SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 913-927, September 1990

USE OF THE P[4] AND P[5] ALGORITHMS FOR IN-CORE FACTORIZATION OF SPARSE MATRICES*

M. ARIOLI†§, I. S. DUFF‡, N. I. M. GOULD‡, AND J. K. REID‡

Abstract. Variants of the P[4] algorithm of Hellerman and Rarick and the P[5] algorithm of Erisman, Grimes, Lewis, and Poole, used for generating a bordered block triangular form for the in-core solution of sparse sets of linear equations, are considered.

A particular concern is with maintaining numerical stability. Methods for ensuring stability and the extra cost that they entail are discussed.

Different factorization schemes are also examined. The uses of matrix modification and iterative refinement are considered, and the best variant is compared with an established code for the solution of unsymmetric sparse sets of linear equations. The established code is usually found to be the most effective method.

Key words. sparse matrices, tearing, linear programming, bordered block triangular form, Gaussian elimination, numerical stability

AMS(MOS) subject classifications. 65F50, 65K05, 65F05

1. Introduction.

For solving sparse unsymmetric sets of linear equations

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

Hellerman and Rarick (1971) introduced an algorithm for permuting **A** to bordered block triangular form, which they called the preassigned pivot procedure (P[3]). A little later, Hellerman and Rarick (1972) suggested that the matrix should initially be permuted to block triangular form and that the P[3] algorithm should be applied to each diagonal block; they called this the partitioned preassigned pivot procedure (P[4]). A potential problem with both of these algorithms is that, when Gaussian elimination is applied to the reordered matrix, some of the pivots may be small or even zero. This leads to numerical instability or breakdown of the algorithm. They intended that small pivots should be avoided, but the published explanation of their algorithm is lacking in detail.

Saunders (1976, p. 222) used column interchanges to avoid this difficulty. Erisman, Grimes, Lewis, and Poole (1985) proposed a cautious variant of P[4] that they called the precautionary partitioned preassigned pivot procedure (P[5]). P[5] avoids structurally zero pivots away from the border, but does not address problems associated with small pivots.

Erisman et al. (1985), (1987) performed some extensive numerical tests using as a benchmark the Harwell code MA28 (Duff (1977), Duff and Reid (1979)), which uses the pivotal strategy of Markowitz (1957) and a relative pivot test

$$|a_{kk}^{(k)}| \geqq u \max_{j>k} |a_{kj}^{(k)}| \tag{1.2}$$

* Received by the editors December 28, 1987, accepted for publication (in revised form) July 24, 1989.

† Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, 42 Avenue G. Coriolis, 31057 Toulouse Cedex, France.

‡ Computer Science and Systems Division, Harwell Laboratory, Oxfordshire OX11 0RA, United Kingdom. Present address, Central Computing Department, Atlas Centre, Rutherford Appleton Laboratory, Oxfordshire OX11 0QX, United Kingdom.

§ This research was performed while this author was visiting Harwell Laboratory and was funded by a grant from the Italian National Council of Research (CNR), Istituto di Elaborazione dell'Informazione, CNR, via. S. Maria 46, 56100 Pisa, Italy.

on the elements $a_{kj}^{(k)}$ of the kth pivot row. Here u is a preassigned factor, usually set to 0.1. The Erisman et al. (1985) tests showed their P[4] algorithm encountering zero pivots and therefore failing in more than half the test cases. This illustrates that provision for reordering is an essential part of a reliable algorithm. Erisman et al. (1987) found a full 2×2 block that was exactly singular, which illustrates that it is not sufficient to ensure that the diagonal entries are structurally nonzero. They concluded (1987) in favour of the standard Markowitz approach, as represented by MA28.

To make this paper self-contained, we summarize the properties of the reduction to block triangular form in § 2 and of the P[5] and P[4] algorithms in §§ 3 and 4. For detailed descriptions, we refer the reader to Duff, Erisman, and Reid (1986) or Erisman et al. (1985). Both algorithms permute the matrix to a form that is lower triangular with a few "spike columns" projecting into the upper-triangular part. A practical implementation of P[4] or P[5] needs some provision for reordering to avoid small pivots. We consider this in § 5. We have constructed an experimental code to explore these ideas, and the results are presented in § 6. Finally, we present conclusions in § 7.

We assume throughout the paper that there is enough main storage for the computation to be performed without the use of any form of auxiliary storage.

2. Block triangular matrices. Both the P[4] and the P[5] algorithms start by permuting the matrix to block triangular form

$$\begin{pmatrix} \mathbf{A}_{11} & & & & \\ \mathbf{A}_{21} & \mathbf{A}_{22} & & & \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} & & \\ \cdot & \cdot & \cdot & \cdot & \\ \mathbf{A}_{N1} & \mathbf{A}_{N2} & \mathbf{A}_{N3} & \cdot & \mathbf{A}_{NN} \end{pmatrix}, \tag{2.1}$$

which allows the system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.2}$$

to be solved by block forward substitution

$$\mathbf{A}_{ii}\mathbf{x}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \mathbf{A}_{ij}\mathbf{x}_j, \qquad i = 1, 2, \cdots, N. \tag{2.3}$$

We assume that each block $\mathbf{A}_{ii}$ is irreducible, that is it cannot itself be permuted to block triangular form. There are well-established and successful algorithms for reducing a matrix to this form (see Duff et al. (1986), Chap. 6, for example), and good software is available. It is the treatment of the blocks $\mathbf{A}_{ii}$ that is our concern here.

Because we will subsequently perform a block decomposition of the $\mathbf{A}_{ii}$ blocks, we will use the graph theoretic equivalent term "*strong component*" to identify a block $\mathbf{A}_{ii}$ in the following text.

3. The P[5] algorithm. The P[5] algorithm (Erisman et al. (1985)) first permutes the matrix to block triangular form and then further permutes each strong component to the form illustrated in Fig. 3.1. The general form is of a matrix

$$\begin{pmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{D} & \mathbf{E} \end{pmatrix}, \tag{3.1}$$

where $\mathbf{B}$ is block lower triangular with full diagonal blocks and each column of $\mathbf{C}$ has a leading block of nonzero entries in the rows of a diagonal block of $\mathbf{B}$ and extends upwards at least as far as the preceding column. We refer to the columns of $\binom{\mathbf{C}}{\mathbf{E}}$ as the

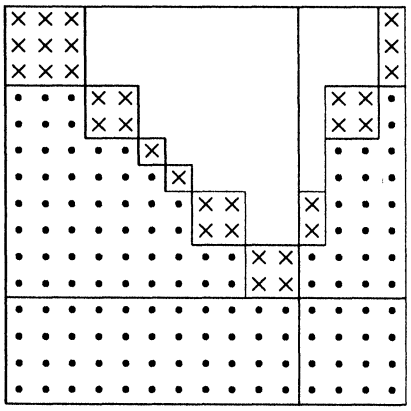

FIG. 3.1. *A strong component after the application of* P^5. *Entries shown as "×" are nonzero and entries shown as "•" may be zero or nonzero.*

"border." For consistency of notation, we refer to any column that projects above the diagonal as a "spike" column, even though they do not look like spikes in the case of the P^5 algorithm. A border column or a column that is not the first of a diagonal block is a spike column, and there are no other spike columns. We refer to the part of a spike column that lies above the diagonal as a "spike."

4. The P^4 algorithm. The P^4 algorithm leads to the same number of spike columns as the P^5 algorithm, but some of the border columns are moved forward. They still have the desirable property that each spike acts as the border of a bordered block triangular matrix in a properly nested set of such matrices. For example, the matrix of Fig. 3.1 might have the form shown in Fig. 4.1. The two spikes in the middle of the border have moved forward and become separated. At the outer level, the blocks have sizes 3, 6, and 5. The first is a full 3×3 matrix. The last is a bordered form with inner blocks that are full 2×2 matrices. The middle one is a bordered form with inner blocks of sizes 4 and 1, the first of which is a bordered form with blocks of sizes 2 and 1.

Our implementation is as described by Duff et al. (1986). Note, in particular, that this version ensures that the diagonal entries are nonzero unless they are in the border.

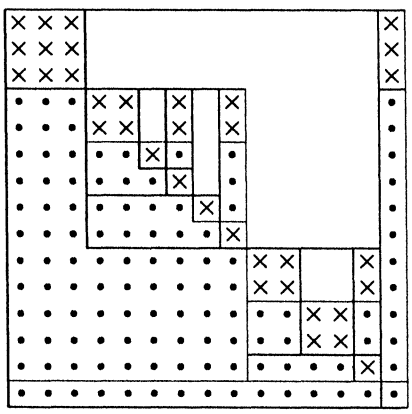

FIG. 4.1. *The matrix of Fig.* 3.1 *after application of* P^4. *Entries shown as "×" are nonzero and entries shown as "•" may be zero or nonzero.*

5. The treatment of spiked matrices. If Gaussian elimination without interchanges is applied to any sparse matrix, including those produced by P^4 and P^5, all fill-in is confined to the spike columns (if column j is not a spike column, it is inactive during steps $1, 2, \cdots, j-1$ of the elimination). Since this produces triangular factorizations of the blocks, we will refer to this as "LU."

An interesting property of the matrices produced by P^4 and P^5 is that they produce a properly nested set; that is, given any pair of spikes, either the set of rows that are cut by the first spike on or above its diagonal is a subset of the corresponding set for the second spike or the two sets of rows do not overlap. Therefore, if Gauss–Jordan elimination is performed by applying row operations to eliminate any entry in the lower triangular part of row 2 (that is, entry (2, 1), if present), then any entry in the upper triangular part of column 2 (that is, entry (1, 2), if present), then all the entries in the lower triangular part of row 3, then all the entries in the upper triangular part of column 3, etc., all fill-in is confined to the spikes (that is, the parts of spike columns that project above the diagonal). Note that if the spikes were not properly nested, fill-in would lengthen some of them. Gauss–Jordan elimination is sometimes called "product form of the inverse" or PFI for short. It is more usual to perform the Gauss–Jordan eliminations column by column.

For numerical stability, Saunders (1976) suggests considering a column interchange whenever the inequality

$$|a_{kk}^{(k)}| \geqq u_1 \max_{i>k} |a_{ik}^{(k)}| \tag{5.1}$$

is not satisfied, where u_1 is a small threshold (often 0.001). He took the largest $a_{kj}^{(k)}$, $j = k, \cdots, n$, as the pivot, but now recommends (private communication) choosing the first spike column, l, such that

$$|a_{kl}^{(k)}| \geqq u_2 \max_{j \geqq k} |a_{kj}^{(k)}|, \tag{5.2}$$

where u_2 is another threshold (usually 0.1), in order that the structure is corrupted least. Indeed, it is possible for a pivot to fail the test (5.1) and yet pass the test (5.2). Note that a column interchange may corrupt the property of the previous paragraph and hence lead to fill-ins that lengthen later spikes.

In both the P^4 and P^5 algorithms, even after column interchanges, we have the block form

$$\begin{pmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{D} & \mathbf{E} \end{pmatrix}, \tag{5.3}$$

where $\mathbf{B}$ and $\mathbf{E}$ are square and the second block column contains all the border columns. We assume that implicit factorization of this block form is used (see George (1974); see also Duff et al. (1986), p. 61); that is, $\mathbf{B}$ is factorized and the Schur complement $\mathbf{E} - \mathbf{D}\mathbf{B}^{-1}\mathbf{C}$ is formed as a full matrix and factorized using conventional interchanges, but $\mathbf{C}$ and $\mathbf{D}$ are stored in their original form without fill-ins. The Schur complement is formed naturally when LU or Gauss–Jordan factorization is used and there is no need to calculate $\mathbf{B}^{-1}$ explicitly, but note that the number of operations for forming it by the two methods is usually different. Where $\mathbf{B}$ has a block structure, we could also use an implicit factorization for $\mathbf{B}$ so that no fill-in is held in its off-diagonal blocks, but have not done this in our experiments because we found that the inner borders were too small to justify the extra complication.

6. Numerical experiments. For our numerical experiments, we have taken the matrices studied by Erisman et al. (1985) and included a few more. We have applied the Harwell code MA28, which performs Gaussian elimination with the ordering of Markowitz (1957) and threshold pivoting. We have used the usual value, $u = 0.1$, for the threshold. MA28 transforms the matrices to block triangular form so that fill-in is confined to the strong components, but implicit factorization of the strong components (see end of § 5) is not available for MA28. We passed the strong components to the Harwell code MC33 that has options for P^4 and P^5 ordering. We then applied the algorithms of § 5 with (u_1, u_2) values of (0.1, 0.1), (0.001, 0.1), and (0.0, 0.0).

The first comment that is worth making concerning our experience is that the block triangular form usually has few nontrivial strong components. Many matrices that occur in practice are irreducible (have only one strong component) and those that are reducible usually have many trivial strong components. The chemical engineering problems considered by Erisman et al. are all reducible and we show the sizes of their strong components in Table 6.1. We also show in Table 6.2 the sizes of the strong components for the linear programming bases BP0, $\cdots$, BP1600 in the Harwell set of sparse matrices. We also use a set of matrices from François Cachard of Grenoble. These matrices arose in the simulation of computing systems (Cachard (1981)) and are all irreducible. The matrices in the test set have some entries that are numerically zero. We processed this data to remove zeros so that the number of entries is less than indicated in the set distributed by Duff, Grimes, and Lewis (1987). In most instances, the runs on the BP matrices showed the same trend as the other sets and did not add to our understanding of the algorithms. We have therefore omitted tables for most of the BP runs. We have also run the remaining problems referenced in the papers of Erisman et al. and have found the relative performance similar to the results displayed.

When the P^4 and P^5 algorithms are applied to the strong components, the final border plays a very important role. Most of the spikes lie within it, as Tables 6.3–6.5 and Figs. 6.1–6.2 show. For the P^5 algorithm, this implies that most of the diagonal blocks are of size 1. We have observed that those not of size 1 are usually of size 2.

TABLE 6.1
The sizes of the strong components of Westerberg's matrices.

Order	No. of components of size 1 or 2	Other sizes
67	1	66
132	55	77
156	133	23
167	90	77
381	5	375
479	165	308
497	291	92; 57; 57
655	197	452
989	269	720
1,505	403	1,099; 3
2,021	521	1,500

TABLE 6.2
The strong component sizes for BP0, · · · , BP1600. *All the matrices have order* 822.

Identifier	No. of components of size 1 or 2	No. of components of size 3 to 9	Other sizes
BP0	822	0	None
BP200	658	12	32, 39, 40
BP400	575	14	22, 161
BP600	523	11	12, 18, 216
BP800	475	15	33, 244
BP1000	446	13	33, 56, 19, 207
BP1200	426	17	33, 65, 220
BP1400	390	12	372
BP1600	431	16	32, 69, 217

TABLE 6.3
The numbers of spikes and border sizes for Westerberg's matrices.

Order	Component size	No. of spikes	Border size, P^5	Border size, P^4
67	66	14	13	11
132	77	6	4	3
156	23	4	4	3
167	77	6	4	3
381	375	75	53	52
479	308	61	42	38
497	92	19	18	16
497	57	1	1	1
497	57	1	1	1
655	452	93	66	54
989	720	98	84	77
1,505	1,099	148	127	116
2,021	1,500	205	175	160

TABLE 6.4
The numbers of spikes and border sizes for the Grenoble set of irreducible matrices.

Order	No. of spikes	Border size, P^5	Border size, P^4
115	19	15	15
185	28	28	28
216	25	25	24
216	25	25	24
343	52	52	42
512	55	55	50
1,107	283	113	100

TABLE 6.5

The numbers of spikes and border sizes for strong components of sizes 10 *or more from the* BP *matrices.*

Identifier	Component size	No. of spikes	Border size, P^5	Border size, P^4
BP200	32	4	2	2
	39	4	3	2
	40	7	3	3
BP400	22	5	2	2
	161	30	21	17
BP600	12	4	3	1
	18	4	3	2
	216	42	27	21
BP800	33	9	6	4
	244	49	38	30
BP1000	33	9	7	5
	56	13	10	6
	19	2	2	2
	207	51	32	26
BP1200	33	9	5	4
	65	15	14	8
	220	49	34	25
BP1400	372	83	54	45
BP1600	32	9	5	4
	69	20	16	8
	217	44	29	24

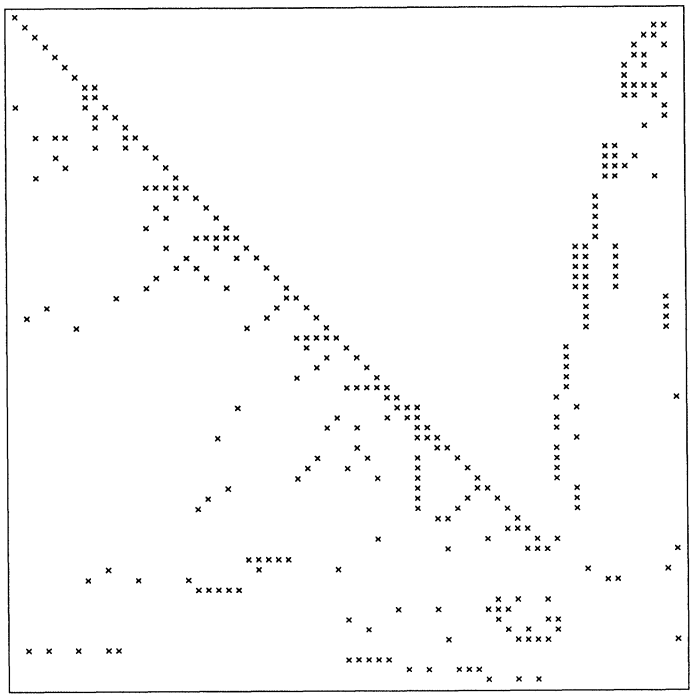

FIG. 6.1. *The matrix of order* 67 *from the Westerberg set after application of* P^5.

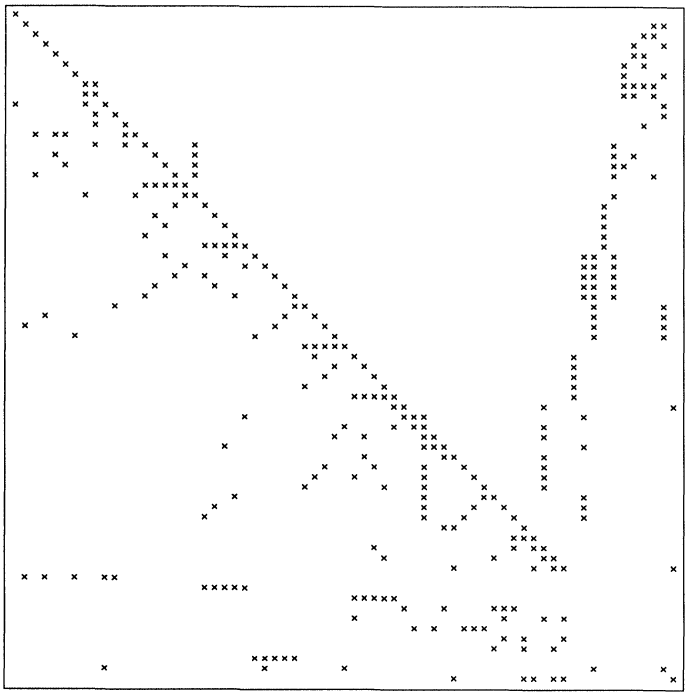

FIG. 6.2. *The matrix of Fig.* 6.1 *after application of* P^4.

In all the tables describing the P^4 and P^5 experiments, we indicate by flop count the number of multiplications, divisions, and additions performed, and we count only the fill-in blocks **B** and **E** of (5.3) since we are using the implicit form of the factorization.

Our experience with Gauss-Jordan elimination, as described in § 5, is that it does indeed often involve less fill-in, but the additional elimination steps lead to a very substantial increase in operation counts; see Tables 6.6 and 6.7. So substantial is this increase that we do not consider this variant further.

TABLE 6.6

Comparison between Gauss-Jordan (GJ) *and* LU *on Westerberg's matrices, using the* P^4 *algorithm and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	GJ	LU	GJ	LU
67	294	53	113	7.4	3.2
132	413	5	17	1.2	0.9
156	362	8	14	0.4	0.2
167	506	13	20	1.3	0.9
381	2,134	1,332	2,708	834	137
479	1,888	704	1,307	336	44
497	1,721	129	254	16	8.0
655	2,808	1,447	2,704	931	119
89	3,518	2,889	4,098	2,345	171
1,505	5,414	6,177	7,474	6,060	368
2,021	7,310	13,225	19,506	23,714	1,756

TABLE 6.7
Comparison between Gauss-Jordan (GJ) *and* LU *on Grenoble matrices, using the* P^4 *algorithm and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	GJ	LU	GJ	LU
115	421	153	373	43	13
185	975	388	1,085	239	77
216	812	299	571	121	25
216	812	339	3,000	320	127
343	1,310	935	1,853	569	89
512	1,976	1,278	2,515	1,235	161
1,107	5,664	6,731	11,382	14,849	1,744

To discover whether our code is sensitive to the way that the choice is made when the heuristics of the algorithms say that more than one column is equally good (that is, sensitivity to tie-breaking), we ran several problems with their columns randomly permuted. We found little sensitivity. For example, 10 runs of the Grenoble case of order 512 had P^5 borders varying between 51 and 55 and fill-in with LU factorization ($u_1 = u_2 = 0.1$) varying between 2,594 and 3,008.

It can be seen from Tables 6.3 and 6.4 that the P^4 algorithm leads to relatively few spikes being moved forward from the border. We might therefore expect that it would not make a large difference to the fill-in or operation count. This is confirmed in Tables 6.8 and 6.9. Usually, but not always, moving spikes forward leads to an improvement. We therefore prefer the P^4 algorithm.

We have stressed the need for interchanges to avoid small pivots. This is essential if accurate solutions are to be obtained, as is illustrated in Tables 6.10 and 6.11, where the relative residuals are

$$\frac{|r_i|}{|b_i| + \sum_j |a_{ij}||\tilde{x}_j|}, \quad i = 1, 2, \cdots \tag{6.1}$$

TABLE 6.8
Comparison between P^5 *and* P^4 *on Westerberg's matrices, using* LU *factorization and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	P^5	P^4	P^5	P^4
67	294	125	113	3.1	3.2
132	413	21	17	0.9	0.9
156	362	15	14	0.2	0.2
167	506	17	20	0.9	0.9
381	2,134	2,712	2,708	139	137
479	1,888	1,576	1,307	53	44
497	1,721	301	254	8.2	8.0
655	2,808	3,776	2,704	177	119
989	3,518	4,317	4,098	199	171
1,505	5,414	11,761	7,474	750	368
2,021	7,310	22,395	19,506	2,064	1,756

TABLE 6.9
Comparison between P^5 *and* P^4 *on Grenoble matrices, using* LU *factorization and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	P^5	P^4	P^5	P^4
115	421	373	373	13	13
185	975	1,085	1,085	77	77
216	812	619	571	27	25
216	812	3,014	3,000	132	127
343	1,310	2,657	1,853	132	89
512	1,976	2,996	2,515	193	161
1,107	5,664	13,848	11,382	2,151	1,744

TABLE 6.10
Comparison between $u_1 = 0.0$, 0.001, *and* 0.1 *on Westerberg's matrices, using* P^4 *and* LU *factorization with* $u_2 = 0.1$. *The arithmetic is* IBM *double precision.*

		Max. relative residual			No. of col. interchanges		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
67	294	5×10^{-15}	5×10^{-15}	6×10^{-15}	0	0	0
132	413	3×10^{-9}	1×10^{-11}	2×10^{-15}	0	2	2
156	362	4×10^{-15}	9×10^{-16}	9×10^{-16}	0	1	1
167	506	3×10^{-9}	2×10^{-13}	4×10^{-15}	0	2	3
381	2,134	fails	4×10^{-10}	1×10^{-12}	0	2	6
479	1,888	1×10^{-6}	4×10^{-11}	4×10^{-14}	0	6	10
497	1,721	fails	4×10^{-11}	4×10^{-14}	0	0	3
655	2,808	3×10^{-7}	6×10^{-11}	1×10^{-13}	0	6	10
989	3,518	4×10^{-7}	7×10^{-12}	4×10^{-14}	0	28	39
1,505	5,414	1×10^{-7}	3×10^{-10}	1×10^{-13}	0	42	63
2,021	7,310	4×10^{-6}	1×10^{-9}	2×10^{-13}	0	56	77

TABLE 6.11
Comparison between $u_1 = 0.0$, 0.001, *and* 0.1 *on Grenoble matrices, using* P^4 *and* LU *factorization with* $u_2 = 0.1$. *The arithmetic is* IBM *double precision.*

		Max. relative residual			No. of col. interchanges		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
115	421	5×10^{-10}	5×10^{-10}	1×10^{-13}	0	0	11
185	975	2×10^{-8}	2×10^{-8}	2×10^{-12}	0	0	16
216	812	1×10^{-9}	1×10^{-9}	1×10^{-9}	0	0	0
216	812	fails	fails	4×10^{-14}	0	0	85
343	1,310	1×10^{-9}	1×10^{-9}	1×10^{-9}	0	0	0
512	1,976	4×10^{-2}	4×10^{-2}	4×10^{-2}	0	0	0
1,107	5,664	fails	fails	fails	0	3	24

and $\tilde{\mathbf{x}}$ is the computed solution. The number of interchanges is shown in the tables. Interchanges usually lead to an increase in fill-in and operation count (illustrated in Tables 6.12 and 6.13) principally because of exchanges between the border and nonborder columns.

For the Westerberg matrices, the choice of $u_1 = 0.1$ gives good residuals, without an excessive increase in fill-in or operation count; the choice $u_1 = 0.001$ yields higher residuals, but they are reasonably satisfactory.

For the Grenoble matrices, the choice $u_1 = 0.001$ yields unsatisfactory residuals in cases 4, 6, and 7, and the choice $u_1 = 0.1$ yields unsatisfactory residuals in cases 6 and 7. We regard the use of the small value of 0.001 as "living dangerously" and are not surprised by an occasional poor result, but the last two results with the value 0.1 prompted us to investigate further. We found that they were indeed caused by large growth in the size of the matrix entries. For case 6 (order 512), we found that increasing u_1 to 0.2 did not help but that increasing it to 0.5 reduced the maximum relative residual to 1×10^{-6}. For case 7 (order 1107), the values 0.2 and 0.5 reduced the residuals to 1×10^{-9} and 6×10^{-13}, respectively. Increasing u_2 made little difference to the results.

TABLE 6.12

Comparison between $u_1 = 0.0$, 0.001, and 0.1 on Westerberg's matrices, using P^4 and LU factorization with $u_2 = 0.1$.

		Fill-in			Factorization flop count (thousands)		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
67	294	117	117	113	3.2	3.2	3.2
132	413	18	30	17	1.0	1.1	0.9
156	362	13	14	14	0.2	0.2	0.2
167	506	16	22	20	0.9	1.0	0.9
381	2,134	2,626	2,707	2,708	134	148	137
479	1,888	1,299	1,438	1,307	41	50	44
497	1,721	255	256	254	9.3	9.6	8.0
655	2,808	2,421	2,918	2,704	93	131	119
989	3,518	3,232	2,938	4,098	112	98	171
1,505	5,414	5,834	6,352	7,474	263	260	368
2,021	7,310	16,501	16,353	19,506	1,263	1,178	1,756

TABLE 6.13

Comparison between $u_1 = 0.0$, 0.001, and 0.1 on Grenoble matrices, using P^4 and LU factorization with $u_2 = 0.1$.

		Fill-in			Factorization flop count (thousands)		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
115	421	227	227	373	6.9	6.9	13
185	975	771	771	1,085	42	42	77
216	812	571	571	571	25	25	25
216	812	584	584	3,000	25	25	127
343	1,310	1,853	1,853	1,853	89	89	89
512	1,976	2,515	2,515	2,515	161	161	161
1,107	5,664	10,281	10,784	11,382	1,113	1,265	1,744

These results illustrate the potential pitfalls with any particular choice of u_1, but our general recommendation is nevertheless for the value 0.1, which we use for the later comparisons in this paper. Note that iterative refinement may be used to improve any solution; in particular, we found it to be successful for the poor factorizations discussed in the previous paragraph.

We have also worked with the strategy of performing a column interchange whenever the inequality (1.2) is not satisfied, which has software advantages if storage by rows is in use. This strategy is equivalent to the use of a value of u_1 that is greater than 1.0. We found that it almost always leads to more column interchanges and hence to more fill-in and work, though its stability properties are better (for instance, we did not have such serious difficulties with the last two Grenoble matrices).

As an alternative to performing column interchanges, we also considered changing the diagonal elements $a_{kk}^{(k)}$ whenever they do not satisfy the pivot test (1.2). The solution of (1.1) may then be obtained using the modification method (see, for example, Duff et al. (1986), pp. 244–247). This technique has the advantage of allowing predefined storage structures but when r diagonal elements are altered it has the disadvantage of requiring the solutions of $r+1$ linear systems, each of whose coefficient matrix is the perturbed matrix. Unfortunately, the size of r is normally about the same as the number of column interchanges that would otherwise have been performed with $u_1 = u_2 = 0.1$, and this makes such an approach prohibitively expensive (see Tables 6.10 and 6.11).

Another alternative is to ignore the pivot test (1.2) but to check the absolute value of each entry on the diagonal and to increase it by a value μ if it is less than that value. Usually the Schur complement is full and we can factorize it by an LU decomposition with interchanges that makes the pivots the largest entries of the columns (or rows). This strategy is particularly useful for P^5, because the fill-in can be confined to the Schur complement. Obviously the solution obtained may be poor, but it is possible to improve the error by performing a few steps of iterative refinement. In Tables 6.14 and 6.15, we show results for Westerberg's matrices and those from Grenoble using a value for μ of 1×10^{-8} (approximately the square root of the machine precision).

This approach does not guarantee that iterative refinement converges because the error in the factorization could be too large. For example, with the Grenoble matrix of order 1107 iterative refinement does not converge, because the factorization is

TABLE 6.14

Results on Westerberg's matrices, using LU *factorization with iterative refinement and* P^5, *incrementing the pivot by* 1×10^{-8} *if it is less than* 1×10^{-8}. *The arithmetic is* IBM *double precision.*

Order	Nonzeros	Fill-in	Flop count (thousands)		Num. iter. steps	Error
			Factorization	Solution		
67	294	125	3.1	1.4	0	5×10^{-15}
132	413	15	0.9	0.9	0	8×10^{-10}
156	362	13	0.2	0.6	0	3×10^{-8}
167	506	13	0.9	1.1	0	4×10^{-10}
381	2,134	2,721	139	33	1	2×10^{-12}
479	1,888	1,478	46	16	1	4×10^{-11}
497	1,721	293	9.1	4.4	0	2×10^{-10}
655	2,808	2,756	106	30	1	5×10^{-11}
989	3,518	3,121	104	30	1	2×10^{-10}
1,505	5,414	10,172	598	69	1	3×10^{-10}
2,021	7,310	18,078	1,397	115	1	2×10^{-9}

TABLE 6.15

Results on Grenoble matrices, using LU *factorization with iterative refinement and* P^5, *incrementing the pivot by* 1×10^{-8} *if it is less than* 1×10^{-8}. *The arithmetic is* IBM *double precision* (*note that the fourth matrix is very ill-conditiond*).

Order	Nonzeros	Fill-in	Flop count (thousands)		Num. iter. steps	Error
			Factorization	Solution		
115	421	222	6.9	6.1	1	8×10^{-16}
185	975	771	42	20	1	5×10^{-13}
216	812	617	27	19	1	4×10^{-17}
216	812	617	27	80	7	6×10^{-2}
343	1,310	2,657	132	50	1	2×10^{-16}
512	1,976	2,996	193	265	5	4×10^{-13}
1,107	5,664	12,723	1,458		Diverged	

unstable with entries of size 1×10^{33}. Our view is that this approach does not show sufficient advantages to compensate for the lack of robustness, and we therefore reject it.

Finally, in Tables 6.16–6.18, we show a comparison between the most satisfactory of our algorithms, P^4 with LU factorization and thresholds $u_1 = u_2 = 0.1$, and the Harwell Markowitz code MA28, with threshold $u = 0.1$. The P^4 algorithm often involves less fill-in, but in most cases it requires more operations, sometimes considerably more.

7. Conclusions. We have compared the P^4 and P^5 variants of the Hellerman–Rarick algorithm and found that P^4 is usually better than P^5, but not by much. The special form of Gauss–Jordan elimination that confines fill-in to the spikes themselves usually requires far more operations than LU factorization and does not always lead to less fill-in. We therefore prefer P^4 with LU factorization.

The use of interchanges is essential if a reliable solution is to be obtained, even though the interchanges may lead to an increase in computation.

We also tried to maintain the structure during the factorization by modifying the pivot when it was too small. We examined the possibility of using updating schemes

TABLE 6.16

Comparison between MA28 *and* P^4 *on Westerberg's matrices, using* LU *factorization and thresholds* $u = u_1 = u_2 = 0.1$.

Order	Nonzeros	Fill-in		Factorization flop count (thousands)		Solution flop count (hundreds)	
		MA28	P^4	MA28	P^4	MA28	P^4
67	294	267	113	2.1	3.2	11	8.5
132	413	105	17	0.5	0.9	6.3	5.3
156	362	26	14	0.1	0.2	1.5	1.2
167	506	97	20	0.5	0.9	6.2	5.4
381	2,134	1,941	2,708	25	137	76	102
479	1,888	1,104	1,307	9.8	44	44	54
497	1,721	299	254	2.4	8.0	19	20
655	2,808	2,223	2,704	22	119	80	101
989	3,518	1,194	4,098	7.2	171	69	137
1,505	5,414	1,984	7,474	12	368	109	233
2,021	7,310	2,659	19,506	16	1,756	147	505

TABLE 6.17
Comparison between MA28 *and* P^4 *on Grenoble matrices, using* LU *factorization and thresholds* $u = u_1 = u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)		Solution flop count (hundreds)	
Order	Nonzeros	MA28	P^4	MA28	P^4	MA28	P^4
115	421	654	373	5.6	13	20	19
185	975	3,134	1,085	63	77	80	53
216	812	2,544	571	39	25	65	29
216	812	2,156	3,000	25	127	57	128
343	1,310	5,334	1,853	119	89	129	67
512	1,976	11,545	2,515	402	161	265	94
1,107	5,664	39,316	11,382	1,928	1,744	889	403

TABLE 6.18
Comparison between MA28 *and* P^4 *on* BP *matrices, using* LU *factorization and thresholds* $u = u_1 = u_2 = 0.1$.

			Fill-in		Factorization flop count (thousands)		Solution flop count (hundreds)	
Identifier	Order	Nonzeros	MA28	P^4	MA28	P^4	MA28	P^4
BP0	822	3,276	0	0	0	0	0	0
BP200	822	3,802	106	25	0.8	1.3	11	10
BP400	822	4,028	266	196	2.0	5.2	20	22
BP600	822	4,172	484	358	3.5	11	30	34
BP800	822	4,534	681	874	5.9	31	41	56
BP1000	822	4,661	766	827	7.1	31	47	63
BP1200	822	4,726	959	843	9.2	37	54	69
BP1400	822	4,790	1,295	1,747	11.4	105	65	91
BP1600	822	4,841	872	734	8.4	34	53	68

but found them to be prohibitively expensive. We also considered iterative refinement and found this sometimes to be very competitive, but disliked its lack of robustness.

Apart from numerical considerations, Erisman et al. (1985), (1987) found the P^5 algorithm to be competitive with the Markowitz algorithm, but were pessimistic about being able to ensure numerical stability. Our comparisons, using the same threshold factor in the two algorithms and taking the number of operations into account, confirm that the Markowitz algorithm is comparable with respect to fill-in and indicate that it is usually superior with respect to factorization operation count. Of course, it should be borne in mind that the Hellerman–Rarick algorithms never need access by rows and are therefore better suited to out-of-core working. They also have a less expensive analysis phase. For example, on the Grenoble matrix of order 1107, our analysis time was 1.2 seconds, whereas MA28 took 60 seconds for the phase that performs both analysis and factorization.

Acknowledgments. We express our thanks to both referees, whose comments have substantially improved the paper. We are also indebted to Michael Saunders for drawing our attention to the double-threshold strategy for column interchanges.

REFERENCES

F. CACHARD (1981), *Logiciel numerique associé à une modelisation de systèmes informatiques*, Thèse, Université Scientifique et Médicale de Grenoble, et l'Institut National Polytechnique de Grenoble, Grenoble, France.

I. S. DUFF (1977), MA28—*A set of Fortran subroutines for sparse unsymmetric linear equations*, Report AERE R8730, Her Majesty's Stationery Office, London, United Kingdom.

I. S. DUFF AND J. K. REID (1979), *Some design features of a sparse matrix code*, ACM Trans. Math. Software, 5, pp. 18-35.

I. S. DUFF, A. M. ERISMAN, AND J. K. REID (1986), *Direct Methods for Sparse Matrices*, Oxford University Press, London.

I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS (1987), *Sparse matrix test problems*, ACM Trans. Math. Software, 15, pp. 1-14.

A. M. ERISMAN, R. G. GRIMES, J. G. LEWIS, AND W. G. POOLE, JR. (1985), *A structurally stable modification of Hellerman-Rarick's P^4 algorithm for reordering unsymmetric sparse matrices*, SIAM J. Numer. Anal., 22, pp. 369-385.

A. M. ERISMAN, R. G. GRIMES, J. G. LEWIS, W. G. POOLE, JR., AND H. D. SIMON (1987), *Evaluation of orderings for unsymmetric sparse matrices*, SIAM J. Sci. Statist. Comput., 8, pp. 600-624.

A. GEORGE (1974), *On block elimination for sparse linear systems*, SIAM J. Numer. Anal., 11, pp. 585-603.

E. HELLERMAN AND D. C. RARICK (1971), *Reinversion with the preassigned pivot procedure*, Math. Programming, 1, pp. 195-216.

——— (1972), *The partitioned preassigned pivot procedure* (P^4), in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, pp. 67-76.

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

M. A. SAUNDERS (1976), *A fast, stable implementation of the simplex method using Bartels-Golub updating*, in Sparse Matrix Computations, Bunch and Rose, eds., Academic Press, New York, pp. 213-226.