# PRACTICAL USE OF SOME KRYLOV SUBSPACE METHODS FOR SOLVING INDEFINITE AND UNSYMMETRIC LINEAR SYSTEMS

Y. SAAD

Technical Report # 214
January 28, 1982

This work was supported in part by the U.S. Office of Naval Research under grant N000014-76-C-0277 and in part by NSF Grant.

## Abstract

The main purpose of this paper is to develop stable algorithms for some Krylov subspace methods. Like for the SYMMLQ algorithm in the symmetric case, our algorithms are based on stable factorizations of the banded Hessenberg matrix representing the projected part of the matrix A on the Krylov subspace. We show how an algorithm similar to Paige and Saunders' SYMMLQ can be derived for unsymmetric 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 unsymmetric or symmetric indefinite or both unsymmetric and indefinite systems of linear equations can be handled by a unique algorithm.

## 1. Introduction

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

$$A\,x = b \tag{1.1}$$

where A is an N x N real matrix. Recent work by Vinsome [13], Axelsson [1], Elman Eisenstat and Schultz [3], Young and Jea [6] and Saad [11] concerns Krylov subspace methods for the case where A is unsymmetric. A common feature of all of these methods is that the approximate solution belongs to the affine space $x_0 + K_m$ where the $K_m$ is the Krylov subspace

$$K_m = \text{span}[r_0, Ar_0, \ldots, A^{m-1}r_0]$$

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, possibly different from $K_m$ [11]. It is also possible to regard these processes as realizations of Galerkin's method on $K_m$. The representation of the projected part of A on $K_m$ is then in general a Hessenberg matrix instead of a tridiagonal matrix as is the case when A is symmetric.

In [12] the Incomplete Orthogonalization Method (IOM), closely related to the Galerkin process, has been proposed. It has the advantage of requiring fewer core memory than its counterparts based on direct updating of the solution at every step, but uses secondary storage. For IOM, the above mentioned representation is a banded Hessemberg matrix.

In this paper we will develop an alternative version of the IOM, that does not require secondary storage and which ,as IOM, has the advantage that it can be successfully applied to positive definite systems as well to

'mildly' indefinite systems (here positive definiteness relates to the symmetric part $(A+A^T)/2$). The principle of our approach is very similar to the one adopted by Paige and Saunders [8] which led to the successful SYMMLQ algorithm for the symmetric indefinite problems. Let us recall that SYMMLQ was based upon the stable LQ factorization of the representation of A in $K_m$, which is a tridiagonal matrix when A is symmetric. In our case we will have to factor a banded Hessenberg matrix by resorting to the LU factorization with partial pivoting. Note that such a factorization can also be applied to tridiagonal matrices which means that, when the matrix A is symmetric, we obtain an alternative of the SYMMLQ algorithm which in fact is 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. It should however be understood that when the matrix A has a large unsymmetric part and a symmetric part which has the origin well inside its 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 has a nearly positive definite symmetric part or both.

In section 2 we will briefly recall the Incomplete Orthogonalization Method. Section 3 describes an alternative version of the same algorithm, which will be called the Direct Incomplete Orthogonalization Method (DIOM). Then 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 section 5 and some numerical experiments will be reported in the last section.

## 2. Krylov Subspace Methods

### 2.1. The Full Orthogonalization Method

In the basic full orthogonalization method (or Arnoldi's method) presented in [12], one first constructs an orthonormal basis $V_m=[v_1,v_2,\ldots v_m]$ of $K_m$. An orthogonal projection method on $K_m$ is by definition a method which obtains an approximate solution of the form

$$x_m = x_0 + V_m y_m \tag{2.1}$$

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

$$V_m^T(b-Ax_m) = 0 \tag{2.2}$$

and therefore

$$y_m = [V_m^T A V_m]^{-1} V_m^T r_0 \tag{2.3}$$

The basis $V_m$ is built from the following recurrence:

$$h_{j+1,j} v_{j+1} = A v_j - \sum_{i=1}^{j} h_{ij} v_i \tag{2.4}$$

where the coefficients $h_{ij}$, i=1,..j+1 are chosen such that the vector $v_{j+1}$ is orthogonal to all the previous $v_i$'s and $||v_{j+1}||=1$.

An important property is that the matrix $V_m^T A V_m$ in 2.3 is precisely the m x m upper Hessenberg matrix whose nonzero elements $h_{ij}$ are defined by the algorithm. Furthermore the vector $V_m^T r_0$ in 2.3 is equal to $||r_0||e_1$.

Therefore the solution $y_m$ of equation 2.3 simplifies into

$$y_m = H_m^{-1} (\beta e_1) \tag{2.5}$$

where we have denoted by $\beta$ 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...m compute the vectors $v_j$ and the coefficients $h_{i,j}$ by 2.4
3. Compute the approximate solution $x_m$ from equations 2.1 and 2.5.

This will be referred to as the full orthogonalization method .

## 2.2. The Incomplete Orthogonalization Method

A serious drawback of the above algorithm is that as j increases the process becomes intolerably expensive and requires the storage of the the whole set of previous vectors $v_i$ since these are used every step. A remedy to this is to orthogonalize the current vector $A v_j$ against the k previous vectors instead of all the previous vectors. We will assume throughout that[1] $k \geq 2$. The derived algorithm, has been proposed in [12], and is called the Incomplete Orthogonalization Method. The only difference with the above

---

[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.

algorithm of full orthogonalization lies in the definition of $v_{j+1}$, which now becomes:

$$v_{j+1} = \frac{1}{h_{j+1,j}} [A v_j - \sum_{i=j-k+1}^{j} h_{ij} v_i] \tag{2.6}$$

In the above summation j-k+1 is to be replaced by 1 whenever j<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,..j , that is:

$$h_{i,j} = (Av_j, v_i) \qquad i=j-k+1,..j \tag{2.7}$$

$$h_{j+1,j} = ||A v_j - \sum_{i=j-k+1}^{j} h_{ij} v_i|| \tag{2.8}$$

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

$$H_m = \begin{vmatrix} x & x & x & x & & & & & \\ x & x & x & x & x & & & & \\ & x & x & x & x & x & & & \\ & & x & x & x & x & x & & \\ & & & x & x & x & x & x & \\ & & & & x & x & x & x & x \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \end{vmatrix} \tag{2.9}$$

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 $V_m = [v_1, v_2, \ldots, v_m]$ and $H_m$ by using formulas 2.6 and 2.7 and 2.8, for j=1,...m.

3. Compute

$$y_m := \beta H_m^{-1} e_1 \tag{2.10}$$

and form the approximate solution

$$x_m := x_0 + V_m y_m \tag{2.11}$$

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 Orthogonalizatiom 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 algorithm concerns the choice of the number of steps m. If the algorithm were to be applied with an arbitrarily chosen m, we will certainly have to restart the algorithm if the accuracy provided is insufficient. In other words the above algorithm is restarted with the initial vector replaced by the newly computed approximate solution, and this is repeated until convergence. But it is also possible that m is 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 estimating the residual norm of the intermediate approximation $x_j$ <u>without forming</u> $x_j$ . Fortunately such a formula exists and is given by (see e.g. [12])

$$||b - A x_m|| = h_{m+1,m}|e_m^T y_m| \tag{2.12}$$

As will be seen later, updating the estimate 2.12 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 x m system involved in 2.10, it is not more costly to factor $H_m$ by updating the factorixation at each step, and hence the computation of the estimate 2.12 is virtually free. It should be added that 2.12 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 [12] for more details). First choose an initial vector $x_0$, the parameter k and a maximum number of steps mmax. Compute $r_0$, and $v_1=r_0/||r_0||$. Then for j=1,2.. , compute $h_{i,j}$ and $v_{j+1}$ by 2.4. Write in secondary storage every vector $v_{j+1}$ thus generated, one by one. Simultaneously update the LU factorization of $H_j$ at each step j and the estimate 2.12 of the residual. If at step j this residual norm estimate is small enough, recall the $v_i$'s from secondary storage one by one and form the approximate solution $x_j$ by formulas 2.10 and 2.11. If j reaches mmax and the residual norm is still unsatisfactory form $x_{mmax}$ and restart the algorithm.

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 the LU factorization with partial pivoting can also be updated at each step together with the estimate 2.12 of the residual norm. These observations have already been used in our earlier paper [12]. 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 Schultz [3], Young and Jea [6], 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 the updating at every step of the LU factorization of the banded Hessenberg matrix $H_m$.

For the sake of clarity let us first present a version that does not use partial pivoting. The more stable algorithm using partial pivoting will be the object of subsection 3.2.

### 3.1. Derivation of the algorithm.

At each step m. the approximate solution $x_m$ is given by the formula

$$x_m = x_0 + \beta\, V_m\, H_m^{-1}\, e_1. \tag{3.1}$$

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

Let $H_m$ be factored as

$$H_m = L_m\, U_m \tag{3.2}$$

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

$$H_m = L_m U_m$$

$$\begin{pmatrix} x & x & x & x & & & & \\ x & x & x & x & x & & & \\ & x & x & x & x & x & & \\ & & x & x & x & x & x & \\ & & & x & x & x & x & x \\ & & & & x & x & x & x \\ & & & & & x & x & x \\ & & & & & & x & x \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & \\ 1_2 & 1 & & & & & & \\ & \ddots & \ddots & & & & & \\ & & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & & & \\ & & & & \ddots & \ddots & & \\ & & & & & 1_m & 1 & \end{pmatrix} \begin{pmatrix} x & x & x & x & & & & \\ & x & x & x & x & & & \\ & & x & x & x & x & & \\ & & & x & x & x & x & \\ & & & & x & x & x & x \\ & & & & & x & x & x \\ & & & & & & x & x \\ & & & & & & & x \end{pmatrix}$$

From 3.1 and 3.2 the solution $x_m$ satisfies

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1)$$

Following Paige and Saunders [8] let us set

$$W_m = V_m U_m^{-1} \tag{3.3}$$

and

$$z_m = \beta L_m^{-1} e_1 \tag{3.4}$$

We will denote by $w_i$ the i-th column of the N by 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 $\xi_m$ which is equal to $\xi_m = -1_m \xi_{m-1}$. In other words we have:

$$z_m = \begin{pmatrix} z_{m-1} \\ \hline \xi_m \end{pmatrix} \tag{3.5}$$

It is then clear that $x_m$ can be updated at every step through the formula:

$$x_m = x_{m-1} + \xi_m w_m \tag{3.6}$$

where the last element $\xi_m$ of the vector $z_m$ satisfies the recurrence

$$\xi_m = -1_m \xi_{m-1} , \quad m=2,3... \tag{3.7}$$

starting with $\xi_1 = \beta = ||r_0||$. Recall that we denote by $1_m$ the element in position m,m-1 of the matrix $L_m$.

The vectors $w_m$ can in turn be updated quite simply since we have from their definition 3.3

$$w_m = [v_m - \sum_{i=m-k+1}^{m-1} u_{im} w_i ]/u_{mm} \tag{3.8}$$

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,...until convergence do:
   - Compute $h_{im}$, i=m-k+1,..m+1, and $v_{m+1}$ by formulas 2.6, 2.7, and 2.8.
   - 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 $\xi_m$ from 3.7.
   - Compute

$$w_m = [v_m - \sum_{i=m-k+1}^{m-1} u_{im} w_i]/u_{mm}$$

   - 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 the formula 2.12 can be efficiently used, because the last element of $y_m$ is precisely $\xi_m/u_{mm}$, hence

$$||b-Ax_m|| = h_{m+1,m} |\xi_m/u_{mm}| \tag{3.9}$$

which requires only 2 operations at each step.

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 $Av_m$). Note that the division by $u_{mm}$ in 3.8 can be avoided by simply using the vectors $w'_j=u_{jj}w_j$ in place of the $w_j$'s and by keeping track of the scaling factors $u_{jj}$.

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 unstable LU factorization of $H_m$ and can result in an instable 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 3.2 we now introduce the following LU factorization with partial pivoting of the matrix $H_m$

$$H_m = P_2E_2P_3E_3 \cdots P_mE_m U_m \tag{3.10}$$

where each $P_j$ is an elementary permutation matrix, and $E_j$ is an elementary transformation( [14]) having the structure:

$$E_j = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & 1 & 1 & 0 \; 0 \; 0 \\ & & & & 0^{j} & & \\ & & & & 0 & & \\ & & & & 0 & & \\ & & & & \vdots & & \end{bmatrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \text{. .j-th row} \\ \\ \\ \\ \\ \end{matrix}$$

(j-1)st column

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.

As before the approximation $x_m$ is defined by 3.1. Letting again

$$W_m = V_m \, U_m^{-1} \tag{3.11}$$

we get

$$x_m = x_0 + W_m \, z_m$$

where $z_m$ is now defined as

$$z_m = E_m^{-1} \, P_m \, E_{m-1}^{-1} P_{m-1} \cdots E_2^{-1} \, P_2 \, (\beta e_1)$$

Note that the vectors $w_j$ can be updated in the same way as before by use of 3.8, except that the summation is now form m-k to m-1.

We claim that there are formulas similar to 3.6, 3.7 for updating the vectors $z_m$. This is because we have

$$z_m = E_m^{-1} P_m \bar{z}_{m-1} \tag{3.12}$$

with

$$\bar{z}_{m-1} = \begin{vmatrix} \\ z_{m-1} \\ \\ \hline 0 \end{vmatrix} \tag{3.13}$$

Equations 3.12 and 3.13 show that there are two cases, depending on whether interchange has or has not been performed at previous step:

1. Either the rows m−1 and m have not been permuted. In this case the vector $z_m$ is defined as before by 3.5 and 3.7.
2. Or there has been a permutation of rows (m−1) and m, in which case:

$$z_m = \begin{vmatrix} \\ z_{m-2} \\ \\ \hline 0 \\ \xi_{m-1} \end{vmatrix} \tag{3.14}$$

where $\xi_{m-1}$ 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 formulas 3.6 and 3.7. If on the other hand rows m−1 and m have been permuted then the expression 3.14 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 $\xi_m$ as equal to $\xi_{m-1}$. This means that when a permutation occurs, all we have to do is skip the application of

the updating formulas 3.6, 3.7 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 m-th step as well as the element $u_{mm}$, since the factorization is updated at each step.

Typically the updating of the factorization of $H_m$ at the m-th step can be performed as follows. First, using the k+1 previous pivots $1_{m-k}$, $1_{m-k+1}$,...$1_m$ tranform 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 m-th 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 $1_{m+1}$. Clearly the matrices $H_m$, $E_m$, $U_m$, need not be saved. All we need is the k+1 previous pivots $1_j$, the logical information perm(j) j=m-k,..m, defined as perm(j)=true if rows j and j+1 are permuted, and the k+1 nonzero elements of the m-th column of $H_m$, which is transformed into the m-th column of $U_m$. This constitutes little storage as k is in general small (typically k<10). We describe below the Direct Imcomplete Orthogonalization Method (DIOM(k)) algorithm derived from the above suggestions .

Algorithm 4 : DIOM(k)

1. Start. Pick an initial vector $x_0$, define $r_0 := b - Ax_0$.

2. Iterate. For m=1,2,...until convergence do

   - Compute $v_{m+1}$ and the $m^{th}$ column of $H_m$ by formulas 2.6, 2.7, and 2.8.

   - Update the LU factorization with partial pivoting of $H_m$, i.e. find the m-th column of $U_m$ and the pivotal information to be used in the next step.

   - Compute

$$w_m := [\, v_m - \sum_{i=m-k}^{m-1} u_{im} w_i \,]/u_{mm} \qquad (3.15)$$

   - If perm(m+1)=false then compute:

$$\xi_m := -1_m \, \xi_{m-1}$$

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

$$\rho_m := \beta_{m+1} |\xi_m / u_{mm}|$$

$$\text{if } \rho_m \leq \varepsilon \text{ stop}$$

   - else define:

$$\xi_m := \xi_{m-1}$$

$$x_m := x_{m-1}$$

In the above algorithm $\rho_m$ represents the estimate 2.12 of the residual norm. Thus the process is stopped when this estimate is smaller than the tolerance $\varepsilon$.

Once again DIOM($\infty$) refers to the case of full orthogonalization, that is to the case where the summations in 2.6 and 3.15 start from 1.

The amount of work is approximately the same as that of algorithm 3. Indeed when a permutation takes place we save one vector update of the form 3.6, 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 an extra N 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 k-1, i.e. we need a total of 2k+2 vectors of storage, instead of 2k+1 in Algorithm 3.

One might question the need of using algorithm 4 instead of algorithm 3 in some cases. In particular is pivoting a necessity at all if the original matrix A has a positive definite symmetric part? 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 definite suggests that it is in general better to use the more stable version of Algorithm 4, instead of Algorithm 3, in spite of the extra vector of storage involved.

### 3.3. Using the stable QR factorization.

In the SYMMLQ algorithm described by Paige and Saunders in [8] one uses the LQ factorization $H_m=L_mQ_m$. A similar algorithm using the stable QR factorization can also be developed for the incomplete orthogonalization method. Consider the orthogonal QR factorization:

$$H_m = Q_mU_m \tag{3.16}$$

where $U_m$ is, as in subsection 3.1, an mxm upper triangular matrix and $Q_m$ is an unitary orthogonal matrix, i.e. $Q_m^TQ_m=I$ . A remark which is essential is that

since $H_m$ is banded upper Hessenberg, $U_m$ will be banded upper triangular.

The reason why we prefer a 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 rotations $F_j$ of the form:

$$F_j = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & c_j & -s_j & & & \\ & & & s_j & c_j & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \quad \leftarrow \text{--- row } j$$

where $c_j = \cos(\theta_j)$, $s_j = \sin(\theta_j)$ . We will show next that $Q_m$ is the product of m-1 such rotation matrices, more precisely

$$Q_m = F_2 F_3 \ldots F_m \tag{3.17}$$

Letting as previously

$$W_m = V_m U_m^{-1} \tag{3.18}$$

$$z_m = Q_m^{-1}(\beta e_1) = Q_m^T(\beta e_1) \tag{3.19}$$

we observe that the approximate solution $x_m$ is again defined by

$$x_m = x_0 + W_m z_m \tag{3.20}$$

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

Next we would like to show how to update the factorization 3.16 and the vector $z_m$ at every step.

Suppose that at a given step we have the factorization 3.16 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 3.17 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)x(m+1) matrix obtained by completing the jxj matrix $Q_j$ by adding zeroes 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

$$\bar{Q}_m^T H_{m+1} = \begin{vmatrix} \ddots & & & & : & 0 \\ & \ddots & & 0 & : & 0 \\ & & U_m & & : & x \\ & 0 & & \ddots & : & x \\ & & & u_{mm} & : & x \\ \cdots & \cdots & \cdots & \cdots & : & \cdots \\ 0 & 0 & 0 & \beta_{m+1} & : & \alpha_{m+1} \end{vmatrix} \qquad (3.21)$$

where we have denoted for convenience by $\beta_{m+1}$ and $\alpha_{m+1}$ 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,..m. In order to cancel the element $\beta_{m+1}$ 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

$$c_{m+1} = u_{mm} / (u_{mm}^2 + \beta_{m+1}^2)^{1/2} \qquad (3.22)$$

$$s_{m+1} = \beta_{m+1} / (u_{mm}^2 + \beta_{m+1}^2)^{1/2}$$

which finally determines the next plane rotation . Note that the last column of $U_{m+1}$ is now completely available by premultiplying the last column of the matrix 3.21 by the rotation $F_{m+1}^T$ Since the elements $h_{i,m+1}$ are zeroes

for i=1,2,..j-k, only the k previous 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

$$z_{m+1} = F_{m+1} \bar{z}_j.$$

where

$$\bar{z}_m = \begin{vmatrix} \\ z_m \\ \hline 0 \end{vmatrix}$$

Hence

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

where

$$\bar{\xi}_m = c_{m+1} \xi_m$$

$$\xi_{m+1} = s_{m+1} \xi_m$$

In the above $\xi_j$ denotes the last elememt of $z_j$. Hence the approximate solution may be updated form the equation

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

Such an updating formula is however not the most economical. The reason for this is that, exactly as in subsection 3.3, at the m-th step we have all the information for obtaining the next plane rotation. In our algorithm we should therefore look ahead at step m and get 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 an updating formula passing from $x_m$ to $x_{m+1}$ can be obtained. Only at the the final step do we have to use a slightly different formula. Notice that similar arguments have been used in [8].

An algorithm based on the above developments can easily be implemented but we believe 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 its equivalent DIOM(k). This would not have been a high price to pay if it results in a substantial improvement. Such is not the case however as a number of numerical experiments show (see subsection 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 3.1, but mainly in the construction of the vectors $v_j$. Solving a banded Hessenberg system, by Gaussian elimination with partial pivoting is sufficiently stable in general. However there might exist particularly difficult cases where the more stable QR factorization has a better performance and although we prefer algorithm 4 in general, 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 algortihm 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. [14]).

Proposition 3.1: If the minimal polynomial of $v_1$ is of degree not less than j, then at least j steps of algorithm 4 are feasible.

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

$$h_{j+1,j}p_{j+1}(z) = zp_j(z) - \sum_{i=j-k+1}^{i=j} h_{ij}p_i(z) \qquad (3.23)$$

starting with $p_1(z)= 1$.

Denote by $\tilde{p}_{j+1}$ the polynomial on the right hand side of 3.23. This is a polynomial of degree j such that $h_{i+1,i}=||\tilde{p}_{i+1}(A)v_1||$, i=1,2.. 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 assuming 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.

Q.E.D.

It is important to note that in the case where $h_{m+1,m}=0$ and $u_{mm}\neq 0$ then the algorithm produces the exact solution at step m. This observation can be regarded as a consequence of equation 2.12. and is often referred to as a

'happy breakdown'. We do not know under what circumstances this may happen.

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 unsymmetric case. For the residual vectors, it was shown in [11] that at each step j the residual $r_j$ is orthogonal the the previous k residuals. This fact is a simple consequence of the following lemma:

Lemma 3.2: The residual vectors $r_m$ produced by m steps of either algorithm 3 or algorithm 4 satisfy the relation

$$r_m = - h_{m+1,m}\, e_m^T y_m\, v_{m+1} \tag{3.24}$$

where $y_m$ is defined by 2.10.

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

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1}\, e_m^T \tag{3.25}$$

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

$$r_m = b - Ax_m = b - A[x_0 + V_m y_m] = r_0 - AV_m y_m$$

$$= \beta v_1 - [V_m H_m + h_{m+1,m} v_{m+1}\, e_m^T] y_m$$

The result 3.25 follows form the fact that $H_m y_m = \beta e_1$.

Q.E.D.

The lemma asserts that the residual vectors are equal to the $v_i$'s apart from a multiplicative constant. Since the coefficients $h_{i,j}$ are chosen such that $(v_{m+1},v_i)=0$ $i=m-k+1,..m$, it follows from 3.24 that ( [12]) :

Proposition 3.3: The residual vectors produced by algorithm 3 or 4 satisfy the orthogonality relation:

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

In other words the current residual is orthogonal to the previous k residuals. Note that the above lemma also proves the result 2.12.

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 unnecessary to take $k>2$, because all of the algorithms DIOM(k) with $k\geq 2$ are equivalent to DIOM(2) (See [11]). 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 3.4: For the direct version of the Full Orthogonalization Method (DIOM(∞)), with no pivoting, the following semi-conjugacy relation is satisfied by the vectors $w_i$:

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

Proof: Multiplying equation 3.25 on the right by $U_m^{-1}$ gives:

$$AW_m = V_m L_m + (h_{m+1,m}/u_{mm}) \; v_{m+1} \; e_m^T \tag{3.26}$$

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 x m matrix $[V_m^T V_m]$ is the identity. Finally we get

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

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

Q.E.D.

From the above proof it is easy to see that the proposition does not extend to the case of Incomplete Orthogonalization. A somehow weaker result is, however, proved below.

Proposition 3.5: The following orthogonality relations hold for DIOM(k) with no pivoting:

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

Proof: Let us start with equation 3.26, which is still valid, and multiply both sides on the left by $V_m^T$ to get

$$V_m^T A W_m = V_m^T V_m L_m + \mu V_m^T v_{m+1} e_m^T \tag{3.27}$$

With $\mu$ defined in the proof of the previous proposition. Careful matrix interpretation of equation 3.27 shows that $V_m^T A W_m$ has the following stucture:

$V_m^T A W_m =$

$V_m^T V_m$ $L_m$ $V_m^T v_{m+1} e_m^T$

$$\begin{vmatrix} 1 & & x & \cdots & \\ & 1 & & x & \cdots \\ & & \cdot & 0 & x & \cdot \\ x & & & \cdot & & x \\ \cdot & x & 0 & \cdot & & \\ \cdot & \cdot & x & & \cdot & \\ \cdot & \cdot & \cdot & x & & 1 \end{vmatrix} \begin{vmatrix} 1 & & & & \\ x & 1 & & & \\ & x & \cdot & & 0 \\ & & x & \cdot & \\ & & & x & \cdot \\ & 0 & & x & \cdot \\ & & & & x \ 1 \end{vmatrix} + \mu \begin{vmatrix} & & & 0 & x \\ & & & 0 & x \\ & & & \cdot & x \\ & & 0 & \cdot & x \\ & & & \cdot & 0 \\ & & & & 0 \\ & & & 0 & 0 \end{vmatrix} \leftarrow \text{row } m-k+1$$

Hence

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

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.

Q.E.D.

## 4. Application to other Krylov subspace methods.

As indicated in [11], many algorithms using Krylov subspaces can be described with equations similar to those of IOM. A sequence of vectors $v_j$,

representing the residuals rescaled as in lemma 3.2, is generated by requiring some orthogonality conditions. Then the solution is obtained by the equations 2.10 and 2.11. 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 decribed earlier can also be adapted to other Krylov subspace methods. The main reason why such versions are sought is that when the matrix A does not have a positive definite part, then the regular versions face the risk of breakdown and instability, 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 [11]).

### 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 [4, 7, 11].

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..m do:

$$\tilde{v}_{j+1} := Av_j - \alpha_j v_j - \beta_j v_{j-1} \tag{4.1}$$

$$\tilde{u}_{j+1} := A^T u_j - \alpha_j u_j - \delta_j u_{j-1} \tag{4.2}$$

with $\alpha_j := (Av_j, u_j)$

$$\delta_{j+1} := |(\tilde{v}_{j+1}, \tilde{u}_{j+1})|^{1/2} , \quad \beta_{j+1} := \text{sign}[(\tilde{v}_{j+1}, \tilde{u}_{j+1})]\ \delta_{j+1} \tag{4.3}$$

$$v_{j+1} := \tilde{v}_{j+1}/\delta_{j+1} , \quad u_{j+1} := \tilde{u}_{j+1}/\beta_{j+1} \tag{4.4}$$

3. Form the approximate solution

$$x_m := x_0 + V_m y_m \tag{4.5}$$

in which as before $V_m = [v_1, v_2, .. v_m]$ and

$$y_m = H_m^{-1}(\beta e_1) \tag{4.6}$$

where $H_m$ is the tridiagonal matrix defined by

$$H_m = \begin{vmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_m \\ & & & \delta_m & \alpha_m \end{vmatrix} \tag{4.7}$$

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 [4], [11].

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 the one for DIOM(k) described in section 3, and we will omit the details. This does not, however, overcome all of difficulties associated with this algorithm, because the process of building the sequence $\{v_i\}_{i=1,2..}$ faces itself risks of breakdown. The problem of building the Lanczos vectors in the unsymmetric case was addressed by Parlett and Taylor [10] who suggest an alternative algorithm which takes care of the breakdowns associated with the construction of the sequence $v_i$. Thus a combination of Parlett and Taylors' Look ahead Lanczos [10] 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.**

Similar considerations hold for the ORTHOMIN(k), GCR methods(see [13], [3]). However let us start with an important remark. Based on the orthogonality properties, we may say that the methods considered in [3] are in a way generalizations of the (symmetric) minimal residual method [8], or conjugate residual method [2] while DIOM(k) generalizes the conjugate gradient method. One might therefore think that a generalization of the corresponding MINRES algorithm of Paige and Saunders [8] to the unsymmetric case, is straightforward . A more carefull analysis seems to show that, unfortunately, such is not the case. This comes mainly form the fact that the vectors $v_i$ are not orthogonal.

However one can certainly generate a direct version as suggested earlier 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 A-conjugate [3], which means that the $v_i$'s should satisfy:

$$(Av_j, v_i) = 0 \quad i=1,2, \ldots j-1 \tag{4.8}$$

The above sequence of vectors would lead to the Generalized Conjugate Residual Method, or ORTHOMIN [3].

Again the computation of the system of vectors stisfying 4.7( 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 no longer equivalent to the truncated version ORTHOMIN(k) of ORTHOMIN.

Computing the sequence of vectors $v_i$, by the recurrence 4.7' or by its truncated form faces as in the Lanczos algorithm, risks of breakdown because we are attempting to orthogonalize a sequence of vectors with respect to an indefinite inner product, i.e. we can have $(Av,v)=0$ for $v \neq 0$ . It is obvious that 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 implicitely deal with the same problem of constructing a sequence of orthogonal polynomials with respect to some indefinite inner product. See [11, 5]. 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 an algorithm having the advantage of being more robust for indefinite systems.

## 5. Practical Considerations.

### 5.1. General comments.

As mentioned in the introduction, one attractive feature of DIOM(k), is its ability to deal efficiently with symmetric indefinite systems and unsymmetric systems. A comparison with SYMMLQ, would indeed show that DIOM(2) requires the same storage as SYMMLQ, while computationally each step of DIOM(2) requires 7N additions/multiplications against 9N for SYMMLQ. 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.

Note that other methods also exist which are less expensive than SYMMLQ. For example Chandra's SYMMBK version [2] based upon the use of 2x2 pivots in the LU factorization of $H_m$ also requires 7N adds/multiply's. However these versions do not deal with unsymmetric problems.

From the point of view of the difficulty in the numerical solution of large linear systems we may consider that there are four classes problems:

1. A is symmetric positive definite
2. A is symmetric indefinite.
3. A is unsymmetric with a positive definite symmetric part.
4. A is unsymmetric with a non positive definite part.

For the first class of problems, there are many methods that can effectively be used, e.g. the conjugate gradient method.

It seems that a commun way of dealing with the second class is the SYMMLQ

or MINRES algorithm. The third class can be treated by several efficient methods including ORTHOMIN(k), IOM(k), DIOM(k), Chebyshev iteration, etc..

The fourth class of problems is more difficult. Our DIOM(k) can be applied, especially in those cases where the unsymmetric 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 indefinite unsymmetric systems with large unsymmetric parts, or such that the origin is well inside threir spectra. In fact since k must be large (e.g. k>9) it might well be preferrable to solve the normal equations using e.g. the conjugate gradient method.

## 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 3.5 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 j-th 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 reduce k by as much as there are small elements $h_{ij}$ where i is between j-2 and j-k+1. Specifically we redefine k from

$$k_{new} := k - \max_{\mu} \{\mu \text{ s.t. } \sum_{i=j-k+1}^{j-\mu+1} |h_{ij}| < tol_1 \sum_{i=j-k+1}^{j} |h_{ij}| \}$$

where $tol_1$ is some tolerance parameter ( In our code $tol_1$ was set to 1.e-03.). The formula above should be modified such 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 preciselly the following heuristics have been found to be effective

$$\text{If } (\rho_j/\rho_{j-p}) > tol_2 \text{ then restart}$$

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 subsection 6.3) but does not change much when convergence is fast enough.

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

## 6. Numerical Experiments.

All the numerical experiments have been performed on a DEC-20 computer. Single precision (unit round off $\simeq$3.7e-09) is used in the first experiment while double precision (unit roundoff $\simeq$ 1.e-19) is used elsewhere.

### 6.1. Symmetric Indefinite Problems.

We begin with an experiment comparing SYMMLQ the conjugate Gradient method and DIOM(2) on a symmetric indefinite problem. We choose to take an example from [8], 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 $\mu$ 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 a similar behavior on this example, we use single precision

arithmetic. The problem solved is Ax=b, where b = Ae, $e=(1,1,1,\ldots 1)^T$. The initial vector is a random vector, the same for the three methods DIOM(2), SYMMLQ, CG. Figure 6-1 compares the behaviours of the three methods for the steps m=38,39...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 but the difference is not significant. More significant is the difference obtained when a less efficient way of generating the Lanczos vectors $v_i$'s is utilized. Indeed in our first experiments with the above example we found DIOM(2) significantly more efficient 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. Indeed as mentioned in [9], it is more efficient to generate the Lanczos vectors by the recurrence

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

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

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

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

rather than

$$q := Av_j$$

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

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

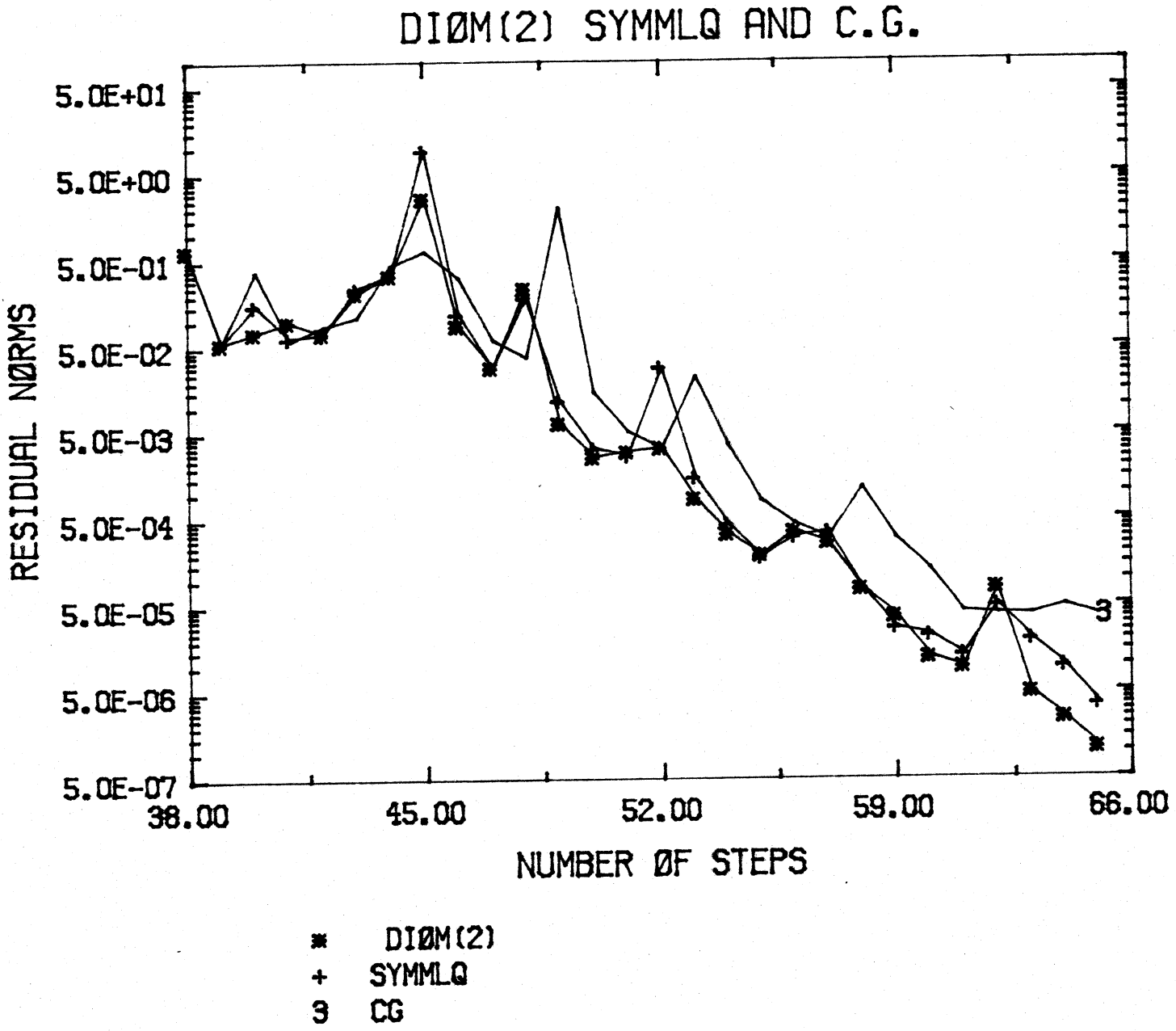


**Figure 6-1: Comparison of DIOM(2), SYMMLQ, CG on a symmetric indefinite problem, of dimension 50**

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

Remarks

1. The plot in figure 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 figure 6.1 are obtained for DIOM(2) by the estimate 2.12 and for SYMMLQ by an equivalent estimate. Both estimates have been checked to produce accurate result in the final step. Note that after step 65, these estimates deteroriate slowly as the maximum possible accuracy given the norm of A and the unit roundoff is being reached, so the estimates become meaningless.

All this demonstrates the important fact mentioned earlier that the main source of errors lies in the $v_i$'s rather than in the formation of the approximate solution by formulas 2.10 and 2.11.

## 6.2. Unsymmetric problems with positive definite symmetric parts.

In this test we compare the direct version DIOM(k) with the indirect version IOM(k) on the following model problem:

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

and dim(A) = N = 200, dim(B) = n = 10
$t_1 = -1+\delta$, $t_2 = -1-\delta$

where $\delta$ 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 $\delta=0.5$, $b = Ae$, $e=(1,1,\ldots,1)^T$. This test was performed in double precision. For simplicity none of the heuristics has been used, i.e. the algorithm is never 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[illegible] steps. As expected , the 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 storage while DIOM(4) requires 9N. It is interesting to decompose the runtimes 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'=1.39sec and 'read time'=0.91sec. This distribution is obviously very much machine dependent, and may change completely for other architechtures. 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 computations and the I/O in parallel.

### 6.3. Indefinite and unsymmetric 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 $\delta$ set again to 0.5 and where $\mu$ is chosen equal to 0.25 . This is an indefinite and unsymmetric 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 have used as preconditionning matrix M the incomplete Choleski factorization associated with the Laplace operator, i.e. the incomplete Choleski factorization of $(A + A^T)/2$. The system $M^{-1}Ax = M^{-1}b$ is 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 $\mu$ 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 restating strategy used was the one described in subsection 5.2 with tol2=1. The criterion for restarting was tested every p=5 steps and at least 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 tol1=0, which means that k was constantly equal to 7.

When $\mu$ becomes even larger, the above preconditoning 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.

## REFERENCES


[1] O. Axelsson. Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations. Lin. Alg. and its Appl. 29:1-16, 1980.

[2] R. Chandra. Conjugate Gradient Methods for Partial Differential Equations. Technical Report 129, Yale University, 1981. PhD Thesis.

[3] S.C. Eisenstat, H.C. Elman and M.H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. Technical Report 209, Yale University, 1980.

[4] R. Fletcher. Conjugate Gradient Methods for Indefinite Systems. In G.A. Watson, Editor, Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974, Springer Verlag, 1975, pp. 73-89.

[5] W.B. Gragg. Matrix interpretation and applications of continued fraction algorithm. Rocky Mountain J. of Math. 4 #2:213-225, 1974.

[6] D.M. Young and K.C. Jea. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. Lin. Alg. and its Appl. 34:159-194, 1980.

[7] C. Lanczos. Solution of systems of linear equations by minimized itearations. J. of Res. NBS 49π:33-53, 1952.

[8] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations . SIAM j. on Numer. Anal. 12:617-624, 1975.

[9] B.N. Parlett. The Symmetric Eigenvalue Problem. Prentice Hall, Englewood Cliffs, 1980.

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

[11] Y.Saad. The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems. Technical Report UIUCDCS-R-80-1036, University of Illinois at Urbana Champaign, 1980.

[12] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. Mathenatics of Computation 37:105-126, 1981.

[13] P.K.W. Vinsome. ORTHOMIN, an iterative method for solving sparse sets of simultaneous linear equations. In Society of Petroleum Engineers of AIME, Proceedings of the Fourth Symposium on Resevoir Simulation, 1976, pp. 149-159.

[14] J. H. Wilkinson. The Algebraic Eigenvalue Problem. Clarendon Press, Oxford, 1965.