SIAM J. SCI. COMPUT.
Vol. 17, No. 4, pp. 848–869, July 1996

004

EFFICIENT ALGORITHMS FOR COMPUTING A STRONG RANK-REVEALING QR FACTORIZATION*

MING GU† AND STANLEY C. EISENSTAT‡

Abstract. Given an $m \times n$ matrix M with $m \geq n$, it is shown that there exists a permutation Π and an integer k such that the QR factorization

$$M\,\Pi = Q \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$$

reveals the numerical rank of M: the $k \times k$ upper-triangular matrix A_k is well conditioned, $\|C_k\|_2$ is small, and B_k is linearly dependent on A_k with coefficients bounded by a low-degree polynomial in n. Existing rank-revealing QR (RRQR) algorithms are related to such factorizations and two algorithms are presented for computing them. The new algorithms are nearly as efficient as QR with column pivoting for most problems and take $O(mn^2)$ floating-point operations in the worst case.

Key words. orthogonal factorization, rank-revealing factorization, numerical rank

AMS subject classifications. 65F25, 15A23, 65F35

1. Introduction.

Given a matrix $M \in \mathbf{R}^{m\times n}$ with $m \geq n$, we consider partial QR factorizations of the form

$$M\,\Pi = QR \equiv Q \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}, \tag{1}$$

where $Q \in \mathbf{R}^{m\times m}$ is orthogonal, $A_k \in \mathbf{R}^{k\times k}$ is upper triangular with nonnegative diagonal elements, $B_k \in \mathbf{R}^{k\times(n-k)}$, $C_k \in \mathbf{R}^{(m-k)\times(n-k)}$, and $\Pi \in \mathbf{R}^{n\times n}$ is a permutation matrix chosen to reveal linear dependence among the columns of M. Usually k is chosen to be the smallest integer $1 \leq k \leq n$ for which $\|C_k\|_2$ is sufficiently small [24, p. 235].

Golub [20] introduced these factorizations and, with Businger [13], developed the first algorithm (QR with column pivoting) for computing them. Applications include least-squares computations [11, 12, 17, 20, 21, 23, 36], subset selection and linear dependency analysis [12, 18, 22, 34, 44], subspace tracking [7], rank determination [10, 39], and nonsymmetric eigenproblems [2, 15, 26, 35]. Such factorizations are also related to condition estimation [4, 5, 25, 40] and the URV and ULV decompositions [14, 41, 42].

1.1. RRQR factorizations.

By the interlacing property [24, Cor. 8.3.3], for *any* permutation Π we have[1]

$$\sigma_i(A_k) \leq \sigma_i(M) \quad \text{and} \quad \sigma_j(C_k) \geq \sigma_{k+j}(M) \tag{2}$$

for $1 \leq i \leq k$ and $1 \leq j \leq n-k$. Thus,

$$\sigma_{\min}(A_k) \leq \sigma_k(M) \quad \text{and} \quad \sigma_{\max}(C_k) \geq \sigma_{k+1}(M). \tag{3}$$

Assume that $\sigma_k(M) \gg \sigma_{k+1}(M) \approx 0$, so that the numerical rank of M is k. Then we would like to find a Π for which $\sigma_{\min}(A_k)$ is sufficiently large and $\sigma_{\max}(C_k)$ is sufficiently

*Received by the editors May 13, 1994; accepted for publication (in revised form) March 8, 1995. This research was supported in part by U. S. Army Research Office contract DAAL03-91-G-0032.

†Department of Mathematics and Lawrence Berkeley Laboratory, University of California, Berkeley, CA 94720 (minggu@math.berkeley.edu).

‡Department of Computer Science, Yale University, P. O. Box 208285, New Haven, CT 06520-8285 (eisenstat-stan@cs.yale.edu).

[1]Here $\sigma_i(X)$, $\sigma_{\max}(X)$, and $\sigma_{\min}(X)$ denote the ith largest, the largest, and the smallest singular values of the matrix X, respectively.

small. We call the factorization (1) a rank-revealing QR (RRQR) factorization if it satisfies (cf. (3))

$$\sigma_{\min}(A_k) \geq \frac{\sigma_k(M)}{p(k,n)} \quad \text{and} \quad \sigma_{\max}(C_k) \leq \sigma_{k+1}(M)\ p(k,n), \tag{4}$$

where $p(k,n)$ is a function bounded by a low-degree polynomial in k and n [13, 28]. Other, less restrictive definitions are discussed in [13] and [37]. The term "rank-revealing QR factorization" is due to Chan [10].

The Businger and Golub algorithm [8, 20] works well in practice, but there are examples where it fails to produce a factorization satisfying (4) (see Example 1 in §2). Other algorithms fail on similar examples [13]. Recently, Hong and Pan [28] showed that there exist RRQR factorizations with $p(k,n) = \sqrt{k(n-k) + \min(k, n-k)}$, and Chandrasekaran and Ipsen [13] developed an algorithm that computes one efficiently in practice,[2] given k.

1.2. Strong RRQR factorizations. In some applications it is necessary to find a basis for the approximate right null space of M, as in rank-deficient least-squares computations [23, 24] and subspace tracking [7], or to separate the linearly independent columns of M from the linearly dependent ones, as in subset selection and linear dependency analysis [12, 18, 22, 34, 44]. The RRQR factorization does not lead to a stable algorithm because the elements of $A_k^{-1}B_k$ can be very large (see Example 2 in §2).

In this paper we show that there exist QR factorizations that meet this need. We call the factorization (1) a *strong* RRQR factorization if it satisfies (cf. (2))

$$\sigma_i(A_k) \geq \frac{\sigma_i(M)}{q_1(k,n)} \quad \text{and} \quad \sigma_j(C_k) \leq \sigma_{k+j}(M)\ q_1(k,n) \tag{5}$$

and

$$\left|\left(A_k^{-1}B_k\right)_{i,j}\right| \leq q_2(k,n), \tag{6}$$

for $1 \leq i \leq k$ and $1 \leq j \leq n-k$, where $q_1(k,n)$ and $q_2(k,n)$ are functions bounded by low-degree polynomials in k and n. Clearly a strong RRQR factorization is also a RRQR factorization. In addition, condition (6) makes

$$\Pi \begin{pmatrix} -A_k^{-1}B_k \\ I_{n-k} \end{pmatrix}$$

an approximate right null space of M with a small residual independent of the condition number of A_k, provided that A_k is not too ill conditioned [38, pp. 192–198]. See [26] for another application.

We show that there exists a permutation Π for which conditions (5) and (6) hold with

$$q_1(k,n) = \sqrt{1 + k(n-k)} \quad \text{and} \quad q_2(k,n) = 1.$$

Since this permutation might take exponential time to compute, we present algorithms that, given $f \geq 1$, find a Π for which (5) and (6) hold with

$$q_1(k,n) = \sqrt{1 + f^2 k(n-k)} \quad \text{and} \quad q_2(k,n) = f.$$

Here k can be either an input parameter (Algorithm 4) or the smallest integer for which $\sigma_{\max}(C_k)$ is sufficiently small (Algorithm 5). When $f > 1$, these algorithms require $O\left((m + n\log_f n)n^2\right)$ floating-point operations. In particular, when f is a small power of n (e.g., $\sqrt{n}$ or n), they take $O(mn^2)$ time (see §4.4).

[2] In the worst case the runtime might be exponential in k or n. The algorithm proposed by Golub, Klema, and Stewart [22] also computes an RRQR factorization [30], but requires an orthogonal basis for the right null space.

Recently, Pan and Tang [37] presented an algorithm that, given $f > 1$, computes an RRQR factorization with $p(k,n) = f\sqrt{k(n-k) + \max(k, n-k)}$. This algorithm can be shown to be mathematically equivalent to Algorithm 5 and thus computes a strong RRQR factorization with $q_1(k,n) = \sqrt{1 + f^2 k(n-k)}$ and $q_2(k,n) = f$. However, it is much less efficient. Pan and Tang [37] also present two practical modifications to their algorithm, but they do not always compute strong RRQR factorizations.

1.3. Overview. In §2 we review QR with column pivoting [8, 20] and the Chandrasekaran and Ipsen algorithm [13] for computing an RRQR factorization. In §3 we give a constructive existence proof for the strong RRQR factorization. In §4 we present an algorithm (Algorithm 5) that computes a strong RRQR factorization and bound the total number of operations required when $f > 1$; and in §5 we show that this algorithm is numerically stable. In §6 we report the results of some numerical experiments. In §7 we show that the concept of a strong RRQR factorization is not completely new in that the QR factorization given by the Businger and Golub algorithm [8, 20] satisfies (5) and (6) with $q_1(k,n)$ and $q_2(k,n)$ functions that grow exponentially with k. Finally, in §8 we present some extensions of this work, including a version of Algorithm 5 that is nearly as fast as QR with column pivoting for most problems and takes $O(mn^2)$ floating-point operations in the worst case.

1.4. Notation. By convention, A_k, $\bar{A}_k \in \mathbf{R}^{k\times k}$ denote upper-triangular matrices with nonnegative diagonal elements, and B_k, $\bar{B}_k \in \mathbf{R}^{k\times(n-k)}$ and C_k, $\bar{C}_k \in \mathbf{R}^{(m-k)\times(n-k)}$ denote general matrices.

In the partial QR factorization

$$X = Q \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$$

of a matrix $X \in \mathbf{R}^{m\times n}$ (where the diagonal elements of A_k are nonnegative), we write

$$\mathcal{A}_k(X) = A_k, \quad \mathcal{C}_k(X) = C_k, \quad \text{and} \quad \mathcal{R}_k(X) = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}.$$

For A, a nonsingular $\ell \times \ell$ matrix, $1/\omega_i(A)$ denotes the 2-norm of the ith *row* of A^{-1} and $\omega_*(A) = (\omega_1(A), \ldots, \omega_\ell(A))^T$. For C, a matrix with ℓ columns, $\gamma_j(C)$ denotes the 2-norm of the jth *column* of C and $\gamma_*(C) = (\gamma_1(C), \ldots, \gamma_\ell(C))$.

$\Pi_{i,j}$ denotes the permutation that interchanges the ith and jth columns of a matrix.

A *flop* is a floating-point operation $\alpha \circ \beta$, where α and β are floating-point numbers and $\circ$ is one of $+$, $-$, $\times$, and $\div$. Taking the absolute value or comparing two floating-point numbers is also counted as a flop.

2. RRQR algorithms. QR with column pivoting [8, 20] is a modification of the ordinary QR algorithm.

ALGORITHM 1. QR with column pivoting.

$k := 0$; $R := M$; $\Pi := I$;
while $\max_{1\le j\le n-k} \gamma_j(\mathcal{C}_k(R)) \ge \delta$ **do**
  $j_{\max} := \operatorname{argmax}_{1\le j\le n-k} \gamma_j(\mathcal{C}_k(R))$;
  $k := k + 1$;
  Compute $R := \mathcal{R}_k(R\,\Pi_{k,k+j_{\max}-1})$ and $\Pi := \Pi\,\Pi_{k,k+j_{\max}-1}$;
endfor;

When Algorithm 1 halts, we have

$$\sigma_{\max}(\mathcal{C}_k(M\,\Pi)) \le \sqrt{n-k} \max_{1\le j\le n-k} \gamma_j(\mathcal{C}_k(M\,\Pi)) \le \sqrt{n-k}\,\delta,$$

and if δ is sufficiently small, then the numerical rank of M is at most k. If the vector of column norms $\gamma_*(\mathcal{C}_k(R))$ is updated rather than recomputed from scratch each time, then Algorithm 1 takes about $4mnk - 2k^2(m+n) + 4k^3/3$ flops [24, p. 236].

Algorithm 1 uses a greedy strategy for finding well-conditioned columns: having determined the first k columns, it picks a column from the remaining $n-k$ columns that maximizes $\det[\mathcal{A}_{k+1}(R)]$ (see [13]). When there are only a few well-conditioned columns, this strategy is guaranteed to find a strong RRQR factorization (see §7). It also works well in general, but it fails to find an RRQR factorization for the following example.

Example 1 (Kahan [33]). Let $M = S_n K_n$, where

$$S_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \varsigma & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \varsigma^{n-1} \end{pmatrix} \quad \text{and} \quad K_n = \begin{pmatrix} 1 & -\varphi & \cdots & -\varphi \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\varphi \\ 0 & \cdots & 0 & 1 \end{pmatrix}, \tag{7}$$

with $\varphi, \varsigma > 0$ and $\varphi^2 + \varsigma^2 = 1$. Let $k = n-1$. Then Algorithm 1 does not permute the columns of M, yet it can be shown that

$$\frac{\sigma_k(M)}{\sigma_{\min}(A_k)} \geq \frac{\varphi^3(1+\varphi)^{n-4}}{2\varsigma},$$

and the right-hand side grows faster than any polynomial in k and n.

When $m = n$ and the numerical rank of M is close to n, Stewart [39] suggests applying Algorithm 1 to M^{-1}. Recently, Chandrasekaran and Ipsen [13] combined these ideas to construct an algorithm Hybrid-III(k) that is guaranteed to find an RRQR factorization, given k. We present it in a different form here to motivate our constructive proof of the existence of a strong RRQR factorization.

ALGORITHM 2. Hybrid-III(k).

$R := M$; $\Pi := I$;
repeat
  $i_{\min} := \operatorname{argmin}_{1 \leq i \leq k} \omega_i(\mathcal{A}_k(R))$;
  if there exists a j such that $\det[\mathcal{A}_k(R\,\Pi_{i_{\min}, j+k})] / \det[\mathcal{A}_k(R)] > 1$ **then**
    Find such a j;
    Compute $R := \mathcal{R}_k(R\,\Pi_{i_{\min}, j+k})$ and $\Pi := \Pi\,\Pi_{i_{\min}, j+k}$;
  endif;
  $j_{\max} := \operatorname{argmax}_{1 \leq j \leq n-k} \gamma_j(\mathcal{C}_k(R))$;
  if there exists an i such that $\det[\mathcal{A}_k(R\,\Pi_{i, j_{\max}+k})] / \det[\mathcal{A}_k(R)] > 1$ **then**
    Find such an i;
    Compute $R := \mathcal{R}_k(R\,\Pi_{i, j_{\max}+k})$ and $\Pi := \Pi\,\Pi_{i, j_{\max}+k}$;
  endif;
until no interchange occurs;

Since the objective is to find a permutation Π for which $\sigma_{\min}(\mathcal{A}_k(M\,\Pi))$ is sufficiently large and $\sigma_{\max}(\mathcal{C}_k(M\,\Pi))$ is sufficiently small, Algorithm 2 keeps interchanging the most "dependent" of the first k columns (column $i_{\min}$) with one of the last $n-k$ columns, and interchanging the most "independent" of the last $n-k$ columns (column $j_{\max}$) with one of the first k columns, as long as $\det[\mathcal{A}_k(R)]$ strictly increases.

Since $\det[\mathcal{A}_k(R)]$ strictly increases with every interchange, no permutation repeats; and since there are only a finite number of permutations, Algorithm 2 eventually halts. Chandrasekaran and Ipsen [13] also show that it computes an RRQR factorization, given k. Due to efficiency considerations, they suggest that it be run as a postprocessor to Algorithm 1.

But Algorithm 2 may not compute a strong RRQR factorization either.

Example 2. Let $k = n - 2$ and let

$$M \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} = \begin{pmatrix} S_{k-1}K_{k-1} & 0 & 0 & -\varphi S_{k-1}c_{k-1} \\ & \mu & 0 & 0 \\ & & \mu & 0 \\ & & & \mu \end{pmatrix},$$

where S_{k-1} and K_{k-1} are defined as in (7), $c_{k-1} = (1, \ldots, 1)^T \in \mathbf{R}^{k-1}$, and

$$\mu = \frac{1}{\sqrt{k}} \min_{1 \le i \le k-1} \omega_i(S_{k-1}K_{k-1}).$$

Then Algorithm 2 does not permute the columns of M (note that $i_{\min} = k$ and $j_{\max} = k + 1$), yet it can be shown that

$$\frac{\sigma_{k-1}(M)}{\sigma_{k-1}(A_k)} \ge \frac{\varphi^3(1+\varphi)^{k-4}}{2\varsigma} \quad \text{and} \quad \|A_k^{-1}B_k\|_\infty = \varphi(1+\varphi)^{k-2},$$

and the right-hand sides grow faster than any polynomial in k and n.

Since Algorithm 1 does not permute the columns of M, this example also shows that Algorithm 2 may not compute a strong RRQR factorization even when it is run as a postprocessor to Algorithm 1.

3. The existence of a strong RRQR factorization. A strong RRQR factorization satisfies three conditions: *every* singular value of A_k is sufficiently large, *every* singular value of C_k is sufficiently small, and *every* element of $A_k^{-1}B_k$ is bounded. Since

$$\det(A_k) = \prod_{i=1}^{k} \sigma_i(A_k) = \sqrt{\det(M^T M)} \Big/ \prod_{j=1}^{n-k} \sigma_j(C_k),$$

a strong RRQR factorization also results in a large $\det(A_k)$. Given k and $f \ge 1$, Algorithm 3 below[3] constructs a strong RRQR factorization by using column interchanges to try to maximize $\det(A_k)$.

ALGORITHM 3. Compute a strong RRQR factorization, given k.

$R := \mathcal{R}_k(M)$; $\Pi := I$;

while there exist i and j such that $\det(\bar{A}_k))/\det(A_k) > f$,

where $R = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$ and $\mathcal{R}_k(R\,\Pi_{i,j+k}) = \begin{pmatrix} \bar{A}_k & \bar{B}_k \\ & \bar{C}_k \end{pmatrix}$, **do**

Find such an i and j;

Compute $R := \mathcal{R}_k(R\,\Pi_{i,j+k})$ and $\Pi := \Pi\,\Pi_{i,j+k}$;

endwhile;

While Algorithm 2 interchanges either the most "dependent" column of A_k or the most "independent" column of C_k, Algorithm 3 interchanges any pair of columns that sufficiently increases $\det(A_k)$. As before, there are only a finite number of permutations and none can repeat, so that it eventually halts.

[3]The algorithms in this section are only intended to prove the existence of a strong RRQR factorization. Efficient algorithms will be presented in §§4 and 8.

To prove that Algorithm 3 computes a strong RRQR factorization, we first express $\det(\bar{A}_k)/\det(A_k)$ in terms of $\omega_i(A_k)$, $\gamma_j(C_k)$, and $\left(A_k^{-1}B_k\right)_{i,j}$.

LEMMA 3.1. *Let*

$$R = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} \quad \textit{and} \quad \mathcal{R}_k(R\,\Pi_{i,j+k}) = \begin{pmatrix} \bar{A}_k & \bar{B}_k \\ & \bar{C}_k \end{pmatrix},$$

where A_k has positive diagonal elements. Then

$$\frac{\det(\bar{A}_k)}{\det(A_k)} = \sqrt{\left(A_k^{-1}B_k\right)_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2}.$$

Proof. First, assume that $i < k$ or that $j > 1$. Let $A_k\,\Pi_{i,k} = \tilde{Q}\tilde{A}_k$ be the QR factorization of $A_k\,\Pi_{i,k}$, let $\tilde{B}_k = \tilde{Q}^T B_k\,\Pi_{1,j}$ and $\tilde{C}_k = C_k\,\Pi_{1,j}$, and let $\tilde{\Pi} = \operatorname{diag}(\Pi_{i,k}, \Pi_{1,j})$. Then

$$R\,\tilde{\Pi} \equiv \begin{pmatrix} A_k\,\Pi_{i,k} & B_k\,\Pi_{1,j} \\ & C_k\,\Pi_{1,j} \end{pmatrix} = \begin{pmatrix} \tilde{Q} & \\ & I_{m-k} \end{pmatrix} \begin{pmatrix} \tilde{A}_k & \tilde{B}_k \\ & \tilde{C}_k \end{pmatrix}$$

is the QR factorization of $R\,\tilde{\Pi}$. Since both A_k and $\tilde{A}_k$ have positive diagonal elements, we have $\det(A_k) = \det(\tilde{A}_k)$. Since $\tilde{A}_k^{-1}\tilde{B}_k = \Pi_{i,k}^T A_k^{-1} B_k \Pi_{1,j}$, we have $(A_k^{-1}B_k)_{i,j} = (\tilde{A}_k^{-1}\tilde{B}_k)_{k,1}$. Since $\tilde{A}_k^{-1} = \Pi_{i,k}^T A_k^{-1} B_k \tilde{Q}$ and postmultiplication by an orthogonal matrix leaves the 2-norms of the rows unchanged, we have $\omega_i(A_k) = \omega_k(\tilde{A}_k)$. Finally, we have $\gamma_j(C_k) = \gamma_1(\tilde{C}_k)$. Thus it suffices to consider the special case $i = k$ and $j = 1$.

Partition

$$\mathcal{R}_{k+1}(R) = \begin{pmatrix} A_{k-1} & b_1 & b_2 & B \\ & \gamma_1 & \beta & c_1^T \\ & & \gamma_2 & c_2^T \\ & & & C_{k+1} \end{pmatrix}.$$

Then $\omega_i(A_k) = \gamma_1$, $\gamma_j(C_k) = \gamma_2$, and $\left(A_k^{-1}B_k\right)_{i,j} = \beta/\gamma_1$. But $\det(A_k) = \det(A_{k-1})\,\gamma_1$ and $\det(\bar{A}_k) = \det(A_{k-1})\sqrt{\beta^2 + \gamma_2^2}$, so that

$$\frac{\det(\bar{A}_k)}{\det(A_k)} = \sqrt{(\beta/\gamma_1)^2 + (\gamma_2/\gamma_1)^2} = \sqrt{\left(A_k^{-1}B_k\right)_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2},$$

which is the result required. □

Let

$$\rho(R,k) = \max_{1\le i\le k,\ 1\le j\le n-k} \sqrt{\left(A_k^{-1}B_k\right)_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2}.$$

Then by Lemma 3.1, Algorithm 3 can be rewritten as the following.

ALGORITHM 4. Compute a strong RRQR factorization, given k.

Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(M)$ and $\Pi = I$;

while $\rho(R,k) > f$ **do**

Find i and j such that $\sqrt{\left(A_k^{-1}B_k\right)_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2} > f$;

Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\,\Pi_{i,j+k})$ and $\Pi := \Pi\,\Pi_{i,j+k}$;

endwhile;

Since Algorithm 4 is equivalent to Algorithm 3, it eventually halts and finds a permutation Π for which $\rho(\mathcal{R}_k(M\,\Pi), k) \le f$. This implies (6) with $q_2(k,n) = f$. Now we show that this also implies (5) with $q_1(k,n) = \sqrt{1 + f^2k(n-k)}$, i.e., that Algorithms 3 and 4 compute a strong RRQR factorization, given k.

THEOREM 3.2. *Let*

$$R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} = \mathcal{R}_k(M\,\Pi)$$

satisfy $\rho(R,k) \le f$. *Then*

$$\sigma_i(A_k) \ge \frac{\sigma_i(M)}{\sqrt{1 + f^2k(n-k)}}, \qquad 1 \le i \le k, \tag{8}$$

and

$$\sigma_j(C_k) \le \sigma_{j+k}(M)\sqrt{1 + f^2k(n-k)}, \qquad 1 \le j \le n-k. \tag{9}$$

Proof. For simplicity we assume that M (and therefore R) has full column rank. Let $\alpha = \sigma_{\max}(C_k)/\sigma_{\min}(A_k)$, and write

$$R = \begin{pmatrix} A_k & \\ & C_k/\alpha \end{pmatrix} \begin{pmatrix} I_k & A_k^{-1}B_k \\ & \alpha I_{n-k} \end{pmatrix} \equiv \tilde{R}_1 W_1.$$

Then by [29, Thm. 3.3.16],

$$\sigma_i(R) \le \sigma_i(\tilde{R}_1)\,\|W_1\|_2, \qquad 1 \le i \le n. \tag{10}$$

Since $\sigma_{\min}(A_k) = \sigma_{\max}(C_k/\alpha)$, we have $\sigma_i(\tilde{R}_1) = \sigma_i(A_k)$ for $1 \le i \le k$. Moreover,

$$\begin{aligned} \|W_1\|_2^2 &\le 1 + \|A_k^{-1}B_k\|_2^2 + \alpha^2 \\ &= 1 + \|A_k^{-1}B_k\|_2^2 + \|C_k\|_2^2\|A_k^{-1}\|_2^2 \\ &\le 1 + \|A_k^{-1}B_k\|_F^2 + \|C_k\|_F^2\|A_k^{-1}\|_F^2 \\ &= 1 + \sum_{i=1}^{k}\sum_{j=1}^{n-k}\left\{\left(A_k^{-1}B_k\right)_{i,j}^2 + \gamma_j(C_k)^2/\omega_i(A_k)^2\right\} \\ &\le 1 + f^2k(n-k), \end{aligned}$$

so that $\|W_1\|_2 \le \sqrt{1 + f^2k(n-k)}$. Plugging these relations into (10), we get (8). Similarly, let

$$\tilde{R}_2 \equiv \begin{pmatrix} \alpha A_k & \\ & C_k \end{pmatrix} = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} \begin{pmatrix} \alpha I_k & -A_k^{-1}B_k \\ & I_{n-k} \end{pmatrix} \equiv R W_2.$$

Then

$$\sigma_j(C_k) = \sigma_{j+k}(\tilde{R}_2) \le \sigma_{j+k}(R)\|W_2\|_2 \le \sigma_{j+k}(M)\sqrt{1 + f^2k(n-k)},$$

which is (9). □

4. Computing a strong RRQR factorization. Given $f \ge 1$ and a tolerance $\delta > 0$, Algorithm 5 below computes both k and a strong RRQR factorization. It is a combination of the ideas in Algorithms 1 and 4 but uses

$$\hat{\rho}(R,k) = \max_{1\le i\le k,\ 1\le j\le n-k} \max\left\{\left|\left(A_k^{-1}B_k\right)_{i,j}\right|,\ \gamma_j(C_k)/\omega_i(A_k)\right\}$$

instead of $\rho(R,k)$ and computes $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$ recursively for greater efficiency.

ALGORITHM 5. Compute k and a strong RRQR factorization.

$k := 0$; $R \equiv C_k := M$; $\Pi := I$;
Initialize $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$;
while $\max_{1\le j\le n-k} \gamma_j(C_k) \ge \delta$ **do**
  $j_{\max} := \operatorname{argmax}_{1\le j\le n-k} \gamma_j(C_k)$;
  $k := k + 1$;
  Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\,\Pi_{k,k+j_{\max}-1})$ and $\Pi := \Pi\,\Pi_{k,k+j_{\max}-1}$;
  Update $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$;
  while $\hat{\rho}(R,k) > f$ **do**
    Find i and j such that $\left|\left(A_k^{-1}B_k\right)_{i,j}\right| > f$ or $\gamma_j(C_k)/\omega_i(A_k) > f$;
    Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\,\Pi_{i,j+k})$ and $\Pi := \Pi\,\Pi_{i,j+k}$;
    Modify $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$;
  endwhile;
endwhile;

Since its inner **while**-loop is essentially equivalent to Algorithm 4, Algorithm 5 must eventually halt, having found k and a permutation Π for which $\hat{\rho}(R,k) \le f$. This implies that $\rho(\mathcal{R}_k(M\,\Pi),k) \le \sqrt{2}\,f$, so that (5) and (6) are satisfied with[4] $q_1(k,n) = \sqrt{1+2f^2k(n-k)}$ and $q_2(k,n) = \sqrt{2}f$.

Remark 1. Note that

$$\frac{\sigma_{k+1}(M)}{\sigma_k(M)} \ge \frac{1}{q_1(k,n)^2}\,\frac{\sigma_{\max}(C_k)}{\sigma_{\min}(A_k)} \ge \frac{1}{q_1(k,n)^2} \max_{1\le i\le k,\ 1\le j\le n-k} \frac{\gamma_j(C_k)}{\omega_i(A_k)}$$

and

$$\frac{\sigma_{k+1}(M)}{\sigma_k(M)} \le \frac{\sigma_{\max}(C_k)}{\sigma_{\min}(A_k)} \le \sqrt{k(n-k)} \max_{1\le i\le k,\ 1\le j\le n-k} \frac{\gamma_j(C_k)}{\omega_i(A_k)}.$$

Thus Algorithm 5 can detect a sufficiently large gap in the singular values of M if we change the condition in the outer **while**-loop to

$$\max_{1\le j\le n-k} \gamma_j(C_k) \ge \delta \quad \textbf{or} \quad \max_{1\le i\le k,\ 1\le j\le n-k} \gamma_j(C_k)/\omega_i(A_k) \ge \zeta,$$

where ζ is some tolerance. This is useful when solving rank-deficient least-squares problems using RRQR factorizations (see [11, 12] and the references therein).

In §§4.1–4.3 we show how to update A_k, B_k, C_k, $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$ after k increases and to modify them after an interchange. In §4.4 we bound the total number of interchanges and the total number of operations. We will discuss numerical stability in §5.

4.1. Updating formulas. Let

$$R = \begin{pmatrix} A_{k-1} & B_{k-1} \\ & C_{k-1} \end{pmatrix} \quad \text{and} \quad \mathcal{R}_k(R\,\Pi_{k,k+j_{\max}-1}) = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}.$$

Assume that we have already computed A_{k-1}, B_{k-1}, C_{k-1}, $\omega_*(A_{k-1})$, $\gamma_*(C_{k-1})$, and $A_{k-1}^{-1}B_{k-1}$. In this subsection we show how to compute A_k, B_k, C_k, $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$. For simplicity we assume that $j_{\max} = 1$, so that $\gamma_1(C_{k-1}) \ge \gamma_j(C_{k-1})$ for $1 \le j \le n-k+1$.

[4] To get $q_1(k,n) = \sqrt{1+f^2k(n-k)}$ and $q_2(k,n) = f$, replace $\hat{\rho}(R,k)$ by $\rho(R,k)$ or replace f by $f/\sqrt{2}$ (assuming that $f > \sqrt{2}$) in Algorithm 5.

Let $H \in \mathbf{R}^{(m-k)\times(m-k)}$ be an orthogonal matrix that zeroes out the elements below the diagonal in the first column of C_{k-1}, and let

$$B_{k-1} = \begin{pmatrix} b & B \end{pmatrix} \quad \text{and} \quad HC_{k-1} = \begin{pmatrix} \gamma & c^T \\ & C \end{pmatrix},$$

where $\gamma = \gamma_1(C_{k-1})$. Then

$$\begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} = \begin{pmatrix} A_{k-1} & b & B \\ & \gamma & c^T \\ & & C \end{pmatrix},$$

so that

$$A_k = \begin{pmatrix} A_{k-1} & b \\ & \gamma \end{pmatrix}, \quad B_k = \begin{pmatrix} B \\ c^T \end{pmatrix}, \quad \text{and} \quad C_k = C.$$

Let $A_{k-1}^{-1} B_{k-1} = \begin{pmatrix} u & U \end{pmatrix}$. Then

$$A_k^{-1} = \begin{pmatrix} A_{k-1}^{-1} & -u/\gamma \\ & 1/\gamma \end{pmatrix} \quad \text{and} \quad A_k^{-1} B_k = \begin{pmatrix} U - uc^T/\gamma \\ c^T/\gamma \end{pmatrix}.$$

Let $u = (\mu_1, \ldots, \mu_{k-1})^T$ and $c = (\nu_1, \ldots, \nu_{n-k})^T$. Then $\omega_*(A_k)$ and $\gamma_*(C_k)$ can be computed from

$$\omega_k(A_k) = \gamma \quad \text{and} \quad 1/\omega_i(A_k)^2 = 1/\omega_i(A_{k-1})^2 + \mu_i^2/\gamma^2, \quad 1 \le i \le k-1$$

and

$$\gamma_j(C_k)^2 = \gamma_{j+1}(C_{k-1})^2 - \nu_j^2, \quad 1 \le j \le n-k.$$

The main cost of the updating procedure is in computing HC_{k-1} and $U - uc^T/\gamma$, which take about $4(m-k)(n-k)$ and $2k(n-k)$ flops, respectively, for a total of about $2(2m-k)(n-k)$ flops.

Remark 2. Since $f \ge 1$, $\rho(R, k-1) \le \sqrt{2}\, f$, and $\gamma \ge \gamma_{j+1}(C_{k-1}) \ge \nu_j$, for $1 \le j \le n-k$, we have

$$\left| \left(A_k^{-1} B_k\right)_{i,j} \right| \le 2f \quad \text{and} \quad \gamma_j(C_k)/\omega_i(A_k) \le \sqrt{2}\, f,$$

so that

$$\rho(\mathcal{R}_k(R\,\Pi_{k,j_{\max}}), k) \le \sqrt{6}\, f.$$

This bound will be used in §5.1.

4.2. Reducing a general interchange to a special one. Assume that there is an interchange between the ith and $(j+k)$th columns of R. In this subsection we show how to reduce this to the special case $i = k$ and $j = 1$.

Let

$$R = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}.$$

If $j > 1$, then interchange the $(k+1)$st and $(k+j)$th columns of R. This only interchanges the corresponding columns in B_k, C_k, $\gamma_*(C_k)$, and $A_k^{-1} B_k$. Henceforth we assume that $i < k$ and $j = 1$.

Partition

$$A_k = \begin{pmatrix} A_{1,1} & a_1 & A_{1,2} \\ & \alpha & a_2^T \\ & & A_{2,2} \end{pmatrix},$$

where $A_{1,1} \in \mathbf{R}^{(i-1)\times(i-1)}$ and $A_{2,2} \in \mathbf{R}^{(k-i)\times(k-i)}$ are upper triangular. Let Π_k be the permutation that cyclically shifts the last $k-i+1$ columns of A_k to the left, so that

$$A_k \Pi_k = \begin{pmatrix} A_{1,1} & A_{1,2} & a_1 \\ & a_2^T & \alpha \\ & A_{2,2} & \end{pmatrix}.$$

Note that $A_k \Pi_k$ is an upper-Hessenberg matrix with nonzero subdiagonal elements in columns $i, i+1, \ldots, k-1$.

To retriangularize $A_k \Pi_k$, we apply Givens rotations to successively zero out the nonzero subdiagonal elements in columns $i, i+1, \ldots, k-1$ (see [19, 24]). Let Q_k^T be the product of these Givens rotations, so that $Q_k^T A_k \Pi_k$ is upper triangular.

Let $\tilde{\Pi} = \mathrm{diag}(\Pi_k, I_{n-k})$, so that the ith column of R is the kth column of $R\,\tilde{\Pi}$. Then

$$R\,\tilde{\Pi} = \begin{pmatrix} A_k \Pi_k & B_k \\ & C_k \end{pmatrix} \quad \text{and} \quad \mathcal{R}_k(R\,\tilde{\Pi}) \equiv \begin{pmatrix} \bar{A}_k & \bar{B}_k \\ & \bar{C}_k \end{pmatrix} = \begin{pmatrix} Q_k^T A_k \Pi_k & Q_k^T B_k \\ & C_k \end{pmatrix}.$$

Since $\bar{A}_k^{-1} = \Pi_k^T A_k^{-1} Q_k$ and postmultiplication by an orthogonal matrix leaves the 2-norms of the rows unchanged, it follows that

$$\omega_*(\bar{A}_k) = \Pi_k^T\, \omega_*(A_k), \quad \gamma_*(\bar{C}_k) = \gamma_*(C_k), \quad \text{and} \quad \bar{A}_k^{-1}\bar{B}_k = \Pi_k^T \left(A_k^{-1} B_k\right).$$

The main cost of this reduction is in computing $Q_k^T A_k \Pi_k$ and $Q_k^T B_k$, which takes about $3\left((n-i)^2 - (n-k)^2\right) \le 3k(2n-k)$ flops.

4.3. Modifying formulas. In this subsection we show how to modify $A_k, B_k, C_k, \omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$ when there is an interchange between the kth and $(k+1)$st columns of R. We assume that we have already zeroed out the elements below the diagonal in the $(k+1)$st column.

Writing

$$R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} = \begin{pmatrix} A_{k-1} & b_1 & b_2 & B \\ & \gamma & \gamma\mu & c_1^T \\ & & \gamma\nu & c_2^T \\ & & & C_{k+1} \end{pmatrix},$$

we have

$$\mathcal{R}_{k+1}(R\,\Pi_{k,k+1}) \equiv \begin{pmatrix} \bar{A}_k & \bar{B}_k \\ & \bar{C}_k \end{pmatrix} = \begin{pmatrix} A_{k-1} & b_2 & b_1 & B \\ & \bar{\gamma} & \gamma\mu/\rho & \bar{c}_1^T \\ & & \gamma\nu/\rho & \bar{c}_2^T \\ & & & C_{k+1} \end{pmatrix},$$

where $\rho = \sqrt{\mu^2+\nu^2}$, $\bar{\gamma} = \gamma\rho$, $\bar{c}_1 = (\mu c_1 + \nu c_2)/\rho$, and $\bar{c}_2 = (\nu c_1 - \mu c_2)/\rho$.

From the expression for R, we also have

$$A_k^{-1} = \begin{pmatrix} A_{k-1}^{-1} & -u/\gamma \\ & 1/\gamma \end{pmatrix},$$

where $u = A_{k-1}^{-1} b_1$. Since A_{k-1} is upper triangular, we can compute u using back-substitution. Moreover,

$$A_k^{-1} B_k \equiv \begin{pmatrix} u_1 & U \\ \mu & u_2^T \end{pmatrix} = \begin{pmatrix} A_{k-1}^{-1} & -u/\gamma \\ & 1/\gamma \end{pmatrix} \begin{pmatrix} b_2 & B \\ \gamma\mu & c_1^T \end{pmatrix},$$

so that

$$A_{k-1}^{-1} b_2 = u_1 + \mu u \quad \text{and} \quad A_{k-1}^{-1} B = U + u c_1^T/\gamma.$$

It follows that

$$\bar{A}_k^{-1} = \begin{pmatrix} A_{k-1}^{-1} & -A_{k-1}^{-1} b_2/\bar{\gamma} \\ & 1/\bar{\gamma} \end{pmatrix} = \begin{pmatrix} A_{k-1}^{-1} & -(u_1 + \mu u)/\bar{\gamma} \\ & 1/\bar{\gamma} \end{pmatrix}$$

and

$$\begin{aligned} \bar{A}_k^{-1} \bar{B}_k &= \begin{pmatrix} A_{k-1}^{-1} & -(u_1 + \mu u)/\bar{\gamma} \\ & 1/\bar{\gamma} \end{pmatrix} \begin{pmatrix} b_1 & B \\ \gamma\mu/\rho & \bar{c}_1^T \end{pmatrix} \\ &= \begin{pmatrix} \left(1 - \gamma\mu^2/(\bar{\gamma}\rho)\right) u - \left(\gamma\mu/(\bar{\gamma}\rho)\right) u_1 & A_{k-1}^{-1} B - (u_1 + \mu u)\bar{c}_1^T/\bar{\gamma} \\ \gamma\mu/(\bar{\gamma}\rho) & \bar{c}_1^T/\bar{\gamma} \end{pmatrix}. \end{aligned} \tag{11}$$

Simplifying,

$$1 - \gamma\mu^2/(\bar{\gamma}\rho) = 1 - \mu^2/\rho^2 = \nu^2/\rho^2 \quad \text{and} \quad \gamma\mu/(\bar{\gamma}\rho) = \mu/\rho^2.$$

We also have

$$\begin{aligned} A_{k-1}^{-1} B - (u_1 + \mu u)\bar{c}_1^T/\bar{\gamma} &= U + u c_1^T/\gamma - \mu u \bar{c}_1^T/\bar{\gamma} - u_1 \bar{c}_1^T/\bar{\gamma} \\ &= U + u\left(\rho c_1 - \mu \bar{c}_1\right)^T/\bar{\gamma} - u_1 \bar{c}_1^T/\bar{\gamma} \\ &= U + \nu u \bar{c}_2^T/\bar{\gamma} - u_1 \bar{c}_1^T/\bar{\gamma}. \end{aligned}$$

Plugging these relations into (11), we get

$$\bar{A}_k^{-1} \bar{B}_k = \begin{pmatrix} \left(\nu^2 u - \mu u_1\right)/\rho^2 & U + \left(\nu u \bar{c}_2^T - u_1 \bar{c}_1^T\right)/\bar{\gamma} \\ \mu/\rho^2 & \bar{c}_1^T/\bar{\gamma} \end{pmatrix}.$$

Let

$$u = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_{k-1} \end{pmatrix}, \quad u_1 + \mu u = \begin{pmatrix} \bar{\mu}_1 \\ \vdots \\ \bar{\mu}_{k-1} \end{pmatrix}, \quad c_2 = \begin{pmatrix} \nu_2 \\ \vdots \\ \nu_{n-k} \end{pmatrix}, \quad \text{and} \quad \bar{c}_2 = \begin{pmatrix} \bar{\nu}_2 \\ \vdots \\ \bar{\nu}_{n-k} \end{pmatrix}.$$

Then $\omega_*(A_k)$ and $\gamma_*(C_k)$ can be computed from

$$\omega_k(\bar{A}_k) = \bar{\gamma} \quad \text{and} \quad \omega_i(\bar{A}_k)^2 = \omega_i(A_k)^2 + \bar{\mu}_i^2/\bar{\gamma}^2 - \mu_i^2/\gamma^2, \quad 1 \le i \le k-1,$$

and

$$\gamma_1(\bar{C}_k) = \gamma\nu/\rho \quad \text{and} \quad \gamma_j(\bar{C}_k)^2 = \gamma_j(C_k)^2 + \bar{\nu}_j^2 - \nu_j^2, \quad 2 \le j \le n-k.$$

The cost of zeroing out the elements below the diagonal in the $(k+1)$st column is about $4(m-k)(n-k)$ flops, the cost of computing u is about k^2 flops, and the cost of computing $\bar{A}_k^{-1} \bar{B}_k$ is about $4k(n-k)$ flops. Thus the total cost of the modification is about $4m(n-k) + k^2$ flops.

4.4. Efficiency. In this subsection we derive an upper bound on the total number of interchanges and bound the total number of flops. We only consider the case $f > 1$.

Let τ_k be the number of interchanges performed for a particular value of k (i.e., within the inner **while**-loop), and let Δ_k be the determinant of A_k after these interchanges are complete (by convention, $\Delta_0 = 1$). Since $\det(A_k) = \Delta_{k-1}\, \gamma_{j_{\max}}(C_{k-1})$ before the interchanges, and each interchange increases $\det(A_k)$ by at least a factor of f, it follows that

$$\Delta_k \geq \Delta_{k-1}\, \gamma_{j_{\max}}(C_{k-1})\, f^{\tau_k}.$$

By (3), we have

$$\sigma_{l+1}(M) \leq \sigma_{\max}(\mathcal{C}_l(M)) \leq \|\mathcal{C}_l(M)\|_F \leq \sqrt{n-l}\, \gamma_{j_{\max}}(\mathcal{C}_l(M)),$$

for $1 \leq l < n$, so that

$$\Delta_k \geq \Delta_{k-1} \frac{1}{\sqrt{n}} \sigma_k(M)\, f^{\tau_k} \geq \left(\frac{1}{\sqrt{n}}\right)^k \left\{\prod_{i=1}^{k} \sigma_i(M)\right\} f^{t_k},$$

where $t_k = \sum_{i=1}^{k} \tau_i$ is the total number of interchanges up to this point. On the other hand, from (2) we also have

$$\Delta_k = \prod_{i=1}^{k} \sigma_i(A_k) \leq \prod_{i=1}^{k} \sigma_i(M).$$

Combining these relations, we have $f^{t_k} \leq (\sqrt{n})^k$, so that $t_k \leq k \log_f \sqrt{n}$.

The cost of the updating procedure is about $2(2m-k)(n-k)$ flops (see §4.1), the cost of the reduction procedure is at most about $3k(2n-k)$ flops (see §4.2), and the cost of the modifying procedure is about $4m(n-k)+k^2$ flops (see §4.3). For each increase in k and each interchange, the cost of finding $\hat{\rho}(R,k)$ is about $2k(n-k)$ flops (taking $k(n-k)$ absolute values and making $k(n-k)$ comparisons).

Let k_f be the final value of k when Algorithm 5 halts. Then the total number of interchanges t_{k_f} is bounded by $k_f \log_f \sqrt{n}$, which is $O(k_f)$ when f is taken to be a small power of n (e.g., $\sqrt{n}$ or n). Thus the total cost is at most about

$$\begin{aligned}
&\sum_{k=1}^{k_f} [2(2m-k)(n-k) + 2k(n-k)] \\
&\qquad + t_{k_f} \max_{1 \leq k \leq k_f} \left[3k(2n-k) + 4m(n-k) + k^2 + 2k(n-k)\right] \\
&\quad \leq 2mk_f(2n-k_f) + 4t_{k_f} n(m+n)
\end{aligned}$$

flops. When f is taken to be a small power of n (e.g., $\sqrt{n}$ or n), the total cost is $O(mnk_f)$ flops. Normally t_{k_f} is quite small (see §6), and thus the cost is about $2mk_f(2n-k_f)$ flops. When $m \gg n$, Algorithm 5 is almost as fast as Algorithm 1; when $m \approx n$, Algorithm 5 is about 50% more expensive. We will discuss efficiency further in §§6 and 8.

5. Numerical stability. Since we update and modify $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$ rather than recompute them, we might expect some loss of accuracy. But since we only use these quantities for deciding which pairs of columns to interchange, Algorithm 5 could only be unstable if they were extremely inaccurate.

In §5.1 we give an upper bound for $\rho(R,k)$ during the interchanges. Since this bound grows slowly with k, Theorem 3.2 asserts that A_k can never be extremely ill conditioned, provided that $\sigma_k(M)$ is not very much smaller than $\|M\|_2$. This implies that the elements of $A_k^{-1}B_k$ cannot be too inaccurate. In §5.2 we discuss the numerical stability of updating and modifying $\omega_*(A_k)$ and $\gamma_*(C_k)$.

5.1. An upper bound on $\rho(R, k)$ during interchanges. We only consider the case $f > 1$.

LEMMA 5.1. *Let $A, C, U \in \mathbf{R}^{k\times k}$, where A is upper triangular with positive diagonal elements and $U = (u_{i,j})$. If*

$$\sqrt{u_{i,j}^2 + \left(\gamma_j(C)/\omega_i(A)\right)^2} \le f, \qquad 1 \le i, j \le k,$$

then

$$\sqrt{\det\left[(AU)^T AU + C^T C\right]} \le \det(A)\ \left(\sqrt{2k}\, f\right)^k.$$

Proof. First, note that

$$\sqrt{\det\left[(AU)^T AU + C^T C\right]} = \prod_{i=1}^{k} \sigma_i\left(\begin{pmatrix} AU \\ C \end{pmatrix}\right).$$

Let $\alpha = \sigma_{\min}(A)$, and write

$$W \equiv \begin{pmatrix} AU \\ C \end{pmatrix} = \begin{pmatrix} A & \\ & \alpha I_k \end{pmatrix} \begin{pmatrix} U \\ C/\alpha \end{pmatrix} \equiv \tilde{D}\, \tilde{W}.$$

By [29, Thm. 3.3.4], we have

$$\prod_{i=1}^{k} \sigma_i(W) \le \prod_{i=1}^{k} \sigma_i(\tilde{D})\, \sigma_i(\tilde{W}).$$

Since $\sigma_i(\tilde{D}) = \sigma_i(A)$, for $1 \le i \le k$, we have

$$\prod_{i=1}^{k} \sigma_i(\tilde{D}) = \prod_{i=1}^{k} \sigma_i(A) = \det(A).$$

Now, since $\tilde{W}^T \tilde{W}$ is symmetric and positive definite,

$$\prod_{i=1}^{k} \sigma_i(\tilde{W}) = \sqrt{\det(\tilde{W}^T \tilde{W})} \le \sqrt{\prod_{i=1}^{k} (\tilde{W}^T \tilde{W})_{i,i}} = \prod_{i=1}^{k} \|\tilde{W} e_i\|_2,$$

and, since

$$\frac{1}{\alpha} = \|A^{-1}\|_2 \le \sqrt{k} \max_{1\le i\le k} \frac{1}{\omega_i(A)} = \frac{\sqrt{k}}{\min\limits_{1\le i\le k} \omega_i(A)},$$

we have

$$\|\tilde{W} e_j\|_2^2 = \sum_{i=1}^{k} u_{ij}^2 + \frac{\gamma_j(C)^2}{\alpha^2} \le k f^2 + \frac{k\gamma_j(C)^2}{\min\limits_{1\le i\le k} \omega_i(A)^2} \le 2k f^2.$$

The result follows immediately. □

To derive an upper bound on $\rho(R, k)$ during the interchanges, we use techniques similar to those used by Wilkinson [43] to bound the growth factor for Gaussian elimination with complete pivoting.[5] Let

$$\mathcal{W}(\tau) = \left(\tau \prod_{s=2}^{\tau} s^{1/(s-1)}\right)^{1/2},$$

[5]See [13] for a connection between the growth factor for Gaussian elimination with *partial* pivoting and the failure of RRQR algorithms.

which is Wilkinson's upper bound on the growth factor for Gaussian elimination with complete pivoting on a $\tau \times \tau$ matrix. Although $\mathcal{W}(\tau)$ is not a polynomial in τ, it grows rather slowly [43]: $\mathcal{W}(\tau) = O\left(\tau^{1+\frac{1}{4}\log\tau}\right)$.

THEOREM 5.2. *If Algorithm* 5 *performs* τ *interchanges for some* $k > 1$, *then*

$$\rho(\mathcal{R}_k(M\,\Pi), k) \leq 2\sqrt{6}\, f\, (\tau+1)\, \mathcal{W}(\tau+1).$$

Proof. Assume that Algorithm 5 will perform at least one interchange for this value of k; otherwise the result holds trivially.

Let $\Pi^{(l)}$ be the permutation after the first l interchanges, where $0 \leq l \leq \tau+1$. Partition

$$M\,\Pi^{(l)} = \begin{pmatrix} M_k^{(l)} & M_{n-k}^{(l)} \end{pmatrix},$$

where $M_k^{(l)} \in \mathbf{R}^{m\times k}$ and $M_{n-k}^{(l)} \in \mathbf{R}^{m\times(n-k)}$. Assume that $\eta(l,\tau)$ columns of $M_k^{(\tau+1)}$ are from $M_{n-k}^{(l)}$, and that the rest are from $M_k^{(l)}$. Since there are $\tau - l + 1$ more interchanges, we have[6] $\eta(l,\tau) \leq \tau - l + 1$.

Without loss of generality, we assume that the first $k - \eta(l,\tau)$ columns of $M_k^{(\tau+1)}$ are the first $k - \eta(l,\tau)$ columns of $M_k^{(l)}$, and that the last $\eta(l,\tau)$ columns of $M_k^{(\tau+1)}$ are the first $\eta(l,\tau)$ columns of $M_{n-k}^{(l)}$. Then we can write

$$R^{(l)} \equiv \mathcal{R}_k(M\,\Pi^{(l)}) \equiv \begin{pmatrix} A_k^{(l)} & B_k^{(l)} \\ & C_k^{(l)} \end{pmatrix} = \begin{pmatrix} A_{1,1} & A_{1,2} & B_{1,1} & B_{1,2} \\ & A_{2,2} & B_{2,1} & B_{2,2} \\ & & C_{1,1} & C_{1,2} \\ & & & C_{2,2} \end{pmatrix},$$

where $A_{2,2}, C_{1,1} \in \mathbf{R}^{\eta(l,\tau)\times\eta(l,\tau)}$ and the partition is such that

$$R^{(\tau+1)} \equiv \mathcal{R}_k(M\,\Pi^{(\tau+1)}) \equiv \begin{pmatrix} A_k^{(\tau+1)} & B_k^{(\tau+1)} \\ & C_k^{(\tau+1)} \end{pmatrix} = \mathcal{R}_k \begin{pmatrix} A_{1,1} & B_{1,1} & A_{1,2} & B_{1,2} \\ & B_{2,1} & A_{2,2} & B_{2,2} \\ & C_{1,1} & & C_{1,2} \\ & & & C_{2,2} \end{pmatrix}.$$

These relations imply that

$$\det(A_k^{(l)}) = \det(A_{1,1})\ \det(A_{2,2}) \tag{12}$$

and

$$\det(A_k^{(\tau+1)}) = \det(A_{1,1}) \sqrt{\det\left[B_{2,1}^T B_{2,1} + C_{1,1}^T C_{1,1}\right]}. \tag{13}$$

Let $f^{(l)} = \rho(R^{(l)}, k)$. By the definition of $\rho(R,k)$, we have

$$\sqrt{\left(A_{2,2}^{-1}B_{2,1}\right)_{i,j}^2 + \left(\gamma_j(C_{1,1})/\omega_i(A_{2,2})\right)^2} \leq f^{(l)}$$

for $1 \leq i, j \leq \eta(l,\tau)$. Applying Lemma 5.1 and recalling that $\eta(l,\tau) \leq \tau - l + 1$, we have

$$\sqrt{\det\left[B_{2,1}^T B_{2,1} + C_{1,1}^T C_{1,1}\right]} \leq \det(A_{2,2}) \left(\sqrt{2(\tau-l+1)}\ f^{(l)}\right)^{\tau-l+1}.$$

Combining with (12) and (13), we get

$$\det(A_k^{(\tau+1)}) \leq \det(A_k^{(l)}) \left(\sqrt{2(\tau-l+1)}\ f^{(l)}\right)^{\tau-l+1}.$$

[6] It is possible that $\eta(l,\tau) < \tau - l + 1$ since a column may be interchanged more than once.

On the other hand, Algorithm 5 ensures that

$$\det(A_k^{(\tau+1)}) \geq \det(A_k^{(l)}) \left(f^{(l)}/\sqrt{2}\right) \cdots \left(f^{(\tau)}/\sqrt{2}\right).$$

Comparing these two relations, we have

$$f^{(l)} \cdots f^{(\tau)} \leq \left(2\sqrt{\tau - l + 1}\, f^{(l)}\right)^{\tau-l+1}, \quad 0 \leq l \leq \tau. \tag{14}$$

Since

$$\sum_{l=1}^{s-1} \frac{1}{(\tau - l)(\tau - l + 1)} + \frac{1}{\tau} = \frac{1}{\tau - s},$$

taking the product of the $(\tau - l)(\tau - l + 1)$st root of (14) with $l = 1, 2, \ldots, \tau - 1$ and the τth root of (14) with $l = 0$, we have

$$\begin{aligned}\prod_{s=0}^{\tau-1} \left(f^{(s)}\right)^{1/(\tau-s)} f^{(\tau)} &\leq \left(\prod_{l=1}^{\tau-1} \left(2\sqrt{\tau - l + 1}\, f^{(l)}\right)^{1/(\tau-l)}\right) \left(2\sqrt{\tau+1}\, f^{(0)}\right)^{(\tau+1)/\tau} \\ &\leq 2^{1+\sum_{l=0}^{\tau-1} 1/(\tau-l)} \left((\tau+1) \prod_{l=0}^{\tau-1} (\tau - l + 1)^{1/(\tau-l)}\right)^{1/2} \left(\prod_{l=0}^{\tau-1} \left(f^{(l)}\right)^{1/(\tau-l)}\right) f^{(0)},\end{aligned}$$

which simplifies to

$$f^{(\tau)} \leq f^{(0)}\, 2^{1+\sum_{s=1}^{\tau} 1/s} \left((\tau+1) \prod_{s=2}^{\tau+1} s^{1/(s-1)}\right)^{1/2} \leq 2 f^{(0)}\, (\tau+1) \mathcal{W}(\tau+1).$$

Remark 2 at the end of §4.1 implies that $f^{(0)} \leq \sqrt{6}\, f$. Plugging this into the last relation proves the result. □

From §4.4 we have $\tau_k \leq k \log_f \sqrt{n}$. For example, when $\sqrt{n} \leq f \leq n$, we have $\tau_k \leq k$, so that $\rho(R, k) \leq O(n\, k\, \mathcal{W}(k))$.

5.2. Computing the row norms of A_k^{-1} and the column norms of C_k. In this section we discuss the numerical stability of updating and modifying $\omega_*(A_k)$ and $\gamma_*(C_k)$ as a result of interchanges, assuming that f is a small power of n.

For any $\alpha > 0$, we let $\mathcal{O}_n(\alpha)$ denote a positive number that is bounded by α times a slowly increasing function of n. By Theorems 3.2 and 5.2, $\|A_k^{-1}\|_2 = \mathcal{O}_n(1/\sigma_k(M))$ and $\|C_k\|_2 = \mathcal{O}_n(\sigma_{k+1}(M))$ after each interchange. As Algorithm 5 progresses, $\|A_k^{-1}\|_2$ increases from $\mathcal{O}_n(1/\sigma_1(M))$ to $\mathcal{O}_n(1/\sigma_k(M))$, while $\|C_k\|_2$ decreases from $\mathcal{O}_n(\sigma_1(M))$ to $\mathcal{O}_n(\sigma_{k+1}(M))$. A straightforward error analysis shows that the errors in $1/\omega_i(A_k)^2$ and $\gamma_j(C_k)^2$ are bounded by $\mathcal{O}_n\left(\epsilon/\sigma_k^2(M)\right)$ and $\mathcal{O}_n\left(\epsilon\, \sigma_1^2(M)\right)$, respectively, where ϵ is the machine precision. Hence the error in $1/\omega_i(A_k)^2$ is less serious than the error in $\gamma_j(C_k)^2$, which can be larger than $\|C_k\|_2^2$ when $\|C_k\|_2 \leq \mathcal{O}_n\left(\sqrt{\epsilon}\, \sigma_1(M)\right)$.

Algorithm 5 uses the computed values of $\omega_*(A_k)$ and $\gamma_*(C_k)$ only to decide which columns to interchange. But although these values do not need to be very accurate, we do need to avoid the situation where they have no accuracy at all. Thus we recompute rather than update or modify $\gamma_*(C_k)$ when $\max_{1 \leq j \leq n-k} \gamma_j(C_k) = \mathcal{O}_n\left(\sqrt{\epsilon}\, \sigma_1(M)\right)$. This needs to be done at most twice if one wants to compute a strong RRQR factorization with A_k numerically nonsingular. A similar approach is taken in `xgeqpf`, the LAPACK [1] implementation of Algorithm 1.

6. Numerical experiments. In this section we report some numerical results for a Fortran implementation (SRRQR) of Algorithm 5 and the all-Fortran implementation (DGEQPF) of Algorithm 1 in LAPACK [1]. The computations were done on a SPARCstation/10 in double precision where the machine precision is $\epsilon = 1.1 \times 10^{-16}$.

We use the following sets of $n \times n$ test matrices:

1. *Random*: a random matrix with elements uniformly distributed in $[-1, 1]$;
2. *Scaled random*: a random matrix whose ith row is scaled by the factor $\eta^{i/n}$, where $\eta > 0$;
3. *GKS*: an upper-triangular matrix whose jth diagonal element is $1/\sqrt{j}$ and whose (i, j) element is $-1/\sqrt{j}$, for $j > i$ (see Golub, Klema, and Stewart [22]);
4. *Kahan* (see Example 1 in §2);
5. *Extended Kahan*: the matrix $M = S_{3l} R_{3l}$, where

$$S_{3l} = \operatorname{diag}(1, \varsigma, \varsigma^2, \ldots, \varsigma^{3l-1}) \quad \text{and} \quad R_{3l} = \begin{pmatrix} I_l & -\varphi H_l & 0 \\ & I_l & \varphi H_l \\ & & \mu I_l \end{pmatrix};$$

l is a power of 2; $\varsigma > 0$, $\varphi > 1/\sqrt[4]{4l-1}$, and $\varsigma^2 + \varphi^2 = 1$; $0 < \mu \ll 1$; and $H_l \in \mathbf{R}^{l \times l}$ is a symmetric Hadamard matrix (i.e., $H_l^2 = l\, I_l$ and every component of H_l is ± 1).

In particular, we chose $\eta = 20\epsilon$, $\varphi = 0.285$, and $\mu = 20\epsilon/\sqrt{n}$.

In exact arithmetic Algorithm 1 does not perform any interchanges for the Kahan and extended Kahan matrices. To preserve this behavior in DGEQPF we scaled the jth columns of these matrices by $1 - 100j\sqrt{\epsilon}$ and $1 - 10j\epsilon$, respectively, for $1 \le j \le n$. To prevent DGEQPF from taking advantage of the upper-triangular structure we replaced all of the zero entries by random numbers of the order ϵ^2.

For each test matrix, we took $n = 96$, 192, and 384, and set $f = 10\sqrt{n}$ and $\delta \approx 3 \times 10^{-13} \times \|M\|_2$ in SRRQR. For the extended Kahan matrix, we also used $f = \varphi^2 l 1$ and $\delta = 4l^2 \sigma_{2l+1}(M)$; these results are labeled *Extended Kahan**.

The results are summarized in Tables 1 and 2. Execution time is in seconds; rank is the value of k computed by SRRQR; t_{k_f} is the total number of interchanges in the inner **while**-loop of SRRQR; and

$$q_1(k, n) = \sqrt{1 + 2f^2 k(n-k)} \quad \text{and} \quad q_2(k, n) = \begin{cases} f & \text{if } k < n \\ 0 & \text{if } k = n \end{cases}$$

are the theoretical upper bounds on

$$\max_{1 \le i \le k,\ 1 \le j \le n-k} \left(\frac{\sigma_i(M)}{\sigma_i(A_k)}, \frac{\sigma_j(C_k)}{\sigma_{k+j}(M)} \right) \quad \text{and} \quad \max_{1 \le i \le k,\ 1 \le j \le n-k} \left| \left(A_k^{-1} B_k\right)_{i,j} \right|,$$

respectively, for SRRQR.

The execution times confirm that Algorithm 5 is about 50% more expensive than Algorithm 1 on matrices that require only a small number t_{k_f} of interchanges. And as predicted, Algorithm 1 failed to reveal the numerical rank of the Kahan matrix. Finally, the results suggest that the theoretical upper bounds $q_1(k, n)$ and $q_2(k, n)$ are much too large for $0 < k < n$.

For the extended Kahan matrices with $f = \varphi^2 l$ there were no interchanges until the $2l$th step, when the ith column was interchanged with the $(2l + i)$th column for $i = 1, 2, \ldots, l$. These $l = n/3$ column interchanges show that Algorithm 5 may have to perform $O(n)$ interchanges before finding a strong RRQR factorization for a given f (see §4.4) and can be more than twice as expensive as Algorithm 1. However, the extended Kahan matrix is already a strong RRQR factorization with $f = 10\sqrt{n}$ for the values of n used here, which is why no interchanges were necessary.

TABLE 1
SRRQR versus DGEQPF: Execution time.

Matrix type	Order N	Execution time		Rank k	t_{k_f}
		SRRQR	DGEQPF		
Random	96	0.20	0.13	96	0
	192	1.57	0.98	192	0
	384	16.2	11.0	384	0
Scaled random	96	0.19	0.15	74	0
	192	1.48	1.16	147	0
	384	14.5	11.4	290	0
GKS	96	0.20	0.13	95	0
	192	1.58	1.00	191	0
	384	15.5	10.7	383	0
Kahan	96	0.21	0.13	95	1
	192	1.59	0.99	191	1
	384	15.7	10.5	383	1
Extended Kahan	96	0.17	0.15	64	0
	192	1.38	1.16	128	0
	384	13.4	11.4	256	0
Extended Kahan*	96	0.38	0.15	64	32
	192	3.21	1.16	128	64
	384	31.4	11.7	256	128

TABLE 2
SRRQR versus DGEQPF: Bounds.

Matrix type	Order N	$\max_{i,j}\left(\frac{\sigma_i(M)}{\sigma_i(A_k)}, \frac{\sigma_j(C_k)}{\sigma_{k+j}(M)}\right)$			$\max_{i,j}\left\|\left(A_k^{-1}B_k\right)_{i,j}\right\|$		
		SRRQR	$q_1(k,n)$	DGEQPF	SRRQR	$q_2(k,n)$	DGEQPF
Random	96	1	1	1	0	0	0
	192	1	1	1	0	0	0
	384	1	1	1	0	0	0
Scaled random	96	2.93	5.59×10^3	2.38	1.71	98.0	1.75
	192	3.39	1.13×10^4	3.04	1.58	1.96×10^2	1.41
	384	3.75	2.29×10^4	3.76	1.37	3.92×10^2	1.16
GKS	96	1.12	1.35×10^3	1.12	0.71	98.0	0.71
	192	1.09	1.91×10^3	1.09	0.71	1.96×10^2	0.71
	384	1.07	2.71×10^3	1.07	0.71	3.92×10^2	0.71
Kahan	96	1.04	1.35×10^3	1.04×10^{10}	0.78	98.0	4.92×10^9
	192	1.04	1.91×10^3	1.86×10^{17}	0.78	1.96×10^2	1.40×10^{20}
	384	1.04	2.71×10^3	5.98×10^{16}	0.78	3.92×10^2	1.27×10^{23}
Extended Kahan	96	3.22	6.27×10^3	3.22	2.60	98.0	2.60
	192	5.76	1.25×10^4	5.76	5.20	1.96×10^2	5.20
	384	10.9	2.51×10^4	10.9	10.4	3.92×10^2	10.4
Extended Kahan*	96	1.17	1.66×10^2	3.22	0.38	2.60	2.60
	192	1.09	6.65×10^2	5.76	0.19	5.20	5.20
	384	1.05	2.66×10^3	10.9	0.10	10.4	10.4

7. Algorithm 1 and the strong RRQR factorization. Using the techniques developed in §3, we now show that Algorithm 1 satisfies (5) and (6) with $q_1(k,n)$ and $q_2(k,n)$ functions that grow exponentially with k. We need the following lemma.

LEMMA 7.1 (Faddeev, Kublanovskaya, and Faddeeva [16]). *Let $W = (w_{i,j}) \in \mathbf{R}^{n\times n}$ be an upper-triangular matrix with $w_{i,i} = 1$ and $|w_{i,j}| \le 1$ for $1 \le i \le j \le n$. Then*

$$|(W^{-1})_{i,j}| \le 2^{n-2}, \quad 1 \le i, j \le n, \qquad \text{and} \qquad \|W^{-1}\|_F \le \frac{\sqrt{4^n + 6n - 1}}{3}.$$

THEOREM 7.2. *Let* Π *be the permutation chosen by Algorithm* 1, *and let*

$$R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} = \mathcal{R}_k(M\,\Pi).$$

Then

$$\sigma_i(A_k) \geq \frac{\sigma_i(M)}{\sqrt{n-i}\,2^i}, \tag{15}$$

$$\sigma_j(C_k) \leq \sigma_{k+j}(M)\,\sqrt{n-k}\,2^k, \tag{16}$$

and

$$\left|\left(A_k^{-1}B_k\right)_{i,j}\right| \leq 2^{k-i}, \tag{17}$$

for $1 \leq i \leq k$ *and* $1 \leq j \leq n-k$.

Proof. For simplicity we assume that M (and therefore R) has full rank.

Let

$$R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} = D \begin{pmatrix} W_{1,1} & W_{1,2} \\ & W_{2,2} \end{pmatrix} \equiv DW \quad \text{and} \quad \widetilde{W}_j = \begin{pmatrix} W_{1,1} & w_j \\ & 1 \end{pmatrix},$$

where $D = \operatorname{diag}(d_1, d_2, \ldots, d_m)$ is the diagonal of R, $W_{1,1} \in \mathbf{R}^{k\times k}$ is unit upper triangular, $W_{1,2} \in \mathbf{R}^{k\times(n-k)}$, $W_{2,2} \in \mathbf{R}^{(m-k)\times(n-k)}$, and $w_j \in \mathbf{R}^k$ is the jth column of $W_{1,2}$. Since Algorithm 1 would not cause any column interchanges if it were applied to R, it follows that $d_1 \geq d_2 \geq \cdots \geq d_k$ and that no component of $\widetilde{W}_j$ has absolute value larger than 1.

Let $u_{i,j} = \left(A_k^{-1}B_k\right)_{i,j}$. Then $-u_{i,j}$ is the $(i, k+1)$ component of $\widetilde{W}_j^{-1}$. Applying the first result in Lemma 7.1 to the lower right $(k-i+2)\times(k-i+2)$ submatrix of $\widetilde{W}_j$, we have $|u_{i,j}| \leq 2^{k-i}$, which is (17).

As in the proof of Theorem 3.2, let $\alpha = \sigma_{\max}(C_k)/\sigma_{\min}(A_k)$, and write

$$\tilde{R}_2 = \begin{pmatrix} \alpha A_k & \\ & C_k \end{pmatrix} = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} \begin{pmatrix} \alpha I_k & -A_k^{-1}B_k \\ & I_{n-k} \end{pmatrix} \equiv RW_2.$$

Then

$$\sigma_j(C_k) = \sigma_{j+k}(\tilde{R}_2) \leq \sigma_{j+k}(R)\,\|W_2\|_2 = \sigma_{j+k}(M)\,\|W_2\|_2.$$

But

$$\begin{aligned} \|W_2\|_2^2 &\leq 1 + \|A_k^{-1}B_k\|_2^2 + \alpha^2 \\ &\leq 1 + \sum_{i=1}^{k}\sum_{j=1}^{n-k} u_{i,j}^2 + \|C_k\|_F^2\|A_k^{-1}\|_F^2 \\ &= 1 + \sum_{i=1}^{k}\sum_{j=1}^{n-k}\{u_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2\}. \end{aligned}$$

Since $1/\omega_i(A_k) \leq 1/(d_k\omega_i(W_{1,1}))$ and $\gamma_j(C_k) \leq d_k$, we have

$$u_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2 \leq (\widetilde{W}_j^{-1})_{i,k+1}^2 + 1/\omega_i(W_{1,1})^2 = 1/\omega_i(\widetilde{W}_j)^2.$$

Using the second result in Lemma 7.1, it follows that

$$\sum_{i=1}^{k}\left\{u_{i,j}^2 + \left(\gamma_j(C_k)/\omega_i(A_k)\right)^2\right\} \leq \sum_{i=1}^{k} 1/\omega_i(\widetilde{W}_j)^2 = \|\widetilde{W}_j^{-1}\|_F^2 - 1 \leq 4^k - 1,$$

so that $\|W_2\|_2^2 \leq 4^k(n-k)$, which gives (16).

Similarly, writing

$$R = \begin{pmatrix} A_k & \\ & C_k/\alpha \end{pmatrix} \begin{pmatrix} I_k & A_k^{-1}B_k \\ & \alpha I_{n-k} \end{pmatrix} \equiv \tilde{R}_1 W_1,$$

we have

$$\sigma_k(M) = \sigma_k(R) \leq \sigma_k(\tilde{R}_1)\, \|W_1\|_2 \leq \sigma_k(A_k)\, \sqrt{n-k}\, 2^k.$$

Taking $k = i$ and noting that $\sigma_i(A_i) \leq \sigma_i(A_k)$ by the interlacing property of the singular values [24, Cor. 8.3.3], we get (15). □

If R has very few linearly independent columns, then we can stop Algorithm 1 with a small value of k and are guaranteed to have a strong RRQR factorization. Results similar to Theorem 7.2 can be obtained for the RRQR algorithms in [10, 18, 25], and [39].

8. Some extensions. We have proved the existence of a strong RRQR factorization for a matrix $M \in \mathbf{R}^{m\times n}$ with $m \geq n$ and presented an efficient algorithm for computing it. In this section, we describe some further improvements and extensions of these results.

Since Algorithm 1 seems to work well in practice [5, 10, 11, 13], Algorithm 5 tends to perform very few (and quite often no) interchanges in its inner **while**-loop. This suggests using Algorithm 1 as an initial phase (cf. [13] and [37]), and then using Algorithm 4 to remove any dependent columns from A_k, reducing k as needed (cf. [10] and [18]). In many respects the resulting algorithm is equivalent to applying Algorithm 5 to M^{-1} (cf. Stewart [39]).

ALGORITHM 6. Compute k and a strong RRQR factorization.

$k := 0$; $R \equiv C_k := M$; $\Pi := I$;
Compute $\gamma_*(C_k)$;
while $\max_{1\leq j\leq n-k} \gamma_j(C_k) \geq \delta$ **do**
  $j_{\max} := \operatorname{argmax}_{1\leq j\leq n-k}\ \gamma_j(C_k)$;
  $k := k + 1$;
  Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\, \Pi_{k,k+j_{\max}-1})$ and $\Pi := \Pi\, \Pi_{k,k+j_{\max}-1}$;
  Update $\gamma_*(C_k)$;
endwhile;
Compute $\omega_*(A_k)$ and $A_k^{-1}B_k$;
repeat
  while $\hat{\rho}(R,k) > f$ **do**
    Find i and j such that $\left|\left(A_k^{-1}B_k\right)_{i,j}\right| > f$ or $\gamma_j(C_k)/\omega_i(A_k) > f$;
    Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\, \Pi_{i,j+k})$ and $\Pi := \Pi\, \Pi_{i,j+k}$;
    Modify $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$;
  endwhile;
  if $\min_{1\leq i\leq k} \omega_i(A_k) \leq \delta$ **then**
    $i_{\min} := \operatorname{argmin}_{1\leq i\leq k}\ \omega_i(A_k)$;
    $k := k - 1$;
    Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\, \Pi_{i_{\max},k+1})$ and $\Pi := \Pi\, \Pi_{i_{\max},k+1}$;
    Downdate $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$;
  endif;
until k is unchanged;

As before, Algorithm 6 eventually halts and finds k and a strong RRQR factorization. The total number of interchanges t_k is bounded by $(n-k)\log_f \sqrt{n}$, which is $O(n-k)$ when f is taken to be a small power of n (see §4.4). The formulas for downdating $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1}B_k$ are analogous to those in §4.1.

Algorithm 6 initializes $\omega_*(A_k)$ and $A_k^{-1}B_k$ after the first **while**-loop, at a cost of $O(k^2n)$ flops. However, since they are only used to decide which (if any) columns to interchange and whether to decrease k, they do not need to be computed very accurately. To make the algorithm more efficient, we could instead use the condition estimation techniques in [4, 5, 10, 27], and [40] to generate unit vectors u and v such that

$$\|A_k^{-1}u\|_2 \approx \|A_k^{-1}\|_2 \quad \text{and} \quad \|B_k^T A_k^{-T} v\|_2 \approx \|B_k^T A_k^{-T}\|_2 .$$

Let the $i_{\max}$th component of $A_k^{-1}u$ be the largest in absolute value. To find the smallest entry in $\omega_*(A_k)$, we note that

$$1/\omega_{i_{\max}}(A_k) \approx \max_{1\le i\le k} 1/\omega_i(A_k) \approx \left|(A_k^{-1}u)_{i_{\max}}\right| .$$

Similarly, let the $j_{\max}$th component of $B_k^T A_k^{-T} v$ be the largest in absolute value. To find the largest entry of $A_k^{-1}B_k$ in absolute value, we compute the $j_{\max}$th column of $A_k^{-1}B_k$ and look for the largest component in absolute value. Since the condition estimates cost $O(n^2)$ flops, the resulting algorithm will take nearly the same number of flops as QR with column pivoting when at most a few interchanges are needed. As Algorithm 6 could take $O(n)$ interchanges and all condition estimation techniques can fail, Algorithm 6 could be very inefficient and can fail as well, although we believe that this is quite unlikely in practical applications.

Most of the floating-point operations in Algorithms 5 and 6 can be expressed as Level-2 BLAS. Using ideas similar to those in [3] and [6], it should be straightforward to develop block versions of these algorithms so that most of the floating-point operations are performed as Level-3 BLAS.

The restriction $m \ge n$ is not essential and can be removed with minor modifications to Algorithms 5 and 6. Thus these algorithms can also be used to compute a strong RRQR factorization for M^T, which may be preferable when one wants to compute an orthogonal basis for the right approximate null space.

Finally, the techniques developed in this paper can easily be adopted to compute *rank-revealing* LU factorizations [9, 13, 31, 32]. This result will be reported at a later date.

Acknowledgments. The authors thank Shivkumar Chandrasekaran and Ilse Ipsen for many helpful discussions and suggestions, and Gene Golub, Per Christian Hansen, W. Kahan, and Pete Stewart for suggestions that improved the presentation.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994.

[2] T. Beelen and P. Van Dooren, *An improved algorithm for the computation of Kronecker's canonical form of a singular pencil*, Linear Algebra Appl., 105 (1988), pp. 9–65.

[3] C. H. Bischof, *A block QR factorization algorithm using restricted pivoting*, in Proceedings, Supercomputing '89, ACM Press, New York, 1989, pp. 248–256.

[4] ———, *Incremental condition estimation*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 312–322.

[5] C. H. Bischof and P. C. Hansen, *Structure preserving and rank-revealing QR-factorizations*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1332–1350.

[6] ———, *A block algorithm for computing rank-revealing QR factorizations*, Numerical Algorithms, 2 (1992), pp. 371–392.

[7] C. H. BISCHOF AND G. M. SHROFF, *On updating signal subspaces*, IEEE Trans. Signal Processing, 40 (1992), pp. 96–105.
[8] P. A. BUSINGER AND G. H. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.
[9] T. F. CHAN, *On the existence and computation of LU-factorizations with small pivots*, Math. Comp., 42 (1984), pp. 535–547.
[10] ———, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.
[11] T. F. CHAN AND P. C. HANSEN, *Computing truncated singular value decomposition least squares solutions by rank revealing QR-factorizations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 519–530.
[12] ———, *Some applications of the rank revealing QR factorization*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 727–741.
[13] S. CHANDRASEKARAN AND I. IPSEN, *On rank-revealing QR factorisations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 592–622.
[14] ———, *Analysis of a QR algorithm for computing singular values*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 520–535.
[15] J. DEMMEL, M. HEATH, AND H. VAN DER VORST, *Parallel numerical linear algebra*, in Acta Numerica, A. Iserles, ed., Vol. 2, Cambridge University Press, Cambridge, 1993, pp. 111–197.
[16] D. K. FADDEEV, V. N. KUBLANOVSKAYA, AND V. N. FADDEEVA, *Solution of linear algebraic systems with rectangular matrices*, Proc. Steklov Inst. Math., 96 (1968), pp. 93–111.
[17] ———, *Sur les systemes lineaires algebriques de matrices rectangulaires et malconditionees*, in Programmation en Mathematiques Numeriques, Editions Centre Nat. Recherche Sci., Paris VII, 1968, pp. 161–170.
[18] L. V. FOSTER, *Rank and null space calculations using matrix decomposition without column interchanges*, Linear Algebra Appl., 74 (1986), pp. 47–71.
[19] P. E. GILL, G. H. GOLUB, W. MURRAY, AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.
[20] G. H. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.
[21] ———, *Matrix decompositions and statistical computation*, in Statistical Computation, R. C. Milton and J. A. Nelder, eds., Academic Press, New York, 1969, pp. 365–397.
[22] G. H. GOLUB, V. KLEMA, AND G. W. STEWART, *Rank Degeneracy and Least Squares Problems*, Tech. Report TR–456, Dept. of Computer Science, University of Maryland, College Park, MD, 1976.
[23] G. H. GOLUB AND V. PEREYRA, *The differentiation of pseudo-inverses, separable nonlinear least squares problems and other tales*, in Generalized Inverses and Applications, M. Z. Nashed, ed., Academic Press, New York, 1976, pp. 303–324.
[24] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.
[25] W. B. GRAGG AND G. W. STEWART, *A stable variant of the secant method for solving nonlinear equations*, SIAM J. Numer. Anal., 13 (1976), pp. 880–903.
[26] M. GU, *Finding Well-Conditioned Similarities to Block-Diagonalize Nonsymmetric Matrices is NP-Hard*, J. of Complexity, 11 (1995), pp. 377–391.
[27] W. W. HAGER, *Condition estimates*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 311–316.
[28] Y. P. HONG AND C.-T. PAN, *Rank-revealing QR factorizations and the singular value decomposition*, Math. Comp., 58 (1992), pp. 213–232.
[29] R. A. HORN AND C. R. JOHNSON, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, 1991.
[30] T.-M. HWANG, W.-W. LIN, AND D. PIERCE, *An Alternative Column Selection Criterion for a Rank-Revealing QR Factorization*, Tech. Report BCSTECH-93-021, Boeing Computer Services, Seattle, WA, July 1993. To appear in *Math. Comp.*.
[31] ———, *Improved Bound for Rank Revealing LU Factorizations*, Tech. Report BCSTECH-93-007, Boeing Computer Services, Seattle, WA, Oct. 1993.
[32] T.-M. HWANG, W.-W. LIN, AND E. K. YANG, *Rank revealing LU factorizations*, Linear Algebra Appl., 175 (1992), pp. 115–141.
[33] W. KAHAN, *Numerical linear algebra*, Canad. Math. Bull., 9 (1966), pp. 757–801.
[34] V. E. KANE, R. C. WARD, AND G. J. DAVIS, *Assessment of linear dependencies in multivariate data*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 1022–1032.
[35] V. N. KUBLANOVSKAYA, *AB-Algorithm and its modifications for the spectral problems of linear pencils of matrices*, Numer. Math., 43 (1984), pp. 329–342.
[36] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[37] C.-T. PAN AND P. T. P. TANG, *Bounds on Singular Values Revealed by QR Factorizations*, Preprint MCS-P332-1092, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, Oct. 1992.
[38] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.
[39] ———, *Rank degeneracy*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 403–413.

[40] G. W. STEWART, *Incremental Condition Calculation and Column Selection*, Tech. Report CS TR–2495, Dept. of Computer Science, University of Maryland, College Park, MD, July 1990.

[41] ———, *An updating algorithm for subspace tracking*, IEEE Trans. Signal Processing, 40 (1992), pp. 1535–1541.

[42] ———, *Updating a rank-revealing ULV decomposition*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 494–499.

[43] J. H. WILKINSON, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Mach., 8 (1961), pp. 281–330.

[44] S. WOLD, A. RUHE, H. WOLD, AND W. J. DUNN, III, *The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 735–743.