SIAM J. MATRIX ANAL. APPL.
Vol. 28, No. 2, pp. 524–550

MIQR: A MULTILEVEL INCOMPLETE QR PRECONDITIONER FOR LARGE SPARSE LEAST-SQUARES PROBLEMS*

NA LI† AND YOUSEF SAAD†

Abstract. This paper describes a multilevel incomplete QR factorization for solving large sparse least-squares problems. The algorithm builds the factorization by exploiting structural orthogonality in general sparse matrices. At any given step, the algorithm finds an independent set of columns, i.e., a set of columns that have orthogonal patterns. The other columns are then block orthogonalized against columns of the independent set, and the process is repeated recursively for a certain number of levels on these remaining columns. The final level matrix is processed with a standard QR or incomplete QR factorization. Dropping strategies are employed throughout the levels in order to maintain a good level of sparsity. A few improvements to this basic scheme are explored. Among these is the relaxation of the requirement of independent sets of columns. Numerical tests are proposed which compare this scheme with the standard incomplete QR preconditioner, the robust incomplete factorization preconditioner, and the algebraic recursive multilevel solver (on normal equations).

Key words. multilevel incomplete QR factorization, CGLS, QR factorization, orthogonal factorization, incomplete QR, preconditioning, iterative methods, large least-squares problems, normal equations

AMS subject classifications. 65F10, 65F20, 65F50

DOI. 10.1137/050633032

1. Introduction.

This paper considers iterative solution methods for linear least-squares problems of the form

$$\min_x \|b - Ax\|_2, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is a large sparse matrix with full rank. Problems of this type arise in many scientific and engineering applications including data analysis, computational fluid dynamics, simulation, signal processing, and control problems, to name just a few. As engineers and scientists are benefiting from increased availability of data as well as computational resources, these problems are inevitably becoming harder to solve due to their size as well as their ill-conditioning. For example, the papers [4, 34] mention a problem of this type which arises from an animal breeding study with 60 million unknowns. In the very different area of three-dimensional computer graphics, one encounters certain least-squares problems which have complexity proportional to the number of geometry primitives [23], which, in desirable models, should include millions of polygons. Problems from such applications are usually very sparse and can be solved iteratively or by sparse orthogonal factorizations. Iterative solution methods may have an advantage over direct methods, depending on the underlying sparsity pattern.

However, if iterative methods are to be used, then preconditioning is essential. Although it is known that iterative solution algorithms are not effective without pre-

*Received by the editors June 3, 2005; accepted for publication (in revised form) by D. B. Szyld February 2, 2006; published electronically July 31, 2006. This work was supported by NSF grants ACI-0305120 and INT-0003274 and by the Minnesota Supercomputing Institute.
http://www.siam.org/journals/simax/28-2/63303.html
†Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455 (nli@cs.umn.edu, saad@cs.umn.edu).

conditioning, there has been little effort made in developing preconditioners for least-squares problems in recent years. This is in contrast with the solution of standard (square) linear systems, where enormous progress has been made in designing both general purpose preconditioners and specialized preconditioners that are tailored to specific applications. Part of the difficulty stems from the fact that many methods solve the system (1.1) by implicitly solving the normal equations

$$A^T Ax = A^T b, \tag{1.2}$$

whose solution is the same as that of (1.1). The condition number of the coefficient matrix of the normal equations system (1.2) is the square of that of the original matrix A. As a result, the normal equations will tend to be very ill-conditioned. In this situation preconditioning is critical for robustness. However, severe ill-conditioning of the matrix will also tend to make it difficult to obtain a good preconditioner.

Though it is possible to solve the least-squares problem (1.1) by solving normal equations (1.2), forming the system of normal equations explicitly and then solving it is not a recommended approach in general as this suffers from various numerical difficulties; see [6, 15] for details. For small dense problems, the best overall solution method is to use a good orthogonal factorization algorithm such as the Householder QR; see, e.g., [15]. If $A = QR$ is the "thin" QR factorization of A [15], then the solution of (1.1) can be obtained by solving $Rx = Q^T b$ for x. For a comprehensive survey of direct methods, see [6].

Alternatively, iterative methods such as LSQR [28] and SOR [27] have been advocated for solving least-squares problems of the type (1.1) when A is large. A well-known approach is one that is based on solving the normal equations by the conjugate gradient (CG) method. The resulting algorithm is sometimes termed CGNR [31] and sometimes CGLS [6]. The latter acronym is adopted here. This paper only considers CGLS as the accelerator and focuses on developing effective preconditioners. Since we refer to preconditioned CGLS throughout the paper, we now give a brief description of the algorithm, assuming that a preconditioner M for $A^T A$ is available. Recall that a preconditioner is a certain matrix M which approximates the original coefficient matrix (in this case $A^T A$) such that it is inexpensive to solve an arbitrary linear system $Mx = b$.

ALGORITHM 1.1. Left-preconditioned CGLS

1. Compute $r_0 := b - Ax_0$, $\widetilde{r}_0 := A^T r_0$, $z_0 := M^{-1}\widetilde{r}_0$, $p_0 := z_0$.
2. For $i = 0, \ldots,$ until convergence *Do:*
3. $\quad w_i := Ap_i$.
4. $\quad \alpha_i := (z_i, \widetilde{r}_i)/\|w_i\|_2^2$.
5. $\quad x_{i+1} := x_i + \alpha_i p_i$.
6. $\quad r_{i+1} := r_i - \alpha_i w_i$.
7. $\quad \widetilde{r}_{i+1} := A^T r_{i+1}$.
8. $\quad z_{i+1} := M^{-1}\widetilde{r}_{i+1}$.
9. $\quad \beta_i := (z_{i+1}, \widetilde{r}_{i+1})/(z_i, \widetilde{r}_i)$.
10. $\quad p_{i+1} := z_{i+1} + \beta_i p_i$.
11. *EndDo*

Many variants of the above algorithm exist. In particular, when M is available in the form of a product $M = LL^T$, where L is lower triangular, then the preconditioning operation can be split into two parts and a split-preconditioned CGLS option can be derived. A right-preconditioned option can be developed as well. We consider only the left-preconditioned variant in this paper.

Developing preconditioners for the normal equations, or for problem (1.1), can be approached in a number of ways. A naive approach would be to form the squared matrix A^TA and try to find an incomplete Cholesky factorization of this matrix. The fact that this matrix is symmetric positive definite does not make it easy to find a preconditioner for it. Indeed, most of the theory for preconditioning techniques relies on some form of diagonal dominance. In addition, forming the normal equations suffers from other disadvantages, some of which are the same as those mentioned above for the dense case; in particular there is some loss of information when forming A^TA [6]. Moreover, A^TA can be much denser than the original matrix. In fact one dense row of A will make the entire A^TA matrix dense.

Another approach, one that is taken here, is to try to compute an approximate orthogonal factorization of A. This approach is not new, as will be seen in section 2 which discusses related work. If $A \approx QR$, then $A^TA \approx R^TR$ and this matrix can be used as a preconditioner M. Notice that this approach ignores the factor Q which is not used. In this paper we exploit multilevel ideas similar to those defined for the algebraic recursive multilevel solver (ARMS) in [33, 25]. The idea of multilevel incomplete QR (MIQR) factorization can be easily described with the help of recursion. It is important to observe at the outset that when A is sparse, then many of its columns will be orthogonal because of their structure. These are called structurally orthogonal columns. It is therefore possible to find a large set S of *structurally* orthogonal columns. This set is called an *independent set* of columns. Independent sets are the main ingredient used in ARMS [33, 25]. Once the first independent set S is obtained, we can block orthogonalize the remaining columns against the columns in S. Since the matrix of the remaining columns will still be sparse in general, it is natural to think of recursively repeating the process until a small number of columns are left which can be orthogonalized with standard methods. The end result is a QR factorization of a column-permuted A. With this simple strategy MIQR gradually reduces a large sparse least-squares system into one with a significantly smaller size. It is worth pointing out that although we focus on overdetermined systems ($m > n$), the techniques described are applicable to square matrices ($m = n$) and underdetermined matrices ($m < n$) as well.

Recent developments in the solution of standard linear systems have shown that multilevel preconditioners have excellent scalability and robustness properties; see, e.g., [33, 9, 31, 1, 2, 3]. However, it appears that when it comes to the solution of large general sparse least-squares problems, similar multilevel methods have not been considered so far, in spite of an increasing demand for solving such problems.

The remainder of this paper is organized as follows. After a short section on related work (section 2), we discuss in section 3 the issue of finding independent sets of columns, as this is an important ingredient used in MIQR. Then, a detailed description of MIQR is presented in section 4 followed by strategies to improve the performance of MIQR as well as other implementation details. Numerical results are reported in section 5, and the paper ends with concluding remarks in section 6.

2. Related work. Several general-purpose preconditioners based on techniques such as SSOR, incomplete orthogonal factorization, and incomplete Cholesky factorization have been proposed and analyzed in the literature.

In 1979, Björck introduced a preconditioner based on the SSOR method [5]. In the proposed method, A^TA is written as $A^TA = L + D + L^T$, where L is lower triangular. The normal equations are then preconditioned by

$$M = \omega(2-\omega)(D+\omega L)D(D+\omega L^T).$$

To avoid forming A^TA explicitly, row (or column) projection methods have also been exploited and applied to normal equations [10]. In these methods, only a row or a column of A is needed at any given relaxation step. Block versions of these methods have also been studied [7, 20].

In a 1984 pioneering article, Jennings and Ajiz proposed preconditioners based on incomplete versions of Givens rotations and the Gram–Schmidt process [18]. Since then, several other preconditioners based on incomplete orthogonal factorizations have been studied [30, 35, 29]. If $A = QR$ is the exact thin QR factorization of A, where R is an $n \times n$ upper triangular matrix and Q is an $m \times n$ orthogonal matrix, then $A^TA = R^TR$, and it is usually inexpensive to solve the equation $R^TRx = y$. The incomplete version of the QR factorization (IQR) can be used as a preconditioner for (1.2). Unlike the matrix Q produced by incomplete Givens rotations, which is always orthogonal, the factor Q produced by the incomplete Gram–Schmidt process is not necessarily orthogonal. Nonetheless, the incomplete Gram–Schmidt-based preconditioners are robust and can avoid breakdown when A has full rank. In this approach, dropping strategies can be employed in Q as well as R to reduce intermediate storage requirements. Let P_Q and P_R be zero patterns chosen for matrices Q and R, respectively. The incomplete QR factorization based on the Gram–Schmidt process can be described by the following modification of the incomplete LQ (ILQ) algorithm given in [30]. Note that in practice P_Q and P_R are normally determined dynamically based on the magnitude of the elements generated.

ALGORITHM 2.1. Incomplete QR factorization (IQR)

1. For $j = 1, \ldots, n$ *Do:*
2. Compute $r_{ij} := (a_j, q_i)$ for $i = 1, 2, \ldots, j-1$.
3. Replace r_{ij} by zero if $(i, j) \in P_R$.
4. Compute $q_j := a_j - \sum_{i=1}^{j-1} r_{ij} q_i$.
5. Replace q_{ij} by zero if $(i, j) \in P_Q$, $i = 1, 2, \ldots, m$.
6. Compute $r_{jj} := \|q_j\|_2$.
7. If $r_{jj} == 0$, then stop; Else compute $q_j := q_j / r_{jj}$.
8. *EndDo*

In the above algorithm, the step represented by line 2 computes the inner products of the jth column of A with all previous columns of Q. Most of these inner products are equal to zero because of sparsity. Therefore, it is important to ensure that only the nonzero inner products are calculated for efficiency. The strategy proposed in [30] calculates these inner products as a linear combination of sparse vectors. Specifically, let $r_j = [r_{1j}, r_{2j}, \ldots, r_{j-1,j}]^T$ and $Q_{j-1} = [q_1, q_2, \ldots, q_{j-1}]$; then $r_j = Q_{j-1}^T a_j$ is a sparse matrix by sparse vector product. This product can be computed as a linear combination of the rows in Q_{j-1}; i.e., only the rows corresponding to the nonzero elements in a_j are linearly combined. Since the matrix Q is normally stored column-wise, a linked list pointing to the elements in each row of Q needs to be dynamically maintained. This strategy is also utilized in the implementation of the proposed MIQR algorithm.

Preconditioners based on the incomplete modified Gram–Schmidt process have also been developed. The Cholesky incomplete modified Gram–Schmidt (CIMGS) algorithm of Wang, Gallivan, and Bramley is an incomplete orthogonal factorization preconditioner based on the modified Gram–Schmidt process [35]. The paper explores rigorous strategies for defining incomplete Cholesky factorizations, based on the relation between the Cholesky factorization of A^TA and the QR factorization of A. Other authors studied direct ways to obtain the Cholesky factorization [19, 6]. This type

of approach obtains the incomplete Cholesky factorization of $C = A^T A$, where C may or may not be formed explicitly. As an alternative, Benzi and Tůma proposed a robust incomplete factorization (RIF) preconditioner which computes an incomplete LDL^T factorization of $A^T A$ without explicitly forming it [4]. Their approach utilizes a conjugate Gram–Schmidt process to calculate the factorization $Z^T C Z = D$, where Z is unit upper triangular and D is diagonal. Using the fact that $Z^T = L^{-1}$, they showed that $L_{ji} = (z_j^T C z_i)/(z_j^T C z_j)$. Therefore, the L factor of C can be obtained as a side product of the conjugate orthogonalization process without any extra cost. In section 5 a few comparisons are made between this approach and the MIQR technique proposed in this paper.

There were also a number of attempts to precondition positive definite matrices which may be far from diagonally dominant. In a 1980 paper, Manteuffel [26] suggested shifting a positive definite matrix to get an incomplete Cholesky factorization. This work was pursued more recently in [32], where other diagonal shifting techniques were studied for both incomplete orthogonal factorizations and incomplete Cholesky factorizations.

The idea of utilizing independent sets of columns (rows) in the context of least-squares, or more precisely for normal equations, is not new; see, e.g., [20, 21]. The main goal of these two papers was to exploit independent sets to improve parallelism. Independent sets of columns will be the main ingredient in obtaining an MIQR factorization. In terms of parallel algorithms, Elmroth and Gustavson developed recursive parallel QR factorizations that can be used in direct solvers for dense normal equations [12, 13].

3. Independent sets of columns. The MIQR algorithm proposed in this paper exploits successive independent sets of columns. This section discusses column independent set orderings.

Given a matrix $A = [a_1, a_2, \ldots, a_n]$, where $a_1, a_2, \ldots, a_n$ are column vectors, a subset $\{a_{j_1}, a_{j_2}, \ldots, a_{j_s}\}$ is called *an independent set of columns* of A if columns l and k of A are structurally orthogonal for any $l, k \in \{j_1, j_2, \ldots, j_s\}$ and $l \neq k$. Figure 3.1(a) shows an example of such an independent set of five columns (marked as open circles). The issue of finding independent sets of columns is not new and has been discussed in depth in the literature in different contexts; see, e.g., [11, 22, 24, 31] or [14] for a more comprehensive review. Here, we formalize the problem into that of finding an independent set in a graph.

3.1. Finding independent sets of columns. Two columns a_i and a_j of A will be said to be *adjacent* if their patterns overlap. This means that if $\hat{a}_k$ is the column vector obtained from a_k by replacing all its nonzero entries by ones, then a_i and a_j are adjacent if and only if $\hat{a}_i^T \hat{a}_j \neq 0$. The opposite of adjacent is *structurally orthogonal*: two columns a_i and a_j are structurally orthogonal if $\hat{a}_i^T \hat{a}_j = 0$.

Let $\hat{A}$ be the pattern matrix obtained from A by replacing all its nonzero entries by ones. Then, the *column intersection graph* (CIG) (see [11]) of A is the graph with n vertices representing the n columns of A, and with edges defined by the nonzero pattern of $\hat{A}^T \hat{A}$. This means that there is an edge between vertex i and j if and only if $\hat{a}_i$ and $\hat{a}_j$ are adjacent.

Note that an edge from vertex i to vertex j is defined if $\cos\theta_{ij} \neq 0$, where θ_{ij} is the angle between vectors $\hat{a}_i$ and $\hat{a}_j$. Define the following matrices:

$$B = \left[\frac{\hat{a}_1}{\|\hat{a}_1\|}, \frac{\hat{a}_2}{\|\hat{a}_2\|}, \ldots, \frac{\hat{a}_n}{\|\hat{a}_n\|} \right] \quad \text{and} \quad C = B^T B. \tag{3.1}$$

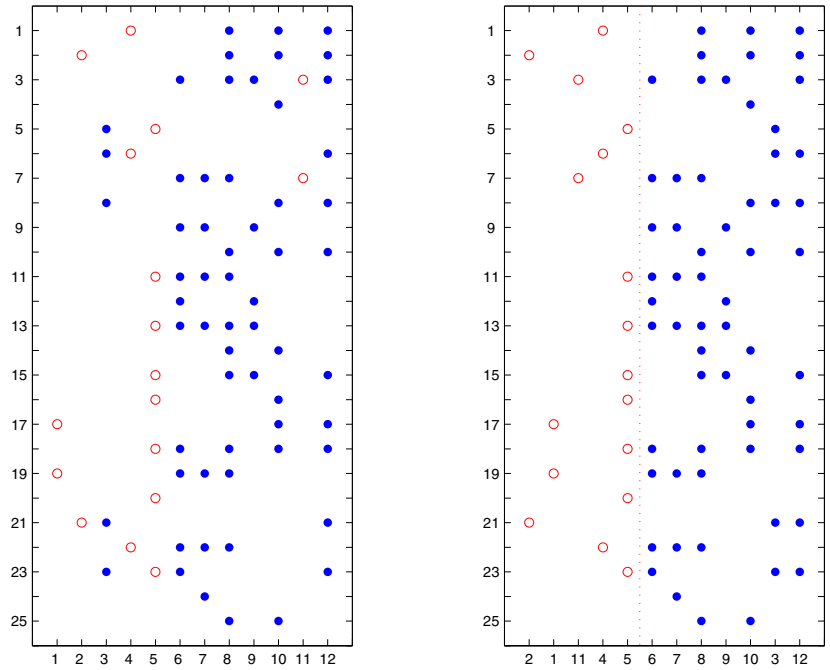

FIG. 3.1. (a) *An independent set of five columns (open circles) in a* 25×12 *matrix.* (b) *The independent set of columns are permuted to the first five columns of the matrix.*

Since the generic entry c_{ij} of C can be written as $c_{ij} \equiv \cos\theta_{ij}$, the graph $CIG(A)$ is nothing but the adjacency graph of C. Therefore, the problem is to find a maximal independent set of a graph. Let E be the set of all edges in $CIG(A)$ and let V be the set of its vertices. Recall that an *independent set* S is a subset of the vertex set V such that

$$\text{if } x \in S, \text{ then } [(x,y) \in E \text{ or } (y,x) \in E] \Rightarrow y \notin S;$$

i.e., any vertex in S is not allowed to be adjacent with any other vertex in S either by incoming or outgoing edges. An independent set S is *maximal* if

$$S' \supseteq S \text{ is an independent set} \Rightarrow S' = S.$$

Note that the maximal independent set is not necessarily the independent set with maximum cardinality. In fact, to find the latter is NP hard. In the following, the term independent set will always mean a maximal independent set. The following greedy algorithm (see, e.g., [31]) can be used to find an independent set S. In the algorithm, U is the set of all unmarked vertices, which initially includes all the vertices.

ALGORITHM 3.1. Independent set ordering

1. Let $S := \phi$ and $U := \{1, 2, \ldots, n\}$. $j := 1$.
2. *Do* While $U \neq \phi$ and $j \leq maxSteps$:
3. Let $k :=$ next unmarked vertex in U.
4. $S := S \cup \{k\}$. Mark and remove k from U.
5. Mark all vertices adjacent to k and remove them from U.
6. $j := j + 1$.
7. *EndDo*

Let $|S|$ be the size of S. Assume that the maximum degree of all vertices in S is d_S. According to Algorithm 3.1, the total number of marked vertices is $n - |S|$. At the same time, whenever a vertex is added to S, at most d_S vertices will be marked, which means the total number of vertices marked is at most $d_S|S|$. Therefore, we have $n - |S| \leq d_S|S|$ and as a result

$$|S| \geq \frac{n}{1 + d_S}. \tag{3.2}$$

This suggests that we may obtain S with a larger number of vertices by first visiting the vertices with smaller degrees [31].

ALGORITHM 3.2. Independent set ordering with increasing degree traversal

1. Find an ordering $i_1, i_2, \ldots, i_n$ of the vertices by increasing degree.
2. Let $S := \phi$ and $U := \{i_1, i_2, \ldots, i_n\}$. $j := 1$.
3. While $U \neq \phi$ and $j \leq maxSteps$:
4. Let $i_k :=$ next unmarked vertex in U.
5. $S := S \cup \{i_k\}$. Mark and remove i_k from U.
6. Mark all i_k's adjacent vertices and remove them from U.
7. $j := j + 1$.
8. *EndDo*

Algorithm 3.2 first sorts the vertices in increasing degree order and then applies the greedy algorithm. In general, Algorithm 3.2 will find a larger independent set at the cost of an initial sorting of the vertices. Algorithm 3.2 is used in the implementation of MIQR.

3.2. Estimates for the size of the independent set. The lower bound of the independent set size given by (3.2) is a rough one. The goal of this section is to find a more accurate estimate of the size of the independent set using a simple probabilistic model.

Consider an $m \times n$ sparse matrix A with N_{nz} nonzero entries and assume that these nonzero entries are randomly distributed. In particular each column will have on average the same number of nonzero entries, which is $\nu \equiv N_{nz}/n$. Under this assumption, Algorithms 3.1 and 3.2 would be equivalent and therefore, we can restrict our study to Algorithm 3.1. We denote by μ the average number of nonzero entries per row; thus $\mu \equiv N_{nz}/m$.

For any column vector a of A, we first calculate the expected number of column vectors that are *not* structurally orthogonal to a. If a has only one nonzero element, then there are on average $n - 1 - (\mu - 1) = n - \mu$ possible columns among $n - 1$ that will be orthogonal to a; thus the probability that any given column is orthogonal to a is $(n - \mu)/(n - 1)$. Since a has ν nonzero elements on average, the probability that any given column is orthogonal to a is

$$p = \left(\frac{n - \mu}{n - 1}\right)^{\nu}. \tag{3.3}$$

As a result the probability that any given column is not orthogonal to a is $1 - p$. Thus, the expected number of column vectors that are *not* structurally orthogonal to a is

$$\eta = (n - 1) \times \left(1 - \left(\frac{n - \mu}{n - 1}\right)^{\nu}\right). \tag{3.4}$$

Note that η is simply the average degree of a node in $CIG(A)$ in the very first step of Algorithm 3.1, since it represents the average number of columns that are not orthogonal to a given column of A.

Consider now an arbitrary step j of Algorithm 3.1. We will call N_j the number of columns left to be considered, i.e., the number of unmarked columns in U at the end of step j of Algorithm 3.1.

LEMMA 3.1. *Let N_j be the expected number of unmarked columns at the end of the jth step of Algorithm* 3.1*, with $N_0 = n$. Then N_j satisfies the following recurrence relation:*

$$N_j = \left(\left(1 - \frac{\nu}{m}\right) \frac{N_{j-1}}{N_{j-1} - 1}\right)^{\nu} (N_{j-1} - 1). \tag{3.5}$$

Proof. We begin by observing that if we consider the matrix consisting of the unmarked columns of A at any given step, then its average number of nonzero entries per column remains unchanged and equal to ν. In contrast, the removal of one column will change μ. If μ_j is the average number of nonzero entries per row for the matrix of unmarked columns, then

$$\mu_j = \frac{\nu}{m} \times N_j. \tag{3.6}$$

Assume that the independent set obtained is $S = \{i_1, i_2, \ldots, i_s\}$, where i_1 is the first vertex added into S, i_2 is the second vertex added into S, and so on. When i_j is added, the estimated number of vertices that are *newly* marked in line 5 in Algorithm 3.1 is simply the expected number of columns that are not orthogonal to a given column for the matrix of unmarked columns. This is simply the expression (3.4) in the very first step, i.e., when $j = 1$. For a general step it will be the same expression with n replaced by N_j and μ by μ_j. Note that i_j itself is also marked. The new number of unmarked columns is therefore

$$\begin{aligned} N_j &= N_{j-1} - 1 - (N_{j-1} - 1) \times \left(1 - \left(\frac{N_{j-1} - \mu_{j-1}}{N_{j-1} - 1}\right)^{\nu}\right) \\ &= (N_{j-1} - 1) \left(\frac{N_{j-1} - \mu_{j-1}}{N_{j-1} - 1}\right)^{\nu}. \end{aligned}$$

We now introduce $n_j \equiv N_j - 1$ to simplify notation. The above equalities become

$$n_j = \left(\frac{n_{j-1} + 1 - \mu_{j-1}}{n_{j-1}}\right)^{\nu} n_{j-1} - 1 = \left(1 - \frac{\mu_{j-1} - 1}{n_{j-1}}\right)^{\nu} n_{j-1} - 1.$$

Substituting μ_{j-1} given by (3.6) gives

$$\begin{aligned} n_j &= \left(1 - \frac{\frac{(n_{j-1}+1)\nu}{m} - 1}{n_{j-1}}\right)^{\nu} n_{j-1} - 1 = \left(1 - \frac{\nu}{m} + \frac{1 - \frac{\nu}{m}}{n_{j-1}}\right)^{\nu} n_{j-1} - 1 \\ &= \left(\left(1 - \frac{\nu}{m}\right)\left(1 + \frac{1}{n_{j-1}}\right)\right)^{\nu} n_{j-1} - 1, \end{aligned}$$

which is the expression to be proved when $N_j = n_j + 1$ is substituted. ∎

The lemma should be interpreted in a probabilistic sense: Given a large number of matrices, with the same size (n and m) and the same ν, on average the number

of unmarked columns at the jth step is given by the number resulting from solving the recurrence equation (3.5). The actual number may, of course, be larger or smaller than the N_j given by the lemma.

It is important to note that (3.5) is an exact equality. However, it does not seem possible to obtain a simple closed form expression for N_j. One would be tempted to make the approximation $1/N_j \approx 0$ but this is not valid since toward the end N_j will become small. On the other hand, one can find rough bounds for N_j and substitute them above.

Thus, since $N_j \leq N$ for all j we have

$$N_j \geq \left[\left(1 - \frac{\nu}{m}\right)\frac{n}{n-1}\right]^{\nu}(N_{j-1} - 1).$$

Define

$$\alpha \equiv \left[\left(1 - \frac{\nu}{m}\right)\frac{n}{n-1}\right]^{\nu} \quad \text{and} \quad \gamma \equiv \frac{\alpha}{1-\alpha}.$$

Clearly, we have $\alpha \leq (n/(n-1))^{\nu}$. For large n and m, α will typically be smaller than 1 for a sparse matrix. For example, noting that $\alpha = [(1-\nu/m)(1+1/(n-1)]^{\nu}$, we have

$$\frac{1}{n-1} \leq \frac{\nu}{m} \quad \rightarrow \quad \alpha \leq \left(1 - \left(\frac{\nu}{m}\right)^2\right)^{\nu} < 1.$$

As can be easily seen, the assumption $1/(n-1) \leq \nu/m$ is equivalent to $\mu \geq 1 + \nu/m$, which is generally verified because $\nu \ll m$. We will make the assumption that $\alpha < 1$. This condition is equivalent to

$$\left(1 - \frac{\nu}{m}\right)\frac{n}{n-1} < 1 \quad \leftrightarrow \quad \left(1 - \frac{\nu}{m}\right) < 1 - \frac{1}{n} \quad \leftrightarrow \quad \frac{\nu n}{m} > 1 \quad \leftrightarrow \quad \mu > 1.$$

Then, we have $N_j \geq \alpha N_{j-1} - \alpha$ and since $-\alpha = \alpha\gamma - \gamma$ this becomes $(N_j + \gamma) \geq \alpha(N_{j-1} + \gamma)$. Using this it is possible to estimate the total number of steps which will result in the algorithm, which will be the size of S, since each step will add one more member to S. Let s be the last step of the algorithm and note that we have $N_s \leq N_{s-1} \leq \cdots \leq N_0 = n$. Then the above inequality will yield

$$(N_s + \gamma) \geq \alpha(N_{s-1} + \gamma) \geq \cdots \geq \alpha^s(N_0 + \gamma) \rightarrow (N_s + \gamma) \geq \alpha^s(N_0 + \gamma).$$

The step s at which the algorithm is stopped corresponds to a final size of $N_s = 1$. This means that

$$\alpha^s \leq \frac{1+\gamma}{n+\gamma} = \frac{1}{(1-\alpha)(n + \alpha/(1-\alpha))} = \frac{1}{(1-\alpha)n + \alpha}.$$

Taking logarithms and recalling that $\alpha < 1$ yields

$$s \geq s_{min} \equiv \frac{\log\left[\alpha + (1-\alpha)n\right]}{-\log\alpha}. \tag{3.7}$$

The accuracy of the estimates derived above will be tested in section 5. It will be verified in the experiments that the lower bound (3.7) is not sharp. In fact the experiments indicate, with good consistency, that it is better to use $2s_{min}$ as an estimate of the actual size of S. On the other hand, the estimate given by the direct application of the formula (3.5) can be quite accurate in spite of the simplicity of the underlying model.

4. Multilevel incomplete QR (MIQR) factorizations. This section presents the MIQR preconditioning method for solving sparse least-squares systems. It begins with a discussion of the complete version of the multilevel QR factorization (section 4.1). Then, strategies are proposed to approximate the factorization for preconditioning purposes (section 4.2).

4.1. Multilevel QR factorization (MQR). When the matrix A in (1.1) is sparse, it will most likely have an independent set of columns $a_{j_1}, a_{j_2}, \ldots, a_{j_s}$. Let P_1^T be the permutation matrix which permutes $a_{j_1}, a_{j_2}, \ldots, a_{j_s}$ into the first s columns. Then we have

$$AP_1^T = [A^{(1)}, A^{(2)}], \tag{4.1}$$

where $A^{(1)} = [a_{j_1}, a_{j_2}, \ldots, a_{j_s}]$ is an $m \times s$ matrix and $A^{(2)}$ is an $m \times (n-s)$ matrix. Figure 3.1(b) shows an example of such an ordering. Without loss of generality and for simplicity, we still use $[a_1, a_2, \ldots, a_s]$ and $[a_{s+1}, a_{s+2}, \ldots, a_n]$ to denote the columns of $A^{(1)}$ and $A^{(2)}$, respectively.

Since the columns in $A^{(1)}$ are orthogonal to each other, $(A^{(1)})^T A^{(1)}$ is a diagonal matrix. Then $A^{(1)}$ can be trivially factored as $A^{(1)} = Q_1 D_1$ with

$$Q_1 = \left[\frac{a_1}{\|a_1\|_2}, \frac{a_2}{\|a_2\|_2}, \ldots, \frac{a_s}{\|a_s\|_2} \right] \quad \text{and} \quad D_1 = \operatorname{diag}(\|a_1\|_2, \|a_2\|_2, \ldots, \|a_s\|_2).$$

Now let

$$\begin{aligned} F_1 &= Q_1^T A^{(2)}, \\ A_1 &= A^{(2)} - Q_1 F_1. \end{aligned}$$

Then (4.1) can be rewritten as

$$AP_1^T = [A^{(1)}, A^{(2)}] = [Q_1, A_1] \begin{bmatrix} D_1 & F_1 \\ 0 & I \end{bmatrix}. \tag{4.2}$$

This is a block version of the Gram–Schmidt process, and we have

$$Q_1^T A_1 = 0, \tag{4.3}$$

because $Q_1^T A_1 = Q_1^T (A^{(2)} - Q_1 F_1) = Q_1^T A^{(2)} - F_1 = 0$.

In the simplest one-level method, we apply a standard QR factorization to the reduced $m \times (n-s)$ system A_1:

$$A_1 \widetilde{P}_2^T = Q_2 \widetilde{R}_2,$$

where $\widetilde{P}_2^T$ is an $(n-s) \times (n-s)$ permutation matrix ($\widetilde{P}_2^T$ is the identity matrix when pivoting is not used), Q_2 is an $m \times (n-s)$ orthogonal matrix, and $\widetilde{R}_2$ is an $(n-s) \times (n-s)$ upper triangular matrix. Equation (4.2) can then be rewritten as

$$A = [Q_1, Q_2] \begin{bmatrix} I & 0 \\ 0 & \widetilde{R}_2 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \widetilde{P}_2 \end{bmatrix} \begin{bmatrix} D_1 & F_1 \\ 0 & I \end{bmatrix} P_1 \tag{4.4}$$

or

$$A = Q R_2 P_2 R_1 P_1 = Q \widehat{R} \tag{4.5}$$

if we use the following notations:

$$Q = [Q_1, Q_2], \quad R_1 = \begin{bmatrix} D_1 & F_1 \\ 0 & I \end{bmatrix}, \quad R_2 = \begin{bmatrix} I & 0 \\ 0 & \widetilde{R}_2 \end{bmatrix}, \quad P_2 = \begin{bmatrix} I & 0 \\ 0 & \widetilde{P}_2 \end{bmatrix},$$

$$\text{and } \widehat{R} = R_2 P_2 R_1 P_1.$$

If A has full rank, then $\widetilde{R}_2$ is nonsingular. It is easy to show that Q is orthogonal because

$$Q_1^T Q_2 = Q_1^T (A_1 \widetilde{P}_2^T \widetilde{R}_2^{-1}) = (Q_1^T A_1) \widetilde{P}_2^T \widetilde{R}_2^{-1} = 0.$$

As is the case in similar situations related to Gram–Schmidt with pivoting, the final result is equivalent to applying the standard Gram–Schmidt process to a matrix obtained from A by permuting its columns. Indeed, starting with (4.4), we have

$$\begin{aligned} AP_1^T &= [Q_1, Q_2] \begin{bmatrix} I & 0 \\ 0 & \widetilde{R}_2 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \widetilde{P}_2 \end{bmatrix} \begin{bmatrix} D_1 & F_1 \\ 0 & I \end{bmatrix} \\ &= [Q_1, Q_2] \begin{bmatrix} I & 0 \\ 0 & \widetilde{R}_2 \end{bmatrix} \begin{bmatrix} D_1 & F_1 \\ 0 & \widetilde{P}_2 \end{bmatrix} \\ &= [Q_1, Q_2] \begin{bmatrix} I & 0 \\ 0 & \widetilde{R}_2 \end{bmatrix} \begin{bmatrix} D_1 & F_1 \widetilde{P}_2^T \\ 0 & I \end{bmatrix} P_2, \end{aligned}$$

which yields the following QR factorization of a column-permuted A:

$$AP_1^T P_2^T = [Q_1, Q_2] \begin{bmatrix} D_1 & F_1 \widetilde{P}_2^T \\ 0 & \widetilde{R}_2 \end{bmatrix}. \tag{4.6}$$

Since A is sparse, F_1 and A_1 are usually sparse as well. Sparsity can also be improved by relaxing the orthogonality and applying dropping strategies in the incomplete version, as will be discussed in section 4.2. Moreover, because A_1 is likely to still be large, the above reduction process can be applied to A_1 recursively instead of obtaining its QR factorization with a standard algorithm. The recursion continues until the reduced matrix is small enough or the matrix cannot be further reduced.

Let $A_0 \equiv A$. Then, generally, the factorization at levels $i = 1, 2, \ldots, p$ can be recursively defined as follows:

$$A_{i-1} \widetilde{P}_i^T = [A_{i-1}^{(1)}, A_{i-1}^{(2)}] = [Q_i, A_i] \begin{bmatrix} D_i & F_i \\ 0 & I \end{bmatrix}, \tag{4.7}$$

where $A_{i-1}^{(1)}$ has s_i columns and, similarly to the one-level case, $\widetilde{P}_i^T$ is the column permutation which orders the set of independent columns first. Let

$$D_i = \text{diag}\left(\|A_{i-1}^{(1)} e_j\|_2 \right)_{j=1,\ldots,s_i}, \tag{4.8}$$

$$Q_i = A_{i-1}^{(1)} D_i^{-1}, \tag{4.9}$$

$$F_i = Q_i^T A_{i-1}^{(2)}, \tag{4.10}$$

$$A_i = A_{i-1}^{(2)} - Q_i F_i. \tag{4.11}$$

We will also define as before

$$P_i = \begin{bmatrix} I & 0 \\ 0 & \widetilde{P}_i \end{bmatrix},$$

where the identity block completes the matrix $\widetilde{P}_i$ into an $n \times n$ matrix.

The MQR algorithm can be simply defined as follows.

ALGORITHM 4.1. MQR

0. $A_0 \equiv A$.
1. For $i = 1, \ldots, p$ *Do:*
2. Compute permutation $\widetilde{P}_i$ and apply it to A_{i-1}: $A_{i-1}\widetilde{P}_i^T = [A_{i-1}^{(1)}, A_{i-1}^{(2)}]$.
3. Compute Q_i, D_i, $F_i = Q_i^T A_{i-1}^{(2)}$, and $A_i = A_{i-1}^{(2)} - Q_i F_i$.
4. *EndDo*
5. $A_p \widetilde{P}_{p+1}^T = Q_{p+1}\widetilde{R}_{p+1}$ (standard QR with/without pivoting).

We can now establish a result which generalizes the relation (4.6).

LEMMA 4.1. *At the ith step of the MQR procedure, the following relation holds:*

$$AP_1^T \ldots P_i^T = [Q_1, \ldots, Q_i \mid A_i] \begin{bmatrix} R_{11} & R_{12} \\ 0 & I \end{bmatrix}, \tag{4.12}$$

where R_{11} is a (square) upper triangular matrix with a size equal to the column size of $[Q_1, \ldots, Q_i]$.

Proof. The proof is by induction on i. We begin by pointing out that $\widetilde{P}_1 \equiv P_1$. Since $A_0 \equiv A$, (4.7) shows that the result is trivially true for $i = 1$. We now assume that (4.12) is true for i and will show that it is true for $i+1$. From (4.7) we can write

$$A_i = [Q_{i+1}, A_{i+1}] \begin{bmatrix} D_{i+1} & F_{i+1} \\ 0 & I \end{bmatrix} \widetilde{P}_{i+1}$$

which, when substituted in (4.12) yields

$$\begin{aligned}
&[Q_1, \ldots, Q_i, A_i] \begin{bmatrix} R_{11} & R_{12} \\ 0 & I \end{bmatrix} \\
&\quad = \left[Q_1, \ldots, Q_i, [Q_{i+1}, A_{i+1}] \begin{bmatrix} D_{i+1} & F_{i+1} \\ 0 & I \end{bmatrix} \widetilde{P}_{i+1} \right] \begin{bmatrix} R_{11} & R_{12} \\ 0 & I \end{bmatrix} \\
&\quad = [Q_1, \ldots, Q_i, [Q_{i+1}, A_{i+1}]] \begin{bmatrix} R_{11} & R_{12} \\ 0 & \begin{bmatrix} D_{i+1} & F_{i+1} \\ 0 & I \end{bmatrix} \widetilde{P}_{i+1} \end{bmatrix} \\
&\quad = [Q_1, \ldots, Q_i, Q_{i+1}, A_{i+1}] \begin{bmatrix} R_{11} & R_{12}\widetilde{P}_{i+1}^T \\ 0 & \begin{bmatrix} D_{i+1} & F_{i+1} \\ 0 & I \end{bmatrix} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0\widetilde{P}_{i+1} & \end{bmatrix}.
\end{aligned}$$

This shows that

$$AP_1^T \cdots P_i^T P_{i+1}^T = [Q_1, \ldots, Q_i, Q_{i+1} \mid A_{i+1}] \begin{bmatrix} R_{11} & R_{12}\widetilde{P}_{i+1}^T \\ 0 & \begin{bmatrix} D_{i+1} & F_{i+1} \\ 0 & I \end{bmatrix} \end{bmatrix},$$

which is the desired result for level $i+1$. ∎

If the procedure stops at the pth level, then A_p is the final reduced system and we factor it as

$$A_p \widetilde{P}_{p+1}^T = Q_{p+1}\widetilde{R}_{p+1}.$$

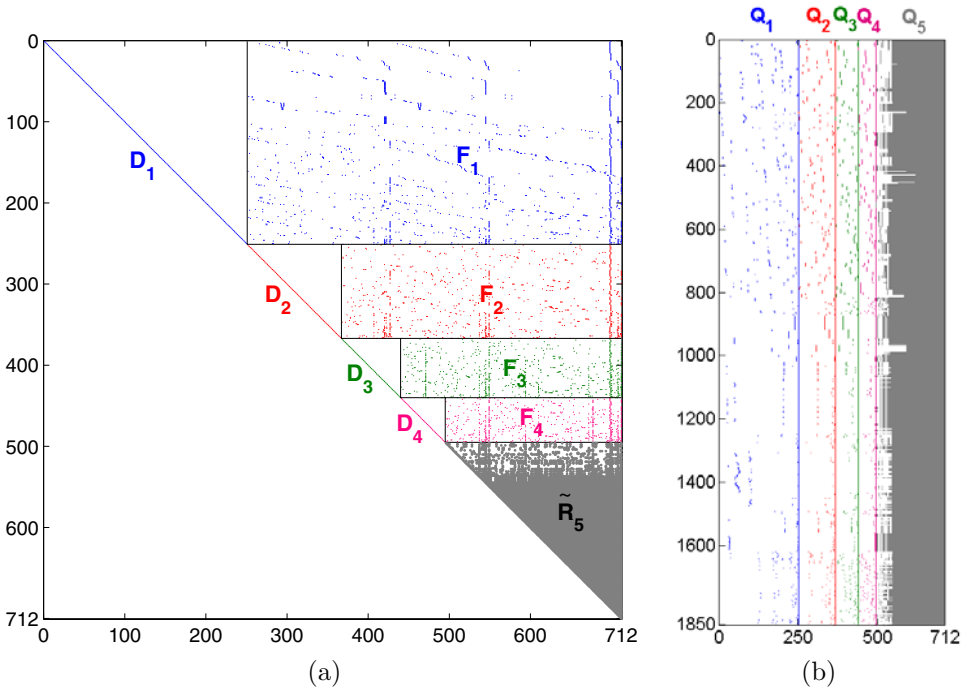

FIG. 4.1. *The MQR structure for matrix* WELL1850 (1,850 × 712, $nnz = 8{,}758$).

Note that $\widetilde{P}_{p+1}^T$ is the identity matrix when pivoting is not used. Then, the above lemma shows that

$$AP_1^T \cdots P_p^T P_{p+1}^T = [Q_1, \ldots, Q_p, Q_{p+1}] \begin{bmatrix} R_{11} & R_{12}\widetilde{P}_{p+1}^T \\ 0 & \widetilde{R}_{p+1} \end{bmatrix}. \tag{4.13}$$

This yields a permuted QR factorization, since it is easily shown that the columns of $[Q_1, Q_2, \ldots, Q_{p+1}]$ are orthonormal.

LEMMA 4.2. *Let A be of full rank, and define $\mathcal{P}^T = P_1^T P_2^T \cdots P_{p+1}^T$ and $\mathcal{Q} = [Q_1, \ldots, Q_p, Q_{p+1}]$. Then $\mathcal{Q}$ is unitary and the MQR procedure computes a permuted QR factorization of A; i.e., we have*

$$A\mathcal{P}^T = \mathcal{Q}\mathcal{R}, \tag{4.14}$$

where $\mathcal{R}$ is an upper triangular matrix.

Proof. Part of the result is established in (4.13). The only situation when the algorithm will break down is when a diagonal entry in D_i is zero or when the last factorization fails. This is impossible when A is of full rank. It remains only to show that the matrix $[Q_1, \ldots, Q_p, Q_{p+1}]$ in (4.13) is indeed unitary. Within the same block Q_i the columns are orthogonal structurally and are normalized. So $Q_i^T Q_i = I$. For $j > i$ we have $Q_i^T Q_j = 0$. Indeed, the columns of Q_j are linear combinations of columns of A_i because $j > i$. However, by construction A_i is orthogonal to Q_i so we have $Q_i^T Q_j = 0$. ∎

An illustration of the sizes and the positions of $D_1, F_1, \ldots, D_p, F_p$, and $\widetilde{R}_{p+1}$ can be visualized in Figure 4.1(a), where a four-level QR factorization process ($p = 4$) has been applied to matrix WELL1850. (Some information on this matrix can be found

in section 5.) Figure 4.1(b) shows the corresponding matrix $Q = [Q_1, \ldots, Q_p, Q_{p+1}]$. Note that in order to obtain a better quality picture, the Q and R factors are scaled differently in the figure (the column/row size of R is the same as the column size of Q).

Another formulation of the factorization, which will be used later, is to write

$$A = Q\widehat{R}, \tag{4.15}$$

where $\widehat{R} = R_{p+1}P_{p+1}R_pP_p \cdots R_1P_1 \in \mathbb{R}^{n\times n}$ and $Q = [Q_1, \ldots, Q_p, Q_{p+1}] \in \mathbb{R}^{m\times n}$. Under similar notations as the one-level process, R_i has the form

$$R_i = \begin{bmatrix} I & 0 & 0 \\ 0 & D_i & F_i \\ 0 & 0 & I \end{bmatrix}, \quad i = 1, 2, \ldots, p, \quad \text{and} \quad R_{p+1} = \begin{bmatrix} I & 0 \\ 0 & \widetilde{R}_{p+1} \end{bmatrix}.$$

4.2. Multilevel incomplete QR factorization. If the MQR factorization is complete, we have

$$A^TA = \widehat{R}^T\widehat{R}.$$

Since $\widehat{R}$ is a product of permutation matrices and upper triangular matrices, it is normally inexpensive to solve the equation $(\widehat{R}^T\widehat{R})x = y$. Therefore, an approximation $M \approx \widehat{R}^T\widehat{R}$, obtained through an incomplete multilevel QR factorization process, can be used as a preconditioner for solving (1.2). In the following, a few strategies are considered for developing practical variants of the exact MQR algorithm just described. These will lead to the MIQR preconditioner.

4.2.1. Relaxed independent set of columns. At each level of MQR, we would like to find a larger independent set of columns so that the reduced matrix is smaller. However, there are cases when an independent set with a large size does not even exist. For example, in an extreme case where all entries in one row of a matrix are nonzero, any two column vectors of the matrix are adjacent to each other; i.e., an edge exists between any two vertices in $CIG(A)$. In this case, the largest independent set will consist of only one vertex.

For the purpose of preconditioning, the orthogonality requirement can be somewhat relaxed since only an approximation of the factorization is needed. Therefore, in order to obtain a larger independent set, as well as to reduce fill-in, we will treat two column vectors as being "orthogonal" whenever the acute angle between them is "close" to a right angle. Specifically, for a given small value $\tau_\theta > 0$, an edge from vertex i to vertex j is considered to belong to $CIG(A)$ if $|\cos\theta_{ij}| \geq \tau_\theta$. This replaces the original condition that $\cos\theta_{ij} \neq 0$. The scalar τ_θ is termed the *angle threshold*. We denote the CIG obtained under the angle threshold τ_θ by $CIG(A, \tau_\theta)$.

Let $C(\tau_\theta)$ be the matrix obtained by replacing all elements less than τ_θ in absolute value in the matrix C defined by (3.1) with 0. Clearly, the graph $CIG(A, \tau_\theta)$ is the adjacency graph of the matrix $C(\tau_\theta)$. Recall that the entries in C (and $C(\tau_\theta)$) are cosines of columns of B, which represent the patterns of the columns of A. Alternatively, the cosines can be calculated using the real values in A instead. To do so, $B = [\frac{a_1}{\|a_1\|}, \frac{a_2}{\|a_2\|}, \ldots, \frac{a_n}{\|a_n\|}]$ is used to calculate C instead of B given by (3.1). In our implementation the cosines were evaluated using the real values in A. Once $CIG(A, \tau_\theta)$ is obtained, Algorithm 3.1 or 3.2 can be applied to $CIG(A, \tau_\theta)$ to find an independent set. The independent set found in this way is in general significantly larger than that found by applying the same algorithm on $CIG(A)$. The effectiveness of this relaxed independent set ordering strategy is illustrated in section 5.

4.2.2. Dropping strategies. The multilevel process yields denser and denser intermediate matrices F_i in general. To ensure a moderate memory usage, we usually drop small terms from F_i. Since a relaxed orthogonality strategy is employed (see previous section), this same strategy is applied when computing the matrix F_i. Recall that $F_i = Q_i^T A_{i-1}^{(2)}$, where Q_i includes normalized columns in the independent set S found at level i, and $A_{i-1}^{(2)}$ includes all remaining columns which are not in S. Therefore, any element in F_i is an inner product between a column vector in S and another column vector not in S. For a given angle threshold τ_θ, the element is replaced by 0 if the cosine of the angle between these two column vectors is less than τ_θ in absolute value. Assume that $F_i = \{f_{uv}\}$, $Q_i = [q_1, q_2, \ldots, q_s]$, and $A_{i-1}^{(2)} = [a_1, a_2, \ldots, a_t]$. Then $f_{uv} = q_u^T a_v = \|a_v\|_2 \cos\theta_{uv}$, where θ_{uv} is the angle between q_u and a_v. Thus, f_{uv} is dropped if $|f_{uv}| < \tau_\theta \|a_v\|_2$.

The final reduced matrix is normally much denser than the original matrix A. If it is small enough (e.g., around 100 columns), a standard QR factorization can be applied. Note that this matrix can now be treated as dense in order to take advantage of effective block computations. Otherwise, an incomplete QR factorization is applied. In this paper, we aim to compare MIQR with the IQR preconditioner outlined in Algorithm 2.1. Therefore, we use the same IQR algorithm on the reduced matrix to ensure that the implementations are as close as possible. In the implementation of IQR, fill-ins are dropped dynamically when the columns of Q and R are being formed based on the magnitude of the columns generated.

4.2.3. MIQR. With the relaxed independent sets of columns and dropping strategies described above, the MIQR algorithm can be described as follows.

ALGORITHM 4.2. Multilevel incomplete QR factorization with angle threshold (MIQR(τ_θ))

1. $k := 0$; $A^{(0)} = A$; $p = maxlev$.
2. While $k < p$ *Do:*
3. Construct column intersection graph under angle threshold τ_θ: $CIG(A_k, \tau_\theta)$.
4. Find an independent set permutation $\widetilde{P}_{k+1}$ for $CIG(A_k, \tau_\theta)$.
5. Apply permutation $[A_k^{(1)}, A_k^{(2)}] = A_k \widetilde{P}_{k+1}^T$.
6. Let $D_{k+1} := \mathrm{diag}(||a_1^{(k)}||, \ldots, ||a_s^{(k)}||)$, where $A_k^{(1)} = [a_1^{(k)}, \ldots, a_s^{(k)}]$.
7. Let $Q_{k+1} := A_k^{(1)} D_{k+1}^{-1}$ and $F_{k+1} := Q_{k+1}^T A_k^{(2)}$.
8. Apply a dropping strategy to F_{k+1}.
9. $A_{k+1} := A_k^{(2)} - Q_{k+1} F_{k+1}$.
10. Apply a dropping strategy to A_{k+1}.
11. $k := k + 1$.
12. *EndDo*
13. Apply IQR (or QR) on A_p: $A_p \widetilde{P}_{p+1}^T \approx Q_{p+1} \widetilde{R}_{p+1}$.

Some implementation details are now discussed. Theoretically, we can continue the multilevel incomplete QR factorization process until the reduced matrix is very small or the system cannot be further reduced. Practically, the overhead of the multilevel process increases substantially as more levels are taken. At the same time, since the multilevel process yields denser and denser matrices, the number of independent columns available becomes much smaller. Therefore, it is best to stop the multilevel process when a certain number of levels (*maxlev* in Algorithm 4.2) is reached or the size of the reduced problem is not significant (e.g., less than 30% of the previous problem size).

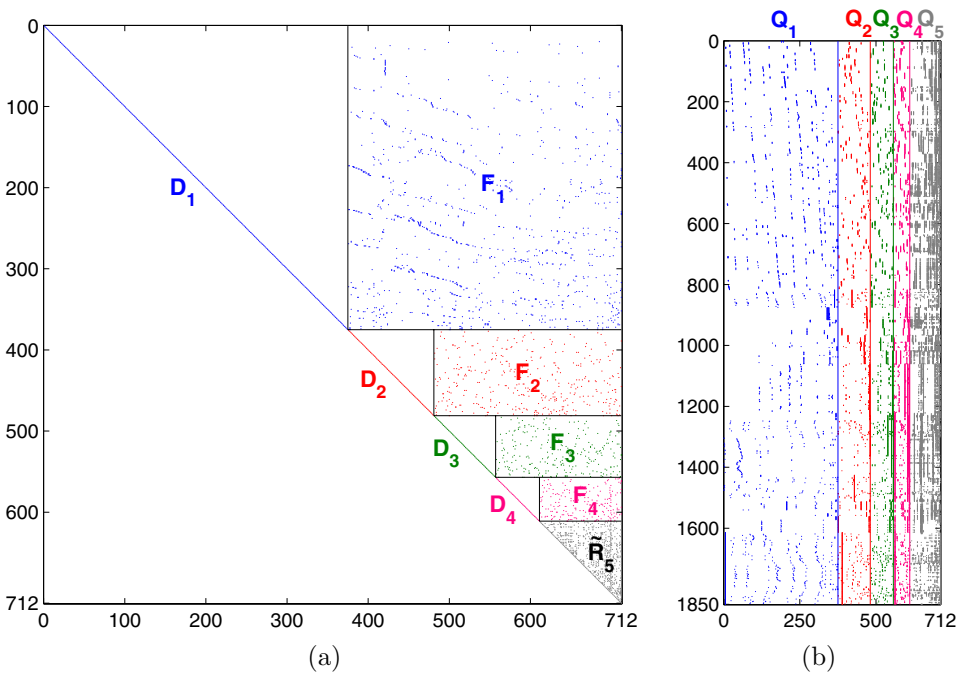

FIG. 4.2. *The MIQR structure for matrix* WELL1850 (1,850 × 712, $nnz = 8{,}758$).

Recall that at each level $CIG(A, \tau_\theta)$ is the adjacency graph of $C(\tau_\theta)$ which is available from the matrix $C = B^T B$; see (3.1). However, the matrix C need not be calculated explicitly. Since we determine the degrees of the vertices one by one, only a single row of C is needed at any given time. In other words, to determine the degree of vertex i, only the ith row of C needs to be calculated. This row contains the inner products between the ith column of B and all other columns. As indicated in [30] (see also section 2), these inner products can be efficiently calculated as a linear combination of the rows of B. For this reason, although a reduced matrix is naturally formed and stored columnwise during the MQR factorization process, we maintain an index array for easily accessing its elements rowwise. Furthermore, since C is symmetric, only its upper part (i.e., the inner products between the ith column of B and columns from $i+1$ to n) needs to be calculated.

As is standard practice, the permutation matrices $\widetilde{P}_k$ are not formed explicitly. Instead, a permutation array $perm^{(k)}$ is employed to hold the new ordering of the columns at each level, along with the inverse permutation array $iperm^{(k)}$. With this strategy, the columns of matrix A_k are kept in their original ordering and the permutation step (line 5 in Algorithm 4.2) can be avoided. To construct an MIQR preconditioner, the matrix array $\{D_1, F_1, D_2, F_2, \ldots, D_p, F_p, \widetilde{R}_{p+1}\}$ and the permutation arrays $perm^{(k)}$ and $iperm^{(k)}$ are stored. Similarly to Figure 4.1, this matrix array can be organized in an $n \times n$ matrix as illustrated in Figure 4.2, where $\tau_\theta = 0.1$ is used and IQR is applied to the final reduced matrix. Moreover, since the matrices Q_k are not needed for the preconditioning purpose, they are discarded at the end of the kth level recursion, respectively.

Other than the dropping strategies discussed in section 4.2.2, an optional dropping rule may be applied to A_{k+1} as well (see line 10 of Algorithm 4.2). This is used mainly

as a means of preventing A_{k+1} from becoming too dense. For example, when needed, any nonzero entry in A_{k+1} whose absolute value is less than a threshold τ times a certain norm of the column under consideration will be dropped. However, dropping in A_{k+1} should be applied sparingly as we observed in our experiments that this can negatively impact the robustness of the resulting preconditioner. Using a small value of the threshold value τ is recommended if dropping is to be done on A_{k+1}, as dropping very small terms is less harmful. For the test problems used in this paper, all the intermediate matrices generated at line 10 in Algorithm 4.2 were reasonably sparse. Therefore, for the sake of maintaining robustness, no dropping was applied to A_{k+1} in the tests reported in section 5, (i.e., we use $\tau = 0$).

As mentioned before, the matrix $\widehat{R}$ defined as in (4.15) is a product of permutation matrices and upper triangular matrices. To precondition the normal equations, we need to solve the systems $\widehat{R}x = y$ and $\widehat{R}^T x = y$. Let $s_k (k = 1, 2, \ldots, p)$ be the size of the independent set of columns at level k. Define $r_1 = 1$ and $r_k = r_{k-1} + s_{k-1}$ for $k = 2, \ldots, p+1$. Algorithms 4.3 and 4.4 are used to solve the systems $\widehat{R}x = y$ and $\widehat{R}^T x = y$, respectively.

ALGORITHM 4.3. Solving $\widehat{R}x = y$

1. $x(1:n) := y(1:n)$.
2. For $k = 1 : p$ *Do:*
3. Apply permutation $perm^{(k)}$ to $x(r_k : n)$.
4. $x(r_k : r_{k+1} - 1) := D_k^{-1} x(r_k : r_{k+1} - 1)$.
5. $x(r_{k+1} : n) := x(r_{k+1} : n) - F_k^T x(r_k : r_{k+1} - 1)$.
6. *EndDo*
7. Solve $\widetilde{R}_{p+1} z = x(r_{p+1} : n)$ for z.
8. $x(r_{p+1} : n) := z$.

ALGORITHM 4.4. Solving $\widehat{R}^T x = y$

1. $x(1:n) := y(1:n)$.
2. Solve $\widetilde{R}_{p+1} z = y(r_{p+1} : n)$ for z.
3. $x(r_{p+1} : n) := z$.
4. For $k = p : -1 : 1$ *Do:*
5. $x(r_k : r_{k+1} - 1) := D_k^{-1}[x(r_k : r_{k+1} - 1) - F_k x(r_{k+1} : n)]$.
6. Apply permutation $iperm^{(k)}$ to $x(r_k : n)$.
7. *EndDo*

4.3. Analysis. In this section we will analyze the errors generated by the MIQR factorization that are due to dropping small terms. Two questions are important to consider. The first is, How far does the result deviate from satisfying the relation $A\mathcal{P}^T = \mathcal{Q}\mathcal{R}$? The second is, How much does $\mathcal{Q}$ deviate from orthogonality in the presence of dropping?

The basic step of MIQR can be described by approximate versions of the relations (4.7)–(4.11). Specifically, the equations that define F_i and A_i will change while Q_i, D_i can be assumed to be exact:

$$D_i = \text{diag}\left(\|A_{i-1}^{(1)} e_j\|_2 \right)_{j=1,\ldots,s_i}, \tag{4.16}$$

$$Q_i = A_{i-1}^{(1)} D_i^{-1}, \tag{4.17}$$

$$\widetilde{F}_i = Q_i^T A_{i-1}^{(2)} + E_{F,i}, \tag{4.18}$$

$$\widetilde{A}_i = A_{i-1}^{(2)} - Q_i \widetilde{F}_i + E_i. \tag{4.19}$$

The term E_i comes from dropping entries when forming the block A_i while the term $E_{F,i}$ comes from dropping when forming the matrix F_i; see Algorithm 4.2. Then assuming A_{i-1} is exact, the relation (4.7) becomes

$$A_{i-1}\widetilde{P}_i^T = [A_{i-1}^{(1)}, A_{i-1}^{(2)}] = [Q_i, \widetilde{A}_i]\begin{bmatrix} D_i & \widetilde{F}_i \\ 0 & I \end{bmatrix} + [0, E_i]. \tag{4.20}$$

Note the remarkable absence of $E_{F,i}$ from the error in (4.20). The error is imbedded in $\widetilde{F}_i$. Before continuing, we can further note that if there is no dropping when building A_i, then $E_i \equiv 0$ and the relation (4.7) becomes exact. This means that in the end we would expect to have the relation $A\mathcal{P}^T = \mathcal{Q}\mathcal{R}$ exactly satisfied, but $\mathcal{Q}$ is not necessarily unitary. Now we consider the general case and will attempt to analyze the difference between $A\mathcal{P}^T$ and $\mathcal{Q}\mathcal{R}$. This case is important to consider because Algorithm 4.14 may include dropping in A_{k+1} after it is constructed; see line 10 of the algorithm. (Although we did not apply this dropping in our tests reported in this paper, it helps to keep A_{k+1} sparse in general.)

Consider the result of Lemma (4.1) which would be desirable to generalize. In the following we attempt to extend the argument used in the proof of Lemma 4.1. We write the above relation for level $i+1$ as

$$\widetilde{A}_i = \Big[Q_{i+1}, \widetilde{A}_{i+1}\Big] \underbrace{\begin{bmatrix} D_{i+1} & \widetilde{F}_{i+1} \\ 0 & I \end{bmatrix}}_{Z_{i+1}} \widetilde{P}_{i+1} + \underbrace{[0, E_{i+1}]}_{G_{i+1}} \widetilde{P}_{i+1}.$$

Substituting in (4.12) (where A_i is replaced by the computed $\widetilde{A}_i$ at the ith step) yields

$$\begin{aligned}
&\Big[Q_1, \ldots, Q_i, \widetilde{A}_i\Big] \begin{bmatrix} R_{11} & R_{12} \\ 0 & I \end{bmatrix} \\
&\quad = \Big[Q_1, \ldots, Q_i \mid [Q_{i+1}, \widetilde{A}_{i+1}]Z_{i+1}\widetilde{P}_{i+1} + G_{i+1}\widetilde{P}_{i+1}\Big] \left[\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & I \end{array}\right] \\
&\quad = \Big[Q_1, \ldots, Q_i \mid [Q_{i+1}, \widetilde{A}_{i+1}]\Big] \left[\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & Z_{i+1}\widetilde{P}_{i+1} \end{array}\right] + [0, G_{i+1}\widetilde{P}_{i+1}] \\
&\quad = \Big[Q_1, \ldots, Q_i, Q_{i+1}, \widetilde{A}_{i+1}\Big] \begin{bmatrix} R_{11} & R_{12}\widetilde{P}_{i+1}^T \\ 0 & Z_{i+1} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \widetilde{P}_{i+1} \end{bmatrix} \\
&\qquad + [0, G_{i+1}] \begin{bmatrix} I & 0 \\ 0 & \widetilde{P}_{i+1} \end{bmatrix}.
\end{aligned}$$

In what follows we will denote by $\mathcal{R}_i$ the matrix $\left[\begin{smallmatrix} R_{11} & R_{12} \\ 0 & I \end{smallmatrix}\right]$ at step i. So the above relation translates into

$$\Big[Q_1, \ldots, Q_i, \widetilde{A}_i\Big]\mathcal{R}_i = \Big[Q_1, \ldots, Q_i, Q_{i+1}, \widetilde{A}_{i+1}\Big]\mathcal{R}_{i+1}P_{i+1} + [0, G_{i+1}]P_{i+1}. \tag{4.21}$$

The left-hand side of the above relation is equal to $AP_1^T \cdots P_i^T + \mathcal{E}_i$, where $\mathcal{E}_i$ denotes the total error made at step i of the factorization. Using a similar notation for the first term of the right-hand side will transform (4.21) into

$$AP_1^T \cdots P_i^T + \mathcal{E}_i = (AP_1^T \cdots P_iP_{i+1}^T + \mathcal{E}_{i+1})P_{i+1} + [0, G_{i+1}]P_{i+1}. \tag{4.22}$$

This means that $\mathcal{E}_i = \mathcal{E}_{i+1}P_{i+1} + [0, G_{i+1}]P_{i+1}$, and it establishes the remarkably simple recurrence relation for the total error

$$\mathcal{E}_{i+1} = \mathcal{E}_i P_{i+1}^T - [0, E_{i+1}], \tag{4.23}$$

where the zero block in the right-hand side has the same number of columns as $[Q_1, Q_2, \ldots, Q_i]$. In particular we have

$$\|\mathcal{E}_{p+1}\| \leq \sum_{i=1}^{p+1} \|E_{i+1}\|.$$

However, this inequality does not say everything about the errors. For example, it is clear that the last columns (after permutation) will undergo more perturbations than the first ones and they will therefore be less accurate. This is understandable since, for example, the columns of the first level are not perturbed by the other columns.

Consider now the accuracy of the process with respect to orthogonality. For simplicity, we will consider only the situation where there is no dropping in forming $\widetilde{A}_i$; i.e., the case where $E_i = 0$. Furthermore, we assume that each Q_i, considered individually, is exactly orthonormal; i.e., $Q_i^T Q_i = I$. In particular this means that τ_θ should be equal to zero. In this case, it is easy to see from (4.23) that the relation $A\mathcal{P}^T = \mathcal{Q}\mathcal{R}$ is exactly satisfied. However, dropping entries in F_i will cause loss of orthogonality.

Consider only one step of the process. From (4.17), (4.18), and (4.19) (with $E_i = 0$) we obtain

$$Q_i^T \widetilde{A}_i = Q_i^T (A_{i-1}^{(2)} - Q_i \widetilde{F}_i) = Q_i^T A_{i-1}^{(2)} - (Q_i^T A_{i-1}^{(2)} + E_{F,i}) = -E_{F,i}.$$

Next we wish to establish a relation between this term and $Q_i^T \mathcal{Q}$. Specifically, because of the recursive nature of the algorithm, at step i we have a relation similar to that given by Lemma 4.2. A little additional notation is needed. Let $\mathcal{Q}_i = [Q_i, \ldots, Q_{p+1}]$ and $\mathcal{P}_i^T = P_i^T \cdots P_{p+1}^T$. Then,

$$\widetilde{A}_i \mathcal{P}_i = \mathcal{Q}_i \mathcal{R}_i,$$

where $\mathcal{R}_i$ is an upper triangular matrix. Multiplying to the left by Q_i^T yields

$$Q_i^T \widetilde{A}_i \mathcal{P}_i = Q_i^T \mathcal{Q}_i \mathcal{R}_i$$

so that

$$-E_{F,i} \mathcal{P}_i \mathcal{R}_i^{-1} = Q_i^T [Q_i, \ldots, Q_{p+1}].$$

The above relation shows in a simple way how the error made at step i will propagate to the matrices $Q_i^T Q_j$ for $j > i$. The result involves the inverse of an unknown triangular matrix. The matrix $\mathcal{R}_i$ establishes the relation between the Q_j's and the matrix $\widetilde{A}_i$. For example, for $i = 1$ we would get all the matrices $Q_1^T Q_j$ for $j > 1$ in terms of $E_{F,1}$, the error related to dropping in the F matrix in the first step.

An interesting and important question which we do not address in this paper is the issue of effective dropping. In the papers [8, 9], an idea was considered for dropping in such a way that the preconditioned matrix is close to the identity. This is in contrast with other methods which try to make the preconditioner close to A. Though this idea involves the inverse of the preconditioner, heuristics can be used to provide quite effective methods. In the context of IQR, this may be doable but it will undoubtedly be more complex.

TABLE 5.1

Information on the test problems: Size = $m \times n$; nnz = the number of nonzeros; μ = the average number of nonzeros per row; ν = the average number of nonzeros per column.

Matrix	m	n	nnz	μ	ν	Source
ILLC1850	1,850	712	8,758	4.73	12.30	Surveying
WELL1850	1,850	712	8,758	4.73	12.30	Surveying
MESHPAR1	31,258	15,994	187,498	6.00	11.72	3D mesh parameterization
MESHPAR2	75,650	38,384	453,846	6.00	11.82	3D mesh parameterization
SMALL2	6,280	3,976	25,530	4.07	6.42	Animal breeding
MEDIUM2	18,794	12,238	75,039	3.99	6.13	Animal breeding
LARGE	28,254	17,264	75,018	2.66	4.35	Animal breeding
LARGE2	56,508	34,528	225,054	3.98	6.52	Animal breeding
VERYL	174,193	105,882	463,303	2.66	4.38	Animal breeding
VERYL2	348,386	211,764	1,389,909	3.99	6.56	Animal breeding

TABLE 5.2

Independent set sizes: Comparison of the lower bounds (3.7)*, the estimated numbers* (3.5)*, the actual sizes, and the rough lower bounds* (3.2)*.*

Matrix	New low. s	Est. val. s	Real val. s	Old low. s
ILLC1850	59	196	220	16
WELL1850	59	196	237	16
MESHPAR1	1,116	3,594	2,191	269
MESHPAR2	2,660	8,613	5,090	639
SMALL2	612	1,438	1,171	193
MEDIUM2	1,976	4,544	3,297	633
LARGE	5,048	9,648	9,690	2108
LARGE2	5,359	12,681	9,957	1,690
VERYL	30,788	59,020	60,454	12,815
VERYL2	32,661	77,552	61,638	10,269

5. Numerical results. In this section, we test the performance of the MIQR method on ten least-squares problems from real applications. Table 5.1 provides some basic information about the test matrices. In the table, m is the number of rows, n is the number of columns, nnz is the total number of nonzeros, μ is the average number of nonzeros per row, and ν is the average number of nonzeros per column. Matrices ILLC1850 and WELL1850 are available from the Matrix Market.[1] The next two matrices are from a three-dimensional mesh parameterization problem.[2] These matrices are generated using the method of least-squares conformal maps as described in [23]. The last six matrices (SMALL2, MEDIUM2, LARGE, LARGE2, VERYL, VERYL2) arise in animal breeding studies [16, 17] and can be downloaded from ftp://ftp.cerfacs.fr/pub/algo/matrices/animal. The matrices SMALL2, MEDIUM2, LARGE2, and VERYL2 are generated from the code `conv2.f` while LARGE and VERYL are generated using `conv.f`. Both `conv.f` and `conv2.f` are included in the package available from the website.[3]

We first test the accuracy of the estimate on the number of independent sets of columns as described in section 3.2. Table 5.2 shows the lower bounds (in field "New low. s") estimated by (3.7) and the values estimated by solving (3.5) numerically (in

[1] http://math.nist.gov/MatrixMarket/.

[2] This problem was provided to us by Minh Nguyen from the Graphics group at the University of Minnesota, Department of Computer Science and Engineering.

[3] The difference between `conv.f` and `conv2.f` is explained in READ_ME as follows: "The code conv.f constructs problems where only the single trait 'weight increase' is considered. The code `conv2.f` deals with both traits. The matrices constructed using `conv.f` are smaller than those constructed using conv2.f." For more details see the references cited above.

TABLE 5.3
Numbers of independent columns found using different angle tolerances for the first two levels.

Matrix	Level	$\tau_\theta = 0.00$	$\tau_\theta = 0.05$	$\tau_\theta = 0.10$	$\tau_\theta = 0.15$	$\tau_\theta = 0.20$
ILLC1850	1	220	327	338	343	350
	2	130	146	152	159	170
WELL1850	1	237	346	372	395	432
	2	122	101	115	119	134
MESHPAR1	1	2,191	6,151	6,594	6,471	6,681
	2	2,191	2,483	3,397	4,213	5,411
MESHPAR2	1	5,090	15,685	16,829	15,818	15,655
	2	5,090	5,506	7,973	9,823	13,918
SMALL2	1	1,171	1,623	1,995	2,793	2,884
	2	1,147	1,022	1,087	683	684
MEDIUM2	1	3,297	4,836	6,154	8,092	8,308
	2	3,110	3,274	2,941	2,273	2,340
LARGE	1	9,690	10,280	10,222	10,545	12,554
	2	3,794	4,113	4,563	4,509	3,320
LARGE2	1	9,957	14,681	18,523	24,994	25,724
	2	9,312	9,891	8,552	6,038	6,131
VERYL	1	60,454	61,032	64,107	66,579	77,343
	2	22,095	23,495	25,781	25,000	20,203
VERYL2	1	61,638	89,500	114,752	153,992	157,109
	2	57,924	58,804	51,687	36,741	37,615

field "Est. val. s") for the ten matrices. These lower bounds and estimated values are compared with the real values calculated by Algorithm 3.2 (in field "Real val. s"). For reference, we also calculate the rough lower bounds estimated by (3.2) (in field "Old low. s"), where the average degree η (3.4) is used as the value of d_S. It is clear that the values calculated using (3.7) provide much closer lower bounds. Recall that (3.5) and (3.7) are derived under the assumption that the nonzero elements of a matrix are randomly distributed. In spite of this assumption, the estimated values can still provide good approximations for the matrices from real applications.

Table 5.3 presents the results of finding the independent columns using different angle tolerances τ_θ as described in section 4.2.1. In the table, $\tau_\theta = 0.00$, 0.05, 0.10, 0.15 and 0.20 (corresponding to angles 90°, 87.13°, 84.26°, 81.37°, and 78.46°, respectively) are tested. Algorithm 3.2 is used for all tests. For each τ_θ, we list the number of independent columns found in the first two levels. From the table, as expected, the number of independent columns found in each level is significantly increased as the angle tolerance increases. As an example, for matrix VERYL2, without relaxing the criterion of finding independent columns (i.e., $\tau_\theta = 0.00$), only 61,638 and 57,924 independent columns are found in the first two levels of reduction respectively; i.e., the problem size reduced after the first two levels is 119,562. With the angle tolerances, the problem sizes reduced after the first two levels increase to 148,304, 166,439, 190,733, and 194,724, respectively, for $\tau_\theta = 0.05$, 0.10, 0.15, and 0.20.

Next, we test MIQR on the ten least-squares problems and compare the results with IQR (Algorithm 2.1), RIF [4] preconditioners, and ARMS (on the normal equations). MIQR, IQR, and ARMS were coded in C. RIF provided by Benzi and Tůma was in FORTRAN90.[4] All codes were compiled in 64-bit mode with the -O2 opti-

[4]The RIF package (rifsrnri.tar.gz) is also available at http://www.cs.cas.cz/~tuma/sparslab.html. We only changed the two drop tolerances in the `rifsrnri` source code, one for the SAINV process (to approximate $A^{-1}b$) and the other for the postfiltration of RIF, to achieve the desired fill-in factors for comparison purposes in our tests. All other parameters in the code were left unchanged.

TABLE 5.4
Performance of MIQR under different angle tolerances ($\tau_\theta = 0.000$ and $\tau_\theta = 0.015$).

Matrix	τ_θ	Levels	Res.#	Fill-in	Pre.T	ITS	Its.T	Tot.T
ILLC1850	0.00	2	360	0.668	0.04	290	0.35	0.39
	0.10	4	77	0.328	0.04	172	0.18	0.22
	0.20	4	34	0.219	0.02	183	0.17	0.19
WELL1850	0.00	2	353	0.482	0.04	85	0.10	0.14
	0.10	5	60	0.322	0.06	68	0.06	0.12
	0.20	5	10	0.224	0.02	133	0.12	0.14
MESHPAR1	0.00	2	11,612	0.763	7.25	460	14.27	21.52
	0.04	2	7,536	0.567	5.52	405	10.81	16.33
	0.08	2	6,424	0.513	4.60	530	13.59	18.19
MESHPAR2	0.00	3	25,458	1.097	34.89	731	71.13	106.02
	0.05	3	12,672	0.650	22.02	800	60.84	82.86
	0.10	3	8,938	0.495	13.00	1,357	91.39	104.39
SMALL2	0.00	3	1,299	1.073	0.21	247	1.19	1.40
	0.10	3	506	0.530	0.15	241	0.93	1.08
	0.20	3	132	0.334	0.08	284	0.89	0.97
MEDIUM2	0.00	3	4,724	1.299	0.88	223	3.89	4.77
	0.10	3	1,756	0.628	0.55	235	3.03	3.58
	0.25	3	254	0.304	0.16	407	4.09	4.25
LARGE	0.00	3	2,134	1.247	0.72	44	0.86	1.58
	0.05	3	1,446	1.014	0.52	69	1.21	1.73
	0.10	4	594	0.867	2.22	128	2.12	4.34
LARGE2	0.00	3	12,087	1.234	3.13	361	20.62	23.75
	0.10	3	3,769	0.515	2.25	442	18.29	20.54
	0.20	3	292	0.251	0.37	461	15.47	15.84
VERYL	0.00	4	8,176	1.120	6.88	118	17.11	23.99
	0.18	4	613	0.512	2.19	221	25.59	27.78
	0.20	4	259	0.447	1.56	196	21.51	23.07
VERYL2	0.00	1	150,126	1.430	40.66	916	401.60	442.26
	0.10	4	11,586	0.502	26.50	497	157.71	184.21
	0.25	3	1,522	0.256	3.22	737	186.02	189.24

mization option. All experiments were performed on an IBM SP machine, which has four 222MHz processors sharing 16GB memory. Note that we did not take advantage of parallelism in our tests; i.e., only one of the four processors was used. The right-hand sides available from the original data were used. This is in contrast to [4] where artificial right-hand sides were employed. Algorithm 1.1 with a zero initial guess was used to solve all the problems. The iterations were stopped when

$$\|A^T b - A^T A x^{(k)}\|_2 < 10^{-8} \|A^T b - A^T A x^{(0)}\|_2$$

or the maximum iteration count of 2,000 was reached. To better compare the preconditioners, we use an indicator called a fill-in factor to indicate the memory usage for each method. The fill-in factor is defined as the ratio between the memory used in a preconditioner and the memory used in the original matrix. The memory used in MIQR is represented by the total number of nonzero entries in matrices D_1, $F_1, \ldots, D_p$, F_p, $\tilde{R}_{p+1}$ as shown in Figure 4.2. We wish to compare the preconditioners under similar fill-in factors.

In Table 5.4, we test MIQR on the ten matrices under different angle tolerances τ_θ. In the table, "Levels" is the number of levels used, "Res.#" is the number of columns of the final reduced matrix, "Fill-in" is the fill-in factor, "Pre.T" is the preconditioning time in seconds, "ITS" is the number of iterations for CGLS to reach convergence, "Its.T" is the iteration time in seconds, and "Tot.T" is the total time in

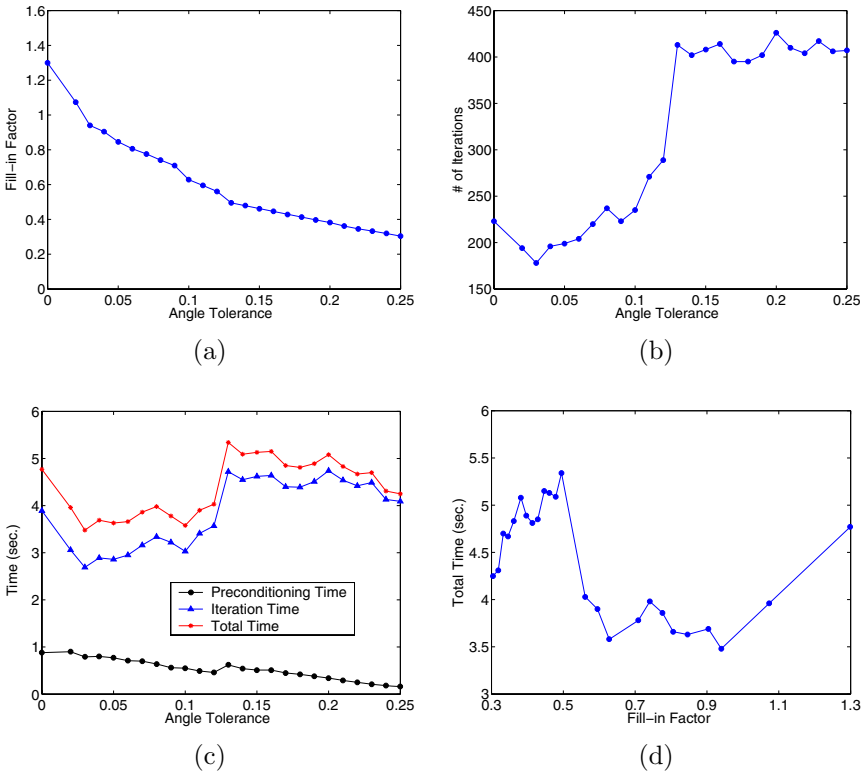

FIG. 5.1. *MIQR performance on matrix* MEDIUM2*:* (a) *Fill-in factor versus angle tolerance.* (b) *Total number of iterations versus angle tolerance.* (c) *Preconditioning time, iteration time, and total time versus angle tolerance.* (d) *Total Time versus fill-in factor.*

seconds. According to the table, and as expected, the size of the final reduced system decreases as the angle tolerance increases. For example, after the same number of reduction levels, the reduced system sizes for matrix MEDIUM2 are 4,724, 1,756, and 254 under angle tolerances 0.00, 0.10, and 0.25, respectively. As a result, the memory usage of MIQR decreases correspondingly. It is also apparent that allowing more fill-ins for the MIQR preconditioner does not necessarily provide faster convergence rates. For matrix ILLC1850, as an example, it takes CGLS 290 iterations to converge under a fill-in factor of 0.668 when $\tau_\theta = 0.00$. However, it takes only 172 iterations under a smaller fill-in factor of 0.328 when $\tau_\theta = 0.10$. This is because the accuracy of the MIQR preconditioner is determined not only by the fill-ins allowed but by many other factors. For example, it is not known how dropping affects the orthogonality of the underlying $\mathcal{Q}$ factor. Recall that $A^TA - M$ is not as important as $I - M^{-1}A^TA$ when it comes to analyzing convergence. From the table, we also observe that only one level is applied to VERYL2 when $\tau_\theta = 0.0$. In this case, it is due to the fact that the reduced size in the first level (61,638 according to Table 5.3) is only a small portion of the original problem size (211,764), which is less than the threshold we set to stop the multilevel process (see section 4.2.3).

We further examine the relationships among the angle tolerance, the fill-in factor, the number of iterations, and the execution times for matrix MEDIUM2 in Figure 5.1. Figure 5.1(a) shows the fill-in factors of the MIQR preconditioner as a function of the angle tolerance τ_θ. Figure 5.1(b) shows the number of iterations required

TABLE 5.5
A comparison of MIQR, IQR, and RIF.

Matrix	Method	Fill-in	Pre.T	ITS	Its.T	Tot.T
ILLC1850	MIQR	0.328	0.04	172	0.18	0.22
	IQR	0.332	0.02	1,512	1.45	1.47
	RIF	0.332	0.19	406	0.28	0.47
WELL1850	MIQR	0.322	0.06	68	0.06	0.12
	IQR	0.333	0.01	439	0.43	0.44
	RIF	0.324	0.21	90	0.07	0.28
MESHPAR1	MIQR	0.567	5.52	405	10.81	16.33
	IQR	0.582	2.06	666	17.31	19.37
	RIF	0.586	3.23	700	12.44	15.67
MESHPAR2	MIQR	0.650	22.02	800	60.84	82.86
	IQR	0.650	5.36	1,462	96.27	101.63
	RIF	0.642	12.30	1,567	74.14	86.44
SMALL2	MIQR	0.334	0.08	284	0.89	0.97
	IQR	0.330	0.03	621	2.30	2.33
	RIF	0.321	6.13	285	0.75	6.88
MEDIUM2	MIQR	0.304	0.16	407	4.09	4.25
	IQR	0.334	0.08	1,315	15.93	16.01
	RIF	0.326	22.49	526	4.50	26.99
LARGE	MIQR	1.014	0.53	69	1.23	1.76
	IQR	1.013	0.43	157	3.02	3.45
	RIF	1.014	2.44	59	0.76	3.20
LARGE2	MIQR	0.251	0.37	461	15.47	15.84
	IQR	0.319	0.22	-	-	-
	RIF	0.266	88.86	634	17.27	106.13
VERYL	MIQR	0.512	2.19	221	25.59	27.78
	IQR	0.515	0.70	997	117.28	117.98
	RIF	0.538	4.44	188	15.56	20.00
VERYL2	MIQR	0.256	3.22	737	186.02	189.24
	IQR	1.090	7.38	-	-	-
	RIF	0.288	1,306.88	1,220	226.70	1,533.58

for CGLS to converge as a function of the angle tolerance τ_θ. Figure 5.1(c) shows the preconditioning time, the iteration time, and the total time as a function of the angle tolerance τ_θ. Finally, Figure 5.1(d) shows the total time used to solve problem MEDIUM2 as a function of the fill-in factor of the MIQR preconditioner.

Table 5.5 compares MIQR with IQR and RIF. The symbol "-" in the table indicates that convergence was not obtained in 2,000 iterations. Note that matrices ILLC1850, WELL1850, LARGE, and VERYL have also been tested in [4] but with using artificial right-hand sides. In order to obtain comparable runs, we have put much effort into adjusting parameters in order to obtain similar fill factors for the same matrix. One should view the comparisons in this section with the following interpretation: How do the techniques compare for the same matrix if roughly the same fill-in is allowed for each method? Recall that MIQR implicitly uses a reordering of the columns and this is in fact the essence of the method. The other methods tested do not use any reordering of the columns. The superior performance of MIQR clearly shows that reordering can be vital. This observation is in agreement with [29], where it is shown that an IQR preconditioner can be made much more effective by applying a permutation prior to computing it. The table indicates that the set-up times for MIQR are slightly higher than those of IQR under similar memory usage, but that both its robustness and total execution times are significantly better. For matrices LARGE2 and VERYL2, IQR failed to converge in 2,000 iterations, even when much more fill-in was allowed. We also observe that MIQR had better overall performances

TABLE 5.6
A comparison of $nnz(A)$ and $nnz(A^TA)$.

Matrix	$nnz(A)$	$nnz(A^TA)$
ILLC1850	8,758	9,126
WELL1850	8,758	9,126
MESHPAR1	187,498	220,916
MESHPAR2	453,846	532,816
SMALL2	25,530	58,848
MEDIUM2	75,039	177,066
LARGE	75,018	128,798
LARGE2	225,054	524,036
VERYL	463,303	782,772
VERYL2	1,389,909	3,184,684

TABLE 5.7
A comparison of MIQR and ARMS (on the normal equations).

Matrix	Method	Levels	Fill-in	Pre.T	ITS	Its.T	Tot.T
ILLC1850	MIQR	4	0.328	0.04	172	0.18	0.22
	ARMS	3	1.329	0.05	94	0.20	0.25
WELL1850	MIQR	5	0.322	0.06	68	0.06	0.12
	ARMS	3	1.120	0.05	70	0.14	0.19
MESHPAR1	MIQR	2	0.567	5.52	405	10.81	16.33
	ARMS	2	1.401	2.78	-	-	-
MESHPAR2	MIQR	3	0.650	22.02	800	60.84	82.86
	ARMS	3	1.262	6.97	-	-	-
SMALL2	MIQR	3	0.334	0.08	284	0.89	0.97
	ARMS	3	1.216	0.32	261	3.92	4.24
MEDIUM2	MIQR	3	0.304	0.16	407	4.09	4.25
	ARMS	4	1.281	1.20	182	9.20	10.40
LARGE	MIQR	3	1.014	0.53	69	1.23	1.76
	ARMS	3	1.010	1.66	427	24.90	26.56
LARGE2	MIQR	3	0.251	0.37	461	15.47	15.84
	ARMS	3	1.356	5.30	-	-	-
VERYL	MIQR	4	0.512	2.19	221	25.59	27.78
	ARMS	4	1.025	37.10	55	21.80	58.90
VERYL2	MIQR	3	0.256	3.22	737	186.02	189.24
	ARMS	4	1.200	73.20	-	-	-

than RIF in general. For most matrices, MIQR required fewer iterations to converge than RIF. This is especially true for matrices MESHPAR1, MESHPAR2, and VERYL2, for which MIQR required significantly fewer iterations under similar memory costs. We caution, however, that comparisons are always difficult with iterative methods because of the large number of parameters that can be adjusted.

Finally, we compared MIQR with ARMS applied to the corresponding normal equations. The results are shown in Table 5.7. The symbol "-" in the table indicates failure to converge in 2,000 iterations or less. GMRES(100) was used as the accelerator for ARMS. In contrast to MIQR, the fill-in factors for ARMS are calculated based on the number of nonzeros of the normal equations (see Table 5.6). This makes it less relevant to compare the MIQR and ARMS-GMRES techniques from the point of view of memory usage (fill factors have a different meaning). Therefore, we choose the best results (in terms of overall time and memory usage) for ARMS in our tests. The results in Table 5.7 clearly show that solving the least-squares problems with MIQR is more effective than solving the corresponding normal equations by ARMS.

6. Conclusion. We have presented a preconditioning technique for solving large sparse least-squares systems that is based on a MIQR factorization. The algorithm exploits a divide-and-conquer strategy which takes advantage of structurally orthogonal columns. This allows us to gradually reduce a large problem to a significantly smaller one with little computational effort. The algorithm first finds an independent set of columns, which are structurally orthogonal. The remaining columns are then orthogonalized against this first set of columns, and the resulting set is orthogonalized recursively. In order to increase the size of the independent sets of columns, we proposed a strategy which consists of relaxing the orthogonality requirement. Numerical results have shown that this strategy is quite effective in finding independent column sets with large cardinality. The MIQR preconditioner has been tested and compared with a standard IQR factorization and with the RIF. The numerical tests show that MIQR is robust and efficient. We have not implemented a parallel version of the algorithm. However, the method has been designed with parallelism in mind and a parallel implementation should scale well.

In section 5, we have observed that the performances of MIQR may be very different when the angle tolerance varies. It remains to investigate a systematic way of selecting a good angle tolerance for a given problem. As mentioned in section 4.2.2, the current dropping strategies at all levels use simple techniques which tend to yield a factorization that is accurate, i.e., such that $A\mathcal{P}^T - \mathcal{Q}\mathcal{R}$ is small. As is the case for ILU factorizations, this is not necessarily a good strategy [8]. It may be possible to adapt Bollhöfer's work [8] to this context and develop more sophisticated dropping strategies which will in all likelihood improve the robustness of the scheme. Finally, as a further improvement, we would like to investigate more sophisticated IQR methods for the final reduced matrix, including using some effective reordering techniques as discussed in [29].

Acknowledgments. We would like to thank Michele Benzi and Miroslav Tůma, who provided the RIF source code used in section 5. We also thank Minh Nguyen for supplying the test matrices MESHPAR1 and MESHPAR2, and Sui Ruan for the valuable discussion on calculating the expected size of independent sets. We are grateful to three anonymous referees whose numerous suggestions helped improve the quality of this paper.

REFERENCES

[1] O. Axelsson and P. Vassilevski, *Algebraic multilevel preconditioning methods.* I. Numer. Math., 56 (1989), pp. 157–177.

[2] O. Axelsson and P. S. Vassilevski, *Algebraic multilevel preconditioning methods,* II, SIAM J. Numer. Anal., 27 (1990), pp. 1569–1590.

[3] R. Bank and C. Wagner, *Multilevel ILU decomposition*, Numer. Math., 82 (1999), pp. 543–576.

[4] M. Benzi and M. Tůma, *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM J. Sci. Comput., 25 (2003), pp. 499–512.

[5] A. Björck, *SSOR preconditioning methods for sparse least squares problems*, in Proceedings of the Computer Science and Statistics 12th Annual Symposium on the Interface, University of Waterloo, Waterloo, ON, Canada, 1979, pp. 21–25.

[6] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[7] A. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[8] M. Bollhöfer, *A robust ILU with pivoting based on monitoring the growth of the inverse factors*, Linear Algebra Appl., 338 (2001), pp. 201–213.

[9] M. Bollhöfer and Y. Saad, *Multilevel preconditioners constructed from inverse-based ILUs*, SIAM J. Sci. Comput., 27 (2006), pp. 1627–1650.

[10] R. Bramley and A. Sameh, *Row projection methods for large nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 168–193.
[11] T. F. Coleman and J. J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.
[12] E. Elmroth and F. Gustavson, *New serial and parallel recursive QR factorization algorithms for SMP systems*, in Applied Parallel Computing, Lecture Notes in Comput. Sci. 1541, Springer-Verlag, Berlin, 1998, pp. 120–128.
[13] E. Elmroth and F. Gustavson, *Applying recursion to serial and parallel QR factorization leads to better performance*, IBM J. Res. Develop., 44 (2000), p. 605–624.
[14] A. H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Rev., 47 (2005), pp. 629–705.
[15] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[16] M. Hegland, *On the computation of breeding values*, in Proceedings of CONPAR 90 - VAPP IV, Joint International Conference on Vector and Parallel Processing, Lecture Notes in Comput. Sci. 457, Springer-Verlag, New York, 1990, pp. 232–242.
[17] M. Hegland, *Description and Use of Animal Breeding Data for Large Least Squares Problems*, Tech. Report TR/PA/93/50, CERFACS, Toulouse, France, 1993.
[18] A. Jennings and M. A. Ajiz, *Incomplete methods for solving $A^T Ax = b$*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 978–987.
[19] M. Jones and P. Plassmann, *An improved incomplete Cholesky factorization*, ACM Trans. Math. Software, 21 (1995), pp. 5–17.
[20] C. Kamath, *Solution of Nonsymmetric Systems of Equations on a Multiprocessor*, Ph.D. thesis, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, IL, 1986.
[21] C. Kamath and A. H. Sameh, *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Comput., 9 (1989), pp. 291–312.
[22] R. Leuze, *Independent set orderings for parallel matrix factorizations by Gaussian elimination*, Parallel Comput., 10 (1989), pp. 177–191.
[23] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, *Least squares conformal maps for automatic texture atlas generation*, in ACM Trans. Graphics, 21 (2002), pp. 362–371.
[24] J. G. Lewis, B. W. Peyton, and A. Pothen, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1146–1173.
[25] Z. Li, Y. Saad, and M. Sosonkina, *pARMS: A parallel version of the algebraic recursive multilevel solver*, Numer. Linear Algebra Appl., 10 (2003), pp. 485–509.
[26] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497.
[27] W. Niethammer, J. de Pillis, and R. Varga, *Convergence of block iterative methods applied to sparse least-squares problems*, Linear Algebra Appl., 58 (1984), pp. 327–341.
[28] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.
[29] A. T. Papadopoulos, I. S. Duff, and A. J. Wathen, *A class of incomplete orthogonal factorization methods.* II: *Implementation and results*, BIT, 45 (2005), pp. 159–179.
[30] Y. Saad, *Preconditioning techniques for indefinite and nonsymmetric linear systems*, J. Comput. Appl. Math., 24 (1988), pp. 89–105.
[31] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[32] Y. Saad and M. Sosonkina, *Enhanced Preconditioners for Large Sparse Least Squares Problems*, Tech. Report umsi-2001-1, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.
[33] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.
[34] I. Strandén, S. Tsuruta, and I. Misztal, *Simple preconditioners for the conjugate gradient method: Experience with test day models*, J. Anim. Breed. Genet., 119 (2002), pp. 166–174.
[35] X. Wang, K. A. Gallivan, and R. Bramley, *CIMGS: An incomplete orthogonal factorization preconditioner*, SIAM J. Sci. Comput., 18 (1997), pp. 516–536.