Report no. 00/21

Understanding MA27 - A Numerical Example

C. Keller
Oxford University Computing Laboratory, Wolfson Building,
Parks Road, Oxford OX1 3QD, UK

The Harwell Subroutine Library code MA27 is a collection of FORTRAN subroutines for solving sparse sets of symmetric linear equations of the form $\mathcal{A}x = b$ by Gaussian elimination. In order to achieve a high degree of efficiency, Duff and Reid [5, 6] implement a large number of sophisticated methods including the minimum-degree ordering for reducing fill-in in the factors, the depth-first search algorithm for determining an efficient pivot ordering within a pre-computed elimination tree, and the ideas of frontal elimination for permitting a stable numerical factorisation to be performed based on a symbolically chosen pivot sequence. Understanding the interaction of all these methods within the subroutine library is not an easy task and requires the careful study of the source code. In this article an attempt is made to visualise the interactions between individual subroutines in the library by considering a specific example, and by discussing the more important theories that are implemented. We hope that by 'talking the reader through' a particular example, the beauty of the code can be appreciated and understood more easily.

We remark that permission has been given by the authors of the MA27 solver to publish this work.

Subject classifications: AMS(MOS): 65F10, 65F15, 65F50

Key words and phrases: Direct methods, Harwell Subroutine Library, MA27

This work was supported by an EPSRC Research Studentship and by a CASE Award with the Rutherford Appleton Laboratory (RAL).

Oxford University Computing Laboratory
Numerical Analysis Group
Wolfson Building
Parks Road
Oxford, England OX1 3QD
E-mail: ck@comlab.oxford.ac.uk

September, 2000

Contents

1 Introduction

Comparing direct and iterative methods for solving large linear systems of equations of the form

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

where $\mathcal{A} \in \mathbb{R}^{n \times n}$ is sparse, $b \in \mathbb{R}^n$ is the right-hand side vector, and $x \in \mathbb{R}^n$ is the solution vector, often results in the identification of the advantages and disadvantages of the two different solution strategies. For example, one of the main disadvantages of solving (1.1) using a direct method is its restrictive applicability in the context of large dimensional systems of equations. Not only is it near to impossible to find enough computer storage for an increasing number of (zero) entries in $\mathcal{A}$, but the unnecessary work of performing arithmetic with zero entries is another problematic issue that requires attention.

In the last twenty years, a large number of authors have addressed the problem of solving the linear system (1.1) in an efficient manner—see, for instance, Duff, Erisman and Reid [3], Saad [12], or George and Liu [8]. The overall aim has been to develop efficient (sparse) algorithms that solve (1.1), and which reduce the complexity of $\mathcal{O}(n^3)$ in the dense case, towards a complexity in time and space which is proportional to $\mathcal{O}(n) + \mathcal{O}(\tau)$; here, n represents the order of the given coefficient matrix, and τ denotes the number of non-zeros of $\mathcal{A}$.

Most of the advances that have been made over the years exploit the simple fact that a reordering of $\mathcal{A}$ can have drastic implications on how much work is involved in solving (1.1). In this context, the majority of these refinements are based on direct methods that use some sort of Gaussian elimination, in which case the standard approach is not to compute the LU factorisation of $\mathcal{A}$, but to factorise some permutation of the coefficient matrix. If $\mathcal{P}$ and $\mathcal{Q}$ are permutation matrices, then the 'permuted' factorisation may be written as

$$\mathcal{P}\mathcal{A}\mathcal{Q} = \mathcal{L}\mathcal{U}, \tag{1.2}$$

where $\mathcal{L}$ and $\mathcal{U}$ are lower- and upper-triangular matrices, respectively. Once the factorisation (1.2) has been performed, and the factors $\mathcal{L}$ and $\mathcal{U}$ are known, equation (1.1) is solved through the application of the *forward substitution*

$$\mathcal{L}y = \mathcal{P}b,$$

and the *back substitution*

$$\mathcal{U}(\mathcal{Q}^T x) = y.$$

Special situations arise when the coefficient matrix $\mathcal{A}$ is symmetric and positive definite, or symmetric and indefinite. In the former case, the factorisation (1.2) simplifies to

$$\mathcal{P}\mathcal{A}\mathcal{P}^T = \mathcal{L}\mathcal{L}^T, \tag{1.3}$$

where this decomposition is commonly referred to as the *Cholesky factorisation* of $\mathcal{A}$. In the latter case, the factorisation (1.2) is further modified to give the *root-free Cholesky factorisation* $\mathcal{PAP}^T = \mathcal{LDL}^T$.

In practice, the different phases of the above solution process are usually divided into a number of distinct steps, namely

(1) an *analysis phase* that examines the matrix structure in order to produce a suitable ordering and corresponding data structures for an efficient factorisation;

(2) a *factorisation phase* that performs the actual (numerical) factorisation; and

(3) a *solution phase* which uses the factors of step (2) in order to perform the forward and back substitutions.

The Harwell Subroutine Library [9] code $MA27$ is a collection of FORTRAN subroutines for solving sparse sets of linear equations of the form (1.1) by Gaussian elimination—see Duff and Reid [5,6]. No special structure apart from symmetry is assumed for the coefficient matrix $\mathcal{A}$. The generality of the implementation, which is mainly based on a frontal method approach [10,11], requires the code to be able to deal with a variety of different coefficient matrices, most eminently those which are indefinite or positive definite. Dealing with positive definiteness of the coefficient matrix is comparatively elementary and can be exploited by choosing pivots on sparsity grounds alone, simply because any choice of pivot from the diagonal results in a numerically stable factorisation. On the other hand, the possible indefiniteness of $\mathcal{A}$ requires special care due to potential zero pivots occurring during the numerical factorisation—see below for details of how $MA27$ overcomes this problem.

By applying a frontal method approach when solving (1.1), Duff and Reid describe their method in terms of finite-element problems, in which case the coefficient matrix $\mathcal{A}$ can usually be written as

$$\mathcal{A} = \sum_{l=1}^{m} \mathcal{A}^{[l]} \qquad m < n. \tag{1.4}$$

Here, each matrix $\mathcal{A}^{[l]}$ is zero except in a small number of rows and columns. The advantage of this approach is that the elimination steps

$$a_{ij} \Leftarrow a_{ij} - a_{ik} \left[a_{kk}\right]^{-1} a_{kj} \tag{1.5}$$

can often be performed before all the assembly steps

$$a_{ij} \Leftarrow a_{ij} + a_{ij}^{[l]} \tag{1.6}$$

are completed. The operations involving (1.5) can be executed as soon as the pivot row and column is fully summed, that is, as soon as all operations (1.6) have been completed for the individual elements—see the Duff and Reid's original papers [5,6] for details.

Structure within the code is accomplished by using the categorisation that we introduced above. In particular, each of the subroutines is associated with exactly one of the three tasks of

- *analysing* the sparsity pattern of the given coefficient matrix and preparing for handling numerical values;
- *factorising* the previously analysed coefficient matrix numerically; and
- *solving* the set of linear equations that represents the already factorised coefficient matrix.

Assuming that no pivot sequence has been specified on entry, the overall solution process of the MA27 multifrontal solver can be represented by Figure 1.

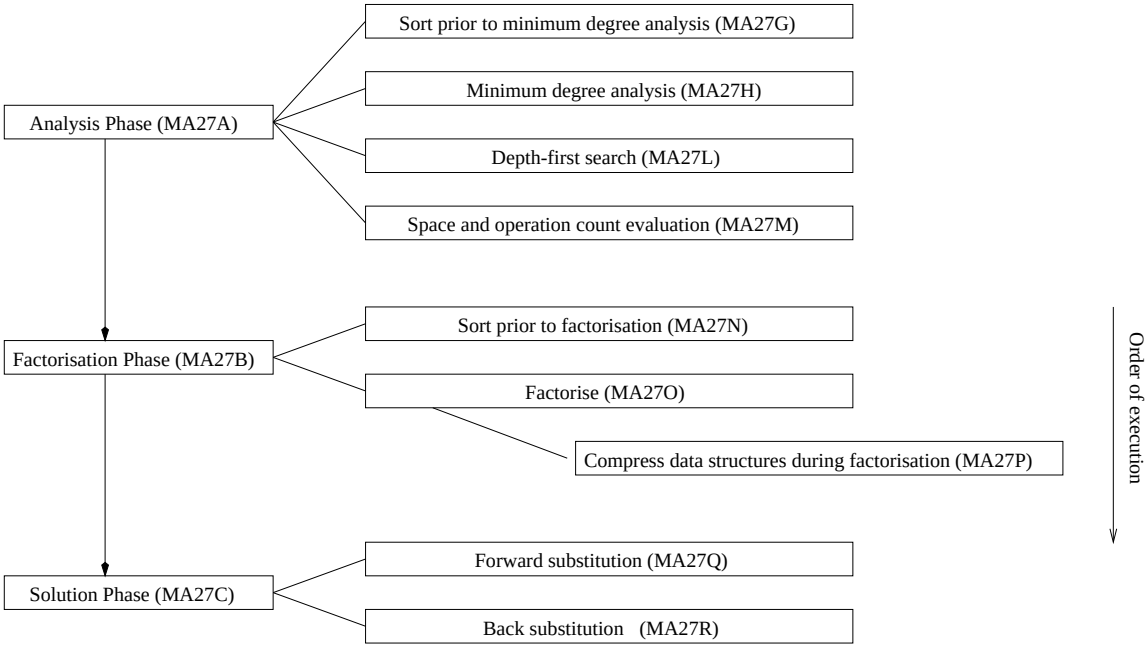


Figure 1: Call tree of the MA27 multifrontal solver.

The pivot sequence, which is determined during the *analyse phase*, can have an inadmissible effect on the remaining two solution phases. As indicated above, if the coefficient matrix $\mathcal{A}$ is symmetric and positive definite, then any choice of pivots from the diagonal gives a stable (numerical) factorisation and also preserves symmetry. The advantageous aspect of this result is that pivots may be chosen on sparsity grounds alone, and that a suitable choice of diagonal pivots can simply be made by symbolically processing on the sparsity pattern of the coefficient matrix. On the other hand, if $\mathcal{A}$ is indefinite, then this approach may fail due to (possible) zero pivots. MA27 overcomes the problem of zero pivots by using a mixture of 1×1 and 2×2 pivots which are chosen during the numerical factorisation of $\mathcal{A}$—see Duff, Reid, Munksgaard, and Neilsen [4], and also Bunch and Parlett [1], for details. The *factorisation phase*, which follows the initial analysis, is mainly characterised by its application of the frontal method—see Duff and Reid's original reports [5,6] for details.

In this report, we take a close look at the workings of the $MA27$ routines by identifying a number of important sections within the code and by trying to give a coherent description of how the different methods, which are implemented in the code, work and interact together. This is achieved by taking a particular coefficient matrix $\mathcal{A}$ and by using the chosen example to understand the code at a very low level. The example matrix, which is a random symmetric and indefinite matrix of dimension 20×20, is shown in Figure 2 (the corresponding numerical values of the coefficient matrix $\mathcal{A}$ are given in Section 3.1).

The two original papers by Duff and Reid cover the methods implemented within the subroutine library [5] as well as the code itself [6]. However, even though the descriptions give a general overview of how individual methods function within the code, and also describe Fortran specific implementation details, they do not go into great detail in doing so. Here, an attempt is made to visualise how the code performs its individual tasks and how the information constructed by the subroutines is exchanged. To this extent, we concentrate on the major methods implemented and therefore do not cover routines such as those used to compress the data structure.

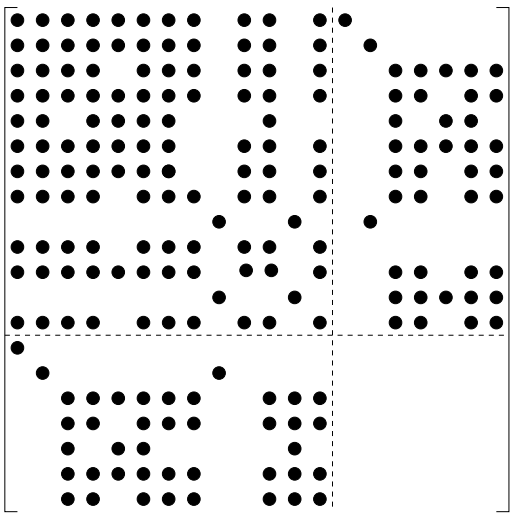


Figure 2: A 20×20 example matrix.

In Section 2, we discuss the routines that implement the analysis phase within the $MA27$ code. We focus on the implementation of the minimum degree algorithm (Section 2.1) and the depth-first algorithm (Section 2.2). Section 3 describes the individual routines that make up the factorisation phase within the code and, in particular, discuss the sorting routine $MA27N$ (Section 3.1) and the numerical factorisation that is implemented within $MA27O$ (Section 3.2). In Section 4, we describe how the factors $\mathcal{U}$ and $\mathcal{D}$, which are stored within the $MA27$ data structure, are used to perform the forward substitution (Section 4.1) and the back substitution (Section 4.2). Finally, in Section 5 we give an insight of how the block-diagonal matrix $\mathcal{D}$ and the upper-triangular matrix $\mathcal{U}$ are stored within the $MA27$ data structure, and how they can be extracted in order to be able to perform the explicit product $\mathcal{P}\mathcal{A}\mathcal{P}^T = \mathcal{U}^T\mathcal{D}\mathcal{U}$ if required.

2 The analysis of the sparsity pattern

2.1 The minimum degree analysis - MA27H

As outlined in Duff and Reid's original paper [5], the subroutine MA27H considers the non-zero pattern of the coefficient matrix in order to perform a fill-in reducing minimum degree ordering strategy. At this stage no check is performed to identify potential zero pivots. If a zero pivot occurs during the factorisation process, then the subsequent factorisation routine either delays the application of the proposed (zero) pivot, or a 2×2 pivot is used instead.

In this article, we do not state the theoretical aspects of the minimum degree algorithm, but instead describe how the actual minimum degree algorithm is implemented within the MA27H subroutine by looking at the example described in the previous section. Within $MA27H$ the non-zero pattern of the coefficient matrix $\mathcal{A}$ is stored using the INTEGER array IPE of length n, and the INTEGER*2 work-array IW. The entries of IPE are used as pointers to the lists of the columns indices of the non-zeros in the rows of $\mathcal{A}$, where these lists are stored in the array IW. If row i has no off-diagonal entries, then $IPE(i)$ is zero. During execution IPE is used to store the elimination tree that is derived by the minimum degree algorithm.

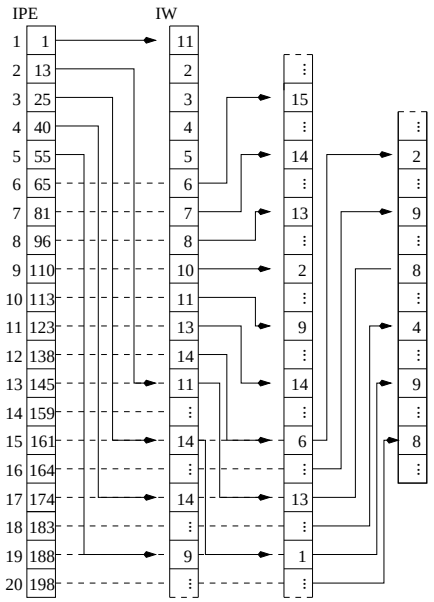


Figure 3: Storage of the non-zero pattern of $\mathcal{A}$

If a supervariable i is absorbed into supervariable j, then $IPE(i) = -j$. [1] On the other hand, if the supervariable i is eliminated, then $IPE(i)$ either points to the list of supervariables for the created element i, or it is zero if the created element is null. If element i is absorbed into element j, then $IPE(i) = -j$. Using this strategy, the coefficient matrix $\mathcal{A}$, that is depicted in Figure 2, can be represented by Figure 3.

As indicated in Figure 3, $IPE(1)$ is a pointer to the first entry of array $IW(1)$. Here, $IW(1)$ is the number of non-zero off-diagonal entries in the row 1 of $\mathcal{A}$. The entries $IW(2)$-$IW(12)$ then refer to the column indices of the off-diagonal entries. Similarly, $IPE(2)$ points to location $IW(13)$, which is the start of row 2 of $\mathcal{A}$. The remaining entries follow accordingly and are not given here.

The degree of a diagonal entry of $\mathcal{A}$, subsequently referred to as a variable, is the number of non-zero entries in the corresponding row of the coefficient matrix $\mathcal{A}$. In order to be able to update the data structure that holds the degrees of individual pivots, all

[1] A supervariable is an element consisting of two independent variables which have the same sparsity pattern.

variables of equal degree are linked together with the help of a double linked list. Three n dimensional INTEGER arrays, IPD, NXT and LST, are used for this purpose. For the above example, there is no variable of degree one; this is indicated by the array entry $IPD(1) = 0$. Similarly, there is only one variable of degree two; this is expressed by the entry $IPD(2) = 14$ and also the array entries $NXT(14) = 0$ and $LST(14) = -2$, which verify that only variable 14 has degree two. Overall, the arrays IPD, NXT and LST are

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPD	0	14	15	0	18	0	12	0	20	19	0	2	0	13	11	6	0	0	0	0

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
NXT	0	1	0	3	0	0	4	0	0	5	7	0	8	0	9	10	0	0	16	17

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LST	2	−12	4	7	10	−16	11	13	15	16	−15	−7	−14	−2	−3	19	20	−5	−10	−9

The entries of IPD refer to the degrees of a particular variable, the entries of NXT link variables of equal degree together from left to right ($NXT = 0$ indicates the last element of the list going from left to right), and the entries of LST ($LST = -d$ ($1 \leq d \leq 20$) indicates the last element of degree d in the list going from right to left) link the variables from right to left. Visually, the double linked list can be established as shown in Figure 4.

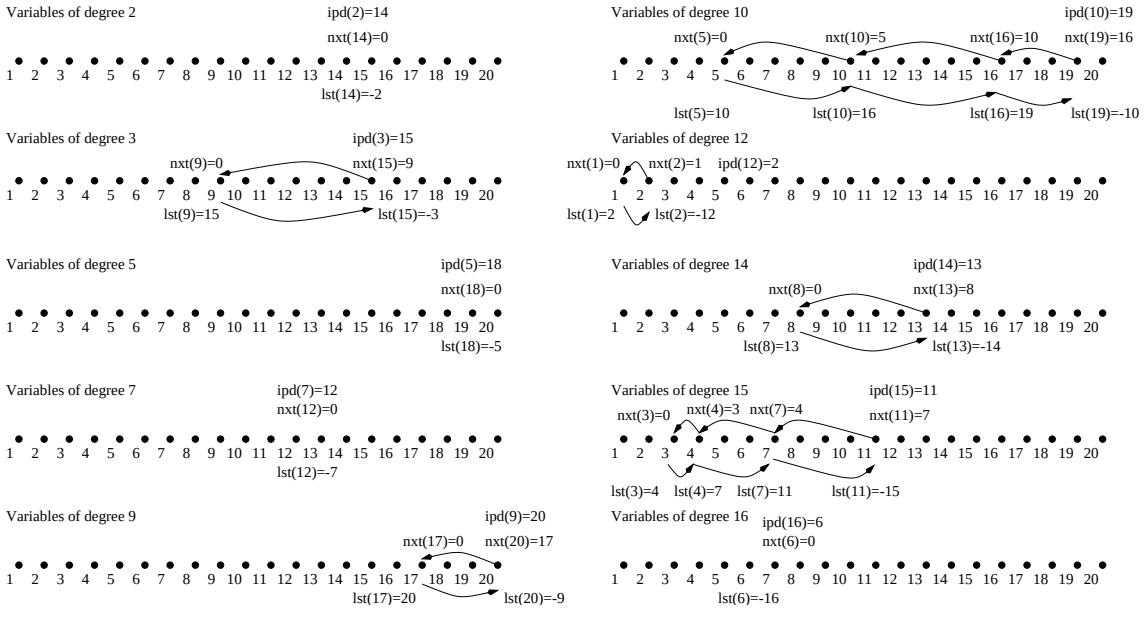
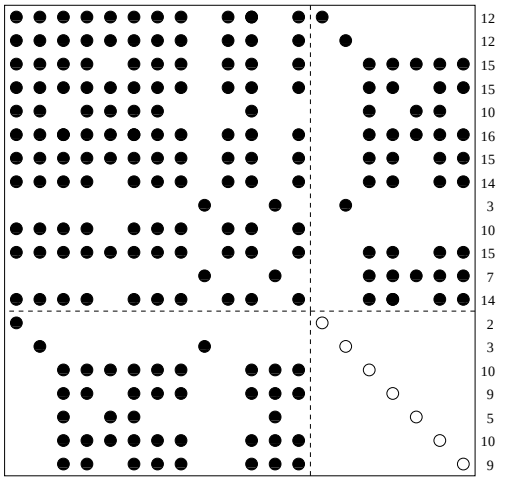


Figure 4: The minimum degree variables of $\mathcal{A}$.

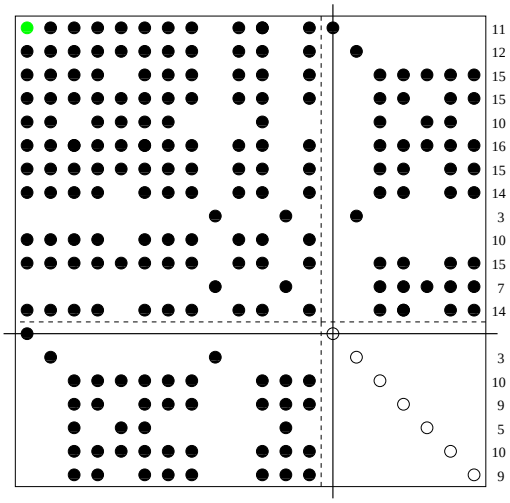
The next step after the construction of the double linked list data structure is the search for and removal of the next variable of minimum degree. We observe that within

each individual linked list of degree d, the variable with the greatest row number is eliminated first. Eliminating the variable of minimum degree is achieved by going through the array $IPD(i) \quad (i = 1, \ldots, 20)$ sequentially. A particular variable is removed simply by readjusting the pointers of the linked lists. In order to establish how this is achieved within the MA27H source, we shall consider the above example.

In the following, the elimination process at each stage can be read as follows. Black circles represent the initial non-zero entries of $\mathcal{A}$, light grey circles represent fill-in, dark grey surfaces represent eliminated pivots/elements, dotted lines stand for amalgamated rows/columns [2], and a solid line represents the element that is to be eliminated next.



Looking at the first graph, we note that pivot 14, which has degree two [3], is the variable of minimum degree. The elimination of individual pivots during the elimination process may induce fill-in which we have to take into consideration. As indicated in Figure 3, the non-zero structure of the coefficient matrix $\mathcal{A}$ is stored in the pointer array IPE and the work-array IW. At stage zero of the elimination process, the last entry of IW occurs at position 206 (this is because $IPE(20) = 198$ and there are eight non-zero elements in the current row). The original pointer $IPE(14)$ points to position 159 in the work-array IW, thus indicating that the non-zero pattern of row 14 is referenced at this position in IW.



In fact, the value of entry $IW(159)$ is 1 indicating that there is only one non-zero off-diagonal. The entry $IW(160) = 1$ and thus the non-zero off-diagonal entry occurs in column one. After the first elimination, $IPE(14)$ points to $IW(207)$ indicating that fill-in has occurred. Here, $IW(207) = 1$ (the length of the fill-in list) and $IW(208) = 1$ indicate that fill-in has occurred in column 1.

To determine the rows in which fill-in has occurred in these positions, we need to check the work-array IW. Here, each occurrence of the value 14 indicates that fill-in has occurred in the row which is represented by $IW(*) = 14$. In our example, we observe that $IW(2) = 14$ and also $IW(25) = IW(40) = IW(81) = IW(123) = 14$. The first

[2] If two rows have the same sparsity pattern, then these rows can be combined, or amalgamated, during the subsequent elimination process.

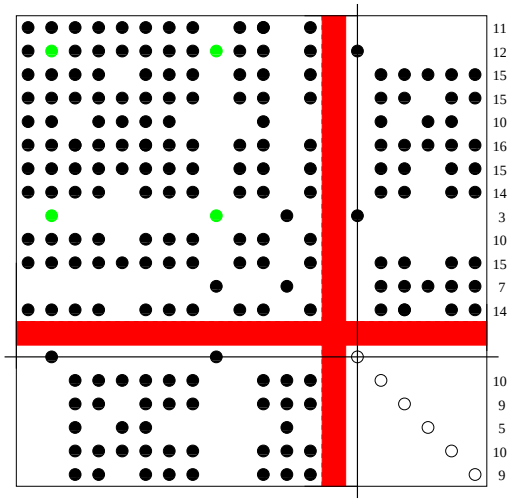
[3] In fact, row 14 has degree one (the diagonal entry $(14, 14)$ is zero). However, since the minimum degree algorithm does not look at the numerical values, and the off-diagonal is assumed to be non-zero, the degree counter returns two rather than one.

occurrence of 14, i.e., $IW(2) = 14$, indicates that fill-in has occurred in row 1. This behaviour is readily verified in the second figure in which the only fill-in has occurred in the position where row/column 14 have a non-zero entry. The remaining occurrences of the value 14 do not refer to fill-in that has occurred during the factorisation, but refer to the length of individual sub-lists in the overall list—as indicated above, positions 25, 40, 81 and 123 are the first entries of corresponding rows pointed to by the pointers in the array IPE. Thus, after the elimination of the first variable, the array IPE is given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	1	13	25	40	55	65	81	96	110	113	123	138	145	207	161	164	174	183	188	198

In the following, we look specifically at the entries of the pointer array IPE and the work-array IW in order to determine the fill-in that occurs during individual elimination stages. As indicated above, the elimination of pivot 14 implies an update of the $(1, 1)$ entry of $\mathcal{A}$. This is specified by the light-gray circles.

As indicated by the degrees of individual rows (specified by the numbers on the right-hand side in the previous figure), the next pivot to be eliminated, i.e., the one of minimum degree, is element 15.



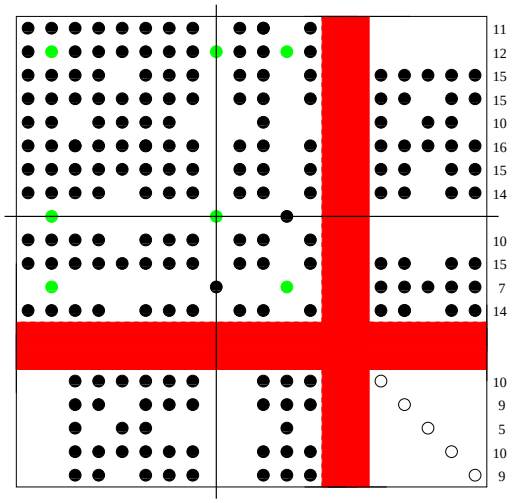
Again, looking at the individual entries of arrays IPE and IW, we note that the initial value of $IPE(15) = 161$ has been updated to $IPE(15) = 209$, thus indicating that fill-in occurred in the elimination of this variable. We have $IW(209) = 2$ (the length of the fill-in list), $IW(210) = 9$ and $IW(212) = 2$, and so fill-in occurred in columns 9 and 2. The corresponding rows can, as before, be found by checking array IW for the occurrence of the value 15. We obtain, $IW(14) = 15$ (row 2) and $IW(111) = 15$ (row 9), so that row 2 and 9 have fill-in in columns 2 and 9. This is indicated in the figure by light-gray circles. As before, the entry $IW(65) = 15$ refers to the length of the corresponding row—in this case row 6—and not to an entry that has filled-in. The updated pointer array IPE is now given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	1	13	25	40	55	65	81	96	110	113	123	138	145	207	209	164	174	183	188	198

Checking the current degree counts, we observe that the next pivot to be eliminated is 9, which has degree 3. The original value of $IPE(9) = 110$ is changed to $IPE(9) = 212$, thus indicating that fill-in has occurred during the elimination. We have $IW(212) = 2$ (the length of the fill-in list), $IW(213) = 12$ and $IW(214) = 2$. The corresponding rows

that have a non-zero entry in column 9 are $IW(14) = 9$ (row 2) and $IW(139) = 9$ (row 12). It follows that the elimination of pivot 9 has induced fill-in in rows 2 and 12, which all have new entries in columns 2 and 12. This is again indicated in the pictures by light-gray circles. As before, the entries $IW(55) = IW(113) = IW(164) = IW(188) = 9$ refer to the length of corresponding sub-lists within IW.

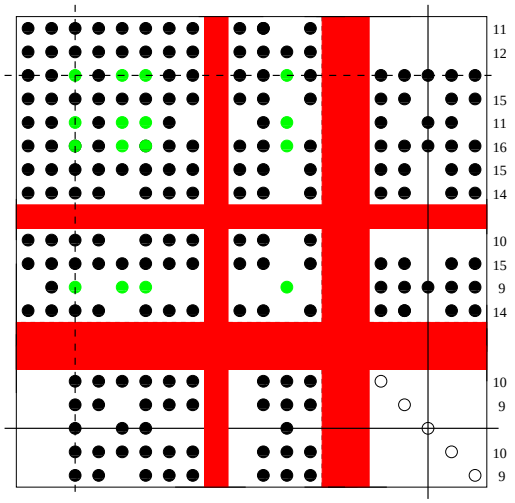
Also, the entry $IW(210)$ arises from the elimination of pivot 15 and corresponds to row 9; it is currently being eliminated. The updated pointer array IPE is given by



i	1	2	3	4	5	6	7	8	9	10
IPE	1	13	25	40	55	65	81	96	212	113
i	11	12	13	14	15	16	17	18	19	20
IPE	123	138	145	207	−9	164	174	183	188	198

The negative entry of $IPE(15)$, i.e., $IPE(15) = -9$, indicates that the eliminated element 15 has been absorbed into element 9. It follows that the elimination tree, which is stored in IPE in form of son to father tree pointers, has its first entry by means of element 15 being the son of its father 9. The overall elimination tree is given in the next section.

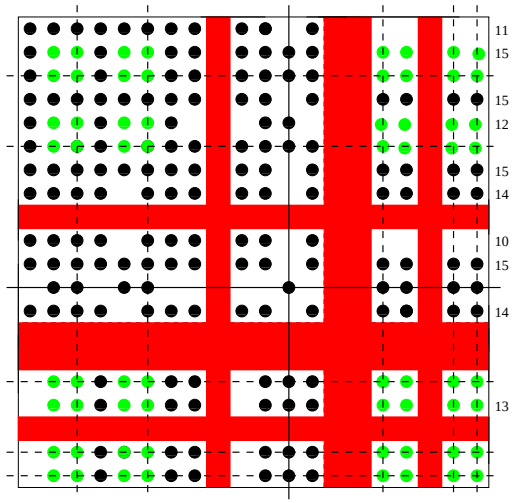
The next pivot to be eliminated is 18. Checking the relevant indices, we find that the original value of $IPE(18) = 183$ has been changed to $IPE(18) = 215$. The corresponding work-array entries are $IW(215) = 5$ (the length of the fill-in list), $IW(216) = 6$, $IW(217) = 12$ and $IW(218) = 5$. Again, looking through array IW, we find that $IW(26) = 18$ (row 3), $IW(56) = 18$ (row 5), $IW(66) = 18$ (row 6) and $IW(139) = 18$ (row 12). Thus, the elimination of pivot 18 has induced fill-in in rows 3, 5, 6 and 12 and for each row in columns 3, 5, 6 and 12. One peculiarity of the current state is that rows 3 and 6 have the same sparsity pattern, which is due to the fill-in induced by previous eliminations. It follows that rows 3 and 6 can be amalgamated, i.e., $3 \to 6$. This state is indicated by dotted lines through the respective row/column. As before, we need to update the pointer array IPE in order to incorporate the current state of the elimination tree, i.e.,



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	1	13	−6	40	55	65	81	96	212	113	123	138	145	207	−9	164	174	215	188	198

As shown above, element 3 has been amalgamated into element 6, which is indicated by the fact that $IPE(3) = -6$.

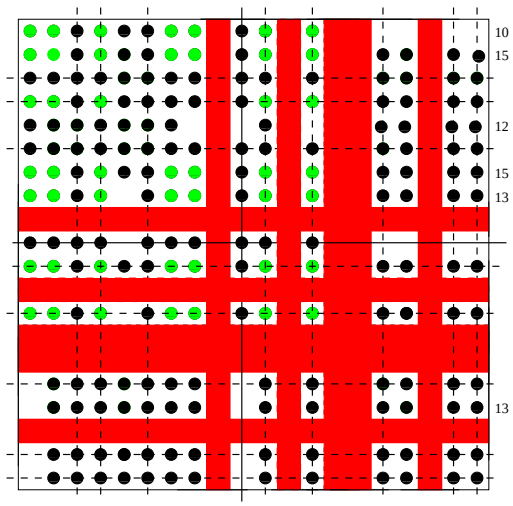
As before, checking relevant degree counts, we observe that the next pivot of minimum degree is element 12 which has degree 9. The initial value of $IPE(12) = 138$ is changed to $IPE(12) = 219$, thus indicating that fill-in has occurred during the elimination process. We obtain $IW(219) = 3$ (the length of the fill-in list), $IW(220) = 2$, $IW(221) = 17$, and $IW(222) = 5$. Checking for the value 12 in the work-array, we obtain $IW(14) = 12$ (row 2), $IW(56) = 12$ (row 5), and $IW(175) = 12$ (row 17). It follows that rows 2, 5 and 17 all have fill-in in columns 2, 5, and 17. Note the amalgamation of row $6 \rightarrow 3$, row $19 \rightarrow 20$, row $20 \rightarrow 16$, and row $16 \rightarrow 17$. The corresponding pointer array is



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	1	13	−6	40	55	−2	81	96	−12	113	123	219	145	207	−9	−17	174	−12	−20	−16

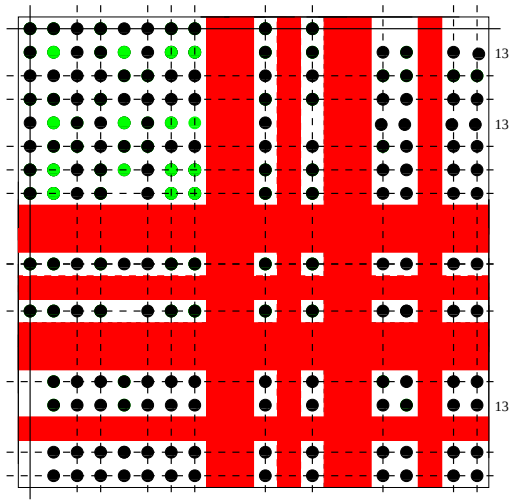
As indicated above, element 6 has been amalgamated into variable 2 giving $IPE(3) = -6$, element 19 has been amalgamated into variable 20 giving $IPE(19) = -20$, element 20 has been amalgamated into variable 16 giving $IPE(20) = -16$, and element 16 has been amalgamated into element 17 giving $IPE(16) = -17$. Furthermore, supervariable 9 has been absorbed into element 12 giving $IPE(9) = -12$, and supervariable 18 has been absorbed into element 12 giving $IPE(18) = -12$.

The next pivot to be eliminated is element 10, which has degree 10. The original value of $IPE(10) = 113$ is changed to $IPE(1) = 223$ and so fill-in has occurred. We have $IW(223) = 4$ (the length of the fill-in list), $IW(224) = 2$, $IW(225) = 7$, $IW(226) = 8$, and $IW(227) = 1$. Again, going through the work-array in order to find the rows which have a non-zero entry in column 10, we obtain $IW(2) = 10$ (row 2), $IW(14) = 10$ (row 2), $IW(82) = 10$ (row 7), and $IW(97) = 10$ (row 8). It is thus observed that the elimination of pivot 10 induces fill-in in rows 1, 2, 7 and 8, where each row has fill-in in columns 1, 2, 7 and 8. Note also the amalgamation of row $11 \rightarrow 2$, row $4 \rightarrow 7$, and row $13 \rightarrow 8$. As before, the corresponding pointer array IPE has been updated to incorporate the current state of the elimination tree, i.e.,



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	1	13	−6	−7	55	−2	81	96	−12	223	−2	219	−8	−10	−9	−17	174	−12	−20	−16

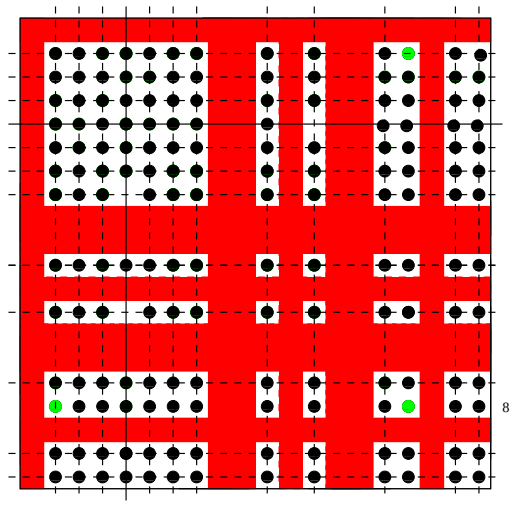
At this stage, element 11 has been amalgamated with element 2 giving $IPE(11) = -2$, element 4 has been amalgamated with element 7 giving $IPE(4) = -7$, and element 13 has been amalgamated with element 8 giving $IPE(13) = -8$. Furthermore, supervariable 14 has been absorbed into supervariable 10 giving $IPE(14) = -10$.



The next variable of minimum degree is 1. The original value of $IPE(1) = 1$ is changed to $IPE(1) = 228$, which is an indication that fill-in has occurred during the elimination of pivot 1. We obtain $IW(228) = 2$ (the length of the fill-in list), $IW(229) = 5$ and $IW(230) = 2$. Going through the work-array, and looking for variables which have the value 1, we observe that $IW(14) = 1$ (row 2) and $IW(56) = 1$ (row 5). It follows that the elimination of variable 1 introduces fill-in in rows/columns 2 and 5. Note the amalgamation of row $7 \rightarrow 8$ and row $8 \rightarrow 5$. The corresponding pointer array, IPE, which has been updated to reflect the current state of the elimination tree, is given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	228	13	−6	−7	55	−2	−8	−5	−12	−1	−2	219	−8	−10	−9	−17	174	−12	−20	−16

At this stage, element 7 has been amalgamated with element 8 giving $IPE(7) = -8$ and element 8 has been amalgamated with element 5 giving $IPE(8) = -5$. Furthermore, supervariable 10 has been absorbed into element 1 giving $IPE(10) = -1$.

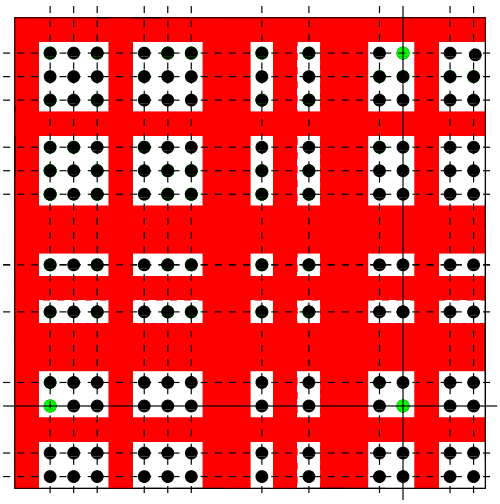


The next pivot of minimum degree is 5. The elimination of this variable introduces fill-in, which can be seen by observing that the original value of $IPE(5) = 55$ has been changed to $IPE(5) = 231$. If we now look at the value of $IPE(231)$ we find that the length of the corresponding list is 1, i.e., $IPE(231) = 1$ and also that $IPE(232) = 17$. Going through the array IW as before, and looking for the value 5, we observe that $IW(175) = 5$ (row 17). It follows that the elimination of variable 5 introduces fill-in in row/column 17. Note the amalgamation of row $2 \rightarrow 17$. The updated pointer array IPE is defined by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPE	−5	−17	−6	−7	231	−2	−8	−5	−12	−1	−2	−5	−8	−10	−9	−17	174	−12	−20	−16

Similar to the above, element 2 has been amalgamated with element 17 giving $IPE(2) = -17$. In addition, supervariable 2 has been absorbed into element 5 giving $IPE(2) = -5$, and supervariable 12 has been absorbed into element 5 giving $IPE(12) = -5$.

The last pivot that needs to be eliminated is element 17, which has degree 11. The elimination of this variable does not induce any fill-in, which is readily verified by the zero value of $IPE(17) = 0$. The final state of the pointer array is given by



i	1	2	3	4	5	6	7	8	9	10
IPE	−5	−17	−6	−7	−17	−2	−8	−5	−12	−1
i	11	12	13	14	15	16	17	18	19	20
IPE	−2	−5	−8	−10	−9	−17	0	−12	−20	−16

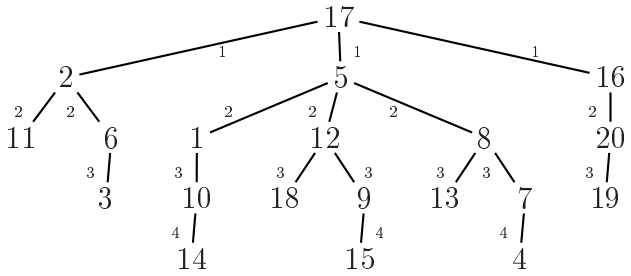
Here, supervariable 5 has been absorbed into element 17 giving $IPE(5) = -17$, and supervariable 17 has been set to null, i.e., $IPE(17) = 0$, indicating that element 17 is the root of the elimination tree. In the next section, we describe how the elimination tree can be constructed from the values stored in the pointer array IPE.

2.2 The depth-first tree search - MA27L

In this section, we investigate how to construct a graphical representation of the elimination tree that is stored in the pointer array IPE in form of son to father tree pointers. Furthermore, we discuss how the depth-first search algorithm is used to construct the permutation vector that is used to permute the rows and columns of the original coefficient matrix $\mathcal{A}$.

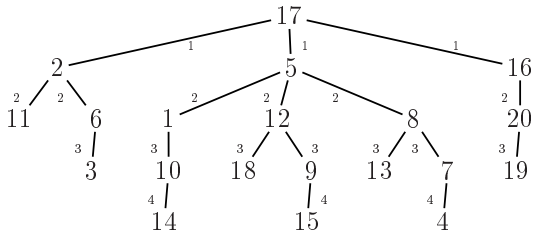
The graphical representation of the elimination tree, which is stored in the array IPE on exit from the subroutine $MA27H$, can easily be obtained by considering the son-father relationship between individual tree nodes. For instance, for the example that is discussed in this article, the entries of IPE indicate that the tree node is 17 ($IPE(17) = 0$). Having established that the root of the elimination tree is 17, the next step is to find all the nodes of the tree that are connected to the root. This is done by searching for the value -17 in the array IPE, since the corresponding indices refer to the sons of the root. In this example, the array entries $IPE(2)$, $IPE(5)$, and $IPE(16)$ all satisfy this condition and thus 2, 5 and 16 are the son nodes of the father/root node 17. Similarly, to find the son(s) of tree node 2, we would search for the value -2 in IPE giving $IPE(11) = -2$ and $IPE(6) = -2$. Thus, the nodes 11 and 6 are sons of the tree node 2. The remaining nodes of the tree are constructed in a similar fashion.

Completing the construction of the graphical representation of the elimination tree, we ascertain



In the graph, the node numbers refer to the pivot/row numbers of the coefficient matrix and the small numbers next to the connections between individual nodes refer to the depth of a particular pivot within the elimination tree. In order to establish the permutation vector, the n-dimensional array IPS is used. The array must not be set on entry into $MA27L$, but during the elimination it is used temporarily to hold $-(eldest\ son\ of\ node\ i)$ if it has one or 0 otherwise. Eventually, it is set to hold the position of node i in the order.

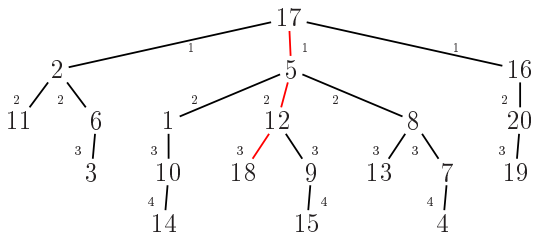
We are now in a position to see how the subroutine $MA27L$ performs the depth-first search using the elimination tree that is constructed by the subroutine $MA27H$. The initial elimination tree and the corresponding entries of arrays IPS and IPE are



i	1	2	3	4	5
IPS/IPE	−10/8	−11/ − 17	0/ − 6	0/ − 7	−12/16
i	6	7	8	9	10
IPS/IPE	−3/ − 2	−4/ − 8	−13/ − 5	−15/ − 12	−14/ − 1
i	11	12	13	14	15
IPS/IPE	0/6	−18/1	0/7	0/ − 10	0/ − 9
i	16	17	18	19	20
IPS/IPE	−20/2	−5/0	0/9	0/ − 20	−19/ − 16

As discussed above, the entries of $IPS(i)$ are initially set to $-(eldest\ son\ of\ node\ i)$. Similarly, on entry the array $IPE(i)$ is set equal to $-(father\ of\ node\ i)$ or zero if the node is the root node. Before the application of the depth-first search algorithm the entries of $IPE(i)$ $(i = 1, \ldots, 20)$ are altered to point to the next younger brother of node i if it has one but otherwise is not changed. Thus, for the above example $IPS(1) = -10$ denotes that node 10 is the eldest son of node 1 and $IPE(1) = 8$ denotes that 8 is the next younger brother of node 1. Similarly, $IPS(2) = -11$ denotes that node 11 is the eldest son of node 2 and $IPE(2) = -17$ implies that node 2 does not have a younger brother. All other relationships are derived in a similar fashion.

For this example, the first pivot that is returned by the depth-first search algorithm is element 18. For step $k = 1$ $(k = 1, \ldots, 20)$ the elimination tree and its corresponding arrays IPS and IPE are given by



i	1	2	3	4	5
IPS/IPE	−10/8	−11/ − 17	0/ − 6	0/ − 7	0/16
i	6	7	8	9	10
IPS/IPE	−3/ − 2	−4/ − 8	−13/ − 5	−15/ − 12	−14/ − 1
i	11	12	13	14	15
IPS/IPE	0/6	0/1	0/7	0/ − 10	0/ − 9
i	16	17	18	19	20
IPS/IPE	−20/2	0/0	1/9	0/ − 20	−19/ − 16

Here, the light gray connections show the path of the depth first search from the root to the corresponding element in the tree. In later graphs the dashed connections refer to elements that have already been eliminated.

We note that the entries of $IPS(5)$, $IPS(12)$ and $IPS(17)$ have all been cleared during the elimination of node 18 and also that $IPS(18) = 1$. The latter assignment indicates that node 18 has position 1 in the reordering. The former assignments clear the father-son pointers in IPS as these are used. This is necessary in order to prevent their reuse at a later stage in the elimination process.

At this stage we introduce four additional arrays NV, NE, NA and ND which have the following purpose during the elimination process:

> Array $NV(i)$ is set to zero if no variables are eliminated at node i and to the degree otherwise. Only leaf nodes can have zero values of $NV(i)$.
>
> Array $NE(i)$ is set to the number of variables eliminated at stage i of the elimination.
>
> Array $NA(i)$ is set to the number of variables assembled at stage i of the elimination.
>
> Array $ND(i)$ is set to the degree at stage i of the elimination.

The initial degree count and the values of the above arrays at stage one are

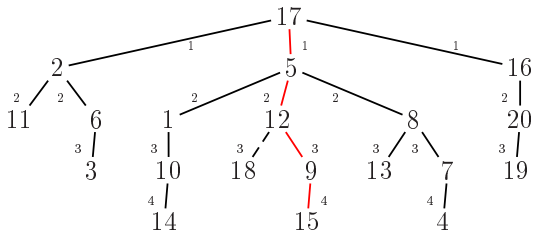
Degree	Variables
2	14
3	9 15
7	12
9	17 20
10	5 10 16 19
11	1
12	2
14	8 13
15	3 4 7 11
16	6

```
NV ->  10 0 0 0 13 0 0 0 3 10 0 9 0 2 3 0 8 5 0 0
NE ->   1 0 0 0  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
NA ->   0 0 0 0  0 0 0 0 0  0 0 0 0 0 0 0 0 1 0 0
ND ->   5 0 0 0  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
```

In the context of the current elimination stage, the arrays NV, NE, NA and ND can be read as follows. From array NV we observe that node 18 has degree 5 ($NV(18) = 5$), and from arrays NE and ND that, at stage 1, one element is eliminated, where this

variable has degree 5 (i.e., $ND(1) = 5$). We also observe that no elements are assembled at stage 1, i.e., $NA(1) = 0$.

At the second stage of the elimination process the depth-first search backtracks one and chooses element 15 as the next pivot, giving



i	1	2	3	4	5
IPS/IPE	$-10/8$	$-11/-17$	$0/-6$	$0/-7$	$0/16$
i	6	7	8	9	10
IPS/IPE	$-3/-2$	$-4/-8$	$-13/-5$	$0/-12$	$-14/-1$
i	11	12	13	14	15
IPS/IPE	$0/6$	$0/1$	$0/7$	$0/-10$	$2/-9$
i	16	17	18	19	20
IPS/IPE	$-20/2$	$0/0$	$1/9$	$0/-20$	$-19/-16$

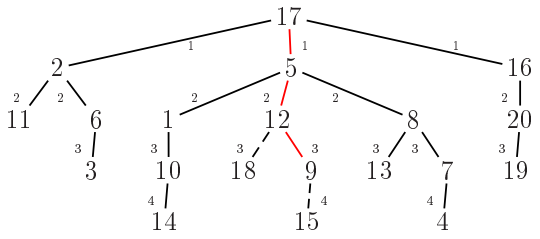
We observe that the entry of $IPS(9)$ has been cleared during the elimination and that $IPS(15) = 2$ indicates that node 15 has position 2 in the reordering. The initial degree count and the values of the arrays NV, NE, NA and ND are

Degree	Variables
2	14
3	9
7	12
9	17 20
10	5 10 16 19
11	1
12	2
14	8 13
15	3 4 7 11
16	6

```
NV ->  10 0 0 0 13 0 0 0 3 10 0 9 0 2 3 0 8 5 0 0
NE ->   1 1 0 0  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
NA ->   0 0 0 0  0 0 0 0 0  0 0 0 0 0 0 0 1 1 0 0
ND ->   5 3 0 0  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
```

As before, we observe from array NV that node 15 has degree 3 ($NV(15) = 3$), and from arrays NE and ND that, at stage 2, one element is eliminated, where this variable has degree 3 (i.e., $ND(2) = 3$). We also note that no elements are assembled at stage 2, i.e., $NA(2) = 0$.

At the third stage of the elimination process element 9 is chosen as the next pivot, giving



i	1	2	3	4	5
IPS/IPE	$-10/8$	$-11/-17$	$0/-6$	$0/-7$	$0/16$
i	6	7	8	9	10
IPS/IPE	$-3/-2$	$-4/-8$	$-13/-5$	$3/-12$	$-14/-1$
i	11	12	13	14	15
IPS/IPE	$0/6$	$0/1$	$0/7$	$0/-10$	$2/-9$
i	16	17	18	19	20
IPS/IPE	$-20/2$	$0/0$	$1/9$	$0/-20$	$-19/-16$

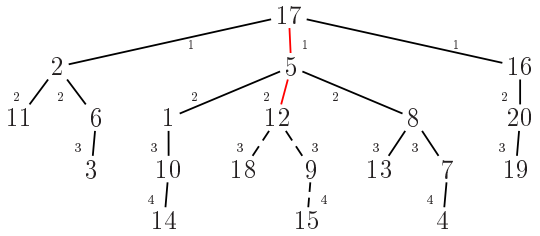
At the current stage, the entry $IPS(9)$ has been changed to $IPS(9) = 3$ indicating that node 9 has position 3 in the reordering. Furthermore, we ascertain

Degree	Variables
2	14
7	12
9	17 20
10	5 10 16 19
11	1
12	2
14	8 13
15	3 4 7 11
16	6

```
NV ->  10 0 0 0 13 0 0 0 3 10 0 9 0 2 3 0 8 5 0 0
NE ->   1 1 1 0  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
NA ->   0 0 1 0  0 0 0 0 0  0 0 0 0 0 0 0 1 2 0 0
ND ->   5 3 3 0  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
```

Here, the entry $NV(9) = 3$ indicates that the current pivot has degree 3 and the array entries $NE(3) = 1$, $NA(3) = 1$, and $ND(3) = 3$ imply that one variable has been eliminated, one element has been assembled at the current stage (node 15), and $ND(3) = 3$ verifies that this element has degree 3.

The next pivot to be chosen is element 12, which is indicated in the following elimination tree.



i	1	2	3	4	5
IPS/IPE	$-10/8$	$-11/-17$	$0/-6$	$0/-7$	$0/16$
i	6	7	8	9	10
IPS/IPE	$-3/-2$	$-4/-8$	$-13/-5$	$3/-12$	$-14/-1$
i	11	12	13	14	15
IPS/IPE	$0/6$	$4/1$	$0/7$	$0/-10$	$2/-9$
i	16	17	18	19	20
IPS/IPE	$-20/2$	$0/0$	$1/9$	$0/-20$	$-19/-16$

Here, the only entry of IPS that needs to be updated is $IPS(12)$. The assignment $IPS(12) = 4$ indicates, as before, that the current node has position 4 in the reordering. The corresponding degree counts, elimination and assembly information is given below.

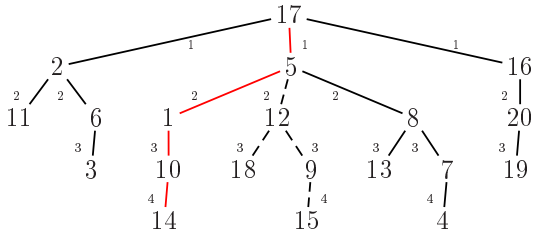
Degree	Variables
2	14
9	17 20
10	5 10 16 19
11	1
12	2
14	8 13
15	3 4 7 11
16	6

```
NV ->  10 0 0 0 13 0 0 0 3 10 0 9 0 2 3 0 8 5 0 0
NE ->   1 1 1 1  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
NA ->   0 0 1 2  0 0 0 0 0  0 0 0 0 0 0 0 1 0 1 0
ND ->   5 3 3 9  0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
```

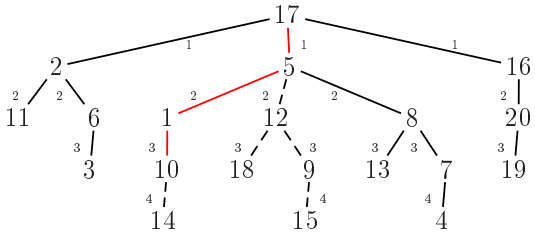
We note that $NA(4) = 2$ indicates that two elements have been assembled at the current stage. Consulting the corresponding elimination tree above, we observe that the two elements that have been assembled at this stage are those resulting from nodes 18 and 9. Also, $IPS(12) = 4$ indicates that node 12 has position 4 in the reordering.

After the elimination of node 12, the depth-first search algorithm backtracks one stage and continues with one of the brother nodes of node 12. Since the overall procedure is similar to what we just observed, we shall, for the remainder of the elimination process, only give the elimination trees. As before, the light grey lines indicate the current pivot that is eliminated and black dashed lines refer to already eliminated nodes.

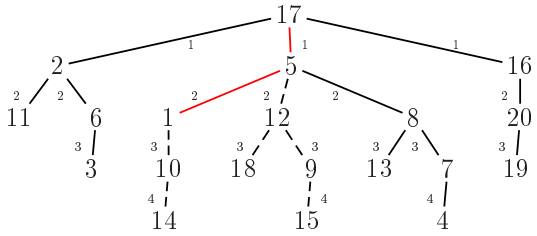
Stage 5



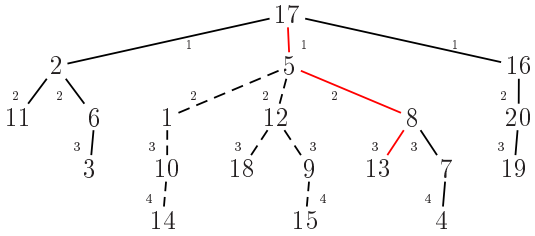
Stage 6



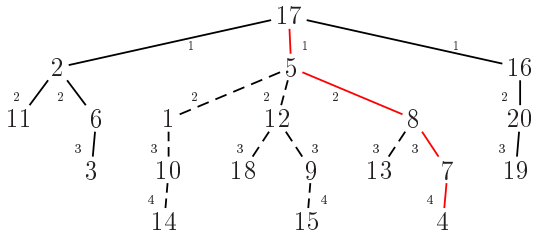
Stage 7



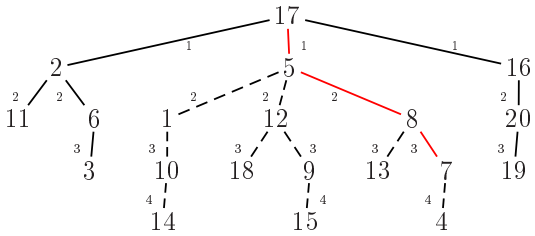
Stage 8



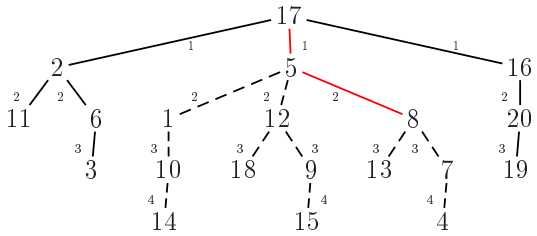
Stage 9



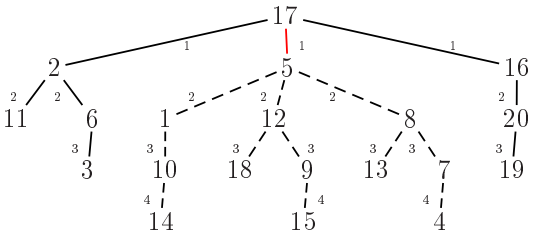
Stage 10



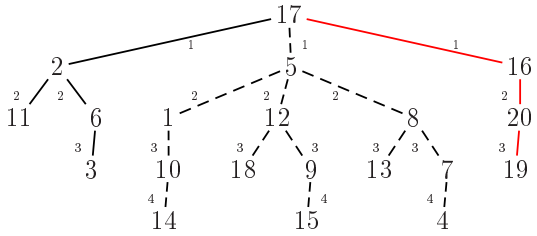
Stage 11



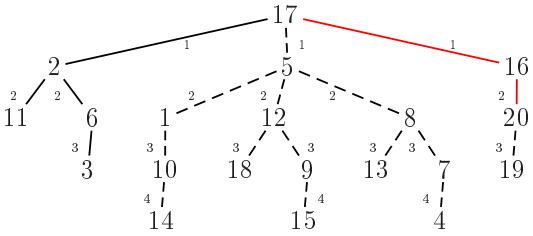
Stage 12



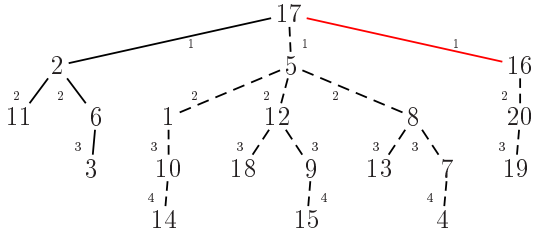
Stage 13



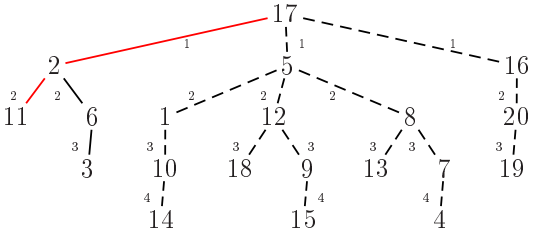
Stage 14



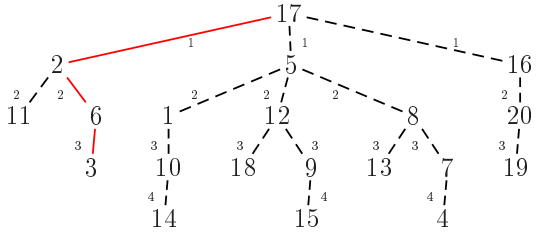
Stage 15



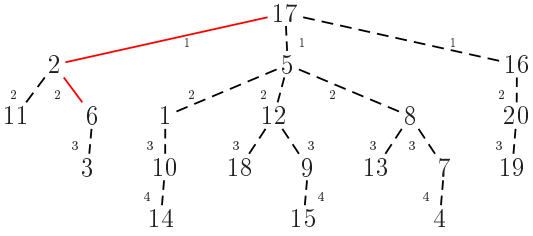
Stage 16



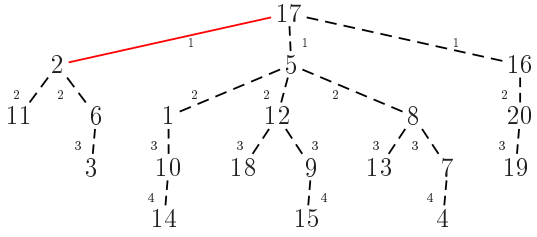
Stage 17



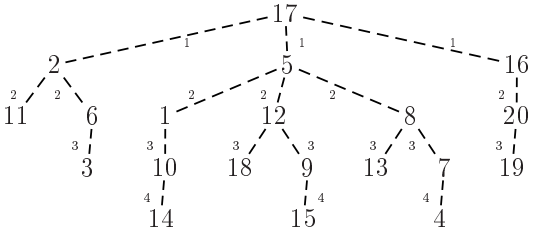
Stage 18



Stage 19



Stage 20



At stage 20, all nodes in the elimination tree have been eliminated. This is indicated by the black dashed lines in the graph. The final permutation vector IPS is given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPS	7	19	17	9	12	18	10	11	3	6	16	4	8	5	2	15	20	1	13	14

This completes the depth-first search algorithm as implemented by subroutine $MA27L$. In the next section, we discuss how the information constructed in the analysis phase of the MA27 algorithm is applied in order to implement an efficient (numerical) factorisation strategy.

3 The numerical factorisation

3.1 Sorting prior to factorisation - MA27N

In this section, we consider the $MA27$ sorting routine $MA27N$. $MA27N$ uses the permutation vector IPS to permute the rows and columns of the coefficient matrix $\mathcal{A}$. For the example we consider in this article, the numerical values of the coefficient matrix are given by the following matrix. Note that the entries in gray refer to the $(1, 1)$ block which has size 13×13.

$\mathcal{A} =$

.8024	−.0234	−.0471	−.0170	.0064	−.0006	−.0581	−.0036		.0307	−.0393		−.0025	1.0000						
	.8197	−.0212	−.0010	−.0084	−.0004	−.0034	.0566		.0138	.0482		−.0041		.8313					
		.7631	−.0198		−.0009	−.0161			.1094	−.0004		−.0019			.8723	−.0041	−.0792	.0050	−.0009
			.8146	−.0078	−.0003	−.0107	−.0624		.0129	.0517		−.0046			−.0041	.8467		.0270	−.0015
				.8795	.0001	.0022				−.0082					−.0792		.8741	−.0050	
					.8357	−.0011	−.0001		.0006	−.0007					.0050	.0270	−.0050	.8857	.0057
						.9527	.0127		.0105	−.0134		−.0008			−.0009	−.0015		.0057	.8763
							.9515			.0024		−.0003			.0004	−.0346		−.0024	.0001
								.6739			.0033			.0289					
									.8593	.0003		.0013							
										.7518		.0064			−.0064	.0029		.0422	−.0023
											.7502				−.0374	.0049	−.0391	−.0053	.0010
												.7213			−.0007	.0206		.0048	−.0087
													.0000						
														.0000					
															.0000				
																.0000			
																	.0000		
																		.0000	
																			.0000

As indicated above, the purpose of the $MA27N$ sorting routine is to reorder the user's input, i.e., the original coefficient matrix, so that the upper triangle of the permuted matrix, including the diagonal, is held ordered by rows at the end of the storage arrays A and IW. In fact, on entry into $MA27N$, the array $A(i)$ holds the value of the original matrix element in position $(IRN(i), ICN(i)) \quad (i = 1, \ldots, nz)$, and, on exit,

$A(LA - NZ1 + i)$ $(i = 1, \ldots, NZ1)$ holds the upper triangle of the permuted matrix by rows with the diagonal entry although there is no further ordering within the rows themselves. Here, the array $IRN(i)$ is set to hold the row index of entry $A(i)$, the array $ICN(i)$ holds the column index of entry $A(i)$, LA is the length of array A, and $NZ1$ is, on exit, the number of entries in the sorted matrix. Similarly, the array IW is used as a workspace and on exit the entries $IW(LIW - NZ1 + i)$ $(i = 1, \ldots, NZ1)$ hold the column indices (the original unpermuted indices) of the corresponding entries of A with the first entry for each row flagged negative. Here, LIW is the length of the array IW.

For the above example, the relevant entries of A and IW are respectively given by

$A(530-534)$	0.00000	−0.03913	−0.07919	−0.00495	0.87415
$A(535-539)$	0.00000	0.83125	0.02893	0.67390	0.00335
$A(540-544)$	0.75022	0.00103	−0.00527	0.00486	−0.03742
$A(545-549)$	0.00000	1.00000	0.85930	−0.00001	0.00126
$A(550-554)$	0.01290	0.03075	0.00058	0.01051	0.01384
$A(555-559)$	0.10941	0.00027	0.80244	−0.00245	−0.03926
$A(560-564)$	−0.00356	0.00638	−0.05811	−0.00063	−0.01703
$A(565-569)$	−0.04711	−0.02337	0.72129	−0.00871	0.00478
$A(570-574)$	0.02059	−0.00072	−0.00027	−0.00192	−0.00455
$A(575-579)$	−0.00405	−0.00084	0.00635	−0.00005	0.81461
$A(580-584)$	0.05174	−0.00101	−0.00409	0.84672	0.02700
$A(585-589)$	−0.00149	−0.00783	−0.00032	−0.01976	−0.01074
$A(590-594)$	−0.06242	0.95266	−0.00087	−0.00149	0.00572
$A(595-599)$	−0.00110	0.01270	0.00218	−0.01610	0.87634
$A(600-604)$	−0.00340	−0.01342	0.95150	−0.00235	0.00237
$A(605-609)$	0.00036	0.00013	−0.03458	−0.00007	0.00002
$A(610-614)$	0.05658	0.87952	−0.07919	−0.00838	−0.00495
$A(615-619)$	0.00012	−0.00820	0.00000	0.04222	0.00498
$A(620-624)$	0.88567	0.00000	−0.00233	0.00572	−0.00087
$A(625-629)$	0.00000	−0.00640	0.87233	0.00498	0.75182
$A(630-634)$	0.00294	−0.00074	0.04822	−0.00041	0.76310
$A(635-639)$	−0.00409	−0.02121	−0.00089	0.83568	0.02700
$A(640-644)$	−0.00044	0.81969	0.00000		

$IW(163-167)$	−18	12	3	6	5
$IW(168-172)$	−15	2	9	−9	12
$IW(173-177)$	−12	20	19	17	16
$IW(178-182)$	−14	1	−10	8	13
$IW(183-187)$	4	1	6	7	2
$IW(188-192)$	3	11	−1	13	11
$IW(193-197)$	8	5	7	6	4
$IW(198-202)$	3	2	−13	20	19
$IW(203-207)$	17	16	8	3	4
$IW(208-212)$	2	7	11	6	−4
$IW(213-217)$	11	2	16	17	19
$IW(218-222)$	20	5	6	3	7
$IW(223-227)$	8	−7	16	17	19
$IW(228-232)$	6	8	5	3	20
$IW(233-237)$	2	11	−8	19	11

$$
\begin{array}{lrrrrr}
IW(238-242) & 16 & 20 & 17 & 6 & 3 \\
IW(243-247) & 2 & -5 & 16 & 2 & 19 \\
IW(248-252) & 6 & 11 & -19 & 11 & 3 \\
IW(253-257) & 6 & -20 & 11 & 6 & 3 \\
IW(258-262) & -16 & 11 & 3 & 6 & -11 \\
IW(263-267) & 17 & 6 & 2 & 3 & -3 \\
IW(268-272) & 17 & 2 & 6 & -6 & 17 \\
IW(273-277) & 2 & -2 & -17 & &
\end{array}
$$

As already mentioned above, negative entries in the array IW refer to the first entry in each row, whereas nonnegative entries refer to the original unpermuted column indices. Hence, for the above arrays IW and A we ascertain the following reordered coefficient matrix.

$$
\begin{array}{lrr}
\text{Row 1} & (18,18)\rightarrow & 0.00000 \\
 & (18,12)\rightarrow & -0.03913 \\
 & (18,3)\ \rightarrow & -0.07919 \\
 & (18,6)\ \rightarrow & -0.00495 \\
 & (18,5)\ \rightarrow & 0.87415
\end{array}
\rightarrow
\begin{array}{rrrr}
-18 & 12 & 3 & 6 \\
 & 5 & &
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 2} & (15,15)\rightarrow & 0.00000 \\
 & (15,2)\ \rightarrow & 0.83125 \\
 & (15,9)\ \rightarrow & 0.02893
\end{array}
\rightarrow
\begin{array}{rrr}
-15 & 2 & 9
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 3} & (9,9)\ \rightarrow & 0.67390 \\
 & (9,12)\ \rightarrow & 0.00335
\end{array}
\rightarrow
\begin{array}{rr}
-9 & 12
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 4} & (12,12)\rightarrow & 0.75022 \\
 & (12,20)\rightarrow & 0.00103 \\
 & (12,19)\rightarrow & -0.00527 \\
 & (12,17)\rightarrow & 0.00486 \\
 & (12,16)\rightarrow & -0.03742
\end{array}
\rightarrow
\begin{array}{rrrr}
-12 & 20 & 19 & 17 \\
 & 16 & &
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 5} & (14,14)\rightarrow & 0.00000 \\
 & (14,1)\ \rightarrow & 1.00000
\end{array}
\rightarrow
\begin{array}{rr}
-14 & 1
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 6} & (10,10)\rightarrow & 0.85930 \\
 & (10,8)\ \rightarrow & -0.00001 \\
 & (10,13)\rightarrow & 0.00126 \\
 & (10,4)\ \rightarrow & 0.01290 \\
 & (10,1)\ \rightarrow & 0.03075 \\
 & (10,6)\ \rightarrow & 0.00058 \\
 & (10,7)\ \rightarrow & 0.01051 \\
 & (10,2)\ \rightarrow & 0.01384 \\
 & (10,3)\ \rightarrow & 0.10941 \\
 & (10,11)\rightarrow & 0.00027
\end{array}
\rightarrow
\begin{array}{rrrr}
-10 & 8 & 13 & 4 \\
 & 1 & 6 & 7 \\
 & 2 & 3 & 11
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 7} & (1,1)\ \rightarrow & 0.80244 \\
 & (1,13)\ \rightarrow & -0.00245 \\
 & (1,11)\ \rightarrow & -0.03926 \\
 & (1,8)\ \rightarrow & -0.00356 \\
 & (1,5)\ \rightarrow & 0.00638 \\
 & (1,7)\ \rightarrow & -0.05811 \\
 & (1,6)\ \rightarrow & -0.00063 \\
 & (1,4)\ \rightarrow & -0.01703 \\
 & (1,3)\ \rightarrow & -0.04711 \\
 & (1,2)\ \rightarrow & -0.02337
\end{array}
\rightarrow
\begin{array}{rrrr}
-1 & 13 & 11 & 8 \\
 & 5 & 7 & 6 \\
 & 4 & 3 & 2
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 8} & (13,13)\rightarrow & 0.72129 \\
 & (13,20)\rightarrow & -0.00871 \\
 & (13,19)\rightarrow & 0.00478 \\
 & (13,17)\rightarrow & 0.02059 \\
 & (13,16)\rightarrow & -0.00072 \\
 & (13,18)\rightarrow & -0.00027 \\
 & (13,3)\ \rightarrow & -0.00192 \\
 & (13,4)\ \rightarrow & -0.00455 \\
 & (13,2)\ \rightarrow & -0.00405 \\
 & (13,7)\ \rightarrow & -0.00084 \\
 & (13,11)\rightarrow & 0.00635 \\
 & (13,6)\ \rightarrow & -0.00005
\end{array}
\rightarrow
\begin{array}{rrrr}
-13 & 20 & 19 & 17 \\
 & 16 & 8 & 3 \\
 & 4 & 2 & 7 \\
 & 11 & 6 &
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 9} & (4,4)\ \rightarrow & 0.81461 \\
 & (4,11)\ \rightarrow & 0.05174 \\
 & (4,2)\ \rightarrow & -0.00101 \\
 & (4,16)\ \rightarrow & -0.00409 \\
 & (4,17)\ \rightarrow & 0.84672 \\
 & (4,19)\ \rightarrow & 0.02700 \\
 & (4,20)\ \rightarrow & -0.00149 \\
 & (4,5)\ \rightarrow & -0.00783 \\
 & (4,6)\ \rightarrow & -0.00032 \\
 & (4,3)\ \rightarrow & -0.01976 \\
 & (4,7)\ \rightarrow & -0.01074 \\
 & (4,8)\ \rightarrow & -0.06242
\end{array}
\rightarrow
\begin{array}{rrrr}
-4 & 11 & 2 & 16 \\
 & 17 & 19 & 20 \\
 & 5 & 6 & 3 \\
 & 7 & 8 &
\end{array}
$$

$$
\begin{array}{lrr}
\text{Row 10} & (7,7)\ \rightarrow & 0.95266 \\
 & (7,16)\ \rightarrow & -0.00087 \\
 & (7,17)\ \rightarrow & -0.00149 \\
 & (7,19)\ \rightarrow & 0.00572 \\
 & (7,6)\ \rightarrow & -0.00110 \\
 & (7,8)\ \rightarrow & 0.01270 \\
 & (7,5)\ \rightarrow & 0.00218 \\
 & (7,3)\ \rightarrow & -0.01610 \\
 & (7,20)\ \rightarrow & 0.87634 \\
 & (7,2)\ \rightarrow & -0.00340 \\
 & (7,11)\ \rightarrow & -0.01342
\end{array}
\rightarrow
\begin{array}{rrrr}
-7 & 16 & 17 & 19 \\
 & 6 & 8 & 5 \\
 & 3 & 20 & 2 \\
 & 11 & &
\end{array}
$$

$$
\begin{array}{lll}
\text{Row 11} & \left.\begin{array}{ll}(8,8) & \to \quad 0.95150\\ (8,19) & \to -0.00235\\ (8,11) & \to \quad 0.00237\\ (8,16) & \to \quad 0.00036\\ (8,20) & \to \quad 0.00013\\ (8,17) & \to -0.03458\\ (8,6) & \to -0.00007\\ (8,3) & \to \quad 0.00002\\ (8,2) & \to \quad 0.05658\end{array}\right\} & \to \begin{array}{cccc} -8 & 19 & 11 & 16\\ & 20 & 17 & 6\\ & 3 & 2 & \end{array}\\
\text{Row 12} & \left.\begin{array}{ll}(5,5) & \to \quad 0.87952\\ (5,16) & \to -0.07919\\ (5,2) & \to -0.00838\\ (5,19) & \to -0.00495\\ (5,6) & \to \quad 0.00012\\ (5,11) & \to -0.00820\end{array}\right\} & \to \begin{array}{cccc} -5 & 16 & 2 & 19\\ & 6 & 11 & \end{array}\\
\text{Row 13} & \left.\begin{array}{ll}(19,19) & \to \quad 0.00000\\ (19,11) & \to \quad 0.04222\\ (19,3) & \to \quad 0.00498\\ (19,6) & \to \quad 0.88567\end{array}\right\} & \to -19 \quad 11 \quad 3 \quad 6\\
\text{Row 14} & \left.\begin{array}{ll}(20,20) & \to \quad 0.00000\\ (20,11) & \to -0.00233\\ (20,6) & \to \quad 0.00572\\ (20,3) & \to -0.00087\end{array}\right\} & \to -20 \quad 11 \quad 6 \quad 3
\end{array}
$$

$$
\begin{array}{lll}
\text{Row 15} & \left.\begin{array}{ll}(16,16) & \to \quad 0.00000\\ (16,11) & \to -0.00640\\ (16,3) & \to \quad 0.87233\\ (16,6) & \to \quad 0.00498\end{array}\right\} & \to -16 \quad 11 \quad 3 \quad 6\\
\text{Row 16} & \left.\begin{array}{ll}(11,11) & \to \quad 0.75182\\ (11,17) & \to \quad 0.00294\\ (11,6) & \to -0.00074\\ (11,2) & \to \quad 0.04822\\ (11,3) & \to -0.00041\end{array}\right\} & \to \begin{array}{cccc} -11 & 17 & 6 & 2\\ & & 3 & \end{array}\\
\text{Row 17} & \left.\begin{array}{ll}(3,3) & \to \quad 0.76310\\ (3,17) & \to -0.00409\\ (3,2) & \to -0.02121\\ (3,6) & \to -0.00089\end{array}\right\} & \to -3 \quad 17 \quad 2 \quad 6\\
\text{Row 18} & \left.\begin{array}{ll}(6,6) & \to \quad 0.83568\\ (6,17) & \to \quad 0.02700\\ (6,2) & \to -0.00044\end{array}\right\} & \to -6 \quad 17 \quad 2\\
\text{Row 19} & (2,2) \quad \to \quad 0.81969 \} & \to -2\\
\text{Row 20} & (17,17) \to \quad 0.00000 \} & \to -17
\end{array}
$$

In order to check the correctness of the above (reordered) array representation, we may consider the tentative pivots that we obtained from performing the minimum degree algorithm of Section 2.1, and the depth-first search algorithm of Section 2.2. We obtained the ordering

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IPS	7	19	17	9	12	18	10	11	3	6	16	4	8	5	2	15	20	1	13	14

Considering the individual entries of array IPS, we establish that row/column 18 has been swapped to become row/column 1, row/column 15 has been swapped to become row/column 2, etc. Overall, this defines the vector PER which is given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
PER	18	15	9	12	14	10	1	13	4	7	8	5	19	20	16	11	3	6	2	17

Comparing $\mathcal{A}(PER, PER)$ with the entries of the arrays A and IW, we conclude that the internal (reordered) representation of $\mathcal{A}$ within the MA27 data structure is identical to the one defined by $\mathcal{A}(PER, PER)$; however, we remark that the column indices stored in the array IW are those of the unordered matrix $\mathcal{A}$, and not of $\mathcal{A}(PER, PER)$.

3.2 The actual numerical factorisation - MA27O

On entry into the factorisation routine $MA27O$, the upper triangular representation of the permuted matrix is copied from the end of the array A to the front, and the relevant entries at the end of array IW are copied to the front of IW. Thus, we have

$A(001-005)$	0.00000	−0.03913	−0.07919	−0.00495	0.87415
$A(006-010)$	0.00000	0.83125	0.02893	0.67390	0.00335
$A(011-015)$	0.75022	0.00103	−0.00527	0.00486	−0.03742
$A(016-020)$	0.00000	1.00000	0.85930	−0.00001	0.00126
$A(021-025)$	0.01290	0.03075	0.00058	0.01051	0.01384
$A(026-030)$	0.10941	0.00027	0.80244	−0.00245	−0.03926
$A(031-035)$	−0.00356	0.00638	−0.05811	−0.00063	−0.01703
$A(036-040)$	−0.04711	−0.02337	0.72129	−0.00871	0.00478
$A(041-045)$	0.02059	−0.00072	−0.00027	−0.00192	−0.00455
$A(046-050)$	−0.00405	−0.00084	0.00635	−0.00005	0.81461
$A(051-055)$	0.05174	−0.00101	−0.00409	0.84672	0.02700
$A(056-060)$	−0.00149	−0.00783	−0.00032	−0.01976	−0.01074
$A(061-065)$	−0.06242	0.95266	−0.00087	−0.00149	0.00572
$A(066-070)$	−0.00110	0.01270	0.00218	−0.01610	0.87634
$A(071-075)$	−0.00340	−0.01342	0.95150	−0.00235	0.00237
$A(076-080)$	0.00036	0.00013	−0.03458	−0.00007	0.00002
$A(081-085)$	0.05658	0.87952	−0.07919	−0.00838	−0.00495
$A(086-090)$	0.00012	−0.00820	0.00000	0.04222	0.00498
$A(091-095)$	0.88567	0.00000	−0.00233	0.00572	−0.00087
$A(096-100)$	0.00000	−0.00640	0.87233	0.00498	0.75182
$A(101-105)$	0.00294	−0.00074	0.04822	−0.00041	0.76310
$A(106-110)$	−0.00409	−0.02121	−0.00089	0.83568	0.02700
$A(111-115)$	−0.00044	0.81969	0.00000	0.00000	0.00000

$IW(001-005)$	−18	12	3	6	5
$IW(006-010)$	−15	2	9	−9	12
$IW(011-015)$	−12	20	19	17	16
$IW(016-020)$	−14	1	−10	8	13
$IW(021-025)$	4	1	6	7	2
$IW(026-030)$	3	11	−1	13	11
$IW(031-035)$	8	5	7	6	4
$IW(036-040)$	3	2	−13	20	19
$IW(041-045)$	17	16	8	3	4
$IW(046-050)$	2	7	11	6	−4
$IW(051-055)$	11	2	16	17	19
$IW(056-060)$	20	5	6	3	7
$IW(061-065)$	8	−7	16	17	19
$IW(066-070)$	6	8	5	3	20
$IW(071-075)$	2	11	−8	19	11
$IW(076-080)$	16	20	17	6	3
$IW(081-085)$	2	−5	16	2	19
$IW(086-090)$	6	11	−19	11	3
$IW(091-095)$	6	−20	11	6	3
$IW(096-100)$	−16	11	3	6	−11
$IW(101-105)$	17	6	2	3	−3
$IW(106-110)$	17	2	6	−6	17
$IW(111-115)$	2	−2	−17	11	13

There are three distinctive parts within the $MA27O$ subroutine which can be summarised as follows:

(i) The assembly of the frontal matrix from (possible) stack elements, i.e., factorised rows from a previous elimination step, and entries from original rows. This process can be visualised by considering Figure 5.

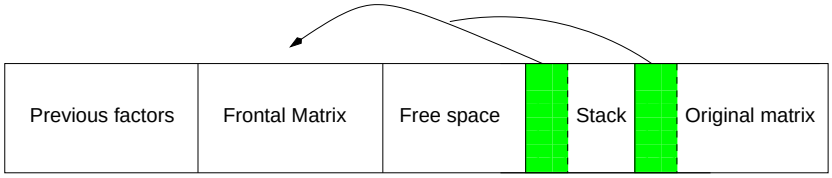


Figure 5: Assembly of the frontal matrix.

(ii) After the assembly of the frontal matrix, the next step in the elimination process is the actual application of the Gaussian elimination process. Potential pivots are taken from the rows and columns of the frontal matrix, i.e., the shaded region in Figure 6.

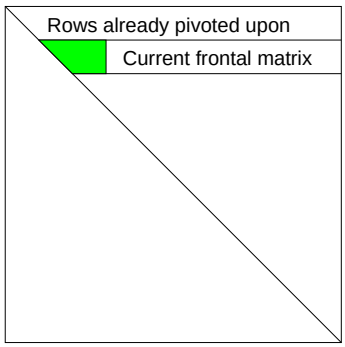


Figure 6: The frontal matrix at an intermediate stage.

(iii) The final part in the elimination process is the placement of the remainder of the current front back on the stack. This process is visualised in Figure 7.

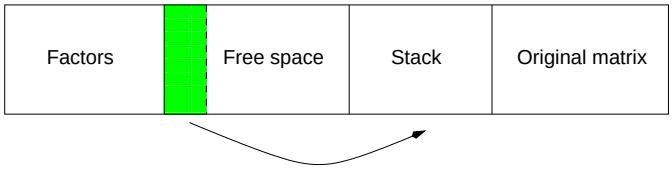


Figure 7: Placing remainder of the front on the stack.

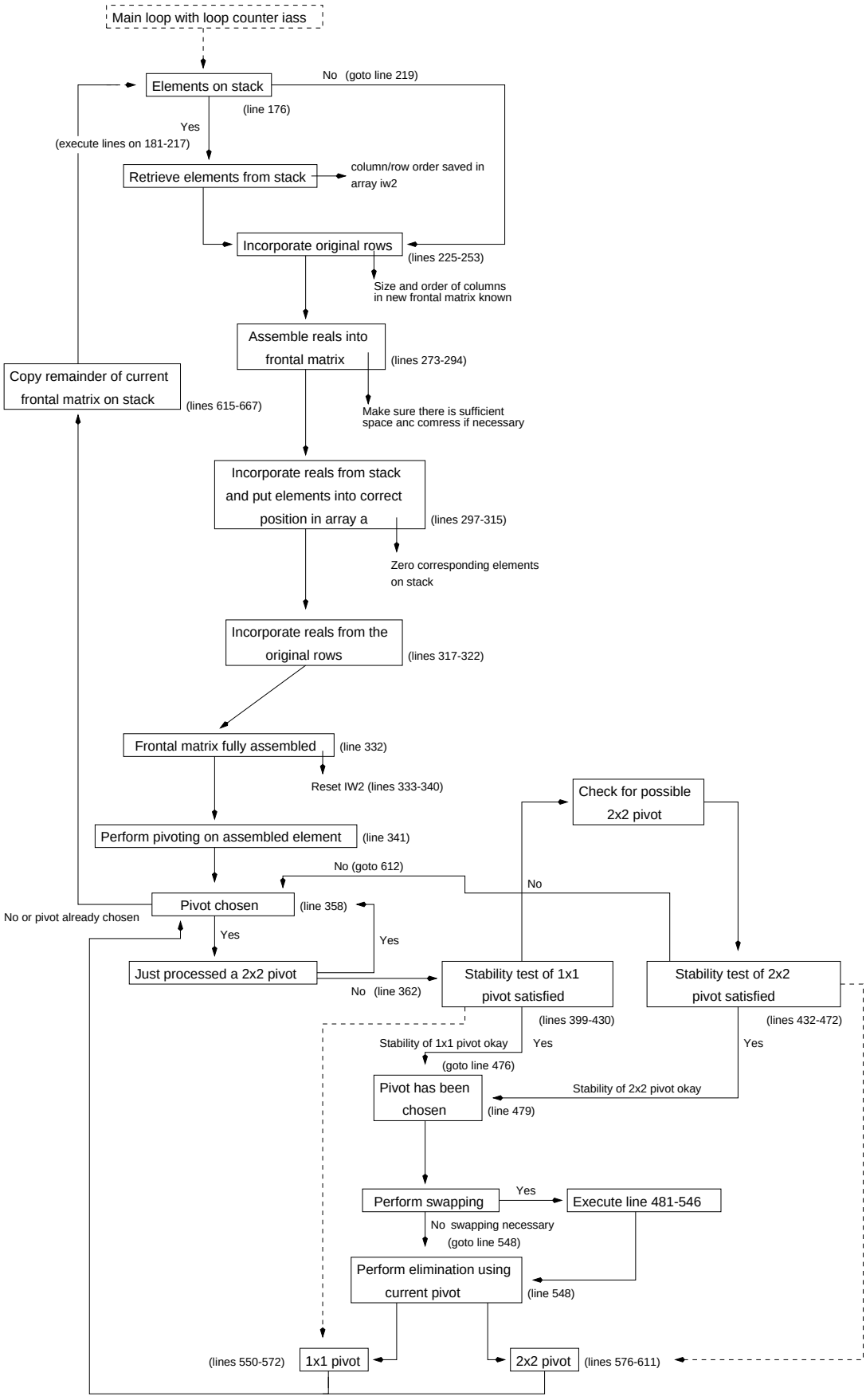


Figure 8: Execution order of the MA27OD factorisation routine.

In order for us to understand the above structure, we shall consider the above factorisation structures for the first few elimination steps of the 20×20 matrix defined in Section 3.1. In Figure 8, we give a flow-chart outlining the internal workings of the factorisation routine.

3.2.1 Overall process at stage one of the elimination process

The actual factorisation process starts on line 155 with a loop of the form

```
    do 760 iass=1,nsteps
      .
      . (see original code for details)
      .
760 continue
```

Since $iass = 1$ at the current stage, nothing has been put on the stack and therefore no stack elements are being assembled at this stage (line 176), i.e.,

```
    if (numstk.eq.0) go to 80
```

During the symbolic assembly, we hold the column indices so far placed in the front in a linked list in the array $IW2$ (lines 223-253).

```
80  numorg=nelim(iass)
    j1 = -iw(j1)
    do 150 iorg=1, numorg
       .
       . (see original code for details)
       .
150 continue
```

At stage one, the entries of array $IW2$ before line 225 are given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

whereas the same array has the entries

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	0	6	0	3	21	0	0	0	0	0	5	0	0	0	0	0	12	0	0

after line 253, thus indicating that columns 18, 12, 5, 3 and 6 have been assembled into the current frontal matrix. Looking at the initial coefficient matrix in Figure 2, we can verify that row 18 has non-zero entries in columns 3, 5, 6 and 12, and therefore the elimination of pivot 18 involves updates in those columns.

The next step in the construction of the frontal matrix is the assembly of the reals into the frontal matrix. We first make sure that there is sufficient space in the arrays to

$a(1) - a(15)$ before line 276

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	−.03913	−.07919	−.00495	.87415	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
.83125	.02893	.67390	.00335	.75022	.00103
$a(13)$	$a(14)$	$a(15)$	$a(16)$	$a(17)$	$a(18)$
−.00527	.00486	−.03742			

$a(1) - a(15)$ after line 295

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(13)$	$a(14)$	$a(15)$	$a(16)$	$a(17)$	$a(18)$
.00000	.00000	.00000			

hold the new frontal matrix and perform a simple compress using $MA27P$ if necessary (lines 276-295).

However, at this stage no stack elements are being assembled and so execution is transferred to line 317.

The final step in the construction of the frontal matrix is the incorporation of the reals from the original rows (lines $317 - 332$). The integer and real information is copied directly into place as indicated by Figure 5. We notice that during this assembly, the entries of the linked list are converted into relative positions during the integer copy to facilitate the move of the reals.

$a(1) - a(15)$ before line 317

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(13)$	$a(14)$	$a(15)$	$a(16)$	$a(17)$	$a(18)$
.00000	.00000	.00000			

$a(1) - a(15)$ after line 332

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	−.03913	.87415	−.07919	−.00495	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(13)$	$a(14)$	$a(15)$	$a(16)$	$a(17)$	$a(18)$
.00000	.00000	.00000			

This completes the assembly of the current frontal matrix, where the array entries $a(1) - a(15)$ after line 332 correspond to the front

$$\begin{bmatrix} .00000 & -.03913 & .87415 & -.07919 & -.00495 \\ & .00000 & .00000 & .00000 & .00000 \\ & & .00000 & .00000 & .00000 \\ & & & .00000 & .00000 \\ & & & & .00000 \end{bmatrix}.$$

After resetting the array $IW2$ and the variable $numass$ (lines 333-340), which is the number of accumulated rows assembled so far, we are now in a position to choose pivots in the frontal matrix and perform the eliminations. The maximum number of pivots is the number of tentative eliminations plus the number of variables in stacked elements which were not previously eliminated because of stability considerations. Since we assembled in tentative pivot order, these rows and columns come first in the corresponding data structure (lines 341-364).

If the user has indicated that the matrix is definite by setting $u \leq .00000$, we do not check for stability but perform the eliminations as long as the pivot is non-zero and has not changed sign. A zero pivot, when the matrix has been declared definite, raises an error condition as does a change in sign when u is set to zero. When u is set negative, changes in sign are flagged and a warning is printed but the decomposition continues (lines 365-398).

The frontal matrix—at an intermediate stage—has the form shown in Figure 6. When $u > 0$, we test the next potential pivot for stability in the following manner. We first search the whole of the potential pivot row determining the largest entry in the potential pivot columns (shaded region) and the largest entry of the rest (unshaded region). The largest entry to the right of the diagonal in the row of prospective pivots in the fully-summed part of the matrix is determined, and we also record the column of this entry (lines 399-410). We do the same as above for the non-fully summed part, only here we do not need to record the column (lines 410-417). For the above example, the largest entry that is returned by the loop is $tmax = .874148119$. Finally, we calculate the largest entry in the other part of the row (lines 418-428), for which we also obtain $rmax = .874148119$.

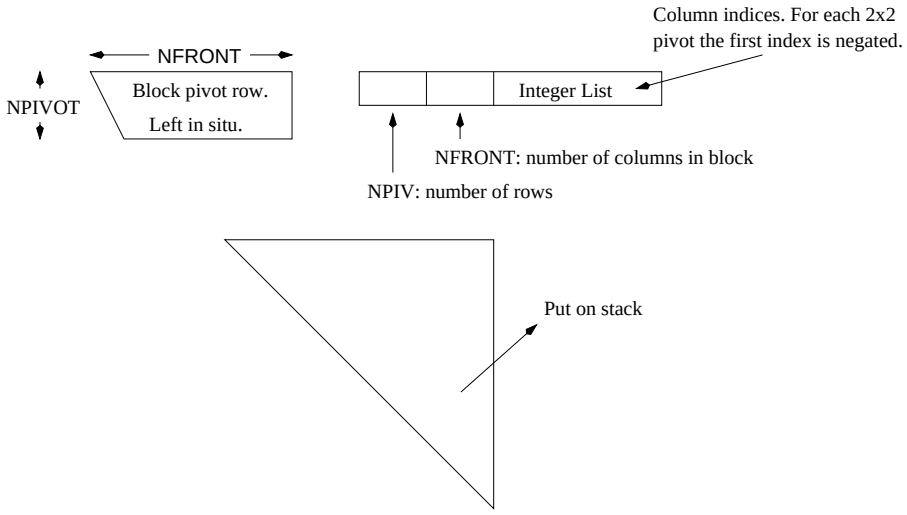


Figure 9: Disposition of the frontal matrix.

The stability test follows on line 430, where, for a 1×1 pivot, we apply the chosen pivot a_{jj} if $\mid a_{jj} \mid > u \mid a_{jk} \mid$ holds. At this stage of the elimination process, we ascertain $.00000 > .00000$, and so the stability test is not satisfied. Next, the stability of a possible 2×2 pivot must be checked, i.e.,

$$\left\| \begin{bmatrix} a_{kk} & a_{k,k+1} \\ a_{k+1,k} & a_{k+1,k+1} \end{bmatrix} \right\|_{\infty} \max_{j \neq k,k+1} [\max (\mid a_{kj} \mid, \mid a_{k+1,j} \mid)] \leq u^{-1}.$$

The numerical stability test for 2×2 pivots, which is performed on lines 432-474, is not satisfied ($amax = 0.0000$) in this particular case, and so neither the 1×1 nor the 2×2 pivot can be used. Execution is transferred to the main elimination loops (line $611 - 612$) and consequently, since no eliminations can be performed at this stage, the elements of the current front are put back onto the stack.

If an elimination had taken place at this point, the rows from which pivots have been chosen would have been at the beginning of the frontal matrix. This is visualised in

Figure 7. It follows that pivoted rows can be left in place as the next block pivot row in the factors. The remainder of the frontal matrix must be placed on the stack. The data movement in Figure 7 is expanded in Figure 9, where the integer storage for each block pivot row is explicitly shown.

Once the frontal matrix has been disposed of as shown in Figure 9, further assemblies can be performed (lines 615-667). Notice that any null rows or columns or any zero block are effectively swept to the end of the factored form and will not ever need to be stored. The total number of pivots ($NTOTPV$) is, in this case, less than n, the order of the system, but the decomposition is still valid for the non-singular submatrix of order $NTOTPV$. We elaborate on this behaviour at a later stage in the elimination process.

However, since no elimination has taken place at this stage, the entire front must be put back onto the stack. We ascertain

$a(1) - a(15)$ before line 615

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	−.03913	.87415	−.07919	−.00495	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(13)$	$a(14)$	$a(15)$	$a(16)$	$a(17)$	$a(18)$
.00000	.00000	.00000			

$a(515) - a(529)$ after line 615

$a(515)$	$a(516)$	$a(517)$	$a(518)$	$a(519)$	$a(520)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(522)$	$a(523)$	$a(524)$	$a(525)$	$a(526)$	$a(527)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(528)$	$a(529)$	$a(530)$	$a(531)$	$a(532)$	$a(533)$
.00000	.00000	.00000			

and also

$a(1) - a(15)$ before line 667

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(13)$	$a(14)$	$a(15)$	$a(16)$	$a(17)$	$a(18)$
.00000	.00000	.00000			

$a(515) - a(529)$ after line 667

$a(515)$	$a(516)$	$a(517)$	$a(518)$	$a(519)$	$a(520)$
.00000	−.03913	.87415	−.07919	−.00495	.00000
$a(522)$	$a(523)$	$a(524)$	$a(525)$	$a(526)$	$a(527)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(528)$	$a(529)$	$a(530)$	$a(531)$	$a(532)$	$a(533)$
.00000	.00000	.00000			

The array entries $a(1)$-$a(15)$ (shaded region below) are put on the stack $a(515)$-$a(529)$, and the initial entries $a(1)$-$a(15)$ are zeroed out.

3.2.2 Overall process at stage two of the elimination process

As in the previous section, the actual factorisation process starts on line 155 with the loop

```
    do 760 iass=1,nsteps
      .
      . (see original code for details)
      .
760 continue
```

and where $iass = 2$. Usually, the assembly of the frontal matrix for stages two or later of the elimination process starts by retrieving the stack elements from a previous stage of the elimination. This is accomplished by a loop of the form (lines 181-217)

```
    do 70 iell=1,numstk
       .
       . (see original source code for details)
       .
70 continue
```

However, since no elimination has taken place so far and therefore no (processed) elements are on the stack, the execution is transferred to line 223 (see above) and the assembly is continued by incorporating the original rows into the front (lines 223-253). We ascertain the following entries for array $IW2$ before line 223:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0

After the incorporation of the original rows (line 253), the same array has the entries

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	21	0	0	0	0	0	0	0	0	0	2	0	0	9	0	0	0	0	0

thus indicating that columns 15, 9, and 2 have been assembled into the current frontal matrix.

As before, the elimination process continues by incorporating the reals into the current front. Since the overall process is equivalent to the one at stage two, we shall not discuss it here again, but state the overall frontal matrix (line 332) which is given by

$$\begin{bmatrix} .00000 & .02893 & .83125 \\ & .00000 & .00000 \\ & & .00000 \end{bmatrix}.$$

Performing the same preparations for checking the stability of the current pivot as above (lines 333-430), we conclude on line 430 that in the current case, just as above, the stability test for the 1×1 pivot is not satisfied ($.00000 \not> .00000$). Furthermore, the 2×2 pivot does not satisfy the stability test either ($amax = .00000$ on line 432) and therefore execution is transferred back to the main elimination loops (line 611 - 612). Consequently, since no elimination has taken place, all the elements of the current front are put back onto the stack. As before, we ascertain the following array entries

$a(1) - a(6)$ after line 667

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
.00000	.00000	.00000	.00000	.00000	.00000

$a(509) - a(514)$ after line 667

$a(509)$	$a(510)$	$a(511)$	$a(512)$	$a(513)$	$a(514)$
.00000	.83125	.00000	.00000	.00000	.00000

3.2.3 Overall process at stage three of the elimination process

Since no eliminations have taken place so far, stage three of the elimination process ($iass = 3$) starts, as before, by assembling the frontal matrix (lines 171-253). However, the driver routine $MA27A$ returns that one stack element is generated at stage three ($nstk(3) = 1$) [4] and so the loop

```
    do 70 iell=1,numstk
       .
       .
       .
70 continue
```

is executed once. The corresponding entries of array $IW2$ before line 181 are given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

whereas the same array has the entries

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	21	0	0	0	0	0	0	2	0	0	0	0	0	9	0	0	0	0	0

after line 217, thus indicating that columns 15, 9 and 2 have been assembled from the stack.

The next step in the construction of the frontal matrix is, as before, the incorporation of the original rows into the frontal matrix. The entries of array $IW2$ before line 225 are given by

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	21	0	0	0	0	0	0	2	0	0	0	0	0	9	0	0	0	0	0

whereas the same array has the entries

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	0	21	0	0	0	0	0	0	12	0	0	2	0	0	9	0	0	0	0	0

after line 253, thus indicating that columns 15, 9, 12 and 2 have been assembled into the current frontal matrix.

As indicated by our elaboration of stage one, the elimination process continues by incorporating the reals into the current front, giving

[4] The elimination of pivot 9 implies the assembly of node 15—see the elimination tree on page 18 for details.

$$\begin{bmatrix} .00000 & .02893 & .00000 & .83125 \\ & .67390 & .00335 & .00000 \\ & & .00000 & .00000 \\ & & & .00000 \end{bmatrix}. \tag{3.1}$$

Similar to the fronts of stage one and two in the elimination process, the stability of the 1×1 pivot of the current front is not satisfied ($.00000 \not> .00000$ on line 430). However, the stability of the 2×2 pivot

$$\begin{bmatrix} .00000 & .02893 \\ .02893 & .67390 \end{bmatrix}$$

is satisfied (lines 431-474) and thus execution continues on line 479 with a loop (lines 483 - 545) that attempts to move the pivot block to the top left hand corner of the frontal matrix if required. However, since the chosen 2×2 pivot is already in the top left hand corner of the frontal matrix in this particular case, execution is transferred to line 547—the start of the actual elimination process—and consequently to line 576 since we are dealing with a 2×2 pivot.

A pseudo algorithm that visualises the 2×2 elimination process can be stated as follows.

Algorithm 3.1 (Gaussian elimination for 2×2 pivots)
Choose 2×2 *pivot* $A(1:2,1:2)$ *and compute inverse* $A(1:2,1:2)^{-1}$
Compute $A(1:2,1:n) = A(1:2,1:2)^{-1}A(1:2,1:n)$
For $i = 3, \ldots, n$
    For $j = 3, \ldots, n$
        $A(i,j) = A(i,j) - A(j,1)A(1,i) - A(j,2)A(2,i)$
    End for
End for.

Now, writing the front (3.1) in its full storage representation we ascertain

$$\begin{bmatrix} .00000 & .02893 & .00000 & .83125 \\ .02893 & .67390 & .00335 & .00000 \\ .00000 & .00335 & .00000 & .00000 \\ .83125 & .00000 & .00000 & .00000 \end{bmatrix}. \tag{3.2}$$

Here, the grey cells represent the chosen 2×2 pivot as defined above. Computing the inverse of the $A(1:2,1:2)$, which is given by

$$\begin{bmatrix} -805.29771 & 34.56838 \\ 34.56838 & 0.00000 \end{bmatrix},$$

and multiplying $A(1:2,1:n)$ by the inverse, we obtain

$$\begin{bmatrix} 1.00000 & .00000 & -.11568 & 669.40429 \\ .00000 & 1.0000 & .00000 & -28.73499 \\ .00000 & .00335 & .00000 & .00000 \\ .83125 & .00000 & .00000 & .00000 \end{bmatrix}. \tag{3.3}$$

This completes step one and two of Algorithm 3.1. What remains is to compute the entries $A(3,3)$, $A(3,4)$ and $A(3,5)$. This is achieved by the loop in step three of the above algorithm, giving

$$\begin{aligned} A(3,3) &= A(3,3) - A(3,1)A(1,3) - A(3,2)A(2,3) \\ &= .00000 - .00000 * .11568 - .00335 * .00000 \\ &= .00000 \end{aligned}$$

$$\begin{aligned} A(3,4) &= A(4,3) \\ &= A(3,4) - A(4,1)A(1,3) - A(4,2)A(2,3) \\ &= .00000 - .11568 * .83125 - .00000 * .00000 \\ &= -.09616 \end{aligned}$$

$$\begin{aligned} A(4,4) &= A(4,4) - A(4,1)A(1,4) - A(4,2)A(2,4) \\ &= .00000 - .83125 * 669.40429 - .00000 * 28.73499 \\ &= 556.44278. \end{aligned}$$

It follows that the overall frontal matrix for stage three of the elimination process is given by

$$\begin{bmatrix} -805.29772 & 34.56838 & -.11568 & 669.40428 \\ & .00000 & .00000 & -28.73499 \\ & & .00000 & -.09616 \\ & & & 556.44278 \end{bmatrix}.$$

Here, the entries $A(1,1) = -805.29772$, $A(1,2) = 34.56838$ and $A(2,2) = .00000$ correspond the relevant entries in (3.3) but include the values of the inverse of the 2×2 pivot in the data structure A within $MA27O$.

What remains after the actual factorisation is to put the factorised terms in gray back onto the stack, i.e.,

$a(1) - a(12)$ before line 615

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
−805.29772	34.56838	−.11568	669.40428	.00000	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
−28.73499	.00000	−0.09616	556.44278	.00000	.00000

$a(1) - a(12)$ after line 667

$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$
−805.29772	34.56838	−.11568	669.40428	.00000	.00000
$a(7)$	$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$
−28.73499	.00000	.00000	.00000	.00000	.00000

The corresponding stack elements are

$a(512) - a(517)$ before line 615

$a(512)$	$a(513)$	$a(514)$	$a(515)$	$a(516)$	$a(517)$
.00000000	.00000000	.00000000	.00000000	−.03913	.87415

$a(512) - a(517)$ after line 667

$a(512)$	$a(513)$	$a(514)$	$a(515)$	$a(516)$	$a(517)$
.00000000	−.09616	556.44278	.00000000	−.03913	.87415

3.2.4 Overall process at stages four and five of the elimination process

Stage four of the elimination proceeds along the same lines as the elimination in stage three, where the assembled front includes the columns

$$18 \to 12 \to 5 \to 19 \to 20 \to 16 \to 3 \to 6 \to 2 \to 17$$

As before, since the diagonal entry of row/column 18 is zero, the stability test for the 1×1 pivot fails. However, the stability of the 2×2 pivot is satisfactory and thus elimination proceeds as in the previous stage, giving the overall intermediate elimination stage corresponding to

$$\begin{bmatrix}
-805.29772 & 34.56838 & & -.11568 & & & & & \\
 & .00000 & & .00000 & & & & & \\
 & & -489.94323 & -25.55509 & 428.28295 & -.13477 & & & \\
 & & & .00000 & 22.33894 & .00000 & & & \\
 & & & & 374.38274 & -0.11780 & & & \\
 & & & & & .00000 & & & \\
 & & & & & & & 669.40428 & \\
 & & & & & & & -28.73499 & \\
 & & & .02634 & -.95617 & -38.79879 & -2.42625 & -2.45739 & .12432 \\
 & & & .00000 & .00000 & -2.02372 & -.12655 & .00000 & .00000 \\
 & & & .02303 & -.83584 & -33.91589 & -2.12090 & -2.14813 & .10868 \\
 & & & .00000 & .00000 & .01067 & .00067 & .00000 & .00000 \\
 & & & .00000 & .00000 & -.00209 & -.00013 & .00000 & .00000 \\
 & & & & .00000 & .07572 & .00474 & .00000 & .00000 \\
 & & & & & 3.07249 & .19214 & .19460 & -.00985 \\
 & & & & & & .01201 & .01217 & -.00062 \\
 & & & & & & & 556.44278 & .00000 \\
 & & & & & & & & .00000
\end{bmatrix}.$$

The entries in grey are put back onto the stack in order to be used in subsequent eliminations and execution is continued by assembling the fifth frontal matrix.

The fifth frontal matrix consists simply of the two columns

$$14 \to 1$$

Since no fill-in has occurred in the diagonal entry $(14, 14)$ and which is consequently zero, the stability test for the 1×1 pivot fails. Checking the stability of the 2×2 pivot reveals that its stability is not satisfied either, and thus execution continues at stage six. The sixth stage of the elimination process, which incorporates a peculiarity, is discussed in the next section.

3.2.5 Overall process at stages six of the elimination process

The assembly of the front at stage six, as well as the corresponding elimination process, is similar to the elimination described in previous sections. However, the fully summed

variable 14, which was not pivoted on in the previous section due to the failure of the numerical test, is included in the new front, indicating that two fully summed rows/column (14 and 10) are present. It follows that in case of the numerical instability of the first pivot, a second potential pivot is available in the front which may be eliminated. The importance of this observation is emphasised below.

The initial assembly of stack elements and original rows at the current stage returns the front

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IW2	13	21	6	7	0	2	8	11	0	1	3	0	4	10	0	0	0	0	0	0

indicating that columns 14, 10, 1, 13, 4, 7, 8, 11, 3, 6 and 2 have been assembled in the current frontal matrix.

Again, going through the individual steps of checking the stability of the 1×1 pivot, i.e., element $(14, 14)$, and the 2×2 pivot, which is defined as

$$\begin{bmatrix} (14,14) & (14,10) \\ (10,14) & (10,10) \end{bmatrix},$$

we notice that neither pivot satisfies the given stability criteria. However, element $(10, 10)$ in the current front, which has been fully summed, is the second pivot that may be eliminated. Thus, the second run through loop 640 chooses the entry $(10, 10)$ as pivot and performs the suitable stability test which is satisfied—line 430 giving $.859300795 > .000000001$. Before the elimination can take place the data structure

$$14 \to 10 \to 1 \to 13 \to 4 \to 7 \to 8 \to 11 \to 3 \to 6 \to 2$$

must be updated in order to incorporate the correct ordering

$$10 \to 14 \to 1 \to 13 \to 4 \to 7 \to 8 \to 11 \to 3 \to 6 \to 2$$

Here, the first two positions in the current front have been swapped such that the current pivot is in the top left corner. Writing the initial front in its column representation, we ascertain

$$\begin{array}{cccccccccccccc} (14,14) & (14,10) & (14,1) & (14,13) & (14,4) & (14,4) & (14,7) & (14,8) & (14,11) & (14,3) & (14,6) & (7,23) & (14,2) \\ \llcorner 1 \lrcorner\uparrow & \llcorner 2 \lrcorner\uparrow & \llcorner 3 \lrcorner\uparrow & \llcorner 4 \lrcorner\uparrow & \llcorner 5 \lrcorner\uparrow & \llcorner 6 \lrcorner\uparrow & \llcorner 7 \lrcorner\uparrow & \llcorner 8 \lrcorner\uparrow & \llcorner 9 \lrcorner\uparrow & \llcorner 10 \lrcorner\uparrow & \llcorner 11 \lrcorner\uparrow & \llcorner 12 \lrcorner\uparrow \end{array}$$

The subsequent entries of the front can be defined by reducing the number of columns by one, i.e., neglecting the first entry of the previous row, in each subsequent row. For instance, the second row in the front is given by

$$(10,10)\ (10,1)\ (10,13)\ (10,4)\ (10,7)\ (10,8)\ (10,11)\ (10,3)\ (10,6)\ (10,2)$$
$$\llcorner 1 \lrcorner\uparrow\ \llcorner 2 \lrcorner\uparrow\llcorner 3 \lrcorner\uparrow\llcorner 4 \lrcorner\uparrow\llcorner 5 \lrcorner\uparrow\llcorner 6 \lrcorner\uparrow\llcorner 7 \lrcorner\uparrow\llcorner 8 \lrcorner\uparrow\llcorner 9 \lrcorner\uparrow$$

Within the $MA27O$ source code the swapping is performed between lines 483-547. If the current pivot is not in the top left corner of the current front (as in this particular case), then the loop

```
    do 550 krow=1,pivsiz
       .
       .
       .
550 continue
```

is executed. The overall process can be visualised quite easily by considering the indices of the array locations occupied by the current front within the storage space A and by writing them in their frontal representation, i.e.,

$$\begin{bmatrix}
27 & 28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 \\
 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\
 & & 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\
 & & & 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \\
 & & & & 65 & 66 & 67 & 68 & 69 & 70 & 71 \\
 & & & & & 72 & 73 & 74 & 75 & 76 & 77 \\
 & & & & & & 78 & 79 & 80 & 81 & 82 \\
 & & & & & & & 83 & 84 & 85 & 86 \\
 & & & & & & & & 87 & 88 & 89 \\
 & & & & & & & & & 90 & 91 \\
 & & & & & & & & & & 92
\end{bmatrix}.$$

For $krow = 1$, the loop `do 470`, which `swaps the portion of rows whose column indices are greater than a later row`, returns the indices $j1 = 29$, $j2 = 37$ and $apos2 = 39$. It follows that the first row is swapped with the second row. Here, the individual entries correspond to the following locations within the matrix.

$$\begin{array}{cccccccccccc}
{\scriptstyle(14,14)} & {\scriptstyle(14,10)} & {\scriptstyle(14,1)} & {\scriptstyle(14,13)} & {\scriptstyle(14,4)} & {\scriptstyle(14,7)} & {\scriptstyle(14,8)} & {\scriptstyle(14,11)} & {\scriptstyle(14,3)} & {\scriptstyle(14,3)} & {\scriptstyle(14,2)} \\
27 & 28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 \\
\updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\
38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 & \\
{\scriptstyle(10,10)} & {\scriptstyle(10,1)} & {\scriptstyle(10,13)} & {\scriptstyle(10,4)} & {\scriptstyle(10,7)} & {\scriptstyle(10,8)} & {\scriptstyle(10,11)} & {\scriptstyle(10,3)} & {\scriptstyle(10,6)} & {\scriptstyle(10,2)} &
\end{array}$$

The last step in the current swapping process is to swap the diagonal entries, which is accomplished on lines 524-533 of the original code. The code returns $apos = 38$ and $posfac = 27$ and so $a(38)\ (10,10) \leftrightarrow a(27)\ (14,14)$. This completes the swapping process which has swapped rows/columns 14 and 10. It follows that the initial row/column order given above is now defined by

$$(10,10)\ (10,14)\ (10,1)\ (10,13)\ (10,4)\ (10,4)\ (10,7)\ (10,8)\ (10,11)\ (10,3)\ (10,6)\ \ (10,2)$$
$$\llcorner 1 \lrcorner\uparrow\ \llcorner 2 \lrcorner\uparrow\llcorner 3 \lrcorner\uparrow\llcorner 4 \lrcorner\uparrow\llcorner 5 \lrcorner\uparrow\llcorner 6 \lrcorner\uparrow\llcorner 7 \lrcorner\uparrow\llcorner 8 \lrcorner\uparrow\llcorner 9 \lrcorner\uparrow\llcorner 10 \lrcorner\uparrow\llcorner 11 \lrcorner\uparrow$$

and, in terms of the above index matrix, we ascertain

$$\begin{bmatrix} 38 & 28 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ & 27 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 \\ & & 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\ & & & 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \\ & & & & 65 & 66 & 67 & 68 & 69 & 70 & 71 \\ & & & & & 72 & 73 & 74 & 75 & 76 & 77 \\ & & & & & & 78 & 79 & 80 & 81 & 82 \\ & & & & & & & 83 & 84 & 85 & 86 \\ & & & & & & & & 87 & 88 & 89 \\ & & & & & & & & & 90 & 91 \\ & & & & & & & & & & 92 \end{bmatrix}.$$

Having chosen the correct pivot, which has already been swapped to the top left hand corner and which satisfies the 1×1 stability criterion, we are now in a position to perform the actual factorisation. The process is initiated on line 548 and concludes on line 572. Here, the corresponding entries of the elimination matrix A are defined as follows.

$a(27) - a(92)$ before line 548

$a(27)$	$a(28)$	$a(29)$	$a(30)$	$a(31)$	$a(32)$
.85930	.00000	.03075	.00126	.01290	.01051
$a(33)$	$a(34)$	$a(35)$	$a(36)$	$a(37)$	$a(38)$
−.00001	.00027	.10941	.00058	.01384	.00000
$a(39)$	$a(40)$	$a(41)$	$a(42)$	$a(43)$	$a(44)$
1.00000	.00000	.00000	.00000	.00000	.00000
$a(45)$	$a(46)$	$a(47)$	$a(48)$	$a(49)$	$a(50)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(51)$	$a(52)$	$a(53)$	$a(54)$	$a(55)$	$a(56)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(57)$	$a(58)$	$a(59)$	$a(60)$	$a(61)$	$a(62)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(63)$	$a(64)$	$a(65)$	$a(66)$	$a(67)$	$a(68)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(69)$	$a(70)$	$a(71)$	$a(72)$	$a(73)$	$a(74)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(75)$	$a(76)$	$a(77)$	$a(78)$	$a(79)$	$a(80)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(81)$	$a(82)$	$a(83)$	$a(84)$	$a(85)$	$a(86)$
.00000	.00000	.00000	.00000	.00000	.00000
$a(87)$	$a(88)$	$a(89)$	$a(90)$	$a(91)$	$a(92)$
.00000	.00000	.00000	.00000	.00000	.00000

$a(27) - a(92)$ after line 572

$a(27)$	$a(28)$	$a(29)$	$a(30)$	$a(31)$	$a(32)$
1.16374	.00000	−.03578	−.00146	−.01501	−.01223
$a(33)$	$a(34)$	$a(35)$	$a(36)$	$a(37)$	$a(38)$
.00001	−.00031	−.12733	−.00067	−.01611	.00000
$a(39)$	$a(40)$	$a(41)$	$a(42)$	$a(43)$	$a(44)$
1.00000	.00000	.00000	.00000	.00000	.00000
$a(45)$	$a(46)$	$a(47)$	$a(48)$	$a(49)$	$a(50)$
.00000	.00000	.00000	−.00110	−.00004	−.00046
$a(51)$	$a(52)$	$a(53)$	$a(54)$	$a(55)$	$a(56)$
−.00038	.00000	−.00001	−.00392	−.00002	−.00050
$a(57)$	$a(58)$	$a(59)$	$a(60)$	$a(61)$	$a(62)$
.00000	−.00002	−.00002	.00000	.00000	−.00016
$a(63)$	$a(64)$	$a(65)$	$a(66)$	$a(67)$	$a(68)$
.00000	−.00002	−.00019	−.00016	.00000	.00000
$a(69)$	$a(70)$	$a(71)$	$a(72)$	$a(73)$	$a(74)$
−.00164	−.00001	−.00021	−.00013	.00000	.00000
$a(75)$	$a(76)$	$a(77)$	$a(78)$	$a(79)$	$a(80)$
−.00134	−.00001	−.00017	.00000	.00000	.00000
$a(81)$	$a(82)$	$a(83)$	$a(84)$	$a(85)$	$a(86)$
.00000	.00000	.00000	−.00003	.00000	.00000
$a(87)$	$a(88)$	$a(89)$	$a(90)$	$a(91)$	$a(92)$
−.01393	−.00007	−.00176	.00000	−.00001	−.00022

The entries of $a(27) - a(92)$ after line 572 now correspond to the entries of the frontal matrix after Gaussian elimination. In order to verify the correctness of these results, consider the ordinary implementation of the Gaussian elimination algorithm as defined by Golub and Van Loan [7, page 99].

Algorithm 3.2 (Gaussian elimination for 1×1 pivots)

For $k = 1, 2, \ldots, n-1$
  Compute $A(k+1:n, k) = A(k+1:n/k)/A(k,k)$
  For $i = k+1, \ldots, n$
    For $j = k+1, \ldots, n$
      $A(i,j) = A(i,j) - A(i,k)A(k,j)$

End for
End for
End for.

Performing the individual elimination stages, we ascertain, for $k = 1$,

$$
\begin{array}{lrcl}
A(2:11,1) = A(2:11,1)/A(1,1) & \to A(2,1) & = & A(2,1)/A(1,1) \\
 & & = & .000000000/.859300795 \\
 & & = & .000000000 \\
 & \to A(3,1) & = & A(3,1)/A(1,1) \\
 & & = & .030748033/.859300795 \\
 & & = & .035782619 \\
 & \to A(4,1) & = & A(4,1)/A(1,1) \\
 & & = & .001256151/.859300795 \\
 & & = & .001461830 \\
 & \to A(5,1) & = & A(5,1)/A(1,1) \\
 & & = & .012897162/.859300795 \\
 & & = & .015008902 \\
 & \to A(6,1) & = & A(6,1)/A(1,1) \\
 & & = & .010508927/.859300795 \\
 & & = & .012229625 \\
 & \to A(7,1) & = & A(7,1)/A(1,1) \\
 & & = & -.000011455/.859300795 \\
 & & = & -.000013330 \\
 & \to A(8,1) & = & A(8,1)/A(1,1) \\
 & & = & .000267988/.859300795 \\
 & & = & .000311868 \\
 & \to A(9,1) & = & A(9,1)/A(1,1) \\
 & & = & .109413267/.859300795 \\
 & & = & .12732825 \\
 & \to A(10,1) & = & A(10,1)/A(1,1) \\
 & & = & .000579578/.859300795 \\
 & & = & .000674476 \\
 & \to A(10,1) & = & A(10,1)/A(1,1) \\
 & & = & .013842891/.859300795 \\
 & & = & .016109483
\end{array}
$$

The inner loops, i.e., $i = 2:11$ and $j = 2:11$, give

$$\begin{aligned}
\rightarrow A(2,2) &= A(2,2) - A(2,1)A(1,2) \\
&= .000000000 - (.000000000)(.000000000) \\
&= .000000000 \\
\rightarrow A(2,3) &= A(2,3) - A(2,1)A(1,3) \\
&= 1.000000000 - (.000000000)(.030748033) \\
&= 1.000000000 \\
\rightarrow A(2,4) &= A(2,4) - A(2,1)A(1,4) \\
&= .000000000 - (.000000000)(.001256151) \\
&= .000000000 \\
\rightarrow A(2,5) &= A(2,5) - A(2,1)A(1,5) \\
&= .000000000 - (.000000000)(.012897160) \\
&= .000000000 \\
\rightarrow A(2,6) &= A(2,6) - A(2,1)A(1,6) \\
&= .000000000 - (.000000000)(.010508920) \\
&= .000000000 \\
\rightarrow A(2,7) &= A(2,7) - A(2,1)A(1,7) \\
&= .000000000 - (.000000000)(-.000011450) \\
&= .000000000 \\
\rightarrow A(2,8) &= A(2,8) - A(2,1)A(1,8) \\
&= .000000000 - (.000000000)(.000267980) \\
&= .000000000 \\
\rightarrow A(2,9) &= A(2,9) - A(2,1)A(1,9) \\
&= .000000000 - (.000000000)(.109413260) \\
&= .000000000 \\
\rightarrow A(2,10) &= A(2,10) - A(2,1)A(1,10) \\
&= .000000000 - (.000000000)(.000579578) \\
&= .000000000 \\
\rightarrow A(2,11) &= A(2,11) - A(2,1)A(1,11) \\
&= .000000000 - (.000000000)(.013842891) \\
&= .000000000 \\
\rightarrow A(3,3) &= A(3,3) - A(3,1)A(1,3) \\
&= .000000000 - (.035782619)(.030748033) \\
&= -.001100245 \\
&\vdots
\end{aligned}$$

Comparing the entries of the frontal matrix after line 572 with the entries obtained from the Gaussian elimination algorithm verifies the correctness of the results stated above.

This completes the factorisation of the current frontal matrix. As before, entries in the pivot row are stored at the front of the storage array A and do not have to be accessed again, whereas entries from non-pivot rows are put on the stack in order to be considered by future fronts.

The remaining fronts formed in order to complete the entire factorisation of the coefficient matrix follow along the same lines as those described above; they are not given here.

In the next section, we consider the factored matrix—stored at the beginning of the array A–and describe how back substitution and forward substitution is applied to find the solution to the linear system of equations.

4 The solution phase

4.1 The forward substitution routine - MA27Q

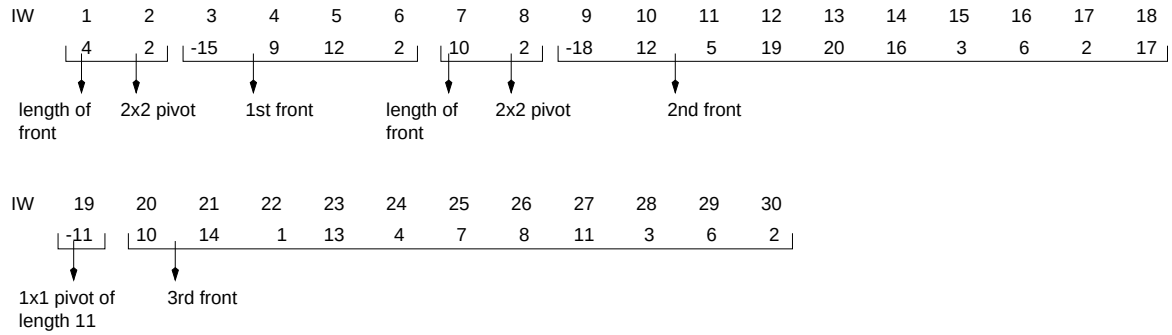
The factors that are produced by the factorisation routine $MA27O$, and which are stored in the arrays A and IW, are used by $MA27Q$ to solve the system

$$\mathcal{U}^T y = b. \tag{4.1}$$

Here, the solution vector y overwrites the right-hand side vector b held in the array RHS.

The substitution process for each individual block pivot is initiated on lines 66-92 by determining whether it is computationally better to use direct or indirect addressing in the innermost substitution loop [2, §3.5]. When *direct addressing* is applied, an additional vector W is used to store those components of the RHS vector that correspond to the rows of the block pivot. The substitutions are then performed considering W rather than RHS. On completion the components of W are copied back to the appropriate components of RHS. *Indirect addressing* does not require an auxiliary vector, but computations are performed using the RHS vector directly—see Duff and Reid [6] for details.

In the context of the example in this article, the first thirty entries of the work-array IW, which correspond to the first three fronts built during the elimination process, are given by



Within the work-array, the fronts corresponding to 2×2 pivots are preceded by two entries, which refer to the length of the front and the indication that a 2×2 pivot is used. On the other hand, 1×1 pivots are simply preceded by a single entry that refers to the length of the front involving the 1×1 pivot.

As outlined in the previous section, the first block pivot is a 2×2 pivot involving a front of dimension four, i.e.,

$$\begin{bmatrix} -805.29772 & 34.56838 & -.11568 & 669.40428 \\ & .00000 & .00000 & -28.73499 \\ & & .00000 & -.09616 \\ & & & 556.44278 \end{bmatrix}.$$

Here, the 2×2 pivot is located in the top left corner (elements in grey) and the elements which were put back on the stack during the elimination process are in the bottom right corner (also in grey). Forward substitution is performed using the elements in the top right corner of the above front, i.e., those elements which involve neither the pivot itself nor the elements that were put back on the stack. In this particular instance the substitution is executed using indirect addressing (line 186), giving

```
rhs( 12)    = rhs( 12)   + rhs( 15) * a( 3)      + rhs( 9) * a( 6)
0.88431811  = 1.00000000 + 1.000000 * -0.1156810 + 1.00000 * 0.000000

rhs( 2)     = rhs( 2)    + rhs( 15) * a( 4)      + rhs( 9) * a( 7)
641.669291  = 1.00000000 + 1.000000 * 669.404285 + 1.00000 * -28.7349
```

Consulting the entries of the data structure stored in the array A, i.e.,

```
A(1-5) -805.2977168  34.5683843  -0.1156818 669.4042847   0.0000000
A(6-7)    0.0000000 -28.7349934
```

confirms the above theory. The first two entries of A, as well as the fourth entry, correspond to the 2×2 block pivot, whereas the entries $A(3)$, $A(4)$, $A(6)$ and $A(7)$ are substituted on.

The second block pivot, which is also of dimension 2, is again applied using indirect addressing. The corresponding operations (line 186) are

```
rhs(  5)    = rhs(  5)   + rhs( 18)   * a( 10)      + rhs( 12)  * a( 19)
449.037678  = 1.00000000 + 1.00000000 * 428.282951  + 0.884318  * 22.3389

rhs( 19)    = rhs( 19)   + rhs( 18)   * a( 11)      + rhs( 12)  * a( 20)
0.865234    = 1.00000000 + 1.00000000 * -0.1347650  + 0.884318  * 0.00000

rhs( 20)    = rhs( 20)   + rhs( 18)   * a( 12)      + rhs( 12)  * a( 21)
1.026343    = 1.00000000 + 1.00000000 * 0.02634300  + 0.884318  * 0.00000

rhs( 16)    = rhs( 16)   + rhs( 18)   * a( 13)      + rhs( 12)  * a( 22)
0.043827    = 1.00000000 + 1.00000000 * -0.9561720  + 0.884318  * 0.00000

rhs(  3)    = rhs(  3)   + rhs( 18)   * a( 14)      + rhs( 12)  * a( 23)
-39.588401  = 1.00000000 + 1.00000000 * -38.798791  + 0.884318  * -2.0237

rhs( 6)     = rhs( 6)    + rhs( 18)   * a( 15)      + rhs( 12)  * a( 24)
-1.538156   = 1.00000000 + 1.00000000 * -2.426245   + 0.884318  * -0.1265

rhs( 2)     = rhs( 2)    + rhs( 18)   * a( 16)      + rhs( 12)  * a( 25)
639.211897  = 641.669200 + 1.00000000 * -2.45739400 + 0.884318  * 0.00000
```

```
rhs( 17)    = rhs( 17)   + rhs( 18)   * a( 17)      + rhs( 12)  * a( 26)
1.124324    = 1.00000000 + 1.00000000 * 0.1243246   + 0.884318  * 0.00000
```

Again, checking the entries of the work-array IW reveals the agreement between the columns that make up the second front in the elimination process (see previous section) and also the operations listed above. Only the elements involving the pivot rows (except the 2×2 pivot itself) are used in the substitution. The corresponding front is defined as

$$
\begin{bmatrix}
-805.29772 & 34.56838 & & -.11568 & & \\
 & .00000 & & .00000 & & \\
 & & -489.94323 & -25.55509 & 428.28295 & -.13477 \\
 & & & .00000 & 22.33894 & .00000 \\
 & & & & 374.38274 & -0.11780 \\
 & & & & & .00000 \\
 & & & & & \\
 & & & & & \\
 & & & & & \\
 & & & & &
\end{bmatrix}
\begin{bmatrix}
 & & & & 669.40428 & \\
 & & & & -28.73499 & \\
.02634 & -.95617 & -38.79879 & -2.42625 & -2.45739 & .12432 \\
.00000 & .00000 & -2.02372 & -.12655 & .00000 & .00000 \\
.02303 & -.83584 & -33.91589 & -2.12090 & -2.14813 & .10868 \\
.00000 & .00000 & .01067 & .00067 & .00000 & .00000 \\
.00000 & .00000 & -.00209 & -.00013 & .00000 & .00000 \\
 & .00000 & .07572 & .00474 & .00000 & .00000 \\
 & & 3.07249 & .19214 & .19460 & -.00985 \\
 & & & .01201 & .01217 & -.00062 \\
 & & & & 556.44278 & .00000 \\
 & & & & & .00000
\end{bmatrix},
$$

with the relevant entries in the array A being given by

```
A( 1- 5) -805.2977168  34.5683843   -0.1156818  669.4042847   0.0000000
A( 6-10)    0.0000000 -28.7349934 -489.9432282  -25.5550938 428.2829513
A(11-15)   -0.1347651   0.0263438   -0.9561726  -38.7987914  -2.4262451
A(16-20)   -2.4573944   0.1243246    0.0000000   22.3389372   0.0000000
A(21-25)    0.0000000   0.0000000   -2.0237176   -0.1265512   0.0000000
A(26-26)    0.0000000
```

Here, the first two rows in the above front (entries $A(1)$-$A(7)$) refer to the first block pivot of dimension 2×2 and the entries in rows five to twelve are the factored elements that were put back on the stack in order to be incorporated in later fronts.

The third front in the elimination process had dimension eleven and involved a 1×1 pivot—as indicated above, $IW(19) = -11$. The substitution is again performed using indirect addressing (line 168) and involves the operations

```
rhs( 14) = rhs( 14) + a( 28)   * rhs( 10)
1.00000  = 1.000000 + 0.000000 * 1.000000
```

```
rhs( 1)  = rhs( 1)  + a( 29)   * rhs( 10)
0.96421  = 1.000000 + -0.03578 * 1.000000

rhs( 13) = rhs( 13) + a( 30)   * rhs( 10)
0.99853  = 1.000000 + -0.00146 * 1.000000

rhs( 4)  = rhs( 4)  + a( 31)   * rhs( 10)
0.98499  = 1.000000 + -0.01500 * 1.000000

rhs( 7)  = rhs( 7)  + a( 32)   * rhs( 10)
0.98777  = 1.000000 + -0.01222 * 1.000000

rhs( 8)  = rhs( 8 ) + a( 33)   * rhs( 10)
1.00001  = 1.000000 + 0.000013 * 1.000000

rhs( 11) = rhs( 11) + a( 34)   * rhs( 10)
0.99968  = 1.000000 + -0.00031 * 1.000000

rhs( 3)  = rhs( 3)  + a( 35)   * rhs( 10)
-39.715  = -39.5884 + -0.12732 * 1.000000

rhs( 6)  = rhs( 6)  + a( 36)   * rhs( 10)
-1.5388  = -1.53815 + -0.00067 * 1.000000

rhs( 2)  = rhs( 2)  + a( 37)   * rhs( 10)
639.195  = 639.2118 + -0.01610 * 1.000000
```

The corresponding array entries of A are

```
A( 1- 5) -805.2977168  34.5683843   -0.1156818  669.4042847   0.0000000
A( 6-10)    0.0000000 -28.7349934 -489.9432282  -25.5550938 428.2829513
A(11-15)   -0.1347651   0.0263438   -0.9561726  -38.7987914  -2.4262451
A(16-20)   -2.4573944   0.1243246    0.0000000   22.3389372   0.0000000
A(21-25)    0.0000000   0.0000000   -2.0237176   -0.1265512   0.0000000
A(26-30)    0.0000000   1.1637368    0.0000000   -0.0357826  -0.0014618
A(31-35)   -0.0150089  -0.0122296    0.0000133   -0.0003118  -0.1273282
A(36-37)   -0.0006744  -0.0161094
```

The remaining substitution steps, which are not outlined here, are performed in a similar manner. References to the vector RHS, which stores the intermediate result of the forward substitution, and which is passed to the back substitution routine $MA27R$, do not refer to the original ordering of the rows and columns of the coefficient matrix $\mathcal{A}$, but to its permuted representation. In Section 5, we outline a procedure which may be used to identify where individual entries of RHS are located within the original ordering.

4.2 The back substitution routine - MA27R

The back substitution routine $MA27R$ works along the same lines as its corresponding forward substitution routine, which we discussed in the previous section. However, the final substitution operations are not performed from top to bottom, but in a similar manner to the ordinary forward/backward substitution theory by computing the final solution vector from bottom to top. Thus, the substitutions start with the final block pivot— or the final front in the elimination—and conclude with the first block pivot.

Starting with our discussion from the third but last block pivot, which is a 1×1 pivot with the operations performed on a front of dimension eleven, reveals the following computations. As before, the substitutions are executed using indirect addressing with the column indices and the relevant frontal information being stored in the work-array IW. The backward substitutions involving the current 1×1 block pivot are initiated on line 166 by handling the pivot itself, i.e.,

```
rhs( 10)    = rhs( 10)  * a( 27)
1.16373685  = 1.0000000 * 1.16373685
```

The corresponding incorporation of the relevant columns of the current front are performed within a loop (lines 170-174), giving

```
rhs( 10)    = rhs( 10)   + a( 28)         * rhs( 14)
1.16373685  = 1.16373685 + 0.00000000000 * 0.39417215

rhs( 10)    = rhs( 10)   + a( 29)         * rhs( 1)
1.12795423  = 1.16373685 + -0.0357826193 * 1.00000000

rhs( 10)    = rhs( 10)   + a( 30)         * rhs( 13)
1.12592055  = 1.12795423 + -0.0014618300 * 1.39118609

rhs( 10)    = rhs( 10)   + a( 31)         * rhs( 4)
1.10860324  = 1.12592055 + -0.0150089029 * 1.15380297

rhs( 10)    = rhs( 10)   + a( 32)         * rhs( 7)
1.09450153  = 1.10860324 + -0.0122296258 * 1.15307701

rhs( 10)    = rhs( 10)   + a( 33)         * rhs( 8)
1.09451552  = 1.09450153 + 0.00001333033 * 1.04942995

rhs( 10)    = rhs( 10)   + a( 34)         * rhs( 11)
1.09412851  = 1.09451552 + -0.0003118685 * 1.24095618

rhs( 10)    = rhs( 10)   + a( 35)         * rhs( 3)
0.92426730  = 1.09412851 + -0.1273282500 * 1.33404182
```

```
rhs( 10)    = rhs( 10)   + a( 36)        * rhs( 6)
0.923572339 = 0.92426729 + -0.0006744766 * 1.03036833

rhs( 10)    = rhs( 10)   + a( 37)        * rhs( 2)
0.905020812 = 0.92357233 + -0.0161094830 * 1.15159046

rhs( 10)    = 0.905020812
```

This completes the computations involving the current 1×1 pivot. At this point element $rhs(10)$ is fully assembled and is equal to the corresponding entry in the solution vector x, i.e., $x(10) = 0.905020812$ refers to the original ordering.

The second but last block pivot is of order 2 and is equivalent to the pivot used in the second front in the subroutine $MA27Q$. Indirect addressing is used to initiate the substitution involving the current pivot (lines 184-185), with the operations comprising the actual pivot being given by

```
rhs( 18)    = rhs( 18)    * a( 8)       + rhs( 12)    * a( 9)
-512.542061 = 1.00000000  * -489.943228 + 0.884318112 * -25.555094

rhs( 12)    = rhs( 18)    * a( 9)       + rhs( 12)    * a( 18)
-25.5550939 = 1.00000000  * -25.5550939 + 0.884318112 * 0.00000000
```

The backward substitutions involving the non-pivot entries of the corresponding front are again inaugurated by a loop (lines 190-196), giving

```
rhs( 18)    = rhs( 18)   + rhs( 5)       * a( 10)
56.9060659  = -512.54206 + 1.32960727    * 428.282951

rhs( 12)    = rhs( 12)   + rhs( 5)       * a( 19)
4.14691953  = -25.555093 + 1.32960727    * 22.3389372

rhs( 18)    = rhs( 18)   + rhs( 19)      * a( 11)
56.8849538  = 56.9060659 + 0.156658622   * -0.1347652

rhs( 12)    = rhs( 12)   + rhs( 19)      * a( 20)
4.14691953  = 4.14691953 + 0.156658622   * 0.00000000

rhs( 18)    = rhs( 18)   + rhs( 20)      * a( 12)
56.8846507  = 56.8849538 + -0.0115081547 * 0.02634384

rhs( 12)    = rhs( 12)   + rhs( 20)      * a( 21)
4.14691953  = 4.14691953 + -0.0115081547 * 0.00000000

rhs( 18)    = rhs( 18)   + rhs( 16)      * a( 13)
56.8998457  = 56.8846507 + -0.0158915837 * -0.9561727
```

```
rhs( 12)    = rhs( 12)   + rhs( 16)      * a( 22)
4.14691953  = 4.14691953 + -0.0158915837 * 0.00000000

rhs( 18)    = rhs( 18)   + rhs( 3)       * a( 14)
5.14063534  = 56.8998457 + 1.33404182    * -38.798791

rhs( 12)    = rhs( 12)    + rhs( 3)      * a( 23)
1.44719555  = 4.14691953  + 1.33404182   * -2.0237177

rhs( 18)    = rhs( 18)    + rhs( 6)      * a( 15)
2.64070920  = 5.14063534  + 1.03036833   * -2.4262451

rhs( 12)    = rhs( 12)    + rhs( 6)      * a( 24)
1.31680116  = 1.44719555  + 1.03036833   * -0.1265512

rhs( 18)    = rhs( 18)    + rhs( 2)      * a( 16)
-0.18920278 = 2.6407092   + 1.15159046   * -2.4573944

rhs( 12)    = rhs( 12)    + rhs( 2)      * a( 25)
1.31680116  = 1.31680116  + 1.15159046   * 0.00000000

rhs( 18)    = rhs( 18)    + rhs( 17)     * a( 17)
-0.17166994 = -0.18920278 + 0.141024596  * 0.12432469

rhs( 12)    = rhs( 12)    + rhs( 17)     * a( 26)
1.31680116  = 1.31680116  + 0.141024596  * 0.00000000

rhs( 18) -> -0.171669949
rhs( 12) ->  1.316801160
```

As indicated above, the entries $rhs(18)$ and $rhs(12)$ refer to the original entries of the solution vector, respectively given by $x(12) = 1.31680116$ and $x(18) = -0.171669949$.

The operations involving the last block pivot of dimension 2, and thus the remaining two entries of the solution vector x that have not been defined yet, follow the along the same lines as indicated above (lines 190-196) and are given by

```
rhs( 15)    = rhs( 15)   * a( 1)       + rhs( 9)     * a( 2)
-770.729332 = 1.00000000 * -805.297717 + 1.00000000  * 34.5683844

rhs(  9)    = rhs( 15)   * a( 2)       + rhs( 9)     * a( 5)
34.5683844  = 1.00000000 * 34.5683844  + 1.00000000  * 0.00000000
```

The corresponding backward substitutions of the non-pivot entries of the current front are

```
rhs( 15)    = rhs( 15)    + rhs( 12)     * a( 3)
-770.881663 = -770.729332 + 1.31680116   * -0.11568189

rhs(  9)    = rhs(  9)    + rhs( 12)     * a( 6)
34.5683844  = 34.5683844  + 1.31680116   * 0.000000000

rhs( 15)    = rhs( 15)    + rhs( 2)      * a( 4)
-0.00207227 = -770.881663 + 1.15159046   * 669.4042850

rhs(  9)    = rhs(  9)    + rhs( 2)      * a( 7)
1.47743996  = 34.5683844  + 1.15159046   * -28.7349934

rhs( 15) -> -0.00207228
rhs( 9)  ->  1.47743996
```

This completes the overall backward substitution process. At this point all fronts built during the elimination process have been eliminated and all elements of the solution vector x have been fully assembled, with the remaining two entries being defined by $x(9) = 1.47743996$ and $x(15) = -0.00207227864$. The overall solution vector x is given by

$$x = \begin{bmatrix} 1.000000000000000 \\ 1.151590463117462 \\ 1.334041821966219 \\ 1.153802973500643 \\ 1.329607271879568 \\ 1.030368333316440 \\ 1.153077010727039 \\ 1.049429948405755 \\ 1.477439964526797 \\ 0.905020811566008 \\ 1.240956184514922 \\ 1.316801160512123 \\ 1.391186092818604 \\ 0.394172151185358 \\ -0.002072278644448 \\ -0.015891583689995 \\ 0.141024596188653 \\ -0.171669949204010 \\ 0.156658621818426 \\ -0.011508154676598 \end{bmatrix}.$$

Performing the matrix-vector product $\mathcal{A}x$ reveals the right-hand side vector b, indicating that the above solution vector satisfies the equality (5.2).

5 Extracting the factors $\mathcal{U}$ and $\mathcal{D}$ from the $MA27$ data structure

As indicated in the introduction of this article, the $MA27$ subroutine library applies the factorisation

$$\mathcal{P}\mathcal{A}\mathcal{P}^T = \mathcal{U}^T\mathcal{D}\mathcal{U} \tag{5.1}$$

to solve the system

$$\mathcal{A}x = b. \tag{5.2}$$

Here, $\mathcal{P}$ is the permutation vector, $\mathcal{U}$ is an upper-triangular matrix, and $\mathcal{D}$ is a block-diagonal matrix consisting of 1×1 and possibly 2×2 blocks. The solution to (5.2) is obtained by applying the decomposition (5.1) in the following manner. Firstly, forward substitution solves the system

$$\mathcal{U}^T y = \mathcal{P}b, \tag{5.3}$$

and then back substitution is applied to solve the system

$$\mathcal{D}\mathcal{U}(\mathcal{P}x) = y. \tag{5.4}$$

Even though the explicit representation of the factors $\mathcal{U}$ and $\mathcal{D}$ is not necessary in order to understand the workings of the $MA27$ subroutine library, it may be useful to obtain an explicit representation such that the matrix products in (5.1) can be computed. The possible application of 2×2 pivots, and particularly their deep incorporation within the $MA27$ data structure, makes it rather difficult and complicated to extract the desired information.

As indicated in the previous section, the forward-substitution subroutine $MA27Q$ can be divided into four subsections, i.e.,

- The operations are performed using direct addressing;
  - lines 113-126 in the case of 1×1 pivots;
  - lines 127-142 in the case of 2×2 pivots.
- The operations are performed using indirect addressing.
  - lines 158-172 in the case of 1×1 pivots;
  - lines 173-190 in the case of 2×2 pivots.

Since the factors $\mathcal{U}$ and $\mathcal{D}$ are not stored in their original form, but in their permuted representation, it is necessary to construct a number of permutation vectors which indicate where a particular entry comes from in the original ordering. The first of these vectors is the *permutation vector* p; it determines the row/column swapping as defined

by the minimum degree and depth-first search algorithms. In the context of the example in this article, the vector p is defined by

$$p = \left[\ 7 \quad 19 \quad 17 \quad 9 \quad 12 \quad 18 \quad 10 \quad 11 \quad 3 \quad 6 \quad 16 \quad 4 \quad 8 \quad 5 \quad 2 \quad 15 \quad 20 \quad 1 \quad 13 \quad 14\ \right];$$

it results from the information kept in the array `ikeep` within the $MA27$ source code.

Before we can start to assemble the factors, a second vector referring to the sequence of pivot rows needs to be constructed. This vector can simply be obtained by considering the backward substitution routine $MA27R$ and by storing relevant pivot rows whenever a backward substitution takes place, i.e.,

```
1       subroutine ma27rd(n,a,la,iw,liw,w,maxfnt,rhs,iw2,nblk,latop,alg)
2  c this subroutine performs backward elimination operations
3             .
4             .
5             .
6             open(72,file='rdd.dat',status='new')
7             .
8             .
9             .
10
11            ipiv = npiv - iipiv + 1
12            if (ipiv.eq.1) go to 30
13 c jump if we have a 2 by 2 pivot.
14            if (iw(jpos-1).lt.0) go to 60
15 c perform back-substitution using 1 by 1 pivot.
16   30       .
17            .
18            .
19            w1 = w(ipiv)*a(apos)
20            write(72,*)alg(ipiv)
21            .
22            .
23            .
24 c perform back-substitution operations with 2 by 2 pivot
25   60       .
26            .
27            .
28            w1 = w(ipiv-1)*a(apos) + w(ipiv)*a(apos+1)
29            write(72,*)alg(ipiv)
30            write(72,*)alg(ipiv-1)
31            .
32            .
33            .
34            w2 = w(ipiv-1)*a(apos+1) + w(ipiv)*a(apos2)
```

```
35            .
36            .
37            .
38  c perform back-substitution using 1 by 1 pivot.
39            .
40            .
41            .
42            iirhs = iw(jpos)
43            w1 = rhs(iirhs)*a(apos)
44            write(72,*)iirhs
45            .
46            .
47            .
48  c perform operations with 2 by 2 pivot
49            .
50            .
51            .
52            i1rhs = -iw(jpos-1)
53            i2rhs = iw(jpos)
54            w1 = rhs(i1rhs)*a(apos) + rhs(i2rhs)*a(apos+1)
55            write(72,*)i2rhs
56            write(72,*)i1rhs
57            .
58            .
59            .
60            close(72)
61            end
```

For the example of this section, we ascertain the vector

$$rdd = \left[\ 17 \quad 2 \quad 6 \quad 3 \quad 11 \quad 16 \quad 20 \quad 19 \quad 5 \quad 8 \quad 7 \quad 4 \quad 13 \quad 1 \quad 14 \quad 10 \quad 12 \quad 18 \quad 9 \quad 15\ \right].$$

Using the two vectors p and rdd is sufficient in order to construct the final permutation vector $permut$. Considering the last entry in rdd, which has the value 15, and identifying its position within the vector p, we ascertain the first entry of the vector $permut$, which is 16. Similarly, considering the second but last entry of rdd, which has the value 9, and identifying its position within p, defines the second entry of $permut$ which is 4. Overall, the process just described can be written in a programming context as

```
1       do 20 i=n,1,-1
2          do 30 j=1,n
3             if(rdd(i).eq.p(j)) then
4                permut(counter)=j
5                write(*,*)permut(counter)
6                counter=counter+1
```

```
7              endif
8   30       continue
9   20     continue
```

defining the overall vector

$$permut = \begin{bmatrix} 16 & 4 & 6 & 5 & 7 & 20 & 18 & 19 & 12 & 1 & 13 & 14 & 2 & 17 & 11 & 8 & 9 & 10 & 15 & 3 \end{bmatrix}.$$

The vector *permut* defines the overall structure of the block diagonal matrix $\mathcal{D}$, i.e.,

$$\mathcal{D} = \begin{bmatrix} (16,16) & & & & & & & & & & & & & & & & & & & \\ & (4,4) & & & & & & & & & & & & & & & & & & \\ & & (6,6) & & & & & & & & & & & & & & & & & \\ & & & (5,5) & & & & & & & & & & & & & & & & \\ & & & & (7,7) & & & & & & & & & & & & & & & \\ & & & & & (20,20) & & & & & & & & & & & & & & \\ & & & & & & (18,18) & & & & & & & & & & & & & \\ & & & & & & & (19,19) & & & & & & & & & & & & \\ & & & & & & & & (12,12) & & & & & & & & & & \\ & & & & & & & & & (1,1) & & & & & & & & & \\ & & & & & & & & & & (13,13) & & & & & & & & \\ & & & & & & & & & & & (14,14) & & & & & & & \\ & & & & & & & & & & & & (2,2) & & & & & & \\ & & & & & & & & & & & & & (17,17) & & & & & \\ & & & & & & & & & & & & & & (11,11) & & & & \\ & & & & & & & & & & & & & & & (8,8) & & & \\ & & & & & & & & & & & & & & & & (9,9) & & \\ & & & & & & & & & & & & & & & & & (10,10) & \\ & & & & & & & & & & & & & & & & & & (15,15) \\ & & & & & & & & & & & & & & & & & & & (3,3) \end{bmatrix}.$$

Here, the numbers in brackets refer to the initial row/column of the pivot within the matrix $\mathcal{A}$. However, the above representation does not yet take the 2×2 pivots into account. The relevant information required to specify the actual representation is derived below.

Before the factor $\mathcal{U}$ can be constructed, it is necessary to reorder the solution vector y whose final specification is known once the overall controlling process leaves the subroutine $MA27Q$. This reordering is required in order for the individual entries of the factor $\mathcal{U}$ to be multiplied by the correct entries of the vector y. The first step in the assembly of the permutation vector pp, which is consequently used to perform the reordering, is to go through the vector rdd one by one and by looking for the corresponding index of the entry in the vector p. The index is the looked for in the vector $permut$ and whose index is the entry of the vector pp. For example, taking the first entry of vector rdd returns the value 17. Looking for the value 17 in the vector p returns the index 3, and searching for this index in the vector $permut$ gives 20. It follows that the first entry of pp is 20. The overall process of defining the vector pp can be described by the following FORTRAN code:

```
1     do 45 i=1,n
2           current=rdd(i)
3           do 46 j=1,n
4              if(current.eq.p(j)) then
5                 current1=j
6                 goto 500
7              endif
8   46        continue
9   500       do 47 j=1,n
10             if(current1.eq.permut(j)) then
11                pp(current)=j
12                goto 501
13             endif
14  47        continue
15  501       continue
16  45    continue
```

giving

$$pp = \begin{bmatrix} 20 & 19 & 18 & 17 & 16 & 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}.$$

Reordering the solution vector y can now be performed by applying the following code:

```
1       do 50 i=1,n
2          yy(pp(i))=y(i)
3   50    continue
```

defining the reordered form of the solution vector y

$$yy = \begin{bmatrix} 1.000000000000000 \\ 1.000000000000000 \\ 1.000000000000000 \\ 0.884318111764627 \\ 1.000000000000000 \\ 1.000000000000000 \\ 0.964217380739730 \\ 1.001033684383166 \\ 1.008828274756829 \\ 1.060944145416257 \\ 1.068056212297313 \\ 449.039387744984310 \\ 0.965337265722774 \\ -5.207969041404278 \\ 0.988697286953270 \\ 35.575878845224274 \\ 457.018773956291511 \\ 254.437958499165148 \\ 641.716463629558348 \\ -0.124619366787324 \end{bmatrix}.$$

It is now possible to start the assembly of the factors $\mathcal{U}$ and $\mathcal{D}$ by using the permutation vector *permut* and the vector p, which are defined above. The actual substitution steps on lines 122, 138, 168 and 186 within the subroutine $MA27Q$ refer to particular entries of the rhs vectors—the right hand side of the original equation— and memory locations within the data structure A that hold the factored entries. Since these entries are not stored in their original order, it is necessary to define the row and column indices of each individual entry that is stored in the array A. The definition of where a specific entry occurs within the upper triangular matrix A can be derived as described below. For instance, consider the entry $rhs[1]$. Looking for the index 1 in the vector p returns 18, i.e., $p[18] = 1$. Furthermore, identifying where the index 18 occurs in the permutation vector *permut*, we obtain $permut[7] = 18$. It follows that $rhs[1]$ within the subroutine $MA27Q$ refers to the entry $y[7]$ in Equation 5.2. Overall, the individual references are derived as follows:

$$\begin{aligned}
y_1 &\text{ is defined as } permut[\ 1] = 16 \rightarrow p[16] = 15 \rightarrow rhs(15) \\
y_2 &\text{ is defined as } permut[\ 2] = 4 \rightarrow p[\ 4] = 9 \rightarrow rhs(\ 9) \\
y_3 &\text{ is defined as } permut[\ 3] = 6 \rightarrow p[\ 6] = 18 \rightarrow rhs(18) \\
y_4 &\text{ is defined as } permut[\ 4] = 5 \rightarrow p[\ 5] = 12 \rightarrow rhs(12) \\
y_5 &\text{ is defined as } permut[\ 5] = 7 \rightarrow p[\ 7] = 10 \rightarrow rhs(10) \\
y_6 &\text{ is defined as } permut[\ 6] = 20 \rightarrow p[20] = 14 \rightarrow rhs(14)
\end{aligned}$$

$$
\begin{array}{lrcrcrcl}
y_{7} & \text{is defined as } permut[\ 7] = & 18 & \rightarrow & p[18] & = & 1 & \rightarrow rhs(\ 1) \\
y_{8} & \text{is defined as } permut[\ 8] = & 19 & \rightarrow & p[19] & = & 13 & \rightarrow rhs(13) \\
y_{9} & \text{is defined as } permut[\ 9] = & 12 & \rightarrow & p[12] & = & 4 & \rightarrow rhs(\ 4) \\
y_{10} & \text{is defined as } permut[10] = & 1 & \rightarrow & p[\ 1] & = & 7 & \rightarrow rhs(\ 7) \\
y_{11} & \text{is defined as } permut[11] = & 13 & \rightarrow & p[13] & = & 8 & \rightarrow rhs(\ 8) \\
y_{12} & \text{is defined as } permut[12] = & 14 & \rightarrow & p[14] & = & 5 & \rightarrow rhs(\ 5) \\
y_{13} & \text{is defined as } permut[13] = & 2 & \rightarrow & p[\ 2] & = & 19 & \rightarrow rhs(19) \\
y_{14} & \text{is defined as } permut[14] = & 17 & \rightarrow & p[17] & = & 20 & \rightarrow rhs(20) \\
y_{15} & \text{is defined as } permut[15] = & 11 & \rightarrow & p[11] & = & 16 & \rightarrow rhs(16) \\
y_{16} & \text{is defined as } permut[16] = & 8 & \rightarrow & p[\ 8] & = & 11 & \rightarrow rhs(11) \\
y_{17} & \text{is defined as } permut[17] = & 9 & \rightarrow & p[\ 9] & = & 3 & \rightarrow rhs(\ 3) \\
y_{18} & \text{is defined as } permut[18] = & 10 & \rightarrow & p[10] & = & 6 & \rightarrow rhs(\ 6) \\
y_{19} & \text{is defined as } permut[19] = & 15 & \rightarrow & p[15] & = & 2 & \rightarrow rhs(\ 2) \\
y_{20} & \text{is defined as } permut[20] = & 3 & \rightarrow & p[\ 3] & = & 17 & \rightarrow rhs(17)
\end{array}
$$

The first elimination that takes place within the $MA27Q$ routine is given by

```
rhs( 12) = rhs( 12) + a( 3) * rhs( 15) + a( 6) * rhs( 9)
rhs( 12) = 1.000000 -0.115681 * 1.000000 + 0.000000 * 1.000000
rhs( 12) = 0.884318112
```

Going through the above list, we identify that $rhs(12)$ refers to row 4 in the original ordering and thus elements $a(3)$ and $a(6)$ are entries of that row. To identify their respective columns all that is required is to check which column $rhs(15)$ represents—column 1 from above—and which column $rhs(9)$ represents—column 2 from above. It follows that $u_{41} = .115681$ (entries of $\mathcal{U}$ are stored in negated from within $MA27$), $u_{42} = .000000$ and also that $u_{44} = 1.000000$ by definition.

The second elimination is

```
rhs( 2) = rhs( 2) + a( 4) * rhs( 15) + a( 7) * rhs( 9)
rhs( 2) = 1.000000 + 669.404284 * 1.000000 -28.734993 * 1.000000
rhs( 2) = 641.669291
```

Here, $rhs(2)$ refers to row 19 in the original ordering. Similarly, $a(4)$ multiplies $rhs(15)$, which refers to row 1, and thus $a(4) = -669.404284$ must be the entry $u_{19,1}$. The entry $a(7)$ multiplies $rhs(9)$, which refers to row 2, and thus $a(7) = 28.734993$ must be the entry $u_{19,2}$.

The other entries of the factor $\mathcal{U}$ are derived accordingly. However, there is a slight complication if the elimination is performed using *direct addressing*. In this case the above elimination takes the form

```
w( 2) = w( 2) + a( 60) * w( 1)
w( 2) = 1.002484 + 0.006337 * 1.001033
w( 2) =   1.00882827
```

In order to decipher we must use the following *conversion table* which is also used within the $MA27$ algorithm:

```
w(  1) = rhs( 13)
w(  2) = rhs(  4)
w(  3) = rhs(  7)
w(  4) = rhs(  8)
w(  5) = rhs(  5)
w(  6) = rhs( 19)
w(  7) = rhs( 20)
w(  8) = rhs( 16)
w(  9) = rhs( 11)
w( 10) = rhs(  3)
w( 11) = rhs(  6)
w( 12) = rhs(  2)
w( 13) = rhs( 17)
```

It is easily seen that $w(2)$ refers to $rhs(4)$ and thus row 9. Similarly, $w(1)$ refers to $rhs(13)$ and thus row 8. It follows that the above substitution refers to element u_{98}, i.e., $u_{98} = -0.006337$.

The remaining entries involving both direct and indirect addressing are devised accordingly. However, there is another problem that has to be dealt with. It can sometimes happen that the above process does not return the correct columns in $\mathcal{U}$. In order to solve this problem, a backup strategy must be implemented which tests whether the columns, as devised by the above process, are the correct ones. This is done by comparing the values of the rhs and w entries with the corresponding values in yy. If they agree with the column depicted by the the above process, then the derivation is continued accordingly, otherwise a search is initiated that looks for the values of rhs and w in the vector yy and assigns the relevant indices. The overall process of devising $\mathcal{U}$ can now be described by the following FORTRAN program:

```
1     program main
2
3     implicit none
4
5     integer n, max
6     integer i,j,k
7     integer row, column
8     double precision value
9     integer rowF, columnF
10    integer counter, foo
```

```
11        integer current, currentcol, current1
12        double precision currentcolv
13
14        parameter(n=20, max=126)
15
16        integer position(n), vec(n), permut(n), final(n)
17        double precision rhs(n), rrhs(n)
18
19        integer rrow(max), col(max)
20        double precision a(max), rh(max)
21
22        open(67,file='rdd.dat',status='old')
23        open(68,file='p.dat',status='old')
24        open(69,file='rhs_reordered.dat',status='new')
25        open(70,file='rhs.dat',status='old')
26        open(71,file='permut.dat',status='new')
27        open(72,file='qdd.dat',status='old')
28        open(73,file='fooU.dat',status='new')
29        open(74,file='pp.dat',status='new')
30
31        counter=1
32
33        do 10 i=1,n
34           read(67,*)position(i)
35           read(68,*)vec(i)
36           write(73,*)i,i,'           1.0000000000000000'
37    10   continue
38
39        do 20 i=n,1,-1
40           do 30 j=1,n
41              if(position(i).eq.vec(j)) then
42                 permut(counter)=j
43                 write(71,*)permut(counter)
44                 counter=counter+1
45              endif
46    30        continue
47    20   continue
48
49        do 40 i=1,n
50           read(70,*)rhs(i)
51           rrhs(i)=0d0
52    40   continue
53
54        do 45 i=1,n
```

```
55          current=position(i)
56          do 46 j=1,n
57             if(current.eq.vec(j)) then
58                current1=j
59                goto 500
60             endif
61    46       continue
62
63    500      do 47 j=1,n
64             if(current1.eq.permut(j)) then
65                final(current)=j
66                write(74,1001)final(current)
67                goto 501
68             endif
69    47       continue
70    501      continue
71    45    continue
72
73          do 50 i=1,n
74             foo=final(i)
75             rrhs(foo)=rhs(i)
76    50    continue
77
78          do 60 i=1,n
79             write(69,1002)rrhs(i)
80    60    continue
81
82          do 70 i=1,max
83             read(72,*)rrow(i),a(i),rh(i),col(i)
84    70    continue
85
86          do 80 i=1,n
87             current=vec(permut(i))
88             do 90 j=1,max
89                if(rrow(j).eq.current) then
90                   currentcolv=rh(j)
91                   do 95 k=1,n
92                      if(col(j).eq.vec(k)) then
93                         currentcol=k
94                         goto 96
95                      endif
96    95             continue
97    96             do 97 k=1,n
98                      if(currentcol.eq.permut(k)) then
```

```
99                      currentcol=k
100                     goto 98
101                  endif
102 97               continue
103 98               if(abs(currentcolv−rrhs(currentcol)).le.1e−8) then
104                     goto 110
105                  else
106                     do 100 k=1,n
107                        if(abs(currentcolv−rrhs(k)).le.1e−8) then
108                           currentcol=k
109                           goto 110
110                        endif
111 100                 continue
112                  endif
113 110                 continue
114                  write(73,1000)i,currentcol,−a(j)
115               endif
116 90         continue
117 80     continue
118
119        close(67)
120        close(68)
121        close(69)
122        close(70)
123        close(71)
124        close(72)
125        close(73)
126
127 1000 format(i4,i4,f25.15)
128 1001 format(i4)
129 1002 format(f25.15)
130        end
```

Thus, using the above strategy, the factor $\mathcal{U}$ is given by

$$\mathcal{U} = \begin{bmatrix}
1.0000 & & & 0.1157 & & & & & & & & & \\
 & 1.0000 & & 0.0000 & & & & & & & & & \\
 & & 1.0000 & & & & & & & & & & \\
 & & & 1.0000 & & & & & & & & & \\
 & & & & 1.0000 & 0.0000 & 0.0358 & 0.0015 & 0.0150 & 0.0122 & & & \\
 & & & & & 1.0000 & & −0.0025 & −0.0175 & −0.0585 & & & \\
 & & & & & & 1.0000 & 0.00000 & 0.0000 & 0.0000 & & & \\
 & & & & & & & 1.0000 & −0.0063 & −0.0012 & & & \\
 & & & & & & & & 1.0000 & −0.0134 & & & \\
 & & & & & & & & & 1.0000 & & & \\
 & & & & & & & & & & −669.4043 & & \\
 & & & & & & & & & & 28.7349 & & \\
 & −428.2830 & 0.1348 & −0.0263 & 0.9562 & & 38.7988 & 2.4262 & 2.4574 & −0.1243 & & & \\
 & −22.3389 & 0.0000 & 0.0000 & 0.0000 & & 2.0237 & 0.1266 & 0.0000 & 0.0000 & & & \\
−0.0001 & & & & & 0.0003 & 0.1273 & 0.0007 & 0.0161 & & & & \\
−0.0036 & 0.0064 & & & & −0.0393 & −0.0510 & −0.0006 & −0.0239 & & & & \\
0.0000 & 0.0000 & & & & 0.0000 & 0.0000 & 0.0000 & 0.0000 & & & & \\
−0.0004 & 0.0000 & 0.0066 & −0.0121 & −0.0010 & 0.0088 & −0.0029 & −0.0001 & −0.0056 & 0.0286 & & & \\
−0.0766 & −0.0096 & 0.0332 & −0.0019 & −0.0050 & 0.0636 & −0.0263 & −0.0004 & −0.0015 & 1.0399 & & & \\
0.0125 & 0.002179 & 0.0064 & 0.9201 & −0.0010 & −0.0134 & −0.0186 & −0.0012 & −0.0038 & 0.0104 & & & \\
1.0000 & −0.0007 & −0.0004 & −0.0115 & 0.0001 & 0.0069 & −0.0015 & −0.0001 & 0.0597 & 0.0319 & & & \\
 & 1.0000 & −0.0003 & 0.0001 & −0.0024 & −0.0001 & −0.0904 & −0.0057 & −0.0057 & 0.0003 & & & \\
 & & 1.0000 & 5.4480 & 0.1509 & −40.2316 & −5.3747 & −878.8796 & 0.5870 & 28.0389 & & & \\
 & & & 1.0000 & −0.0022 & 0.2720 & 0.0184 & 6.204 & −0.0093 & −0.1893 & & & \\
 & & & & 1.0000 & 5.7245 & −388.1569 & 62.7445 & 2.3218 & −4.1140 & & & \\
 & & & & & 1.0000 & −1.8832 & 15.0378 & 0.0206 & −0.5101 & & & \\
 & & & & & & 1.0000 & 0.0662 & −0.0059 & 0.0031 & & & \\
 & & & & & & & 1.0000 & −0.0036 & −0.0285 & & & \\
 & & & & & & & & 1.0000 & 0.0000 & & & \\
 & & & & & & & & & 1.0000 & & &
\end{bmatrix}.$$

The corresponding entries of the block diagonal matrix $\mathcal{D}$ can be ascertained by considering the subroutine $MA27R$ and by storing any reference to the storage locations A on line 117 (1×1 pivot used with direct addressing), lines 132-133 (2×2 pivot used with direct addressing), line 166 (1×1 pivot used with indirect addressing) or lines 184-185 (2×2 pivot used with indirect addressing). The block-diagonal entries of $\mathcal{D}$, which are defined by the storage locations referenced on these lines [5], have indices starting from n. Decrementing these indices each time a pivot on one of the above lines is used, gives

[5] Actually, the block-diagonal matrix $\mathcal{D}$ is stored as $\mathcal{D}^{-1}$ within the MA27 data structure.

$$
\mathcal{D} = \begin{bmatrix}
0.0000 & 0.0289 \\
0.0289 & 0.6739 \\
 & & 0.0000 & -0.0391 \\
 & & -0.0391 & 0.7502 \\
 & & & & 0.8593 \\
 & & & & & 0.0000 & 1.0000 \\
 & & & & & 1.0000 & 0.8013 \\
 & & & & & & & 0.7213 \\
 & & & & & & & & 0.8143 \\
 & & & & & & & & & 0.9524 \\
 & & & & & & & & & & 0.9466 \\
 & & & & & & & & & & & 375.2621 \\
 & & & & & & & & & & & & -0.0010 \\
 & & & & & & & & & & & & & -0.7766 \\
 & & & & & & & & & & & & & & -0.0022 \\
 & & & & & & & & & & & & & & & 2.5098 \\
 & & & & & & & & & & & & & & & & 327.3969 \\
 & & & & & & & & & & & & & & & & & 248.9059 \\
 & & & & & & & & & & & & & & & & & & 557.2432 \\
 & & & & & & & & & & & & & & & & & & & -0.8837
\end{bmatrix}.
$$

With the explicit representation of the factors $\mathcal{U}$ and $\mathcal{D}$ available, it is now possible to verify Equation (5.1). By extracting the individual blocks H, B and B^T from both matrix products, it is easily verified that the matrix product $\mathcal{P}^T\mathcal{U}^T\mathcal{D}\mathcal{U}\mathcal{P}$ is identical to the original coefficient matrix, that is

$$
\begin{aligned}
\max \mid H_{\mathcal{A}} - H_{\mathcal{U}^T\mathcal{D}\mathcal{U}} \mid &= 1.47883441603547e^{-13}, \\
\max \mid B_{\mathcal{A}} - B_{\mathcal{U}^T\mathcal{D}\mathcal{U}} \mid &= 2.67954061716758e^{-13}, \\
\max \mid B_{\mathcal{A}}^T - B_{\mathcal{U}^T\mathcal{D}\mathcal{U}}^T \mid &= 2.67954061716758e^{-13}.
\end{aligned}
$$

Figure 10 shows the spy plots of the block-diagonal matrix $\mathcal{D}$, the upper triangular matrix $\mathcal{U}$, the original coefficient matrix $\mathcal{A}$, and the matrix product $\mathcal{P}^T\mathcal{U}^T\mathcal{D}\mathcal{U}\mathcal{P}$.

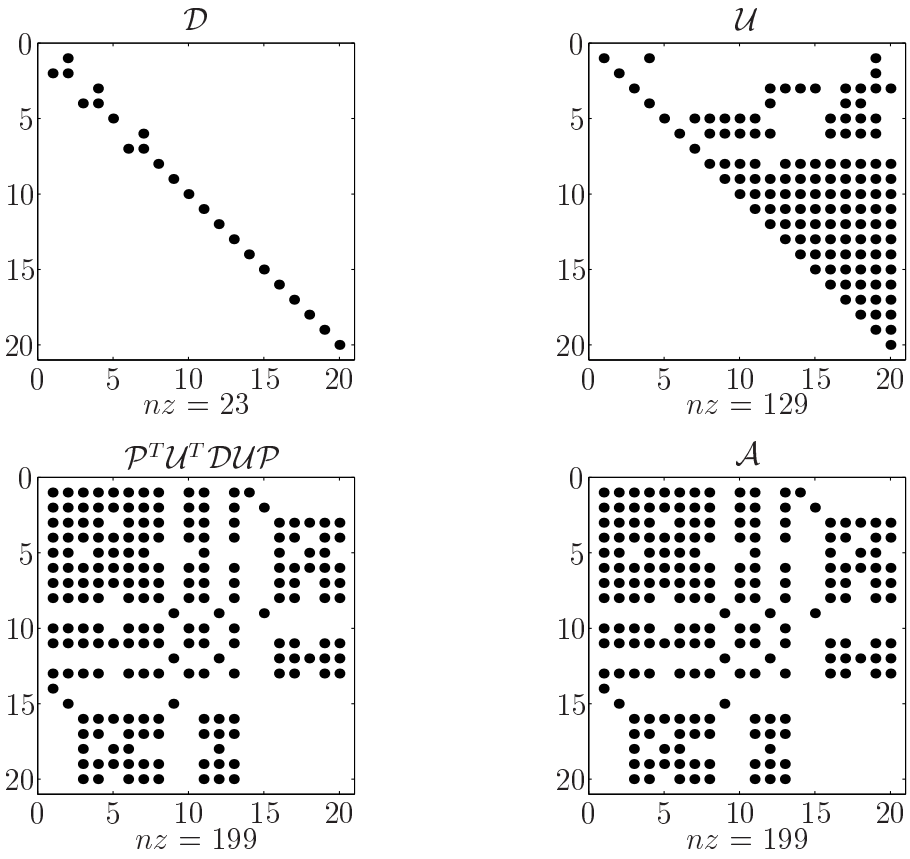


Figure 10: The implicit representations of the block diagonal matrix $\mathcal{D}$, the upper-triangular matrix $\mathcal{U}$, the original coefficient matrix $\mathcal{A}$ and the matrix product $\mathcal{P}^T\mathcal{U}^T\mathcal{D}\mathcal{U}\mathcal{P}$.

References

[1] J. R. Bunch and B. N. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal., 8 (1971), pp. 639-655.

[2] J. J. Dongarra, I. S. Duff, D. C. Sorensen and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, SIAM Press, Philadelphia, 1998.

[3] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1989.

[4] I. S. Duff, J. K. Reid, N. Munksgaard and H. B. Neilsen, *Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite*, Journal of the Institute of Mathematics and its Applications, 23 (1979), pp. 235-250.

[5] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Transactions on Mathematical Software, 9 (3) (1983), pp. 302-325.

[6] I. S. Duff and J. K. Reid, *MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Her Majesty's Stationery Office, AERE R10533, 1982

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1996.

[8] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, 1981.

[9] The Harwell Subroutine Library (HSL), http://www.dci.clrc.ac.uk/Activity.asp?HSL, 2000.

[10] P. Hood, *Frontal solution program for unsymmetric matrices*, Int. J. Numer. Meth. Eng., 10 (1976), pp. 379-400.

[11] B. M. Irons, *A frontal solution program for finite element analysis*, Int. J. Numer. Method. Eng., 2 (1970), pp. 5-32.

[12] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1996.