SIAM J. NUMER. ANAL.
Vol. 13, No. 1, March 1976

LINEAR PROGRAMMING VIA A NONDIFFERENTIABLE PENALTY FUNCTION*

A. R. CONN†

Abstract. A numerically stable form of an algorithm that is closely related to the work of Gill and Murray [5] and Conn [3] is presented.

Among other reasons, the penalty function approach has never been available for linear programming in a viable sense because of the inherent nonlinearities introduced. The nondifferentiable penalty function is unique in that it is a piecewise linear function and hence maintains a computational efficiency comparable with, and in general, better than, the standard form.

The method admits nonsimplex steps, and this feature enables it to be readily generalized to quadratic programming.

Introduction. In recent years several authors, e.g., [5] have noted that the unfortunate terminology associated with linear programming has served to isolate the subject from the mainstream of linear algebra and numerical analysis.

In this paper, the point of view of numerical analysts [1], [2] and [5] is preferred insofar as it emphasizes the close relationship between linear and nonlinear programming. This is not surprising since the main content of this paper is to apply a nonlinear programming technique, namely, the nondifferentiable penalty technique of Conn [3], to the specialized case of linear programming. By showing the parallel between this technique and the simplex algorithm, we hope to further elucidate the structure of the algorithm and the advantages that can arise from such an approach.

Among other reasons, the penalty function approach has never been available for linear programming in a viable sense because of the inherent nonlinearities. The nondifferentiable penalty function is unique in that it is a piecewise linear function.

Geometrically we may consider the method in the following way. By virtue of the penalty function we are able to define a convex polytope of degree K which defines a region such that the penalty function has the same optimum value within this region as the original problem. Furthermore, the facets of the polytope are always a subset of the original defining facets of the possible region and $K \leqq l$, the degree of the original polytope. (By the degree of a polytope we mean the number of its vertices.)

1. Preliminaries. This paper is concerned with the solution of the following linear programming problem:

$$\min \{z = c^T x\}$$

* Received by the editors March 12, 1974.

† Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1. This research was supported in part by the National Research Council of Canada under Grant A8639.

subject to the constraints

$$A^T x \geqq b,$$

where A is an $n \times m$ matrix, $m \geqq n$.

Following Gill and Murray [5], it is assumed that (since [5] is essential background to this paper, whenever convenient I have kept to this terminology):

(i) There exists at least one $x \in E^n$ for which $A^T x \geqq b$.

(ii) z is bounded in the feasible region.

(iii) A and $[A; c]$ satisfy the Haar condition for matrices.

The final condition ensures that degeneracy and cycling cannot occur during the simplex algorithm.

We shall deal with a more general statement of the problem and weakening of the conditions later.

We shall begin by considering how Conn's algorithm behaves on the problem

$$\min f(x, y) = y + \tfrac{1}{2}x$$

subject to

$$\phi_1(x, y) \equiv y - x \geqq 0,$$

$$\phi_2(x, y) \equiv y + x \geqq 0,$$

$$\phi_3(x, y) \equiv -y \geqq -2.$$

This problem is illustrated in Fig. 1.

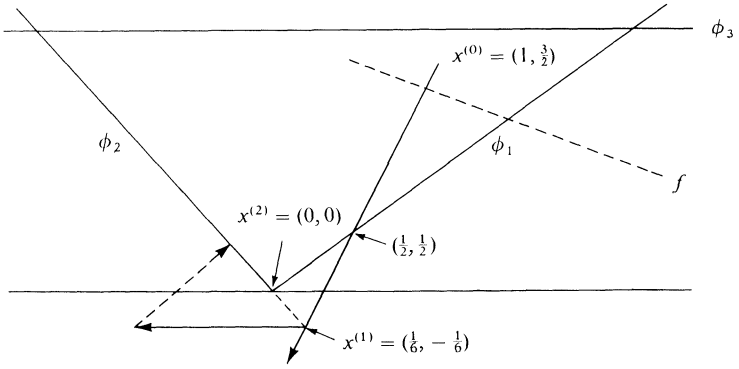

FIG. 1

We shall use the notation of [3] noting that in this special case (i.e., linear programming) we may assume that $\varepsilon = 0$. Let our starting point be $x^{(0)} = (1, \frac{3}{2})$, with $\mu = 1$. (The choice of μ will come under discussion at a later date.) Then no constraints are active and the initial direction of search is $z_1 = -\nabla f = (-\frac{1}{2}, -1)$. Proceeding in this direction, we eventually reach $(\frac{1}{2}, \frac{1}{2})$. Now $p_{x_0}(x, y) = y + \frac{1}{2}x - \sum_{123} \min(\phi_i, 0)$. When $\phi_1, \phi_2, \phi_3 \geqq 0$, $p_{x_0}(x, y) = y + \frac{1}{2}x$; as we pass $(\frac{1}{2}, \frac{1}{2})$ in the direction z_1, ϕ_1 becomes negative and $p_{x_0}(x, y) = 3x/2$. Moreover, the direction of decrease of p_{x_0}, u, is given by $u = (-\frac{3}{2}, 0)$. Since $u^T z_1 > 0$, p_{x_0} still decreases in the direction z_1.

Thus we proceed further until $(\frac{1}{6}, -\frac{1}{6})$ is reached; as we pass $(\frac{1}{6}, -\frac{1}{6})$ in addition to $\phi_1 < 0$, we have $\phi_2 < 0$; thus $p_{x_0}(x, y) = x/2 - y$, and the direction of decrease of p_{x_0}, v, is given by $v = (-\frac{1}{2}, 1)$. However, $u^T z_1 < 0$, and so $x^{(1)} = (\frac{1}{6}, -\frac{1}{6})$.

Now ϕ_2 is an active constraint and $p_{x_1}(x, y) = y + \frac{1}{2}x - \sum_{1,2,3} \min(\phi_i, 0)$. Thus we now proceed in the direction $(-\frac{3}{2}, 0)$ projected along ϕ_2; i.e., $z_2 = (-1, 1)$ until we reach $(0, 0)$. At $(0, 0)$ and beyond, in the direction z_2, $\phi_1 \geqq 0$ and $\phi_2 = 0$ $(\phi_3 \geqq 0)$; consequently $p_{x_1}(x, y) = y + \frac{1}{2}x$ and the direction of decrease of p_{x_1}, $w = (-\frac{1}{2}, -1)$. Now $w^T z_2 < 0$. Thus $x_2 = (0, 0)$. Finally, at $(0, 0)$ the conditions for a minimum of p_{x_2} are satisfied. In this case $\mu < \mu_0$ and x_2 is the solution to the problem.

We shall begin by considering the basic algorithm. This is very similar to the simplex algorithm in the sense that we are moving from vertex to vertex, the essential difference being in the selection of the constraints that leave and enter the basis at each iteration. However, it is certainly distinct from any of the standard forms, e.g., primal, dual or primal-dual algorithms. This is because the function being minimized is the penalty function and not the objective function of the original problem. Consequently the corresponding Kuhn–Tucker variables (Lagrange multipliers) are different, and these determine the change of basis.

2. The algorithms.

The basic algorithm. At the beginning of the ith iteration the following matrices and vectors are available:

(i) $A^{(i)}$, an $n \times q$ matrix,
(ii) $\bar{A}^{(i)}$, an $n \times (m-q)$ matrix,
(iii) $b^{(i)}$, a $q \times 1$ column vector,
(iv) $\bar{b}^{(i)}$, an $(m-q) \times 1$ column vector,
(v) $x^{(i)}$, an $n \times 1$ column vector,

where the matrix of constraints is partitioned in the form

$$A^T = \begin{bmatrix} A^{(i)^T} \\ \bar{A}^{(i)^T} \end{bmatrix}. \tag{2.1}$$

The rows of $A^{(i)^T}$, $\bar{A}^{(i)^T}$ are labeled $1(1)q$, $1(1)m-q$, respectively, giving

$$b = \begin{bmatrix} b^{(i)} \\ \bar{b}^{(i)} \end{bmatrix}, \tag{2.2}$$

with

$$A^{(i)^T} x^{(i)} = b^{(i)}, \tag{2.3}$$

$$\bar{A}^{(i)^T} x^{(i)} \neq \bar{b}^{(i)} \tag{2.4}$$

and

(vi) $L^{(i)}$ a lower triangular matrix such that

$$L^{(i)} L^{(i)^T} = A^{(i)^T} A^{(i)}, \tag{2.5}$$

$$d^{(i)} = A^{(i)^T} c^{(i)}, \tag{2.6}$$ (vii)

$$w^i = \bar{A}^{(i)^T} x^{(i)} - \bar{b}^{(i)}, \tag{2.7}$$ (viii)

where c_i^* is defined as follows:

$$\overset{*}{c}{}^i = \mu c - B^{(i)^T}\varepsilon^{(i)}, \tag{2.8}$$

where

$$\varepsilon^{(i)^T} = [1, \cdots, 1], \text{ a } 1 \times s \text{ row vector},$$
$$B^{(i)^T} = [a_{i_1}, \cdots, a_{i_s}], \text{ an } n \times s \text{ matrix},$$

and

$$a_j^T x^{(i)} < b^{(i)},$$
$$j \in \mathcal{Q}\xi = \{i_1, \cdots, i_s\};$$

i.e., $\mathcal{Q}\xi$ is the index set for the unsatisfied constraints.

We note that if $q = n$ and (2.4) is replaced by

$$(\bar{A}^{(i)})^T x^i > \bar{b}^1,$$

(i.e., $\mathcal{Q}\xi = \Phi$) then, since $\overset{*}{c}{}^i$ becomes μc, we have analogous intermediate information to that required by Gill and Murray [5] in their basic iteration for taking simplex steps. We shall begin by stating the algorithm under the assumption that $q = n$ at each stage.

ALGORITHM 1.

Step 1. Determine the Kuhn–Tucker multipliers u by solving the equations

$$L^{(i)}L^{(i)^T}u = d^{(i)}. \tag{2.9}$$

(a) If $u_j \geqq 0$, $j = 1, 2, \cdots, n$, then $x^{(i)}$ is the optimal solution.

(b) If some $u_j < 0$, then choose index t such that

$$u_t = \min\{u_j : j = 1, 2, \cdots, n\}.$$

Step 2. Determine p^T, the tth row of $(A^{(i)})^{-1}$ by solving the equations

$$L^{(i)}L^{(i)^T}y = e_t, \tag{2.10}$$

where e_t is the tth column of the identity matrix; then

$$p = A^{(i)}y. \tag{2.11}$$

Step 3. Order the λ_K's such that

$$\lambda_{K_1} \leqq \lambda_{K_2} \leqq \lambda_{K_3} \leqq \cdots \leqq \lambda_{K_j}, \tag{2.12}$$

where

$$K_j \in \{1, 2, \cdots, m-n | \lambda_{K_j} > 0\}. \tag{2.13}$$

Here

$$\lambda_j = -w^{(i)}/v_j \tag{2.14}$$

and

$$v_j = \bar{A}^{(i)^T}p. \tag{2.15}$$

Step 4. Determine $\lambda_{(i)}$ and update x^i by using Algorithm 1.1 below.

Step 5. Set

$$d_j^{(i+1)} = d_j^{(i)}, \qquad j < t, \tag{2.16}$$

$$d_j^{(i+1)} = d_{j+1}^{(i)}, \qquad t \leqq j < n, \tag{2.17}$$

$$d_n^{(i+1)} = \bar{a}_K^T \overset{*}{c}^{(i+1)}, \tag{2.18}$$

where $\bar{a}_K$ is the corresponding Kth column of $\bar{A}^{(i)}$.

Step 6. Update the residuals of the inactive constraints

$$w_j^{(i+1)} = w_j^{(i)} + \lambda_{(i)}\, v_j, \qquad 1 \leqq j \leqq m-n, \quad j \neq K, \tag{2.19}$$

$$w_K^{(i+1)} = \lambda_{(i)}. \tag{2.20}$$

Step 7. The tth column of $A^{(i)}$ is removed to become the Kth column of $\bar{A}^{(i+1)}$. $A^{(i+1)}$ is formed by relabeling the remaining columns of $A^{(i)}$ from 1 to $n-1$ and adding $\bar{a}_K$ in the nth position. Similarly the tth element of $b^{(i)}$ is removed and placed in the Kth position of $\bar{b}^{(i+1)}$. $b^{(i+1)}$ is formed by relabeling the remaining elements of $b^{(i)}$ from 1 to $n-1$ and adding $\bar{b}_K^{(i)}$ in the nth position.

Step 8. The lower triangular factors $L^{(i)}$ are modified in two stages. When the tth column of $A^{(i)}$ is removed, an intermediate lower triangular matrix L^* is found by one of the methods of Gill and Murray [5]. Similarly, this matrix is in turn modified when $\bar{a}_K$ is added in the nth position.

Algorithm 1.1 used in Step 4, now follows below.

ALGORITHM 1.1.

Step 1. Set

$$j = 0, \quad x_0^{(i)} = x^{(i)}, \quad \overset{*}{c}_0^{(i)} = \overset{*}{c}^{(i)}. \tag{2.21}$$

Step 2. Set

$$j = j + 1. \tag{2.22}$$

Step 3. Set

$$x_j^{(i+1)} = x_j^{(i)} + \lambda_{K_j} p. \tag{2.23}$$

$$\text{If} \quad K_j \in \mathcal{Q}\xi, \tag{2.24}$$

$$\text{set} \quad \overset{*}{c}_j^{(i+1)} = \overset{*}{c}_{j-1}^{(i)} + a_{K_j}, \tag{2.25}$$

$$\text{otherwise} \quad \overset{*}{c}_j^{(i+1)} = \overset{*}{c}_{j-1}^{(i)} - a_{K_k}. \tag{2.26}$$

Step 4. If

$$p^T \overset{*}{c}_j^{(i+1)} > 0, \tag{2.27}$$

$$\text{set} \quad x^{(i+1)} = x_j^{(i+1)}, \qquad \overset{*}{c}^{(i+1)} = \overset{*}{c}_{j-1}^{(i)}. \tag{2.28}$$

$$\text{Let} \quad K = K_j \tag{2.29}$$

and return to Algorithm 1.

$$\text{If} \quad p^T \overset{*}{c}_j^{i+1} \leqq 0 \tag{2.30}$$

return to Step 2.

Some remarks on Algorithm 1.

Remark 1. For simplicity we shall assume that μ is sufficiently small. In actuality the algorithm handles this difficulty automatically, but the details will be reserved for the more general algorithm that follows, i.e., Algorithm 2.

Remark 2. It is clear that p is exactly the direction of search used in Conn's algorithm [3] for the nonlinear programming problem corresponding to this special case.

Remark 3. The only genuine variation between the algorithm just stated above and that of Gill and Murray is contained in the statements (2.8) and Step 4. (i.e., Algorithm 1.1).

The vector given by (2.8) is merely the gradient of the penalty function with the contribution of the active constraints neglected, henceforth referred to as the pseudo-gradient of the penalty function.

Algorithm 1.1 is quite straightforward. The λ_K's are already ordered such that on leaving $x^{(i)}$ in direction p we first encounter the constraint corresponding to λ_{K_1}, then that corresponding to λ_{K_2} and so on. If the constraint encountered was previously unsatisfied, then we update the pseudo-gradient of the penalty function according to (2.25). Similarly, if we now intend to violate a previously satisfied constraint we must use the update given by (2.26). Finally, we only continue searching in the direction p if (2.29) is satisfied, i.e., if the penalty function can still be decreased in the direction p.

Remark 4. Some computational improvements [6] can be made on the updates of Gill and Murray [5], including iterative refinement, but these have been omitted so as not to obscure the statement of the main principles of the algorithm. We now consider the more general case when q is not necessarily equal to n.

To begin with we shall require some additional notation.

Let $\mathscr{A}$ be the index set for the active constraints (at the current point $x^{(i)}$) i.e.,

$$\mathscr{A} = \{j | a_j^T x^{(i)} = b_j\}.$$

Similarly let $\mathscr{S}$ be the index set for the inactive but satisfied constraints, i.e.,

$$\mathscr{S} = \{1, 2, 3, \cdots, m\} \backslash (\mathscr{A} \cup \mathscr{Q}\xi).$$

We now state the modified algorithm.

ALGORITHM 2. As for Algorithm 1 we are assuming that at the beginning of the ith iteration (i)–(viii) of the basic algorithm are available.

Step 1. Determine the Kuhn–Tucker multipliers u by solving the equations

$$L^{(i)} L^{(i)^T} u = d^{(i)}.$$

(a) If $u_j \geqq 0$, $j = 1, 2, \cdots, q$, and $\mathscr{Q}\xi = \Phi$, then $x^{(i)}$ is the optimal solution.

(b) If $u_j \geqq 0$, $j = 1, 2, \cdots, q$, and $\mathscr{Q}\xi \neq \Phi$, then we may conclude that μ is too large. Hence set $\mu \leftarrow \mu/10$ and return to Step 1, after first updating d^i and $\overset{*}{c}^{(i)}$ (cf. (vii) of the basic algorithm).

(c) If some $u_j < 0$, then choose index t such that $u_t = \min \{u_j : j = 1, 2, \cdots, q\}$.

Step 2. Choose p so that

$$\overset{*}{A}^{(i)T}p = 0,$$
$$\overset{*}{c}^{(i)T}p < 0,$$
$$\bar{A}^{T}p > 0,$$

where

$$A^{(i)} = [\overset{*}{A}^{(i)} : \overset{*}{\bar{A}}^{(i)}]$$

and $\mathcal{V} = \{v_1, \cdots, v_r\}$ is the index set for the columns of $\overset{*}{A}^{(i)}$. Furthermore, $t \notin \mathcal{V}$ if $q = n$: Note in general this requires reordering of $A^{(i)}$ – see Remark 1 on Algorithm 2.

Step 3. Order the λ_K's such that

$$\lambda_{K_1} \leqq \lambda_{K_2} \leqq \cdots \leqq \lambda_{K_j},$$

where

$$K_j \in \{j | j \in \mathcal{Q}\xi u\mathcal{S} \text{ and } \lambda_j > 0\}.$$

Here

$$\lambda_j = -w_j^{(i)}/v_j$$

and

$$v_j = \bar{A}^{(i)T}p.$$

Step 4. As in Algorithm 1 (but see Remark 3, below).

Step 5. Set

$$d_j^{(i+1)} = d_j^{(i)}, \qquad j \in \mathcal{V},$$
$$d_{r+1}^{(i+1)} = a_K^{T}\overset{*}{c}^{(i+1)},$$

where $\bar{a}_K$ is the corresponding Kth column of $\bar{A}^i$.

$$\text{Set} \quad q = r + 1.$$

Step 6. Update the residuals of the inactive constraints

$$w_j^{(i+1)} = w_j^{(i)} + \lambda_{(i)}v_j, \qquad 1 \leqq j \leqq m - n, \quad j \neq K,$$
$$w_K^{(i+1)} = \lambda_i.$$

Step 7. As in Algorithm 1.

Step 8. As in Algorithm 1.

Some remarks on Algorithm 2.

Remark 1. The direction p as chosen in Step 2 requires rearrangement of the columns of $A^{(i)}$.

If $r < [2q/3]$ Gill and Murray [5] recommend building up the active basis afresh. Otherwise the A^i can be rearranged as follows.

Suppose l is the first column in A^i such that $l \in \mathcal{V}$. Then the lth column is removed to become the qth column of $A^{(i)}$ with all the previous columns from $l+1$ to n being relabeled l to $n-1$. This is repeated until $l = r+1$ (similarly to the $b_i^{(i)}$).

Remark 2. One may readily choose p in Step 2 in a manner analogous to Conn and Pietrzykowski [4]. Following Gill and Murray [5] we shall take, using their notation,

$$p = p_c = \beta_c\{\overset{*}{A}{}^i(\overset{*}{A}{}^{(i)^T}\overset{*}{A}{}^{(i)})^{-1}\overset{*}{A}{}^{(i)^T}\overset{*}{c}{}^{(i)} - \overset{*}{c}{}^{(i)}\}$$

which we obtain as follows, whenever $t \notin \mathcal{V}$,

$$p_c = \beta_c\{A^{(i)}u - \overset{*}{c}{}^{(i)} - \frac{u_t}{\gamma}A^i y\},$$

where $L^{(i)}L^{(i)^T}y = e_t$ and γ is the (t, t)th element of $(A^{(i)^T}A^{(i)})^{-1}$.

Remark 3. Finally it is necessary to make one modification to Algorithm 1.1 as added in Step 4 to account for the possibility of the parameter μ being too large. The modification is as follows. Condition (2.30) becomes:

If $p^T\overset{*}{c}{}_j^{(i+1)} \leqq 0$ and λ_{K_j} does not correspond to the last constraint that it is possible to cross by moving in the direction p, then go to Step 2. Otherwise set $\mu \leftarrow \mu/10$ and return to Step 1, after first updating d^i and $\overset{*}{c}{}^{(i)}$.

It should be noted that it is easy to ascertain when the condition for μ to be reduced is realized since we already have all the λ_K's ordered.

3. Proof of convergence. First consider the basic Algorithm 1 with the modification on μ as indicated in the second algorithm. Under these conditions finite convergence is guaranteed as follows immediately from a specialization of the results of Conn [3], [4]. The algorithm ensures that μ will become sufficiently small in a finite number of iterations. Finite convergence to the optimum is certain because at each stage one proceeds from vertex to vertex (not necessarily feasible vertices) and there are only a finite number of constraints and hence only a finite number of vertices and forms of the penalty function.

As for Algorithm 1, convergence for Algorithm 2 is an immediate consequence of earlier results [3], [4]. Finite convergence, however, is not quite so easy to prove. In fact, without the following antizigzag rule, viz., "If a constraint that has previously been left is returned to, it is kept in the constraint basis until the minimal solution with those constraints active has been determined"; finite convergence has not been proved. However, I believe that zigzagging cannot occur for Algorithm 2 and consequently that finite convergence can be proved for the algorithm in its most general form.

4. Some comments with respect to efficiency and numerical experience. It is a major hope of this paper that the algorithm presented here is, in general, superior to the simplex algorithm. This optimism is based on the surmise that, in general, the number of pivots (changes of bases) is fewer than the simplex method. It is therefore necessary to point out computational requirements per pivot.

Gill and Murray [5] indicate that the average amount of work for a change of basis, using elementary Hermitian matrices for the updates [their method 3], is

$$\frac{9}{2}n^2 + n(m-n) + O(n).$$

Furthermore, the inclusion of less than n constraints in the projection reduces the

amount of work. If n' constraints are present, then the work reduces to

$$\frac{11}{6}(n')^2+2nn'+n(m-n')+O(n-n').$$

When average figures of $m=3n$ and $n'=n/2$ are assumed, the average work becomes

$$3\frac{23}{24}n^2+O(n).$$

Under the supposition that (2.30) is satisfied at least once (otherwise the analysis is identical to Gill and Murray's) we have an additional $n^2+O(n)$ multiplications in updating d [cf. (2.6)]. Thus for example, the average amount of work per change of basis for Algorithm 1, using elementary Hermitian matrices for the updates, is $11n^2/2+n(m-n)+O(n)$.

Analogously to Gill and Murray, we are able to make a direct comparison with the explicit inverse version of the simplex algorithm. Assuming

$$A^T=\begin{bmatrix}B\\I\end{bmatrix} \quad \text{and} \quad b=\begin{bmatrix}b\\0\end{bmatrix},$$

where B is an $r\times n$ matrix with $r<n$ and $m=n+r$. Then with an average figure of $n=3r$ the amount of work becomes

$$5r^2+O(r) \quad \text{multiplications.}$$

In contrast, using Gill and Murray's results for handling simple constraints (i.e., constraints of the form $\pm x_j \geqq b_j$), and letting r' correspond to the number of simple constaints that are not in the current basis, we obtain with an expected figure of $n=3r$ and $r'=r/2$,

$$3\frac{1}{12}r^2+O(r)$$

(i.e., an additional $r^2/4$ over Gill and Murray to account for the update on d).

Furthermore, if fewer than r constraints are in the projection matrix, then additional savings can be made. It should also be noted that if the update on d involves only simple constraints the amount of work per iteration is

$$2\frac{5}{6}r^2+O(r)$$

as for Gill and Murray.

Thus we have an algorithm that can be expected to require less work per change of basis than the simplex algorithm. What is perhaps more significant is that we can expect it to require fewer changes of basis in general to solve a given LP problem. Nevertheless, it should be pointed out that no theoretical results are yet available to substantiate this hypothesis. However, it is clear that the algorithm contained in this paper is a penalty function approach and as such does not require us to follow any boundaries. Moreover, the amount of work required to ascertain the direction of search and to minimize the penalty function is directly comparable

to the amount of work required to perform one iteration of the simplex algorithm. Consequently one might expect more efficiency. This expectation has been realized on the few problems solved by the author. About one dozen small LP's have been run, with the largest problem containing 8 variables and 16 constraints. An average saving of at least 20% has occurred. However, clearly much more computational experience is required.

It should be noted that the amount of work per iteration is greater than that of Gill and Murray, wherever an update in d is required. However, on the apparently reasonable assumption that, on the average, an update in d will indicate fewer changes of basis, we anticipate that the algorithm above is superior to theirs.

5. Comments and conclusions. An algorithm has been presented which is competitive with the standard simplex method. Moreover, the method is numerically stable. Also, because of the approach taken in this algorithm, i.e., that of nonlinear programming, the methods used here should be able to be extended readily to quadratic programming and to the minimization of a nonlinear function with linear constraints.

Acknowledgment. I am grateful to Walter Murray for his helpful discussions and advice.

REFERENCES

[1] R. H. BARTELS, *A numerical investigation of the simplex method*, Tech. Rep. CS104, Computer Science Department, Stanford University, Stanford, Calif., 1968.

[2] R. H. BARTELS AND G. H. GOLUB, *The simplex method of linear programming using LU decomposition*, Comm. ACM 12 (1969), pp. 266–268.

[3] A. R. CONN, *Constrained optimization using a nondifferentiable penalty function*, this Journal, 10 (1973), pp. 760–784.

[4] A. R. CONN AND T. PIETRZYKOWSKI, *A penalty function method converging directly to a constrained optimum*, Res. Rep. 73-11, Dept. of Combinatorics and Optimization, Univ. of Waterloo, 1973.

[5] P. E. GILL AND W. MURRAY, *A numerically stable form of the simplex algorithm*, Linear Algebra Appl., 7 (1973), pp. 99–138.

[6] W. MURRAY, Private communication, 1974.