RAL-TR-96-010

MA46, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations from finite-element applications

by
A. C. Damhaug* and J. K. Reid

Abstract

We describe the design of a new code for direct solution of sparse unsymmetric linear systems of equations from finite-element applications. The code accepts both the finite-element structure and the matrix coefficients in the form of finite elements. We show that the sparsity analysis using the knowledge about the finite-element structure is economic in time and space and that the matrix can be factored and the equations can be solved efficiently by a multifrontal technique.

Categories and subject descriptors: G.1.3 [**Numerical Linear Algebra**]: Linear systems (direct methods), sparse and very large systems.

General Terms: Algorithms, performance.

Additional Key Words and Phrases: Sparse unsymmetric matrices, the finite-element method, Gaussian elimination, multifrontal organization, BLAS.

Computing and Information Systems Department,
Rutherford Appleton Laboratory,
Chilton, Didcot,
Oxfordshire OX11 0QX.

January 30, 1996.

---

* Det Norske Veritas Research AS
Veritasveien 1,
N-1322 Høvik,
NORWAY.

CONTENTS

1 Introduction

We consider the direct solution of a set of linear equations

$$AX = B, \tag{1.1}$$

where the matrix A arises from a finite-element calculation and is large, sparse, and unsymmetric. We require the user to specify the number of finite-element nodes, and the number of variables at each of the nodes. This allows considerable storage and efficiency gains for any problem that has a significant number of nodes with more than one variable. The analysis is performed in terms of the super-variables formed by the sets of variables at the nodes. We will refer also to 'super-rows' and 'super-columns' for the corresponding sets of rows and columns.

The k-th finite element gives rise to a matrix

$$A^{(k)} \tag{1.2}$$

that is zero except in a small number of rows and columns. It may be represented by a list of indices of the nodes associated with the finite element and a square full matrix whose order is the total number of variables at the nodes. Because of nodal quantities, such as masses, springs and dampers, which are often used to modify the diagonal of the matrix A, we allow for an additional diagonal matrix A_d to give the overall form

$$A = \sum_{k=1}^{m} A^{(k)} + A_d. \tag{1.3}$$

Note that the structure is symmetric.

We use the multifrontal method (see, for example [6] and [7]), which is a variant of Gaussian elimination and produces the triangular factorization of a permutation of A.

Our approach is to assume initially that any diagonal entry is suitable as a pivot. This allows us to use exactly the same pivot choice strategies as for the symmetric and positive-definite case. We permit the user to supply a symmetric permutation in the form of an ordering for the nodes, but normally this is chosen automatically by MA46 from the structure of the matrix. In pivotal order, the matrix is

$$\bar{A} = P_s A P_s^T, \tag{1.4}$$

where P_s is the permutation matrix corresponding to the node order supplied by the user or generated by MA46. The order of the variables within any node is not changed by P_s.

Given such a permutation, and again working only with the matrix structure, we construct a tree that has a node for each finite-element node. The links of this tree are determined by the structure

of the super-rows when pivotal. If the off-diagonal entry of super-row *i* that is earliest in the pivot sequence is in super-column *j*, node *i* has node *j* as its parent. It is straightforward to show that any other off-diagonal entry of super-row *i* corresponds to an ancestor of node *i*. Following Liu [15], we refer to this as the elimination tree.

We will associate each finite element with the tree node corresponding to the finite-element node that is first in the pivot sequence among the nodes of the element. Because each element matrix is full, all the other nodes of the element must correspond to ancestors.

Suppose we consider the matrix obtained by summing the elements associated with a leaf node of the elimination tree. The super-row and super-column corresponding to the variables of the leaf node will be the same as for the matrix *A*, that is, they are fully-summed. We can take advantage of the fact that elimination steps

$$a_{ij} \leftarrow a_{ij} - a_{il} a_{ll}^{-1} a_{lj} \tag{1.5}$$

may be performed before all the assembly steps

$$a_{ij} \leftarrow a_{ij} + a_{ij}^{(k)} \tag{1.6}$$

are complete. It is necessary only that the terms in the triple product be fully-summed. We can therefore perform the elimination operations within a full temporary matrix (the frontal matrix) of order the number of variables associated with the nodes of the entries in the pivot super-row. The pivot rows and columns are stored away and the Schur complement is added into the frontal matrix. Once all the elimination operations of the node have been completed, we are left with a reduced matrix that has a status like that of an element matrix; we will refer to it as a generated element matrix, keep it in temporary storage, and associate it with the parent node.

There is considerable freedom in the ordering of the operations. What is required is that all the operations at the children of a node be completed before those at the node. To simplify the organization of temporary storage, we postorder the nodes following a depth-first search of the elimination tree. This allows a stack to be used to hold the generated elements awaiting assembly.

In the elimination tree, suppose there is a sequence of nodes $n_0, n_1, n_2, \ldots, n_k$ such that, for $i = 1, 2, \ldots, k$,

(a) node n_i has node n_{i-1} as its only child and

(b) the entries of the pivot super-row at node n_i are the off-diagonal entries of the pivot row at node n_{i-1},

then the corresponding rows and columns can be treated as blocks with no loss of sparsity. The elimination tree may be condensed by merging each such chain of nodes into a supernode, and we call the resulting tree the supernode elimination tree. Working with such blocks allows us to take

advantage of the additional efficiency associated with the use of full-matrix code and the BLAS (Basic Linear Algebra Subprograms) [4,5,12] during factorization.

If a supernode has one or more original elements as children, the code must somehow access real data supplied by the user. We have chosen to use reverse communication. The code must be called NB times, where NB is the number of supernodes with which original elements are associated. On each return the user is told which original element matrices are required. This avoids having to store all the original element matrices. Of course, the user may choose to store them all, but other alternatives may be convenient, such as generating them as required or holding them in a file.

For numerical stability, it is necessary to introduce further row and column interchanges into this process. It is usual for sparse unsymmetric matrices to require every pivot to satisfy the relative pivot tolerance

$$|a_{ll}| \geq u \cdot \max_{i>l} |a_{il}| \ , \tag{1.7}$$

where u is a fixed parameter, and this is what we do. The pivots must be chosen from the square submatrix that is fully-summed and we choose as many as possible, which may leave a few rows and columns uneliminated. Such rows and columns are simply passed to the parent as part of the generated element matrix. If we use the notation P and Q for the permutation matrices of the row and column interchanges introduced for stability, the final factorization is

$$LU = P\bar{A}Q = PP_sAP_s^TQ, \tag{1.8}$$

where L is lower triangular and U is upper triangular.

The subroutines are named according to the naming convention of the Harwell Subroutine Library [2]. We describe the single-precision versions which have names that commence with MA46 and have one more letter. The corresponding double-precision versions have the additional letter D. The code itself is available from AEA Technology, Harwell; the contact is Dr Scott Roberts or Mr Richard Lee, AEA Technology, Bldg 552, Harwell, Didcot, Oxon OX11 0RA, tel (44) 1235 434714 or (44) 1235 435690, Fax (44) 1235 434136, email: scott.roberts@aeat.co.uk or richard.lee@aeat.co.uk, who will provide details of price and conditions of use.

There are four subroutines that are called directly by the user:

Initialize. MA46I provides default values for the arrays CNTL and ICNTL that together control the execution of the package. For details, see appendix B.

Analyse. MA46A is called to analyse the sparsity pattern. If a pivot order is not provided by the user, the routine chooses one. It then prepares data structures for assembly and factorization, and computes the number of assembly steps NB.

Factorize. MA46B assembles and factorizes the matrix A based on the information computed by MA46A and chooses permutations P and Q for numerical stability. The routine must be called NB times in order to assemble and factorize the matrix. The routine may be called for several finite-element matrices A with the same sparsity pattern without the need for a new call to MA46A. This is common practice in non-linear finite-element packages, for instance when a Newton-Raphson iterative scheme is used.

Solve. MA46C uses the factorization produced by MA46B to solve the equation $AX=B$. Note that several right-hand side matrices B may be solved for the same matrix A without the need for a new sequence of calls to MA46B. This is common practice in linear finite-element packages when several load cases are analysed.

2 MA46A: analysis

This section describes the analyse subroutine MA46A. MA46A is logically divided into three parts: preparation, pivot order choice, and tree construction and tree analysis.

2.1 Preparation

We require the user to provide:

The number of finite elements, NELS.

The number of finite-element nodes, NNODS.

The number of variables, NEQNS.

An array IELT that holds the list of nodes for element 1, followed by the list of nodes for element 2, etc.

An array IPIELT of length NELS+1 that holds the position in IELT of the first node of element i, for i = 1, 2,..., NELS, and the first unused position in IELT.

An array IVAR of length NNODS that holds the number of variables at node i, for i = 1, 2, ... , NNODS.

Optionally, the first NNODS locations of an array KEEPA may be set to specify the pivot order. The node to be used in position i of the pivot order must be placed in KEEPA(i), i = 1, 2,..., NNODS.

The routine first checks the validity of the input data and exits with an appropriate error message if errors are found. Then the routine proceeds with three preparatory steps.

The first step is to compute the number of finite-element nodes that are active (have one or more variables) and the total number of variables. This provides a check on the value NEQNS provided by the user. If the total number of variables is not equal to NEQNS, the routine exits with an appropriate error message.

The second step is to order the active nodes ahead of the others. This permutation of the nodes is done regardless of whether or not a pivot order is provided. When it is provided, the relative order of the nodes with variables is retained. This permutation information is saved in KEEPA in a sub-

array denoted BSPERM. BSPERM is thus, in this stage of the analysis, the permutation from the initial order to the order provided by the user. BSPERM is needed in order to find the indices that the user associated with the finite-element nodes.

The third step is to compute a representation of the element-node connectivity information provided by the user in the array pair IPIELT, IELT. The representation consists of four sub-arrays that are tailored for efficient execution of the ordering step, if requested, and the subsequent tree construction and tree analysis. The first two arrays are denoted XELNOD, ELNOD and give a compressed version of IPIELT, IELT, that is, a compressed element-node connectivity structure. The compression is done in order to disregard the nodes that have no variables. The two last arrays are denoted XNODEL, NODEL and give the node-element connectivity and may be regarded as the inverse arrays to XELNOD, ELNOD. The four arrays are referred to as the implicit adjacency structure, or as the implicit graph structure, of the assembled coefficient matrix *A*. Note that the implicit adjacency structure represents the nodal structure of the coefficient matrix and not the variable structure.

2.2 Pivot order choice

A pivot order does not need to be chosen if it is provided by the user. In this case, the internal permutation arrays PERM and INVP are set to the identity. Otherwise, the routine MA46F is used to compute an ordering of the nodes that is stored in the arrays PERM and INVP by means of a minimum-degree type algorithm. The minimum-degree algorithm symbolically simulates the factorization of a sparse matrix. For each step in the algorithm, a node of minimum degree is chosen and eliminated. This symbolic elimination procedure is performed on some graph representation of the sparse matrix structure and creates a sequence of graphs, which are usually referred to as elimination graphs. MA46F uses a generalized element representation of this sequence of elimination graphs. The benefit is that the storage needed is no more than that needed for the original structure.

We have chosen to minimize the ‘external degree’, that is to choose each pivot supervariable to minimize the number of entries in the pivot row that lie outside the pivot block. This was introduced by Liu [13], who found that the number of entries in the factors was between 3% and 7% less than with ‘true minimum degree’ for his test problems. Amestoy, Davis and Duff report cases with bigger gains, including one with a reduction of over 50% in the number of entries in the factors [1].

The code implements a standard minimum-external-degree algorithm. That is, for each node of minimum external degree, the routine performs a graph elimination step and a degree update step. The routine is implemented to exploit indistinguishable nodes [13] (indistinguishable nodes are nodes that have the same list of connected elements) and uses incomplete degree update [10]. That is, the routine does a merge of nodes that have the same adjacency set in the current elimina-

tion graph and does not update the degree of nodes that are known not to be of minimum external degree after a degree update step. Incomplete degree update is often implemented as a search for outmatched nodes [13] (an outmatched node is a node whose list of connected elements includes all those connected to a neighbouring node). Since the routine uses an element representation, it is customary to formulate both the requirements in terms of generated elements. In most implementations of the minimum-degree algorithm, a simplified search for outmatched nodes is used. In this implementation, a complete search is used since the search procedure is very efficient in a generated element setting and often produces orderings of higher quality.

By default, the standard minimum-external-degree algorithm is extended with a multiple-elimination step as described by Liu [13]. Multiple-elimination allows more than one node of minimum external degree to be eliminated before the degree update step. The consequence is that more than one generated element may emerge in a multiple-elimination step. The nodes must be independent, that is no node may be involved in a new generated element other than its own.

An option is an extension of the multiple-elimination version of the minimum-external-degree algorithm to include independent nodes of degree exceeding the minimum by a user-specified amount. After all the nodes of minimum external degree have been eliminated the algorithm continues its search for independent nodes with external degree one higher, two higher and so on until the limit is reached, and eliminates these nodes together with the minimum external degree nodes in a multiple-elimination step.

2.3 Tree construction and tree analysis

When the preparation and pivot ordering steps have been completed, the routine continues with the tree construction and tree analysis, which consists of six steps. All the steps use the implicit adjacency structure that was computed in the preparation stage and the internal permutation arrays PERM and INVP. The steps are implemented separately with modularity in mind, which will make it easy to change parts of the code should new and better algorithms appear.

The first step is to compute the finite-element node-based elimination tree and the corresponding postordering. The work is done by the routine MA46G. At first glance, it might be assumed that the nodal elimination tree is not needed since the routine attempts to amalgamate nodes to make supernodes and the associated supernode elimination tree. This is true, but there is no great cost associated with the computation of the nodal elimination tree and the subsequent steps in the matrix analysis are more efficient if the structure is present. The method used to compute the elimination tree and its postordering is straightforward and is described in [15].

The second step is to compute the number of entries in the pivot rows at each node. The algorithm used is due to Gilbert, Ng and Peyton [11]. The algorithm is implemented in routine MA46H. For efficiency, it makes use of the postordered nodal elimination tree in addition to the implicit adja-

cency structure.

In step three, after the pivot row lengths are known, the nodes are grouped into supernodes by routine MA46J, as explained on page 2. Such supernodes were introduced by Duff and Reid [6] and called ‘fundamental supernodes’ by Ashcraft and Grimes [3].

The fourth step is the computation of an optimal postordering of the supernode elimination tree. We use a result of Liu [14]. Suppose the children of a node are $n[i]$, $i = 1, 2, ..., k$, that the size of the generated element at node $n[i]$ is $g[i]$, and that the temporary stack space needed when working on node $n[i]$ is $s[i]$. Liu showed that the total stack size needed for work on all the children is minimized if they are ordered so that $s[i]$-$g[i]$, $i = 1, 2, ... , k$ is a monotonic decreasing sequence. Such an ordering can therefore be determined, along with the total stack space needed, provided $s[i]$ and $g[i]$ are known for all the children. Any postorder following a depth-first search allows us to do this.

Pivoting due to numerical stability considerations may increase the size of the frontal and generated element matrices. This implies that the order found during the analysis stage need not be the best when the matrix is factorized. We have, however, not found it feasible to try any kind of sub-optimization during the factorization. Our numerical experiments indicate that the size of the working stack storage changes little and we believe that the order found is close to the best overall order that may be computed after the factorization has been completed.

The fifth step is to update the internal permutation vectors PERM and INVP and the representation of the supernodal elimination tree to correspond with the supernode postordering computed in step four. This step is done by routine MA46L.

The sixth and final step is to compute the number of assembly steps and some factorization statistics. The step also updates BSPERM by the information collected in the internal permutation arrays PERM and INVP during the previous matrix analysis steps. The work is done by routine MA46M. The number of assembly steps may be less than the number of supernodes found in step three, since there may be many nodes that have no original elements associated with them. We often see that the number of assembly steps is between 50-75% of the number of supernodes. Therefore, we have found it convenient to compute an assembly tree in order to reduce the number of calls to MA46B. The assembly tree consists of amalgamated supernodes and the start of a new node in the assembly tree is defined by the need for original finite-element coefficients. The routine checks each supernode to see if it needs coefficients from finite elements. If not, its right-most child in the supernode elimination tree is merged into it and thus an assembly tree is created. The procedure used to create the assembly tree assures that the postorder of the supernodal elimination tree is maintained as required by the stack management in routine MA46B.

3 MA46B: assembly and factorization

Before we start the description of MA46B, we give a skeleton of the basic multifrontal factorization algorithm in order to motivate the different choices we have made in the development and implementation of the code. The skeleton of the factorization is shown in Figure 3.1.

Note that the algorithm needs to process the supernodes in postorder for the internal stack management to work. In order to facilitate such an arrangement, we assume that a supernode elimination tree representation in postorder is available for the factorization algorithm.

We observe from Figure 3.1 that no distinction has been made between original finite-element matrices and generated element matrices and that an implementation of the algorithm will need data from the user for each supernode. That is, if we were to organize the factorization as is in Figure 3.1, MA46B would have to be called for each supernode, unless the coefficients were assembled into the factor submatrices in advance. This latter option would need pre-allocated storage for the triangular factors, which is not practical in a code where pivoting may alter the sizes. In addition, this arrangement would exclude the possibility of overlap between the stack and the triangular factors if we store the finalized triangular factors, the current frontal matrix, and the stack all in the same array. Such an overlap is implemented in MA46B in order to reduce the total storage needed.

In addition to the observation made in the previous paragraph, we use the following three observations to further refine the factorization algorithm:

1. Previously fully-summed variables that were not eliminated for stability reasons appear naturally in the leading part of the generated element matrices to be assembled into the current frontal matrix. These sets of variables must be disjoint since they arrive from disjoint subtrees in the supernode elimination tree, and thus they may be assembled directly into the index list in the symbolic assembly step.
2. Newly fully-summed variables are on entry to the factorization step still in the order arising from the analysis stage.
3. Row and column pivoting in the fully-summed part of the frontal matrix does not affect the order of the other rows and columns of the frontal matrix.

Figure 3.2 shows the final assembly and factorization algorithm. The implementation is based on calls to MA46B for each assembly step, and thus it is the part of the algorithm that starts with “For each supernode in the assembly step do” that is found inside MA46B. Note that the algorithm needs the supernodes in postorder for the internal management of the stack to work and that many details have been removed in order to make Figure 3.2 easy to read.

3.1 The symbolic assembly

The assembly starts with the computation of the indices of the variables that are active in the current step. This process is denoted symbolic assembly and is organized in two parts:

```
For each supernode in postorder do
    Merge the index lists of the children of the supernode to form the index list of the frontal matrix;
    Allocate space on top of the working stack for the frontal matrix of the supernode and initialize it to zero;
    For each child of the supernode do
        Pop the generated element matrix associated with the child off the stack and assemble it into the
        frontal matrix of the supernode;
    End For
    Perform the eliminations that are possible on the fully-summed rows and columns;
    Move the L/U submatrices associated with the performed eliminations to permanent storage for the factors;
    Move the generated element of the supernode to the top of the stack;
End For
```

Figure 3.1: Skeleton of the factorization algorithm.

```
For each assembly step do
    For each supernode in the assembly step do
        ! Symbolic assembly
        Merge the index lists of the children of the supernode to form the index list of the frontal matrix;
        ! Actual assembly
        Allocate space on top of the stack for the parent frontal matrix and initialize it to zero;
        For each generated element child do
            Assemble the generated element of the child into the parent frontal matrix;
        End For
        If this supernode is the first in the assembly step then do
            For each original finite element that participate in this assembly step do
                Assemble the original finite-element matrix into the parent frontal matrix;
            End For
        End If
        ! Elimination
        Perform elimination of the fully-summed rows and columns that may be eliminated;
        ! Management of the triangular factors and the stack
        Move the L/U submatrices computed in the elimination step to permanent storage for the
        submatrices;
        Compress the generated element matrix and move it to the new top of the stack;
    End For
End For
```

Figure 3.2: Final version of the assembly and factorization algorithm.

(i) the index lists of the children of the current supernode are merged together to form the index list of the parent, and

(ii) if the supernode is the first in an assembly step, index lists from original finite elements are merged with the parent list.

Since there may be a need to use off-diagonal pivots, we prepare by generating column indices for the fully-summed part, as well as row indices for the generated element. If at least one of the incoming generated elements has different row and column indices or an off-diagonal pivot is selected, we need to keep the column indices for the rows and columns of the factors that are associated with the supernode.

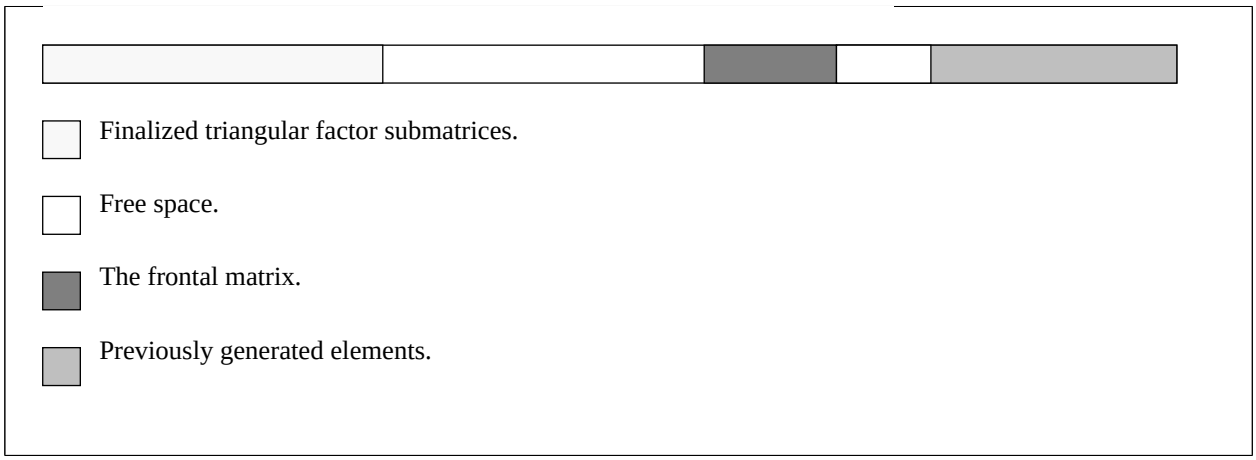


Figure 3.3: The structure of the workspace for triangular factors and the stack after assembly but before the eliminations have been carried out.

3.2 The numerical assembly

The active frontal matrix is stored on the top of the stack. Before the assembly process starts, workspace on top of the stack is allocated for the frontal matrix and it is initialized to zero. The frontal matrix is then assembled and finally the eliminations that are possible are performed on the matrix. Figure 3.3 shows the structure of the workspace after assembly but before the eliminations have been carried out.

As for the symbolic assembly step, the actual assembly step is divided into two separate parts:

(i) assembly of coefficients from generated element matrices, and

(ii) if the supernode is the first in an assembly step, assembly of coefficients from original finite-element matrices.

3.3 The actual factorization

For each assembly step, one or more supernode elimination steps are carried out. These block factorization steps involve eliminations of the fully-summed rows and columns of the frontal matrix. Each block factorization step is organized around the pivot search and subsequent submatrix update. The pivot search is done within the block of fully-summed rows and columns that are not yet eliminated. For better numerical stability, we choose the largest off-diagonal entry of the fully-summed part of the column even if the diagonal element satisfies the criterion for pivot choice. This in contrast to how the MA37 package from the Harwell Subroutine Library [2] selects the next pivot in such a case.

Initially, we planned to provide to the user two versions of the factorization:

(i) a version based on matrix-vector updates, i.e. Level 2 BLAS, and

(ii) a version based on matrix-matrix updates, i.e. Level 3 BLAS.

The final testing showed, however, that this was not necessary since the matrix-matrix version is the overall best.

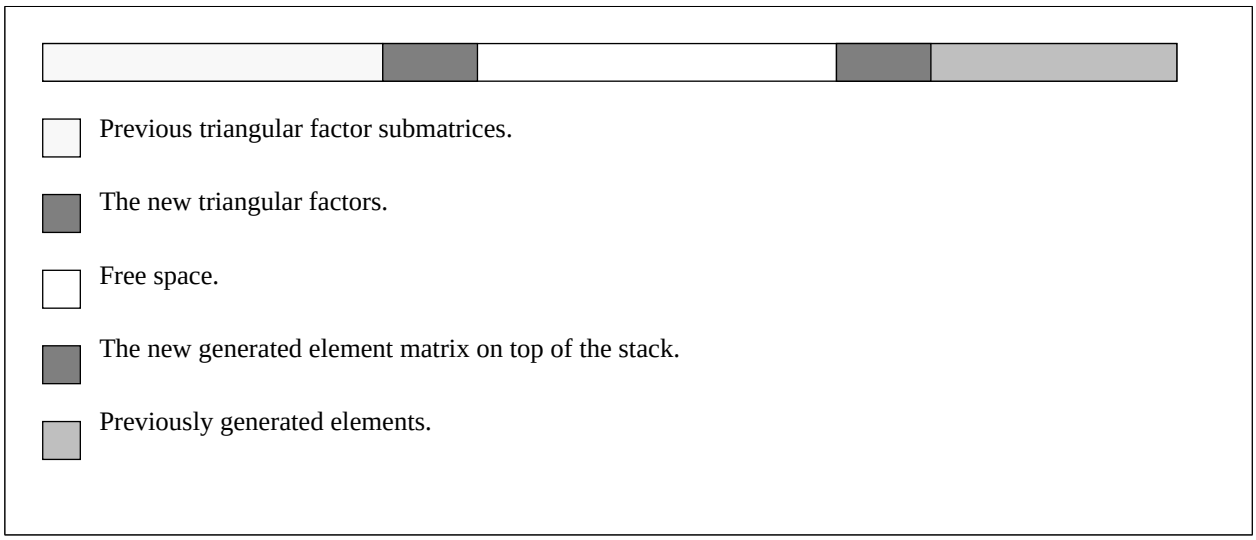


Figure 3.4: The structure of the workspace for triangular factors and stack after the eliminations and workspace compression.

In order to make good use of cache memory, we ask the user to provide its size in ICNTL(8) (see Appendix B), and divide the matrix-matrix update into blocks. Both versions use standard BLAS routines, that is, I_AMAX, _SCAL, _SWAP, _GER, _TRSM, and _GEMM in the most demanding sections of the elimination. Let the frontal matrix be *F* and have order *f*. Further, let it have *k* fully-summed rows and columns. We ensure that these are at the front of *F*.

The matrix-vector update kernel is implemented as follows: in elimination step *j*, where $1 \leq j \leq k$, a pivot is selected from *F*(*j:k, j:k*), and permuted to position *F*(*j, j*). The pivot search and the permutations are done by the Level 1 BLAS routines I_AMAX and _SWAP. The next step is to scale the column vector *F*(*j+1:f, j*) with the pivot using the Level 1 BLAS routine _SCAL and then the trailing submatrix *F*(*j+1:f, j+1:f*) is rank-one updated by Level 2 BLAS routine _GER.

As many such elimination steps as possible are performed and the resulting matrix is passed as the generated element matrix to the parent in the supernode elimination tree.

For the unblocked matrix-matrix update kernel, in elimination step *j*, a pivot is selected from the submatrix *F*(*j:k, j:k*), permuted to the *F*(*j, j*) position and the column *F*(*j+1:f, j*) is scaled by the pivot inverse, exactly as for the matrix-vector update kernel. Now the Level 2 BLAS routine _GER is restricted to a rank-one update of the submatrix *F*(*j+1:f, j+1:k*). After all possible such elimination steps are completed, say *l* steps, we perform forward solves on the submatrix *F*(*1:l, k+1:f*) using the Level 3 BLAS routine _TRSM, and finally update the submatrix *F*(*l+1:f, k+1:f*) using the Level 3 BLAS routine _GEMM. The resulting matrix *F*(*l+1:f, l+1:f*) is passed as the generated element matrix to the parent in the supernode elimination tree.

We introduced the blocked version of the factorization algorithm in order to increase the amount of Level 3 BLAS work and avoid swapping between cache and main memory. The block size *kb*

is chosen so that $(kb+1)f \leq \mathrm{ICNTL}(8)$, to ensure that a block column fits in the cache. We commence processing as if only the first *kb* rows and columns were fully-summed. Once this is complete, we regard the next *kb* rows and columns as fully-summed and process these together with any we were unable to pivot upon while processing the first block. We continue in this way with one more block of *kb* rows and columns at a time. The treatment is essentially the same as we would have obtained had we limited the merges into supervariables to blocks of size *kb*.

After a factorization step has been completed, the newly computed submatrices of the triangular factors are moved from the frontal matrix to permanent storage and the generated element matrix is moved to the top of the stack. During its movement, the generated element matrix is compressed in order to eliminate the unused space above the columns. Figure 3.4 shows the structure of the workspace after the eliminations and workspace compression have been carried out.

4 MA46C: solve

In contrast to MA46A and MA46B, there is no use of finite-element input to the solution routine MA46C. The reason is that the solve step, and thus MA46C, then has a cleaner interface. The organization of the solve routine is, however, based on the supernode elimination tree which means that the equations are solved as a sequence of submatrix solves. The forward elimination is driven by traversal of the supernode elimination tree in a postorder and the backward elimination is driven by traversal of the supernode elimination tree in the inverse order. MA46C may be called more than once for the same triangular factors. On each call, the number of right-hand sides (columns of B) must be specified in NRHS. Both the forward and backward elimination steps use matrix-matrix computational kernels, i.e. Level 3 BLAS, when NRHS>1. It should be noted that B is involved in this matrix-matrix product and that the kernels become more efficient with an increased number of columns in B. When there is one right-hand side, i.e. NRHS=1, Level 2 BLAS routines are used since this gives an improvement over the Level 3 BLAS because there is less administration overhead.

5 Performance results

For performance testing, we have taken a subset of the problems in the Harwell-Boeing collection, see [8], and some problems collected from structural engineering applications at Det Norske Veritas Research AS. Table 5.1 shows a summary of the problems and those from the Harwell-Boeing collection are marked with superscript a. The storage format of the problems from the Harwell-Boeing collection does not take advantage of the fact that there may be more than one variable at a finite-element node. This means that all nodes in these problems have at most one variable and that the matrix analysis of MA46 may perform worse than usual for finite-element problems. There are, however, many performance results available for these matrices, see for

Table 5.1: The test problems

Problem	Description	Number of variables	Number of elements
1	3D model of a container ship.	10,110	3,431
2	CEGB2802[a].	2,694	108
3	CEGB2919[a].	2,859	128
4	CEGB3024[a].	2,996	551
5	CEGB3306[a].	3,222	791
6	3D model of a corrugated plate field.	18,010	3,152
7	3D model of a flywheel.	4,368	248
8	LOCK1074[a].	1,038	323
9	LOCK2232[a].	2,208	944
10	LOCK3491[a].	3,416	684
11	LOCK 700[a].	691	324
12	MAN5976[a].	5,882	784
13	3D model of part of a condeep cylinder.	15,449	977
14	3D model of a sandwich beam[b].	2,508	3,429

a. See [8] for a description of the problems taken from the Harwell-Boeing collection.
b. The 3D model of the beam consists of randomly distributed 2-noded beam elements.

instance [9], and this choice of problems makes it easier to compare the performance of MA46 with other packages. On the other hand, the problems from Det Norske Veritas Research AS are presented to MA46 as we expect the code to be implemented in finite-element packages; they are extracted directly from finite-element applications and most of the nodes have more than one variable.

The matrices of Table 5.1 have been used to test the code on

1. a DEC 3000-400 with operating system OSF/1 V2.0 and using release V3.4-480 of the f77 compiler with the options -O5 and -fast. There are vendor versions of BLAS available on this platform.
2. a SUN 4 with operating system Sun OS release 4.1.2 and using release SC 1.0 Fortran V1.4 of the f77 compiler with options -O3 -cg89 -dalign. There are no vendor version of BLAS available on this platform.
3. one processor of a Cray Y-MPI/8-128 using release 6.0 (6.52) of the CF77 compiling system with default options. There are vendor versions of BLAS available on this platform.

The double precision version of MA46 was used on the DEC 3000-400 and SUN 4 and the single precision version was used on the Cray Y-MP. The pseudo-random number generator FA04 from the Harwell Subroutine Library [2] was used to generate values for the element coefficient matrices and the right-hand sides. Each problem was run enough times for each combination of options to take at least one second and the average CPU times in seconds are reported.

Table 5.2: Results from MA46A using default value 0 for ICNTL(5).

Problem	Number of variables	Number of assemblies	Number of supernodes	Size of factorization (thousands)		Stack size (thousands)	
				integers	reals	with reordering	without reordering
1	10,110	791	828	98	2,196	756	2,059
2	2,694	86	163	15	267	80	177
3	2,859	98	192	19	361	204	235
4	2,996	398	657	18	111	19	39
5	3,222	382	510	11	69	23	28
6	18,010	929	1,653	136	2,825	991	1,545
7	4,368	152	271	40	1,167	793	971
8	1,038	101	112	6	61	30	47
9	2,208	293	293	10	73	10	42
10	3,416	274	401	22	223	87	110
11	691	117	126	4	26	31	36
12	5,882	499	956	44	489	197	242
13	15,449	564	1,051	150	5,522	3,763	5,180
14	2,508	811	811	15	64	12	34

Table 5.3: Results from MA46A, ICNTL(5)=-1 and ICNTL(5)=4.

	ICNTL(5)=-1				ICNTL(5)=4			
Problem	Number of assemblies	Number of supernodes	Size of factorization (thousands)		Number of assemblies	Number of supernodes	Size of factorization (thousands)	
			integers	reals			integers	reals
1	797	836	98	2,157	783	829	98	2,167
2	86	163	14	236	86	165	14	248
3	101	192	19	341	100	192	19	357
4	385	656	19	113	394	657	19	117
5	514	514	11	68	382	510	11	69
6	942	1,673	137	2,783	903	1,634	134	2,732
7	152	271	39	1,093	152	271	40	1,167
8	103	113	6	61	103	112	6	61
9	293	293	10	73	292	293	10	74
10	274	400	22	223	273	399	22	224
11	117	127	4	25	114	126	4	26
12	499	956	44	493	499	956	44	510
13	562	1,051	150	5,420	566	1,051	150	5,476
14	810	810	15	64	794	794	16	69

5.1 Matrix analysis options

Our first experiments concern the matrix analysis options of MA46A. We have run the test problems for each of the options:

(1) standard minimum external degree, ICNTL(5)<0,

(2) multiple minimum external degree, ICNTL(5)=0, the default, and

(3) relaxed multiple minimum external degree, ICNTL(5)>0,

Table 5.4: Timing results from MA46A, CPU-seconds on DEC 3000-400.

Problem	ICNTL(5)=-1	ICNTL(5)=0	ICNTL(5)=4
1	0.39	0.37	0.36
2	0.23	0.23	0.23
3	0.29	0.29	0.29
4	0.15	0.14	0.14
5	0.12	0.12	0.12
6	0.19	0.17	0.16
7	0.13	0.12	0.12
8	0.09	0.08	0.09
9	0.13	0.13	0.13
10	0.23	0.23	0.23
11	0.06	0.05	0.05
12	0.32	0.32	0.32
13	0.51	0.47	0.47
14	0.19	0.18	0.19

to see the effect on the number of block factorization steps and assembly steps; the size of the index information, the triangular factors, and the working stack; and on the CPU time. Table 5.2 shows the results from MA46A for the default value of ICNTL(5).

We see from Table 5.2 that the number of supernodes is often below 10% of the total number of variables. High percentages are often found for problems that are mainly assembled from bar and beam elements, such as the problems 5, 8, 9, 10, 11 and 14.

The number of assembly steps is for most problems between 50-75% of the number of super-nodes. Finite-element problems for which the value is higher usually have a high element-to-node ratio. The most obvious example is problem 14 where there are more elements than nodes in the finite-element mesh. The problems 1, 8, 9 and 11 all have a relatively high element-to-node ratio and this is reflected in the number of assembly steps being greater than 75% of the number of supernodes.

For all the problems, we find that the number of indices stored to represent the triangular factors is small compared with the number of reals for the triangular factors themselves.

In the final two columns of Table 5.2, we show the stack sizes with and without reordering of the children of each node. We found that the additional CPU time for this reordering is negligible (not measurable within the uncertainty of the timer). It is clear that this reordering is very worthwhile.

Table 5.3 shows the results from MA46A with the options ICNTL(5)=-1, 4. For almost all the problems, the number of assembly steps, the number of supernodes and the number of indices stored do not differ much from the values shown in Table 5.2.

We conclude that the options for different forms of the minimum-degree algorithm do not have a big effect on these quantities. None of the algorithms is consistently better than the other two.

Table 5.4 shows the CPU-time consumptions in MA46A for the three forms of the minimum-

Table 5.5: The effect of block size on factorization time. Results from DEC 3000-400.

	Level 2 BLAS ICNTL(8)=-1		Level 3 BLAS ICNTL(8)=0		Level 3 BLAS ICNTL(8)=32		Level 3 BLAS ICNTL(8)=64	
Problem	DXML BLAS	F77 BLAS	DXML BLAS	F77 BLAS	DXML BLAS	F77 BLAS	DXML BLAS	F77 BLAS
1	98.24	95.21	42.39	54.97	42.63	46.60	**34.07**	42.94
2	3.11	2.67	1.96	2.14	**1.86**	2.04	1.94	2.07
3	6.68	6.22	3.47	3.82	3.32	3.75	**3.29**	3.73
4	0.66	0.60	0.59	0.55	0.59	**0.54**	0.59	**0.54**
5	0.34	0.31	0.33	**0.29**	0.33	**0.29**	0.33	0.30
6	118.90	115.42	46.14	62.84	55.98	57.20	**41.09**	50.24
7	59.92	58.48	23.06	30.10	27.35	28.08	**20.53**	25.65
8	0.49	0.43	0.37	**0.35**	0.36	**0.35**	0.37	**0.35**
9	0.39	0.35	0.36	**0.32**	0.36	**0.32**	0.36	**0.32**
10	2.11	1.87	1.47	1.52	**1.42**	1.49	1.44	1.49
11	0.18	0.16	0.17	0.15	0.16	**0.14**	0.17	0.15
12	6.87	6.16	4.20	4.58	3.90	4.37	**3.82**	4.35
13	540.87	539.02	**196.65**	341.74	480.86	413.78	214.63	228.92
14	0.35	**0.32**	0.38	**0.32**	0.38	**0.32**	0.38	**0.32**

Table 5.6: Solution times. Results from DEC 3000-400.

Problem	Number of right-hand sides									
	NRHS=1				NRHS=3		NRHS=10		NRHS=50	
	DXML BLAS		F77 BLAS		DXML BLAS	F77 BLAS	DXML BLAS	F77 BLAS	DXML BLAS	F77 BLAS
	Level 2	Level 3	Level 2	Level 3						
1	0.69	1.13	**0.62**	0.65	1.62	**1.29**	**2.91**	3.58	**10.64**	17.08
2	0.09	0.15	**0.08**	0.09	0.21	**0.16**	**0.38**	0.42	**1.35**	1.93
3	0.12	0.20	**0.11**	**0.11**	0.28	**0.21**	**0.50**	0.55	**1.82**	2.59
4	0.07	0.11	**0.06**	0.07	0.17	**0.12**	0.33	**0.30**	**1.22**	1.43
5	**0.05**	0.07	**0.05**	**0.05**	0.12	**0.09**	0.22	**0.21**	**0.83**	1.02
6	0.92	1.52	**0.83**	0.85	2.14	**1.68**	**3.81**	4.55	**13.65**	21.54
7	0.35	0.67	**0.32**	**0.32**	0.91	**0.65**	**1.53**	1.79	**5.20**	8.61
8	0.03	0.04	**0.02**	0.03	0.06	**0.05**	**0.12**	**0.12**	**0.42**	0.57
9	**0.04**	0.06	**0.04**	**0.04**	0.10	**0.07**	**0.18**	**0.18**	**0.65**	0.83
10	0.10	0.15	**0.09**	**0.09**	0.22	**0.17**	**0.41**	0.42	**1.51**	1.97
11	**0.01**	0.02	**0.01**	**0.01**	0.04	**0.03**	**0.07**	**0.07**	**0.25**	0.33
12	0.20	0.33	**0.18**	0.19	0.48	**0.36**	**0.88**	0.92	**3.27**	4.37
13	1.62	2.80	**1.46**	1.49	3.79	**3.38**	**6.36**	10.07	**22.64**	49.30
14	0.06	0.09	**0.05**	0.06	0.15	**0.10**	0.29	**0.26**	**1.08**	1.22

external-degree algorithm. It may be seen that the differences are not great. We have therefore decided not to offer the ICNTL(5) < 0 option to users. The option ICNTL(5)>0 is offered to the users since there are some problems, often very big problems modelled with solid finite-elements, where it is better than the default option.

5.2 Block size and factorization

In this section, we report the effect on the CPU time used to factorize the matrices of Table 5.1 of using Level 2 BLAS and Level 3 BLAS with varying block sizes. The options are controlled by ICNTL(8) and we have limited our trials to the values:

-1	Level 2 BLAS,
0	Level 3 BLAS without blocking,
32	Level 3 BLAS with block columns of size less than 32 kbytes, and
64	Level 3 BLAS with block columns of size less than 64 kbytes.

We also consider the effect of using vendor-supplied BLAS. The most important BLAS routines for MA46B and MA46C are _TRSM and _GEMM. In the Fortran 77 versions, we have made the following modifications in order to improve their performance: _TRSM has been modified to use level-two unrolling in its inner loop:

```
      DO 82, I = K + 1, M
         B(I,J) = B(I,J) - TEMP1*A(I,K)
         B(I,J+1) = B(I,J+1) - TEMP2*A(I,K)
82    CONTINUE
```

and _GEMM has been modified to use level-eight unrolling in its inner loop:

```
      DO 78, I = 1, M
         C(I,J) = C(I,J) + TEMP1*A(I,L) + TEMP2*A(I,L+1)
     &             + TEMP3*A(I,L+2) + TEMP4*A(I,L+3)
     &             + TEMP5*A(I,L+4) + TEMP6*A(I,L+5)
     &             + TEMP7*A(I,L+6) + TEMP8*A(I,L+7)
78    CONTINUE
```

The choice of unrolling level is dependent of the computer architecture. For MA46, the two above code segments give the best performance on DEC 3000-400 and SUN 4.

The results from runs on the DEC 3000-400 are summarized in Tables 5.5 and 5.6. Here and in Tables 5.7, 5.8 and 5.9, we show the best result for each problem in bold. The columns labelled DXML BLAS show the results when the vendor-supplied versions of the BLAS routines are used and the other columns show the results of using our modified Fortran 77 source. On this machine, the size of the local cache memory is 64 kbytes and we therefore expect the option ICNTL(8)=64 to perform best. Using the Fortran 77 BLAS, this option is the best in 11 cases and is very near the best (within 8%) in all the others.

On the smaller problems, we must expect that all the fully assembled columns will often fit into the cache anyway, so blocking will have little effect. In fact, with no blocking, some overheads in the factorization routine are avoided, but the effect on timing is not great. These remarks may be verified in the table.

It may also be seen in Table 5.5 that the vendor-supplied Level 3 BLAS outperform the Fortran Level 3 BLAS for most of the bigger problems. For the smaller problems our modified Fortran 77

Table 5.7: The effect of block size on factorization time, and average solution times for NRHS=3. Results from SUN 4.

	Factorization time				Solution time	
	Level 2 BLAS	Level 3 BLAS	Level 3 BLAS	Level 3 BLAS	Level 2 BLAS	Level 3 BLAS
Problem	ICNTL(8)=-1	ICNTL(8)=0	ICNTL(8)=32	ICNTL(8)=64		
1	404.73	331.34	**239.36**	241.49	2.40	**2.23**
2	18.54	13.00	**12.24**	12.41	0.29	**0.25**
3	33.89	24.76	**21.36**	21.61	0.40	**0.34**
4	3.41	3.09	3.17	**3.07**	0.21	**0.17**
5	1.79	**1.58**	**1.58**	1.60	0.14	**0.12**
6	490.75	394.59	**297.22**	299.56	3.29	**2.97**
7	238.70	194.06	**142.15**	143.75	1.13	**1.11**
8	2.60	2.14	**2.13**	2.14	0.08	**0.07**
9	2.02	**1.77**	1.81	1.79	0.13	**0.10**
10	11.70	9.04	**8.61**	8.62	0.30	**0.26**
11	1.04	0.91	0.89	**0.88**	0.05	**0.04**
12	40.54	30.12	**25.68**	25.80	0.65	**0.56**
13	1951.39	1607.49	1589.05	**1149.34**	15.20	**9.31**
14	1.84	1.82	1.77	**1.74**	0.16	**0.14**

Table 5.8: The effect of block size on factorization time. Results from Cray Y-MP.

	Level 2 BLAS ICNTL(8)=-1		Level 3 BLAS ICNTL(8)=0		Level 3 BLAS ICNTL(8)=32		Level 3 BLAS ICNTL(8)=64	
Problem	Cray BLAS	F77 BLAS	Cray BLAS	F77 BLAS	Cray BLAS	F77 BLAS	Cray BLAS	F77 BLAS
1	6.35	10.19	**6.33**	8.59	7.71	8.90	6.82	8.44
2	**0.48**	0.72	0.49	0.70	0.49	0.71	0.49	0.71
3	**0.74**	1.13	**0.74**	1.04	0.77	1.06	0.75	1.04
4	**0.23**	0.32	0.25	0.37	0.25	0.37	0.25	0.37
5	**0.17**	0.23	0.18	0.27	0.18	0.27	0.18	0.27
6	7.92	12.52	**7.60**	10.67	9.86	11.13	8.33	10.63
7	3.76	5.75	**3.54**	4.79	4.56	4.98	3.91	4.78
8	**0.12**	0.17	**0.12**	0.18	**0.12**	0.18	**0.12**	0.18
9	**0.16**	0.22	0.17	0.25	0.17	0.25	0.17	0.25
10	**0.42**	0.61	0.44	0.65	0.45	0.66	0.44	0.65
11	**0.06**	0.08	0.07	0.10	0.07	0.10	0.07	0.10
12	**0.98**	1.48	1.00	1.46	1.04	1.49	1.01	1.46
13	28.35	42.84	**26.62**	34.53	61.05	43.55	36.06	34.73
14	**0.17**	0.23	0.19	0.28	0.20	0.28	0.20	0.28

source is competitive which seems to indicate that the vendor was concentrating on performance for bigger cases.

From Table 5.5, we see that the results with Level 2 BLAS were almost always worse and often significantly worse than those with Level 3 BLAS. Except for problem 14 where the Level 2 BLAS are as fast as Level 3 BLAS, for no case was the best time obtained with Level 2 BLAS.

In order to study the effect of vendor-supplied Level 3 BLAS more carefully, we have tabulated

Table 5.9: Average solution times for NRHS=3. Results from Cray Y-MP.

	Level 2 BLAS		Level 3 BLAS	
Problem	Cray BLAS	F77 BLAS	Cray BLAS	F77 BLAS
1	0.069	0.133	**0.060**	0.099
2	0.014	0.026	**0.011**	0.017
3	0.016	0.031	**0.013**	0.021
4	0.025	0.059	**0.017**	0.031
5	0.020	0.047	**0.013**	0.025
6	0.106	0.219	**0.092**	0.153
7	0.031	0.054	**0.028**	0.043
8	0.006	0.013	**0.005**	0.008
9	0.013	0.030	**0.009**	0.016
10	0.021	0.045	**0.016**	0.027
11	0.005	0.012	**0.004**	0.006
12	0.045	0.099	**0.033**	0.058
13	0.130	0.224	**0.118**	0.179
14	0.026	0.065	**0.017**	0.033

the solution times for different number of right-hand sides in Table 5.6. It can be seen that our Fortran BLAS are faster than DXML BLAS when there are few right-hand sides and that the opposite is true when there are many right-hand sides.

Results for the SUN 4 are shown in Table 5.7. Here we find that ICNTL(8)=64 is best or near best in all cases. Vendor supplied BLAS were not available on this machine and only Fortran 77 BLAS were used.

Results for the Cray Y-MP are shown in Tables 5.8 and 5.9. This computer does not use cache memory and we therefore do not expect blocking to be helpful. In Table 5.8, we can see that this indeed is the case. The vendor versions of BLAS are usually better here. An exception is for problem 13 where our modified Fortran 77 source is better than the vendor versions for ICNTL(8)=32, 64, but the best performance is obtained for the unblocked Level 3 BLAS when vendor supplied BLAS are used. For the Cray, we also see that Level 2 BLAS are competitive with Level 3 BLAS. Level 2 BLAS are best for the smaller problems and Level 3 BLAS are best for the bigger problems, but with small margins in both cases. Table 5.9 shows a comparison between Level 2 and Level 3 BLAS for the solution step. NRHS was set to 3 and we have reported the average solution times. We can see that vendor supplied Level 3 BLAS are consistently the best on all the problems.

We conclude that the block strategy of MA46B works as expected on the problems of Table 5.1 on all the three platforms we considered. We have decided not to offer the Level 2 BLAS option to users because its performance is inferior to the Level 3 BLAS option except for small problems on the Cray, where the difference is slight.

The most important BLAS routines in MA46B are _TRSM and _GEMM and we recommend that vendor supplied versions of at least these two should be used when available.

Table 5.10: Comparison between MA37 and MA46. CPU-time consumptions for MA37 divided by the best corresponding results for MA46.

	DEC 3000-400			SUN 4			Cray Y-MP		
Problem	Analyse	Factorize	Solve	Analyse	Factorize	Solve	Analyse	Factorize	Solve
1	13.73	3.67	1.53	11.49	2.02	1.19	6.99	3.18	1.32
2	4.37	2.57	1.44	3.63	1.79	1.05	2.07	4.11	1.37
3	4.34	2.90	1.59	3.62	2.06	1.10	2.08	3.93	1.35
4	2.29	2.71	1.16	1.85	2.03	0.92	1.47	3.25	0.91
5	2.28	2.22	0.94	1.98	1.72	0.83	1.42	3.18	1.01
6	29.82	10.07	2.18	21.61	4.85	3.57	16.17	5.73	1.56
7	21.74	4.81	1.79	18.29	2.46	1.18	10.95	3.59	1.52
8	3.22	2.72	1.36	2.92	1.92	1.04	1.70	4.53	1.13
9	2.48	2.48	1.06	2.12	1.83	0.92	1.46	3.71	1.02
10	3.33	2.73	1.41	2.88	1.99	1.06	1.73	4.02	1.27
11	2.26	2.41	0.80	2.19	1.78	0.82	1.36	4.03	0.99
12	2.93	4.77	1.65	2.35	3.00	1.20	1.64	3.81	1.11
13	19.09	3.71	1.60	15.84	2.16	1.98	9.67	2.77	1.46
14	0.87	1.61	0.79	0.86	1.24	0.68	0.66	2.19	0.79

Table 5.11: Comparison between MA37 and MA46 on the number of indices anticipated in analyse and stored in factorize for each problem. Also compared is the length of the array for the triangular factors and stack as anticipated in analyse and needed in factorize. The results are obtained on DEC 3000-400. Default options are used for both packages.

	ANALYSE				FACTORIZE			
	Size of factorization (thousands)				Size of factorization (thousands)			
	integers		reals, including stack		integers		reals, including stack	
Problem	MA46	MA37	MA46	MA37	MA46	MA37	MA46	MA37
1	98	201	4,820	4,848	108	201	4,854	4,839
2	15	29	565	563	18	29	570	566
3	19	39	851	903	22	39	858	907
4	19	40	231	279	21	40	235	281
5	11	24	149	154	15	24	152	156
6	136	296	6,156	10,781	154	296	6,211	10,445
7	40	82	2,719	3,395	44	82	2,738	3,355
8	6	13	135	153	7	13	136	154
9	10	21	149	198	13	21	153	199
10	22	47	480	541	26	47	485	542
11	4	8	68	83	5	8	69	84
12	44	94	1,100	1,465	50	94	1,118	1,451
13	150	307	12,476	14,815	166	307	12,545	14,713
14	16	32	133	133	18	33	135	135

5.3 Comparison with MA37

To evaluate the efficiency of the new code we have compared it to the MA37 package [7] from the Harwell Subroutine Library [2].

The main differences between MA37 and MA46 are that MA37:

(a) creates an explicit graph structure based on variables and an assembled coefficient matrix,
(b) makes no use of Level 2 and Level 3 BLAS in the computationally expensive parts of the code,
(c) always holds both row and columns indices of frontal matrices, and
(d) does not reorder siblings in the tree to reduce the size of the stack.

Our numerical experiments were run with default options for both codes. We have always compared MA37 to the best result for MA46, regardless if this result was obtained with vendor versions of the BLAS or with our modified Fortran 77 BLAS. This applies for DEC 3000-400 and Cray Y-MP. MA37 does workspace compressions in the ANALYSE and FACTORIZE steps only when this is necessary and in the tests we made sure that the integer and real work arrays were great enough to avoid this. We did not make provisions to avoid workspace compressions in the ANALYSE step of MA46, and the FACTORIZE step of MA46 always does a compression of the workspace after a block factorization step is finished.

Table 5.10 shows the CPU-time consumptions on all three computers for MA37 in ANALYSE, FACTORIZE and SOLVE divided by the corresponding results obtained by MA46 and shown in Tables 5.5, 5.6, 5.7, 5.8 and 5.9 respectively.

We see that MA46 is faster than MA37 in all the three steps for most of the problems considered on all three computers. In ANALYSE, only problem 14 is performed faster by MA37 than MA46. The reason is that this is a problem with many elements, which makes the ANALYSE step of MA46 expensive. In FACTORIZE, MA46 performs better than MA37 for all the problems.

Table 5.11 shows a comparison between MA46 and MA37 on the size of the integer array needed for the triangular factors, and the length of the work array needed in FACTORIZE. We show both the sizes anticipated by ANALYSE on the assumption of no interchanges and the actual sizes. Almost all the results favour MA46. The ordering routine of MA37 is a standard minimum degree ordering that is comparable with the one obtained for ICNTL(5)=-1 in MA46 except that it uses true degree instead of external degree. The results from the analysis obtained by MA37 on the problems reflects this since they are like the results shown in Table 5.3. The difference in the length of the work array for the two routines must therefore be attributed to the requirement for the stack that is needed in MA37B. This is consistent with the results shown in Table 5.2 where we saw that the stack was much larger without reordering of siblings.

We have recorded the numerical errors in the resulting solution and due to the fact that MA46

Table 5.12: Comparison between MA37 and MA46 on the relative numerical errors. The results are obtained on DEC 3000-400. Default options are used for both packages.

	right-hand side 1 $B(i,1)=i$		right-hand side 2 $B(i,2)=1$		right-hand side 3 $B(i,3)=[-1,1]$	
Problem	MA46	MA37	MA46	MA37	MA46	MA37
1	4.3E-14	2.4E-13	3.9E-14	2.2E-13	4.7E-14	2.8E-13
2	1.3E-14	4.8E-14	1.2E-14	4.6E-14	2.1E-14	7.6E-14
3	3.1E-14	7.4E-14	2.4E-14	1.3E-13	3.9E-14	1.2E-13
4	7.8E-15	5.9E-14	2.6E-14	5.6E-14	7.1E-15	4.4E-14
5	6.5E-15	1.2E-14	9.7E-15	2.3E-14	8.5E-15	3.7E-14
6	2.5E-13	3.6E-13	7.6E-14	2.6E-12	4.8E-13	2.8E-12
7	4.5E-14	3.3E-13	7.3E-14	3.7E-13	6.3E-14	2.4E-13
8	6.4E-15	3.9E-14	9.7E-15	6.7E-14	8.0E-15	1.9E-14
9	3.0E-14	3.5E-14	9.4E-15	4.1E-14	6.0E-15	1.9E-14
10	1.4E-14	1.2E-13	1.3E-14	3.9E-14	2.7E-14	4.4E-14
11	1.6E-14	3.1E-14	2.5E-14	1.9E-14	1.0E-14	2.4E-14
12	1.8E-14	1.1E-13	2.9E-14	1.4E-13	3.2E-14	1.4E-13
13	1.1E-13	4.1E-13	1.6E-13	1.6E-12	1.6E-13	9.7E-13
14	1.0E-14	8.5E-14	1.2E-14	4.9E-14	6.9E-15	4.7E-14

always chooses the largest element in a column as the next pivot, we get reduced errors compared with MA37. The reduced error in the solution for MA46 over MA37 is maintained on all the three platforms that we considered. Table 5.12 shows the errors as computed on DEC 3000-400 for three vectors stored in B, i.e. NRHS=3, with components B(i, 1) = i, B(i, 2) = 1, and B(i, 3) a pseudo-random number in [-1,1] generated by the Harwell Subroutine Library code FA04 [2].

The relative numerical errors are computed as:

$$\varepsilon = \frac{\| b - Ax \|}{\| Ax \|}, \tag{5.1}$$

where the max norm is used and b, x refer to one column in B and X, respectively.

6 Summary and conclusions

This report has described the implementation of a new code for the solution of sets of linear equations where the matrices and the structure are of finite-element form. We have given a brief description of the input philosophy and the design of the code. The numerical experiments show that the code performs well and that it is faster than the code MA37 from the Harwell Subroutine Library [2].

7 Acknowledgement

We would like to thank Iain Duff, Nick Gould and Jennifer Scott for useful discussions and remarks that have improved the report.

8 References

1 Amestoy, P.R., Davis, T.A., and Duff, I.S.: "An approximate minimum degree ordering algorithm", To appear in SIAM J. Matrix Anal. and Applics., 1996.

2 Anon. Harwell Subroutine Library Catalogue (Release 12). AEA Technology, Harwell Laboratory, Oxfordshire, 1995.

3 Ashcraft, C.C., and Grimes, R.G.: "The influence of relaxed supernode partitions on the multifrontal method", *Technical Report ETA-TR-60-R1*, Boeing Computer Services, 1988.

4 Dongarra, J.J., Du Groz, J., Hammarling, S., and Hanson, R.J.: "An extended set of Fortran basic linear algebra subprograms", *ACM Trans. Math. Softw.* **14**, 1-17, 1988.

5 Dongarra, J.J., Du Groz, J., Duff, I.S., and Hammarling, S.: "A set of level 3 basic linear algebra subprograms", *ACM Trans. Math. Softw.* **16**, 1-17, 1990.

6 Duff, I.S., and Reid, J.K.: "The multifrontal solution of indefinite sparse symmetric linear systems", *ACM Trans. Math. Softw.* **9**, 302-325, 1983.

7 Duff, I.S., and Reid, J.K.: "The multifrontal solution of unsymmetric sets of linear equations", *SIAM J. Sci. Stat. Comput.* **5**, 633-641, 1984.

8 Duff, I.S., Grimes, R.G., and Lewis, J.G.: "Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release 1)", *Technical Report RAL-92-086*, Rutherford Appleton Laboratory, Chilton DIDCOT Oxon OX11 0QX, UK, December 1992.

9 Duff, I.S., and Scott, J.A.: "MA42-A new frontal code for solving sparse unsymmetric systems", *Technical Report RAL-93-064*, Rutherford Appleton Laboratory, Chilton Didcot Oxon OX11 0QX, UK, September 1993.

10 Eisenstat, S.C., Schultz, M.H., and Sherman, A.H.: "Yale sparse matrix package I: The symmetric codes", *Int J. Num. Methods Eng.*, **18**, 1145-1151, 1982.

11 Gilbert, J.R., Ng, E.G. and Peyton, B.W.: "An efficient algorithm to compute row and column counts for sparse Cholesky factorization", *Technical Report ORNL/TM-12195*, Oak Ridge National Laboratory, Oak Ridge, TN, USA, September 1992.

12 Lawson, C., Hanson, R., Kincaid, D., and Krogh, F.: "Basic linear algebra subprograms for Fortran usage", *ACM Trans. Math. Softw.* **5**, 308-329, 1979.

13 Liu, J.W.H.: "Modification of the minimum-degree algorithm by multiple elimination", *ACM Trans. Math. Softw.* **11**, 141-153, 1985.

14 Liu, J.W.H.: "On the storage requirement in the out-of-core multifrontal method for sparse factorization", *ACM Trans. Math. Softw.* **12**, 249-264, 1986.

15 Liu, J.W.H.: "The role of elimination trees in sparse factorization", *SIAM J. Matrix Anal. Appl.*, **11**, 134-172, 1990.

Appendix A. Auxiliary routines and data structures used in MA46

This appendix describes the auxiliary routines and the main internal data structures of the MA46 package. The package consists of two set of subroutines MA46 and MA56. The former includes the user-callable routines along with auxilliary routines, while the latter includes only auxilliary routines needed by the package.

Tables A.1 and A.2 list the auxiliary routines of the MA46 package and explain their tasks.

Table A.1: The MA46* auxiliary routines.

Routine	Task
MA46D	Given a permutation, it computes the inverse permutation.
MA46E	Given an element-node connectivity structure, it computes the corresponding node-element connectivity structure.
MA46F	It computes a minimum-degree ordering of the nodes.
MA46G	It computes the nodal elimination tree and the corresponding postordering.
MA46H	It computes the length of each nodal column in the triangular factor L, or the row length of the triangular factor U.
MA46J	Given the nodal postordered elimination tree and the column length of the triangular factor L, it computes the corresponding fundamental supernode partition of the nodes. It also computes the adjacency set representation of the supernode elimination tree.
MA46K	Given the supernode elimination tree, it computes an optimal or a standard depth-first postorder of the supernode elimination tree.
MA46L	It updates the permutation and the supernode elimination tree after a depth-first search of the supernode elimination tree as performed by MA46K.
MA46M	It computes the number of assembly steps, the assembly sequence of the elements, the element assembly tree, and does the final updates of the permutation vectors and the supernode partition. In addition it computes factorization statistics.
MA46N	It performs a symbolic assembly step of generated elements in a block elimination step.
MA46O	It performs a symbolic assembly step of original finite elements in an assembly step.
MA46P	It finalizes the index information needed for a supernode in a block elimination step.
MA46Q	It performs coefficient assembly of generated elements into the frontal matrix in a block elimination step.
MA46R	It performs coefficient assembly of original finite elements into the frontal matrix in an assembly step.
MA46S	It performs pivot search in a block elimination step.
MA46T	It prints triangular factors to standard output unit defined by ICNTL(2).
MA46U	It performs the block forward substitution steps of the right-hand sides.
MA46V	It checks the forward solved right-hand sides for consistency with the matrix system in case of a rank deficient system.
MA46W	It performs the block backward substitution steps of the right-hand sides.
MA46X	Called from MA46T to print an M times N matrix to standard output defined by ICNTL(2).
MA46Y	It copies an integer vector from vector IX to vector IY.
MA46Z	It updates the permutation given a permutation increment.

Table A.2: The MA56* auxiliary routines.

Routine	Task
MA56A	It initializes data structures for the minimum-degree routine.
MA56B	It updates the graph representation due to the elimination of a minimum-degree node.
MA56C	It updates the degree of nodes after a multiple elimination step in the minimum-degree routine.
MA56D	After the nodes are eliminated in the minimum-degree routine, it computes the final permutation of the nodes.
MA56E	It computes the nodal elimination tree.
MA56F	It computes the first child-sibling vectors of the nodal elimination tree to facilitate fast postordering of the tree.
MA56G	It postorders the nodal elimination tree by a depth-first search and computes the corresponding permutation increment
MA56H	It updates the permutation with the increment computed by the depth-first search of the nodal elimination tree.
MA56I	It computes the adjacency set representation of an elimination tree represented as a parent vector.
MA56J	It sorts a list of integers in decreasing order of their keys using insertion sort.
MA56K	It computes the stack storage for a non-trivial supernode, i.e. an updated postordering of the supernode elimination tree.
MA56L	Given the information computed by MA56K, it computes a postordering of the supernode elimination.
MA56M	It sorts a list of integers into ascending order.
MA56N	It compresses lists held by MA56B for the generated elements and adjusts the pointers.
MA56O	It computes the leftmost unexplored children of a node in an elimination tree in connection with depth-first search of the tree.
MA56P	It computes the right sibling of a child in an elimination tree in connection with depth-first search of the tree.

Tables A.3, A.4 and A.5 show the structure of the calls in MA46A, MA46B and MA46C, including the calls to the BLAS routines that are used in the package.

Table A.3: Structure of the calls in MA46A.

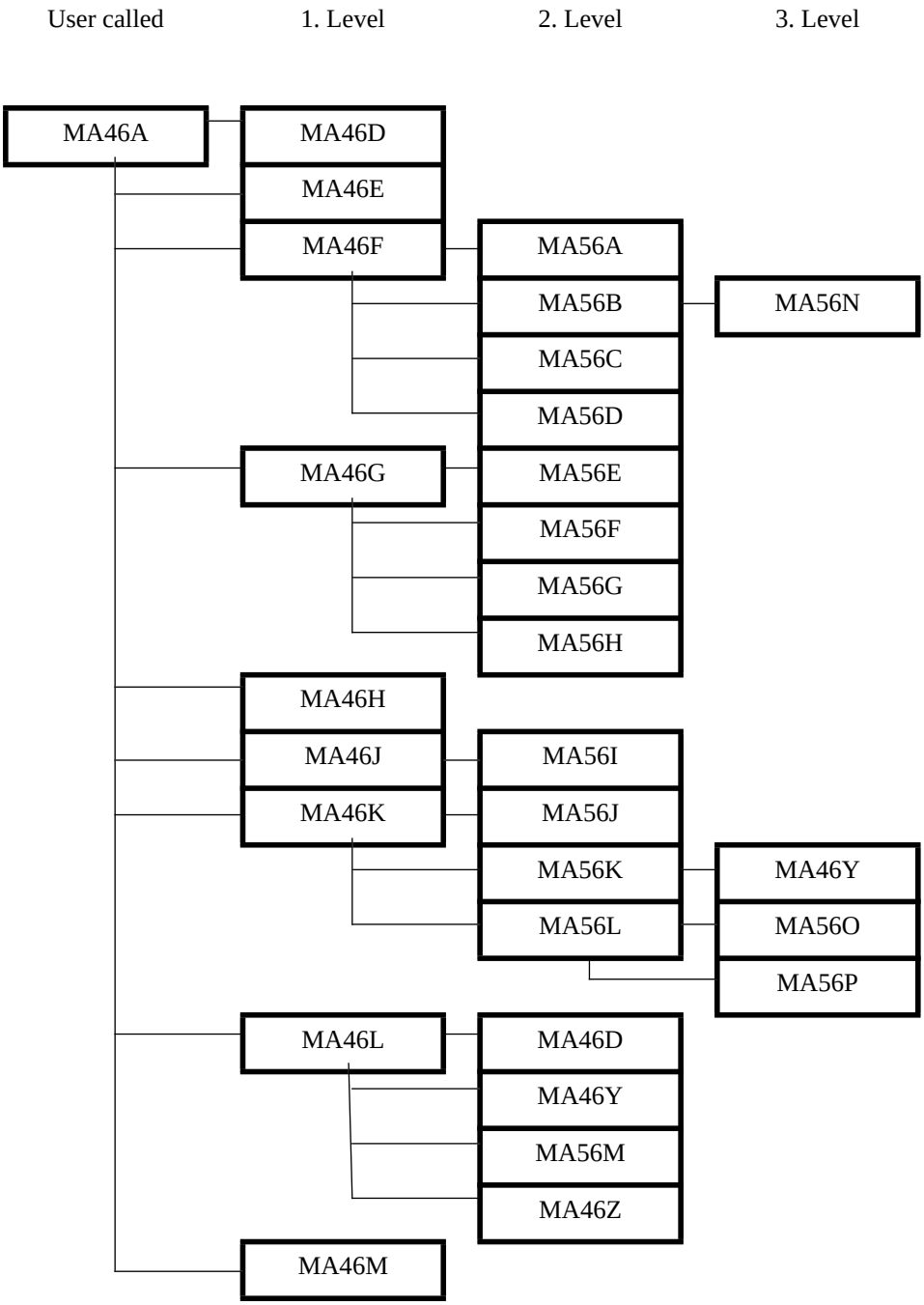


Table A.4: Structure of the calls in MA46B.

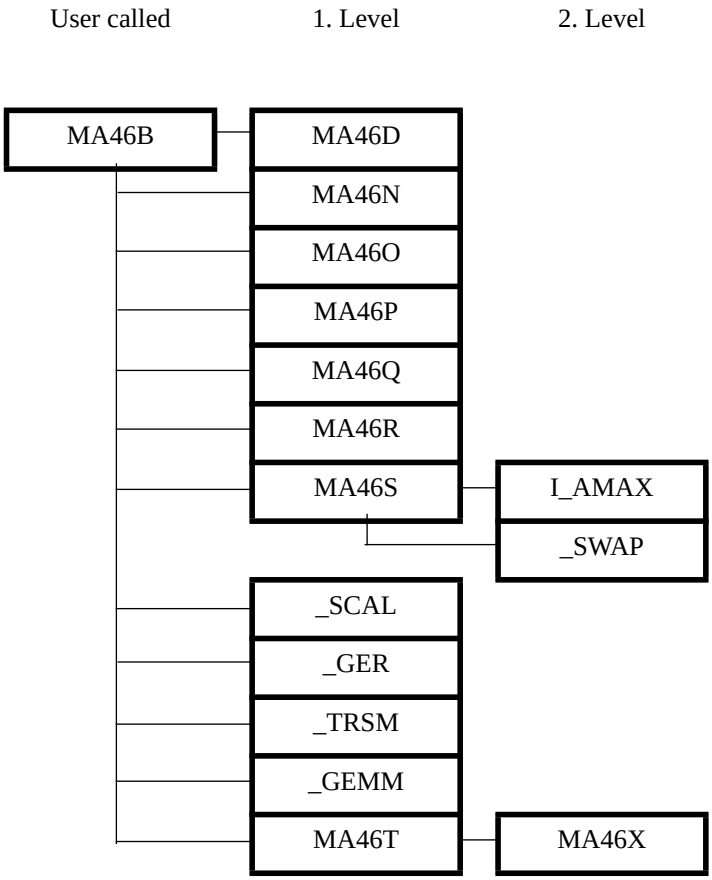


Table A.5: Structure of the calls in MA46C.

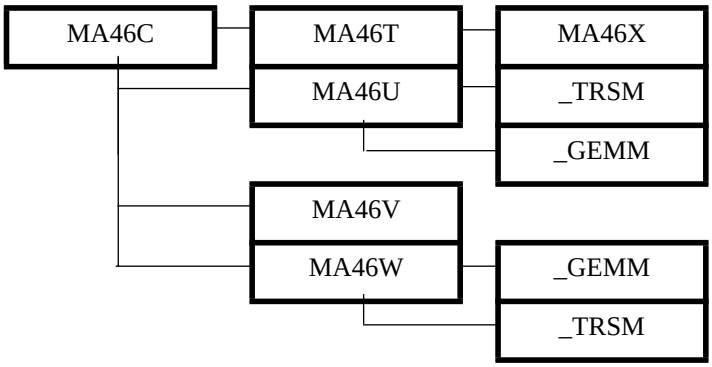


Table A.6 defines the parameter array MNPAR that is stored in the first 50 locations of KEEPA.

Table A.6: Definition of the sub array MNPAR stored in the first 50 locations of KEEPA. The length of MNPAR currently in use is 31.

Location	Name	Definition
1	STAGE	Stage control.
2	NB	Number of assembly steps.
3	NSUPER	Number of block factorization steps.
4	NACTIV	Number of nodes that is active ascomputed by MA46A.
5	NACTEQ	Number of equations as computed by MA46A.
6	NNODS	Number of nodes as input by the user.
7	NEQNS	Number of equations as input by the user.
8	NELS	Number of elements as input by the user.
9	NELNOD	Length of the implicit graph representation arrays.
10	NCOMPR	Number of workspace compressions performed by MA46F.
11	MAXSUP	The largest supernode as computed by MA46A. (Block factorization node).
12	MAXFRT	Order of the largest front matrix as computed MA46A.
13	NOFSUB	Number of indices needed to represent the factors as computed by MA46A.
14	NZEROU	Size of the upper triangular factor as computed by MA46A.
15	NZSTCK	Size of the stack as computed by MA46A.
16	DPSTCK	Depth of the stack.
17	KPUSED	Length of KEEPA after MA46A.
18	MAXLIW	The maximum length of IW that was used in MA46A.
19	NELACT	Number of elements that actually participate in the assembly.
20	LKEEPB	Length of KEEPB as anticipated in MA46A.
21	LFACT	Length of FACT for triangular factors and stack as computed by MA46A.
22	LKEEPB	Length of KEEPB after factorization.
23	LFACT	Length of FACT for triangular factors.
24	TPSTCK	Top of the stack.
25	MAXFRT	Order of the largest front matrix after factorization.
26	MAXSUP	The largest supernode after factorization. (Block factorization node).
27	NOFSUB	The number of indices needed to represent the factors.
28	NZEROU	The number of real storage locations needed to hold the upper triangular factor U.
29	NZSTCK	The maximal number of real storage locations needed to hold the stack.
30	NELIMS	Number of eliminations performed by MA46B.
31	MAXLAW	Minimum length of FACT for successful factorization.

Table A.7 shows the structure of KEEPA on exit from MA46A, i. e. the form of the array as passed to MA46B and MA46C, and Table A.8 explains the contents of each sub-array except for MNPAR that is explained in Table A.6.

Table A.7: Structure of KEEPA on exit from MA46A.

KEEPA	MNPAR	XELSEQ	ELSEQ	BSPERM	XSUPER

KEEPA	XSIBL	SIBL	SUPLEN	XBLOCK

Table A.8: Definition of the sub-arrays that are stored in KEEPA on exit from MA46A.

Sub-array	Length on entry	Length on exit	Definition
XELSEQ	NNODS+1	NB+1	Pointer to the list of original finite elements stored in ELSEQ that are needed in the assembly step *i*, for *i*=1:NB.
ELSEQ	NELS	NELACT	The lists of original finite elements that are needed in each assembly step.
BSPERM	NNODS	NNODS	The new to old permutation of the nodes from the final order of the nodes computed by MA46A to the order of the nodes as input by the user.
XSUPER	NNODS+1	NSUPER+1	The supernode partition related to the variables.
XSIBL	NNODS+1	NSUPER+1	Pointer to the supernode elimination tree adjacency set that is stored in SIBL for each supernode.
SIBL	NNODS+1	NSUPER+1	The lists of supernode children belonging to supernode *i*, for *i*=1:NSUPER. IP=SIBL[NSUPER+1] gives the number of supernodes that are roots of connected trees in the supernode elimination tree, and these roots are found in locations SIBL[NSUPER-IP+1] to SIBL[NSUPER].
SUPLEN	NNODS	NSUPER	The column length of each supernode in the triangular factor *L* or the length of each row in the triangular factor *U*.
XBLOCK	NNODS+1	NB+1	The assembly tree partition of the supernodes. I.e. the structure of the assembly tree.

From Tables A.6 and A.8 we have:
length of KEEPA on entry to MA46A=NELS+7*NNODS+55, and:
length of KEEPA on exit from MA46A=NELACT+4*NSUPER+2*NB+NNODS+55.

Tables A.9 shows the structure of KEEPB on exit from MA46B, and Table A.10 explains the content of each sub-array.

Table A.9: Structure of KEEPB on exit from MA46B.

KEEPB	XFINDX	FINDX

Table A.10: Definition of the sub-arrays that are stored in KEEPB on exit from MA46B.

Sub-array	Length on entry	Length on exit	Definition
XFINDX	NSUPER	NSUPER	XFINDX[*i*] points to the start in FINDX for the index information stored for supernode, or block elimination step *i*.
FINDX	FLIMIT[a]	MAXSUB[b]	See Table A.11 for a definition of the quantities that are stored for each supernode, or block elimination step.

a. FLIMIT=3*NSUPER+NOFSUB+NACTEQ, see the explanation of these quantities in Table A.6.
b. For a diagonally dominant matrix we have MAXSUB=3*NSUPER+NOFSUB where NSUPER and MAXSUB are as anticipated by MA46A.

Table A.11: Initial, intermediate and final structure of FINDX for supernode, or block elimination step *i* in routine MA46B.

A[*i*]	B[*i*]	C[*i*]	Row indices for [*i*]	Column indices for [*i*]

A[*i*]	(i) it is set to the anticipated size of the front matrix for supernode *i* as computed in MA46A. (ii) after symbolic assembly of generated element indices it is updated with the number of delayed eliminations received from its incoming children, i.e. it holds the active size of the new frontal matrix. If at least one child had negative value of A[*child*], then A[*i*] is also negated to signal that column indices for the variables in the supernode need be stored as well as the row indices. (iii) after numerical eliminations in supernode *i* it is not changed if A[*i*] was already set to a negative value in step (ii), if it was positive from (ii) it is set to a negative value if at least one off-diagonal pivot was selected.
B[*i*]	(i) it is set to the anticipated number of delayed eliminations to be received from the children of the supernode, i.e. B[*i*]=0. (ii) after symbolic assembly of generated elements it is incremented with the total number of delayed eliminations that have been received from its incoming children which is its final form.
C[*i*]	(i) it is set to the size of the supernode, i.e. to the number of eliminations to be performed in the supernode, i.e. C[*i*]=XSUPER[*i+1*]-XSUPER[*i*]. (ii) after symbolic assembly it is set to the number of indices found so far. (iii) after elimination it holds the number of eliminations performed.
Row indices for [*i*]	After elimination of supernode, or block elimination step *i* it holds the global row indices in ascending order for the supernode and the column of L below the supernode.

Table A.11: Initial, intermediate and final structure of FINDX for supernode, or block elimination step *i* in routine MA46B.

Column indices for [*i*]	After elimination of supernode, or block elimination step *i* it is not stored if A[*i*] is positive. If A[*i*] is negative it holds the column indices for the pivots found during the elimination of the supernode. The number of pivots found is C[*i*].

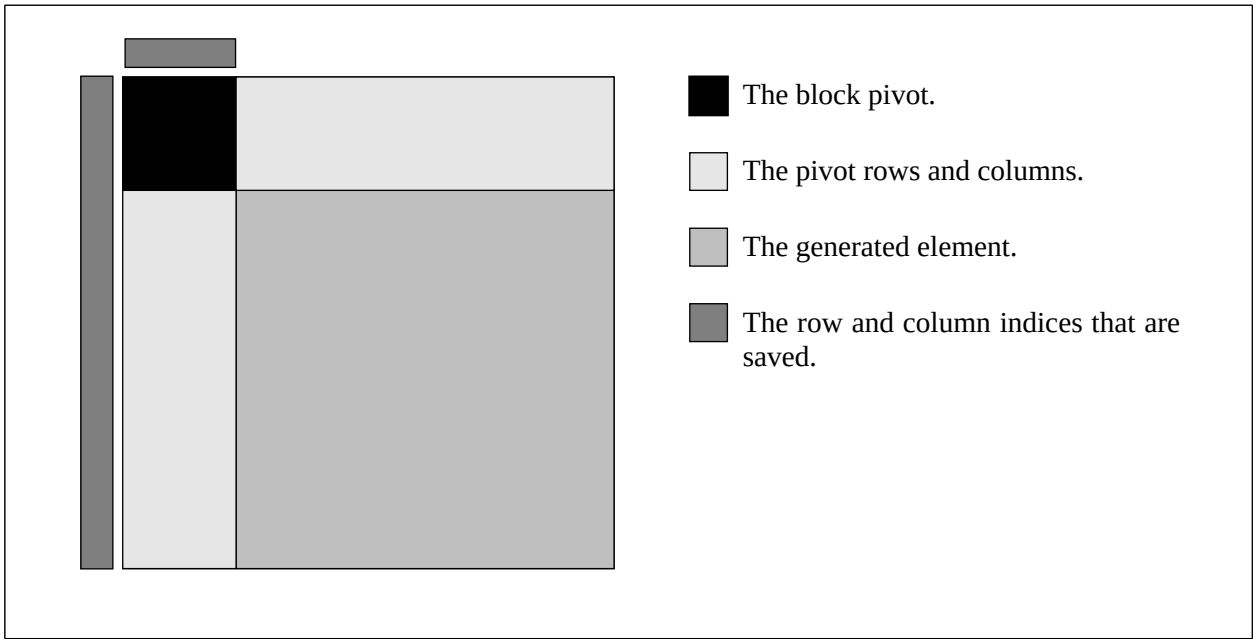


Figure A.1: The figure shows a frontal matrix after the eliminations have been carried out and shows the row and column indices that are saved when at least one off-diagonal pivot was selected.

Table A.12 defines the contents of FACT on exit from MA46B and Figure A.2 shows the finalized *L* and *U* blocks and indicates the workspace compression that is performed in MA46 after each supernode, or block elimination step has been carried out.

Table A.12: The contents of FACT on exit from MA46B. The *L*/*U* factors are stored as a sequence of submatrices.

FACT	*L*/*U*[1]	*L*/*U*[2]	*L*/*U*[3]	*L*/*U*[4]	*L*/*U*[5]	. . .	*L*/*U*[NSUPER]

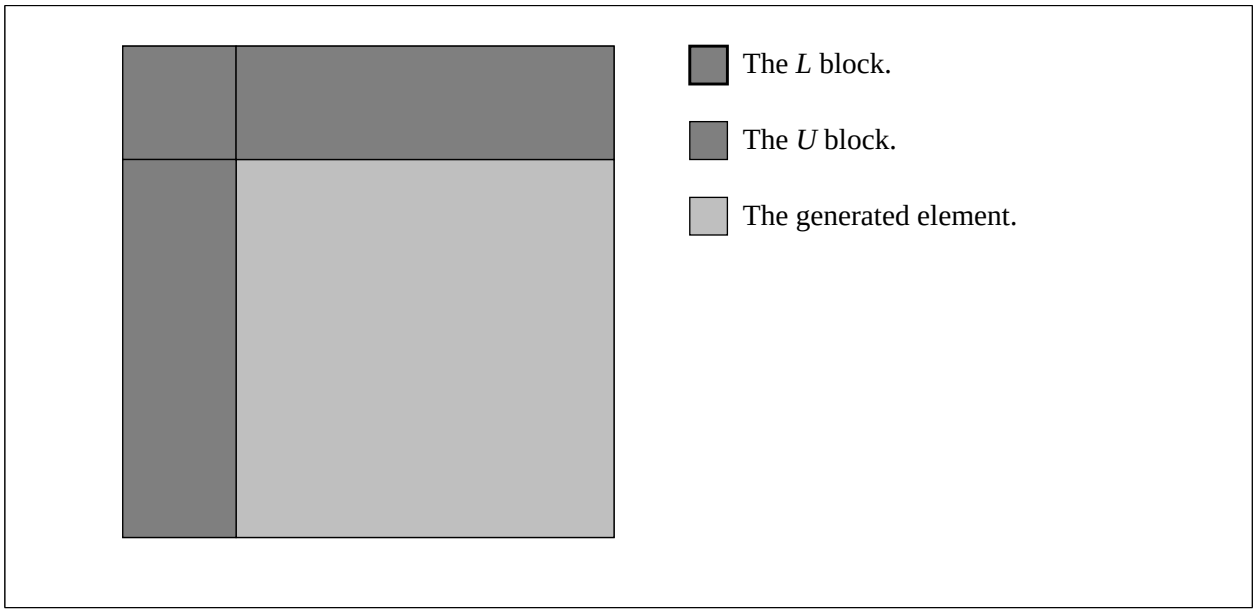


Figure A.2: The *L* and *U* blocks of the finalized frontal matrix.

The workspace compression is done as follows: (i) the *L* block is moved from the frontal matrix to the final storage for supernode submatrices, (ii) the *U* block is moved and compressed, and (iii) the generated element is moved to the new stack top and compressed at the same time. It is the space between the column of the finalized *U* block segments, or between the column segments of the generated element, that is taken into account and freed after every elimination step.

Appendix B. The specification document for MA46

In this appendix, we include a copy of the specification document for MA46. The code itself is available from AEA Technology, Harwell; the contact is Dr Scott Roberts or Mr Richard Lee, AEA Technology, Bldg 552, Harwell, Didcot, Oxon OX11 0RA, tel (44) 1235 434714 or (44) 1235 435690, Fax (44) 1235 434136, email: scott.roberts@aeat.co.uk or richard.lee@aeat.co.uk, who will provide details of price and conditions of use.

1 SUMMARY

To **solve one or more set of sparse unsymmetric linear equations Ax = B from finite-element applications, using a multifrontal elimination scheme.** The matrix **A** must be input by elements and be of the form

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)}$$

where $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables of the nodes of the k-th element. Optionally, the user may pass an additional matrix $\mathbf{A}_d$ of coefficients for the diagonal. **A** is then of the form

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)} + \mathbf{A}_d$$

The right-hand side **B** should be assembled through the summation

$$\mathbf{B} = \sum_{k=1}^{m} \mathbf{B}^{(k)},$$

before calling the solution routine.

ATTRIBUTES — Versions: MA46A, MA46AD. **Calls:** MA56, _GEMM, _GEMV, _GER, I_AMAX, _SCAL, _SWAP, _TRSM, _TRSV. **Origin:** A.C. Damhaug, Det Norske Veritas Research AS and J.K. Reid, Rutherford Appleton Laboratory. **Date:** September 1995. **Conditions on external use:** (i), (ii), (iii) and (iv).

2 HOW TO USE THE PACKAGE

2.1 Argument lists and calling sequences

There are four routines that can be called by the user:

(a) MA46I/ID sets default values for the control parameters for the other routines.

(b) MA46A/AD accepts the matrix pattern by element-node connectivity lists and chooses diagonal pivots for Gaussian elimination to preserve sparsity while disregarding numerical values. It also constructs information for the numerical factorization to be performed by MA46B/BD. The user may provide a pivot sequence by means of node numbers, in which case the necessary information for MA46B/BD will be generated.

(c) MA46B/BD factorizes the finite-element matrix **A** by NB calls, where NB is the number of assembly steps computed by routine MA46A/AD. For all the elements involving a node, the variables at the node must be in the same order. The actual pivot sequence may differ from that specified by MA46A/AD or provided by the user, due to numerical stability considerations.

(d) MA46C/CD uses the factors generated by MA46B/BD to solve the set of linear equations. The solution overwrites the right-hand side.

Normally, the user will call MA46I/ID prior to the call of any other routine in the package. If non-default values for any of the control parameters are required, they should be set immediately after the call to MA46I/ID. A call to MA46C/CD must be preceded by a call to MA46B/BD, which in turn must be preceded by a call to MA46A/AD. Since the information passed from one routine to the next is not corrupted by the second, several sequences of calls to MA46B/BD for matrices with the same sparsity pattern but different values may follow a single call to MA46A/AD, and similarly MA46C/CD can be used repeatedly to solve for different sets of right-hand sides **B**.

2.1.1 To set default values for control parameters

The single precision version

```
CALL MA46I(CNTL,ICNTL)
```

The double precision version

```
CALL MA46ID(CNTL,ICNTL)
```

CNTL is a REAL (DOUBLE PRECISION in the D version) array of length 2 that need not be set by the user. On return

it contains default values. For further information see Section 2.2.

ICNTL is an INTEGER array of length 10 that need not be set by the user. On return it contains default values. For further information see Section 2.2.

2.1.2 To perform ordering and generate assembly tree

The single precision version

```
      CALL MA46A(NELS,NNODS,NEQNS,IPIELT,IELT,LIELT,IVAR,NB,KEEPA,LKEEPA,
     $           IW,LIW,ICNTL,RINFO,INFO)
```

The double precision version

```
      CALL MA46AD(NELS,NNODS,NEQNS,IPIELT,IELT,LIELT,IVAR,NB,KEEPA,LKEEPA,
     $            IW,LIW,ICNTL,RINFO,INFO)
```

NELS is an INTEGER variable that must be set by the user to the largest integer that is used to index a finite element. It is not altered by the routine.

NNODS is an INTEGER variable that must be set by the user to the largest integer that is used to index a finite-element node. It is not altered by the routine.

NEQNS is an INTEGER variable that must be set by the user to the number of variables. It is not altered by the routine.

IPIELT is an INTEGER array of length NELS+1. It must be set by the user so that the nodes connected to element I are in IELT(IPIELT(I)), IELT(IPIELT(I)+1), ..., IELT(IPIELT(I+1)-1) for I = 1, 2, ... , NELS. It is not altered by the routine.

IELT is an INTEGER array of length LIELT that must be set by the user to contain the lists of nodes in each element. Its length must be at least IPIELT(NELS+1)-1. It is not altered by the routine.

LIELT is an INTEGER variable that must be set by the user to the length of IELT. It is not altered by the routine.

IVAR is an INTEGER array of length NNODS that must be set by the user. It gives the number of variables for each node. It may contain values equal to zero. A node, I, I = 1, 2, ... , NNODS that has IVAR(I)=0 is not processed. It is not altered by the routine.

NB is an INTEGER variable that need not be set by the user. On exit it holds the number of assembly steps needed to factor the matrix. This variable must be preserved between a call to MA46A/AD and a sequence of calls to MA46B/BD.

KEEPA is an INTEGER array of length at least NELS+7*NNODS+55. If the user wishes to provide an ordering for the nodes, the index of the node in position i must be placed in KEEPA(i), i = 1, 2, ... , NNODS and ICNTL(4) must be set to 1. The given order is likely to be replaced by one that is equivalent apart from reordering of additions and subtractions. Otherwise, KEEPA need not be set by the user. On exit, KEEPA contains, in locations KEEPA(51:51+NB), a pointer array into KEEPA for the sequence of finite elements needed in each assembly step. For assembly step, IBL, IBL = 1, 2, ... , NB, the index of the first element required by MA46B/BD is found in location KEEPA(KEEPA(IBL)+NB+51) and the last element index is found in location KEEPA(KEEPA(IBL+1)-1+NB+51). The number of elements needed in assembly step IBL is thus KEEPA(50+IBL+1)-KEEPA(50+IBL). KEEPA must be preserved between a call to MA46A/AD and other routines.

LKEEPA is an INTEGER variable that must be set by the user to the length of KEEPA. It is not altered by the routine.

IW is an INTEGER array of length LIW that need not be set by the user. It is used as workspace by the routine. Its length must be at least $\max(l_1,l_2)$, where l_1 = 3*NELS+2*NNODS+4*LIELT+8*NEQNS+2 (or NELS+NNODS+2*LIELT+2 if the pivot order is specified in KEEPA), and l_2 = NELS+11*NNODS+2*LIELT+5.

LIW is an INTEGER variable that must be set by the user to the length of IW. It is not altered by the routine.

ICNTL is an INTEGER array of length 10 that contains control parameters and must be set by the user. Default values for the components may be set by a call to MA46I/ID. Details of the control parameters are given in Section 2.2. It is not altered by the routine.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of length 6 that need not be set by the user. For the meaning of the values of components of RINFO set by MA46A/AD, see Section 2.2.

INFO is an INTEGER array of length 16 that need not be set by the user. On return from MA46A/AD, a value of zero for INFO(1) indicates that the routine has performed successfully. For nonzero values, see Section 2.3. For the meaning of the value of other components of INFO set by MA46A/AD, see Section 2.2.

2.1.3 To factorize a matrix

To factorize the matrix, MA46B/BD uses 'reverse communication' which means that the routine must be called by the user NB times, where NB is the number of assembly steps determined by MA46A/AD. In each call, the user must pass a specified sequence of finite-element coefficient matrices to the routine.

The single precision version

```
      CALL MA46B(IBL,NELS,NNODS,IPIELT,IELT,LIELT,IVAR,KEEPA,LKEEPA,KEEPB,
     $           LKEEPB,ELMAT,A,LA,AD,LAD,IW,LIW,CNTL,ICNTL,RINFO,INFO)
```

The double precision version

```
      CALL MA46BD(IBL,NELS,NNODS,IPIELT,IELT,LIELT,IVAR,KEEPA,LKEEPA,KEEPB,
     $            LKEEPB,ELMAT,A,LA,AD,LAD,IW,LIW,CNTL,ICNTL,RINFO,INFO)
```

IBL is an INTEGER variable that must be set by the user to the current assembly step. Calls to the routine must be in the order IBL = 1, 2, ..., NB. It is not altered by the routine.

NELS, NNODS, IPIELT, IELT, LIELT and IVAR are as in the preceding call to MA46A/AD and their values must not have changed. They are not altered by the routine.

KEEPA is an INTEGER array of length LKEEPA. It must be as on exit from MA46A/AD. It is not altered by the routine.

LKEEPA is an INTEGER variable that must be set by the user to the length of KEEPA. It must be at least as great as INFO(2) as output from MA46A/AD (see Section 2.2). It is not altered by the routine.

KEEPB is an INTEGER array of length at least LKEEPB that need not be set by the user. It is used as workspace by MA46B/BD and on exit holds integer index information on the matrix factors. It must be preserved by the user between the calls to this routine and subsequent calls to MA46C/CD.

LKEEPB is an INTEGER variable that must be set by the user to the length of KEEPB. It must be at least as great as INFO(8) as output from MA46A/AD (see Section 2.2). A greater value is recommended because numerical pivoting may increase storage requirements. It is not altered by the routine.

ELMAT is a REAL (DOUBLE PRECISION in the D version) array that must be set by the user to hold the element coefficient matrices for this assembly step, column by column in the sequence defined by KEEPA(FIRST), KEEPA(FIRST+1), ..., KEEPA(LAST), where FIRST = KEEPA(IBL)+51+NB and LAST = KEEPA(IBL+1)+50+NB. It is not altered by the routine.

A is a REAL (DOUBLE PRECISION in the D version) array of length LA that need not be set on the first entry to MA46B/BD. It must be preserved between the calls to MA46B/BD and for subsequent calls to MA46C/CD. On exit from each intermediate call, A will hold the entries of the factors of the matrix **A** that have been completed. On exit from the final call, A holds the factors needed by MA46C/CD.

LA is an INTEGER variable that must be set by the user to the length of A. It must be at least as great as INFO(9) as output from MA46A/AD (see Section 2.2). It is advisable to allow a greater value because the use of numerical pivoting may increase storage requirements. It is not altered by the routine.

AD is a REAL (DOUBLE PRECISION in the D version) array of length LAD that need not be set if ICNTL(10) has its default value (see Section 2.2). Otherwise, its NEQNS first positions must hold the coefficients for the diagonal of **A**. It is assumed by the routine that variables at nodes are stored consecutively and that nodes are in the initial order. MA46B/BD alters the order of the entries according to the tentative pivot order computed by MA46A/AD.

LAD is an INTEGER variable that must be set by the user to the length of AD. It must be set to at least 1 if ICNTL(10) has its default value. Otherwise, it must be set to a value as least as great as NEQNS as input to MA46A/AD.

IW is an INTEGER array of length LIW that need not be set by the user. It is used as workspace by the routine.

LIW is an INTEGER variable that must be set by the user to the length of IW. It must be at least as great as 3*(NNODS+NEQNS)+1. NNODS and NEQNS are as input to MA46A/AD. It is not altered by the routine.

CNTL is a REAL (DOUBLE PRECISION in the D version) array of length 2 that contains control parameters and must be set by the user. Default values for the components may be set by a call to MA46I/ID. Details of the control parameters are given in Section 2.2. It is not altered by the routine.

ICNTL is an INTEGER array of length 10 that contains control parameters and must be set by the user. Default values for the components may be set by a call to MA46I/ID. Details of the control parameters are given in Section 2.2. It is not altered by the routine.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of length 6 that need not be set by the user. For the meaning of the values of components of RINFO set by MA46B/BD, see Section 2.2.

INFO is an INTEGER array of length 16 that need not be set by the user. On return from MA46B/BD, a value of zero for INFO(1) indicates that the routine has performed successfully. For nonzero values, see Section 2.3. For the meaning of the value of other components of INFO set by MA46B/BD, see Section 2.2.

2.1.4 To solve equations, given the factorization

The single precision version

```
      CALL MA46C(IVAR,NNODS,KEEPA,LKEEPA,KEEPB,LKEEPB,A,LA,B,LDB,NRHS,
     $           IW,LIW,RW,LRW,ICNTL,INFO)
```

The double precision version

```
      CALL MA46CD(IVAR,NNODS,KEEPA,LKEEPA,KEEPB,LKEEPB,A,LA,B,LDB,NRHS,
     $            IW,LIW,RW,LRW,ICNTL,INFO)
```

NNODS and IVAR are as in the preceding call to MA46A/AD and their values must not have changed. They are not altered by the routine.

KEEPA is an INTEGER array of length at least LKEEPA. The first INFO(2) components must be as on exit from MA46B/BD.

LKEEPA is an INTEGER variable that must be set by the user to the length of KEEPA. It must be at least as great as INFO(2) as output from MA46A/AD (see Section 2.2). It is not altered by the routine.

KEEPB is an INTEGER array of length at least LKEEPB. The first INFO(8) components must be as on exit from MA46B/BD.

LKEEPB is an INTEGER variable that must be set by the user to the length of KEEPB. It must be at least as great as INFO(8) as output from MA46B/BD (see Section 2.2). It is not altered by the routine.

A is a REAL (DOUBLE PRECISION in the D version) array of length LA that must be unchanged since the call to MA46B/BD. It is not altered by the routine.

LA is an INTEGER variable that must be set by the user to the length of A. It must be at least as great as INFO(9) as output from MA46B/BD which may be smaller than predicted in MA46A/AD (see Section 2.2). It is not altered by the routine.

B is a REAL (DOUBLE PRECISION in the D version) array of leading dimension LDB, whose first NRHS columns must be set by the user to hold the right-hand sides. It is assumed that the right-hand side is passed to MA46C/CD in the input node order with the variables at each node stored consecutively. On exit, the solution overwrites the right hand side and the initial nodal order with variables at each node stored consecutively is maintained.

LDB is an INTEGER variable that must be set by the user to the leading dimension of B. It must be at least as great as NEQNS. It is not altered by the routine.

NRHS is an INTEGER variable that must be set by the user to hold the number of right hand sides to be solved in this call to MA46C/CD. It is not altered by the routine.

IW is an INTEGER array of length LIW that need not be set by the user. It is used as workspace by the routine and must be preserved between the calls to the routine.

LIW is an INTEGER variable that must be set by the user to the length of IW. It must be at least as great as NNODS+NEQNS+1. NNODS and NEQNS are as input to MA46A/AD. It is not altered by the routine.

RW is a REAL (DOUBLE PRECISION in the D version) work array of length as least as great as INFO(15) as output from MA46B/BD, that need not be set by the user. It is used as workspace by the routine.

LRW is an INTEGER variable that must be set by the user to the length of RW. It is not altered by the routine.

ICNTL is an INTEGER array of length 10 that contains control parameters and must be set by the user. Default values for the components may be set by a call to MA46I/ID. Details of the control parameters are given in Section 2.2. It is not altered by the routine.

INFO is an INTEGER array of length 16 that need not be set by the user. On return from MA46C/CD, a value of zero for INFO(1) indicates that the routine has performed successfully. For nonzero values, see Section 2.3. For the meaning of the value of other components of INFO set by MA46C/CD, see Section 2.2.

2.2 Arrays for control and information

The elements of the arrays CNTL and ICNTL control the action of MA46A/AD, MA46B/BD and MA46C/CD. Default values for the elements are set by MA46I/ID. The elements of the arrays RINFO and INFO provide information on the action of MA46A/AD, MA46B/BD and MA46C/CD.

CNTL(1) has default value 0.1 and is used for pivoting by MA46B/BD. Values greater than 1.0 are treated as 1.0 and less than zero as zero.

CNTL(2) has default value zero. If it is set to a positive value, MA46B/BD will treat any pivot whose modulus is less than CNTL(2) as zero.

ICNTL(1) has default value 6 and holds the unit number to which the error messages are sent.

ICNTL(2) has default value 6 and holds the unit number to which warning messages and additional printing is sent.

ICNTL(3) is used by the routines to control printing. It has default value 1. Possible values are:

- 0 No printing.
- 1 Error messages only.
- 2 Error and warning messages only.
- 3 Scalar parameters and a few entries of arrays on entry and exit from routines.
- 4 All parameter values printed on entry and exit from routines.

ICNTL(4) has default value 0. It must be set by the user to a value of 1 when calling MA46A/AD if a pivot sequence is being supplied by the user in array KEEPA.

ICNTL(5) has default value 0. This option is related to the node ordering step of MA46A/AD. If the value is zero or less, the minimum external degree algorithm is used. Multiple elimination is used when the value is zero. If value is greater than zero, multiple elimination is still in effect, but the minimum external degree condition is relaxed (see Section 4).

ICNTL(6) has default value 0. With this value, MA46A/AD reorders the assembly steps to reduce the temporary working storage required by MA46B/BD while computing the triangular factors. If ICNTL(6) is set to 1, MA46A/AD uses a standard depth first postordering of the assembly steps.

ICNTL(7) has default value 0. It is ignored in the present version, but the intention is for a later version to have the option of amalgamating tree nodes into supernodes even if this introduces additional structural zeros.

ICNTL(8) has default value 64. MA46B/BD is written to make good use of the cache memory if its size in kBytes is ICNTL(8). Setting the value to zero will mean that the routine assumes that the computer has no cache.

ICNTL(9) has default value 0. It is ignored in the present version, but the intention is for a later version to have the option of using indirect addressing in the solve step of MA46C/CD.

ICNTL(10) has default value 0. This means that no diagonal matrix A_d is used to specify the diagonal matrix nonzero coefficients, otherwise ICNTL(10) must be set to 1.

RINFO(1) gives the number of floating-point additions used to assemble the original finite-element matrix coefficients.

RINFO(2) gives the number of floating-point additions used to assemble the generated elements if the tentative pivot sequence calculated by MA47A/AD is acceptable numerically.

RINFO(3) gives the sum of floating-point additions, multiplications and divisions used to factorize the matrix if it the tentative pivot sequence calculated by MA47A/AD is acceptable numerically.

RINFO(4) gives the number of floating-point additions used to assemble the generated elements in MA46B/BD.

RINFO(5) gives the sum of floating-point additions, multiplications and divisions used to factorize the matrix in MA46B/BD.

RINFO(6) gives the sum of floating-point additions, multiplications and divisions used to solve one set of linear equations in MA46C/CD.

INFO(1) has the value zero if the call was successful, and a negative value in the event of an error (see Section 2.3).

INFO(2) gives the required size of KEEPA in MA46B/BD and MA46C/CD on exit from MA46A/AD if INFO(1)=0. If INFO(1)=–1 it gives the required size of KEEPA needed in MA46A/AD.

INFO(3) gives the size of IW that has been used in MA46A/AD or in MA46B/BD if INFO(1)=0. If INFO(1)=−1 it gives the required size of IW needed in MA46A/AD. If INFO(1)=−6 it gives the required size of IW needed in MA46B/BD.

INFO(4) gives the number of entries out of range for INFO(1)=−2.

INFO(5) gives the number of duplicate entries for INFO(1)=−2.

INFO(6) gives the number of active nodes computed by MA46A/AD if INFO(1)=0.

INFO(7) gives the number of variables computed by MA46A/AD if INFO(1)=0.

INFO(8) gives the minimum required length of KEEPB in MA46B/BD on exit from MA46A/AD and the required length of KEEPB in MA46C/CD on exit from MA46B/BD if INFO(1)=0. If INFO(1)=−6 on exit from MA46B/BD it gives the minimum required length of KEEPB for a successful exit.

INFO(9) gives the minimum required length of A in MA46BD on exit from MA46A/AD and the required length of A in MA46C/CD on exit from MA46B/BD if INFO(1)=0. If INFO(1)=−7 on exit from MA46B/BD it gives the minimum required length of A for a successful exit.

INFO(10) gives the order of the largest front matrix if INFO(1)=0.

INFO(11) gives the number indices in the factorized matrix if INFO(1)=0.

INFO(12) gives the number of entries in the factorized matrix if INFO(1)=0.

INFO(13) gives the number of assembly steps if INFO(1)=0.

INFO(14) gives the number of elements if INFO(1)=0.

INFO(15) gives the size of the largest front matrix that occured in the factorization step if INFO(1)=0.

INFO(16) gives the number of eliminations done by MA46BD if INFO(1)=0.

2.3 Error diagnostics

A successful return from MA46A/AD or MA46B/BD is indicated by a value of INFO(1) equal to zero. Possible nonzero values for INFO(1) are given below.

A nonzero flag value is associated with an error message that will be output on unit ICNTL(1).

-1 The length of KEEPA and/or IW is not great enough (MA46A/AD).

-2 Entries in KEEPA are out of range and/or are duplicates (MA46A/AD).

-3 NEQNS less than the number of variables computed by MA46A/AD or the number of variables computed is less than one (MA46A/AD).

-4 Indices out of range in IELT (MA46A/AD).

-5 NELS ≤ 0, and/or NNODS ≤ 0, and/or NEQNS ≤ 0 (MA46A/AD).

-6 The length of KEEPB and/or IW is not great enough. (MA46B/BD).

-7 The length of A is not great enough. (MA46B/BD).

-8 Error from previously called routine is not cleared (MA46B/BD or MA46C/CD).

-9 Error in the symbolic assembly step in MA46B/BD. Signals that KEEPA may have been altered before the call to MA46B/BD.

2.4 Singular systems

If the matrix is singular, MA46B/BD factorizes a nonsingular submatrix. A warning message is written if the right-hand side is not consistent with the factorization.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: MA46D/DD, MA46E/ED, MA46F/FD, MA46G/GD, MA46H/HD, MA46J/JD, MA46K/KD, MA46L/LD, MA46M/MD, MA46N/ND, MA46O/OD, MA46P/PD, MA46Q/QD, MA46R/RD, MA46S/SD, MA46T/TD, MA46U/UD, MA46V/VD, MA46W/WD, MA46X/XD, MA46Y/YD, MA46Z/ZD, MA56A/AD, MA56B/BD, MA56C/CD, MA56D/DD, MA56E/ED, MA56F/FD, MA56G/GD, MA56H/HD, MA56I/ID, MA56J/JD, MA56K/KD,

MA56L/LD, MA56M/MD, MA56N/ND, MA56O/OD and MA56P/PD.

The package uses the Basic Linear Algebra Subprograms SGEMM/DGEMM, SGEMV/DGEMV, SGER/DGER, SSCAL/DSCAL, ISAMAX/IDAMAX, STRSM/DTRSM and SSWAP/DSWAP.

Input/output: Error messages on unit ICNTL(1). Warning messages and additional printing on unit ICNTL(2). Each has default value 6.

4 METHOD

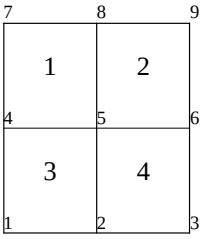
The method used is a direct method using multifrontal sparse Gaussian elimination. The matrix structure is passed to the routine in the form of element-node connectivity lists. The matrix analyse step (MA47A/AD) uses this 'unassembled' form to find the ordering of the nodes, and to build the necessary information for the factorise and solve steps. The ordering is done with the minimum degree heuristic. It is possible to relax this by altering the default value given by ICNTL(5). Setting it greater than zero has the effect of allowing nodes with degree ICNTL(5) greater than the minimum to be eliminated together with the nodes of minimum degree. Sometimes, this helps to reduce the size of the decomposition. The final assembly tree is reordered in an attempt to reduce the size of the working stack. The default value of option ICNTL(6) gives this and is recommended. The factorization step (MA47B/BD) is provided by the analyse step with a tentative pivot sequence, which it uses except when this would be numerically unstable. The numerical stability criterion is the relative pivot tolerance given by CNTL(1), with a default value of 0.1. In general, increasing its value gives a more stable factorization, but increases in the size of the decomposition. A value of 1.0 gives partial pivoting as defined for the dense matrix case.

Reference A.C. Damhaug and J.K. Reid (1994) MA46, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations from finite-element applications. Rutherford Appleton Laboratory Report, to appear.

5 EXAMPLE OF USE

We give an example of the code required to solve a set of equations using the MA46 package. The example illustrates the use of MA46 when no input order and additional diagonal matrix $\mathbf{A}_d$ is provided by the user. There are two right-hand sides to solve for.

We wish to solve the following simple finite-element problem in which the finite-element mesh consists of four 4-noded elements with two degrees of freedom at each node. The nodes 1, 4, and 7 are assumed constrained, which means that they do not contribute to the matrix system to be solved.



The input to the routine is then:

```
NELS   = 4
NNODS  = 9
NEQNS  = 12
LIELT  = 16
IVAR   = [0,2,2,0,2,2,0,2,2]
IPIELT = [1,5,9,13,17]
IELT   = [4,5,8,7,5,6,9,8,1,2,5,4,2,3,6,5]
```

The four elemental matrices $\mathbf{A}^{(k)}$ $(1 \le k \le 4)$ are

$$\begin{array}{c} 5 \\ 8 \end{array}\begin{pmatrix} 6. & 2. & 3. & 4. \\ 1. & 5. & 4. & 3. \\ 2. & 3. & 4. & 4. \\ 3. & 2. & 3. & 1. \end{pmatrix}$$

$$\begin{array}{c} 5 \\ 6 \\ 9 \\ 8 \end{array}\begin{pmatrix} 4. & 4. & 3. & 4. & 5. & 4. & 3. & 4. \\ 3. & 1. & 4. & 3. & 4. & 2. & 4. & 3. \\ 2. & 3. & 6. & 2. & 3. & 4. & 7. & 2. \\ 3. & 2. & 1. & 5. & 4. & 3. & 2. & 6. \\ 4. & 3. & 2. & 3. & 4. & 4. & 3. & 4. \\ 3. & 1. & 3. & 2. & 3. & 1. & 4. & 3. \\ 2. & 3. & 6. & 1. & 2. & 3. & 6. & 2. \\ 3. & 2. & 1. & 5. & 3. & 2. & 1. & 5. \end{pmatrix},$$

$$\begin{array}{c} 2 \\ 5 \end{array}\begin{pmatrix} 6. & 2. & 3. & 4. \\ 1. & 5. & 4. & 3. \\ 2. & 3. & 4. & 4. \\ 3. & 2. & 3. & 1. \end{pmatrix} \qquad \begin{array}{c} 2 \\ 3 \\ 6 \\ 5 \end{array}\begin{pmatrix} 4. & 4. & 3. & 4. & 5. & 4. & 3. & 4. \\ 3. & 1. & 4. & 3. & 4. & 2. & 4. & 3. \\ 2. & 3. & 6. & 2. & 3. & 4. & 7. & 2. \\ 3. & 2. & 1. & 5. & 4. & 3. & 2. & 6. \\ 4. & 3. & 2. & 3. & 4. & 4. & 3. & 4. \\ 3. & 1. & 3. & 2. & 3. & 1. & 4. & 3. \\ 2. & 3. & 6. & 1. & 2. & 3. & 6. & 2. \\ 3. & 2. & 1. & 5. & 3. & 2. & 1. & 5. \end{pmatrix},$$

where the node numbers are indicated by the integers before each matrix (columns are identified symmetrically to rows). The two right-hand side vectors $\mathbf{b}^{(k)}$ $(1 \le k \le 2)$ are

$$\begin{array}{c} 2 \\ 3 \\ 5 \\ 6 \\ 8 \\ 9 \end{array}\begin{pmatrix} 0. \\ 0. \\ 1. \\ 0. \\ 0. \\ 0. \\ 1. \\ 0. \\ 0. \\ 0. \\ 1. \\ 0. \end{pmatrix} \qquad \begin{array}{c} 2 \\ 3 \\ 5 \\ 6 \\ 8 \\ 9 \end{array}\begin{pmatrix} 0. \\ 0. \\ 0. \\ 1. \\ 0. \\ 0. \\ 0. \\ 2. \\ 0. \\ 0. \\ 0. \\ 1. \end{pmatrix},$$

where the node numbers are indicated by the integers before each vector.

The following program is used to solve this problem.

```
      INTEGER           NELS    , NNODS    , NEQNS     , LIELT
      PARAMETER       ( NELS = 4, NNODS = 9, NEQNS = 12, LIELT = 16 )
      INTEGER           IVAR(NNODS), IPIELT(NELS+1), IELT(LIELT)
      INTEGER           KEEPA(200), KEEPB(200), IW(300), XELMAT(10),
     $                  ELSIZE(10)
      INTEGER           NB    , LKEEPA, LIW   , LKEEPB, LA    , LAD   ,
     $                  IBL   , IPEL  , LDB   , NRHS  , LRW   , LAMAX ,
     $                  LELMAT, L1    , L2    , I     , J     , K     ,
     $                  MNPAR , NODE  , NVAR  , XELSEQ, ELEMNT, ELSEQ ,
     $                  LORDER, ISTRT , ISTOP
      PARAMETER       ( LAMAX = 200, LELMAT = 200, MNPAR = 50 )
      DOUBLE PRECISION  ELMAT(LELMAT), A(LAMAX), AD(NEQNS), B(NEQNS,2),
     $                  RW(NEQNS),RELMAT(LELMAT)
      INTEGER           ICNTL(10), INFO(16)
      DOUBLE PRECISION  CNTL(2), RINFO(6)

      NRHS=2

*     ---------------------
*     READ IN THE DATA SET.
*     ---------------------
      READ(5,'(10I3)') (IVAR(I),I=1,NNODS)
      READ(5,'(10I3)') (IPIELT(I),I=1,NELS+1)
      READ(5,'(10I3)') (IELT(I),I=1,LIELT)
      READ(5,'(8F5.0)') (ELMAT(I),I=1,160)
      READ(5,'(12F5.0)') ((B(I,J),I=1,NEQNS),J=1,NRHS)

*     ----------------------------------------
*     COMPUTE THE ORDER OF THE ELEMENT MATRICES.
*     ----------------------------------------
      DO 200 I = 1, NELS
         ELSIZE(I) = 0
         DO 100 J = IPIELT(I), IPIELT(I+1)-1
            NODE = IELT(J)
            NVAR = IVAR(NODE)
            IF ( NVAR .GT. 0 )
     $         ELSIZE(I)=ELSIZE(I)+NVAR
  100    CONTINUE
  200 CONTINUE

*     ---------------------------------------
*     CALL MA46ID TO INITIALIZE CONTROL ARRAYS.
*     ---------------------------------------
```

```
      CALL MA46ID(CNTL,ICNTL)

*     ------------------------------------------------
*     ANALYSE THE SPARSITY PATTERN BY A CALL TO MA46AD.
*     ------------------------------------------------
      LKEEPA = NELS+7*NNODS+55
      L1     = 3*NELS+2*NNODS+4*LIELT+8*NEQNS+2
      L2     = NELS+11*NNODS+2*LIELT+5
      LIW    = MAX(L1,L2)
      IF ( LKEEPA .GT. 200 .OR.
     $     LIW    .GT. 300      ) GOTO 8000

      CALL MA46AD(NELS,NNODS,NEQNS,IPIELT,IELT,LIELT,IVAR,NB,KEEPA,
     $            LKEEPA,IW,LIW,ICNTL,RINFO,INFO)
      IF ( INFO(1) .NE. 0 ) GOTO 8000

*     --------------------------------
*     STORE THE ELEMENT MATRICES IN THE
*     SEQUENCE DETERMINED BY MA46AD.
*     --------------------------------
      XELSEQ = MNPAR
      ELSEQ  = XELSEQ+NB+1
      IPEL = 1
      XELMAT(1) = IPEL
      DO 600 IBL = 1, NB
         DO 500 I = KEEPA(XELSEQ+IBL), KEEPA(XELSEQ+IBL+1)-1
            ELEMNT = KEEPA(ELSEQ+I)
            LORDER = ELSIZE(ELEMNT)
            K = 0
            DO 300 J = 1, ELEMNT-1
               K = K + ELSIZE(J)*ELSIZE(J)
  300       CONTINUE
            XELMAT(IBL+1) = XELMAT(IBL) + LORDER*LORDER
            DO 400 J = IPEL, IPEL+LORDER*LORDER-1
               K = K + 1
               RELMAT(J) = ELMAT(K)
  400       CONTINUE
            IPEL = IPEL + LORDER*LORDER
  500    CONTINUE
         XELMAT(IBL+1) = IPEL
  600 CONTINUE

*     ---------------------------------------
*     SET UP THE STORAGE REQUIRED FOR MA46BD.
*     ---------------------------------------
      LKEEPA = INFO(2)
      LKEEPB = INFO(8)
      LA     = INFO(9)
      LAD    = 1
      LIW    = 3*(NNODS+NEQNS) + 1
      IF ( LKEEPA .GT. 200 .OR.
     $     LKEEPB .GT. 200 .OR.
     $     LA     .GT. 200 .OR.
     $     LIW    .GT. 300      ) GOTO 8000

*     -----------------------------------------
*     FACTORIZE THE MATRIX BY NB CALLS TO MA46BD.
*     -----------------------------------------
      DO 700 IBL = 1, NB
         IPEL = XELMAT(IBL)
         CALL MA46BD(IBL,NELS,NNODS,IPIELT,IELT,LIELT,IVAR,KEEPA,LKEEPA,
     $               KEEPB,LKEEPB,RELMAT(IPEL),A,LA,AD,LAD,IW,LIW,CNTL,
     $               ICNTL,RINFO,INFO)
         IF ( INFO(1) .NE. 0 ) GOTO 8000
  700 CONTINUE

*     ---------------------------------------
*     SET UP THE STORAGE REQUIRED FOR MA46CD.
*     ---------------------------------------
      LKEEPB = INFO(8)
      LA     = INFO(9)
```

```
      LIW    = NNODS + NEQNS + 1
      LRW    = INFO(15)
      IF ( LKEEPB .GT. 200 .OR.
     $     LA     .GT. 200 .OR.
     $     LIW    .GT. 300 .OR.
     $     LRW    .GT. NEQNS    ) GOTO 8000
      LDB = NEQNS

*     ------------------------------------
*     SOLVE THE SYSTEMS BY A CALL TO MA46CD.
*     ------------------------------------
      CALL MA46CD(IVAR,NNODS,KEEPA,LKEEPA,KEEPB,LKEEPB,A,LA,B,LDB,NRHS,
     $            IW,LIW,RW,LRW,ICNTL,INFO)

*     --------------------------
*     PRINT THE SOLUTION VECTORS.
*     --------------------------
      ISTRT = 1
      DO 1000 NODE = 1, NNODS
         IF ( IVAR(NODE) .GT. 0 )
     $   THEN
            ISTOP = ISTRT + IVAR(NODE) - 1
            DO 900 J = 1, NRHS
               WRITE(6,'(A,I6,A,I6)')
     $         'SOLUTION VECTOR ',J,'     FOR NODE :', NODE
               WRITE(6,'(45X,1PE12.5)')
     $         (B(I,J),I=ISTRT,ISTOP)
  900       CONTINUE
            ISTRT = ISTOP + 1
         ELSE
            WRITE(6,'(/A,I6/)')
     $      'NO VARIABLES AT NODE :', NODE
         ENDIF
 1000 CONTINUE

*     ----------------------
*     PRINT SOME STATISTICS:
*     ----------------------
      WRITE(6,'(A,I6)')
     $'NUMBER OF ASSEMBLY STEPS              ',INFO(13)
      WRITE(6,'(A,I6)')
     $'NUMBER OF ELIMINATIONS PERFORMED      ',INFO(16)
      WRITE(6,'(A,I6)')
     $'ORDER OF THE LARGEST FRONT MATRIX     ',INFO(15)
      WRITE(6,'(A,I6)')
     $'LENGTH OF THE UPPER TRIANGULAR FACTOR ',INFO(12)
      WRITE(6,'(A,I6)')
     $'SIZE OF THE INDEX INFORMATION         ',INFO(11)

      STOP
 8000 CONTINUE
*     ------------------------------------
*     ERROR CONDITION, PRINT THE INFO ARRAY.
*     ------------------------------------
      WRITE(6,'(10I5)') (INFO(I),I=1,16)
      STOP
      END
```

The input data used for this problem is:

```
   0  2  2  0  2  2  0  2  2
   1  5  9 13 17
   4  5  8  7  5  6  9  8  1  2
   5  4  2  3  6  5
    6.   2.   3.   4.   1.   5.   4.   3.
    2.   3.   4.   4.   3.   2.   3.   1.
    4.   4.   3.   4.   5.   4.   3.   4.
    3.   1.   4.   3.   4.   2.   4.   3.
    2.   3.   6.   2.   3.   4.   7.   2.
    3.   2.   1.   5.   4.   3.   2.   6.
    4.   3.   2.   3.   4.   4.   3.   4.
```

```
   3.   1.   3.   2.   3.   1.   4.   3.
   2.   3.   6.   1.   2.   3.   6.   2.
   3.   2.   1.   5.   3.   2.   1.   5.
   6.   2.   3.   4.   1.   5.   4.   3.
   2.   3.   4.   4.   3.   2.   3.   1.
   4.   4.   3.   4.   5.   4.   3.   4.
   3.   1.   4.   3.   4.   2.   4.   3.
   2.   3.   6.   2.   3.   4.   7.   2.
   3.   2.   1.   5.   4.   3.   2.   6.
   4.   3.   2.   3.   4.   4.   3.   4.
   3.   1.   3.   2.   3.   1.   4.   3.
   2.   3.   6.   1.   2.   3.   6.   2.
   3.   2.   1.   5.   3.   2.   1.   5.
   0.   0.   1.   0.   0.   0.   1.   0.   0.   0.   1.   0.
   0.   0.   0.   1.   0.   0.   0.   2.   0.   0.   0.   1.
```

The program produces the following output:

```
 NO VARIABLES AT NODE :     1

 SOLUTION VECTOR      1     FOR NODE :     2
                                          -9.80559E-01
                                          -8.62854E-02
 SOLUTION VECTOR      2     FOR NODE :     2
                                           6.86096E-01
                                           9.45142E-02
 SOLUTION VECTOR      1     FOR NODE :     3
                                          -6.57556E-01
                                          -8.17085E-01
 SOLUTION VECTOR      2     FOR NODE :     3
                                           9.30706E-01
                                           5.84907E-01

 NO VARIABLES AT NODE :     4

 SOLUTION VECTOR      1     FOR NODE :     5
                                           4.80762E-01
                                           7.99617E-01
 SOLUTION VECTOR      2     FOR NODE :     5
                                          -6.56310E-01
                                          -4.63449E-01
 SOLUTION VECTOR      1     FOR NODE :     6
                                           9.15647E-01
                                           1.17792E+00
 SOLUTION VECTOR      2     FOR NODE :     6
                                          -6.22518E-01
                                          -9.86380E-01

 NO VARIABLES AT NODE :     7

 SOLUTION VECTOR      1     FOR NODE :     8
                                          -7.58728E-01
                                          -8.15599E-01
 SOLUTION VECTOR      2     FOR NODE :     8
                                           4.50445E-01
                                           8.14764E-01
 SOLUTION VECTOR      1     FOR NODE :     9
                                          -1.54214E+00
                                          -6.42698E-01
 SOLUTION VECTOR      2     FOR NODE :     9
                                           1.68228E+00
                                           2.91376E-01
 NUMBER OF ASSEMBLY STEPS                   2
 NUMBER OF ELIMINATIONS PERFORMED          12
 ORDER OF THE LARGEST FRONT MATRIX          8
 LENGTH OF THE UPPER TRIANGULAR FACTOR     62
 SIZE OF THE INDEX INFORMATION             24
```