STAR ADAPATION FOR TWO ALGORITHMS USED ON SERIAL COMPUTERS

by Lona M. Howser and Jules J. Lambiotte, Jr.

Langley Research Center
Hampton, Va. 23665
### Abstract

Two representative algorithms used on a serial computer and presently executed on the Control Data Corporation 6000 computer were adapted to execute efficiently on the Control Data STAR-100 computer. Gaussian elimination for the solution of simultaneous linear equations and the Gauss-Legendre quadrature formula for the approximation of an integral are the two algorithms discussed. This paper describes how the programs were adapted for STAR and why these adaptations were necessary to obtain an efficient STAR program. Some points to consider when adapting an algorithm for STAR are discussed. Program listings of the 6000 version coded in 6000 FORTRAN, the adapted STAR version coded in 6000 FORTRAN, and the STAR version coded in STAR FORTRAN are presented in the appendices.
STAR ADAPTATION FOR TWO ALGORITHMS
USED ON SERIAL COMPUTERS

By Lona M. Howser and Jules J. Lambiotte, Jr.
Langley Research Center

SUMMARY

Two representative algorithms used on a serial computer and presently executed on the Control Data Corporation 6000 computer were adapted to execute efficiently on the Control Data STAR-100 computer. Gaussian elimination for the solution of simultaneous linear equations and the Gauss-Legendre quadrature formula for the approximation of an integral are the two algorithms discussed. This paper describes how the programs were adapted for STAR and why these adaptations were necessary to obtain an efficient STAR program. Some points to consider when adapting an algorithm for STAR are discussed. Program listings of the 6000 version coded in 6000 FORTRAN, the adapted STAR version coded in 6000 FORTRAN, and the STAR version coded in STAR FORTRAN are presented in the appendices.

INTRODUCTION

Many algorithms which are presently used on the Control Data Corporation 6000 computer and executed in a serial mode are suitable for the Control Data STAR-100 computer. However, if these algorithms were converted line-by-line to the STAR coding, it is not likely that they would make an efficient STAR program and might actually produce code which would run inefficiently on a vector computer such as STAR. The 6000 code will need to be adapted for STAR to produce an efficient STAR code. This paper discusses two algorithms of this nature. A comparison of the 6000 coding and STAR coding of the identical algorithm is made for two different algorithms: one for the solution of simultaneous linear equations using Gaussian elimination with partial pivoting and the other for numerical evaluation of an integral using the Gauss-Legendre quadrature formula. The paper discusses how the 6000 program was adapted for STAR, the reasons for the adaptations, and some factors which should be considered when adapting a program. FORTRAN codings of the algorithms are presented in the appendices. The STAR codings use the FORTRAN language defined in reference 1.
AIDS FOR ADAPTING AN ALGORITHM

When beginning to adapt an algorithm for STAR, it is not enough to look at just segments in the 6000 coding, but it is necessary to look at the entire algorithm to get the total picture. Some questions to pose are: What is the final result?, What is needed at various steps in the algorithm?, and What is computed independent of other steps and what is repeated?

It will be helpful to review a few definitions and terms which are important to remember when formulating a program for the STAR computer. (For a comprehensive discussion of STAR architecture and hardware instructions, see ref. 2.)

(1) A vector is a set of elements stored in contiguous locations in memory. (Array and vector are used interchangeably in this discussion.)

(2) Vector timing is the time required by the central processing unit to process a vector. It is obtained by the equation \[ T = S + \frac{l}{c}, \]

where

- \( T \) time in clocks (1 clock represents 40 nanoseconds)
- \( S \) startup time, different for each vector instruction or macro
- \( l \) length of vector (number of elements in the vector)
- \( c \) constant depending on the type of instruction and whether the vector elements are each stored in 32- or 64-bit words

(3) A page is a block of storage which contains data or instructions. A program is made up of one or more pages. (See ref. 3.)

(4) A program's working set is the smallest set of pages which must be in central memory for the program to operate efficiently.

(5) A page fault occurs when a program references a page which is not contained in central memory; that is, it is not in the program's present working set.

(6) Paging is the process of bringing a page into central memory or releasing a page.

When adapting the 6000 code to the STAR code, the following factors are important:

(1) Use vector instructions. This requirement may mean reordering steps in an algorithm so that elements in contiguous locations can be operated on or it may mean rearranging storage.

(2) Use long vectors. If a choice is available whether to use many short vectors or a few long vectors, use the long vectors unless the overhead to create the long vectors
is too great. This procedure reduces the effect of the startup time associated with each vector instruction.

(3) Avoid or use sparingly coding which will generate costly vector instructions and macros, that is, costly as compared with some of the faster vector instructions. Refer to the most current timings available. Examples of such instructions and macros are divide, transmit indexed list, and dot product.

(4) Avoid unnecessary paging problems by creating a reasonable working set for the program. When working with a particular array, perform all the operations possible with this array before working with another array. This factor will be more critical with long arrays, but it is a good habit to form and should help reduce page faults.

(5) Investigate the feasibility of creating more answers than are really needed. Because of the high result rate of vector instructions, it may be advantageous to use an approach which generates a larger number of results than are needed in order to avoid scalar computation. This idea, however, should be used cautiously. (See the discussion in ref. 4 on parallel algorithms for tri-diagonal equation solvers.)

ADAPTATION OF AN ALGORITHM FOR THE SOLUTION OF SIMULTANEOUS LINEAR EQUATIONS

A Langley Research Center 6000 library subroutine GELIM (see listing in appendix A) uses Gaussian elimination with partial pivoting to obtain the solution of the set of simultaneous linear equations, $AX = B$, where $A$ is the square matrix of coefficients of order $n$, $X$ is a vector of unknowns of length $n$, and $B$ is a constant matrix of order $n \times r$ where $r$ is the number of right-hand sides. The matrix $A$ is factored into a lower unit triangular matrix and an upper triangular matrix. (For numerical details of the algorithm, see any numerical analysis text, such as ref. 5.) The subroutine also contains an option for the evaluation of the determinant of matrix $A$.

The version of GELIM on the Langley Research Center 6000 library performs Gaussian elimination as generally defined by operating on one row of $A$ at a time. At the $k$th step, column $k$ is searched for the largest element; then row $k$ and the row which contained the largest element are interchanged. The pivot element is then used to obtain zeroes in all positions in its column below the diagonal. This procedure requires multiplying a row of $A$ by the appropriate scalar and subtracting this product from another row of $A$.

Since these row modifications are the usual way the steps in Gaussian elimination are thought of being performed, it would seem normal that for STAR the matrix would be
stored by rows. This method of storage means that elements in one row of a matrix would be stored in contiguous locations so a row of the matrix could be a vector and the steps would be performed by using vector instructions.

In the present version of STAR FORTRAN, two-dimensional arrays are stored columnwise and an optional storage arrangement by rows is not available. This column storage means that elements in one row of matrix \( A \) are not stored in contiguous locations and modifying the matrix a row at a time would mean that few vector instructions could be used. This fact makes it desirable to see whether Gaussian elimination can be performed by modifying matrix \( A \) by using columns so that vector instructions can be used. As will be shown below, Gaussian elimination can be performed by operating on columns of the matrix.

An option to store matrices by rows may be available in a later version of STAR FORTRAN, but when the 6000 code was modified to perform Gaussian elimination by columns, many advantages for the column storage over the row storage appeared; therefore, column storage is recommended. The following section will show how Gaussian elimination can be performed by using vector instructions when the matrix is stored by columns and will identify these advantages. The section entitled "Row Storage" shows the sequence of steps performed in the row storage which is identical to the present 6000 algorithm.

### Column Storage

Gaussian elimination can be performed by modifying one column of the matrix at a time. This is done by a reordering of the operations from the usual row operations. Accomplishing the triangularization of matrix \( A \) by performing the work on columns makes efficient use of STAR and does the identical arithmetic normally done when performing the work on rows.

The \( k \)th step of the triangularization can be performed as shown below, where \( n \) is the number of equations and \( r \) is the number of right-hand sides. All references to the \( k \)th column refer to column entries below the diagonal.

1. Divide the \( k \)th column of \( A \) by the \( a_{kk} \) element and store in the \( k \)th column. This is a vector divided by a scalar:

\[
a_{ik} = \frac{a_{ik}}{a_{kk}} \quad (i = k + 1, \ldots, n)
\]

2. Multiply the \( k \)th column by the \( a_{kj} \) element and subtract this result from the \( j \)th column. This is a vector multiplied by a scalar and then a vector subtract:
\( a_{ij} = a_{ij} - a_{ik}b_{kj} \) \quad (i = k + 1, \ldots, n)

(3) Repeat the sequence of vector instructions in step (2) for the columns of A \((j = k + 1, \ldots, n)\).

(4) Multiply the kth column of A by the \(b_{kj}\) element of B and subtract this result from the jth column of B. This is a vector multiplied by a scalar and then a vector subtract:

\( b_{ij} = b_{ij} - a_{ik}b_{kj} \) \quad (i = k + 1, \ldots, n)

(5) Repeat the vector instructions in step (4) for the remaining right-hand sides of B \((j = 1, 2, \ldots, r)\).

(6) Repeat steps (1) to (5) until the triangularization of matrix A is complete \((k = 1, 2, \ldots, n - 1)\).

Often a subroutine for the solution of simultaneous equations requires the user to append the right-hand sides to the original matrix. The column storage arrangement allows the right-hand sides to be done as separate vector instructions without having to append the right-hand sides to the original matrix. This arrangement is less cumbersome for the user since the right-hand side can be a separate array.

GELIM uses partial pivoting which means that rows will need to be interchanged at times. At the kth step, column k is searched for the largest element and the row containing the largest element and the kth row are interchanged. The column search to find the largest element can easily make use of vector instructions, but the row interchange presents a problem. Elements of rows will need to be interchanged and none of them will be stored in contiguous locations. Not only are they not contiguously stored, but for a very large matrix a column could use one or more pages. For a large matrix it is unlikely that a program will be allowed a working set large enough to contain the entire matrix. This situation would mean that the row elements to be interchanged would be on separate pages and would have to be brought into and then out of core only to reference two elements on the page. Then when the column modifications are performed, the same pages will need to be brought back into core again. This procedure would be extremely inefficient.

A form of indexing could be set up to achieve row interchange, but it would need one of the more time-consuming instructions (transmit indexed list) and the referencing across page boundaries would still be present. No vector instruction can help perform the interchange efficiently; thus, a scalar interchange will be just as efficient.
The possible paging problem can be alleviated by not interchanging an entire row at one time as it is performed on the present 6000 version. At the kth step when the largest element in column k is found, interchange the two elements only in column k. When working with the jth column, both column k and column j are needed. Both of the elements which need interchanging are in the jth column which will be in the program's working set at that time. Therefore, before beginning the operations on the jth column, interchange the two elements. In this way, the page or pages containing that column will need to be brought into core only once per step of the algorithm.

The subroutine also computes the value of the determinant of matrix A which is equal to the product of the elements of the diagonal of the triangular matrix. In the 6000 version, the product is computed after the triangularization is completed. For the STAR, this computation presents a similar situation as the rows interchange; the diagonal elements may be on separate pages and are not stored contiguously. Therefore, vector instructions cannot be utilized. The product will be scalar multiplication, but after the kth step has been completed, the partial product can be formed by using the diagonal element of column k while column k is still in the program's working set. This grouping of row interchange, triangularization of the matrix, and evaluation of the determinant should create an efficient working set for the program with a minimum of paging.

The remaining task of the subroutine is to perform the back substitution. Back substitution generally uses the dot product of a row and the solution vector. This method is used on the 6000 version, but presents problems for STAR column storage since it uses the costly dot product macro and references elements of one array by rows and references elements of the other array by columns. This problem can be eliminated by reordering the steps needed to perform back substitution and all the work can be performed on columns.

The steps to find the kth unknown by back substitution by using vector instructions but not using the dot product are as follows:

1. A scalar divide is always necessary, \( b_{kj} = \frac{b_{kj}}{a_{kk}} \). This step obtains the kth unknown for the jth right-hand side and stores the unknown in the right-hand side vector.

2. Multiply column k of A by the kth unknown obtained in step (1) and subtract this result from the jth column of B. This is a vector multiplied by a scalar and then a vector subtract:

\[
b_{ij} = b_{ij} - a_{ik}b_{kj} \quad (i = 1, 2, \ldots, k - 1)
\]

3. Steps (1) and (2) are repeated for all the right-hand sides (columns of B) \((j = 1, 2, \ldots, r)\).
(4) Steps (1), (2), and (3) are repeated for all the unknowns \( k = n, \ldots, 2, 1 \). When \( k = 1 \), step (2) is omitted.

Row Storage

The way of performing the steps in Gaussian elimination as commonly seen in texts is by operating with rows. The elements in one row would be stored contiguously and the triangularization of matrix \( A \) would make efficient use of vector instructions.

To make the triangularization most efficient, the right-hand sides must be appended to the matrix \( A \) for row storage. The length of the longest vector used would be \( n + r - 1 \) where \( n \) is the number of unknowns and \( r \) is the number of right-hand sides. If a separate array is used for the right-hand side, two identical vector instructions would be needed for each operation, one of length \( n \) and one of length \( r \). This method would be inefficient because the startup times would be multiplied by a factor of 2 and if \( r \) is 1, it would mean using vector instructions of length 1. Appending the right-hand side is a disadvantage in that it is awkward for the user.

Step \( k \) of the triangularization when the matrix is stored by rows is as follows:

1. Perform a scalar divide, \( a_{ik}/a_{kk} \) and store in \( a_{ik} \).
2. Multiply row \( k \) of \( A \) by \( a_{ik} \) and subtract from the \( i \)th row. This is a vector multiplied by a scalar and then a vector subtract:

\[
    a_{ij} = a_{ij} - a_{ik}a_{kj} \quad (j = k + 1, \ldots, n + r)
\]

3. Repeat steps (1) and (2) for all rows \( (i = k + 1, \ldots, n) \).
4. Repeat steps (1), (2), and (3) for all columns until the triangularization is complete \( (k = 1, 2, \ldots, n - 1) \).

The row interchanges necessary for partial pivoting are very easy when the matrix is stored by rows, but the column search will be scalar operations. In addition, there will be no way to avoid possible paging problems for a large matrix during the column search. The elements on each row of the column may be on separate pages, but the search has to be completed before any row operations can begin.

The determinant evaluation could be performed the same way as in the column storage, after a step in the Gaussian elimination is completed. The pages for a large matrix would not have to be brought in only to get one element for the evaluation of the determinant, but the information would be used while the page was still in memory.

When performing the back substitution, if the matrix is stored by rows, there is no way of using the scheme devised to eliminate the need for the dot product macro. The
back substitution references a row of the matrix and a column of the right-hand side. To
be able to use vector instructions, a vector would need to be created which would contain
the elements in the column. This method would be expensive and inefficient, and it is
likely that the vector code would be no better than the scalar code. Either would be
inefficient here, because of the referencing of a row and a column.

Table I summarizes the comparison of the two storage arrangements at the various
steps in the algorithm. The column storage is more advantageous for STAR than the row
storage arrangement.

| TABLE I. - COMPARISON OF ROWWISE AND COLUMNWISE
STORAGE OF MATRIX |

<table>
<thead>
<tr>
<th>Step</th>
<th>Rowwise</th>
<th>Columnwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column search</td>
<td>Scalar: no way to avoid possible paging problems</td>
<td>Vector</td>
</tr>
<tr>
<td>Row interchange</td>
<td>Vector: easy</td>
<td>Scalar: can avoid possible paging problems</td>
</tr>
<tr>
<td>Triangularization</td>
<td>Vector: usual way it is done</td>
<td>Vector: with steps reordered</td>
</tr>
<tr>
<td>Back substitution</td>
<td>Dot product macro: costly, referencing row and column, possibly not vectorizable</td>
<td>Vectors: columns only, no dot product, more efficient</td>
</tr>
<tr>
<td>Determinant evaluation</td>
<td>Scalar</td>
<td>Scalar</td>
</tr>
<tr>
<td>Treatment of right-hand sides</td>
<td>Must be appended to matrix for efficient vector use</td>
<td>Can be separate variable and still use vectors efficiently</td>
</tr>
</tbody>
</table>

Flow Chart

Subroutine GELIM was adapted for use on the STAR computer by using the same
numerical method that was used on the 6000 computer. The matrix is stored by columns.
By reordering computational steps, vector instructions can be used and a reasonable
working set for the program established. Figure 1 shows a flow chart of the 6000 and
STAR versions of the algorithm.
Figure 1.- Flow chart of 6000 and STAR version of segments of Gaussian elimination. An asterisk denotes use of vector instructions.
Figure 1.- Concluded.
Appendix A contains coding of the versions of the algorithm. The 6000 version and the STAR version coded in 6000 FORTRAN and in STAR FORTRAN are included. The calling sequence of the subroutine remains the same and no additional storage was necessary.

Additional Comment

An additional interesting fact was noted when the STAR version of subroutine GELIM was executed on the 6000 computer. The STAR version ran faster than the 6000 version; as a result, there was about a 10-percent decrease in execution time.

ADAPTATION OF A NUMERICAL INTEGRATION ALGORITHM

Subroutine GLEGEN (see listing in appendix B) uses the Gauss-Legendre quadrature formula to evaluate simultaneously an array of integrals

\[ \int_a^b f_k(x) \, dx \quad (k = 1, 2, \ldots, N) \]

where \( N \) is the number of functions. This subroutine is an example of the manner in which vectors rather than single variables can be used on STAR. As in the preceding example, the sequence of instructions is not the same as in the 6000 version. The subroutine subdivides the integration interval \((a, b)\) into \(NQ\) panels and the Gauss-Legendre quadrature formula is applied to each panel by using a 3- or 10-point formula. The resulting integral \( I_k \) is

\[ I_k = \Delta \sum_{j=1}^{NQ} \sum_{i=1}^{NP} r_i f_k(p_i) \quad (k = 1, 2, \ldots, N) \]

where

- \( f_k \) function
- \( NQ \) number of quadratures or panels
- \( NP \) number of points per quadrature
- \( p_i = l_j + \Delta x_i \) (where \( l_j \) is the lower limit of integration for quadrature \( j \))
- \( r_i \) weights for the point formula used
The 6000 version of the subroutine obtains the first point in a quadrature, evaluates the function at that point, multiplies the function by the appropriate weight, and adds the product to a sum. It continues in the same manner until all the points have been evaluated and used in that quadrature; then the process is repeated for the remaining quadratures until the integral has been evaluated over the desired range. The sum is then multiplied by the appropriate delta to obtain the value of the integral.

Looking at this sequence of instructions shows that much of this work is actually computed independently. The STAR version takes advantage of this. All the information required to compute the points is known so all the points for all the quadratures can be computed to form a long array. This array can be passed to the subroutine to evaluate the functions and a savings can be obtained by using vector instructions to evaluate the functions. This routine will now return an array of function values. All the weights are known so an array can be formed which contains the weights for all the quadratures. The remaining task is to multiply the weights by the functions and sum the results. This computation can use the dot product macro; even though it is costly compared with most vector instructions, it is less expensive than a vector multiply followed by scalar adds. This formulation requires the use of the dot product only once. A scalar multiply of the sum by the appropriate delta completes the integral evaluation.

The weights and abscissas for each quadrature are identical; therefore, the length of these arrays in the 6000 version is equal to 10 for the 10-point formula. A version of the subroutine using short vectors could have been formed. The STAR version could have used vector instructions and computed results using one quadrature at a time with the same array of weights and array length as in the 6000 version. Computing the results one quadrature at a time means that the startup time associated with each vector instruction and the startup times used in the function evaluation also would be multiplied by the number of quadratures used. Depending upon the nature of the function, this amount of time could be significant.

Vector transmits could be used to create a long vector of weights which is really a repetition of the short vector. The long vector is

\[ \begin{align*}
q \text{ repetitions} \\
(\{r_1', r_2', \ldots, r_m', r_1, r_2, \ldots, r_m, \ldots, r_1', r_2', \ldots, r_m\})
\end{align*} \]
where \( m \) depends upon the quadrature formula and \( q \) is the number of quadratures. The cost of setting up this vector seems to be minimal when compared with the multiplication of the startup times which would have prevailed in the short-vector version.

A problem of this type (excluding the function evaluation) would probably not produce a significant paging problem. All the arrays are one dimensional and will most likely not be extremely long. However, consideration should still be given to the working set which is created. Currently, there is a tendency to initialize variables at the beginning of the program. The vector of weights could be created at the beginning of the program, but they are not used until after the functions are evaluated; therefore, the vector is not created until just before it is used. Forming the vector of weights and then immediately using them results in less paging; therefore, a better working set is created than would have been created by initializing the vector.

Subroutine GLEGEN was adapted for the STAR computer by using the same numerical method as was used on the 6000. An efficient STAR routine was obtained by using long vectors and reordering the sequence of instructions. Figure 2 shows a flow chart of the 6000 and STAR versions.

Appendix B contains the coding of the two versions of the algorithm. The 6000 version and the STAR version in 6000 FORTRAN and STAR FORTRAN are included. Notice that the calling sequence of the STAR version reflects the use of long arrays in that more working space is needed.

CONCLUDING REMARKS

Two algorithms used on the Control Data Corporation 6000 computer were adapted for use on the STAR computer. This adaptation required a rethinking of the flow of the entire problem. Array variables are used where the 6000 code used a single-value variable and the steps in the algorithm are not computed in the same order for the two versions. This reordering of steps and changes in variable assignments allow vector instructions to be used and a reasonable working set can be established for the program; thereby efficient use of the STAR computer and some implied improvements for the Control Data Corporation 6000 computer are made.

Langley Research Center,
National Aeronautics and Space Administration,
Hampton, Va., February 20, 1974.
Figure 2. - Flow chart of 6000 and STAR versions of Gauss-Legendre quadrature formula for numerical integration. An asterisk denotes use of vector instructions.
FORTRAN CODING OF 6000 AND STAR VERSIONS OF ALGORITHM FOR THE
SOLUTION OF A SET OF SIMULTANEOUS EQUATIONS

6000 Version in 6000 FORTRAN

SUBROUTINE GELIM(AN,RNKHS,MAXN,PIVOT,IP,JDETERM,ISCALE)
DIMENSION A(MAXN,MAXN),R(MAXN,NRHS),PIVOT(MAXN)

A(MAXN,MAXN) = SQUARE MATRIX OF COEFFICIENTS (A IS DESTROYED)
N = NUMBER OF ROWS AND COLUMNS IN A
B(MAXN,NRHS) = MATRIX OF CONSTANTS (REPLACED BY SOLUTION MATRIX)
MAXN = NUMBER OF COLUMNS IN B
MAXN = MAXIMUM NUMBER OF ROWS AND COLUMNS IN A
IOP = MAXIMUM NUMBER OF ROW INTERCHANGES
IP - IOP=1, EVALUATE DETERMINANT, IOP=0, SKIP DETERMINANT EVALUATION
DETERM = GIVES VALUE OF DETERMINANT DETERM=(10**100)**ISCALE*DETERM
ISCALE = SCALE FACTOR COMPUTED BY SUBROUTINE IN DETERM EVALUATION
TO KEEP DETERM WITHIN THE FLOATING POINT WORD SIZE OF THE

** Compitew

SIGN=1.0
DO 35 1=1,N
35 IPIVOT(1)=I
ISCALE=0
R1=10.0**100
R2=1.0/R1
10 I = DETERM=1.0
NM=1=N-1
IF(NM.E.0)
100**G rider to 1000
*** FIND LARGEST REMAINING TERM IN I-TH COLUMN FOR PIVOT
DO 200 I=1,NM1
BIG=0.
DO 50 K=1,N
TERM=ABS(A(K,I))
IF(TERM>BIIG)50,50,30
BIG=TERM
L=K
50 CONTINUE
IPIVOT(1)=L
IF(BIG.100.0,60,80)
30 BIG=TERM
CONTINUE
IPIVOT(1)=L
IF(BIG.100.0,60,80)
50 CONTINUE
60 CONTINUE
RETURN
80 IF(I-I1)90,120,90
*** SWAP ROWS OF A AND B, SET IPIVOT(J)=K.
90 SIGN=-SIGN
DO 100 J=1,N
TEMP=A(I,J)
A(I,J)=A(I,I)
A(I,I)=TEMP
100 A(I,J)=TEMP
DO 101 J=1,NRHS
TEMP2=A(I,J)
A(I,J)=R(I,J)
R(I,J)=TEMP2
101 A(I,J)=TEMP2
CONTINUE
*** STORE PIVOT IN A(I,J) MULTPLY A AND B BY PIVOT.
IPI=I+1
DO 120 1=1,N
R(I,I)=R(I,I)*IPI
A(I,J)=A(I,J)*IPI
120 CONTINUE

END
APPENDIX A – Continued

32  \[ A(II,K) = A(II,K) - X3 \times A(I,K) \]
    DO 33 K = 1, NRHS
33  \[ B(II,K) = B(II,K) - X3 \times R(I,K) \]
31  CONTINUE
200 CONTINUE
    IF (TOP.EQ.15C) GO TO 221

C
***** SCALE THE DETERMINANT
C
    DO 122 I = 1, N
       PIVOTI = A(I,I)
    1035 IF (ABS(DETERM) .LT. 1.035) GO TO 1040
    1040 DETERM = DETERM / R1
       ISCALE = ISCALE + 1
1050 DETERM = DETERM / R1
       ISCALE = ISCALE + 1
1060 DETERM = DETERM / R1
       ISCALE = ISCALE + 1
1070 PIVOTI = PIVOTI / R1
       ISCALE = ISCALE + 1
1080 PIVOTI = PIVOTI / R1
       ISCALE = ISCALE + 1
GOTO 320
1090 IF (ABS(DETERM) .LT. 1.1090) GO TO 1100
1100 PIVOTI = PIVOTI / R1
       ISCALE = ISCALE + 1
1110 PIVOTI = PIVOTI / R1
       ISCALE = ISCALE + 1
1120 PIVOTI = PIVOTI / R1
       ISCALE = ISCALE + 1
1130 DETERM = DETERM * PIVOTI
122 CONTINUE
C
***** PERFORM BACK SUBSTITUTION
C
221 CONTINUE
    DO 57 IC = 1, NRHS
       B(IN, IC) = B(IN, IC) / A(N, N)
57 CONTINUE
    DO 61 KK = 1, NM1
       I = N - KK
       II = I + 1
    DO 61 J = 1, NRHS
       SUM = B(I, J)
62 SUM = SUM - A(I, K) * B(K, J)
    DO 62 K = II, N
   61 B(I, J) = SUM / A(I, I)
    DETERM = DETERM * SIGN
    RETURN
1000 IF (A(I, I) .EQ. 0) GO TO 60
    DO 1500 J = 1, NRHS
1500 B(I, J) = B(I, J) / A(I, I)
    DETERM = A(I, I)
    RETURN
END
SUBROUTINE GELIM(AN,B,NRHS,MAXN,IPIVOT,IOP,DETERM,ISCALE)
DIMENSION A(MAXN,MAXN),B(MAXN,NRHS),IPIVOT(MAXN)
A(MAXN,MAXN) = SQUARE MATRIX OF COEFFICIENTS (A IS DESTROYED)
N = NUMBER OF ROWS AND COLUMNS IN A
B(MAXN,NRHS) = MATRIX OF CONSTANTS (REPLACED BY SOLUTION MATRIX)
NRHS = NUMBER OF COLUMNS IN B
MAXN = MAXIMUM NUMBER OF ROWS AND COLUMNS IN A
IPIVOT(MAXN) = RECORD OF ROW INTERCHANGES
IOP = IOP=1, EVALUATE DETERMINANT. IOP=0, SKIP DETERMINANT EVALUATION
DETERM = GIVES VALUE OF DETERMINANT DETERM=(10**100)**ISCALE*DETERM
ISCALE = SCALE FACTOR COMPUTED BY SUBROUTINE IN DETERM EVALUATION
TO KEEP DETERM WITHIN THE FLOATING POINT WORD SIZE OF THE COMPUTER
ID=0
SIGN=1.0
DO 3 I=1,N
3 IPIVOT(I)=I
5 ISCALE=0
6 R1=10.0**100
7 R2=1.0/R1
10 DETERM=1.0
NMI=N-1
IF (NMI.EQ.0) GO TO 600
NPI=N+1
:*****FIND LARGEST REMAINING TERM IN I-TH COLUMN FOR PIVOT
DO 200 I=1,NMI
200 BIG=0.
DO 50 K=I,N
TERM=ABS(A(K,I))
IF(TERM>BIG)50,50,30
30 RIG=TERM
L=K
50 CONTINUE
IPIVOT(I)=L
IF(BIG.EQ.0) GO To 600
IPN=I
:*****INTERCHANGE ELEMENTS IN COLUMN I ONLY
90 SIGN=-SIGN
100 A(I,I)=AL(I,I)
A(L,I)=A(I,L)
120 CONTINUE
:*****STORE PIVOT IN A(I,I). MULTIPLY A AND B BY PIVOT.
IP1=I+1
:*****OBTAIN COLUMN OF PIVOTS
DO 128 K=IP1,N
128 A(K,I)=A(K,I)/A(I,I)
:*****INTERCHANGE ELEMENTS ONLY IN COLUMN J
DO 132 J=1,NRHS
TEMP1= B(J,J)
B(J,J)=B(1,J)
B(1,J)=TEMP1
DO 130 K=1,K-1,N
130 B(K,J)=B(K,J)-A(K,J)*B(I,J)
132 CONTINUE
:*****INTERCHANGE ELEMENTS ONLY IN COLUMN J
DO 140 J=IP1,N
TEMP1= A(I,J)
A(I,J)=A(L,J)
A(L,J)=TEMP1
DO 135 K=IP1,N
135 K=IP1,N

APPENDIX A – Continued
APPENDIX A – Continued

135  A(K,J)=A(K,J)-A(K,I)*A(I,J)  
140  CONTINUE
   IF (I.0.EQ.0) GO TO 200

C  SCALE THE DETERMINANT
   PIVOTI=A(I,I)
1005 IF(ABS(DETERM)-R1)1030,1010,1010
1010  DETERM=DETERM/R1
   ISCALE=ISCALE+1
1020  DETERM=DETERM/R1
   ISCALE=ISCALE+1
   GO TO 1060
1030 IF(ABS(DETERM)-R2)1040,1040,1060
1040  DETERM=DETERM/R1
   ISCALE=ISCALE-1
1050 IF(ABS(DETERM)-R2)1050,1050,1060
1060  DETERM=DETERM/R1
   ISCALE=ISCALE-1
   GO TO 320
1090 IF(ABS(PIVOTI)-R2)1090,1070,1070
1070  PIVOTI=PIVOTI/R1
   ISCALE=ISCALE+1
1080  PIVOTI=PIVOTI/R1
   ISCALE=ISCALE+1
   GO TO 200
200  CONTINUE

C  OBTAIN PARTIAL PRODUCT FOR DETERMINANT
320  DETERM=DETERM*PIVOTI
   IF (I.NE.NMI) GO TO 200
   IF (I.EQ.11) GO TO 200
   PIVOTI=A(N,N)
   ID=1
   GO TO 1005
200  CONTINUE

C  PERFORM BACK SUBSTITUTION
340  CONTINUE
   DO 450 KK=1,N
   K=NPI-KK
   SCALAR DIVIDE
   DO 430 J=1,NRHS
   B(K,J)=B(K,J)/A(K,K)
   IF (K.EQ.1) GO TO 430
   KM1=K-1
   COLUMN K BY UNKNOWN AND SUBTRACT
   DO 420 I=1,KM1
420  B(I,J) = B(I,J) - A(I,K)*B(K,J)
430  CONTINUE
450  DETERM=DETERM*SIGN
   RETURN
600  IF (A(1,1).EQ.0.) GO TO 60
   DO 800 J=1,NRHS
800  B(1,J)=B(1,J)/A(1,1)
   DETERM=A(1,1)
   RETURN
END
APPENDIX A – Continued

STAR Version in STAR FORTRAN

SUBROUTINE GELIM(ANBNRHS,MAXN,IPIVOT,IOP,DETERM,ISCALE)
DIMENSION A(MAXN,MAXN),B(MAXN,NRHS),IPIVOT(MAXN)

N = NUMBER OF ROWS AND COLUMNS IN A
B(MAXN,NRHS) = MATRIX OF CONSTANTS (REPLACED BY SOLUTION MATRIX)
NRHS = NUMBER OF COLUMNS IN B
MAXN = MAXIMUM NUMBER OF ROWS AND COLUMNS IN A
IPIVOT(MAXN) = RECORD OF ROW INTERCHANGES
IOP = 1, EVALUATE DETERMINANT. IOP = 0, SKIP DETERMINANT EVALUATION
DETERM = GIVES VALUE OF DETERMINANT (DETERM = (1O**100)**ISCALE*ETERM)
ISCALE = SCALE FACTOR COMPUTED BY SUBROUTINE IN DETERM EVALUATION
TO KEEP DETERM WITHIN THE FLOATING POINT WORD SIZE OF THE COMPUTER

ID=0
SIGN=1.0
DO 3 I=1,N
3 IPIVOT(I)=I
5 ISCALE=0
6 R1=1.0D100
7 R2=1.0/R1
10 DETERM=1.0
NM1=N-1
IF (NM1.EQ.0) GO TO 600
NPI=N+1
******FIND LARGEST REMAINING TERM IN I-TH COLUMN FOR PIVOT
DO 200 I=INM1
BIG=0.
00 50 K=I,N
TERM=ABS(A(K,I))
IF (TERM-BIG)50,50,30
30 BIG=TERM
L=K
200 CONTINUE
IPIVOT(I)=L
IF (BIG)80,60,80
60 DETERM=0.0
RETURN
80 IF (I-L)90,120,90
SIGN=-SIGN
******INTERCHANGE ELEMENTS ONLY IN COLUMN I
TEMP1= A(I,I)
A(I,I)=A(L,I)
A(L,I)=TEMP1
120 CONTINUE
******STORE PIVOT IN A(I,J). MULTIPLY A AND B BY PIVOT.
IPI=I+1
******OBTAIN COLUMN OF PIVOTS
A(IPI:N,J)= A(IPI:N,J)/A(I,J)
******INTERCHANGE ELEMENTS ONLY IN COLUMN J
DO131 J=1,NRHS
TEMPI= B(I,J)
B(I,J)=B(L,J)
B(L,J)=TEMPI
******INTERCHANGE ELEMENTS ONLY IN COLUMN J
DO134 J=1P1,N
TEMPI= A(I,J)
A(I,J)=A(L,J)
A(L,J)=TEMPI
134 A(I+1,N,J) = A(I+1,N,J) - A(I+1,N,1) * A(I,J)

IF (IOP.EQ.0) GO TO 200

***** SCALE THE DETERMINANT

PIVOTI = A(I,I)

1005 IF (IABS(DETERM) .LT. R11030, 1010, 1010
1010 DETERM = DETERM / R1
ISCALE = ISCALE + 1
IF (IABS(DETERM) .LT. R11060, 1020, 1020
1020 DETERM = DETERM / R1
ISCALE = ISCALE + 1
GO TO 1060

1030 IF (IABS(DETERM) .LT. R211040, 1040, 1040
1040 DETERM = DETERM / R1
ISCALE = ISCALE - 1
IF (IABS(DETERM) .LT. R211050, 1050, 1050
1050 DETERM = DETERM / R1
ISCALE = ISCALE - 1
GO TO 320

1090 IF (IABS(DETERM) .LT. R212000, 2000, 2000
2000 DETERM = DETERM / R1
ISCALE = ISCALE - 1
IF (IABS(DETERM) .LT. R212010, 2010, 2010
2010 DETERM = DETERM / R1
ISCALE = ISCALE - 1
GO TO 320

320 DETERM = DETERM ** PIVOTI

IF (I .NE. NMII) GO TO 200
IF (ID .EQ. 1) GO TO 200
PIVOTI = A(N,N)
ID = 1
GO TO 1005

200 CONTINUE

***** PERFORM BACK SUBSTITUTION

340 CONTINUE
DO 450 KK = 1, N
K = NP1 - KK

**** SCALAR DIVIDE
DO 430 J = 1, NRHS
B(K, J) = B(K, J) / A(K, K)
IF (K .EQ. 1) GO TO 430
KM1 = K - 1

**** COLUMN K BY UNKNOWN AND SUBTRACT
B(1:KM1, J) = B(1:KM1, J) - A(1:KM1, K) * B(K, J)

430 CONTINUE

450 CONTINUE
DETERM = DETERM * SIGN
RETURN

600 IF (AIL, 1) .EQ. 0) GO TO 60
B(1, 1:NRHS) = B(1, 1:NRHS) / A(1, 1)
DETERM = A(1, 1)
RETURN
END
APPENDIX B

FORTRAN CODING OF 6000 AND STAR VERSIONS OF ALGORITHM
FOR NUMERICAL INTEGRATION EVALUATION

6000 Version in 6000 FORTRAN

SUBROUTINE GLEGEN(SUM,ICODE,A,B,FX,FOFX,NFX,NQ,NPQ)

** PURPOSE: **
TO COMPUTE THE INTEGRALS, F(I) OF X * DX FROM A TO B USING THE GAUSS-LEGENDRE QUADRATURE FORMULA.

** USE:**
CALL GLEGEN (SUM,ICODE,A,B,FX,FOFX,NFX,NQ,NPQ)

** PARAMETERS: **
SUM: THE ARRAY FOR THE VALUE(S) OF THE INTEGRAL(S)
ICODE: AN INTEGER TEST CODE RETURNED BY THE ROUTINE
0: NORMAL RETURN
1: NFX NOT PROPERLY SPECIFIED
2: NQ NOT PROPERLY SPECIFIED (NOT 3 OR 10)

A: LOWER LIMIT OF INTEGRATION
B: UPPER LIMIT OF INTEGRATION
FX: THE NAME OF A USER SUPPLIED SUBROUTINE WITH ARGUMENTS X AND FOFX TO EVALUATE THE FUNCTIONS
IT MUST BE DECLARED AS EXTERNAL.
FOFX: THE ARRAY TO STORE THE VALUES OF THE FUNCTIONS
NFX: AN INTEGER, THE NO. OF FUNCTIONS TO INTEGRATE
NQ: AN INTEGRAL, THE NUMBER OF QUADRATURES
NPQ: AN INTEGER, THE NO. OF POINTS PER QUADRATURE

** REQUIRED ROUTINES:** NONE

** SOURCE/IMPLEMENTER:** COMPUTER SCIENCES CORP./G. W. HAIGLER

** DATE RELEASED:** NOV. 14, 1972

** LATEST REVISION:** NOV. 15, 1972
APPENDIX B – Continued

DIMENSION UI(3), U2(10), RI(3), R2(10), SUM(I), FOFX(1)
EQUIVALENCE (UI(1),U(1),(U2(1),U(4)),(RI(1),R(1)),(R2(1),R(1)))
DATA UI/.011270166537925,.5,.887298334620742/, U2/.013046735741414, ...
L.061468316655507,.160295215850488..28330320935376,.42556283050918,
25..574437169490816,.716697967064624,.839704784149512,.932531683344,
5..13463359564998,.147762112357377,.147762112357377,.13463359564998,
68..1095431811257991,.07425674575291,.033335672154344/
ICODE = 0
IF(NFX.LE.0)ICODE = 1
IF(NQ.LE.0)ICODE = 3
IF(NPQ.EQ.3)GO TO 5
IF(NPQ.NE.10)ICODE = 2
5 IF(ICODE.GT.0) RETURN
J=3
IF(NPQ.EQ.3) J=0
DELTA = (B-A)/FLOATINO)
DO 10 I = 1,NFX
10 SUM(I) = 0.0
DO 80 JB=1,NFX
80 SUM(JB) = FOFX(JB)*FACT+SUM(JB)
DO 100 II=1,NFX
100 SUM(II) = SUM(II)*DELTA
RETURN
END

STAR Version in 6000 FORTRAN

SUBROUTINE GLEGENS(SUM,ICODE,A,B,FX,FOFX,NFX,NQ,NPQ,UU,FF,UD,MAXI)

P  PURPOSE: TO COMPUTE THE INTEGRALS, F(I) OF X * DX FROM A TO B USING THE GAUSS-LEGENDRE QUADRATURE

P  USE: CALL GLEGEN (SUM,ICODE,A,B,FX,FOFX,NFX,NQ,NPQ,UU,FF,UD,MAXI)

P  PARAMETERS:
SUM THE ARRAY FOR THE VALUE(S) OF THE INTEGRAL(S)
ICODE AN INTEGER TEST CODE RETURNED BY THE ROUTINE
  = 0, NORMAL RETURN
  = 1, NFX NOT PROPERLY SPECIFIED
  = 2, NQ NOT PROPERLY SPECIFIED (NOT 3 OR 10)
  = 3, NQ NOT PROPERLY SPECIFIED
THIS PARAMATER SHOULD BE TESTED UPON RETURN BY THE CALLING PROGRAM

GLEN001
APPENDIX B – Continued

A LOWER LIMIT OF INTEGRATION

B UPPER LIMIT OF INTEGRATION

FX THE NAME OF A USER SUPPLIED SUBROUTINE WITH ARGUMENTS X AND FOFX TO EVALUATE THE FUNCTIONS. IT MUST BE DECLARED AS EXTERNAL.

FOFX A TWO DIMENSIONED ARRAY TO STORE VALUES OF THE FUNCTIONS, FOFX(MAXI,NFX) IN CALLING PROGRAM.

NFX AN INTEGER, THE NO. OF FUNCTIONS TO INTEGRATE

NO AN INTEGER, THE NUMBER OF QUADRATURES

NPQ AN INTEGER, THE NO. OF POINTS PER QUADRATURE. IT MUST BE 3 OR 10

UU ARRAY TO STORE ALL POINTS AT WHICH FUNCTION IS EVALUATED, ALSO STORES WEIGHTS

FF ARRAY OF WORKING SPACE, CONTAINS LOWER LIMIT OF INTEGRATION FOR EACH QUADRATURE

MAXI MAXIMUM ROW DIMENSIONS OF FOFX IN CALLING PROGRAM

NQ Required routines: NONE

* SOURCE/IMPLEMENTER: COMPUTER SCIENCES CORP./G. W. HAIGLER

* DATE RELEASED: NOV. 14, 1972

* LATEST REVISION: NOV. 15, 1972

*****+********* *******4 ***************************************

DIMENSION U1(3),U2(LO),RI(3),R2(101U3),R(13),SUM(I),

1 FOFX(MAXI,1 ),UU(1 ),UD(1 ),FF(1 )

EQUIVALENCE (U1(1),U(11I,(U2(11,U(4)),(RI(1),R(1IIdRZ(1I,R(4))

DATA U1/112701665379259..5..887298334626742/2/U2/.013046735741414,GLE65057
1.06766816655507,.160295215850488,.283302302935376,.42556283050918,GLE65058
5..57437169490816,.71697697064624,.83970478449512,.93253168334415,GLE65059
3493..986953264258586/2/1/.277777777777777,.444444444444444,.277777777777777,GLE65060
477777777777777,.R2/.03335672154344,,.074725674575291,.109543181257991,GLE65061
5.13463359654998,.147762112357377,.147762112357377,.13463359654998,GLE65062
68,.109543181257991,.074725674575291,.03335672154344/ GLE65063

ICODE = 0

NT = TOTAL NUMBER OF POINTS AT WHICH FUNCTION WILL BE EVALUATED

IF(NPQ NE.0)ICODE = 1

IF(INPQ LE.0)ICODE = 3

IF(INPQ NE.0)GO TO 5

5 IF(ICODE LT.0)ICODE = 2

J=3

IF(INPQ .EQ. 3) J=0

DEL! = (R-1)/FLOATING

DO 20 K=1,NPQ

J1 = J*K

20 UD(K) = U(J1)*DEL! 

***COMPUTE ARRAY TO BE ADDED TO FIRST POINT IN EACH QUADRATURE

20 UD(K) = U(J1)*DEL!
APPENDIX B - Continued

****COMPUTE FIRST POINT IN EACH QUADRATURE
DO 30 K=1,NQ
   XI =K-1
30   FF(K) =A + XI*DELT

C CREATE VECTOR OF ALL POINTS FOR ALL QUADRATURES
DO 40 K=1,NQ
   K1 =K-1*NPQ
   DO 35 I=1,NPQ
      UU(K1+I) = FF(K) + UD(I)
35 CONTINUE
40 CONTINUE

CREATE VECTOR OF ALL POINTS FOR ALL QUADRATURES
DO 50 K=1,NQ
   K1 =K-1*NPQ
   DO 45 I=1,NPQ
      UU(K1+I) = R(J,I)
45 CONTINUE
50 CONTINUE

SUM OF UU(K1)*FOFX(K1)
SUM(II) = DELT * SUM(II)

EVALUATE FUNCTION AT ALL POINTS
NOTE CHANGE IN CALLING SEQUENCE FOR FUNCTION EVALUATION SUBROUTINE
CALL FXS(UU,FOFX,MAXI,NT)

CREATE VECTOR OF WEIGHTS
DO 60 K=1,NQ
   K1 =K-1)*NPQ
   DO 55 I=1,NPQ
      UU(K1+I) = R(J,I)
55 CONTINUE
60 CONTINUE

SUM OF UU(K1)*FOFX(K1)
SUM(II) = DELT * SUM(II)

RETURN
END

STAR Version in STAR FORTRAN

SUBROUTINE GLEGEN(SUM,ICODE,A,B,FX,FOFX,NFX,NQ,NPQ,UD,MAXI)

C**********************************************************************
C PURPOSE: TO COMPUTE THE INTEGRALS, F(I) OF X * DX FROM A TO B USING
C THE GAUSS-LEGENDRE QUADRATURE FORMULA.
C USE: CALL GLEGEN (SUM,ICODE,A,B,FX,FOFX,NFX,NQ,NPQ,UD,MAXI)
C PARAMETERS: SUM THE ARRAY FOR THE VALUE(S) OF THE INTEGRAL(S)
ICODE AN INTEGER TEST CODE RETURNED BY THE ROUTINE
 = 0, NORMAL RETURN
 = 1, NFX NOT PROPERLY SPECIFIED
 = 2, NPQ NOT PROPERLY SPECIFIED (NOT 3 OR 10)
 = 3, NO NOT PROPERLY SPECIFIED
THIS PARAMETER SHOULD BE TESTED UPON RETURN BY THE CALLING PROGRAM
A LOWER LIMIT OF INTEGRATION
B UPPER LIMIT OF INTEGRATION
FX THE NAME OF A USER SUPPLIED SUBROUTINE WITH
ARGUMENTS X AND FOFX TO EVALUATE THE FUNCTIONS. IT MUST BE DECLARED AS EXTERNAL.
FOFX A TWO DIMENSIONED ARRAY TO STORE VALUES OF THE FUNCTIONS. FOFX(MAXI,NFX) IN CALLING PROGRAM

GLES084
GLES085
GLES086
GLES087
GLES088
GLES089
GLES090
GLES091
GLES092
GLES093
GLES094
GLES095
GLES096
GLES097
GLES098
GLES099
GLES100
GLES101
GLES102
GLES103
GLES104
GLES105
GLES106
GLES107
GLES108
GLES109
GLES001
GLES002
GLES003
GLES004
GLES005
GLES006
GLES007
GLES008
GLES009
GLES010
GLES011
GLES012
GLES013
GLES014
GLES015
GLES016
GLES017
GLES018
GLES019
GLES020
GLES021
GLES022
GLES023
GLES024
GLES025
GLES026
GLES027
GLES028
GLES029
GLES030
GLES031

24
APPENDIX B – Continued

* NFX AN INTEGER, THE NO. OF FUNCTIONS TO INTEGRATE
* NQ AN INTEGER, THE NUMBER OF QUADRATURES
* NPQ AN INTEGER, THE NO. OF POINTS PER QUADRATURE
  IT MUST BE 3 OR 10
* UU ARRAY TO STORE ALL POINTS AT WHICH FUNCTION IS
  EVALUATED, ALSO STORES WEIGHTS
* FF ARRAY OF WORKING SPACE, CONTAINS LOWER LIMIT OF
  INTEGRATION FOR EACH QUADRATURE
* UD ARRAY OF WORKING SPACE, ABSCISSAS* Delt
* MAXI MAXIMUM ROW DIMENSIONS OF FOFX IN CALLING PROG.
* REQUIRED ROUTINES: NONE
* SOURCE/IMPLEMENTER: COMPUTER SCIENCES CORP./ G. W. HAIGLER
* DATE RELEASED: NOV. 14, 1972
* LATEST REVISION: NOV. 15, 1972

DIMENSION U1(3), U2(10), R1(3), R2(10), U(13), R(13), SUM(1),
1 FOFX(MAXI,1), UU(MAXI,1), UD(1,1), FF(1,1)

EQUIVALENCE (U1(1), U1(1)), (U2(1), U2(1)), (R1(1), R1(1)), (R2(1), R1(1))
DATA U1/1.112701665379259, .5, .887298334620742/, U2/1.06746316655507, .160295215850488, .283302302935376, .42556283050918, .574437169490816, .71697697064624, .839704784149512, .93253168334493, .986951264258586/, R1/1.277777777777777, .444444444444444, .277777777777777, .074725674575291, .033335672154344/,
1 R2/1.03335672154344, .074725674575291, .033335672154344, .109543181257991, .134633359654998, .147762112357377, .147762112357377, .134633359654998/,
11 SUM/1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0/,
12 FOFX/1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0/,
13 UD/1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0/,
14 FF/1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0/,
15 ICODE = 0
C NT = TOTAL NUMBER OF POINTS AT WHICH FUNCTION WILL BE EVALUATED
NT=NQ*NPQ
1 IF(NFX.LE.0) ICODE = 1
2 IF(NQ.LE.0) ICODE = 3
3 IF(NPQ.EQ.3) GO TO 5
4 IF(NPQ.NE.10) ICODE = 2
5 IF(ICODE.GT.0) RETURN
J=3
IF (NPQ .EQ. 3) J=0
DELT= (B-A)/FLOAT(NQ)
J1 = J+1
NJ= J + NPQ
C****COMPUTE ARRAY TO BE ADDED TO FIRST POINT IN EACH QUADRATURE
UD1(NPQ) = U(J1:NJ) * DELT
C****COMPUTE FIRST POINT IN EACH QUADRATURE
DO 30 K=1,NQ
X1 = X-1
30 FF(K) = X1 + X1*DELT
C****COMPUTE ARRAY OF ALL POINTS FOR ALL QUADRATURES
DO 40 K=1,NQ
X1 = (K-1)*NPQ
40 CONTINUE
C EVALUATE FUNCTION AT ALL POINTS
APPENDIX B — Concluded

C NOTE CHANGE IN CALLING SEQUENCE FOR FUNCTION EVALUATION SUBROUTINE
  CALL FXS(UU,FOFX,MAXI,NT)
C CREATE VECTOR OF WEIGHTS
DO 50 K=1,NQ
  K1 = (K-1)*NPQ
  NK = K + NPQ
  UU(K1+1:NK) = R(J1:NJ)
50 CONTINUE
C THESE STATEMENTS CAN BE REPLACED BY DOT PRODUCT FUNCTION IF AVAILABLE
C SUCH AS
  SUM(I) = DOTP (UU(I:NT),FOFX(1:NT, I))
C **
  DO 80 I=1,NFX
      SUM(I) = 0.0
  DO 70 K=1,NT
    70 SUM(I) = SUM(I) + UU(K)*FOFX(I,K)
  SUM(I) = DELT * SUM(I)
80 CONTINUE
C **
RETURN
END

C
REFERENCES


