



UNIVERSITÄT KARLSRUHE

Symmetric Permutations for I-matrices to Delay and
Avoid Small Pivots During Factorization

Jan Mayer

Preprint Nr. 06/08

Institut für Wissenschaftliches Rechnen
und Mathematische Modellbildung



76128 Karlsruhe

Anschrift des Verfassers:

Dr. Jan Mayer
Institut für Angewandte und Numerische Mathematik
Universität Karlsruhe
D-76128 Karlsruhe

Symmetric Permutations for I-matrices to Delay and Avoid Small Pivots During Factorization

JAN MAYER

INSTITUT FÜR ANGEWANDTE UND NUMERISCHE MATHEMATIK
UNIVERSITÄT KARLSRUHE

Abstract

In this article, we present several new permutations for I-matrices making these more suitable for incomplete LDU-factorization preconditioners used in solving linear systems by iterative methods. A general matrix can be transformed by row permutation as well as row and columns scaling into an I-matrix, i.e. a matrix having elements of modulus 1 on the diagonal and elements of modulus of no more than 1 elsewhere. Reordering rows and columns by the same permutation clearly preserves I-matrices. In this article, we consider such reordering techniques which make the permuted matrix more suitable for an incomplete LDU-factorization preconditioner than the original I-matrix. We use a multilevel ILUC, an incomplete LDU-factorization preconditioner using Crout's implementation of Gaussian elimination without pivoting to test these reorderings. The combination of I-matrix preprocessing with the various algorithms presented here and the multilevel incomplete LDU-factorizations forms a powerful preconditioning method for unsymmetric, highly indefinite problems.

Key words: preconditioning, I-matrix, reordering, ILUC, iterative methods, sparse linear systems.

AMS subject classification: 65F10, 65F50.

1 Introduction

Reordering rows and columns is a very successful technique for making sparse matrices more suitable for incomplete LU-factorization and subsequent solving by an iterative method. Initially, when mostly symmetric, positive definite problems were being investigated, the focus was on Reverse Cuthill-McKee, approximate minimal degree, nested dissection or similar permutations. These are "symmetric" permutations, i.e. the same permutation is applied to rows and columns, so as to preserve the symmetry and positive definiteness of the coefficient matrix A , while changing the structure of the matrix in such a manner that less fill-in will occur during factorization. Without dropping, no zero pivot can occur for a Cholesky factorization of a symmetric, positive definite matrix so that these reorderings aim only at reducing fill-in, rather than avoiding small (or zero) pivots. Furthermore, dropping elements often has the additional, desirable effect, of increasing pivots, so a zero or small pivot is generally not an issue for symmetric, positive definite problems. Also recall that the reorderings mentioned above only make use of the structure of the matrix (i.e. whether a certain element is zero or not), but not of the actual elements themselves.

These reorderings are also useful for non-symmetric matrices A , whenever zero pivots are not a primary concern. This is the case, for example, if the matrix is diagonally dominant. Then, the reordering is calculated using the non-zero structure of $A + A^T$, making this approach particularly attractive for symmetrically structured problems. However, for many non-symmetric problems, a zero pivot can occur during the course of factorization so that simply using a symmetric permutation is not very promising for a large number of problems. Indeed, if a matrix is not symmetric, particularly if it is not even symmetrically structured, there is no reason to restrict oneself to using symmetric permutations for preprocessing. But before non-symmetric permutations were considered, developments went in a slightly different direction. The main reason for this is probably the fact that no non-symmetric permutation can be constructed easily which would circumvent pivoting in all cases. Hence, the focus was on designing new incomplete LU-factorizations such as ILUTP, see [15], or ILUCP, see [11], which implement pivoting by columns. Although preprocessing with one of the symmetric permutations mentioned before could still be useful for some matrices, generally the pivoting destroyed the reordering to such a degree that it offered little benefit.

However, non-symmetric permutations seemed natural for use with non-symmetric matrices, so they did receive more attention, see for example [1] for an overview. One of the most promising approaches was initiated by Neumaier and Olschowska, see [14]. It is clear that the rows of a matrix can be permuted in such a manner that the product of the absolute values of the diagonal elements become maximal. Neumaier and Olschowska construct an algorithm for finding that permutation efficiently for a dense matrix. Additionally, they prove that the rows and columns of the permuted matrix can be scaled in such a manner that we obtain an I-matrix, i.e. a matrix having elements of modulus 1 on the diagonal and elements of modulus of at most 1 elsewhere. (Note that Duff and Koster developed algorithms for sparse matrices, see [6] and [7].) Intuitively, it is reasonable to expect that I-matrices are more suitable for Gaussian elimination and numerical experiments do indicate that this is the case, see [1], but zero pivots can be encountered nonetheless.

A somewhat different approach is due to Saad, see [16]. He constructs “PQ reorderings” which aim at improving diagonal dominance of the rows of a matrix upto a particular index, i.e. of an initial block matrix. Although there are no guarantees, frequently pivoting can be avoided in this block. After factorization of this block has been completed, an approximate Schur complement is calculated, which is then factored as before. This recursion results in a multilevel scheme.

Similarly, it is natural and certainly possible to factor (incompletely) an I-matrix until the absolute value of a pivot becomes too small, to calculate an (approximate) Schur complement and to proceed recursively. However, this approach does not appear to have been considered extensively. As it is obvious that I-matrices are preserved by symmetric permutations, it seems natural to further preprocess I-matrices using one of the symmetric permutations mentioned above. Although this approach may reduce fill-in, it does not take into account that for many matrices we should still expect zero (or small) pivots. Using these permutations often results in a higher number of zero pivots than necessary, so that more levels are needed to factor A , which is fairly expensive. In some cases, the number of levels required makes an incomplete factorization practically impossible in reasonable time. So instead, we will investigate in this article symmetric permutations

that are more suitable for I-matrices. They aim at both reducing fill-in and avoiding small pivots. The basic idea is to eliminate those rows and columns first which are likely to alter the element on the diagonal as little as possible. This suggests that rows and columns having few elements or whose off-diagonal elements have small modulus should be eliminated first.

The idea for constructing permutations in this manner, which can also be interpreted as an attempt to improve diagonal dominance, is of course, not new and at least some of permutations can be seen as symmetrized versions of the PQ reorderings mentioned before. Nevertheless, there are several important differences. PQ reorderings select rows having an element with good dominance properties (i.e. the modulus of one element is larger than the others or the weighted sum of the others), move these rows to the top of the matrix and permute the columns such that the element having large modulus become the diagonal element. As these permutations are designed for general matrices and not I-matrices, these permutations will generally not be symmetric. Furthermore, the elements with largest modulus of different rows may lie in the same column, so that one of the rows cannot be used to improve diagonal dominance and must be rejected, i.e. moved to a higher index arbitrarily. Thus, PQ reorderings generally do not improve the diagonal dominance properties for the entire matrix, but only upto a particular index. This index is a natural point to terminate a level in factoring, especially if no pivoting is used, and to proceed in calculating the Schur complement. However, for I-matrices, this problem can be avoided. For I-matrices, no off-diagonal element has a modulus larger than the diagonal element, hence we can always choose the column corresponding to a particular row index, thus preserving the diagonal. The algorithm simply needs to be modified slightly to do so. In the unmodified form, whenever several elements in a particular row have modulus 1, the “wrong” column may be selected, i.e. a column having a index different from the row.

The first permutation that we will consider in this article is in fact this symmetrized PQ reordering. The other permutations will be variants of this simple idea, all moving rows and columns having the good dominance properties to low indices, so that they will be eliminated first. However, for these permutations, we know that the reordered matrix will also be an I-matrix, so there is no reason to terminate a level at the particular index indicated by a PQ reordering. Instead, we only terminate a level if the modulus of the pivot becomes too small. Generally, this works well and requires fewer levels than the other approach.

After describing the new symmetric permutations for I-matrices in the next section, we will continue with numerical experiments illustrating the usefulness of this technique. We will use a simple multilevel ILUC (modified slightly to have an incomplete LDU-factorization) with levels being terminated whenever the absolute value of a pivot becomes too small. See [4] for details on ILUC as well as [2], [3] and [16] for information on multilevel preconditioners. As a dropping rule, we used an error propagation reduction strategy that also aims at reducing the error in L and U , see [10] and [12]. We also considered further preprocessing, such as applying a multilevel nested dissection permutation to reduce fill-in, see [9], prior to using the various permutations presented here. As can be expected, the effects of further preprocessing were generally negligible. This is not surprising because the final symmetric permutation is likely to destroy any advantage that the multilevel nested dissection permutation might have provided.

2 Symmetric Permutations for I-Matrices

There are basically two types of permutations which will be considered. The idea of the construction of the first type is very simple. Given an $(n \times n)$ I-matrix A , weights are assigned to each index k , $k = 1, \dots, n$ based on the diagonal dominance properties as well as the number of non-zero elements of the k -th row and column of A in such a manner that low weights indicate good properties. Then, the rows and columns are reordered by increasing weights. The main advantage of this approach lies in the ease of implementation and the fact that A only needs to be available in either compressed sparse row or column format, which is usually the case. The other type of permutation proceeds inductively. Assuming that k rows and columns with indices $I_k = i_1, \dots, i_k$ have already been selected such that the submatrix corresponding to I_k has improved properties, we select the index i_{k+1} such that the submatrix corresponding to $I_{k+1} = i_1, \dots, i_{k+1}$ has the “best” properties for all possible choices of i_{k+1} . Implementing this approach is more involved, as it requires more data to be manipulated and also requires A to be available so that arbitrary rows and columns can be accessed easily. De facto means that A must be available in both compressed sparse row and column format. In the sequel, we assume familiarity with these formats. Details can be found in [15].

For the first type of algorithm, we allow the weight w_k associated with index k to depend on the norm $\|a_{k,:}\|_1$ of the k -th row of A , on the norm $\|a_{:,k}\|_1$ of the k -th column of A , on the number $\text{nnz}(a_{k,:})$ of non-zero elements of the k -th row of A and on the number $\text{nnz}(a_{:,k})$ of non-zero elements of the k -th column of A . These are obvious candidates for calculating the weights. Having few non-zero elements will likely result in little fill-in in, avoiding modifications of the pivot. Furthermore, little fill-in means that fewer elements have to be dropped to attain a prescribed level of sparsity, usually resulting in a more accurate factorization. Furthermore, having to drop fewer elements in the initial phase of factorization generally means that fewer errors are propagated during the course of factorization. As the diagonal element always has modulus 1, the norms of the rows and columns are a measure of the diagonal dominance of rows and columns respectively. They are diagonally dominant, if and only if the respective norm is less than 2. Hence, sorting by increasing norms is equivalent to sorting by decreasing diagonal dominance. So in order to implement this approach, we must calculate the quantities mentioned above efficiently, then calculate the weight and finally select the indices.

We will assume that A is stored in compressed sparse row format, so that the rows of A can be accessed efficiently. The implementation for sparse compressed column format is similar. Even though this approach is simple and straightforward, for the sake of completeness, we summarize it in algorithm 1. It is important to point out that if the weights do not actually depend on all the quantities listed, then this allows for some obvious simplifications. Let nz_r , nz_c , nr and nc denote vectors of dimension n used for storing the number of non-zero elements and the norms of the rows and columns of A . The vector w will store the n weights and P will be the permutation to reorder A .

Algorithm 1: simple greedy selection

1. Initialize nz_r , nz_c , nr and nc as zero vectors of dimension n
2. Initialize P as the identity

3. **for** $k = 1, \dots, n$
4. Calculate $\text{nz_r}(k)$ using the pointers associated with A
5. **for** $i = 1, \dots, n$ and $a(k, i) \neq 0$
6. $\text{nr}(k) = \text{nr}(k) + |a(k, i)|$
7. $\text{nc}(i) = \text{nc}(i) + |a(k, i)|$
8. $\text{nz_c}(i) = \text{nz_c}(i) + 1$
9. **end for** i
10. **end for** k
11. **for** $k = 1, \dots, n$
12. Calculate $w(k)$ using $\text{nz_r}(k)$, $\text{nz_c}(k)$, $\text{nr}(k)$ and $\text{nc}(k)$
13. **end for** k
14. Sort w and permute the elements of P analogously

There are some obvious choices for calculating the weights w . Among others, we considered the following:

- sPQ) $w(k) = \text{nr}(k) \cdot \text{nz_r}(k)$;
- a) $w(k) = \text{nr}(k) + \text{nc}(k)$;
 - b) $w(k) = \text{nz_r}(k) + \text{nz_c}(k)$;
 - c) $w(k) = (\text{nr}(k) + \text{nc}(k)) \cdot (\text{nz_r}(k) + \text{nz_c}(k))$;
 - d) $w(k) = \text{nr}(k) \cdot \text{nz_r}(k) + \text{nc}(k) \cdot \text{nz_c}(k)$;

The implementation sPQ) is essentially a symmetrized version of the simple greedy PQ-algorithm mentioned in the introduction, see algorithm 3.2 in [16] aiming to improve the (weighted) diagonal dominance of rows. In other words, it guarantees that the permutation used for the rows and columns is the same. Given that we use an LDU-factorization and that we have a large degree of symmetry with respect to rows and columns, just using rows as in sPQ) in determining a permutation is perhaps somewhat unnatural. So even though sPQ) is a natural starting point for investigating suitable permutations for I-matrices, it lacks this symmetry and indeed all other permutations considered are completely symmetric with respect to rows and columns, i.e. the weights for A and A^T are equal. Hence, they aim at improving the dominance properties of both rows and columns. Although [16] indicates that other PQ algorithms, such as algorithm 4.3 using dynamic averages, perform better, our numerical experiments did not confirm this for I-matrices. For this reason and the simple fact that these other PQ algorithms in [16] do not appear to allow for simple symmetrization, we focus on the simple greedy approach outlined in algorithm 3.2 of [16] and its modifications.

Note that no weighing strategy requires all vectors listed. To implement sPQ), we just need a single vector, nr , to store the norms of the rows. After these norms have been calculated, nr can be multiplied elementwise by the number of elements per row and sorted. Similarly, the weights in a) and b) can be calculated directly and stored in w , so that w is actually the only vector we need. Approach c) requires separate vectors to store the norms and the number of non-zero elements, but one of these can be overwritten with the weights, so that in total only two vectors of dimension n are needed. Finally, to implement strategy d), we need three vectors to store nr , nc and nz_c . Using this data and the fact that the number of elements in a row can be calculated immediately using

the sparse compressed row format, the weights can be calculated and stored in one of these vectors.

Although these approaches generally worked well, the number of small pivots encountered during factorization was fairly high for a number of matrices, leading to a large number of levels or even to a failure of the preconditioner. So instead of selecting rows and columns to have the best diagonal dominance and sparsity properties overall, we considered alternatives which only aim at improving the diagonal dominance and sparsity of the initial $(k \times k)$ -block consisting of rows and columns having indices $1, \dots, k$. For this purpose, we assume that in the k -th step, we have already selected a set of indices I_k and that we have a vector w of weights. The index of the smallest element of w not yet in I_k should indicate the most suitable row and column to be eliminated at the $(k+1)$ -th step. Hence, this index is added to I_k to form I_{k+1} and then w is updated. In order to find the smallest element of w quickly, it is best to implement w as a binary tree. Furthermore, a vector of length n needs to point to the various elements of w so that w can be accessed by indices as well. This is needed so that the elements of w can be updated efficiently (by removing and reinserting into the binary tree). Using these ideas, we obtain algorithm 2:

Algorithm 2: greedy selection for initial block

1. Initialize w as a binary tree having n elements, all set to 0.
2. **for** $k = 1, \dots, n$
3. $P(k) = \operatorname{argmin}(w)$
4. remove $w(P(k))$ from w
5. **for** $i = 1, \dots, n$ and $w(i)$ not yet removed
6. Update $w(i)$
7. **end for** i
8. **end for** k

Again, there are several possibilities for calculating and updating the weights in step 6:

- a) $w(i) = w(i) + |a(i, P(k))| + |a(P(k), i)|$
- b) **if** $a(i, P(k)) \neq 0$ **then** $w(i) = w(i) + 1$
 if $a(P(k), i) \neq 0$ **then** $w(i) = w(i) + 1$
- c) $w(i) = w(i) + (|a(i, P(k))| + |a(P(k), i)|) \cdot (\operatorname{nz_r}(i) + \operatorname{nz_c}(i))$
- d) $w(i) = w(i) + |a(i, P(k))| \cdot \operatorname{nz_r}(i) + |a(P(k), i)| \cdot \operatorname{nz_c}(i)$

The weights in strategy b) are the number of non-zero elements belonging to the i -th row plus those belonging to the i -th column. Hence, in the k -th step, an index will be selected so that the least number of non-zero elements are moved into the k -th row and column. Similarly, strategy a) minimizes the sum of the norms of the k -th row and column, i.e. the permutation attempts to select the most diagonally dominant rows and columns. Approaches c) and d) are weighted alternatives to b). Note that the number of non-zero elements in a row or column of A again does not actually need to be calculated and stored in $\operatorname{nz_r}$ and $\operatorname{nz_c}$. Instead, these quantities can be determined easily using the compressed sparse row and column formats of A .

Finally, we do need to point out that in order to access A by arbitrary rows and columns at any time as is needed to update w , we require that A be available both in compressed row and column format. Generally, this should not be a problem, because memory needs to be available for the incomplete factorization anyhow. Hence, it should be possible to allocate enough memory to store A in both formats, to calculate the permutation, to free the memory for one format, and to subsequently calculate the incomplete factorization. Needless to say, this also requires more calculation time. However, both the memory requirements and the additional calculation time are not unreasonable or more than what other symmetric permutations require. Recall that most of these (e.g. reverse Cuthill-McKee, multilevel nested dissection, etc.) require the graph of $A + A^T$ for their implementation, so that the additional memory and setup times are comparable.

Algorithm 3: selection for diagonally dominant initial block

1. Initialize w as a binary tree having n elements, all set to 0.
2. Initialize nr and nc as zero vectors.
3. **for** $k = 1, \dots, n$
4. $k^* = \operatorname{argmin}(w)$
5. **if for all** $i = 1, \dots, k - 1$
6. $nr(i) + |a(P(i), k^*)| < 2$ **and** $nc(i) + |a(k^*, P(i))| < 2$
7. **then** $P(k) = k^*$
8. **else** mark k^* as unsuitable
9. **end if**
10. remove $w(k^*)$ from w
11. **for** $i = 1, \dots, n$ and $w(i)$ not yet removed
12. $w(i) = w(i) + |a(i, k^*)| + |a(k^*, i)|$
13. **end for** i
14. **end for** k
15. Initialize w for the rejected indices
16. Complete P as in algorithm 2a)

Algorithm 2 combined with a) selects rows and columns so as to improve diagonal dominance, but there is no guarantee that diagonal dominance is actually achieved for a fairly large submatrix. Although obtaining a large diagonally dominant block is not always possible, it is possible to modify algorithm 2, such that a larger diagonally dominant block becomes more likely by rejecting any index that would destroy diagonal dominance. In other words, if we assume that the submatrix at step k associated with I_k is diagonally dominant, then we should only select $P(k) = \operatorname{argmin}(w)$ if next submatrix will also be diagonally dominant. If it is not, we should reject $\operatorname{argmin}(w)$ and mark this index as being unsuitable. We continue selecting the index of the smallest element of the remaining elements of w to be a candidate for the next row and column until we find one preserving diagonal dominance. We continue in this manner until w is empty, i.e. until no more suitable indices are left. These rejected indices should then be sorted using algorithm 2 with weighing strategy a) and without the check for diagonal dominance. To implement this strategy efficiently, we simply need two additional arrays to keep track of the norms of both the rows and columns of the submatrix. Selecting a particular index will preserve diagonal dominance if and only if norms of the rows and columns of the updated matrix

would all less than 2, but this is easy to check. In this manner, we are guaranteed to obtain a diagonally dominant initial submatrix. Algorithm 3 summarizes this approach. Before continuing, it is important to point out that all the weights in algorithm 1 can be calculated in $\mathcal{O}(\text{nnz})$ operations, nnz being the number of non-zero elements of A , but strategies a)-d) require both the norms of rows and columns and hence approximately twice as many operations as sPQ). Additionally, a vector with n elements must be sorted, requiring $\mathcal{O}(n \log n)$ operations if quicksort is used. On the other hand, the main work for algorithm 2 is updating the binary tree, which essentially requires nnz updating operations, each consisting of removing, modifying, and reinserting a weight into a binary tree of n elements, amounting to approximately $\mathcal{O}(\text{nnz} \log n)$ operations. The actual number of arithmetic operations depends on the strategy chosen, but is again $\mathcal{O}(\text{nnz})$, with the more involved strategies requiring about twice as many operations as the simpler ones.

3 Numerical Results

In this section, we will report on numerical results for the different algorithms and different strategies for calculating the weights. All matrices tested were initially preprocessed to become I-matrices. We give results for no further preprocessing and for further preprocessing using multilevel nested dissection, the simple greedy PQ-strategy (algorithm 3.2 of [16]) and for all algorithms listed in the previous section. Recall that simple greedy PQ provides an index, upto which diagonal dominance was improved. For this reordering, we give results both for terminating a level based on this index and based on the absolute value of the pivot, analogous to the implementation of the other reorderings.

For testing the different permutations for a particular matrix, we first created an artificial right hand side, such that the exact solution was a vector consisting of all 1's. Next, the matrix was permuted and scaled to become an I-matrix using a subroutine of PARDISO, see [17], [18], [19] and [20]. Next, the rows and columns of the I-matrix were permuted using the permutation to be tested. For comparisons, we also used the multilevel nested dissection routine in METIS, see [8]. Next, we calculated a multilevel ILUC preconditioner. A particular level was terminated whenever the absolute value of the pivot was less than 0.01. In this case, the approximate Schur complement was calculated and it was treated as we just described for the initial matrix. We used the weighted dropping strategy described in [10] and [12] aimed at reducing the propagation of errors and the errors in L and U to ensure sparsity and varied the threshold parameter to obtain preconditioners having approximately the same fill-in. The Schur complement was calculated without additional dropping for the smaller matrices. For larger matrices, the dropping parameter τ was decreased by 3 orders of magnitude for calculating the Schur complement. We selected the zero vector as the initial guess for the solution and iterated with BiCGstab. If the initial residual was reduced by 8 orders of magnitude and the final residual was less than 10^{-8} in no more than 600 iterations, we considered the iterative method with that preconditioner to converge and denoted a failure otherwise. The code was programmed in C++ and compiled with the option -O3. The binary trees that are needed for some of the algorithms were implemented using the class "multimap" from the Standard Template Library. The calculations were performed on a 3.0 GHz Athlon XP computer using Linux.

We tested the same 58 matrices from the Harwell-Boeing Collection as in [16] as made available by Matrix Market [13] as well as 28 mostly larger matrices from the University of Florida Collection [5]. Details on these matrices can be found in these collections. For a number of matrices, particularly those that are not too difficult to precondition, the results were often fairly similar. Hence, we selected 17 matrices of the 58 from the Harwell-Boeing Collection which better illustrate the effects of the different approaches. Generally, we try to report on preconditioners having similar fill-in, but this is not always possible, as even for the same matrix, some strategies required significantly more fill-in for convergence than others. Furthermore, varying τ to produce a preconditioner having a specified amount of fill-in, was not always possible in some cases, as fill-in sometimes does not appear to depend continuously on τ . This is especially the case if varying τ also changes the number of levels. In any of these cases, the values for fill-in are different, but in most cases, meaningful comparisons are still possible. These results can be found in table 1. Although we also tested multilevel nested dissection combined with the algorithms presented in this paper, these results were very similar to those without the multilevel nested dissection, so we will not report on them. This is not surprising, as the algorithms presented in this article are likely to reorder a matrix significantly, destroying any structural advantage obtained by applying any symmetric permutation.

The first observation of these results indicates that much can be gained by further preprocessing I-matrices. Multilevel nested dissection as implemented by METIS does yield better results for most matrices. Particularly, whenever small pivots are not an issue, it seems that METIS is quite successful. However, in a number of cases, where near zero-pivots occurred frequently, METIS required a large number of levels and at times much fill-in. Considering that preprocessing is performed for each level change, this is an expensive option. Most methods presented in this paper appear to perform better overall.

matrix		Reordering applied to I-matrix													
		\emptyset	M	PQ*	PQ	sPQ	1a	1b	1c	1d	2a	2b	2c	2d	3
bp_800	γ	0.7	0.5	0.7	0.7	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
	L	3	1	5	3	3	2	3	3	2	2	3	3	2	
	I	51	75	28	28	24	24	97	36	29	40	58	59	46	
bp_1600	γ	1.0	1.0	1.0	1.1	1.0	1.0	1.0	0.9	1.0	1.0	1.0	1.0	1.0	
	L	3	3	4	3	2	2	4	3	3	2	2	2	2	
	I	47	10	10	19	13	11	20	10	9	10	17	12	10	
fs_760_2	γ	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	
	L	2	5	3	2	2	8	2	2	2	2	2	2	2	
	I	163	85	10	10	10	13	12	12	12	16	23	15	12	
fs_760_3	γ	3.5	2.2	2.2	2.0	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.0	2.0	
	L	3	14	4	3	16	18	2	14	12	2	2	3	3	
	I	511	234	110	195	67	85	87	51	46	36	68	41	50	
gemat11	γ	1.5	1.8	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	
	L	16	14	3	7	8	14	5	7	9	3	6	4	3	
	I	387	153	25	35	30	87	44	38	43	222	39	172	88	
gemat12	γ	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	
	L	20	17	6	10	9	11	7	7	8	3	4	3	3	
	I	212	41	46	69	29	36	25	15	16	43	24	39	47	

Table 1: Specific Numerical Results for Smaller Matrices (continued on next page).

matrix	Reordering applied to I-matrix														
	\emptyset	M	PQ*	PQ	sPQ	1a	1b	1c	1d	2a	2b	2c	2d	3	
lms_511	γ	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
	<i>L</i>	2	2	6	2	2	1	2	1	2	1	2	1	2	
	<i>I</i>	69	27	115	86	26	24	26	21	24	43	72	40	32	39
lms_3937	γ	2.0	2.1	NC	2.0	2.0	2.1	1.8	2.5	2.4	2.3	2.1	2.3	2.2	2.1
	<i>L</i>	2	2	NC	7	2	2	2	2	3	1	4	1	1	3
	<i>I</i>	469	34	NC	463	45	71	247	125	135	329	49	466	310	48
nnc666	γ	5.0	2.9	5.3	4.8	4.6	2.9	2.9	3.8	2.9	2.7	2.7	2.9	3.0	2.8
	<i>L</i>	11	11	5	3	10	10	9	10	9	3	3	3	2	2
	<i>I</i>	190	86	56	83	71	62	45	109	72	28	51	42	54	41
nnc1374	γ	13	5.9	NC	11	28	8.0	6.1	6.1	6.2	8.0	5.1	5.1	5.1	6.1
	<i>L</i>	52	30	NC	11	47	18	21	26	26	6	8	4	3	8
	<i>I</i>	68	12	NC	26	528	21	15	156	48	6	10	21	4	28
psmigr_2	γ	16	9.2	8.6	8.5	9.0	6.9	11	9.3	9.0	6.9	9.0	9.0	9.0	6.7
	<i>L</i>	23	16	6	7	6	11	10	7	14	9	18	13	10	8
	<i>I</i>	215	171	147	109	269	101	37	166	48	22	25	11	16	28
saylr4	γ	NE	5.2	0.7	2.6	2.6	1.4	4.9	3.3	3.3	0.7	0.7	0.7	0.7	0.7
	<i>L</i>	NE	65	2	31	33	73	67	48	48	3	4	3	3	3
	<i>I</i>	NE	366	46	48	49	50	48	49	49	46	49	48	47	47
watt_2	γ	1.0	1.0	1.0	0.9	0.9	0.9	1.0	1.0	1.0	1.0	0.8	1.0	1.0	1.0
	<i>L</i>	1	1	1	3	1	1	1	1	1	1	1	1	1	1
	<i>I</i>	48	29	21	83	54	64	65	55	53	54	62	56	51	65
west0655	γ	0.9	0.8	1.2	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	<i>L</i>	1	4	4	7	2	2	4	2	2	2	3	2	2	2
	<i>I</i>	349	205	386	155	48	43	39	32	41	33	35	91	66	33
west0989	γ	0.9	0.5	1.1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	<i>L</i>	5	3	5	5	1	2	3	1	1	1	3	2	2	2
	<i>I</i>	191	172	339	82	40	46	54	37	36	60	61	53	60	56
west1505	γ	1.4	0.7	1.3	0.7	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	<i>L</i>	4	3	5	4	3	3	3	2	2	2	2	3	2	2
	<i>I</i>	107	44	59	40	163	29	30	54	24	36	79	35	33	35
west2021	γ	1.1	0.7	1.0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	<i>L</i>	3	4	4	5	1	1	3	1	2	2	3	3	3	3
	<i>I</i>	88	72	29	21	20	23	69	42	27	60	37	36	27	40

Abbreviations used:

\emptyset	no further preprocessing used
M	multilevel nested dissection, level termination by pivot size
PQ*	standard PQ, level termination by PQ, not pivot size
PQ	standard PQ, level termination by pivot size
sPQ	symmetrized PQ, level termination by pivot size
γ	fill-in
<i>L</i>	number of levels used
<i>I</i>	number of iterations needed for convergence
NC	none of the preconditioners converged
NE	preconditioner does not exist

Table 1: Specific Numerical Results for Smaller Matrices.

Comparing the results for the greedy PQ reordering, we notice that terminating a level when the reordering indicates (denoted by PQ*) rather than whenever the absolute value of a pivot becomes too small (denoted by PQ) seems to be worse. Indeed, the latter strategy yielded suitable preconditioners for all matrices, whereas the former approach failed for two matrices entirely. However, either termination criterion was better for some matrices in terms of the number of iterations required for convergence than the other. Finally, enforcing symmetry for PQ reorderings generally seems to be a good idea yielding better results, except for nnc1374 and west1505 (and psimgr_2 to a lesser degree).

Unfortunately, there is no best strategy overall, but it seems that algorithm 3 is most robust. For many matrices, it is amongst the best strategies and for the others it seldomly amongst the worst, except perhaps for gemat11. However, it is the most involved strategy requiring the most memory (although not unreasonably much) and the most work. For a number of matrices, algorithm 1 requires significantly more levels than the others, making it fairly expensive and probably unsuitable for many larger matrices. In this regard, algorithm 2 is often a good compromise between algorithm 1 and 3.

It seems that no particular method of calculating the weights is universally best. Strategy c) for both algorithm 1 and 2 does not appear to work particularly well. Usually, strategy d) is significantly better or only slightly worse. Although for some matrices, using a) or b) to calculate weights produces the best results, their behavior is somewhat unpredictable, making d) probably the best choice overall. This result can be expected to some degree, as it can be viewed as the symmetrization of the weights used in sPQ), which is the weighing strategy suggested in [16].

Next, we will look at the results for some of the larger matrices from the University of Florida Collection. Except for the results for I-matrix preprocessing (without further permutation) or for I-matrix and METIS preprocessing, we provide the same results as for the small matrices. Both of these approaches yielded extremely poor results for a large number of matrices. Often the preconditioning failed because the calculations required more than 100 levels, which was the maximum permitted. In other cases, the preconditioner failed to converge or required an extremely large number of iterations. Hence, often the calculation times were formidable, so that for many matrices, it was not feasible to vary the threshold parameter sufficiently, making meaningful comparisons impossible. Hence, only the results for the other preprocessing methods can be found in table 2.

These results seem to confirm the observations already made. Much can be gained by using the absolute value of the pivot to terminate a level. The advantage of enforcing symmetry on the PQ reordering is, however, not as clear. For a few matrices, a symmetric PQ reordering improves the preconditioner further, but there are notable exceptions. Furthermore, for one matrix, scircuit, symmetric PQ fails. Nevertheless, for 6 out of 12 matrices (goodwin, graham1, igbt3, nmos, onetone1 and utm5940), the results for either of the nonsymmetrized PQ methods were significantly worse than any other method, whereas symmetrized PQ performs quite well for most of these matrices. Hence, symmetrized PQ is not necessarily competitive, but certainly neither of the non-symmetrized PQ reorderings can be considered attractive alternatives. By comparison, a number of the new algorithms work well for all matrices with a reasonable amount of fill-in and with a reasonable number of iterations. Of these, algorithm 3 again appears to be best overall.

matrix		Reordering applied to I-matrix											
		PQ*	PQ	sPQ	1a	1b	1c	1d	2a	2b	2c	2d	3
bayer01	γ	NC	1.1	1.8	1.8	1.8	1.9	2.0	1.2	1.9	1.1	1.1	1.1
	L	NC	8	1	1	1	1	1	1	1	1	1	1
	I	NC	9	38	20	20	166	181	11	35	10	18	10
	T_s	NC	0.8	2.3	2.1	2.5	2.3	2.5	2.3	2.4	2.3	2.3	2.2
	T_i	NC	1.0	4.5	2.0	2.4	19	22	1.1	4.0	1.0	1.8	1.0
bayer10	γ	NC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	L	NC	7	2	3	3	3	3	2	2	2	3	2
	I	NC	24	30	15	31	24	18	18	22	19	25	21
	T_s	NC	0.3	0.4	0.4	0.3	0.3	0.3	0.5	0.4	0.4	0.5	0.5
	T_i	NC	0.5	0.5	0.3	0.6	0.5	0.4	0.3	0.4	0.3	0.5	0.4
epb3	γ	2.9	2.9	2.9	2.9	2.8	3.0	3.0	2.9	3.0	2.9	3.0	2.9
	L	3	3	1	1	1	1	1	1	1	1	1	1
	I	124	128	127	65	103	87	93	52	90	57	56	47
	T_s	3.9	3.4	3.1	3.1	3.4	3.2	3.4	3.8	3.7	3.8	3.8	3.9
	T_i	29	30	30	15	24	21	22	12	22	14	14	11
goodwin	γ	NC	3.4	3.3	2.8	3.3	2.8	2.8	3.3	3.2	3.3	4.0	3.1
	L	NC	13	9	3	25	8	4	5	17	3	7	6
	I	NC	145	58	79	38	67	75	104	51	33	61	80
	T_s	NC	5.0	4.1	4.7	8.2	3.6	3.0	8.1	9.5	4.3	6.5	3.9
	T_i	NC	14	6.6	8.0	4.6	6.5	7.7	12	5.6	3.9	8.3	8.7
graham1	γ	NC	5.2	5.3	5.3	4.3	3.7	4.9	3.6	3.7	3.5	3.1	3.2
	L	NC	10	9	5	18	16	8	6	12	5	7	8
	I	NC	168	47	11	49	23	19	15	51	22	33	21
	T_s	NC	14	11	13	15	8.9	11	6.9	12	5.8	6.0	5.6
	T_i	NC	24	8.8	2.1	7.1	3.4	5.5	2.0	7.0	2.9	2.7	2.5
igbt3	γ	1.0	1.1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	L	6	5	2	4	2	3	3	2	2	2	2	2
	I	125	226	51	55	28	71	57	32	27	33	31	35
	T_s	0.5	0.5	0.4	0.4	0.4	0.5	0.4	0.6	0.7	0.6	0.6	0.7
	T_i	4.6	7.4	2.0	2.1	1.1	2.8	2.2	1.2	1.1	1.2	1.1	1.3
nmos	γ	1.7	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.9	1.0	1.0	0.9
	L	7	8	6	6	3	4	4	3	2	2	2	3
	I	204	142	46	41	40	53	61	37	40	35	34	36
	T_s	1.6	1.2	1.4	1.0	0.9	1.1	1.1	1.3	1.2	1.3	1.3	1.5
	T_i	18	9.9	5.3	2.6	2.8	3.6	4.2	2.2	2.4	2.2	2.2	2.1
onetone1	γ	NC	5.0	5.0	3.7	3.2	3.2	3.1	3.3	3.3	3.5	3.7	3.9
	L	NC	30	32	10	33	14	19	6	23	8	9	6
	I	NC	12	12	12	11	7	10	24	19	29	25	28
	T_s	NC	11	12	4.6	12	5.1	6.5	4.7	15	7.5	8.1	6.2
	T_i	NC	3.0	3.1	2.1	1.8	1.3	1.7	3.7	3.2	5.1	4.5	4.9
onetone2	γ	NC	3.5	3.4	NC	2.7	2.7	2.5	NC	3.2	3.3	3.6	3.6
	L	NC	19	15	NC	31	19	16	NC	23	6	8	7
	I	NC	20	30	NC	11	21	19	NC	40	21	24	26
	T_s	NC	2.8	3.1	NC	3.8	2.7	2.8	NC	6.1	3.4	3.9	3.6
	T_i	NC	2.5	4.4	NC	1.3	2.4	2.2	NC	5.2	2.6	3.2	3.3

Table 2: Specific Numerical Results for Larger Matrices (continued on next page).

matrix		Reordering applied to I-matrix											
		PQ*	PQ	sPQ	1a	1b	1c	1d	2a	2b	2c	2d	3
raefsky3	γ	1.9	1.9	1.9	1.9	1.6	1.9	1.9	2.1	2.1	1.6	1.6	2.1
	L	3	1	2	1	1	1	1	1	1	1	1	2
	I	43	35	41	16	11	16	16	20	12	33	34	19
	T_s	8.8	6.3	6.3	6.6	3.8	6.7	6.7	9.4	8.9	7.7	7.8	10
	T_i	14	11	13	5.0	3.1	5.1	5.3	6.7	4.2	9.2	9.6	6.5
scircuit	γ	2.5	2.5	NC	NC	2.5	NC	2.2	1.9	1.9	1.7	1.8	1.8
	L	3	73	NC	NC	62	NC	79	5	21	7	8	6
	I	5	5	NC	NC	6	NC	263	8	6	8	8	8
	T_s	10	15	NC	NC	17	NC	11	9.5	12	11	11	9.6
	T_i	1.0	3.7	NC	NC	6.6	NC	305	3.8	3.6	3.9	4.0	3.6
utm5940	γ	13	6.9	2.0	2.2	2.7	2.0	2.1	2.3	2.5	2.1	2.2	2.2
	L	6	5	3	4	4	3	3	4	2	4	3	3
	I	166	124	112	129	126	235	155	209	119	217	273	238
	T_s	4.0	2.1	0.3	0.4	0.6	0.3	0.3	0.5	0.3	0.5	0.5	0.6
	T_i	18	6.1	2.4	3.1	3.6	5.6	4.3	5.0	1.3	4.7	6.4	5.4

Abbreviations used:

PQ*	standard PQ, level termination by PQ, not pivot size
PQ	standard PQ, level termination by pivot size
sPQ	symmetrized PQ, level termination by pivot size
γ	fill-in
L	number of levels used
I	number of iterations needed for convergence
T_s	setup time for the preconditioner (in seconds)
T_i	iteration time (in seconds)
NC	none of the preconditioners converged
NE	preconditioner does not exist

Table 2: Specific Numerical Results for Larger Matrices.

Let us consider the results for specific matrices and compare the various PQ reorderings and algorithm 3: PQ (using the pivot size to terminate a level) is significantly better than all other approaches only for bayer01. Although symmetrized PQ is never the best approach, it is among the best methods for utm5940. On the other hand, algorithm 3 yields the best results for epb3, graham1, and scircuit. Furthermore, it is almost as good as the best preprocessing for igbt3 and nmos. Hence, algorithm 3 certainly seems to be the best method overall. Finally, if we look that the 5 remaining matrices, we observe that bayer10, the results for all methods are similar. Furthermore, for goodwin, symmetrized PQ is better than algorithm 3, but not the best method overall. For onetone2, PQ with pivot size terminating the level is better than algorithm 3, but again, not best. Finally, for onetone1 and raefsky3, algorithm 3 is better than either of the PQ methods. Hence, it seems that algorithm 3 is generally a better choice than any of the PQ-methods.

Overall, the simplest algorithm, algorithm 1, seems to be disappointing, although some variant of it is best for onetone1 (1c) and raefsky (1b). Algorithm 2 generally performs quite well with each variant sometimes being the best. However, in most cases, algorithm 3 is almost as good or better, so that the latter is usually the method of choice.

Finally, some testing was performed by varying the threshold for switching levels. Generally speaking, in most cases, varying between 10^{-4} and 10^{-1} had little effect, although

there are exceptions, hence we choose 10^{-2} as a standard. Usually, selecting 10^{-1} resulted in somewhat more levels, but this had little effect on the quality of the preconditioner or on calculation times, whereas selected the threshold closer to 10^{-4} seemed to have even less effect.

4 Conclusion

The three new algorithms presented here calculate row and column permutations for I-matrices making these more suitable for incomplete LU-factorizations. This approach reduces setup and iteration times substantially. Additionally, we obtain better robustness, resulting in convergence where other methods fail. This improvement is due to two factors: firstly, terminating a level based on pivot size is better than using suggestion for terminating a level which standard PQ reordering provides. Secondly, these reorderings are specifically designed for I-matrices. In particular, they permute rows and columns analogously to preserve the I-matrix structure, resulting in further improvement.

Combined with preprocessing to produce an I-matrix and an incomplete LU-factorization as preconditioner, these algorithms are a powerful tool for solving unsymmetric, indefinite linear systems with iterative methods. Generally, both fill-in and iterations required for convergence can be reduced significantly by using these strategies.

References

- [1] Michelle Benzi, John C. Haws, and Miroslav Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comp.*, 22(4):1333–1353, 2000.
- [2] Matthias Bollhöfer. A robust and efficient ILU that incorporates the growth of the inverse triangular factors. *SIAM J. Sci. Comp.*, 25(1):86–103, 2003.
- [3] Matthias Bollhöfer and Yousef Saad. Multilevel preconditioners constructed from inverse-based ILUs. Preprint.
- [4] Edmond Chow, Na Li, and Yousef Saad. Crout versions of ILU for sparse matrices. *SIAM J. Sci. Comput.*, 25(2):715–728, 2003.
- [5] Tim Davis. University of Florida Sparse Matrix Collection. <http://www.cise.ufl.edu/research/sparse/matrices>, <ftp://ftp.cise.ufl.edu/pub/faculty/davis/matrices>, Sept. 2006.
- [6] Iain S. Duff and Jacko Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20(1):889–901, 1999.
- [7] Iain S. Duff and Jacko Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996, 2001.
- [8] George Karypis. METIS. <http://glaros.dtc.umn.edu/gkhome/views/metis/>, Sept. 2006.

- [9] George Kayrpi and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comp.*, 20(1):359–392, 1998.
- [10] Jan Mayer. A multilevel crout ILU preconditioner with pivoting and row permutation. Preprint.
- [11] Jan Mayer. ILUCP: a Crout ILU preconditioner with pivoting. *Numer. Linear Algebra Appl.*, 12(9):941–955, 2005.
- [12] Jan Mayer. Alternative weighted dropping strategies for ILUTP. *SIAM J. Sci. Comput.*, 27(4):1424–1437, 2006.
- [13] National Institute of Standards. Matrix Market. <http://math.nist.gov/MatrixMarket/>, Sept. 2006.
- [14] Markus Olschowska and Arnold Neumaier. A new pivoting strategy for Gaussian elimination. *Lin. Alg. Appl.*, 220:131–151, 1996.
- [15] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2003.
- [16] Yousef Saad. Multilevel ILU with reorderings for diagonal dominance. *SIAM J. Sci. Comput.*, 27:1032–1057, 2006.
- [17] Olaf Schenk and Klaus Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. Technical report, Department of Computer Science, University of Basel, 2004.
- [18] Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004.
- [19] Olaf Schenk and Klaus Gärtner. PARDISO. <http://www.computational.unibas.ch/cs/scicomp/software/pardiso/>, Sept. 2006.
- [20] Olaf Schenk, Klaus Gärtner, and Wolfgang Fichtner. Efficient sparse LU factorization with left-looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.

IWRMM-Preprints seit 2004

- Nr. 04/01 Andreas Rieder: Inexact Newton Regularization Using Conjugate Gradients as Inner Iteration Michael
- Nr. 04/02 Jan Mayer: The ILUCP preconditioner
- Nr. 04/03 Andreas Rieder: Runge-Kutta Integrators Yield Optimal Regularization Schemes
- Nr. 04/04 Vincent Heuveline: Adaptive Finite Elements for the Steady Free Fall of a Body in a Newtonian Fluid
- Nr. 05/01 Götz Alefeld, Zhengyu Wang: Verification of Solutions for Almost Linear Complementarity Problems
- Nr. 05/02 Vincent Heuveline, Friedhelm Schieweck: Constrained H^1 -interpolation on quadrilateral and hexahedral meshes with hanging nodes
- Nr. 05/03 Michael Plum, Christian Wieners: Enclosures for variational inequalities
- Nr. 05/04 Jan Mayer: ILUCDP: A Crout ILU Preconditioner with Pivoting and Row Permutation
- Nr. 05/05 Reinhard Kirchner, Ulrich Kulisch: Hardware Support for Interval Arithmetic
- Nr. 05/06 Jan Mayer: ILUCDP: A Multilevel Crout ILU Preconditioner with Pivoting and Row Permutation
- Nr. 06/01 Willy Dörfler, Vincent Heuveline: Convergence of an adaptive hp finite element strategy in one dimension
- Nr. 06/02 Vincent Heuveline, Hoang Nam-Dung: On two Numerical Approaches for the Boundary Control Stabilization of Semi-linear Parabolic Systems: A Comparison
- Nr. 06/03 Andreas Rieder, Armin Lechleiter: Newton Regularizations for Impedance Tomography: A Numerical Study
- Nr. 06/04 Götz Alefeld, Xiaojun Chen: A Regularized Projection Method for Complementarity Problems with Non-Lipschitzian Functions
- Nr. 06/05 Ulrich Kulisch: Letters to the IEEE Computer Arithmetic Standards Revision Group
- Nr. 06/06 Frank Strauss, Vincent Heuveline, Ben Schweizer: Existence and approximation results for shape optimization problems in rotordynamics
- Nr. 06/07 Kai Sandfort, Joachim Ohser: Labeling of n -dimensional images with choosable adjacency of the pixels
- Nr. 06/08 Jan Mayer: Symmetric Permutations for I-matrices to Delay and Avoid Small Pivots During Factorization
- Nr. 06/09 Andreas Rieder, Arne Schneck: Optimality of the fully discrete filtered Backprojection Algorithm for Tomographic Inversion
- Nr. 06/10 Patrizio Neff, Krzysztof Chelminski, Wolfgang Müller, Christian Wieners: A numerical solution method for an infinitesimal elasto-plastic Cosserat model
- Nr. 06/11 Christian Wieners: Nonlinear solution methods for infinitesimal perfect plasticity

Eine aktuelle Liste aller IWRMM-Preprints finden Sie auf:

www.mathematik.uni-karlsruhe.de/iwrmm/seite/preprints