Mathematical Programming 14 (1978) 41–72.
North-Holland Publishing Company

LARGE-SCALE LINEARLY CONSTRAINED OPTIMIZATION*

B.A. MURTAGH

University of New South Wales, Sydney, Australia

M.A. SAUNDERS

DSIR, Wellington, New Zealand
Stanford University, Stanford, CA, U.S.A.

Received 27 September 1976
Revised manuscript received 11 July 1977

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 quasi-Newton methods for handling the nonlinearities. A general-purpose production code (MINOS) is described, along with computational experience on a wide variety of problems.

Key words: Large-scale Systems, Linear Constraints, Linear Programming, Nonlinear Programming, Optimization, Quasi-Newton Method, Reduced-gradient Method, Simplex Method, Sparse Matrix, Variable-metric Method.

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 standard form:

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

$$\text{subject to} \quad A\boldsymbol{x} = \boldsymbol{b}, \tag{2}$$

$$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{3}$$

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

* This research was supported by the U.S. Office of Naval Research (Contract N00014-75-C-0267), the National Science Foundation (Grants MCS71-03341 A04, DCR75-04544), the U.S. Energy Research and Development Administration (Contract E(04-3)-326 PA #18), the Victoria University of Wellington, New Zealand, and the Department of Scientific and Industrial Research Wellington, New Zealand.

portion x^N:

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

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

$$\nabla f(x^N) = g(x^N),$$

and we assume that both f and g can be computed at any feasible point x^N.

The research work reported here was stimulated by some of the deficiencies in the algorithms of Murtagh and Sargent [44, 50] and Goldfarb [30], 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

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

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.

2. Basis of the method

2.1. Basic solutions; justification for standard form

Before proceeding to the nonlinear problem, we need to introduce some linear programming background. In particular, equations (1)–(3) with $f(x^N) = 0$ are the standard form for stating linear programs and for solving them in practical

implementations of the simplex method. A *basic solution* is characterized by having at most m "basic" variables lying between their bounds while the remaining $n-m$ "nonbasic" variables are equal to one bound or other. An associated square basis matrix B is drawn from the columns of the constraint matrix A, and as the simplex method proceeds the columns of B are replaced one at a time.

Note that it is assumed in the standard form that A and $\boldsymbol{x}$ contain a full identity matrix and a full set of slack variables, respectively. (General equality and inequality constraints are accommodated by placing suitable upper and lower bounds on the slacks.) There are many practical reasons for retaining the standard form here. Full justification would require much implementation background, but very briefly, a sparse triangular factorization of B can be maintained more easily if columns (but not rows) of B are altered. Further, although the total number of variables is apparently greater, it is very easy to take advantage of the unit vectors associated with slacks, whenever B is re-factorized.

2.2. *Superbasic variables*

One virtue of the concept of basic solutions is the emphasis thereby given to the upper and lower bounds, $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$. It is misleading to regard these as "sparse constraints"; more importantly, they serve directly to eliminate a large percentage of the variables. The simplex method is therefore free to focus its attention on transforming (factorizing) just B, rather than the whole of A. (When B is large and sparse, this is problem enough.)

With nonlinear problems we cannot expect an optimal point to be a basic solution. However, if the number of nonlinear variables is small, it seems reasonable to suppose that an optimal solution will be "nearly" basic. Thus, as a simple generalization we introduce the notion of *superbasic* variables and partition the set of general constraints (2) as follows:

$$\begin{array}{c} \begin{array}{ccc} m & s & n-m-s \end{array} \\ A\boldsymbol{x} = \left[\begin{array}{ccc} B & S & N \end{array}\right] \left[\begin{array}{c} \boldsymbol{x}_B \\ \boldsymbol{x}_S \\ \boldsymbol{x}_N \end{array}\right] = \boldsymbol{b}. \\ \begin{array}{ccc} \text{basics} & \text{super-} & \text{nonbasics} \\ & \text{basics} & \end{array} \end{array} \tag{4}$$

The matrix B is square and nonsingular as in the simplex method, S is $m \times s$ with $0 \leq s \leq n-m$, and N is the remaining columns of A. The associated variables $\boldsymbol{x}_B, \boldsymbol{x}_S, \boldsymbol{x}_N$ are called the *basics*, *superbasics* and *nonbasics* respectively. Both basics and superbasics are free to vary between their bounds. The

name is chosen to highlight the superbasics' role as "driving force"; they may be moved in any direction at all (preferably one that improves the objective value), and the basics are then obliged to change in a definite way to maintain feasibility with respect to the constraints $A\boldsymbol{x} = \boldsymbol{b}$.

Our expectation that solutions will often be "nearly" basic is confirmed by the following theorem:

Theorem 1. *Suppose a nonlinear program has t variables occurring nonlinearly (in either the objective or the constraints). 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 if the nonlinear variables are now regarded as superbasic in the original problem. (At first sight $s = t$, but if any of the nonlinear variables are actually on a bound we can label them nonbasic, and then $s < t$.)

2.3. Derivation of method

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

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x} + \tfrac{1}{2}\Delta x^{\mathrm{T}} G(\boldsymbol{x} + \gamma\Delta\boldsymbol{x})\Delta\boldsymbol{x} \tag{5}$$

where $0 \leq \gamma \leq 1$, and $G(\boldsymbol{x} + \gamma\Delta\boldsymbol{x})$ is the Hessian matrix of second partial derivatives evaluated at some point between $\boldsymbol{x}$ and $\boldsymbol{x} + \Delta\boldsymbol{x}$. Note that G is a constant matrix if $f(\boldsymbol{x})$ is a quadratic function.

Let us partition $\Delta\boldsymbol{x}$ and $\boldsymbol{g}(\boldsymbol{x})$ corresponding to the partitioning of A.

If $f(\boldsymbol{x})$ were truly quadratic, we could obtain a constrained stationary point at $\boldsymbol{x} + \Delta\boldsymbol{x}$ by requiring two properties of the step $\Delta\boldsymbol{x}$:

Property 1.

$$\begin{bmatrix} B & S & N \\ 0 & 0 & \mathrm{I} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}_B \\ \Delta\boldsymbol{x}_S \\ \Delta\boldsymbol{x}_N \end{bmatrix} = \boldsymbol{0}, \tag{6}$$

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

Property 2.

$$\begin{bmatrix} \boldsymbol{g}_B \\ \boldsymbol{g}_S \\ \boldsymbol{g}_N \end{bmatrix} + G \begin{bmatrix} \Delta\boldsymbol{x}_B \\ \Delta\boldsymbol{x}_S \\ \Delta\boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} B^{\mathrm{T}} & 0 \\ S^{\mathrm{T}} & 0 \\ N^{\mathrm{T}} & \mathrm{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\lambda} \end{bmatrix}, \tag{7}$$

i.e., the gradient at $x + \Delta x$ (given by the left hand-side of (7)) is orthogonal to the surface of active constraints and is therefore expressible as a linear combination of the active constraint normals.

For a more general function $f(x)$, the step Δx may not lead directly to a stationary point, but we shall use Properties 1 and 2 to determine a feasible descent direction.

From (6) we have:

$$\Delta x_N = 0, \tag{8}$$

and

$$\Delta x_B = -W\Delta x_S, \tag{9}$$

where

$$W = B^{-1}S. \tag{10}$$

Thus,

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

Equation (7) simplifies when multiplied by the matrix

$$\begin{bmatrix} \mathrm{I} & 0 & 0 \\ -W^{\mathrm{T}} & \mathrm{I} & 0 \\ 0 & 0 & \mathrm{I} \end{bmatrix}. \tag{11}$$

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

$$B^{\mathrm{T}}\mu = g_B + [\mathrm{I} \quad 0 \quad 0]G \begin{bmatrix} -W \\ \mathrm{I} \\ 0 \end{bmatrix} \Delta x_S. \tag{12}$$

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

$$B^{\mathrm{T}}\mu = g_B \tag{13}$$

in which case μ is analogous to the pricing vector π in the revised simplex method. (From now on we shall denote the solution of (13) by π.) Next we have from (7) that

$$\lambda = g_N - N^{\mathrm{T}}\mu + [0 \quad 0 \quad \mathrm{I}]G \begin{bmatrix} -W \\ \mathrm{I} \\ 0 \end{bmatrix} \Delta x_S \tag{14}$$

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

$$\lambda = g_N - N^{\mathrm{T}}\pi \tag{15}$$

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

The third result from equation (7), following pre-multiplication by the matrix

(11), is an expression for the appropriate step:

$$[-W^{\mathrm{T}}\ \mathrm{I}\ 0]G\begin{bmatrix}-W\\ \mathrm{I}\\ 0\end{bmatrix}\Delta x_S = -h \tag{16}$$

where

$$h = [-W^{\mathrm{T}}\ \mathrm{I}\ 0]g = g_S - W^{\mathrm{T}}g_B = g_S - S^{\mathrm{T}}\pi. \tag{17}$$

The form of equation (16) suggests that

$$[-W^{\mathrm{T}}\ \mathrm{I}\ 0]G\begin{bmatrix}-W\\ \mathrm{I}\\ 0\end{bmatrix} \tag{18}$$

can be regarded as a "reduced" Hessian and $h = [-W^{\mathrm{T}}\ \mathrm{I}\ 0]g$ a reduced gradient, with (16) giving a Newton step in the independent variables Δx_S. Note that $\|h\| = 0$ becomes a necessary condition for a stationary point on the current set of active constraints, which, if the reduced Hessian is nonsingular, implies that $\|\Delta x_S\| = 0$.

2.4. Summary

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 $\hat{A}x = \hat{b}$ is the current set of $n - s$ active constraints, Z is an $n \times s$ matrix such that

$$\hat{A}Z = 0. \tag{19}$$

This characterization may be used to describe several published algorithms, which are discussed and compared in [25] as well as in the review papers by Fletcher [17] and Sargent [49].

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

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

(B) Form some approximation to the reduced Hessian, viz.

$$G_A \doteq Z^{\mathrm{T}}GZ.$$

(C) Obtain an approximate solution to the system of equations

$$Z^{\mathrm{T}}GZp_A = -Z^{\mathrm{T}}g \tag{20}$$

by solving the system

$$G_A p_A = -g_A.$$

(D) Compute the search direction $p = Zp_A$.

(E) Perform a linesearch to find an approximation to α^*, where

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

Apart from having full column rank, eq. (19) 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

$$Z = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \begin{matrix} \}m \\ \}s \\ \}n-m-s. \end{matrix} \tag{21}$$

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 S and a triangular (LU) factorization of B. The matrix Z itself is never computed.

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 (20)). 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 $[B\ S]$:

$$[B\ S] = [L\ 0]DV$$

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 S has more than 1 or 2 columns, this factorization will always be substantially more dense than an LU factorization of B. Thus on the grounds of efficiency we proceed with the Z in (21). At the same time we are conscious (from the unwelcome appearance of B^{-1}) that B 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 considerably 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[1] 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].[2]

(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.[3]

(5) Imbedding of non-spike columns of L within A.

(6) Stable updating of the LU factors of B 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, as implemented by J.A. Tomlin, following ideas due to Rarick [48] and Conn [10].

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 and S 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 linesearch. The particular routine used is a Fortran translation of Gill and Murray's Algol 60 procedure *delinsearch*,[4] 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.

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

[2] This has been dispensed with in later versions of MINOS, since in a pure Fortran code it results in only moderate storage savings and considerable loss in execution speed.

[3] The block-triangular structure of B is currently being found using subroutines MC13 and MC21 from the Harwell Subroutine Library (Duff [14], Duff and Reid [15]). Hellerman and Rarick's P^3 [32] is then applied to each block.

[4] More recently implemented as Fortran subroutines LNSRCH and NEWPTC by Gill et al. [59].

Assume we have the following:

(a) A feasible vector $\boldsymbol{x}$ satisfying $[B\ S\ N]\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$.

(b) The corresponding function value $f(\boldsymbol{x})$ and gradient vector $\boldsymbol{g}(\boldsymbol{x}) = [\boldsymbol{g}_B\ \boldsymbol{g}_S\ \boldsymbol{g}_N]^T$.

(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.

(e) A factorization, R^TR, of a quasi-Newton approximation to the $s \times s$ matrix Z^TGZ. (Note that G, Z and Z^TGZ are never actually computed.)

(f) A vector $\boldsymbol{\pi}$ satisfying $B^T\boldsymbol{\pi} = \boldsymbol{g}_B$.

(g) The reduced-gradient vector $\boldsymbol{h} = \boldsymbol{g}_S - S^T\boldsymbol{\pi}$.

(h) Small positive convergence tolerances TOLRG and TOLDJ.

Step 1. (Test for convergence in the current subspace). If $\|\boldsymbol{h}\| >$ TOLRG go to step 3.

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

(a) Calculate $\boldsymbol{\lambda} = \boldsymbol{g}_N - N^T\boldsymbol{\pi}$.

(b) Select $\lambda_{q_1} < -$TOLDJ $(\lambda_{q_2} > +$TOLDJ), the largest elements of $\boldsymbol{\lambda}$ 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 $\boldsymbol{a}_q$ as a new column of S;

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

(iv) add a suitable new column to R.

(d) Increase s by 1.

(Note: MINOS also has a MULTIPLE PRICE option which allows more than one nonbasic variable to become superbasic.)

Step 3. (Compute direction of search, $\boldsymbol{p} = Z\boldsymbol{p}_S$).

(a) Solve $R^TR\boldsymbol{p}_S = -\boldsymbol{h}$.

(b) Solve $\mathrm{LU}\boldsymbol{p}_B = -S\boldsymbol{p}_S$.

(c) Set $\boldsymbol{p} = \begin{bmatrix} \boldsymbol{p}_B \\ \boldsymbol{p}_S \\ 0 \end{bmatrix}$.

Step 4. (Ratio test, "CHUZR").

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

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

Step 5. (Linesearch).

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

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

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

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

(a) Solve $U^TL^T\boldsymbol{\pi} = \boldsymbol{g}_B$.

(b) Compute the new reduced gradient, $\bar{\boldsymbol{h}} = \boldsymbol{g}_S - S^T\boldsymbol{\pi}$.

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

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

(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 if necessary; 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 $[B\ S]$ 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

$$\begin{bmatrix} B \\ \boldsymbol{x}_B^T \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} S \\ \boldsymbol{x}_S^T \end{bmatrix}$$

respectively, where q is chosen to keep B nonsingular (this requires a vector $\boldsymbol{\pi}_p$ which satisfies $U^TL^T\boldsymbol{\pi}_p = \boldsymbol{e}_p$);

(ii) modify L, U, R and $\boldsymbol{\pi}$ to reflect this change in B;

(iii) compute the new reduced gradient $\boldsymbol{h} = \boldsymbol{g}_S - S^T\boldsymbol{\pi}$;

(iv) go to (c).

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

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

(i) delete the q-th columns of

$$\begin{bmatrix} S \\ \boldsymbol{x}_S^T \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} R \\ \boldsymbol{h}^T \end{bmatrix};$$

(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 $\|\boldsymbol{h}\|$ 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 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 time per iteration as if the problem were entirely linear. (The total number of iterations required is, of course, undetermined.)

3.3. Updating the matrix factorizations

As in the simplex method, a stable factorization of the basis matrix B is important for solving equations of the form $By = b$ or $B^Tz = c$. Here we use an implementation of the method of Bartels and Golub [2, 3] for updating the factorization $B = LU$. Details are given in Saunders [52]. We normally refactorize B 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 $\bar{R}$ will be used to represent R after any particular modification. To ensure stability, all modifications to R have been implemented using elementary orthogonal matrices $Q_{j,k}$ (plane rotations) whose non-trivial elements are at the intersection of the j-th and k-th rows and columns, and are of the form

$$\begin{bmatrix} c & s \\ s & -c \end{bmatrix}, \quad \text{where } c^2 + s^2 = 1.$$

3.3.1. Quasi-Newton 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

$$\text{COMDFP:} \quad \bar{R}^T\bar{R} = R^TR + \frac{1}{\alpha y^T p_S} yy^T + \frac{1}{h^T p_S} hh^T.$$

The Rank-one Formula

$$\text{RANK1:} \quad \bar{R}^T\bar{R} = R^TR + \frac{1}{\alpha w^T p_S} ww^T,$$

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 $\alpha_{\max}$ is very small it is possible that the computed value of y will be meaningless. Following the suggestion of M.J.D. Powell (private com-

munication) we allow for this by monitoring the change in directional derivative and modifying R only if

$$\bar{\boldsymbol{h}}^{\mathrm{T}}\boldsymbol{p}_S > 0.9\boldsymbol{h}^{\mathrm{T}}\boldsymbol{p}_S.$$

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

$$\eta \equiv \frac{-\bar{\boldsymbol{h}}^{\mathrm{T}}\boldsymbol{p}_S}{|\boldsymbol{h}^{\mathrm{T}}\boldsymbol{p}_S|} < 0.9,$$

which will normally be true if a value *eta* < 0.9 is given to the parameter of procedure *delinsearch*, which uses $|\eta| \leq eta$ as one criterion for a successful search. (Note that $\boldsymbol{g}^{\mathrm{T}}\boldsymbol{p} = \boldsymbol{g}^{\mathrm{T}}Z\boldsymbol{p}_S = \boldsymbol{h}^{\mathrm{T}}\boldsymbol{p}_S$.) The test also ensures that the COMDFP update will preserve positive definiteness.

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

R1ADD: $\bar{R}^{\mathrm{T}}\bar{R} = R^{\mathrm{T}}R + \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$,

R1SUB: $\bar{R}^{\mathrm{T}}\bar{R} = R^{\mathrm{T}}R - \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$.

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 qth 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

$$\bar{Z} = Z(\mathrm{I} + \boldsymbol{e}_q\boldsymbol{v}^{\mathrm{T}}) \tag{22}$$

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

$$\begin{aligned} B^{\mathrm{T}}\boldsymbol{\pi}_p &= \boldsymbol{e}_p, \\ \boldsymbol{y} &= S^{\mathrm{T}}\boldsymbol{\pi}_p, \\ y_q &= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{e}_q, \\ \boldsymbol{v} &= -\frac{1}{y_q}(\boldsymbol{y} + \boldsymbol{e}_q). \end{aligned}$$

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 $\boldsymbol{y}$, viz.

$$y_j = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{e}_j = \boldsymbol{\pi}_p^{\mathrm{T}}S\boldsymbol{e}_j = \boldsymbol{e}_p^{\mathrm{T}}B^{-1}(S\boldsymbol{e}_j)$$

is the pivot element that would arise if the j-th column of S were selected for the basis change. Hence $\boldsymbol{y}$ can be used as a guide for determining q. Broadly

speaking, the condition of B will be preserved as well as possible if y_q is the largest available pivot element (assuming the columns of S 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, with j ranging over the superbasic set, we define q by the following:

$$y_{\max} = \max |y_j|,$$
$$d_j = \min\{|x_j - l_j|, |x_j - u_j|\} \text{ (for each } j),$$
$$d_q = \max\{d_j \mid |y_j| \geq 0.1 y_{\max}\}.$$

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

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

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) $\boldsymbol{\pi}_p$ can also be used to update the LU factors of B (Tomlin [54]). Conversely, the updated LU factors of B can provide $\boldsymbol{\pi}_p$ more cheaply than solving $B^{\mathrm{T}}\boldsymbol{\pi}_p = \boldsymbol{e}_p$ (Goldfarb [31]).

The modification to R corresponding to eq. (22) is accomplished as follows:

R1PROD: $\bar{R}^{\mathrm{T}}\bar{R} = (\mathrm{I} + \boldsymbol{v}\boldsymbol{e}_q^{\mathrm{T}})R^{\mathrm{T}}R(\mathrm{I} + \boldsymbol{e}_q\boldsymbol{v}^{\mathrm{T}})$.

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

$$\bar{R}^{\mathrm{T}}\bar{R} = (R^{\mathrm{T}} + \boldsymbol{v}\boldsymbol{r}_q^{\mathrm{T}})(R + \boldsymbol{r}_q\boldsymbol{v}^{\mathrm{T}}).$$

A partial backward sweep of plane rotations $Q_{q,j}$ $(j = q-1, \ldots, 1)$ reduces $\boldsymbol{r}_q$ to a multiple of $\boldsymbol{e}_q$, filling in the q-th row of R. A multiple of $\boldsymbol{v}$ is added to this row, and then a partial forward sweep of rotations $Q_{j,q}$ $(j = 1, \ldots, q-1)$ restores R to triangular form. (We could use other methods designed for a general modifying matrix $\mathrm{I} + \boldsymbol{w}\boldsymbol{v}^{\mathrm{T}}$, but the method described takes full advantage of the special case $\boldsymbol{w} = \boldsymbol{e}_q$. It also allows some rotations in the backward sweep to be skipped if the corresponding elements of $\boldsymbol{r}_q$ are zero.)

3.3.3. Removal of one superbasic variable (step (7c))

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 $Q_{j,j+1}$. $(j = q, \ldots, s-1)$:

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

3.3.4. Addition of one superbasic variable (step 2(c))

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

$$\bar{Z} = [Z \;\; z], \quad \text{where } z = \begin{bmatrix} -B^{-1}a_q \\ e_s \\ 0 \end{bmatrix}.$$

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

$$v = \frac{g(x + \delta z) - g(x)}{\delta} = Gz + O(\delta \|z\|^2),$$

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

$$\text{ADDCOL:} \begin{cases} \text{Solve} \quad R^T r = Z^T v, \\ \text{Compute} \quad \sigma = z^T v - \|r\|^2, \rho = |\sigma|^{1/2}, \\ \text{Take} \quad \bar{R} = \begin{bmatrix} R & r \\ & \rho \end{bmatrix}. \end{cases}$$

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

Comparison of

$$\bar{R}^T\bar{R} = \begin{bmatrix} R^T & \\ r^T & \rho \end{bmatrix}\begin{bmatrix} R & r \\ & \rho \end{bmatrix} = \begin{bmatrix} R^TR & Z^Tv \\ v^TZ & z^Tv \end{bmatrix}$$

and

$$\bar{Z}^TG\bar{Z} = \begin{bmatrix} Z^T \\ z^T \end{bmatrix} G[Z \;\; z] = \begin{bmatrix} Z^TGZ & Z^TGz \\ z^TGZ & z^TGz \end{bmatrix}$$

shows that if R^TR provides a good approximation to Z^TGZ then $\bar{R}^T\bar{R}$ has some chance of being a useful approximation to $\bar{Z}^TG\bar{Z}$. The main work involved here is in computing $B^{-1}a_q$, the gradient vector $g(x + \delta z)$, and the reduction $\bar{Z}^T v$. This work is essentially wasted if the expression for σ is not positive, which may happen for many reasons, e.g., if $\bar{Z}^TG\bar{Z}$ 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 $r = 0$ and take ρ to be either $(z^Tv)^{1/2}$ or 1.0, thus:

$$\bar{R} = \begin{bmatrix} R & 0 \\ & \rho \end{bmatrix}. \tag{23}$$

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 r and ρ as shown then leads to significantly fewer iterations than if (23) 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 Gz 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, 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 (23) 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 $\|v\|$ and resort to (23) whenever $\|v\|$ is significantly larger than $d_{\max}$ or smaller than $d_{\min}$. (As a side benefit, the expense of computing $\bar{Z}^T v$ and 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 certainly retains more information 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 estimate Lagrange multipliers (reduced costs, λ) and add one more superbasic variable.

Suppose that after a particular iteration we have

$$\begin{aligned} \Delta x_S &= \text{the change in the superbasic variables,} \\ \Delta f &= \text{the change in } f, \\ \pi &= \text{the new pricing vector,} \\ h &= Z^T g, \text{ the new reduced gradient,} \\ \epsilon_x, \epsilon_f, \text{TOLRG}, \epsilon_g &= \text{positive scalars,} \\ \epsilon &= \text{machine precision,} \end{aligned}$$

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

T_1: $\|\Delta x_S\| \leq (\epsilon_x + \epsilon^{1/2})(1 + \|x_S\|)$,

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

T_3: $\|h\| \leq \text{TOLRG}$,

T_4: $\|h\| \leq \epsilon_g \|\pi\|$.

In place of the simple test

if T_3 *then* compute $\boldsymbol{\lambda}$,

the following combined test is used:

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

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 $\boldsymbol{x}$ very close to an optimum in the current subspace (e.g., the quasi-Newton procedure should achieve this regularly if $f(\boldsymbol{x})$ 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 $\lambda_q = g_q - \boldsymbol{\pi}^T \boldsymbol{a}_q$. If the corresponding variable x_q becomes superbasic, the reduced gradient for the expanded subspace will be

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

Now recall from eq. (14) that unless $\boldsymbol{h}$ is reasonably small, even one further iteration could change $\boldsymbol{\pi}$ 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 $\|\boldsymbol{h}\|_\infty \leq 0.9|\lambda_q|$, which implies

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

We then reset TOLRG for the new subspace to be

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

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

In the discussion so far, and in the existing implementation, we have assumed that both $f(\boldsymbol{x})$ and its gradient $g(\boldsymbol{x})$ are available via a user-written subroutine. We do not store the matrix Z explicitly and we make no use of the Hessian

matrix $G(\boldsymbol{x})$. (Instead we maintain a quasi-Newton approximation to the reduced Hessian, $Z^{\mathrm{T}}GZ$.)

Some discussion of potential 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 $\boldsymbol{p} = Z\boldsymbol{p}_S$ and $\boldsymbol{h} = Z^{\mathrm{T}}\boldsymbol{g}$. (These calculations are analogous to the FTRAN and BTRAN operations in linear programming.)

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

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

(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. $\boldsymbol{h} \approx Z^{\mathrm{T}}\hat{\boldsymbol{g}}$. For greater accuracy, central differences may be used, at the cost of extra function calculations.

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

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

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

(3) If $\boldsymbol{g}(\boldsymbol{x})$ and perhaps $G(\boldsymbol{x})$ are available, the system of equations

$$Z^{\mathrm{T}}GZ\boldsymbol{p}_S = -Z^{\mathrm{T}}\boldsymbol{g} \tag{24}$$

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

$$Z^{\mathrm{T}}GZ = [\boldsymbol{z}_i^{\mathrm{T}}G\boldsymbol{z}_j]$$

or differencing $\boldsymbol{g}(\boldsymbol{x})$ thus:

$$\boldsymbol{v}_j = \frac{\boldsymbol{g}(\boldsymbol{x} + \delta \boldsymbol{z}_j) - \boldsymbol{g}(\boldsymbol{x})}{\delta} \equiv V\boldsymbol{e}_j,$$

$$Z^{\mathrm{T}}GZ \approx \tfrac{1}{2}(Z^{\mathrm{T}}V + V^{\mathrm{T}}Z).$$

However the need for the vectors $\boldsymbol{z}_j$ again presents severe difficulties for large problems.

(4) If G is large and sparse, eq. (24) 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 $Z^{\mathrm{T}}GZ$ or any approximation to it. However if Z has s columns the method would typically require $O(s)$ products of the form $Z^{\mathrm{T}}(G(Z\boldsymbol{v}))$.

(5) A final (more promising) alternative is to abandon eq. (24) 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) $\bar{\boldsymbol{h}} = -Z^{\mathrm{T}}\bar{\boldsymbol{g}}$

(b) *if* restart *then* $\bar{\boldsymbol{p}}_S = -\bar{\boldsymbol{h}}$

else $\bar{\boldsymbol{p}}_S = -\bar{\boldsymbol{h}} + \beta \boldsymbol{p}_S$

(c) $\boldsymbol{p} = Z\bar{\boldsymbol{p}}_S$

where $\boldsymbol{p}_S, \bar{\boldsymbol{p}}_S$ 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]:	$\beta = \|\bar{\boldsymbol{h}}\|^2/\|\boldsymbol{h}\|^2$;
Polak and Ribiere [46]:	$\beta = \bar{\boldsymbol{h}}^{\mathrm{T}}(\bar{\boldsymbol{h}} - \boldsymbol{h})/\|\boldsymbol{h}\|^2$;
Perry [45]:	$\beta = \bar{\boldsymbol{h}}^{\mathrm{T}}(\bar{\boldsymbol{h}} - \boldsymbol{h} - \alpha \boldsymbol{p}_S)/\boldsymbol{p}_S^{\mathrm{T}}(\bar{\boldsymbol{h}} - \boldsymbol{h})$.

In MINOS, one of these methods is used if, at a particular iteration, the number of superbasics s is 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 Subsection 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. Even if the *exact* Hessian matrix were available (unless it were of very special form) it seems that 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 200 (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

$$R^{\mathrm{T}}R = Z^{\mathrm{T}}GZ \tag{25}$$

can often be maintained exactly without recomputing $Z^{\mathrm{T}}GZ$ every iteration. Such a specialization has been described by Gill and Murray [26], along with the measures required to allow for $Z^{\mathrm{T}}GZ$ being indefinite. The present quasi-

Newton algorithm could conceivably be specialized as follows:

(1) Initialize R at the start of a run to satisfy (25). (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 (Subsection 3.3.4) compute the vector $v = Gz$ directly rather than by differencing the gradient.

(3) Suppress the quasi-Newton updates to R (COMDFP and RANK1 in Subsection 3.3.1).

However it is worth noting that the difference approximation to $v = Gz$ will be essentially exact, so that if (25) ever holds at any stage then ADDCOL will maintain (25) almost exactly when a column is added to Z. A step $\alpha = 1.0$ along the next search direction will then move x to the new subspace minimum. Now it is easily verified that the subsequent quasi-Newton 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 quasi-Newton updates will then alter R, (25) 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 $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ 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

$$f(\boldsymbol{x}) = \sum_k \left[\sum_j x_{jk} \left(c_{jk} + \ln \left(x_{jk} / \sum_i x_{ik} \right) \right) \right].$$

(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

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

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 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 Subsection 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 $\|\boldsymbol{h}\|/\|\boldsymbol{\pi}\|$ was reduced to 10^{-6} or less at optimality.

5.2.1. The chemical equilibrium problem (problem 3)

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

$$f = \sum_i x_i g_i,$$

whose gradient components are

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

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

Problem no.	Rows	Columns	Nonzero elements	Nonlinear variables	Iterations[a]	Evaluations of $f(x)$, $g(x)$	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	504	24	103	72	0	42.43	0.51
8	324	425	1404	91	348	215	0	538.3	6.48

[a] Includes Phase 1 iterations.

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

Problem no.[a]	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

[a] 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 = 12BOTH, restart from 10b.

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 $f(\boldsymbol{x})$ 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.

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

Lo-bound ϵ_k	No. of superbasics	$f(\boldsymbol{x})$	Iterations[a]	Evaluations[a] of f, g	Estimate of[b] $\kappa(R^TR)$
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	

[a] Additional to previous run.

[b] A lower bound on the condition number of the reduced Hessian approximation R^TR is the square of the ratio of the largest and smallest diagonals of R.

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.

For interest, the last row of Table 4 shows the values obtained by the program SUMT as reported by Himmelblau [34]. For the 8 smallest x_j the two results differ in all significant figures. (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.

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

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

(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 $f(\boldsymbol{x} + \alpha \boldsymbol{p})$ 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 $f(\boldsymbol{x})$ 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

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

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 Section 3);

η_g = the tolerance for minimization within each subspace (see Subsection 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)

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–Ribière		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

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.

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

η	quasi-Newton		Fletcher–Reeves		Polak–Ribière		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

To illustrate that the cg methods are not to be discarded immediately, in Fig. 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.

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

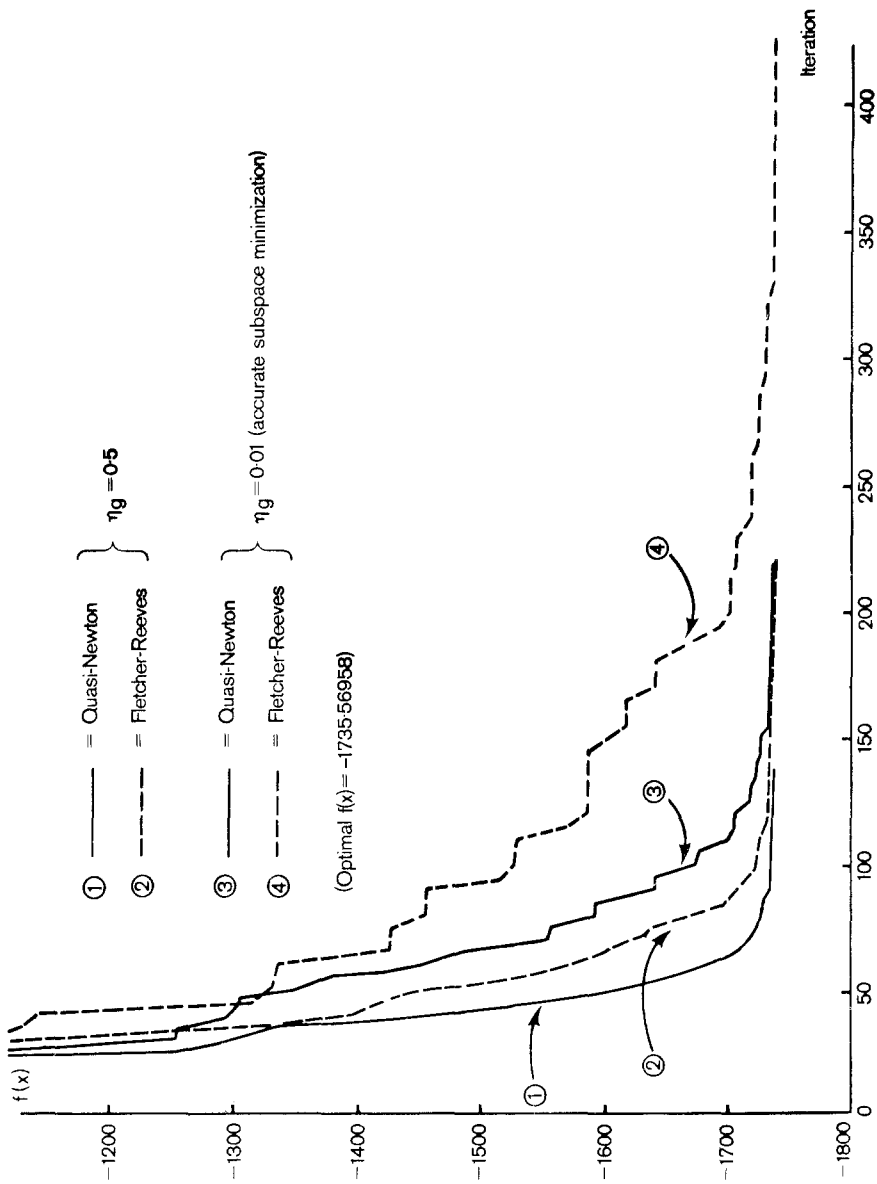

Fig. 1. Comparison of quasi-Newton (complementary DFP) and conjugate gradients (Fletcher-Reeves). Objective value is plotted against iteration number for the weapon assignment problem, using $\eta = 0.01$ (accurate linesearch).

keeping the number of superbasic variables as small as possible and changing that number by a small amount (usually 1) each iteration.[5] With the quasi-Newton approach, this strategy retains maximum information about the reduced Hessian. Even though the proof of convergence [41] for the variable-reduction method depended on regular resetting of the reduced 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 reduced Hessian are normally very ill-conditioned). Zig-zagging is controlled effectively by the tolerance η_g and the logic described in Subsection 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 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. If there are *many* quadratic terms and if the optimal solution has most of the variables away from their bounds, then a sparse-matrix implementation of one of the complementarity methods will be preferable (e.g., Tomlin's implementation [60] of Lemke's method).

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 (standard 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

[5] 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.

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 problems, and it should provide a new dimension of utility to an already substantial body of large-scale linear programming models.

Acknowledgment

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. Revision suggestions by the referees and by C.C. Paige are also gratefully acknowledged.

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", *Numerische Mathematik* 16 (1971) 414–434.
[3] R.H. Bartels and G.H. Golub, "The simplex method of linear programming using LU decomposition", *Communications of 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", *Communications of ACM* 16 (1973) 105–109.
[7] C.G. Broyden, "Quasi-Newton methods", in: W. Murray, ed., *Numerical methods for unconstrained optimization* (Academic Press, New York, 1972) pp. 87–106.
[8] A. Buckley, "An alternate implementation of Goldfarb's minimization algorithm", *Mathematical Programming* 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 Journal of Numerical Analysis* 13 (1) (1976) 145–154.
[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, NJ, 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.

[15] I.S. Duff and J.K. Reid, "An implementation of Tarjan's algorithm for the block triangularization 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 Journal* 6 (1963) 163–168.
[19] R. Fletcher and C.M. Reeves, "Function minimization by conjugate gradients", *Computer Journal* 7 (1964) 149–154.
[20] P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders, "Methods for modifying matrix factorizations", *Mathematics of Computation* 28 (1974) 505–535.
[21] P.E. Gill and W. Murray, "Quasi-Newton methods for unconstrained optimization", *Journal of Institute of Mathematics and its Applications* 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", *Mathematical Programming* 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, 1977), Proceedings 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", *Mathematics of Computation* 29 (1975) 1051–1077.
[30] D. Goldfarb, "Extension of Davidon's variable metric method to maximization under linear inequality and equality constraints", *SIAM Journal of Applied Mathematics* 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", *Mathematical Programming* 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) pp. 67–76.
[34] D.M. Himmelblau, *Applied nonlinear programming* (McGraw-Hill, New York, 1972).
[35] A. Jain, L.S. Lasdon and M.A. Saunders, "An in-core nonlinear mathematical programming system for large sparse nonlinear programs", 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 Science* 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", *Bell Journal of Economics* (Autumn 1976) 381–406.
[41] G.P. McCormick, "The variable-reduction method for nonlinear programming", *Management Science* 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, New York, 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, MA, U.S.A.
[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", *Mathematical Programming* 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, CA, U.S.A.
[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, 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. Operational Research* 4 (2) (1976) 101–117.
[54] J.A. Tomlin, "On pricing and backward transformation in linear programming", *Mathematical Programming* 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.
[59] P.E. Gill, W. Murray, S.M. Picken, H.M. Barber and M.H. Wright, Subroutine LNSRCH, NPL Algorithms Library, Reference No. E4/16/0/Fortran/02/76 (February 1976).
[60] J.A. Tomlin, "Robust implementation of Lemke's method for the linear complementarity problem", Technical Report SOL 76-24, Systems Optimization Laboratory, Stanford University (September 1976).