SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

0196-5204/84/0501-0015 $01.25/0

PRACTICAL USE OF SOME KRYLOV SUBSPACE METHODS FOR SOLVING INDEFINITE AND NONSYMMETRIC LINEAR SYSTEMS*

YOUCEF SAAD†

Abstract. The main purpose of this paper is to develop stable versions of some Krylov subspace methods for solving linear systems of equations $Ax = b$. As in the case of Paige and Saunders's SYMMLQ [SIAM J. Numer. Anal., 12 (1975), pp. 617–624], our algorithms are based on stable factorizations of the banded Hessenberg matrix representing the restriction of the linear application A to a Krylov subspace. We will show how an algorithm similar to the SYMMLQ can be derived for nonsymmetric problems and we will describe a more economical algorithm based upon the LU factorization with partial pivoting. In the particular case where A is symmetric indefinite the new algorithm is theoretically equivalent to SYMMLQ but slightly more economical. As a consequence, an advantage of the new approach is that nonsymmetric or symmetric indefinite or both nonsymmetric and indefinite systems of linear equations can be handled by a single algorithm.

Key words. numerical linear algebra, iterative methods, nonsymmetric systems, conjugate gradients

1. Introduction.

In the previous few years considerable attention has been devoted to solving large sparse sets of equations of the form

$$Ax = b \tag{1}$$

where A is an $N \times N$ real matrix. Linear systems of equations fall into four distinct classes:

1. A is symmetric positive definite;
2. A is symmetric indefinite;
3. A is nonsymmetric positive real, i.e., its symmetric part $(A + A^T)/2$ is positive definite;
4. A is nonsymmetric nonpositive real, i.e. its symmetric part is indefinite. We will then often say that A is indefinite.

Roughly speaking, the four classes above are ordered according to increasing difficulty of solution.

While the problems of the first class are well understood, the other classes have attracted much of the recent research in numerical linear algebra and are still under intensive investigation. Recent work by Vinsome [18], Axelsson [1], Elman, Eisenstat and Schultz [3], Elman [4], [5], Young and Jea [8] and Saad [14] concerns Krylov subspace methods for the case where A is nonsymmetric. A common feature of all of these methods is that the approximate solution x_m belongs to the affine space $x_0 + K_m$ where K_m is the Krylov subspace $K_m = \text{span}\,[r_0, Ar_0, \cdots, A^{m-1}r_0]$ and r_0 is the initial residual vector $r_0 = b - Ax_0$. Their principle is to attempt to make the residual vector r_m orthogonal to some subspace L_m, usually different from K_m [15]. These processes can also be regarded as different realizations of Galerkin projection methods on K_m, whereby the original problem is replaced by an m-dimensional problem with the linear operator $A_m = P_m A|_{K_m}$, where P_m is the projector onto K_m parallel to L_m. We will refer to A_m as a *section of A in* K_m.

Many of the Krylov subspace methods developed in the literature assume that the matrix A has a positive definite symmetric part, i.e. they deal with problems of the third class. Problems of the harder class 4 often occur when a preconditioning

* Received by the editors February 9, 1982, and in final revised form November 12, 1982. This work was supported in part by the U.S. Air Force grant AFOSR-81-0193 and in part by the National Science Foundation grant MCS-8104874.

† Computer Science Department, Yale University, New Haven, Connecticut 06520.

technique is used in conjunction with the iterative method. Then the resulting iteration matrix is nonpositive real even though the original matrix is positive real (see [5]).

The Incomplete Orthogonalization Method (IOM), closely related to the Galerkin process, was introduced in [14]. The IOM was devised to handle problems of classes 2, 3 and 4. Experience shows that it is effective on problems of the second and third class and on problems of class 4 that are mildly indefinite or mildly nonsymmetric. A peculiarity of the original version of IOM is that the solution is not available at each ep. Instead, one saves the basis vectors of the subspace K_m in secondary storage and forms the solution as a combination of these vectors only at the end of the process. A simple formula provides, at no extra cost, the residual norm at each step, thus determining when to stop. This implementation of the method is *indirect* in that the solution is not directly formed. It requires less core memory and computational work than its counterparts based on direct updating of the solution at every step, but has the disadvantage of using secondary storage. The purpose of this paper is to develop an equivalent version of the IOM that does not require secondary storage, while keeping the stability properties of the original version.

The principle of our approach is similar to that adopted by Paige and Saunders [10] which led to the successful SYMMLQ algorithm for solving symmetric indefinite problems. Let us recall that SYMMLQ was based upon the stable LQ factorization of a tridiagonal matrix representing the section A_m of A in K_m. For IOM, this representation becomes a banded Hessenberg matrix, which we will factor by resorting to the *LU factorization with partial pivoting.* Note that such a factorization can also be applied to tridiagonal matrices and therefore when the matrix A is symmetric, we obtain an alternative of the SYMMLQ algorithm which, incidentally, is slightly more economical than SYMMLQ. As a consequence, the new algorithm has the attractive feature that the same code can be used for any linear system regardless of symmetry or definiteness. However, it should be pointed out that when the skew-symmetric part of A is large and its symmetric part has the origin well inside the spectrum, the Krylov subspace methods are not recommended. The Krylov subspace methods are most effective in those cases where A is either nearly symmetric or nearly positive real or both.

We will compare our method with other Krylov subspace methods and will prove in particular that DIOM(k) is equivalent to Jea and Young's ORTHORES(k) algorithm.

In § 2 we will briefly recall the Incomplete Orthogonalization Method in its original form [14]. Section 3 describes an alternative version of the same algorithm, which will be called the Direct Incomplete Orthogonalization Method (DIOM). Then, in § 4, we will briefly indicate how similar techniques can be derived for Krylov subspace methods other than the IOM. A few practical considerations and heuristics will be presented in § 5 and some numerical experiments will be reported in the last section.

2. Krylov subspace methods.

2.1. The full orthogonalization method. Given a set of m linearly independent vectors $V_m = [v_1, v_2, \cdots, v_m]$, an orthogonal projection method on the subspace $K_m \equiv \text{span}[V_m]$, is by definition a method which obtains an approximate solution to (1) of the form

$$x_m = x_0 + V_m y_m \tag{2}$$

for which the residual $r_m = b - Ax_m$ is orthogonal to the subspace K_m. This Galerkin condition gives

$$V_m^T(b - Ax_m) = 0 \tag{3}$$

and therefore

$$y_m = [V_m^T A V_m]^{-1} V_m^T r_0. \tag{4}$$

In the basic full orthogonalization method (or Arnoldi's method) presented in [14], one first constructs an orthonormal sequence $V_m = [v_1, v_2, \cdots, v_m]$ from the recurrence:

$$h_{j+1,j} v_{j+1} = A v_j - \sum_{i=1}^{j} h_{ij} v_i \tag{5}$$

starting with the vector $v_1 = r_0/\|r_0\|$. In (5), the coefficients h_{ij}, $i = 1, \cdots, j+1$ are chosen such that the vector v_{j+1} is orthogonal to all the previous v_i's and $\|v_{j+1}\| = 1$.

The system V_m constitutes an orthonormal basis of the Krylov subspace $K_m = \operatorname{span}\{r_0, Ar_0, \cdots, A^{m-1} r_0\}$. An important property is that the matrix $V_m^T A V_m$ in (4) is precisely the $m \times m$ upper Hessenberg matrix whose nonzero entries are the h_{ij}'s defined in the algorithm. Furthermore, the vector $V_m^T r_0$ in (4) is equal to $\|r_0\| e_1$, with $e_1 = (1, 0, 0, \cdots, 0)^T$, by the definition of v_1. Therefore the solution y_m of (4) simplifies into

$$y_m = H_m^{-1} (\beta e_1) \tag{6}$$

where we have denoted by β the scalar $\|r_0\|$ for convenience.

An algorithm based upon the above considerations can be briefly described as follows:

ALGORITHM 1.
1. Compute $r_0 := b - Ax_0$, take $v_1 := r_0/(\beta := \|r_0\|)$ and choose an integer m.
2. For $j = 1, 2, \cdots, m$ compute the vectors v_j and the coefficient $h_{i,j}$ by (5).
3. Compute the approximate solution x_m from (2) and (6).

This will be referred to as the full orthogonalization method.

2.2. The incomplete orthogonalization method. A serious drawback of Algorithm 1 is that as j increases the process becomes intolerably expensive and requires the storage of the whole set of previous vectors v_i since these are used at every step. A remedy to this is to orthogonalize the current vector Av_j against k previous vectors instead of all the previous vectors. We will assume throughout that[1] $k \geqq 2$. The derived algorithm has been proposed in [14], and is called the Incomplete Orthogonalization Method (IOM). The only difference with the above algorithm of full orthogonalization lies in the definition of v_{j+1}, which now becomes:

$$v_{j+1} = \frac{1}{h_{j+1,j}} \left[A v_j - \sum_{i=j-k+1}^{j} h_{ij} v_i \right]. \tag{7}$$

In the above summation $j - k + 1$ is to be replaced by 1 whenever $j \leqq k - 1$. Here the coefficients $h_{i,j}$ are chosen such as to make v_{j+1} of norm one and orthogonal to the vectors v_i, $i = j - k + 1, \cdots, j$, that is:

$$h_{i,j} = (Av_j, v_i), \qquad i = j - k + 1, \cdots, j, \tag{8}$$

$$h_{j+1,j} = \left\| A v_j - \sum_{i=j-k+1}^{j} h_{ij} v_i \right\|. \tag{9}$$

[1] The method can also be defined for $k = 1$, and corresponds to the method of steepest descent in the symmetric positive definite case. We want to avoid this trivial case.

The new Hessenberg matrix which will still be denoted by H_m has the following banded structure when, for example, $m=9$, $k=4$:

$$H_m=\begin{bmatrix} x & x & x & x & & & & & \\ x & x & x & x & x & & & & \\ & x & x & x & x & x & & & \\ & & x & x & x & x & x & & \\ & & & x & x & x & x & x & \\ & & & & x & x & x & x & x \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \end{bmatrix}. \tag{10}$$

A simplistic version of the IOM algorithm can be described as follows.

ALGORITHM 2.
1. Compute $r_0 := b - Ax_0$, take $v_1 := r_0/(\beta := \|r_0\|)$ and choose an integer m.
2. Compute $V_m = [v_1, v_2, \cdots, v_m]$ and H_m by using (7) and (8) and (9), for $j = 1, \cdots, m$.
3. Compute

$$y_m := \beta H_m^{-1} e_1 \tag{11}$$

and form the approximate solution

$$x_m := x_0 + V_m y_m. \tag{12}$$

We will refer to the above algorithm as IOM (k) or simply IOM if there is no ambiguity. It is clear that when the number of steps m does not exceed k, the above algorithm is equivalent to the full orthogonalization method. For this reason we will denote by IOM (∞) the full orthogonalization method, since full orthogonalization corresponds to taking k larger than any step number m in the above algorithm.

One of the important details not made clear in the above implementation concerns the choice of the number of steps m. If the algorithm were to be applied with an arbitrarily chosen m, we would certainly have to restart the algorithm whenever m is so small that the accuracy of x_m is insufficient. In other words we would have to restart the above algorithm with the initial vector x_0 replaced by the latest computed approximate solution x_m, and this would be repeated until convergence. But it is also possible that m might be too large and that convergence would occur for some $m_0 < m$. This means that we must be able to test for convergence anywhere between $j=1$ and $j=m$. In fact all we need is a formula for computing the residual norm of the intermediate approximation x_j *without forming* x_j. Fortunately such a formula exists and is given by (see e.g. [14])

$$\|b - Ax_m\| = h_{m+1,m} |e_m^T y_m|. \tag{13}$$

As will be seen later, updating (13) requires only 2 multiplications per step provided that the factorization of H_j is updated at each step (this fact will be shown in the next section). Since the final factorization of H_m is needed for the solution of the $m \times m$ system involved in (11), it is not more costly to factor H_m by updating the factorization at each step, and hence the computation of the estimate (13) is virtually free. It should be added that (13) gives a quite accurate approximation of the residual norm in practice.

The above remarks suggest an efficient implementation of IOM which we briefly outline here (see [14] for more details). First choose an initial vector x_0, the parameter k and a maximum number of steps m max. Compute r_0, and $v_1 = r_0/\|r_0\|$. Then for $j = 1, 2, \cdots$, compute $h_{i,j}$ and v_{j+1} by (5). Write in secondary storage every vector v_{j+1} thus generated, one by one. At each step j, simultaneously update the LU factorization of H_j and the residual norm (13). If at step j the residual norm is small enough, recall the v_i's from secondary storage one by one and form the approximate solution x_j by (11) and (12). If j reaches m max and the residual norm is still unsatisfactory, form $x_{m\,\max}$ and restart the algorithm with this as initial vector.

We must emphasize that the central idea of the algorithm lies in the fact that it is possible to update at each step the factorization $H_j = L_j U_j$ with L_j unitary lower triangular, U_j upper triangular. Even more interesting is the fact that LU factorization *with partial pivoting* can also be updated at each step together with the estimate (13) of the residual norm. These observations have already been exploited in the previous paper [14]. The algorithm given above will be referred to as the *indirect version of* IOM as the approximate solution x_m is not updated at every step. We now show how to derive a few direct versions which are theoretically equivalent.

3. Incomplete orthogonalization method: direct versions.

In all of the algorithms proposed by Axelsson [1], Eisenstat, Elman and Schulz [3], Elman [5], Young and Jea [8] and Vinsome [18], the approximate solution x_m is updated at every step in a conjugate-gradient-like algorithm. We show here that it is also possible to write similar versions for the IOM algorithm. The algorithms we are about to describe are based upon updating the LU factorization of H_m at each step. For the sake of clarity we first present a version that does not use partial pivoting. The more stable algorithm using partial pivoting will be the object of § 3.2.

3.1. Derivation of the algorithm.

At each step m of IOM, the approximate solution x_m is given by the formula

$$x_m = x_0 + \beta V_m H_m^{-1} e_1 \tag{14}$$

where $\|r_0\|$ has been replaced by β for convenience.

Let H_m be factored as

$$H_m = L_m U_m \tag{15}$$

where L_m is an $m \times m$ lower bidiagonal matrix with diagonal elements equal to one, and U_m is a banded upper triangular matrix with k diagonals. That is:

$$\begin{array}{ccccc} H_m & = & L_m & & U_m \\ \begin{bmatrix} x & x & x & x & & & & & \\ x & x & x & x & x & & & & \\ & x & x & x & x & x & & & \\ & & x & x & x & x & x & & \\ & & & x & x & x & x & x & \\ & & & & x & x & x & x & x \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \end{bmatrix} & = & \begin{bmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & & 1 & & \\ & & \ddots & \ddots & \\ & & & l_m & 1 \end{bmatrix} & & \begin{bmatrix} x & x & x & x & & & & & \\ & x & x & x & x & & & & \\ & & x & x & x & x & & & \\ & & & x & x & x & x & & \\ & & & & x & x & x & x & \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \\ & & & & & & & & x \end{bmatrix} \end{array}$$

From (14) and (15) the solution x_m satisfies $x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1)$.

Following Paige and Saunders [10] let us set

$$W_m = V_m U_m^{-1} \tag{16}$$

and

$$z_m = \beta L_m^{-1} e_1. \tag{17}$$

We will denote by w_i the ith column of the $N \times m$ matrix W_m.

The key observation here is that we pass from the $(m-1)$-dimensional vector z_{m-1} to the m-dimensional vector z_m by just appending one component. In other words:

$$z_m = \begin{bmatrix} z_{m-1} \\ \\ \xi_m \end{bmatrix}. \tag{18}$$

The last element ξ_m of the vector z_m satisfies the recurrence

$$\xi_m = -l_m \xi_{m-1}, \qquad m = 2, 3, \cdots, \tag{19}$$

starting with $\xi_1 = \beta = \|r_0\|$. Recall that we denote by l_m the element in position m, $m-1$ of the matrix L_m. It is then clear that x_m can be updated at every step through the formula:

$$x_m = x_{m-1} + \xi_m w_m. \tag{20}$$

The vectors w_m can in turn be updated quite simply since we have from their definition (16)

$$w_m = \frac{v_m - \sum_{i=m-k+1}^{m-1} u_{im} w_i}{u_{mm}}. \tag{21}$$

Our first direct version of IOM (k) can therefore be summarized as follows:

ALGORITHM 3.
1. *Start.* Choose an initial vector x_0 and compute $r_0 = b - Ax_0$.
2. *Iterate.* For $m = 1, 2, \cdots$ until convergence do:
   - Compute h_{im}, $i = m-k+1, \cdots, m+1$, and v_{m+1} by formulas (7), (8) and (9).
   - Update the LU factorization of H_m, i.e. obtain the last column of U_m and the last row of L_m. Then compute ξ_m from (19).
   - Compute

$$w_m = \frac{v_m - \sum_{i=m-k+1}^{m-1} u_{im} w_i}{u_{mm}}.$$

   - Compute

$$x_m = x_{m-1} + \xi_m w_m.$$

We have intentionally skipped some of the details concerning in particular the way the LU factorization of H_m is updated. An important remark here is that (13) can easily be updated because the last element of y_m is just ξ_m / u_{mm}, hence

$$\|b - Ax_m\| = h_{m+1,m} \left| \frac{\xi_m}{u_{mm}} \right|, \tag{22}$$

which requires only two arithmetic operations per step.

Note that the division by u_{mm} in (21) can be avoided by simply using the vectors $w_j' = u_{jj}w_j$ in place of w_j and by keeping track of the scaling factors u_{jj}. With this, the cost per step of the above algorithm is approximately $(3k+2)N$ additions/multiplications plus one matrix by vector multiplication, and we need $(2k+1)$ vectors of storage (these are: x_m, $k-1$ vectors w_j, k vectors v_j, plus an extra vector for Av_m).

The above algorithm breaks down whenever u_{mm} vanishes. In fact even if u_{mm} does not vanish it is not recommended to use the above algorithm as it is based upon the potentially unstable LU factorization of H_m and can result in an unstable algorithm for solving $Ax = b$. This brings up the use of partial pivoting.

3.2. Using the *LU* factorization with partial pivoting. Instead of the straightforward factorization (15) we now introduce the following LU factorization with partial pivoting of the matrix H_m

$$H_m = P_2E_2P_3E_3 \cdots P_mE_mU_m \tag{23}$$

where each P_j is an elementary permutation matrix, and E_j is an elementary transformation [19] having the structure:

$$E_j = \left[\begin{array}{ccccccccc}
1 &&&&&&&& \\
& 1 &&&&&&& \\
&& 1 &&&&&& \\
&&& \ddots &&&&& \\
&&&& 1 &&&& \\
&&&& l_j & 1 & 0 & 0 & 0 \\
&&&& 0 && \ddots && \\
&&&& 0 &&& \ddots & \\
&&&& 0 &&&& 1
\end{array}\right]
\begin{array}{l} \\ \\ \\ \\ \\ \leftarrow j\text{th row} \\ \\ \\ \\ \end{array}$$

$$\begin{array}{c}\uparrow \\ (j-1)\text{st column}\end{array}$$

The elementary permutation matrix P_{j+1} is the one used to permute the rows j and $j+1$ if needed, i.e., if the element $h_{j+1,j}$ is larger in absolute value than the element u_{jj}. The matrix U_m is again a banded upper triangular matrix, this time with $k+1$ diagonals instead of k due to the permutations. The LU factorization with partial pivoting is a very stable process when the matrix H_m is tridiagonal since the elements obtained at any step r of the factorization, $r \leqq m$, are no longer than $2 \max\{|h_{ij}|, i = 1, m; j = 1, m\}$ (see Wilkinson [19, p. 219]). For banded Hessenberg matrices of bandwidth $k+1$, it is easy to generalize the above bound and show that the elements obtained at any step of the factorization do not exceed $k \max\{|h_{ij}|, i = 1, m; j = 1, m\}$.

As before the approximation x_m is defined by (14). Letting again

$$W_m = V_mU_m^{-1} \tag{24}$$

we get

$$x_m = x_0 + W_mz_m$$

where z_m is now defined as

$$z_m = E_m^{-1}P_mE_{m-1}^{-1}P_{m-1} \cdots E_2^{-1}P_2(\beta e_1).$$

Note that the vectors w_j can be updated in the same way as before by use of (21), except that the summation is now from $m-k$ to $m-1$. We claim that there are formulas similar to (19), (20) for updating the vectors z_m. This is because we have

$$z_m = E_m^{-1} P_m \bar{z}_{m-1} \tag{25}$$

with

$$\bar{z}_{m-1} = \begin{bmatrix} z_{m-1} \\ \\ 0 \end{bmatrix}. \tag{26}$$

Equations (25) and (26) show that there are two cases, depending on whether interchange has or has not been performed at the previous step:

1. either rows $m-1$ and m have *not* been interchanged, in which case the vector z_m is defined as before by (18) and (19);
2. or there has been an interchange of rows $(m-1)$ and m, in which case

$$z_m = \begin{bmatrix} z_{m-2} \\ 0 \\ \xi_{m-1} \end{bmatrix} \tag{27}$$

where ξ_{m-1} is the last element of z_{m-1}.

Practically, the use of the above observations is particularly simple. If interchange has not been performed at step $m-1$, then the approximate solution x_m is updated from x_{m-1} as before by (20) and (19). If, on the other hand, rows $m-1$ and m have been permuted, then the expression (27) of z_m shows that the only difference with the previous case is that the approximation x_m is now defined by $x_m = x_{m-2} + \xi_{m-1} w_m$. In other words x_{m-1} could be redefined as equal to x_{m-2} and the scalar ξ_m as equal to ξ_{m-1} in the formula (20). In other words if a permutation occurs all we have to do is *skip the application of the updating formulas* (20), (19) at the next step. In a practical implementation we must look ahead at the current step m and check whether permutation will or will not be necessary *in the next step* $m+1$. This is fortunately possible because the element $h_{m+1,m}$ is available at the mth step as well as the element u_{mm}, since the factorization is updated at each step.

Concerning the updating of the factorization of H_m at the mth step we can proceed as follows. First, using the $k+1$ previous pivots $l_{m-k}, l_{m-k+1}, \cdots, l_m$ transform the column $\{h_{im}\}_{i=1,m}$ into the column $\{u_{im}\}_{i=1,m}$. In order for this to be possible we must save these $k+1$ pivots. Note that the mth column of H_m (resp. U_m) has at most k (resp. $k+1$) nonzero elements. Second, compare the element u_{mm} thus obtained with $h_{m+1,m}$ to determine if interchange is needed. If $|u_{mm}| < h_{m+1,m}$, permute the elements $h_{m+1,m}$ and u_{mm} and compute the next pivot l_{m+1}. Clearly the matrices H_m, E_m, U_m need not be saved. All we need is the previous $k+1$ pivots l_j, the logical information perm $(j+1)$, $j = m-k, \cdots, m$, defined as perm $(j+1)$ = true if rows j and $j+1$ are permuted, and the $k+1$ nonzero elements of the mth column of H_m, which is transformed into the mth column of U_m. This constitutes little storage as k is small in general (typically $k < 10$). We describe below the Direct Incomplete Orthogonalization Method (DIOM (k)) algorithm derived from the above suggestions.

```
ALGORITHM 4. DIOM (k).
1. Start. Pick an iniital vector $x_0$, define $r_0 := b - Ax_0$, $\rho_0 := \|r_0\|$.
2. Iterate. For $m = 1, 2, \cdots$ do:
    • Compute $v_{m+1}$ and the $m$th column of $H_m$ by (7), (8), and (9).
    • Update the LU factorization with partial pivoting of $H_m$, i.e. find the $m$th
      column of $U_m$ and compute $\rho_m := \beta_{m+1}|\xi_m/u_{mm}|$. Then interchange $u_{mm}$ and
      $h_{m+1,m}$ if necessary, and get the pivotal information to be used in the next step.
    • Compute
(28)      $w_m := \frac{v_m - \sum_{i=m-k}^{m-1} u_{im} w_i}{u_{mm}}$.
    • If perm $(m+1)$ = true and $\rho_m > \varepsilon$ then:
          $\xi_m := \xi_{m-1}$,    $x_m := x_{m-1}$.
    • Else compute
          $\xi_m := -l_m \xi_{m-1}$,    $x_m := x_{m-1} + \xi_m w_m$.
    • If $\rho_m \leqq \varepsilon$ stop.
```

Once again DIOM (∞) refers to the case of full orthogonalization, that is, to the case where the summations in (7) and (28) start from 1. In the above algorithm ρ_m represents the residual norm that would be obtained if we stop at step m. Thus the process is stopped when ρ_m is smaller than the tolerance ε. We should stress that u_{mm} in item 3 of iterate is the one obtained *after* interchange while the one used for computing ρ_m in item 2 is *before* interchange.

Notice that we force the application of formulas (19) and (20) when it is known from the residual norm ρ_m that the process will stop at the current step. Indeed, the algorithm *artificially* defines x_m to be equal to x_{m-1} whenever it is known that interchange will be necessary at the *next step*. When it is known from ρ_m that convergence is achieved, i.e. that the current step is the last one, we must imperatively define x_m by (20) and (19), regardless of the value of perm $(m+1)$. This is the reason for the more complicated test in item 4 of iterate. Another, perhaps simpler, way of correcting this is to always define perm $(m+1)$ as false if m is known to be the last step.

Because of the artifice used to define x_m when interchange occurs, the vectors x_m defined here are not the same as the vectors x_m of Algorithm 3 except when interchange is not performed. Thus, while the final solutions delivered by Algorithms 3 and 4 are identical, the intermediate vectors x_j are not necessarily the same. When interchange occurs, the approximation x_j defined by (14) is not computed but is defined as being equal to the previous approximation. For example, if interchange is needed at *every step*, the new sequence of approximations x_j obtained from Algorithm 4 is *stationary* until the final step is reached, i.e. until $\rho_m \leqq \varepsilon$. The residual estimate ρ_m corresponds to the *actual approximate solution* that would have been obtained if we had had to stop at the current step. Clearly, this may be infinite in case u_{ij} (before interchange) is zero, which reflects the fact that the approximate solution x_j defined by (14) does not exist when H_j is singular. But, while Algorithm 3 will break down in this situation, Algorithm 4 is able to construct the next approximations $x_{j+1}, x_{j+2}, \cdots$.

The amount of work is essentially the same as required for Algorithm 3. Indeed, when a permutation takes place we save one vector update of the form (20); i.e., we save N additions/multiplications. But then the permutation introduces an extra nonzero element in the triangular matrix U_{m+k}, which means N extra additions/multiplications when updating the vectors w_{m+k} at step $m+k$, and this compensates exactly the savings made at the current step. Concerning the storage, we need one more vector of storage, as the sum defining w_m is now from $m-k$ to $m-1$; i.e., we need a total of $2k+2$ vectors of storage instead of the $2k+1$ of Algorithm 3.

One might question the need for using Algorithm 4 instead of Algorithm 3 in some cases. In particular, is pivoting necessary at all if the original matrix A is positive real? Our numerical experiments show that interchange will indeed occur even in those cases. The fact that some pivots are quite small even when A is almost positive real suggests that it is in general better to use the more stable version of Algorithm 4, instead of Algorithm 3. Moreover, as shown above, it is not more costly to use pivoting except for one additional vector of storage required.

3.3. Using the *QR* factorization. The SYMMLQ algorithm described by Paige and Saunders in [10] uses the LQ factorization $H_m = L_m Q_m$. A similar algorithm using the stable QR factorization can also be developed for the incomplete orthogonalization method. Consider the orthogonal QR factorization

$$H_m = Q_m U_m \tag{29}$$

where U_m is, as in § 3.1, an $m \times m$ upper triangular matrix and Q_m is a unitary orthogonal matrix, i.e. $Q_m^T Q_m = I$. A remark which is essential is that since H_m is banded upper Hessenberg, U_m will be banded upper triangular.

The reason we prefer the QR factorization to the LQ factorization is that the implementation with QR is quite simple and resembles that of Algorithm 4. The only difference is that instead of the elementary matrices E_j we now use elementary rotation matrices F_j of the form

$$F_j = \begin{array}{c} \left[\begin{array}{cccccccc} 1 &&&&&&& \\ & 1 &&&&&& \\ && \ddots &&&&& \\ &&& c_j & -s_j &&& \\ &&& s_j & c_j &&& \\ &&&&& 1 && \\ &&&&&& \ddots & \\ &&&&&&& 1 \end{array}\right] \end{array} \begin{array}{l} \\ \\ \\ \\ \leftarrow \text{row } j \\ \\ \\ \\ \end{array}$$

where $c_j = \cos(\theta_j)$, $s_j = \sin(\theta_j)$. It will be shown below that Q_m in (29) is the product of $m-1$ such rotation matrices; more precisely,

$$Q_m = F_2 F_3 \cdots F_m. \tag{30}$$

Letting as previously

$$W_m = V_m U_m^{-1}, \tag{31}$$

$$z_m = Q_m^{-1}(\beta e_1) = Q_m^T(\beta e_1), \tag{32}$$

we observe that the approximate solution x_m is again defined by

$$x_m = x_0 + W_m z_m \tag{33}$$

and that the updating of the vectors w_j is essentially the same as in the previous algorithms, since U_m is banded upper triangular.

Next we would like to show how to update the factorization (29) and the vector z_m at every step. Suppose that at a given step we have the factorization (29) and let us assume that one more step is performed in the sequence giving the v_i's, such that we now have the $(m+1)$st column of H_{m+1} available. We want to compute the next orthogonal matrix Q_{m+1}, or according to (30) the next rotation F_{m+1}, and the next last column of U_{m+1}. Let us denote by $\bar{Q}_j$ the $(m+1)\times(m+1)$ matrix obtained by completing the $j\times j$ matrix Q_j by adding zeros in all nondiagonal positions and ones in the diagonal positions. We will define in the same way the matrix $\bar{F}_j$.

Then we have

$$(34)\qquad \bar{Q}_m^T H_{m+1} = \left[\begin{array}{cccccccc|c} x & x & x & x & & & & & 0 \\ & x & x & x & x & & 0 & & 0 \\ & & x & x & x & x & & & 0 \\ & & & x & U_m & & x & & 0 \\ & 0 & & & x & x & x & x & 0 \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & u_{mm} & x \\ \hline & & & & & & & \beta_{m+1} & \alpha_{m+1} \end{array}\right]$$

where we have denoted for convenience by β_{m+1} and α_{m+1} the elements $h_{m+1,m}$ and $h_{m+1,m+1}$ respectively. The elements of the last column of the above matrix are obtained by multiplying the last column of H_{m+1} by the successive rotations F_j^T, $j=2,3,\cdots,m$. In order to eliminate the element β_{m+1} in position $(m+1,m)$ it is clear that we need to premultiply the above matrix by the plane rotation F_{m+1}^T with c_{m+1} and s_{m+1} defined by

$$(35)\qquad c_{m+1}=\frac{u_{mm}}{(u_{mm}^2+\beta_{m+1}^2)^{1/2}},\qquad s_{m+1}=\frac{\beta_{m+1}}{(u_{mm}^2+\beta_{m+1}^2)^{1/2}},$$

which determines the next plane rotation. Note that the last column of U_{m+1} is now entirely available by premultiplying the last column of the matrix (34) by the rotation F_{m+1}^T. Since the elements $h_{i,m+1}$ are zeros for $i=1,2,\cdots,j-k$, only the previous k plane rotations are needed. The upper triangular matrix U_{m-1} will have $k+1$ diagonals as the premultiplication by the plane rotations will introduce an extra diagonal.

Consider now the vector z_{m+1} which we would like to update from z_m. This is possible because $z_{m+1}=F_{m+1}\bar{z}_m$ where

$$\bar{z}_m=\begin{bmatrix} z_m \\ 0 \end{bmatrix}.$$

Hence

$$z_{m+1}=\begin{bmatrix} z_{m-1} \\ \bar{\xi}_m \\ \xi_{m+1} \end{bmatrix}$$

where $\bar{\xi}_m = c_{m+1}\xi_m$, $\xi_{m+1} = s_{m+1}\xi_m$. In the above, ξ_j denotes the last element of z_j. Hence the approximate solution may be updated from the equation

$$x_{m+1} = x_{m-1} + \bar{\xi}_m w_m + \xi_{m+1} w_{m+1}. \tag{36}$$

Clearly, such an updating formula is not the most economical, because in the computation of x_{m+1}, each w_i is accumulated twice with different coefficients as is seen from (36). However, a more efficient accumulation can be used. Consider instead of x_m the vector $\bar{x}_m$ defined by the recurrence $\bar{x}_m = \bar{x}_{m-1} + \bar{\xi}_m w_m$. Unlike x_{m+1}, this can be accumulated with N multiplications per step instead of $2N$. It is only at the last step that we need to compute x_{m+1}, which can be obtained from $x_{m+1} = \bar{x}_m + \xi_{m+1} w_{m+1}$. Just as pointed out in § 3.3, at the mth step we have all the information necessary to obtain the next plane rotation. Hence, we should look ahead at step m and obtain the rotation F_{m+1}. In this manner we have not only z_m but also the element $\bar{\xi}_m$ in position m of z_{m+1}. Since this will not be changed by the subsequent rotations, we see that the updating formula passing from $\bar{x}_{m-1}$ to $\bar{x}_m$ can be performed. Notice that similar arguments have been used in [10].

An algorithm based on the above developments can easily be implemented, but our experience contradicted the expectation, formulated in [14], that this approach is superior. Indeed, we found that it is needlessly more expensive and complicated than the version DIOM(k) of Algorithm 4. In effect, each step now requires N more operations than the equivalent method DIOM(k). This would not have been a high price to pay if it resulted in a substantial improvement. Such is not the case, however, as a number of numerical experiments show (see § 6.1). We think that the reason for this is that the main source of errors is not in the factorization of H_m and the formation of the solution as defined by (14), but rather mainly in the construction of the vectors v_j. Solving a banded Hessenberg system by Gaussian elimination with partial pivoting is a very stable process, as was seen in § 3.2. However, there might exist particularly difficult cases where the more stable QR factorization would be more effective, and although we prefer Algorithm 4, the technique outlined in this subsection should not be completely discarded.

3.4. Properties of the Incomplete Orthogonalization Method. In this subsection we wish to show a number of properties of Algorithm 4 assuming exact arithmetic. We would like first to establish a sufficient condition under which Algorithm 4 does not break down. Similar sufficient conditions for the symmetric Lanczos algorithm are expressed in terms of the minimal polynomial of the initial vector v_1 (or r_0). We will call minimal polynomial of a vector v with respect to the matrix A the polynomial p_s of smallest degree s such that $p_s(A)v = 0$ (cf. [19]).

PROPOSITION 1. *Assume that the minimal degree of r_0 is not less than m and that the mth approximate solution x_m as defined by* (14) *exists,* i.e. *that H_m is nonsingular. Then x_m can be computed by Algorithm* 4.

Proof. To prove this property consider the polynomial defined by the recurrence

$$h_{j+1,j}p_{j+1}(\lambda) = \lambda p_j(\lambda) - \sum_{i=j-k+1}^{j} h_{ij}p_i(\lambda), \qquad j = 1, 2, \cdots, m, \cdots \tag{37}$$

starting with $p_1(\lambda) = 1$. Denote by $\tilde{p}_{j+1}$ the polynomial on the right-hand side of (37). This is a polynomial of degree j such that $h_{i+1,i} = \|\tilde{p}_{i+1}(A)v_1\|$, $i = 1, 2, \cdots$. If at step j we had $h_{j+1,j} = 0$, this would mean that $\tilde{p}_{j+1}(A)v_1 = 0$. Since $\tilde{p}_{j+1}$ is of degree j, and because the minimal polynomial of v_1 is of degree larger than j, we conclude that $h_{j+1,j}$ cannot be equal to zero. This means that the algorithm does not break down at step j, because firstly the next vector v_{j+1} can be computed, and secondly since $\max\{h_{j+1,j}, |u_{jj}|\}$ is nonzero, the factorization of H_j does not breakdown and hence the vector w_{j+1} can be computed. The only difficulty may be encountered at the final step m where u_{mm} can be equal to zero. But this is excluded by the assumption that H_m is nonsingular. □

If at the jth step, the Hessenberg matrix H_j is singular, then the jth approximation x_j as defined by (14) does not exist. Thus Algorithm 3 fails because it attempts to explicitly compute x_j for all j, while the key advantage of Algorithm 4 is that it does not need x_j to obtain the subsequent approximations. Note that when $h_{m+1,m}=0$ and $u_{mm}\neq 0$, the algorithm produces the exact solution at step m. This uncommon situation is a consequence of (13) and is often referred to as a "happy breakdown".

Next we would like to prove some orthogonality relations characterizing the residual vectors r_j and the directions w_j produced by the algorithm. In the symmetric case it is known that the residual of the approximate solution produced by the conjugate gradient algorithm is orthogonal to all the previous residuals and that the directions w_j, are conjugate with respect to A, i.e. $(Aw_j, w_i)=0$ if $i\neq j$. The situation is not as simple in the nonsymmetric case. For the residual vectors, it was shown in [14] that at each step j the residual r_j is orthogonal to the previous k residuals. This fact is a simple consequence of the following lemma:

LEMMA 2. *The residual vectors r_m produced by m steps of either Algorithm* 3 *or Algorithm* 4 *satisfy the relation*

$$r_m = -h_{m+1,m}e_m^T y_m v_{m+1} \tag{38}$$

where y_m is defined by (11).

Proof. The lemma is a consequence of the following important relation derived from the definition of the vectors v_j:

$$AV_m = V_m H_m + h_{m+1,m}v_{m+1}e_m^T \tag{39}$$

where $V_m=[v_1, v_2, \cdots, v_m]$. The residual r_m is such that

$$\begin{aligned} r_m &= b - Ax_m = b - A[x_0 + V_m y_m] = r_0 - AV_m y_m \\ &= \beta v_1 - [V_m H_m + h_{m+1,m}v_{m+1}e_m^T]y_m. \end{aligned}$$

The result (39) follows from the fact that $H_m y_m = \beta e_1$. □

Among other things, the lemma asserts that the residual vectors are equal to the v_i except for length. Since the coefficients $h_{i,j}$ are chosen such that $(v_{m+1}, v_i)=0$, $i=m-k+1, \cdots, m$, it follows from (38) that ([14]):

PROPOSITION 3. *The residual vectors produced by Algorithm* 3 *or* 4 *satisfy the orthogonality relation*:

$$(r_m, r_i)=0, \qquad i=m-k, m-k+1, \cdots, m-1.$$

In other words the current residual is orthogonal to the previous k residuals. Note that the above lemma also proves the result (13).

An important consequence of the above proposition is that when A is symmetric then the algorithm DIOM (2) is theoretically equivalent to the conjugate gradient method (A positive definite) or the SYMMLQ algorithm (A symmetric indefinite), for which it is known that the residuals satisfy the same property. In the symmetric case it is not necessary to take $k>2$, because all of the algorithms DIOM (k) with $k\geqq 2$ are equivalent to DIOM (2) (see [14]). The matrix H_m is then tridiagonal symmetric and the process of building the v_i's is nothing but the usual symmetric Lanczos process.

Concerning the directions w_i, it is natural to ask whether a conjugacy relation similar to that of the conjugate gradient method holds. Consider first the case of full orthogonalization. For the remainder of this subsection we will assume that no pivoting is performed throughout the process, i.e. we assume Algorithm 3 is used instead of Algorithm 4.

PROPOSITION 4. *For the direct version of the Full Orthogonalization Method* (DIOM (∞)) *with no pivoting, the following semiconjugacy relation is satisfied by the vectors* w_i:

$$(Aw_j, w_i) = 0 \quad \text{for } i < j, \quad j = 1, 2, \cdots.$$

Proof. Multiplying (39) on the right by U_m^{-1} gives

$$AW_m = V_m L_m + \left(\frac{h_{m+1,m}}{u_{mm}}\right) v_{m+1} e_m^T U_m^{-1}. \tag{40}$$

Multiplying the above equation by W_m^T on the left, and letting $\mu = h_{m+1,m}/u_{mm}$, we obtain

$$W_m^T A W_m = U_m^{-T}[V_m^T V_m] L_m + \mu U_m^{-T} V_m^T v_{m+1} e_m^T.$$

The last term in the above equation is a null vector because v_{m+1} is orthogonal to all previous vectors (full orthogonalization). Also because of the orthogonality of the v_i's, the $m \times m$ matrix $[V_m^T V_m]$ is the identity matrix. Finally we get

$$W_m^T A W_m = U_m^{-T} L_m,$$

which is a lower triangular matrix. This completes the proof. □

From the above proof it is easily seen that the proposition does not extend to the case of incomplete orthogonalization. A somehow weaker result is, however, proved below.

PROPOSITION 5. *The following orthogonality relations hold for* DIOM (k) *with no pivoting*:

$$(Aw_j, v_i) = 0 \quad \text{for } j - k + 1 < i < j, \quad j = 2, 3, \cdots.$$

Proof. Let us start with equation (40), which is still valid, and multiply both sides on the left by V_m^T to get

$$V_m^T A W_m = V_m^T V_m L_m + \mu V_m^T v_{m+1} e_m^T, \tag{41}$$

with μ defined in the proof of the previous proposition. Careful matrix interpretation of (41) shows that $V_m^T A W_m$ has the following structure:

$$V_m^T A W_m =$$

$$\begin{array}{ccccc} V_m^T V_m & + & L_m & + & \mu V_m^T v_{m+1} \theta_m^T \\ \begin{bmatrix} 1 & & & x & \cdot & \cdot & \cdot \\ & 1 & & & x & & \cdot \\ & & \cdot & 0 & & x & \cdot \\ x & & & \cdot & & & x \\ \cdot & x & & 0 & \cdot & & \\ \cdot & & x & & & \cdot & \\ \cdot & \cdot & \cdot & x & & & 1 \end{bmatrix} & + & \begin{bmatrix} 1 & & & & & & \\ x & 1 & & & & & \\ & x & \cdot & & 0 & & \\ & & x & \cdot & & & \\ & & & x & \cdot & & \\ & 0 & & & x & \cdot & \\ & & & & & x & 1 \end{bmatrix} & + & \begin{bmatrix} & & & & & 0 & x \\ & & & & & 0 & x \\ & & & & & \cdot & x \\ 0 & & & & & \cdot & x \\ & & & & & \cdot & 0 \\ & & & & & & 0 \\ & & & & & 0 & 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow \text{row } m-k+1. \\ \\ \\ \\ \end{matrix} \end{array}$$

Hence

$$V_m^T A W_m = \begin{bmatrix} & 1 & & & x & & \cdot & \cdot & \cdot \\ & x & 1 & & & x & & & \cdot \\ & & x & \cdot & & 0 & x & & \cdot \\ x & & & x & \cdot & & & x & \\ \cdot & x & 0 & & x & \cdot & & & x \\ \cdot & & x & & & x & \cdot & & \\ \cdot & \cdot & \cdot & x & & & x & 1 & \end{bmatrix}.$$

The upper part of the above matrix has zero elements in position i, j whenever $j-k+1<i<j$ (post multiplication of $V_m^T V_m$ by L_m introduces an extra diagonal). The proof is complete. □

3.5. Comparison with other methods. In this section we will compare the Incomplete Orthogonalization Methods with other Krylov subspace methods and will in particular prove that DIOM (k) is theoretically equivalent to Young and Jea's ORTHORES (k) algorithm. The class of Krylov subspace methods includes the ORTHOMIN (k) algorithm [18], [3], [5], Young and Jea's ORTHODIR (k) and ORTHORES (k) algorithms and Axelsson's method [1]. A comparison of the work per step for each of the above methods must take into account the fact that the parameter k does not always have the same meaning. Thus ORTHOMIN (1) is equivalent to the conjugate residual method in the symmetric case while our DIOM (2) would be equivalent to the conjugate gradient method.[2] It is therefore more meaningful to compare ORTHOMIN (k) with DIOM $(k+1)$. With this in mind, the next table compares the work and storage requirements for a few representative Krylov subspace methods.

TABLE 1

Method	Mult./step	Storage
IOM $(k+1)$*	$(2k+5)N$	$(k+2)N$
DIOM $(k+1)$	$(3k+5)N$	$(2k+4)N$
ORTHOMIN (k)	$(3k+4)N$	$(2k+3)N$
ORTHODIR (k)	$(3k+4)N$	$(2k+3)N$
ORTHORES $(k+1)$	$(3k+6)N$	$(2k+3)N$
Axelsson (k)	$(3k+4)N$	$(2k+3)N$

* This method uses secondary storage. The table reports core memory requirement only.

Note that the number of multiplications in DIOM (k) can be reduced to roughly $(3k+4)N$ by normalizing the vectors v_i only when their norms become large (or small). Thus one might say that the number of multiplications for DIOM $(k+1)$, ORTHOMIN (k) and ORTHODIR (k) is nearly the same. The slight difference in storage is not significant. The indirect method IOM (k) is attractive in computing environments in which the I/O with disks is inexpensive, but this may vary significantly from one site to another.

[2] It is assumed in this discussion that the auxiliary matrix Z used in Young and Jea's notation [8] is the identity matrix.

Various experiments have been conducted to compare the above methods, in particular by Elman [4], [5]. The nontruncated versions ($k=\infty$) are impractical since one has to keep all the previous vectors in core memory. The truncated versions ($k<\infty$), on the other hand, are more suitable computationally but more difficult to study theoretically. The comparisons show that, in most cases, none of the above methods is superior by a large margin. The ORTHOMIN (k) methods, however, have a better theoretical foundation since proofs of convergence have been established for them [3], [5]. Some examples exhibited in [5] also show that ORTHODIR (k) may diverge. We know that this may happen for DIOM (k) as well when the symmetric part of A is not positive definite. It is not known whether this can occur in the positive real case, but all the numerical experiments conducted thus far have yielded convergence in such cases. It seems likely that for k sufficiently large the process will converge in the general case but this has not been proved so far.

Besides convergence, an important comparison criterion is the risk of breakdown presented by the different methods. In this respect, DIOM (k) is reliable, as shown by Proposition 1. ORTHOMIN (k) is known to be feasible when A is positive real and can breakdown otherwise. There is no proof that ORTHORES (k) *does not* break down for positive real problems, but examples showing that it does break down for nonpositive real problems are easy to construct [8].

It will be shown below that ORTHORES (k) is in fact equivalent to IOM (k)–DIOM (k). The result that ORTHORES (∞) is equivalent to IOM (∞) is straightforward and was mentioned in [5]. That the truncated versions are also equivalent does not seem to have been noticed.

We begin by briefly recalling the ORTHORES (k) methods and a few of its properties. For more details see [8].

ALGORITHM ORTHORES (k).

1. *Start.* Choose x_0, and compute $r_0=b-Ax_0$.
2. *Iterate.* For $i=0,1,\cdots$ until convergence do:

$$\begin{aligned}
&a_j^{(i)}=(Ar_i,r_j)/(r_j,r_j),\ j=i,i-1,\cdots,i-k+1,\\
&b_i=\left[\sum_{j=i-k+1}^{i}a_j^{(i)}\right]^{-1},\\
&c_j^{(i)}=b_ic_j^{(i)},\ j=i-k+1,\cdots,i,\\
&x_{i+1}=b_ir_i+\sum_{j=i-k+1}^{i}c_j^{(i)}x_j,\\
&r_{i+1}=-b_iAr_i+\sum_{j=i-k+1}^{i}c_j^{(i)}r_j.
\end{aligned}\tag{42}$$

The scalars $c_j^{(i)}$ are determined so that the residual r_{i+1} is orthogonal to the previous k residuals. The nontruncated version of the above algorithm, named ORTHORES (∞), consists in starting the above sums from 1 instead of $i-k+1$. The process may break down at any step because a division by zero can occur in computing b_i. It is not difficult to exhibit an example for which this happens at the first step [8].

The nontruncated version ORTHORES (∞) is equivalent to IOM (∞) (or DIOM (∞)) because for both algorithms and residuals satisfy the same orthogonality relation. Note that the truncated versions DIOM (k) and ORTHORES (k) also satisfy the same orthogonality relation, namely $(r_i,r_j)=0$, for $i-k+1<j<i$. This is a hint that the truncated versions are also equivalent, but a more careful proof is needed.

Let us recall that by Lemma 2 the residual vector r_n obtained at the nth step of DIOM (k) is equal to a scalar times the vector v_{n+1} of the truncated Arnoldi process. The residual vectors r_n obtained from any Krylov subspace method are known to satisfy $r_n = p_n(A)r_0$ where p_n is a polynomial of degree n, normalized so that $p_n(0) = 1$.

We will need the following lemma.

LEMMA 6. *Let the residual vectors r_i and r'_i produced by two Krylov subspace methods be such that $r_i = \alpha_i r'_i$, $i = 0, 1, \cdots, n$, where α_i is a scalar with $\alpha_0 = 1$. Assume that the degree of the minimal polynomial for the initial residual vector r_0 is not less than n. Then $r_j = r'_j$, $j = 0, 1, \cdots, n$.*

In other words, two Krylov subspace methods that start with the same initial vector and which produce proportional residual vectors will in fact produce identical iterates.

Proof. For the first method we have $r_i = p_i(A)r_0$ and for the second $r'_i = p'_i(A)r_0$ where the two polynomials p_n and p'_n are such that

$$p_n(0) = p'_n(0) = 1. \tag{43}$$

Since the degree of the minimal polynomial of r_0 is not less than n, the relation

$$r_i = \alpha_i r'_i, \qquad i = 0, 1, \cdots, n, \tag{44}$$

yields

$$p_i(\lambda) \equiv \alpha_i p'_i(\lambda), \qquad i = 0, 1, \cdots, n. \tag{45}$$

From (45) and (43) we get $\alpha_i = 1$, $i = 0, 1, \cdots, n$, which completes the proof. □

PROPOSITION 7. *Assume that n steps of both* ORTHORES (k) *and* DIOM (k) *with no pivoting can be achieved starting with the same initial vector x_0 and that the degree of the minimal polynomial of r_0 is not less than n. Then the iterates x_j, $j = 0, 1, \cdots, n$, produced by both algorithms are identical.*

Proof. Let r_i be the residual vectors for ORTHORES (k). According to the lemma all we have to show is that this is proportional to the residual vector produced by DIOM (k), or, equivalently, to the vector v_{i+1} produced by the incomplete Arnoldi process. Note that (42) can be written:

$$r_{i+1} = -b_i\left[Ar_i - \sum_{j=i-k+1}^{i} a_j^{(i)} r_j\right], \tag{46}$$

which is of the same form as (7). Using the fact that the sequences $\{r_i\}_{i=0,1,\cdots}$ and $\{v_i\}_{i=1,2,\cdots}$ are generated by the similar formulas (7) and (46), and that for both sequences the current vector is orthogonal to the previous k vectors, it is easy to show by induction that r_i differs from v_{i+1} only by a multiplicative factor. This completes the proof. □

Clearly, this result can be extended to DIOM (k) with pivoting (Algorithm 4), by considering only the iterates corresponding to the steps where pivoting has not occurred. Though theoretically equivalent, the two methods are quite different in their implementation. ORTHORES (k) is slightly more expensive than DIOM (k) and IOM (k), as shown in Table 1, but requires less storage than DIOM (k). However, these differences are not significant because they represent a small fraction of the total work and storage in general. The computation of b_i in ORTHORES (k) may cause the algorithm to break down, and no result explaining when this may happen seems to exist. Implicitly, the ORTHORES (k) process attempts, like Algorithm 3, to compute the approximate solutions x_j defined by (14) for all j. When one of the intermediate Hessenberg matrices H_j is singular, this solution does not exist and the process breaks down. Algorithm 4, on the other hand, discards x_j in this situation by defining it as being equal to x_{j-1} and computes x_{j+1} instead (or some x_{j+i}, $i \geqq 1$).

4. Application to other Krylov subspace methods. As indicated in [15], many algorithms using Krylov subspaces can be described with equations similar to those of IOM. A sequence of vectors v_j, respesenting the residuals rescaled as in Lemma 2, is generated by requiring some orthogonality conditions. Then the solution is obtained by (11) and (12). The only difference between the various methods resides in the orthogonality conditions forced upon the residual vectors r_j, or equivalently the v_j's. An interesting question is whether the algorithms described earlier can also be adapted to other Krylov subspace methods. The main reason why such versions are sought is that when the matrix A is not positive real, then the regular versions may break down or become unstable, because they *implicitly* solve an upper Hessenberg system with the potentially unstable Gaussian elimination with no pivoting.

The use of pivoting will be very helpful in particular for the Lanczos algorithm, considered next, as the original direct version, called the biconjugate gradient algorithm, faces serious risks of instability and breakdown (see [15]).

4.1. The Lanczos biorthogonalization algorithm. For the following discussion we recall the essential of the Lanczos algorithm for solving a linear system of the form $Ax = b$. See [6], [9], [15].

ALGORITHM 5 (Lanczos).

1. Choose an initial vector x_0 and compute $r_0 = b - Ax_0$. Define $v_1 := u_1 := r_0/(\beta := \|r_0\|)$.
2. For $j = 1, 2, \cdots, m$ do:

$$\tilde{v}_{j+1} := Av_j - \alpha_j v_j - \beta_j v_{j-1}, \tag{47}$$

$$\tilde{u}_{j+1} := A^T u_j - \alpha_j u_j - \delta_j u_{j-1} \quad \text{with } \alpha_j := (Av_j, u_j), \tag{48}$$

$$\delta_{j+1} := |(\tilde{v}_{j+1}, \tilde{u}_{j+1})|^{1/2}, \qquad \beta_{j+1} := \text{sign}\,[(\tilde{v}_{j+1}, \tilde{u}_{j+1})]\delta_{j+1}, \tag{49}$$

$$v_{j+1} := \frac{\tilde{v}_{j+1}}{\delta_{j+1}}, \qquad u_{j+1} := \frac{\tilde{u}_{j+1}}{\beta_{j+1}}. \tag{50}$$

3. Form the approximate solution

$$x_m := x_0 + V_m y_m \tag{51}$$

in which $V_m = [v_1, v_2, \cdots, v_m]$ and

$$y_m = H_m^{-1}(\beta e_1) \tag{52}$$

where H_m is the tridiagonal matrix defined by

$$H_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \beta_m \\ & & & \delta_m & \alpha_m \end{bmatrix}. \tag{53}$$

A direct version of this algorithm exists and was due to Lanczos himself. The algorithm was neglected for a long time because it faces serious stability problems; see [6], [15].

The similarity of part 3 of this algorithm with part 3 of Algorithm 2 indicates that a direct version which uses the *LU* factorization with partial pivoting can easily be formulated. The development of the new algorithm is identical with that of DIOM (k) described in § 3, and we will omit the details. This does not, however, overcome all of the difficulties associated with this algorithm, because the process of building the sequence $\{v_i\}_{i=1,2,\cdots}$ can itself be troublesome. The problem of building the Lanczos vectors in the nonsymmetric case was addressed by Parlett and Taylor [13], who suggest an alternative algorithm which handles the breakdowns associated with the construction of the sequence v_i. This a combination of Parlett and Taylors' "look-ahead" Lanczos [13] process for constructing the Lanczos vectors v_j and our technique of obtaining the approximation x_m as implemented in Algorithm 4 constitute a more reliable version of Algorithm 5.

4.2. Krylov subspace methods based on conjugate residuals. In the context of symmetric indefinite problems, Paige and Saunders [10] proposed an algorithm, named MINRES, which at each step minimizes the residual norm of the residual vector over the Krylov subspace K_m. MINRES uses the same *LQ* factorization of the matrix H_m as SYMMLQ. The algorithm proposed in § 3.3 can also be extended in a similar way to yield a generalization of Paige and Saunders's MINRES to nonsymmetric problems. The details of the algorithm and its properties are beyond the scope of this paper. However, we would like to make an important remark. The nontruncated version of the resulting MINRES algorithm is equivalent to the ORTHOMIN (∞) algorithm [18], [5] which has the property that its residual vectors are semiconjugate, i.e. $(Ar_j, r_i) = 0$, $i < j$. One might then think that, if we truncate the process as is done for the SYMMLQ-like method of § 3.3, we will find an equivalent (and hopefully better) version of ORTHOMIN (k), the truncated version of ORTHOMIN (∞). A more careful analysis shows that this is not the case, i.e., that the nonsymmetric truncated MINRES-like extension of the method described in § 3.3 is not equivalent to ORTHOMIN (k).

An alternative is to generate a direct version by imposing on the v_i's the orthogonality condition known to be satisfied by the residual vectors of the original algorithm. In our case we would like the residual vectors to be semiconjugate [3], which means that the v_i's should satisfy

$$(Av_j, v_i) = 0, \qquad i = 1, 2, \cdots, j-1. \tag{54}$$

The above sequence of vectors would lead to the generalized conjugate residual method, or ORTHOMIN (∞) [3]. Again the computation of the system of vectors satisfying (54) becomes uneconomical as j increases, and a natural idea is to replace such a condition by an incomplete orthogonality condition. Note that the incomplete version of the algorithm thus obtained is equivalent neither to the truncated version ORTHOMIN (k) of ORTHOMIN nor to the truncated version of MINRES.

Computing the sequence of vectors v_i by the recurrence (7) in which the v_i's satisfy the new orthogonality conditions $(Av_j, v_i) = 0$, $i = j-k+1, j-k+2, \cdots, j-1$, and $\|Av_{j+1}\| = 1$, may break down or become unstable. The reason for this is that we are attempting to orthonormalize a sequence of vectors with respect to an indefinite inner product; i.e., we can have $(Av, v) = 0$ for $v \neq 0$. A process similar to the one suggested by Parlett and Taylor can be applied to the sequence of v_i's because in both cases we implicilty deal with the same problem of constructing a sequence of orthogonal polynomials with respect to some indefinite inner product; see [15], [7]. The combination of such a process and the technique using partial pivoting for forming the approximate solution of Algorithm 4 can again be combined to yield a more robust algorithm.

5. Practical considerations.

5.1. General comments. As mentioned in the introduction, one attractive feature of DIOM (k) is its ability to deal with symmetric indefinite systems and nonsymmetric systems. A comparison with SYMMLQ would indeed show that DIOM (2) requires one more vector of storage than SYMMLQ, while computationally each step of DIOM (2) requires $7N$ multiplications against $9N$ for SYMMLQ (see Table 2.) Note that according to the comments following Algorithm 4, each step of DIOM (k) would cost $(3k+2)N$, which for $k=2$ gives $8N$ operations instead of the $7N$ claimed. However, because of the symmetry of the problem, we save one inner product in the formation of v_{j+1}, which explains the result.

There exist other methods which are also less expensive than SYMMLQ. For example Chandra's SYMMBK version [2] based upon the use of 2×2 pivots in the LU factorization of H_m also requires $7N$ multiplications per step. However, the Bunch and Kaufman factorization is still potentially unstable while the LU factorization with partial pivoting is always stable for tridiagonal matrices, as is well known [19, p. 219]. Table 2 shows the work and storage required for several methods dealing with symmetric indefinite problems, including SYMMLQ [10], SYMMBK [2], MCR [2], Stoer and Freund's method [17]. The indirect Lanczos method, equivalent to IOM (2), was described by Parlett [11], who observed that it can be competitive in some environments where the I/O can be realized inexpensively. See also [16].

TABLE 2
Work and storage of some methods for solving symmetric indefinite systems.

Method	Mult./step	Storage
DIOM (2)	$7N$	$6N$
SYMMLQ	$9N$	$5N$
SYMMBK	$7N$	$6N$
MCR	$7N$–$9N$	$7N$
Stoer & Freund	$8N$	$7N$
Parlett (*)	$6N$	$3N$

* This method uses secondary storage. The table reports core memory requirement only.

These methods deal with symmetric problems, but most of them can easily be generalized to nonsymmetric problems, although this does not seem to have been considered so far in the literature.

Consider the four classes of problems enumerated in the introduction. For the class of symmetric positive definite linear systems, there are many effective methods. It seems that a common way of dealing with symmetric indefinite problems is the SYMMLQ algorithm. The third class of problems, dealing with nonsymmetric systems with indefinite symmetric parts, can be effectively handled by several methods including ORTHOMIN (k), IOM (k), DIOM (k), Chebyshev iteration, Concus, Golub and Widlund's generalized conjugate gradient method, etc.

The last class of problems, i.e. the class of nonpositive real problems, is more difficult. Our algorithm DIOM (k) can be applied especially in the cases where the nonsymmetric part $(A-A^T)/2$ is small compared with the symmetric part. When this is *not the case*, we observe that in order for the process to converge, the parameter k must be quite large, thus rendering the method uneconomical. The process may diverge or be very slow if k is too small.

These observations suggest that the Krylov subspace methods may not be suitable for nonpositive real systems with large nonsymmetric parts or such that the origin is well inside their spectra. In fact since k must be large (e.g. $k>9$) for convergence to hold, one might find it preferable to solve the normal equations using e.g. the conjugate gradient method. Moreover, note that in the extreme case where A is unitary, the eigenvalues are on the unit circle, and, as suggested by the theory [14], the convergence of Krylov subspace methods can become very slow. However, the normal equations are clearly trivial to solve since A^TA is the identity matrix. This suggests that the choice between solving the normal equations and using a Krylov subspace method is not always an easy one to make.

5.2. Heuristics. In order to enhance the efficiency of a code based upon IOM (k) or DIOM (k), a number of heuristics are needed. The most important of them are described below.

Dynamic choice of the parameter k. In an efficient implementation of IOM (k), or DIOM (k), we must include a process which chooses automatically the parameter k. Indeed, the user does not in general have any idea of a reasonable choice for k. The possibility of choosing k in a dynamical way is based on the fact that k can be *reduced* during the algorithm without changing the orthogonality relation of Proposition 5. Note, however, that k cannot be reincreased. What this means is that we can start the algorithm with some large k (in our code k starts with the value 9) and then reduce it progressively according to some criterion. The criterion that we use is related to the fact that when the matrix is almost symmetric (or skew-symmetric), then the elements $h_{i,j}$ of the jth column of H_m with $i<j-2$ are small, and can therefore be neglected. This suggests that at a given step j we should subtract from k the number of small elements h_{ij}, where i is between $j-2$ and $j-k+1$. Specifically we redefine k from

$$k := k - \max_{\mu}\left\{\mu \text{ s.t. } \sum_{i=j-k+1}^{j-\mu+1} |h_{ij}| < \text{tol}_1 \sum_{i=j-k+1}^{j} |h_{ij}|\right\}$$

where tol_1 is some tolerance parameter (In our code tol_1 was set to 1.e-03.). The formula above should be modified so as not to yield k less than 2.

With this empirical formula, symmetry or near-symmetry is easily detected, and as a consequence the computational work may be significantly reduced.

Restarting. In IOM (k), the version using secondary storage, it is a necessity to restart the algorithm because of storage. It is also often more effective to include a restarting strategy even for the direct version DIOM (k). Such a strategy would restart the algorithm whenever the convergence becomes unsatisfactory. More precisely the following heuristics have been found to be effective:

$$\text{If } \left(\frac{\rho_j}{\rho_{j-p}}\right) > \text{tol}_2 \quad \text{then restart}$$

where p is some fixed integer (e.g. 5 as in our code), and tol_2 is some positive tolerance parameter. The above criterion for restarting has the following interpretation: restart if the residual norms do not decrease by a sufficient amount in p steps. For IOM (k) we can restart with the vector x_{j-p} which has a smaller residual, but this cannot be done for DIOM (k) unless we save the vector x_{j-p}. The restarting strategy was found to be quite effective for some difficult cases (see § 6.3), but does not change much when convergence is fast enough.

Preconditioning. The efficiency of the incomplete orthogonalization methods can be improved by using preconditioning techniques. One difficulty, however, is that most of the known preconditions assume (directly or indirectly) that $(A+A^T)/2$ is positive definite. A very simple remedy is to add αI to the original matrix, where α is a scalar such that $B=A+\alpha I$ is positive real, and use as preconditioning the preconditioning associated with B. This can be effective in case α is not too large.

6. Numerical experiments. All the numerical experiments have been performed on a DEC-20 computer. Single precision (unit roundoff $\approx 3.7\times 10^{-09}$) is used in the first experiment while double precision (unit roundoff $\approx 10^{-19}$) is used elsewhere.

6.1. Symmetric indefinite problems. We begin with an experiment illustrating the behaviors of DIOM (2), SYMMLQ and the regular conjugate gradient (CG) method on a symmetric indefinite problem. Note that the regular conjugate gradient method may break down if A is indefinite, and is therefore not recommended in general for indefinite problems. It is applied here for the sole purpose of illustration. We choose to take an example from [10], in which the matrix A is of the form $A=B^2-\mu I$, where B is the tridiagonal matrix with typical nonzero row elements $(-1, 2, -1)$ and $\mu=\sqrt{3}$. Note that μ is not an eigenvalue of A, and that it is not near either extremity of the spectrum. In order to demonstrate the fact that IOM (2) and SYMMLQ have similar behavior on this example, we use single precision arithmetic. The problem solved is $Ax=b$, where $b=Ae$, $e=(1, 1, 1, \cdots, 1)^T$. The initial vector is a random vector, the same for the three methods DIOM (2), SYMMLQ, CG. Figure 6.1 compares the behaviors of the three methods for the steps $m=38, 39, \cdots, 65$. The first 37 steps yield almost identical residual norms for the three algorithms and have not been reproduced.

The figure shows that both SYMMLQ and DIOM (2) are superior to the regular conjugate gradient method, but SYMMLQ is not superior to DIOM (2). In fact DIOM (2) is slightly better on this example through the difference is not significant. More significant is the difference obtained when a less efficient way of generating the Lanczos vectors v_i is utilized. Indeed, in our first experiments with the above example we found DIOM (2) significantly more accurate than SYMMLQ. A careful analysis showed that the reason for this superiority was due to the fact that the original code of SYMMLQ was not using the best formula for generating the Lanczos vectors. As mentioned in [12], it is more accurate to generate the Lanczos vectors by the recurrence

$$q := Av_j - \beta_j v_j,$$

$$\alpha_j := (q, v_j),$$

$$q := q - \alpha_j v_j,$$

$$v_{j+1} := \frac{q}{(\beta_{j+1} := \|q\|)}$$

rather than by

$$q := Av_j,$$

$$\alpha_j := (q, v_j),$$

$$q := q - \alpha_j v_j - \beta_j v_j,$$

$$v_{j+1} := \frac{q}{(\beta_{j+1} := \|q\|)}.$$

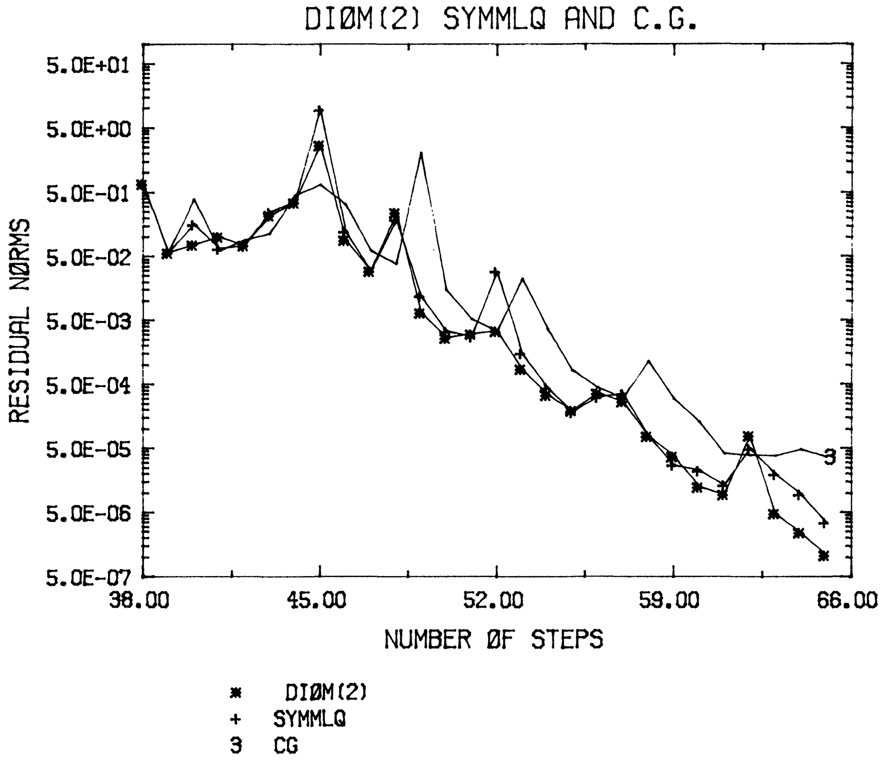

FIG. 6.1. *Comparison of* DIOM (2), SYMMLQ, CG *on a symmetric indefinite problem of dimension* 50.

Remarks. 1. The plot of Fig. 6.1 reports the numerical results corresponding to the first (best) of the above formulations for *both* SYMMLQ and DIOM (2).

2. Similar observations are made when double precision is used.

3. The residual norms in Fig. 6.1 are obtained for DIOM (2) by the formula (13) and for SYMMLQ by an equivalent formula. Both formulas have been checked to produce accurate result in the final step. Note that after step 65, these formulas deteriorate slowly as the maximum possible accuracy given the norm of A and the unit roundoff is being reached, so the estimates of the residual norms become meaningless.

All this demonstrates the important fact mentioned earlier that the main source of error lies in the computation of the v_i's rather than in the formation of the approximate solution by formulas (11) and (12).

6.2. Nonsymmetric problems with positive real matrices. In this test we compare the direct version DIOM (k) with the indirect version IOM (k) on the following model problem:

$$A=\begin{bmatrix} B & -I & & & \\ -I & B & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & B \end{bmatrix} \quad \text{with } B=\begin{bmatrix} 4 & t_1 & & & \\ t_2 & 4 & \ddots & & \\ & \ddots & 4 & \ddots & \\ & & \ddots & \ddots & t_1 \\ & & & t_2 & 4 \end{bmatrix}$$

and $\dim(A)=N=200$, $\dim(B)=n=10$, $t_1=-1+\delta$, $t_2=-1-\delta$, where δ is some parameter. The above example originates from the centered difference discretization of the operator $-\Delta+c\partial/\partial x$, where c is a constant.

The performances of IOM (4) and DIOM (4) have been compared on the above example with $\delta=0.5$, $b=Ae$, $e=(1,1,\cdots,1)^T$. This test was performed in double precision. For simplicity none of the heuristics has been used, i.e. the algorithm is not restarted and k is constantly equal to 4. The process is stopped as soon as the residual is less than 10^{-5}. This has required 57 steps. As expected, the residual norms and the final iterates produced by both algorithms are identical. The run times on the DEC-20 are approximately as follows:

IOM (4): 5.60 sec, DIOM (4): 3.60 sec.

Note that IOM (4) requires $5N$ vectors of core memory storage while DIOM (4) requires $9N$. It is interesting to decompose the run times for IOM (4) into I/O time and computing time. The time for writing the vectors v_i's into disk memory and reading them back when forming the solution is about 2.50 sec, i.e. 47% of the overall CPU time. The I/O time can be further decomposed into write time = 1.39 sec and read time = 0.91 sec. This distribution is obviously very much machine dependent, and the comparison may change completely for other architectures. It may even happen that IOM becomes faster than DIOM in cases where the I/O time can be masked by performing much of the computation and the I/O in parallel.

6.3. Indefinite and nonsymmetric problems. In this example we test DIOM (k) on the matrix $B=(A-\mu I)$, where A is defined as in the previous experiment, with δ set again to 0.5 and where μ is chosen equal to 0.25. This is a nonsymmetric and indefinite problem.

The right-hand side is defined as previously, and the initial vector is again a random vector with an initial residual norm of 19.08 .... The process is stopped as soon as the residual norm is below 10^{-5}. A straightforward application of DIOM (4), with a fixed k and no restarting, converged in 89 steps. Then we used a preconditioning matrix M the incomplete Choleski factorization associated with the Laplace operator, i.e. the incomplete Choleski factorization of $(A+A^T)/2$. The system $M^{-1}Ax=M^{-1}b$ was then solved by a call to DIOM (2). This preconditioned DIOM produced a generalized residual vector $M^{-1}r_n$ of norm less than 10^{-5} in 31 steps, thus significantly improving the previous performance. Note that, surprisingly, DIOM (k) with $k>2$ did not perform better than with $k=2$ since it took 41 steps for DIOM (3) to converge and 48 steps for DIOM (4).

As μ increases, the problem becomes more difficult to solve. When $\mu = 0.5$, for example, DIOM (k) either diverges (e.g. for $k = 2$) or showed signs of very slow convergence. Using the same preconditioning as above, IOM (7) converged in 125 steps. The restarting strategy used was the one described in § 5.2 with tol $2 = 1$. The criterion for restarting was tested every $p = 5$ steps and, at a minimum, 10 steps were taken at each iteration. With this strategy the process was restarted at steps 20, 30, 70, and 110. Note that we took tol $1 = 0$, which means that k was constantly equal to 7.

When μ becomes even larger, the above preconditioning does not improve the convergence, which can become very poor. It seems more appropriate to use the conjugate gradient method applied to the normal equations in those cases.

Acknowledgments. The mathematical equivalence between ORTHORES (k) and IOM (k) (§ 3.5) was established after a suggestive question was raised by one of the referees. The author is indebted to both referees for their instructive and helpful comments.

REFERENCES

[1] O. Axelsson, *Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 1–16.

[2] R. Chandra, *Conjugate gradient methods for partial differential equations*, PhD thesis, Tech. Rep. 129, Yale Univ., New Haven, CT, 1981.

[3] S. C. Eisenstat, H. C. Elman and M. H. Schulz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[4] H. C. Elman, *Preconditioned conjugate gradient methods for nonsymmetric systems of linear equations*, Advances in Computer Methods for Partial Differential Equations-IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, NJ, 1981, pp. 409–417.

[5] ———, *Iterative methods for large sparse nonsymmetric systems of linear equations*, PhD thesis, Technical Report 229, Yale Univ., New Haven, CT, 1982.

[6] R. Fletcher, *Conjugate gradient methods for indefinite systems*, in Proceedings of the Dundee Biennial Conference on Numerical Analysis 1974, Univ. of Dundee, Scotland, New York, 1975, pp. 73–89.

[7] W. B. Gragg, *Matrix interpretation and applications of continued fraction algorithm*, Rocky Mountain J. Math., 4 (1974), pp. 213–225.

[8] K. C. Jea and D. M. Young, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

[9] C. Lanczos, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[10] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–624.

[11] B. N. Parlett, *A new look at the Lanczos algorithm for solving symmetric systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 323–246.

[12] ———, *The Symmetric Eignevalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[13] B. N. Parlett and D. Taylor, *A look ahead Lanczos algorithm for unsymmetric matrices*, Technical Report PAM-43, Center for Pure and Applied Mathematics, Berkeley, CA, 1981.

[14] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comput., 37 (1981), pp. 105–126.

[15] ———, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982), pp. 470–484.

[16] H. D. Simon, *The Lanczos algorithm for solving symmetric linear systems*, PhD Thesis, Univ. of California at Berkeley, Berkeley, CA, 1982.

[17] J. Stoer and R. Freund, *On the solution of large indefinite systems of linear equations by conjugate gradient algorithms*, Technical Report, Institut für Angewandte Mathematik und Statistik,Universität Würzburg, Würzburg, W. Germany.

[18] P. K. W. VINSOME, ORTHOMIN, *an iterative method for solving sparse sets of simultaneous linear equations*, in Proceedings of the Fourth Symposium on Reservoir Simulation, Society of Petroleum Engineers of the American Inst. of Mechanical Engineers, 1976, pp. 149–159.

[19] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.