

Experiments in preprocessing and scaling symmetric problems for multifrontal solution

Iain. S. Duff* and Stéphane Pralet†

Working Notes WN/PA/04/17

February 18, 2004

Abstract

We consider ways of implementing a reordering and scaling based on MC64 for symmetric systems and show the effect of using this technique with two multifrontal codes for sparse symmetric indefinite systems.

*i.s.duff@rl.ac.uk, CERFACS, Toulouse, and Atlas Centre, RAL, Oxon OX11 0QX, England.

†Stephane.Pralet@cerfacs.fr. CERFACS, 42, av. G. Coriolis, 31057 Toulouse Cedex 01, France.

Contents

1	Introduction	3
2	The use of MC64 on symmetric matrices	3
2.1	Symmetric scaling	3
2.2	Selection of 2 by 2 pivots	4
2.3	Compression and decompression according to a set of candidate pivots	6
3	Strategies in MA47	7
4	Strategies in MA57	9
5	Structurally singular matrices	10
5.1	How to get a maximum weighted matching or an approximation ? . .	10
5.2	How to get a symmetric scaling from MC64 ?	11
6	Experimental results	11
6.1	Experimental environment	11
6.2	Memory and factorization time	13
6.2.1	Number of failures	13
6.2.2	Factorization time	14
6.2.3	MA47/MA57 comparison	16
6.3	Precision of the solution and iterative refinement	20
6.4	Analysis estimations and factorization memory	22
7	Conclusions and future work	25

1 Introduction

We study techniques for scaling and choosing pivots when using multifrontal methods in the LDL^T factorization of symmetric indefinite matrices where L is a lower triangular matrix and D is a block diagonal matrix with 1×1 and 2×2 blocks.

For the LU factorization of a matrix A , MC64 [6] can be used to get a maximum weighted matching so that the corresponding permutation will place large entries on the diagonal. The matrix can then be scaled so that diagonal entries have modulus one and off-diagonals have modulus less than or equal to one. This has been found to greatly improve the numerical stability of the subsequent LU factorization.

If, however, MC64 is applied to a symmetric matrix the resulting permutation will not normally preserve symmetry. In this paper, we examine ways in which MC64 can be used while still preserving symmetry. Then an ordering (for example, AMD [1]) can be computed on the permuted matrix to get a symmetric permutation in order to decrease the fill-in in the factors.

We will use our “symmetric” MC64 preprocessing with two symmetric multifrontal codes MA47 [8] and MA57 [5] to validate our preprocessing heuristics on real test problems.

We will describe :

- MC64 modifications to get a symmetric scaling and a set of 1×1 and 2×2 numerically acceptable pivots,
- Modifications of the ordering and factorization phases of MA47 and MA57 to exploit this preprocessing.

We will test our different strategies on three sets of matrices: structurally singular augmented systems, structurally nonsingular augmented systems, and general indefinite matrices.

2 The use of MC64 on symmetric matrices

2.1 Symmetric scaling

Definition 1 A matrix $B = (b_{ij})$ is said to satisfy the MC64 constraints if and only if

$$\exists \text{ a permutation } \sigma, \text{ such that } \forall i, |b_{i\sigma(i)}| = \|b_{\cdot\sigma(i)}\|_\infty = \|b_i\|_\infty = 1.$$

Property 1 Let \mathcal{M} be the maximum matching of the symmetric matrix A returned by MC64 and $D_r = (d_{r_i})$, $D_c = (d_{c_i})$ be the row and column scalings respectively. Let $D = (d_i) = \sqrt{D_r D_c}$. DAD is a symmetrically scaled matrix which satisfies the MC64 constraints.

Proof The entry of $|DAD|$ in position (i, j) is $|d_i d_j a_{ij}| = \sqrt{|d_{r_i} d_{c_j} a_{ij}|} \sqrt{|d_{r_j} d_{c_i} a_{ji}|} \leq 1$ because it is the square root of the product of 2 entries of $|D_r A D_c|$. Let σ be the column permutation associated with \mathcal{M} . Thus, for all i , $|d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}| = 1$. Let $\lambda_i = |d_{c_i} d_{r_{\sigma(i)}} a_{i\sigma(i)}| = |d_{c_i} d_{r_{\sigma(i)}} a_{\sigma(i)i}|$. $\forall i$, $\lambda_i \leq 1$ because it corresponds to an entry of $|D_r A D_c|$.

$$\prod_{1 \leq i \leq n} \lambda_i = \left(\prod_{1 \leq i \leq n} d_{r_i} \right) \left(\prod_{1 \leq i \leq n} d_{c_i} \right) \left(\prod_{1 \leq i \leq n} a_{i\sigma(i)} \right) = \prod_{1 \leq i \leq n} (d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}) = 1$$

(we exchange the terms in the products to get the product of the scaled terms in the MC64 maximum matching). The product of the numbers, λ_i , each of which is less than or equal to one, is one, so that

$$\lambda_i = 1, \forall i$$

and so

$$\forall i, |d_i d_{\sigma(i)} a_{i\sigma(i)}|^2 = |d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}| |d_{r_{\sigma(i)}} d_{c_i} a_{i\sigma(i)}| = 1 \times \lambda_i = 1.$$

2.2 Selection of 2 by 2 pivots

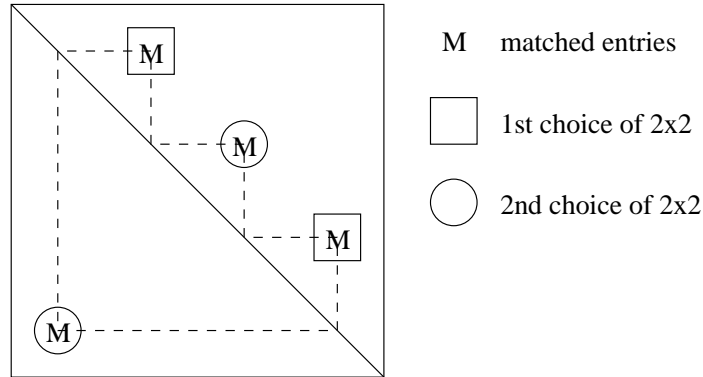


Figure 2.1: Example of selection of 2×2 pivots in an even loop.

We first use MC64 to compute a maximum weighted matching \mathcal{M} . This will normally have many entries that are not on the diagonal of A . Any diagonal entries that are in the matching are immediately considered as potential 1×1 pivots and are held in the set $\mathcal{M}_{1 \times 1}$. We then build a set $\mathcal{M}_{2 \times 2}$ of potential 2×2 pivots. We use the 2×2 pivot selection strategy suggested by [4], but with a structural metric instead of a numerical criterion. The basis for this strategy is to express the computed permutation, σ , in terms of its component cycles. Because of the scaling, all the entries in the cycles of σ are 1 so we choose a structural criterion to select the potential 2×2 pivots. Cycles of length 1 correspond to a matching

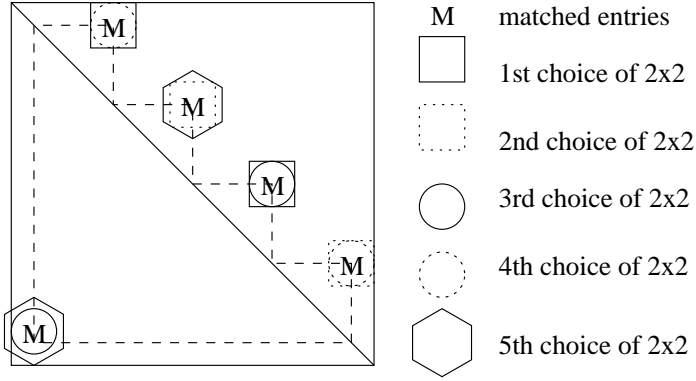


Figure 2.2: Example of selection of 2×2 pivots in an odd loop.

on the diagonal. k 2×2 pivots can be extracted from even cycles of length $2k$ or from odd cycles of length $2k + 1$. For even cycles there are only 2 possibilities of extraction (see example in Figure 2.1) and for odd cycles of length $2k + 1$ there are $2k + 1$ possible combinations of 2×2 pivots (see example in Figure 2.2). For each path in σ , we take 2×2 pivots such that the sequence of 2×2 pivots (i_k, j_k) maximizes $\prod |Row_{i_k} \cap Row_{j_k}| / |Row_{i_k} \cup Row_{j_k}|$. This extraction can be done in a double pass over the matching and thus in $\mathcal{O}(n) \times \mathcal{O}(c) = \mathcal{O}(nc)$, where c is the cost of the computation of the structural score of a 2×2 pivot. For the above metric, $|Row_{i_k} \cap Row_{j_k}|$ and $|Row_{i_k} \cup Row_{j_k}|$ can be computed in a time bounded by $\mathcal{O}(\max_i |Row_i|)$ and thus the total complexity of the extraction is $\mathcal{O}(n \max_i |Row_i|)$. Note that this total cost can be reduced to $\mathcal{O}(nnz(A))$, if we use the flags set during the computation of $|Row_{i_p} \cap Row_{i_{p+1}}|$ for the computation of $|Row_{i_{p+1}} \cap Row_{i_{p+2}}|$, when i_p, i_{p+1}, i_{p+2} are three consecutive indices in a cycle.

We define \mathcal{M}_s as $\mathcal{M}_{1 \times 1} \cup \mathcal{M}_{2 \times 2}$ corresponding to a symmetric matching obtained from \mathcal{M} . We note that $\mathcal{M}_{1 \times 1} \cap \mathcal{M}_{2 \times 2} = \emptyset$. Let \mathcal{M}_s^c be the set $\{(i, i) \text{ such that } i \text{ does not belong to any pivots in } \mathcal{M}_s\}$ that is, the complementary set to \mathcal{M}_s . It may be non empty because of odd cycles. We define \mathcal{P}_s , the candidate pivot set, by $\mathcal{M}_s^c \cup \mathcal{M}_s$. We illustrate these sets by the small 3×3 example in Figure 2.3, where the matching is shown as m and the candidate 2×2 pivot is defined by the off-diagonal entry $(1,2)$. Thus, the set $\mathcal{M}_{2 \times 2}$ is $(1,2)/(2,1)$, the set $\mathcal{M}_{1 \times 1}$ is null, and the set \mathcal{M}_s^c is $(3,3)$.

$$\begin{pmatrix} 0 & m & \times \\ \times & 0 & m \\ m & \times & 0 \end{pmatrix}$$

Figure 2.3: Matching and potential pivots on 3×3 example.

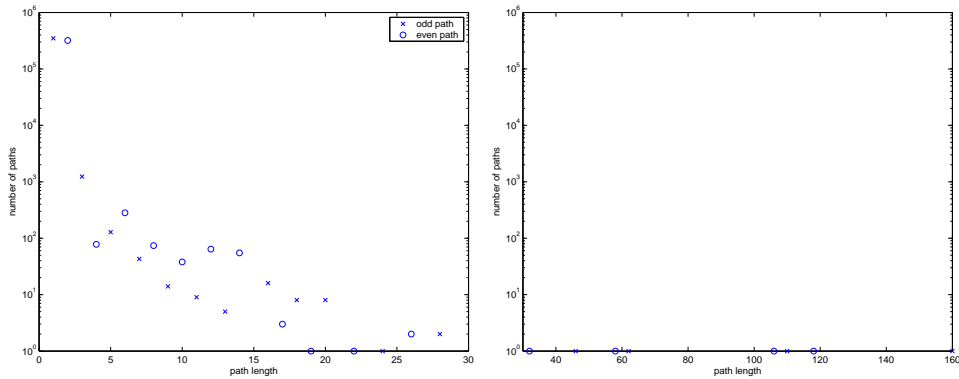


Figure 2.4: Length of MC64 paths.

In practice, most of the cycles from the MC64 permutation are of length 1 or 2. We illustrate this in Figure 2.4 where we show the total number of paths of varying lengths for runs over all of our test matrices. That is why we will not try different structural criteria to select the 2×2 pivots. We note that even cycles of length greater than 2 cannot occur if each off-diagonal entry has a different modulus so that there is no tie-breaking in MC64. Moreover, the entries of \mathcal{M}_s^c may not be directly selectable as a pivot because they correspond to a zero entry (for example, entry (3,3) of Figure 2.3). More generally, at each step k of the symbolic factorization the set \mathcal{P}_s of candidate pivots is split into two parts \mathcal{A}_s^k and \mathcal{U}_s^k . \mathcal{A}_s^k is the set of **selectable** (or **acceptable**) pivots and \mathcal{U}_s^k is the set of **unselectable** (or **unacceptable**) pivots. For example, at the first step of the Gaussian elimination we can set $\mathcal{A}_s^1 = \mathcal{M}_s$ and $\mathcal{U}_s^1 = \mathcal{M}_s^c$. Then, on the example in Figure 2.3, (3,3) can move from \mathcal{U}_s^1 to \mathcal{A}_s^2 because its diagonal is filled.

In some of our strategies, we will select pivots only from the candidate set \mathcal{P}_s . This selection can be relaxed by enlarging the set of acceptable pivots. If $B = \{b_{ij}\} = DAD$, is the scaled matrix, then we say that an entry is numerically acceptable as a pivot if it is larger in modulus than a threshold value, where our default value for the threshold is 0.99. We define \mathcal{S} as the set $\{(i, j) \text{ such that } |b_{ij}| > \text{threshold}\}$.

2.3 Compression and decompression according to a set of candidate pivots

Let \mathcal{M} be a maximum matching on A from which we have obtained a set of 1×1 and 2×2 candidate pivots, \mathcal{M}_s . The undirected graph \mathcal{G} associated with the matrix A has a set of vertices V_A , corresponding to the rows (and columns) of A and a set of unordered edges E_A , where the unordered pair $(i, j) \in E_A$ if and only if $a_{ij} \neq 0$.

We define R , the **reduced matrix of A relative to \mathcal{M}_s** as the square matrix of order the number of pivots in \mathcal{M}_s with vertices \mathcal{I}_i . Each \mathcal{I}_i is either associated with

a row/column of A corresponding to a candidate 1×1 pivot or a pair of rows/columns (i_k, i_l) corresponding to a candidate 2×2 pivot. The edge set E_R consists of the unordered pairs $(\mathcal{I}_i, \mathcal{I}_j)$ where there exists an edge between one of the constituent vertices of \mathcal{I}_i and \mathcal{I}_j that is in E_A . In other words, the candidate 2×2 pivots of \mathcal{M}_s are compressed into one vertex and the union of their adjacency defines the adjacency of the new vertex. Note that the rows/columns of A which are not represented in \mathcal{M}_s will not be represented in the reduced matrix.

Let P_{red} be a symmetric permutation on R . Then it is easy to extend P_{red} to a permutation P_{aug} on A by expanding each component corresponding to a 2×2 composite node to the two rows/columns of A associated with that composite node and by putting the rows/columns of A which were not represented in R (\mathcal{M}_s^c entries) at the end of this permutation. Clearly this expansion can be done in a single pass through P_{red} . Note that P_{aug} is not unique: firstly, when a node corresponding to a 2×2 pivot (i, j) is expanded, we have the choice of taking i or j first in P_{aug} ; secondly, when we visit the entries in \mathcal{M}_s^c , there is no a priori order for placing them in P_{aug} .

To illustrate the compression and the expansion, let take the following matrices:

$$A = \begin{pmatrix} 2 & -1 & 1 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \text{ and } R = \begin{pmatrix} X & X & X \\ X & X & 0 \\ X & 0 & X \end{pmatrix}.$$

In A , we detect a 2×2 pivot in rows/columns 3,4 and two 1×1 pivots on the diagonal. We have $\mathcal{M}_s = \{(1, 1), (2, 2), (3, 4)/(4, 3)\}$ and $\mathcal{M}_s^c = \{(5, 5)\}$. R is a reduced matrix of A relative to \mathcal{M}_s where row/column 3 corresponds to the 2×2 pivot. If $P_{red} = [3, 2, 1]$ then $P_{aug} = [3, 4, 2, 1, 5]$ or $[4, 3, 2, 1, 5]$ is an expansion of P_{red} to A .

We now consider the effect of our pivoting strategy when used with two multifrontal codes: MA47 that attempts to exploit the zero block of an augmented system and MA57 that ignores this block although uses 2×2 pivoting to effect a stable factorization of indefinite systems.

3 Strategies in MA47

We ran the MA47 factorization with a pivoting threshold equal to 10^{-2} .

- **Strategy MA47_0:** This corresponds to the standard MA47. We do not apply any scaling and we use the standard analysis of MA47.
- **Strategy MA47_1:** MA47_0 with the matrix first scaled using MC30 [2].
- **Strategy MA47_2:** MA47_0 with the matrix first scaled using MC77 (infinite norm) [10].

- **Strategy MA47_3:** MA47_0 with the matrix first scaled using MC77 (one norm).
- **Strategy MA47_4:** MA47_0 with the matrix first scaled using the symmetric MC64 scaling.
- **Strategy MA47_5:** We first apply the symmetric MC64 scaling and get \mathcal{P}_s . We use a modified version of the MA47 analysis. A pivot can be selected during the analysis if and only if it is a 2×2 candidate pivot in \mathcal{P}_s or is a 1×1 candidate pivot in \mathcal{P}_s that corresponds to a nonzero diagonal entry at the current step of the elimination. At each step of the analysis we take the selectable pivot of minimum metric. The metric used is a variant of the Markowitz cost [7].
- **Strategy MA47_6:** As for strategy MA47_5 but the candidate pivot set is enlarged to \mathcal{S} . For a successful completion of this analysis phase we need to keep a maximum matching on the remaining set of numerically acceptable pivots. That is why we use Algorithm 1 to update \mathcal{S} and \mathcal{M}_s (otherwise \mathcal{S} can become empty on structurally nonsingular matrices).

We have the following cases during the update of the set of numerically acceptable pivots (\times corresponds to the current pivots, m corresponds to the old matching entries, f corresponds to new matching entries):

$$\begin{array}{l}
 \text{(case 1)} \quad \begin{array}{c} p \quad i \\ p \quad \left(\begin{array}{cc} \times & m \\ m & f \end{array} \right) \\ i \end{array} \quad , \quad \text{(case 2)} \quad \begin{array}{c} i \quad j \quad j_1 \\ i \quad \left(\begin{array}{ccc} m & \times & \\ \times & & m \\ & m & f \end{array} \right) \\ j \\ j_1 \end{array} \quad , \\
 \\
 \text{(case 3)} \quad \begin{array}{c} j \quad i \quad i_1 \\ j \quad \left(\begin{array}{ccc} m & \times & \\ \times & & m \\ & m & f \end{array} \right) \\ i \\ i_1 \end{array} \quad , \quad \text{(case 4)} \quad \begin{array}{c} i \quad j \quad i_1 \quad j_1 \\ i \quad \left(\begin{array}{cccc} & \times & m & \\ \times & & & m \\ m & & & f \\ & m & f & \end{array} \right) \\ j \\ i_1 \\ j_1 \end{array} \quad .
 \end{array}$$

Algorithm 1 Update of the set of numerically acceptable pivots.

```
if the selected pivot is in  $\mathcal{M}_s$  then
  Remove the current pivot from  $\mathcal{M}_s$  and  $\mathcal{S}$ .
else
  if the selected pivot is a  $1 \times 1$  pivot  $p$  then
    Let  $i$  be such that  $(i, p) \in \mathcal{M}_s$ .
    Add  $(i, i)$  to  $\mathcal{M}_s$  and  $\mathcal{S}$ , remove  $(i, p)$  from  $\mathcal{M}_s$  and  $\mathcal{S}$  (case 1).
  else
    Let  $(i, j)$ ,  $i \neq j$  be the  $2 \times 2$  pivot,  $i_1$  be such that  $(i, i_1) \in \mathcal{M}_s$  and  $j_1$  be such
    that  $(j, j_1) \in \mathcal{M}_s$ .
    if  $i_1 = i$  and  $j_1 \neq j$  then
      Add  $(j_1, j_1)$  to  $\mathcal{M}_s$  and  $\mathcal{S}$  (case 2).
    else if  $j_1 = j$  and  $i_1 \neq i$  then
      Add  $(i_1, i_1)$  to  $\mathcal{M}_s$  and  $\mathcal{S}$  (case 3).
    else if  $i \neq i_1$  and  $j \neq j_1$  then
      Add  $(i_1, j_1)$  to  $\mathcal{M}_s$  and  $\mathcal{S}$  (case 4).
    end if
    Remove  $(i, i_1)$  and  $(j, j_1)$  from  $\mathcal{M}_s$  and  $\mathcal{S}$ .
  end if
end if
```

4 Strategies in MA57

When we used MA47 we did not modify the factorization step. It will be different in MA57 with the strategy MA57_6. For the other strategies, we use the standard MA57 factorization step with a pivoting threshold of 10^{-2} .

- **Strategy MA57_0:** This corresponds to the standard MA57. We do not apply any scaling and we use the standard analysis of MA57.
- **Strategy MA57_1:** MA57_0 plus MC30 scaling.
- **Strategy MA57_2:** MA57_0 plus MC77 scaling in infinite norm.
- **Strategy MA57_3:** MA57_0 plus MC77 scaling in one norm.
- **Strategy MA57_4:** MA57_0 plus symmetric MC64 scaling.
- **Strategy MA57_5:** We first apply a symmetric MC64 scaling and get \mathcal{M}_s . We then compute R , a reduced matrix of A relative to \mathcal{M}_s . We apply an AMD based ordering on R to get a symmetric permutation P_{red} on R . We compute P_{aug} , an extension of P_{red} relative to \mathcal{M}_s and place the components of \mathcal{M}_s^c in the last positions (this ensures that potentially zero pivots in \mathcal{M}_s^c will be

filled by the previous eliminations if we assume that no numerical cancellation occurs and the matrix is structurally nonsingular). Then we run the MA57 factorization on the scaled matrix with this ordering.

- **Strategy MA57_6:** This starts with the same as strategy MA57_5 but we modify the MA57 factorization so that we do no further numerical pivoting in the lower part of the tree. If the chosen pivot is too small, we add to it a perturbation. To be more precise, we do the same as in SuperLU_DIST [9] for a 1×1 pivot and replace it by the value $\sqrt{\epsilon} \|A\|_1$, where ϵ is the machine precision. For a small 2×2 pivot P , we use Δ , a perturbation such that

$$\|(P + \Delta)^{-1}\|_\infty \leq \frac{1}{\sqrt{\epsilon} \|A\|_1}.$$

The point in the tree at which we stop this static pivoting is governed by a parameter and our logic is that MC64 will normally have done a good job for the early pivots so that there is less likelihood of static pivoting causing many changes to the early pivots.

5 Structurally singular matrices

In this section, we consider the case when A is a structurally singular matrix. Structurally singular matrices are quite common in optimization. The notion of a maximum weighted matching with the product metric is not well defined for MC64 [6]. In this case, for MC64 all the maximum matchings will have zero weight. Moreover, it is impossible to find scaling factors that satisfy MC64 constraints for structurally singular matrices.

5.1 How to get a maximum weighted matching or an approximation ?

Suppose that the weight of a matching on a structurally singular matrix is given by the product of the absolute values of the matched entries. Then we can define a maximum weighted matching on a structurally singular matrix as a matching of maximum weight among the matchings of maximum size. We did not modify the MC64 code to get this kind of matching, but we used a less complicated algorithm to get an approximation to a maximum weighted matching. We first apply MC64 to A . Then we extract \mathcal{M}_s from the matching obtained, and we define C as the submatrix containing the row and column indices in \mathcal{M}_s . C is structurally nonsingular. We again apply MC64 this time on C and recompute \mathcal{M}_s according to the maximum matching of C .

5.2 How to get a symmetric scaling from MC64 ?

We relax the MC64 constraints on the scaling. Let D_r and D_c be the scaling factors returned by MC64 on C . We build D so that if i corresponds to an index of R ,

$$d_i = \sqrt{d_{r_i} d_{c_i}},$$

otherwise

$$d_i = \frac{1}{\max\{\max_{k \in \text{index}(R)} |a_{ik} d_k|, \max_{k \notin \text{index}(R)} \sqrt{|a_{ik}|}\}}.$$

We note that the entries of $|DAD|$ are less than 1 and entries in \mathcal{M}_s are 1.

6 Experimental results

6.1 Experimental environment

Table 6.1: Structurally singular augmented systems (set 1).

Matrix	Order	nnz	Zero diag		“Constraints min size”	
AUG2DC	30200	40000	30200	100%	10000	33%
AUG2D	29008	38416	20003	67%	9604	33%
DTOC3	7493	14982	7493	100%	2997	40%
bloweybl	30003	70001	20002	67%	10001	33%
dtoc	24993	49982	24993	100%	9997	40%
funny1	1030	7141	42	4%	0	0%
laser	3002	6002	2002	67%	1000	33%
sloan	2295	11100	2282	99%	305	13%

We conduct our experiments on a number of test problems which can be divided into three sets. The two sets in Tables 6.1 and 6.2 come from optimization problems and correspond to augmented systems of the form

$$\mathcal{K}_{H,A} = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix}.$$

These two sets correspond to structurally singular and structurally nonsingular matrices respectively. The third set, in Table 6.3, corresponds to general (nonzero diagonal) indefinite symmetric matrices.

For the sets 1 and 2 we give an estimate of the number of constraints. Firstly, we note that this estimate is less than or equal to the number of zeros on the diagonal.

Table 6.2: Structurally nonsingular augmented systems (set 2).

Matrix	Order	nnz	Zero diag		“Constraints min size”	
AUG2DCQP	4880	12960	1600	33%	1600	33%
BLOWEYA	30004	90006	20003	67%	10002	33%
BOYD1	93279	745507	18	0%	18	0%
CONT-201	80595	249996	40198	50%	40198	50%
NCVXQP1	12111	47648	5000	41%	5000	41%
NCVXQP5	62500	287481	12500	20%	12500	20%
NCVXQP7	87500	362481	37500	43%	37500	43%
bratu3d	8288	28538	3375	41%	3375	41%
cvxqp3	17500	69981	7500	43%	7500	43%
mario	38434	114643	15304	40%	15304	40%
ngee	4419	32938	1408	32%	1	0%
qpband	20000	49999	5000	25%	5000	25%
sawpath1	1359	4066	580	43%	2	0%
stcqp1	5036	38045	939	19%	939	19%
waltz1	3500	6498	500	14%	0	0%
waltz2	2415	4715	609	25%	0	0%
waltz3	2415	4715	609	25%	0	0%
waltz4	2415	4715	609	25%	0	0%

Table 6.3: General symmetric matrices (set 3).

Matrix	Order	nnz
HELM3D01	32226	230335
SPARSINE	50000	799494
bloweybq	10001	39996
c-68	64810	315408
c-71	76638	468096
copter2	55476	407714
dawson5	51537	531157
vibrobox	12328	177578

Secondly, we estimate (in column “Constraint min size” of Table 6.1 and 6.2) this by $n - \max\{j \text{ such that there is an entry in the lower part of column } j \text{ of } \mathcal{K}_{H,A}\}$. This estimate is exact if no reordering has been done on $\mathcal{K}_{H,A}$.

In the following tables, we will only show the results on matrices which are representative of a global behaviour or which illustrate an important aspect. Concerning the SPARSINE and NCVXQP7 matrices, we will not give any results about them in the tables because they are too large for our target machine.

6.2 Memory and factorization time

Our experiments are conducted on one node of a PC cluster at CERFACS. There is 1 Gbyte of memory shared between 2 PIII-933 processors per node and we disable one of the processors so that we can use all the memory of the node with the other single processor. We use the Portland Fortran compiler, pgf90 with -O option. If the factorization of a matrix takes more than 800 MBytes or requires more than 10 minutes, we consider that the factorization is not successful.

We compare our codes using the performance profiling system of [3]. In these profiles, if α is the value on the x-axis, then the y-axis measures the fraction of times that each code is within a factor α of the best code.

6.2.1 Number of failures

Table 6.4: MA47 error output, 0 : success, CPU : CPU time exceeded, MAA not enough memory after analysis. Only matrices on which at least one version of MA47 did not succeed are reported.

Matrix	MA47_0	MA47_1	MA47_2	MA47_3	MA47_4	MA47_5	MA47_6
BLOWEYA	CPU	0	0	0	0	0	0
BOYD1	CPU	CPU	CPU	CPU	CPU	0	0
CONT-201	CPU	CPU	CPU	CPU	CPU	0	CPU
NCVXQP1	CPU	CPU	CPU	CPU	0	0	0
NCVXQP5	MAA	MAA	MAA	MAA	MAA	MAA	MAA
cvxqp3	CPU	CPU	CPU	CPU	CPU	0	0
HELM3D01	0	0	0	0	0	MAA	MAA
c-68	0	0	0	0	0	CPU	CPU
c-71	CPU	CPU	CPU	CPU	CPU	MAA	MAA
copter2	0	0	0	0	0	MAA	MAA
dawson5	0	0	0	0	0	MAA	CPU
vibrobox	0	0	0	0	0	CPU	CPU

In Table 6.4, we see that MA47_0 is the least robust approach, it exceeds the CPU time limit for 6 matrices (CPU error in the table). MA47_5 and MA47_6 are clearly the more robust approaches on augmented systems but are not reliable on general

Table 6.5: MA57 output error, ≥ 0 : success, MAA : not enough memory after analysis, MAF : not enough memory after factorization, CPU : CPU time exceeded. Only matrices on which at least one version of MA47 did not succeed are reported.

Matrix	MA57_0	MA57_1	MA57_2	MA57_3	MA57_4	MA57_5	MA57_6
AUG2DC	MAF	MAF	MAF	MAF	MAF	MAA	MAA
AUG2D	MAF	MAF	MAF	MAF	MAF	MAA	MAA
bloweybl	CPU	4	4	4	4	4	4
dtoc	CPU	CPU	CPU	CPU	CPU	MAA	MAA
BLOWEYA	CPU	0	0	0	0	0	0
BOYD1	CPU	CPU	CPU	CPU	CPU	0	0
NCVXQP1	CPU	0	0	0	0	0	0
NCVXQP5	MAF	CPU	CPU	CPU	CPU	MAA	MAA
cvxqp3	MAF	0	0	0	0	0	0
c-68	0	0	0	0	0	CPU	CPU
c-71	CPU	CPU	CPU	CPU	CPU	CPU	CPU

indefinite matrices. On these matrices, they can be far from the best ordering because of the constraints that we put in the analysis phase and because of the errors in the computation of the Markowitz cost of the full 2×2 pivots. Indeed, during the analysis, the metric of a full 2×2 pivot (i, j) is approximated by $(deg(i) + deg(j) - 2)^2/2$, whereas its real cost is $|(Row_i \cup Row_j) \setminus \{i, j\}|^2/2$.

In Table 6.5, we see that using the strategy MA57_5 or MA57_6 improves the robustness of the solver on augmented systems. Indeed, in these cases we never have a memory problem during the factorization. A drawback of these two approaches is that they do more computation during the factorization because of the structural quality of the analysis. That is why we sometimes (c-68 matrix) hit the CPU time limit, when other approaches succeed. On the c-68 matrix, MA57_4 forecast 7.2×10^6 entries in the factors and 1.2×10^{10} operations. After factorization there are 8.4×10^6 entries in the factors and 1.5×10^{10} operations are performed. On this matrix, the MA57_5 and MA57_6 analysis forecast 15.8×10^6 entries in the factors and 3.7×10^{10} operations, which requires more than 10 minutes on our target machine. The use of scaling improves MA57. MA57_0 fails on 10 matrices, whereas MA57 plus any scaling fails on less than 6 matrices.

6.2.2 Factorization time

On structurally singular augmented systems, the analysis phase of MA47 will identify a structurally nonsingular block and will permute a block of zeros to the end. We call this the structural kernel. On these systems, MA47_5 has the least factorization

Table 6.6: MA47 factorization CPU time in seconds.

Matrix	MA47_0	MA47_1	MA47_2	MA47_3	MA47_4	MA47_5	MA47_6
AUG2DC	0.03	0.04	0.03	0.04	0.03	0.03	0.03
DTOC3	0.00	0.00	0.01	0.01	0.01	0.00	0.01
sloan	11.9	6.94	7.00	5.78	10.7	4.52	11.3
AUG2DCQP	0.01	0.02	0.02	0.01	0.02	0.86	0.01
BLOWEYA		0.08	0.08	0.07	0.08	0.08	0.07
BOYD1						0.40	0.40
CONT-201						47.1	
NCVXQP1					251.	124.	108.
bratu3d	391.	392.	391.	384.	389.	72.6	75.4
cvxqp3						225.	211.
mario	0.36	0.38	0.36	0.36	0.36	2.18	0.48
ngee	1.73	1.70	0.06	0.04	0.04	0.83	0.02
sawpath1	6.09	1.42	1.82	0.18	0.00	0.00	0.01
waltz4	1.02	0.84	0.84	0.81	0.62	0.02	0.02
HELM3D01	173.	173.	173.	173.	172.		
c-68	288.	454.	269.	245.	257.		
vibrobox	45.5	172.	34.9	36.4	35.1		

time (see Table 6.6 and Figure 6.1). There are less delayed pivots with this approach. Moreover, any numerical problem will destroy the structural kernel prediction by the analysis. That is why the MA47_6 approach which involves less strict numerical constraints during the analysis is a bit too risky for this set of matrices. On structurally nonsingular augmented systems, MA47_6 seems to be the best of the MA47 strategies. On one hand, it uses information from MC64 to improve the numerical stability and, on the other hand, its flexible analysis offers quite a good structural quality for the ordering. On the indefinite non-augmented matrices, MA47_3 and MA47_4 are the best approaches. Here again we see the drawback of the analysis phase of MA47_5 and MA47_6. MC30 scaling does not improve the performance. Contrary to the other scalings (MC64, MC77), it does not take into account structural information when doing the scaling and uniformly tries to scale all entries to be as close to one as possible. We suspect that the structural decision of the analysis is less compatible with MC30 scaling. This difference can give us a partial answer about the degradation of the performance with MA47_1, whereas the other scalings have a positive effect on general indefinite matrices.

MA57 is not well designed to solve structurally singular problems, it solves only 60% of set 1 whereas MA47 solves 100%. On nonsingular augmented systems MA57_5 and MA57_6 seem to be the best MA57 approaches. MA57_6 has the advantage of giving

Table 6.7: MA57 factorization CPU time in seconds.

Matrix	MA57_0	MA57_1	MA57_2	MA57_3	MA57_4	MA57_5	MA57_6
DTOC3	79.5	62.1	62.0	61.9	79.5	36.1	35.4
laser	0.00	0.00	0.00	0.01	0.00	0.01	0.01
sloan	3.69	3.76	3.89	5.47	4.63	5.53	5.49
BOYD1						0.60	0.61
CONT-201	37.7	37.8	37.7	38.1	37.8	7.97	7.30
NCVXQP1		294.	77.1	110.	66.5	60.4	58.8
bratu3d	11.6	11.5	11.4	11.1	11.5	3.24	3.14
cvxqp3		340.	234.	241.	213.	295.	266.
mario	0.33	0.33	0.32	0.33	0.32	0.56	0.56
ngee	0.24	1.56	0.06	0.02	0.02	0.02	0.03
sawpath1	1.02	0.24	0.43	0.03	0.00	0.00	0.01
waltz4	0.02	0.01	0.01	0.01	0.01	0.01	0.00
HELM3D01	86.9	87.5	86.9	87.5	87.1	158.	158.
c-68	166.	532.	154.	188.	144.		
vibrobox	9.43	40.7	7.73	7.74	7.74	10.1	10.1

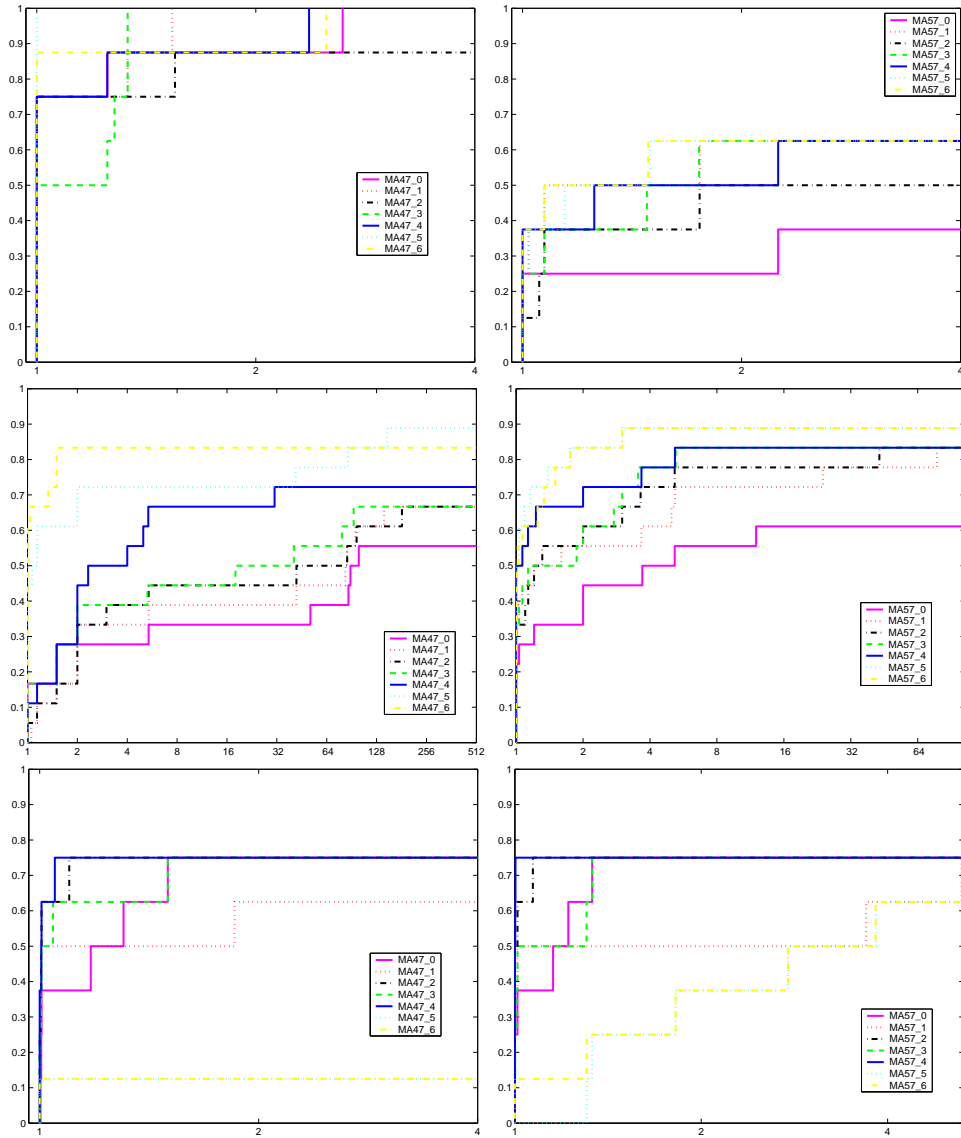
an exact prediction of the memory needed by the factorization at the bottom of the assembly tree. Nevertheless we have to check the numerical quality of the solution because static pivoting does not ensure that we have a reasonable error. That is why we keep the two codes, MA57_5 and MA57_6 for the next comparison. On the indefinite matrices, MA57_4 seems to be the best strategy. MA57_5 and MA57_6 have too many constraints in the analysis, which significantly increases fill-in and operations. As with MA47, the MC30 scaling again often degrades the performance.

6.2.3 MA47/MA57 comparison

Regarding the above comparison of factorization times, we decided to compare MA47_5 and MA57_6 for the set 1 (see Figure 6.2), MA47_6, MA57_5 and MA57_6 for the set 2 (see Figure 6.3), and MA47_4 and MA57_4 for the set 3 (see Figure 6.4). The right-hand side profiles of Figures 6.2, 6.3 and 6.4 compare the memory needed for a successful factorization, and the left-hand side profiles compare the CPU factorization time.

It seems clear that MA47 is faster and more robust than MA57 on the set of structurally singular augmented systems (Figure 6.2). Moreover it needs less memory than MA57 on this set of matrices. In particular it does not store the zero block of the structural kernel. For example, AUG2D has a structural kernel of size 10000 and MA47 avoids storing the more than 380 MBytes required for this part.

Figure 6.1: CPU profile MA47/MA57. row 1: set 1 (augmented singular systems). row 2: set 2 (augmented nonsingular systems). row 3: set 3 (general indefinite symmetric matrices).



For nonsingular augmented systems (Figure 6.3), MA47 and MA57 are comparable in half of the cases and it is not clear which code is the best. Sometimes MA47 takes advantage of the zero blocks and avoids a lot of the computation and storage done by MA57. (compare for example the number of operation and the factorization time of the two codes, MA47_6 and MA57_6, on the BLOWEYA and stcp1 matrices in Table 6.8). When numerical pivoting does not change analysis predictions, MA47 can better manage the zero blocks of tiles and oxo pivots than MA57. The drawback of

Figure 6.2: MA47/MA57 comparison on set 1. Left figure : CPU time profile. Right figure : profile of memory used by factorization.

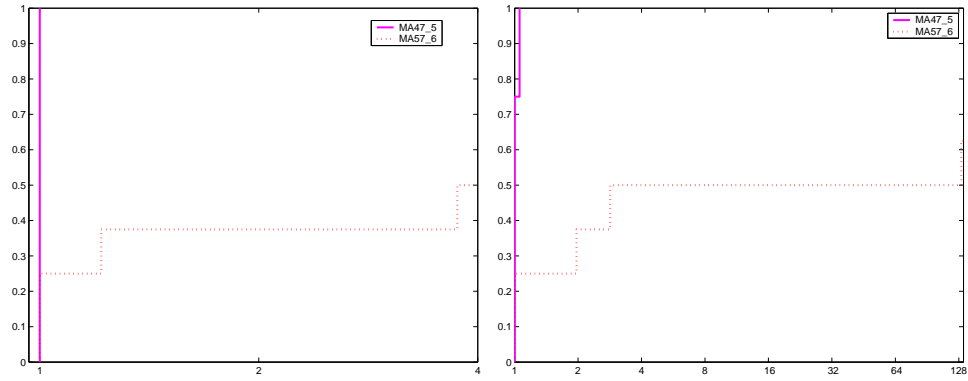


Figure 6.3: MA47/MA57 comparison on set 2. Left figure : CPU time profile. Right figure : profile of memory used by factorization.

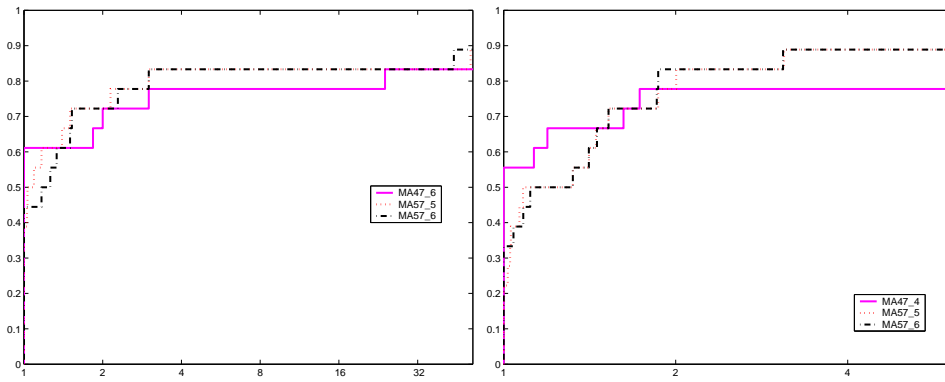
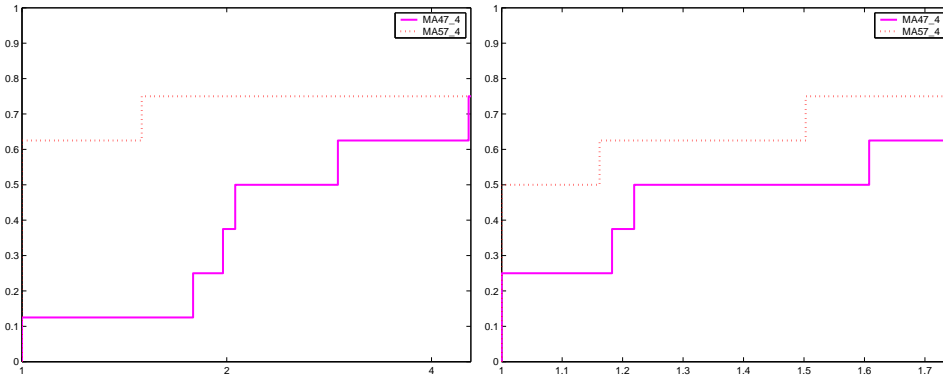


Table 6.8: Comparison of MA47_6 and MA57_6 number of operation and factorization CPU time in seconds on selected matrices from set 2.

Matrix	MA47_6		MA57_6	
	Time	Forecast / Real number of operations	Time	Real number of operations
BLOWEYA	0.07	4.8 / 4.8×10^5	0.16	8.8×10^6
cvxqp3	211.	0.5 / 3.4×10^9	266.	2.1×10^{10}
mario	0.48	3.9 / 4.2×10^7	0.56	7.5×10^7
stcqp1	0.01	7.7 / 5.6×10^4	0.44	4.5×10^7

Figure 6.4: MA47/MA57 comparison on set 3. Left figure : CPU time profile. Right figure : profile of memory used by factorization.



the MA47 approach is that the structures are more complicated with some indirect addressing. That is why, for an equal number of floating-point operations, MA57 will be faster than MA47. For example (see Table 6.8), on the mario matrix, MA47_6 and MA57_6 factorizations take nearly the same time but MA57_6 does twice as many operations as MA47_6. Furthermore, numerical pivoting will increase this effect. The more numerical pivoting occurs, the more the indirect addressing changes and the slower MA47 becomes. This effect is clearly illustrated by the MA47_6 behaviour on cvxqp3 (see Table 6.8), where the increase in the number of operations between analysis and factorization reveals numerical problems and delayed pivots.

On general indefinite matrices, MA57 is clearly better than MA47 in terms of CPU time and is slightly better in terms of memory (see Figure 6.4).

Table 6.9: MA47 precision of the solution : componentwise backward error $berr = \max_i \frac{|Ax-b|_i}{(|b|+|A||x|)_i}$ and number of iterative refinement steps in parenthesis.

Matrix	MA47_0	MA47_1	MA47_2	MA47_3	MA47_4	MA47_5	MA47_6
DTOC3	7E-14 (2)	2E-13 (3)	7E-14 (2)	6E-14 (2)	7E-14 (2)	1E-13 (3)	7E-14 (2)
bloweybl	8E-16 (2)	1E-16 (2)	6E-16 (2)	4E-16 (2)	1E-16 (2)	3E-16 (2)	3E-16 (2)
funny1	1E-00 (1)	2E-16 (2)	3E-16 (2)	2E-16 (2)	1E-16 (2)	1E-16 (2)	2E-16 (2)
laser	1E-16 (1)	1E-16 (1)	1E-16 (1)	2E-16 (1)	1E-16 (1)	1E-16 (1)	1E-16 (1)
sloan	1E-00 (1)	1E-00 (1)	1E-00 (1)	1E-00 (1)	2E-01 (2)	9E-15 (3)	4E-12 (3)
BLOWEYA		2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)
BOYD1						4E-15 (3)	1E-14 (3)
CONT-201						2E-16 (2)	
NCVXQP1					5E-01 (1)	2E-16 (2)	2E-16 (2)
bratu3d	1E-16 (2)	1E-16 (2)	2E-16 (2)	1E-16 (2)	2E-16 (2)	1E-16 (2)	1E-16 (2)
cvxqp3						3E-16 (2)	3E-16 (2)
mario	2E-16 (2)	2E-16 (2)	1E-16 (2)	2E-16 (2)	2E-16 (2)	1E-16 (2)	1E-16 (2)
sawpath1	7E-16 (1)	5E-16 (2)	6E-16 (2)	6E-16 (2)	9E-16 (2)	4E-16 (2)	7E-16 (2)
stcqp1	6E-16 (1)	5E-16 (1)	6E-16 (1)	7E-16 (1)	7E-16 (1)	8E-16 (1)	7E-16 (1)
waltz1	1E-16 (2)	1E-16 (2)	1E-16 (2)	1E-16 (2)	2E-16 (2)	1E-16 (2)	1E-16 (2)
waltz3	1E-00 (1)	1E-16 (2)	1E-00 (1)	2E-16 (2)	1E-16 (2)	1E-16 (2)	1E-16 (2)
HELM3D01	3E-16 (2)	3E-16 (2)	3E-16 (2)	4E-16 (2)	3E-16 (2)		
bloweybq	1E-16 (1)	1E-16 (1)	2E-16 (1)	1E-16 (1)	2E-16 (1)	7E-16 (1)	7E-16 (1)
copter2	2E-16 (2)	3E-16 (2)	2E-16 (2)	3E-16 (2)	3E-16 (2)		
dawson5	3E-16 (2)	3E-16 (2)	4E-16 (2)	3E-16 (2)	3E-16 (2)		
vibrobox	1E-14 (2)	4E-14 (3)	5E-16 (2)	5E-16 (2)	5E-16 (2)		

6.3 Precision of the solution and iterative refinement

Tables 6.9 and 6.10 show the componentwise backward error of the solution and the number of iterative refinement steps. We limited the number of iterative refinement steps to 10 and stopped when the backward error is comparable to the machine precision (in practice we stop if it is $< 10^{-15}$) or when the convergence is too slow ($berr_n > rat \times berr_{n-1}$ with $rat = 0.5$). For most cases we note that the backward error for the different versions of MA47 and MA57 is comparable.

In general, the backward errors for MA57 are good although some, for example MA57_3 and MA57_4 on laser, are a little worse. These backward errors can be slightly improved if we relax the ratio test ($rat = 0$). With this relaxed ratio test, on laser, the MA57_3 backward error decreases to 9×10^{-13} after 6 iterations and the MA57_4 backward error decreases to 7×10^{-12} after 3 iterations.

The MA57_6 strategy that uses a static pivoting approach in the lower part of the tree gives good backward error in all cases and uses a reasonable number of iterative

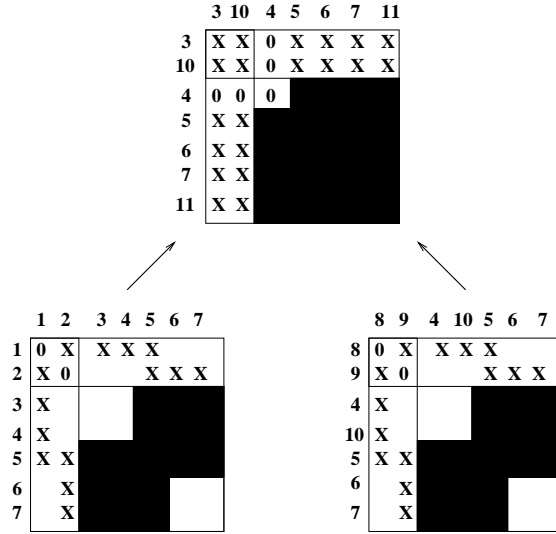
Table 6.10: MA57 precision of the solution : componentwise backward error $berr = \max_i \frac{|Ax-b|_i}{(|b|+|A||x|)_i}$ and number of iterative refinement steps in parenthesis.

Matrix	MA57_0	MA57_1	MA57_2	MA57_3	MA57_4	MA57_5	MA57_6
DTOC3	1E-13 (3)	6E-14 (4)	2E-13 (3)	1E-12 (2)	1E-13 (3)	7E-14 (2)	7E-14 (3)
bloweybl		3E-16 (2)	3E-16 (2)	1E-16 (3)	1E-16 (3)	8E-16 (2)	2E-16 (5)
funny1	1E-16 (2)	2E-16 (2)	4E-16 (2)	1E-16 (2)	1E-16 (2)	1E-16 (2)	5E-16 (3)
laser	9E-13 (2)	8E-14 (4)	8E-13 (2)	9E-12 (3)	6E-11 (2)	2E-16 (2)	2E-16 (3)
sloan	2E-15 (4)	1E-14 (3)	9E-14 (3)	1E-14 (3)	7E-14 (3)	4E-11 (3)	9E-13 (3)
BLOWEYA		2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (4)
BOYD1						9E-15 (3)	8E-15 (4)
CONT-201	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (3)
NCVXQP1		2E-16 (2)	2E-16 (2)	2E-16 (2)	4E-16 (2)	2E-16 (2)	4E-16 (3)
bratu3d	2E-16 (2)	1E-16 (2)	1E-16 (2)	2E-16 (2)	1E-16 (2)	1E-16 (2)	2E-16 (3)
cvxqp3		4E-16 (2)	3E-16 (2)	4E-16 (2)	4E-16 (2)	3E-16 (2)	4E-16 (3)
mario	1E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (2)	1E-16 (2)	2E-16 (2)	2E-16 (4)
sawpath1	5E-16 (2)	6E-16 (2)	5E-16 (2)	6E-16 (2)	4E-16 (2)	5E-16 (2)	5E-16 (3)
stcqp1	5E-16 (2)	6E-16 (2)	5E-16 (2)	6E-16 (2)	6E-16 (2)	6E-16 (2)	7E-16 (3)
waltz1	1E-16 (2)	1E-16 (2)	1E-16 (2)	1E-16 (2)	1E-16 (2)	2E-16 (2)	5E-15 (10)
waltz3	1E-16 (2)	1E-16 (2)	1E-16 (2)	2E-16 (2)	1E-16 (2)	1E-16 (2)	1E-16 (3)
HELM3D01	4E-16 (2)	3E-16 (2)	3E-16 (2)	3E-16 (2)	3E-16 (2)	3E-16 (2)	3E-16 (3)
bloweybq	1E-16 (1)	1E-16 (1)	1E-16 (1)	1E-16 (1)	1E-16 (1)	1E-16 (2)	1E-16 (3)
copter2	2E-16 (2)	2E-16 (2)	3E-16 (2)	2E-16 (2)	3E-16 (2)	2E-16 (2)	2E-16 (4)
dawson5	3E-16 (2)	2E-16 (2)	3E-16 (2)	3E-16 (2)	2E-16 (2)	2E-16 (2)	2E-16 (3)
vibrobox	5E-13 (2)	6E-16 (2)	1E-15 (2)	3E-15 (3)	1E-15 (2)	7E-16 (2)	4E-16 (3)

refinements compared to the other versions of MA57, except for waltz1 (it is the only case when we perform 10 iterations whereas the other versions converge in less than 2 iterations). Note that if we authorize 12 (respectively 13) iterations the backward error is 9×10^{-16} (respectively 3×10^{-16}) on this matrix.

For MA47, we sometimes get precision degradation. For example, on the funny matrix with strategy 0, on the sloan matrix with strategy 0 to 4, on NCVXQP1 with strategy 4, and on waltz3 with strategy 0 and 2. If we relax the ratio test on all these cases we do not improve the backward error and thus do not converge to an acceptable solution, except for NCVXQP1 with MA47_4 for which we get a solution with the following iterations: step 1, backward error of 5×10^{-1} , step 2, backward error of 3×10^{-14} and step 3, backward error of 2×10^{-16} .

Figure 6.5: An example of removable zeros in MA57 factors. **X** corresponds to a nonzero and shaded areas to nonzeros in the contribution blocks.



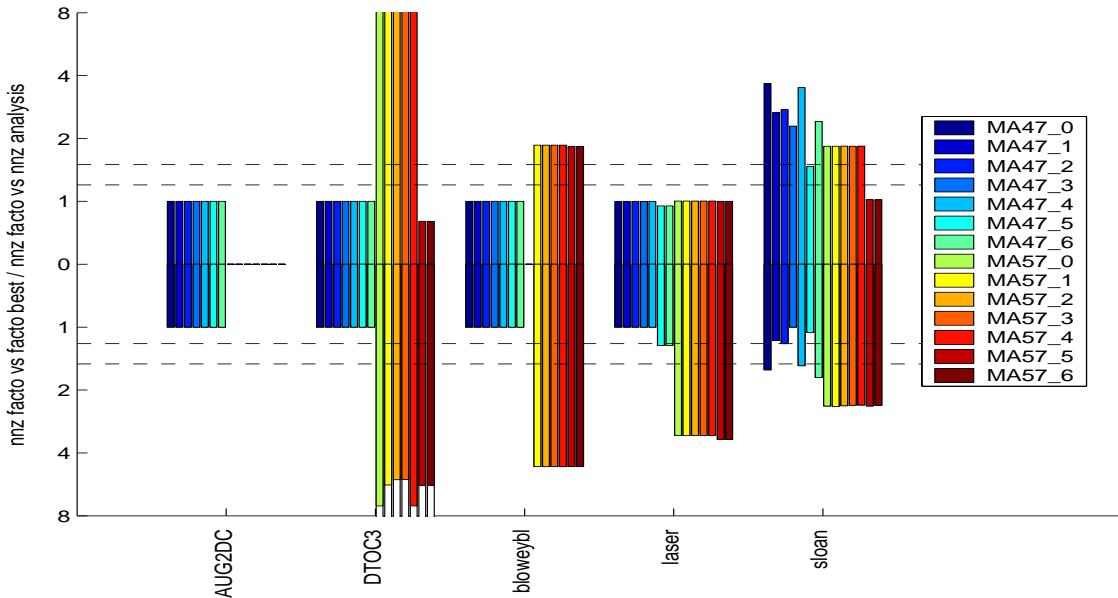
6.4 Analysis estimations and factorization memory

In the previous section, we studied the robustness of the factorization where we only allowed 800 MBytes of memory and a factorization time of less than 10 minutes. A critical problem is the evaluation during the analysis of the memory needed by the factorization. We would like to have an estimation of the required memory after the analysis that is not too far from the space actually needed by the factorization. It is standard practice to increase this estimate and allocate rather more storage in the hope that the factorization will be successful even if some additional numerical pivoting occurs. In this section, we will study the robustness of the different approaches in terms of memory estimation and the size of the factors.

In Figures 6.6, 6.7 and 6.8, we plot with positive values the ratio of the number of entries after factorization to the forecast number of entries after the analysis. This measure shows the reliability of the analysis estimation. The first dashed line represents 20% greater storage required by the factorization than predicted by the analysis, and the second represents 50% more. We plot with a negative value the ratio of the number of entries in the factors after factorization to the number for the best strategy. A value equal to 0 means that the factorization was not successful.

For MA57, we can drop some entries in the factors during the factorization in order to have a faster solve phase. The fully summed rows of a node can contain zero columns which can be removed from the factors after having computed the contribution block. For example, in Figure 6.5, the zero entries in column 4 and rows 3 and 10 can be removed. This is due to the fact that the zeros blocks of oxo or tiles of its children are explicitly stored in MA57. The white bars in Figures 6.6 and 6.7

Figure 6.6: Set 1 : ratio between analysis estimation and number of nonzeros after factorization (positive values), ratio between number of nonzeros after factorization and the best strategy.



represent the ratios if this dropping is not done. Sometimes the number of such entries can be very significant. For example, in an extreme case, this corresponds to 98% of the factors of DTOC3 computed by MA57_1. It is important to note that this dropping does not change the number of flops for the factorization. Because, in a multifrontal approach, information can only be passed from child to parent, we have to compute the contribution block before dropping any entries. We could remove them before the Schur computation but it would involve a lot of indirect addressing.

On the set of structurally singular augmented systems, MA47_5 is the only reliable analysis. It is the only approach under the 20% limit except for the sloan matrix where the factorization requires between 20% and 50% more than the prediction. Moreover, the MA47_5 size is always quite good compared to the best approach.

On structurally nonsingular matrices (Figure 6.7), we see that using only a scaling can be very dangerous. Sometimes the computation stops because memory limits are exceeded (see Table 6.5); sometimes the actual number of entries in the factors is far from the analysis estimates (see matrices CONT-201, NCVXQP1, cvxqp3, sawpath ... with MA57 and matrices CONT-201, bratu3d, sawpath ... for MA47). The increase in memory during factorization is always less than 20% with MA57_5 except for the BLOWEYA matrix, but the number of entries in the factors in half of the cases is more than 1.5 times greater than the best strategy. MA47_6 behaves in quite the opposite way. It is always within 1.2 times as good as the best code, except for bratu3d, but its analysis estimates are less reliable than MA57_5

Figure 6.7: Set 2 : ratio between analysis estimation and number of nonzeros after factorization (positive values), ratio between number of nonzeros after factorization and the best strategy.

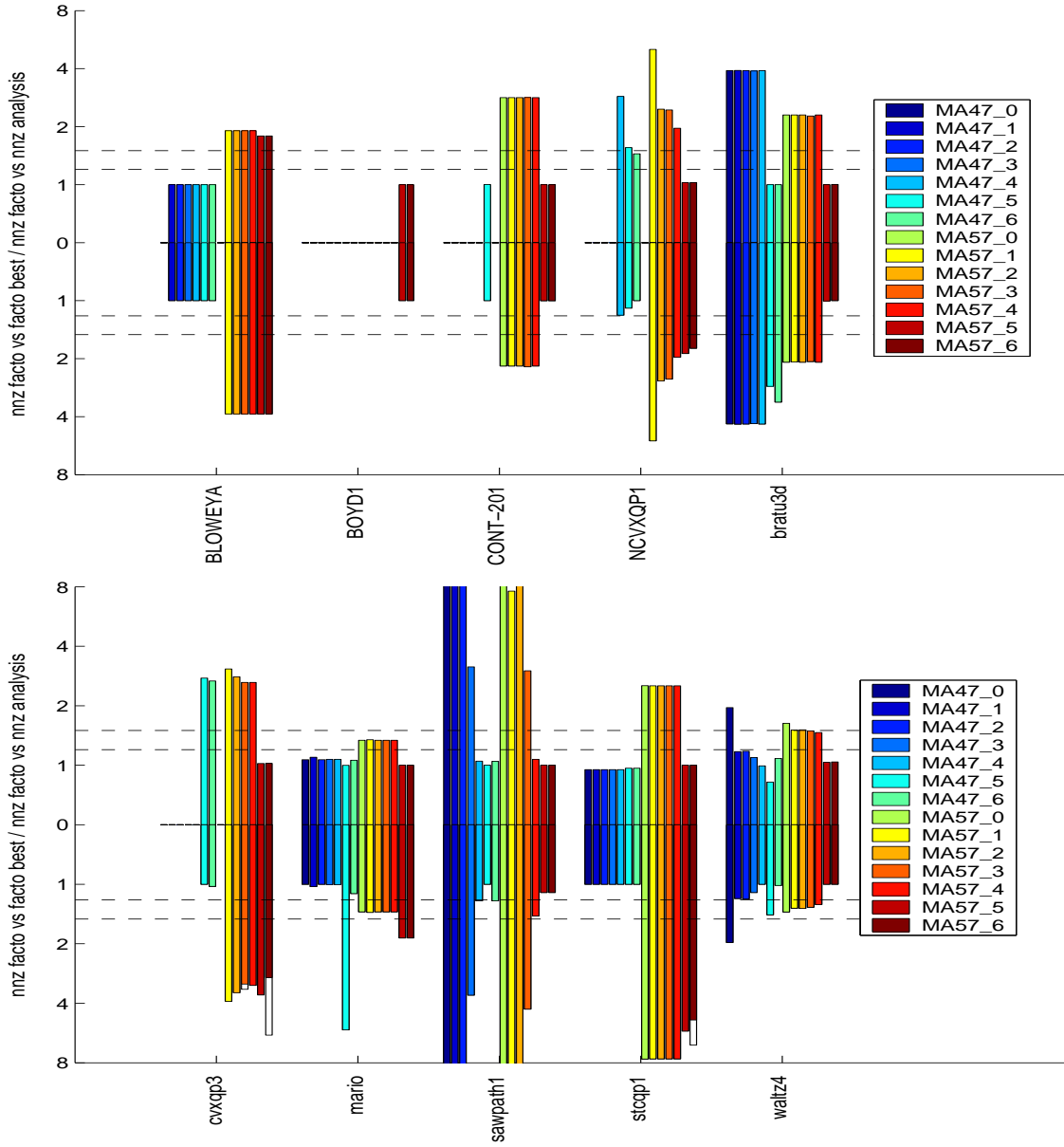
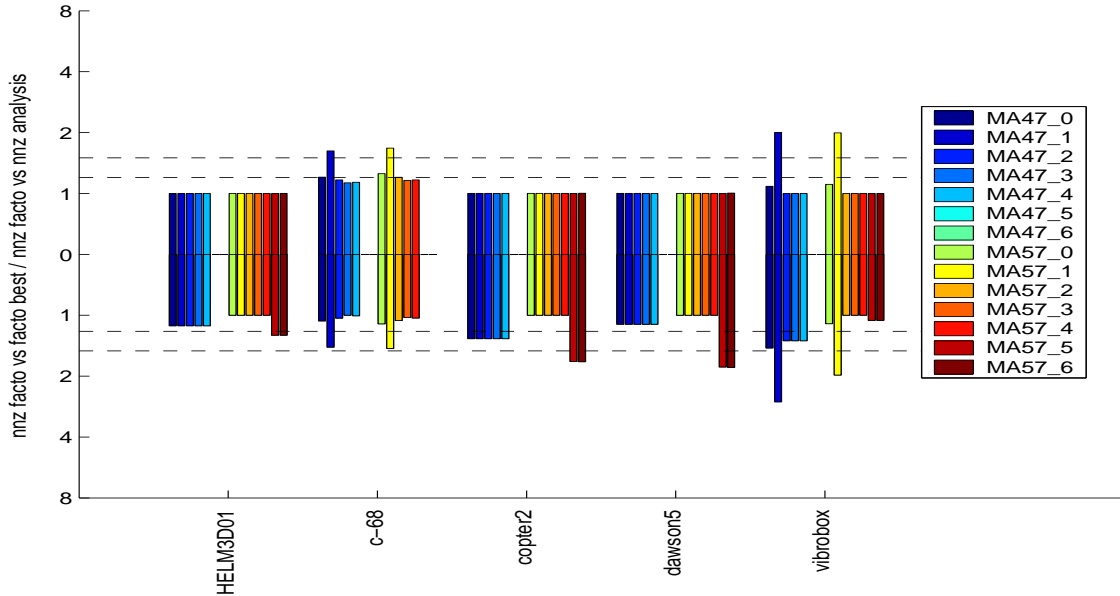


Figure 6.8: Set 3 : ratio between analysis estimation and number of nonzeros after factorization (positive values), ratio between number of nonzeros after factorization and the best strategy.



(the size of L is greater than the 20% limit on three matrices). That is why we recommend running MA47_6 allocating between 50% and 100% more memory for the factorization than predicted by the analysis, but only 20% more for MA57_5/6 on structurally nonsingular augmented systems.

The set of general indefinite symmetric problems seems not to have significant numerical troubles. MA57_4 and MA57_3 have good estimations from the analysis phase and are nearly always the best approach on these problems. MA57_5 and MA57_6 estimations are always within 20% of the memory actually used, but sometimes their factors have 50% more entries than the best code. Perhaps the tested matrices are too “easy” to solve and so do not illustrate the advantages of these two codes. We see that the MA47 analysis predictions are good; but, on this set of matrices, the fill-in when using MA57 is less than for MA47. This explains the slower factorization time for MA47 in this general case.

7 Conclusions and future work

We have shown how MC64 can be used when the matrix is symmetric to effect a symmetric scaling and identify potential 2×2 pivots. We noticed that the use of an appropriate scaling (here MC64 symmetrized scaling) solves a lot of computational difficulties. Indeed all the best approaches on the three sets used it. We have

illustrated the performance of strategies using this extension with two multifrontal codes from HSL. The performance of the two codes depends very much on the nature of the matrix, but we have shown that both codes can benefit from this preprocessing, sometimes with no change, or only minor changes to the multifrontal code. Another benefit of our work is that the analysis phase gives a better indication of the work and storage required by the subsequent factorization. In our future work we will try to answer to the following open questions:

- **How to improve the structural quality of pivot extraction?**

The memory and the number of floating-point operations predicted and performed by the strategies which use the algorithm of extraction of 1×1 and 2×2 pivots, are most of the time larger than the other estimation strategies. The problem is that, when compressing the graph, we lose information the structure of the two rows and columns involved in the 2×2 candidate pivots. These rows/columns are treated as if they had the same structure although they might not have. Perhaps the strategy for pivot extraction can be refined. For example, we can change the structural criterion of the extraction and/or apply another maximum transversal on a matrix containing the entries of the set \mathcal{S} but with a metric based on structural information instead of their numerical values.

- **How to improve the numerical quality of the pivots?**

We saw the gain in terms of reliability of the analysis and discussed some pathological cases. Perhaps the numerical quality of the selected pivot during the analysis can be improved. For example, we want to avoid the selection of 2×2 pivots of the form $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. An answer to this question may be to apply MC64 on a matrix Λ which contains spectral information about the 2×2 pivots (for example $\lambda_{ij} = \min\{\text{singular values of the pivot } (i, j)\}$).

- **What is the more robust way to perturb 2×2 pivots?**

In particular, we may want to have different perturbations in a version of MA47 with static pivoting and in a version of MA57 with static pivoting.

- **Can \mathcal{M}_s^c be included in the reduced matrix?**

In our implementation, we suppressed \mathcal{M}_s^c from the reduced matrix. Then the ordering does not take into account the fill-in in the rows/columns of \mathcal{M}_s^c . It can be taken into account if \mathcal{M}_s^c is included in the reduced matrix and if it is seen as a separator (the rows/columns cannot be selected but the fill-in in this part of the matrix is counted in the structural metric). Another alternative can be to apply an ordering on the reduced matrix without any differentiation between indices representing entries of \mathcal{M}_s and indices representing entries of \mathcal{M}_s^c . On the one hand this solution can be dangerous in term of numerical issues, on the other hand it can improve the forecast number of nonzeros in

the factors and it is likely that the fill-in will create large enough entries on the part of the diagonal corresponding to \mathcal{M}_s^c by the time its variables are eliminated.

- **How can we monitor the choice of the solver on the set 2 ?**

We saw that on structurally nonsingular augmented systems, there is no a best approach for all the cases between MA47_6, MA57_5/6. Perhaps it is possible to detect which code will be the best before the factorization. For example, we remarked that if the analysis of MA47_6 and MA57_5/6 predict the same fill-in or the same number of operations there is more chance that MA57 has the fastest factorization time. This decision would require two analyses. Is there a cheaper decision ? Can we subdivide the set 2 into subsets on which it is possible to detect easily which factorization will be the best?

References

- [1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.
- [2] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for Gaussian elimination. *J. Inst. Maths. Applics.*, 10:118–124, 1972.
- [3] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
- [4] I. S. Duff and J. R. Gilbert. Maximum-weight matching and block pivoting for symmetric indefinite systems. In *Abstract book of Householder Symposium XV, June 17-21, 2002*.
- [5] I. S. Duff. MA57 - a new code for the solution of sparse symmetric definite and indefinite systems. Technical report RAL-TR-2002-024, Rutherford Appleton Laboratory, 2002.
- [6] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(4):889–901, 1999.
- [7] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. Factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11:181–204, 1991.
- [8] I. S. Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL 95-001, Rutherford Appleton Laboratory, 1995.

- [9] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2), 2003.
- [10] D. Ruiz. A scaling algorithm to equilibrate both row and column norms in matrices. Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, 2001.