NOTICE

THIS DOCUMENT HAS BEEN REPRODUCED FROM MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE
A Computer Program for Determining Truncation Error Coefficients for Runge-Kutta Methods

Mission Planning and Analysis Division

July 1980

NASA
National Aeronautics and Space Administration

Lyndon B. Johnson Space Center
Houston, Texas
SHUTTLE PROGRAM

A COMPUTER PROGRAM FOR DETERMINING TRUNCATION ERROR COEFFICIENTS FOR RUNGE-KUTTA METHODS

By M. Kathleen Horn*

JSC Task Monitor: Victor Bond
Software Development Branch

Approved: Elric N. McHenry
Elric N. McHenry, Chief
Software Development Branch

Approved: Ronalf L. Berry, Chief
Mission Planning and Analysis Division

Mission Planning and Analysis Division
National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas
July 1980
*National Research Council Research Associate
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2.0 THE GENERATION OF TRUNCATION ERROR COEFFICIENTS</td>
<td>3</td>
</tr>
<tr>
<td>3.0 DESCRIPTION OF THE FORTRAN PROGRAM</td>
<td>6</td>
</tr>
<tr>
<td>4.0 CONCLUDING REMARKS</td>
<td>8</td>
</tr>
<tr>
<td>5.0 REFERENCES</td>
<td>9</td>
</tr>
<tr>
<td>APPENDIX A - SUBROUTINE RKEQ</td>
<td>A-1</td>
</tr>
<tr>
<td>Table</td>
<td>DIMENSIONING PARAMETER, LS, FOR ORDERS 1 THROUGH 12</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

The solution of the initial value problem for ordinary differential equations (ODE's)

\[ \frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0 \]  \hspace{1cm} (1)

may be treated by several different numerical methods. Runge-Kutta (RK) algorithms are a type of method well suited to solving equation (1) for many classes of functions, $f$, because of their simplicity and their accuracy. The RK algorithm is derived using a direct comparison with a truncated Taylor series, giving the accuracy of the Taylor series without the difficulty of determining complicated partial derivatives. The comparison between the Taylor series expansion of the solution vector and the solution determined by the RK algorithm results in a number of expressions referred to as truncation error coefficients, $T_{i,j}$. Associated with each term of order $i$ in the Taylor series (or with each power of $h$, the integration stepsize) are $\lambda_i$ truncation error coefficients. For an RK algorithm to be of order $p$, the $T_{i,j}$ coefficients must be identically zero for $i = 1, \ldots, p$; $j = 1, \ldots, \lambda_i$. These vanishing truncation error coefficients are referred to as equations of condition. The nonvanishing error coefficients, however, are of equally great importance since they indicate how closely the RK solution approximates a Taylor series solution of higher order. The equations of condition determine the validity of an RK algorithm; the nonvanishing error coefficients explain the differences between particular RK algorithms of the same order. While a user may apply an RK algorithm, never considering the truncation error coefficients, awareness of the effect of these terms is important both in the selection of a specific algorithm and in the analysis of difficulties encountered during the solution of a particular ODE.

D. G. Bettis,a has developed an algorithm for generating the truncation error coefficients for RK methods designed to treat systems of both first- and second-order ODE's directly. The recursive nature of this algorithm lends itself readily to computer programming, generating high order error coefficients with little added difficulty. Such an algorithm, implemented in a numerical code, is an essential tool for anyone developing coefficients for RK algorithms and is of interest to the user of RK methods in analyzing the effectiveness of specific RK algorithms. A Fortran subroutine, RKEQN, written to accompany reference 1, generated the truncation error coefficients through order 10 but required a great amount of storage location, particularly when a double precision version of the program was needed. The basic structure of this original program has been reformulated to reduce storage requirements significantly and to accommodate variable dimensioning. This new Fortran program, SUBROUTINE RKEQ, determines truncation error coefficients for RK algorithms in the sequence presented in reference 1 for orders 1 through 10 and extends the order of coefficients

---

aFrom a private communication with D. G. Bettis, 1978.
through 12 with the 11th- and 12th-order terms determined following the patterns used to establish the lower order coefficients. Both subroutines (RKEQN and RKEQ) are also written to treat RK m-fold methods (refs. 2 and 3) which utilize m known derivatives of f to increase the order of the algorithm. Setting m = 0 gives the classical RK algorithm.
2.0 THE GENERATION OF TRUNCATION ERROR COEFFICIENTS

The solution of equation (1) at \( t_1 = t_0 + h \), using the RK algorithm, is written

\[
y_1 = y_0 + \sum_{k=0}^{p} C_k f_k
\]

where

\[
f_o = f(t_0, y_0)
\]

\[
f_k = f(t_0 + \alpha_k h, y_0 + h \sum_{\lambda=0}^{k-1} \beta_{k,\lambda} f_{\lambda})
\]

where \( p + 1 \), the number of evaluations of \( f \) computed, is referred to as the number of stages. The truncation error coefficients, \( T_{i,j} \), determined by comparing the Taylor series expansion of equation (2) with the Taylor series expansion of the solution about \( t_0 \), are nonlinear combinations of the \( C \), \( \alpha \), and \( \beta \) coefficients. For the classical RK algorithm, the \( j \)th error coefficient of order \( i \) assumes the form

\[
T_{i,j} = \left\{ \frac{F_{i,j} - 1/(pA_{i,j})}{B_{i,j}} \right\}
\]

\( j = 1, \ldots, \lambda_i \) where \( p = i \). For the \( m \)-fold algorithm, \( p = m + i \) and corresponds to the order of the term. (For \( m = 0 \), the \( m \)-fold algorithm is identical to the classical RK formula.) The \( A_{i,j} \) and \( B_{i,j} \) terms are constants (or functions of \( m \) for \( m \)-fold methods) and may be determined by recursive relations. (One should note that while references to \( m \)-fold RK algorithms may appear to complicate matters, the inclusion of these methods in RKEQ (or RKEQN) involves the insertion of only a few additional lines of coding. Once these additions are made, the classical RK error coefficients and the \( m \)-fold error coefficients are determined identically.) The complicated expression to generate in equation (3) is the \( F_{i,j} \) term, which is a combination of the \( C \), \( \alpha \), and \( \beta \) coefficients

\[
F_{i,j} = \sum_{k=k_0}^{p} C_k S_{i,j,k}
\]

with \( S_{i,j,k} \) being a combination of \( \alpha \) and \( \beta \) coefficients and where \( k_0 \) depends upon the number of summations embedded in \( S_{i,j,k} \).
The algorithm developed by Bettis and used for generating these $S_{i,j,k}$, $A_{i,j}$, and $B_{i,j}$ terms is documented in reference 1, where the $S_{i,j,k}$ terms are written in an abbreviated notation, e.g., $S_{8,13} = \alpha^2 \beta_8 \beta_\alpha$, with the subscript- ing and embedded summations being suppressed. The rules for writing the entire $S_{i,j,k}$ terms are also described thoroughly in reference 1. For the sake of interpreting the program, however, a few features need to be known about generating the terms in abbreviated notation. Denoting the number of truncation error coefficients of order $\mu$ by $\lambda_\mu$, and suppressing the $k$ subscript, the first $2\lambda_{\mu-1}$ $S_{i,j,k}$ expressions are generated from the $S_{i-1,j,k}$ terms. The remaining $\lambda_\mu - 2\lambda_{\mu-1}$ expressions, referred to as composite sums, are formulated as products of lower order $S$ terms. The $A_{i,j}$ and $B_{i,j}$ constants are also generated from simple relationships involving previous $A$ and $B$ terms. In generating the first $\lambda_{i-1}$ terms of order $i$, the $S_{i-1,j}$ expressions are premultiplied by an $\alpha$. (Adjacent $\alpha$'s represent actual multiplication.) Thus, $S_{9,13} = \alpha S_8$, $S_{13} = \alpha^2 \beta_8 \beta_\alpha$. The next $\lambda_{i-1}$ terms, $S_{i,j}$, $j = \lambda_{i-1} + 1, \ldots, 2\lambda_{i-1}$ are generated by premultiplying the $S_{i-1,j-\lambda_{i-1}}$ expressions by a $\beta$, e.g., $S_{9,128} = \beta S_{8,13} = \beta \alpha^2 \beta_8 \beta_\alpha$. (Adjacent $\beta$'s do not represent multiplication of the $\beta$ coefficients since a summation sign precedes each $\beta$ when the entire $S$ term is written, e.g.,

$$S_{6,4,k} = \alpha^2 \beta_\alpha = \alpha^k \sum_{k=2}^{k=1} \beta_{k,l} \sum_{l=1}^{l=1} \beta_{l,m}$$  \hspace{1cm} (5)

($S_{6,4,k} \neq \alpha^2 \beta_2 \alpha$.)) The recursion relations, then, for the first $\lambda_{i-1}$ terms of order $i$, $(i > 2)$, are

$$S_{i,j} = \alpha S_{i-1,j}$$

$$A_{i,j} = A_{i-1,j}$$

$$B_{i,j} = \mu B_{i-1,j}$$

for $j = 1, 2, \ldots, \lambda_{i-1}$, where $\mu$ is the power of the leading $\alpha$ in the $S_{i-1,j}$ term, and

$$S_{i,j} = \beta S_{i-1,j-\lambda_{i-1}}$$

$$A_{i,j} = (m+i-1) A_{i-1,j-\lambda_{i-1}}$$
\[ B_{i,j} = B_{i-1,j-\lambda_{i-1}} \]

for \( j = \lambda_{i-1} + 1, \ldots, 2\lambda_{i-1} \). (\( S_{1,1} = 1 \), and \( S_{2,1} = \alpha \).) The first \( \lambda_{i-1} \) \( S \) expressions are referred to as alpha terms in subroutine RKEQ, while the next \( \lambda_{i-1} \) expressions are called beta terms. The remaining terms, composite sums, are generated by considering the weight factors of the \( S \) terms. The \( S_{\mu,j} \) expressions have a weight factor of \( \mu-1 \) (i.e., the number of \( \alpha \) and \( \beta \) coefficients included in the \( S \) term). The composite sums of order \( i \) are all products of \( S_{\mu,j} \) terms having initial \( \beta \) coefficients, whose weight factors add up to \( i-1 \). Subroutine RKEQ determines these composite sums in a separate block of the subprogram, calling subroutine CROSS to perform the multiplication of the \( S \), \( A \), and \( B \) terms. The \( A_{i,j} \) and \( B_{i,j} \) terms of a composite sum are the products of the \( A \) and \( B \) constants whose corresponding \( S \) terms form the composite sum. (When an \( S \) term is raised to the power \( k \), an additional \( k! \) multiplies the \( B \) constant.)
3.0 DESCRIPTION OF THE FORTRAN PROGRAM

Subroutine RKEQ determines the truncation error coefficients (TEC) for a given set of RK coefficients and returns TERROR,

\[
TERROR = \left( \sum_{j=1}^{K} T_i^2 \right)^{1/2}
\]

for a specified order \( i \). Since RK algorithms with embedded pairs of solutions, e.g. RK-Fehlberg formulas, are often studied, RKEQ is written to treat two algorithms simultaneously, which use identical \( \alpha \) and \( \beta \) coefficients but different \( C_k \) and \( \hat{C}_k \) coefficients. (TERROR is formed using the \( \hat{C}_k \) coefficients.) The Greek letters \( \alpha_i \) and \( \beta_i \) are replaced by \( A(I) \) and \( B(I) \) and the \( A_{i,j} \) and \( B_{i,j} \) constants are denoted \( AA(I,J) \) and \( BB(I,J) \), respectively.

The input parameters for RKEQ are:

(1) The RK coefficients
   
   (a) \( A(K) \) \( a_k \), the alpha coefficients
   
   (b) \( C_0, C(K) \) \( C_0, C_k \), the \( C_k \) coefficients for the first solution
   
   (c) \( CH_0, CH(K) \) \( \hat{C}_0, \hat{C}_k \), the \( \hat{C}_k \) coefficients for the second solution
      
      used to form TERROR
   
   (d) \( B_0(K), B(K,L) \) \( \beta_k, \beta_k \), the beta coefficients

   where \( K = 1,2,\ldots,R \), \( L = 1,2,\ldots, K - 1 \), \( R \) an integer with \( R + 1 \) being the number of stages of the algorithm, and

(2) The integers controlling orders and options

   (a) \( R \) = the index for dimensioning the RK coefficients
   
   (b) \( IORDER \) = the maximum order to be treated
   
   (c) \( ITERR \) = the order of TEC used to form TERROR
   
   (d) \( IOPT = \) the options for operating the program. For \( IOPT = 1 \), RKEQ computes and prints all TEC(I,J) for \( I = 1,\ldots,IORDER \). For \( IOPT = 2 \), RKEQ computes and prints TEC(IORDER, J) only. For \( IOPT = 3 \), RKEQ computes but does not print TEC(ITERR, J). (For all options TERROR is computed, which may require internal adjustments to the order.)

   (e) \( MFOLD = \) an integer giving the number of known derivatives of \( f \). For the classical RK algorithm, \( MFOLD = 0 \).
(f) LS = a dimensioning index for the work arrays, S, AA and BB.

The output parameter for RKEQ is TERROR, the Euclidean norm of the TEC of order ITERR. Depending upon the option used, RKEQ may print values of TEC, but these are not returned to the main program.

Parameters S, AA, and BB are used internally by RK5 to compute the TEC terms. To take advantage of variable dimensioning, these parameters are given in the calling sequence with dimensions S(LS,R), AA(LS), BB(LS). The RK coefficients should be dimensioned A(R), C(R), CH(R), B(R, R), BO(R), R an integer, where R + 1 is the number of stages for the algorithm.

The calling sequence for RKEQ is

```
```

which, if a printing option is used, will give the TEC from the C solution in the first column and the TEC from the CH solution in the second column. Integers IORDER, ITERR, and IOPT are reset within the subroutine, and any adjustments made to protect against exceeding dimension or option limits are made to the new variables, so that the user may enter constant values in the calling sequence of the driving program.

A sample calling sequence for a six-stage, fifth-order algorithm is

```
CALL RKEQ(A, C, CO, CH, CHO, B, BO, 5, 7, 6, 1, 0, TERROR, S, 48, AA, BB)
```

which computes and prints all TEC through order 7 for the classical RK formulas, using the 6th-order terms of the CH solution to form TERROR. Using the calling sequence

```
CALL RKEQ(A, CH, CHO, C, CO, B, BO, 5, 7, 6, 1, 0, TERROR, S, 48, AA, BB)
```

generates similar information except that TERROR is formed by the C solution (and the C TEC terms are now printed in the first column.)

The minimum value of LS for a given ORDER, I, is found in table I. A listing of subroutines RKEQ and CROSS may be found in the appendix.

<table>
<thead>
<tr>
<th>ORDER</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS_MIN</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>20</td>
<td>48</td>
<td>115</td>
<td>286</td>
<td>719</td>
<td>1842</td>
<td>4766</td>
</tr>
</tbody>
</table>
4.0 CONCLUDING REMARKS

A program, which evaluates the truncation error coefficients, is an essential tool in the development of Runge-Kutta algorithms and in the comparison of existing RK algorithms. By structuring the routine in the given form, a substantial savings in storage occurs in generating these truncation error coefficients using the recursive formulation presented by D. G. Bettis (ref. 1). The extension to orders higher than 12 is relatively simple but not of great practical use at the present time.
REFERENCES


APPENDIX A

SUBROUTINE RKEQ
SUBROUTINE RKEQ(A,C,CO,CH,BO,R,
1 ORDER,ITERR,IOPT,MP,D,TERROR,S,LS,AA,BB)
INTEGER R,ORDER,IOPT
DOUBLE PRECISION A(R),C(R),CH(R),BO(R)
DOUBLE PRECISION S(R),AA(LS),BB(LS)
DOUBLE PRECISION M(12),MP(12),FACT(12)
DOUBLE PRECISION P,PP1,PM1,PM2,ZAPP,TERROR
LOGICAL EVEN

DATA LIMIT/1,1,2,4,9,20,48,115,186,286,719,1842,4766/
DATA INDEX/1,1,2,5,11,28,67,171,433,1123,2924/
DATA ZERO,UNITY,TWO/ZERO,1.0D0,2.0D0,1.0D-14/
DATA MAXORD/12/
DATA KPRINT/0/

SUBROUTINE RKEQ IS A FORTRAN SUBROUTINE WRITTEN BY
M.K. HORN WHICH IMPLEMENTS THE ALGORITHM DEVELOPED
BY D.G. BETTIS TO GENERATE TRUNCATION ERROR COEFFI-
CIENTS FOR RUNGE-KUTTA ALGORITHMS.

REFERENCE: BETTIS, D.G. AND M.K. HORN, "COMPUTATION
OF TRUNCATION ERROR TERMS FOR RUNGE-KUTTA METHODS,"
TICOM REPORT 77-14, DECEMBER, 1978.

SUBROUTINE RKEQ DETERMINES THE TRUNCATION ERROR
COEFFICIENTS (TEC) FOR A RUNGE-KUTTA ALGORITHM HAVING
AN EMBEDDED PAIR OF SOLUTIONS

\[ Y = Y + H \sum_{k=0}^{K} C F_k \]

\[ R \]

\[ YH = Y + H \sum_{k=0}^{K} CH F_k \]
C

K=0

C

WHERE

\[ F = F(T, Y) \]
\[ F = F(T + A H, Y + H \sum_{K,L} F) \]

C

FOR MFOLD = 0, RKEQ FORMS THE TEC FOR THE CLASSICAL RK FORMULAS OF ORDER = IORDER. FOR MFOLD = M, RKEQ FORMS THE TEC FOR MFOLD-RK FORMULAS OF ORDER = M + IORDER.

C

OPTION .EQ. 1 COMPUTES AND PRINTS ALL TRUNCATION ERROR COEFFICIENTS THROUGH ORDER = IORDER
C

OPTION .EQ. 2 COMPUTES AND PRINTS ONLY T.E.C. OF ORDER = IORDER
C

OPTION .EQ. 3 COMPUTES T.E.C. AS IN OPTION = 2 BUT DOES NOT PRINT

C

ADJUSTS INPUT PARAMETERS IF THESE ARE NOT WITHIN ALLOWABLE RANGE

ORDER = IORDER
LIMITS = ITERR
IF (ORDER .LT. LIMITS) ORDER = LIMITS
IF (LS. LT. LIMIT(MAXORD)) GO TO 1
IF (ORDER .LE. MAXORD) GO TO 1
ORDER = MAXORD
PRINT 507, MAXORD

507 FORMAT(52H ORDER REQUESTED IS BEYOND CAPABILITY OF THE PROGRAM
1 ,/ , 24H ORDER LOWERED--ORDER = , I2)
IF (ITERR .LE. ORDER) GO TO 1
PRINT 526, LIMITS

526 FORMAT(45H ITERR REQUESTED IS LARGER THAN MAXIMUM ORDER
1 ,/ , 35H ITERR HAS BEEN REDUCED TO ORDER = , I2 )
CONTINUE

TERROR = ZERO
OPTION = IOPT
IF (IOPT .LE. 0 .OR. IOPT .GT. 3) OPTION = 1
IF (LS .GE. LIMIT(ORDER)) GO TO 3
2 CONTINUE
ORDER = ORDER - 1
IF (LS .LT. LIMIT(ORDER)) GO TO 2
PRINT 505, ORDER, IORDER, LIMIT(IORDER)
505 FORMAT(50H THE ORDER SPECIFIED HAS BEEN REDUCED TO ORDER =
1, /, I3, /, 32H BECAUSE OF INSUFFICIENT STORAGE, /,
2 14H FOR ORDER = , I3, 20H LS MUST BE .GE., , I5,
3 /, 56H TERROR = SQRT(SUM(T.E.C.(I,J)*T.E.C.(I,J))) IS COMPUTED
4 , /, 14H FOR I = ORDER )
C
IF (LIMITS .LE. ORDER) GO TO 3
LIMITS = ORDER
PRINT 527, LIMITS
527 FORMAT(45H ITERR REQUESTED IS LARGER THAN THE PROVIDED
1, /, 50H DIMENSIONING. ITERR HAS BEEN REDUCED TO ITERR = ,
2 I2)
3 CONTINUE
C
IF (ORDER .GT. MAXORD) ORDER = MAXORD
C
INDEX -- COUNTS THE NUMBER OF S(J,K) WITH A GIVEN A**K
AS THE FIRST TERM IN THE EXPRESSION, E.G.,
FOR J=7, THERE ARE 1 A**6, 1 A**4, 2 A**3,
4 A**2, AND 9 A**1 @S

X1 = A(1) - BO(1)
JJ = 1
550 FORMAT(15H ERROR IN BETA(, I2, 1OH SUM = , D15.7)
IF (DABS(X1) .GE. ZAPP) PRINT 550, JJ, X1
DO 5 J = 2, R
X1 = A(J) - BO(J)
JLOW = J-1

DO 4 K = 1, JLOW
5 CONTINUE

4 X1 = X1 - B(J,K)
IF (DABS(X1) .GE. ZAPP) PRINT 550, J, X1
CONTINUE

C
IF (KPRINT .EQ. 1) READ PRINT 499
C
SETS VALUES OF FACTORIALS USED IN THE PROGRAM.
ADJUSTMENTS IN THE PROGRAM TO ACCOMODATE M-FOLD RUNGE-KUTTA ALGORITHMS OCCUR HERE. IF MFOLD = 0,
THE CLASSICAL RUNGE-KUTTA T.E.C. OCCUR

IF (MFOLD .GT. 0) GO TO 6
EVEN = .TRUE.
GO TO 8
6 CONTINUE
N1 = MFOLD / 2
N2 = 2 * N1
EVEN = .FALSE.
IF (N2 .EQ. MFOLD) EVEN = .TRUE.
8 CONTINUE
MP(1) = DFLOAT(MFOLD)+UNITY
M(1) = 1
X1 = UNITY
FACT(1) = UNITY
DO 10 I = 2, ORDER
X1 = X1 + UNITY
FACT(I) = FACT(I-1)*X1
MP(I) = MP(I-1) + UNITY
10 M(I) = M(I-1) + 1
M2 = 1
M1 = 0
11 CONTINUE
M1 = M1 + 1
M2 = M2*M1
IF (M1 .LT. MFOLD+1) GO TO 11
MM(1) = DFLOAT(M2)
DO 12 I = 2, ORDER
12 MM(I) = MP(I)* MM(I-1)
CONTINUE

IF (KPRINT .EQ. 1) PRINT 499
C
SETS INITIAL VALUES OF S(I,J) EQUAL TO ZERO
AND SETS AA(J) AND BB(J) EQUAL TO UNITY
C
DO 14 J = 1, LS
AA(J) = UNITY

BB(J) = UNITY
DO 14 K = 1,R
14 S(J,K) = ZERO

C IF (OPTION .GT. 1 .AND. LIMITS .NE. 1) GO TO 21
C EVALUATES T.E.C. OF ORDER 1
C SUM1 = UNITY/DFLOAT(MFOLD + 1) - CO
SUM3 = UNITY/DFLOAT(MFOLD + 1) - CHO
DO 20 I = 1,R
SUM1 = SUM1 - C(I)
SUM3 = SUM3 - CH(I)
20 CONTINUE

C JJ1 = 1
IF (LIMITS .EQ. 1) TERROR = SUM3*SUM3
IF (OPTION .EQ. 3) GO TO 21
PRINT 500,JJ1
PRINT 501,JJ1,SUM1,SUM3
500 FORMAT(36H TRUNCATION ERROR TERMS X-ORDER = ,I2,2X,1,0/0)
501 FORMAT(2(2X,I4),2(D15.7))
C 21 CONTINUE
C SETS S(1,J) TERMS
C KOUNT = 1
C DO 26 I = 1,R
IF (MFOLD .GT. 0) GO TO 22
S(1,I) = A(I)
GO TO 26
22 CONTINUE
IF (EVEN) GO TO 24
S(1,I) = DABS(A(I))**MP(1)
GO TO 26
24 CONTINUE
X1 = DABS(A(I))**MP(1)
S(1,I) = DSIGN(X1,A(I))
26 CONTINUE
C EVALUATES T.E.C. FOR ORDER = KOUNT = 2,3
C AA(1) = UNITY
BB(1) = MM(1)
28 CONTINUE
P = TWO + DFLOAT(MFOLD)

PP1 = P+UNITY

30 CONTINUE

IF (LIMITS .EQ. KOUNT) JJ1 = KOUNT - 1
IF (OPTION .GT. 1 .AND. LIMITS .NE. KOUNT) GO TO 34
JJ2 = 0
JJ1 = KOUNT + 1
PRINT 500, JJ1
DO 33 K = 1, KOUNT
JJ2 = JJ2 + 1
SUM1 = UNITY/(AA(K)*P)
SUM3 = SUM1
DO 32 I = 1, R
SUM1 = SUM1 - C(I)*S(K, I)
SUM3 = SUM3 - CH(I)*S(K, I)
32 CONTINUE
IF (LIMITS .EQ. KOUNT) TERROR = TERROR + SUM3*SUM3
IF (OPTION .EQ. 3) GO TO 33
PRINT 501, JJ1, JJ2, SUM1, SUM3
33 CONTINUE

C

34 CONTINUE
C

IF (KOUNT .EQ. 2) GO TO 38
C

SETS S(1, J), S(2, J) FOR THIRD ORDER T.E.C.
C

KOUNT = 2
P = P + UNITY
PP1 = PP1 + UNITY
DO 35 I = 2, R
S(2, I) = ZERO
IM1 = I-1
DO 35 J = 1, IM1
S(2, I) = S(2, I) + B(I, J)*S(1, J)
35 S(2, I) = S(2, I) + B(I, J)*S(1, J)
DO 36 I = 1, R
S(1, I) = S(1, I)*A(I)
AA(2) = TWO
BB(1) = MM(2)
BB(2) = MM(1)
GO TO 30
38 CONTINUE
IF (ORDER .LE. 3) GO TO 182
LIM1 = 3
C

EVALUATES T.E.C. FOR ORDERS GREATER THAN THREE
C

DO 180 J = 4, ORDER
P = P + UNITY
PP1 = P + UNITY
A-8
PM1 = P - UNITY

PM2 = P - TWO

IF (KPRINT .EQ. 1) READ 497,II

LIM1 = LIM1 + 1
LIMA = LIMIT(J-1)
LIMB = LIMA

COMPUTES S(1,I)--ALL OTHER S(J,K) TERMS INVOLVING A
LEADING ALPHA ARE ALREADY DETERMINED
EXCEPT FOR THE POWER OF ALPHA WHICH
IS DETERMINED BY IN INDEX AND J

AA(1) = AA(1)
BB(1) = BB(1) * MP(LIM1-1)

DO 42 I = 1,R

42 S(1,I) = S(1,I)*A(I)

LIM1 = INTEGER P

BETA TERMS

MARKB = LIM1+1
DO 61 I = 2,R

S(MARKB,I) = ZERO
IM1 = I-1
DO 61 K = 1,IM1

61 S(MARKB,I) = S(MARKB,I)+B(I,K)*A(K)**(MFOLD+LIM1-2)

AA(MARKB)=PM1
BB(MARKB) = MM(LIM1-2)
IND2 = 1
IND1 = MARKB
LL = LIM1 - 4
IF (LIM1 .EQ. 4) GO TO 66
DO 65 K = 1,LL

LL1 = INDEX(K+1)
IPOW = LL-K+1
DO 65 KK = 'I , LL1

IND1 = IND1 + 1
IND2 = IND2 + 1
DO 64 I = 2,R

S(IND1,I) = ZERO
IM1 = I-1
DO 64 L = 1,IM1

64 S(IND1,I) = S(IND1,I)+B(I,L)*S(IND2,L)*A(L)**M(IPOW)
BB(IND1) = BB(IND2)*FACT(IPOW)
AA(IND1) = AA(IND2)*PM1
CONTINUE

LL = LIMB - IND2
DO 68 K = 1, LL

IND1 = IND1 + 1
IND2 = IND2 + 1
DO 67 I = 2, R
S(IND1, I) = ZERO
IM1 = I - 1
DO 67 L = 1, IM1
67 S(IND1, I) = S(IND1, I) + B(I, L)*S(IND2, L)
BB(IND1) = BB(IND2)
68 AA(IND1) = AA(IND2)*PM1
883 CONTINUE
JM3 = J - 3
GO TO (150, 100, 105, 110, 115, 120, 125, 130, 135), JM3

C
C CROSS PRODUCT TERMS
C
100 CONTINUE
C
5 TH ORDER TERMS
C
IND = 2*L I M I T(4) + 1
CALL CROSS(LS, R, S, AA, BB, IND, 2, 2, 0, 0)
GO TO 150
C
6 TH ORDER TERMS
C
105 CONTINUE
IND = 2*L I M I T(5) + 1
CALL CROSS(LS, R, S, AA, BB, IND, 2, 1, 3, 4)
GO TO 150
C
7 TH ORDER TERMS
C
110 CONTINUE
IND = 2*L I M I T(6) + 1
CALL CROSS(LS, R, S, AA, BB, IND, 2, 1, 5, 8)
CALL CROSS(LS, R, S, AA, BB, IND, 2, 3, 0, 0)
CALL CROSS(LS, R, S, AA, BB, IND, 3, 2, 0, 0)
CALL CROSS(LS, R, S, AA, BB, IND, 4, 2, 0, 0)
CALL CROSS(LS, R, S, AA, BB, IND, 3, 1, 4, 4)
GO TO 150
C
8 TH ORDER TERMS
C
115 CONTINUE
IND = 2*L I M I T(7) + 1
CALL CROSS(LS, R, S, AA, BB, IND, 2, 1, 10, 18)
CALL CROSS(LS, R, S, AA, BB, IND, 3, 1, 5, 8)
CALL CROSS(LS, R, S, AA, BB, IND, 4, 1, 5, 6)
CALL CROSS(LS,R,S,AA,BB,IND,2,2,3,4) 00040000

GO TO 150 00040100
C 9 TH ORDER TERMS 00040200
120 CONTINUE 00040300
   IND = 2*LIMIT(8)+1 00040400
   CALL CROSS(LS,R,S,AA,BB,IND,2,1,21,40) 00040500
   CALL CROSS(LS,R,S,AA,BB,IND,3,1,10,13) 00040600
   CALL CROSS(LS,R,S,AA,BB,IND,4,1,10,18) 00040700
   DO 121 K = 5,8 00040800
   121 CALL CROSS(LS,R,S,AA,BB,IND,K,1,K,8) 00040900
   CALL CROSS(LS,R,S,AA,BB,IND,9,1,5,8) 00041000
   CALL CROSS(LS,R,S,AA,BB,IND,2,4,0,0) 00041100
   CALL CROSS(LS,R,S,AA,BB,IND,4,1,21,40) 00041200
   GO TO 150 00041300
C 10 TH ORDER TERMS 00041400
C 125 CONTINUE 00041500
   IND = 2*LIMIT(9) + 1 00041600
   CALL CROSS(LS,R,S,AA,BB,IND,2,1,49,96) 00041700
   CALL CROSS(LS,R,S,AA,BB,IND,3,1,21,40) 00041800
   CALL CROSS(LS,R,S,AA,BB,IND,4,1,21,40) 00041900
   DO 126 K = 5,9 00042000
   126 CALL CROSS(LS,R,S,AA,BB,IND,K,1,10,18) 00042100
   CALL CROSS(LS,R,S,AA,BB,IND,2,1,106,113) 00042200
   CALL CROSS(LS,R,S,AA,BB,IND,2,3,3,4) 00042300
   CALL CROSS(LS,R,S,AA,BB,IND,3,3,0,0) 00042400
   CALL CROSS(LS,R,S,AA,BB,IND,4,3,0,0) 00042500
   CALL CROSS(LS,R,S,AA,BB,IND,4,2,3,3) 00042600
   CALL CROSS(LS,R,S,AA,BB,IND,3,2,4,4) 00042700
   GO TO 150 00042800
C 130 CONTINUE 00042900
C 11TH ORDER TERMS 00043000
C 131 CONTINUE 00043100
   IND = 2*LIMIT(10) + 1 00043200
   CALL CROSS(LS,R,S,AA,BB,IND,2,1,116,230) 00043300
   CALL CROSS(LS,R,S,AA,BB,IND,3,1,49,96) 00043400
   CALL CROSS(LS,R,S,AA,BB,IND,4,1,49,96) 00043500
   DO 131 K = 5,8 00043600
   131 CALL CROSS(LS,R,S,AA,BB,IND,K,1,21,40) 00043700
   CALL CROSS(LS,R,S,AA,BB,IND,19,1,10,18) 00043800
   CALL CROSS(LS,R,S,AA,BB,IND,2,3,1,18) 00043900
   CALL CROSS(LS,R,S,AA,BB,IND,4,1,21,40) 00044000
   DO 132 K = 10,18 00044100
   132 CALL CROSS(LS,R,S,AA,BB,IND,K,1,1,K,18) 00044200
   CALL CROSS(LS,R,S,AA,BB,IND,2,2,21,40) 00044300
   CALL CROSS(LS,R,S,AA,BB,IND,19,1,10,18) 00044400
   CALL CROSS(LS,R,S,AA,BB,IND,19,1,10,18) 00044500
CALL CROSS(LS,R,S,AA,BB,IND,20,1,10,18)

CALL CROSS(LS,R,S,AA,BB,IND,2,1,269,278)
DO 133 K = 46,48
133 CALL CROSS(LS,R,S,AA,BB,IND,K,1,5,8)
CALL CROSS(LS,R,S,AA,BB,IND,2,3,5,8)
CALL CROSS(LS,R,S,AA,BB,IND,2,2,46,48)
CALL CROSS(LS,R,S,AA,BB,IND,2,5,0,0)
GO TO 150
C 12TH ORDER TERMS
C 135 CONTINUE
C
C  IND = 2 LIMIT(11) + 1
C
C CALL CROSS(LS,R,S,AA,BB,IND,2,1,287,572)
DO 141 K = 5,8
141 CALL CROSS(LS,R,S,AA,BB,IND,K,1,49,96)
DO 142 K = 10,18
142 CALL CROSS(LS,R,S,AA,BB,IND,K,1,21,40)
CALL CROSS(LS,R,S,AA,BB,IND,2,2,49,96)
CALL CROSS(LS,R,S,AA,BB,IND,19,1,21,40)
CALL CROSS(LS,R,S,AA,BB,IND,20,1,21,40)
DO 143 K = 41,48
143 CALL CROSS(LS,R,S,AA,BB,IND,K,1,10,18)
CALL CROSS(LS,R,S,AA,BB,IND,114,1,5,8)
CALL CROSS(LS,R,S,AA,BB,IND,115,1,5,8)
DO 144 K = 269,278
144 CALL CROSS(LS,R,S,AA,BB,IND,K,1,3,4)
CALL CROSS(LS,R,S,AA,BB,IND,3,3,2,2)
CALL CROSS(LS,R,S,AA,BB,IND,4,3,2,2)
CALL CROSS(LS,R,S,AA,BB,IND,3,2,20,20)
CALL CROSS(LS,R,S,AA,BB,IND,4,2,19,19)
CALL CROSS(LS,R,S,AA,BB,IND,2,4,3,4)
CONTINUE
C TEMPORARY INSERT TO CHECK VALUES OF AA AND BB COEFF
C
C LL = LIMIT(LIM1)
C IFAKE = 0
C DO 153 K = 1,LL
C IFAKE = IFAKE + 1
C IF (IFAKE .LT. 40) GO TO 153
C IF (KPRINT .EQ. 1) READ 497,II
C IFAKE = 0
C 153 PRINT 506,K,AA(K),K,BB(K)
A-12
C 506 FORMAT( 4H AA(,I4, 4H) = ,D15.7,2X,4H BB(,
.
C 1 I4,4H) = ,D15.7)
   IF (KPRINT .EQ. 1) PRINT 499
C
C IF (LIMITS .EQ. J) JJ1 = J - 1
   IF (OPTION .GT. 1 .AND. LIMITS .NE. J) GO TO 180
C
JJ2 = 1
JJ1 = J
C
C EVALUATES FIRST T.E.C. OF ORDER J
C
SUM1 = UNITY/(AA(1)*P)
SUM3 = SUM1
DO 154 I = 1,R
SUM1 = SUM1 - C(I)*S(1,I)
SUM3 = SUM3 - CH(I)*S(1,I)
154 CONTINUE
SUM1 = SUM1 / BB(1)
SUM3 = SUM3 / BB(1)
IF (LIMITS .EQ. J) TERROR = TERROR + SUM3*SUM3
IF (OPTION .EQ. 3) GO TO 159
PRINT 501,JJ1,JJ2,SUM1,SUM3
155 CONTINUE
C
C EVALUATES T.E.C. FOR S(I,J) TERMS WITH ALPHA AS LEADING COEFFICIENT (I .NE. 2)
C
IFAKE = 1
K = 1
KNT = 2
IPOW = LIM1 - 3
LIMD = INDEX(KNT)
156 CONTINUE
DO 159 KK = 1,LIMD
K = K + 1
SUM1 = UNITY/(AA(K)*P)
SUM3 = SUM1
DO 157 I = 1,R
SUM1 = SUM1 - C(I)*A(I)**IPOW*S(K,I)
SUM3 = SUM3 - CH(I)*A(I)**IPOW*S(K,I)
157 CONTINUE
SUM1 = SUM1 / (BB(K)*FACT(IPOW))
SUM3 = SUM3 / (BB(K)*FACT(IPOW))
IF (LIMITS .EQ. J) TERROR = TERROR + SUM3*SUM3
IF (OPTION .EQ. 3) GO TO 159
JJ2 = JJ2 + 1
IFAKE = IFAKE + 1

IF (IFAKE .LT. 40) GO TO 158
IFAKE = 0
IF (KPRINT .EQ. 1) PRINT 499
158 CONTINUE
497 FORMAT(I3)
PRINT 501,JJ1,JJ2,SUM1,SUM3
159 CONTINUE

C
KNT = KNT + 1
LIMD = INDEX(KNT)
IPOW = IPOW - 1
IF (IPOW .GE. 1) GO TO 156

C
LIMD = LIMIT(LIM1) - LIMA
C
EVALUATES T.E.C. FOR S(I,J) TERMS WITH BETA AS
C
LEADING COEFFICIENT
C
DO 162 KK = 1,LIMD
K = K + 1
SUM1 = UNITY/(AA(K)*P)
SUM3 = SUM1
DO 160 I = 1,R
SUM1 = SUM1 - C(I)*S(K,I)
SUM3 = SUM3 - CH(I)*S(K,I)
160 CONTINUE
SUM1 = SUM1 / BB(K)
SUM3 = SUM3 / BB(K)
IF (LIMITS .EQ. J) TERROR = TERROR + SUM3*SUM3
IF (OPTION .EQ. 3) GO TO 162
IFAKE = IFAKE + 1
IF (IFAKE .LT. 40) GO TO 161
IFAKE = 0
IF (KPRINT .EQ. 1) PRINT 499
161 CONTINUE
JJ2 = JJ2 + 1
C
PRINT 556,K,LIM1,AA(K),P,PP1
556 FORMAT(2(2X,I4),3(2X,D15.7))
PRINT 501,JJ1,JJ2,SUM1,SUM3
162 CONTINUE
IF (KPRINT .EQ. 1) PRINT 499
499 FORMAT(/)
180 CONTINUE
182 CONTINUE
TERROR = DSQRT(TERROR)
RETURN
END
SUBROUTINE CROSS(LS,R,S,AA,BB,INDEX,TERM1,POWER,TERM2,  
            TERM3)
    INTEGER TERM1,TERM2,TERM3,POWER,R
    DOUBLE PRECISION S(LS,R),AA(LS),BB(LS),FACT(7)
    DATA FACT/1.0D0,2.0D0,6.0D0,24.0D0,  
            120.0D0,720.0D0,5040.0D0/

    IF (TERM2 .EQ. 0) GO TO 20

    COMPUTES S(INDEX+J,I)=S(TERM1,I)**POWER * S(TERM2+J,I)  
    FOR J=0,1,...,TERM3-TERM2

    KK = 1
    DO 10 K = TERM2,TERM3
        KK = KK + 1
        INDEX = INDEX + KK
        AA(INDEX) = AA(TERM1)**POWER * AA(K)
        IPOW = POWER
        IF (TERM1 .EQ. K) IPOW = IPOW + 1
        BB(INDEX) = BB(TERM1)**POWER * BB(K) * FACT(IPOW)
    DO 10 12,R
       10 S(INDEX,I) = S(TERM1,I)**POWER * S(K,I)
       INDEX = INDEX + TERM3 - TERM2 + 1

    RETURN

    COMPUTES S(INDEX,I) = S(TERM1,I)**POWER

    CONTINUE
    AA(INDEX) = AA(TERM1)**POWER
    BB(INDEX) = BB(TERM1)**POWER * FACT(POWER)
    DO 25 I = 2,R
       25 S(INDEX,I) = S(TERM1,I)**POWER
       INDEX = INDEX + 1

    RETURN
END