80-FM-13

JSC-16429

AUG 5 1980

A Computer Program for Determining Truncation Error Coefficients for Runge-Kutta Methods

(NASA-TM-81143) A COMPUTER PROGRAM FOR DETERMINING TRUNCATION ERROR COEFFICIENTS FOR RUNGE-KUTTA METHODS (NASA) 27 p HC A03/MF A01 CSCL 12A

N80-30095

Unclas

G3/64 26923

Mission Planning and Analysis Division

July 1980

NASA

National Aeronautics and Space Administration

Lyndon B. Johnson Space Center

Houston, Texas

80FM13

80-FM-13

JSC-16429

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:
Ronald 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

CONTENTS

TABLES

1.0 INTRODUCTION

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

$$\frac{dy}{dt} = f(t,y), \quad y(t_o) = y_o \tag{1}$$

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 λ_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 programing, 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

---

[a]From 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 $t_1 = t_o + h$, using the RK algorithm, is written

$$y_1 = y_o + \sum_{k=o}^{\rho} C_k f_k \tag{2}$$

where

$$f_o = f(t_o, y_o)$$

$$f_k = f(t_o + \alpha_k h,\ y_o + h \sum_{\lambda=o}^{k-1} \beta_{k,\lambda} f_\lambda)$$

where $\rho + 1$, 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_o, are nonlinear combinations of the C, α, and β coefficients. For the classical RK algorithm, the jth error coefficient of order i assumes the form

$$T_{i,j} = \left\{F_{i,j} - 1/(pA_{i,j})\right\} / B_{i,j} \tag{3}$$

$j = 1,\ldots,\lambda_i$ 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, α, and β coefficients

$$F_{i,j} = \sum_{k=k_o}^{\rho} C_k S_{i,j,k} \tag{4}$$

with $S_{i,j,k}$ being a combination of α and β coefficients and where k_o 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., $S_{8,13} = \alpha^2\beta\alpha\beta\beta\alpha$, with the subscripting 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 μ, by λ_μ, 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 λ_{i-1} terms of order i, the $S_{i-1,j}$ expressions are premultiplied by an α. (Adjacent α's represent actual multiplication.) Thus, $S_{9,13} = \alpha S_{8,13} = \alpha^3\beta\alpha\beta\beta\alpha$. The next λ_{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 β, e.g., $S_{9,128} = \beta S_{8,13} = \beta\alpha^2\beta\alpha\beta\beta\alpha$. (Adjacent β's do not represent multiplication of the β coefficients since a summation sign precedes each β when the entire S term is written, e.g.,

$$S_{6,4,k} = \alpha^2\beta\beta\alpha = \alpha_k^2 \sum_{\ell=2}^{k-1} \beta_{k,\ell} \sum_{m=1}^{\ell-1} \beta_{\ell,m}\alpha_m \tag{5}$$

($S_{6,4,k} \neq \alpha^2\beta^2\alpha$.)) The recursion relations, then, for the first λ_{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 μ is the power of the leading α 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 λ_{i-1} S expressions are referred to as alpha terms in subroutine RKEQ, while the next λ_{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 α and β coefficients included in the S term). The composite sums of order i are all products of $S_{\mu,j}$ terms having initial β 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,

$$\text{TERROR} = \left\{ \sum_{j=1}^{\lambda_i} T_{i,j}^2 \right\}^{1/2}$$

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 α and β coefficients but different C_k and $\hat{C}_k$ coefficients. (TERROR is formed using the $\hat{C}_k$ coefficients.) The Greek letters α_i and β_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) α_k, the alpha coefficients
- (b) CO, C(K) C_o, C_k, the C_k coefficients for the first solution
- (c) CHO, CH(K) $\hat{C}_o$, $\hat{C}_k$, the $\hat{C}_k$ coefficients for the second solution used to form TERROR
- (d) BO(K), B(K,L) $\beta_{k,o}$, $\beta_{k,\ell}$, the beta coefficients

where K = 1,2,...,R, L = 1,2,..., 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,...,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 RKEQ 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

```
SUBROUTINE RKEQ(A, C, CO, CH, CHO, B, BO, R, IORDER, ITERR, IOPT, MFOLD,
TERROR, S, LS, AA, BB).
```

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 $\hat{C}$ 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 I.- DIMENSIONING PARAMETER, LS, FOR ORDERS 1 THROUGH 12

ORDER	1	2	3	4	5	6	7	8	9	10	11	12
LS_{MIN}	1	1	2	4	9	20	48	115	286	719	1842	4766

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

1. Bettis, D. G.; and Horn, M. K.: Computation of Truncation Error Terms for Runge-Kutta Methods. TICOM Report 77-14, December 1977.

2. Fehlberg, E.: New High-Order Runge-Kutta Formulas with Step-size Control for Systems of First- and Second-Order Differential Equations. ZAMM, Vol. 44, 1964.

3. Fehlberg, E.: New High-Order Runge-Kutta Formulas with an Arbitrarily Small Truncation Error. ZAMM, Vol. 6, 1966.

APPENDIX A

SUBROUTINE RKEQ

```
      SUBROUTINE RKEQ(A,C,CO,CH,CHO,B,BO,R,
     1  IORDER,ITERR,IOPT,MFOLD,TERROR,S,LS,AA,BB)
      INTEGER R,ORDER,OPTION
      DOUBLE PRECISION A(R),C(R),CH(R),B(R,R),BO(R)
      DOUBLE PRECISION CO CHO
      DOUBLE PRECISION S(LS,R),AA(LS),BB(LS)
      DOUBLE PRECISION SUM1,SUM3,X1,ZERO,UNITY,TWO,
     1  MP(12),MM(12),FACT(12)
      DOUBLE PRECISION P,PP1,PM1,PM2,ZAPP,TERROR
      DIMENSION M(12),LIMIT(12),INDEX(11)
      LOGICAL EVEN
C
      DATA LIMIT/1,1,2,4,9,20,48,115,286,719,1842,4766/
      DATA INDEX/1,1,2,5,11,28,67,171,433,1123,2924/
      DATA ZERO,UNITY,TWO,ZAPP/0.0D0,1.0D0,2.0D0,1.0D-14/
      DATA MAXORD/12/
C
C******************************
C
      DATA KPRINT/0/
C
C******************************
C
C
C
C     SUBROUTINE RKEQ IS A FORTRAN SUBROUTINE WRITTEN BY
C     M.K. HORN WHICH IMPLEMENTS THE ALGORITHM DEVELOPED
C     BY D.G. BETTIS TO GENERATE TRUNCATION ERROR COEFFI-
C     IENTS FOR RUNGE-KUTTA ALGORITHMS.
C
C     REFERENCE: BETTIS,D.G. AND M.K. HORN, 'COMPUTATION
C     OF TRUNCATION ERROR TERMS FOR RUNGE-KUTTA METHODS,'
C     TICOM REPORT 77-14, DECEMBER, 1978.
C
C     SUBROUTINE RKEQ DETERMINES THE TRUNCATION ERROR
C     COEFFICIENTS (TEC) FOR A RUNGE-KUTTA ALGORITHM HAVING
C     AN EMBEDDED PAIR OF SOLUTIONS
C
C                  R
C
C     Y  = Y  + H SUM C  F
C      1    0          K  K
C                 K=0
C
C
C                   R
C
C     YH  = Y  + H SUM CH  F
C       1    0           K  K
```


PAGE A-2 INTENTIONALLY BLANK

```
C                    K=0

C
C
C
C     WHERE     F  = F(T ,Y )
C                0      0  0
C
C                                            K-1
C
C               F  = F(T  + A H, Y  + H * SUM B    F  )
C                K      0    K    0            K,L  L
C                                            L=0
C
C     FOR MFOLD = 0, RKEQ FORMS THE TEC FOR THE CLASSICAL RK
C     FORMULAS OF ORDER = IORDER.  FOR MFOLD = M, RKEQ FORMS
C     THE TEC FOR MFOLD-RK FORMULAS OF ORDER = M + IORDER.
C
C
C
C -------
C
C     OPTION .EQ. 1   COMPUTES AND PRINTS ALL TRUNCATION
C                     ERROR COEFFICIENTS THROUGH ORDER =
C                     IORDER
C     OPTION .EQ. 2   COMPUTES AND PRINTS ONLY T.E.C. OF
C                     ORDER = IORDER
C     OPTION .EQ. 3   COMPUTES T.E.C. AS IN OPTION = 2 BUT
C                     DOES NOT PRINT
C
C -------
C
C
C     ADJUSTS INPUT PARAMETERS IF THESE ARE NOT WITHIN
C     ALLOWABLE RANGE
C
C
      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
      LIMITS = ORDER
      PRINT 526,LIMITS
  526 FORMAT(45H ITERR REQUESTED IS LARGER THAN MAXIMUM ORDER
     1  ,/,35H ITERR HAS BEEN REDUCED TO ORDER =  ,I2  )
```

```
C
      1 CONTINUE
C
        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
C
C
        IF (ORDER .GT. MAXORD)   ORDER = MAXORD
C
C
C -------
C
C       MP(J) = (MFOLD+J)          D.P.
C       M(J) = J                   INTEGER
C       FACT(J) = J                D.P.
C       MM(J) = (MFOLD+J)          D.P.
C
C       INDEX--COUNTS THE NUMBER OF S(J,K) WITH A GIVEN A**K
C              AS THE FIRST TERM IN THE EXPRESSION, E.G.,
C              FOR J=7, THERE ARE 1 A**6, 1 A**4, 2 A**3,
C              4 A**2, AND 9 A**1 @S
C
C -------
C
        X1 = A(1) - B0(1)
        JJ = 1
    550 FORMAT(15H ERROR IN BETA(,I2,10H)   SUM =  ,D15.7)
        IF (DABS(X1) .GE. ZAPP)  PRINT 550,JJ,X1
        DO 5 J = 2,R
        X1 = A(J) - B0(J)
```

```
      JLOW = J-1

      DO 4 K = 1,JLOW
    4 X1 = X1 - B(J,K)
      IF (DABS(X1) .GE. ZAPP)  PRINT 550,J,X1
    5 CONTINUE
C
      IF (KPRINT .EQ. 1) READ PRINT 499
C
C
C     SETS VALUES OF FACTORIALS USED IN THE PROGRAM.
C     ADJUSTMENTS IN THE PROGRAM TO ACCOMODATE M-FOLD
C     RUNGE-KUTTA ALGORITHMS OCCUR HERE.  IF MFOLD = 0,
C     THE CLASSICAL RUNGE-KUTTA T.E.C. OCCUR
C
C
      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)
C
C
      IF (KPRINT .EQ. 1) PRINT 499
C
C     SETS INITIAL VALUES OF S(I,J) EQUAL TO ZERO
C     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
C     EVALUATES T.E.C. OF ORDER 1
C
      SUM1 = UNITY/DFLOAT(MFOLD + 1) -  C0
      SUM3 = UNITY/DFLOAT(MFOLD + 1) - CH0
      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,JJ1,SUM1,SUM3
  500 FORMAT(36H TRUNCATION ERROR TERMS   X-ORDER = ,I2,2X
     1  ,//)
  501 FORMAT(2(2X,I4),2(D15.7))
C
   21 CONTINUE
C
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
C     EVALUATES T.E.C. FOR ORDER = KOUNT = 2,3
C
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
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
   35 S(2,I) = S(2,I) + B(I,J)*S(1,J)
      DO 36 I = 1,R
   36 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
C     EVALUATES T.E.C. FOR ORDERS GREATER THAN THREE
C
      DO 180 J = 4,ORDER
      P = P + UNITY
      PP1 = P + UNITY
```

```
      PM1 = P - UNITY

      PM2 = P - TWO
      IF (KPRINT .EQ. 1) READ 497,II
      LIM1 = LIM1 + 1
      LIMA = LIMIT(J-1)
      LIMB = LIMA
C
C     COMPUTES S(1,I)--ALL OTHER S(J,K) TERMS INVOLVING A
C                      LEADING ALPHA ARE ALREADY DETERMINED
C                      EXCEPT FOR THE POWER OF ALPHA WHICH
C                      IS DETERMINED BY IN INDEX AND J
C
C     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)
C
C     LIM1 = INTEGER P
C
C
C
C     BETA TERMS
C
      MARKB = LIMA+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 = 1,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)
   65 AA(IND1) = AA(IND2)*PM1
   66 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
C     5 TH ORDER TERMS
C
      IND = 2*LIMIT(4)+1
      CALL CROSS(LS,R,S,AA,BB,IND,2,2,0,0)
      GO TO 150
C
C      6 TH ORDER TERMS
  105 CONTINUE
      IND = 2*LIMIT(5)+1
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,3,4)
      GO TO 150

C
C      7 TH ORDER TERMS
C
  110 CONTINUE
      IND = 2*LIMIT(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
C      8 TH ORDER TERMS
C
  115 CONTINUE
      IND = 2*LIMIT(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,4,1,5,8)
```

```
      CALL CROSS(LS,R,S,AA,BB,IND,2,2,3,4)

      GO TO 150
C
C     9 TH ORDER TERMS
  120 CONTINUE
      IND = 2*LIMIT(8)+1
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,21,40)
      CALL CROSS(LS,R,S,AA,BB,IND,3,1,10,18)
      CALL CROSS(LS,R,S,AA,BB,IND,4,1,10,18)
      DO 121 K = 5,8
  121 CALL CROSS(LS,R,S,AA,BB,IND,K,1,K,8)
      CALL CROSS(LS,R,S,AA,BB,IND,9,1,5,8)
      CALL CROSS(LS,R,S,AA,BB,IND,2,4,0,0)
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,46,48)
      GO TO 150
C
C    10 TH ORDER TERMS
C
  125 CONTINUE
      IND = 2*LIMIT(9) + 1
C
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,49,96)
      CALL CROSS(LS,R,S,AA,BB,IND,3,1,21,40)
      CALL CROSS(LS,R,S,AA,BB,IND,4,1,21,40)
      DO 126 K = 5,9
  126 CALL CROSS(LS,R,S,AA,BB,IND,K,1,10,18)
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,106,113)
      CALL CROSS(LS,R,S,AA,BB,IND,2,3,3,4)
      CALL CROSS(LS,R,S,AA,BB,IND,3,3,0,0)
      CALL CROSS(LS,R,S,AA,BB,IND,4,3,0,0)
      CALL CROSS(LS,R,S,AA,BB,IND,4,2,3,3)
      CALL CROSS(LS,R,S,AA,BB,IND,3,2,4,4)
C
      GO TO 150
C
  130 CONTINUE
C
C     11TH ORDER TERMS
C
      IND = 2*LIMIT(10) + 1
C
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,116,230)
      CALL CROSS(LS,R,S,AA,BB,IND,3,1,49,96)
      CALL CROSS(LS,R,S,AA,BB,IND,4,1,49,96)
      DO 131  K = 5,8
  131 CALL CROSS(LS,R,S,AA,BB,IND,K,1,21,40)
      DO 132  K = 10,18
  132 CALL CROSS(LS,R,S,AA,BB,IND,K,1,K,18)
      CALL CROSS(LS,R,S,AA,BB,IND,2,2,21,40)
      CALL CROSS(LS,R,S,AA,BB,IND,19,1,10,18)
```

```
      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)
C
      GO TO 150
C
C     12TH ORDER TERMS
C
  135 CONTINUE
C
      IND = 2*LIMIT(11) + 1
C
C
      CALL CROSS(LS,R,S,AA,BB,IND,2,1,287,572)
      CALL CROSS(LS,R,S,AA,BB,IND,3,1,116,230)
      CALL CROSS(LS,R,S,AA,BB,IND,4,1,116,230)
      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)
  150 CONTINUE
C
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)
```

```
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
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 155
      PRINT 501,JJ1,JJ2,SUM1,SUM3
  155 CONTINUE
C
C
C     EVALUATES T.E.C. FOR S(I,J) TERMS WITH ALPHA AS
C     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
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,
     1    TERM3)
      INTEGER TERM1,TERM2,TERM3,POWER,R
      DOUBLE PRECISION S(LS,R),AA(LS),BB(LS),FACT(7)
C
      DATA FACT/1.0D0,2.0D0,6.0D0,24.0D0,
     1    120.0D0,720.0D0,5040.0D0/
C
C
C
      IF (TERM2 .EQ. 0)  GO TO 20
C
C     COMPUTES S(INDEX+J,I)=S(TERM1,I)**POWER * S(TERM2+J,I)
C              FOR  J=0,1,...,TERM3-TERM2
C
      KK = -1
      DO 10 K = TERM2,TERM3
      KK = KK + 1
      INDX = INDEX + KK
      AA(INDX) = AA(TERM1)**POWER * AA(K)
      IPOW = POWER
      IF (TERM1 .EQ. K)  IPOW = IPOW + 1
      BB(INDX) = BB(TERM1)**POWER * BB(K) *FACT(IPOW)
      DO 10 I = 2,R
   10 S(INDX,I) = S(TERM1,I)**POWER * S(K,I)
      INDEX = INDEX + TERM3 - TERM2 + 1
C
C
      RETURN
C
C     COMPUTES S(INDEX,I) = S(TERM1,I)**POWER
C
   20 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
C
C
      RETURN
      END
```