1990

ALGORITHM 686, FORTRAN Subroutines for Updating the QR Decomposition

Reichel, L.

ALGORITHM 686 FORTRAN Subroutines for Updating the QR Decomposition

L. REICHEL
Bergen Scientific Center
and
W. B. GRAGG
Naval Postgraduate School

Let the matrix $A \in \mathbf{R}^{m \times n}$, $m \geq n$, have a QR decomposition $A = \mathrm{QR}$, where $\mathrm{Q} \in \mathbf{R}^{m \times n}$ has orthonormal columns, and $\mathrm{R} \in \mathbf{R}^{n \times n}$ is upper triangular. Assume that Q and R are explicitly known. We present FORTRAN subroutines that update the QR decomposition in a numerically stable manner when A is modified by a matrix of rank one, or when a row or a column is inserted or deleted. These subroutines are modifications of the Algol procedures in Daniel et al. [5]. We also present a subroutine that permutes the columns of A and updates the QR decomposition so that the elements in the lower right corner of R will generally be small if the columns of A are nearly linearly dependent. This subroutine is an implementation of the rank-revealing QR decomposition scheme recently proposed by Chan [3].

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

General Terms: Algorithms

Additional Key Words and Phrases: QR decomposition, subset selection, updating

1. INTRODUCTION

We present several FORTRAN subroutines for updating the QR decomposition of a matrix. Let $A \in \mathbf{R}^{m \times n}$, $m \geq n$, have a QR decomposition $A = \mathrm{QR}$, where $\mathrm{Q} \in \mathbf{R}^{m \times n}$ has orthonormal columns, and $\mathrm{R} \in \mathbf{R}^{n \times n}$ is upper triangular. Assume that the elements of Q and R are explicitly known. Let $\bar{A} \in \mathbf{R}^{p \times q}$, $p \geq q$, be obtained from A by inserting or deleting a row or a column, or let $\bar{A}$ be a rank-one modification of A (i.e., $\bar{A} = A + vu^T$, where $u \in \mathbf{R}^n$, $v \in \mathbf{R}^m$). Then, a QR-decomposition of $\bar{A}$, $\bar{A} = \bar{\mathrm{Q}}\bar{\mathrm{R}}$, where $\bar{\mathrm{Q}} \in \mathbf{R}^{p \times q}$ has orthonormal columns and

This research was supported by NSF grant DMS-8704196 and by the Foundation Research Program of the Naval Postgraduate School.

Authors' addresses: L. Reichel, Bergen Scientific Center, Thormøhlensgaten 55, 36, N-5008, Bergen, Norway and Department of Mathematics, University of Kentucky, Lexington, KY 40506; W. B. Gragg, Department of Mathematics, Naval Postgraduate School, Monterey, CA 93943.

$\bar{R} \in \mathbf{R}^{q \times q}$ is upper triangular, can be computed in $O(mn)$ arithmetic operations by updating Q and R; see Daniel et al. [5]. The updating is done by applying Givens reflectors. The operation count for updating Q and R compares favorably with the $O(mn^2)$ arithmetic operations necessary to compute a QR decomposition of a general $m \times n$ matrix.

Algol procedures for computing $\bar{Q}$ and $\bar{R}$ from Q and R are presented by Daniel et al. [5]. Buckley [2] translated these procedures into FORTRAN. Our FORTRAN subroutines implement modifications of the algorithms in [5]. These modifications reduce the operation count. Also, the structure of our code differs from [2]. We use subroutines such as the Basic Linear Algebra Subprograms (BLAS), described in [12], for simple matrix and vector operations. This makes it fairly easy to adapt the frequently executed parts of the code in order to achieve faster execution. The subroutines run quickly on an IBM 3090 VF vector computer.

Several program libraries, such as LINPACK [6] and NAG [15], provide subroutines for updating R, but contain no routines for updating the complete QR decomposition. Two advantages of updating both Q and R are that QR decomposition can be modified stably when a row or a column is deleted from A, and that the individual elements of projections are easily accessible; see LINPACK [6, p. 10.23], Daniel et al. [5], and Stewart [18].

The first comprehensive survey of updating algorithms was presented by Gill et al. [8]; and a recent discussion with references to applications can be found in Golub and Van Loan [10, Ch. 12.6]. The applications include linear least squares problems, regression analysis, and the solution of nonlinear systems of equations. (See Allen [1], Goldfarb [9], Gragg and Stewart [11], Moré and Sorensen [14].) The algorithms also appear to be applicable to recursive least squares problems of signal processing; see Ling et al. [13].

We also present a subroutine that implements the rank-revealing QR decomposition method recently proposed by Chan [3]. In this method the QR decomposition A = QR is updated to yield the QR decomposition $\bar{A} = \bar{Q}\bar{R}$, where $\bar{A}$ is obtained from A by column permutation. This permutation is selected so that, in general, the element(s) in the lower right corner of $\bar{R}$ are small if A has nearly linearly dependent columns. The subroutine can be used to solve the subset selection problem; see Golub and Van Loan [10]. Table I lists the FORTRAN subroutines for updating the QR decomposition. All subroutines use double precision arithmetic and are written in FORTRAN 77. Section 2 contains programming details for the subroutines of Table I and for certain auxiliary subprograms. For all subroutines of Table I, except DRRPM, the numerical method as well as Algol procedures have been presented in [5]. For these subroutines we discuss differences between our FORTRAN subroutines and the Algol procedures only.

Section 3 discusses programming details for the subroutines for simple matrix and vector operations. These subroutines have been written to vectorize well on an IBM 3090 VF vector computer when the vectorizing compiler VS FORTRAN 2.3.0 is used. The last section, Section 4, contains some timing results for our subroutines and comparisons with the code [2].

Table I. Subroutines for Updating a QR Decomposition $A = \mathrm{QR}$ to Yield a QR Decomposition $\bar{A} = \bar{\mathrm{Q}}\bar{\mathrm{R}}$

Subroutine	Purpose
DDECL	Computes $\bar{\mathrm{Q}}$, $\bar{\mathrm{R}}$ from Q, R when $\bar{A}$ is obtained from A by deleting a column; see [5].
DDELR	Computes $\bar{\mathrm{Q}}$, $\bar{\mathrm{R}}$ from Q, R when $\bar{A}$ is obtained from A by deleting a row; see [5].
DINSC	Computes $\bar{\mathrm{Q}}$, $\bar{\mathrm{R}}$ from Q, R when $\bar{A}$ is obtained from A by inserting a column; see [5].
DINSR	Computes $\bar{\mathrm{Q}}$, $\bar{\mathrm{R}}$ from Q, R when $\bar{A}$ is obtained from A by inserting a row; see [5].
DRNK1	Computes $\bar{\mathrm{Q}}$, $\bar{\mathrm{R}}$ from Q, R when $\bar{A}$ is a rank-one modification of A; see [5].
DRRPM	Computes $\bar{\mathrm{Q}}$, $\bar{\mathrm{R}}$ from Q, R when $\bar{A}$ is obtained by permuting the columns of A in a manner that generally reveals if columns of A are nearly linearly dependent; see [3].

2. THE UPDATING SUBROUTINES

We consider the subroutines of Table I in order. These subroutines use auxiliary subroutines, which we need to introduce first. They are listed in Table II.

2.1 Subroutines DORTHO and DORTHX

Given a matrix $\mathrm{Q} \in \mathbf{R}^{m\times n}$, $m \geq n$, with orthonormal columns and a vector $w \in \mathbf{R}^m$, the subroutine DORTHO computes the Fourier coefficients $s := \mathrm{Q}^T w$ and the orthogonal projection of w into the null space of Q^T, $v := (I - \mathrm{QQ}^T)w$. At most one reorthogonalization is carried out. Since the subroutine DORTHO differs from the corresponding Algol procedure "orthogonalize" in [5], we discuss DORTHO and its use in some detail.

Subroutine DORTHO is called by routine DINSC, which updates the QR factorization of a matrix $A = \mathrm{QR} \in \mathbf{R}^{m\times n}$, $m > n$, when a column w is inserted into A. Updating may not be meaningful if w is a nearly linear combination of the columns of Q. Therefore, DORTHO computes the condition number of the matrix $\tilde{\mathrm{Q}} := [\mathrm{Q}, w/\|w\|] \in \mathbf{R}^{m\times(n+1)}$, where $\|\ \|$ is the Euclidean norm. Using $\mathrm{Q}^T\mathrm{Q} = I$, we obtain the following expressions for the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n+1}$ of $\tilde{\mathrm{Q}}$:

$$\sigma_1 = (1 + \|\mathrm{Q}^T w\|/\|w\|)^{1/2}, \tag{2.1a}$$

$$\sigma_j = 1, \qquad 2 \leq j \leq n, \tag{2.1b}$$

$$\sigma_{n+1} = (1 - \|\mathrm{Q}^T w\|/\|w\|)^{1/2}. \tag{2.1c}$$

Further, for $v := (I - \mathrm{QQ}^t)w/\|w\|$,

$$\|v\| = \sigma_1\sigma_{n+1}. \tag{2.2}$$

Since $1 \leq \sigma_1 \leq \sqrt{2}$, σ_{n+1} is also an accurate estimate of the length of the orthogonal projection of $w/\|w\|$ into the null space of Q^T. In order to avoid severe cancellation of significant digits in (2.1c), we first determine σ_1 from (2.1a) and then σ_{n+1} from (2.2).

Subroutines DINSC and DORTHO have an input parameter RCOND which is a lower bound for the reciprocal condition number. The computations are discontinued and an error flag is set if RCOND $> \sigma_{n+1}/\sigma_1$. On exit, RCOND $:= \sigma_{n+1}/\sigma_1$.

Table II. Auxiliary Subroutines

Auxiliary subroutine	Called by subroutine	Purpose
DORTHO	DINSC, DRNK1	Compute $s := Q^T w$, $v := (I - QQ^T)w$ with reorthogonalization for arbitrary vector w.
DORTHX	DDELR	Compute $s := Q^T e_j$, $v := (I - QQ^T)e_j$, with reorthogonalization for axis vector e_j.
DINVIT	DRRPM	Compute approximation of a right singular vector corresponding to a least singular value of R. A first approximation is obtained from the LINPACK condition number estimator DTRCO, and is improved by inverse iteration.
DTRLSL	DINVIT	Solve lower triangular system of equations with frequent rescalings in order to avoid overflow. Similar to part of DTRCO.
DTRUSL	DINVIT	Solve upper triangular linear system of equations with frequent rescalings in order to avoid overflow. Similar to part of DTRCO.

Assume now that the input value of RCOND $\leq \sigma_{n+1}/\sigma_1$. Then DORTHO computes $s := Q^T w$ and $v := (I - QQ^T)w$ by a scheme analogous to the method described by Parlett [15, p. 107] for orthogonalizing a vector against another vector. For definiteness, we present the orthogonalization scheme. References to σ_1, σ_{n+1}, and RCOND are neglected for simplicity.

Orthogonalization Algorithm. Input $Q \in \mathbf{R}^{m \times n}$ (Q has orthonormal columns), m, n ($m > n$), $w \in \mathbf{R}^m$ ($w \neq 0$); output v ($v = (I - QQ^T)w$), s ($s = Q^T w$);

```
$\tilde{w} := w/\|w\|$;
$s := Q^T\tilde{w}$;   $v := \tilde{w} - Qs$;                                   (2.3)
if $\|v\| \geq 0.707$ then
    $v := v/\|v\|$;   $s := s\|w\|$;   exit;   * $\|v\| = 1$,   $Q^T v = 0$ *
$s' := Q^T v$;   $v' := v - Qs'$;                                            (2.4)
if $\|v'\| \leq 0.707\|v\|$ then
    * $w$ lies in span{Q} numerically *
    $v := 0$;   $s := (s + s')\|w\|$;   set flag;   exit;
$v := (v + v')/\|v + v'\|$;   $s := (s + s')\|w\|$;   exit;   * $\|v\| = 1$,   $Q^T v = 0$ *
```

The proof in Parlett [16, pp. 107–108], that one reorthogonalization suffices, carries over the present algorithm, using $Q^T Q = I$.

We note that there are other ways to carry out the computations on lines (2.3)–(2.4). In [5], v and v' are updated immediately after a component of s is computed. Our scheme has the advantage of being faster on vector computers because it allows matrix vector operations; it is also, generally, more accurate, since there is less accumulation of rounding errors. The latter advantage can be shown easily, but we omit the details.

We turn to subroutine DORTHX. This is a faster version of subroutine DORTHO. DORTHX assumes that w in the orthogonalization algorithm is an axis vector. This simplifies the computations in (2.3). DORTHX may perform nearly twice as fast as DORTHO.

2.2 Subroutines DINVIT, DTRLSL, and DTRUSL

Given a nonsingular upper triangular matrix $U = [\mu_{jk}] \in \mathbf{R}^{n \times n}$ and a vector $b = [\beta_j] \in \mathbf{R}^n$, DTRUSL solves $Ux = b\rho$, where $|\rho| \le 1$ is a scaling factor such that $|\beta_j \rho / \mu_{jj}| \le 1$ for all j. The scaling factor is introduced in order to avoid overflow when solving very ill-conditioned linear systems of equations. DTRLSL is an analogous subroutine for lower triangular systems.

DTRLSL and DTRUSL are called by DINVIT, a subroutine for computing an approximation of a right singular vector belonging to a least singular value of a right triangular matrix R. If R is singular, then such a singular vector is computed by solving a triangular linear system of equations. Otherwise an initial approximate right singular vector $a^{(o)} = \{\alpha_j^{(o)}\}_{j=1}^n$ is obtained by the LINPACK condition number estimator DTRCO, and inverse iteration with $R^T R$ is used to obtain improved approximations $a^{(j)}, j = 1, 2, \ldots,$ NMBIT, where NMBIT is an input parameter to DINVIT and DRRPM. On exit from DINVIT and DRRPM, IPOS(j) contains the least index k such that $|\alpha_k^{(j)}| \ge |\alpha_l^{(j)}|$, $1 \le l \le n$, $0 \le j \le$ NMBIT. On return from DINVIT and DRRPM, the parameter DELTA is given by DELTA $:= \| R^T R a^{(\mathrm{NMBIT})} \| / \| a^{(\mathrm{NMBIT})} \|$. Hence, DELTA is an upper bound for the least singular value of R.

2.3 Updating Subroutines

We are now in a position to consider the subroutines of Table I. The vectorization is done mainly in the subroutines for simple matrix and vector operations, discussed in Section 3, but some loops of the subroutines of Table I vectorize as well. Comments in the source code reveal which loops vectorize or are eligible for vectorization on an IBM 3090 VF computer with compiler VS FORTRAN 2.3.0 when the default vectorization directives are used. For applications to particular problem classes, changing the default vectorization by compiler directives may decrease execution time.

We list the differences between the subroutines of Table I and the corresponding Algol procedures of [5]. Some of these modifications are suggested in [5], but not implemented in the Algol procedures. In subroutine DDELC, the column deleted in $A :=$ QR is determined optionally. Not computing this column saves $O(mn)$ arithmetic operations. In subroutine DDELR, the auxiliary subroutine DORTHX is used instead of DORTHO. As indicated in Section 2.1, the former subroutine may perform nearly twice as fast. In subroutine DINSC, a column w is inserted into $A :=$ QR only if the reciprocal condition number of the matrix $[Q, w/\|w\|]$ is larger than a bound given by the parameter RCOND on entry. The parameter RCOND can be used to prevent updating when $w/\|w\|$ is nearly in the range of Q. Finally, DRNK1 performs slightly faster if the updated matrix $A + vu^T$ is such that v lies numerically in the range of A.

The subroutine DRRPM implements an algorithm presented by Chan [3]. The computation of an approximate right singular vector corresponding to a least singular value is done by subroutine DINVIT, and has already been discussed. The position of the component of largest magnitude of this singular vector has to be determined, and we found, in agreement with Chan's suggestion [3], that two inverse iterations suffice. In fact, in all computed examples, one inverse iteration was sufficient, even for problems with multiple or close least singular values. The subroutine permutes the order of columns 1 through k of $A\Pi$ where k is an input parameter, $A \in \mathbf{R}^{m\times n}$, $m \geq n$, and Π is a permutation matrix. Typically, DRRPM is called with $k = n$, $n - 1$, $n - 2$, ... until no further permutation is made or until the computer upper bound DELTA for the least singular value of the matrix consisting of the first k columns of $A\Pi$ is not small.

The subroutines of Table I neither require nor produce a factorization with nonnegative diagonal elements of the upper triangular matrix.

3. SUBROUTINES FOR SIMPLE MATRIX AND VECTOR OPERATIONS

Much computational experience on a variety of computers led Dongarra and Sorensen [7] to conclude that nearly optimal performance of numerical linear algebra code can be achieved if the subroutines for simple matrix and vector operations, such as BLAS [12], are written to perform on the computer at hand. We have written these subroutines so that they run efficiently on an IBM 3090 VF vector computer when the compiler VS FORTRAN 2.3.0 with parameter vlev = 2 is used. No vectorization directives are required in the code. In particular, we note that when implementing the code for updating the QR decomposition of a matrix on other (vector) computers, it may be possible to gain execution speed by modifying the subroutines for simple linear algebra tasks. We illustrate this with an example that describes a partricular feature of the VS FORTRAN 2.3.0 compiler. By introducing a temporary scalar variable, denoted ACC in the subroutine DAPX in Example 3.1 below, this compiler generates code that avoids unnecessary loads and stores.

During execution, ACC should be thought of as a vector variable stored in a vector register. Timings for DAPX and comparison with code with explicitly unrolled loops have been carried out by Robert and Squazzero [17]. These timings show subroutine DAPX to perform better than equivalent subroutines with explicitly unrolled loops. However, when implemented on other computers with other compilers, the subroutine DAPX might perform better if the order of the DO-loops is interchanged.

Example 3.1 *Subroutine for Matrix Vector Multiplication*

```
      SUBROUTINE DAPX(A,LDA,M,N,X,Y)
C
C     DAPX COMPUTES Y := A*X.
C
      INTEGER LDA,M,N,I,J
      REAL*8 A(LDA,N), X(N), Y(M), ACC
C
C     OUTER LOOP VECTORIZES.
C
      DO 10 I=1, M
        ACC=ODO
        DO 20 J=1, N
          ACC=ACC+A(I,J)*X(J)
20        CONTINUE
      Y(I)=ACC
10    CONTINUE
      RETURN
      END
```

Table III. Timings for DDELCO

		CPU time in seconds		
m	n	Scalar arithmetic	Vector arithmetic	$\frac{\text{Scalar time}}{\text{Vector time}}$
10	10	$4 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	1
20	10	$4 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	1
30	10	$5 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	1.5
50	10	$6 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	1.5
75	10	$8 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	2
128	10	$1 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	2
1024	10	$7 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	3.5
1280	10	$9 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	3.5

Table IV. Timings for DRNK1

		CPU time in seconds		
m	n	Scalar arithmetic	Vector arithmetic	$\frac{\text{Scalar time}}{\text{Vector time}}$
16	12	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	1
32	25	$4 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	1.5
64	50	$2 \cdot 10^{-2}$	$7 \cdot 10^{-3}$	2
128	100	$6 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	2.5
1024	100	$4 \cdot 10^{-1}$	$8 \cdot 10^{-2}$	4.5
1250	100	$5 \cdot 10^{-1}$	$9 \cdot 10^{-1}$	5

4. COMPUTED EXAMPLES

Example 4.1 Execution times for subroutines DDELCO and DRNK1 are compared for scalar and vector arithmetic. The measured CPU times differ somewhat between different executions of the same code. The times reported are therefore rounded to one significant digit and the quotient of measured CPU times rounded to the nearest multiple of $\frac{1}{2}$.

Table III shows the CPU times for DDELCO. This routine and its subroutines are compiled with the VS FORTRAN 2.3.0 compiler. The times for vector arithmetic are obtained from code generated with compiler option vlev = 2, which makes the compiler generate code that utilizes the vector registers and arithmetic. The times for scalar arithmetic are obtained from code generated with compiler option vlev = 0, which makes the compiler generate code that does not use vector instructions. Given a QR decomposition of a matrix $A \in \mathbf{R}^{m \times n}$, Table III shows the CPU time required by DDELCO to compute the QR decomposition of $\bar{A} \in \mathbf{R}^{m \times (n-1)}$ obtained by deleting column one of A.

Table IV is similar to Table III, and contains execution times for DRNK1. The reduction in execution time obtained by using vector instructions is of the same order of magnitude for the other updating routines, too.

Example 4.2 Execution times for subroutines written by Buckley [2] and those of Table I are compared. The vectorized and scalar codes were generated as

Table V. The First Row of $A = QR$ is Deleted. CPU Times for Vectorized Code for Updating Q and R are Given in Seconds; $n = 10$.

m	Time for DDELR	Time for DELROW [2]	Time for DELROW [2] / Time for DDELR
10	$3 \cdot 10^{-5}$	$9 \cdot 10^{-4}$	$3.5 \cdot 10^{1}$
64	$7 \cdot 10^{-5}$	$2 \cdot 10^{-3}$	$2 \cdot 10^{1}$
128	$8 \cdot 10^{-4}$	$3 \cdot 10^{-3}$	3
1024	$4 \cdot 10^{-3}$	$2 \cdot 10^{-2}$	4

Table VI. The Last Column of $A = QR$ is deleted. CPU Times for Vectorized Code for Updating Q and R are Given in Seconds. DDELC Does Not Compute the Last Column of A. (IFLAG = 0 on Entry).

m	n	Time for DELC	Time for DELCOL [2]	Time for DELCOL [2] / Time for DDELC
1024	10	$1 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	2
1024	100	$1 \cdot 10^{-4}$	$7 \cdot 10^{-3}$	$7.5 \cdot 10^{1}$
1280	100	$1 \cdot 10^{-4}$	$9 \cdot 10^{-3}$	$9.5 \cdot 10^{1}$

Table VII. A New First Column is Inserted into $A = QR$. CPU Times for Vectorized Code for Updating Q and R are Given in Seconds; $n = 10$.

m	Time for DINSC	Time for INSCOL [2]	Time for INSCOL [2] / Time for DINSC
64	$1 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	2
128	$1 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	2
1024	$7 \cdot 10^{-3}$	$2 \cdot 10^{-2}$	2.5

Table VIII. The First Row of $A = QR$ is Deleted. CPU Times for Scalar Code for Updating Q and R are Given in Seconds; $n = 10$.

m	Time for DDELR	Time for DELROW [2]	Time for DELROW [2] / Time for DDELR
10	$2 \cdot 10^{-5}$	$6 \cdot 10^{-4}$	$3 \cdot 10^{1}$
64	$1 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	1.5
128	$2 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	1.5
1024	$1 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	1.5

explained in Example 4.1. We found that vectorization of the subroutines in [2] did not change the execution times significantly, generally less than 20%. In all computed examples the vectorized subroutines in [2] required at least twice as much execution time as the vectorized subroutines of Table I. For certain problems, our vectorized code executed up to 95 times faster than the vectorized code in [2]. For scalar code the differences in execution time often decreased with increasing matrix size. Tables V–VIII present some sample timings.

Tables V–VII present timings for vectorized code. The next table shows timings for scalar code for the same updatings as in Table V. Table VIII shows that, without vectorization, DELROW [2] requires 50% more CPU time then DDELR for moderately large problems.

ACKNOWLEDGMENTS

One of the authors (L.R.) would like to thank Jack Dongarra and Pat Gaffney for valuable discussions, and Aladin Kamel for help in handling the computer.

REFERENCES

1. ALLEN, D. M. Mean square error of prediction as a criterion for selecting variables. *Technometrics 13* (1971), 469–475.
2. BUCKLEY, A. Algorithm 580, QRUP: A set of FORTRAN routines for updating QR factorizations. *ACM Trans. Math. Softw. 7* (1981), 548–549 and *8* (1982), 405.
3. CHAN, T. F. Rank revealing QR factorizations. *Lin. Alg. Appl. 88/89* (1987), 67–82.
4. CLINE, A. K., MOLER, C. B., STEWART, G. W., AND WILKINSON, J. H. An estimate for the condition number of a matrix. *SIAM J. Numer. Anal. 16* (1979), 368–375.
5. DANIEL, J. W., GRAGG, W. B., KAUFMAN, L., AND STEWART, G. W. Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. *Math. Comput. 30* (1976), 772–795.
6. DONGARRA, J. J., BUNCH, J. R., MOLER, C. B., AND STEWART, G. W. *Linpack Users' Guide*. SIAM, Philadelphia, 1979.
7. DONGARRA, J. J., AND SORENSEN, D. C. Linear algebra on high performance computers. In *Applications of Supercomputers*, D. F. Lockhart, and D. L. Hicks, Eds., Elsevier, Amsterdam, 1986, 57–88.
8. GILL, P. E., GOLUB, G. H., MURRAY, W., AND SAUNDERS, M. A. Methods for modifying martrix factorizations. *Math. Comput. 28* (1974), 505–535.
9. GOLDFARB, D. Factorized variable metric methods for unconstrained optimization. *Math. Comput. 30* (1976), 796–811.
10. GOLUB, G. H., AND VAN LOAN, C. F. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Md., 1983.
11. GRAGG, W. B., AND STEWART, G. W. A stable variant of the secant method for solving nonlinear equations. *SIAM J. Numer. Anal. 13* (1976), 889–903.
12. LAWSON, C., HANSON, R., KINCAID, D., AND KROGH, F. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Softw. 5* (1979), 308–323.
13. LING, F., MANOLAKIS, D., AND PROAKIS, J. G. A recursive modified Gram–Schmidt algorithm for least-squares estimation. *IEEE Trans. Acoustics, Speech and Signal Process. ASSP-34* (1986), 829–835.
14. MORÉ, J. J., AND SORENSEN, D. C. Newton methods. In *Studies in Numerical Analysis*, G. H. Golub, Ed., The Mathematical Association of America, Washington, D.C., 1984, 29–82.
15. *NAG FORTRAN Library Manual*, Mark 12. Numerical Algorithms Group, 1987.
16. PARLETT, B. N. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, N.J., 1980.
17. ROBERT, Y., AND SGUAZZERO, P. The LU decomposition algorithm and its efficient FORTRAN implementation on the IBM 3090 vector multiprocessor. Tech. Rep. ICE-0006, IBM European Center for Scientific and Engineering Computing, Rome, 1987.
18. STEWART, G. W. The effect of rounding error on an algorithm for downdating a Cholesky factorization. *J. Inst. Math. Applic. 23* (1979), 203–213.

Received October 1988; revised August 1989; accepted September 1989