005280 293

MODIFIED SUBSPACE COMPUTATION FOR EIGENVALUES AND EIGENVECTORS

by

Robert L. Taylor and James M. Rollins

Consulting Report 78-1

Prepared for

PMB Systems Engineering Inc.
500 Sansome Street
San Francisco, California 94111

April 1978 (Mod. June 1978)

Table of Contents

	Page
Table of Contents	i
1. Introduction	1
2. Isolation of Subgroups by Bisection	4
3. Subspace Iteration on a Subgroup	6
4. Selection of Start Vectors	8
5. Modification of Convergence Test	10
6. Removal of "Ghost" Vectors	11
References	12
Figures	13

1. Introduction

The calculation of eigenpairs for large structural systems often consumes a large amount of total computer time. Much effort has been expended on procedures to reliably extract selected eigenpairs at a reasonable cost. One popular procedure is the subspace method as given by Wilson and Bathe [1] and implemented in the SAPIV structural analysis system [2].

The general eigenproblem in the SAPIV program is defined by

$$\underline{A}\,\underline{v} = \underline{M}\,\underline{v}\,\underline{\Lambda} \tag{1}$$

where $\underline{A}$ and $\underline{M}$ are the symmetric stiffness and "lumped" mass matrices (i.e., $\underline{M}$ is diagonal), $\underline{v}$ is the set of eigenvectors which are mass orthonormal, and $\underline{\Lambda}$ is the array (diagonal) of eigenvalues. The diagonal arrays are stored internally in vectors.

In a typical problem the stiffness and mass arrays are of large size, and it is required to find the lowest set of eigenvalues. Often the number of eigenvalues required is much smaller than the total number, and it is in this regard that the subspace method is an extremely viable algorithm for extracting the required eigenvalues and their associated vector.

The basic subspace algorithm can be summarized as follows:

(1) Perform an $\underline{L}\,\underline{D}\,\underline{L}^T$ decomposition of $\underline{A}$.

(2) For the kth iteration, set

$$\underline{y}^k = \underline{M}\,\underline{v}^k$$

and solve the inverse iteration steps

$$\underline{A}\,\underline{x}^{k+1} = \underline{y}^k$$

for $\underline{x}^{k+1}$.

(3) Construct the subspaces

$$\underline{A}^* = (\underline{x}^{k+1})^T \underline{A}\, \underline{x}^{k+1}$$

and

$$\underline{M}^* = (\underline{x}^{k+1})^T \underline{M}\, \underline{x}^{k+1}$$

(4) Solve the reduced general eigenproblem

$$\underline{A}^* \underline{Q} = \underline{M}^* \underline{Q}\, \underline{\Lambda}^*$$

(5) Compute $\underline{v}^{k+1} = \underline{x}^{k+1} \underline{Q}$, then repeat (2) to (5) until "converged."

In use the set of vectors $\underline{v}$ consists of the p-vectors corresponding to the values of the p-lowest eigenvectors desired, plus some extra vectors which are often called "guard vectors." Let q be the size of the total number of vectors retained. Further, let n be the size of $\underline{A}$ and $\underline{M}$, and m the half-band of $\underline{A}$.

Thus, the operation counts for each step in the iteration process are:

(1) $\frac{1}{2}nm^2$ operations to construct $\underline{L}\,\underline{D}\,\underline{L}^T$ decomposition.

, nq operations to get $\underline{y}^k$ and 2mnq operations to get $\underline{x}^{k+1}$ (forward solution and backsubstitution).

(3) $\frac{1}{2}nq^2 + nq$ operations to get $\underline{M}^*$, and $\frac{1}{2}nq^2 + 2nmq$ operations to get $\underline{A}^*$.

(4) Solution of the general eigenproblem is:

C(i)i q^3, where C(i) and i are iteration parameters in the double Jacobi algorithm and depend on the threshhold and number of iterations to get convergence.

(5) nq^2 multiplications to get $\underline{v}^{k+1}$.

Usually convergence will be achieved in a few iterations; however, as the number of vectors increases, the number of iterations will also increase.

This is primarily due to the fact that convergence of the subspace is dominated by the ratio of the eigenvalue just outside the subspace (λ_{q+1} in Fig. 1) to the largest eigenvalue desired (λ_p in Fig. 1). As the size of the subspace increases, this ratio reduces (unless an extremely large set of guard vectors is retained). Another related disadvantage is that the lower values converge before the larger values, thus much wasted effort is expended to get the larger values.

It is clear from the operation counts that situations for which $q \geq m$ become inefficient in the sense that subspace iterations are expensive relative to the triangular decompositions. Furthermore, as q increases the cost of the Jacobi solution goes up extensively — the number of iterations to diagonalize $\underline{A}^*$ and $\underline{M}^*$ are of order 10-20 for some large problems, C(i) is a constant depending on the threshhold algorithm used in the Jacobi routine. If $q > 0.1\ n$, the Jacobi iteration itself can exceed a factorization cost.

From the above discussion, it is clear that as the number of sought eigenvalues increases the subspace becomes prohibitably more and more costly and some modifications are required. In this report we discuss one such modification to the algorithm provided in SAPIV. The modification consists of dividing the subspace into small groups and then using a shifted subspace algorithm to get the eigenvalues and vectors in each small group. A binary search routine using the Sturm sequence property is used to determine the bounds on the groups. Once the groups are isolated, the subspace is used with shifts to determine the eigenvalues and vectors in each small group. In this regard, special considerations are necessary to select the initial subspace subgroup vectors in order to achieve convergence. Some new difficulties are also encountered in that

"ghost" vectors occasionally appear in the interval. A strategy is employed to detect and remove these vectors. Finally, consideration of the I/O is necessary since, in general, many more iterations are necessary to get all the vectors. We discuss the aspects of our modifications in more detail in the following sections.

2. Isolation of Subgroups by Bisection

In order to establish the subgroups of size $\leq r$ (i.e., we limit the number of vectors in each subspace to be equal to or less than r), a binary bisection algorithm is employed. The value of r is determined as

$$r = \max\left(4, \sqrt{\frac{MBAND}{2}} + 1\right)$$

$$r = \min\left(r, \text{core space limitation}\right)$$

where MBAND = m the semi (half) band of $\underline{A}$. The chosen value is to balance costs of a subspace iteration and the triangular decomposition. For very large (n) problems, it may be necessary to further reduce r to utilize I/O more efficiently or because of core space limitations.

Once the value of r is determined, it is necessary to know how many eigenvalues are required and estimate the largest eigenvalues wanted (upper bound estimate) or a cut-off frequency (i.e., a frequency above which no eigenvalues are wanted). If no cut-off (upper bound) frequency is specified, the program will use the Gershgorin disk's center value

$$\beta = \text{CUTOFF} = 2 \cdot \max_{i = 1, n} \left(\frac{A_{ii}}{M_{ii}}\right)$$

Only values with nonzero M_{ii} are used. (N.B. this simple algorithm does not suffice if a consistent mass is used).

Starting from the cut-off value, a Sturm check is made using the triangular decomposition of the shifted stiffness. Accordingly, we compute

$$\underline{C} = \underline{A} - \beta\underline{M} = \underline{\tilde{L}}\,\underline{\tilde{D}}\,\underline{\tilde{L}}^T$$

and count the number of sign changes in $\tilde{D}$ (this gives the Sturm check). We should note that $\underline{C}$ is not positive definite and that we do not use pivots or double precision in the decomposition. Consequently, there is a remote possibility of the decomposition failing (none has appeared to occur at this time). From the Sturm property, we now know how many values are below β. We record the shift and the number of values by storing in

$$AT(NVAL) = \beta$$

where NVAL is number of values less than β. This records the necessary data to recover the shift and Sturm property (AT is in common /EM/ and currently is dimensioned 500; it can be increased to a much larger value without increasing core demands). We next halve the interval which has the largest number of eigenvalues; thus, in the second step we set β (i.e., SHIFT) to $\beta/2$ and repeat the process. We continue bisection until we isolate all the subgroups, i.e., see Fig. 2 [3]. Each subgroup has less than or exactly r-values between recorded shifts. It is possible that two adjacent subgroups combined have less than r-values. Consequently, we regroup these into a single group recording a final group configuration, which is then output to the line printer. At this time we are prepared to start the subspace computations for each subgroup in turn. We begin with the lower values and work to the upper ones.

3. Subspace Iteration on a Subgroup

After isolating each subgroup, the subspace algorithm is used to determine the eigenvalues and vectors in the subgroup. The algorithm given above is modified in two basic ways: (a) we use only $\tilde{q} \leq 2r$ vectors in each subspace — r values serve as guard vectors and (b) we shift to the mid-interval as evaluated by the bisection bounds (see Fig. 3). We can summarize the steps for a typical subgroup as:

(1) Compute $\underline{\tilde{L}}\,\underline{\tilde{D}}\,\underline{\tilde{L}}^T$ from $\underline{A} - \alpha\underline{M} = \underline{C}$, where α is the shift to the mid-interval.

(2) For the kth iteration set,

$$\underline{y}^k = \underline{M}\,\underline{\tilde{v}}^k \text{ and solve } (\underline{A} - \alpha\underline{M})\,\underline{x}^{k+1} = \underline{y}^k$$

where now $\underline{\tilde{v}}^k$ are the subspace vectors for the current subgroup; i.e., $\tilde{q} \times n$.

(3) Construct the subspaces

$$\underline{\tilde{A}}^* = (\underline{x}^{k+1})^T\,\underline{A}\,\underline{x}^{k+1}$$

and

$$\underline{\tilde{B}}^* = (\underline{x}^{k+1})^T\,\underline{M}\,\underline{x}^{k+1}$$

which are now $\tilde{q} \times \tilde{q}$ arrays (symmetric).

(4) Solve the reduced general eigenproblem:

$$(\underline{\tilde{A}}^* + \alpha\underline{\tilde{M}}^*)\,\underline{Q} = \underline{\tilde{M}}^*\,\underline{Q}\,\underline{\Lambda}^*$$

(5) Compute $\underline{\tilde{v}}^{k+1} = \underline{x}^{k+1}\,\underline{Q}$ and repeat (2) to (5) until $\underline{\Lambda}^* \rightarrow \underline{\tilde{\Lambda}}$ (i.e., values in interval converge).

(6) Repeat (1) to (5) for all subgroups.

Some further comments are necessary on this algorithm. These deal with how to select "good" start vectors, which will then control convergence, how to deal with any "adverse" vectors which appear (we call these

"ghost" vectors), how to utilize core and I/O effectively, and last, some comments on computed $\tilde{\underline{A}}^*$ and $\tilde{\underline{B}}^*$. We begin with the last two, namely, storage I/O and computing $\tilde{\underline{A}}^*$ and $\tilde{\underline{B}}^*$.

We do not want to compute $\tilde{\underline{A}}^*$ as shown above. The reason is that we will need two copies of an array of size $\tilde{\underline{v}}$ in core and a block of $\underline{A}$ or we will have to make a large number of I/O operations with $\underline{A}$; both are objectionable. If we look at the original implementation (i.e., see Fig. 4a), this is precisely what was done — two copies $\underline{VL}$ and $\underline{VR}$ and a block of $\underline{A}$. If q is large, the size of a block (NEQB) is controlled by the subspace computation of $\underline{A}^*$ and $\underline{B}^*$. We now have small size subgroups and will store all of $\tilde{\underline{v}}$ in core plus a block of $\underline{A}(\underline{C})$ and all the mass. In this way we will reduce I/O to just $\underline{A}$, being transferred by blocks once forward and once backward. To get all of $\tilde{\underline{v}}$ in core restricts the subspace size further for large problems where I/O would begin to dominate costs. Thus, the modified core utilization for subspace computation of $\underline{A}^*$ and $\underline{B}^*$ is as shown in Fig. 4b. We note particularly that $\underline{v}$ appears on left and right side of equations — no second copy being necessary. This implies that $\tilde{\underline{v}}^k$, $\underline{y}^k$, and $\underline{x}^k$ all are stored in the same array. This is a substantial change from that originally given in Fig. 4a.

In order to use this storage scheme, we must look further at how $\tilde{\underline{A}}^*$ can be computed; $\tilde{\underline{B}}^*$ can be computed simply since $\underline{M}$ is diagonal. Using the definitions for $\underline{C}$ and $\tilde{\underline{A}}^*$, we note that

$$\tilde{\underline{A}}^* = (\underline{x}^{k+1})^T \; \underline{C} \; (\underline{x}^{k+1})$$

$$= (\underline{x}^{k+1})^T \; \tilde{\underline{L}} \, \underline{D} \, \tilde{\underline{L}}^T \; (\underline{x}^{k+1})$$

Now let us look at the solution for $\underline{x}^{k+1}$,

$$\underline{C}\,(\underline{x}^{k+1}) = \underline{y}^{k} = \underline{\tilde{L}}\,\underline{D}\,\underline{\tilde{L}}^{T}\,(\underline{x}^{k+1})$$

which we compute by a forward solution

$$\underline{L}\ \underline{\underline{z}}_{c} = \underline{y}$$

and the backsubstitution

$$\underline{L}^{T}\,(\underline{x}^{k+1}) = \underline{D}^{-1}\ \underline{\underline{z}}_{c}$$

We note now that $\underline{z}$ can be used to compute $\underline{\tilde{A}}^{*}$ as follows. Substitute for $\underline{L}^{T}\ \underline{x}^{k+1}$ the $\underline{D}^{-1}\ \underline{z}$ and obtain

$$\underline{\tilde{A}}^{*} = \underline{\underline{z}}_{c}^{T}\ \underline{D}^{-1}\ \underline{\underline{z}}_{c}$$

$\underline{z}$ is stored in $\underline{\tilde{v}}$ also.

This is identical (with exception of reciprocal diagonal matrix) to the algorithm for getting $\underline{\tilde{B}}^{*}$. It can be computed as we compute the forward solution to the equations by blocks since $\underline{D}^{-1}$ is diagonal (otherwise we would need more than one block at a time) and all $\underline{z}$ is in core. The completion of backsubstitution gives $\underline{x}^{k+1}$ and we can then compute $\underline{\tilde{B}}^{*}$ again easily since $\underline{M}$ is diagonal. These revisions are incorporated into the rewritten subroutine REDBAK. Core revision is necessary in SAP4, MODES, SSPACEB, where NEQB is determined together with r.

4. Selection of Start Vectors

The selection of $\underline{\tilde{v}}^{0}$ for each subgroup will affect (significantly) the subspace computations in each subgroup. If a $\underline{\tilde{v}}^{0}$ is selected which has a very poor content of the eigenvectors in the subgroup, convergence may never occur in a reasonable number of iterations (e.g., say 16). Several

types of vectors have been tried. These include random vectors and selected unit vectors similar to that originally used. The vectors were mass orthogonalized in some calculations using a Gram-Schmidt process (G-S). Random vectors did not perform reliably — convergence not being achieved or eigenvalues were missed. The later subgroups especially suffered. Since very few vectors are used, the unit vectors were often not "rich enough" to get all the eigenvalues. The strategy employed was to use the largest values of

$$\underline{e}_i = \frac{M_{ii}}{(A_{ii} - \alpha M_{ii})}$$

as the unit vectors (note they are orthogonal). In the first subgroup, $\underline{M}$ is the first $\underline{\tilde{v}}$.

In performing the calculations, the vectors of the guard values which were above the previous subspace subgroup were used also. When fewer than r values were used, there were often cases in which <u>no</u> values were above. By using r values we usually have at least one being brought forward. This definitely improves the behavior — especially if the unit vectors are mass orthogonalized to these vectors. The start vectors currently used consist of the vectors above the previous subgroup, with the remaining vectors being made up from all the elements of

$$\frac{M_{ii}}{(A_{ii} - \alpha M_{ii})}$$

divided equally into the $\underline{\tilde{v}}^o$ as shown in Fig. 5. These vectors are all made mass orthonormal by the G-S process. We note that they remain mass orthonormal (but not stiffness) for all subsequent iterations; thus, the only

loss in orthogonality which can come in is that between subgroups.

The start vectors in each subgroup are also orthogonalized to the previous converged vectors to ensure that they are Ritz vectors. In theory this should preclude the possibility of ever having "ghost" vectors. In practice, however, we usually compute the eigenpairs to less than machine precision. Thus, it is possible for the vectors to still contain small portions of all the eigenvectors. In setting the start vectors these "small errors" may eventually result in start vectors which have significant components of eigenvectors below the subgroup and these lead to "ghost" vectors. A "ghost" vector is merely a vector which is a combination of vectors whose eigenvalues are both above and below the subgroup. The combination gives a value in the subspace (not an eigenvalue) which corrupts the result. The effect of these vectors can be minimized by computing eigenvalues to a precision of at least half the machine word length. In the next two sections, we discuss the procedure adopted to detect and remove the "ghost" vector from the calculations. The setting of the start vectors is performed in SINVEC; ORTHOG is used to make them mass orthogonal and NORM makes them mass orthonormal.

5. Modification of Convergence Test

To reliably detect good and bad vectors in a subgroup, we have modified the convergence test in EIGSOL. We define the tolerance as follows:

$$RTOLV_i = \min_j \left(\frac{ABS(DL(j) - D(i))}{D(i)} \right)$$

This ensures that "ghost" vectors moving through the group do not affect a determination of good vectors. In practice we then find that "ghost" vectors produce large RTOLV compared with good vectors.

6. Removal of "Ghost" Vectors

In some subgroups we have noted that more eigenvalues exist (between the boundaries defined by bisection) after a few iterations than there should be in the interval. If iteration continues, these tend to converge to an eigenvalue already in the interval and remain. In some instances, they may eventually "drift" to the next value and the next till they pass out of the interval. We call these "ghost" vectors. If we did not know better from the Sturm property during bisection, we might think they were eigenvalues — they are not! They must be removed and the sooner the better. More or less by trial and error, we have found that by the end of four subspace iterations in a subgroup we can detect the presence of bad copies quite reliably. If full print out is used, this appears at the end of each iteration as the values of L and NF. The value of L is the number of values currently between the boundaries of the subgroup, and NF is the number which should be in the interval. If at the end of four iterations we detect ghost vectors, we delete the vector in the interval which has the largest change in the eigenvalue from the previous iteration. If after two more iterations we still have a ghost, another deletion is made. This will reduce the number of guard vectors but, to date, has worked well. The routine SWEEP is called from EIGSOL to delete a vector.

References

1. K. J. Bathe and E. L. Wilson, Numerical Methods in Finite Element Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

2. E. L. Wilson, K. J. Bathe, and F. E. Peterson, "SAPIV - A Structural Analysis Program for Static and Dynamic Response of Linear Systems," EERC Report 73-11, University of California, Berkeley, 1973 (revised 1974).

3. A. Jennings and T.J.A. Agar, "Hybrid Sturm Sequence and Simultaneous Iteration Method," Symp. on Appl. of Comp. Meth. in Engr., University of Southern California, Los Angeles, 1977.

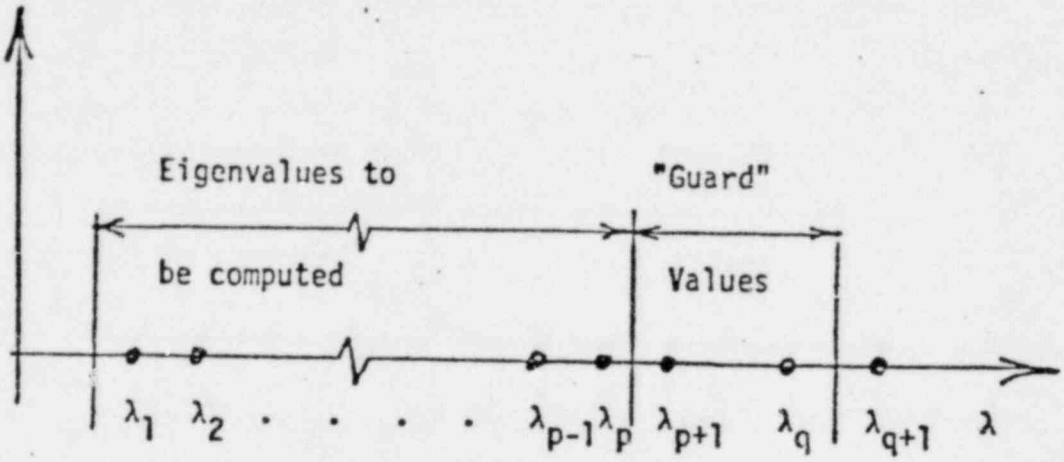


FIG. 1. SUBSPACE EIGENVALUES

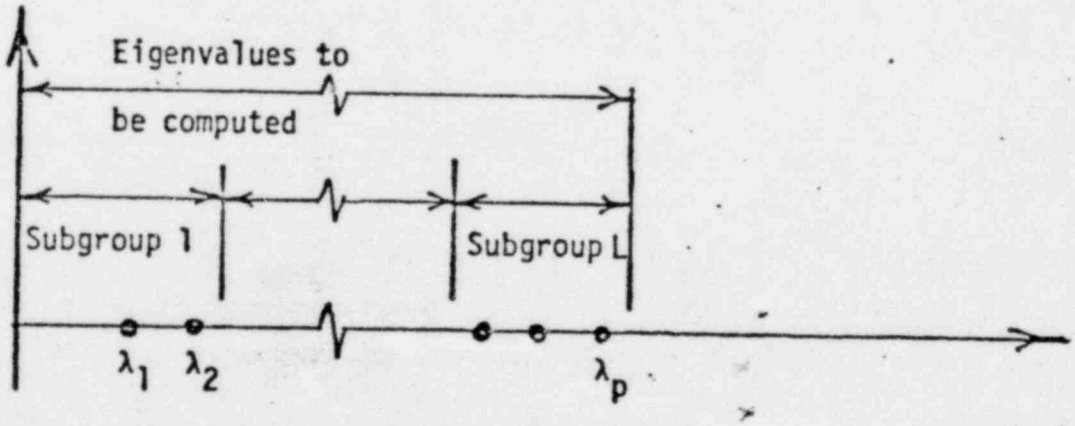


FIG. 2. SUBGROUPS FOR SUBSPACE

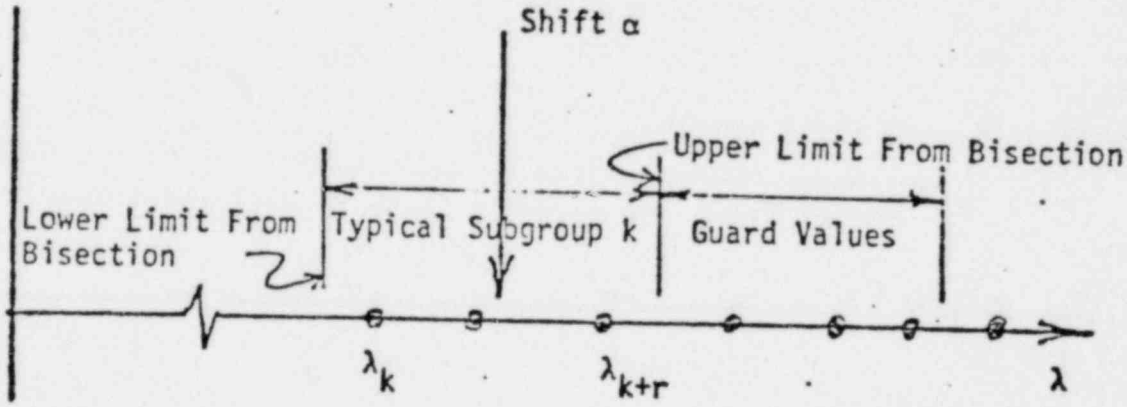
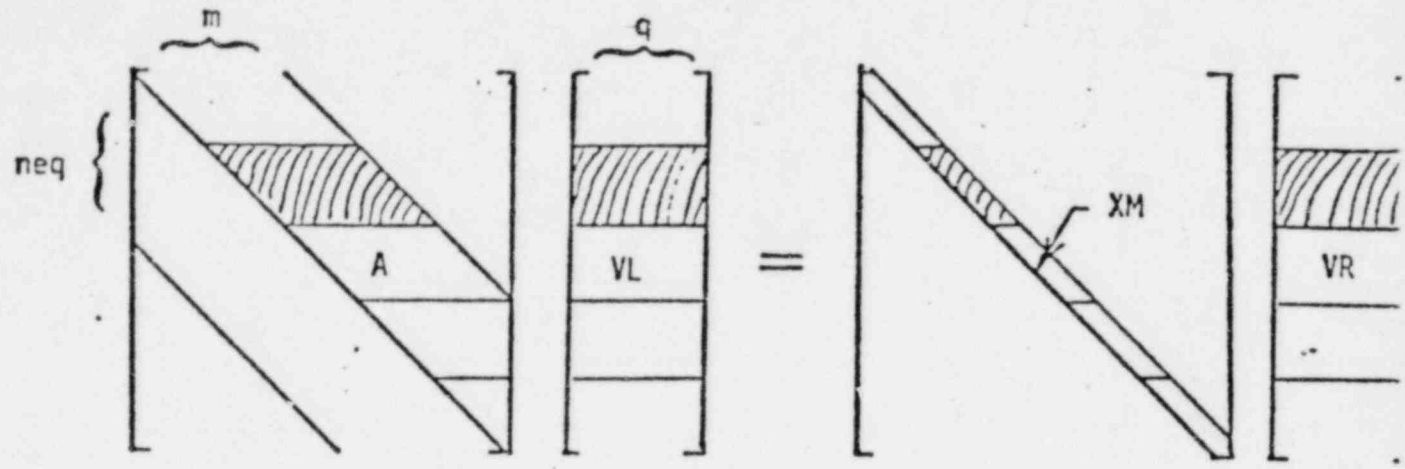
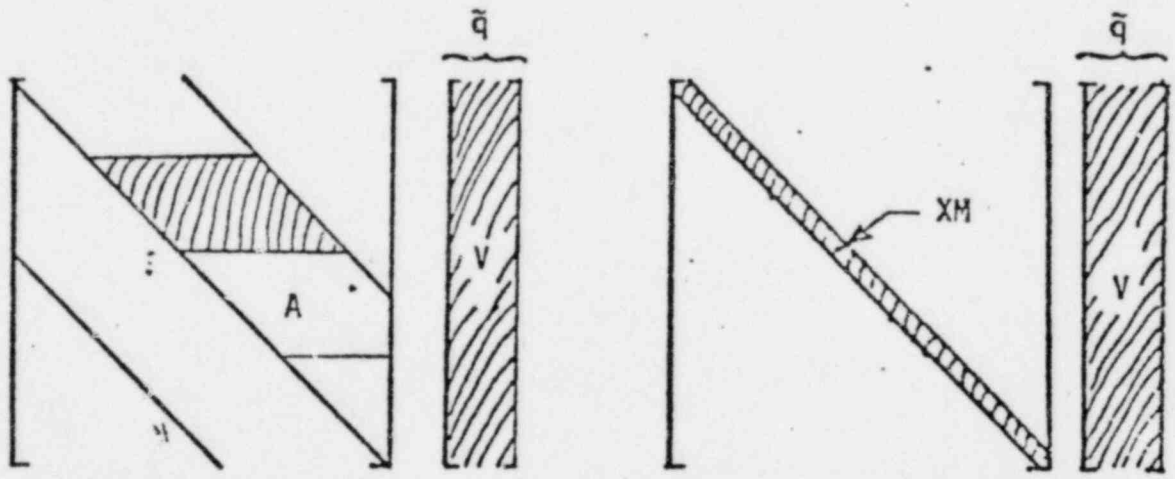


FIG. 3. TYPICAL SUBGROUP IN SUBSPACE

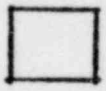


(a) Original "Blocking" Scheme



(b) Modified "Blocking" Scheme

— Information In-Core



— Information in Backing Store

FIG. 4. ARRAYS IN HIGH SPEED CORE FOR SUBSPACE COMPUTATIONS

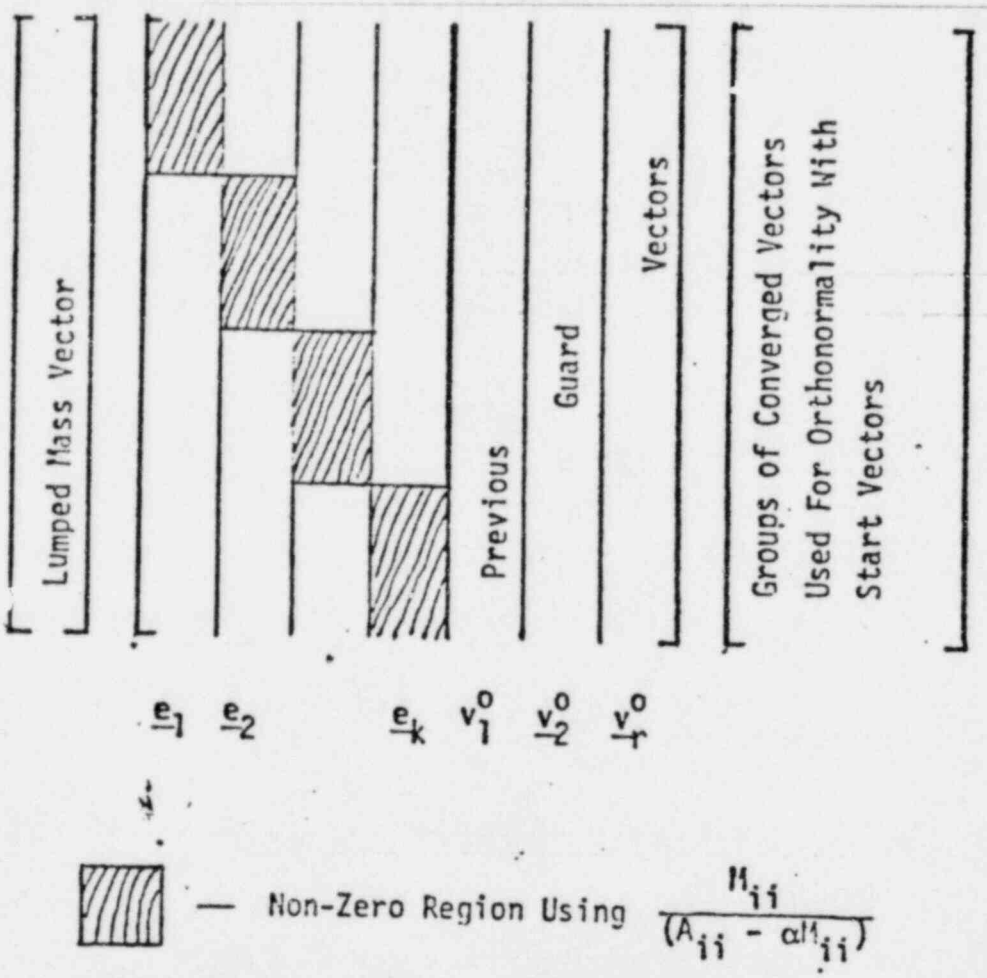


FIG. 5. START VECTORS FOR A SUBGROUP