RAL-TR-95-040

Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems

by

I. S. Duff and J. K. Reid

Abstract

We describe the design of a new code for the solution of sparse indefinite symmetric linear systems of equations. The principal difference between this new code and earlier work lies in the exploitation of the additional sparsity available when the matrix has a significant number of zero diagonal entries. Other new features have been included to enhance the execution speed, particularly on vector and parallel machines.

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

General Terms: Algorithms, performance.

Additional Key Words and Phrases: sparse indefinite symmetric matrices, augmented systems, Gaussian elimination, zero diagonal entries, 2×2 pivots, BLAS.

Computing and Information Systems Department,
Rutherford Appleton Laboratory,
Oxon OX11 0QX.

August 1995.

CONTENTS

1 Introduction

This paper describes the design of a collection of Fortran subroutines for the direct solution of sparse symmetric sets of *n* linear equations

$$\mathbf{Ax} = \mathbf{b}, \tag{1.1}$$

when the matrix **A** is symmetric and has a significant number of zero diagonal entries. An example of applications in which such linear systems arise is the equality-constrained least-squares problem

$$\underset{\mathbf{x}}{\text{minimize}} \ \|\mathbf{Bx} - \mathbf{c}\|_2 \tag{1.2}$$

subject to

$$\mathbf{Cx} = \mathbf{d}. \tag{1.3}$$

This is equivalent to solving the sparse symmetric linear system

$$\begin{pmatrix} \mathbf{I} & & \mathbf{B} \\ & \mathbf{0} & \mathbf{C} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{C}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \lambda \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{c} \\ \mathbf{d} \\ \mathbf{0} \end{pmatrix}. \tag{1.4}$$

Another example is the quadratic programming problem

$$\underset{\mathbf{x}}{\text{minimize}} \ \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Hx} + \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{1.5}$$

subject to the linear equality constraints (1.3), where **H** is a symmetric matrix. Such problems arise both in their own right and as subproblems in constrained optimization calculations. Under a suitable inertial condition, the problem is equivalent to solving the symmetric but indefinite system of linear equations

$$\begin{pmatrix} \mathbf{H} & \mathbf{C}^{\mathrm{T}} \\ \mathbf{C} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} -\mathbf{c} \\ \mathbf{d} \end{pmatrix}. \tag{1.6}$$

Our earlier Harwell Subroutine Library code `MA27` (Duff and Reid 1982 and 1983) uses a multifrontal solution technique and is unusual in being able to handle indefinite matrices. It has a preliminary analysis phase that chooses a tentative pivot sequence from the sparsity pattern alone, assuming that the matrix is definite so that all the diagonal entries are nonzero and suitable as 1×1 pivots. For the indefinite case, this tentative pivot sequence is modified in the factorization phase to maintain stability by delaying the use of a pivot if it is too small or by replacing two pivots by a 2×2 block pivot (Bunch and Parlett 1971).

The assumption that all the diagonal entries are nonzero is clearly violated in the above examples. For such problems, the fill-in during the factorization phase of `MA27` can be significantly greater than predicted by the analysis phase. Duff, Gould, Reid, Scott, and Turner (1991) found that the use of 2×2 pivots with zeros on the diagonal alleviated this problem and also assisted the preservation of sparsity during the analysis phase. Our new code, `MA47`, is based upon this work and, like MA27, uses a multifrontal method. It will work for the definite case, but there are many opportunities for simplifications and efficiency improvements, so we plan to provide a separate code for this special case.

The factorization used has the form

$$\mathbf{A} = \mathbf{P}\,\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T\mathbf{P}^T \tag{1.7}$$

where $\mathbf{P}$ is a permutation, $\mathbf{L}$ is unit lower triangular, and $\mathbf{D}$ is block diagonal. A tentative choice of the permutation and blocking is made by working symbolically from the zero/nonzero structure of the matrix $\mathbf{A}$. We call this the ***analyse*** phase. Given the results from this phase, we ***factorize*** a matrix of the given structure, including symmetric permutations for the sake of numerical stability, but trying to keep close to the tentative pivotal sequence and block structure. Finally, we use the factorization to ***solve*** for a particular vector $\mathbf{b}$.

Throughout this paper, we use the term ***entry*** for a matrix coefficient that is nonzero or might be nonzero. Note that sometimes an entry may have the value zero, but a coefficient that is not an entry is always zero. If a coefficient of the reduced matrix is obtained by modification of an entry, we regard the result as an entry even if it has the value zero since it might be nonzero for another matrix of the same pattern. Also, the user may find it convenient to treat a zero as an entry during the analyse phase in anticipation of a later matrix having a nonzero in the corresponding position.

MA47 accepts an $n \times n$ symmetric sparse matrix whose entries are stored in any order in a real array with their row and column indices stored in corresponding positions in integer arrays. Each pair of off-diagonal entries a_{ij} and a_{ji} is represented by either entry. Multiple entries are permitted and are summed. This is the most user-friendly format that we have been able to devise, and is the same as that of MA27.

We describe the algorithm in Section 2. There are four subroutines that are called directly by the user:

Initialize. MA47I provides default values for the arrays that together control the execution of the package.

Analyse. MA47A accepts the pattern of $\mathbf{A}$ and makes a tentative choice of block pivots. It also calculates other data needed for actual factorization. The user may provide a pivotal sequence, in which case the necessary data will be generated.

Factorize. MA47B accepts a matrix $\mathbf{A}$ together with a set of recommended block pivots. It performs the factorization, including additional permutations when they are needed for numerical stability.

Solve. MA47C uses the factorization produced by MA47B to solve the equations $\mathbf{Ax} = \mathbf{b}$.

These are described in detail in a separate report (Duff and Reid 1995), in which the specification document is included as an appendix. Section 3 is devoted to our experience of the actual running of the code. The code has been placed in the Harwell Subroutine Library and is available from AEA Technology, Harwell; the contact is Dr John Harding, Harwell Subroutine Library, Bldg. 552, AEA Technology, Harwell, Didcot, Oxon OX11 0RA, tel (44) 1235 434573, fax (44) 1235 434340, email john.harding@aeat.co.uk, who will provide details of price and conditions of use. We also provide a complex version of the code, which handles symmetric complex matrices. We have chosen not to offer a version for Hermitian matrices because significant changes would have been needed to keep track of which off-diagonal entries move between the two triangular halves as permutations are made.

2 The algorithm

Our algorithm is based on the work of Duff, Gould, Reid, Scott, and Turner (1991) which uses 2×2 pivots with zeros on the diagonal to assist in the preservation of sparsity. Additionally, it is advantageous to perform elimination with several pivots simultaneously which we do by accumulating them into a block pivot.

We use block pivots that may be

(i) of the form

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}_1 \\ \mathbf{A}_1^T & \mathbf{0} \end{pmatrix} \tag{2.1}$$

with $\mathbf{A}_1$ square, which we call an ***oxo*** pivot;

(ii) of the form

$$\begin{pmatrix} \mathbf{A}_2 & \mathbf{A}_1 \\ \mathbf{A}_1^T & \mathbf{0} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \mathbf{0} & \mathbf{A}_1 \\ \mathbf{A}_1^T & \mathbf{A}_2 \end{pmatrix} \tag{2.2}$$

with $\mathbf{A}_1$ square, which we call a ***tile*** pivot; or

(iii) of any other form, which we call ***full.***

We will use the term ***structured*** for a pivot that is either a tile or an oxo pivot. The blocks $\mathbf{A}_1$ and $\mathbf{A}_2$ are usually full and we always store them as full matrices. Note that the inverse of a tile is a tile with its zero block in the other diagonal position. The inverse of an oxo is an oxo.

The matrix modifications of a block pivotal step that lie outside the pivot rows and columns (which we call the Schur update) are not applied at the time but are stored in a ***generated element matrix.*** This has entries only in a principal submatrix that after permutation has the general form

$$\begin{pmatrix} \mathbf{0} & \mathbf{B}_2 & \mathbf{B}_3 \\ \mathbf{B}_2^T & \mathbf{B}_1 & \mathbf{B}_4 \\ \mathbf{B}_3^T & \mathbf{B}_4^T & \mathbf{0} \end{pmatrix}, \tag{2.3}$$

where the blocks on the diagonal are square. This is the form of the submatrix altered by a pivotal step with an oxo pivot and is illustrated in Figure 2.1. The blocks $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$, and $\mathbf{B}_4$ are usually full and we always store them as full matrices. A tile pivot produces the special case of this form where the first or last block row and column is null. It is illustrated in Figure 2.2. For a full pivot, the generated element is held as a full matrix, which is the special case where the first and last block row and column are both null.

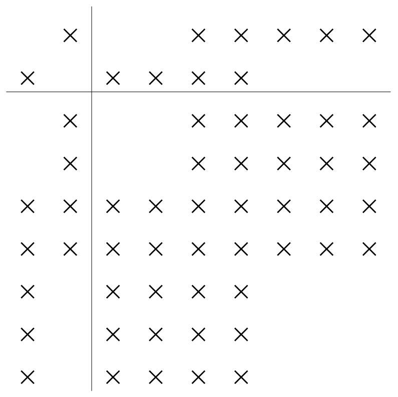

Figure 2.1. An oxo pivot, its pivot rows and columns, and fill-in pattern (generated element).

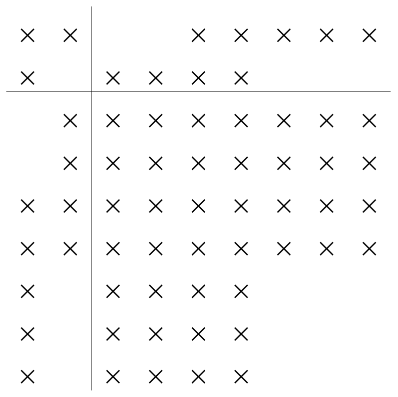

Figure 2.2. A tile pivot, its pivot rows and columns, and fill-in pattern (generated element).

We have chosen the multifrontal technique (Duff and Reid 1983) for the sake of efficiency during the analyse phase and to permit extensive use of full-matrix code and the BLAS (Basic Linear Algebra Subprograms: Lawson, Hanson, Kincaid, and Krogh 1979, Dongarra, Du Croz, Hammarling, and Hanson 1988, Dongarra, Du Croz, Duff, and Hammarling 1990) during factorization. We will use the notation $\mathbf{B}^{(l)}$ for the generated element matrix from the l-th (block) pivotal step and the notation $\mathbf{A}_k$ and $\mathbf{B}_k^{(l)}$ to denote the submatrices of $\mathbf{A}$ and $\mathbf{B}^{(l)}$ obtained by removing the rows and columns corresponding to the first k (block) pivotal steps. Following (block) step k, the reduced matrix is held as

$$\mathbf{A}_k + \sum_{l \in I_k} \mathbf{B}_k^{(l)} \tag{2.4}$$

where I_k is the set of indices of element matrices that are active then. If $\mathbf{B}_{k-1}^{(l)}$ has entries only in the pivotal rows and columns, $\mathbf{B}_k^{(l)}$ will be zero and l is omitted from the index set I_k. Other $\mathbf{B}_k^{(l)}$ may have entries that lie entirely within the pattern of the newly generated element $\mathbf{B}^{(k)}$; for efficiency, such a $\mathbf{B}_k^{(l)}$ is added into $\mathbf{B}^{(k)}$ and l is omitted from I_k. We say that $\mathbf{B}_k^{(l)}$ is ***absorbed*** into $\mathbf{B}^{(k)}$.

Such absorption certainly takes place if the pivot is full and overlaps one or more of the diagonal entries of $\mathbf{B}_k^{(l)}$ since in this case the pivot row has an entry for every index of $\mathbf{B}_k^{(l)}$. If all the pivots are full, all generated elements are full and therefore any generated element that is involved in a pivotal step is absorbed. This is the situation for a definite matrix.

In the definite case, the whole process may be represented by a tree, known as the ***assembly tree***, which has a node for each block pivotal step. The sons of a node correspond to the element matrices that contribute to the pivotal row(s) and are absorbed in the generated element. Here, it is efficient to add all the generated elements from the sons and the pivot rows from the original matrix into a temporary matrix known as the ***frontal matrix,*** which can be held in a square array of size the number of rows and columns with entries involved. The rows and columns are known as the ***front.*** For a fuller description of this case, see Duff, Erisman, and Reid (1986), Sections 10.5 to 10.9.

Given an assembly tree, there is considerable freedom in the ordering of the block pivotal steps during an actual matrix factorization. The operations are the same for any ordering such that the pivotal operations at a node follow those at a descendant of the node (apart from round-off effects caused by performing additions in a different order). Subject to this requirement, the order may be chosen for organizational convenience. For a uni-processor implementation, it is usual to base it on postordering following a depth-first search of the tree, which allows a stack to be used to store the generated elements awaiting assembly. We follow this practice.

When there are some structured pivots, we employ the same assembly tree, but a generated element is not necessarily absorbed at its father node. Instead, it may persist for several generations, making contributions to several pivotal rows, until it is eventually absorbed. As an illustration of absorption not occurring, a simple 1×1 pivot might overlap the leading (zero) block of (2.3). In such a case, $\mathbf{B}_1$ and $\mathbf{B}_4$ are absorbed but the non-pivotal rows of $\mathbf{B}_2$ and $\mathbf{B}_3$ are inactive during this step (unless made active by entries from other generated element matrices). Absorption of $\mathbf{B}_k^{(l)}$ occurs for a structured pivot if an off-diagonal entry of $\mathbf{B}_k^{(l)}$ overlaps the off-diagonal block $\mathbf{A}_1$ of the pivot. This is seen by regarding the structured pivot as a sequence of 1×1 pivots, starting with the off-diagonal entry and its symmetrically placed partner. To handle the structured case efficiently, we sum only the contributions to the pivot rows, form the Schur update, and then add into it any generated elements from the sons that can be absorbed. The frontal matrix is thus more complicated, but we still refer to the set of rows and columns involved as the front.

Similarly, the stack is more complicated. Access will be needed to generated elements corresponding to descendants (not just children), but these will still be nearer the top of the stack than any generated elements that do not correspond to descendants. When a generated element is absorbed, it may leave a hole in the stack. These holes are tolerated until the stack become too large for the available memory, at which point we perform a simple data compression. To aid both access to data in the stack and to aid stack compression, we merge adjacent holes as soon as they appear.

It is interesting to consider the effect of using full frontal matrices of sufficient size that each can accommodate all the elements and generated elements of its sons. Apart from round-off effects caused by performing additions in a different order, the operations performed on nonzeros will be the same, but there will be many additional operations involving zeros and there will be many stored zeros. Delaying the assemblies is our way of exploiting the sparsity.

Previously (Duff, Gould, Reid, Scott, and Turner 1991), we had anticipated working with generated elements (after permutation) of the forms

$$\begin{pmatrix} \mathbf{B}_1 & \mathbf{B}_2 \\ \mathbf{B}_2^T & \mathbf{0} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mathbf{0} & \mathbf{B}_3 \\ \mathbf{B}_3^T & \mathbf{0} \end{pmatrix}. \tag{2.5}$$

A tile generated element is of the first form and an oxo generated matrix can be represented as the sum of a matrix of the first form plus one of the second form:

$$\begin{pmatrix} \mathbf{0} & \mathbf{B}_2 & \mathbf{B}_3 \\ \mathbf{B}_2^T & \mathbf{B}_1 & \mathbf{B}_4 \\ \mathbf{B}_3^T & \mathbf{B}_4^T & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{B}_2 & \mathbf{0} \\ \mathbf{B}_2^T & \mathbf{B}_1 & \mathbf{B}_4 \\ \mathbf{0} & \mathbf{B}_4^T & \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{B}_3 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_3^T & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{2.6}$$

We have decided to use the form (2.3) because, in the case of an oxo generated element,

(i) the duplication of the index lists of the first and third blocks is avoided;

(ii) for a row of the first or third block, one rather than two elements involve it and need to be included in a list of elements associated with the row (such lists are needed during the analyse phase); and

(iii) a link would need to be maintained between the two parts of an oxo generated element in order to recognize that both parts of the element can be absorbed when an off-diagonal entry overlaps the off-diagonal block $\mathbf{A}_1$ of a structured pivot.

2.1 The design of the factorized form

Two considerations had a profound effect on our design for the factorized form:

(i) the wish to use block operations during both factorization and solution, and

(ii) the wish to be able readily to modify the factorization so that it is a factorization of a positive-definite matrix, needed in some optimization calculations.

For a full pivot, both are easy to achieve. If the pivot $\mathbf{A}_{11}$ is full, we factorize it as

$$\mathbf{A}_{11} \; = \; \mathbf{L}\,\mathbf{D}\,\mathbf{L}^T, \tag{2.7}$$

where **L** is unit lower triangular and **D** is a block diagonal matrix with blocks of order 1 or 2, and have the matrix factorization

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} \end{pmatrix} \; = \; \begin{pmatrix} \mathbf{L} & \\ \mathbf{M}^T & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D} & \\ & \mathbf{A}_{22} - \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{L}^T & \mathbf{M} \\ & \mathbf{I} \end{pmatrix}, \tag{2.8}$$

where **M** is the matrix of multipliers

$$\mathbf{M} \; = \; \mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{A}_{12} \tag{2.9}$$

and $\mathbf{A}_{22} - \mathbf{S}$ is the Schur complement, where

$$\mathbf{S} \; = \; \mathbf{M}^T\mathbf{D}\,\mathbf{M} \; = \; \mathbf{A}_{12}^T\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \tag{2.10}$$

is the Schur update. **D** may be perturbed to a positive-definite matrix by examining its (1×1 and 2×2) diagonal blocks and changing the diagonal entries as necessary.

A comparable factorization to (2.7) for a block tile pivot is

$$\mathbf{A}_{11} \; = \; \begin{pmatrix} \mathbf{0} & \mathbf{A}_1 \\ \mathbf{A}_1^T & \mathbf{A}_2 \end{pmatrix} \; = \; \begin{pmatrix} \mathbf{L}_1 & \\ \mathbf{U}_2^T & \mathbf{U}_1^T \end{pmatrix} \begin{pmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{D}_2 \end{pmatrix} \begin{pmatrix} \mathbf{L}_1^T & \mathbf{U}_2 \\ & \mathbf{U}_1 \end{pmatrix}, \tag{2.11}$$

where $\mathbf{L}_1$ is unit lower triangular, $\mathbf{U}_1$ is unit upper triangular, $\mathbf{U}_2$ is strictly upper triangular, and $\mathbf{D}_1$ and $\mathbf{D}_2$ are diagonal. The special case $\mathbf{U}_2 = \mathbf{D}_2 = \mathbf{0}$ corresponds to a block oxo pivot. Note that the relationship

$$\mathbf{A}_1 \; = \; \mathbf{L}_1\,\mathbf{D}_1\,\mathbf{U}_1 \tag{2.12}$$

holds, so that a conventional triangular factorization of $\mathbf{A}_1$ is included. We show that (2.11), with $\mathbf{U}_2$ strictly upper triangular, is a correct factorization by performing a symmetric permutation to place the rows and columns in the order 1, r+1, 2, r+2, 3, r+3, ..., where $2r$ is the order of the matrix in equation (2.11). This gives the block tile form of the symmetric matrix

$$\begin{pmatrix} t_{11} & t_{12} & . & . & t_{1r} \\ t_{21} & t_{22} & . & . & t_{2r} \\ . & . & . & . & . \\ . & . & . & . & . \\ t_{r1} & t_{r2} & . & . & t_{rr} \end{pmatrix} \tag{2.13}$$

where each t_{ij} is a 2×2 matrix whose (1,1) element is zero (each t_{ij} is a tile). Using $t_{11}, t_{22}, ..., t_{rr}$ as pivots, this matrix has the factorization

$$\begin{pmatrix} \mathbf{I} & & & & \\ l_{21} & \mathbf{I} & & & \\ . & . & . & & \\ . & . & . & . & \\ l_{r1} & l_{r2} & . & . & \mathbf{I} \end{pmatrix} \begin{pmatrix} d_{11} & & & & \\ & d_{22} & & & \\ & & . & & \\ & & & . & \\ & & & & d_{rr} \end{pmatrix} \begin{pmatrix} \mathbf{I} & l_{21}^T & . & . & l_{r1}^T \\ & \mathbf{I} & . & . & l_{r2}^T \\ & & . & . & . \\ & & & . & . \\ & & & & \mathbf{I} \end{pmatrix}, \tag{2.14}$$

where each l_{ij} has a zero in position (2,1) and each d_{ii} is a symmetric tile. If we now apply the inverse permutation, we get the form (2.11) with the lower triangular entries of $\mathbf{L}_1$, $\mathbf{U}_2^T$, $\mathbf{U}_1^T$, respectively, being the (1,1), (2,1), (2,2) entries of l_{ij}, and the diagonal entries of $\mathbf{D}_1$ and $\mathbf{D}_2$, respectively, being the (2,1) and (2,2) entries of d_{ii}. An alternative derivation of (2.11) is by application of a sequence of elementary row and column operations that reduce $\begin{pmatrix} \mathbf{0} & \mathbf{A}_1 \\ \mathbf{A}_1^T & \mathbf{A}_2 \end{pmatrix}$ to the form $\begin{pmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{D}_2 \end{pmatrix}$ in column 1, row 1, column r+1, row r+1, column 2, row 2, column r+2, row r+2, ... Note that, apart from the effects of reordering additions and subtractions, the forms (2.11) and (2.14) yield the same numerical values. We use both forms in our code. We compute the factors with (2.14). We apply the factors in solutions and in Schur updates using (2.11), which permits use of block operations.

The diagonal entries of $\begin{pmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{D}_2 \end{pmatrix}$ can be changed to make it positive definite as easily as those of $\mathbf{D}$ in (2.7). Thus, we may regard (2.7) as applicable to the tile case with

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_1 & \\ \mathbf{U}_2^T & \mathbf{U}_1^T \end{pmatrix} \quad \text{and} \quad \mathbf{D} = \begin{pmatrix} \mathbf{D}_3 & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{D}_2 \end{pmatrix}. \tag{2.15}$$

The special case $\mathbf{U}_2 = \mathbf{0}$ corresponds to an oxo pivot.

We therefore store, for each block pivot, $\mathbf{L}$, $\mathbf{D}$, and $\mathbf{M}$. The matrices $\mathbf{L}$ and $\mathbf{D}$ may be packed to take advantage of their form. For an oxo pivot, the nonzero columns of $\mathbf{M}$ are ordered to the form

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_5 & \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix}. \tag{2.16}$$

and only the blocks $\mathbf{M}_5$, $\mathbf{M}_6$, $\mathbf{M}_7$, and $\mathbf{M}_8$ are stored. For a tile pivot, the first block column is absent. These forms are discussed further in Section 2.3.

2.2 Analyse phase

In the analyse phase, we simulate the operations of the factorization, representing each generated element by three index lists and assuming that every pivot chosen is acceptable numerically. This allows the processing to be efficient and there is no need for any numerical values, but the pivotal sequence chosen has to be regarded as tentative. The analyse phase is faster than it would be if numerical values were taken into account and its storage demands are much more modest.

For pivotal strategy, we use the variant of the Markowitz (1957) criterion recommended by Duff, Gould, Reid, Scott, and Turner (1991). The Markowitz cost

$$(r_i - 1)^2 \tag{2.17}$$

for a diagonal entry a_{ii}, with row count r_i (number of entries in the row), is extended to

$$(r_i - 1)\,(r_i + r_j - 3) \tag{2.18}$$

for a tile pivot with nonzero a_{ii}, and

$$(r_i - 1)\,(r_j - 1) \tag{2.19}$$

for an oxo pivot. Applying a structured pivot is mathematically equivalent to pivoting in turn on the two off-diagonal entries of the pivot. In the oxo case, expression (2.19) is the Markowitz cost of either of these pivots. In the tile case, expression (2.18) is a bound for the Markowitz costs.

We find a pivot that minimizes this extended Markowitz cost by searching the rows in order of increasing row count as follows:

```
main_loop: for $r$ := 1 step 1 until $n$ do
    if there is a row $i$ with row count $r$ such that $a_{ii} \neq 0$ then
        accept it as a $1 \times 1$ pivot and exit main_loop
    end if
    for each row $i$ with row count $r$ do
        for each variable $j$ for which there is a nonzero entry $(i,j)$ in
            the current reduced matrix do
            if the Markowitz cost $\leq (r-1)^2$ then
                accept $(i,j)$ as a $2 \times 2$ pivot and exit main_loop
            end if
            if the Markowitz cost is the smallest so far found, store it as such
        end do
    end do
    if there is a stored $2 \times 2$ pivot with Markowitz cost $\leq r^2$ then
        accept the $2 \times 2$ pivot and exit main_loop
    end if
end do
```

We also provide an option to limit the search to a given number of rows with least entries.

We found it convenient to build a tree that represents the grouping of variables into blocks as well as the assemblies. It has a node for every variable, rather than for every block pivot. In the positive-definite case, this bigger tree is the elimination tree (see, for example, Liu 1990), so we will call it the ***elimination tree*** in this more general setting. The variables of a full pivot or of either of the halves of a structured pivot are linked together in a chain whose head we call the ***principal variable.*** The two principal variables of a structured pivot are linked together as father and son. Only these father nodes and the principal variables of full pivots have a node in the assembly tree.

It is interesting that if the same pivot sequence is a applied to the matrix obtained by adding entries to all the positions corresponding to zeros in the pivots and the elimination tree is formed, it is identical to our elimination tree apart from the detail in our tree within each tile and oxo pivot.

One of the ways to speed the analyse phase is to recognize rows with the same structure, both in the original matrix and the successive reduced matrices. The set of variables that correspond to such a set of rows is called a ***supervariable*** and we represent the matrix pattern in terms of supervariables. We allow a supervariable to consist of a single variable. If the rows do not have entries on the diagonal, we say that the supervariable is ***defective***. Each supervariable is indexed by one of its variables, which is its principal variable.

A simple example of a matrix and its elimination tree is shown in Figure 2.3. Here there are just two block pivots; the first is an oxo pivot of order 6 with variable 4 as its first principal variable and the second is a full pivot of order 2 with variable 7 as its principal variable. Variable 1 is the principal variable of the other half of the oxo pivot and node 1 is the son of node 4 in the elimination tree. The other variables are linked in chains to the three principal variables.

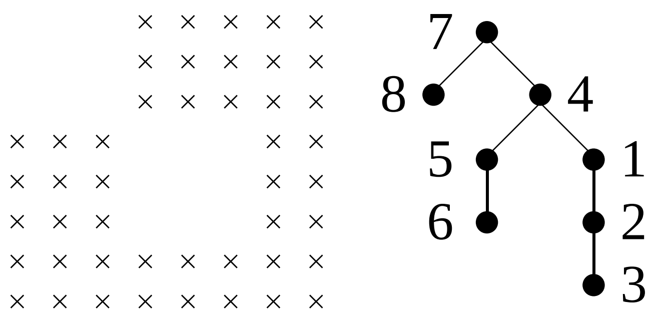

Figure 2.3. A matrix pattern with its elimination tree.

We begin the analyse phase by recognizing rows with identical structure and forming supervariables. With careful choice of data structures (see Section 2.7), the amount of work performed for each matrix entry is bounded by a constant, so the whole process executes in $O(n+\tau)$ time, where τ is the number of entries. This leaves us with a forest with a node for each variable and the variables of each supervariable in a chain with the principal variable at its head.

This forest is gradually modified until it becomes the elimination tree. At an intermediate stage, each tree of the forest has a root that represents the principal variable of either

(i) a supervariable that has not yet been pivotal, or
(ii) a supervariable that has been pivotal and whose generated element matrix is zero or has not yet contributed to a pivotal row.

The node of a principal variable that has been eliminated may be regarded as also representing the element generated when it was eliminated. When a pivot block is chosen, the node of its principal variable is given as sons the roots of all the trees that contain one or more generated elements that contribute to the pivot rows. If the pivot is a structured pivot, it is also given as a son the principal variable of the second part. Some amalgamation of supervariables may follow the pivotal step, since the fill-ins may cause some rows to have identical structures in the reduced matrix.

We have noted that a generated element matrix need not be absorbed at its father node. Indeed, it may persist for many generations of the tree, contributing to pivotal rows at each stage. This may mean that all that is left is a zero matrix, which is all or part of one of the zero blocks of (2.3). It might be thought that such a zero matrix could be discarded, but to do so risks the loss of a property of an assembly tree that we wish to exploit: the block pivotal steps may be reordered provided those at a node follow those at all descendants of the node. We must therefore retain such a zero element matrix and whenever one of its variables becomes pivotal, make the root of its tree a son of the pivot node.

In the irreducible case, this eventually yields the elimination tree, whose root is the only node with a zero generated element matrix. The node represents the final block pivot, which obviously leaves a null Schur complement. If the original matrix is reducible, it must be block diagonal since it is symmetric. In this case, the problem consists of several independent subproblems and there will be a tree for each. It is not difficult to allow for this and our code does so. For simplicity of description, however, we assume that the matrix is irreducible.

A depth-first search of the elimination tree allows the pivotal sequence to be found, including the grouping of variables into blocks that are eliminated together. At the same time, the assembly tree is constructed.

The opportunity is taken to look for father-son amalgamations that can take place with no extra fill-in, since most of them can be found by a simple test involving the number of variables eliminated at the son and the row counts before elimination. We have chosen to rely on this as the only mechanism that we employ to find block pivots when the user specifies the pivotal order. It is also useful when the pivotal order and tree are chosen by the code. Many block pivots will be found through the use of supervariables, but not all since it would be costly to ensure that every possible supervariable is identified (we see no way of doing this without at least one sweep of all the index lists for the supervariables).

If a father and son are both full pivots and the number of variables eliminated at the son is equal to the difference between the son's row count before elimination and the father's before elimination, there is no extra fill-in if they are amalgamated. The frontal matrix for the father node has the pattern of the element generated by the son and there is no advantage in treating them separately. This is particularly likely to happen near the root because the reduced matrix is often full in the final stages.

Similarly, a father-son pair of tiles can be amalgamated without extra fill-in if the counts for the rows with nonzero diagonal entries differ by the number of variables eliminated and the counts for the rows with zero diagonal entries differ by half the number of variables eliminated. A simple example is shown in Figure 2.4. Here there is no fill-in and the row counts at the time of elimination are (5,3) and (3,2).

$$\begin{array}{cc|cc|c}
\times & \times & \times & \times & \times \\
\times & & \times & & \times \\
\hline
\times & \times & \times & \times & \times \\
\times & & \times & & \times \\
\hline
\times & \times & \times & \times & \times
\end{array}$$

Figure 2.4. A father-son pair of tile pivots for which amalgamation is possible.

To see if a father-son pair of oxos can be amalgamated without extra fill-in, we test the differences for each of the halves against half the number of variables eliminated. Unfortunately, when the row counts for the two halves are identical, we cannot be sure that the two halves of the son do not need to be interchanged if extra fill-in is to be avoided. A simple case is shown in Figure 2.5. where variables 1 and 2 are eliminated at the son node and variables 3 and 4 at the father node. Checking for such an event requires more information at the nodes than is available at this time. We therefore do not amalgamate oxo nodes where the row counts of the two halves are the same.

$$\begin{array}{cc|cc|c}
 & \times & \times & & \times \\
\times & & & \times & \times \\
\hline
\times & & & \times & \times \\
 & \times & \times & & \times \\
\hline
\times & \times & \times & \times & \times
\end{array}$$

Figure 2.5. A father-son pair of oxo pivots for which amalgamation is possible if the variables of the father are interchanged.

We also perform some amalgamation that does involve extra fill-in because of the advantages of reasonably large block sizes, particularly on a vector or parallel computer. Interfering with a structured pivot may lead to greatly increased fill-in, so we do this only for father-son pairs that are full. Also, we require that all ancestors be full, too, to avoid indirectly affecting a structured pivot. This leads us to using a depth-first search to look for father-son amalgamations. These amalgamations are controlled by a parameter (with default value 5) which is a limit on the number of variables at a node that is amalgamated with another.

2.3 Factorization

The factorization is controlled by the assembly tree created at the end of the analyse phase. For stability, all the pivots are tested numerically. If the updated entries of the fully-summed rows in the front are f_{ij}, the test for a 1×1 pivot is

$$|f_{kk}| \geq u \max_{j\neq k} |f_{kj}|, \tag{2.20}$$

where u is a pivot ***threshold parameter*** given a default value during initialization (see discussion in Section 3), and the test for a 2×2 pivot is

$$\left|\begin{pmatrix} f_{kk} & f_{k,k+1} \\ f_{k+1,k} & f_{k+1,k+1}\end{pmatrix}^{-1}\right| \begin{pmatrix} \max\limits_{j\neq k,\, k+1} |f_{kj}| \\ \max\limits_{j\neq k,\, k+1} |f_{k+1,j}| \end{pmatrix} \leq \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}. \tag{2.21}$$

For a tile pivot, it is possible for this test to fail and yet taking its two diagonal entries in turn as 1×1 pivots would lead to two 1×1 pivots that satisfy inequality (2.20). Using this pair is mathematically equivalent. We therefore accept the tile pivot in this case, too. For the second 1×1 pivot, we test the inequality

$$\left| f_{k+1,k+1} - \frac{f_{k+1,k}^2}{f_{k,k}} \right| \geq u \max_{j\neq k,\, k+1} \left| f_{k+1,j} - \frac{f_{k+1,k}}{f_{k,k}} f_{kj} \right|. \tag{2.22}$$

As well as the relative tests we have just described, we also apply an absolute test on the size of a 1×1 pivot or the off-diagonal entry of a 2×2 pivot. By default, the value of this ***pivot tolerance*** is zero.

These numerical tests may mean that some rows that we expected to eliminate during a block pivotal step remain uneliminated at the end of the step. These rows are stored alongside the generated element for treatment at the father node. We call these rows ***fully-summed*** since we know that there are no contributions to be added from elsewhere, unlike the rows of the generated element. At the father node, the possible pivot rows consist of old fully-summed rows coming from this son and perhaps other sons, too, and the new fully-summed rows that were recommended as pivots by the analyse phase.

If a full block pivot was recommended, we choose a simple 1×1 or 2×2 pivot and perform the corresponding elimination operations on the pivot rows before choosing the next simple 1×1 or 2×2 pivot. Both old and new fully-summed rows are candidates. We know of no other strategy for ensuring that the block pivot as a whole is satisfactory. Note, however, that the calculations for the Schur update can be delayed and performed as a block operation once all pivots are chosen.

If a structured block was recommended, the analyse phase expects that the new fully-summed rows have the form

$$\begin{pmatrix} \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_5 & \mathbf{A}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_1^T & \mathbf{0} & \mathbf{0} & \mathbf{A}_7 & \mathbf{A}_8 & \mathbf{0} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \mathbf{0} & \mathbf{A}_1 & \mathbf{A}_5 & \mathbf{A}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_1^T & \mathbf{0} & \mathbf{0} & \mathbf{A}_7 & \mathbf{A}_8 & \mathbf{0} \end{pmatrix} \tag{2.23}$$

after suitable permutations. We need to check that the potential pivots (leading two block columns in 2.22) still have this form since earlier changes to the pivotal sequence may destroy it.

Assuming that the form is still there, we again choose simple pivots one at a time and perform the corresponding elimination operations on the pivot rows before choosing the next simple pivot. We restrict these pivots to 2×2 pivots of the desired form in the new fully-summed rows, noting that each is identified by an entry of the $\mathbf{A}_1$ block. We therefore search $\mathbf{A}_1$ or its reduced form, again using the test (2.21) for stability.

If any fully-summed rows remain in the front after completion of this sequence of simple 2×2 pivot operations, we look for simple 1×1 and 2×2 pivots in these rows, exactly as for the case when full pivots were recommended. Note that these rows may be new fully-summed rows in which we failed to find a structured pivot or old fully-summed rows from the sons of the node. If any 1×1 or 2×2 full pivots are chosen, we regard the generated element as full, but by forming the Schur update for the rows of the structured pivots separately, we can at least take some advantage of the zero block or blocks within it.

Provided enough pivots are selected in the block pivot step, we use Level 3 BLAS (Dongarra, Du Croz, Duff, and Hammarling 1990) for constructing the Schur update. In the case of a full pivot, the Schur update is

$$\mathbf{S} \;=\; \mathbf{M}^T\mathbf{D}\,\mathbf{M} \tag{2.24}$$

where $\mathbf{M}$ is a rectangular matrix and $\mathbf{D}$ is a diagonal matrix with blocks of order 1 and 2 (see equation (2.10)). Within the frontal matrix, we hold a compact representation of $\mathbf{M}$ that excludes zero columns. Therefore, our real concern is the efficient formation of the product (2.24) for a full rectangular matrix $\mathbf{M}$. Unfortunately, there are no BLAS routines for forming a symmetric matrix as the product of two rectangular matrices, so we cannot form $\mathbf{D}\,\mathbf{M}$ and use a BLAS routine for calculating $\mathbf{M}^T(\mathbf{D}\,\mathbf{M})$ without doing about twice as much work as necessary. We therefore choose a block size b (with default size 5 set by the initialization routine) and divide the rows of $\mathbf{M}$ and $\mathbf{D}\,\mathbf{M}$ into strips that start in rows 1, b+1, 2b+1, ... This allows us to compute the block upper triangular part of each corresponding strip of $\mathbf{S}$ in turn, using `SGEMM` for each strip except the last. For the last (undersize) strip, we use simple Fortran code and take full advantage of the symmetry. Note that if $b>n$, simple Fortran code will be used all the time.

For an oxo pivot, the matrix $\mathbf{D}\,\mathbf{M}$ has the form

$$\mathbf{D}\,\mathbf{M} \;=\; \begin{pmatrix} \mathbf{L}_1 & \\ & \mathbf{U}_1^T \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{A}_5 & \mathbf{A}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_7 & \mathbf{A}_8 & \mathbf{0} \end{pmatrix} \;=\; \begin{pmatrix} \mathbf{M}_5 & \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix} \tag{2.25}$$

and the matrix $\mathbf{M}$ has the form

$$\mathbf{M} \;=\; \begin{pmatrix} & \mathbf{D}_1^{-1} \\ \mathbf{D}_1^{-1} & \end{pmatrix} \begin{pmatrix} \mathbf{M}_5 & \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix} \;=\; \begin{pmatrix} \mathbf{0} & \mathbf{M}_9 & \mathbf{M}_{10} & \mathbf{0} \\ \mathbf{M}_{11} & \mathbf{M}_{12} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{2.26}$$

We may form the Schur update

$$\begin{pmatrix} \mathbf{0} & \mathbf{B}_2 & \mathbf{B}_3 & \mathbf{0} \\ \mathbf{B}_2^T & \mathbf{B}_1 & \mathbf{B}_4 & \mathbf{0} \\ \mathbf{B}_3^T & \mathbf{B}_4^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{M}_{11}^T \\ \mathbf{M}_9^T & \mathbf{M}_{12}^T \\ \mathbf{M}_{10}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{M}_5 & \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix} \tag{2.27}$$

by the calculations

$$\begin{aligned} \begin{pmatrix} \mathbf{B}_2 & \mathbf{B}_3 \end{pmatrix} &= \mathbf{M}_{11}^T \begin{pmatrix} \mathbf{M}_7 & \mathbf{M}_8 \end{pmatrix}, \\ \mathbf{B}_4 &= \mathbf{M}_{12}^T \mathbf{M}_8, \\ \mathbf{B}_1 &= \begin{pmatrix} \mathbf{M}_9^T & \mathbf{M}_{12}^T \end{pmatrix} \begin{pmatrix} \mathbf{M}_6 \\ \mathbf{M}_7 \end{pmatrix}. \end{aligned} \tag{2.28}$$

We may use the BLAS 3 routine `SGEMM` directly for the first two calculations. For the symmetric matrix $\mathbf{B}_1$, we subdivide the computation into strips, as for the full-pivot case.

For a tile pivot, the sparsity in the first block column of the above form is lost:

$$\mathbf{D}\mathbf{M} = \begin{pmatrix} \mathbf{L}_1 & \\ \mathbf{U}_2^T & \mathbf{U}_1^T \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{A}_5 & \mathbf{A}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_7 & \mathbf{A}_8 & \mathbf{0} \end{pmatrix}. \tag{2.29}$$

We therefore amalgamate the first two blocks to give

$$\mathbf{D}\mathbf{M} = \begin{pmatrix} \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix} \tag{2.30}$$

$$\mathbf{M} = \begin{pmatrix} -\mathbf{D}_1^{-1}\mathbf{D}_2\mathbf{D}_1^{-1} & \mathbf{D}_1^{-1} \\ \mathbf{D}_1^{-1} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{M}_9 & \mathbf{M}_{10} & \mathbf{0} \\ \mathbf{M}_{12} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{2.31}$$

The Schur update calculation is therefore as in the oxo case except that the first block row and column is not present and we have only the calculations for $\mathbf{B}_1$ and $\mathbf{B}_4$.

2.4 Solution

The solution is conveniently performed in two stages. The first, forward substitution, consists of solving

$$(\mathbf{P}\mathbf{L}\mathbf{P}^T)\mathbf{y} = \mathbf{b} \tag{2.32}$$

and the second, back-substitution, consists of solving

$$(\mathbf{P}\mathbf{D}\mathbf{L}^T\mathbf{P}^T)\mathbf{x} = \mathbf{y}. \tag{2.33}$$

For the first step of the forward substitution, let

$$\mathbf{P}^{-1}\mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} \tag{2.34}$$

where $\mathbf{b}_1$ corresponds to the block pivot and $\mathbf{b}_2$ corresponds to the rest of the first front. We need to solve the equation

$$\begin{pmatrix} \mathbf{L} & & \\ \mathbf{M}^T & \mathbf{I} & \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{b}'_1 \\ \mathbf{b}'_2 \\ \mathbf{b}'_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix}. \tag{2.35}$$

This involves the forward substitution

$$\mathbf{L}\,\mathbf{b}'_1 = \mathbf{b}_1 \tag{2.36}$$

and the modification

$$\mathbf{b}'_2 = \mathbf{b}_2 - \mathbf{M}^T\mathbf{b}'_1. \tag{2.37}$$

In the case of a full pivot, we can employ the Level 2 BLAS routine `STPSV` in equation (2.36) (we pack the triangular array $\mathbf{L}$ to save storage) and the Level 2 BLAS routine `SGEMV` in equation (2.37). For a structured pivot, it is slightly more complicated. Now $\mathbf{L}$ has the form $\begin{pmatrix} \mathbf{L}_1 & \\ \mathbf{U}_2^T & \mathbf{U}_1^T \end{pmatrix}$, so solving equation (2.36) requires two applications of `STRSV` (here, we pack the arrays $\mathbf{L}_1$, $\mathbf{D}_1$ and $\mathbf{U}_1$ together in a square array) and one application of `STPMV.` Also, $\mathbf{M}$ has the form $\begin{pmatrix} \mathbf{M}_5 & \mathbf{M}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_7 & \mathbf{M}_8 & \mathbf{0} \end{pmatrix}$ so two applications of `SGEMV` are needed for equation (2.37).

Similar considerations apply to the other steps and to the back-substitution, too.

2.5 Recognition of supervariables

The analyse phase recognizes sets of structurally identical rows in the original matrix and forms supervariables. With careful choice of data structures, the whole process can be made to execute in $O(n+\tau)$ time, where τ is the number of entries. We work progressively so that after j steps we have the supervariable structure for the submatrix of the first j columns. We start with all variables in one supervariable (for the submatrix with no columns), then split it into two according to which rows do or do not have an entry in column 1, then split these according to the entries in column 2, etc. The splitting is done by moving the variables one at a time to the new supervariable.

The variables of each supervariable are held in a circular chain. We use two integer arrays of length n to hold links from each variable to the next and previous variable in its chain. This allows rapid removal of a variable from a supervariable and rapid insertion of a variable in the chain that holds another variable. We begin with all the variables linked in a single chain.

We use other integer arrays of length n as follows:–

svar(i) is the index of the supervariable to which variable i belongs.

flag is initially set to zero. *flag*(s) is set to j when supervariable s is encountered in column j.

var(s) holds the index of a variable that was in supervariable s.

Using this data structure, the details of our algorithm are as follows:–

```
for $j$ := 1 step 1 until $n$ do
    for each entry $i$ in column $j$ do
        $s = svar(i)$ ! $s$ is $i$'s old supervariable
        if $flag(s) < j$ then ! first occurrence of $s$ for column $j$
            $flag(s) = j$
            if $s$ has more than one variable then
                remove $i$ from $s$
                create a new supervariable $ns$ with $i$ as its only variable
                $svar(i) = ns$
                $var(s) = i$
            end if
        else ! second or later occurrence of $s$ for column $j$
            $k = var(s)$ ! $k$ is the first variable of $s$ encountered in column $j$
            $svar(i) = svar(k)$
            move $i$ from its present chain to the chain containing $k$
        end if
    end do
end do
```

The elements of *var* that do not correspond to supervariables may be employed to hold a chain of indices not currently in use for supervariables.

3 Numerical Experience

3.1 Introduction

In this section, we examine the performance of the `MA47` code on a range of test problems on a SUN SPARCstation 10 and a CRAY Y-MP. We study the influence of various parameter settings on the performance of `MA47` and determine the values for the defaults. We also compare the `MA47` code with the code `MA27`.

As always, the selection of test problems is a compromise between choosing sufficient to obtain meaningful statistics while keeping run times and this section of the report manageable. In Tables 1 and 2, we list the test problems used for the numerical experiments in this section. In choosing this set, many runs were performed on other matrices so we do feel that this selection is broadly representative of a far larger set. Our set of 22 matrices can be divided into three distinct sets. The first (matrices 1 to 10), in Table 1, are obtained from the CUTE collection of nonlinear optimization problems (Bongartz, Conn, Gould, and Toint, 1993). The problems in that set are parameterized, and we have chosen the parameters to obtain a linear problem of order between about 1000 and 20000. In each case, we solve a linear system whose coefficient matrix is the Kuhn-Tucker matrix of the form

$$\begin{pmatrix} \mathbf{H} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0} \end{pmatrix},$$

where **H** is an $m \times m$ symmetric matrix and **A** is of dimension $m \times n$. We are grateful to Ali Bouaricha and Nick Gould for extracting these matrices for us from CUTE. The second and third sets are obtained by forming augmented systems of the form

$$\begin{pmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mathbf{D} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0} \end{pmatrix}, \quad \text{with} \quad \mathbf{D} = \begin{pmatrix} \mathbf{I} & \\ & \mathbf{0} \end{pmatrix}$$

respectively, where the matrix **A** is from the Harwell-Boeing Collection (Duff, Grimes, and Lewis, 1992) or the netlib LP collection (Gay, 1985). The matrix **D** has $m–n$ unit diagonal entries and n zeros on the diagonal. We use the six matrices from these collections that are shown in Table 2, which gives us another twelve test cases according to the two forms of augmentation shown above. In all cases, the dimensions are given in the tables and are the number of rows in **A** and the total order of the augmented system. The number of entries is the total number in the upper triangular part of the augmented matrix.

Case	Identifier	Order m	Order $m+n$	Number of entries	Description
1	BRITGAS	3102	5802	15282	British gas pipe network distribution problem.
2	BIGBANK	2230	3342	8056	Nonlinear network problem.
3	MINPERM	8347	16551	16863	Minimize the permanent of a doubly stochastic matrix.
4	SVANBERG	14000	21000	91000	Structural optimization.
5	BRATU2D	5184	10084	29684	Finite-difference discretization of nonlinear PDE on unit square.
6	BRATU3D	4913	8288	28538	Finite-difference discretization of nonlinear PDE on unit cube.
7	GRIDNETC	7564	11408	30256	A nonlinear network problem on a square grid.
8	QPCSTAIR	614	970	4617	STAIR LP with additional convex Hessian.
9	KSIP	1021	2022	22023	Discretization of a semi-infinite QP.
10	AUG3DQP	3873	4873	10419	QP from nine-point formulation of 3-D PDE.

Table 1. The CUTE matrices used for performance testing.

Case	Identifier	Order m	Order $m+n$	Number of entries	Description
11/17	FFFFF800	1028	1552	7429	LP. Oil industry.
12/18	PILOT	4860	6301	40235	LP. Energy model from Stanford.
13/19	ORSIRR 2	886	1772	6861	Oil reservoir simulation.
14/20	JPWH 991	991	1982	7018	Circuit physics modelling.
15/21	BCSSTK27	1224	2448	29899	Structural engineering. Buckling analysis.
16/22	NNC1374	1374	2748	9980	Nuclear reactor core modelling.

Table 2. Matrices used for augmented systems. Matrices 11 to 16 have **I** as leading block and matrices 17 to 22 have **D**.

Before conducting a systematic study of the parameters, we first experimented to see the effect of using the extended Markowitz cost computed in the analyse phase (see equations (2.17), (2.18), and (2.19)) to decide whether to allow prospective pivots in the factorize phase. The logic of this scheme, which was proposed by Duff, Reid, Gould, Scott, and Turner (1991), is that changes to the pivotal sequence from the analyse phase may mean that later pivots in their recommended position in the pivotal sequence would be poor choices. There is, however, a counter-balancing effect whereby even if the pivot is very much poorer on sparsity grounds than predicted, it may be worse to delay using it because of further build up of unanticipated fill-in. In effect, it may be better to "bite the bullet" and take the bad pivot early rather than late.

When we tested this option on the examples in Table 1, we found that there was usually little to choose between including the Markowitz test or not. However, in most cases it was slightly worse to use the Markowitz test and, in one case, the factorization time was increased by nearly a factor of 14 and the total storage for the factorization by nearly a factor of 6. Since there were no examples so dramatically favouring the Markowitz test, we have decided to drop it from the code. We have, however, left in the structure for the test and have only commented out the test itself. We believe that possible future changes to the way we handle fully summed blocks might make the test useful in a later version of the code.

In our program for running the numerical experiments, we have an option to prescale the matrices using Harwell Subroutine `MC30`. In nearly all the cases, there is very little difference in performance or accuracy whether scaling is used or not. There were, however, two cases where additional pivoting on the unscaled systems caused a significant increase in time and storage for the factorization, with only one case significantly the other way. We feel more comfortable assessing the performance on scaled systems, so we use this option in all the runs in this paper.

In the following subsections, we examine the relative performance when a single parameter is changed by means of the median, upper-quartile, and lower-quartile ratios over the 22 test problems. We use these values rather than means and variances to give some protection against stray results caused either by the timer or by particular features of the problems. We remind the reader that half the results lie between the quartile values. Full tables of ratios are available by anonymous ftp from numerical.cc.rl.ac.uk (130.246.8.23) in the file pub/reports/ma47.tables. In Section 3.2, we consider the effect of choosing the option to restrict the pivot choice to a small number of columns, and we examine the effect of altering the pivot threshold in Section 3.3. We study the effect of changing the amount of node amalgamation in Section 3.4. In Sections 3.5 and 3.6, we examine the use of higher level BLAS; the Level 3 BLAS in the factorization in the first section and the use of Level 2 BLAS in the solution phase in the second. Finally, in Section 3.7, we show the performance of our code with default parameter values on the test problems and compare it with the Harwell Subroutine Library code `MA27`.

3.2 Effect of restricting pivot selection

If we restrict the number of rows that we search to find a pivot, we might expect to reduce the time for the analyse phase at the cost, perhaps, of more storage and time in the factorization. Clearly, the choice depends on the relative importance of the phases. We show the results of runs varying the search limit in Tables 3 to 5. The times in these tables are in seconds on the SUN SPARCstation 10.

	Total storage		Storage for factors		Time (SUN)			
	Predicted	Actual	Predicted	Actual	Analyse	Factorize	Solve	One-off
lower q.	0.97	1.00	0.98	1.00	0.95	0.98	0.97	0.97
median	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00
upper q.	1.00	1.04	1.00	1.02	1.01	1.06	1.03	1.03

Table 3. Results with search limit of 2 rows divided by those with limit of 4.

	Total storage		Storage for factors		Time (SUN)			
	Predicted	Actual	Predicted	Actual	Analyse	Factorize	Solve	One-off
lower q.	1.00	0.97	1.00	0.99	0.98	0.96	0.99	0.97
median	1.00	1.00	1.00	1.00	0.99	1.01	1.00	1.01
upper q.	1.08	1.09	1.02	1.01	1.01	1.10	1.03	1.07

Table 4. Results with search limit of 4 rows divided by those with limit of 10.

	Total storage		Storage for factors		Time (SUN)			
	Predicted	Actual	Predicted	Actual	Analyse	Factorize	Solve	One-off
lower q.	1.00	1.00	0.97	1.00	0.73	0.95	0.96	0.94
median	1.00	1.00	1.00	1.00	0.95	1.00	0.98	0.99
upper q.	1.06	1.76	1.00	1.35	1.00	1.86	1.28	1.43

Table 5. Results with search limit of 10 divided by those with no search limit.

The medians are near 1.0, showing slight gains to the analyse time by restricting the pivot search at a cost of a slightly more expensive factorization. The costs for a “one-off” run of a single analysis followed by one factorization and solution, show that there is really quite a balance in the competing trends. The upper quartile figures markedly support having no search limit (that is, using a Markowitz count), so we have decided to keep that as the default and use it for the later runs in this paper.

3.3 Effect of change in pivot threshold

In `MA27`, a value of 0.1 was chosen for the default value of the threshold parameter, u (see equations (2.20) and (2.21)), since smaller values gave little appreciable benefit to sparsity in the experiments that we conducted at that time. However, the more complicated data structures in `MA47` and the greater penalties for not being able to follow the pivotal sequence recommended by the analyse phase penalizes higher values of u to a greater extent than in the earlier code. We investigate this in Tables 6 to 10.

	Storage required		Time (SUN)		
	Total	Factors	Analyse	Factorize	Solve
lower q.	1.00	1.00	1.00	1.08	1.01
median	1.14	1.09	1.00	1.78	1.07
upper q.	2.14	1.53	1.00	3.32	1.42

Table 6. Results with threshold *u* set to 0.5 divided by those with *u* = 0.1.

	Storage required		Time (SUN)		
	Total	Factors	Analyse	Factorize	Solve
lower q.	1.00	1.00	0.96	0.99	1.00
median	1.02	1.05	0.98	1.07	1.03
upper q.	1.28	1.23	1.00	1.97	1.22

Table 7. Results with threshold *u* set to 0.1 divided by those with *u* = 0.01.

	Storage required		Time (SUN)		
	Total	Factors	Analyse	Factorize	Solve
lower q.	1.00	1.00	0.94	1.00	0.98
median	1.00	1.03	0.99	1.11	1.02
upper q.	1.14	1.16	1.03	1.63	1.12

Table 8. Results with threshold *u* set to 0.01 divided by those with *u* = 0.001.

	Storage required		Time (SUN)		
	Total	Factors	Analyse	Factorize	Solve
lower q.	1.00	1.00	1.00	1.03	1.02
median	1.00	1.00	1.03	1.08	1.07
upper q.	1.07	1.05	1.08	1.42	1.10

Table 9. Results with threshold *u* set to 0.001 divided by those with *u* = 0.0001.

	Storage required		Time (SUN)		
	Total	Factors	Analyse	Factorize	Solve
lower q.	1.00	1.00	0.99	1.00	0.99
median	1.00	1.00	1.00	1.01	1.00
upper q.	1.01	1.02	1.00	1.09	1.03

Table 10. Results with threshold *u* set to 0.0001 divided by those with *u* = 0.00001.

We also, of course, monitored the numerical performance for all these runs. Although the results from using a threshold value of 0.5 were better than 0.1 for a couple of test problems, notably the NNC1374 example, it was substantially more expensive to use such a high value for the threshold. For lower values of the threshold, the scaled residual was remarkably flat for all of the test problems until values of the threshold less than 10^{-6} when poorer results were obtained on three of the examples.

From the performance figures in Tables 6 to 10, the execution times and storage decline almost monotonically but have begun to level off by about 0.0001 (although there are a couple of outliers). However, we are anxious not to compromise stability and recognize that our numerical experience is necessarily limited. We have therefore decided to choose as default a threshold value of 0.001 since, for many problems, much of the sparsity benefit has been realized at this value.

3.4 Effect of node amalgamation

The node amalgamation parameter, discussed at the end of Section 2.2, controls the amalgamation of neighbouring nodes in the assembly tree to obtain larger blocks (and more eliminations within each node) at the cost of extra fill-in and arithmetic. No node is amalgamated with another unless its number of variables is less than the parameter value. This feature was also present in the `MA27` code. One intention of performing more amalgamations is that there should be more scope for the use of Level 3 BLAS, which might be expected to benefit platforms with efficient Level 3 BLAS kernels.

We show the results of running with various levels of amalgamation in Tables 11 to 13. As expected, there is a difference of performance between the two machines. On the CRAY, a higher amount of amalgamation is beneficial. On the SUN, there is a slight gain from some amalgamation, but the effect is reversed before the amalgamation parameter reaches 10. In the interests of choosing a default value that is satisfactory on several platforms (even if not optimal), we have chosen the value 5. Note that people running extensively on a vector machine, like a CRAY, may wish to increase this (say to 20).

		Total storage		Storage for factors		Time			
		Predicted	Actual	Predicted	Actual	Analyse	Factorize	Solve	One-off
CRAY	lower q.	0.78	0.88	0.77	0.84	1.00	1.00	0.73	1.00
	median	0.95	0.95	0.87	0.92	1.01	1.13	0.83	1.03
	upper q.	1.00	1.00	1.00	1.00	1.03	1.37	1.00	1.10
SUN	lower q.	0.78	0.90	0.77	0.84	1.00	1.00	0.82	1.00
	median	0.95	0.95	0.87	0.92	1.02	1.03	0.92	1.01
	upper q.	1.00	1.00	1.00	1.00	1.03	1.11	1.00	1.07

Table 11. Results with no amalgamation divided by those with parameter 5.

		Total storage		Storage for factors		Time			
		Predicted	Actual	Predicted	Actual	Analyse	Factorize	Solve	One-off
CRAY	lower q.	0.84	0.85	0.80	0.81	1.00	1.00	1.00	1.00
	median	0.91	0.96	0.87	0.92	1.00	1.02	1.11	1.01
	upper q.	1.00	1.00	1.00	1.00	1.01	1.14	1.25	1.05
SUN	lower q.	0.84	0.86	0.80	0.81	0.98	0.95	0.91	0.96
	median	0.91	0.96	0.87	0.92	0.98	0.97	0.96	0.98
	upper q.	1.00	1.00	1.00	1.00	1.00	1.00	1.01	0.99

Table 12. Results with amalgamation parameter set to 5 divided by those with parameter 10.

		Total storage		Storage for factors		Time			
		Predicted	Actual	Predicted	Actual	Analyse	Factorize	Solve	One-off
CRAY	lower q.	0.82	0.83	0.77	0.80	1.00	1.00	1.00	1.00
	median	0.89	0.94	0.86	0.88	1.00	1.00	1.10	1.00
	upper q.	1.00	1.00	1.00	1.00	1.01	1.05	1.22	1.01
SUN	lower q.	0.82	0.83	0.77	0.80	1.00	0.87	0.91	0.98
	median	0.89	0.94	0.86	0.87	1.02	0.98	0.98	1.00
	upper q.	1.00	1.00	1.00	1.00	1.05	1.03	1.00	1.03

Table 13. Results with amalgamation parameter set to 10 divided by those with parameter 20.

3.5 Effect of change of block size for Level 3 BLAS during factorization

A major benefit of multifrontal methods is that the floating-point arithmetic is performed on dense submatrices. In particular, if we perform several pivot steps on a particular frontal matrix, Level 3 BLAS can be used. However, in the present case, we also wish to maintain symmetry and the current Level 3 BLAS suite does not have an appropriate kernel. We thus, as discussed in Section 2.3, need to split the frontal matrix into strips, starting at rows 1, b+1, 2b+1, ..., so that we can use Level 3 BLAS without doubling the arithmetic count. In fact, in a block of size b on the diagonal, the extra work is $b*(b-1)$ floating-point operations. Clearly this means that, while we would like to increase b for Level 3 BLAS efficiency, by doing so we increase the amount of arithmetic. In this section, we examine the trade off between these competing trends.

We show results for various values of the block-size parameter, b, in Tables 14 to 16. It would seem, from these results, that a modest value is best and we choose 5 as the default value on the basis of these figures.

		Analyse	Factorize	Solve	One-off
CRAY	lower q.	1.00	1.01	0.99	1.00
	median	1.00	1.05	1.00	1.01
	upper q.	1.00	1.14	1.00	1.03
SUN	lower q.	1.00	1.00	0.98	0.99
	median	1.01	1.02	1.00	1.01
	upper q.	1.01	1.06	1.02	1.03

Table 14. Times with block-size parameter b set to 1 divided by those with b = 5.

		Analyse	Factorize	Solve	One-off
CRAY	lower q.	1.00	0.95	1.00	0.98
	median	1.00	0.98	1.00	0.99
	upper q.	1.00	1.00	1.00	1.00
SUN	lower q.	0.99	0.99	0.99	0.99
	median	1.00	0.99	1.00	0.99
	upper q.	1.01	1.01	1.01	1.00

Table 15. Times with block-size parameter b set to 5 divided by those with b = 10.

		Analyse	Factorize	Solve	One-off
CRAY	lower q.	1.00	0.73	1.00	0.91
	median	1.00	0.90	1.00	0.97
	upper q.	1.00	1.00	1.00	1.00
SUN	lower q.	0.98	0.92	0.98	0.94
	median	1.00	0.96	0.99	0.98
	upper q.	1.01	0.99	1.00	1.00

Table 16. Times with block-size parameter b set to 10 divided by those with b = 20.

We show in Table 17 a comparison between results with $b=5$ and $b>n$, which corresponds to no blocking and the use of simple Fortran code. Interestingly, the results are almost identical.

		Analyse	Factorize	Solve	One-off
CRAY	lower q.	1.00	0.97	1.00	0.99
	median	1.00	0.99	1.00	1.00
	upper q.	1.00	1.01	1.02	1.00
SUN	lower q.	1.00	0.97	0.98	0.99
	median	1.00	1.00	1.00	1.00
	upper q.	1.03	1.02	1.03	1.01

Table 17. Times with block-size parameter b set to 5 divided by those with $b > n$.

3.6 Effect of change of block size for level 2 BLAS during solution

In a single block pivot stage of the solution phase, one can use indirect addressing for every operation. Alternatively, one can load the appropriate entries of the right-hand side vector into a small full vector corresponding to the rows in the current front, update this vector with Level 2 BLAS operations, and finally scatter it back to the full vector.

We have experimented with the parameter that determines whether to use indirect or direct addressing in the solution phase. Direct addressing is used (and Level 2 BLAS called) if the number of pivots at a step is more than this parameter. Thus, for high values of the parameter, there will be less use of Level 2 BLAS. We show a summary of our results in Table 18. As can be seen, the results are quite flat. On the largest of our problems, there was some gain by using a value of 4 for the Level 2 blocking and so we have chosen that as our default.

Parameter ratio		1:2	2:4	4:8	8:16
CRAY	lower q.	1.00	1.00	1.00	1.00
	median	1.04	1.03	1.03	1.00
	upper q.	1.10	1.06	1.13	1.07
SUN	lower q.	0.97	0.99	0.94	0.94
	median	1.00	1.00	0.98	0.98
	upper q.	1.02	1.03	1.06	1.04

Table 18. Ratios of solve times with different values for the parameter for indirect addressing.

3.7 Performance of MA47 and comparison with MA27

In the past five subsections, we have considered the effect of various controlling parameters on the performance of `MA47`. We now examine the performance of our code with the default values for the parameters on both the SUN SPARCstation 10 and the CRAY Y-MP. The storage counts and times for the problems of Tables 1 and 2 are shown in Tables 18 and 19. It was our original intention that this new `MA47` code would replace `MA27` in the Harwell Subroutine Library. However, the added complexity of the new code will penalize it if it is unable to take advantage of the structure. We thus might expect that sometimes `MA47` would be better and sometimes `MA27`. We illustrate this by showing the comparison ratios for the two codes in Tables 20 and 21.

Case	Storage (millions of words) Total Predicted	Total Actual	For factors Predicted	For factors Actual	Time Analyse	Time Factorize	Time Solve
1 BRITGAS	0.52	0.59	0.28	0.34	12.40	10.96	0.16
2 BIGBANK	0.09	0.09	0.07	0.07	1.22	0.37	0.04
3 MINPERM	0.12	0.12	0.09	0.09	0.60	0.45	0.13
4 SVANBERG	1.07	0.89	0.94	0.77	12.43	3.17	0.36
5 BRATU2D	1.22	1.22	0.95	0.95	80.35	8.29	0.33
6 BRATU3D	4.09	3.94	2.32	2.18	18.00	70.53	0.65
7 GRIDNETC	0.43	0.43	0.36	0.36	1.62	1.45	0.19
8 QPCSTAIR	0.13	0.14	0.07	0.08	0.44	0.80	0.03
9 KSIP	0.14	0.70	0.11	0.20	4.68	19.28	0.07
10 AUG3DQP	0.29	0.29	0.20	0.20	0.70	1.16	0.09
11 FFFFF800	0.18	0.17	0.12	0.09	1.24	1.08	0.04
12 PILOT	1.58	1.74	0.80	0.84	9.80	15.07	0.28
13 ORSIRR 2	0.44	0.44	0.24	0.25	1.48	2.87	0.08
14 JPWH 991	0.84	0.94	0.43	0.35	1.57	6.99	0.11
15 BCSSTK27	0.18	0.18	0.10	0.10	0.94	0.21	0.04
16 NNC1374	0.38	1.32	0.32	0.48	1.88	29.89	0.16
17 FFFFF800	0.04	0.06	0.03	0.04	0.77	1.29	0.02
18 PILOT	0.29	0.29	0.16	0.19	4.97	2.76	0.10
19 ORSIRR 2	0.28	0.93	0.13	0.30	1.21	25.14	0.11
20 JPWH 991	0.57	0.99	0.14	0.26	2.37	24.08	0.10
21 BCSSTK27	0.17	0.17	0.10	0.10	0.98	0.22	0.04
22 NNC1374	0.11	0.25	0.09	0.14	2.47	5.54	0.06

Table 19. Performance of runs of MA47 code on the SUN SPARC-10.

Case	Storage (millions of words) Total Predicted	Total Actual	For factors Predicted	For factors Actual	Time Analyse	Time Factorize	Time Solve
1 BRITGAS	0.52	0.59	0.28	0.35	9.63	3.46	0.06
2 BIGBANK	0.09	0.09	0.07	0.07	0.99	0.19	0.03
3 MINPERM	0.12	0.12	0.09	0.09	16.27	0.35	0.09
4 SVANBERG	1.07	1.07	0.94	0.95	10.06	1.63	0.27
5 BRATU2D	1.22	1.22	0.95	0.95	75.64	1.79	0.12
6 BRATU3D	4.09	3.94	2.32	2.18	16.69	7.17	0.11
7 GRIDNETC	0.43	0.43	0.36	0.36	1.59	0.71	0.15
8 QPCSTAIR	0.13	0.14	0.07	0.08	0.39	0.28	0.01
9 KSIP	0.14	0.70	0.11	0.20	6.24	2.97	0.02
10 AUG3DQP	0.29	0.29	0.20	0.20	0.58	0.41	0.06
11 FFFFF800	0.18	0.17	0.12	0.09	1.16	0.35	0.02
12 PILOT	1.58	1.74	0.80	0.84	9.06	2.78	0.08
13 ORSIRR 2	0.44	0.44	0.24	0.25	1.38	0.56	0.02
14 JPWH 991	0.84	0.94	0.43	0.35	1.30	1.25	0.02
15 BCSSTK27	0.18	0.18	0.10	0.10	0.91	0.16	0.02
16 NNC1374	0.38	1.32	0.32	0.47	1.51	2.43	0.03
17 FFFFF800	0.04	0.06	0.03	0.04	0.78	0.68	0.01
18 PILOT	0.29	0.29	0.16	0.19	3.94	1.63	0.05
19 ORSIRR 2	0.28	0.93	0.13	0.30	1.17	3.38	0.02
20 JPWH 991	0.57	0.99	0.14	0.26	2.39	4.92	0.02
21 BCSSTK27	0.17	0.17	0.10	0.10	0.92	0.17	0.02
22 NNC1374	0.11	0.23	0.09	0.14	2.00	1.51	0.03

Table 20. Performance of runs of MA47 code on the CRAY Y-MP.

Case		Total storage	Storage for factors		Time			
		Predicted	Predicted	Actual	Analyse	Factorize	Solve	One-off
1	BRITGAS	1.90	1.11	1.25	29.74	7.66	1.47	12.04
2	BIGBANK	1.49	1.31	1.29	4.52	1.86	1.69	3.31
3	MINPERM	1.01	1.12	1.12	1.85	1.31	2.95	1.66
4	SVANBERG	0.96	1.36	1.11	2.75	1.03	1.16	2.02
5	BRATU2D	1.36	1.15	0.47	83.26	0.23	0.55	2.36
6	BRATU3D	1.48	1.10	0.48	10.23	0.26	0.47	0.32
7	GRIDNETC	1.56	1.37	1.38	2.72	1.29	1.47	1.77
8	QPCSTAIR	2.52	1.57	1.64	3.52	3.17	1.53	3.20
9	KSIP	1.30	0.96	0.44	15.00	0.58	0.43	0.72
10	AUG3DQP	1.86	1.46	1.46	2.33	1.25	1.46	1.52
11	FFFFF800	1.80	1.63	1.01	3.00	1.42	1.09	1.95
12	PILOT	1.55	1.20	1.15	1.25	0.99	1.00	1.08
13	ORSIRR 2	1.61	1.15	0.42	4.11	0.26	0.52	0.38
14	JPWH 991	2.04	1.48	0.43	3.29	0.24	0.47	0.30
15	BCSSTK27	0.80	0.45	0.27	1.41	0.07	0.41	0.31
16	NNC1374	2.43	2.19	1.10	5.88	3.39	1.12	3.44
17	FFFFF800	0.41	0.40	0.39	1.83	1.14	0.59	1.31
18	PILOT	0.28	0.25	0.15	0.65	0.06	0.24	0.15
19	ORSIRR 2	1.04	0.61	0.50	3.39	2.31	0.72	2.32
20	JPWH 991	1.40	0.47	0.31	4.97	0.96	0.46	1.03
21	BCSSTK27	0.76	0.45	0.27	1.44	0.07	0.44	0.32
22	NNC1374	0.70	0.59	0.81	7.79	6.72	1.13	6.74
lquart		0.96	0.59	0.42	1.85	0.26	0.47	0.38
median		1.44	1.14	0.66	3.34	1.09	0.86	1.59
uquart		1.80	1.37	1.15	5.88	1.86	1.46	2.36

Table 21. Ratio of performance of `MA47` code to `MA27` code on the SUN SPARC-10.

Case		Total storage	Storage for factors		Time			
		Predicted	Predicted	Actual	Analyse	Factorize	Solve	One-off
1	BRITGAS	1.90	1.11	1.28	23.21	9.40	2.22	16.24
2	BIGBANK	1.49	1.31	1.29	3.08	1.70	1.89	2.69
3	MINPERM	1.01	1.12	1.12	59.17	1.42	2.53	30.05
4	SVANBERG	0.96	1.36	1.37	1.68	1.29	2.14	1.62
5	BRATU2D	1.36	1.15	0.47	71.49	0.51	1.93	16.71
6	BRATU3D	1.48	1.10	0.48	7.79	0.46	1.29	1.34
7	GRIDNETC	1.56	1.37	1.38	2.70	1.49	2.11	2.16
8	QPCSTAIR	2.52	1.57	1.64	2.35	3.03	1.83	2.57
9	KSIP	1.30	0.96	0.44	14.83	0.30	1.20	0.88
10	AUG3DQP	1.86	1.46	1.46	1.91	1.62	1.97	1.79
11	FFFFF800	1.80	1.63	1.01	2.03	2.05	1.80	2.03
12	PILOT	1.55	1.20	1.15	0.86	1.35	1.78	0.95
13	ORSIRR 2	1.61	1.15	0.42	3.00	0.54	1.83	1.30
14	JPWH 991	2.04	1.48	0.43	2.30	0.54	1.44	0.88
15	BCSSTK27	0.80	0.45	0.27	1.01	0.22	1.46	0.67
16	NNC1374	2.43	2.19	1.10	4.04	1.90	1.72	2.38
17	FFFFF800	0.41	0.39	0.38	1.36	3.14	1.44	1.84
18	PILOT	0.28	0.25	0.15	0.38	0.40	1.10	0.38
19	ORSIRR 2	1.04	0.61	0.51	2.53	3.33	1.67	3.07
20	JPWH 991	1.40	0.47	0.31	4.24	2.37	1.40	2.77
21	BCSSTK27	0.76	0.45	0.27	1.02	0.25	1.38	0.69
22	NNC1374	0.70	0.59	0.81	5.35	5.86	2.00	5.48
lquart		0.96	0.59	0.42	1.68	0.51	1.44	0.95
median		1.44	1.14	0.66	2.61	1.45	1.79	1.93
uquart		1.80	1.37	1.28	5.35	2.37	1.97	2.77

Table 22. Ratio of performance of `MA47` code to `MA27` code on the CRAY Y-MP.

We show the full results in these tables as well as the medians and quartiles since the performance can vary very widely. On the SUN, the new code factorizes matrix 18 (PILOT) over sixty times faster than `MA27` but is nearly 7 times slower than `MA27` on matrix 1 (BRITGAS). With the exception of matrix 18, the analyse phase times are always greater for the new code, once by over a factor of 80 (matrix 5, BRATU2D). The variation is only slightly less dramatic on the CRAY.

We have also run the codes on a set of ten Harwell-Boeing matrices with nonzero diagonal entries (BCSSTK14/15/16/17/18/26/27/28 and BCSSTM26/27) and the results are summarized in Table 23. On the SUN, `MA47` just outperforms `MA27` for the factorize and solve phases, but otherwise is inferior. We therefore recommend that where the matrix has nonzeros on the diagonal, `MA27` should continue to be used. We plan to make a new version of `MA27` that incorporates the BLAS and some other minor improvements. We anticipate that the revised `MA27` will always outperform `MA47` on this kind of matrix.

		Total storage	Storage for factors		Time			
		Predicted	Predicted	Actual	Analyse	Factorize	Solve	One-off
CRAY	lower q.	1.25	0.99	0.99	1.54	1.11	1.75	1.26
	median	1.34	1.03	1.02	1.56	1.15	1.93	1.29
	upper q.	1.38	1.07	1.04	1.80	1.21	2.00	1.30
SUN	lower q.	1.25	0.99	0.99	1.55	0.86	0.87	0.89
	median	1.34	1.03	1.02	1.70	0.94	0.94	0.98
	upper q.	1.38	1.07	1.04	1.98	1.02	0.98	1.06

Table 23. `MA47` to `MA27` ratios on 10 matrices with nonzero entries on the diagonal.

In these comparisons, we have used the same parameter settings (where applicable) for both codes. In particular, we have run `MA27` with a value for the threshold, *u*, of 0.001 which is 100 times less than the default value for `MA27`. However, it would appear empirically that the stability of `MA27` is more sensitive to the threshold value than `MA47`, so we have also compared `MA47` with `MA27` using its default threshold. A summary of the results in Table 24 indicates that, in general, `MA47` with its default outperforms `MA27` with its default on the SUN, but the variation in relative performance over different problem classes is still substantial. However, the roles are reversed on the CRAY due to the greater penalty for the extra integer manipulation in `MA47`. We were disappointed to find this outweighed the benefits of BLAS on the CRAY.

		Total storage	Storage for factors		Time			
		Predicted	Predicted	Actual	Analyse	Factorize	Solve	One-off
CRAY	lower q.	0.96	0.59	0.31	1.69	0.53	1.44	0.93
	median	1.45	1.15	0.57	2.64	1.38	1.82	1.80
	upper q.	1.86	1.46	1.21	5.41	2.04	2.18	3.13
SUN	lower q.	0.96	0.59	0.31	1.88	0.25	0.45	0.33
	median	1.45	1.15	0.57	3.32	0.96	0.75	1.38
	upper q.	1.86	1.46	1.12	5.58	1.88	1.40	2.53

Table 24 . `MA47` to `MA27` ratios using default threshold value for `MA27`.

In summary, these results indicate that it is “horses for courses”. MA47 does well on our augmented systems when the (1,2) block is nearly square or when several of the diagonals of the (1,1) block are zero. However, the efficiency of MA47 is very dependent on the details of the assembly tree structure so it is often difficult to judge the relative performance in advance. We find this fragility to be the most disturbing aspect of the present code and hope that further work will improve the performance of MA47 on the “bad” cases.

4 Acknowledgment

We would like to thank our colleagues Nick Gould and Jennifer Scott and particularly the referee John Lewis for reading drafts of the manuscript and for making many helpful suggestions for the presentation.

5 References

Anon (1993). Harwell Subroutine Library Catalogue (Release 11). Theoretical Studies Department, AEA Technology, Harwell.

Bongartz, I., Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1993). CUTE: Constrained and Unconstrained Testing Environment. Technical Report TR/PA/93/10, CERFACS, Toulouse, France. ACM Trans Math Softw (to appear).

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

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). A set of level 3 basic linear algebra subroutines. *ACM Trans. Math. Softw.* **16**, 1-17.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**, 1-17 and 18-32.

Duff, I. S. and Reid, J. K. (1982). MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Report AERE R10533, HMSO, London.

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

Duff, I. S. and Reid, J. K. (1995). MA47, a Fortran code for direct solution of indefinite symmetric linear systems of equations. Report RAL 95-001, Rutherford Appleton Laboratory, Oxfordshire.

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). Direct methods for sparse matrices. Oxford University Press, London.

Duff, I. S., Gould, N. I. M., Reid, J. K., Scott, J. A. and Turner, K. (1991). The factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.* **11**, 181-204.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1992). Users’ Guide for the Harwell-Boeing Sparse Matrix Collection. Report RAL 92-086, Rutherford Appleton Laboratory, Oxfordshire.

Gay, D. M. (1985). Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter.*

Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for Fortran use. *ACM Trans. Math. Softw.* **5**, 308-325.

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

Markowitz, H. M. (1957). The elimination form of the inverse and its application to linear programming. *Management Sci.* **3**, 255-269.