# Computing Rank-Revealing QR Factorizations of Dense Matrices

Christian H. Bischof
Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439. bischof@mcs.anl.gov
and
Gregorio Quintana-Ortí
Departamento de Informática, Universidad Jaime I, Campus Penyeta Roja, 12071 Castellón, Spain. gquintan@inf.uji.es.

---

We develop algorithms and implementations for computing rank-revealing QR factorizations (RRQR factorizations) of dense matrices. First, we develop an efficient block algorithm for approximating an RRQR factorization, employing a windowed version of the commonly used Businger/Golub pivoting strategy, safeguarded by incremental condition estimation. Second, we develop efficiently implementable variants of guaranteed reliable RRQR algorithms for triangular matrices originally suggested by Chandrasekaran and Ipsen and by Pan and Tang. We suggest algorithmic improvements with respect to condition estimation, termination criteria, and Givens updating. By combining the block algorithm with one of the triangular postprocessing steps, we arrive at an efficient and reliable algorithm for computing an RRQR factorization of a dense matrix. Experimental results on IBM RS/6000 and SGI R8000 platforms show that this approach performs up to three times faster than the less reliable QR factorization with column pivoting as it is currently implemented in LAPACK, and comes within 15% of the performance of the LAPACK block algorithm for computing a QR factorization without any column exchanges. Thus, we expect this routine to be useful in many circumstances where numerical rank deficiency cannot be ruled out but currently has been ignored because of the computational cost of dealing with it.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematical Software**]

Additional Key Words and Phrases: rank-revealing orthogonal factorization, numerical rank, block algorithm, QR factorization, least-squares systems

---

*Argonne Preprint ANL-MCS-P559-0196*
*submitted to ACMTransactions on Mathematical Software*

---

This work was supported by the Applied and Computational Mathematics Program, Advanced Research Projects Agency, under contracts DM28E04120 and P-95006.
Bischof was also supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U. S. Department of Energy, under Contract W-31-109-Eng-38.
Quintana also received support through the European ESPRIT Project 9072–GEPPCOM and the Spanish Research Agency CICYT under grant TIC-91-1157-C03-02. During part of this work, Quintana was a research fellow of the Spanish Ministry of Education and Science of the Valencian Government at the Universidad Politécnica de Valencia and a visiting scientist at the Mathematics and Computer Science Division at Argonne National Laboratory.

## 1. INTRODUCTION

We briefly summarize the properties of a *rank-revealing QR factorization* (RRQR factorization). Let $A$ be an $m \times n$ matrix (w.l.o.g. $m \geq n$) with singular values

$$\sigma_1 \geq \sigma_2 \ \geq \ldots \geq \sigma_n \geq 0, \tag{1}$$

and define the numerical rank $r$ of $A$ with respect to a threshold $\tau$ as follows:

$$\frac{\sigma_1}{\sigma_r} \leq \tau < \frac{\sigma_1}{\sigma_{r+1}}.$$

Also, let $A$ have a QR factorization of the form

$$A\,P = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{2}$$

where $P$ is a permutation matrix, $Q$ has orthonormal columns, $R$ is upper triangular and $R_{11}$ is of order $r$. Further, let $\kappa(A)$ denote the two-norm condition number of a matrix $A$. We then say that (2) is an RRQR factorization of $A$ if the following properties are satisfied:

$$\kappa(R_{11}) \approx \sigma_1/\sigma_r \text{ and } \|R_{22}\|_2 = \sigma_{max}(R_{22}) \approx \sigma_{r+1}. \tag{3}$$

Whenever there is a well-determined gap in the singular value spectrum between $\sigma_r$ and $\sigma_{r+1}$, and hence the numerical rank $r$ is well defined, the RRQR factorization (2) reveals the numerical rank of $A$ by having a well-conditioned leading submatrix $R_{11}$ and a trailing submatrix $R_{22}$ of small norm. We also note that the matrix

$$P^T \begin{pmatrix} R_{11}^{-1} R_{12} \\ -I \end{pmatrix},$$

which can be easily computed from (2), is usually a good approximation of the nullvectors, and a few steps of subspace iteration suffice to compute nullvectors that are correct to working precision [Chan and Hansen 1992].

The RRQR factorization is a valuable tool in numerical linear algebra because it provides accurate information about rank and numerical nullspace. Its main use arises in the solution of rank-deficient least-squares problems, for example, in geodesy [Golub et al. 1986], computer-aided design [Grandine 1989], nonlinear least-squares problems [Moré 1978], the solution of integral equations [Eldén and Schreiber 1986], and the calculation of splines [Grandine 1987]. Other applications arise in beamforming [Bischof and Shroff 1992], spectral estimation [Hsieh et al. 1991], and regularization [Hansen 1990; Hansen et al. 1992; Waldén 1991].

Stewart [1990] suggested another alternative to the singular value decomposition (SVD), a complete orthogonal decomposition called URV decomposition. This factorization decomposes

$$A = U \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} V^T,$$

where $U$ and $V$ are orthogonal and both $\|R_{12}\|_2$ and $\|R_{22}\|_2$ are of the order $\sigma_{r+1}$. In particular, compared with RRQR factorizations, URV decompositions employ a general orthogonal matrix $V$ instead of the permutation matrix $P$. URV decompositions are more expensive to compute, but they are well suited for nullspace updating. RRQR factorizations, on the other hand, are more suited for the least-squares setting, since one need not store the orthogonal matrix $V$ (the other orthogonal matrix is usually applied to the right-hand side "on the fly"). Of course, RRQR factorizations can be used to compute an initial URV decomposition, where $U = Q$ and $V = P$.

We briefly review the history of RRQR algorithms. From the interlacing theorem for singular values [Golub and Loan 1983, Corollary 8.3.3], we have

$$\sigma_{min}(R(1:k,1:k)) \leq \sigma_k(A) \text{ and } \sigma_{max}(R(k+1:n,k+1:n)) \geq \sigma_{k+1}(A)\,. \quad (4)$$

Hence, to satisfy condition (3), we need to pursue two tasks:

*Task 1.* Find a permutation $P$ that maximizes $\sigma_{min}(R_{11})$.

*Task 2.* Find a permutation $P$ that minimizes $\sigma_{max}(R_{22})$.

Businger and Golub [1965] suggested what is commonly called the "QR factorization with column pivoting." Given a set of already selected columns, this algorithm chooses as the next pivot column the one that is "farthest away" in the Euclidean norm from the subspace spanned by the columns already chosen [Golub and Loan 1983, p.168, P.6.4-5]. This intuitive strategy addresses task 1.

While this greedy algorithm is known to fail on the so-called Kahan matrices [Golub and Loan 1989, p. 245, Example 5.5.1], it works well in practice and forms the basis of the LINPACK [Dongarra et al. 1979] and LAPACK [Anderson et al. 1992a; Anderson et al. 1994b] implementations. Recently, Quintana-Ortí, Sun, and Bischof [1995] developed an implementation of the Businger/Golub algorithm that allows half of the work to be performed with BLAS-3 kernels. Bischof also had developed restricted-pivoting variants of the Businger/Golub strategy to enable the use of BLAS-3 type kernels [1989] for almost all of the work and to reduce communication cost on distributed-memory machines [1991].

One approach to task-2 is based, in essence, on the following fact, which is proved in [Chan and Hansen 1992].

LEMMA 1. *For any $R \in \mathbb{R}^{n\times n}$ and any $W = \begin{pmatrix} W_1 \\ W_2 \end{pmatrix} \in \mathbb{R}^{n\times p}$ with a nonsingular $W_2 \in \mathbb{R}^{p\times p}$, we have*

$$\|R(n-p+1:n, n-p+1:n)\|_2 \leq \|R\,W\|_2\,\|W_2^{-1}\|_2. \quad (5)$$

This means that if we can determine a matrix $W$ with $p$ linearly independent columns, all of which lie approximately in the nullspace of $R$ (i.e., $\|R\,W\|_2$ is small), and if $W_2$ is well conditioned such that $(\sigma_{\min}(W_2))^{-1} = \|W_2^{-1}\|_2$ is not large, then we are guaranteed that the elements of the bottom right $p \times p$ block of $R$ will be small.

Algorithms based on computing well-conditioned nullspace bases for $A$ include these by Golub, Klema, and Stewart [1976], Chan [1987], and Foster [1986]. Other algorithms addressing task-2 are these by Stewart [1984] and Gragg and Stewart [1976]. Algorithms addressing task 1 include those of Chan and Hansen [1994] and Golub, Klema, and Stewart [1976]. In fact, the latter achieves both task 1 and task 2 and, therefore, reveals the rank, but it is too expensive in comparison with the others.

Bischof and Hansen combined a restricted-pivoting strategy with Chan's algorithm [Chan 1987] to arrive at an algorithm for sparse matrices [Bischof and Hansen

1991] and also developed a block-variant of Chan's algorithm [Bischof and Hansen 1992]. A Fortran 77 implementation of Chan's algorithm was provided by Reichel and Gragg [1990].

Chan's algorithm [Chan 1987] guaranteed

$$\frac{\sigma_i}{\sqrt{n(n-i+1)2^{n-i}}} \leq \sigma_{min}(R(1:i,1:i)) \leq \sigma_i \tag{6}$$

and

$$\sigma_i \leq \sigma_{max}(R(i:n,i:n)) \leq \sigma_i \sqrt{n(n-i+1)2^{n-i}}\,. \tag{7}$$

That is, as long as the rank of the matrix is close to $n$, the algorithm is guaranteed to produce reliable bounds, but reliability may decrease with the rank of the matrix.

Hong and Pan [1992] then showed that there exists a permutation matrix $P$ such that for the triangular factor $R$ partitioned as in (2) we have

$$||R_{22}||_2 \leq \sigma_{r+1}(A) p_1(r,n) \tag{8}$$

and

$$\sigma_{\min}(R_{11}) \geq \sigma_r(A)\,\frac{1}{p_2(r,n)}, \tag{9}$$

where $p_1$ and $p_2$ are low-order polynomials in $n$ and $r$ (versus an exponential factor in Chan's algorithm).

Chandrasekaran and Ipsen [1994] were the first to develop RRQR algorithms that satisfy (8) and (9). Their paper also reviews and provides a common framework for the previously devised strategies. In particular, they introduce the so-called unification principle, which says that running a task-1 algorithm on the rows of the inverse of the matrix yields a task-2 algorithm. They suggest hybrid algorithms that alternate between task-1 and task-2 steps to refine the separation of the singular values of $R$.

Pan and Tang [1992] and Gu and Eisenstat [1992] presented different classes of algorithms for achieving (8) and (9), addressing the possibility of nontermination of the algorithms because of floating-point inaccuracies.

The goal of our work was to develop an efficient and reliable RRQR algorithm and implementation suitable for inclusion in a numerical library such as LAPACK. Specifically, we wished to develop an implementation that was both reliable and close in performance to the QR factorization without any pivoting. Such an implementation would provide algorithm developers with an efficient tool for addressing potential numerical rank deficiency by minimizing the computational penalty for addressing *potential* rank deficiency. Our strategy involves the following ingredients:

—An efficient block-algorithm for computing an approximate RRQR factorization based on the work by Bischof [1989], and

—efficient implementations of RRQR algorithms well suited for triangular matrices based on the work by Chandrasekaran and Ipsen [1994] and Pan and Tang [1992]. These algorithms seemed better suited for triangular matrices than those suggested by Gu and Eisenstat [1992].

We expect that

$P = I$;
**foreach** $i \in \{1, \ldots, n\}$ **do** $res_i = \|a(:,i)\|_2$ **end do**
**for** $i = 1$ **to** $\min(m,n)$ **do**
  Let $i \le pvt \le n$ be such that $res_{pvt}$ is maximal
  $P(:,i) \leftrightarrow P(:,pvt)$ ; $a(:,i) \leftrightarrow a(:,pvt)$ ; $res_{pvt} := res_i$ ;
  $[u_i, a(i\!:\!m\,,i)] := genhh(a(i\!:\!m\,,i))$ ;
  $a(i\!:\!m\,,i\!+\!1\!:\!n) := apphh(u_i\,, a(i\!:\!m\,,i\!+\!1\!:\!n))$ ;
  **foreach** $j \in \{i+1, \ldots, n\}$ **do**
    $res_j := \sqrt{res_j^2 - a(i,j)^2}$ ;
  **end foreach**
**end for**

Fig. 1. The QR Factorization Algorithm with Traditional Column Pivoting

—in most cases the approximate RRQR factorization computed by the block algorithm is very close to the desired RRQR factorization, requiring little postprocessing, and

—the almost entirely BLAS-3 based preprocessing algorithm performs considerably faster than the QR factorization with column pivoting and close to the performance of the QR factorization without pivoting.

The paper is structured as follows. In the next section, we review the block algorithm for computing an approximate RRQR factorization based on a restricted-pivoting approach. In Section 3, we describe our modifications to Chandrasekaran and Ipsen's "Hybrid-III" algorithm and Pan and Tang's "Algorithm 3." Section 4 presents our experimental results on IBM RS/6000, and SGI R8000 platforms. In Section 5, we summarize our results.

## 2. A BLOCK QR FACTORIZATION WITH RESTRICTED PIVOTING

In this section, we describe a block QR factorization algorithm which employs a restricted pivoting strategy to approximately compute a RRQR factorization, employing the ideas described in Bischof [1989].

We compute $Q$ by a sequence of Householder matrices

$$H \equiv H(u) = I - 2uu^T \;, \|u\|_2 = 1. \tag{10}$$

For any given vector $x$, we can choose a vector $u$ so that $H(u)x = \alpha e_1$, where $e_1$ is the first canonical unit vector and $\mid \alpha \mid = \|x\|_2$ (see, for example,[Golub and Loan 1989, p. 196]). The application of a Householder matrix $B := H(u)A$ involves a matrix-vector product $z := A^T u$ and a rank-one update $B := A - 2uz^T$.

Figure 1 describes the Businger/Golub Householder QR factorization algorithm with traditional column pivoting [Businger and Golub 1965] for computing the QR decomposition of an $m \times n$ matrix $A$. The primitive operation $[u, y] := genhh(x)$ computes $u$ such that $y = H(u)x$ is a multiple of $e_1$, while the primitive operation $B := apphh(u, A)$ overwrites $B$ with $H(u)A$.

After step $i$ is completed, the values $res_j, j = i+1, \ldots, n$ are the length of the projections of the $j$th column of the currently permuted $AP$ onto the orthogonal complement of the subspace spanned by the first $i$ columns of $AP$. The values $res_j$ can be updated easily and do not have to be recomputed at every step, although

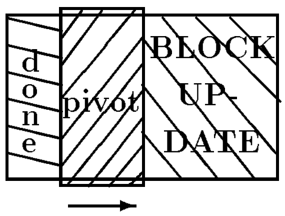


Fig. 2. Restricting Pivoting for a Block Algorithm

roundoff errors may make it necessary to recompute $res_j = \|(a(i:m,j))\|_2, j = i+1,\ldots,n$ periodically [Dongarra et al. 1979, p. 9.17] (we suppressed this detail in line 9 of Figure 1).

The bulk of the computational work in this algorithm is performed in the *apphh* kernel, which relies on matrix-vector operations. However, on today's cache-based architectures (ranging from workstations to supercomputers) matrix-matrix operations perform much better. Matrix-matrix operations are exploited by using so-called block algorithms, whose top-level unit of computation is matrix blocks instead of vectors. Such algorithms play a central role, for example, in the LAPACK implementations [Anderson et al. 1992a; Anderson et al. 1994b]. LAPACK employs the so-called compact WY representation of products of Householder matrices [Schreiber and Van Loan 1989], which expresses the product

$$Q = H_1 \cdots H_{nb}$$

of a series of $m \times m$ Householder matrices (10) as

$$Q = I + YTY^T\,, \tag{11}$$

where $Y$ is an $m \times nb$ matrix and $T$ is an $nb \times nb$ upper triangular matrix. Stable implementations for generating Householder vectors as well as forming and applying compact WY factors are provided in LAPACK.

To arrive at a block QR factorization algorithm, we would like to avoid updating part of $A$ until several Householder transformations have been computed. This strategy is impossible with traditional pivoting, since we must update $res_j$ before we can choose the next pivot column. While we can modify the traditional approach to do half of the work using block transformations, this is the best we can do (these issues are discussed in detail in [Quintana-Ortí et al. 1995]). Therefore, we instead limit the scope of pivoting as suggested in [Bischof 1989], Thus, we do not have to update the remaining columns until we have computed enough Householder transformations to make a block update worthwhile.

The idea is graphically depicted in Figure 2. At a given stage we are done with the columns to the left of the pivot window. We then try to select the next pivot columns exclusively from the columns in the pivot window, not touching the part of $A$ to the right of the pivot window. Only when we have combined the Householder vectors defined by the next batch of pivot columns into a compact WY factor, do we apply this block update to the columns on the right.

Since the leading block of $R$ is supposed to approximate the large singular values of $A$, we must be able to guard against pivot columns that are close to the span of columns already selected. That is, given the upper triangular matrix $R_i$ defined by the first $i$ columns of $Q^T AP$ and a new column $\begin{pmatrix} v \\ \gamma \end{pmatrix}$ determined by the new candidate pivot column, we must determine whether

$$R_{i+1} = \begin{pmatrix} R_i & v \\ 0 & \gamma \end{pmatrix}$$

has a condition number that is larger than a threshold $\tau$, which defines what we consider a rank-deficient matrix.

We approximate

$$\sigma_{max}(R_{i+1}) \approx \widehat{\sigma}_{max}(R_{i+1}) \equiv n^{\frac{1}{3}} \max_{1 \leq k \leq i} \|R(1:k,k)\|_2 \tag{12}$$

which is easy to compute. To cheaply estimate $\sigma_{min}(R_{i+1})$, we employ incremental condition estimation (ICE) [Bischof 1990; Bischof and Tang 1991]. Given a good estimate $\widehat{\sigma}_{min}(R_i) = 1/\|x\|_2$ defined by a large norm solution $x$ to $R_i^T x = d$, $\|d\|_2 = 1$ and a new column $\begin{pmatrix} v \\ \gamma \end{pmatrix}$, incremental condition estimation, with only $3k$ flops, computes $s$ and $c$, $s^2 + c^2 = 1$, such that

$$\sigma_{min}(R_{i+1}) \approx \widehat{\sigma}_{min}(R_{i+1}) = 1/\|\begin{pmatrix} sx \\ c \end{pmatrix}\|_2 \,. \tag{13}$$

A stable implementation of ICE based on the formulation in [Bischof and Tang 1991] is provided by the LAPACK routine xLAIC1.[1] ICE is an order of magnitude cheaper than other condition estimators (see, for example, [Higham 1986]). Moreover, it is considerably more reliable than simply using $\mid \gamma \mid$ as an estimate for $\sigma_{min}(R_{i+1})$ (see, for example, [Bischof 1991]). We also define

$$\widehat{\kappa}(R_i) \equiv \frac{\widehat{\sigma}_{max}(R_i)}{\widehat{\sigma}_{min}(R_i)} \,. \tag{14}$$

The restricted block pivoting algorithm proceeds in four phases:

*Phase 1: Pivoting of largest column into first position.* This step is motivated by the fact that the norm of the largest column of $A$ is usually a good estimate for $\sigma_1(A)$.

*Phase 2: Block QR factorization with restricted pivoting.* Given a desired block size $nb$ and a window size $ws$, $ws \geq nb$, we try to generate $nb$ Householder transformations by applying the Businger/Golub pivoting strategy *only to the columns in the pivot window*, using ICE to assess the impact of a column selection on the condition number via ICE. When the pivot column chosen from the pivot window would lead to a leading triangular factor whose condition number exceeds $\tau$, we mark *all* remaining columns in the pivot window ($k$, say) as "rejected," pivot them to the end of the matrix, generate a block transformation (of width not more than $nb$), apply it to the remainder of the matrix, and then reposition the pivot window

[1] Here as in the sequel we use the convention that the prefix "x" generically refers to the appropriate one of the four different precision instantiations: SLAIC1, DLAIC1, CLAIC1, or ZLAIC1.

to encompass the next $ws$ not yet rejected columns. When all columns have been either accepted as part of the leading triangular factor or rejected at some stage of the algorithm, this phase stops.

Assuming we have included $r_2$ columns in the leading triangular factor, we have at this point computed an $r_2 \times r_2$ upper triangular matrix $R_{r_2} = R(1:r_2, 1:r_2)$ that satisfies

$$\widehat{\kappa}(R_{r_2}) \leq \tau \,. \tag{15}$$

That is, $r_2$ is our estimate of the numerical rank with respect to the threshold $\tau$ at this point.

In our experiments, we chose

$$ws = nb + \max\{10, \frac{nb}{2} + 0.05n\} \,. \tag{16}$$

This choice tries to ensure a suitable pivot window and "loosens up" a bit as matrix size increases. A pivot window that is too large will reduce performance because of the overhead in generating block orthogonal transformations and the larger number of unblocked operations. On the other hand, a pivot window that is too small will reduce the pivoting flexibility and thus increase the likelihood that the restricted pivoting strategy will fail to produce a good approximate RRQR factorization. In our experiments, the choice of $w$ had only a small impact (not more than 5%) on overall performance and negligible impact on the numerical behavior.

*Phase 3: Traditional pivoting strategy among "rejected" columns.* Since phase 2 rejects all remaining columns in the pivot window when the pivot candidate is rejected, a column may have been pivoted to the end that should not have been rejected. Hence, we now continue with the traditional Businger/Golub pivoting strategy on the remaining $n - r_2$ columns, updating (14) as an estimate of the condition number. This phase ends at column $r_3$, say, where

$$\widehat{\kappa}(R_{r_3}) \leq \tau \,, \tag{17}$$

and the inclusion of the next pivot column would have pushed the condition number beyond the threshold. We do not expect many columns (if any) to be selected in this phase. It is mainly intended as a cheap safeguard against possible failure of the initial restricted-pivoting strategy.

*Phase 4: Block QR factorization without pivoting on remaining columns.* The columns not yet factored (columns $r_3 + 1 : n$) are with great probability linearly dependent on the previous ones, since they have been rejected in both phase 2 and phase 3. Hence, it is unlikely that any kind of column exchanges among the remaining columns would change our rank estimate, and the standard BLAS-3 block QR factorization as implemented in the LAPACK routine xGEQRF is the fastest way to complete the triangularization.

After the completion of phase 4, we have computed a QR factorization Kthat satisfies

$$\widehat{\kappa}(R_{r_3}) \leq \tau,$$

and for any column $y$ in $R(:, r_3+1:n)$ we have

$$\widehat{\kappa}(\begin{pmatrix} R_{r_3} \\ 0 \end{pmatrix}, y) > \tau.$$

This result suggests that this QR factorization is a good approximation to a RRQR factorization and $r_3$ is a good estimate of the rank.

However, this $QR$ factorization *does not guarantee* to reveal the numerical rank correctly. Thus, we back up this algorithm with the guaranteed reliable RRQR implementations introduced in the next two sections.

## 3. POSTPROCESSING ALGORITHMS FOR AN APPROXIMATE RRQR FACTORIZATION

In 1991, Chandrasekaran and Ipsen [1994] introduced a unified framework for RRQR algorithms and developed an algorithm guaranteed to satisfy (8) and (9) and thus to properly reveal the rank. Their algorithm assumes that the initial matrix is triangular and thus is well suited as a postprocessing step to the algorithm presented in the prexeding section. Shortly thereafter, Pan and Tang [1992] introduced another guaranteed reliable RRQR algorithm for triangular matrices. In the following subsections, we describe our improvements and implementations of these algorithms.

### 3.1 The RRQR Algorithm by Pan and Tang

We implement a variant of what Pan and Tang [1992] call "Algorithm 3." Pseudocode for our algorithm is shown in Figure 3. It assumes as input an upper triangular matrix $R$. $\Pi^R(i,j), i<j$, denotes a right cyclic permutation that exchanges columns $i$ and $j$, e.g., $i \rightarrow i+1, \ldots, j-1 \rightarrow j, j \rightarrow i$, whereas $\Pi^L_{i,j}, i<j$ denotes a left cyclic permutation that exchanges columns $i$ and $j$, i.e., $j \leftarrow i, i \leftarrow i+1, \ldots, j-1 \leftarrow j$. In the algorithm, $\mathrm{triu}(A)$ denotes the upper triangular factor $R$ in a QR factorization $A = QR$ of $A$. As can be seen from Figure 3, we use this notation as shorthand for retriangularizations of $R$ after column exchanges.

Given a value for $k$, and a so-called $f$-factor $0 < f \leq 1/\sqrt{k+1}$, the algorithm is guaranteed to halt and produce a triangular factorization that satisfies

$$\sigma_{\min}(R_{11}) \geq \frac{f}{\sqrt{k(n-k+1)}}\sigma_k(A) \tag{18}$$

$$\sigma_{\max}(R_{22}) \leq \frac{\sqrt{(k+1)(n-k)}}{f}\sigma_{k+1}(A)\,. \tag{19}$$

Our implementation incorporates the following features:

(1) Incremental condition estimation (ICE) is used to arrive at estimates for smallest singular values and vectors. Thus, $\sigma$ (line 5) and $v$ (line 9) of Figure 3 can be computed inexpensively from $u$ (line 2). The use of ICE significantly reduces implementation cost.

(2) The QR factorization update (line 4) must be performed only when the if-test (line 6) is false. Thus, we delay it if possible.

(3) For the algorithm to terminate, all columns need to be checked, and no new permutations must occur. In Pan and Tang's algorithm, rechecking of columns

Algorithm PT3M(f,k)

$i = k+1$; accepted_col $= 0$; $\Pi = I$;
$u$ := left singular vector corresponding to $\sigma_{\min}(R(1{:}k, 1{:}k))$
**while** ( accepted_col $\leq n-k$ ) **do**
  $R := \mathrm{triu}(R \cdot \Pi^R_{k+1,i})$ ; $\Pi := \Pi \cdot \Pi^R_{k+1,i}$
  Compute $\sigma \approx \sigma_{\min}(R(1{:}k+1, 1{:}k+1))$
  **if** $(\sigma \geq f \cdot |R(k+1,k+1)|)$ **then**
    accepted_col := accepted_col + 1;
  **else**
    $v$ := right singular vector corresponding to $\sigma$
    Find index $q$, $1 \leq q \leq k+1$, such that: $|v_q| = \max_j |v_j|$
    $R := \mathrm{triu}(R \cdot \Pi^L_{q,k+1})$ ; $\Pi := \Pi \cdot \Pi^L_{q,k+1}$; accepted_col = 0;
    $u$ := left singular vector corresponding to $\sigma_{\min}(R(1{:}k, 1{:}k))$
  **end if**
  **if** $(i == n)$ **then** $i := k+1$ **else** $i := i+1$ **end if**
**end while**

Fig. 3. Variant of Pan/Tang RRQR Algorithm

after a permutation *always* starts at column $k+1$. We instead begin checking at the column right after the one that just caused a permutation. Thus, we first concentrate on the columns that have not just been "worked over."

(4) The left cyclic shift permutes the triangular matrix into an upper Hessenberg form, which is then retriangularized with Givens rotations. Applying Givens rotations to rows of $R$ in the obvious fashion (as done, for example, in [Reichel and Gragg 1990]) is expensive in terms of data movement, because of the column-oriented nature of Fortran data layout. Thus, we apply Givens rotations in an aggregated fashion, updating matrix strips $(R(1:jb, (j-1)b+1:jb))$ of width $b$ with *all* previously computed Givens rotations.

Similarly, the right cyclic shift introduces a "spike" in column $j$, which is eliminated with Givens rotations in a bottom-up fashion. To aggregate Givens rotations, we first compute all rotations only touching the "spike" and the diagonal of $R$, and then apply all of them one block column at a time. In our experiments, we choose the width $b$ of the matrix strips to be the same as the blocksize $nb$ of the preprocessing.

Compared with a straightforward implementation of Pan and Tang's "Algorithm 3," improvements (1) through (3) on average decreased runtime by a factor of five on $200 \times 200$ matrices on an Alliant FX/80. When retriangularizations were frequent, improvement (4) had the most noticeable impact, resulting in a twofold to fourfold performance gain on matrices of order 500 and 1000 on an IBM RS/6000-370.

Pan and Tang introduced the $f$-factor to prevent cycling of the algorithm. The higher $f$ is, the tighter the bounds in (18) and (19), and the better the approximations to the $k$ and $k+1$st singular values of $R$. However, if $f$ is too large, it introduces more column exchanges and therefore more iterations, and, because of round-off errors, it might present convergence problems. We used $f = 0.9/\sqrt{k+1}$ in our work.

Algorithm Hybrid-III-sf(f,k)

1. $\Pi = I$
2. **repeat**
3. Golub-I-sf(f,k)
4. Golub-I-sf(f,k+1)
5. Chan-II-sf(f,k+1)
6. Chan-II-sf(f,k)
7. **until** none of the four subalgorithms modified the column ordering

Fig. 4. Variant of Chandrasekaran/Ipsen Hybrid-III algorithm

Algorithm Golub-I-sf(f,k)

1. Find smallest index $j$, $k \leq j \leq n$, such that
2. $\|R(k{:}j, j)\|_2 = \max_{k \leq i \leq n} \|R(k{:}i, i)\|_2$
3. **if** $f \cdot \|R(k{:}j, j)\|_2 > | R(k,k) |$ **then**
4. $R := \mathrm{triu}(R \cdot \Pi^R_{k,j}); \Pi := \Pi \cdot \Pi^R_{k,j}$
5. **end if**

Fig. 5. "f-factor" Variant of Golub-I Algorithm

### 3.2 The RRQR Algorithm by Chandrasekaran and Ipsen

Chandrasekaran and Ipsen introduced algorithms that achieve bounds (18) and (19) with $f = 1$. We implemented a variant of the so-called Hybrid-III algorithm, pseudocode for which is shown in Figures 4 – 6.

Compared with the original Hybrid-III algorithm, our implementation incorporates the following features:

(1) We employ the Chan-II strategy (an $O(n^2)$ algorithm) instead of the so-called Stewart-II strategy (an $O(n^3)$ algorithm because of the need for the inversion of $R(1:k, 1:k)$) that Ipsen and Chandrasekaran employed in their experiments.

(2) The original Hybrid-III algorithm contained two subloops, with the first one looping over Golub-I(k) and Chan-II(k) till convergence, the second one looping over Golub-I(k+1) and Chan-II(k+1). We present a different loop ordering in our variant, one that is simpler and seems to enhance convergence. On matrices that required considerable postprocessing, the new loop ordering required about 7% less steps for $1000 \times 1000$ matrices (one step being a call to Golub-I or Chan-II) than Chandrasekaran and Ipsen's original algorithm. In addition, the new ordering speeds up detection of convergence, as shown below.

Algorithm Chan-II-sf(k)

1. $v$ := right singular vector corresponding to $\sigma_{min}(R(1{:}k, 1{:}k))$.
2. Find largest index $j$, $1 \leq j \leq k$, such that: $|v_j| = \max_{1 \leq i \leq k} |v_i|$
3. **if** $f \cdot |v_j| > |v_k|$ **then**
4. $R := \mathrm{triu}(R \cdot \Pi^L_{j,k}); \Pi := \Pi \cdot \Pi^L_{j,k}$
5. **end if**

Fig. 6. "f-factor" Variant of Chan-II Algorithm

(3) As in our implementation of the Pan/Tang algorithm, we use ICE for estimating singular values and vectors, and the efficient "aggregated" Givens scheme for the retriangularizations.

(4) We employ a generalization of the $f$-factor technique to guarantee termination in the presence of rounding errors. The pivoting method assigns to every column a "weight," namely, $\|R(k\colon i, i)\|_2$ in Golub-I(k) and $v_i$ in Chan-II(k), where $v$ is the right singular vector corresponding to the smallest singular value of $R(1\colon k, 1\colon k)$. To ensure termination, Chandrasekaran and Ipsen suggested pivoting a column only when its weight exceeded that of the current column by at least $n^2\epsilon$, where $\epsilon$ is the computer precision; they did not analyze the impact of this change on the bounds obtained by the algorithm. In contrast, we use a multiplicative tolerance factor $f$ like Pan and Tang; the analysis in [Quintana-Ortí and Quintana-Ortí 1996] proves that our algorithm achieves the bounds

$$\sigma_{\min}(R_{11}) \geq \frac{f^2}{\sqrt{k(n-k+1)}}\sigma_k(A), \text{ and} \tag{20}$$

$$\sigma_{\max}(R_{22}) \leq \frac{\sqrt{(k+1)(n-k)}}{f^2}\sigma_{k+1}(A) . \tag{21}$$

These bounds are identical to (18) and (19), except that an $f^2$ instead of an $f$ enters into the equation and that now $0 < f \leq 1$. We used $f = 0.5$ in our implementation.

We claimed before that the new loop ordering can avoid unnecessary steps when the algorithm is about to terminate. To illustrate, consider the situation where we apply Chandrasekaran and Ipsen's original ordering to a matrix that almost reveals the rank:

| | | |
|---|---|---|
| 1. | Golub-I(k) | Final permutation occurs here.<br>Now the rank is revealed. |
| 2. | Chan-II(k) | |
| 3. | Golub-I(k) | Another iteration of inner k-loop<br>since permutation occurred. |
| 4. | Chan-II(k) | |
| 5. | Golub-I(k+1) | Inner loop for $k+1$ |
| 6. | Chan-II(k+1) | |
| 7. | Golub-I(k) | Another iteration of the main loop<br>since permutation occurred in last pass. |
| 8. | Chan-II(k) | |
| 9. | Golub-I(k+1) | |
| 10. | Chan-II(k+1) | Termination |

In contrast, the Hybrid-III-sf algorithm terminates in four steps:

| | | |
|---|---|---|
| 1. | Golub-I-sf(k) | Final permutation |
| 2. | Golub-I-sf(k+1) | |
| 3. | Chan-II-sf(k+1) | |
| 4. | Chan-II-sf(k) | Termination |

Algorithm RRQR(f,k)

**repeat**
  **call** Hybrid-III-sf(f,k) or PT3M(f,k)
  $\alpha := \kappa(R(1{:}\,k, 1{:}\,k))$
  $\beta := \kappa(R(1{:}\,k+1, 1{:}\,k+1))$
  **if** (( $\alpha \le \tau$ ) **and** ( $\beta > \tau$ )) **then**
    rank := $k$; **stop**
  **else if** ( ( $\alpha \le \tau$ ) **and** ($\beta \le \tau$ ) )**then**
    $k := k + 1$
  **else if** ( ( $\alpha \ge \tau$ ) **and** ( $\beta \ge \tau$ ) )**then**
    $k := k - 1$
  **end if**

Fig. 7. Algorithm for Computing Rank-Revealing QR Factorization

### 3.3 Determining the Numerical Rank

As Stewart [1993] pointed out, both the Chandrasekaran/Ipsen and Pan/Tang algorithms, as well as our versions of those algorithms, do not reveal the rank of a matrix per se. Rather, given an integer $k$, they compute tight estimates for $\sigma_k(A) \approx \sigma_{\min}(R(1{:}\,k, 1{:}\,k))$ and $\sigma_{k+1}(A) \approx \sigma_{\max}(R(k+1{:}\,n, k+1{:}\,n))$.

To obtain the numerical rank with respect to a given threshold $\tau$, given an initial estimate for the rank (as provided, for example, by the algorithm described in Section 2), we employ the algorithm shown in Figure 7. In our actual implementation, $\alpha$ and $\beta$ are computed in Hybrid-III-sf or PT3M.

## 4. EXPERIMENTAL RESULTS

We report in this section experimental results with the double-precision implementations of the algorithms presented in the preceding section. We consider the following codes:

*DGEQPF.* The implementation of the QR factorization with column pivoting currently provided in LAPACK.

*DGEQPB.* An implementation of the "windowed" QR factorization scheme described in Section 2.

*DGEQPX.* DGEQPB followed by an implementation of the variant of the Chandrasekaran/Ipsen algorithm described in subsections 3.2 and 3.3.

*DGEQPY.* DGEQPB followed by an implementation of the variant of the Pan/Tang algorithm described in subsections 3.1 and 3.3.

*DGEQRF.* The block QR factorization without any pivoting provided in LAPACK.

In the implementation of our algorithms, we make heavy use of available LAPACK infrastructure. The code used in our experiments, including test and timing drivers and test matrix generators, is available as `rrqr.tar.gz` in `pub/prism` on `ftp.super.org`.

We tested matrices of size $100, 150, 250, 500$, and $1000$ on an IBM RS/6000 Model 370 and SGI R8000. In each case, we employed the vendor-supplied BLAS in the ESSL and SGIMATH libraries, respectively.

### 4.1 Numerical Reliability

We employed 18 different matrix types to test the algorithms, with various singular value distributions and numerical rank ranging from 3 to full rank. Details of the test matrix generation are beyond the scope of this paper, and we give only a brief synopsis here. For details, the reader is referred to the code.

Test matrices 1 through 5 were designed to exercise column pivoting. Matrix 6 was designed to test the behavior of the condition estimation in the presence of clusters for the smallest singular value. For the other cases, we employed the LAPACK matrix generator `xLATMS`, which generates random symmetric matrices by multiplying a diagonal matrix with prescribed singular values by random orthogonal matrices from the left and right. For the break1 distribution, all singular values are 1.0 except for one. In the arithmetic and geometric distributions, they decay from 1.0 to a specified smallest singular value in an arithmetic and geometric fashion, respectively. In the "reversed" distributions, the order of the diagonal entries was reversed. For test cases 7 though 12, we used xLATMS to generate a matrix of order $\frac{n}{2}+1$ with smallest singular value 5.0e-4, and then interspersed random linear combinations of these "full-rank" columns to pad the matrix to order $n$. For test cases 13 through 18, we used `xLATMS` to generate matrices of order $n$ with the smallest singular value being 2.0e-7. We believe this set to be representative of matrices that can be encountered in practice.

We report in this section on results for matrices of size $n = 1000$, noting that identical qualitative behavior was observed for smaller matrix sizes. We decided to report on the largest matrix sizes because the possibility for failure in general increases with the number of numerical steps involved. Numerical results obtained on the three platforms agreed to machine precision. For this case, we list in Table 1 the numerical rank $r$ with respect to a condition threshold of $\tau = 1.0e5$, the largest singular value $\sigma_{max}$, the $r$-th singular value $\sigma_r$, the $(r+1)$st singular value $\sigma_{r+1}$, and the smallest singular value $\sigma_{\min}$ for our test cases.

Figures 8 and 9 display the ratio

$$\Theta := \frac{(\sigma_1/\sigma_r)}{\widehat{\kappa}(R_r)}, \tag{22}$$

where $\widehat{\kappa}(R)$ as defined in (14) is the computed estimate of the condition number of $R$ after DGEQPB (Figure 8) and DGEQPX and DGEQPY (Figure 9). Thus, $\Theta$ is the ratio between the ideal condition number and the estimate of the condition number of the leading triangular factor identified in the RRQR factorization. If this ratio is close to 1, and $\widehat{\kappa}$ is a good condition estimate, our RRQR factorizations do a good job of capturing the "large" singular values of $A$. Since the pivoting strategy and hence the numerical behavior of DGEQPB is potentially affected by the block size chosen, Figures 8 and 9 contain seven panels, each of which shows the results obtained with the 18 test matrices and a block size ranging from 1 to 24 (shown in the top of each panel).

We see that except for matrix type 1 in Figure 8, the block size does not play much of a rule numerically, although close inspection reveals subtle variations in both Figure 8 and 9. With block size 1, DGEQPB just becomes the standard Businger/Golub pivoting strategy. Thus, the first panel in Figure 8 corroborates the experimentally robust behavior of this algorithm. We also see that except for

Table 1. Test Matrix Types ($r$ = rank for $n = 1000$):

| | Description | $r$ | $\sigma_{\max}$ | $\sigma_r$ | $\sigma_{r+1}$ | $\sigma_{\min}$ |
|---|---|---|---|---|---|---|
| 1 | Matrix with rank $\frac{min(m,n)}{2} - 1$ | 499 | 1.0e0 | 1.0e0 | 2.0e-7 | 1.2e-19 |
| 2 | $A(:, 2 : \min(m, n))$ has full rank<br>$\mathcal{R}(A) = \mathcal{R}(A(:, 2 : \min(m, n)))$ | 999 | 1.0e0 | 5.0e-4 | 6.7e-19 | 6.7e-19 |
| 3 | Full rank | 1000 | 1.0e0 | 5.0e-4 | 5.0e-4 | 5.0e-4 |
| 4 | $A(:, 1 : 3)$ small in norm<br>$A(:, 4 : n)$ of full rank | 997 | 2.9e+1 | 5.0e-4 | 2.4e-4 | 4.2e-5 |
| 5 | $A(:, 1 : 3)$ small in norm<br>$\mathcal{R}(A) = \mathcal{R}(A(:, 1 : 3))$ | 3 | 1.0e0 | 5.0e-4 | 5.5e-14 | 7.6e-21 |
| 6 | 5 smallest sing. values clustered | 1000 | 1.0e0 | 7.0e-4 | 7.0e-4-3 | 7.0e-4 |
| 7 | Break1 distribution | 501 | 1.0e0 | 5.0e-4 | 1.7e-15 | 1.0e-26 |
| 8 | Reversed break1 distribution | 501 | 1.0e0 | 5.0e-4 | 1.7e-15 | 1.2e-27 |
| 9 | Geometric distribution | 501 | 1.0e0 | 5.0e-4 | 3.3e-16 | 1.9e-35 |
| 10 | Reversed geometric distribution | 501 | 1.0e0 | 5.0e-4 | 3.2e-16 | 5.4e-35 |
| 11 | Arithmetic distribution | 501 | 1.0e0 | 5.0e-4 | 9.7e-16 | 1.4e-34 |
| 12 | Reversed arithmetic distribution | 501 | 1.0e0 | 5.0e-4 | 9.7e-16 | 1.2e-34 |
| 13 | Break1 distribution | 999 | 1.0e0 | 1.0e0 | 2.0e-7 | 2.0e-7 |
| 14 | Reversed break1 distribution | 999 | 1.0e0 | 1.0e0 | 2.0e-7 | 2.0e-7 |
| 15 | Geometric distribution | 746 | 1.0e0 | 5.0e-5 | 9.9e-6 | 2.0e-7 |
| 16 | Reversed geometric distribution | 746 | 1.0e0 | 5.0e-5 | 9.9e-6 | 2.0e-7 |
| 17 | Arithmetic distribution | 999 | 1.0e0 | 1.0e-1 | 2.0e-7 | 2.0e-7 |
| 18 | Reversed arithmetic distribution | 999 | 1.0e0 | 1.0e-1 | 2.0e-7 | 2.0e-7 |

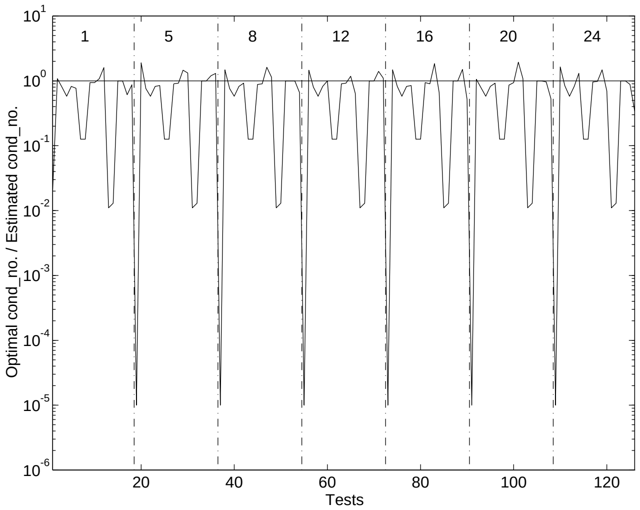


Fig. 8. Ratio between Optimal and Estimated Condition Number for DGEQPB

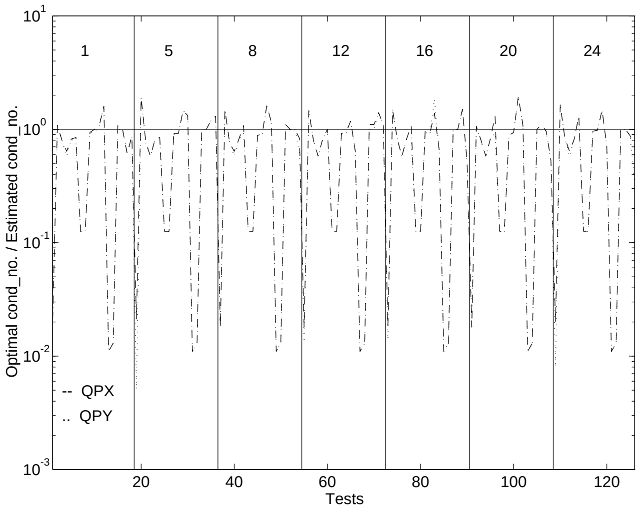


Fig. 9. Ratio between Optimal and Estimated Condition Number for DGEQPX (solid line) and DGEQPY (dashed)

matrix type 1, the restricted pivoting strategy employed in DGEQPB does not have much impact on numerical behavior. For matrix type 1, however, it performs much worse. Matrix 1 is constructed by generating $\frac{n}{2}-1$ independent columns and generating the leading $\frac{n}{2}+1$ as random linear combinations of those columns, scaled by $\epsilon^{\frac{1}{4}}$, where $\epsilon$ is the machine precision. Thus, the restricted pivoting strategy, in its myopic view of the matrix, gets stuck, so to speak, in these columns.

The postprocessing of these approximate RRQR factorizations, on the other hand, remedies potential shortcomings in the preprocessing step. As can be seen from Figure 9, the inaccurate factorization of matrix 1 is corrected, while the other, in essence correct, factorizations get improved only slightly. Except for small variations, DGEQPX and DGEQPY deliver identical results.

We also computed the exact condition number of the leading triangular submatrices identified in the triangularizations by DGEQPB, DGEQPX, and DGEQPY, and compared it with our condition estimate. Figure 10 shows the ratio of the exact condition number to the estimated condition number of the leading triangular factor. We observe excellent agreement, within an order of magnitude in all cases.

Hence, the "spikes" for test matrices 13 and 14 in Figures 8 and 9 are not due to errors in our estimators. Rather, they show that all algorithms have difficulties when confronted with dense clusters of singular values. We also note that in this context, the notion of rank is numerically illdefined, since there is no sensible place to draw the line. The "rank" derived via the SVD is 746 in both cases, and our algorithms deliver estimates between 680 and 710, with minimal changes in the condition number of their corresponding leading triangular factors.

In summary, these results show that DGEQPX and DGEQPY are reliable algorithms for revealing numerical rank. They produce RRQR factorizations whose

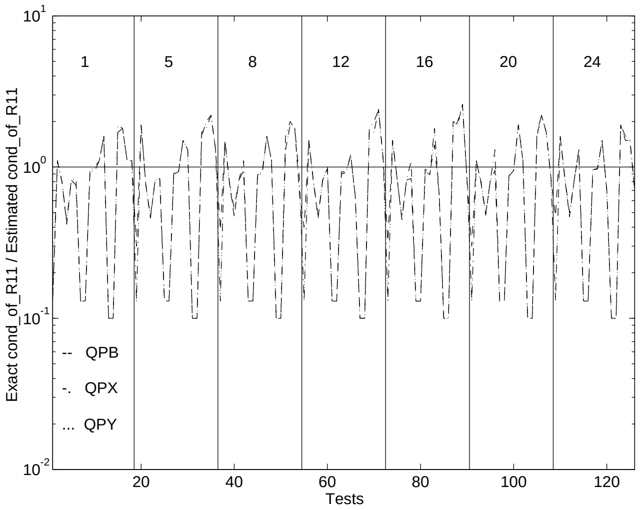


Fig. 10. Ratio between Exact and Estimated Condition Number of Leading Triangular Factor for DGEQPB (dashed), DGEQPX (dashed-dotted) and DGEQPY (dotted)

leading triangular factors accurately capture the desired part of the spectrum of $A$, and thus reliable and numerically sensible rank estimates. Thus, the RRQR factorization takes advantage of the efficiency and simplicity of the QR factorization, yet it produces information that is almost as reliable as that computed by means of the more expensive singular value decomposition.

### 4.2 Computing Performance

In this section we report on the performance of the LAPACK codes DGEQPF and DGEQRF as well as the new DGEQPB, DGEQPX, and DGEQPY codes. For these codes, as well as all others presented in this section, the Mflop rate was obtained by dividing the number of operations required for the unblocked version of DGEQRF by the runtime. This normalized Mflop rate readily allows for timing comparisons. We report on matrix sizes 100, 250, 500, and 1000, using block sizes ($nb$) of 1, 5, 8, 12, 16, 20, and 24.

Figures 11 through 12 show the Mflop performance (averaged over the 18 matrix types) versus block size on the IBM and SGI platforms. The dotted line denotes the performance of **DGEQPF**, the solid one that of **DGEQRF** and the dashed one that of **DGEQPB**; the 'x' and '+' symbols indicate **DGEQPX** and **DGEQPY**, respectively.

On all three machines, the performance of the two new algorithms for computing RRQR is robust with respect to variations in the block size. The two new block algorithms for computing RRQR factorization are, except for small matrices on the SGI, faster than LAPACK's **DGEQPF** for all matrix sizes. We note that the SGI has a data cache of 4 MB, while the IBM platform has only a 32 KB data cache. Thus, matrices up to order 500 fit into the SGI cache, but matrices of order 1000 do not. Therefore, for matrices of size 500 or less we observe limited benefits from the

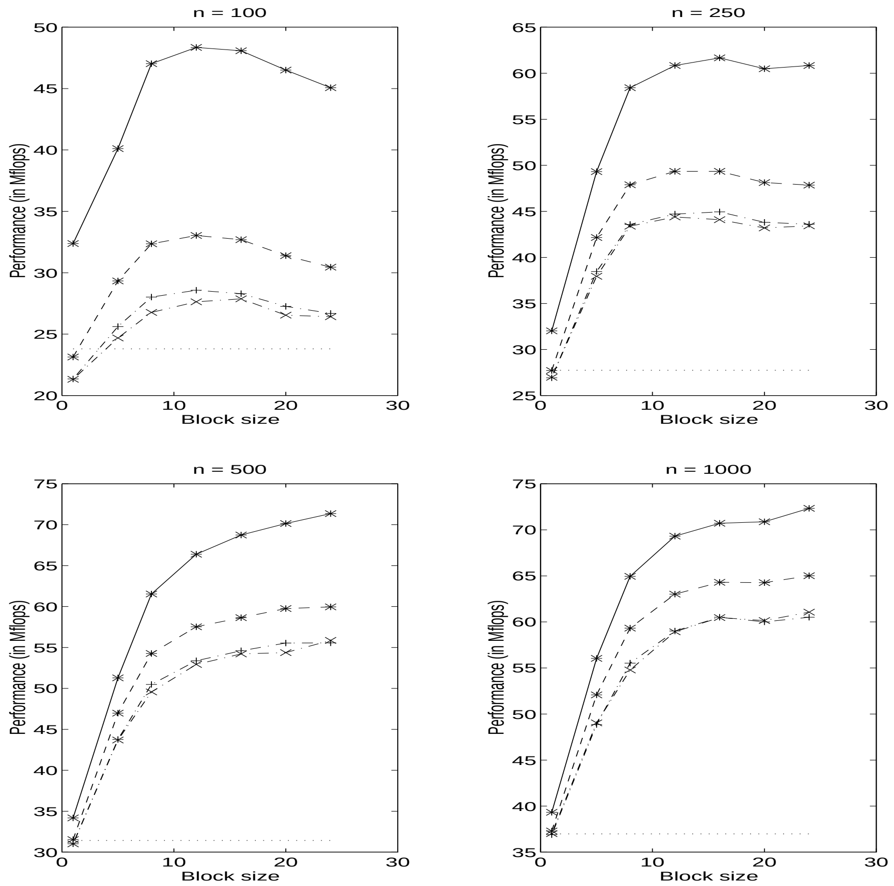


Fig. 11. Performance versus Block Size on IBM RS/6000-370: DGEQPF (···), DGEQRF (—), DGEQPB (- -), DGEQPX (-·-x), DGEQPY (-·-+)

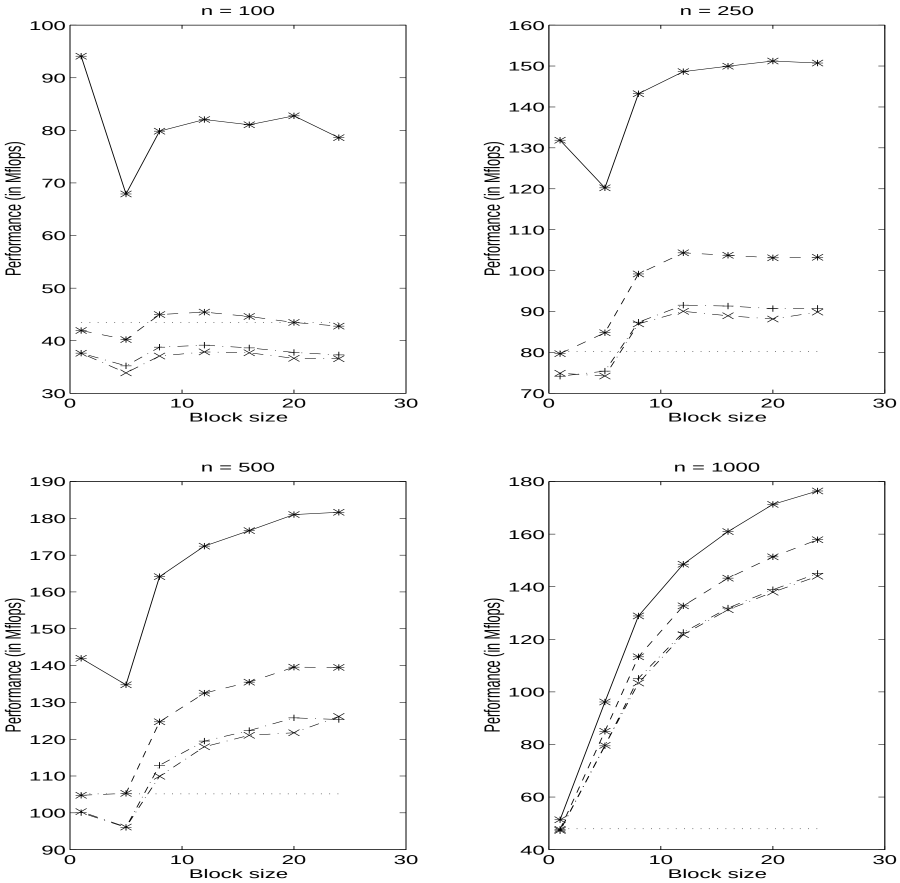


Fig. 12. Performance versus Block Size on SGI R8000: DGEQPF (···), DGEQRF (—), DGEQPB (- -), DGEQPX (-·-x), DGEQPY (-·-+)

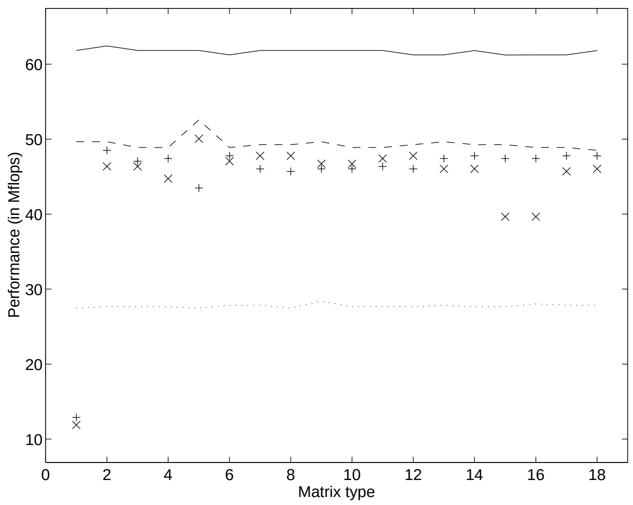


Fig. 13. Performance versus Matrix Type on an IBM RS/6000-370 for $n = 250$ and $nb = 16$: DGEQPF (···), DGEQRF (—), DGEQPB (- -), DGEQPX (x), DGEQPY (+)

better inherent data locality of the BLAS 3 implementation in this computer. These results also show that DGEQPX and DGEQPY exhibit comparable performance.

Figures 13 through 14 offer a closer look at the performance of the various test matrices. We chose $nb = 16$ and $n = 250$ as a representative example. Similar behavior was observed in the other cases.

We see that on the IBM platforms (Figure 13), the performance of DGEQRF and DGEQPF does not depend on the matrix type. We also see that, except for matrix types 1, 5, 15, and 16, the postprocessing of the initial approximate RRQR factorization takes up very little time, with DGEQPX and DGEQPY performing similarly. For matrix type 1, considerable work is required to improve the initial QR factorization. For matrix types 5 and 15, the performance of DGEQPX and DGEQPY differ noticeably on the IBM platform, but there is no clear winner. We also note that matrix type 5 is suitable for DGEQPB, since the independent columns are up front and thus are revealed quickly, and the rest of the matrix is factored with DGEQRF.

The SGI platform (Figure 14) offers a different picture. The performance of all algorithms shows more dependence on the matrix type, and DGEQPB performs *worse* on matrix type 5 than on all others. Nonetheless, except for matrix 1, DGEQPX and DGEQPY do not require much postprocessing effort.

The pictures for other matrix sizes are similar. The cost for DGEQPX and DGEQPY decreases as the matrix size increases, except for matrix type 1, where it increases as expected. We also note that Figures 11 though 12 would have looked even more favorable for our algorithm had we omitted matrix 1 or chosen the median (instead of the average) performance.

Figure 15 shows the percentage of the actual amount of flops spent in monitoring

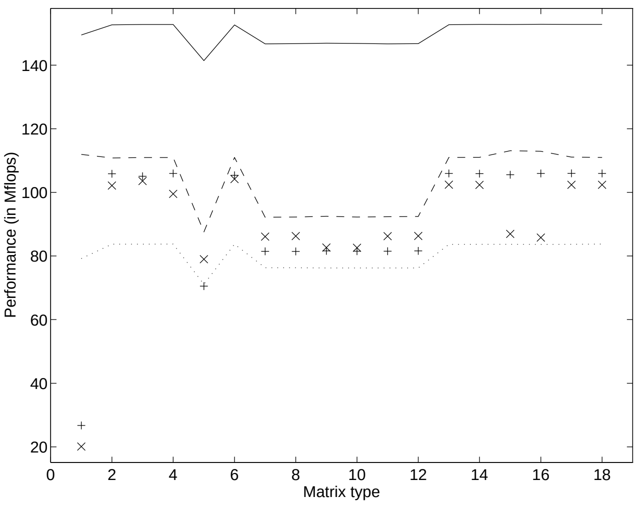


Fig. 14. Performance versus Matrix Type on an SGI R8000 for $n = 250$ and $nb = 16$: DGEQPF (···), DGEQRF (—), DGEQPB (- -), DGEQPX (x), DGEQPY (+)

the rank in DGEQPB and in postprocessing the initial QR factorization for different matrix sizes on the IBM RS/6000. We show only matrix types 2 through 18, since the behavior of matrix type 1 is rather different: in this special case, roughly 50% of the overall flops is expended in the postprocessing. Note that the actual performance penalty due to these operations is, while small, still considerably higher than the flop count suggests. This is not surprising given the relatively fine-grained nature of the condition estimation and postprocessing operations.

Lastly, one may wonder whether the use of DGEQRF to compute the initial QR factorization would lead to better results, since DGEQRF is the fastest QR factorization algorithm. This is not the case, since DGEQRF does not provide any rank preordering, and thus performance gains from DGEQRF are annihilated in the postprocessing steps. For example, for matrices of order 250 on an IBM RS/6000-370, the average Mflop rate, excluding matrix 5, was 4.5, with a standard deviation of 1.4. The percentage of flops spent in postprocessing in these cases was on average 76.8 %, with a standard deviation of 6.7. For matrix 5, we are lucky, since the matrix is of low rank and all independent columns are at the front of the matrix. Thus, we spend only 3% in postprocessing, obtaining a performance of 49.1 Mflops overall. In all other cases, though, considerable effort is expended in the postprocessing phase, leading to overall disappointing performance. These results show that the preordering done by DGEQPB is essential for the efficiency of the overall algorithm.

## 5. CONCLUSIONS

In this paper, we presented rank-revealing QR factorization (RRQR) algorithms that combine an initial QR factorization employing a restricted pivoting scheme

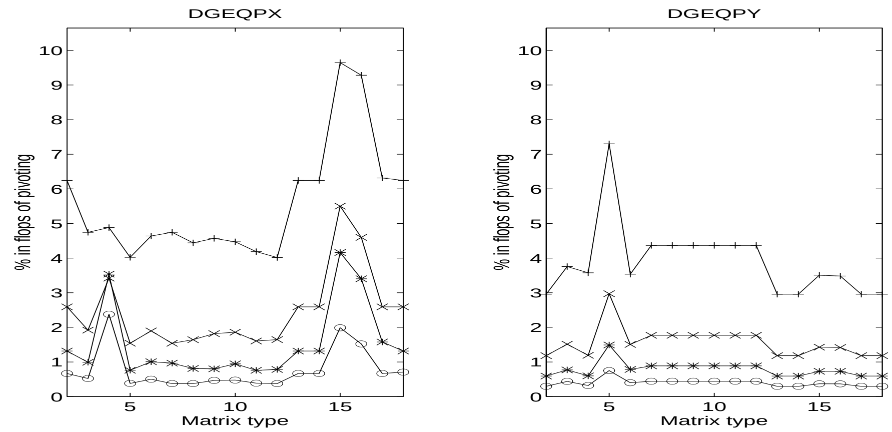


Fig. 15. Cost of Pivoting (in % of flops) versus Matrix Types of Algorithms **DGEQPX** and **DGEQPY** on an IBM RS/6000-370 for Matrix Sizes 100 (+), 250 (x), 500 (*) and 1000 (o).

with postprocessing steps based on variants of algorithms suggested by Chandrasekaran and Ipsen and Pan and Tang.

The restricted-pivoting strategy results in an initial QR factorization that is almost entirely based on BLAS-3 kernels, yet still achieves at a good approximation of an RRQR factorization most of the time. To guarantee the reliability of the initial RRQR factorization and improve it if need be, we improved an algorithm suggested by Pan and Tang, relying heavily on incremental condition estimation and "blocked" Givens rotation updates for computational efficiency. As an alternative, we implemented a version of an algorithm by Chandrasekaran and Ipsen, which among other improvements uses the $f$-factor technique suggested by Pan and Tang to avoid cycling in the presence of roundoff errors.

Numerical experiments on eighteen different matrix types with matrices ranging in size from 100 to 1000 on IBM RS/6000 and SGI R8000 platforms show that this approach produces reliable rank estimates while outperforming the (less reliable) QR factorization with column pivoting, the currently most common approach for computing an RRQR factorization of a dense matrix.

ACKNOWLEDGMENTS

We thank Xiaobai Sun, Peter Tang and Enrique S. Quintana-Ortí for stimulating discussions on the subject.

## References

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1992a . *LAPACK User's Guide*. SIAM, Philadelphia.

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1994b . *LAPACK User's Guide Release 2.0*. SIAM, Philadelphia.

Bischof, C. H. 1989. A block QR factorization algorithm using restricted pivoting. In *Proceedings SUPERCOMPUTING '89* (Baltimore, Md., 1989), pp. 248–256. ACM Press.

Bischof, C. H. 1990. Incremental condition estimation. *SIAM Journal on Matrix Analysis and Applications 11*, 2, 312–322.

Bischof, C. H. 1991. A parallel QR factorization algorithm with controlled local pivoting. *SIAM Journal on Scientific and Statistical Computing 12*, 1, 36–57.

Bischof, C. H. and Hansen, P. C. 1991. Structure-preserving and rank-revealing QR factorizations. *SIAM Journal on Scientific and Statistical Computing 12*, 6 (November), 1332–1350.

Bischof, C. H. and Hansen, P. C. 1992. A block algorithm for computing rank-revealing QR factorizations. *Numerical Algorithms 2*, 3-4, 371–392.

Bischof, C. H. and Shroff, G. M. 1992. On updating signal subspaces. *IEEE Transactions on Signal Processing 40*, 1, 96–105.

Bischof, C. H. and Tang, P. T. P. 1991. Robust incremental condition estimation. Preprint MCS-P225-0391, Mathematics and Computer Science Division, Argonne National Laboratory.

Businger, P. A. and Golub, G. H. 1965. Linear least squares solution by Householder transformation. *Numerische Mathematik 7*, 269–276.

Chan, T. F. 1987. Rank revealing QR factorizations. *Linear Algebra and Its Applications 88/89*, 67–82.

Chan, T. F. and Hansen, P. C. 1992. Some applications of the rank revealing QR factorization. *SIAM Journal on Scientific and Statistical Computing 13*, 3, 727–741.

Chan, T. F. and Hansen, P. C. 1994. Low-rank revealing QR factorizations. *Numerical Linear Algebra and Applications 1*, 1, 33–44.

Chandrasekaran, S. and Ipsen, I. 1994. On rank-revealing QR factorizations. *SIAM Journal on Matrix Analysis and Applications 15*, 2.

Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W. 1979. *LINPACK Users' Guide*. SIAM, Philadelphia.

Eldén, L. and Schreiber, R. 1986. An application of systolic arrays to linear discrete ill-posed problems. *SIAM Journal on Scientific and Statistical Computing 7*, 892–903.

Foster, L. V. 1986. Rank and null space calculations using matrix decomposition without column interchanges. *Linear Algebra and Its Applications 74*, 47–71.

Golub, G. H., Klema, V., and Stewart, G. W. 1976. Rank degeneracy and least squares problems. Technical Report TR–456, University of Maryland, Dept. of Computer Science.

Golub, G. H. and Loan, C. F. V. 1983. *Matrix Computations*. The Johns Hopkins University Press.

Golub, G. H. and Loan, C. F. V. 1989. *Matrix Computations* (2nd ed.). The Johns Hopkins University Press, Baltimore, Maryland.

Golub, G. H., Manneback, P., and Toint, P. L. 1986. A comparison between some direct and iterative methods for certain large scale geodetic least-squares problem. *SIAM Journal on Scientific and Statistical Computing 7*, 799–816.

Gragg, W. B. and Stewart, G. W. 1976. A stable variant of the secant method for solving nonlinear equations. *SIAM Journal on Numerical Analysis 13*, 6, 889–903.

Grandine, T. A. 1987. An iterative method for computing multivariate $C^1$ piecewise polynomial interpolants. *Computer Aided Geometric Design 4*, 307–319.

Grandine, T. A. 1989. Rank deficient interpolation and optimal design: An example. Technical Report SCA–TR–113 (February), Boeing Computer Services, Engineering and Scientific Services Division.

Gu, M. and Eisenstat, S. 1992. A stable and efficient algorithm for the rank-one modification of the symmmetric eigenproblem. Technical Report YALEU/DCS/RR-916, Yale University, Department of Computer Science.

Hansen, P. C. 1990. Truncated SVD solutions to discrete ill-posed problems with ill-determined numerical rank. *SIAM Journal on Matrix Analysis and Applications 11*, 3, 503 – 518.

Hansen, P. C., Sekii, T., and Shibahashi, H. 1992. The modified truncated SVD-method for regularization in general form. *SIAM Journal on Scientific Computing 13*, 1142–1150.

Higham, N. J. 1986. Efficient algorithms for computing the condition number of a tridiagonal matrix. *SIAM Journal on Scientific and Statistical Computing 7*, 150–165.

Hong, Y. P. and Pan, C.-T. 1992. The rank revealing QR decomposition and SVD. *Mathematics of Computation 58*, 213–232.

Hsieh, S. F., Liu, K. J. R., and Yao, K. 1991. Comparisons of truncated QR and SVD methods for AR spectral estimations. In R. J. Vaccaro Ed., *SVD and Signal Processing II* (Amsterdam, 1991), pp. 403–418. Elsevier Science Publishers.

Moré, J. 1978. The Levenberg-Marquardt algorithm: Implementation and theory. In G. A. Watson Ed., *Proceedings of the Dundee Conference on Numerical Analysis* (Berlin, 1978). Springer Verlag.

Pan, C.-T. and Tang, P. T. P. 1992. Bounds on singular values revealed by QR factorizaton. Technical Report MCS-P332-1092, Mathematics and Computer Science Division, Argonne National Laboratory.

Quintana-Ortí, G. and Quintana-Ortí, E. S. 1996. Guaranteeing termination of Chandrasekaran & Ipsen's algorithm for computing rank-revealing QR factorizations. Preprint MCS-P564-0196, Mathematics and Computer Science Division, Argonne National Laboratory.

Quintana-Ortí, G., Sun, X., and Bischof, C. H. 1995. A BLAS-3 version of the QR factorization with column pivoting. Preprint MCS-P551-1295, Mathematics and Computer Science Division, Argonne National Laboratory.

Reichel, L. and Gragg, W. 1990. Fortran subroutines for updating the QR factorization. *ACM Transactions on Mathematical Software 16*, 369–377.

Schreiber, R. and Van Loan, C. 1989. A storage efficient WY representation for products of Householder transformations. *SIAM Journal on Scientific and Statistical Computing 10*, 1, 53–57.

Stewart, G. W. 1984. Rank degeneracy. *SIAM Journal on Scientific and Statistical Computing 5*, 403–413.

Stewart, G. W. 1990. An updating algorithm for subspace tracking. Technical Report CS-TR-2494, University of Maryland, Department of Computer Science.

Stewart, G. W. 1993. Determining rank in the presence of error. Technical Report CS-TR-2972, Dept. of Computer Science, University of Maryland.

Waldén, B. 1991. Using a fast signal processor to solve the inverse kinematic problem with special emphasis on the singularity problem. Ph. D. thesis, Linköping University, Dept. of Mathematics.