MIT/LCS/TM-167

COMPUTER PROGRAMS FOR RESEARCH IN GRAVITATION AND DIFFERENTIAL GEOMETRY

Richard Pavelle
Michael Wester

June 1980

Computer Programs for Research in Gravitation and Differential Geometry
(chapter 11 of the MACSYMA REFERENCE MANUAL)

Richard Pavelle
Lincoln Laboratory
Massachusetts Institute of Technology
Lexington, Massachusetts 02173

Michael Wester*
Department of Mathematics
University of New Mexico
Albuquerque, New Mexico 87131

June 4, 1980

ABSTRACT:

This report contains a description of all current functions and features (with many examples) of the programs CTENSR and ITENSR which are available with MACSYMA. CTENSR is a standard Component TENSoR manipulation system which means that geometrical tensor objects are represented as arrays or matrices. Tensor operations such as contraction or covariant differentiation are carried out by actually summing over repeated (dummy) indices with DO statements. ITENSR, is a unique Indicial TENSoR manipulation system which is implemented by representing tensors as functions of their covariant, contravariant and derivative indices. Tensor operations such as contraction or covariant differentiation are performed by manipulating the indices themselves rather than the components to which they correspond. The programs are connected in the sense that one can obtain an expression in ITENSR and have the corresponding expression generated in the CTENSR format automatically.

KEY WORDS: MACSYMA, algebraic manipulation, computer science, general relativity, gravitation, tensor analysis, differential geometry.

* Part of this research was carried out when one of us (MW) was a Research Staff Member at the Laboratory for Computer Science, MIT during the summer of 1979.

This research was supported in part by the United States Department of Energy and under contract No. E(11-1)-3070 and in part by the National Aeronautics and Space Administration under grant No. NSG 1323.

1.1 Tensor Manipulation Programs- Introduction

MACSYMA implements symbolic tensor manipulation of two distinct types: **component** tensor manipulation and **indicial** tensor manipulation.

Component tensor manipulation means that geometrical tensor objects are represented as arrays or matrices. Tensor operations such as contraction or covariant differentiation are carried out by actually summing over repeated (dummy) indices with DO statements. That is, one explicitly performs operations on the appropriate tensor components stored in an array or matrix.

Indicial tensor manipulation is implemented by representing tensors as functions of their covariant, contravariant and derivative indices. Tensor operations such as contraction or covariant differentiation are performed by manipulating the indices themselves rather than the components to which they correspond.

These two approaches to the treatment of differential, algebraic and analytic processes in the context of Riemannian geometry have various advantages and disadvantages which reveal themselves only through the particular nature and difficulty of the user's problem. However, one should keep in mind the following characteristics of the two implementations:

Component Tensor Manipulation (CTENSR)

i) The representation of tensors and tensor operations explicitly in terms of their components makes CTENSR easy to use. Specification of the metric and the computation of the induced tensors and invariants is straightforward.

ii) Although all of MACSYMA's powerful simplification capacity is at hand, a complex metric with intricate functional and coordinate dependencies can easily lead to expressions whose size is excessive and whose structure is hidden. In addition, many calculations involve intermediate expressions which swell causing programs to terminate before completion. Through experience, a user can avoid avoid many of these difficulties.

Indicial Tensor Manipulation (ITENSR)

i) Because of the special way in which tensors and tensor operations are represented in terms of symbolic operations on their indices, expressions which in the *component* representation would be unmanageable can sometimes be greatly simplified by using the special routines for symmetrical objects in

ITENSR. In this way the structure of a large expression may be more transparent.

ii) On the other hand, because of the the special indicial representation in ITENSR, in some cases the user may find difficulty with the specification of the metric, function definition, and the evaluation of differentiated "indexed" objects.

These two tensor manipulation packages, CTENSR and ITENSR, are available to the MACSYMA user on the TENSOR directory. To use the functions in these files, the user can load them in by doing

LOADFILE(CTENSR,FASL,TENSOR); --- for component tensor manipulation.

LOADFILE(ITENSR,FASL,TENSOR); --- for indicial tensor manipulation

Both of these packages enable the user to specify a metric and compute the basic geometrical objects of interest. These routines were written primarily for research in gravitation theory. However, they can also be of use in other areas of physics where Riemannian geometry is applied owing to their generality.

1.2 Component Tensor Manipulation- Basic Functions

To use CTENSR the user does LOADFILE(CTENSR,FASL,TENSOR). The basic function is called

TSETUP*()* which automatically loads the CTENSR package from within MACSYMA (if it is not already loaded) and then prompts the user to make use of it. The user is first asked to specify the dimension of the manifold. If the dimension is 2, 3 or 4 then the list of coordinates defaults to [X,Y], [X,Y,Z] or [X,Y,Z,T] respectively. These names may be changed by assigning a new list of coordinates to the variable OMEGA (described below) and the user is queried about this. Care must be taken to avoid the coordinate names conflicting with other object definitions. Next, the user enters the metric either directly or from a file by specifying its ordinal position. As an example of a file of common metrics, see TENSOR;METRIC FILE. The metric is stored in the matrix LG. Finally, the metric inverse is computed and stored in the matrix UG. One has the option of carrying out all calculations in a power series.

A sample protocol is begun below for the static, spherically symmetric metric (standard coordinates) which will be applied to the problem of deriving Einstein's vacuum equations (which lead to the Schwarzschild solution) as an example. Many of the functions in CTENSR will be displayed for the standard metric as examples.

```
(C2) TSETUP();
Enter the dimension of the coordinate system:
4;
Do you wish to change the coordinate names?
N;
Do you want to
1. Enter a new metric?
2. Enter a metric from a file?
3. Approximate a metric with a Taylor series?
Enter 1, 2 or 3
1;

Is the matrix  1. Diagonal  2. Symmetric  3. Antisymmetric  4. General
Answer 1, 2, 3 or 4
1;
```

```
Row 1 Column 1:  A;

Row 2 Column 2:  X^2;

Row 3 Column 3:  X^2*SIN(Y)^2;

Row 4 Column 4:  -D;

Matrix entered.
Enter functional dependencies with the DEPENDS function or 'N' if none
DEPENDS([A,D],X);
Do you wish to see the metric?
Y;
                          [ A  0      0       0  ]
                          [                      ]
                          [     2                ]
                          [ 0  X      0       0  ]
                          [                      ]
                          [         2    2       ]
                          [ 0  0   X  SIN (Y)  0 ]
                          [                      ]
                          [ 0  0      0      - D ]

Do you wish to see the metric inverse?
N;
```

The other functions and features in CTENSR are now listed below.

CHRISTOF*(arg)* computes the Christoffel symbols of both kinds. The *arg* determines which results are to be immediately displayed. The Christoffel symbols of the first and second kinds are stored in the arrays LCS[i,j,k] and MCS[i,j,k] respectively and defined to be symmetric in the first two indices. If the argument to CHRISTOF is LCS or MCS then the unique non-zero values of LCS[i,j,k] or MCS[i,j,k], respectively, will be displayed. If the argument is ALL then the unique non-zero values of LCS[i,j,k] and MCS[i,j,k] will be displayed. If the argument is FALSE then the display of the elements will not occur. The array elements MCS[i,j,k] are defined in such a manner that the final index is contravariant. For the standard metric one has:

```
(C3) CHRISTOF(MCS);
                                                     A
                                                      X
(E3)                                MCS         = ---
                                       1, 1, 1   2 A

                                                      1
(E4)                                 MCS          = -
                                        1, 2, 2    X

                                                      1
(E5)                                 MCS          = -
                                        1, 3, 3    X

                                                   D
                                                    X
(E6)                                MCS         = ---
                                       1, 4, 4   2 D

                                                     X
(E7)                                MCS         = - -
                                       2, 2, 1       A

                                                COS(Y)
(E8)                              MCS        = ------
                                     2, 3, 3   SIN(Y)

                                                     2
                                                X SIN (Y)
(E9)                            MCS        = - ---------
                                   3, 3, 1          A

(E10)                         MCS        = - COS(Y) SIN(Y)
                                 3, 3, 2

                                                   D
                                                    X
(E11)                               MCS         = ---
                                       4, 4, 1   2 A
```

DIAGMETRIC if TRUE causes special routines to compute all geometrical objects (which contain the metric tensor explicitly) by taking into consideration the diagonality of the metric. Reduced run times will, of course, result. Note: this option is set automatically by TSETUP if a diagonal metric is specified.

DIM is the dimension of the manifold with the default 4. The command DIM : N will reset the dimension to any other integral value.

EINSTEIN*(dis)* computes the mixed Einstein tensor after the Christoffel symbols and Ricci tensor have been obtained. If the argument *dis* is TRUE, then the non-zero values of the mixed Einstein tensor G[i,j] will be displayed where j is the contravariant index. RATEINSTEIN[TRUE] if TRUE will cause the rational simplification on these components. If RATFAC is TRUE then the components will also be factored as the following example, for the standard metric, demonstrates:

```
(C40) EINSTEIN(TRUE);
                                  D  X - A D + D
                                   X
(E40)                       G       = --------------
                             1, 1              2
                                          A D X


                                 2                            2
             2 A D D  X - A D  X - A  D D  X + 2 A D D  - 2 A  D
                    X X       X       X   X          X       X
(E41)  G      = ----------------------------------------------------
        2, 2                              2  2
                                       4 A  D  X


                                 2                            2
             2 A D D  X - A D  X - A  D D  X + 2 A D D  - 2 A  D
                    X X       X       X   X          X       X
(E42)  G      = ----------------------------------------------------
        3, 3                              2  2
                                       4 A  D  X
```

```
                                        2
                                 A X + A  - A
                                   X
(E43)                    G      = - ------------
                          4, 4              2  2
                                           A  X
```

LRICCICOM(*dis*) computes the covariant (symmetric) components LR[i,j] of the Ricci tensor. If the argument *dis* is TRUE, then the non-zero components are displayed. For the standard metric one finds (with RATFAC:TRUE):

```
(C24) RATFAC:TRUE$

(C25) LRICCICOM(TRUE);
```

```
                                         2                            2
                     2 A D D   X - A D  X - A  D D  X - 4 A  D
                           X X        X        X   X          X
(E25)       LR     = - ------------------------------------------------
              1, 1                            2
                                           4 A D  X
```

```
                                              2
                              A D  X - A  D X - 2 A  D + 2 A D
                                   X      X
(E26)              LR     = - --------------------------------
                     2, 2                   2
                                         2 A  D
```

```
                                             2                2
                     (A D  X - A  D X - 2 A  D + 2 A D) SIN (Y)
                          X      X
(E27)        LR     = - ------------------------------------------
               3, 3                          2
                                          2 A  D
```

```
                                           2
                       2 A D D    X - A D   X - A  D D  X + 4 A D D
                              X X        X       X     X            X
(E28)        LR      = -----------------------------------------------
               4, 4                          2
                                         4 A  D X
```

MOTION*(dis)* computes the geodesic equations of motion for a given metric. They are stored in the array EM[i]. If the argument *dis* is TRUE then these equations are displayed.

OMEGA is an option which assigns a list of coordinates to the variable. While normally defined when the function TSETUP is called, one may redefine the coordinates with the assignment OMEGA:[*j1,j2,...jn*] where the *j*'s are the new coordinate names. A call to OMEGA will return the coordinate name list. Also see the function TSETUP above.

RATFAC*(false)* is a switch which, if TRUE, causes the Ricci, Einstein, Riemann, and Weyl tensors and the Scalar Curvature to be factored automatically. Clearly, this should only be set for cases where the tensorial components are known to consist of few terms.

RIEMANN*(dis)* computes the Riemann curvature tensor from the given metric and the corresponding Christoffel symbols. If *dis* is TRUE, the non-zero components R[i,j,k,l] will be displayed. All the indicated indices are covariant. As with the Einstein tensor, various switches set by the user control the simplification of the components of the Riemann tensor. If RATRIEMAN[TRUE] is TRUE then rational simplification will be done. If RATFAC is TRUE then each of the components will also be factored.

RICCICOM*(dis)* This function first computes the covariant components LR[i,j] of the Ricci tensor. Then the mixed Ricci tensor is computed using the contravariant metric tensor. If the value of the argument to RICCICOM is TRUE, then these mixed components, RICCI[i,j] (the index i is covariant and the index j is contravariant), will be displayed directly. Otherwise, RICCICOM(FALSE) will simply compute the entries of the array RICCI[i,j] without displaying the results.

SCURVATURE() returns the Scalar Curvature as the trace of the mixed Ricci tensor. With RATFAC:TRUE this invariant will be factored.

WEYL(*dis*) computes the covariant Weyl conformal tensor. If the argument *dis* is TRUE, the non-zero components W[i,j,k,l] will be displayed. Otherwise, these components will be computed and stored. If the switch RATWEYL[TRUE] is set to TRUE, then the components will be rationally simplified. If RATFAC is TRUE then the results will be factored as well. The following example illustrates the use of the function for an elementary metric which is chosen to be conformally flat.

```
(C7) LG;
                                        [ A  0  0  0  0 ]
                                        [               ]
                                        [ 0  A  0  0  0 ]
                                        [               ]
(D7)                                    [ 0  0  A  0  0 ]
                                        [               ]
                                        [ 0  0  0  A  0 ]
                                        [               ]
                                        [ 0  0  0  0  A ]
(C8) DEPENDENCIES;
(D8)                                          [A(T)]

(C9) RATWEYL:TRUE;
(D9)                                           TRUE

(C10) WEYL(TRUE);
THIS SPACETIME IS CONFORMALLY FLAT
Time= 94320 msec.
(D10)                                          DONE
```

1.3 Component Tensor Manipulation- Auxiliary Functions

CHECKDIV*(tensor)* computes the covariant divergence of the mixed second rank *tensor* (whose first index must be covariant) by printing the corresponding n components of the vector field (the divergence) where n = DIM. If the argument to the function is G then the divergence of the Einstein tensor will be formed and must be zero. In addition, the divergence (vector) is given the array name DIV.

COGRAD*(function,name)* computes the COvariant GRADient of a scalar *function* allowing the user to choose the corresponding vector*name* as the example under CONTRAGRAD illustrates.

CONTRAGRAD*(function,name)* computes the CONTRAvariant GRADient of a scalar *function* allowing the user to choose the corresponding vector*name* as the example below for the standard metric illustrates.

```
(C12) DEPENDS(F,X);
(D12)                        [F(X)]
(C13) COGRAD(F,G1)$
(C14) LISTARRAY(G1);

(D14)                     [F , 0, 0, 0]
                            X
(C15) CONTRAGRAD(F,G2)$
(C16) LISTARRAY(G2);
                           F
                            X
(D16)                     [--, 0, 0, 0]
                           A
```

DELETEN*(list,n)* returns a new list consisting of *list* with the *n*th element deleted.

DSCALAR*(function)* computes the tensor d'Alembertian of the scalar *function* once dependencies have been declared upon the *function*. For the standard metric one has:

```
(C16) DEPENDS(P,X);
(D16)                                    [P(X)]

(C17) FACTOR(DSCALAR(P));
                  2 A D P   X + A D  P  X - A  D P  X + 4 A D P
                        X X       X  X       X  X             X
(D17)             ---------------------------------------------
                                       2
                                   2 A  D X
```

FINDDE*(array,n)* returns a list of the unique differential equations (expressions) corresponding to the elements of the *n* dimensional square *array*. Presently, *n* may be 2 or 3. DEINDEX is a global list containing the indices of *array* corresponding to these unique differential equations. For the Einstein tensor (G) given above, which is a two dimensional array, FINDDE gives the following independent differential equations:

```
(C19) FINDDE(G,2);
                                                  2
(D19) [D  X - A D + D, 2 A D D    X - A D    X - A  D D  X + 2 A D D
        X                   X X        X       X    X             X

                                                         2           2
                                               - 2 A  D , A  X + A  - A]
                                                    X      X
(C20) DEINDEX;
(D20)                    [[1, 1], [2, 2], [4, 4]]
```

NTERMST*(f)* gives the user a quick picture of the "size" of the doubly subscripted tensor (array) *f*. It prints two element lists where the second element corresponds to NTERMS of the components specified by the first elements. In this way, it is possible to quickly find the non-zero expressions and attempt simplification.

RAISERIEMANN*(dis)* returns the *contravariant* components of the Riemann curvature tensor as array elements UR[i,j,k,l]. These are displayed if *dis* is TRUE.

RINVARIANT(*)* forms the Kretschmann invariant obtained by contracting the tensors R[i,j,k,l]*UR[i,j,k,l]. This object not automatically simplified since it can be very large. For the standard metric, however, the invariant is small and easily factored. One finds:

```
(C20) FACTOR(RINVARIANT());

           2  2     2  4      2     2        4          2          4
(D20) (4  A  D  D     X  - 4 A  D D   D    X - 4 A A  D  D  D    X
                 X X               X   X X          X     X  X X

     2   4  4              3  4     2  2   2  4     2  2   2  2
  + A  D   X  + 2 A A  D D   X  + A   D  D   X + 8 A  D  D   X
        X            X    X        X      X               X

        2  4  2       4  4       3  4      2  4      4  4  4
  + 8 A   D  X  + 16 A  D  - 32 A  D + 16 A  D )/(4 A  D  X )
       X
```

TRANSFORM(*matrix)* will perform a coordinate transformation upon an arbitrary square symmetric *matrix*. The user must input the functions which define the transformation as in C8 below. The following example considers the transformation from Cartesian to spherical coordinates:

```
(C5) DIM:3$

(C6) OMEGA:[X,Y,Z]$

(C7) LG:MATRIX([1,0,0],[0,1,0],[0,0,1]);

                                   [ 1  0  0 ]
                                   [         ]
(D7)                               [ 0  1  0 ]
                                   [         ]
                                   [ 0  0  1 ]
```

```
(C8) TRANSFORM(LG)$

TRANSFORM # 1
X*SIN(Y)*SIN(Z);
TRANSFORM # 2
X*SIN(Y)*COS(Z);
TRANSFORM # 3
X*COS(Y);

(C9) /* a substitution which reduces the transformed matrix */

EV(%,COS(Y) = SQRT(1-SIN(Y)^2),SIN(Z) = SQRT(1-COS(Z)^2),RATSIMP);

                                 [ 1  0      0      ]
                                 [                  ]
                                 [     2            ]
(D9)                             [ 0  X      0      ]
                                 [                  ]
                                 [         2    2   ]
                                 [ 0  0   X  SIN (Y) ]
```

1.4 Component Tensor Manipulation- Alternate Gravity Theories

BDVAC() generates the covariant components of the vacuum field equations of the Brans- Dicke gravitational theory. There are two field equations. The components of the second rank covariant field tensor are represented by the array BD2. The scalar field equation requires the user to input the name of a scalar and declare its functional dependencies. This field equation is represented by the scalar BD0.

INVARIANT1() generates the mixed Euler- Lagrange tensor (field equations) for the invariant density of R^2. The field equations are the components of an array named INV1.

INVARIANT2() generates the mixed Euler- Lagrange tensor (field equations) for the invariant density of LR[i,j]*UR[i,j]. The field equations are the components of an array named INV2.

BIMETRIC() generates the field equations of Rosen's bimetric theory. The field equations are the components of an array named ROSEN.

[1.5] Indicial Tensor Manipulation

In ITENSR a tensor is represented as an "indexed object" . This is a function of 3 groups of indices which represent the covariant, contravariant and derivative indices. The covariant indices are specified by a list as the first argument to the indexed object, and the contravariant indices by a list as the second argument. If the indexed object lacks either of these groups of indices then the empty list [] is given as the corresponding argument. Thus, G([a,b],[c]) represents an indexed object called G which has two covariant indices (a,b), one contravariant index (c) and no derivative indices.

The derivative indices, if they are present, are appended as additional arguments to the symbolic function representing the tensor. They can be explicitly specified by the user or be created in the process of differentiation with respect to some coordinate variable. Since ordinary differentiation is commutative, the derivative indices are sorted alphanumerically. This canonical ordering makes it possible for MACSYMA to recognize that, for example, T([a],[b],i,j) is the same as T([a],[b],j,i). Differentiation of an indexed object with respect to some coordinate whose index does not appear as an argument to the indexed object would normally yield zero. This is because MACSYMA would not know that the tensor represented by the indexed object might depend implicitly on the corresponding coordinate. By modifying the existing MACSYMA function DIFF in ITENSR, MACSYMA now assumes that all indexed objects depend on any variable of differentiation unless otherwise stated. This makes it possible for the summation convention to be extended to derivative indices. <u>It should be noted that ITENSR does not possess the capabilities of raising derivative indices, and so they are always treated as covariant.</u>

The following functions are available in the tensor package for manipulating indexed objects. At present, with respect to the simplification routines, it is assumed that all indexed objects are completely symmetric in their lists of covariant indices and symmetric in their lists of contravariant indices. This can be overridden by setting the variable ALLSYM[TRUE] to FALSE which will result in no symmetry assumptions in these two sets of indices. <u>However, the simplification routines may no longer operate completely.</u>

In what follows, general indexed objects will be denoted by *tensor1, tensor2,* ... . The symbols *L1, L2,...* denote lists which are arguments to indexed objects. Optional arguments are enclosed in angle brackets.

1.6 Indicial Tensor Manipulation- Basic Functions

CHR1*([i,j,k])* yields the Christoffel symbol of the first kind via the definition

$$(g_{ik,j} + g_{jk,i} - g_{ij,k})/2 .$$

To evaluate the Christoffel symbols for a particular metric, the variable METRIC must be assigned a name as in the example under CHR2.

CHR2*([i,j],[k])* yields the Christoffel symbol of the second kind defined by the relation

$$\mathrm{CHR2}([i,j],[k]) = g^{ks} (g_{is,j} + g_{js,i} - g_{ij,s})/2$$

As an example we consider a conformally flat metric and find the Christoffel symbols of both kinds:

```
(C7) DECLARE(E,CONSTANT)$

(C8) METRIC(G)$

(C9) COMPONENTS(G([I,J],[]),E([I,J],[])*P([],[]))$

(C10) COMPONENTS(G([],[I,J]),E([],[I,J])/P([],[]))$

(C11) SHOW(G([I,J],[]));
(D11)                              P E
                                      I J

(C12) SHOW(G([],[I,J]));
                                     I J
                                   E
(D12)                              ----
                                    P


(C13) SHOW(FACTOR(CHR1([I,J,K])));
                       P   E    + P   E    - P   E
                        ,I  J K    ,J  I K    ,K  I J
(D13)                  ------------------------------
                                     2
```

```
(C30) SHOW(FACTOR(CHR2([I,J],[K])));
                 %1 K
                E     (P   E     - P   E      - P   E     )
                          ,%1  I J     ,I  %1 J     ,J  %1 I
(D31)          -------------------------------------------
                                     2 P
```

COMPONENTS(*tensor,exp*) permits one to assign an indicial value to an expression *exp* giving the values of the components of *tensor*. These are automatically substituted for the *tensor* whenever it occurs with all of its indices. The *tensor* must be of the form T([...],[...]) where either list may be empty. *Exp* can be any indexed expression involving other objects with the same free indices as *tensor*. When used to assign values to the metric tensor wherein the components contain dummy indices one must be careful to define these indices to avoid the generation of multiple dummy indices. Removal of this assignment is given to the function REMCOMPS described below.

The example under DEFCON (C9 - D12) demonstrates the use of the *COMPONENTS* function to define an *algebraically special metric* and also shows how the null property of the vector field can be given with the property assignment functions. The example above under CHR2 gives the basic syntax used in the COMPONENTS statement.

CONTRACT(*exp*) carries out the tensorial contractions in *exp* which may be any combination of sums and products. This function uses the information given to the DEFCON function. When using CONTRACT, *exp* must be fully expanded. Also see the function METRIC and the example under *DIM*.

COVDIFF(*exp,v1,v2,...*) yields the covariant derivative of *exp* with respect to the variables *vi* in terms of the Christoffel symbols of the second kind (CHR2). In order to evaluate these, one can use EV(*exp*,CHR2).

```
(C3) ENTERTENSOR();
Enter tensor name: A;
Enter a list of the covariant indices: [I,J];
Enter a list of the contravariant indices: [K];
Enter a list of the derivative indices: [];
                                         K
(D3)                                    A
                                         I J
```

```
(C4) SHOW(COVDIFF(%,S));
               K        %1    K        %1     K          K     %1
(D4)        - A     CHR2   - A     CHR2   + A      + CHR2      A
               I %1     J S   %1 J     I S    I J,S      %1 S  I J
```

DIFF*(exp,v1,n1,v2,n2,...)* is the usual MACSYMA differentiation function which has been expanded in its abilities for ITENSR. It takes the derivative of *exp* with respect to *v1 n1* times, with respect to *v2 n2* times, etc. For the tensor package, the following modifications have been incorporated (also see the function UNDIFF):

1) the derivatives of any indexed objects in *exp* will have the variables *vi* appended as additional arguments. Subsequently, all derivative indices will be sorted.

2) the *vi* may be integers from 1 up to the value of the variable DIM[4]. This will cause the differentiation to be carried out with respect to the *vi*th member of the list COORDINATES which should be set to a list of the names of the coordinates, e.g., [x,y,z,t] . If COORDINATES is bound to an atomic variable, then that variable subscripted by *vi* will be used for the variable of differentiation. This permits an array of coordinate names or subscripted names like X[1], X[2],... to be used. If COORDINATES has not been assigned a value, then the variables will be treated as in 1) above.

3) one may now differentiate the determinant of the metric tensor. Thus, if METRIC has been bound to G then DIFF(DETERMINANT(G),K) will return 2*DETERMINANT(G)*CHR2([%i,K],[%i]) where the dummy index has been appropriately chosen.

DIM is the dimension of the manifold with the default 4. The command DIM : N will reset the dimension to any other integral value. The following example demonstrates the contraction property of the Kronecker delta.

```
(C4) CONTRACT(KDELTA([A],[B])*KDELTA([B],[A]));
(D4)                            KDELTA([], [])

(C5) EV(%,KDELTA);
(D5)                                  4
```

ENTERTENSOR*(<name>)* is a function which, by prompting, allows one to create an indexed object called *name* with any number of tensorial and derivative indices. Either a single index or a list of indices (which may be null) is acceptable input (see the example under COVDIFF).

GEODESIC*(exp,name)* enables the user to cause undifferentiated Christoffel symbols and first derivatives of the metric tensor vanish in *exp*. The *name* in the GEODESIC function refers to the metric *name* (if it appears in *exp*) while the connection coefficients must be called with the names CHR1 and/or CHR2. The following example demonstrates the verification of the cyclic identity upon the Riemann tensor using RENAME while also showing the use of the GEODESIC function.

```
(C2) EXP:RIEMANN([R,S,T],[U])+RIEMANN([S,T,R],[U])+RIEMANN([T,R,S],[U])$

(C3) SHOW(EXP);
              U             U          %6        U            U          %6       U
(D3) - CHR2        - CHR2        CHR2     + CHR2       + CHR2      CHR2     + CHR2
           T S,R          %6 R       T S         T R,S        %6 S      T R        S T,R

          U          %5         U             U         %5         U
 + CHR2        CHR2      - CHR2     - CHR2        CHR2      - CHR2
       %5 R         S T         S R,T       %5 T        S R        R T,S

          U          %4         U             U         %4
 - CHR2        CHR2      + CHR2       + CHR2       CHR2
       %4 S         R T         R S,T        %4 T        R S

(C4) SHOW(GEODESIC(EXP,CHR2));
             U            U            U            U            U            U
(D4) - CHR2       + CHR2       + CHR2       - CHR2       - CHR2       + CHR2
           T S,R        T R,S        S T,R        S R,T        R T,S        R S,T

(C5) SHOW(RENAME(EXP));
(D5)                                      0
```

INDEXED*(tensor)* must be executed before assigning components to a *tensor* for which a built in value already exists as with CHR1, CHR2, RIEMANN. See the example under RIEMANN.

KDELTA*(L1,L2)* is the generalized Kronecker delta function with *L1* the list of covariant indices and *L2* the list of contravariant indices. KDELTA([i],[j]) returns the ordinary Kronecker delta. The command EV(EXP,KDELTA) causes the evaluation of an expression containing KDELTA([],[]) to the dimension of the manifold (see the example under DIM).

```
(C3) KDELTA([A,B,C],[R,S,T])$

(C4) SHOW(EV(%));
                R        S        T         T        S
(D4) - KDELTA  (KDELTA  KDELTA  - KDELTA  KDELTA )
                A        B        C         B        C

          R        T        S        S        T
 - KDELTA  (KDELTA  KDELTA  - KDELTA  KDELTA )
          B        A        C        A        C

          S        T          T        S        R
 - (KDELTA  KDELTA  - KDELTA  KDELTA ) KDELTA
          A        B          A        B        C
```

LC*(L)* is the permutation (or Levi-Civita) tensor density which yields 1 if the list *L* consists of an even permutation of integers, -1 if it consists of an odd permutation, and 0 if some indices in *L* are repeated.

METRIC*(name)* specifies *name* as the metric name by assigning the variable METRIC:*name*. In addition, the contraction properties of the metric *name* are set up by executing the commands DEFCON(*name*), DEFCON(*name, name,* KDELTA). See, for example, the example under RIEMANN.

RATEXPAND*(exp)* is the fastest way to expand products and powers of sums of indexed objects generated by ITENSR within MACSYMA.

RENAME*(exp, <count>)* returns an expression equivalent to *exp* but with the dummy indices in each term chosen from the set [%1, %2,...], if the optional second argument is omitted. Otherwise, the dummy indices are indexed beginning at the value of *count*. Each dummy index in a product will be different. For a sum, RENAME will operate upon each term in the sum resetting the counter with each term. In this way RENAME can serve as a tensorial simplifier. In addition, the indices will be sorted alphanumerically (if ALLSYM is TRUE) with respect to covariant or contravariant indices depending upon the value of FLIPFLAG. If FLIPFLAG is FALSE then the indices will be renamed according to the order of the covariant indices otherwise according to the order of the contravariant indices. It often happens that the combined effect of the two renamings will reduce an expression more than either one by itself.

```
(C41) SHOW(EXP);
         %4 %5  %6 %7     %3        U        %1        %2
(D41) G        G       CHR2      CHR2      CHR2      CHR2
                          %1 %4     %2 %3     %5 %6     %7 R

                      %4 %5  %6 %7      U        %1        %3        %2
                   - G        G       CHR2      CHR2      CHR2      CHR2
                                         %1 %2     %3 %5     %4 %6     %7 R

(C42) FLIPFLAG;
(D42)                                    FALSE

(C43) SHOW(RENAME(EXP));
         %2 %5  %6 %7     %4        U        %1        %3
(D43) G        G       CHR2      CHR2      CHR2      CHR2
                          %1 %2     %3 %4     %5 %6     %7 R

                      %4 %5  %6 %7      U        %1        %3        %2
                   - G        G       CHR2      CHR2      CHR2      CHR2
                                         %1 %2     %3 %4     %5 %6     %7 R
(C44) FLIPFLAG:TRUE$

(C45) RENAME(D42);
(D45)                                  0

(C46) [FIRST(D42),LAST(D42)]$
```

```
(C46) SHOW(RENAME(%));
         %1 %2  %3 %4      %5          %6       %7       U
(D46) [G       G       CHR2       CHR2       CHR2     CHR2      ,
                           %1 %6      %2 %3      %4 R      %5 %7

                      %1 %2  %3 %4      %5          %6          %7          U
                    - G       G       CHR2       CHR2       CHR2     CHR2      ]
                                          %1 %6      %2 %3      %4 R      %5 %7
```

RIEMANN(*[i,j,k]*,*[h]*) yields the Riemann curvature tensor in terms of the Christoffel symbols of the second kind (CHR2). The following notation is used:

```
                h          h         h       %1      h
     RIEMANN      = - CHR2     - CHR2    CHR2    + CHR2
                i j k         i k,j      %1 j     i k       i j,k

                          h          %1
                   + CHR2     CHR2
                          %1 k      i j
```

Suppose the name specified by the value of METRIC corresponds to a tensor which has been given some structure via the COMPONENTS command. In order to evaluate an *expression* involving the Riemann tensor and incorporate this given definition of the metric explicitly into the result, the user can do *expression*, EVAL as the following example for the weak field metric demonstrates:

```
(C5) INDEXED(CHR2)$

(C6) DECLARE(E,CONSTANT)$

(C7) METRIC:G$

(C8) COMPONENTS(G([M,N],[]),E([M,N],[])+2*L*P([M,N],[]))$

(C9) COMPONENTS(G([],[M,N]),E([],[M,N])-2*L*P([],[M,N]))$

(C10) SHOW(G([I,J],[]));
(D10)                      2 L P    + E
                                I J    I J
```

```
(C11) SHOW(G([],[I,J]));

                                        I J    I J
(D11)                                  E   - 2 P   L

(C12) (RATVARS(L),RATWEIGHT(L,1),RATWTLVL:1)$

(C13) RIEMANN([S,U,N],[Y])$

(C14) %,EVAL$

(C15) SHOW(CANFORM(CONTRACT(RENAME(RATEXPAND(%)))))$

           %1 Y                    %1 Y                   %1 Y
(D15) - E       L P            + E      L P           + E      P          L
                    S U,%1 N                N S,%1 U             %1 U,N S

                                                              %1 Y
                                                           - E     P          L
                                                                    %1 N,S U
```

SHOW(*exp*) displays *exp* with the indexed objects in it shown having their covariant indices as subscripts and contravariant indices as superscripts. The derivative indices are displayed as subscripts, separated from the covariant indices by a comma (see the example above).

UNDIFF(*exp*) returns an expression equivalent to *exp* but with all derivatives of indexed objects replaced by the noun form of the DIFF function. Its arguments would yield that indexed object if the differentiation were carried out. This is useful when it is desired to replace a differentiated indexed object with some function definition resulting in *exp* and then carry out the differentiation by saying EV(*exp*, DIFF).

1.7 Indicial Tensor Manipulation- Simplification Functions

ALLSYM*(true)* if TRUE then all indexed objects are assumed symmetric in all of their covariant and contravariant indices. If FALSE then no symmetries of any kind are assumed in these indices. Derivative indices are always taken to be symmetric.

CANFORM*(exp)* simplifies *exp* by renaming dummy indices and reordering all indices as dictated by symmetry conditions imposed on them. If ALLSYM is TRUE then all indices are assumed symmetric, otherwise symmetry information provided by DECSYM declarations will be used. The dummy indices are renamed in the same manner as in the RENAME function. When CANFORM is applied to a large expression the calculation may take a considerable amount of time. This time can be shortened by calling RENAME on the expression first. Also see the example under DECSYM. Note: CANFORM may not be able to reduce an expression completely to its simplest form although it will always return a mathematically correct result.

CANTEN*(exp)* simplifies *exp* by renaming (see RENAME) and permuting dummy indices. CANTEN is restricted to sums of tensor products in which no derivatives are present. As such it is limited and should only be used if CANFORM is not capable of carrying out the required simplification.

CHANGENAME*(old,new,exp)* will change the name of all indexed objects called *old* to *new* in *exp*. *Old* may be either a symbol or a list of the form *[name, m, n]* in which case only those indexed objects called *name* with *m* covariant and *n* contravariant indices will be renamed to *new*.

CONMETDERIV*(exp,tensor)* is used to simplify expressions containing ordinary derivatives of both covariant and contravariant forms of the metric tensor (the current restriction). For example, CONMETDERIV can relate the derivative of the contravariant metric tensor with the Christoffel symbols as seen from the following:

```
(C8) SHOW(G([],[A,B],C))$

                                A B
(D8)                           G
                                ,C
```

```
(C9) SHOW(CONMETDERIV(%,G));
                                 %1 B      A         %1 A      B
(D9)                          - G     CHR2        - G     CHR2
                                           %1 C                %1 C
```

FLIPFLAG*(false)* if FALSE then the indices will be renamed according to the order of the covariant indices otherwise according to the order of the contravariant indices. The function influences RENAME in the following way: If FLIPFLAG is FALSE then RENAME forms a list of the covariant indices as they are encountered from left to right (if TRUE then of the contravariant indices). The first dummy index in the list is renamed to %1, the next to %2, etc. Then sorting occurs after the RENAMEing (see the example under RENAME).

FLUSH*(exp,tensor1,tensor2,...)* will set to zero, in *exp*, all occurrences of the *tensori* that have no derivative indices.

FLUSHD*(exp,tensor1,tensor2,...)* will set to zero, in *exp*, all occurrences of the *tensori* that have derivative indices.

FLUSHND*(exp,tensor,n)* will set to zero, in *exp*, all occurrences of the differentiated object *tensor* that have *n* or more derivative indices as the following example demonstrates.

```
(C3) SHOW(A([I],[J,R],K,R)+A([I],[J,R,S],K,R,S));
                                     J R S      J R
(D3)                                A         + A
                                     I,K R S    I,K R

(C4) SHOW(FLUSHND(D3,A,3));
                                          J R
(D4)                                     A
                                          I,K R
```

FLUSH1DERIV*(exp,tensor)* will set to zero, in *exp*, all occurrences of *tensor* that have exactly one derivative index.

LORENTZ*(exp, <tensor1, tensor2, ...>)* imposes a generalized Lorentz condition on *exp* replacing by zero those *tensori* which have a derivative index identical to a contravariant index. If no *tensori* are specified, this process will be performed on all indexed objects in *exp* (see the example under MAKEBOX).

MAKEBOX*(exp,tensor)* will display, with the symbol [], all occurrences of the flat-space d'Alembertian operator acting upon *tensor* in *exp*. The name of the flat-space metric appears in the argument to the function. In the following example EIN is the weak field approximation of the Einstein tensor for the metric which is given and L is small.

```
(C56) SHOW(G([I,J]));

(D56)                       P    L + E
                             I J      I J


(C57) SHOW(EIN);
         %1  %2  I J        %1 %2   I J             %1 %2  I J
(D57) - E       P      L - P       E    L + P      E      E    L
                 ,%1 %2     ,%1 %2           ,%1 %2

               %1 I  %2 J       %1 I    %2 J             %1 I  %2 J
            + E     P      L + P       E   L - P        E     E     L
                     ,%1 %2      ,%1 %2           ,%1 %2

(C58) SHOW(LORENTZ(%,P));
         %1  %2  I J                %1 %2  I J
(D58) - E       P      L + P       E     E     L
                 ,%1 %2     ,%1 %2

                                                         %1 I  %2 J
                                               - P      E     E     L
                                                  ,%1 %2


(C59) SHOW(MAKEBOX(%,E));
                 I J          I J            %1 I  %2 J
(D59)       - []P   L + []P E    L - P      E     E     L
                                      ,%1 %2
```

1.8 Indicial Tensor Manipulation- Property Assignment Functions

COORD*(tensor1,tensor2,...)* gives *tensori* the coordinate differentiation property that the derivative of contravariant vector whose name is one of the *tensori* yields a Kronecker delta. For example, if COORD(X) has been done then DIFF(X([],[I]),J) gives KDELTA([I],[J]). COORD is a list of all indexed objects having this property.

DECLARE*(object,property)* allows the specification of certain *properties* upon the *object*. For example, we can specify that an indexed object is independent of all coordinate variables. Whereas DIFF(W([],[I,J]),K) normally results in W([],[I,J],K), with the command DECLARE(W,CONSTANT) given, the result of the differentiation will be 0. Similarly, one can declare a vector to be null (see the example under the DEFCON function).

DECSYM*(tensor, m, n, [cov1,cov2,...], [contr1,contr2,...])* declares symmetry properties for *tensor* of *m* covariant and *n* contravariant indices. The *covi* and *contri* are pseudofunctions expressing symmetry relations among the covariant and contravariant indices respectively. These are of the form *symoper*(*index1*, *index2*,...) where *symoper* is one of SYM, ANTI or CYC and the *indexi* are integers indicating the position of the index in the *tensor*. This will declare *tensor* to be symmetric, antisymmetric or cyclic respectively in the *indexi*. *symoper*(ALL) is also an allowable form which indicates all indices obey the symmetry condition. For example, given an object B with 5 covariant indices, DECSYM (B,5,3,[SYM(1,2),ANTI(3,4)],[CYC(ALL)]) declares B symmetric in its first and second and antisymmetric in its third and fourth covariant indices, and cyclic in all of its contravariant indices. Either list of symmetry declarations may be null. The function which performs the simplifications is CANFORM as the example below illustrates.

```
(C4) EXP:A([K,J,I],[])+A([K,I,J],[])+A([J,K,I],[])+
         A([J,I,K],[])+A([I,K,J],[])+A([I,J,K],[])$

(C5) SHOW(EXP);
(D5)          A      + A      + A      + A      + A      + A
               K J I    K I J    J K I    J I K    I K J    I J K

(C6) ALLSYM;
(D6)                              TRUE
```

```
(C7) SHOW(CANFORM(EXP));

(D7)                                6 A
                                       I J K

(C8) ALLSYM:FALSE$

(C9) DECSYM(A,3,0,[ANTI(ALL)],[])$

(C10) DISPSYM(A,3,0);
(D10)                        [[ANTI, [[1, 2, 3]], []]]

(C11) SHOW(CANFORM(EXP));
(D11)                                  0

(C12) REMSYM(A,3,0)$

(C13) DECSYM(A,3,0,[CYC(ALL)],[])$

(C14) SHOW(CANFORM(EXP));
(D14)                         3 A     + 3 A
                                 I K J     I J K
```

DEFCON*(tensor1,<tensor2,tensor3>)* gives *tensor1* the property that the contraction of a product of *tensor1* and *tensor2* results in *tensor3* with the appropriate indices. If only one argument, *tensor1*, is given, then the contraction of the product of *tensor1* with any indexed object having the appropriate indices (say *tensor*) will yield an indexed object with that name, i.e. *tensor*, and with a new set of indices reflecting the contractions performed. For example, if METRIC:G, then DEFCON(G) will implement the raising and lowering of indices through contraction with the metric tensor. CONTRACTIONS is a list of those indexed objects which have been given contraction properties with DEFCON.

The following example for an *algebraically special metric* shows how the null property of a vector field may be assigned as well as demonstrating that more than one DEFCON assignment can be given for the same indexed object.

```
(C4) DECLARE(E,CONSTANT)$

(C5) DEFCON(E)$
```

```
(C6) DEFCON(E,E,KDELTA)$

(C7) DEFCON(L,L,W)$

(C8) W(L1,L2):=0$

(C9) COMPONENTS(G([P,Q],[]),E([P,Q],[])+2*M*L([P],[])*L([Q],[]))$

(C10) COMPONENTS(G([],[A,B]),E([],[A,B])-2*M*L([],[A])*L([],[B]))$

(C11) SHOW(G([I,J],[]));
(D11)                                2 L  L  M + E
                                        I  J      I J

(C12) SHOW(G([],[I,J]));
                                   I J      I  J
(D12)                            E     - 2 L  L  M

(C13) METRIC(G)$

(C14) CONTRACT(RENAME(EXPAND(G([I,J],[])*G([],[J,K]))))$

(C15) SHOW(%);
                                              K
(D15)                                   KDELTA
                                              I

(C16) DISPCON(ALL);
(D16)      [[[E, E, KDELTA], [E]], [[L, L, W]], [[G, G, KDELTA], [G]]]
```

1.9 Indicial Tensor Manipulation- Property Display Functions

DISPCON*(tensor1,tensor2,...)* displays the contraction properties of the *tensori* which were given to DEFCON. DISPCON(ALL) displays all defined contraction properties as the example under DEFCON illustrates.

DISPSYM*(tensor, m, n)* displays symmetries declared by DECSYM as a list of lists or returns [] if there are none (see the example under DECSYM). The first element of the inner list is one of the atoms SYM, ANTI or CYC. The second element is a list of lists of the index positions that have this property in the covariant indices of *tensor*. The third element is the same except that it is for the contravariant indices.

1.10 Indicial Tensor Manipulation- Property Removal Functions

REMCOMPS*(tensor)* unbinds all values from *tensor* which were assigned with the COMPONENTS function.

REMCOORD*(tensor1,tensor2,...)* removes the coordinate differentiation property from the *tensori* that was established by the function COORD. REMCOORD(ALL) removes this property from all indexed objects.

REMCON*(tensor1,tensor2,...)* removes all the contraction properties from the *tensori*. REMCON(ALL) removes all contraction properties from all indexed objects.

REMSYM*(tensor,m,n)* removes all symmetry properties from *tensor* which has *m* covariant indices and *n* contravariant indices.

1.11 Indicial Tensor Manipulation- Indexing Functions

COUNTER determines the numerical suffix to be used in generating the next dummy index. It may also be used to set the counter to any value (see the example under INDICES).

DUMMY*()* increments COUNTER and returns as its value an index of the form %n where n is a positive integer. This guarantees that dummy indices which are needed in forming expressions will not conflict with indices already in use (see the example under INDICES).

DUMMYX is the prefix for dummy indices (see the example under INDICES).

INDICES*(exp)* returns a list of two elements. The first is a list of the free indices in *exp* (those that occur only once). The second is the list of the dummy indices in *exp* (those that occur exactly twice) as the following example demonstrates.

```
(C3) SHOW(RIEMANN([I,J,K],[L])*RIEMANN([A,B,C],[D]));
                D             D           %2           D             D           %2
(D3) (- CHR2          - CHR2       CHR2     + CHR2         + CHR2        CHR2     )
                A C,B         %2 B        A C          A B,C         %2 C        A B

                   L             L           %1           L             L           %1
          (- CHR2          - CHR2       CHR2     + CHR2         + CHR2        CHR2     )
                   I K,J         %1 J        I K          I J,K         %1 K        I J

(C4) INDICES(%);
(D4)                   [[D, C, A, B, L, K, I, J], [%2, %1]]

(C5) COUNTER;
(D5)                                 2

(C6) COUNTER:11$

(C7) ''C3;
                D             D            %13          D             D            %13
(D7) (- CHR2          - CHR2        CHR2     + CHR2         + CHR2         CHR2     )
                A C,B         %13 B        A C          A B,C         %13 C        A B
```

```
                 L              L                  L              L
                                      %12                                   %12
        (- CHR2        - CHR2        CHR2    + CHR2        + CHR2         CHR2     )
                 I K,J          %12 J      I K         I J,K         %12 K      I J

(C8) DUMMYX;
(D8)                                      %

(C9) DUMMYX:&$

(C10) ''C3;
                  D              D                  D              D
                                       &15                                   &15
(D10) (- CHR2        - CHR2        CHR2    + CHR2        + CHR2         CHR2     )
                  A C,B          &15 B      A C        A B,C         &15 C      A B

                 L              L                  L              L
                                      &14                                   &14
        (- CHR2        - CHR2        CHR2    + CHR2        + CHR2         CHR2     )
                 I K,J          &14 J      I K         I J,K         &14 K      I J
```

1.12 Indicial Tensor Manipulation- ITENSR --> CTENSR

GENERATE*(eqn)* converts an ITENSR equation *eqn* to a CTENSR assignment statement. Implied sums over dummy indices are made explicit while indexed objects are transformed into arrays (the array subscripts are in the order of covariant followed by contravariant indices of the indexed objects). The derivative of an indexed object will be replaced by the noun form of DIFF taken with respect to OMEGA subscripted by the derivative index. The Christoffel symbols CHR1 and CHR2 will be translated to LCS and MCS respectively and if METRICCONVERT[TRUE] is TRUE then all occurrences of the metric with two covariant (contravariant) indices will be renamed to LG (UG). In addition, DO loops will be introduced summing over all free indices so that the transformed assignment statement can be evaluated by just doing EV(...). The following examples demonstrate the features of this function.

```
(C11) SHOW(X);
                                      L        K  I    I   J
(D11)                           G = F  A    (C  B  + D ) E
                                        I J     K         L

(C12) GENERATE(X);
(D12) G : SUM(SUM(SUM(F   A      (SUM(C  B    , K, 1, DIM) + D ) E    ,
                       L  I, J         K  K, I                 I  L, J

                           I, 1, DIM), J, 1, DIM), L, 1, DIM)

(C4) SHOW(T([I],[J]));
                                          J
(D4)                                    T
                                          I

(C5) SHOW(COVDIFF(%,K));
                          J      %1     J         J    %1
(D5)                   - T   CHR2    + T    + CHR2     T
                          %1      I K    I,K       %1 K  I

(C6) METRICCONVERT;
(D6)                                  TRUE
```

```
(C7) GENERATE(H([I,K],[J])=D5);

(D7)

FOR I THRU DIM DO (FOR J THRU DIM DO (FOR K THRU DIM DO H        :
                                                         I, K, J

- SUM(T      MCS            , %1, 1, DIM) + DIFF(T    , OMEGA )
      %1, J     I, K, %1                          I, J        K

 + SUM(MCS           T    , %1, 1, DIM)))
          %1, K, J    I, %1

(C8) METRIC(G)$

(C9) D5,CHR2$

(C10) SHOW(%);
         %1 %3   J
        G       T    (G          - G          + G          )
                 %1    K %3,I      I K,%3       I %3,K
(D10) - -----------------------------------------------
                             2

                  J %2  %1
                 G     T    (G          - G          + G          )
                        I     K %2,%1     %1 K,%2      %1 %2,K        J
             + ------------------------------------------------------ + T
                                       2                                I,K
```

```
(C11) GENERATE(H([I,K],[J])=D10);
(D11)

FOR I THRU DIM DO (FOR J THRU DIM DO (FOR K THRU DIM DO H        :
                                                        I, K, J

- SUM(SUM(UG        T        (DIFF(LG      , OMEGA )
            %1, %3  %1, J             K, %3        I

 - DIFF(LG      , OMEGA   ) + DIFF(LG      , OMEGA )), %1, 1, DIM), %3,
          I, K        %3              I, %3        K

1, DIM)/2 + SUM(SUM(UG        T       (DIFF(LG      , OMEGA   )
                      J, %2  I, %1            K, %2        %1

 - DIFF(LG       , OMEGA   ) + DIFF(LG       , OMEGA )), %1, 1, DIM), %2,
          %1, K        %2              %1, %2        K

1, DIM)/2 + DIFF(T     , OMEGA )))
                  I, J        K
```

[1.13] Acknowledgments

These program systems, first begun in 1973, have undergone several stages of development during the last 7 years. One of us (RP) wishes to thank Alexander Doohovskoy, David Grabel, Richard Zippel and especially Richard Bogen for their efforts in coding the routines described above. In addition, thanks are extended to the many staff members of the Laboratory of Computer Science (past and present) who contributed such a great deal to the current state of this project.