**UNIVERSAL FINITE ELEMENT MATRICES**
**FOR TET3AHED3A**

by

Z.J. Cand^s, ?.U. MinJvis i ?.r>. i-.l/.;:

ORC-i-j-õJ-32

# UNIVERSAL FINITE ELEMENT MATRICES

# FOR TETRAHEDRA

Z. J. Cend*s
Camaegie-Mellon University
Electrical Engineering Dept.
Pittsburgh, PA 15213

F. U. Minhas
Dominion Engineering
Company Limited
Montreal, P.Q.

P. P. Silvester
Electrical Engng.
McGill University
Montreal, P.O.

## ABSTRACT

Methods are described for forming element matrices for a wide variety of operators on tetrahedral finite elements, in a manner similar to that previously employed for line segments and triangles. This technique models the differentiation and product-embedding operators as rectangular matrices, and produces finite element matrices by replacing all required analytic operations by their finite matrix analogues. The method is illustrated by deriving the conventional matrix representation for Laplace*s equation. Brief computer programs are given, which generate universal finite element matrices for use in various applications.

## 1. Introduction.

The finite element analyst has traditionally had two choices for evaluating the matrix elements required for any given finite element model. In one approach, advocated by Zienkiewicz [1], Irons, and others, the matrix elements are evaluated numerically as and when required, using quadrature formulae to compute the necessary integrals. The second approach, first employed by Silvester [2,3]| is to evaluate the matrix elements analytically in terms of parametric factors for a representative element. The precomputed matrix values are then combined in weighted sums to form the overall finite element matrix.

Both of the accepted procedures have advantages and disadvantages. The numerical integration approach is simple, and easy to implement; but it gives rise to high computing costs and sometimes to poor accuracy. Analytic integration is much less costly, but requires precomputing and storing many different numeric matrices for the various differential operators and energy functionals encountered in applications, and their associated functionals.

In recent years a third approach, variously called an "elementary matrix" or "universal matrix" approach, has been developed [4-8]* In this approach, exact numeric representations are developed for certain elementary operators, such as the differentiation operator. Finite element matrices are then generated in specific cases as parametrized combinations of the universal matrices. This third approach shares the precision advantages of the precomputed matrix technique, since all necessary differentiations and integrations are performed analytically, not numerically. Yet it shares much of the numerical integration approach, because the elementary matrices are few and are combined in simple ways. Not surprisingly, the computing time demands of the new method lie between those of the two classical techniques.

In the majority of applications, it is found that at most four elementary matrices suffice to model problems involving arbitrary linear differential operators. For practical use, one has the choice of either tabulating these matrices, or of giving programs capable of generating them as needed. The usual course in the past has been to tabulate the matrices, preferably in the form of integer quotients; for only in that form is full precision preserved. In the present work, the alternative approach is taken: short computer programs are presented which generate the elementary matrices in floating-point form. The disadvantage of finite machine-dependent precision is avoided by employing the same computer, or a computer of at least the same precision, for both the elementary matrix generation and subsequent finite element problem solving.

## 2. Interpolation Polynomials on Tetrahedra

Interpolation polynomials of the closed Newton-Cotes type are commonly used on triangular and tetrahedral elements in field analysis. To set up these polynomials in a convenient form, let

$$\zeta_1 = \begin{vmatrix} x & y & z & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \Bigg/ \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \qquad (1)$$

denote one of the homogeneous (volume) coordinates [9] on a tetrahedron; the remaining three are defined similarly by cyclic interchange of subscripts. Silvester [2] has defined a family of semi-interpolative (one-sidedly interpolative) polynomials by

$$P_m(z) = \prod_{i=1}^{m} \frac{Nz-i+1}{i}, \quad m \geq 1 \qquad (2)$$

- 1 , ~ - o

These are serai-interpolative because they possess zeros at z s (i-D/N, for is 1, ..., m. They are very convenient for defining the set of Lagrangian interpolation polynomials on a tetrahedron, with interpolation nodes of the closed Newton-Cotes pattern. The latter are given by

$$\alpha_{ijkl} = P_i(\zeta_1)\,P_j(\zeta_2)\,P_k(\zeta_3)\,P_l(\zeta_4) \qquad (3)$$

subject to the requirement that i + j + k+ +1 s $N_9$ where N is the degree of the desired polynomial [10]. On a tetrahedron there are M(N) = (N+1)(N+2)(N+3)/6 such nodes and corresponding polynomials. The quadruple index ijkl identifies the polynomial associated with each interpolation node clearly. However, in most applications it is preferable to use single indices to identify the polynomials, so as to avoid cluttering expressions with long subscript strings. In principle, the quadruple indices may be mapped onto single indices in any consistent fashion. In practice, the mapping is usually accomplished by regarding each quadruple index as a four-digit integer, and taking these in descending order.

## 3. The Differentiation Operator

The directional derivative of a polynomial finite element approximation is best expressed in a tetrahedral element by writing the derivative in terms of interpolation polynomials. Consider for example a potential function u, given in a tetrahedron as a polynomial of degree N in the space coordinates,

$$u = \sum_{i=1}^{M(N)} u_i \alpha_i^{(N)}(x,y,z) \tag{4}$$

and suppose that its directional derivative is desired in some direction, say s. If the* interpolation polynomials used for approximating are of degree N, this derivative is clearly a polynomial of degree at most N-1. Thus, one may write

$$\frac{\partial u}{\partial s} = \sum^{M(N)} u_i \sum^{4} \frac{\partial \alpha_i^{(N)}}{\partial \zeta_j} \frac{\partial \zeta_j}{\partial s} \tag{5}$$

where the chain rule of differentiation has been used to move the operation of differentiation from the space direction s to the tetrahedron coordinates* But since the derivative is a polynomial of degree N-1, it may be expressed exactly in terms of the interpolation polynomials of degree N-1:

$$\frac{\partial u}{\partial s} = \sum_{k=1}^{M(N-1)} d_k \alpha_k^{(N*1)} \qquad \text{<*>}$$

The coefficients in eqn. (6) are most easily determined by equating right-hand sides of eqns. (5) and (6), and observing that the summation of eqn. (6) collapses to a single term if evaluated at an interpolation node, say node k, of the family of interpolation polynomials of degree N-1:

$$d_k = \sum_{i=1}^{M(N)} u_i \sum_{j=1}^{4} \frac{\partial \zeta_j}{\partial s} \left[\frac{\partial \alpha_i^{(N)}}{\partial \zeta_j}\right]_{P_k} \qquad (7)$$

Let four purely numeric matrices $D^{(j)}$ be defined by

$$D_{ki}^{(j)} = \frac{\partial \alpha_i^{(N)}}{\partial \zeta_j}\Bigg|_{P^{((}} \qquad (8)$$

These matrices are pure numerics, independent of the size and shape of the tetrahedron. In terms of these matrices, eqn. (7) may be written in the form

$$d = \left( \sum_{j=1}^{4} \frac{\partial \zeta_j}{\partial s} D \right. \qquad (9)$$

It should be observed that although there are in principle four distinct coefficient matrices D, the very nature of homogeneous coordinates dictates that they must be row and column permutations of each other. Thus, tabulation and calculation of only one matrix suffices*

The directional differentiation operator may be regarded as a mapping between the space spanned by the interpolation polynomials of degree N, and the space spanned by those of degree N-1. These spaces are of dimensionality $M(N) = (N+1)(N+2)(N+3)/6$ and $M(N-1) = (N)(N+1)(N+2)/6$, respectively. One possible representation of the finite directional differentiation operator is therefore a rectangular matrix with M(N) columns but only M(N-1) rows. This representation is advantageous in many applications because of its compactness, as well as because the matrices are guaranteed to have full row rank. However, if directional derivative values are desired, this representation suffers from the shortcoming that the values are obtained on an interpolation node set different from that used for the function values. In this circumstance, it is more convenient to express the derivatives in terms of polynomials of degree N. Thus, one may replace eqn. (6) by

$$\frac{\partial u}{\partial s} = \sum_{k=1}^{M(N)} \bar{d}_k \alpha_k^{(N)} \tag{10}$$

This equation is exact, since the directional derivative is a polynomial of degree N-1, and may therefore be expressed in terms of the polynomials of degree N. In this case, the equation corresponding to (9) becomes

$$\bar{d} = \left( \sum_{j=1}^{4} \frac{\partial \zeta_j}{\partial s} \bar{D}^{(j)} \right) u \tag{11}$$

where

$$\bar{D}^{(j)}_{ki} = \left.\frac{\partial \alpha_i^{(N)}}{\partial \zeta_j}\right|_{P_k^{(N)}} \tag{12}$$

the derivatives being evaluated at the interpolation nodes of the set of degree N, not N-1.

Again, the four numeric matrices D are row and column permutations of each other, so that only one needs to be calculated and stored. However, this matrix is square, having M(N) rows and columns. Of course, it has a row nullspace of dimensionality M(N) - M(N-1), and rank M(N-1).

## 3. The Metric Matrices

A matrix frequently required in finite element analysis is the metric of the interpolation polynomials in each element. This matrix is occasionally also termed the "mass matrix" by analysts whose background is rooted in elasticity theory or structural analysis. Given the set of interpolation polynomials of degree N, the metric T is defined as the matrix whose elements are given by

^f*A V CM) L*A
TV = \ *i «i dΩ (13)

Here and in the following, it is assumed that the tetrahedral element has unit volume; for any other element, T must be multiplied by the element volume. There will of course be a distinct metric, of order M(N), for each order of tetrahedral element; orders will be distinguished by superscripts parentheses, as above. Metrics for the first few orders of tetrahedra have been published [3] in the form of integer quotients, so that the first few are known exactly.

An interesting point to observe is that the sequence of metrics T for the various orders of tetrahedron is not independent. Since the interpolation polynomials (3) of the various orders are all complete in the sense of Dunne [11], the family of polynomials of any given order must embed all polynomial families of all lower orders. Consequently, the metric of any given order must also embed, in some sense, the metrics of all lower orders. Just exactly how, will become evident on brief examination of the manner in which the embeddings of the polynomials themselves can be represented.

## 4. Embedding Operators

Suppose that a certain polynomial p has an exact representation in terms of the interpolation polynomials of degree N, say

$$p = \sum_{i=1}^{M(N)} p_i \alpha_i^{(N)} \tag{14}$$

Then it must also have an exact representation in terms of the interpolation polynomials of degree N+1,

$$P = \sum_{j=1}^{M(N+1)} P_j^{(N+1)} \alpha_j^{(N+1)} \tag{15}$$

and it is interesting to enquire how the coefficients in eqn. (15) can be derived from those in eqn. (1U). To determine the necessary mapping, it suffices to equate the right sides of these two equations,

$$\sum_{j=1}^{M(N+1)} P_j^{(N+1)} \alpha_j^{(N+1)} = \sum_{i=1}^{M(N)} P_i^{(N)} \alpha_i^{(N)} \tag{16}$$

and evaluate both sides at interpolation node k of order N+1. Since the polynomials are interpolative, the left-hand summation collapses, leaving only a single surviving term:

$$P_k^{(N+1)} = \sum_{i=1}^{M(N)} P_i^{(N)} \alpha_i^{(N)} \Big|_{P_k^{(N+1)}} \tag{17}$$

Let a rectangular matrix, with M(N+1) rows and M(N) columns, be defined by

$$P_k^{(N+1)} \tag{18}$$

The mapping cf coefficient vectors between eqns. (1M) and (15) is then clearly given, in matrix form, by.

$$P^{(N+1)} = B\, P^{(N)} \tag{19}$$

The matrix B may be termed a finite embedding operator, or an embedding matrix, for it embeds the coefficients related to degree N in the next higher-order set.

While the matrix B could easily be computed and tabulated for various orders, it may be useful to consider another matrix, which is more general than B, but allows B to be derived easily. Consider again the polynomial of eqn. (14); but this time let it be multiplied by some quantity which varies linearly with one of the tetrahedron coordinates. This time,

$$p \zeta_\ell = \sum_{i=1}^{M(N)} p_i^{(N)} \zeta_\ell \alpha_i^{(N)} \tag{20}$$

is of interest, instead of eqn. (14). Equating and evaluating it at node k of the next higher order node set, as above, one is quickly led to define a matrix C by

$$C_{ki}^{(\ell)} = \left[ \zeta_\ell \alpha_i^{(N)} \right]_{P_k^{(N+1)}} \tag{21}$$

Once again there exist four matrices C, one corresponding to weighting p with respect to each tetrahedron coordinate; the appropriate coordinate is identified by the bracketed subscript. The four matrices C are again row and column permutations of each other, so that there is no need to compute mere than one of them.

Since the tetrahedron coordinates must add to exactly unity in any tetrahedron, the matrix B must be given by the sum of the matrices C:

$$B_{ki} = \sum_{\ell=1}^{4} C_{ki}^{(\ell)} \qquad (22)$$

The matrices C provide a more general product embedding operation than does the matrix B. Yet the cost of computing them is virtually the same. Hence the computer programs given in the Appendix calculate and tabulate the matrices $C_f$ rather than B.

## 5. Metrics and Projectors

An interesting special case of embeddings arises when the polynomial p of eqn. (14) is in fact one cf the interpolation polynomials of degree N. In this case the right-hand coefficient vector in eqn. (19) becomes one column of the unit matrix, and

$$\hat{\alpha}^{(N)} = \sum B_{iu} \alpha_i^{(N+1)} \qquad (23)$$

This property is very useful in evaluating projection matrices. Csendes [1] shows that the best approximation to a polynomial of degree N in a subspace spanned by polynomials of degree N-1 is obtained by application of the projector

$$P^{(N)} = \left[T^{(N-1)}\right]^{-1} A^{(N)} \qquad (24)$$

where

$$A_{ij}^{(N)} = \int \alpha_i^{(N-1)} \alpha_j^{(N)} \text{ aa} \tag{25}$$

These matrices are easily evaluated. Substituting eqn. (19) into (25), there immediately results

$$A^{(N)} = [B^{(N-1)}]' T^{(N)} \tag{26}$$

where the prime denotes transposition. A separate evaluation of eqn. (25) from first principles, by actual integration, is *never* required. In a comparable fashion, one easily derives

$$T^{(N-1)} = [B^{(N-1)}]' T^{(N)} B^{(N-1)} \tag{27}$$

This equation indicates that, at least in principle, there is no need for programs to calculate metrics of all orders. If the metric of the highest order element to be employed is known, then the metrics of all lower orders can be derivable by successive applications of the embedding operator. The projector of eqn. (2M) may thus be written in the alternative form

$$P^{(N)} = [(B^{(N-1)})' T^{(N)} B^{(N-1)}]^{-1} B^{(N-1)} T^{(N)} \tag{28}$$

It might be observed in passing that the two forms of differentiation operators, rectangular and square, are also related to each other by an embedding operation:

$$\overline{D}^{(N)(j)} = B^{(N-1)} D^{(} \tag{29}$$

Thus there is no fundamental need to possess both types of differentiation matrices, although it may at times be convenient to do so.

## 6. The Dirichlet Matrix

The Dirichlet matrix is very commonly encountered in finite element analysis of potential field problems, and will be employed to illustrate the use of the universal matrices described here. On a tetrahedral element of unit volume, the Dirichlet matrix is given by

$$S_{ij} = \int \nabla\alpha_i^{(N)} \cdot \nabla\alpha_j^{(N)} \, d\Omega \tag{30}$$

Written out in detail, this equation reads

$$\int \nabla\alpha_i \cdot \nabla\alpha_j \, d\Omega = \sum_{m=1}^{4} \cdots \frac{\partial x}{\partial \zeta} \cdots + \frac{\partial z}{\partial \zeta_m}\frac{\partial z}{\partial \zeta_n}\Big) \int \frac{\partial \alpha_i}{\partial \zeta_m}\frac{\partial \alpha_j}{\partial \zeta_n} \, d\Omega$$

The crucial quantity is obviously the integrand on the right-hand side; the term in parentheses is simply a geometric constant that expresses the relationship of the four homogeneous coordinate directions to the three Cartesian axes. Using the relationships above, however, this integrand is readily written as

$$\frac{\partial \alpha_i}{\partial \zeta_m}\frac{\partial \alpha_j}{\partial \zeta_n} = \sum_{k=1}^{M(N-1)} \sum_{l=1}^{M(N-1)} \left[D_{ik}^{(j)}\right]^{t} T^{(N-1)} D_{jl}^{(j)} \tag{32}$$

in terms of the rectangular differentiation matrices; or as an analogous expression in terms of the square differentiation matrices.

## 7. Conclusions

To derive finite element matrices for tetrahedral elements, using the conventional tetrahedron interpolation polynomials, it suffices to possess the following primitive matrices: (1) a finite differentiation operator, (2) the metric of the interpolation polynomial basis, (3) an embedding operator that maps low-crder polynomials to a representation one order higher, (4) a projection operator that projects polynomials onto a space one order lower. The differentiation operator may be expressed in two different ways, each of which has advantages in certain applications.

The projection operators, and the two forms of the differentiation operator, may be derived easily from the first three primitive matrices above by simple matrix manipulations. Further, and mere importantly, finite element matrix representations of many linear operators may be constructed from three primitives: (1) the rectangular differentiation matrix of order N, (2) the metric of order N, and (3) the embedding between orders N-1 and N. Computation of these matrices is relatively straightforward, and programs for doing so are given in the Appendix.

## 8. References

[1] Zienkiewicz, O. C., The Finite Element Method in Engineering Science. New York: McGraw-Hill, 1971.

[2] Silvester, P., High-order polynomial triangular finite elements for potential problems. Int. Jour. Engrng. Science, vol. 7, 1969, pp. 849 - 861.

[3] Silvester, P., Tetrahedral polynomial finite elements for the Helmholtz equation. Internat. Jour. Numer. Meth. Engrg., vol. 4, 1972, pp. 405 - 413.

[4] Cendes, Z. J., A finite element method for the general solution of ordinary differential equations. Internat. Jour. Numer. Meth. Engrg., vol. 9, 1975, pp. 551 - 561.

[5] Silvester, P., and Haslam, C. R. S., Magnetotelluric modelling by the finite element method. Geophys. Prospect., vol. 20, 1972, pp. 872 - 891.

[6] Cendes, Z. J., A Fortran program to generate finite difference formulas. Int. Jour. Numer. Methods Eng., vol. 9, 1975, pp. 579 - 597.

[7] Kisak, E., Silvester, P., Telford, W. M., A recursive method in the E-polarization of magnetotelluric modelling by high-order finite elements. Acta Geodaet., Geophys., et Montanist. Acad. Sci. Hung., tomus 12, 1977, pp. 255 - 266.

[8] Silvester, P., Construction of triangular finite element universal matrices. Internat. Jour. Numer. Meth. Engrg., vol. 12, 1978, pp. 237 - 244.

[9] Maxwell, E. A., General homogeneous coordinates. Cambridge: University Press, 1960.

[10] Silvester, P., Symmetric quadrature formulae for simplexes. Maths. Comp., vol. 24, 1970, pp. 95 - 100.

[11] Dunne, P. C., Complete polynomial displacement fields for finite element method. Jour. Roy. Aeronaut. Soc.., vol. 72, 1968, pp. 245 - 246.

## 9. Appendix

The three elementary matrices described above are readily generated using the computer programs of this Appendix. The programs are written in near-standard Fortran, and are configured as input-output free subroutines. No file handling and no character handling is involved, so that there should be little trouble in compiling and running the programs at any computer installation.

There are three subroutines to generate the three matrices: DIFMTX, EMBMTX, and METRIC. These in turn call other routines. The second-level routines are:

ADERV1 returns the value of the directional derivative (in the direction toward vertex no. 1) for a specified interpolation function at a specified point in a tetrahedron;

AFUNCT computes the functional value of a specified tetrahedron interpolation function at a specified place in the tetrahedron, see eqn. (3) above;

FACTOR is the factorial function, in double precision, for integer arguments not exceeding 30.

PDERIV returns the first derivative of any one of the semi-interpolative polynomials of eqn. (2) above;

PFUNCT returns values of the semi-interpolative polynomials of eqn. (2);

PRECIS finds the machine precision, i.e. the smallest number s such that (1 + s) is distinguishable from 1.

PSYMBL creates an array of coefficients of the various powers of the argument, thus giving an analytic representation of the semi-interpolative polynomials of eqn. (2);

QUADRA generates the set of closed Newton-Cotes quadrature weight? for a tetrahedron, of degree $2N_f$ by calling WEIGHT;

WEIGHT computes the quadrature weight at a specified quadrature node.

The various routines are designed to be reasonably self-supporting, in the sense that they include a broad variety of error and consistency checks. All floating-point work is done in double precision — which of course will vary considerably from machine to machine and installation to installation. One measure of the precision achievable is the so-called "machine epsilon", the smallest number s such that (1 • s) is distinguishable from unity within the actual operating precision of the machine. This number is fixed for any given installation by the hardware and system software. However, users do not often know the value of this number; the present program suite therefore computes an approximation to it by a sequence of binary chops. The accuracy obtained is fully sufficient for present purposes. A need to know this number arises In several subroutines, where floating-point equality comparisons must be made.

The methods employed for finding the matrices D and C, which do not involve volume integration, are straightforward; the programs amount in essence to no more than computer implementations of eqns. (8) and (21). The method U3ed for the metric differs slightly from those described in earlier literature. Since the integrand in eqn. (13) is exactly polynomial, of degree not higher than 2N, it is known that it can be integrated exactly by a Newton-Cotes quadrature formula of order 2N [10]. Computation of T therefore proceeds in two stages. First, the quadrature weights for a closed Newton-Cotes formula of order 2N are calculated. Secondly, T is computed exactly as it is defined in eqn. (13), save of course that the integration is replaced by a numerical quadrature. It must be emphasized that no numerical approximation is involved here; the quadrature formula is specifically generated of high enough order to render it exact, except for roundoff error. The quadrature formula generating programs are designed to be essentially independent, so that users wishing to make use of these quadrature weights elsewhere may find it convenient to do so.

Program robustness and precision have been considered paramount in the design of the attached subroutines. However, little attention has been paid to memory requirements and to computing time, on the supposition that the elementary matrices will be generated ab initio only very occasionally.

Of the three matrices, T is the most sensitive to numeric stability. Using a 32-bit machine (64 bits in double precision), with a machine epsilon of 1.ME-17, it has been estimated that loss of precision in computation will not exceed 3 decimal figures for sixth-order tetrahedra, i.e, that the results should contain mantissas good to at least 13 - 14 decimal figures. Accuracy deteriorates for higher element orders. But it is rather doubtful that seventh or higher order tetrahedra will find extensive application, since elements with 120 or more nodes are computationally unwieldy!

Computing times rise very rapidly with element order, particularly since the programs do not very seriously attempt to take advantage of subscript symmetries or other possible economies. Should computing times be a factor of importance, the running times of the T matrix routines in particular can probably be reduced by a factor of ten, or more, by clever exploitation of the many symmetries possessed by this matrix. Time requirements were not considered a major issue in program design, because it is likely that the matrix generation programs will be used only a very few times at any one computer installation. The programs as given here were developed and verified on a PDP-11/03 computer with the RT-11 operating system. On this small machine, computation of the matrices for first through fourth orders takes about two hours; of course, only a few minutes are required on a large main-frame machine.

To illustrate the use of this subroutine package, three small driver programs are appended to the subroutines. These read the desired value N of matrix order, call the relevant subroutines, and write out the resulting matrices to the user terminal. While the subroutine package is written to be machine-independent, the driver programs will need modification at every installation, because input-output arrangements invariably differ. However, since these programs only contain about a dozen active Fortran lines each, users should experience no difficulty in adapting them, or providing locally acceptable equivalents.

# MATRIX CENERATOR SUBROUTINES

## Fortran Listings

Z. J. Csendes, F. U. Minhas, P. P. Silvester

July 1980

```
C
C     «ftftftft«ftft»ftXtt«»»ft*»ftftft»«ftft»*ft««»»ftft««*««ft»»ft*««*«*»»»»««»
C
      DOUBLE PRECISION FUNCTION ADERVKUKL, ZETA, IERR)
C
C     «ft««»ftft»ft*ft»»«»»«ft««»ft»ftft«tt««»««»ft««ft»«ft«ftftftft»»ftft»«ft»»ft»
C
C          RETURNS DERIVATIVE OF THE INTERPOLATION FUNCTION OF
C          ORDER N, ALPHA(I,J,K,L), AT THE INTERIOR POINT IN A
C          A TETRAHEDRON GIVEN BY THE ARRAY ZETA.   IERR IS AN
C          ERROR INDICATOR WHICH CARRIES THROUGH THE VALUE OF
C          IERR AS SET BY 'PFUNCT' OR 'PDERIV.  IF ANY ONE OF
C          THE INTEGERS IN IJKL IS NEGATIVE AN ERROR EXIT WITH
C          ARGUMENT IERR SET TO  31,  32, 33, 3*», RESPECTIVELY,
C          IS EFFECTED.  IERR = 35 SIGNIFIES THAT THE FOUR CO-
C          ORDINATES ZETA DID NOT ADD UP TO UNITY.
C
      DOUBLE PRECISION ADERV1, PFUNCT, PDERIV
      DOUBLE PRECISION ZETA, EPSLON, Z
      DIMENSION ZETAU), IJKL(U)
C
C     EPSLON IS A MACHINE-DEPENDENT PRECISION INDICATOR -
      COMMON /PRECSN/ EPSLON
C
C     IS THE ARGUMENT SET ACCEPTABLE?  EXIT IF NOT.
      IERR = 0
      IF (IJKL(D.LT.O) IERR = 31
      IF (IJKL(2).LT.O) IERR = 32
      IF (IJKL(3).LT.O) IERR = 33
      IF (IJKL(U).LT.O) IERR = 3*»
      ADERV1 = -1.0D0
      DO 10 11=1,U
        ADERV1 = ADERV1 • ZETA(II)
   10 CONTINUE
      IF (ADERV1.GT.EPSLON .OR. ADERV1.LT.-EPSLON) IERR = 35
      IF (IERR.NE.O) GO TO MO
C
C     GET STARTED.  SET ORDER N.
      N = 0
      DO 20 11=1,4
        N = N + IJKL(II)
   20 CONTINUE
C
C     COMPUTE ALPHA DERIVATIVE IN 1 - DIRECTION.
      ADERV1 = PDERIV(ZETA(1),IJKL(1),N,IERR)
      IF (IERR.NE.O) GO TO 10
```

```
      DO 30 11=2,4
        IDX = IJKL(II)
        Z = ZETA(II)
        ADERV1 = ADERV1*PFUNCT(Z,IDX,N,IERR)
        IF (IERR.NE.O) GO TO 40
   30 CONTINUE
C
   40 RETURN
      END
```

```
C
C      «»ftttft*ft«*ft»»ftft»ttft»ftftft»»»»ft»»»»»*«**»«»»ftft*»»»«»»ftft*»»ft*ft
C
      DOUBLE PRECISION FUNCTION AFUNCTUJKL, ZETA, IERR)
C
Q      «*«ft»»ft»ft»«»»ftft»ft»*ft»«ftft*«««tt»ft»»ftttft«*»**»ftft»Sft«ft»ft»»»*»
C
C          RETURNS THE VALUE OF THE INTERPOLATION FUNCTION OF
C         ORDER N, ALPHA(I,J,K,L), AT THE INTERIOR POINT IN A
C          A TETRAHEDRON GIVEN BY THE ARRAY ZETA.   IERR IS AN
C          ERROR INDICATOR WHICH CARRIES THROUGH THE VALUE OF
C          IERR AS SET BY •PFUNCT' OR •PDERIV.  IF ANY ONE OF
C          THE INTEGERS IN IJKL IS NEGATIVE AN ERROR EXIT WITH
C          ARGUMENT IERR SET TO  21, 22, 23, 24, RESPECTIVELY,
C          IS EFFECTED.  IERR = 25 SIGNIFIES THAT THE FOUR CO-
C          ORDINATES ZETA DID NOT ADD UP TO UNITY.
C
      'DOUBLE PRECISION AFUNCT, PFUNCT
      DOUBLE PRECISION ZETA, EPSLON, Z
      DIMENSION ZETAC4), IJKL(4)
C
C     EPSLON IS A MACHINE-DEPENDENT PRECISION INDICATOR -
      COMMON /PRECSN/ EPSLON
C
C     IS THE ARGUMENT SET ACCEPTABLE?  EXIT IF NOT.
      IERR = 0
      IF (IJKLO).LT.O) IERR = 21
      IF (IJKL(2).LT.O) IERR = 22
      IF (IJKL(3).LT.O) IERR = 23
      IF (IJKL(4).LT.O) IERR s 24
      AFUNCT s -1.ODO
      DO 10 IIs1,4
        AFUNCT = AFUNCT + ZETA(II)
   10 CONTINUE
      IF (AFUNCT.GT.EPSLON .OR. AFUNCT.LT.-EPSLON) IERR = 25
      IF (IERR.NE.O) GO TO 40
C
C     GET STARTED.  SET ORDER N.
```

```
      N = 0
      DO 20 II=1,4
        N = N + IJKL(II)
   20 CONTINUE
C
C     COMPUTE ALPHA-FUNCTION
      AFUNCT = 1.0D+0
      DO 30 II=1,4
        IDX = IJKL(II)
        Z = ZETA(II)
        AFUNCT = AFUNCT*PFUNCT(Z,IDX,N,IERR)
        IF (IERR.NE.0) GO TO 40
   30 CONTINUE
C
   40 RETURN
      END
```

```
C
C     ****************************************************
C
      SUBROUTINE DIFMTX(N, D1, NI, NJ, IERR)
C
C     ****************************************************
C
C          RETURNS THE DIFFERENTIATION MATRIX D1 OF ORDER N
C          COMPUTED IN DOUBLE PRECISION.
C
C          THE ARGUMENTS NI, NJ ARE MATRIX DIMENSIONS.  THEY
C          MUST BE AT LEAST NI =  (N)(N+1)(N+2)/6   AND
C                           NJ = (N+1)(N+2)(N+3)/6
C          OTHERWISE IERR = 51 IS RETURNED, AND NO OTHER AC-
C          TION IS TAKEN.  OTHER ERROR RETURNS TRACE WHERE
C          THE ERROR OCCURRED, BY SIMPLY PASSING THROUGH THE
C          ERROR-INDICATOR VALUES FROM OTHER ROUTINES.
C
C          SUBROUTINE CALLING STRUCTURE:
C
C          DIFMTX  CALLS  PRECIS
C                  CALLS  ADERV1  CALLS  PDERIV
C                                 CALLS  PFUNCT
C
      DOUBLE PRECISION D1(NI,NJ), ZETA(4), ADERV1
      DIMENSION JARR(4), IARR(4)
C
C     EPSLON IS A MACHINE PRECISION INDICATOR, FOR SET-
C     TING TOLERANCES.  IT IS TAKEN AS FOUR TIMES THE
C     LEAST DEVIATION DISTINGUISHABLE FROM UNITY.
C
```

```
      DOUBLE PRECISION EPSLON
      COMMON /PRECSN/ EPSLON
C
C     START BY SETTING EPSLON
      CALL PRECIS
      EPSLON = 4.D+0«EPSL0N
C
C     CHECK DIMENSIONS IN CASE OF ERROR.
      IF (NI.GE.N*(N+1)«(N+2)/6 .AND. NJ.GE.(N+1)«(N+2)*(N+3)/6) GO TO
     * 10
      IERR = 51
      GO TO 140
   10 CONTINUE
C
C     OUTER LOOP:  GENERATE THE INDEX STRING IARR FOR
C     QUADRUPLE INDICES OF ORDER N-1.  IC IS THE COR-
C     RESPONDING SINGLE INDEX.
C
      IC = 0
      DO 130 J1=1,N
        IARR(1) = N - J1
        M2 = N - IARR(I)
        DO 120 J2=1,M2
          IARRC2) = M2 - J2
          M3 = M2 - IARRC2)
          DO 110 J3=1,M3
            IARR(3) = M3 - J3
            IARR(U) = N - 1
            DO 20 J=1,3
              IARR(4) = IARR(M) - IARR(J)
   20       CONTINUE
            IC = IC + 1
C
C     INNER LOOP:  GENERATE THE INDEX STRING JARR FOR
C     QUADRUPLE INDICES OF ORDER N.  JC IS THE CORRES-
C     PONDING SINGLE INDEX.
C
            JC = 0
            N1 = N + 1
            DO 100 11=1,N1
              JARR(1) = N1 - 11
              N2 = N1 - JARR(1)
              DO 90 12=1,N2
                JARRC2) = N2 - 12
                N3 = N2 - JARR(2)
                DO 80 I3=1,N3
                  JARR(3) = N3 - 13
                  JARR(U) = N
                  DO 30 J=1,3
                    JARR(M) = JARRU) - JARR(J)
   30             CONTINUE
                  JC s JC + 1
```

```
C
C     BOTH INDEX STRINGS ARE NOW IN HAND.  COMPUTE
C     THE COORDINATE VALUES ZETA, AT THE NODE OF OR-
C     DER N-1, AND FIND THE D1 ENTRY AT (IC,JC).
C
                    IF (N.GT.1) GO TO 50
                    DO MO J=1,M
                      ZETA(J) = 0.25D+0
  MO                CONTINUE
                    GO TO 70
C
  50                CONTINUE
                    DO 60 Jsi,H
                      ZETA(J) = IARR(J)
                      ZETA(J) s ZETA(J)/(N-1)
  60                CONTINUE
C
  70                CONTINUE
                    DHIC.JC) = ADERV1(JARR,ZETA,IERR)
C
  80              CONTINUE
  90            CONTINUE
 100          CONTINUE
C
 110        CONTINUE
 120      CONTINUE
 130    CONTINUE
C
 1M0    RETURN
        END



C
C     ft«»*»ft*««ftftftft»»«»»ftftftftftft«»»«ftftftftsft««ftft««ft»»»*ft»»ftftft«»
C
      SUBROUTINE EMBMTXCN, C1, NI, NJ, IERR)
C
C     •ftft»ftftftftft»ftft»«ftft»ftftft»»ft»ftft«ftft»»ft»ftftft«ft»ftftft»«ft»ftftftft»ftft
C
C           RETURNS THE DIFFERENTIATION MATRIX C1 OF ORDER N
C           COMPUTED IN DOUBLE PRECISION.
C
C           THE ARGUMENTS NI, NJ ARE MATRIX DIMENSIONS.  THEY
C           MUST BE AT LEAST NI = (N+2)(N+3)(N+U)/6   AND
C                            NJ = (N+1)(N+2)(N+3)/6
C           OTHERWISE IERR = 61 IS RETURNED, AND NO OTHER AC-
C           TION IS TAKEN.  OTHER ERROR RETURNS TRACE WHERE
C           THE ERROR OCCURRED, BY SIMPLY PASSING THROUGH THE
C           ERROR-INDICATOR VALUES FROM OTHER ROUTINES.
C
```

```
C
C          SUBROUTINE CALLING STRUCTURE:
C
C          EMBMTX  CALLS  AFUNCT  CALLS  PFUNCT
C                  CALLS  PRECIS
C
      DOUBLE PRECISION C1(NI,NJ), ZETAU), AFUNCT
      DIMENSION JARR(U), IARR(U)
      DOUBLE PRECISION EPSLON
      COMMON /PRECSN/ EPSLON
C
C     SET EPSLON TO START.  ALLOW 4 TIMES EPSLON
C     AS THE MARGIN FOR FLOATING-POINT CALCULATION.
      CALL PRECIS
      EPSLON s M.OD+0»EPSLON
C
C     CHECK DIMENSIONS IN CASE OF ERROR.
      IF (NI.GE.N«(N+1)*(N+2)/6 .AND. NJ.GE.(N+1)»(N+2)*(N+3)/6) GO TO
     • 10
      IERR = 61
      GO TO 110
   10 CONTINUE
C
C     OUTER LOOP:  GENERATE THE INDEX STRING IARR FOR
C     QUADRUPLE INDICES OF ORDER N-1.  IC IS THE COR-
C     RESPONDING SINGLE INDEX.
C
      IC = 0
      M1 = N + 2
      DO 100 J1s1,M1
        IARR(1) = M1 - J1
        M2 = M1 - IARR(1)
        DO 90 J2=1,M2
          IARR(2) = M2 - J2
          M3 = M2 - IARR(2)
          DO 80 J3=1,M3
            IARRC3) = M3 - J3
            IARR(U) = N + 1
            DO 20 J=1,3
              IARR(M) = IARR(U) - IARR(J)
   20       CONTINUE
            IC s IC • 1
C
C     INNER LOOP:  GENERATE THE INDEX STRING JARR FOR
C     QUADRUPLE INDICES OF-ORDER N.  JC IS THE CORRES-
C     PONDING SINGLE INDEX.
C
            JC = 0
            N1 = N + 1
            DO 70 11=1,N1
              JARR(1) = N1 - 11
              N2 = N1 - JARRC1)
```

```
                DO 60 12=1,N2
                  JARRC2) = N2 - 12
                  N3 = N2 - JARRC2)
                  DO 50 13=1,N3
                    JARR(3) = N3 - 13
                    JARR(4) = N
                    DO 30 J=1,3
                      JARR(4) = JARR(4) - JARR(J)
 30                 CONTINUE
                    JC = JC + 1
C
C     BOTH INDEX STRINGS ARE NOW IN HAND.  COMPUTE
C     THE COORDINATE VALUES ZETA, AT THE NODE OF OR-
C     DER N-1, AND FIND THE C1 ENTRY AT (IC,JC).
C
                    DO 40 J=1,4
                      ZETA(J) = IARR(J)
                      ZETA(J) = ZETA(J)/(N+1)
 40                 CONTINUE
C
                    CONTINUE
                    C1(IC,JC) s ZETA(1)*AFUNCT(JARR,ZETA,IERR)
C
 50               CONTINUE
 60             CONTINUE
 70           CONTINUE
C
 80         CONTINUE
 90       CONTINUE
100     CONTINUE
C
110     RETURN
        END
```

```
C
C       *«ft*»ft««ft*»*»»*ttft»ftftft»ft»*»ft*»»ft»ftft*ftt««ft»ft»»«»«ft«»»«*»tt»ft«
C
      DOUBLE PRECISION FUNCTION FACTOR(N, IERR)
C

C
C          RETURNS THE DOUBLE-PRECISION FACTORIAL OF THE INTEGER
C          N.  IERR IS SET TO ZERO IF ALL IS WELL; IF N IS NEGA-
C          TIVE, IERR IS RETURNED AS 75.  IF N IS LARGE ENOUGH
C          FOR TRAILING SIGNIFICANT FIGURES TO BE LOST, IERR IS
C          SET TO -76.  IF N EXCEEDS 30, IERR IS SET TO 77.  IF
C          IERR IS POSITIVE, NO CALCULATION IS CARRIED OUT;  IF
C          IERR IS NONPOSITIVE, THE FACTORIAL IS COMPUTED.
C
      DOUBLE PRECISION FACTOR, EPSLON
      COMMON /PRECSN/ EPSLON
C
C     N NONNEGATIVE?  ERROR IF NOT!
      IERR = 0
      IF (N.LT.O) IERR = 75
      IF (N.GT.30) IERR s 77
      IF (IERR.NE.O) GO TO 20
C
C     OK, CALCULATE
      FACTOR = 1.D0
      IF (N.EQ.O) GO TO 20
C
      DO 10 Is1,N
        FACTOR = FACTOR*I
        IF (FACTOR*EPSLON.GT.1.DO) IERR = -76
   10 CONTINUE
C
C     EXIT
   20 RETURN
      END
```

```
C
C
C
      SUBROUTINE METRIC(N, T, ND, IERR)
C
C     ft*«««««»»ftft»ftft«»«*«ftX»ft*»*ft«»ftftftftft»»ft»*ft»««»»«»»»ft»»»*»»
C
C          RETURNS THE METRIC T OF ORDER N, COMPUTED IN DOUBLE
C          PRECISION.
C
C          THE ARGUMENT ND IS THE MATRIX DIMENSION.  IT MUST BE
C          AT LEAST ND =  (N)(N+1)(N+2)/6 OTHERWISE IERR = 101
C          IS RETURNED, AND NO OTHER ACTION IS TAKEN.  OTHER ER--
C          ROR RETURNS TRACE WHERE THE ERROR OCCURRED, BY JUST
C          PASSING THROUGH THE ERROR-INDICATOR VALUES FROM THE
C          CALLED ROUTINES.  NOTE THAT WGT IS THE ARRAY OF QUA-
C          DRATURE NODES, AND MUST BE DIMENSIONED SIMILARLY TO
C          ITS SIZE IN SUBROUTINE QUADRA.
C
C          SUBROUTINE CALLING STRUCTURE:
C
C          METRIC  CALLS  AFUNCT  CALLS  PFUNCT
C                  CALLS  PRECIS
C                  CALLS  QUADRA  CALLS  WEIGHT  CALLS  PSYMBL
C                                                CALLS  FACTOR
C
      DOUBLE PRECISION T(ND,ND), WGT(U55)
      DOUBLE PRECISION AFUNCT
      DIMENSION IARR(U), JARR(M), KARR(M)
      DOUBLE PRECISION EPSLON, SUM, ZETA(M), TERMI, TERMJ
      COMMON /PRECSN/ EPSLON
C
C     SET EPSLON TO START.  ALLOW FOUR TIMES EPSLON
C     AS FLOATING-POINT PRECISION.
      CALL PRECIS
      EPSLON = U.0D+0«EPSLON
C
C     CHECK DIMENSIONS IN CASE OF ERROR.
      IF (ND.GE.(N+1)*(N+2)*(N+3)/6) GO TO 10
      IERR s 101
      GO TO 180
   10 CONTINUE
C
C     NOW MAKE UP THE SET OF QUADRATURE NODES FOR THE
C     TETRAHEDRON OF DEGREE 2N.
      NBY2 = 2«N
      CALL QUADRA(WGT, NBY2, IERR)
C
C     OUTER LOOP:  GENERATE THE INDEX STRING IARR FOR
C     QUADRUPLE INDICES OF ORDER N.  IC IS THE COR-
C     RESPONDING SINGLE INDEX.
C
```

```
      IC = 0
      N1 = N + 1
      DO 170 J1=1,N1
        IARR(1) = N1 - J1
        M2 = N1 - IARRO)
        DO 160 J2=1,M2
          IARR(2) = M2 - J2
          M3 = M2 - IARR(2)
          DO 150 J3=1,M3
            IARRC3) = M3 - J3
            IARR(U) = N
            DO 20 J=1,3
              IARR(U) = IARRU) - IARR(J)
   20       CONTINUE
            IC s IC + 1
C
C     INNER LOOP:  GENERATE THE INDEX STRING JARR FOR
C     QUADRUPLE INDICES OF ORDER N.  JC IS THE CORRES-
C     PONDING SINGLE INDEX.
C
            JC s 0
            DO 140 11=1,N1
              JARR(1) = N1 - 11
              N2 = N1 - JARRC1)
              DO 130 12=1,N2
                JARR(2) = N2 - 12
                N3 - N2 - JARR(2)
                DO 120 13=1,N3
                  JARR(3) = N3 - 13
                  JARRU) = N
                  DO 30 J=1,3
                    JARR(U) = JARRC4) - JARR(J)
   30             CONTINUE
                  JC = JC + 1
C
C     BOTH INDEX STRINGS ARE NOW IN HAND.  COMPUTE
C     THE NEWTON-COTES QUADRATURE AT THIS NODE, BY
C     SCANNING THROUGH QUADRATURE NODES THE SAME AS
C     INTERPOLATION NODES OF ORDER 2*N.
C
C           DO ONLY LOWER TRIANGULAR HALF — GET THE
C           REST FROM SYMMETRY.
                  IF (JC.GT.IC) GO TO 120
                  SUM = O.OD+0
C
C     GENERATE INDEX STRING OF DEGREE 2«N.
C     KC IS THE SINGLE INDEX TO GO WITH IT.
                  KC = 0
                  N21 = NBY2 + 1
                  DO 110 K1=1,N21
                    KARRC1) r N21 - K1
                    N22 = N21 - KARRC1)
```

```
                      DO 100 K2=1,N22
                        KARR(2) s N22 - K2
                        N23 = N22 - KAHRC2)
                        DO 90 K3=1,N23
                          KARR(3) = N23 - K3
                          KARR(U) = 2«N
                          DO 40 J=1,3
                            KARR(M) = KARR(U) - KARR(J)
   40                     CONTINUE
                          KC s KC + 1
                          IF (DABS(WGT(KO).LE.EPSLON) GO TO 90
C
C     FIND COORDINATES AT QUADRATURE NODE
                          IF (N.NE.O) GO TO 60
                          DO 50 J=1,4
                            ZETA(J) = 0.25D+0
   50                     CONTINUE
                          GO TO 80
   60                   DO 70 Js1,4
                            ZETA(J) s KARR(J)
                            ZETA(J) = ZETA(J)/NBY2
   70                     CONTINUE
C
C     ADD NODAL CONTRIBUTION TO SUM
   80                     CONTINUE
                          TERMI = AFUNCT(IARR,ZETAtIERR)
                          IF (IERR.NE.O) GO TO 180
                          IF (DABS(TERMI).LE.EPSLON) GO TO 90
                          TERMJ r AFUNCT(JARR,ZETA,IERR)
                          IF (IERR.NE.O) GO TO 180
                          SUM = SUM + WGT(KC)»TERMI»TERMJ
   90                   CONTINUE
  100                 CONTINUE
  110               CONTINUE
C
                    T(IC,JC) = SUM
                    T(JC,IC) = SUM
C
  120             CONTINUE
  130           CONTINUE
  140         CONTINUE
C
  150       CONTINUE
  160     CONTINUE
  170   CONTINUE
C
  180   RETURN
        END
```

```
C
C     ******************************************************
C
      DOUBLE PRECISION FUNCTION PDERIV(Z, M, N, IERR)
C
C     ******************************************************
C
C          RETURNS THE VALUE, AT ARGUMENT VALUE Z, OF THE DE-
C          RIVATIVE OF THE SEMI-INTERPOLATIVE P-POLYNOMIAL M.
C          HERE N IS THE ORDER OF INTERPOLATION, IERR IS AN
C          ERROR FLAG, SET TO 0 IF ALL IS WELL.  POSSIBLE ER-
C          ROR FLAG SETTINGS ARE:  11 FOR ARGUMENT Z OUT OF
C          RANGE, 12 FOR NEGATIVE VALUE OF N, 13 FOR VALUE OF
C          M OUT OF RANGE.
C
      DOUBLE PRECISION PDERIV, Z, PR, FN, FI, FJ, EPSLON
      COMMON /PRECSN/ EPSLON
C
C     CHECK ARGUMENT VALUES FOR VALIDITY.  SET IERR.
      IERR = 0
      IF (Z.LT.-EPSLON .OR. Z.GT.1.0D0+EPSLON) IERR = 11
      IF (N.LT.0) IERR = 12
      IF (M.LT.0 .OR. M.GT.N) IERR = 13
      IF (IERR.NE.0) GO TO 30
C
C     SET VALUE, RETURN IMMEDIATELY IF M = 0.
      PDERIV = 0.D0
      IF (M.EQ.0) GO TO 30
C
C     COMPUTE DERIVATIVE IF M NONZERO, SUMMING TERMS.
      FN = N
      DO 20 J=1,M
        FJ = J
C
C     PRODUCT FOR ONE VALUE OF J -- OMIT J'TH FACTOR.
        PR = 1.D0
        DO 10 I=1,M
          IF (I.EQ.J) GO TO 10
          FI = I
          PR = PR*(FN*Z-FI+1.D0)/FI
   10   CONTINUE
C
        PDERIV = PDERIV + FN*PR/FJ
   20 CONTINUE
C
C     RETURN TO CALLING PROGRAM WITH VALUE.
   30 RETURN
      END
```

```
C
C
C
      DOUBLE PRECISION FUNCTION PFUNCTU, M, N, IERR)
C
C     ««««tt*»»»ftftft»ft»*ft»»ft*»»»»»«tt»«ft<tft»«tft«»»t«»«»««ft«»»ft*«ft
C
C          RETURNS THE VALUE, AT ARGUMENT VALUE Z, OF THE P-
C          POLYNOMIAL M.  N = ORDER OF INTERPOLATION, IERR =
C          ERROR FLAG, SET TO 0 IF ALL IS WELL.  POSSIBLE ER-
C          ROR FLAG SETTINGS ARE:  1 FOR ARGUMENT Z OUT OF
C          RANGE, 2 FOR NEGATIVE VALUE OF N, 3 FOR VALUE OF M
C          OUT OF RANGE.
C
      DOUBLE PRECISION PFUNCT, Z, FN, FI, EPSLON
      COMMON /PRECSN/ EPSLON
C
C     CHECK ARGUMENT VALUES FOR VALIDITY.  SET IERR.
      IERR = 0
      IF (Z.LT.-EPSLON .OR. Z.GT.1.OD+O+EPSLON) IERR = 1
      IF (N.LT.O) IERR = 2
      IF (M.LT.O .OR. M.GT.N) IERR = 3
      IF (IERR.NE.O) GO TO 20
C
C     SET VALUE, RETURN IMMEDIATELY IF M = 0.
      PFUNCT = 1.D0
      IF (M.EQ.O) GO TO 20
C
C     COMPUTE P IF M NONZERO.
      FN = N
      FN = Z«FN
      DO 10 1=1,M
        FI = I
        PFUNCT = PFUNCT*(FN-FI+1.D0)/FI
   10 CONTINUE
C
C     RETURN TO CALLING PROGRAM WITH VALUE.
   20 RETURN
      END
```

```
C
C
      SUBROUTINE PRECIS
C
C     ****************************************************************
C
C             DETERMINES, BY COMPUTATION, THE DOUBLE PRECISION
C             QUANTITY EPSLON, AND PLACES IT IN LABELLED COMMON.
C             EPSLON IS A MACHINE-DEPENDENT PRECISION INDICATOR -
C             SUCH THAT 1.OD+O AND (1.0D+0 + EPSLON) CAN JUST BE
C             TOLD APART ON THE COMPUTER IN USE.
C
      DOUBLE PRECISION EPSLON, EPSTRY
      COMMON /PRECSN/ EPSLON
C
C     BEGIN BY TAKING A BAD GUESS AT EPSLON
      EPSLON = 1.DO
C
C     KEEP DIVIDING BY 2 UNTIL THE DIFFERENCE BECOMES
C     INVISIBLE TO THE MACHINE.
   10 EPSTRY = EPSL0N/2.D0
      IF (1.D0+EPSTRY.EQ.1.D0) GO TO 20
      EPSLON = EPSTRY
      GO TO 10
C
C     SUCCESS!  EXIT.
   20 RETURN
      END
```

```
C
C
      SUBROUTINE PSYMBL(COEF, M, N, IERR)
C
C
C             RETURNS IN ARRAY fCOEF» THE COEFFICIENTS OF THE
C             SEMI-INTERPOLATIVE FUNCTION PM(Z), OF ORDER N.
C             THE ARRAY ELEMENT COEF(I) CONTAINS THE COEFFICI-
C             ENT OF Z»«(I-1)-0N RETURN.  IERR IS RETURNED AS
C             ZERO IF ALL IS WELL, AS 81 IF M IS OUT OF RANGE
C             RELATIVE TO N.  ARRAY COEF IS DIMENSIONED TO BE
C             SUFFICIENT FOR N = 1M; TO ALTER FOR OTHER POLY-
C             NOMIAL ORDERS, INCREASE NDIM IN DATA STATEMENT
C             BELOW, AND THE DIMENSION OF COEF, TO (N • 1).
C
      DOUBLE PRECISION C0EF(15), DN, DE, F1, F2
```

```
      DATA NDIM /15/
C
C     IS THE REQUEST REASONABLE?
      IERR = 0
      IF (M.LT.O .OR. M.GT.N) IERR = 81
      IF (N.GT.NDIM) IERR * 82
      IF (IERR.NE.O) GO TO 50
C
C     CLEAR THE ARRAY AND START
      DO 10 1=1,NDIM
        COEF(I) = 0.0D0
   10 CONTINUE
C
C     FOR M = 0, POLYNOMIAL IS ALWAYS UNITY.
      C0EF(1) = 1.0D0
      IF (M.EQ.O) GO TO 50
C
C     EVALUATE PRODUCT EXPRESSION RECURSIVELY.
C     I COUNTS THE FACTORS IN THE PRODUCT.
      DO 10 Is1,M
        DN r N
        DE = I
        F1 s DN/DE
        DN r 1 - I
        F2 = DN/DE
C
C     J LOCATES THE TERM OF ORDER (J-1) IN COEF,
        C0EF(I+1) = F1*C0EF(I)
        IF (I.EQ.1) GO TO 30
        DO 20 JBACK=2,I
          J = I - JBACK + 2
          COEF(J) = F1*C0EF(J-1) • F2*C0EF(J)
   20   CONTINUE
   30   C0EF(1) = F2*C0EF(1)
   HO CONTINUE
C
   50 CONTINUE
      RETURN
      END
```

```
C
C       «ft*ft*ft»»»«»ft«*«*ftftft»»«»»««*»ftft»ft»»»»*fttt«*ft«««ft»*»**ftft»«ft»
C
      SUBROUTINE QUADRA(WGT, N, IERR)
C
Q       «»*»Xtt»»»«»»»ftft»tt»»ft»«tt»»ftftft»»»»ft»»»«»«ttS»»«»»«*««*»»ft»»«
C
C            RETURNS THE DOUBLE-PRECISION VECTOR WGT OF WEIGHTS
C            FOR NEWTON-COTES QUADRATURE (CLOSED FORM) ON A TE-
C            TRAHEDRON. THE QUADRATURE IS OF ORDER N. WGT MUST
C            BE DIMENSIONED AT LEAST (N+1)(N+2)(N+3)/6. TO ALTER
C            DIMENSIONING, CHANGE WGT AND ALSO NDIM IN THE DATA
C            STATEMENT BELOW. IERR RETURNS AS 0 IF ALL IS WELL,
C            AS 91 IF DIMENSIONING EXCEEDED.
C
      DIMENSION IARR(U)
      DOUBLE PRECISION WEIGHT, EPSLON, WGTC455)
      COMMON /PRECSN/ EPSLON
      DATA NDIM /12/
C
C     ZERO THE OUTPUT ARRAY AND CHECK ARGUMENTS.
      IERR = 0
      IF (N.GT.NDIM) IERR = 91
      IF (IERR.GT.O) GO TO 60
      NEND = (NDIM+1)»(NDIM+2)*(NDIM+3)
      NEND = NEND/6
      DO 10 1=1,NEND
        WGT(I) = O.OD+0
   10 CONTINUE
C
C     GENERATE INDEX SEQUENCE AND FILL THE ARRAY.
      N1 = N + 1
      IC = 0
      DO 50 I1a1.N1
        IARR(1) = N1 - 11
C
        N2 = N1 - IARR(1)
        DO 40 12=1,N2
          IARRC2) = N2 - 12
C
          N3 = N2 - IARR(2)
          DO 30 13=1,N3
            IARR(3) = N3 - 13
C
            IARRU) = N
            DO 20 J=1,3
              IARR(U) = IARR(U) - IARR(J)
   20       CONTINUE
C
C     FIND WEIGHT FOR EACH SET OF INDICES.
            IC = IC + 1
            WGT(IC) = WEIGHT(IARR,TOTAL,IERR)
```

```
            IF (IERR.GT.O) GO TO 60
   30     CONTINUE
   40   CONTINUE
   50 CONTINUE
C
   60 CONTINUE
      RETURN
      END




C

C
      DOUBLE PRECISION FUNCTION WEIGHTUJKL, TOTAL, IERR)
C

C
C          RETURNS THE NEWTON-COTES QUADRATURE WEIGHT AT THE NODE
C          DESCRIBED BY ARRAY IJKL, ON A TETRAHEDRON.  THE DIMEN-
C          SION OF COEF IS GIVEN BY THE MAXIMUM QUADRATURE ORDER,
C          PLUS ONE, BY 4.  TO ALTER FOR HIGHER ORDERS CHANGE THE
C          DIMENSION OF COEF, ARR AND NDIM IN DATA STATEMENT.  IF
C          IERR IS RETURNED AS 84, THIS DIMENSIONING WAS INSUFFI-
C          CIENT.
C
C          ON RETURNING, THE SINGLE-PRECISION VARIABLE TOTAL CON-
C          TAINS THE SUM OF ABSOLUTE VALUES OF ALL TERMS TOTALLED
C          TO FIND THE QUADRATURE WEIGHT — AN ERROR ESTIMATOR.
C
      DIMENSION IJKL(4)
      DOUBLE PRECISION WEIGHT, C0EF(15,4), FACTOR, EPSLON
      DOUBLE PRECISION TERM, SUMP, SUMN, ARR(15), C2, C3, CM
      COMMON /PRECSN/ EPSLON
      DATA NDIM /15/
C
C     DETERMINE ORDER OF POLYNOMIALS FROM IJKL
      IERR =0
      N = 0
      DO 10 1=1,4
        N = N + IJKL(I)
   10 CONTINUE
      IF (N.GT.NDIM-1) IERR = 84
      IF (IERR.NE.O) GO TO 80
      N1 = N + 1
C
C     GET THE COEFFICIENT STRINGS FOR ALL FOUR P(Z>
      DO 30 1=1,4
        CALL PSYMBLCARR, IJKL(I), N, IERR)
        DO 20 J=1,NDIM
```

```
          COEF(J,I) = ARR(J)
   20     CONTINUE
          IF (IERR.NE.O) GO TO 80
   30   CONTINUE
C
C       MULTIPLY AND INTEGRATE SYMBOLICALLY
        SUMP = 0.0D+0
        SUMN = O.OD+0
        DO 70 HU1.N1
          CM = 6.0D+0*C0EF(IH,i»)*FACTOR(I4-1,IERR)
          IF (IERR.GT.O) GO TO 80
          IF (CM.EQ.O.OD+0) GO TO 70
          DO 60 13=1»N1
            C3 s CM«COEF(I3,3)*FACTOR(I3-1,IERR)
            IF (IERR.GT.O) GO TO 80
            IF (C3-EQ.O.OD+0) GO TO 60
            DO 50 I2s1,N1
              C2 = C3*COEF(I2,2)*FACTOR(I2-1,IERR)
              IF (C2.EQ.0.0D+0) GO TO 50
              IF (IERR.GT.O) GO TO 80
              DO HO IU1,N1
                IF (COEF(I1,1).EQ.O.OD+0) GO TO MO
                TERM = C2*C0EF(I1,1)*FACT0R(I1-1,IERR)/
     «            FACTOR(I1+I2+I3+I4-1,IERR)
                IF (IERR.GT.O) GO TO 80
                IF (TERM..GT.O.OD+0) SUMP s SUMP »• TERM
                IF (TERM..LT.O.OD+0) SUMN s SUMN + TERM
   MO         CONTINUE
   50       CONTINUE
   60     CONTINUE
   70   CONTINUE
        WEIGHT = SUMP • SUMN
        TOTAL = SUMP - SUMN
C
   80   RETURN
        END
```

## EXAMPLE DRIVER PROGRAMS

The following three programs are given to illustrate the use of the matrix generator subroutine package. While the subroutines are written in near-standard (ANSI 1968) Fortran, the driver programs are machine and system dependent; they will probably need modification by the user. The principal nonstandard features used are: (1) Fortran logical unit 7 used for terminal input and output, (2) the PROGRAM statement, (3) free-format terminal input, (4) use of $ as a carriage control character, (5) lower-case characters in Hollerith strings.

```
C
C
      PROGRAM CDEMON
C
C
C          THIS IS A MAIN PROGRAM TO ILLUSTRATE THE OPERATION OF
C          EMBMTX.  IT READS A VALUE OF N FROM THE- USER TERMINAL
C          (UNIT 7) AND PRINTS OUT THE MATRIX AT THE TERMINAL.
C
      DOUBLE PRECISION C1, EPSLON
      DIMENSION C1(84,56)
C
      COMMON /PRECSN/ EPSLON
C
C     NOTE:  NONSTANDARD CARRIAGE CONTROL AND READ FORMAT!
   10 WRITE (7,999)
      READ (7,*) N
      IF (N.LT.O) GO TO 40
      K = (N+1)«(N+2)«(N+3)/6
      M r (N+2)»(N+3)*(N+4)/6
C
      IERR = 0
      CALL EMBMTXCN, C1, 84, 56, IERR)
      IF (IERR.NE.O) GO TO 30
      DO 20 Is1,M
        WRITE (7,998) I, (CKI,J),Js1,K)
   20 CONTINUE
```

```
      GO TO 10
   30 WRITE (7,997) IERR
      GO TO 10
   40 STOP
  999 FORMAT (18H$Please enter N:  )
  998 FORMAT (1X, I2, (3X, 10F7.3))
  997 FORMAT (27H Error encountered; IERR = , I3)
      END




C
C     ***********************************************************
C
      PROGRAM DDEMON
C
C     ***********************************************************
C
C          THIS IS A MAIN PROGRAM TO ILLUSTRATE THE OPERATION OF
C          DIFMTX.  IT READS A VALUE OF N FROM THE USER TERMINAL
C          (UNIT 7) AND PRINTS OUT THE MATRIX AT THE TERMINAL.
C          THE MATRIX IS PRINTED OUT TRANSPOSED, TO MAKE IT FIT
C          THE TERMINAL SCREEN BEST.
C
      DOUBLE PRECISION D1, EPSLON
      DIMENSION D1(35,56)
      COMMON /PRECSN/ EPSLON
C
C     NOTE:  NONSTANDARD CARRIAGE CONTROL AND READ FORMAT!
   10 WRITE (7,999)
      READ (7,*) N
      IF (N.LE.0) GO TO 40
      K = N*(N+1)*(N+2)/6
      M = (N+1)*(N+2)*(N+3)/6
C
      IERR = 0
      CALL DIFMTX(N, D1, 35, 56, IERR)
      IF (IERR.NE.0) GO TO 30
      DO 20 J=1,M
        WRITE (7,998) J, (D1(I,J),I=1,K)
   20 CONTINUE
      GO TO 10
   30 IF (IERR.NE.0) WRITE (7,997) IERR
      GO TO 10
   40 STOP
  999 FORMAT (18H$Please enter N:  )
  998 FORMAT (1X, I2, (3X, 10F7.3))
  997 FORMAT (27H Error encountered; IERR = , I3)
      END
```

```
C
C
      PROGRAM MDEMON
C
Q     »»«»ft»ft»ft»»»*ft«»K*»»ft«»»»«ftft»»*tt«»*l»ft»*»»»«»»*«»»«»»»ft»
C
C          THIS IS A MAIN PROGRAM TO ILLUSTRATE THE OPERATION OF
C          METRIC.  IT READS A VALUE OF N FROM THE USER TERMINAL
C          (UNIT 7) AND PRINTS OUT THE MATRIX T AT THE TERMINAL.
C
      DIMENSION IARR(M)
      DOUBLE PRECISION WEIGHT, EPSLON, T(35,35)
      COMMON /PRECSN/ EPSLON
C
C     NOTE:  NONSTANDARD CARRIAGE CONTROL AND READ FORMAT!
   10 WRITE (7,999)
      READ (7,#) N
      IF (N.LT.O) GO TO MO
      M = (N+1)»(N+2)«(N+3)/6
C
      IERR = 0
      CALL METRICU, T, 35, IERR)
      IF (IERR.NE.O) GO TO 30
      DO 20 J=1,M
        WRITE (7,998) J, (T(I,J),Is1,M)
   20 CONTINUE
      GO TO 10
   30 IF (IERR.NE.O) WRITE (7,997) IERR
      GO TO 10
   HO STOP
  999 FORMAT (18H$Please enter N:   )
  998 FORMAT (1X, 12, (3X, 10F7.3))
  997 FORMAT (27H Error encountered; IERR = , 13) '
      END
```