STAR ADAPTATION OF QR ALGORITHM

Shantilal N. Shah

HAMPTON INSTITUTE
Hampton, Virginia 23668

June 1981
SUMMARY

The QR algorithm used to solve over-determined systems of linear equations was adapted to execute efficiently on the Control Data STAR-100 computer. Using the new vectorized algorithm, the STAR-100 computer solved a system of 250 equations in 50 unknowns in less than 8.5% of the time it took using the original scalar version. This paper describes how the scalar program was adapted for the STAR-100 and indicates why these adaptations yielded an efficient STAR program. Program listings of the old scalar version and of the new vectorized SL/1 version are presented in the appendices. Execution times for the two versions, applied to the same system of linear equations, are compared.

INTRODUCTION

Programs written in standard FORTRAN language for serial computers do run on the Control Data STAR-100 computer, but very inefficiently. To take advantage of the architecture and vector-processing capabilities of the STAR-100 computer it is necessary to vectorize the algorithms in these programs. Frequently one must rearrange the data and computations. This paper describes how the QR algorithm to solve over-determined systems of linear equations was vectorized and what factors were considered in developing an efficient STAR program.

The vectorized program utilizes SL/1, a high level language developed by NASA's Langley Research Center for the STAR-100 computer. SL/1 incorporates many features designed to see that programs it compiles take full advantage of the STAR's architecture and capabilities, including half-word storage and arithmetic. SL/1 is compatible with FORTRAN in the sense that programs written in either language can call subroutines written in the other. In utilizing the program presented in this paper, familiarity with some of the notations used in the SL/1 language will be helpful.
General suggestions concerning the adaptations of algorithms for efficient use on the STAR may be found in the paper, "Star Adaptation for Two Algorithms used on Serial Computer," by Lona M. Howser and Jules J. Lambiotte (see ref. 1).

ADAPTATION OF QR ALGORITHM TO SOLVE OVER-DETERMINED SYSTEMS OF LINEAR EQUATIONS

The NASA computer mathematics library presently has a subroutine called QRASOS, written in FORTRAN for serial computers, to solve an over-determined system of linear equations. This subroutine decomposes the matrix A of the system AX=B using Householder transformations. (For details of this algorithm see ref. 2). To compute these transformations, it uses subroutines SAXPY, SSCAL, SCOPY and function SDOT from the Basic Linear Algebra Subroutines (BLAS). For an mxn (m>n) matrix A it makes $n^2$ calls to these subroutines and functions to solve the given system. Subroutine calls are very expensive on the STAR-100 computer.

In SL/1, a matrix can be stored either column-wise or row-wise. Column storage means that elements in one column of the matrix are stored as one vector (contiguous locations). Similarly, row-storage means that elements in one row are stored as one vector (contiguous locations). In vectorizing this algorithm both row and column storage of matrices A and B were tried.

With row-wise storage, reordering of the scalar-version computations is required but use of the inner product macro to decompose matrix A is avoided. With column-wise storage the computational steps are the same as in the scalar versions with vector instructions replacing scalar instructions, but use of the dot product macro is required. It was expected that, because of the avoidance of the dot product macro, the row-wise approach would offer a considerable saving in CPU time.

Test results show that in using the STAR computer for this algorithm both
vectorized versions offer considerable CPU time savings over the scalar program, but that contrary to expectations column-wise storage is more efficient than row-wise storage (see table).

<table>
<thead>
<tr>
<th>Size of Matrix A</th>
<th>New Vectorized Version (Column Storage)</th>
<th>New Vectorized Version (Row Storage)</th>
<th>Old Scalar Version</th>
<th>New Vectorized Version (Column Storage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>250 x 200</td>
<td>2.169</td>
<td>2.695</td>
<td>26.34</td>
<td>2.239</td>
</tr>
<tr>
<td>120 x 100</td>
<td>.405</td>
<td>.588</td>
<td>3.396</td>
<td>.433</td>
</tr>
<tr>
<td>250 x 50</td>
<td>.158</td>
<td>.863</td>
<td>2.223</td>
<td>.178</td>
</tr>
<tr>
<td>100 x 10</td>
<td>.006</td>
<td>.069</td>
<td>0.056</td>
<td>.009</td>
</tr>
<tr>
<td>200 x 30</td>
<td>.053</td>
<td>.416</td>
<td>0.698</td>
<td>.063</td>
</tr>
</tbody>
</table>

Algorithms for both row-wise storage and column storage to decompose the matrix A are given. Back substitutions are not discussed here. In both algorithms, A is an mxn matrix and WK is a vector of length n.

COLUMN STORAGE

When matrix A is stored column-wise, the decomposition of A is achieved as follows: (Note: All references to $i^{th}$ column refer to column entries on and below the diagonal).

1. Take the inner product of $i^{th}$ column with itself and store in $WK_i$
2. Take the square root of $WK_i$
3. If $WK_i = 0$, go to step (10)
4. $WK_i = WK_i \times \text{Sign of } A_{i,i}$
5. Divide column $i$ by $WK_i$
6. Add 1 to $A_{i,i}$
These 6 steps compute the Householder transformation for column $i$.

To apply this transformation to the columns $K = i+1, \ldots, n$ do the following steps:

1. Take the inner product of column $i$ with column $K$ and store the result in $t$
2. Divide $t$ by $A_{i,i}$ and then store the negative of the result in $t$
3. Multiply column $i$ by $t$ and add the result to column $K$
4. Store $A_{i,i}$ in $t$, $-WK_i$ in $A_{i,i}$ and $t$ in $WK_i$

When $i=n$ perform steps 1 thru 6 and 10.

**ROW STORAGE**

When the matrix $A$ is stored row-wise, the decomposition of $A$ is achieved as follows: (Note: In the steps 1 thru 6 below, all references to the row $j$ in the $i^{th}$ step of decomposition refer to the entries on and to the right of the diagonal. All references to the vector $WK$ refer to its $i^{th}$, $(i+1)^{th}$, ..., $n^{th}$ elements. In steps 7 thru 10 all references to the row $j$ in the $i^{th}$ step of decomposition refer to the entries to the right of the diagonal and all references to the vector $WK$ refer to its $(i+1)^{th}$, $(i+2)^{th}$, ..., $n^{th}$ elements).

At $i^{th}$ step of decomposition ($i=1, 2, \ldots, n-1$).

1. Set $WK=0$
2. For $j=1, 2, \ldots, m$, multiply row $j$ by $A_{j,i}$ and add the result to $WK$
3. Take the square root of $WK_i$
4. If $WK_i=0$ go to step 11
5. Multiply $WK_i$ by sign of $A_{i,i}$
6. Divide $A_{i,i}$ by $WK_i$ and add 1 to the result
7. Divide $WK$ by $WK_i$ and add row $i$ of $A$ to $WK$
8. Divide $WK$ by $-A_{i,i}$
9. For $j=i+1, i+2, \ldots, m$
   Divide $A_{j,i}$ by $WK_i$
(10) For \( j = i, i+1, \ldots, m \), multiply \( WK \) by \( A_{j,i} \) and add the result to row \( j \) of \( A \).

(11) Store \( A_{i,i} \) in \( t \), \(-WK\) in \( A_{i,i} \) and \( t \) in \( WK \).

When \( i = n \), perform Steps 1 thru 6 and 11.

**WHY ROW-STORAGE IS SLOWER THAN COLUMN-STORAGE**

As pointed out earlier, if the matrix \( A \) is stored row-wise, the use of the inner product macro is avoided and the computation of the Householder transformations and their application to other columns at each step of the decomposition is accomplished by the use of a vector multiplication by a scalar and then a vector addition. This should result in a considerable savings of the CPU time for a large matrix. However, our numerical experiments show just the opposite. This can be explained as follows: When an \( mxn \) \( (m \geq n) \) matrix \( A \) is stored row-wise, the vector lengths in that algorithm are proportional to \( n \), the smaller dimension. On the other hand, for column-wise storage the vector lengths are proportional to \( m \), the larger dimension. Equivalently, we see that the row-stored algorithm requires more vector start-ups \( ((m-n)(m-n+1)/2 \) more) to do the same number of total computations as the column-stored algorithm, thus requiring more CPU time to do the same amount of work.

Another factor which makes the row-stored algorithm slower is that the transformation elements are stored in the columns of the decomposed matrix. If the matrix is stored row-wise, this leads to additional scalar computations, notably in step 9 of the algorithm. This slows down the computations considerably. Also, if \( m \) is large, then not all \( m \) vectors in row-wise storage reside in the memory at the same time. Because of need to reference different columns at different steps of algorithm, this could lead to excessive paging. Thus, any advantage gained by avoiding the use of the inner product in the row-wise storage is offset by the need to perform many scalar operations, more iterations and excessive paging.
REFERENCES


APPENDIX A

SL/1 Coding of QR Algorithm
MODULE M1 ; 
\$OPT=1;\$SOURCE(1,72); 

/* PURPOSE */ 
/* TO SOLVE M SIMULTANEOUS EQUATIONS IN N UNKNOWNS WITH IP */ 
/* RIGHT HAND SIDES SO THAT THE SOLUTIONS ARE THE BEST POSSIBLE */ 
/* FIT IN THE LEAST SQUARES SENSE. THE ROUTINE USES HOUSE- */ 
/* HOLDER TRANSFORMATIONS TO PERFORM THE QR DECOMPOSITION */ 
/* OF THE COEFFICIENT MATRIX. */ 
/* */ 
/* */ 
/* CALL Q4QRASOS(MAXM,MAXN,M,N,IP,A,B,WT,JOB,X,RSD,SUM,WK,IERR) */ 
/* */ 
/* PARAMETERS */ 
/* */ 
MAXM AN INPUT INTEGER SPECIFYING THE FIRST DIMENSION OF THE 
A,B, AND RSD ARRAYS IN THE CALLING PROGRAM. MAXM MUST 
BE GREATER THAN OR EQUAL TO M. 
/* */ 
MAXN AN INPUT INTEGER SPECIFYING THE FIRST DIMENSION OF THE 
X ARRAY IN THE CALLING PROGRAM. MAXN MUST BE GREATER 
THAN OR EQUAL TO N. 
/* */ 
M AN INPUT INTEGER SPECIFYING THE NUMBER OF ROWS OF THE 
A AND X ARRAYS. M MUST BE GREATER THAN OR EQUAL TO N. 
/* */ 
N AN INPUT INTEGER SPECIFYING THE NUMBER OF COLUMNS OF 
The A ARRAY. 
/* */ 
IP AN INPUT INTEGER SPECIFYING THE NUMBER OF COLUMNS OF 
The B ARRAY. 
/* */ 
A AN INPUT/OUTPUT TWO-DIMENSIONAL ARRAY WITH FIRST DIMEN- 
SION EQUAL TO MAXM AND SECOND DIMENSION AT LEAST N. 
ON INPUT, A MUST CONTAIN THE MATRIX OF COEFFICIENTS OF 
THE SYSTEM OF EQUATIONS. ON OUTPUT, A CONTAINS INFOR- 
MATION DESCRIBING THE QR DECOMPOSITION OF A. 
/* */ 
B AN INPUT TWO-DIMENSIONAL ARRAY WITH FIRST DIMENSION 
equal to MAXM AND SECOND DIMENSION AT LEAST IP. 
The COLUMNS OF B MUST CONTAIN THE IP RIGHT HAND SIDE 
VECTORS. 
/* */ 
WT AN INPUT ONE-DIMENSIONAL ARRAY OF WEIGHTS. IT MUST 
HAVE LENGTH AT LEAST M. IF WEIGHTING IS DESIRED, 
THE FIRST N LOCATIONS MUST CONTAIN REAL NUMBERS GREATER 
THAN ZERO. IF WEIGHTING IS NOT DESIRED, WT [1] MUST BE 
A NEGATIVE REAL NUMBER. 
/* */ 
JOB AN INPUT INTEGER SPECIFYING RESULTS TO BE COMPUTED. 
/* */ 
-1 COMPUTE SOLUTIONS ONLY. 
-2 COMPUTE RESIDUALS ONLY. 
-3 COMPUTE BOTH SOLUTIONS AND RESIDUALS. 
/* */ 
X AN OUTPUT TWO-DIMENSIONAL ARRAY CONTAINING THE SOLU- 
TIONS. X MUST BE DIMENSIONED WITH FIRST DIMENSION 
equal to MAXN AND SECOND DIMENSION AT LEAST IP. IF 
SOLUTIONS ARE DESIRED INTO MATRIX B THEN MAXN MUST BE 
equal to MAXM FOR THIS PARTICULAR CASE. 
/* */
**Source:** Hampton Institute, Hampton VA.

**Language:** SL/1.

**Date Released:** January 18, 1969.

---

ENTRY PROCEDURE Q4GRASD5 (MAXH,MAXH,H,IP,A,B,NT,JOB,X,RSD, SUM,WK,IERR).

REAL VECTOR [MAXH] ARRAY(H) A;
REAL VECTOR [MAXH] ARRAY(IP) X;
REAL VECTOR [MAXH] ARRAY(IP) B,RSD;
REAL VECTOR [H] WT;
REAL VECTOR [H] WK;
REAL VECTOR [IP] SUM;
AUTOMATIC REAL T;
INTEGER I,J,K,L,H,IP,MAXH,IERR,MAXK,JOB;

CHECK FOR H LESS THAN H.

IF H < H THEN IERR: = 1;
GO TO LAB1

ELSE

CHECK FOR WEIGHTING

IF WT(I) > 0 THEN

CHECK FOR ILLEGAL WEIGHTS

1 = SELT(WT,0);
IF I < H THEN IERR: = 3;
GO TO LAB1

ELSE

WT(I:1:H) = SQRT(WT(I:1:H));
FOR I:=1 TO IP DO
A(I:i:1:H)=A(I:i:1:H)*WT(I:1:H); END;
FOR I:=1 TO IP DO
B(I:i:1:H)=B(I:i:1:H)*WT(I:1:H); END.
ENDI;
ENDI;

/*/ CALL G4SQRDC TO DECOMPOSE MATRIX A. */
CALL G4SQRDC(A,MAXH,M,N,WK);

/*/ CALL G4SQRSL TO SOLVE IP RIGHT HAND SIDES. */
CALL G4SQRSL(MAXH,MAXH,M,N,IP,A,B,WT,JOB,X,RSD,SUM,WK,IERR); IF IERR>0 THEN IERR:=2 ENDI;
LAB1: ENDP;

*******************************************************************************/

PROCEDURE G4SQRDC (A,MAXH,M,N,WK);
REAL VECTOR [MAXH] ARRAY(A); REAL VECTOR [N] WK;
AUTOMATIC REAL T;
INTEGER I,J,K,L,H,IP,HAXH,IERR,MAXH,JOB;

/*/ COMPUTE HH TRANSFORMATION FOR COLUMN I */

FOR I:=1 TO N-1 DO
WK[I]:= A(I)[I:M] .DOT. A(I)[I:M];
WK[I]:= SQRT(WK[I]); IF WK[I] > 0 THEN
WK[I] := WK[I]*ABS(A(I)[I])/A(I)[I];
A(I)[I]:= A(I)[I:M]/WK[I];
A(I)[I]:= A(I)[I]+1;
ENDF;
ENDF;

*******************************************************************************/

PROCEDURE G4SQRSL (MAXH,MAXH,M,N,IP,A,B,WT,JOB,X,RSD,
SUM,WK,IERR);
REAL VECTOR [MAXH] ARRAY(A); REAL VECTOR [MAXH] ARRAY(IP) X;
REAL VECTOR [MAXH] ARRAY (IP) B; REAL VECTOR [M] WK;
REAL VECTOR (M) WT;
REAL VECTOR [MP] SUM;
INTEGER I,J,K,L,H,IP,MAXH,IERR,MAXH,JOB;
AUTOMATIC REAL T;
IERR:=0;
```plaintext
SPECIAL ACTION WHEN M = 1

IF M = 1 THEN
  IF WK[I] = 0 THEN
    IERR := 1;
    GO TO LAB4
  END;
  IF JOB <> 2 THEN
    FOR I := 1 TO IP DO
      X[I][I] := B[I][I]/A[I][I];
    ENDF;
  ENDF;
  IF JOB <> 1 THEN
    RSD[I][1:IP] := 0.0;
    GO TO LAB4;
  ENDF;
END;

  COMPUTE TRANS(Q*B)

  FOR I := 1 TO N DO
    IF WK[I] <> 0 THEN
      FOR J := 1 TO IP DO
        T := A[I][J].DOT. B[J][1:M];
        T := -T/A[I][I];
      ENDF;
    ENDF;
    FOR I := 1 TO IP DO
      X[I][1:N] := B[I][1:N];
    ENDF;
  IF JOB > 1 THEN

  COMPUTE THE RESIDUES

  FOR I := 1 TO IP DO
    RSD[I][1:IP] := 0.0;
  ENDF;
  FOR I := 1 TO IP DO
    K := N+1;
    RSD[I][K:1] := B[I][K:1];
  ENDF;
  FOR K := N DOWNTO 1 DO
    FOR L := 1 TO IP DO
      IF WK[K] = 0 THEN
        IERR := K; GO TO LAB4
      END;
      T := -T/A[K][K];
      SUML := RSD[L][1:M].DOT. RSD[L][1:M];
  ENDF;
  ENDF;
  IF WT[1:M] > 0 THEN
    FOR I := 1 TO IP DO
      RSD[I][1:M] := RSD[I][1:M]/WT[1:M];
    ENDF;
    WT[1:M] := WT[1:M]*WT[1:M];
  ENDF;
END;
```
IF JOB <> 2 THEN

/*
  COMPUTE THE SOLUTIONS
  */

FOR I := N DOWNTO 2 DO
  IF WK[I] = 0 THEN
    IERR:=I; GO TO LAB4
  ELSE
    K:=I-1;
    FOR J := 1 TO IP DO
      X(J)[I]:= -X(J)[I]/WK[I];
      T:= -X(C)[I];
      X(J)[I+K]:= X(J)[I+K] + T*A(1)[I+K];
    ENDF;
    ENDF;
    FOR I := 1 TO IP DO
      IF WK[I] = 0 THEN
        IERR:=I; GO TO LAB4
      ELSE
        X(I)[I]:= -X(I)[I]/WK[I];
      ENDI;
    END;

  /*
  SAVE THE TRANSFORMATION
  */

LAB4: FOR I := 1 TO N DO
  T:=A(I)[I]; A(I)[I]:= -WK[I]; WK[I]:= T;
ENDF;
ENDP;
ENDM;
END OF PROGRAM
APPENDIX B

FORTRAN Coding of QR Algorithm
SUBROUTINE QRASOS(MAXM,MAXN,M,N,IP,A,C,WT,JOB,X,RSD,SUM,WK,IERR) QRAS0010

C  PURPOSE
C  TO SOLVE M SIMULTANEOUS EQUATIONS IN N unknowns with IP
C  RIGHT HAND SIDES so that the solutions are the best possible
C  FIT IN THE LEAST SQUARES SENSE. THE ROUTINE USES HOUSE-
C  HOLDER TRANSFORMATIONS TO PERFORM THE QR DECOMPOSITION
C  OF THE COEFFICIENT MATRIX.
C
C  USE
C
C  CALL QRASOS(MAXM,MAXN,M,N,IP,A,B,WT,JOB,X,RSD,SUM,WK,IERR)
C
C  PARAMETERS
C
C  MAXM AN INPUT INTEGER SPECIFYING THE FIRST DIMENSION OF THE
C  A,B, AND RSD ARRAYS IN THE CALLING PROGRAM. MAXM MUST
C  BE GREATER THAN OR EQUAL TO M.
C
C  MAXN AN INPUT INTEGER SPECIFYING THE FIRST DIMENSION OF THE
C  X ARRAY IN THE CALLING PROGRAM. MAXN MUST BE GREATER
C  THAN OR EQUAL TO N.
C
C  M AN INPUT INTEGER SPECIFYING THE NUMBER OF ROWS OF THE
C  A AND B ARRAYS. M MUST BE GREATER THAN OR EQUAL TO N.
C
C  N AN INPUT INTEGER SPECIFYING THE NUMBER OF COLUMNS OF
C  THE A ARRAY.
C
C  IF AN INPUT INTEGER SPECIFYING THE NUMBER OF COLUMNS OF
THE B ARRAY.

AN INPUT/OUTPUT TWO-DIMENSIONAL ARRAY WITH FIRST DIMENSION EQUAL TO MAXM AND SECOND DIMENSION AT LEAST IP.

ON INPUT, A MUST CONTAIN THE MATRIX OF COEFFICIENTS OF THE SYSTEM OF EQUATIONS. ON OUTPUT, A CONTAINS INFORMATION DESCRIBING THE QP DECOMPOSITION OF A.

AN INPUT TWO-DIMENSIONAL ARRAY WITH FIRST DIMENSION EQUAL TO MAXM AND SECOND DIMENSION AT LEAST IP.

THE COLUMNS OF B MUST CONTAIN THE IP RIGHT HAND SIDE VECTORS.

AN INPUT ONE-DIMENSIONAL ARRAY OF WEIGHTS. IF WEIGHTING IS DESIRED, WT MUST HAVE LENGTH AT LEAST M, AND THE FIRST M LOCATIONS MUST CONTAIN REAL NUMBERS GREATER THAN ZERO. IF WEIGHTING IS NOT DESIRED, WT CAN CONSIST OF A SINGLE LOCATION WHICH MUST CONTAIN A NEGATIVE REAL NUMBER.

AN INPUT INTEGER SPECIFYING RESULTS TO BE COMPUTED.

-1 COMPUTE SOLUTIONS ONLY.
-2 COMPUTE RESIDUALS ONLY.
-3 COMPUTE BOTH SOLUTIONS AND RESIDUALS.

AN OUTPUT TWO-DIMENSIONAL ARRAY CONTAINING THE SOLUTIONS. IF JOB=1 OR JOB=3, X MUST BE DIMENSIONED WITH FIRST DIMENSION EQUAL TO MAXM AND SECOND DIMENSION AT LEAST IP. IF JOB=2, X CAN BE A DUMMY PARAMETER.
RSN  AN OUTPUT TWO-DIMENSIONAL ARRAY CONTAINING THE RESIDUALS. IF JOB=2 OR JOB=3, RSN MUST BE DIMENSIONED WITH FIRST DIMENSION EQUAL TO MAXM AND SECOND DIMENSION AT LEAST IP. IF JOB=1, RSN CAN BE A DUMMY PARAMETER.

SUM  AN OUTPUT ONE-DIMENSIONAL ARRAY CONTAINING THE WEIGHTED SUMS OF SQUARES OF THE RESIDUALS. IF JOB=2 OR JOB=3, SUM MUST BE DIMENSIONED AT LEAST IP. IF JOB=1, SUM CAN BE A DUMMY PARAMETER.

WK   A ONE-DIMENSIONAL WORK ARRAY WHICH MUST BE DIMENSIONED AT LEAST N. ON OUTPUT, WK CONTAINS INFORMATION ON THE QR DECOMPOSITION OF A.

IERR  AN INTEGER ERROR CODE.

-0 NO ERROR DETECTED.
-1 N IS GREATER THAN M.
-2 THE DECOMPOSED MATRIX IS SINGULAR.
-3 WEIGHTING WAS REQUESTED AND ONE OR MORE WEIGHTS IS NEGATIVE.

REQUIRED ROUTINES NORMS,SQRDC2,SQRSL2,SAXPY1,SDOT1,SSCAL

SCOPY

FORTRAN FUNCTIONS ABS,AMAX1,MINO,MOD,SIGN,SQRT

SOURCE COMPUTER SCIENCES CORPORATION,
HAMPSTEAD, VA.

LANGUAGE FORTRAN
DIMENSION A(MAXM,1),B(MAXM,1),X(MAXN,1),RSD(MAXM,1),WT(1),WK(1)
DIMENSION SUM(I)
IERR = 0

CHECK FOR M LESS THAN N.

IF(M .GE. N) GO TO 10
    IERR = 1
    GO TO 160

CHECK FOR NO WEIGHTING

10 IF(WT(1) .LT. 0.0) GO TO 80

CHECK FOR ILLEGAL WEIGHTS

DO 20 I = 2, M
    IF(WT(I) .LE. 0.0) GO TO 30
20  CONTINUE
GO TO 40

IERR = 3
GO TO 160

WEIGHT THE A AND B ARRAYS BY THE SQUARE ROOT
C OF THE WEIGHT ARRAY.

40 DO 70 I = 1, M
   WT(I) = SQRT(WT(I))
   DO 50 J = 1, N
      A(I,J) = WT(I)*A(I,J)
   50 CONTINUE
   DO 60 J = 1, IP
      B(I,J) = WT(I)*B(I,J)
   60 CONTINUE
   70 CONTINUE
   80 CONTINUE

CALL SQRDC2 TO DECOMPOSE A

CALL SQRDC2(A,MAXM,M,N,WK)

CALL SQRSL2 TO SOLVE FOR IP RIGHT HAND SIDES

CALL SQRSL2(A,MAXM,M,N,MAXN,IP,WK,E,X,RSD,JOB,IERF)

IF(IERR .EQ. 0) GO TO 90
   IERR = 2
   GO TO 160

90 CONTINUE

C COMPUTE THE SUM OF WEIGHTED SQUARES OF RESIDUALS.

IF(JOB .EQ. 1) GO TO 140
   DO 110 J = 1, IP
      SUM(J) = 0.0
   110 DO 100 I = 1, M
      SUM(J) = SUM(J) + RSD(I,J)*RSD(I,J)
   100 GO TO 110
100 CONTINUE
110 CONTINUE

C
C COMPUTE UNWEIGHTED RESIDUALS
C
IF(WT(1) .LT. 0.0) GO TO 160
DO 150 I = 1, M
   DO 120 J = 1, IP
      RSD(I,J) = RSD(I,J)/WT(I)
120 CONTINUE
130 CONTINUE
140 CONTINUE
IF(WT(1) .LT. 0.0) GO TO 160
DO 150 I = 1, M
   WT(I) = WT(I)**WT(I)
150 CONTINUE
160 CONTINUE
RETURN
END

SUBROUTINE SQPDC2(X,LDX,N,P,QRAUX)
INTEGER LDX,N,P
REAL X(LDX,1),QRAUX(1)
C
C SQPDC2 USES HOUSEHOLDER TRANSFORMATIONS TO COMPUTE THE QR
C FACTORIZATION OF AN N BY P MATRIX X.
C
C ON ENTRY
C
C X REAL(LDX,P), WHERE LDX .GE. N.
C X CONTAINS THE MATRIX WHOSE DECOMPOSITION IS TO BE
C COMPUTED.
LDX INTEGER.

LDX IS THE LEADING DIMENSION OF THE ARRAY X.

N INTEGER.

N IS THE NUMBER OF ROWS OF THE MATRIX X.

P INTEGER.

P IS THE NUMBER OF COLUMNS OF THE MATRIX X.

ON RETURN

X CONTAINS IN ITS UPPER TRIANGLE THE UPPER
TRIANGULAR MATRIX R OF THE QR FACTORIZATION.
BELOW ITS DIAGONAL X CONTAINS INFORMATION FROM
WHICH THE ORTHOGONAL PART OF THE DECOMPOSITION
CAN BE RECOVERED.

QRAX INTEGER.

QRAX CONTAINS FURTHER INFORMATION REQUIRED TO RECOVER
THE ORTHOGONAL PART OF THE DECOMPOSITION.

LINPACK SUBROUTINE SQRDC VERSION DATED 07/14/77, REVISED BY
COMPUTER SCIENCES CORPORATION, HAMPTON, VA. 10/10/78.

BLAS SAXPY, SDOT, SCAL LPC NORMS

FORTRAN ABSIGN, SQRT, MOD

INTERNAL VARIABLES
INTEGER J,L,LP1
REAL SDOT,NRMXL,T

C
C
C PERFORM THE HOUSEHOLDER REDUCTION OF X.
C
DO 190 L = 1, P
  QRAUX(L) = 0.0E0
  IF (L .EQ. P) GO TO 170
C
  COMPUTE THE HOUSEHOLDER TRANSFORMATION FOR COLUMN L.
C
  NLEN = N-L+1
  CALL NORMS(NLEN,NLEN,1,X(L,L),2,NRMXL)
  IF (NRMXL .EQ. 0.0E0) GO TO 160
    IF (X(L,L) .NE. 0.0E0) NRMXL = SIGN(NRMXL,X(L,L))
    CALL SSCAL(N-L+1,1.0E0/NRMXL,X(L,L),1)
    X(L,L) = 1.0E0 * X(L,L)
C
  APPLY THE TRANSFORMATION TO THE REMAINING COLUMNS.
C
  LP1 = L + 1
  IF (P .LT. LP1) GO TO 150
  GO 140 J = LP1, P
  T = - SDOT1(N-L+1,X(L,L),X(L,J))/X(L,L)
  CALL SAXPY1(N-L+1,T,X(L,L),X(L,J))
140  CONTINUE
150  CONTINUE
C
C
C SAVE THE TRANSFORMATION.
   QRAUX(L) = X(L,L)
   X(L,L) = -HMRXL

160 CONTINUE
170 CONTINUE
180 CONTINUE

RETURN
END

SUBROUTINE SQRSL2(X,LDX,N,K,LDB,IP,QRAUX,Y,BETA,RSD,JOB,INFO)
INTEGER LDX,N,K,LDB,IP,JOB,INFO
REAL X(LDX,1),QRAUX(1),Y(LDX,1),BETA(LDB,1),RSD(LDX,1)

C SQRSL2 APPLIES THE OUTPUT OF THE SUBROUTINE SQRDC2 TO
C COMPUTE A SET OF IP LEAST SQUARES SOLUTIONS AND RESIDUALS. THE
C OUTPUT OF SQRDC2 IS THE DECOMPOSITION OF THE N BY K MATRIX
C X IN THE FORM
C
   X = Q * (R)
   (0)
C WHERE Q IS ORTHOGONAL AND R IS UPPER TRIANGULAR. THIS
C INFORMATION IS CONTAINED IN CODED FORM IN THE ARRAY X
C AND THE ARRAY QRAUX.
C
C ON ENTRY
C
   X    REAL(LDX,K), WHERE LDX .GE. N.
   X CONTAINS THE OUTPUT FROM SQRDC.
C
   LDX    INTEGER.
   LDX IS THE LEADING DIMENSION OF THE ARRAY X.
**C**

**N**  INTEGER.

N IS THE NUMBER OF ROWS OF THE MATRIX X.

**K**  INTEGER.

K IS THE NUMBER OF COLUMNS OF THE MATRIX X.

**LDB**  INTEGER.

LDB IS THE LEADING DIMENSION OF THE ARRAY BETA.

**IP**  INTEGER.

IP IS THE NUMBER OF RIGHT HAND SIDES.

**GRAUX** REAL(K)

GRAUX CONTAINS THE OUTPUT FROM SQRSL2.

**Y** REAL(LDX,IP).

Y IS THE N BY IP RIGHT HAND SIDE MATRIX THAT IS MANIPULATED BY SQRSL2.

**JOB**  INTEGER.

JOB IS A PARAMETER THAT CONTROLS WHAT IS TO BE COMPUTED.

**ON RETURN**

**BETA** REAL(LDB,IP).
C SPECIAL ACTION WHEN N=1

IF (JU .NE. 0) GO TO 20
   IF(X(1,1) .NE. 0.0) GO TO 5
      INFO = 1
   GO TO 220
5 CONTINUE
   DO 10 L = 1, IP
      IF(JOB .NE. 2) BETA(I,L) = Y(I,L)/X(1,1)
      IF(JOB .NE. 1) RSD(1,L) = 0.0E0
10 CONTINUE
   GO TO 220
20 CONTINUE

C C
C COMPUTE TRANS(Q)*Y
C
DO 50 J = 1, JU
   IF (GRAUX(J) .EQ. 0.0E0) GO TO 40
      TEMP = X(J,J)
      X(J,J) = GRAUX(J)
   DO 30 L = 1, IP
      T = -SDOT1(N-J+1,X(J,J),Y(J,L))/X(J,J)
      CALL SAXPY1(N-J+1,T,X(J,J),Y(J,L))
30 CONTINUE
      X(J,J) = TEMP
40 CONTINUE
50 CONTINUE
      KP1 = K + 1
   IF (JOB .EQ. 1 .OR. K .EQ. N) GO TO 70
      DO 60 L = 1, IP
60 CONTINUE
CALL SCOPY(H, K, Y(KPI, L), 1, RSD(KPI, L), 1)

60 CONTINUE

70 CONTINUE

IF (JOB .EQ. 2) GO TO 120

C

C COMPUTE BETA

C

DO 75 L = 1, IP

CALL SCOPY(K, Y(J, L), 1, BETA(J, L), 1)

75 CONTINUE

DO 100 JJ = 1, K

J = K - JJ + 1

IF (X(J, J) .NE. 0.0D0) GO TO 80

INFE = J

C

......EXIT

GO TO 220

80 CONTINUE

DO 95 L = 1, IP

BETA(J, L) = BETA(J, L) / X(J, J)

IF (J .EQ. 1) GO TO 90

T = -BETA(J, L)

CALL SASPY(J-1, T, X(J, J), BETA(J, L))

90 CONTINUE

95 CONTINUE

100 CONTINUE

110 CONTINUE

120 CONTINUE

IF (JOB .EQ. 1) GO TO 210

C

C COMPUTE PSD IF REQUIRED

C

DO 160 L = 1, IP
DO 150 I = I, K
   RS(I,L) = 0.0D0
150    CONTINUE

160 CONTINUE

DO 200 JJ = 1, JU
   J = JU - JJ + 1
   IF (GRAUX(J) .EQ. 0.0D0) GO TO 190
      TEMP = X(J,J)
      X(J,J) = GRAUX(J)
      DO 170 L = 1, IP
         T = -SDOT1(H-J+1,X(J,J),RS(D,J,L)) / X(J,J)
      CALL SAXPY1(H-J+1,T,X(J,J),RS(D,J,L))
170    CONTINUE
      X(J,J) = TEMP
190    CONTINUE
200    CONTINUE

210 CONTINUE

220 CONTINUE
   RETURN
   END
The QR algorithm used on a serial computer and presently executed on the Control Data Corporation 6000 Computer was adapted to execute efficiently on the Control Data STAR-100 computer. This paper describes how the scalar program was adapted for the STAR-100 and indicates why these adaptations yielded an efficient STAR program. Program listings of the old scalar version and the new vectorized SL/I version are presented in the appendices. Execution times for the two versions applied to the same system of linear equations, are compared.