Internat. J. Math. & Math. Sci.
VOL. 16 NO. 2 (1993) 209-224

A CENTER OF A POLYTOPE: AN EXPOSITORY REVIEW AND A PARALLEL IMPLEMENTATION

S.K. SEN, HONGWEI DU, and D.W. FAUSETT

Department of Applied Mathematics
Florida Institute of Technology
Melbourne, FL 32901

(Received March 1992)

ABSTRACT. The solution space of the rectangular linear system $Ax = b$, subject to $x \geq 0$, is called a polytope. An attempt is made to provide a deeper geometric insight, with numerical examples, into the condensed paper by Lord, et al. [1], that presents an algorithm to compute a center of a polytope. The algorithm is readily adopted for either sequential or parallel computer implementation. The computed center provides an initial feasible solution (interior point) of a linear programming problem.

KEY WORDS AND PHRASES. Center of a polytope, consistency check, Euclidean distance, initial feasible solution, linear programming, Moore-Penrose inverse, nonnegative solution, parallel computation.

1991 AMS SUBJECT CLASSIFICATION CODES. 15A06, 15A09, 65F05, 65F20, 65K05.

1. INTRODUCTION.

The solution space ω of a linear system $Ax = b$, where A is an $m \times n$ matrix of rank r, will

- (*i*) be empty if and only if (iff) the system is inconsistent, i.e., iff $r \neq \text{rank } (A, b)$,
- (*ii*) have a unique point (solution) iff $r = \text{rank}(A, b) = m = n$, and
- (*iii*) have infinite points (solutions) or, equivalently, an $n - r$ parameter solution space iff $r = \text{rank}(A,b) \leq m$ when $m < n$ or $r = \text{rank}(A,b) < n$ when $m \geq n$.

A linear system, unlike a nonlinear system, cannot have a solution space having just two solutions or any other finite number of solutions except one. In Case (*i*), the corresponding polytope (defined here by $Ax = b$, $x \geq 0$) is nonexistent, while in Case (*ii*) the polytope will be the (0-dimensional) unique point iff the point x satisfies the condition $x \geq 0$. In both these cases the proposed centering algorithm achieves the same result as that accomplished by an efficient linear equation solving algorithm. In Case (*iii*), the algorithm computes a center of the polytope iff such a center exists. It is readily seen that the polytope is convex. The paper by Lord, et al. [1] suggests a new definition of a 'center' of a convex polytope in Euclidean space - a point is a center of a convex polytope in Euclidean space if it is the center of a sphere that lies within the polytope and it touches a set of bounding hyperplanes that have no common intersection. In general, a center of a polytope in $q = n - r$ dimensional space will be

the center of a sphere touching $q+1$ or more bounding hyperplanes. Degenerate cases correspond to fewer than $q+1$ hyperplanes 'meeting at infinity'. A polytope does not, in general, have a unique center according to the foregoing definition. Even if a uniqueness is defined in some sense, i.e., by imposing certain conditions and thereby increasing complexity, such a uniqueness may introduce nonlinearity, and may not achieve much in practice if one is going to solve linear programming (lp) problems using this unique center. In fact, the centering algorithm has sacrificed uniqueness in favor of preserving linearity and computational simplicity. A center of a polytope always exists if the polytope is a finite (bounded) region of the solution space; and in most applications, uniqueness is not needed. As an example, any 'center', as defined in paper [1], is a good interior (starting) point to solve an lp problem. However, a center is usually defined uniquely as the extremum of a potential function that vanishes on the boundary [2,3,4]. Such a definition adopts a nonlinear concept into what is essentially a linear system. Further, it does not readily adapt itself to computation; in practice, methods based on this definition focus on obtaining just an approximation to the center. The method of finding a center as described here not only encounters no such difficulty but also is simple and straightforward to implement. It provides a very easy means of obtaining an initial feasible solution for an lp problem without enhancing the dimension of the problem, i.e., without introducing artificial variables to check consistency. The intersection of the hyperplanes represented by $Ax = b$ is an $n-r$ dimensional space. It is a point if $n-r=0$. It is a line if $n-r=1$. It is a 2-dimensional plane if $n-r=2$, while it is a q-dimensional hyperspace if $n-r=q$.

In Section 2 we describe the centering algorithm, while in Section 3 we discuss the geometry of the algorithm considering the more basic linear inequality problem, and justify its validity through the properties of n-dimensional Euclidean space. Section 4 presents two examples with comments primarily aimed as an aid for easy computer implementation. Section 5 discusses considerations for parallel implementation of the algorithm, while Section 6 consists of concluding remarks.

2. THE CENTERING ALGORITHM

Consider the linear equations and inequalities

$$Ax = b,\ x \geq 0, \tag{1}$$

where $A = [a_{ij}]$ is an $m \times n$ matrix of rank r, $b \in R_m$, $x \in R_n$. We write

$$A = \left[a_1^t\ \ a_2^t\ \ \dots\ \ a_m^t\right]^t,\ b = \left[b_1\ \ b_2\ \ \dots\ \ b_m\right]^t,\ x = [x_1\ \ x_2\ \ \dots\ \ x_n]^t,$$

where $a_i^t = [a_{i1}\ \ a_{i2}\ \ \dots\ \ a_{in}]$ is the ith row of A and t indicates the transpose. Observe that $a_i^t x = b_i$ represents the ith hyperplane of the linear system (1). The steps of the centering algorithm are then as follows.

S.1 Compute $x = A^+ b$, $P = (I - A^+ A)$, and r by *Algorithm 1.*

S.2 Set $q = n - r$; if any diagonal elements of $P = [p_{ij}]$ vanish, then remove the zero rows and zero columns of P and the corresponding coordinates (viz., the corresponding hyperplanes) from the problem $Ax = b$.

S.3 *Normalize*: $D = diag(1/\sqrt{p_{ii}})$; $S = DP$; $\delta = Dx$.

S.4 Apply *Algorithm 2.*

2.1 ***Algorithm 1*** (*Computing* x, P, *and* r *from the given eqn.* $Ax = b$):

The algorithm [5] is an $O(mn^2)$ direct algorithm to compute the minimum-norm least squares solution $x = A^+b$ of a system of m linear equations, $Ax = b$, with n variables, where m and n can be any positive (finite) integers, the vector b can be zero or nonzero, and A^+ denotes the Moore-Penrose inverse of A. It is concise and matrix-inversion-free. It provides a built-in consistency check, and also produces the rank r of the matrix A. Further, if necessary, it can prune the redundant rows of A and convert A into a full row-rank matrix, thus preserving the complete information of the system. In addition, the algorithm produces the unique projection operator, $P = I_n - A^+A$, that projects the real n-dimensional space orthogonally onto the null space of A and that provides a means of computing a relative error-bound for the solution vector [6,7] and any other solution of the system. The general solution of $Ax = b$ is $x = A^+b - Pz$, where z is an arbitrary vector. Observe that if $b = 0$ (zero column vector) then $x = Pz$.

The algorithm involves no taking of square-roots and a finite number of arithmetic operations; so, it can be readily implemented by error-free (residue) arithmetic [7] to compute x, P and r exactly. In addition, a parallel implementation is easily achieved since the algorithm involves mainly vector operations. In fact, a parallel version of this algorithm has been implemented on an Intel i860 computer.

Consider the equation $Ax = b$ of relations (1). A pseudocode for the algorithm is as follows.

{*Algorithm 1*}

```
begin
   P := I_n ; x := 0 ; r := 0 ;
   for j = 1 to m do
      STEP ( a_j , b_j , P , x , ind ) ;  r := r + ind ;
   end-do
end
procedure STEP ( a , b , P , x , ind ) ;
begin
   ind := 0 ; v := Pa ; y := ||v||^2 ; b := b - a^t x ;
   if (y ≠ 0) then
      P := P - vv^t/y ; x := x + bv/y ; ind := 1 ;
```

```
        else
          if b ≠ 0 then
            terminate ; {error message : = "inconsistent equations"}
          end-if
        end-if
    end
```

The procedure *STEP* computes a normal to the hyperplane represented by the equation $\boldsymbol{a}^t\boldsymbol{x} = b$. After m successive executions of *STEP*, the resulting vector $\boldsymbol{x}$ will be normal to the common intersection of the m hyperplanes represented by $\boldsymbol{a}_j^t\boldsymbol{x} = b_j$ and consequently a solution.

The outputs of *Algorithm 1* are a particular solution $\boldsymbol{x} = A^+\boldsymbol{b}$, known as the minimum-norm least squares solution, of $A\boldsymbol{x} = \boldsymbol{b}$, the rank r (of A), and the projection operator $P = I_n - A^+A$. No explicit computation for the Moore-Penrose inverse [8], denoted by A^+, is required for computing $\boldsymbol{x}$ and P. The solution $\boldsymbol{x}$, in general, does not satisfy the nonnegativity constraint $\boldsymbol{x} \geq \boldsymbol{0}$.

Whenever $y = 0$ and $b = 0$ occur in the procedure *STEP*, the current row of A and the current element (row) of $\boldsymbol{b}$ can be deleted, the following rows of A and $\boldsymbol{b}$ can be popped up, and the number of rows m can be reduced by 1 if A is to be converted to a full row-rank matrix. However, the column-order n of P remains unchanged. To obtain a solution of homogeneous equations $A\boldsymbol{x} = \boldsymbol{0}$, i.e., when $\boldsymbol{b} = \boldsymbol{0}$, execute *Algorithm 1* and then compute $\boldsymbol{x} = P\boldsymbol{z}$, choosing any arbitrary zero or nonzero vector $\boldsymbol{z}$.

Algorithm 1 combines the desirable stability properties of an orthogonal transformations approach with a computational scheme that is well-suited for error-free computation, since the 'intermediate number growth' to any finite extent has no effect on an error-free arithmetic [6,7]. The computationally sensitive steps in the algorithm are those involving division by $\|\boldsymbol{v}\|^2$. Although a test is made to guarantee that $\|\boldsymbol{v}\| \neq 0$, it is possible for $\|\boldsymbol{v}\|$ to be arbitrarily small. When $\|\boldsymbol{v}\|$ is small, it indicates that the current vector $\boldsymbol{v}$ is almost linearly dependent on its predecessors. Such ill-conditioning can cause a severe loss of accuracy in the computed solution, if error-free arithmetic is not used. A thorough discussion of computational issues in solving least-squares problems is presented in Golub and Van Loan [9].

In case of an ill-conditioned system, it may be worth the additional computational effort to modify Algorithm 1 to incorporate row pivoting, so that $\|\boldsymbol{v}\|$ is maximized over the remaining rows at each step. Also the test for $\|\boldsymbol{v}\| = 0$ could be modified to a tolerance test

of the form $\| v \| < tol$, where *tol* is a machine dependent parameter based on the precision of arithmetic operations on a particular computer. For extremely ill-conditioned systems, this method of detecting near rank-deficiency is not always sufficient.

Another alternative in extreme cases is to use some method other than Algorithm 1 to computer x, r, and P before preceding to Algorithm 2. The most widely used method for treating nearly rank-deficient cases is based on the singular value decomposition, or SVD [9]. Rank-revealing methods of orthogonal decomposition is a general area of active research at present [10, 11, 12].

The computational costs of the various approaches described to computing the inputs to Algorithm 2 are presented below.

Solving $Ax = b$ for $x = A^+ b$, P $= I - A^+ A$, and $r =$ rank (A).

Method	Floating Point Operations
Algorithm 1	$m(4n^2 + 7n)$
Modified Algorithm 1 with row pivoting	$m(m+3)n(n+1)$
SVD	$8mn(m+n) + 9n^3$

2.2 ***Algorithm 2*** (*Computing a center from given S and δ*):

Having obtained P and the particular solution $x^p = x$ from *Algorithm 1*, the role of the original matrix A and original vector b in $Ax = b$ is over. We no longer need A and b to compute a center. The inputs of *Algorithm 2* are the particular solution $x^p = A^+ b$, the normalized projection matrix $S = DP$, and the vector of Euclidean distances $\delta = Dx^p$, where $D = diag\left(1/\sqrt{p_{ii}}\right)$ is a diagonal matrix, p_{ii} being the (i,i)-th element of P. Observe that P is (symmetric) idempotent, i.e., $P^2 = P$, and $p_{ii} \geq 0$. Algebraically, the general solution x^g of $Ax = b$ may be given by $x^g = x^p - Pz$, where z is an arbitrary vector; observe that this equation is essentially an identity since it is valid for whatever value we may substitute for the vector z. It can be seen that the equation $Ax = b$ is equivalent to the foregoing equation (identity) which has preserved all the information of $Ax = b$. Let $e = [1 \ \ 1 \ \ \ldots \ \ 1]^t$ be an n-vector and $S = [s_1^t \ \ s_2^t \ \ \ldots \ \ s_n^t]^t$, where s_i^t is the ith row vector of S. We set $Dx^g = -e$ so that the foregoing identity turns out to be an equation in the unknown variable (vector) z. Thus, we have the equation $Sz - \delta = e$. Geometrically, since $\| s_i \| = 1$, the component δ_i of δ is the Euclidean distance of the point x^p from the ith hyperplane $s_i^t z = 1$ of the system of n hyperplanes represented by $Sz = e$. (See *Property* 2 in Section 3). With this background it is easier to appreciate *Algorithm 2* which is as follows.

```
                                        {*Algorithm 2*}
begin
  β := min {i : δ_i = min {δ_k} } ; J := {β} ; j := 1 ;
  Q := I_n ; z := 0 ; ind := 1 ;
  while ( j ≠ q+1 and ind = 1 ) do
    STEP(s_β , 1 , Q , z , ind) ;
    if ind = 1 then
      LAMDA ;
      if ind = 1 then
        j := j+1 ;
      end-if
    end-if
  end-do
end
procedure LAMDA ;
  ind := 0 ; α := D z ;
  for i ∈ J do
    if α_i ≠ α_β then
      λ_i = (δ_i − δ_β)/(α_β − α_i) ;
      if λ_i ≥ 0 then
        if ind = 0 then
          k := i ; ind := 1 ;
        else
          if λ_i < λ_k then
            k := i ;
          end-if
        end-if
      end-if
    end-if
  end-do
  if ind = 1 then
    β := k ; λ := λ_β ; δ := δ + λα ;
    x := x + λz ; J := J ∪ {β} ;
  end-if
```

end

The procedure *LAMDA* computes λ so that $x + \lambda z$ will be equidistant from the current j hyperplanes and one more. *Algorithm* 2 provides a center which is an initial feasible solution for an *lp* problem. Observe that this center is not a Chebyshev point nor is it, in general, unique.

3. GEOMETRY OF THE CENTERING ALGORITHM

Normalized inequalities $Sz \geq e$ $(z \geq 0)$, where now S is an $m \times n$ matrix of rank r, $z \in R_n$, and $e \in R_m$ specify a convex polytope in n dimensional Euclidean space or, simply, n-space bounded by m hyperplanes; some of these hyperplanes may be redundant. The components δ_i of $\delta: = Sx^p - e$ are then the Euclidean distances of the given point x^p from each hyperplane, provided the sign convention is adopted that δ_i is negative if the ith hyperplane separates the point x^p from the polytope.

Let the point x_0 be in the polytope $(\delta \geq 0)$. Then the following procedure will find a center provided the polytope is bounded. Let $\sigma = \min_i\{\delta_i\}$ be the least distance of x_0 from j hyperplanes. If these j hyperplanes do not have a common intersection then x_0 is a center. Otherwise, a line through x_0 perpendicular to the common intersection is a line such that every point of the line is equidistant from the j hyperplanes. Start from x_0 and move along this line in the direction of increasing σ until a point is reached, which is equidistant from $j+1$ hyperplanes, i.e., until the inscribed sphere has been expanded so as to touch one more bounding hyperplane. This must happen if the polytope is bounded. A center is obtained by iterating this procedure until the set of encountered hyperplanes fails to have a common intersection, or $j = q + 1 = n - r + 1$.

The validity of the centering algorithm is based on the following properties [1].

Property 1: The vector s is normal to the hyperplane $s^t z = u$ and is directed into the region $s^t z > u$.

A tangent v to the hyperplane has the form $v = z_1 - z_2$, where z_1 and z_2 are points on the hyperplane. Therefore $s^t v = 0$; so s is normal. If z_1 is on the hyperplane and $\lambda > 0$, then $z = z_1 + \lambda s$ satisfies $s^t z = u + \lambda \| s \|^2 > u$; so the direction of s is as stated.

Property 2: If $\| s \| = 1$ then the Euclidean distance of an arbitrary point y from the hyperplane $s^t z = u$ is $\delta = s^t y - u$ (δ is taken to be negative if $s^t y < u$).

Let z be the orthogonal projection of y on the hyperplane. By *Property* 1, $y = z + \delta s$,

where δ is the signed distance. Hence $s^t y = s^t z + \delta s^t s = u + \delta$.

Property 3: If N is a matrix whose rows are the j unit normals to j given hyperplanes then the particular solution (vector) $z = N^+ e$ of the equation $Nz = e$ is normal to their intersection, where e is as defined before.

If v is tangential to all the j hyperplanes (i.e., if v is a tangent to the intersection of these j hyperplanes) then $Nv = 0$. Therefore, $v^t z = v^t N^+ e = v^t N^t (NN^t)^- e = (Nv)^t (NN^t)^- e = 0$, where A^- is a generalized inverse of A; A^- satisfies the condition $AA^- A = A$.

Property 4: If a point x is equidistant from j hyperplanes $Nx = w$ then so is $x' = x + \lambda z$, where $z = N^+ e$ and λ is arbitrary.

By *Property* 2, if x is at a distance σ from each of the j hyperplanes then $Nx - w = \sigma e$, and the distances of x' from the hyperplanes are given by the components of $\delta = Nx' - w$. Since $Nz = e$, $\delta = (\sigma + \lambda)e$; so, x' is at the same distance $\sigma + \lambda$ from all of the j hyperplanes.

Algorithm 2, the centering algorithm, now follows from these properties. Upon completion of the 'jth iteration (in *Algorithm* 2) we obtain a $j \times q$ matrix N whose rows are the unit normals s_i^t $(i \in J)$ to the encountered hyperplanes. The successive applications of the procedure *STEP* (in *Algorithm 2*) build up N row by row and produce $z = N^+ e$. Thus, by *Property 3*, the vector z is normal to the j hyperplanes already encountered; or, equivalently, z is normal to the intersection of these j hyperplanes. While building up the matrix N, the particular point (solution) x which was initially at the intersection of these hyperplanes, i.e., which was initially equidistant (distance is 0) from these hyperplanes, continues to move along the line whose every point is equidistant from the j hyperplanes. In fact, as soon as the construction of one row of N via *STEP* is over, the point x moves along the foregoing line by an amount λz, where λ is a value computed by the procedure *LAMDA*. Since x is a point equidistant from these j hyperplanes, by *Property 4*, the point $x + \lambda z$ is also equidistant from these j hyperplanes for any λ. Just after building up each row of N, *LAMDA* computes λ so that $x + \lambda z$ will be equidistant from the j hyperplanes and one more. By *Property 2*, the distances of the point x from the n hyperplanes are the components of $\delta = Sx - u$ and those of the point $x + \lambda z$ are $\delta' - S(x + \lambda z) - u = \delta + \lambda\alpha$, where $\alpha = Sz$. We have $\delta_i = \delta_\beta$ (constant) for $i \in J$ and $\alpha_i = \alpha_\beta = 1$ for $i \in J$ (since $Nz = e$). Therefore, $\delta'_\beta = \delta_\beta + \lambda\alpha_\beta$ $(i \in J)$ and $\delta'_\gamma = \delta_\gamma + \lambda\alpha_\gamma$ $(\gamma \notin J)$. For the point $x + \lambda z$ to be equidistant from the j hyperplanes and one more, we should have $\delta'_\beta = \delta'_\gamma$ for some $\gamma \notin J$. Thus, we have $\lambda = (\delta_\beta - \delta_\gamma)/(\alpha_\gamma - \alpha_\beta)$. The procedure *LAMDA* is actually choosing, out of several λ's, a smallest positive λ such that

$x + \lambda z$ will be equidistant from the j encountered hyperplanes plus one more.

4. EXAMPLES

Example 1: Consider the equation $Ax = b$, $x \geq 0$, where $A = \begin{bmatrix} 1 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 2 \end{bmatrix}$, $m = 1$, $n = 2$.

S.1 Find $x = A^+ b$, P, r by *Algorithm* 1:

Initialize $P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $x = \begin{bmatrix} 0 & 0 \end{bmatrix}^t$, $r = 0$;

$j = 1$.

STEP(a_j, b_j, P, x, *ind*)

ind $= 0, v = Pa_j = \begin{bmatrix} 1 & 1 \end{bmatrix}^t$, $y = \| v \|^2 = 2$, $b_j = b_j - a_j^t x = 2$.

$$P = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}, \quad x = \begin{bmatrix} 1 & 1 \end{bmatrix}^t, \quad ind = 1.$$

$x = A^+ b \geq 0$ (already).

$r = r + ind = 1$.

S.2 $q = n - r = 1$.

S.3 $$D = \begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}, \quad S = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}, \quad \delta = \begin{bmatrix} \sqrt{2} & \sqrt{2} \end{bmatrix}^t.$$

S.4 Apply *Algorithm* 2: $\delta_\beta = \min_i \{\delta_i\} = \sqrt{2}; J = \{1\}$; $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $z = \begin{bmatrix} 0 & 0 \end{bmatrix}^t$, $j = 1$.

STEP(s_β, 1, Q, z, *ind*)

ind $= 0$; $v = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}^t$, $y = 1$, $b = 1$, $Q = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$,

$z = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}^t$, *ind* $= 1$.

LAMDA

ind $= 1$; $\alpha = Dz = \begin{bmatrix} 1 & -1 \end{bmatrix}^t$; $\lambda_2 = (\delta_2 - \delta_1)/(\alpha_1 - \alpha_2) = 0$; $\lambda = \lambda_2 = 0$;

$\beta = 2$; $\delta = \begin{bmatrix} \sqrt{2} & \sqrt{2} \end{bmatrix}$, $x = \begin{bmatrix} 1 & 1 \end{bmatrix}^t$; $J = \{1, 2\}$.

$j = 2$. j is now $q + 1$; so, the center is $\begin{bmatrix} 1 & 1 \end{bmatrix}$. It may be seen that this center serves as a good initial feasible solution for the related *lp* problem.

Geometry. The following figure (*Fig. 1*) depicts the geometry of the foregoing polytope and its center.

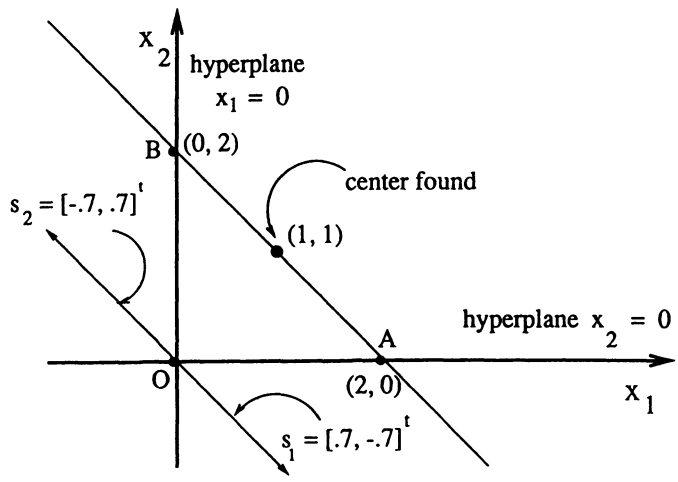

Fig. 1 Geometry of the polytope $\begin{bmatrix} 1 & 1 \end{bmatrix} \boldsymbol{x} = 2$

The finite line AB is the polytope π. The line AB extended infinitely on both sides is the solution space ω. $\delta_1 = \sqrt{2}$ is the Euclidean distance of the center $\begin{bmatrix} 1 & 1 \end{bmatrix}^t$ from the hyperplane $x_1 = 0$ in ω. $\delta_2 = \sqrt{2}$ is that of the center from the hyperplane $x_2 = 0$ in ω. It can be noticed that the hyperplane $x_1 = 0$ in ω is the point B, i.e., $\begin{bmatrix} 0 & 2 \end{bmatrix}^t$, and the hyperplane $x_2 = 0$ in ω is the point A, i.e., $\begin{bmatrix} 2 & 0 \end{bmatrix}^t$. The vector $\boldsymbol{s}_1^t$ is normal to the hyperplane $x_1 = 0$ in ω (i.e., the point B) while the vector $\boldsymbol{s}_2^t$ is normal to the hyperplane $x_2 = 0$ in ω (i.e., the point A). $\boldsymbol{s}_i^t$ is a row vector whose position with invariant direction can be imagined to be present at any other place besides the origin 0. Thus, the row vector $\boldsymbol{s}_1^t$ from B is directed into the polytope BA and $\boldsymbol{s}_2^t$ from A into AB.

Example 2: Consider the equation $A\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$, where

$$A = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} -5 \\ 6 \end{bmatrix}, m = 2, n = 4,$$

S.1 Find $\boldsymbol{x}$, P, r by *Algorithm* 1:

Initialize $P = I_4$; $\boldsymbol{x} = \boldsymbol{0}$; $r = 0$;

$j = 1$.

$STEP(a_1, b_1, P, x, ind)$

$ind = 0$; $v = Pa_1 = [-1 \quad 1 \quad 1 \quad 0]^t$, $y = \|v\|^2 = 3$, $b_1 = -5$;

$$y \neq 0. \quad P = \begin{bmatrix} 2/3 & 1/3 & 1/3 & 0 \\ 1/3 & 2/3 & -1/3 & 0 \\ 1/3 & -1/3 & 2/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad x = [\,5/3 \quad -5/3 \quad -5/3 \quad 0\,]^t, \quad ind = 1;$$

$r = 1$.

$j = 2$.

$STEP(a_2, b_2, P, x, ind)$

$ind = 0$; $v = [\,1 \quad 1 \quad 0 \quad 1\,]^t$; $y = 3$; $b_2 = 6$;

$$y \neq 0. \quad P = \begin{bmatrix} 1/3 & 0 & 1/3 & -1/3 \\ 0 & 1/3 & -1/3 & -1/3 \\ 1/3 & -1/3 & 2/3 & 0 \\ -1/3 & -1/3 & 0 & 2/3 \end{bmatrix};$$

$x = [\,11/3 \quad 1/3 \quad -5/3 \quad 2\,]^t$.

$ind = 1$; $r = 2$.

S.2 $q = n - r = 2$.

S.3 $$D = Diag(1/\sqrt{p_{ii}}) = \begin{bmatrix} 1.7321 & 0 & 0 & 0 \\ 0 & 1.7321 & 0 & 0 \\ 0 & 0 & 1.2247 & 0 \\ 0 & 0 & 0 & 1.2247 \end{bmatrix};$$

$$S = DP = \begin{bmatrix} .5774 & 0 & .5774 & -.5774 \\ 0 & .5774 & -.5774 & -.5774 \\ .4082 & -.4082 & .8165 & 0 \\ -.4082 & -.4082 & 0 & .8165 \end{bmatrix}; \quad \delta = Dx = \begin{bmatrix} 6.3512 \\ .5774 \\ -2.0412 \\ 2.4494 \end{bmatrix}.$$

S.4 Apply *Algorithm* 2.

$\delta_\beta = \min_i\{\delta_i\} = \delta_3 = -2.0412$; $J = \{3\}$; $Q = I_4$; $z = 0$; $j = 1$;

$STEP(s_\beta, 1, Q, z, \text{ind}) = STEP(s_3, 1, Q, z, \text{ind})$;

$ind = 0$; $v = [\,.4082 \quad -.4082 \quad .8165 \quad 0\,]^t$; $y = 1$; $b = 1 - s_3^t z = 1$;

$$y \neq 0;\ Q = Q - vv^t/y = \begin{bmatrix} .8334 & .1666 & -.3333 & 0 \\ .1666 & .8334 & .3333 & 0 \\ -.3333 & .3333 & .3333 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$z = [\ .4082\ \ -.4082\ .8166\ \ 0\ ]^t;\ ind = 1;$

LAMDA

$ind = 1;\ \alpha = Dz = [\ .7071\ \ -.7071\ \ 1\ \ 0\ ]^t;$

$\lambda_i = (\delta_i - \delta_\beta)/(\alpha_\beta - \alpha_i)\ \ (i \notin J,\ \alpha_i \neq \alpha_\beta)$

$\lambda_1 = (\delta_1 - \delta_3)/(\alpha_3 - \alpha_1) = 28.6521;\ \lambda_2 = 1.5339;\ \lambda_4 = 4.4906;$

$\lambda_\beta = min(\lambda_i),\ \lambda_i \geq 0 = \lambda_2 = 1.5339;\ \lambda = 1.5339;$

$\delta = \delta + \lambda\alpha = [7.4358 - .5072 - .50732.4494]^t;$

$x = x + \lambda z = [4.2928 - 0.2928\ 0.4141\ \ 2\ ]^t;$

$J = \{2,3\}.$

Since $ind = 1,\ j = 2.$

STEP$(s_2,\ 1,\ Q,\ z,\ ind)$

$ind = 0;\ v = Qs_2 = [\ .2886\ \ .2888\ \ 0\ \ -.5774\ ]^t;\ y = \|v\|^2 = .5;$

$b = 1 - s_2^t z = 1.7072;$

$$y \neq 0;\ Q = Q - vv^t/y = \begin{bmatrix} .6668 & -.0001 & -.3333 & .3333 \\ -.0001 & .6667 & .3333 & .3334 \\ -.3333 & .3333 & .3333 & 0 \\ .3333 & .3334 & 0 & .3333 \end{bmatrix}$$

$z = z + bv/y = [\ 1.3936\ \ .5776\ \ .8166\ \ -1.9711]^t;\ ind = 1;$

LAMDA

$ind = 1;\ \alpha = Dz = [\ 2.4138\ \ 1.0004\ \ 1.0001\ \ -2.4140\ ]^t;$

$\lambda_i = (\delta_i - \delta_\beta)/(\alpha_\beta - \alpha_i)\ (i \notin J;\ \alpha_i \neq \alpha_\beta)$

$\lambda_1 = (\delta_1 - \delta_2)/(\alpha_2 - \alpha_1) = -5.6198;\ \lambda_4 = .8659;\ \lambda_\beta = \lambda_4 = .8659;$

$\lambda = .8659.$

$\delta = \delta + \delta\alpha = [9.5260\ .3590\ .3587\ .3590\ ]^t;$

$x = x + \lambda z = [5.4996\ .2073\ .2930\ .2932\ ]^t;$

$J = \{2,3,4\},\ j = 3.\ \{\text{since } j = q + 1\}$

stop.

The solution x is a required center of the polytope and hence it is an initial feasible solution of a related linear programming problem.

Remark: If a nonnegative solution of linear equations is required then a center which is a nonnegative solution may be computed.

5. PARALLEL IMPLEMENTATION

Step *S.1* (*Algorithm* 1) of the centering algorithm, coded as a host program and a node program, is implemented on a parallel hypercube computer as follows. The host sends the number of rows m and the number of columns n of the mathix A, the number of processors (nodes) $p = 2^d$ used, and the right-hand side vector $\boldsymbol{b}$ of the equation $A\boldsymbol{x} = \boldsymbol{b}$ to each node processor. Which part (rows) of P and $\boldsymbol{x}$ will be computed by each node processor? In order to consider this question we set $k = [n/p]$ and $k_1 = n - kp$. If $k_1 \neq 0$ then each of Nodes $0,1,2,\ldots,k_1 - 1$ computes $k+1$ rows of P and $\boldsymbol{x}$, and each of Nodes $k_1, k_1+1,\ldots,p-1$ computes k rows of P and $\boldsymbol{x}$. If $k_1 = 0$ then each of Nodes $0,1,2,\ldots,p$ computes k rows of P and $\boldsymbol{x}$. For example, if we have 4 Node processors $0,1,2,3$ and P is 10×10 then $n = 10$, $p = 4$, $k = 2$, and $k_2 = 2$. Node 0 will compute first three rows of P and $\boldsymbol{x}$, Node 1 next three rows of P and $\boldsymbol{x}$, Nodes 2 and 3 the remaining two rows each. Each processor initializes its part of P and its part of $\boldsymbol{x}$. Observe that $\boldsymbol{x}$ is initially zero and P is initially an identity matrix. It then sends one row of A at a time to each processor starting from the first row. Each node processor, on receipt of a row of A, computes its part of the vector $\boldsymbol{v}$, the resulting (partial) y, and b_j. Each node then communicates these partial $\boldsymbol{v}$ and partial y to other nodes so that each node has complete $\boldsymbol{v}$ and complete y (the global sum) in its memory; the number of communications required is d, the dimension of the hypercube. If $y \neq 0$ then each node processor computes its part of P and $\boldsymbol{x}$, and sets $ind = 1$. The process is continued until the host program completes sending all the rows of A. On receipt of one row of A, each node processor will be executing the procedure *STEP* once. Thus each node will execute *STEP* m times to form its own part of P and $\boldsymbol{x}$, and also the rank r of A and the dimension of the solution space $q = n - r$ corresponding to the m rows of A. Since the communication time between the host and a node processor is much larger than that between one node and another, it will be seen later that each processor sends its part of solution to Node processor 0 which appropriately appends the parts of $\boldsymbol{x}$ and forms the required $\boldsymbol{x}$.

The parallelization of Steps *S.2* and *S.3* proceeds along with that of Step *S.1*; no communication among processors is needed. Every processor computes the dimension of the solution space q. It then computes the rows of D, δ, and S corresponding to its parts of P and $\boldsymbol{x}$. Thus, each node processor has its part of D, δ, and S. If $p_{ii} = 0$, i.e., if the ith row and the

ith column of P are zero, for some i then, instead of deleting the ith row and ith column of P and consequently removing the corresponding coordinates as is done in the sequential algorithm, we set ith row and ith column of S to zero as well as (i,i)th element of D zero; we also set $\delta_i = K$, a sufficiently large positive real number, and corresponding component of $x = 0$. The motive of retaining this unnecessary information is to avoid inter-processor communication (that will arise due to index modification) at these steps. In fact, this information will be skipped as and when they are encountered.

For the parallel implementation of Step ***S.4*** (*Algorithm* 2) of the Centering algorithm each node processor needs complete δ in its memory; so, the number of communications that is required to accomplish this task is d. It then finds a smallest component of δ and records its index as β. The processor that contains the βth row of S, viz. s^t_β, sends this row to every other processor. After initializing its part of the matrix Q and vector z each processor enters into the repeat loop containing two procedures *STEP* and *LAMDA*. Having completed the execution of *STEP* as in *Algorithm* 1, each node, if $ind = 1$, executes *LAMDA*, computes its part of α, communicates this part to all other processors so that each processor has complete α in its memory. Thus every processor using the complete α, δ, and the index β computes all λ_i, finds a smallest nonnegative λ_i if not all $\lambda_i < 0$, and then updates complete δ and its part of x. Thus, each processor executes the repeat loop the required number of times; the processor 0 then collects from all other processors their parts of the solution, appends them appropriately and sends the complete solution x to the host.

The number of communications between the host and a node depends on the size of the coefficient matrix A relative to the memory capacity of the nodes. If the capacity of each node is insufficient to store the entire matrix, we can let each node keep only the right-hand side vector b and one row at a time of the matrix A. In this case the host must send the matrix A row by row to each node processor, as described above. The number of communications required is then $m + 3$. If the storage for each node is not a problem, the host can send the entire matrix A and vector b to each node in one communication. The number of communications is then 3. However, the number of communications among nodes in our parallel code is $6d$, where d is the dimension of the hypercube used. If a typical communication is about 500 times more expensive than a flop then the order of the matrix should be at least 130×130 to achieve any speed up over a sequential machine.

6. CONCLUDING REMARKS

If the polytope defined by $Ax = b,\ x \geq 0$ is nonexistent then the centering algorithm detects that condition. If it is unbounded then the algorithm produces a nonnegative solution of $Ax = b$ which may be useful for a physical problem; further, such a solution with an indication of unboundedness is also useful while solving an *lp* problem. Preservation of linearity and simplicity by the algorithm is attractive in interior point methods for *lp* problems.

While *Algorithm* 1 (for the computation of the projection operator $P = (I - A^+A)$, the solution vector $x = A^+b$, and the rank of A) can be implemented error-free, *Algorithm 2* (Steps *S*.3 and *S*.4) however is not implementable error-free because of square-root operations. Thus, a partial error-free computation is possible and may be useful in enhancing the accuracy of the final solution. It can be seen [6,7] that the error-free computation is inherently parallel and immune to 'intermediate number growth'. Since the centering algorithm involves extensive matrix-vector operations, implementation of the algorithm on a vector/super-computer is relatively straight-forward.

It can be seen [13] that a convex polytope is defined, in a more general form, as a set which can be expressed as the intersection of a finite number of closed half spaces.

ACKNOWLEDGEMENTS. S. K. Sen is on leave from SERC, Indian Institute of Science, Bangalore 560012, India and wishes to acknowledge the support of a Fulbright fellowship for the preparation of this paper. D. W. Fausett was supported by NSF Grant #ASC 8821626, New Technologies Program, Division of Advanced Scientific Computing.

The authors thank Dr. Chuck Romine of Oak Ridge National Laboratory for his assistance in developing a parallel code for the centering algorithm on the Intel i860 computer.

REFERENCES

[1] Lord, E.A., Venkaiah, V. Ch., and Sen, S. K., An algorithm to compute a center of a polytope, Proc. CSI-90 (1990) 1 - 8.

[2] Bayer, D.A. and Lagarias, J.C., The nonlinear geometry of linear programming I, Affine and projective scaling trajectories, Trans. Amer. Math. Soc. **314** (1989) 499-526.

[3] Renegar, J., A polynomial-time algorithm based on Newton's method, for linear programming, Mathematical Programming **40** (1988) 59-93.

[4] Vaidya, P.M., An algorithm for linear programming which requires $0(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations, Proc. ACM Annual Symp. on Theory of Computing (1987) 29-38.

[5] Lord, E.A., Sen, S.K., and Venkaiah, V.Ch., A concise algorithm to solve over-/under-determined linear systems, Simulation **5** (1990) 239-240.

[6] Venkaiah, V. Ch. and Sen, S.K., Error-free matrix symmetrizers and equivalent symmetric matrices, Acta Applicande Mathematichae **21** (1990) 291-313.

[7] Venkaiah, V. Ch., Computations in Linear Algebra: A New Look at Residue Arithmetic, *Ph.D. Thesis*, Indian Institute of Science, Bangalore 560012, India, 1987.

[8] Rao, C.R. and Mitra, S.K., Generalized Inverse of Matrices and Its Applications, Wiley, 1974.

[9] Golub, G.H. and Van Loan, C.F., Matrix Computations, 2nd Ed., Johns Hopkins Univ. Press, 1989.

[10] Bischof, C.H. and Hansen, P.C., A block algorithm for computing rank-revealing QR factorizations, *Argonne National Laboratory Preprint* MCS-P251-0791, 1991.

[11] Hong, Y.P. and Pan, C.T., Rank-revealing QR factorizations and the singular value decomposition, *Argonne National Laboratory Preprint* MCS-P188-1090, 1990.

[12] Stewart, G.W., Updating a rank-revealing ULV decomposition, *University of Maryland Technical Report* UMIACS-TR-91-39, 1991.

[13] Luenberger, D.G., Introduction to Linear and Nonlinear Programming, 2nd Ed. Addison-Wesley, 1984.