FORTRAN SUBROUTINES FOR UPDATING
THE QR DECOMPOSITION

William Gragg
Lother Reichel

November 1988

Approved for public release; distribution unlimited
Prepared for: Naval Postgraduate School and the
National Science Foundation, Washington
D.C. 20550
FORTRAN Subroutines For Updating The QR Decomposition

Abstract: Let the matrix \( A \in \mathbb{R}^{m \times n} \), \( m \geq n \), have a QR decomposition \( A = QR \), where \( Q \in \mathbb{R}^{m \times n} \) has orthonormal columns, and \( R \in \mathbb{R}^{n \times n} \) 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]. The subroutines have been written to perform well on a vector computer.
FORTRAN Subroutines for Updating the QR Decomposition*

L. Reichel  
Bergen Scientific Centre  
and  
University of Kentucky

W.B. Gragg  
Naval Postgraduate School  
and  
University of Kentucky

Abstract: Let the matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$, have a QR decomposition $A = QR$, where $Q \in \mathbb{R}^{m \times n}$ has orthonormal columns, and $R \in \mathbb{R}^{n \times n}$ 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]. The subroutines have been written to perform well on a vector computer.

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, updating, subset selection.

Authors' addresses: L. Reichel, Bergen Scientific Centre, Allegaten 36, N-5007 Bergen, Norway; and University of Kentucky, Department of Mathematics, Lexington, KY 40506, USA; W.B. Gragg, Naval Postgraduate School, Department of Mathematics, Monterey, CA 93943, USA.

*Research supported by the National Science Foundation under Grant DMS-870416 and by the Naval Postgraduate School Research Council
1. **INTRODUCTION**

The purpose of this paper is to present several FORTRAN subroutines for updating the QR decomposition of a matrix. Let $A \in \mathbb{R}^{m \times n}$, $m \geq n$, have a QR decomposition $A = QR$, where $Q \in \mathbb{R}^{m \times n}$ has orthonormal columns, and $R \in \mathbb{R}^{n \times n}$ is upper triangular. Assume that the elements of $Q$ and $R$ are explicitly known. Let $\tilde{A} \in \mathbb{R}^{p \times q}$, $p \geq q$, be obtained from $A$ by inserting or deleting a row or a column, or let $\tilde{A}$ be a rank-one modification of $A$, i.e., $\tilde{A} = A + vu^T$, where $u \in \mathbb{R}^n$, $v \in \mathbb{R}^m$. Then a QR-decomposition of $\tilde{A}$, $\tilde{A} = \tilde{Q}\tilde{R}$, where $\tilde{Q} \in \mathbb{R}^{p \times q}$ has orthonormal columns and $\tilde{R} \in \mathbb{R}^{q \times q}$ 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 $\tilde{Q}$ and $\tilde{R}$ 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 Algol procedures in [5]. These modifications speed up the subroutines and make them suitable for use on vector computers. This is illustrated by timing experiments.

Several program libraries, such as LINPACK [6] and NAG [14], provide subroutines for updating $R$ only, but contain no routines for updating the complete QR decomposition. Advantages of updating both $Q$ and $R$ include that downdating can be carried out stably, and that the individual elements of projections are easily accessible; see LINPACK [6, p. 10.23], Daniel et al. [5], and Stewart [17].
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, Chapter 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], More and Sorensen [13]. The algorithms would also appear to be applicable to recursive least squares problems of signal processing; see Ling et al. [12].

We also present a subroutine which 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 1.1 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 1.1 and for certain auxiliary subprograms. For all subroutines of Table 1.1, except DRRPM, the numerical method as well as Algol procedures have been presented in [5]. For these subroutines—we will only discuss differences between our FORTRAN subroutines and the Algol procedures. These differences stem in part from the algorithms being sped up, as well as from the use of simple subroutines, BLAS, for elementary vector and matrix operations.
Subroutine Purpose

DDELC Computes $\bar{Q}, \bar{R}$ from $Q,R$ when $\bar{A}$ is obtained from $A$ by deleting a column; see [5].

DDELR Computes $\bar{Q}, \bar{R}$ from $Q,R$ when $\bar{A}$ is obtained from $A$ by deleting a row; see [5].

DINSC Computes $\bar{Q}, \bar{R}$ from $Q,R$ when $\bar{A}$ is obtained from $A$ by inserting a column; see [5].

DINSR Computes $\bar{Q}, \bar{R}$ from $Q,R$ when $\bar{A}$ is obtained from $A$ by inserting a row; see [5].

DRNK1 Computes $\bar{Q}, \bar{R}$ from $Q,R$ when $\bar{A}$ is a rank-one modification of $A$; see [5].

DRRPM Computes $\bar{Q}, \bar{R}$ 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].

Table 1.1: Subroutines for updating a QR decomposition $A = QR$ to yield a QR decomposition $\bar{A} = \bar{Q}\bar{R}$.

The BLAS are discussed in Section 3. They have been written to vectorize efficiently on a IBM 3090-200VF computer using the vectorizing compiler VS FORTRAN 2.3.0 without special compiler directives. Most BLAS were obtained by modifying LINPACK BLAS [6]. We hope that the provided BLAS vectorize well without excessive timing increases also on other vector computers. Section 4 contains output from a driver illustrating the use of the subroutines. A listing of the source code of the driver is provided in the Appendix. Section 4 also contains some timing results.
2. THE UPDATING SUBROUTINES

We consider the subroutines of Table 1.1 in order. These subroutines use auxiliary subroutines which we need to introduce first. They are listed in Table 2.1.

<table>
<thead>
<tr>
<th>Auxiliary subroutine</th>
<th>Called by subroutine</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DORTHO</td>
<td>DINSC, DRNK1</td>
<td>Compute ( s = Q^T w ), ( v = (I - QQ^T)w ) with reorthogonalization for arbitrary vector ( w ).</td>
</tr>
<tr>
<td>DORTHX</td>
<td>DDELR</td>
<td>Compute ( s = Q^T e_j ), ( v = (I - QQ^T)e_j ), with reorthogonalization for axis vector ( e_j ).</td>
</tr>
<tr>
<td>DINVIT</td>
<td>DRRPM</td>
<td>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.</td>
</tr>
<tr>
<td>DTRLSSL</td>
<td>DINVIT</td>
<td>Solve lower triangular system of equations with frequent rescalings in order to avoid overflow. Similar to part of DTRCO.</td>
</tr>
<tr>
<td>DTRUSL</td>
<td>DINVIT</td>
<td>Solve upper triangular linear system of equations with frequent rescalings in order to avoid overflow. Similar to part of DTRCU.</td>
</tr>
</tbody>
</table>

Table 2.1: Auxiliary subroutines.

2.1 Subroutines DORTHO and DORTHX

Given a matrix \( Q \in \mathbb{R}^{m \times n} \), \( m \geq n \), with orthonormal columns and a vector \( w \in \mathbb{R}^m \), the subroutine DORTHO computes the Fourier coefficients \( s = Q^T w \) and the orthogonal projection of \( w \) into the null-space of \( Q^T \), \( v = (I - QQ^T)w \). 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 \( A = QR \in \mathbb{R}^{mxn}, m > n, \) when a column \( w \) is inserted into \( A. \) Updating may not be meaningful if \( w \) is nearly a linear combination of the columns of \( Q. \) Therefore DORTHO computes the condition number of the matrix \( \tilde{Q} = [Q,w/\|w\|] \in \mathbb{R}^{mx(n+1)}, \) where the norm \( \| \| \) is the Euclidean norm. Using \( Q^TQ = I, \) we obtain the following expressions for the singular values \( \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{n+1} \) of \( \tilde{Q}: \)

\[
\sigma_1 = (1 + \|Q^Tw\|/\|w\|)^{1/2}, \tag{2.1a}
\]

\[
\sigma_j = 1, \quad 2 \leq j \leq n, \tag{2.1b}
\]

\[
\sigma_{n+1} = (1 - \|Q^Tw\|/\|w\|)^{1/2}. \tag{2.1c}
\]

Further, for \( v = (I-QQ^T)w/\|w\|, \)

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

Since \( 1 \leq \sigma_1 \leq \sqrt{2}, \) \( \sigma_{n+1} \) 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 determine first \( \sigma_1 \) from (2.1a) and then \( \sigma_{n+1} \) 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, \( RCOND = \sigma_{n+1}/\sigma_1. \)
Assume now that the input value of $\text{RCOND} \geq \sigma_{n+1}/\sigma_1$. 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 $\sigma_1$, $\sigma_{n+1}$, and $\text{RCOND}$ are neglected for simplicity.

Orthogonalization algorithm: input $Q \in \mathbb{R}^{m \times n}$ ($Q$ has orthonormal columns), $m, n$ ($m > n$), $w \in \mathbb{R}^m$ ($w \neq 0$); output $v$ ($v = (I - QQ^T) w$), $s$ ($s = Q^T w$);

\begin{align*}
\bar{w} & : = w/\|w\|; \\
s & : = Q^T \bar{w}; v & : = \bar{w} - Qs; \\
& \text{if } \|v\| \geq 0.707 \text{ then} \\
& \quad v: = v/\|v\|; s: = s\|w\|; \text{ exit; } \ast \|v\| = 1, Q^Tv = 0 \ast \\
& s': = Q^Tv; v': = v - Qs'; \\
& \text{if } \|v'\| \leq 0.707\|v\| \text{ then} \\
& \quad \ast w \text{ lies in span}\{Q\} \text{ numerically} \ast \\
& \quad v: = 0; s: = (s + s')\|w\|; \text{ set flag; exit; } \\
& \quad v: = (v + v')/\|v + v'\|; s: = (s + s')\|w\|; \text{ exit; } \ast \|v\| = 1, Q^Tv = 0 \ast
\end{align*}

The proof in Parlett [15, pp. 107-108] that one reorthogonalization suffices carries over to the present algorithm, using that $Q^TQ = I$.

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 advantages of being faster on vector computers, since it allows matrix vector operations, and it is also, generally, more accurate, since rounding errors
accumulate less. The latter can easily be shown, and 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 $U = [\mu_{jk}] \in \mathbb{R}^{n \times n}$ and a vector $b = [\beta_j] \in \mathbb{R}^n$, DTRUSL solves $Ux = bp$, where $|\rho| \leq 1$ is a scaling factor such that $|\beta_j \rho / \mu_{jj}| \leq 1$ 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 $a^{(o)} = \{a^{(o)}_j\}_{j=1}^n$ is obtained by the LINPACK condition number estimator DTRCO, and inverse iteration with $R^T R$ is used to obtain improved approximations $a^{(j)}$, $j = 1, 2, \ldots, \text{NMBIT}$, where NMBIT is an input parameter to DINVIT and DRRPM. On exit from DINVIT and DRRPM, $\text{IPOS}(j)$ contains the least index $k$ such that $|a^{(j)}_k| \geq |a^{(j)}_\ell|$, $1 \leq \ell \leq n$, $0 \leq j \leq \text{NMBIT}$. On return from DINVIT and DRRPM the parameter DELTA is given by

$$\text{DELTA} = \left\| R^T a^{(\text{NMBIT})} \right\| / \left\| a^{(\text{NMBIT})} \right\|.$$

Hence, DELTA is an upper bound for the least singular value of $R$. 8
2.3 Updating subroutines

We are in a position to consider the subroutines of Table 1.1. The vectorization is mainly done in the BLAS of the next section, but some loops of the subroutines of Table 1.1 vectorize as well. Comments in the source code reveal which loops vectorize or are eligible for vectorization on an IBM 3090-200VF computer with compiler VS FORTRAN 2.3.0 to where the default vectorization directives are used. For applications to particular problem classes, changing the default vectorization by compiler directives may decrease the execution time.

We list the differences between the subroutines of Table 1.1 and the corresponding Algol procedures of [5]. Some of these modifications were suggested in [5] but not implemented in the Algol procedures [5]. 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 a 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 AΠ where k is an input parameter, $A \in \mathbb{R}^{m \times n}$, $m \geq n$, and Π is a permutation matrix. DRRPM is typically called with $k = n, n-1, n-2, \ldots$ until no further permutation is made or until the computed upper bound DELTA for the least singular value of the matrix consisting of the first k columns of AΠ is not small.

The subroutines of Table 1.1 do neither require nor produce a factorization with nonnegative diagonal elements of the upper triangular matrix.

3. THE BLAS

Much computational experience on a variety of computers led Dongarra and Sorensen [7] to conclude that nearly optimal performance of numerical linear algebra subroutines can be achieved if the subroutines for the basic matrix and vector operations, such as multiplication, addition and inner product computation, are written to perform well on vector computers. We wanted to write a code that performs well on an IBM 3090-200VF computer, and that would not require excessive tuning when moved to other (vector) computers. Therefore we designed the code to vectorize well without special compiler instructions, since the latter would be machine dependent.
A feature of the VS FORTRAN 2.3.0 compiler is that unnecessary vector loads and stores are avoided by introducing a temporary scalar variable, denoted by ACC in the subroutine DAPX in Example 3.1. 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 [16]. These timings show subroutine DAPX to perform better than equivalent subroutines with explicitly unrolled loops.

Example 3.1. Subroutine for matrix vector multiplication.

```fortran
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(N),ACC
C
C OUTER LOOP VECTORIZES.
C
DO 10 I=1,M
   ACC=0D0
   DO 20 J=1,N
      ACC=ACC+A(I,J)*X(J)
   20 CONTINUE
   Y(I)=ACC
10 CONTINUE
RETURN
END
```

Temporary scalar variables have also been used in others of the 17 BLAS used.

4. COMPUTED EXAMPLES

Example 4.1. In this example the QR decomposition of a 4 x 3 matrix A is updated. The use of all subroutines of Table 1.1 is illustrated. The main program producing this output is listed in the Appendix.
Example 4.1

\[
A = Q^* R, \quad \text{MATRIX} A:
\begin{align*}
0.500 & \quad 0.000 & -0.500 \\
0.500 & \quad 0.000 & -1.500 \\
0.500 & \quad -1.000 & 0.500 \\
0.500 & \quad -1.000 & -0.500
\end{align*}
\]

\[
\text{MATRIX} Q:
\begin{align*}
0.500 & \quad 0.500 & 0.500 \\
0.500 & \quad 0.500 & -0.500 \\
0.500 & \quad -0.500 & 0.500 \\
0.500 & \quad -0.500 & -0.500
\end{align*}
\]

\[
\text{MATRIX} R:
\begin{align*}
1.000 & \quad -1.000 & -1.000 \\
1.000 & \quad 1.000 & -1.000 \\
0.000 & \quad 0.000 & 1.000
\end{align*}
\]

DELETE COLUMN 2 OF A AND UPDATE Q AND R BY DDELC
ON RETURN FROM DDELC INFO=0
COLUMN RECOMPUTED BY DDELC: 0.000 0.000 -1.000 -1.000

\[
\text{UPDATED MATRIX} Q:
\begin{align*}
0.500 & \quad 0.000 \\
0.500 & \quad 0.707 \\
0.500 & \quad -0.707 \\
0.500 & \quad 0.000
\end{align*}
\]

\[
\text{UPDATED MATRIX} R:
\begin{align*}
1.000 & \quad -1.000 \\
0.000 & \quad -1.414
\end{align*}
\]

ABS( ELEMENT OF LARGE MAGNITUDE OF A-Q*R ): 0.2E-15
ABS( ELEMENT OF LARGE MAGNITUDE OF Q'*Q-I ): 0.2E-15

DELETE ROW 3 OF A AND UPDATE Q AND R BY DDELR
ON RETURN FROM DDELR INFO=0
ROW RECOMPUTED BY DDELR: 0.500 0.500

\[
\text{UPDATED MATRIX} Q:
\begin{align*}
-0.577 & \quad 0.408 \\
-0.577 & \quad -0.816 \\
-0.577 & \quad 0.408
\end{align*}
\]

\[
\text{UPDATED MATRIX} R:
\begin{align*}
-0.866 & \quad 1.443 \\
0.000 & \quad 0.816
\end{align*}
\]

ABS( ELEMENT OF LARGE MAGNITUDE OF A-Q*R ): 0.2E-15
ABS( ELEMENT OF LARGE MAGNITUDE OF Q'*Q-I ): 0.3E-16

NEW 3RD ROW TO BE INSERTED INTO A BY DINSR: 1.000 2.000 3.000
ON RETURN FROM DINSR INFO=0
COMPUTED RECIPROCAL CONDITION NUMBER BY DINSR: 0.2E+00

\[
\text{UPDATED MATRIX} Q:
\begin{align*}
-0.267 & \quad 0.873 & \quad 0.408 \\
-0.535 & \quad 0.218 & -0.816 \\
-0.802 & \quad -0.436 & 0.408
\end{align*}
\]

\[
\text{UPDATED MATRIX} R:
\begin{align*}
-3.742 & \quad -0.802 & 1.336 \\
0.000 & \quad 0.327 & -0.546 \\
0.000 & \quad 0.000 & 0.816
\end{align*}
\]

ABS( ELEMENT OF LARGE MAGNITUDE OF A-Q*R ): 0.4E-15
ABS( ELEMENT OF LARGE MAGNITUDE OF Q'*Q-I ): 0.2E-15

NEW 3RD ROW TO BE INSERTED INTO A BY DINSR: 1.000 2.000 3.000
ON RETURN FROM DINSR INFO=0
COMPUTED RECIPROCAL CONDITION NUMBER BY DINSR: 0.2E+00

\[
\text{UPDATED MATRIX} Q:
\begin{align*}
-0.258 & \quad 0.095 & -0.470 \\
-0.516 & \quad -0.095 & -0.671 \\
-0.258 & \quad 0.949 & 0.146 \\
-0.775 & \quad -0.285 & 0.353
\end{align*}
\]

\[
\text{UPDATED MATRIX} R:
\begin{align*}
-3.873 & \quad -1.291 & 0.516 \\
0.000 & \quad 1.756 & 3.085 \\
0.000 & \quad 0.000 & 1.403
\end{align*}
\]

ABS( ELEMENT OF LARGE MAGNITUDE OF A-Q*R ): 0.7E-15
ABS( ELEMENT OF LARGE MAGNITUDE OF Q'*Q-I ): 0.7E-15

RANK ONE MATRIX V+U' ADDED TO A, Q AND R UPDATED BY DRNK1

\[
\text{VECTOR} V:
\begin{align*}
0.500 & \quad 0.500 \\
0.200 & \quad 0.500
\end{align*}
\]

\[
\text{VECTOR} U:
\begin{align*}
1.000 & \quad -1.000 \\
1.000
\end{align*}
\]

ON RETURN FROM DRNK1 INFO=1

\[
\text{UPDATED MATRIX} Q:
\begin{align*}
-0.275 & \quad 0.460 & -0.097 \\
-0.458 & \quad 0.598 & -0.398 \\
-0.550 & \quad 0.037 & 0.828 \\
-0.642 & \quad -0.656 & -0.384
\end{align*}
\]

\[
\text{UPDATED MATRIX} R:
\begin{align*}
-5.454 & \quad 0.000 & -2.292 \\
0.000 & \quad 0.000 & -0.412 \\
0.000 & \quad 0.000 & 4.536
\end{align*}
\]

ABS( ELEMENT OF LARGE MAGNITUDE OF A-Q*R ): 0.5E-15
ABS( ELEMENT OF LARGE MAGNITUDE OF Q'*Q-I ): 0.4E-15

RANK REVEALING QR FACTORIZATION BY DRPHM
ON RETURN FROM DRPHM INFO=0
DELTA ON EXIT FROM DRPHM: 0.8E-16
POSITION OF ELEMENTS OF MAX MAGNITUDE OF SUCCESSIVELY COMPUTED SINGULAR VECTORS BY DRPHM: 2 2 2
DRPHM DETERMINED QR FACTORIZATION OF MATRIX A(3,IP(3)), WHERE
IP(1), IP(2), ..., IP(3) = 1 3 2

\[
\text{MATRIX} Q \text{ FOR COLUMN PERMUTATION A:}
\begin{align*}
-0.275 & \quad 0.138 & -0.449 \\
-0.458 & \quad 0.450 & -0.559 \\
-0.550 & \quad -0.021 & -0.112 \\
-0.642 & \quad 0.323 & 0.688
\end{align*}
\]

\[
\text{MATRIX} R \text{ FOR COLUMN PERMUTATION A:}
\begin{align*}
-5.454 & \quad -2.292 & 0.000 \\
0.000 & \quad -4.535 & 0.000 \\
0.000 & \quad 0.000 & 0.000
\end{align*}
\]

ABS( ELEMENT OF LARGE MAGNITUDE OF A-Q*R ): 0.5E-15
ABS( ELEMENT OF LARGE MAGNITUDE OF Q'*Q-I ): 0.4E-15

\[
\text{MATRIX} A \text{ AFTER COLUMN PERMUTATION:}
\begin{align*}
1.500 & \quad 0.000 & 0.000 \\
2.500 & \quad -1.000 & 0.000 \\
3.000 & \quad 5.000 & 0.000 \\
3.500 & \quad 0.000 & 0.000
\end{align*}
\]
Example 4.2. Execution times for subroutines DDELCO and DRNK1 are compared for scalar and vector arithmetic. The measured cpu times differed somewhat between different executions of the same code. Therefore the reported times are rounded to one significant digit and the quotient of measured cpu times are rounded to the nearest multiple of 1/2.

Table 4.1 shows the cpu times for DDELCO. This routine and its subroutines have been 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 $A \in \mathbb{R}^{m \times n}$, Table 4.1 shows the cpu time required by DDELCO to compute the QR decomposition of $\tilde{A} \in \mathbb{R}^{m \times (n-1)}$ obtained by deleting column one of $A$.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>scalar arithmetic</th>
<th>vector arithmetic</th>
<th>scalar time</th>
<th>vector time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>4.10^{-4}</td>
<td>4.10^{-4}</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>4.10^{-4}</td>
<td>4.10^{-4}</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>5.10^{-4}</td>
<td>4.10^{-4}</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>6.10^{-4}</td>
<td>4.10^{-4}</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>75</td>
<td>10</td>
<td>8.10^{-4}</td>
<td>4.10^{-4}</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>128</td>
<td>10</td>
<td>1.10^{-3}</td>
<td>5.10^{-4}</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
<td>7.10^{-3}</td>
<td>2.10^{-3}</td>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>1280</td>
<td>10</td>
<td>9.10^{-3}</td>
<td>3.10^{-3}</td>
<td>3.5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table 4.1: Timings for DDELCO
Table 4.2 is similar to Table 4.1 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.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>scalar arithmetic</th>
<th>vector arithmetic</th>
<th>scalar time</th>
<th>vector time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>12</td>
<td>1.10^{-3}</td>
<td>1.10^{-3}</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>25</td>
<td>4.10^{-3}</td>
<td>3.10^{-3}</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>64</td>
<td>50</td>
<td>2.10^{-2}</td>
<td>7.10^{-3}</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>128</td>
<td>100</td>
<td>6.10^{-2}</td>
<td>2.10^{-2}</td>
<td>2.5</td>
<td>4.5</td>
</tr>
<tr>
<td>1024</td>
<td>100</td>
<td>4.10^{-1}</td>
<td>8.10^{-2}</td>
<td>4.5</td>
<td>5</td>
</tr>
<tr>
<td>1250</td>
<td>100</td>
<td>5.10^{-1}</td>
<td>9.10^{-1}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Timings for DRNK1

Example 4.3. Execution times for subroutines written by Buckley [2] and those of Table 1.1 are compared. The vectorized and scalar codes were generated as explained in Example 4.2. 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 than the vectorized subroutines of Table 1.1. 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 4.3-4.6 present some sample timings.
Table 4.3: The first row of $A = QR$ is deleted. Cpu times for vectorized code for updating $Q$ and $R$ are given in seconds; $n = 10$.

<table>
<thead>
<tr>
<th>$m$</th>
<th>time for</th>
<th>time for</th>
<th>time for DELROW [2]</th>
<th>time for DDELR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$3 \cdot 10^{-5}$</td>
<td>$9 \cdot 10^4$</td>
<td>$3.5 \cdot 10^1$</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>$7 \cdot 10^{-5}$</td>
<td>$2 \cdot 10^3$</td>
<td>$2 \cdot 10^1$</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>$8 \cdot 10^{-4}$</td>
<td>$3 \cdot 10^3$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>1024</td>
<td>$4 \cdot 10^{-3}$</td>
<td>$2 \cdot 10^2$</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.4: 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 $A$, i.e., $IFLAG = 0$ on entry.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>time for</th>
<th>time for</th>
<th>time for DELCOL [2]</th>
<th>time for DDELC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>10</td>
<td>$1 \cdot 10^{-4}$</td>
<td>$2 \cdot 10^{-4}$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1024</td>
<td>100</td>
<td>$1 \cdot 10^{-4}$</td>
<td>$7 \cdot 10^{-3}$</td>
<td></td>
<td>$7.5 \cdot 10^1$</td>
</tr>
<tr>
<td>1280</td>
<td>100</td>
<td>$1 \cdot 10^{-4}$</td>
<td>$9 \cdot 10^{-3}$</td>
<td></td>
<td>$9.5 \cdot 10^1$</td>
</tr>
</tbody>
</table>

Table 4.5: A new first column is inserted into $A = QR$. Cpu times for vectorized code for updating $Q$ and $R$ are given in seconds; $n = 10$.

<table>
<thead>
<tr>
<th>$m$</th>
<th>time for</th>
<th>time for</th>
<th>time for INSCOL [2]</th>
<th>time for DINSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>$1 \cdot 10^{-3}$</td>
<td>$2 \cdot 10^{-3}$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>128</td>
<td>$1 \cdot 10^{-3}$</td>
<td>$3 \cdot 10^{-3}$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1024</td>
<td>$7 \cdot 10^{-3}$</td>
<td>$2 \cdot 10^{-2}$</td>
<td></td>
<td>$2.5$</td>
</tr>
</tbody>
</table>

Tables 4.3-4.5 present timings for vectorized code. The next table shows timings for scalar code for the same updatings as in Table 4.3. Table 4.6 shows that, without vectorization, DELROW [2] requires 50% more cpu time than DDELR for moderately large problems.
<table>
<thead>
<tr>
<th>m</th>
<th>time for DDELR</th>
<th>time for DELROW [2]</th>
<th>time for DELRPW [2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.10^{-5}</td>
<td>6.10^{-4}</td>
<td>3.10^{-1}</td>
</tr>
<tr>
<td>64</td>
<td>1.10^{-3}</td>
<td>2.10^{-3}</td>
<td>1.5</td>
</tr>
<tr>
<td>128</td>
<td>2.10^{-3}</td>
<td>3.10^{-3}</td>
<td>1.5</td>
</tr>
<tr>
<td>1024</td>
<td>1.10^{-2}</td>
<td>2.10^{-2}</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Table 4.6: The first row of $A = QR$ is deleted. Cpu times for scalar code for updating $Q$ and $R$ are given in seconds; $n = 10$.

ACKNOWLEDGEMENTS

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

16
REFERENCES


APPENDIX, Driver for Example 4.1

C DRIVER FOR UPDATING THE QR DECOMPOSITION OF A 4 BY 3 MATRIX.

C

PARAMETER(LDA=4, LDB=3)
INTEGER M,N,I,J,INFO,J,NBIT
INTEGER IPSEV(0:5),IP(LDB)
REAL*8 A(LDA,LDB),Q(LDA,LDB),R(LDB,LDB),B(LDA,LDB)
REAL*8 WORK2(LDA+2*LDB+1),W(200),V(LDA),U(LDB)
REAL*8 RCOND,DELT

C

INITIALIZATION OF A,Q,R,M,N

C

DATA Q(5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1,
5D-1, 5D-1, 5D-1, 5D-1), R(5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1,
5D-1, 5D-1, 5D-1, 5D-1), B(5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1, 5D-1,
5D-1, 5D-1, 5D-1, 5D-1), DATA N/4., N/3., N/2.

C

WRITE(6,10)'A=Q*R, MATRIX A:'
FORMAT(1X,70A)
CALL HATPQ(A,LDA,M,N)
WRITE(6,*)'MATRIX Q:'
CALL HATPQ(Q,LDA,M,N)
WRITE(6,*)'MATRIX R:'
CALL HATPQ(R,LDB,N,N)

C

DELETE COLUMN 2 OF A, UPDATE Q AND R

K=2
DO 20 I=K+1,M
 DO 20 J=1,M
 A(J,J-1)=A(J,J)
 20 CONTINUE

C

INFO=1
WRITE(6,10)'DELETE COLUMN 2 OF A AND UPDATE Q AND R BY DDLC:'
CALL DDLQ(A,LDA,M,N,R,LDB,V,V,INFO)
WRITE(6,40)'DDLC', INFO
FORMAT('ON RETURN FROM ',AS,' INFO=',I1)

C

FOR ALL SUBROUTINES INFO=0 ON EXIT INDICATES SUCCESSFUL TERMINATION

WRITE(6,50)(V(I),I=1,M)
FORMAT('COLUMN RECOMPUTED BY DDLC:',4F8.3)

N=N-1
WRITE(6,10)'UPDATED MATRIX Q:'
CALL HATPQ(Q,LDA,M,N)
WRITE(6,*)'UPDATED MATRIX R:'
CALL HATPQ(R,LDB,N,N)

C

PRINT MAGNITUDE OF ELEMENT OF RESIDUAL MATRIX A-Q*R OF LARGEST
MAGNITUDE, AS WELL AS FOR MATRIX Q**Q-I

CALL MAXRES(A,LDA,Q,R,LDB,M,N)

CALL ORTCKQ(Q,LDA,M,N,WORK)

C

DELETE ROW 3 OF A, UPDATE Q AND R

K=3
DO 100 I=1,N
 DO 100 J=K+1,M
 A(J-1)=A(J,I)
 100 CONTINUE

A(M,1)=0DO

C

WRITE(6,10)'DELETE ROW 3 OF A AND UPDATE Q AND R BY DDEL:'
CALL DDELQ(A,LDA,M,N,R,LDB,V,V,WORK,INFO)
WRITE(6,40)'DDEL', INFO
WRITE(6,120)(U(I),I=1,N)

C

FORMAT('ROW RECOMPUTED BY DDEL: ',4F8.3)

N=N-1
WRITE(6,10)'UPDATED MATRIX Q:'
CALL HATPQ(Q,LDA,M,N)
WRITE(6,*)'UPDATED MATRIX R:'
CALL HATPQ(R,LDB,N,N)

C

PRINT MAGNITUDE OF ELEMENT OF RESIDUAL MATRIX A-Q*R OF LARGEST
MAGNITUDE, AS WELL AS FOR MATRIX Q**Q-I

CALL MAXRES(A,LDA,Q,R,LDB,M,N)
CALL ORTCKQ(Q,LDA,M,N,WORK)

C

INSERT NEW FIRST COLUMN V INTO A, UPDATE Q AND R

K=1
DO 200 I=1,N,K,-1
 DO 200 J=1,M
 A(J,J)=A(J,I)
 200 CONTINUE

C

WRITE(6,230)(V(I),I=1,M)
FORMAT('NEW 1ST COLUMN TO BE INSERTED INTO A BY DINSC:',4F8.3)
CALL DINSCQ(A,LDA,M,N,R,LDB,V,V,WORK,INFO)
WRITE(6,40)'DINSC', INFO
WRITE(6,240)RCOND

C

FORMAT('COMPUTED RECIPROCAL CONDITION NUMBER BY DINSC:',E8.1)
WRITE(6,10)'UPDATED MATRIX Q:'
CALL HATPQ(Q,LDA,M,N)
WRITE(6,*)'UPDATED MATRIX R:'
CALL MATPR1(R,LDR,M,N)
CALL PRINT MAGNITUDE OF ELEMENT OF RESIDUAL MATRIX A-Q*R OF LARGEST MAGNITUDE, AS WELL AS FOR MATRIX Q*R-1
CALL MAXRES(A,LDA,Q,R,LDR,M,N)
CALL ORTHQR(Q,LDA,M,N,WORK)

310 CONTINUE
U(J)=J
A(K,J)=U(J)
300 CONTINUE
M=M+1
WRITE(6,320)(U(I),I=1,N)
320 FORMAT(6,320)(U(I),I=1,N)
WRITE(6,400)DINSR(Q,LDA,M,N,R,LDR,K,WORK,INFO)
WRITE(6,400)DINSR(Q,LDA,M,N,R,LDR,K,WORK,INFO)
WRITE(6,10)UPDATED MATRIX Q:
CALL MATPR1(Q,LDA,M,N)
WRITE(6,10)UPDATED MATRIX R:
CALL MATPR1(R,LDR,M,N)

CALL MAXRES(A,LDA,Q,R,LDR,M,N)
CALL ORTHQR(Q,LDA,M,N,WORK)

C PRINT MAGNITUDE OF ELEMENT OF RESIDUAL MATRIX A-Q*R OF LARGEST MAGNITUDE, AS WELL AS FOR MATRIX Q*R-1
C
CALL MAXRES(A,LDA,Q,R,LDR,M,N)
CALL ORTHQR(Q,LDA,M,N,WORK)
C
C ADD RANK ONE MATRIX U*V TO A, UPDATE Q AND R
C
DO 400 I=1,N
V(I)=50-1
400 CONTINUE
V(3)=200
DO 410 I=1,N
U(I)=(-1)**(I+1)
410 CONTINUE

DO 420 J=1,N
DO 430 I=1,N
A(I,J)=A(I,J)+U(J)*V(I)
430 CONTINUE
420 CONTINUE

WRITE(6,10) RANK ONE MATRIX V*U'" ADDED TO A, Q AND R UPDATED BY DRNK1'
WRITE(6,640)'V*(V(I),I=1,N)

440 FORMAT(/IX,'VECTOR '+IAX', I=1,N)
WRITE(6,450)'U',(U(I),I=1,N)
450 FORMAT(VECTOR '+IAX', I=1,N)
WRITE(6,460)'DRNK1',INFO
460 FORMAT(/IX,'ON RETURN FROM '+'AS', INFO=',,)
WRITE(6,10)'UPDATED MATRIX Q:'
CALL MATPR1(Q,LDA,M,N)
WRITE(6,10)'UPDATED MATRIX R:'
CALL MATPR1(R,LDR,M,N)

C PRINT MAGNITUDE OF ELEMENT OF RESIDUAL MATRIX A-Q*R OF LARGEST MAGNITUDE, AS WELL AS FOR MATRIX Q*R-1. PRINT A.
C
CALL MAXRES(A,LDA,Q,R,LDR,M,N)
CALL ORTHQR(Q,LDA,M,N,WORK)
C
C RANK REVEALING QR FACTORIZATION, INITIALIZE PERMUTATION VECTOR
C
DO 500 J=1,N
IP(J)=J
500 CONTINUE
K=M
K=K+2
DELTA=10

WRITE(6,10)'RANK REVEALING QR FACTORIZATION BY DRPH'
CALL DRPH(Q,LDA,M,N,R,LDR,K,IP,DELTA,MNBT,IPOS,WORK,INFO)
WRITE(6,440)'DRPH',INFO
WRITE(6,510)DELTA

510 FORMAT(DELTA ON EXIT FROM DRPH:', E8.1)
WRITE(6,520)(IPOS(J),J=1,NOBIT)
520 FORMAT(POSITION OF ELEMENTS OF MAX MAGNITUDE OF SUCCESSIVELY ',
, SINGULAR VECTORS BY DRPH:',,513)
WRITE(6,530)(IP(J),J=1,N)
530 FORMAT(DRPH DETERMINED QR FACTORIZATION OF MATRIX ',
, T/A(*, IP(J)), WHERE',/IX', IP(J), IP(2), ... = ',,1013)
WRITE(6,10) 'MATRIX Q FOR COLUMN PERMUTED MATRIX A:'
CALL MATPR1(Q,LDA,M,N)
WRITE(6,10) 'MATRIX R FOR COLUMN PERMUTED MATRIX A:'
CALL MATPR1(R,LDR,M,N)

C PERMUTE COLUMNS OF A ACCORDING TO IP, STORE IN B
C
DO 540 J=1,N
DO 550 K=1,N
B(K,J)=A(K,IP(J))
550 CONTINUE
540 CONTINUE

C PRINT MAGNITUDE OF ELEMENT OF RESIDUAL MATRIX B-Q*R OF LARGEST MAGNITUDE, AS WELL AS FOR MATRIX Q*R-1
C


CALL MAXRES(A,LDA,Q,R,LDR,M,N)
CALL ORTCHK(Q,LDA,M,N,WORK)
C
WRITE(6,10)'MATRIX A AFTER COLUMN PERMUTATION:
CALL MATPRI(A,LDA,M,N)
C
STOP
END
C
C SUBROUTINES FOR COMPUTING RESIDUALS AND OUTPUT
C
---------------------------------------------
SUBROUTINE MATPRI(A,LDA,M,N)
C
MATPRI PRINTS MATRIX A
C
INTEGER LDA,M,N,IROW,ICOL
REAL*8 A(LDA,N)
C
DO 10 IROW=1,M
WRITE(6,20)(A(IROW,ICOL), ICOL=1,N)
10  CONTINUE
WRITE(6,*)
RETURN
20  FORMAT(*F8.3)
END
C
---------------------------------------------
SUBROUTINE ORTCHK(Q,LDA,M,N,WK)
C
ORTCHK COMPUTES Q'*Q-1 FOR THE M BY N MATRIX Q.
C
INTEGER LDA,M,N,IROW,ICOL,K
REAL*8 Q(LDA,M),SUM,HX
C
MX=0DO
DO 10 IROW=1,N
DO 20 ICOL=1,M
SUM=0DO
DO 30 K=1,M
SUM=SUM+Q(K,IROW)*Q(K,ICOL)
30  CONTINUE
IF(IROW.EQ.ICOL)SUM=SUM-1DO
IF(HX.LT.DABS(SUM))HX=DABS(SUM)
20  CONTINUE
10  CONTINUE
C
WRITE(6,40)HX
40  FORMAT(ABS ELEMENT OF LARGEST MAGNITUDE OF Q'*Q-1 :','
*>8.1,1X)
RETURN
END
C
---------------------------------------------
SUBROUTINE MAXRES(A,LDA,Q,R,LDR,M,N)
C
REAL*8 A(LDA,N),Q(LDA,M),R(LDR,N),SUM,HX
DISTRIBUTION LIST

DIRECTOR (2)
DEFENSE TECH. INFORMATION CENTER, CAMERON STATION
ALEXANDRIA, VA 22314

DIRECTOR OF RESEARCH ADMIN.
CODE 012
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

LIBRARY (2)
CODE 0142
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

DEPARTMENT OF MATHEMATICS
CODE 53
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

CENTER FOR NAVAL ANALYSES
4401 FORD AVENUE
ALEXANDRIA, VA 22302-0268

PROFESSOR WILLIAM J. RAGG (15)
CODE 53Gr
DEPARTMENT OF MATHEMATICS
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

NATIONAL SCIENCE FOUNDATION
WASHINGTON, D.C. 20550