Computing Rank-Revealing QR Factorizations of Dense Matrices

CHRISTIAN H. BISCHOF
Argonne National Laboratory
and
GREGORIO QUINTANA-ORTÍ
Universidad Jaime I

We develop algorithms and implementations for computing rank-revealing QR (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 Golub pivoting strategy, aided 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 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms, Performance

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 Spanish Research Agency CICYT under grant TIC96-1062-C03-03. 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 Politecnica de Valencia and a visiting scientist at the Mathematics and Computer Science Division at Argonne National Laboratory.
Authors' addresses: C. H. Bischof, Mathematics and Computer Science Division, Argonne National Laboratory, Building 221, 9700 South Cass Avenue, Argonne, IL 60439; email: bischof@mcs.anl.gov; G. Quintana-Ortí, Departamento de Informática, Universidad Jaime I, Campus Riu Sec, Castellon, 46071, Spain; email: gquintan@inf.uji.es.

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

1. INTRODUCTION

We briefly summarize the properties of a *rank-revealing QR (RRQR) factorization*. Let A be an $m \times n$ matrix (without loss of generality $m \geq n$) 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 τ 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

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

where P is a permutation matrix, Q has orthonormal columns, R is upper triangular, and R_{11} is of order r. Furthermore, 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:

$$\kappa(R_{11}) \approx \sigma_1/\sigma_r \tag{3.1}$$

and

$$\|R_{22}\|_2 = \sigma_{\max}(R_{22}) \approx \sigma_{r+1}. \tag{3.2}$$

Whenever there is a well-determined gap in the singular-value spectrum between σ_r and σ_{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. From now on, whenever the word "rank" appears, it means the "numerical rank with respect to threshold τ." 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 null vectors, and a few steps of subspace iteration suffice to compute null vectors 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 null space. 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 beam forming [Bischof and Shroff 1992], spectral estimation [Hsieh et al. 1991], regularization [Hansen 1990; Hansen et al. 1992; Waldén 1991], and subset selection [Hotelling 1957; de Hoog and Mattheij 1989].

Stewart [1990] suggested another alternative to the singular-value decomposition, 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 where both $\|R_{12}\|_2$ and $\|R_{22}\|_2$ are of the order σ_{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 null space 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 algorithms for computing RRQR factorizations. From the interlacing theorem for singular values [Golub and Van Loan 1983, Corollary 8.3.3], we have

$$\sigma_{\min}(R(1:k, 1:k)) \leq \sigma_k \tag{4.1}$$

and

$$\sigma_{\max}(R(k+1:n, k+1:n)) \geq \sigma_{k+1}(A). \tag{4.2}$$

Hence, to satisfy conditions (3.1) and (3.2), 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 $\sigma_{\max}(R_{22})$.

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 Van 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 Van 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. 1992; 1994] implementations. The drivers from both libraries are point algorithms: LINPACK subroutine xQRDC[1] is based on BLAS-1, and LAPACK subroutine xGEQPF is based on BLAS-2. Recently, Quintana-Ortí et al. [1995] developed an implementation of the Golub algorithm that allows half of the work to be performed with BLAS-3 kernels. Bischof also had developed restricted-pivoting variants of the Golub strategy to enable the use of BLAS-3 type kernels [Bischof 1989] for almost all of the work and to reduce communication cost on distributed-memory machines [Bischof 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*

$$\|R(n-p+1:n, n-p+1:n)\|_2 \le \|RW\|_2 \|W_2^{-1}\|_2. \tag{5}$$

This means that if we can determine a matrix W with p linearly independent columns, all of which lie approximately in the null space of R (i.e., $\|RW\|_2$ is small), and if W_2 is well conditioned such that $(\sigma_{\min}(W_2))^{-1} = \|W_2^{-1}\|_2$ is not large, 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 null space bases for A include those by Golub et al. [1976], Chan [1987], and Foster [1986]. Other algorithms addressing Task 2 are those by Stewart [1984] and Gragg and Stewart [1976]. Algorithms addressing Task 1 include those of Chan and Hansen [1994] and Golub et al. [1976].

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 developed a block variant of Chan's algorithm [Bischof and Hansen 1992]. Reichel and Gragg [1990] supplied some basic tools in Fortran 77 very useful for building Chan's algorithm.

Chan's algorithm [Chan 1987] guaranteed

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

[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: SQRDC, DQRDC, CQRDC, or ZQRDC.

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 algorithm and implementation for computing an RRQR factorization 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 block-oriented algorithm for computing 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 find that

—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 preprocessing algorithm performs considerably faster than the algorithm currently included in LAPACK for computing the QR factorization with column pivoting, and close to the performance of the LAPACK algorithm for computing the QR factorization without pivoting.

The article 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-oriented algorithm that employs a restricted pivoting strategy to approximately compute an RRQR factorization, employing the ideas described by 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 $|\alpha| = \|x\|_2$ (e.g., see Golub and Van 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 $B := A - 2uz^T$.

Golub's algorithm for computing the QR factorization with column pivoting of an $m \times n$ matrix A works as follows [Golub 1965]. It consists of n stages. In stage i, the complete column with largest two-norm in block $A(i : m, i : n)$ is pivoted to the ith position. Then, the Householder reflector that nullifies the elements $i + 1 : m$ in column i is computed and applied to the mentioned block. In order to avoid the recomputation of the norms of the block $A(i : m, i : n)$ for every stage $i = 1,2, \ldots, n$, an auxiliary vector can be used. Initially, the two-norms of the columns of the matrix are saved in this vector, and afterward, in every stage they are downdated. After step i is completed, the values of this vector are the lengths of the projections of the columns of the currently permuted AP onto

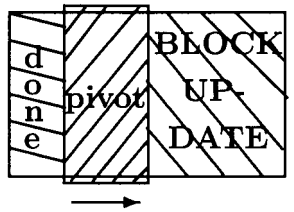

Fig. 1. Restricting pivoting for a block algorithm.

the orthogonal complement of the subspace spanned by the first i columns of AP.

The bulk of the computational work in this algorithm is performed in the applying of the Householder vector to the rest of the matrix, 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. 1992; 1994]. LAPACK employs the so-called compact WY representation of products of Householder matrices [Schreiber and Van Loan 1989], which expresses the product

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

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 where 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-oriented algorithm for computing the QR factorization with pivoting, 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 the vector with partial two-norms 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]. We choose not 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 1. 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^TAP 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 τ, which defines what we consider a rank-deficient matrix.

We approximate

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

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 $\hat{\sigma}_{\min}(R_i) = 1/\|z\|_2$ defined by a large norm solution z to $R_i^T z = d$, $\|d\|_2 = 1$, 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

$$\sigma_{\min}(R_{i+1}) \approx \hat{\sigma}_{\min}(R_{i+1}) = 1 \Big/ \left\| \begin{pmatrix} sz \\ c \end{pmatrix} \right\|_2. \tag{13}$$

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

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

2.1 The Algorithm

The restricted block pivoting algorithm proceeds in four phases:

Phase 1: Pivoting of Column with Largest Two-Norm into First Position. The column with largest two-norm in the full matrix A is pivoted into the first position. This phase is motivated by the fact that the largest two-norm of the columns 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 w, $w \geq nb$, we try to generate nb Householder transformations by applying the 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. When the pivot column chosen from the pivot window would lead to a leading triangular factor whose condition number exceeds τ, 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 to encompass the next w not-yet-rejected columns. When all columns in the matrix 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

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

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

In our experiments, we chose

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

This choice tries to ensure a suitable pivot window and "loosens up" a bit as the 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 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

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

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. Indeed, this phase is not strictly necessary, since the postprocessing can fix bad rank estimates, but this is a much cheaper way to improve the rank estimate since the postprocessings are more expensive.

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 Phases 2 and 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 that satisfies

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

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

$$\hat{\kappa}\left(\begin{pmatrix} R_{r_3} r \\ 0 \end{pmatrix}, y\right) > \tau.$$

This result suggests that this QR factorization is a good approximation to an RRQR factorization and that 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 the algorithms that compute RRQR factorizations, and they 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 preceding section. Shortly thereafter, Pan and Tang [1992] introduced another guaranteed reliable algorithm for computing an RRQR factoriza-

```
Algorithm PT3M(f,k)
1. i = k + 1; accepted_col = 0; Π = I;
2. u := left singular vector corresponding to σ_min(R(1:k, 1:k))
3. while ( accepted_col ≤ n − k ) do
4.    R := triang(R · Π^R_{k+1,i}) ; Π := Π · Π^R_{k+1,i}
5.    Compute σ ≈ σ_min(R(1:k + 1, 1:k + 1))
6.    if (σ ≥ f · |R(k + 1, k + 1)|) then
7.          accepted_col := accepted_col + 1;
8.    else
9.          v := right singular vector corresponding to σ
10.         Find index q, 1 ≤ q ≤ k + 1, such that: |v_q| = max_j |v_j|
11.         R := triang(R · Π^L_{q,k+1}) ; Π := Π · Π^L_{q,k+1}; accepted_col = (
12.         u := left singular vector corresponding to σ_min(R(1:k, 1:k)
13.   end if
14.   if (i == n) then i := k + 1 else i := i + 1 end if
15. end while
16. Find smallest index q, k + 1 ≤ q ≤ n, such that
      ||R(k + 1:q, q)||_2 = max_{k+1≤i≤n} ||R(k + 1:i, i)||_2
17. R := triang(R · Π^R_{k+1,q}) ; Π := Π · Π^R_{k+1,q}
```

Fig. 2. Variant of Pan-Tang RRQR algorithm.

tion for triangular matrices. In the following subsections, we describe our improvements and implementations of these algorithms.

Both the preprocessing and the postprocessing employ Bischof's ICE (implemented in LAPACK's xLAIC1) to estimate the smallest singular value because of its accuracy and speed. We recompute the ICE estimates in the postprocessing since its cost is negligible compared with the total cost. In the postprocessing, to estimate the corresponding right singular vector we first estimate the left singular vector by using ICE and then we estimate the right vector by means of a backsolve.

To estimate the largest singular value we use two different methods. In the preprocessing we use a very cheap estimate: the largest two-norm of the columns times the third root of the matrix dimension. In the postprocessings we use ICE since it is more accurate, though more expensive than that used in the preprocessing.

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 2. 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; in other words, $i \rightarrow i + 1, \ldots, j - 1 \rightarrow j, j \rightarrow i$, whereas $\Pi^L_{i,j}$, $i < j$ denotes a left cyclic permutation that exchanges columns i and j; in other words, $j \leftarrow i, i \leftarrow i + 1, \ldots, j - 1 \leftarrow j$. In the algorithm, triang (A) denotes the upper triangular factor R in the factorization $A = QR$ of A. As can be seen from Figure 2, 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 is used to arrive at estimates for smallest singular values and vectors. Thus, σ (line 5) and v (line 9) of Figure 2 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 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 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.

(5) The final pivoting (lines 16 and 17) improves the estimates for the condition number of $R(1:k+1,1:k+1)$, and makes the rank estimate more accurate.

Improvements (1) through (3) on average decreased runtime by a factor of five on 200×200 matrices on an Alliant FX/80 when compared with a

```
Algorithm Hybrid-III-sf(f,k)
1. Π = 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. 3. Variant of Chandrasekaran-Ipsen Hybrid-III algorithm.

straightforward implementation of the original algorithm by Pan and Tang. 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 are 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.

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 3–6.

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) untill 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% fewer 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 detection of convergence, as shown below.

(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.

Algorithm Golub-I-sf(f,k)
1. Find smallest index j, $k \le j \le n$, such that
2. $\quad \|R(k\!:\!j,j)\|_2 = \max_{k \le i \le n} \|R(k\!:\!i,i)\|_2$
3. **if** $f \cdot \|R(k\!:\!j,j)\|_2 > |\, R(k,k)\,|$ **then**
4. $\quad R := \text{triang}(R \cdot \Pi^R_{k,j}); \Pi := \Pi \cdot \Pi^R_{k,j}$
5. **end if**

Fig. 4. "f-factor" variant of Golub-I algorithm.

Algorithm Chan-II-sf(k)
1. $v :=$ right singular vector corresponding to $\sigma_{\min}(R(1\!:\!k, 1\!:\!k))$.
2. Find largest index j, $1 \le j \le k$, such that: $|v_j| = \max_{1 \le i \le k} |v_i|$
3. **if** $f \cdot |v_j| > |v_k|$ **then**
4. $\quad R := \text{triang}(R \cdot \Pi^L_{j,k}); \Pi := \Pi \cdot \Pi^L_{j,k}$
5. **end if**

Fig. 5. "f-factor" variant of Chan-II algorithm.

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

Fig. 6. Algorithm for computing rank-revealing QR factorization.

(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, $\|R(k : i, i)\|_2$ 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 $R(1 : k, 1 : k)$. 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 ϵ 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 that of Pan and Tang; the analysis in Quintana-Ortí and Quintana-Ortí [1996] proves that our algorithm achieves the bounds

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

and

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

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, we apply Chandrasekaran and Ipsen's original ordering to a matrix that almost reveals the rank:

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

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 $\sigma_{k+1}(A) \approx \sigma_{\max}(R(k+1:n, k+1:n))$.

Thus we need an initial estimate for the rank, which we compute from the upper triangular matrix obtained by the preprocessing, by using the ICE estimator. Given that initial estimate for the rank, to obtain the

numerical rank with respect to a given threshold τ, we employ the algorithm shown in Figure 6. In our actual implementation, α and β are computed in Hybrid-III-sf or PT3M.

This algorithm works as follows. It first calls to algorithms Hybrid-III-rsf or PT3m in order to get a column ordering in A that makes

$$\sigma_k(A) \approx \sigma_{\min}(R_{11}) \text{ and } \sigma_{k+1}(A) \approx \sigma_{\max}(R_{22}), \tag{22}$$

where $R_{11} = R(1 : k, 1 : k)$ and $R_{22} = R(k + 1 : n, k + 1 : n)$, which values α and β can be obtained from. It can be easily determined if the numerical rank with respect to threshold τ is larger, equal, or smaller than the current estimate k. For instance, if $\alpha \leq \tau$ and $\beta > \tau$, then the numerical rank is k and the algorithm ends. If $\alpha \leq \tau$ and $\beta \leq \tau$, then the numerical rank is larger than k. If $\alpha \geq \tau$ and $\beta \geq \tau$, then the numerical rank is smaller than k. As the preprocessing usually offers very accurate estimates for the rank, this algorithm only updates the estimates for the rank in one unit. In the last two cases, the algorithm Hybrid-III-rsf or PT3m must be applied on the new estimates. The current implementation has been specifically designed to avoid cycling (infinite looping) in the presence of clusters of singular values.

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 for computing the QR factorization with column pivoting provided in LAPACK. This is not a block-oriented algorithm and, hence, its performance does not depend on the block-size.

—*DGEQPB:* A BLAS-3 implementation for computing 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 Sections 3.2 and 3.3.

—*DGEQPY:* DGEQPB followed by an implementation of the variant of the line break Pan-Tang algorithm described in Sections 3.1 and 3.3.

—*DGEQRF:* The block-oriented and BLAS-3 implementation for computing the 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_acm.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 MPIS R8000-75MHz. In each case, we em-

Table I. Test Matrix Types (n = rank for $n = 1000$)

	Description	r	$\sigma_{\max}$	σ_r	σ_{r+1}	$\sigma_{\min}$
1	Matrix with rank $(min(m, n)/2) - 1$	499	1.0e0	1.0e0	2.0e-7	1.2e-19
2	$A(:, 2 : \min(m, n))$ has full rank $\mathcal{R}(A) = \mathcal{R}(A(:, 2 : \min(m, n)))$	999	1.0e0	5.0e-4	6.7e-19	6.7e-19
3	Full rank	1000	1.0e0	5.0e-4	NA	5.0e-4
4	$A(:, 1 : 3)$ small in norm $A(:, 4 : n)$ of full rank end	997	2.9e+1	5.0e-4	2.4e-4	4.2e-5
5	$A(:, 1 : 3)$ small in norm $\mathcal{R}(A) = \mathcal{R}(A(:, 1 : 3))$	3	1.0e0	5.0e-4	5.5e-14	7.6e-21
6	5 smallest sing. values clustered	1000	1.0e0	7.0e-4	NA	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

ployed 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 article, 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 $(N/2) + 1$ 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

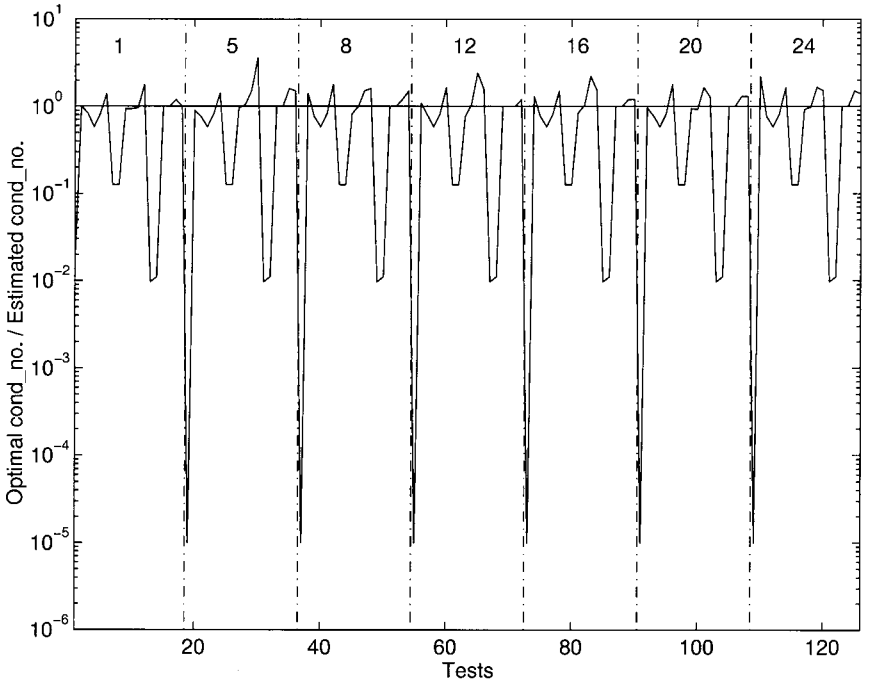

Fig. 7. Ratio between optimal and estimated condition number for DGEQPB. Each panel contains the results for the 18 matrix types on one different block size.

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. For this case, we list in Table I the numerical rank r with respect to a condition threshold of $\tau = 10^5$, the largest singular value $\sigma_{\max}$, the rth singular value σ_r, the $(r + 1)$st singular value σ_{r+1}, and the smallest singular value $\sigma_{\min}$ for our test cases.

The three figures in this subsection (Figures 7–9) 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). Since the pivoting strategy (and hence the numerical behavior of DGEQPB) is potentially affected by the block size chosen, we show the results ordered by block size.

Figures 7 and 8 display the ratio

$$\Theta := \frac{(\sigma_1/\sigma_r)}{\hat{\kappa}(R_r)}, \tag{23}$$

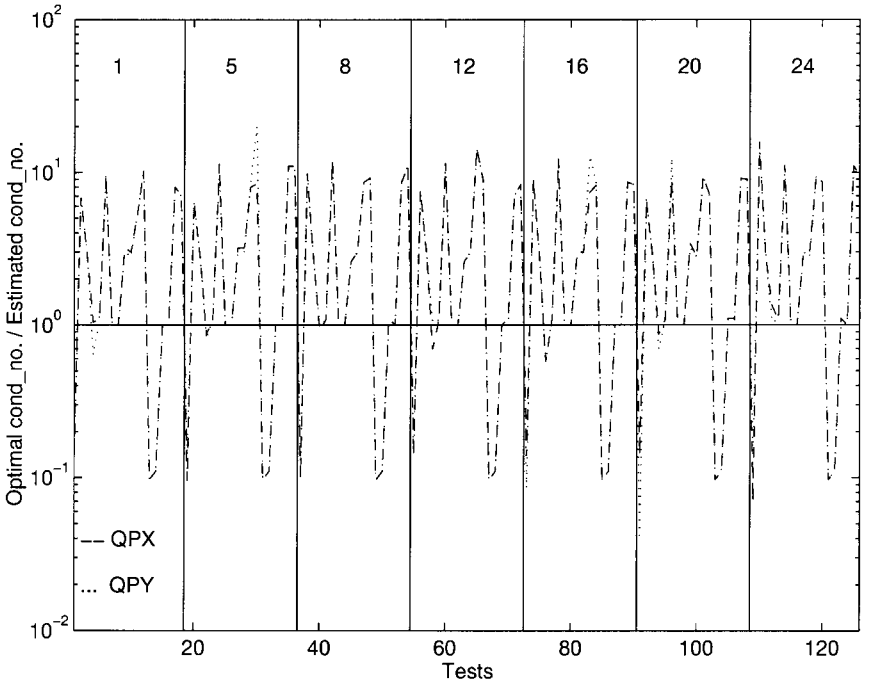

Fig. 8. Ratio between optimal and estimated condition number for DGEQPX (dashed line) and DGEQPY (dotted). Each panel contains the results for the 18 matrix types on one different block size.

where $\hat{\kappa}(R)$ as defined in (14) is the computed estimate of the condition number of R after DGEQPB (Figure 7) and DGEQPX and DGEQPY (Figure 8). Thus, Θ 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 $\hat{\kappa}$ is a good condition estimate, our RRQR factorizations do a good job of capturing the "large" singular values of A.

We see that except for matrix type 1 in Figure 7, the block size does not play much of a role numerically, although close inspection reveals subtle variations in both Figures 7 and 8. With block size 1, DGEQPB just becomes the standard Golub pivoting strategy. Thus, the first panel in Figure 7 corroborates the experimentally robust behavior of this algorithm. We also see that except for 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 $(n/2) - 1$ independent columns and generating the leading $(n/2) + 1$ as random linear combinations of those columns, scaled by $\epsilon^{(1/4)}$, where ϵ is the machine precision. Thus, the restricted pivoting strategy, in its myopic view of the matrix, gets stuck (so to speak) in these columns.

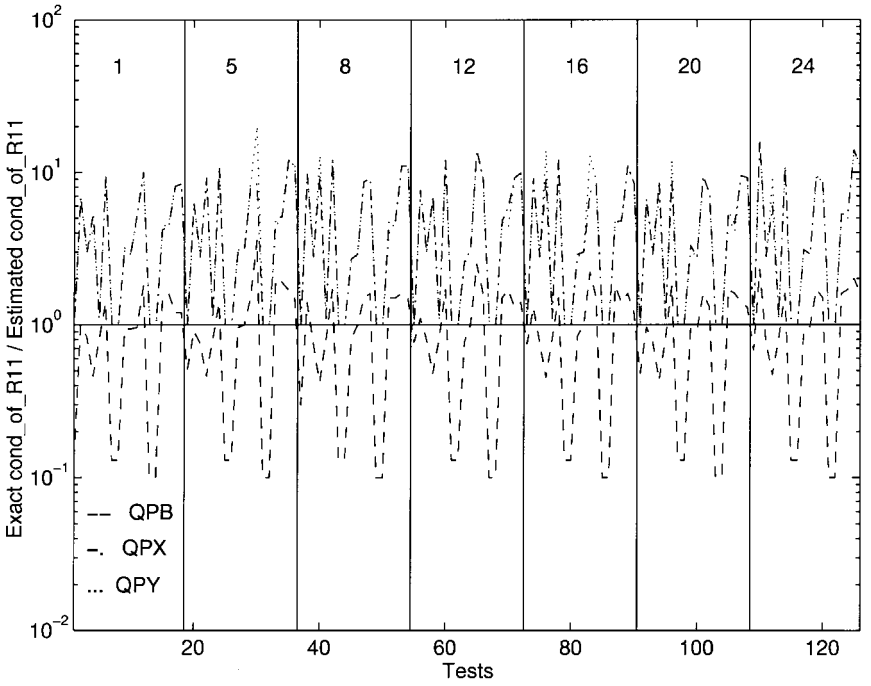

Fig. 9. Ratio between exact and estimated condition number of leading triangular factor for DGEQPB (dashed), DGEQPX (dashed-dotted), and DGEQPY (dotted). Each panel contains the results for the 18 matrix types on one different block size.

The postprocessing of these approximate RRQR factorizations, on the other hand, remedies potential shortcomings in the preprocessing step. As can be seen from Figure 8, 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 9 shows the ratio of the exact condition number to the estimated condition number of the leading triangular factor. We observe excellent agreement, within about an order of magnitude in all cases.

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

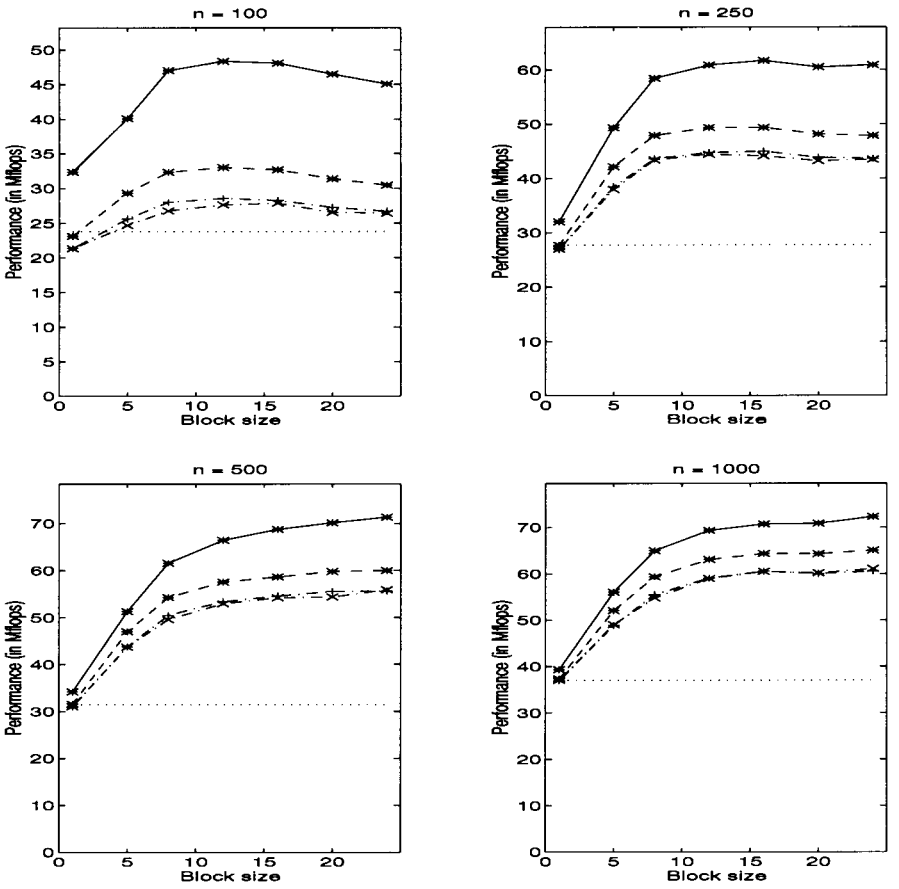

Fig. 10. Performance versus block size on IBM RS/6000-370: DGEQPF (· · ·), DGEQRF (—*), DGEQPB (- –*), DGEQPX (-·-x), DGEQPY (-·-+).

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 10 and 11 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 $\times$ and + symbols indicate DGEQPX and DGEQPY, respectively.

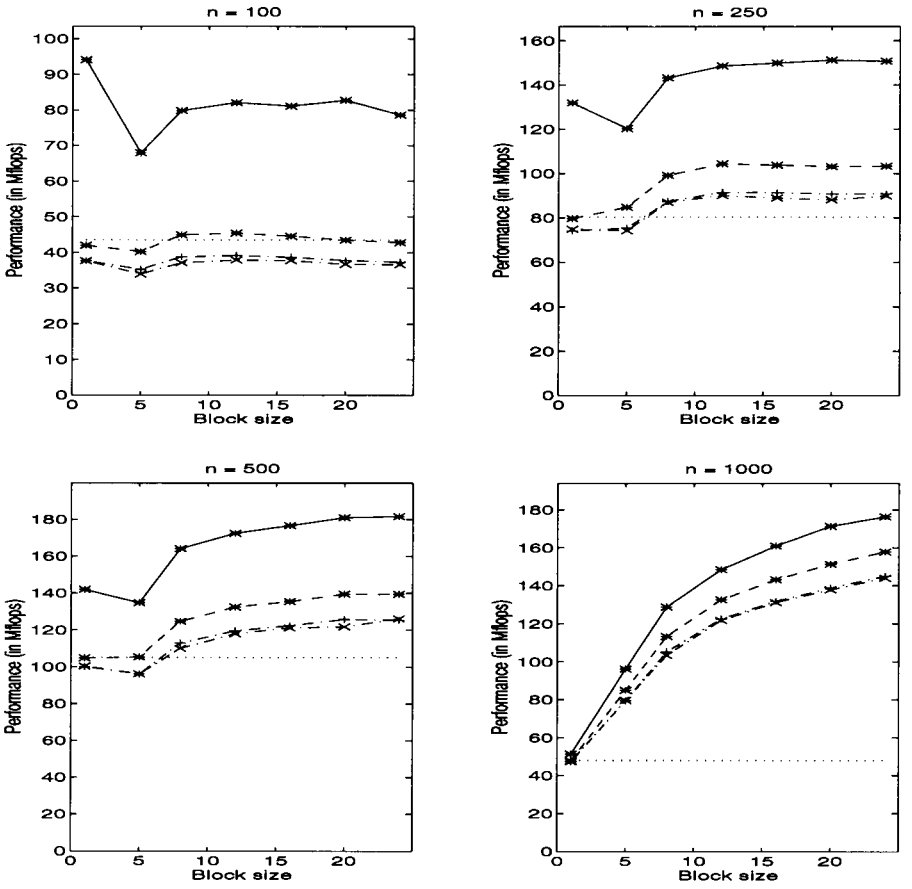

Fig. 11. Performance versus block size on SGI R8000: DGEQPF (· · ·), DGEQRF (—*), DGEQPB (- -*), DGEQPX (-·-x), DGEQPY (-·-+).

On both machines, the performances of the two new algorithms for computing RRQR are 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 4MB, while the IBM platform has only a 32KB 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 better inherent data locality of the BLAS-3 implementation on this computer. These results also show that DGEQPX and DGEQPY exhibit comparable performance.

Figures 12 and 13 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.

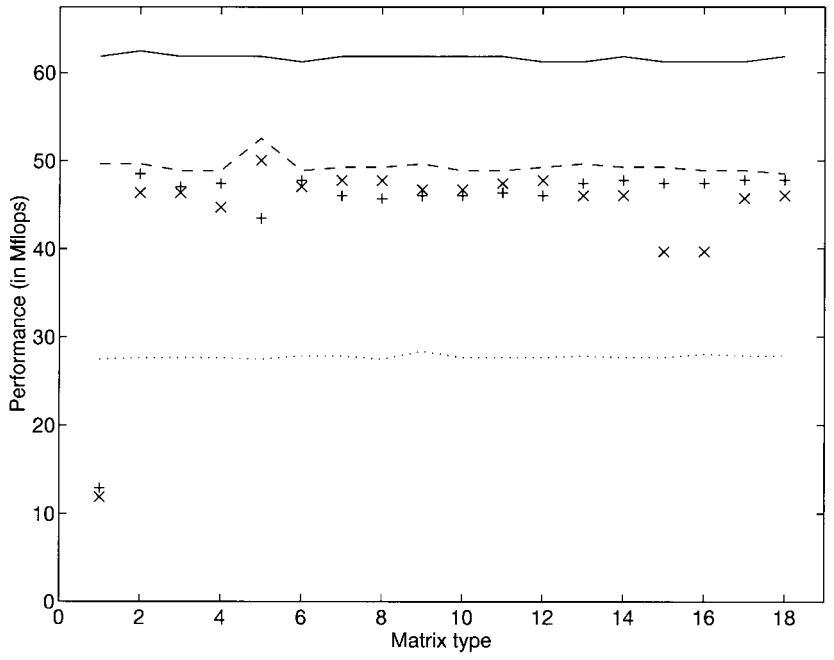

Fig. 12. Performance versus matrix type on an IBM RS/6000-370 for $n = 250$ and $nb = 16$: DGEQPF (· · ·), DGEQRF (—), DGEQPB (- -), DGEQPX (x), DGEQPY (+).

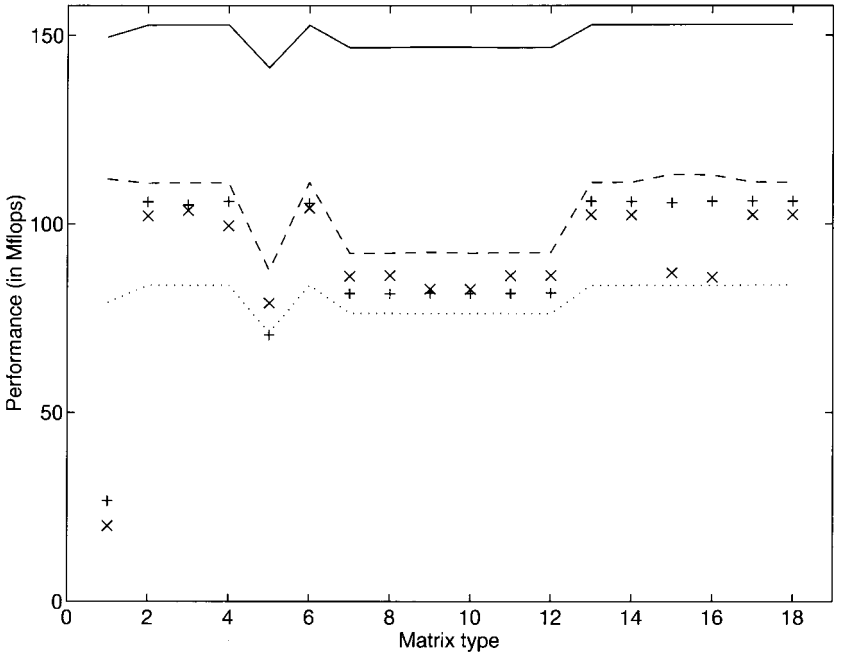

Fig. 13. Performance versus matrix type on an SGI R8000 for $n = 250$ and $nb = 16$: DGEQPF (· · ·), DGEQRF (—), DGEQPB (- -), DGEQPX (x), DGEQPY (+).

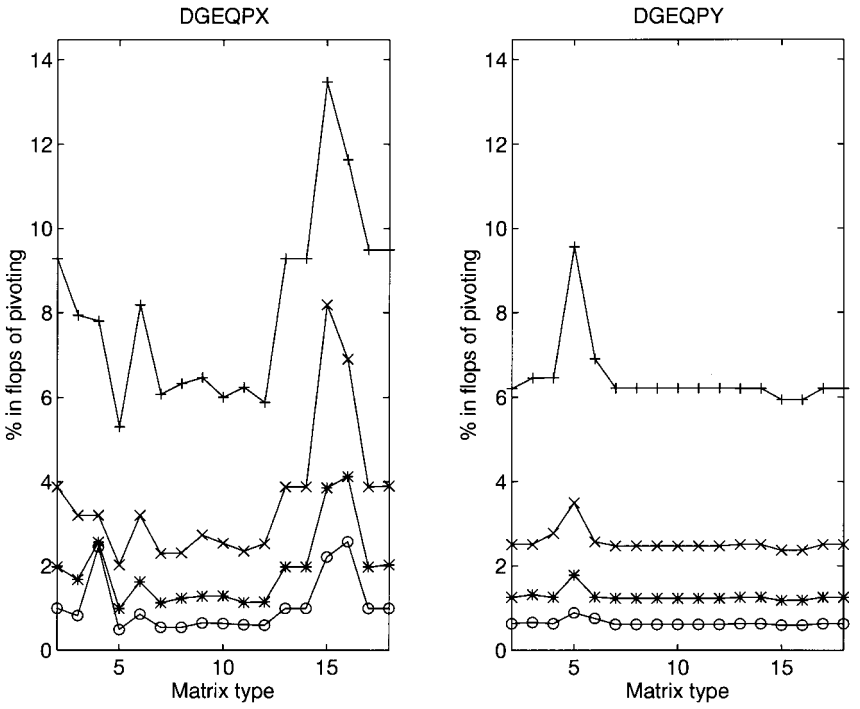

Fig. 14. Cost of pivoting (in % of flops) versus matrix types of algorithms DGEQPX and DGEQPY on an IBM RS/6000-370 for matrix sizes 100 (+), 250 (x), 500 (*) and 1000 (o).

We see that on the IBM platform (Figure 12), 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 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 performances 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 13) 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 10 and 11 would have looked even more favorable for our algorithm had we omitted matrix 1 or chosen the median (instead of the average) performance.

Figure 14 shows the percentage of the actual amount of flops spent in monitoring 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.

One may wonder whether the use of DGEQRF to compute the initial QR factorization would lead to better results, since DGEQRF is the fastest implementation for computing the QR factorization. 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 article, we presented algorithms for computing rank-revealing QR (RRQR) factorizations that combine an initial QR factorization employing a restricted pivoting scheme 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 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.

In our experiments, both postprocessing approaches behaved very similarly and revealed the rank in all cases. However, the theoretical bounds of our version of Chandrasekaran and Ipsen's method are usually tighter than the bounds by Pan and Tang. Thus, in light of our limited experience, we are hesitant to recommend one approach over the other and instead, the

"plug-compatible" design of the codes for the two approaches is meant to encourage users to experiment with both of them.

Numerical experiments on 18 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. We also thank the anonymous referees for their interesting comments.

REFERENCES

ANDERSON, E., BAI, Z., BISCHOF, C., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., AND SORENSEN, D. 1992. *LAPACK User's Guide*. SIAM, Philadelphia, PA.

ANDERSON, E., BAI, Z., BISCHOF, C., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., AND SORENSEN, D. 1994. *LAPACK User's Guide Release 2.0*. SIAM, Philadelphia, PA.

BISCHOF, C. H. 1989. A block QR factorization algorithm using restricted pivoting. In *Proceedings of the 1989 Conference on Supercomputing* (Reno, NV, Nov. 13–17). ACM Press, New York, NY, 248–256.

BISCHOF, C. H. 1990. Incremental condition estimation. *SIAM J. Matrix Anal. Appl. 11*, 2 (Apr.), 312–322.

BISCHOF, C. H. 1991. A parallel QR factorization algorithm with controlled local pivoting. *SIAM J. Sci. Stat. Comput. 12*, 1 (Jan.), 36–57.

BISCHOF, C. H. AND HANSEN, P. C. 1991. Structure-preserving and rank-revealing QR-factorizations. *SIAM J. Sci. Stat. Comput. 12*, 6 (Nov.), 1332–1350.

BISCHOF, C. H. AND HANSEN, P. C. 1992. A block algorithm for computing rank-revealing QR factorizations. *Num. Alg. 2*, 3-4, 371–392.

BISCHOF, C. H. AND SHROFF, G. M. 1992. On updating signal subspaces. *IEEE Trans. Signal Process. 40*, 1, 96–105.

BISCHOF, C. H. AND TANG, P. T. P. 1991. Robust incremental condition estimation. Tech. Rep. CS-91-133. Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL. Also LAPACK Working Note 33.

CHAN, T. F. 1987. Rank revealing QR factorizations. *Lin. Alg. Appl. 88/89*, 67–82.

CHAN, T. F. AND HANSEN, P. C. 1992. Some applications of the rank revealing QR factorization. *SIAM J. Sci. Stat. Comput. 13*, 3 (May), 727–741.

CHAN, T. F. AND HANSEN, P. C. 1994. Low-rank revealing QR factorizations. *Num. Lin. Alg. Appl. 1*, 1, 33–44.

CHANDRASEKARAN, S. AND IPSEN, I. C. F. 1994. On rank-revealing factorisations. *SIAM J. Matrix Anal. Appl. 15*, 2 (Apr.), 592–622.

DE HOOG, F. R. AND MATTHEIJ, R. M. M. 1989. Subset selection for matrices. Tech. Rep. RANA 89-07. Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, Netherlands.

DONGARRA, J. J., MOLER, C. B., BUNCH, J. R., AND STEWART, G. W. 1979. *LINPACK Users' Guide*. SIAM, Philadelphia, PA.

ELDÉN, L. AND SCHREIBER, R. 1986. An application of systolic arrays to linear discrete ill-posed problems. *SIAM J. Sci. Stat. Comput. 7*, 3 (July), 892–903.

FOSTER, L. V. 1986. Rank and null space calculations using matrix decomposition without column interchanges. *Lin. Alg. Appl. 74*, 47–71.

GOLUB, G. H. 1965. Numerical methods for solving linear least squares problems. *Numer. Math. 7*, 206–216.

GOLUB, G. H. AND VAN LOAN, C. F. 1983. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD.

GOLUB, G. AND VAN LOAN, C. F. 1989. *Matrix Computations*. 2nd ed. Johns Hopkins University Press, Baltimore, MD.

GOLUB, G. H., MANNEBACK, P., AND TOINT, P. P. L. 1986. A comparison between some direct and iterative methods for certain large scale geodetic least squares problems. *SIAM J. Sci. Stat. Comput. 7*, 3 (July), 799–816.

GOLUB, G. H., KLEMA, V., AND STEWART, G. W. 1976. Rank degeneracy and least squares problems. Tech. Rep. TR-456. Department of Computer Science, University of Maryland, College Park, MD.

GRAGG, W. B. AND STEWART, G. W. 1976. A stable variant of the secant method for solving nonlinear equations. *SIAM J. Numer. Anal. 13*, 6, 889–903.

GRANDINE, T. A. 1987. An iterative method for computing multivariate C^1 piecewise polynomial interpolants. *Comput. Aided Geom. Des. 4*, 307–319.

GRANDINE, T. A. 1989. Rank deficient interpolation and optimal design: An example. Tech. Rep. SCA-TR-113. Engineering and Scientific Services Division, Boeing Computer Services.

GU, M. AND EISENSTAT, S. 1992. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. Tech. Rep. YALEU/DCS/RR-916. Department of Computer Science, Yale University, New Haven, CT.

HANSEN, P. C. 1990. Truncated SVD solutions to discrete ill-posed problems with ill-determined numerical rank. *SIAM J. Matrix Anal. Appl. 11*, 3 (July), 503–518.

HANSEN, P. C., SEKII, T., AND SHIBAHASHI, H. 1992. The modified truncated SVD-method for regularization in general form. *SIAM J. Sci. Comput. 13*, 1142–1150.

HIGHAM, N. J 1986. Efficient algorithms for computing the condition number of a tridagonal matrix. *SIAM J. Sci. Stat. Comput. 7*, 1 (Jan.), 150–165.

HONG, Y. P. AND PAN, C.-T. 1992. The rank revealing QR decomposition and SVD. *Math. Comput. 58*, 213–232.

HOTELLING, H. 1957. The relation of the newer multivariate statistical methods to factor analysis. *Br. J. Stat. Psychol. 10*, 66–79.

HSIEH, S. F., LIU, K. J. R., AND YAO, K. 1991. Comparisons of truncated QR and SVD methods for AR spectral estimations. In *SVD and Signal Processing II* (Amsterdam), R. J. Vaccaro, Ed. Elsevier Sci. Pub. B. V., Amsterdam, The Netherlands, 403–418.

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

PAN, C.-T. AND TANG, P. T. P. 1992. Bounds on singular values revealed by QR factorization. Tech. Rep. MCS-P332-1092. Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

QUINTANA-ORTÍ, G. 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, Argonne, IL.

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, Argonne, IL. To appear in SIAM J. Sci. Comput.

REIGHEL, L. AND GRAGG, W. B. 1990. Algorithm 686: FORTRAN subroutines for updating the QR decomposition. *ACM Trans. Math. Softw. 16*, 4 (Dec.), 369–377.

SCHREIBER, R. AND VAN LOAN, C. 1989. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Stat. Comput. 10*, 1 (Jan.), 53–57.

STEWART, G. W. 1984. Rank degeneracy. *SIAM J. Sci. Stat. Comput. 5*, 403–413.

STEWART, G. W. 1990. An updating algorithm for subspace tracking. Tech. Rep. UMIACS-TR-90-86. Department of Computer Science, University of Maryland, College Park, MD.

STEWART, G. W. 1992. Determining rank in the presence of error. UMIACS TR-92-108. Department of Computer Science, University of Maryland, College Park, MD.

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

Received: March 1996; revised: January 1997; accepted: October 1997