NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl., **5,** 461–474 (1998)

Computing the Nearest Diagonally Dominant Matrix

María Mendoza[1], Marcos Raydan[*2] and Pablo Tarazaga[3]

[1]*Departamento de Matemáticas, Facultad de Ciencias, Universidad Central de Venezuela, Ap. 20513, Caracas 1020-A, Venezuela*
[2]*Departamento de Computación, Facultad de Ciencias, Universidad Central de Venezuela, Ap. 47002, Caracas 1041-A, Venezuela (e-mail: mraydan@reacciun.ve)*
[3]*Department of Mathematics, University of Puerto Rico, Mayaguez, Puerto Rico 00681-5000, USA*

We solve the problem of minimizing the distance from a given matrix to the set of symmetric and diagonally dominant matrices. First, we characterize the projection onto the cone of diagonally dominant matrices with positive diagonal, and then we apply Dykstra's alternating projection algorithm on this cone and on the subspace of symmetric matrices to obtain the solution. We discuss implementation details and present encouraging preliminary numerical results. © 1998 John Wiley & Sons, Ltd.

KEY WORDS diagonally dominant matrices; matrix cones; Dykstra's algorithm; Kuhn–Tucker conditions

1. Introduction

We consider the problem of minimizing the distance of a given matrix to the set of symmetric and Diagonally Dominant (DD) matrices with positive diagonal. In other words, we consider the following optimization problem

$$\min \|X - A\|_F^2 \tag{1.1}$$

* Correspondence to M. Raydan, Departamento de Computación, Facultad de Ciencias, Universidad Central de Venezuela, Ap. 47002, Caracas 1041-A, Venezuela.

Contract/grant sponsor: CONICIT. Contract/grant sponsor: Numerical Analysis Laboratory at UCV. Contract/grant sponsor: NSF; Contract/grant numbers: HRD-9450448 and CDA-94117362.

CCC 1070–5325/98/060461–14 $17.50
© 1998 John Wiley & Sons, Ltd.

Received 27 July 1997
Revised 25 May 1998

subject to

$$X \in S$$

$$X \in DD^+$$

where A is a real $n \times n$ matrix, and X is the matrix we wish to find in the set S of symmetric matrices and also in the set DD^+ of DD matrices with positive diagonal. To be precise, $X \in DD^+$ if $x_{ii} \geq \sum_{j \neq i} |x_{ij}|$ for all $1 \leq i \leq n$.

For any two $n \times n$ real matrices A and B, we define their inner product by $\langle A, B \rangle = \text{trace}(A^{\mathrm{T}} B)$. In that case, $\|A\|_{\mathrm{F}} = \sqrt{\langle A, A \rangle}$ is the Frobenius norm. In this inner product space, S is a closed subspace and the sets DD^+ and $S \cap DD^+$ (denoted by SDD^+) are closed and convex polyhedral cones. The algebraic and geometric properties of these cones have been studied previously by Barker [3], Barker and Carlson [4], Barker and Loewy [5], and Fiedler and Ptak [15].

Symmetric DD matrices appear frequently in different areas of numerical analysis. In particular, if the diagonal is positive then any DD matrix is also a positive semidefinite matrix. In many optimization problems the solution matrix is constrained to be inside the cone of positive semidefinite matrices. For instance, constrains of this kind appear in the *student test* problem (see Fletcher [16,17]), in least-squares matrix problems (see Escalante and Raydan [13,14], Hu [22] and Hu and Olkin [23]), and also in the design of mathematical models for the economy (see Dantzig [9]). Unfortunately, projecting a matrix onto the cone of positive semidefinite matrices usually requires a significant amount of computational work since it involves good estimation of eigenvalues and eigenvectors (see Allwright [1], Hayden and Wells [19], Higham [20], and Hill and Waters [21]).

An approximation of the nearest symmetric positive semidefinite matrix can also be found by using modified Cholesky factorization techniques. These factorization techniques do not require any information about eigenvalues or eigenvectors (see Gill, Murray and Wright [26, Section 4.4.2.2], Schnabel and Eskow [27], and more recently Cheng and Higham [8]). However, in most cases, they produce a significant amount of *fill-in* that destroys the sparsity pattern of the original matrix A. As a consequence, factorization techniques are suitable only for small or medium size problems. On the other hand, forcing a matrix to be in the SDD^+ cone requires, as we will discuss in this work, a similar amount of computational work that does not destroy the sparsity pattern of the given matrix A. Therefore, projecting on the SDD^+ cone is suitable for large-scale problems.

Using the Kuhn–Tucker conditions we first characterize, in Section 2, the projection onto the DD^+ cone. Then, in Section 3, we apply Dykstra's alternating projection algorithm [6,12], to force symmetry and solve problem (1.1).

Dykstra's algorithm is based on a clever modification of the classical alternating projection method first proposed by von Neumann [25], and studied later by Cheney and Goldstein [7]. It guarantees convergence to the closest point in the intersection of closed and convex sets that are not necessarily closed subspaces. It has been recently applied to find the closest Euclidean distance matrix [18,30], and also to solve constrained least-squares matrix problems [13,14]. For a complete survey on Dykstra's algorithm and applications see Deutsch [10].

In Section 4, we present two different implementations of Dykstra's algorithm applied to problem (1.1) and discuss their behavior on some numerical experiments. Finally, in Section 5, we present some concluding remarks.

2. Projection onto DD^+

Using properties of the Frobenius norm we can observe that the solution of the following optimization problem:

$$\min \|X - A\|_F^2 \tag{2.1}$$

subject to

$$X \in DD^+$$

can be obtained by solving n independent optimization problems of the form

$$\min \{\|x - a_i\|_2^2 : x \in C_i^+\} \tag{2.2}$$

where $C_i^+ = \{x \in \mathbb{R}^n : \quad x_i \geq \sum_{j \neq i} |x_j| \}$, and a_i is the ith row of the given matrix A. Notice that the n independent problems can be solved in parallel and they produce the n rows of the solution matrix of problem (2.1). In this section, we simultaneously solve the closely related optimization problem:

$$\min \{\|x - a_i\|_2^2 : x \in C_i^-\} \tag{2.3}$$

where $C_i^- = \{x \in \mathbb{R}^n : \; -x_i \geq \sum_{j \neq i} |x_j|\}$. Later this will allow us to characterize the projection onto the set of diagonally dominant matrices.

Notice that C_i^+ and C_i^- are closed and convex sets, and also that $\|x - a_i\|_2^2$ is a strictly convex function. Hence, any of the n independent optimization problems (2.2) or (2.3) has a unique solution. In what follows $a = (a_1, a_2, \ldots, a_n)$ denotes the ith row of A. To solve problems (2.2) or (2.3) we assume without any loss of generality, and for the sake of clarity, that

$$\begin{cases} a_j \geq 0, \qquad 1 \leq j \leq n \qquad\qquad (j \neq i) \\ a_1 \leq \ \ldots \leq a_j \leq \ \ldots \leq a_n \quad (j \neq i) \end{cases} \tag{2.4}$$

Our next result will be useful to solve problems (2.2) and (2.3).

Lemma 2.1. *If $\bar{x} = (x_1, x_2, \ldots, x_n)$ solves the following problem:*

$$\min \{\|x - a\|_2^2 : x \in C_i\} \tag{2.5}$$

where $C_i = \{x \in \mathbb{R}^n : |x_i| \geq \sum_{j \neq i} |x_j|\}$; then for each $1 \leq j \leq n$, either $a_j\, x_j > 0$ or $x_j = 0$.

Proof
The proof is straightforward. ■

From Lemma 2.1 and assumption (2.4) we obtain the following equivalent convex programming problem to solve (2.2) and (2.3) for every i:

$$\min \{\|x - a\|_2^2 = (x_1 - a_1)^2 + \ldots + (x_n - a_n)^2 \quad : x \in \mathbb{R}^n\} \tag{2.6}$$

subject to $g_j(x) = x_j \geq 0$ for all $j \neq i$, and $g_i(x) = -s_i x_i - \sum_{j \neq i} x_j \geq 0$, where

$$s_i = \begin{cases} + & \text{for} \quad C_i^- \\ - & \text{for} \quad C_i^+ \end{cases}$$

We now present an algorithm that generates the unique solution $\bar{x}$ of (2.6). First, we need to introduce some notation. Given a, the ith row of A, and an index $m \neq i$ $(1 \leq m \leq n)$, we define $d_m = \sum_{j=m}^{n} a_j + s_i a_i,\ (j \neq i)$,

$$c_m = \begin{cases} n - m + 1 & if \ \ m < i \\ n - m + 2 & if \ \ m > i \end{cases}$$

and $\bar{d}_m = d_m / c_m$.

Algorithm 2.1. *Given a the ith row of A, and an index i, set $x_i = a_i - s_i \bar{d}_k$,*

```
for $1 \leq j \leq k-1 \quad (j \neq i)$
    $x_j = 0$
end
for $k \leq j \leq n \quad (j \neq i)$
    $x_j = a_j - \bar{d}_k$
end
```

where $k \neq i$ is the smallest integer $1 \leq k \leq n$ such that $a_k > 0$ and $\bar{d}_k \leq a_k$.

To illustrate the algorithm we now present a five-dimensional example. Let $a = (1,\ 1,\ 3,\ 4,\ 5)^t$ and $i = 2$. In this case, $\bar{d}_1 = 2.4 > a_1$, and $\bar{d}_3 = 2.75 < a_3$. Therefore, $k = 3$, $\bar{d}_3 = 2.75$, and $\bar{x} = (0,\ 3.75,\ 0.25,\ 1.25,\ 2.25)^t$.

Remark 1

- Since the components of the vector a are in ascending order, then d_k is obtained in Algorithm 2.1 by adding as many components a_j as possible such that $\bar{d}_k \leq a_j\ (j \neq i)$.
- The first $k-1$ components for which $\bar{d}_j > a_j\ (j \neq i)$ will be zero in the solution vector $\bar{x}$.
- The size of the diagonal component a_i is increased and the size of the off-diagonal components are decreased by the factor $\bar{d}_k$ to obtain diagonal dominance with equality in the solution vector $\bar{x}$.

Later in this section, we prove that the algorithm is well defined and solves (2.6), if $\bar{d}_k$ exists. We will now present a few lemmas to characterize the existence of $\bar{d}_k$ and to establish that if $\bar{d}_k$ does not exist then $\bar{x} = 0$. These lemmas are of fundamental importance in the implementation of our algorithms in Section 4. Notice that:

- If $a_i \geq \sum_{j \neq i} a_j$ then the projection onto C_i^+ is given by $\bar{x} = a$.
- If $-a_i \geq \sum_{j \neq i} a_j$ then the projection onto C_i^- is given by $\bar{x} = a$.

Therefore, to study the existence of $\bar{d}_k$ in the next few lemmas, we will ignore these two cases.

Lemma 2.2. *If $a_i \geq 0$ when projecting on C_i^+ or $a_i \leq 0$ when projecting on C_i^- and if $|a_i| < \sum_{j \neq i} a_j$, then there exists $\bar{d}_k$ as defined in Algorithm 2.1.*

Proof
If $i \neq n$, using (2.4) and the fact that $|a_i| < \sum_{j \neq i} a_j$, it follows that $a_n > 0$. Let us now suppose, by way of contradiction, that for all $k \neq i$ either $\bar{d}_k > a_k$ or $a_k = 0$. Hence, $\bar{d}_n > a_n > 0$. Therefore, by definition of $\bar{d}_n$ we have that $(a_n - |a_i|)/2 > a_n$, which is a contradiction and the result follows. Similarly, if $i = n$ we repeat the same arguments with a_{n-1} instead of a_n. ■

Lemma 2.3. *If $a_i < 0$ when projecting on C_i^+ or $a_i > 0$ when projecting on C_i^- and if $|a_i| > a_j, \ 1 \leq j \leq n \ , \ j \neq i$, then*

1. *There is no $\bar{d}_k$, for any k, as defined in Algorithm 2.1.*
2. *The unique solution to problem (2.6) is $\bar{x} = (0 \dots 0)$.*

Proof
Using the definition of d_k and the assumptions, we obtain

$$d_k = \sum_{j=k}^{n} a_j + |a_i| > \sum_{j=k}^{n} a_j \ + a_k \quad (j \neq i)$$

Hence

- If $k < i$ then $d_k > (n-k+1)a_k \ \ (j \neq i)$, and so $\bar{d}_k = (d_k/(n-k+1)) > a_k$.
- If $k > i$ then $d_k > (n-k+2)a_k \ \ (j \neq i)$, and so $\bar{d}_k = (d_k/(n-k+2)) > a_k$.

Therefore, there is no $\bar{d}_k$, for any k, in Algorithm 2.1 and (1) is established.

To establish (2) consider the first-order necessary and sufficient Kuhn–Tucker conditions associated with problem (2.6) (see, for instance, [2, pp. 95–96]):

$$\begin{array}{ll}
\text{(a)} \left\{ \begin{array}{lll} 2(x_j - a_j) - u_j + u_i & = 0 & \text{for } j \neq i \\ 2(x_i - a_i) + s_i u_i & = 0 & \end{array} \right. \\
\text{(b)} \left\{ \begin{array}{lll} g_j(x) & = x_j \geq 0 & \text{for } j \neq i \\ g_i(x) & = -s_i x_i - \sum_{j \neq i} x_j \geq 0 & \end{array} \right. \\
\text{(c)} \left\{ \begin{array}{lll} u_j x_j & = 0 & \text{for } j \neq i \\ u_i(-s_i x_i - \sum_{j \neq i} x_j) & = 0 & \end{array} \right. \\
\text{(d)} \ u_j \geq 0, \qquad 1 \leq j \leq n &
\end{array} \tag{2.7}$$

Let us now consider the vectors $\bar{x} = 0$ and u given by $u_i = 2|a_i| > 0$, and $u_j = -2a_j + u_i$ for all $j \neq i$. Clearly, using (2.4) and the fact that $|a_i| > a_j, \ 1 \leq j \leq n \ , \ j \neq i$, it follows that these two vectors satisfy conditions (2.7). Therefore, $\bar{x} = 0$ is the unique solution of problem (2.6). ■

Lemma 2.4. *If $a_i < 0$ when projecting on C_i^+ or $a_i > 0$ when projecting on C_i^- and if $|a_i| \leq \sum_{j \neq i} a_j$, then*

(i) If there exists $j \neq i$ such that $|a_i| \leq a_j$, then $\bar{d}_k$ exists as in Algorithm 2.1.
(ii) If $|a_i| > a_j$, $1 \leq j \leq n$ $j \neq i$, then there is no $\bar{d}_k$ as in Algorithm 2.1, and $\bar{x} = 0$ is the unique solution of problem (2.6).

Proof
If $i \neq n$ then by assumptions and (2.4) we have that at least $a_n > 0$. Moreover, since $\bar{d}_n = ((a_n + s_i a_i)/2)$, then

$$\bar{d}_n = ((a_n + |a_i|)/2) \leq ((a_n + a_j)/2) \leq a_n$$

and so there exists $\bar{d}_k$ for some k. Similarly, if $i = n$ we repeat the same arguments with a_{n-1} instead of a_n. To finish the proof notice that (ii) follows directly from Lemma 2.3 ■

It is worth mentioning that if $a_i < 0$ when projecting onto C_i^+ or $a_i > 0$ when projecting onto C_i^-, and $|a_i| > \sum_{j \neq i} a_j$ we obtain that $|a_i| > a_j$, $1 \leq j \leq n$ $(j \neq i)$. Consequently, by Lemma 2.3 there is no $\bar{d}_k$ as in Algorithm 2.1, and the unique solution to problem (2.6) is again $\bar{x} = 0$.

Lemma 2.5. *If, under the assumptions of Lemma 2.2 or Lemma 2.4, $\bar{d}_k$ exists for some k then the sequence $\{\bar{d}_l\}_1^k, l \neq i$, is a monotonic increasing sequence of positive real numbers.*

Proof
First, we need to prove that $\bar{d}_l > 0$ for $l \neq i$ and $1 \leq l \leq (k-1)$. Since k is the smallest integer such that $a_k > 0$ and $\bar{d}_k \leq a_k$, then either $\bar{d}_l > a_l \geq 0$ or $a_l = 0$. If $a_l = 0$ then by (2.4) and the assumptions of either Lemma 2.2 or Lemma 2.4 we obtain that

$$d_l = \sum_{j=l}^{n} a_j + s_i a_i = \sum_{j \neq i} a_j + s_i a_i > 0$$

which implies that $\bar{d}_l = d_l / c_l > 0$.

Next, we need to establish that $\{\bar{d}_l\}_1^k$, $l \neq i$, is a monotonic increasing sequence. Let $1 \leq l \leq k-1, l \neq i$, be an arbitrary chosen index. If $l+1 \neq i$, then let us suppose by way of contradiction that $\bar{d}_l > \bar{d}_{l+1}$, which implies by definition of $\bar{d}_l$ that

$$c_{l+1} d_l - c_l d_{l+1} > 0$$

Using now that $c_l = c_{l+1} + 1$ and $d_{l+1} = d_l - a_l$, we obtain

$$a_l > \frac{d_l}{c_l} = \bar{d}_l > 0$$

which is a contradiction, since $l \leq k-1$ and k is the smallest integer with these properties. If $l+1 = i \neq k$ we repeat the same argument supposing that $\bar{d}_l > \bar{d}_{l+2}$, and we obtain a similar contradicton.

Finally, we use the previous two results to obtain that $\bar{d}_k \geq \bar{d}_{k-1} > 0$ if $k > 1$ and $k-1 \neq i$. Similarly, if $k-1 = i$ we obtain $\bar{d}_k \geq \bar{d}_{k-2} > 0$ if $k > 2$. If $k = 1$ or $k = 2$ and $i = 1$ then by definition of $\bar{d}_k$ and the assumptions of either Lemma 2.2 or Lemma 2.4 we obtain that $\bar{d}_k > 0$. ■

Using the Kuhn-Tucker conditions we now establish that Algorithm 2.1 generates the unique solution to problem (2.6) if $\bar{d}_k$ exists.

Lemma 2.6. *If, under the assumptions of Lemma 2.2 or Lemma 2.4, $\bar{d}_k$ exists for some k then $\bar{x}$ generated by Algorithm 2.1 is the unique solution of problem (2.6).*

Proof
Consider the vector $\bar{x}$ generated by Algorithm 2.1 and the vector $u = (u_1, u_2, \ldots, u_n)$ given by $u_k = u_{k+1} = \ldots = u_n = 0$, $u_j = -2a_j + 2\bar{d}_k$ for $1 \le j \le k-1$, $j \ne i$, and $u_i = 2\bar{d}_k$. By direct substitution, we observe that $\bar{x}$ and u satisfy (a) in (2.7).

Notice now that $x_1 = x_2 = \ldots = x_{k-1} = 0$, and so $g_j(\bar{x}) = x_j = 0$ for $1 \le j \le k-1$, $j \ne i$. Notice also that once k has been found in Algorithm 2.1, $\bar{d}_k \le a_k \le \ldots \le a_n$. Thus, $g_j(\bar{x}) = a_j - \bar{d}_k \ge 0$ for $k \le j \le n$, $j \ne i$. Moreover,

$$\begin{aligned} g_i(\bar{x}) &= -s_i x_i - \sum_{j \ne i} x_j \\ &= -s_i a_i + \bar{d}_k - a_k + \bar{d}_k - \ \ldots \ - a_n + \bar{d}_k \\ &= -s_i a_i - (a_k + \ \ldots \ + a_n) + c_k \bar{d}_k \\ &= -s_i a_i - (a_k + \ \ldots \ + a_n) + d_k = 0 \end{aligned}$$

Hence, $\bar{x}$ satisfies (b) in (2.7).

Since $g_j(\bar{x}) = 0$ for $1 \le j \le k-1$, $j \ne i$, $u_j = 0$ for $k \le j \le n$, $j \ne i$, and $g_i(x) = 0$ then $u_j x_j = 0$ for all j. Consequently, $\bar{x}$ and u satisfy (c) in (2.7).

By properties of the index k in Algorithm 2.1 and Lemma 2.5,

$$\bar{d}_k \ge \bar{d}_j > a_j \text{ for } 1 \le j \le k-1, \quad j \ne i$$

Thus, $u_j = 2(\bar{d}_k - a_j) > 0$ for $1 \le j \le k-1$, $j \ne i$. Moreover, $u_i = 2\bar{d}_k > 0$, and $u_k = \ldots = u_n = 0$. Hence, u satisfy (d) in (2.7). Therefore, $\bar{x}$ and u satisfies the Kuhn–Tucker conditions (2.7) and the result follows. ∎

Up to this point we have assumed (2.4) to study the properties of Algorithm 2.1. To finish this section we now present an algorithm that allows negative off-diagonal elements. However, we still want to keep the ascending order assumption, i.e., $|a_1| \le \ldots \le |a_j| \le \ldots \le |a_n|$, $(j \ne i)$. This assumption will be later removed in Section 4.

Algorithm 2.2. *Given a the ith row of A, and an index i, set* $x_i = a_i - s_i \bar{d}_k$.

```
for $1 \le j \le k-1 \quad (j \ne i)$
  $x_j = 0$
end
for $k \le j \le n \quad (j \ne i)$
  if $a_j > 0$ then $x_j = a_j - \bar{d}_k$
    else $x_j = a_j + \bar{d}_k$
  end
end
```

where $\bar{d}_k = d_k / c_k$,

$$d_k = \sum_{j=k}^{n} |a_j| + s_i a_i, \quad (j \ne i)$$

and $k \ne i$ *is the smallest integer* $1 \le k \le n$ *such that* $|a_k| > 0$ *and* $\bar{d}_k \le |a_k|$.

All the lemmas, comments and remarks of this section can be extended in a direct fashion to Algorithm 2.2, and can be summarized in the following main theorem.

Theorem 2.1. *Given $a \in \mathbb{R}^n$ and an index i, the unique solution $\bar{x}$ to either problem (2.2) or problem (2.3) is characterized as follows:*

1. *If $a_i \geq 0$ when projecting on C_i^+ or $a_i \leq 0$ when projecting on C_i^- and $|a_i| \geq \sum_{j \neq i} |a_j|$ then $\bar{x} = a$.*
2. *If $a_i \geq 0$ when projecting on C_i^+ or $a_i \leq 0$ when projecting on C_i^- and $|a_i| < \sum_{j \neq i} |a_j|$ then $\bar{d}_k$ exists and $\bar{x}$ is generated by Algorithm 2.2.*
3. *If $a_i < 0$ when projecting on C_i^+ or $a_i > 0$ when projecting on C_i^- and $|a_i| > \sum_{j \neq i} |a_j|$ then $\bar{x} = 0$.*
4. *If $a_i < 0$ when projecting on C_i^+ or $a_i > 0$ when projecting on C_i^- and $|a_i| \leq \sum_{j \neq i} |a_j|$ then:*
   - *If $|a_i| > |a_j|$ for $1 \leq j \leq n$, $j \neq i$, then $\bar{x} = 0$.*
   - *If there exists $j \neq i$ such that $|a_j| \geq |a_i|$ then $\bar{d}_k$ exists and $\bar{x}$ is generated by Algorithm 2.2.*

Notice that Theorem 2.1 characterizes the solution of problems (2.2) and (2.3) for all possible diagonal and off-diagonal values. Moreover, the implementation of our algorithms, described in Section 4, is based directly on Theorem 2.1.

3. Projection onto SDD^+

In order to solve problem (1.1), including the symmetric constraint $X \in S$, the n rows of A cannot be treated independently as we did in the previous section for the non-symmetric case. Therefore, we now apply Dykstra's alternating projection method [6,12]. The main idea is to project cyclically onto the sets S and DD^+ whose nonempty intersection is the feasible region of (1.1).

In Section 2 we characterized the projection P_{DD^+} onto the convex cone DD^+. The problem of finding the closest symmetric matrix to a given matrix A, in the Frobenius norm, has a simple solution [19,20] given by

$$P_S(A) = \frac{A + A^{\mathrm{T}}}{2}$$

We now present our version of Dykstra's alternating projection algorithm to solve problem (1.1).

Algorithm 3.1. *Given $A \in \mathbb{R}^{n \times n}$, set $G_0 = A$, $I_0 = [0]_{n \times n}$ and $j = 1$.*

```
Step 1. Set $P_S(G_{j-1}) = \frac{G_{j-1} + G_{j-1}^t}{2}$
Step 2. Set $G_j = P_{DD^+}(P_S(G_{j-1}) - I_{j-1})$
Step 3. If convergence tests are satisfied then stop with the solution estimate $G_j$. Else set
        $I_j = G_j - (P_S(G_{j-1}) - I_{j-1})$, $j = j + 1$, and go to Step 1.
```

Here I_j denotes the increment, associated with DD^+, introduced by Dykstra [12]. Since S is a subspace, there is no need to consider the increment associated with S to guarantee convergence to the solution of (1.1) (see [6,13]).

In practice, the algorithm is usually stopped at Step 3 whenever the distance between two consecutive projections onto the same set reaches a pre-stablished tolerance. For example, the process might be stopped when

$$\|G_{j+1} - G_j\|_{\mathrm{F}} \leq \mathrm{TOL} \tag{3.1}$$

where TOL is a given positive number.

Using the convergence result established by Boyle and Dykstra [6, Theorem 2], we obtain the following theorem for Algorithm 3.1.

Theorem 3.1. *For any $A \in \mathbb{R}^{n\times n}$ the sequences $\{P_S(G_j)\}$ and $\{G_j\}$ generated by Algorithm 3.1 converge in the Frobenius norm to the unique solution of problem (1.1).*

Since DD^+ is a convex polyhedral cone, the rate of convergence of Algortihm 3.1 is linear and the factor of convergence depends on the angle between the active faces at the solution. See Deutsch and Hundal [11] for additional details.

4. Numerical results

All experiments in this section were run on an HP Apollo 735/125 workstation in double precision Fortran. The iterations in Algorithm 3.1 were stopped when (3.1) was satisfied for TOL $= 10^{-7}$ as shown below. To find $P_{DD^+}(A)$ we need to project each row of A onto C_i^+. We now present two different implementations of Algorithm 2.2 to obtain the n independent projections at each cycle of Algorithm 3.1. The first one (Q version) makes use of the quick sort algorithm to sort each row of A before we project onto C_i^+. The second one (NQ version) achieves the same goal of projecting the rows of A onto C_i^+ without sorting the rows in advance.

4.1. *Q version*

In this case, we sort each row of A in advance using the quick sort algorithm described in [28, pp. 115–131]. Then, we find $\bar{d}_k$ for each row as in Algorithm 2.2, and we project onto C_i^+ using the following procedure:

- $x_i = a_i + \bar{d}_k$
- $a_j \geq 0, \quad (j \neq i)$ then:

$$\begin{cases} x_j = a_j - \bar{d}_k & \text{if } x_j \geq 0 \\ \text{if } x_j < 0 & \text{then } x_j = 0 \end{cases}$$

- If $a_j < 0, \quad (j \neq i)$ then:

$$\begin{cases} x_j = a_j + \bar{d}_k & \text{if } x_j \leq 0 \\ \text{if } x_j > 0 & \text{then } x_j = 0 \end{cases}$$

It is worth mentioning that the quick sort algorithm that we are using is recursive and requires $O(n \log n)$ comparisons, on average, to sort each row of A. Therefore, using the Q version, Algorithm 3.1 requires on average $O(n^2 \log n)$ floating point operations per iteration.

4.2. NQ version

In this case, we find $\bar{d}_k$ as in Algorithm 2.2 without sorting the rows of A in advance. First, we need to introduce some initial variables:

$$\begin{aligned} d(NQ) &= \sum_{j\neq i} |a_j| - a_i \\ c(NQ) &= n - cdf \\ \bar{d}(NQ) &= d(NQ)/c(NQ) \\ s &= 1 \end{aligned}$$

where cdf is the number of zero off-diagonal components of the vector a.

Algorithm 4.1. *(Computing $\bar{d}_k$)*

```
Do while $s = 1$
$s = 0$
Do for $j = 1, n$  $(j \neq i)$  either Step 1 or Step 2
(1) If $a_j > 0$                      (2) If $a_j < 0$
  $Au = a_j - \bar{d}(NQ)$                  $Au = a_j + \bar{d}(NQ)$
    If $Au < 0$                          If $Au > 0$
    $d(NQ) = d(NQ) - |a_j|$                $d(NQ) = d(NQ) - |a_j|$
    $c(NQ) = c(NQ) - 1$                  $c(NQ) = c(NQ) - 1$
    $a_j = 0$                            $a_j = 0$
    $s = 1$                              $s = 1$
    end if                             end if
  end if                             end if
$\bar{d}_k = \bar{d}(NQ) = d(NQ)/c(NQ)$
end while
```

Once $\bar{d}_k = \bar{d}(NQ)$ has been found, to obtain the projection $\bar{x}$ onto C_i^+ we use the following procedure:

- $x_i = a_i + \bar{d}(NQ)$.
- If $a_j = 0$, $(j \neq i)$ then $x_j = 0$.
- If $a_j > 0$, $(j \neq i)$ then $x_j = a_j - \bar{d}(NQ)$.
- If $a_j < 0$, $(j \neq i)$ then $x_j = a_j + \bar{d}(NQ)$.

Remark 1

- At the end of Algorithm 4.1, when $s = 0$, the value of $d(NQ)$ is the difference between a_i and the summation of all the components a_j of a such that $\bar{d}(NQ) \leq |a_j|$, $(j \neq i)$, where $c(NQ)$ is the number of components of a that have been taking into account in $d(NQ)$.
- The components of a that are ignored to compute $d(NQ)$ will be zero in the solution vector $\bar{x}$.
- The size of the diagonal component a_i is increased and the size of the off-diagonal components is decreased by the factor $\bar{d}(NQ)$ to obtain diagonal dominance with equality in the solution vector $\bar{x}$.

Table 1. Results for Experiment 1

		Q **Version**			NQ **Version**		
TOL	N	IT	Time	Error	IT	Time	Error
10^{-7}	100	30	1.88	8.1×10^{-8}	30	1.4	8.1×10^{-8}
10^{-7}	400	38	41.03	5.26×10^{-8}	38	30.1	5.26×10^{-8}

Therefore, $\bar{d}(NQ)$ obtained in Algorithm 4.1 and $\bar{d}_k$ obtained in Algorithm 2.2 are equal, and so the solution vector $\bar{x}$ using either the Q version or the NQ version coincide. For details see [24].

In the worst possible case, the NQ version of Algorithm 2.2 requires $O(n^2)$ comparisons per row. Hence, using the NQ version, Algorithm 3.1 requires in the worst case $O(n^3)$ floating point operations per iteration. However, in practice it only requires $O(n)$ comparisons per row, and Algorithm 3.1 requires only $O(n^2)$ operations per iteration. We have never seen an example for which Algorithm 3.1 with the NQ version requires $O(n^3)$ operations per cycle. Nevertheless, we would like to stress that we have not presented in this work any theoretical result to support this claim.

4.3. Experiments

In our first experiment we choose a symmetric matrix A given by

$$A = \begin{pmatrix} n & n & n & . & . & . & n \\ n & 2n+2 & -1 & -1 & . & . & -1 \\ n & -1 & 2n+2 & -1 & . & . & -1 \\ . & . & . & . & . & . & . \\ n & -1 & -1 & . & . & -1 & 2n+2 \end{pmatrix}$$

Notice that A is diagonally dominant except for the first row and column. In this case, the solution X to problem (1.1) is given by (see [24]):

$$X = \begin{pmatrix} n+2\beta & n-\beta & n-\beta & . & . & . & n-\beta \\ n-\beta & 2n+2 & -1 & -1 & . & . & -1 \\ n-\beta & -1 & 2n+2 & -1 & . & . & -1 \\ . & . & . & . & . & . & . \\ n-\beta & -1 & -1 & . & . & -1 & 2n+2 \end{pmatrix}$$

where $\beta = (n^2 - 2n)/(n+1)$.

Table 1 shows the CPU time in seconds (Time), the number of iterations or cycles (IT) required by Algorithm 3.1 with both implementations (Q and NQ), and the distance (Error) in the Frobenius norm between the output matrix and the exact solution X.

We observe that the number of iterations and the quality of the approximate solution, for both implementations, are the same. We also observe that the NQ version requires less computational work than the Q version to complete the same number of cycles. Notice also that the $(n-1) \times (n-1)$ lower right block of A remains unchanged in the matrix X as expected.

In the second experiment the initial matrix A is defined as $a_{ij} = i$ for $1 \leq i, j \leq n$. In this case A is not symmetric and clearly not diagonally dominant. Table 2 shows the results

Table 2. Results for Experiment 2

		Q **Version**		NQ **Version**	
TOL	N	IT	Time	IT	Time
10^{-7}	100	530	94.0	530	47.7
10^{-7}	200	1 169	1 304.88	1 169	497.89
10^{-7}	400	2 653	19 332.61	2 653	4 648.68
10^{-7}	500	3 460	49 346.05	3 460	15 209.21

for this experiment. We observe that the number of cycles increases linearly as n increases. It is also clear that the NQ version of Algorithm (3.1) outperforms the Q version in CPU time. Indeed, the NQ version requires approximately one third of the computational work required by the Q version to complete the same number of cycles. We have observed this remarkable difference in most experiments we have run.

5. Concluding remarks

Using Dykstra's alternating projection method we have presented an iterative process that finds the nearest symmetric diagonally dominant matrix to a given matrix. This iterative process involves, at every cycle, the projection onto the cone of diagonally dominant matrices with positive diagonal. This projection has been characterized, and we have also presented two different implementations of the associated algorithm. One of the implementations (NQ version) turns out to be much faster than the other one (Q version).

In Section 2 we simultaneously discussed the projection onto the cone of diagonally dominant matrices with negative diagonal. Moreover, as a by product, we can solve the more general problem of projecting onto the set of diagonally dominant (DD) matrices, i.e., we can also solve the following optimization problem:

$$\min \{\|X - A\|_F^2 : X \in DD\} \tag{5.1}$$

where A is given and $X \in DD$ if $|x_{ii}| \geq \sum_{j \neq i} |x_{ij}|$ for all $1 \leq i \leq n$. In fact, using Lemma 2.1, for each row a_i of A, we obtain:

(1) If $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$ then $\bar{x}_i = a_i$.
(2) If $|a_{ii}| < \sum_{j \neq i} |a_{ij}|$ then

  (a) If $a_{ii} \geq 0$ then $x_{ii} \geq 0$ and $\bar{x}_i \in C_i^+$.
  (b) If $a_{ii} \leq 0$ then $x_{ii} \leq 0$ and $\bar{x}_i \in C_i^-$.

Since we already know how to project onto C_i^+ and C_i^- (see Theorem 2.1), we also know how to solve problem (5.1). Notice that if $a_{ii} = 0$ then there are two possible *closest* diagonally dominant rows x_i.

In practice, an interesting extension is to force each row of X to be dominant by some factor $\delta_i > 0$, i.e.,

$$x_{ii} \geq \sum_{j \neq i} |x_{ij}| + \delta_i \tag{5.2}$$

In this case, the matrix X is in the interior of the SDD^+ cone, and a bound on the condition number can be obtained (see [29]). The closest symmetric matrix that satisfies (5.2) for each row, is given by

$$X = P_{SDD^+}(A - \hat{D}) + \hat{D}$$

where $\hat{D} = \mathrm{diag}(\delta_i)$, and $P_{SDD^+}(A - \hat{D})$ is obtained by Algorithm 3.1. In fact, by the definition of projection we have that

$$\|A - X\|_{\mathrm{F}} = \|(A - \hat{D}) - P_{SDD^+}(A - \hat{D})\|_{\mathrm{F}} \leq \|(A - \hat{D}) - Y\|_{\mathrm{F}} = \|A - (Y + \hat{D})\|_{\mathrm{F}}$$

for all $Y \in SDD^+$.

Finally, we note that large-scale problems are usually sparse and as we can observe from our algorithms, the sparsity pattern of the given matrix A does not change during the process. As a consequence, the computational cost per iteration of the Q version and the NQ version can be reduced in practice to $O((nz)^2 \log(nz))$ and $O((nz)^2)$, respectively, where nz is the number of non-zero off-diagonal elements in A. Since the number of cycles increases linearly with n, then the total computational cost can be reduced to $O(n(nz)^2 \log(nz))$ and $O(n(nz)^2)$, respectively. Moreover, projections onto the set of diagonally dominant matrices and also onto the subspace of symmetric matrices are both highly parallelizable. This is in sharp contrast to the factorization techniques discussed in the Introduction. Parallel implementations of our algorithms that take advantage of the sparsity structure of the matrix are an interesting topic that deserves further investigation.

Acknowledgements

This work was supported by CONICIT, Numerical Analysis Laboratory at UCV, and NSF Grants HRD-9450448 and CDA-94117362.

We wish to thank the referees for providing careful comments which led to substantial improvement in the presentation of the paper. In particular, we thank one of the referees for suggesting the extension to matrices that are dominant by some given positive factors.

REFERENCES

1. J. C. Allwright. Positive semidefinite matrices: characterization via conical hulls and least-squares solution of a matrix equation. *SIAM J. Control and Optimization*, 26, 537–555, 1988.
2. M. Avriel. *Nonlinear Programming: Analysis and Methods.* Prentice-Hall, New Jersey, 1976.
3. G. P. Barker. Theory of cones. *Linear Algebra and Applications*, 39, 263–291, 1981.
4. G. P. Barker and D. Carlson. Cones of diagonally dominant matrices. *Pacific J. Math.*, 57, 15–32, 1975.
5. G. P. Barker and R. Loewy. The structure of cones of matrices. *Linear Algebra and Applications*, 12, 87–94, 1975.
6. J. P. Boyle and R. L. Dykstra. *A method for finding projections onto the intersections of convex sets in Hilbert spaces*, volume 37. Springer–Verlag, 1986.
7. W. Cheney and A. Goldstein. Proximity maps for convex sets. *Proc. Amer. Math. Soc.*, 10, 448–450, 1959.
8. S. H. Cheng and N. J. Higham. A modified Cholesky algorithm based on a symmetric indefinite factorization. Technical Report No. 289, Manchester Centre for Computational Mathematics, Manchester, England, 1996. *SIAM J. Matrix Anal. Appl.*, 19, 1097–1110, 1998.
9. G. B. Dantzig. Deriving an utility function for the economy. Technical Report SOL 85-6R, Department of Operations Research, Stanford University, Stanford, CA, 1985.
10. F. Deutsch. The method of alternating orthogonal projections. In S.P. Singh, editor, *Approximation Theory, Spline Functions and Applications*, pages 105–121. Kluwer Academic Publishers, Netherlands, 1992.

11. F. Deutsch and H. Hundal. The rate of convergence of Dykstra's cyclic projections algorithm: the polyhedral case. *Numer. Funct. Anal. Optimiz.*, 15, 537–565, 1994.
12. R. L. Dykstra. An algorithm for restricted least-squares regression. *J. Amer. Stat. Assoc.*, 78, 837–842, 1983.
13. R. Escalante and M. Raydan. Dykstra's algorithm for a constrained least-squares matrix problem. *Numerical Linear Algebra with Applications*, 3, 459–471, 1996.
14. R. Escalante and M. Raydan. On Dykstra's algorithm for constrained least-squares rectangular matrix problems. *Computers and Mathematics with Applications*, 35, 73–79, 1998.
15. M. Fiedler and V. Ptak. Diagonally dominant matrices. *Czech. Math. J.*, 17(92), 420–433, 1967.
16. R. Fletcher. A nonlinear programming problem in statistics (educational testing). *SIAM J. Sci. Stat. Comp.*, 2, 257–267, 1981.
17. R. Fletcher. Semi-definite matrix constraints in optimization. *SIAM J. Control and Opt.*, 23, 493–513, 1985.
18. N. Gaffke and R. Mathar. A cyclic projection algorithm via duality. *Metrika*, 36, 29–54, 1989.
19. T. L. Hayden and J. Wells. Approximation by matrices positive semidefinite on a subspace. *Linear Algebra and Applications*, 109, 115–130, 1988.
20. N. J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra and Applications*, 103, 103–118, 1988.
21. R. D. Hill and S. R. Waters. On the cone of positive semidefinite matrices. *Linear Algebra and Applications*, 90, 81–88, 1987.
22. H. Hu. Positive definite constrained least-squares estimation of matrices. *Linear Algebra and Applications*, 229, 167–174, 1995.
23. H. Hu and I. Olkin. A numerical procedure for finding the positive definite matrix closest to a patterned matrix. *Statistics and Probability Letters*, 12, 511–515, 1991.
24. M. Mendoza. Proyección sobre el conjunto de las matrices simétricas diagonal dominantes, 1996. Master Thesis, Departamento de Computacion, Universidad Central de Venezuela, Caracas, Venezuela.
25. J. von Neumann. Functional operators vol. II. The geometry of orthogonal spaces. *Annals of Math. Studies*, 22, 1950. Princeton University Press. This is a reprint of mineographed lecture notes first distributed in 1933.
26. W. Murray P. E. Gill and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.
27. R. B. Schnabel and E. Eskow. A new modified Cholesky factorization. *SIAM J. Sci. Stat. Comput.*, 11, 1136–1158, 1990.
28. R. Sedgewick. *Algorithms in C.* Adisson Wesley, London, 1990.
29. J. M. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and Applications*, 11, 3–5, 1975.
30. S. Hong W. Glunt, T. L. Hayden and J. Wells. An alternating projection algorithm for computing the nearest euclidean distance matrix. *SIAM J. Matrix Anal. Appl.*, 11, 589–600, 1990.