

A DIRECT PROJECTION METHOD FOR SPARSE LINEAR SYSTEMS*

MICHELE BENZI[†] AND CARL D. MEYER[‡]

Abstract. An oblique projection method is adapted to solve large, sparse, unstructured systems of linear equations. This row-projection technique is a direct method which can be interpreted as an oblique Kaczmarz-type algorithm, and is also related to other standard solution methods. When a sparsity-preserving pivoting strategy is incorporated, it is demonstrated that the technique can be superior, in terms of both fill-in and arithmetic complexity, to more standard sparse algorithms based on gaussian elimination. This is especially true for systems arising from stiff ordinary differential equations problems in chemical kinetics studies.

Key words. sparse systems of linear equations, sparse matrices, gaussian elimination, projection methods, direct methods, Harwell–Boeing collection

AMS subject classifications. 65F05, 65F30, 65F99

1. Introduction. This paper is concerned with a direct projection algorithm for solving systems of linear algebraic equations. The method is similar to others that have appeared in the literature, and is closely related to such standard techniques as LU decomposition, orthogonalization, and conjugate directions (see [1]–[2], [9]–[18]). After describing the basic algorithm and its geometrical interpretation, we compare it briefly with other (related) methods. Next, we show how the basic algorithm can be adapted to handle sparsity in the coefficient matrix, and we report on numerical experiments with test matrices from the Harwell–Boeing collection. Our approach makes use of threshold pivoting and drop tolerances to preserve sparsity, possibly combined with iterative refinement to improve accuracy in the computed solution. While we do not recommend direct projection techniques as a general purpose sparse solver, we conclude that there are realistic problems for which this approach is well suited.

This work complements an independent study by Tuma [18], who considered an alternative formulation of this algorithm.

2. Description of the basic direct projection method (DPM). Consider a system of linear equations $Ax = b$; for simplicity we shall assume that A is a nonsingular $n \times n$ real matrix, but the basic technique considered in this paper can be applied to more general situations. Let a_i^T denote the i th row of A , and let $\mathcal{H}_i = \{x \mid a_i^T x = b_i\}$ denote the hyperplane defined by the i th equation of the system. If

$$A_k = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_k^T \end{pmatrix}$$

*Received by the editors August 20, 1993; accepted for publication (in revised form) July 19, 1994.

[†]Dipartimento di Matematica, Università degli Studi di Bologna, 40127 Bologna, Italy (benzi@dm.unibo.it).

[‡]Mathematics Department, North Carolina State University, Raleigh, North Carolina 27695-8205 (meyer@math.ncsu.edu). This work was supported in part by National Science Foundation grants DMS-9020915 and DMS-9403224, and by the North Carolina Super Computing Center.

denotes the matrix containing the first k rows of A , and if $\mathcal{N}_k = N(A_k)$ denotes the null space of A_k , then

$$\mathcal{N}_1 \supset \mathcal{N}_2 \supset \cdots \supset \mathcal{N}_{n-1} \supset \mathcal{N}_n = \{0\}.$$

In order to solve the system of equations, we begin by constructing a set of *null vectors*

$$\mathcal{N} = \{z_2, z_3, \dots, z_n\}$$

with the property that $z_k \in \mathcal{N}_{k-1}$ but $z_k \notin \mathcal{N}_k$ (and $z_i \neq 0$); this implies that the vectors in \mathcal{N} are linearly independent. In particular, $\{z_k, z_{k+1}, \dots, z_n\}$ form a basis for \mathcal{N}_{k-1} . The null vectors can be computed as follows.

1. Begin with any basis for $\mathcal{N}_1 = a_1^\perp$, say $\mathcal{Z}_1 = \{z_2^{(1)}, z_3^{(1)}, \dots, z_n^{(1)}\}$. Because $\dim \mathcal{N}_2 = n - 2$, we must have $a_2^T z_i^{(1)} \neq 0$ for some $z_i^{(1)}$, so we can arrange the elements of \mathcal{Z}_1 so that $a_2^T z_2^{(1)} \neq 0$. However, to promote numerical stability, it is preferable to invoke a “pivoting strategy” which chooses $z_2^{(1)}$ to be the vector such that

$$|a_2^T z_2^{(1)}| = \max_{2 \leq i \leq n} |a_2^T z_i^{(1)}|.$$

Geometrically, this means that we choose from \mathcal{Z}_1 the vector of maximum distance from a_2^\perp . This is also equivalent to a form of column pivoting on A , as we shall see. If we agree to “pivot,” then it is clear that $z_2^{(1)} \in \mathcal{N}_1$ but $z_2^{(1)} \notin \mathcal{N}_2$.

2. Successively project $z_2^{(1)}$ onto a_2^\perp through each of the remaining null vectors $z_3^{(1)}, z_4^{(1)}, \dots, z_n^{(1)}$ to produce the new set $\mathcal{Z}_2 = \{z_3^{(2)}, z_4^{(2)}, \dots, z_n^{(2)}\}$. It can be argued that because \mathcal{Z}_1 is a basis for \mathcal{N}_1 , we must have that \mathcal{Z}_2 is a basis for \mathcal{N}_2 . As before, $\dim \mathcal{N}_3 = n - 3$ implies that one of these z -vectors must be such that $a_3^T z_i^{(2)} \neq 0$. If the pivoting strategy is again employed to select $z_3^{(2)}$ such that

$$|a_3^T z_3^{(2)}| = \max_{3 \leq i \leq n} |a_3^T z_i^{(2)}|,$$

then $z_3^{(2)} \in \mathcal{N}_2$ but $z_3^{(2)} \notin \mathcal{N}_3$. Again, $z_3^{(2)}$ is the vector in \mathcal{Z}_2 which maximizes the distance from a_3^\perp .

3. Successively project $z_3^{(2)}$ onto a_3^\perp through $z_j^{(2)}$ for $j = 4, \dots, n$ to produce $\mathcal{Z}_3 = \{z_4^{(3)}, z_5^{(3)}, \dots, z_n^{(3)}\}$. This is a basis for \mathcal{N}_3 , and if $z_4^{(3)}$ is selected to be the vector such that $|a_4^T z_4^{(3)}| = \max_{4 \leq i \leq n} |a_4^T z_i^{(3)}|$, then $z_4^{(3)} \in \mathcal{N}_3$ but $z_4^{(3)} \notin \mathcal{N}_4$.

⋮

This process is repeated $n - 1$ times, and the last step is to project $z_{n-1}^{(n-2)}$ onto a_{n-1}^\perp through $z_n^{(n-2)}$ to produce $\mathcal{Z}_{n-1} = \{z_n^{(n-1)}\}$, which is a basis for \mathcal{N}_{n-1} . The pivot vectors $\mathcal{N} = \{z_2^{(1)}, z_3^{(2)}, \dots, z_n^{(n-1)}\}$ therefore constitute a set of null vectors satisfying $z_k^{(k-1)} \in \mathcal{N}_{k-1}$ but $z_k^{(k-1)} \notin \mathcal{N}_k$.

In practice, a simple normalization can be used as a safeguard against the risk of overflow, so the formal algorithm for computing null vectors is as follows.

THE NULL VECTOR ALGORITHM

Input: Any basis $\{z_2, z_3, \dots, z_n\}$ for $\mathcal{N}_1 = a_1^\perp$.

Output: A set \mathcal{N} of $n - 1$ null vectors together with a corresponding set of $n - 1$ pivots.

```

for  $i = 2 : n$ 
  for  $j = i : n$ 
     $\sigma_j \leftarrow a_i^T z_j$ 
  end
   $m \leftarrow \text{index of } \max_{i \leq j \leq n} |\sigma_j|$ 
   $p_i \leftarrow \sigma_m$  (the  $i$ th pivot)
  if  $m > i$  then interchange  $z_m$  and  $z_i$ 
    (the  $i$ th pivot column = the  $i$ th null vector)
  if  $\|z_i\|_\infty > \textit{tolerance}$  then  $z_i \leftarrow z_i / \|z_i\|_\infty$ 
  if  $i = n$  go to end
  for  $j = i + 1 : n$ 
     $z_j \leftarrow z_j - \frac{\sigma_j}{p_i} z_i$ 
    if  $\|z_j\|_\infty > \textit{tolerance}$  then  $z_j \leftarrow z_j / \|z_j\|_\infty$ 
  end
end

```

If e_i denotes the i th column of the identity matrix I , and if a_{1k} is the entry of maximal magnitude in a_1^T , then one choice for the initial basis for \mathcal{N}_1 is

$$z_{i+1}^{(1)} = \begin{cases} e_i & \text{if } a_{1i} = 0, \\ e_i - \frac{a_{1i}}{a_{1k}} e_k & \text{for } i \neq k \text{ and } a_{1i} \neq 0. \end{cases}$$

Notice that a_{1k} plays the role of the first pivot, so it makes sense to let $p_1 = a_{1k}$. Likewise, e_k is the first pivot vector, and hence we let $z_1 = e_k$. Geometrically, $\{z_2^{(1)}, \dots, z_n^{(1)}\}$ is obtained projecting the unit vector e_k of greatest distance from a_1^\perp through the other unit vectors e_i , with $i \neq k$.

The matrix

$$Z = [z_1, z_2, \dots, z_n]$$

together with the vector of pivots $p = (p_1, p_2, \dots, p_n)$ is intimately related with the LU factorization of A ; we discuss this in the next section. For now it suffices to observe that the final Z was obtained starting from the identity matrix $[e_1, e_2, \dots, e_n]$ and successively projecting its columns onto the null spaces \mathcal{N}_k associated with A . The choice of the identity matrix as the initial matrix is, in a sense, arbitrary, but is clearly justified in terms of simplicity and sparsity. Unless otherwise stated, it is assumed that the null vector algorithm is started with the identity matrix. With this choice it is not difficult to see that the above algorithm returns vectors $\mathcal{N} = \{z_2, z_3, \dots, z_n\}$ in which z_i contains no more than i nonzero entries. Of course, if a_1^T is sparse, then most of the initial vectors will be unit columns, and if A is a sparse matrix, then each pivot column will generally contain far fewer than i nonzero entries. This is one facet of the technique which we will try to exploit. Furthermore, notice that this null vector algorithm is a *row-generation method* in the sense that only one row of A is required at each step. Consequently, single rows of A can be brought from secondary memory or else generated one at a time only as they are needed, as in Kaczmarz's iterative orthogonal projection scheme (see Censor [5]).

It can be shown [2] that for a full $n \times n$ matrix, the null vector algorithm has computational complexity which is dominated by $\frac{1}{3}n^3$ multiplications (see next section for storage requirements). Furthermore, it is not hard to see that in exact arithmetic the null vector algorithm can be carried out without interchanges if and only if all leading principal minors of A are nonzero (see [2]). In practice, however, pivoting is necessary for numerical stability. Computer experiments indicate that this algorithm is at least as accurate as gaussian elimination (hereafter referred to as GE) with partial pivoting. It is also possible to consider alternative pivoting strategies, but the one we propose appears to be the most natural and practical. In §4 we discuss how pivoting can be used to preserve sparsity in the null vectors, without entirely sacrificing accuracy.

Once a set of null vectors $\mathcal{N} = \{z_2, \dots, z_n\}$ is known, the solution to $Ax = b$ can be easily obtained for any right-hand side b by a sequence of $n - 1$ projections along the z_i 's, starting from an arbitrary point on the first hyperplane \mathcal{H}_1 . The procedure is as follows.

1. Start with any $x_1 \in \mathcal{H}_1$, and project x_1 onto \mathcal{H}_2 along z_2 to obtain

$$x_2 = x_1 + \frac{b_2 - a_2^T x_1}{a_2^T z_2} z_2 = x_1 + \frac{b_2 - a_2^T x_1}{p_2} z_2 \in \mathcal{H}_1 \cap \mathcal{H}_2.$$

2. Project $x_2 \in \mathcal{H}_1 \cap \mathcal{H}_2$ onto \mathcal{H}_3 along z_3 to obtain

$$x_3 = x_2 + \frac{b_3 - a_3^T x_2}{a_3^T z_3} z_3 = x_2 + \frac{b_3 - a_3^T x_2}{p_3} z_3 \in \mathcal{H}_1 \cap \mathcal{H}_2 \cap \mathcal{H}_3.$$

⋮

- k. Project $x_k \in \bigcap_{i=1}^k \mathcal{H}_i$ onto \mathcal{H}_{k+1} along z_{k+1} to obtain

$$x_{k+1} = x_k + \frac{b_{k+1} - a_{k+1}^T x_k}{p_{k+1}} z_{k+1} \in \bigcap_{i=1}^{k+1} \mathcal{H}_i.$$

Note that each pivot $p_i = a_i^T z_i$ has already been computed by the null vector algorithm. It is evident that the solution of $Ax = b$ is produced after $n - 1$ of these projections, so we have the following formal algorithm.

THE SOLUTION ALGORITHM

Input: 1. A right-hand side b .
 2. A set of null vectors $\mathcal{N} = \{z_2, z_3, \dots, z_n\}$ for A together with the corresponding set of pivots $p_i = a_i^T z_i$ (e.g., the output from the null vector algorithm).

Output: The solution x to $Ax = b$.

```

x ← any vector in  $\mathcal{H}_1$ 
(a good choice is  $x \leftarrow (b_1/a_{1k})e_k$  where  $|a_{1k}| = \max_j |a_{1j}|$ )
for  $i = 2 : n$ 
   $x \leftarrow x + \frac{b_i - a_i^T x}{p_i} z_i$ 
end

```

The null vector algorithm in conjunction with the solution algorithm constitutes the basic direct projection method (hereafter referred to as DPM).

Clearly, once the null vectors and pivots have been computed, any right-hand side can be processed using no more than $O(n^2)$ flops for dense problems. If A is sparse, then significantly fewer flops are required, and if both A and the null vectors are sparse, then the flop count is reduced even further. The computation of sparse null vectors is one of our primary concerns (see §4). Furthermore, just as in the null vector algorithm, the solution process is a row-projection method requiring only one row of A at each step.

3. Relationship and comparison with other methods. There exist several alternative formulations of the scheme described in the previous section, and various authors have considered one or more of such forms and pointed out their relationship with standard methods such as matrix factorization and others. It is important to emphasize that although all these variants are (in a sense) mathematically equivalent, their properties and behavior can be quite different from the computational point of view.

Perhaps the earliest reference to a variant of DPM for general matrices can be found in Hestenes and Stiefel [11], in their landmark paper on the conjugate gradient method (see pp. 426–427). Other contributions are due to Purcell [15], Pietrzykowski [14], Householder [12], [13], Faddeev and Faddeeva [9], Stewart [17], Sloboda [16], and Abaffy, Broyden, and Spedicato [1]. For symmetric positive definite matrices, the algorithm essentially becomes a conjugate direction method already considered by Fox, Huskey, and Wilkinson [10] and by Hestenes and Stiefel [11]. Other references together with a brief historical overview of this general class of projection methods can be found in [2].

It was recognized early on that DPM can be interpreted in terms of an *implicit* triangular factorization of A . If the null vector algorithm is started with the identity matrix as described in §2, and if $Z = [z_1, z_2, \dots, z_n]$ is the matrix resulting from the output, then the product $L \equiv AZ$ is lower triangular with the pivots p_i on the main diagonal. Furthermore, if Q is the permutation matrix which represents the interchanges performed during the computation of the null vectors, then $Z = QW$ where W is unit upper triangular.¹ Therefore,

$$AQ = LW^{-1},$$

and we recognize in this factorization the familiar LU decomposition of AQ where $U = W^{-1}$. Since the upper triangular factor has 1's on the main diagonal, this is actually Crout's form of the LU decomposition. This shows that in exact arithmetic, no interchanges are required ($Q = I$) if and only if all leading principal minors of A are nonzero.

The null vector algorithm can be viewed as generating a sequence of $n-1$ matrices

$$Z_i = Z_0 P_1 E_1 \cdots P_i E_i, \quad i = 1, 2, \dots, n-1,$$

where P_i denotes a permutation matrix and E_i denotes an elimination matrix (Z_0 is the initial matrix; usually $Z_0 = I$). The i th stage starts with Z_{i-1} and ends with Z_i . The matrix P_i interchanges (if necessary) the i th and m th columns of Z_{i-1} , while E_i annihilates the last $n-i$ elements in the i th row of the matrix $AZ_{i-1}P_i$. Recall that the usual ("left") GE process is composed of $n-1$ stages in which the original

¹ Note that this is not necessarily true when a matrix other than the identity matrix is used to start the null vector algorithm.

matrix A is transformed into an upper triangular matrix U . If partial pivoting is used, then GE produces a sequence of matrices

$$\hat{A}_i = \hat{E}_i \hat{P}_i \cdots \hat{E}_1 \hat{P}_1 A, \quad i = 1, 2, \dots, n-1$$

where \hat{P}_i is a permutation matrix and \hat{E}_i is an elimination matrix that annihilates the last $n-i$ elements in the i th column of $\hat{P}_i \hat{A}_{i-1}$ where $\hat{A}_0 = A$ and $\hat{A}_{n-1} = U$ is an upper triangular matrix. Similarly, “right” GE transforms A into a lower triangular matrix L by generating a sequence of matrices

$$\tilde{A}_i = A \tilde{P}_1 \tilde{E}_1 \cdots \tilde{P}_i \tilde{E}_i, \quad i = 1, 2, \dots, n-1$$

where \tilde{P}_i is a permutation matrix and \tilde{E}_i is an elimination matrix that annihilates the last $n-i$ elements in the i th row of $\tilde{A}_{i-1} \tilde{P}_i$. Here $\tilde{A}_0 = A$ and $\tilde{A}_{n-1} = L$ is a lower triangular matrix. Note that this algorithm can be viewed as the usual GE performed on A^T .

It is easily seen that for $Z_0 = I$ the null vector process is mathematically equivalent to “right” GE. More generally, if $Z_0 \neq I$, then the null vector algorithm is equivalent to “right” GE applied to AZ_0 . This illustrates the close relationship between the null vector algorithm and the LU factorization of A with column pivoting. The most visible difference between the DPM approach and GE is that the matrix L is neither computed nor stored except for the entries on the main diagonal (the pivots). The triangularization is performed only implicitly. While mathematically equivalent, the two methods may behave very differently in practice, especially in applications involving large sparse matrices. While GE stores and updates both \hat{A}_i and the product $\hat{E}_1^{-1} \cdots \hat{E}_i^{-1}$, the null vector algorithm stores A (one row at a time, possibly) and stores and updates the product $E_1 \cdots E_i$. This points to the fact that when applied to sparse matrices, the two methods produce very different fill-in.

In GE, the triangular factors L and U are used in the solution process, and they are both needed if the system has to be solved again subsequently for different right-hand sides. In contrast, DPM avoids forming L explicitly or inverting W (or Z), and is able to solve the system for arbitrary right-hand sides using W , the pivots, and the strictly lower triangular part of AQ . Exactly as in GE, vectors containing the right-hand sides can be overwritten by the corresponding solution vectors. The array which originally contains A can be overwritten by W in the strictly upper triangular part (there is no need to store the 1's on the main diagonal of W), by the pivots on the main diagonal, and by the strictly lower triangular part of AQ . Notice that for a sparse matrix stored in full form, this means that fill-in can occur only in the upper triangular part of the array.

The fact that almost half of matrix A is not altered can be advantageous in all those situations where a copy of matrix A is needed for some reason; for example, if iterative refinement of the solution is called for.

If the system must be solved for a single right-hand side, storage requirements can be greatly reduced by overlapping the solution process with the computation of the null vectors. Once a pivot null vector has been computed and used to update the solution, it is no longer needed for subsequent calculations and can be dropped. The same is true of the corresponding row of the coefficient matrix. For dense matrices which can be generated one row at a time, this scheme requires $O(n^2/4)$ storage locations (see [2], [15]–[16]).

The observation that Z is the inverse of a factor in the triangular factorization of A is quite important. If A is symmetric positive definite, then the computation

of the null vectors can be done without interchanges, so one has

$$AZ = Z^{-T}D \quad \text{where} \quad D = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n \end{pmatrix},$$

and therefore

$$Z^T AZ = D.$$

In other words, the null vectors form a set of conjugate directions with respect to A . Notice also that $A^{-1} = ZD^{-1}Z^T$, which implies that the solution process can now be replaced by the matrix-vector products required to compute $x = ZD^{-1}Z^Tb$. This realization is important for parallel and vector computation because it is known that triangular solves do not vectorize or parallelize as easily as matrix-vector products (also see §6).

When working with sparse matrices, it often happens that the factors are quite sparse, but their inverses are full. As Z , in the symmetric positive definite case, is the inverse of the triangular factor in the (root-free) Cholesky decomposition of A , one might expect that considerably more fill-in will be generated by the DPM approach than with standard GE. While this is true in many cases, it is also true that fill-in can be kept (to some extent) “small” by suitable sparsity-preserving strategies (see §4). Furthermore, it often happens that many entries in the inverse factors are very small, especially far from the main diagonal; this observation led us to make use of drop tolerances, which can sometimes reduce fill-in in a very substantial manner.

However, this method should not be used to factor general full symmetric positive definite matrices because it requires about twice the work as the Cholesky decomposition method. Indeed, DPM is unable to exploit symmetry in the coefficient matrix when solving a linear system.

Notice that in the symmetric positive definite case DPM becomes an orthogonalization method; the null vectors are obtained from the unit vectors e_1, e_2, \dots, e_n by the Gram-Schmidt orthogonalization process with inner product $\langle x, y \rangle = x^T Ay$. This procedure was first proposed by Fox, Huskey, and Wilkinson [10] and also studied by Hestenes and Stiefel [11].

It is worthwhile observing that DPM can also be related to orthogonalization schemes in the following way. Let Z be any nonsingular matrix such that $AZ = L$ where L is a lower triangular matrix. Then the columns of Z form a set of null vectors for A (in the sense of §2), and any linear system $Ax = b$ is easily solved by the solution algorithm presented in §2. The null vector algorithm is just one way to compute such a Z , but other methods could be used. For instance, Z could be a product of Householder or Givens transformations, and the successive projections in the solution process would then be taken along orthogonal directions. This method amounts to an implicit LQ factorization of A where only $Z = Q^T$ and the diagonal entries of L are explicitly computed and stored.

4. Application to sparse matrices. Let us consider two model problems which shed some light on the main issues involved. If

$$(4.1) \quad A = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}$$

then it is obvious that GE does not introduce any fill-in (no pivoting is necessary, of course). If DPM is used without pivoting, and if all pivot vectors are kept because several right-hand sides must be processed, then the Z matrix produced has entries

$$Z_{ij} = \begin{cases} 0 & \text{if } i > j, \\ 1 & \text{if } i = j, \\ i/j & \text{if } i < j, \end{cases}$$

and fill-in is $O(n^2/2)$. The situation is improved if a sparsity-preserving pivoting strategy is used within the projections (see below), in which case no fill-in is introduced. However, pivoting for sparsity is very time-consuming, and hence DPM is still not competitive with GE on this problem. Similar considerations apply to more general banded matrices.

To see that there are problems for which DPM is well suited, consider the unstructured 5×5 matrix

$$A = \begin{pmatrix} 1 & -3 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 & 0 \\ 5 & 0 & -5 & 0 & 6 \end{pmatrix}$$

containing 11 nonzero entries. We will use this example to illustrate pivoting for sparsity. This matrix has no special features, and it is used by Duff, Grimes, and Lewis [7] to illustrate the storage scheme used for the test matrices adopted for the Harwell–Boeing collection. We will use this example to illustrate the “pivoting for sparsity” strategy. The basic (and most effective) technique to preserve sparsity in the context of GE is the Markowitz strategy—see Duff, Erisman, and Reid [6]. This consists in selecting at each stage of GE the nonzero entry in the active matrix which minimizes the Markowitz cost function

$$\text{cost}(a_{ij}^{(k)}) = (r_i - 1)(c_j - 1)$$

where r_i is the number of nonzero entries in the active part of the i th row and c_j is the number of nonzero entries in the active part of the j th column. Row and column interchanges are used to bring the selected entry in position $a_{kk}^{(k)}$ (the pivot). In practice one restricts the search to those candidates which satisfy an inequality of the kind

$$|a_{ij}^{(k)}| \geq u \cdot \max_{l \geq k} |a_{il}^{(k)}| \quad \text{or} \quad |a_{ij}^{(k)}| \geq u \cdot \max_{l \geq k} |a_{lj}^{(k)}|$$

where the “threshold parameter” u is between 0 and 1. This allows a compromise between pivoting for sparsity only ($u = 0$) and pivoting for stability only ($u = 1$). A good tentative value for u is $u = 0.1$. This technique is known to be very effective on many general sparse matrices. If we apply GE with the Markowitz strategy (with $u = 0$) to the 5×5 matrix A above, we obtain the factorization $PAQ = LU$ where P and Q are permutation matrices and

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 0 & 1 & 0 \\ -\frac{5}{2} & -\frac{5}{2} & 0 & \frac{1}{4} & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 3 & 0 \\ 0 & 0 & -\frac{3}{2} & 0 & 0 \\ 0 & 0 & 0 & -4 & 4 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}.$$

For GE, the 1's on the main diagonal of L need not be stored, and we see that 11 nonzeros must be stored if a solution is required for several right-hand sides (just 7 nonzeros if only one right-hand side must be solved for). If for some reason (e.g., iterative refinement) a copy of A must be kept, the total storage requirement for GE is 22 nonzeros.

Now suppose that DPM is used. To preserve sparsity we use as pivot column the sparsest null vector z_i such that $a_i^T z_i \neq 0$ at the i th step. A balance between the need for sparsity and that for stability can be reached by introducing a threshold parameter u between 0 and 1 and restricting the search to all potential candidates z_k which satisfy

$$|a_i^T z_k| \geq u \cdot \max_{i \leq j \leq n} |a_i^T z_j|.$$

Again, $u = 0$ means pivoting for sparsity only while $u = 1$ means pivoting for stability only (it turns out that $u = 0.1$ is, again, a good choice for the threshold parameter). Like Markowitz with threshold pivoting, this is a local, heuristic strategy which attempts, at each step, to minimize fill-in and arithmetic operations.

Begin by permuting the rows of A in order of increasing density.

$$PA = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 3 \\ 0 & 4 & 0 & -4 & 0 \\ 1 & -3 & 0 & -1 & 0 \\ 5 & 0 & -5 & 0 & 6 \end{pmatrix}.$$

If we include pivoting for sparsity in the null vector algorithm (with $u = 0$), the result of applying the algorithm to PA is

$$Z = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

which is a row-permutation of the unit upper triangular matrix

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{2}{3} \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

There are just two essential nonzeros in this matrix.

If Q is the permutation matrix such that $QZ = W$, we have the implicit LU decomposition $PAQ = LW^{-1}$, but, of course, L is not computed nor stored, and W is not actually inverted. For our example we have

$$PAQ = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 3 \\ 0 & 0 & 4 & -4 & 0 \\ 1 & 0 & -3 & -1 & 0 \\ 5 & -5 & 0 & 0 & 6 \end{pmatrix}.$$

If a solution needs to be extracted for several right-hand sides, then the essential nonzeros of W and the strictly lower triangular part of PAQ must be kept. Also, the five pivots $p_1 = 2$, $p_2 = -2$, $p_3 = 4$, $p_4 = -4$, and $p_5 = 8/3$ are needed. Thus, a total of $2 + 4 + 5 = 11$ nonzeros must be stored, the same as in the case of GE with Markowitz pivoting. However, assume now that a copy of the original matrix A must be kept for later use; since a portion of A (the strictly lower triangular part of PAQ) must be kept in storage anyway, only an additional 7 nonzeros, instead of 11, must be stored. In this case the total storage required for DPM is $11 + 7 = 18$ nonzeros, as compared with 22 for GE with the Markowitz strategy. Notice that this figure is hard to improve for GE because Markowitz did not introduce any fill-in! In practice, however, a certain amount of fill-in almost always occurs with both methods. If no reordering of the rows of A is allowed, which could be the case if we want to exploit the row-action nature of DPM and introduce only one row of A at each step, the pivoting strategy previously described produces

$$Z = \begin{pmatrix} 1 & 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{3} & 0 \\ 0 & 1 & 0 & 0 & \frac{3}{2} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

or $Z = QW$ with

$$W = \begin{pmatrix} 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{3}{2} \\ 0 & 0 & 1 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Storage requirements are now slightly higher; from 11 we increase to 12 (from 17 to 19 if A must be kept) but the need is still less than GE with the Markowitz strategy. Further savings are possible if only a single right-hand side is present. It should be emphasized that the pivoting strategy is simpler for the projection method than for GE. For now the row interchanges are performed a priori (or not at all), and only column interchanges (on the Z matrix) are possible at each step. On the other hand, elimination with the Markowitz strategy requires both row and column interchanges on the active matrix at each step. In other words, the search of the pivots is more complicated and time-consuming for this version of GE, although very efficient algorithms have been devised to do the search without having to scan more than a few rows and columns of the active matrix (see Duff, Erisman, and Reid [6] and Zlatev [19]). Of course this also means that Markowitz can choose its pivots from a larger pool of candidates and therefore can be more effective in preserving sparsity. The numerical experiments we performed, however, indicate that this need not be always the case.

Let us now go back for a moment to the tridiagonal matrix (4.1), and consider the 5×5 case. The Z matrix produced by the null vector algorithm with pivoting for sparsity ($u = 0$) is

$$Z = \begin{pmatrix} 1 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \frac{4}{3} \\ 0 & 0 & 1 & 0 & \frac{3}{5} \\ 0 & 0 & 0 & 1 & \frac{15}{2} \end{pmatrix},$$

which means that no fill-in in excess of the initial storage requirement for A is introduced. Hence, pivoting can have a great impact on the performance of the null vector algorithm when applied to structured matrices. Unfortunately, the comparison with GE is still unfavorable because GE requires no pivoting at all to achieve the same result, no fill-in. This suggests that if the null vector algorithm is to have an advantage, it will be for unstructured problems. Also, the null vector algorithm is unable to take advantage of symmetry in the coefficient matrix. For this reason, we focus our attention on unsymmetric problems.

5. Numerical experiments. In this section we summarize the results of a number of numerical experiments we performed in order to study the behavior of the direct projection approach (the null vector algorithm with pivoting and drop tolerance modifications followed by the solution algorithm). We also compare DPM with a sparse implementation of GE. The pivoting strategy for GE which more closely corresponds to the one we used for the projection method (as described in §4) is sometimes called *minimum row count within a priori column ordering*. The columns of the coefficient matrix are first rearranged in increasing density order; then, at each step of the elimination process, pivot $a_{kk}^{(k)}$ is selected to be that nonzero entry of the active part of column k whose row contains the least number of nonzeros. To promote numerical stability, a threshold parameter u between 0 and 1 is introduced and the search is restricted to those candidates which satisfy the inequality

$$|a_{kk}^{(k)}| \geq u \cdot \max_{k \leq j \leq n} |a_{jk}^{(k)}|$$

(for $u = 1$ this is just partial pivoting). Naturally, there exists a column-oriented version of this strategy, called *minimum column count within a priori row ordering*. Again, $u = 0.1$ seems to be a good choice in many cases. This pivoting technique is simpler than the one based on the Markowitz criterion and is known in many cases to give considerably worse performances in terms of fill-in than Markowitz, as reported by Duff, Erisman, and Reid [6, p. 131]. Nevertheless, it is an available option in some sparse codes, and it has been observed that when drop tolerances are used, this strategy can actually be superior to Markowitz (see Zlatev [19, pp. 100–101]).

Therefore, comparing our version of DPM with GE with minimum row count in a priori column ordering is not unreasonable. More sophisticated versions of the pivoting strategy for the projection method are possible, but they appear to be too expensive, at least on sequential computers. Moreover, our experiments show that pivoting for sparsity is only moderately useful for certain problems, which happen to be exactly those problems for which DPM is most effective; the behavior of the algorithm is influenced more by other factors, such as the use of suitable drop tolerances. This is true for both DPM and the scheme based on GE.

In our initial experiments we generated a number of matrices with widely varying sparsity patterns, density, and size. It was soon confirmed that DPM tends to be much more expensive, both in terms of fill-in and operations count, than sparse GE when applied to sparse structured problems. This is true in particular for banded matrices, including the tridiagonal and block tridiagonal cases. On the other hand, DPM often proved better than GE when applied to unstructured problems.

The performance of DPM is particularly interesting for matrices of the form $A = \gamma I + M$ where γ is a nonzero scalar and M is a sparse matrix with a random nonzero pattern in which the nonzero entries are taken from a uniform distribution on the interval $(-1, 1)$. In the first set of experiments, eleven different matrices of

order $n = 1000$ and $\gamma = 10$ were used in which the percentage of nonzero elements varies between 0.3% (each row contains on average only 3 nonzeros) to 3.0% (each row contains on average 30 nonzeros). In Fig. 1

$$y = \log_{10} \frac{\text{Fill for GE}}{\text{Fill for DPM}}$$

is plotted against the percentage of nonzeros in each matrix, and this is done for two tolerances, $\text{Tol} = 10^{-3}$ and $\text{Tol} = 10^{-12}$.

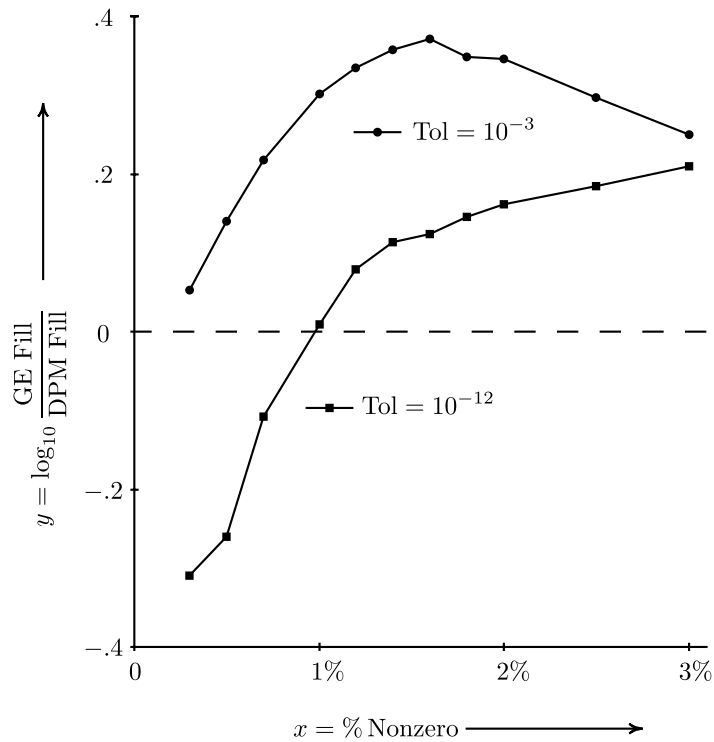


FIG. 1

For matrices whose data points fall above the line $y = 0$, DPM is superior with regard to fill-in, whereas GE is the fill-in winner for points below $y = 0$. The degree of superiority is also evident. For example, GE applied to matrices corresponding to points above the line $y = \log_{10} 2 \approx .3$ generates more than twice the fill-in than does DPM, whereas GE applied to matrices falling below $y = -\log_{10} 2 \approx -.3$ generates less than half of the fill-in as that generated by DPM. It should be mentioned that both methods suffer from severe fill-in as the density moves toward 2%. Threshold pivoting was generally ineffective for our class of test matrices.

For the same eleven matrices used to generate Fig. 1, the number of flops required by GE and by DPM were counted as the computation progressed. In Fig. 2

$$y = \log_{10} \frac{\text{Flop count for GE}}{\text{Flop count for DPM}}$$

is plotted against the percentage of nonzeros in each matrix, and this is done for two drop tolerances, $\text{Tol} = 10^{-3}$ and $\text{Tol} = 10^{-12}$.

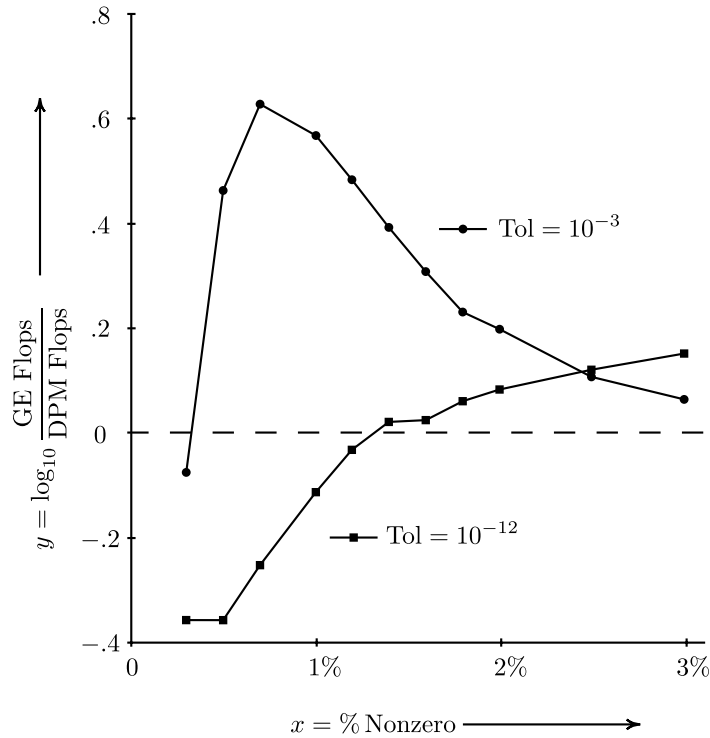


FIG. 2

For matrices of the form $A = \gamma I + M$, Figs. 1 and 2 suggest that for small (or no) drop tolerances, GE is superior (both in terms of fill-in and flop count) when A is less than 1% nonzero, but for densities higher than 1% DPM has the advantage. It is also apparent how the performance of DPM in comparison with that of GE is enhanced by the use of larger drop tolerances. Employing larger drop tolerances greatly reduces both fill-in and flop count for both methods, but, for larger values of Tol, Figs. 1 and 2 show that DPM is significantly better than GE, even for low density matrices. In this regard it is worth noting that matrices of the form $A = \gamma I + M$ are very well conditioned and the relative error in the computed solution of an associated linear system $Ax = b$ is always of the same order of magnitude as the drop tolerance. When more accuracy is required, a few steps of iterative refinement (whose implementation is well-suited for DPM) can be applied.

In order to study the performance of the algorithms on realistic problems, we experimented with test matrices from the Harwell-Boeing collection (Duff, Grimes, and Lewis [7]). Although DPM is outperformed by GE for many problems (such as those considered by Tuma [18]), we found other cases (not discussed by Tuma) for which the projection method is quite efficient and sometimes actually better than GE.

Table 1 lists the Harwell-Boeing (HB) test-matrix name together with its size, the number of nonzero entries, and the application area from which the test matrix is derived (also see [8]).

TABLE 1
Test matrices and their characteristics.

HB-Matrix	Size (n)	Nonzeros	Application
FS 183 1-3-4-6	183	1069	Chemical Kinetics
FS 541 1-2-3-4	541	4285	Chemical Kinetics
FS 680 1-2-3	680	2646	Chemical Kinetics
FS 760 1-2-3	760	5976	Chemical Kinetics
SHL 400	663	1712	Linear Programming
WEST 0167	167	507	Chemical Engineering
NNC 261	261	1500	Nuclear Engineering
IMPCOLE	225	1308	Chemical Engineering
GRE 216 a	216	876	Simul. Computer System
GRE 512	512	2192	Simul. Computer System
HOR 131	434	4710	Flow in Networks
ORSIRR 2	886	5970	Oil Reservoir Simul.
LNSP 511	511	2796	Fluid Flow

For two different values of the threshold parameter, $u = 0.1$ and $u = 1.0$, Table 2 reports the amount of fill-in generated by DPM and by GE when they are applied to the test matrices listed in Table 1. It was observed that operation counts and solution timings tend to be proportional to the amount of fill-in generated. We assume that a system must be solved for several right-hand sides, so we counted the total number of nonzeros in both the L and the U factors for GE. For DPM we counted the number of nonzeros in the null vectors (except for the 1's), the pivots, and the portion of A required to perform the solution process. For GE as well as DPM this usually requires about twice the storage than if only a single right-hand side is involved.

In all our experiments, we used a drop tolerance (usually $10^{-15} \leq \text{Tol} \leq 10^{-12}$) to remove very small elements generated during the computations, and all computations were performed on the CRAY Y-MP 8/464 at the North Carolina Supercomputing Center using CRAY single precision arithmetic.

Right-hand sides b were generated so that the exact solution of $Ax = b$ was known. The accuracy of the computed solutions was checked by calculating the relative error (in the l_2 -norm) $\text{err} = \|x - x^*\| / \|x^*\|$, where x^* is the exact solution and x the computed one. Although the relative error was usually a bit smaller for DPM, the order of magnitude of the relative error was about the same for DPM and GE.

The results in Table 2, which represent about one third of the experiments performed, indicate that DPM can be competitive with and sometimes superior to GE. The performance of DPM compared with GE is particularly interesting for the FS matrices which arise in the numerical solution of large stiff systems of ODEs in chemical kinetics studies. For these problems, DPM is at least comparable with, but often better than GE. These problems are characterized by the fact that the fill is not very sensitive to the value of the threshold parameter (this is true for both methods). Furthermore, the FS class of test matrices somewhat resembles the class of matrices $A = \gamma I + M$ discussed earlier in the sense that matrices in each class have a nonzero diagonal but are otherwise rather unstructured. This is typical of many problems which arise when systems of ODEs are solved by implicit methods.

TABLE 2
Fill for DPM and GE using small drop tolerances.

HB-Matrix Name	DPM		GE	
	$u = 0.1$	$u = 1.0$	$u = 0.1$	$u = 1.0$
FS 183 1	1,731	3,464	2,794	3,940
FS 183 3	5,654	5,112	5,449	8,778
FS 183 4	3,053	4,677	3,410	6,677
FS 183 6	3,241	3,808	2,961	3,643
FS 541 1	6,940	6,912	6,421	6,421
FS 541 2	35,222	37,268	29,556	37,230
FS 541 3	61,746	58,434	47,238	57,581
FS 541 4	43,658	42,614	35,094	43,134
FS 680 1	14,127	15,512	36,759	37,527
FS 680 2	20,095	20,861	51,231	53,041
FS 680 3	22,098	24,487	47,948	48,555
FS 760 1	5,243	5,304	7,047	7,059
FS 760 2	88,873	88,968	78,527	105,050
FS 760 3	217,740	233,744	297,189	391,899
SHL 400	2,019	2,019	2,044	2,044
WESR0167	530	906	872	973
NNC 261	4,176	8,735	5,097	7,275
IMPCOLE	1,940	2,133	2,600	3,602
GRE 216a	9,983	18,289	3,964	6,745
GRE 512	58,268	122,966	15,549	28,568
HOR 131	70,551	87,736	28,591	34,537
ORSIRR 2	103,591	199,008	30,827	52,948
LNSP 511	58,199	82,403	17,333	18,345

On the other hand, DPM was outperformed by GE on nearly all regular grid problems and more generally on band matrices such as the last five listed in Table 2—at least when small drop tolerances were used. These numerical experiments fully support the views expressed in §4—namely that DPM should not be applied as a stand-alone method to structured problems. However, the null vector algorithm implemented with large drop tolerances can be useful for regular grid problems when it is used as a preconditioner for iterative methods; see [2]–[4].

The DPM algorithm occasionally performed satisfactorily on test matrices from application areas other than chemical kinetics (e.g., linear programming, chemical and nuclear plant modelling, etc.), but its behavior was not very consistent. Furthermore, there are other problems, not listed in Table 2, for which DPM is far worse than GE. Consequently, we decided to narrow our experiments to matrices for which DPM seems to be a feasible alternative to GE. Additional experiments with the FS class of test matrices were performed in order to explore the use of “large” drop tolerances. In practice, this course of action can be justified if only a fast approximate solution is

required, or if iterative refinement is going to be used to recover at least some of the lost accuracy.

TABLE 3
Fill for DPM and GE using large drop tolerances.

HB-Matrix	DPM	Fill	GE	Fill
Name (Tol)	$u = .1$ (err)	$u = 1$ (err)	$u = .1$ (err)	$u = 1$ (err)
FS 541 1 (10^{-2})	839 ($4.3E^{-3}$)	840 ($4.3E^{-3}$)	885 ($4.8E^{-3}$)	886 ($4.8E^{-3}$)
FS 541 2 (10^{-5})	7,982 ($6.1E^{-3}$)	7,673 ($4.3E^{-3}$)	8,492 ($9.5E^{-2}$)	9,365 ($5.7E^{-2}$)
FS 541 3 (10^{-5})	15,443 ($6.8E^{-2}$)	13,870 ($7.6E^{-2}$)	15,023 ($2.8E^{-1}$)	17,294 ($7.0E^{-1}$)
FS 541 4 (10^{-5})	9,466 ($1.4E^{-2}$)	9,663 ($2.0E^{-2}$)	10,143 ($1.3E^{-1}$)	11,470 ($9.7E^{-2}$)
FS 680 1 (10^{-3})	2,927 ($2.0E^{-2}$)	3,582 ($2.4E^{-2}$)	8,332 ($1.6E^{-1}$)	8,629 ($1.6E^{-1}$)
FS 680 2 (10^{-3})	2,381 ($1.8E^{-3}$)	3,030 ($1.8E^{-3}$)	3,597 ($3.6E^{-3}$)	5,997 ($3.4E^{-3}$)
FS 680 3 (10^{-3})	2,577 ($3.1E^{-4}$)	3,557 ($4.0E^{-4}$)	2,943 ($4.5E^{-4}$)	7,351 ($4.4E^{-4}$)

Table 3 contains the results of some of these experiments. Just as before, we experimented with two threshold parameters, $u = 0.1$ and $u = 1.0$. The parenthesized quantity (Tol) in the first column of Table 3 is the drop tolerance used for the given problem, while the parenthesized number (err) in the other columns represents the error in the computed solution.

Clearly, using large tolerances is very effective in reducing fill. Again, it appears that using $u < 1$ is seldom justified, especially for DPM (where it is sometimes noxious). The advantage of using DPM instead of GE when large drop tolerances are allowed becomes more evident in connection with iterative refinement.

To perform iterative refinement of the solution, both GE and DPM need the original coefficient matrix A , hence a copy of it must be kept and cannot be overwritten by other quantities. Since DPM needs about half of the matrix A anyway, the additional storage requirements will be smaller than for GE. For example, consider the matrix FS 680 2. For Tol = 10^{-3} and $u = 0.1$, the storage requirements for iterative refinement become, respectively, 3,656 nonzeros for DPM and 6,021 nonzeros for GE. These figures do not include the right-hand side, which is the same for both methods. It can be shown that the operation count for the approximate solution followed by three iterations of the iterative refinement process is about twice as much for GE than for DPM (our experiments show that after three iterations the computed residual is 0 for these matrices).

To fully appreciate the impact of these savings, it should be kept in mind that matrices of the FS type arise in the numerical integration of stiff systems of ODEs, where a large number of linear systems must be solved during the integration process; see Zlatev [19, Chap. 8]. It should be mentioned that for very ill conditioned problems, the same difficulties may arise as for iterative improvement of GE in the sense that the null vector algorithm may break down due to the occurrence of zero pivots. Furthermore, even if the process is successfully completed, iterative improvement could fail to converge in a reasonable number of steps. A more detailed treatment of iterative refinement for DPM is given in [2].

6. Conclusions and future work. Based on extensive numerical experiments we found that our direct projection algorithm in some cases can be competitive and even superior in terms of fill-in and arithmetic complexity when compared with sparse algorithms based on traditional elimination. This approach seems to be best suited when solving linear systems arising in the numerical solution of ODEs in chemical kinetics studies, particularly when high drop tolerances and iterative refinement are used.

On the other hand, we have reached the conclusion that this method should not be used as a general purpose solver or for highly structured problems, particularly when the coefficient matrix is banded with a narrow band.

It is quite possible that much better results might be obtained when this or other variants of DPM are implemented on vector and parallel computers or if the algorithm is used as a preconditioner for iterative methods. For example, if A is symmetric positive definite, then no pivoting is required, and

$$A^{-1} = ZD^{-1}Z^T.$$

If “cheap” approximations $\hat{Z} \approx Z$ and $\hat{D} \approx D$ of Z and D can be obtained, perhaps by using the null vector algorithm with large drop tolerances, then the matrix

$$(6.1) \quad K = \hat{Z}\hat{D}^{-1}\hat{Z}^T$$

represents an approximate inverse of A in the sense that $KA \approx I$. Consequently, K represent a feasible *explicit* preconditioner. A major advantage of explicit preconditioners is the fact that no linear systems must be solved to compute the next iterate—only matrix–vector multiplications are required. This fact is particularly important when the method is to be implemented on vector and parallel computers.

Cheap approximations to Z can be found either by using large drop tolerances or by allowing fill-in to take place only in certain positions within Z , for example along a few diagonals adjacent to the main one. This could be motivated by the fact that many matrices arising in practice have inverses whose entries decrease in magnitude, sometimes very rapidly, away from the main diagonal.

A preconditioning matrix of the form (6.1) can be used in conjunction with many different iterative schemes, the simplest being the stationary first-order iterative method

$$x^{(k+1)} = x^{(k)} + \omega K(b - Ax^{(k)})$$

where ω is a convergence factor—note that $\omega = 1$ corresponds to iterative refinement. Of course, other iterative techniques such as the preconditioned conjugate gradient method can be considered, and encouraging preliminary results along these lines have been reported in [2] and [3]. A more detailed study is forthcoming [4].

Finally, it is worthwhile observing that the sparse null vector algorithm can be adapted to handle singular matrices, and it may also be used to compute a sparse basis for the null space of a rectangular matrix—an important step in many computational procedures in constrained optimization.

Acknowledgment. The authors would like to thank the referees for their valuable suggestions which enhanced the exposition of this article.

REFERENCES

- [1] J. ABAFFY, C. BROYDEN, AND E. SPEDICATO, *A class of direct methods for linear equations*, Numer. Math., 45 (1984), pp. 361–376.

- [2] M. BENZI, *A Direct Row-Projection Method for Sparse Linear Systems*, Ph.D. thesis, Department of Mathematics, North Carolina State University, Raleigh, NC, 1993.
- [3] M. BENZI AND C. D. MEYER, *An explicit preconditioner for the conjugate gradient method*, Proceedings of the Cornelius Lanczos International Centenary Conference, J. Brown, M. Chu, D. Ellison, and R. Plemmons, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1994, pp. 294–296.
- [4] M. BENZI, C. D. MEYER, AND M. TŮMA, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., to appear.
- [5] Y. CENSOR, *Row-action methods for huge and sparse systems and their applications*, SIAM Rev., 23 (1981), pp. 444–466.
- [6] I. S. DUFF, A. M. ERISMAN AND J. K. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, 1986.
- [7] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.
- [8] ———, *User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)*, MEA-TR-202-R1, October 1992.
- [9] D. K. FADDEEV AND V. N. FADDEEVA, *Computational Methods of Linear Algebra*, W. H. Freeman and Co., San Francisco, London, 1963.
- [10] L. FOX, H. D. HUSKEY, AND J. H. WILKINSON, *Notes on the solution of algebraic linear simultaneous equations*, Quart. J. Mech. Appl. Math., 1 (1948), pp. 149–173.
- [11] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.
- [12] A. S. HOUSEHOLDER, *Terminating and nonterminating iterations for solving linear systems*, J. Soc. Indust. Appl. Math., 3 (1955), pp. 67–72.
- [13] ———, *The Theory of Matrices in Numerical Analysis*, Blaisdell Pub. Co., New York, 1964.
- [14] T. PIETRZYKOWSKI, *Projection method*, Zaktadu Aparatów Matematycznych Polskiej Akad. Nauk Praca, A8(1960).
- [15] E. W. PURCELL, *The vector method of solving simultaneous linear equations*, J. Math. Phys., 32 (1953), pp. 180–183.
- [16] F. SLOBODA, *A parallel projection method for linear algebraic systems*, Apl. Mat. Ceskosl. Akad. Ved., 23 (1978), pp. 185–198.
- [17] G. W. STEWART, *Conjugate direction methods for solving systems of linear equations*, Numer. Math., 21 (1973), pp. 283–297.
- [18] M. TŮMA, *Solving sparse unsymmetric sets of linear equations based on implicit Gauss projection*, Tech. Rep. 555, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, April, 1993.
- [19] Z. ZLATEV, *Computational Methods for General Sparse Matrices*, Kluwer Academic Publishers, Dordrecht, 1991.