Mathematical Programming 11 (1976) 128–149.
North-Holland Publishing Company

FINDING THE NEAREST POINT IN A POLYTOPE

Philip WOLFE
IBM Thomas J. Watson Research Center, New York, U.S.A.

Received 18 June 1974
Revised manuscript received 26 January 1976

A terminating algorithm is developed for the problem of finding the point of smallest Euclidean norm in the convex hull of a given finite point set in Euclidean n-space, or equivalently for finding an "optimal" hyperplane separating a given point from a given finite point set. Its efficiency and accuracy are investigated, and its extension to the separation of two sets and other convex programming problems described.

1. Introduction

We develop here a numerical algorithm for finding that point of a given polytope in Euclidean n-space which has smallest Euclidean norm. A polytope is the convex hull of a given point set $P = \{P_1, \ldots, P_m\}$; algebraically, we are required to minimize $|X|^2 = X^{\mathrm{T}}X$ for all X of the form

$$X = \sum_{k=1}^{m} P_k w_k, \qquad \sum_{k=1}^{m} w_k = 1, \quad \text{for all } w_k \geqslant 0. \tag{1.1}$$

The point X is thus the nearest point to the origin in the polytope. Solving the problem for $P = \{P_1 - Y, \ldots, P_m - Y\}$ yields $X + Y$ as the nearest point to Y; for simplicity we keep $Y = 0$ throughout.

That problem, or its dual, arises in many contexts. We have encountered it in the optimization of nondifferentiable functions, in approximation theory, and in pattern recognition, and have felt the need for an efficient and reasonably foolproof way of solving it. We offer what follows as an approximation to that goal.

The problem is, of course, a problem of quadratic programming, for which there are several excellent general algorithms; but the special nature of our problem lets us improve on them. The central feature of our approach is that the representation (1.1) of X is explicitly used, and the set of such X viewed as a polytope, while the usual general algorithm concentrates on the constraints $w \geqslant 0$, taking the description of the set as a polyhedron (the intersection of halfspaces) as fundamental. The algorithm to be described has been trimmed to the present problem; we think it improves on the general-purpose algorithms in computational effort, storage, and roundoff error. Other procedures we have seen (e.g., [2, 9]) are convergent, non-terminating methods, and cannot be properly compared with the present scheme.

The next section presents some elementary notions required to set the problem.

The algorithm is described in purely geometric terms in Section 3, and the corresponding algebraic description given in Sections 4 and 5. Section 6 presents results from tests on easily generated problems of several kinds, while Section 7 considers some possible improvements on the method. Section 8 studies the accuracy of the solution. Section 9 mentions some handy ways to use the method when a problem already solved is modified, and Section 10 describes the use of the algorithm to find "good" or "optimal" separating hyperplanes. Finally, Section 11 discusses some natural extensions of the algorithm to other constraint sets and objective functions.

We are indebted to Daniel Chazan for suggesting this study, and to him, Harlan Crowder, G.W. Stewart, and Christoph Witzgall for many helpful suggestions.

2. Preliminaries

Let $P = \{P_1, \ldots, P_m\}$ be a finite point set in a Euclidean space of dimension n. The smallest flat (translation of a linear subspace) containing P is called the *affine hull* of P,

$$A(P) = \left\{X: X = \sum_{j=1}^{m} P_j w_j, \sum_j w_j = 1\right\}. \tag{2.1}$$

The smallest convex body containing P is the *convex hull* of P,

$$C(P) = \{X: \text{(2.1) holds for some } w \geqslant 0\}.$$

In both cases, the vector $w = (w_1, \ldots, w_m)$ constitutes the *weights*, or barycentric coordinates, of the point X *in* P.

Letting the set P be represented (without ambiguity) by the n-by-m matrix P whose columns are the points P_j, we may write

$$A(P) = \{X: X = Pw, e^{\mathrm{T}}w = 1\},$$
$$C(P) = \{X: X = Pw, e^{\mathrm{T}}w = 1, w \geqslant 0\},$$

where e is the column vector $(1, 1, \ldots, 1)^{\mathrm{T}}$. (Throughout, the number of components of vectors such as e and w is to be inferred from the context. In this case, it is m for both. An inequality relating two vectors is understood to hold simultaneously for each component.)

A point set (or matrix) Q is *affinely independent* if no point of Q belongs to the affine hull of the remaining points. In that case, the weights expressing the point $X = Qw$, $e^{\mathrm{T}}w = 1$ as a member of $A(Q)$ are uniquely determined. Affine dependence is equivalent to the property that the $(n+1)$-by-k matrix

$$\begin{bmatrix} e^{\mathrm{T}} \\ Q \end{bmatrix}$$

(where k is the number of points of Q) has rank k, as well as to the property that the symmetric matrix of order $k+1$

$$\begin{bmatrix} 0 & e^{\mathrm{T}} \\ e & Q^{\mathrm{T}}Q \end{bmatrix} \tag{2.2}$$

have rank $k+1$, that is, be nonsingular. The sets $A(Q)$ and $C(Q)$ then have dimension $k-1$, and $C(Q)$ is a nondegenerate simplex whose vertices are the points of Q, while all the faces of dimension p of that simplex are the convex hulls of all the subsets of $p+1$ points of Q. The relative interior of $C(Q)$ (relative to the smallest affine set, $A(Q)$, containing it) is just the set of points whose weights in Q are all positive, and for any $X \in C(Q)$ there is a unique face having X in its relative interior: its vertices are those points for which the weight of X in Q is positive.

When Q is affinely independent, we can easily minimize $|X|$ on $A(Q)$, that is, solve the problem

$$\begin{aligned} &\text{Minimize} \quad |X|^2 = w^{\mathrm{T}}Q^{\mathrm{T}}Qw, \\ &\text{subject to} \quad e^{\mathrm{T}}w = 1. \end{aligned}$$

Forming the Lagrangian $w^{\mathrm{T}}Q^{\mathrm{T}}Qw + 2\lambda(e^{\mathrm{T}}w - 1)$ and differentiating, we have the necessary conditions

$$\begin{aligned} e^{\mathrm{T}}w &= 1, \\ e\lambda + Q^{\mathrm{T}}Qw &= 0, \end{aligned} \tag{2.3}$$

which have a unique solution owing to the nonsingularity of their matrix (2.2). Since the minimand is convex these conditions are also sufficient; $X = Qw$ minimizes $|X|$ on $A(Q)$.

Our algorithm will repeatedly produce such points X for selected sets Q. If it happens that $w \geqslant 0$, then of course X belongs to $C(Q)$ and minimizes $|X|$ there; and if $Q \subseteq P$ has been suitably chosen, X may even minimize over $C(P)$, and solve the original problem. The test for that, given below, amounts to determining whether the hyperplane

$$H(X) = \{x : X^{\mathrm{T}}x = |X|^2\} \tag{2.4}$$

is a supporting hyperplane of P, and hence of $C(P)$. Since $|X|^2$ is strictly convex, the point which minimizes $|X|$ over $C(P)$ is unique. We refer to it as

$$\mathrm{Nr}\, P.$$

Theorem 2.1 (Optimality). *$X \in C(P)$ is* $\mathrm{Nr}\, P$ *if and only if*

$$X^{\mathrm{T}}P_j \geqslant |X|^2 \quad \text{for all } j.$$

Proof. Let $Y \in C(P)$, $0 \leqslant \theta \leqslant 1$; then $X + \theta(Y - X) \in C(P)$ and $|X + \theta(Y - X)|^2 = |X|^2 + 2\theta(X^{\mathrm{T}}Y - X^{\mathrm{T}}X) + \theta^2|Y - X|^2$, which is less than $|X|^2$ for small θ unless $X^{\mathrm{T}}Y \geqslant X^{\mathrm{T}}X$. It follows that $X = \mathrm{Nr}\, P$ if and only if $X^{\mathrm{T}}Y \geqslant X^{\mathrm{T}}X$ for all $Y \in C(P)$, from which the theorem follows.

The next result is convenient for testing the quality of an approximation to $\mathrm{Nr}\, P$.

Theorem 2.2 (Estimation). *If* $0 \neq X \in C(P)$, *then*

$$|X| \geqslant |\mathrm{Nr}\, P| \geqslant \mathrm{Min}_j\, X^{\mathrm{T}} P_j / |X|. \tag{2.5}$$

Proof. Let $\mathrm{Nr}\, P = \Sigma_j \lambda_j P_j$, $\Sigma_j \lambda_j = 1$, all $\lambda_j \geqslant 0$. Then

$$\operatorname*{Min}_j X^{\mathrm{T}} P_j / |X| \leqslant \sum_j \lambda_j X^{\mathrm{T}} P_j / |X| = X^{\mathrm{T}}(\mathrm{Nr}\, P)/|X| \leqslant |\mathrm{Nr}\, P|.$$

The second inequality of (2.5) of course holds for any $X \neq 0$. Theorem 2.2 states just the useful part of a duality theorem for our problem, which we can state in greater generality for the discussion in Section 10.

Let $|\cdot|$ denote any norm (a positive-homogeneous finite-valued convex function vanishing only at the origin) on E^n, and $|\cdot|^*$ its dual norm

$$|y|^* = \mathrm{Max}\{y^{\mathrm{T}} x : |x| \leqslant 1\}. \tag{2.6}$$

Let

$$g(y) = \operatorname*{Min}_j\, y^{\mathrm{T}} P_j. \tag{2.7}$$

The problems

$$\mathrm{Min}\{|x| : x \in C(P)\}, \tag{2.8}$$

$$\mathrm{Max}\{g(y) : |y|^* \leqslant 1\} \tag{2.9}$$

are dual:

(i) $g(y) \leqslant |x|$ for all $x \in C(P)$, $|y|^* \leqslant 1$;

(ii) the extrema (2.8), (2.9) are equal;

(iii) among the solutions of the two problems are found all the saddlepoints of $y^{\mathrm{T}} x$, that is, all pairs $\bar{x}, \bar{y}$ such that $y^{\mathrm{T}} \bar{x} \leqslant \bar{y}^{\mathrm{T}} \bar{x} \leqslant \bar{y}^{\mathrm{T}} x$ for all $x \in C(P)$, $|y|^* \leqslant 1$.

(This duality is well known in approximation theory, but its first appearance eludes us.)

For the Euclidean norm we have the simplification that $|y|^* = |y|$, and (2.5) follows from the fact that the saddlepoint (iii) is given by $\bar{x} = \mathrm{Nr}\, P$ and $\bar{y} = \bar{x}/|\bar{x}|$ if $\mathrm{Nr}\, P \neq 0$, and by $\bar{x} = \bar{y} = 0$ otherwise.

3. The algorithm: geometry

We call an affinely independent subset Q of P a *corral* if $\mathrm{Nr}\, Q$ is in its relative interior. Note that any singleton is a corral. There is a corral whose convex hull contains the solution of the smallest-norm problem over P, and our algorithm will find it.

The algorithm consists of a finite number of *major cycles*, each of which may involve some number of *minor cycles*. At the start of each major cycle we have a corral Q and the point $\mathrm{Nr}\, Q$. The cycle consists in choosing a new point, adjoining it to Q, and determining whether the result is a corral. If so, the cycle is finished; if not, a minor cycle is begun. At the start of a minor cycle we have just

some affinely independent set Q and some point $X \in C(Q)$. Each minor cycle removes one selected point from Q and alters X. Minor cycles are repeated until a corral is found and $X = \mathrm{Nr}\, Q$, terminating the major cycle.

The initial corral is the singleton given by Step 0 below. Subsequently each major cycle begins at Step 1; the minor cycles, if any, in a major cycle constitute repetititions of Steps 3 and 2. Following the algorithm here we show how it works on the example of Fig. 1.

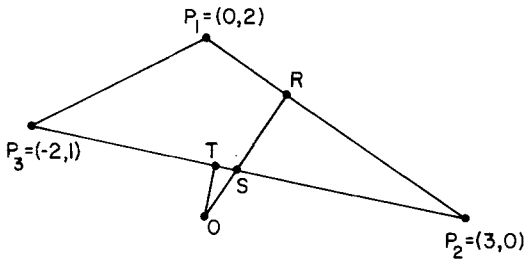

Fig. 1. Example.

Step 0. Find a point of P of minimal norm. Let X be that point and $Q = \{X\}$.

Step 1. If $X = 0$ or if $H(X)$ separates P from the origin, stop. Otherwise choose $P_J \in P$ on the near side of $H(X)$ and replace Q by $Q \cup \{P_J\}$.

Step 2. Let Y be the point of smallest norm in $A(Q)$. If Y is in the relative interior of $C(Q)$, replace X by Y and return to Step 1. Otherwise

Step 3. Let Z be the nearest point to Y on the line segment $C(Q) \cap \overline{XY}$ (thus a boundary point of $C(Q)$). Delete from Q one of the points not on the face of $C(Q)$ in which Z lies, and replace X by Z. Go to Step 2.

Fig. 1 and Table 1 illustrate the algorithm in a simple problem, giving the current Q, X, and Y at the end of each step.

We must show that the algorithm terminates in a solution of the problem. First,

Table 1
Solution of the example

Step	X	Q	Y
0	P_1	P_1	
1	P_1	P_1, P_2	
2	R	do.	R
1	R	P_1, P_2, P_3	
2	R	do.	0
3	S	P_2, P_3	
2	T	do.	T
1		STOP	

$R = (0.923, 1.385)$ is the nearest point to 0 on $\overline{P_1P_2}$.
$S = (0.353, 0.529)$ is the intersection of $\overline{OR}$ and $\overline{P_2P_3}$.
$T = (0.115, 0.577)$ is the answer.

observe that Q is always affinely independent: it changes only by the deletion of single points or by the adjunction of P_J in Step 1. Now the line $\overline{OX}$ is normal to $A(Q)$, since $|X|$ is minimal there; thus $A(Q) \subseteq H(X)$. Since $P_J \notin H(X)$, we know $P_J \notin A(Q)$, so $Q \cup \{P_J\}$ is affinely independent if Q is. Next, there can be no more minor cycles in a major cycle beginning with a given Q than the dimension of $C(Q)$, for when Q is a singleton, Step 2 returns us to Step 1 (indeed, the total number of minor cycles that have been performed from the beginning cannot exceed the number of major cycles). Every time Step 1 is followed by the replacement in Step 2 (the major cycle has no minor cycle) the value of $|X|$ is reduced, since the segment XY intersects the interior of $C(Q \cup \{P_J\})$ and $|X|$ strictly decreases along that segment. For the same reason the first minor cycle, if any, of a major cycle also reduces $|X|$, and subsequent minor cycles cannot increase it. Thus $|X|$ is reduced in each major cycle. Since X is uniquely determined by the corral on hand at Step 1, no corral can enter the algorithm more than once. Since there is but a finite number of corrals the algorithm must terminate, and it can only do so when the problem is solved.

The reader familiar with the Simplex Method for linear programming will notice its close relationship to our algorithm. More is said about this in Section 11.

4. The algorithm: algebra

We describe the algorithm of Section 3 algebraically. The points P_j are the columns of the n-by-m matrix P. The matrix for the corral Q is not explicitly formed; rather, we maintain the set of indices $S \subseteq \{1, \ldots, m\}$ designating those columns of P constituting Q. By $P[S]$ we mean the submatrix of P consisting of those columns, so that $Q = P[S]$. Note that w is a vector of varying length (one, in Step 0). The "notes" at the end of this section should be consulted.

Step 0. Defining J by $|P_J|^2 = \mathrm{Min}\{|P_j|^2, j = 1, \ldots, m\}$, set $S = \{J\}$ and $w = (1)$.

Step 1. (a) Set $X = P[S]w$.
(b) Define J by $X^{\mathrm{T}}P_J = \mathrm{Min}\{X^{\mathrm{T}}P_j, \text{ all } j\}$ (Note 1).
(c) If $X^{\mathrm{T}}P_J > X^{\mathrm{T}}X - Z_1 \,\mathrm{Max}\{|P_J|^2, \mathrm{Max}_{j \in S}|P_j|^2\}$, STOP. (Notes 2,7).
(d) If $J \in S$, then STOP (Note 3).
(e) If not stopped above, then replace S by $S \cup \{J\}$ and w by $(w, 0)$.

Step 2. (a) Solve the equations

$$e^{\mathrm{T}}v = 1, \qquad e\lambda + P[S]^{\mathrm{T}}P[S]v = 0 \tag{4.1}$$

for v. (Section 5 is devoted to that task.)
(b) If $v > Z_2 e$ (Note 4), set $w = v$ and go to Step 1. Otherwise,

Step 3. (a) Let POS be the set of indices $i \in S$ for which $w_i - v_i > Z_3$ (Note 5).
(b) Set $\theta = \mathrm{Min}\{1, \mathrm{Min}\{w_i/(w_i - v_i): i \in \mathrm{POS}\}\}$.
(c) Replace w by $\theta w + (1 - \theta)v$.
(d) Replace by zero all elements of w not greater than Z_2 (Note 6).
(e) Delete from w some zero component, and from S the corresponding index. Go to Step 2.

Notes. 1. This rule for choosing J is the handiest, and keeps the number of cycles low, as well as making approximate affine dependence of P_j on $P[S]$ unlikely. For large-scale problems there are likely better rules: See Section 7.

2. The term subtracted from $X^{\mathrm{T}}X$ is intended to make due allowance for roundoff error in the whole procedure. When $Z_1 = 0$ we have the optimality test of Theorem 1, Section 2. Calculating with sixteen-digit precision we use the generous value $Z_1 = 10^{-12}$. Without some such provision, the optimality test might never be passed.

At the stop in 1(c), X is Nr P, $P[S]$ is the corral which holds it, and w its barycentric coordinates in $P[S]$.

3. Stopping in 1(d) signals temporary disaster: X is so inaccurate that the relation $X^{\mathrm{T}}P_j = X^{\mathrm{T}}X$, which should hold for all $j \in S$, fails to more than the degree measured by Z_1. We have encountered this when P contained some nearly identical points, and generally find than that the stop 1(c) would have occurred if a modestly larger value of Z_1 had been used. That is a happy ending; see Section 8 regarding the effect of Z_1 on the final answer.

Some work would be saved by using only $j \notin S$ in 1(b) and omitting this step, but then some other kind of guard should be kept on the affine independence of Q. The check is also a fine trap for programming errors.

4. Another precaution. We use $Z_2 = 10^{-10}$.

5. We use $Z_3 = 10^{-10}$.

6. The value of θ given by 3(b) is the smaller of 1 and the largest value for which all components of $\theta w + (1-\theta)v$ are nonnegative. If $\theta < 1$, then at least one of those components will vanish. The replacement of 3(d) makes that vanishing decisive. We use the tolerance Z_2 here for consistency with Step 2(b).

7. Answers to well-formed optimization problems are subject to checks which are usually so easy it would be a sin to forgo them. Necessary and sufficient (and redundant) conditions that X, S, and $w \geqslant 0$ solve our problem are the vanishing of these four numbers:

(a) $\quad 1 - e^{\mathrm{T}}w,$

(b) $\quad |X - P[S]w|,$

(c) $\quad \operatorname*{Max}_{j \in S} |X^{\mathrm{T}}P_j - X^{\mathrm{T}}X|,$

(d) $\quad \operatorname*{Min}_{j} X^{\mathrm{T}}P_j - X^{\mathrm{T}}X$

(whose connection with the accuracy of the final solution is discussed in Section 8). If they do not approximate zero to a plausible degree, the algorithm has blundered. A possible fix would be to repeat the calculation, using instead of the matrix P just $P[S]$, and perform the tests again. We have had no experience with unsatisfactory performance when using the methods recommended in the next section.

8. A sometimes most useful feature of the algorithm is that after step 0 the point set P is consulted only in Step 1(b), and that step requires only one point P_J achieving $\mathrm{Min}\{X^{\mathrm{T}}P_j\}$. The set P may well be presented in some form other than that of a simple list, yet permit the step to be executed efficiently. An important

case occurs when P is actually the sum of two other sets: $P = C + D = \{c + d : c \in C, d \in D\}$. Although the number of points in P is the product of those in C and D, it is only necessary to look at each point of C and D once, for the desired member of P is just $\bar{c} + \bar{d}$, where $\bar{c}$, $\bar{d}$ minimize $X^{\mathrm{T}}P$ for $p \in C$ and $p \in D$, respectively. The problem of finding the shortest distance between two convex hulls allows such a formulation, which we exploit in Section 10; and some of the extensions of this algorithm considered in Section 11 likewise depend on such a "decomposition" of P.

5. Solving the equations

We have worked with the four ways A–D below of handling the equations (4.1), and concluded that D is the best. Let s be the number of indices of S – that is, the number of points in the current corral. By "operation" below we mean the execution of one floating multiply and one floating add. In making estimates, we give only the leading term of the polynomial constituting the accurate value. Storage estimates assume that symmetric matrices are stored without redundancy.

Method A. Maintain in storage the inverse E of the matrix of equations (4.1), modifying it when S changes. Initially

$$E = \begin{bmatrix} -P_J^2 & 1 \\ 1 & 0 \end{bmatrix}.$$

When S is increased in Step 1, E is replaced by

$$\begin{bmatrix} E + YY^{\mathrm{T}}/t & -Y/t \\ -Y/t & 1/t \end{bmatrix},$$

where $Y = EQ^{\mathrm{T}}P_J$ and $t = P_J^{\mathrm{T}}P_J^{\mathrm{T}} - P^{\mathrm{T}}QY$. (The "pivot" t is easily shown to be positive [W2]. It is the square of the distance of P_J from $A(P[S])$.) When S is decreased in Step 3, the appropriate column C and diagonal entry d of E are used to form $E - Cd^{-1}C^{\mathrm{T}}$, from which the new inverse results by dropping the same row and column. The solution of the equations is just the top row of E.

Method B. Maintain in storage the "tableau" for the quadratic programming problem (1.1). The tableau will contain E above and all the other data of the problem, suitably transformed by E. This approach has been discussed elsewhere [W14] for its pedagogical value, but its storage ($\frac{1}{2}m^2$) and arithmetic requirements ($\frac{1}{2}m^2$ operations per cycle) rule it out unless $m \ll n$, which is rare. Since it suffers from roundoff error even more than Method A, which is already in agony (see Section 8), we will not pursue it further.

For both procedures below it is efficient to find v of (4.1) by using these equations:

$$(ee^{\mathrm{T}} + P[S]^{\mathrm{T}}P[S])u = e, \tag{5.1}$$

$$v = u/e^{\mathrm{T}}u, \tag{5.2}$$

whose equivalence with (4.1) is easily checked. The order of the system (5.1) is one less than that of (4.1), and the normalization $e^{\mathrm{T}}v = 1$ is almost perfectly achieved.

Method C. The system (5.1) constitutes the normal equations $A^{\mathrm{T}}Au = A^{\mathrm{T}}b$ for the least-square solution of the equations $Au = b$, taking

$$A = \begin{bmatrix} e^{\mathrm{T}} \\ P[S] \end{bmatrix}$$

and $b = (1, 0, \ldots, 0)^{\mathrm{T}}$. In the interpretive language APL/360 the "domino" operator provides such a solution, and (5.1) is solved as

$$u \leftarrow b \boxed{\div} A.$$

This method is of course the most convenient to use in an APL program, and about as accurate as Method D below; in fact, the machine-language implementation of "domino" uses the arithmetic of Method D to compute u. Interestingly, in APL Method C is much quicker than D, although each time C is called, a number of D steps are taken; and it also uses slightly less time than Method A.

Method D. We maintain the matrix of the equations (5.1) in the form $R^{\mathrm{T}}R$, where R is an upper triangular matrix, doing the arithmetic in the manner suggested by Golub and Saunders' treatment of the least-squares problem [6].

We have always

$$R^{\mathrm{T}}R = ee^{\mathrm{T}} + P[S]^{\mathrm{T}}P[S], \tag{5.2}$$

and the solution of (5.1) is the result of solving the two systems

$$\begin{cases} R^{\mathrm{T}}\bar{u} = e \\ Ru = \bar{u}. \end{cases} \tag{5.3}$$

R must be altered whenever S is. The required work changes the algorithm of Section 4 as follows:

(i) Add to Step 0: Let R be the 1-by-1 matrix $[(1 + |P_J|^2)^{1/2}]$.

(ii) Add to Step 1:

(f) Solve for r the system

$$R^{\mathrm{T}}r = e + P[S]^{\mathrm{T}}P_J. \tag{5.4}$$

(g) Adjoin to R, on the right, the column $[r\,\rho]^{\mathrm{T}}$, where

$$\rho = (1 + P_J^{\mathrm{T}}P_J - r^{\mathrm{T}}r)^{1/2}.$$

(iii) Delete "Go to Step 2" from 3(e), and add to Step 3:

(f) Let I be the position of the component deleted in (e); delete the Ith column of R.

(g) If I exceeds the number of columns of (the new) R, go to Step 2. Otherwise let

$$a = R_{I,I}, \qquad b = R_{I+1,I}, \qquad c = (a^2 + b^2)^{1/2};$$

replace $R_{I,.}$ (row I of R) by $(aR_{I,.} + bR_{I+1,.})/c$ and $R_{I+1,.}$ by $(-bR_{I,.} + aR_{I+1,.})/c$; increase I by one, and repeat this step.

It is easy to check that (ii) above and (f) of (iii) maintain the relation (5.2). Step 2(g) uses plane rotations to restore R to upper triangular form after 2(f) has damaged it.

The storage needed for this scheme is the same as for A above. Solving a triangular system of s equations takes $\frac{1}{2}s^2$ operations, so (5.3) takes s^2. Equations (5.4) take $\frac{1}{2}s^2$, while the rotations of 3(g) take $2(s-I)^2$, which has the expected value $\frac{2}{3}s^2$ if I is randomly, uniformly chosen. Thus when S is increased (decreased), $\frac{3}{2}s^2(\frac{5}{3}s^2)$ operations are done, for an average of $\frac{19}{12}s^2$ in the long run.

We find that with this procedure there is virtually no accumulation of roundoff error, unlike A (see Section 8). Since the difference of $\frac{7}{12}s^2$ in the average operation counts of the two methods is swamped by the mn operations taken by Step 1(b) in the typical problem, we prefer D.

6. Experience

Although the amount of computational work done in a single step of the algorithm is well determined (see Section 5), we do not know how to estimate the number of iterations required. That number is of course bounded by the number of possible corrals in P, which is in turn bounded by the number of subsets of P having no more than $n+1$ elements, a calculable number; but the result is preposterously large. As with linear programming, we must resort to experiment to determine when the method is practical.

We have tested the algorithm on four types of problem. All of them start with a set P^0 of m points chosen at random, uniformly distributed over an n-cube of side 2 centered at the origin. (Draw mn integers at random without replacement from $1, 2, \ldots, 10^4$, divide by 5000 and subtract 1.)

Type 0: $P = P^0$.

Type 1: The point X^0 is chosen randomly from $2P^0$, and $P_j = X^0 + P_j^0$ for all j.

Type 2: P^0 is compressed by 10^{-3}, and then displaced one unit, along the x_1 axis:

$$P = \{(1 + 10^{-3}x_1, x_2, \ldots, x_n) : (x_1, x_2, \ldots, x_n) \in P^0\}.$$

Type 3: Like Type 2, but displaced by 0.01 instead:

$$P = \{(10^{-2} + 10^{-3}x_1, x_2, \ldots, x_n) : (x_1, \ldots, x_n) \in P^0\}.$$

For a problem of Type 0, Nr $P = 0$ is almost certain while for Type 1, Nr $P \neq 0$ is. Both problems are very easy; Type 0 since the origin is well-centered in P, Type 1 since the origin is likely to be near a "corner" of P and require only a few points of P to determine the solution. When $m > n$ a Type 0 problem almost always terminates in $n+1$ major, and no minor, cycles: the first corral of dimension $n+1$ constructed contains the origin. Table 6.1 gives the number of major and minor cycles for Type 1 problems of various size. In each case, the number is the average for a sample of ten problems.

Table 6.2 gives the same data for Type 3 problems. We omit the corresponding data for Type 2, since they can be generally described as requiring about 20%

fewer major cycles than do Type 3, and like Type 3 generally have terminal corrals of maximal size.

Note that the sum of the number of major and minor cycles is the total number of times a system of equations is solved. The difference of those numbers is the number of points in the terminal corral. The number of minor cycles can be viewed as a count of the "mistakes" made by the algorithm: the number of points selected which do not wind up in the terminal corral. When that number is zero, the amount of work the algorithm does in equation-solving in the updating versions A,B,D is simply what would be required to determine the weights w if the terminal corral were known in advance. The results tend to support our hunch that for large m the number of cycles increases roughly as $\log m$.

The graphs in Figs. 6.1–6.3 show the convergence of $|X^k|$ and the dual objective $g(X^k)$ (see Section 2) toward their common value $d = |\mathrm{Nr}\, P|$, as a function of the major cycle number k, for three problems having 80 points in E^{20}. Each of the problems 0,2,3 is of the corresponding type, built from the same P^0. The numbers of major cycles to termination were respectively 21,41,54, and minor cycles 0,21,34. The calculations were done using Method C of the previous section in a APL\360 routine on an IBM 360 Model 91. The CPU times required were 1.70, 4.63, and 6.63 seconds.

It appears that the convergence of $|X^k|$ to d is roughly linear, as is that of $g(X^k)$ when $d > 0$. (Note that the plotted points are subject to a discretization error of the width of one dot, and that since the terminal values of $|\cdot|$ and g were within 10^{-15} of d, those points are omitted from the logarithmic graphs.)

Table 6.1.
Cycles for Type 1 problems (major above minor)

$n\backslash m$	20	40	60	80	100	120	140	160
10	3.3	3.5	4.2	4.1	5.4	4.2	4.7	5.0
	0.0	0.2	0.2	0.2	0.1	0.1	0.2	0.6
20	4.2	5.1	5.2	4.6				
	0.1	0.2	0.1	0.1				
30	4.9	5.7						
	0.1	0.0						

Table 6.2.
Cycles for Type 3 problems

$n\backslash m$	20	40	60	80	100	120	140	160
10	14.6	20.2	24.1	25.5	26.0	28.8	30.5	30.7
	5.1	10.2	14.1	15.5	16.0	18.8	20.5	20.7
20	14.5	29.8	29.6	50.0				
	1.3	10.6	19.6	30.0				
30	16.2	27.1						
	1.1	3.3						

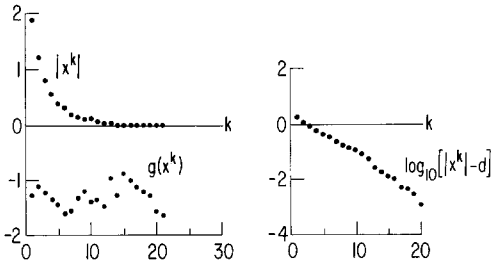

Fig. 6.1. Solving Problem 0.

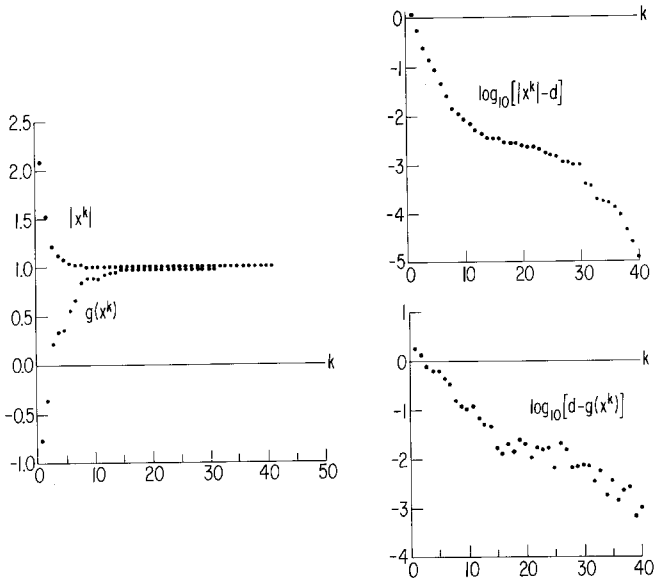

Fig. 6.2. Solving Problem 2.

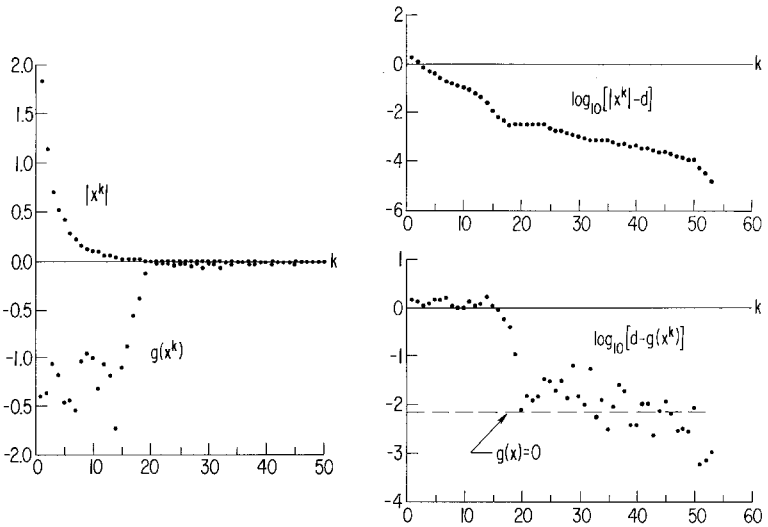

Fig. 6.3. Solving Problem 3.

7. Possible improvements

For a problem having a very large number m of points, almost all the computing time this algorithm requires will be taken in finding the new point P_J in Step 1(b). There are several possibilities for changing the selection criterion given in Section 4 to improve matters.

The first possibility considered is that the criterion used could be improved so that the number of major cycles might be reduced. Step 1(b) as given chooses P_J as that point on the same side of the separating hyperplane $X^{\mathrm{T}}x = X^{\mathrm{T}}X$ as the origin which has greatest distance from the hyperplane. We could, instead, either (i) determine how much reduction in $|X|$ would ensue in adjoining any P_j, (ii) estimate the latter quantity by finding the distance from the origin to the segment XP_j for all P_j, or more crudely (iii) choose P_j so as to maximize the angle between X and $P_j - X$. We deem (i) as too hard to be worthwhile, (ii) as interesting, and (iii) as easy enough to try out quickly, which is what we have done. The results, even for problems with $n = 5$, $m = 100$, were uniformly negative: there was a reduction of a few percent in the number of major cycles, but the extra calculation required almost doubled the total CPU time. A cheap approximation to this is (iv): Choose J to minimize $X^{\mathrm{T}}(P_j - X)/|P_j|$, having calculated $|P_j|$ once and for all. The amount of work is about the same as for the original method, but so is the number of iterations, on the average, for our problems.

We have not yet tried out the other two ideas worth mentioning, although their analogues have proved successful for linear programming problems with very large numbers of variables.

"Cyclic pricing": Only a portion of the points of P are examined before choosing P_J and performing the rest of the algorithm; then another portion is treated the same way; and so on, beginning again when all of P has been examined. Ultimately no candidate will be found in P, and we are done.

"Suboptimization": A small number of points of P are chosen after examining all of them – for instance, the points having the ten lowest values for $X^{\mathrm{T}}P_j$ – and the problem is completely solved on that subset; and this is done repeatedly.

Both procedures above are almost certain to increase the number of major cycles required by the algorithm, but hold promise of greatly decreasing the total work. At this time we have no more specific recommendation as to how to carry them out.

8. Accuracy

In most applications not much accuracy is required of the solution of our problem; but the answer obtained is quite accurate anyway. Here we show that the problem of finding Nr P is well-posed, and examine the round-off error in its solution.

The problem is well-posed if small changes in its data lead to small changes in its answer – i.e. if Nr P is continuous in P. That is the content of the Theorem below, due to G.W. Stewart.

Theorem 8.1. *Suppose that* $X = \mathrm{Nr}\, P$, $\bar{X} = \mathrm{Nr}\, \bar{P}$, *and*

$$|\bar{P}_j - P_j| < \delta \quad \textit{for all } j.$$

Then

$$\big||\bar{X}| - |X|\big| < \delta \tag{8.1}$$

and

$$|\bar{X} - X| < (4|X|\delta + \delta^2)^{1/2}. \tag{8.2}$$

Proof. Write $X = Pw$, $\bar{X} = \bar{P}\bar{w}$, with $e^{\mathrm{T}}w = 1$, $e^{\mathrm{T}}\bar{w} = 1$, $w, \bar{w} \geqslant 0$. Then $|\bar{X}| = |\bar{P}\bar{w}| \leqslant |\bar{P}w| \leqslant |Pw| + |(\bar{P} - P)w| < |X| + \delta$, and similarly $|X| < |\bar{X}| + \delta$, which proves (8.1).

For all j, $X^{\mathrm{T}}P_j \geqslant X^{\mathrm{T}}X$, whence $X^{\mathrm{T}}\bar{P}_j \geqslant X^{\mathrm{T}}X - \delta|X|$; thus $X^{\mathrm{T}}\bar{X} \geqslant X^{\mathrm{T}}X - \delta|X|$. Consequently $|\bar{X} - X|^2 = |\bar{X}|^2 - 2X^{\mathrm{T}}\bar{X} + |X|^2 < (|X| + \delta)^2 - 2(|X|^2 - \delta|X|) + |X|^2 = 4|X|\delta + \delta^2$, completing the proof.

The bound (8.1) is obviously sharp, and the example of Fig. 8.1 shows that (8.2) is not bad. We set $|P_1| = r$, $\overline{P_1P_2} \perp P_1$, $|P_2| = r + \delta$; $\bar{P}_1 = (1 + \delta/r)P_1$, $\bar{P}_2 = P_2/(1 + \delta/r)$. Then $X = \mathrm{Nr}\{P_1, P_2\} = P_1$, $\bar{X} = \mathrm{Nr}\{\bar{P}_1, \bar{P}_2\} = \bar{P}_2$, $|\bar{P}_1 - P_1| = |\bar{P}_2 - P_2| = \delta$, and $|\bar{X} - X| = 2r \sin \frac{1}{2}\theta = [2r^2(1 - \cos\theta)]^{1/2} = [2r^2\delta/(r + \delta)]^{1/2} = [2|X|\delta/(1 + \delta/|X|)]^{1/2}$.

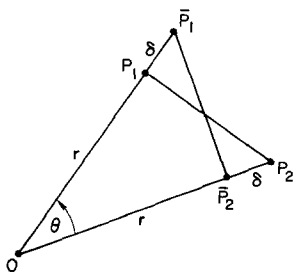

Fig. 8.1. Perturbation of P.

The theorem above permits us to make a simple *a posteriori* error analysis.

Suppose the computation has finished in Step 1(c), as it should, with $X \neq 0$. Let $B = \mathrm{Max}_j |P_j|$. Define

$$\begin{aligned} e_{\mathrm{a}} &= |e^{\mathrm{T}}w - 1|, \\ e_{\mathrm{b}} &= |X - P[S]w|/B, \\ e_{\mathrm{c}} &= \operatorname*{Max}_{j \in S} |X^{\mathrm{T}}P_j - X^{\mathrm{T}}X|/B|X|, \\ e_{\mathrm{d}} &= |\operatorname*{Min}_{j} X^{\mathrm{T}}P_j - X^{\mathrm{T}}X|/B|X|. \end{aligned} \tag{8.3}$$

Theorem 8.2 shows the role of these quantities in a backward error analysis, which is *a posteriori* because we do not know how to give a priori bounds for e_{c} and e_{d}.

Theorem 8.2. *Let P, S, etc. be as above, with $w > 0$, $X \neq 0$, and $e_a < 1$. Set $e = \mathrm{Max}\{e_c, e_d\}$.*

There is a point set $\bar{P}$ such that $X = \mathrm{Nr}\,\bar{P}$ and

$$\operatorname*{Max}_j |\bar{P}_j - P_j| < B(2e_a + 2e_b + e).$$

Proof. Let T be the set of all indices j belonging to S or such that $X^T P_j - X^T X < 0$. Set

$$\bar{P}_j = P_j e^T w + X - Pw - Xa_j,$$

where

$$a_j = X^T(P_j e^T w - Pw)/X^T X,$$

for all $j \in T$, and $\bar{P}_j = P_j$ for all other j. (The required P is readily deduced by altering w, and then P, so as to make the quantities of (8.3) vanish – in the order of their writing.) Let $\bar{w} = w/e^T w$. Then $\bar{w} > 0$, $e^T \bar{w} = 1$, $\Sigma\, \bar{w}_j a_j = 0$, $X = \bar{P}[S]\bar{w}$, and $X^T \bar{P}_j - X^T X = 0$ for $j \in T$ are all easily checked, showing that $X = \mathrm{Nr}\,\bar{P}$.

Now for $j \in T$,

$$|X|^2 a_j = X^T P_J(e^T w - 1) + X^T(P_j - X) + X^T(X - Pw),$$

so $|X| \cdot |a_j| \leq |P_j| e_a + Be + Be_b$. Further, for $j \in T$

$$\bar{P}_j - P_j = P_j(e^T w - 1) + X - Pw - Xa_j,$$

so

$$|\bar{P}_j - P_j| \leq |P_j| e_a + Be_b + |X| \cdot |a_j|,$$

from which the conclusion follows.

In practice the errors e_a, e_b are trifling compared with e, and e_c is small if the equations of the problem are well handled, so that $e_a = e_b = 0$, $e = e_d$ closely represents the normal situation. If the computation for the stop rule 1(c) has been done with reasonable accuracy we should have $\mathrm{Min}_j\, X^T P_j - X^T X \leq Z_1 B^2$, so that $e \leq Z_1 B/|X|$. By Theorem 8.2 the hypotheses of Theorem 8.1 hold with $\delta = Z_1 B^2/|X|$; we conclude that, where X^* is the exact solution of the problem,

$$\|X^*| - |X\| \leq Z_1 B^2/|X|$$

and

$$|X^* - X| \leq 2BZ_1^{1/2}$$

(ignoring the term δ^2 of (8.2)).

As mentioned in Section 4, something like the quantities of (8.3) should always be checked before a solution is accepted. In our routines the calculation of e_a and e_b is actually fallacious: e_b vanishes because the same arithmetic is used to form $X = P[S]w$ as to check it, and calculating e_a amounts largely to determining how accurately the machine represents number close to 1(in our case, to within about 10^{-16}).

Problems 2 and 3 of Section 6 have been run by Methods A and D, as well as by C used in Section 6. The terminal values of e_c and e_d are given in Table 8.1.

Table 8.1
e_c, e_d

Problem	Method A	D
2	1.5×10^{-10}, -1.5×10^{-10}	9.7×10^{-16}, -9.7×10^{-16}
3	abandoned	9.6×10^{-16}, -8.2×10^{-16}

Problem 3 did not terminate properly with Method A. At major cycle 48 the algorithm was abandoned when it calculated a negative pivot t. (Mathematically, $t>0$; $t<0$ is disaster, since the less inaccurate value $t=0$ destroys the logic of the algorithm. We might try to recalculate a better inverse, but there is no guarantee that we could do so using the Method A scheme.) At that point $B|X|e_c$ was 1.9×10^{-5}, a bad value.

Fig. 8.2 plots $\log_{10}(B|X|e_c)$ for the runs of Table 8.1. Method D seems to entail no growth of roundoff error, while Method A, probably in its minor cycles, undergoes exponential growth after a series of good cycles mostly having no minor cycles.

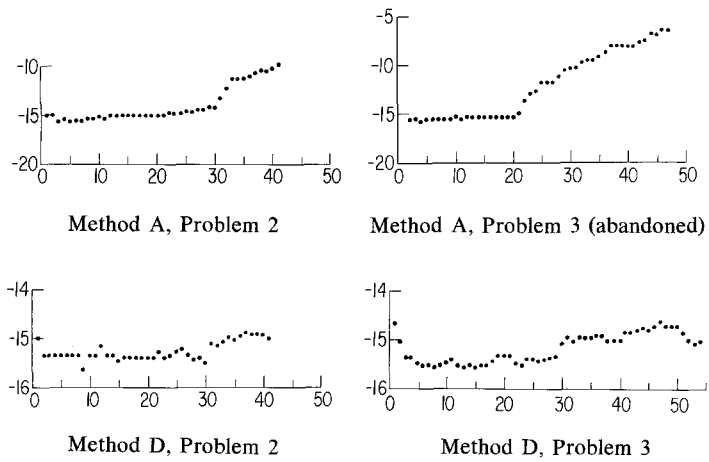

Fig. 8.2. Error in equation solving: $\log_{10}\mathrm{Max}_j\,|X^TP_j - X^TX|$.

9. Convenience features

In some applications the algorithm for Nr P is used repetitively: after a problem has been solved, P is altered by the adjunction or deletion of some points. It will not normally be necessary to go through the whole algorithm with the new set to solve the new problem, provided the data S and w are retained, as well as whatever information is maintained for the solution of the equations.

The routine we use can be entered in any of five modes, three of which are concerned with changing P. Modes 2–5 below assume the problem to have been solved, with S and w at their terminal values.

Mode 1. The entire problem is solved. The routine returns Nr P.

Mode 2. Points are adjoined to P. Here we need only begin the algorithm again at Step 1(a) (or at 1(b) if X has also been retained), and proceed as in Mode 1.

Mode 3. Points not in the terminal corral (i.e., whose indices are not in S) are deleted. Here we discard those points from P and alter appropriately the identifying tags of S; Nr P is unchanged.

Mode 4. Points in the terminal corral are deleted. If there are many such points it might be best to start the problem again from scratch. If not, then the existing information can be utilized by replacing S by S^* (those indices of S retained) and w by w^*, obtained from w by discarding the components in $S \backslash S^*$ and normalizing the result. The algorithm is then entered at Step 2 for method C of handling the equations; likewise for D, after deleting columns of R and restoring triangular form by rotations.

Mode 5. The quantities (a–d) of Note 7, Section 4 are calculated and returned for inspection of the errors.

10. Separating hyperplanes

As Theorem 2.1 shows, a nearest-point algorithm solves the dual problem of finding a hyperplane separating a point from a given finite set. (Hereafter the word "plane" will be used for "hyperplane".) Our procedure may be a fine way to construct planes, but many proposals have been made for that problem – both iterative, nonterminating methods (e.g. [2, 9]) and finite methods using linear or quadratic programming [3, 8] – and we have not attempted any comparisons; so we will deal with the subject only briefly.

Recall the dual objective function (2.7) $g(y) = \mathrm{Min}_j\, y^{\mathrm{T}} P_j$. We take $|y|^* = 1$. The plane

$$G(y) = \{x : y^{\mathrm{T}} x = g(y)\}$$

separates $C(P)$ from the origin when $g(y) \geqslant 0$, and strictly separates when $g(y) > 0$. The distance from the origin to $G(y)$ is

$$\begin{aligned} \mathrm{Min}\{|x| : y^{\mathrm{T}} x = g(y)\} &= \mathrm{Min}\{|x| : y^{\mathrm{T}} x \geqslant g(y)\} \\ &= \mathrm{Min}\{r : y^{\mathrm{T}} x r \geqslant g(y) \quad \text{for some } |x| = 1\} \\ &= \mathrm{Min}\{r : |y|^* r \geqslant g(y)\} = g(y), \end{aligned}$$

so the solution of the dual problem (2.9) $\mathrm{Max}\{g(y) : |y|^* \leqslant 1\}$ gives the "best" separation.

There are only two norms one would want to compute much with other than the Euclidean:

$$|x|_1 = \sum_j |x_j| \quad \text{and} \quad |x|_\infty = \mathrm{Max}_j\, |x_j|.$$

They are dual to each other ($|\cdot|_1^* = |\cdot|_\infty$, $|\cdot|_\infty^* = |\cdot|_1$), and using either of them our primal and dual problems can be stated as the same pair of dual linear

programming problems. Computationally, then, there is essentially one linear programming problem to be solved, and the simplex method is the right way to do that. Properly organized, the resulting algorithm looks much like that for the Euclidean norm.

We suspect that a good measure of difficulty, in number of cycles required, for the problem of finding a single (not the best) separating hyperplane, when one exists, is D/d, where D is the diameter of the projection along Nr P of $C(P)$ and $d = |\mathrm{Nr}\, P|$. In the experiments reported in Section 6, D was about $2(n-1)^{1/2}$; for problems 1,2,3 ($n = 20$) such D/d has the values 2.1, 8.7, 1300; the actual numbers k at which $g(x^k) > 0$ first happened were 1, 3, 33.

A common use of planes is in determining the separability of two point sets. We follow Canon and Cullum [3] in reducing that problem to a nearest-point problem, using these observations: (i) If two closed, compact convex sets C_1, C_2 are disjoint, then points $X \in C_1$ and $Y \in C_2$ can be found such that $|Y - X|$ is minimal, and any plane normal to $Y - X$ may be translated to separate the two sets; (ii) Such X, Y may be found as the resolution $Z = Y - X$ of the point $Z = \mathrm{Nr}\,[C_2 - C_1]$. (Note that $C_2 - C_1$ is defined as $\{y - x : x \in C_1, y \in C_2\}$, and that for any sets $Q, R, C(Q) - C(R) = C(Q - R)$. This reduction of the two-set problem to a one-set problem has been attributed to Caratheodory, but we have no citation.)

Let Q and R be two finite point sets. It is required to determine whether their convex hulls intersect and, if not, to find a well-separating hyperplane – that is, to find a vector X and numbers a, b such that $X^{\mathrm{T}}Q_j \geqslant b$ for all $Q_j \in Q$, $X^{\mathrm{T}}R_i \leqslant a$ for all $R_i \in R$, and $a < b$, with a and b well-separated in the sense that $(b - a)/|X|$ is large.

Let $g(y) = \mathrm{Min}_j\, y^{\mathrm{T}}Q_j - \mathrm{Max}_i\, y^{\mathrm{T}}R_i$. If $g(y) > 0$, then y is the normal of a plane separating Q from R, and the maximum of g for $|y| \leqslant 1$ will give the best separation. Since

$$g(y) = \operatorname*{Min}_{i,j}\, y^{\mathrm{T}}(Q_j - R_i),$$

it is the dual objective for the problem Nr P, setting

$$P = \{Q_j - R_i : \text{all } i, j\}.$$

Write

$$\mathrm{Nr}\, P = \sum_{i,j} (Q_j - R_i) w_{ij}, \qquad w_{ij} \geqslant 0, \qquad \sum_{i,j} w_{ij} = 1.$$

It is easy to check that

$$X_Q = \sum_{i,j} Q_j w_{ij} \in C(Q), \qquad X_R = \sum_{i,j} R_i w_{ij} \in C(R)$$

are the points of $C(Q)$, $C(R)$ for which $|X_Q - X_R|$ is minimized. $X = \mathrm{Nr}\, P = X_Q - X_R$ is the common normal to the two planes $G(X/|X|)$, $G(-X/|X|)$, which constitute the best separation of Q from R.

To apply the algorithm of this paper to the present problem efficiently one

avoids finding the set P explicitly, but works only with Q and R, as foreshadowed above (Section 4, Note 8). For Step 0 of the algorithm of Section 4 we choose a point of Q, find that point of R nearest it, then that point of Q nearest the chosen point of R, and so on. When no new points arise, P_1 is set to the difference of the last member of Q and the last member of R found. (This does not find Min $|P|$, but that is a hard task.) Throughout, the set S is a set of index-pairs (i, j); the first pair identifies the members of Q and R just chosen. In Step 1(b) we find Q_j minimizing $X^{\mathrm{T}}Q_j$ and R_i maximizing $X^{\mathrm{T}}R_i$, forming the new member of P as $Q_j - R_i$. The rest of the algorithm is essentially unchanged.

When Method C of Section 5 is used, it is appropriate to retain those members of P that have been constructed as long as they have positive weights w, and to discard them when their weights vanish in 3(d). In Method D it is not necessary to retain them at all once the inverse has been updated, so the storage requirements are the same as for the problem $\mathrm{Nr}\, Q_{\cup} R$. A few experiments support our intuition that when Q is separable from R the number of steps to find a separating plane is measured by D/d as above, and that the number to find the best plane is about the total required by the separate problems Nr Q, Nr R.

11. Variants and extensions

In this section we point out the relationship of our algorithm to the simplex method for linear programming and to the Frank–Wolfe method for convex programming, and how it can be extended to various types of constraint set and objective function.

The description of the algorithm in Section 3 parallels quite closely Dantzig's "Simplex interpretation of the simplex method" [4, Section 7.3], a visualization of his procedure for the linear programming problem $\mathrm{Min}\{c^{\mathrm{T}}w\colon w \geqslant 0, e^{\mathrm{T}}w = 1, Aw = b\}$ in the space of the vectors $P_j = [c_j; A_j]$, where $\{A_j\}$ are the columns of A. A "basis" for the linear programming problem corresponds to our corral, and a "change of basis" to a complete major cycle containing just one minor cycle. There are differences in detail, of course: unlike our problem, a basis must have n points in order to define a hyperplane by means of which the solution can either be improved or shown to be optimal; and in case of degeneracy a major cycle may not reduce the objective, so special provision must be made to avoid repetition of bases. Our experience has been that the two methods take similar numbers of steps to solve problems of the same size.

The procedure can also be viewed as extending the Frank–Wolfe algorithm. While its original presentation dealt with the minimization of a convex function over a polyhedron (i.e., a set explicitly defined as the intersection of halfspaces), it is best conceived as working with a polytope whose vertices are generated by the solution of linear programming problems in case the constraint set is a polyhedron. The present algorithm reduces to the Frank–Wolfe algorithm if the corral Q is required to be no more than one-dimensional. (The effect of such a restriction on the convergence rate is, of course, drastic.)

Gilbert [5] extended the Frank–Wolfe algorithm to the problem $\mathrm{Min}\{|x|\colon x \in S\}$,

where S is any convex set for which the "support" problem $\mathrm{Min}\{y^{\mathrm{T}}x : x \in S\}$ is readily solved, rather than just a polyhedron, and Barr [1] showed that relaxing the restriction to one dimension considerably improves matters. Barr solves a new quadratic programming problem each time a new point $x \in S$ is generated; it would seem that our algorithm should be used instead, with the support problem replacing our Step 1.

The basic idea of our algorithm seems worth using for some variants of our problem, although we have not had occasion to implement any of them. For a set of vectors $P = \{P_1, \ldots, P_m\}$ let

$$(*) \qquad L(P) = \left\{X : X = \sum_j P_j w_j\right\}$$

be the linear subspace spanned by P and

$$K(P) = \{X : (*) \text{ holds for some } w \geqslant 0\}$$

be the convex cone spanned by P. An algorithm for finding that point of $K(P)$ closest to a given point $R \in \mathrm{E}^n$ closely parallels the algorithm above. (Here $H(X)$ denotes the hyperplane $\{x : (X - R)^{\mathrm{T}}x = |X - R|^2\}$.)

For Step 0, find a ray $K(P_J)$ closest to R (as $\mathrm{Max}_j\, R^{\mathrm{T}}P_j/|P_j|$). Let X be the point of the ray closest to R, and set $Q = \{P_j\}$. The rest of the algorithm is exactly as for Nr P if "$A(Q)$" is replaced by "$L(Q)$" and "$C(Q)$" by "$K(Q)$" throughout. The algebraic algorithm for this problem is, in fact, a bit simpler than that for Nr P, but the numerical optimality criterion requires more thought. The resulting algorithm is almost exactly that introduced by Wilhelmsen [11] in connection with a certain approximation problem.

The most general problem of this kind is not much worse. Any polyhedron – the intersection of finitely many halfspaces – has the form $C(P^1) + K(P^2)$ for some finite sets P^1, P^2. In order to find the nearest point to R in that, we begin by doing Step 0 for both the "C" problem (when P^2 is empty) and the "K" problem (when P^1 is empty) and adding the results. In general one has $Q^1 \subseteq P^1$, $Q^2 \subseteq P^2$ ($Q^1 \cup Q^2$, a generalized corral, has at most $n + 1$ members), and performs Step 1 for either the C problem or the K problem. The remaining steps are as in the C algorithm, with $A(Q)$ replaced by $A(Q^1) + L(Q^2)$ and $C(Q)$ replaced by $C(Q^1) + K(Q^2)$. The algebraic version of this procedure will be very little different from that of the C algorithm, but again the numerical questions are important and unstudied.

When a polyhedron is presented as the intersection of halfspaces it is usually an enormous task to determine P^1, P^2 of the preceding paragraph explicitly. That could be avoided by proceding in the spirit of the original Frank–Wolfe algorithm (or of the "column generation" idea [13]): for each major cycle we require only the solution of $\mathrm{Min}_j (X - R)^{\mathrm{T}}P_j^i$ for $i = 1$ or $i = 2$, and such a solution will be the outcome of minimizing $(X - R)^{\mathrm{T}}x$ over the polyhedron using the simplex method. Such a procedure might compete with the usual quadratic programming algorithms for problems of small dimension with many constraints, but we think that the following ingenious reformulation, due to Witzgall [12], is the right way to proceed. It rests on the fact [10, p. 55] that if K is a convex cone and K^p its dual (or negative polar: $K^p = \{z : y^{\mathrm{T}}z \leqslant 0 \text{ for all } y \in K\}$), x is any point, and y and z

are respectively the nearest points to x in K and K^p, then y and z are orthogonal and $x = y + z$. In other words,

$$(**) \qquad x = x + \mathrm{Nr}\,(K - x) + x + \mathrm{Nr}\,(K^p - x)$$

for any x. Now let R be the polyhedron in n-space defined by the inequalities $A_k^{\mathrm{T}} x + b_k \leqslant 0$, $k = 1, \ldots, K$. Letting B_k be the $(n+1)$-vector $(A_k; b_k)$, define the convex cone in $(n+1)$-space

$$K = \{(x; t) \colon B_k^{\mathrm{T}}(x; t) \leqslant 0, \text{all } k\};$$

then $R = \{x \colon (x; 1) \in K\}$. If R is nonempty and $(y; s)$ is the nearest point in K to $(0, \ldots, 0; 1)$, it is easy to show that $0 < s \leqslant 1$ and that $y/s = \mathrm{Nr}\,R$. The cone K^p is exactly the set of all nonnegative linear combinations of B_k, so that our algorithm for cones can be applied to find $\mathrm{Nr}\,(K^p - (0, \ldots, 0; 1))$, and $(**)$ immediately gives the answer to the same problem for K, from which $\mathrm{Nr}\,R$ is determined.

The algorithm can also in principle be extended to the minimization of any differentiable convex function f on a polytope. Step 1 of the algorithm is then the task of minimizing $\nabla f(X)^{\mathrm{T}} P_j$, and the rest of the algorithm is the same provided one has a means of finding a satisfactory approximation to the minimum of f over any affine set $A(Q)$. The required algebra is easily done when f is a strictly convex quadratic function; the ensuing algorithm differs from that of Section 4 only in the equations to be solved. The general scheme, in connection with generation of vertices of the polytope when it is presented as a polyhedron, has been used in the interesting work of Von Hohenbalken [7], who gives an algorithm, based on the same ideas we have used, for the minimization of any pseudo-convex function over a bounded polyhedron. He has also applied the method to the minimization of a certain nondifferentiable convex function. It would be fascinating to know under what conditions this kind of method is competitive with other schemes for convex programming.

References

[1] R.O. Barr, "An efficient computational procedure for a generalized quadratic programming problem", *SIAM Journal on Control* 7 (1969) 415–429.

[2] L.C. Barbosa and E. Wong, "On a class of iterative algorithms for linear inequalities with applications to pattern classification", in: *Proceedings of the first annual Princeton conference of information sciences and systems*, 1967, pp. 86–89.

[3] M.D. Canon and C.D. Cullum, "The determination of optimum separating hyperplanes I. A finite step procedure", RC 2023, IBM Watson Research Center, Yorktown Heights, New York (February, 1968).

[4] G.B. Dantzig, *Linear programming and extensions* (Princeton University Press, Princeton, N.J. 1963).

[5] E.G. Gilbert, "An iterative procedure for computing the minimum of a quadratic form on a convex set", *SIAM Journal on Control* 4 (1966) 61–80.

[6] G.H. Golub and M.A. Saunders, Linear least squares and quadratic programming, in: J. Abadie, ed., *Integer and nonlinear programming* (North-Holland, Amsterdam, 1970) Ch. 10.

[7] B. von Hohenbalken, "A finite algorithm to maximize certain pseudoconcave functions on polytopes", *Mathematical Programming* 9 (1975) 189–206.

[8] O.L. Mangasarian, "Linear and nonlinear separation of patterns by linear programming", *Operations Research* 13 (1965) 444–452.
[9] B.F. Mitchell, V.F. Demyanov, and V.N. Malozemov, "Finding the point of a polyhedron closest to the origin", *SIAM Journal on Control* 12 (1974) 19–26.
[10] J. Stoer and C. Witzgall, *Convexity and optimization in finite dimensions* I (Springer, Berlin, 1970).
[11] D.R. Wilhelmsen, A linear algorithm for generating positive weight cubatures, *Mathematics of Computation* 30 (1976) to appear.
[12] C. Witzgall, correspondence with P. Wolfe (December, 1974).
[13] P. Wolfe, "Convergence theory in nonlinear programming", in: J. Abadie, ed., *Integer and nonlinear programming*, (North-Holland, Amsterdam, 1970) pp. 1–36.
[14] P. Wolfe, "Algorithm for a least-distance programming problem", *Mathematical Programming Study* 1 (1974) 190–205.