Acta Applicandae Mathematicae 1, 43–77. 0167–8019/83/0011–043$05.25.

Symbolic Computations in Applied Differential Geometry

P. K. H. GRAGERT, P. H. M. KERSTEN, and R. MARTINI
Department of Applied Mathematics, Twente University of Technology, PO Box 217, 7500 AE Enschede, The Netherlands

(Received: 4 November 1982)

Abstract. The main aim of this paper is to contribute to the automatic calculations in differential geometry and its applications, with emphasis on the prolongation theory of Estabrook and Wahlquist, and the calculation of invariance groups of exterior differential systems. A large number of worked examples have been included in the text to demonstrate the concrete manipulations in practice. In the appendix, a list of programs discussed in the paper is added.

AMS (MOS) subject classifications (1980). 35 Partial Differential Equations, 53 Differential Geometry, 68 Computer Science.

Key words. Symbolic computations, differential geometry, prolongation theory, exterior differential calculus, exterior differential systems, partial differential equations, invariance groups

0. Introduction

The impact of the computer on mathematics and its related fields is well-known. For instance, we are all familiar with its applications to numerical analysis, which started some decades ago. Its utility has also been proved in discrete mathematics, number theory and algebra (finite groups).

Perhaps, less well-known is the recent progress of the application of symbolic calculations in the more continuous parts of mathematics, such as mathematical analysis, differential equations, differential geometry and its applications in theoretical physics. In this context, we mention some outstanding results: the Risch algorithm to calculate indefinite integrals [17] and the program package SHEEP for formula manipulation, primarily in General Relativity [6].

The main aim of this paper is to contribute to the automatic calculations in differential geometry and its applications, with emphasis on the prolongation theory of Estabrook and Wahlquist [4, 5], and the calculations of invariance groups of exterior differential systems [7, 10]. A second aim is to encourage the reader to use symbolic calculations in his own field of interest.

In contrast to the strategy of most papers related to symbolic calculations, which only discuss the theory and give final results of computations, we shall present by worked examples the explicit manipulations which have to be carried out in order to get to the results. An obvious but nonnegligible circumstance is the fact that, due to

the use of symbolic calculations, one may carry out easy, although 'long and tedious' calculations with the computer, thus avoiding lots of elementary mistakes, such as wrong signs, missing brackets, omitted symbols, etc.

There are several systems designed for symbolic calculations, e.g., FORMAC, SAC, MACSYMA, REDUCE, and we have chosen the language REDUCE [3] as a basis for our developments. It may be implemented on any computer system supporting LISP [14] and it is cheap and easily available and, consequently, widespread. Last, but not least, we are charmed by the interactive facilities of REDUCE.

PLAN OF THE ARTICLE

Essentially, the paper consists of three parts. First, in order to become more familiar with REDUCE, some typical symbolic calculations are given. The second part is concerned with the basic manipulations of vector fields and differential forms (exterior differential calculus). The third part describes the application to infinitesimal symmetries of exterior differential systems. In the appendix, a list of REDUCE procedures of the described material is included.

Before discussing the three parts in more detail, we shall make some general remarks relevant for all worked examples in each section.

(1) In the examples, a terminal session with the computer is reproduced in printout form.
(2) Input lines always begin with a '*', which is the prompt character of REDUCE, indicating that the system is waiting for a command, while the command itself is written in small letters.
(3) Output from the system is printed in capital letters.
(4) Results of commands terminated by a ';' are printed.
(5) Results of commands terminated by a '$' are not printed.
(6) On each separate input line text following a '%' is ignored by REDUCE, and therefore used as a means of including comment.
(7) In input, the parentheses of functions with a single argument are not required and, therefore, often omitted.

In Section 1, three examples illustrating the capabilities of REDUCE are presented: (a) a procedure for finite Taylor series expansions, which is applied to concrete cases; (b) the integration package; and (c) matrix calculations related to the theory of Lie algebras.

In Section 2 an implementation of differential geometric objects and their operations are described. The concepts of exterior differential calculus are successively expounded. Keywords are: representation of differential forms and vector fields, exterior product, exterior derivative, commutator of vector fields, inner product of vector fields and forms and Lie derivatives of forms.

Section 3 is concerned with the computation of infinitesimal symmetries of exterior differential systems, representing differential equations. Determination of these symmetries gives rise to overdetermined systems of partial differential equations.

These are derived by the machinery of the Section 2. In order to solve such systems, frequently-occurring computations are automized. Further, some auxiliaries are developed. The procedure is illustrated by two examples. Finally, the complete algebra of the infinitesimal symmetries for the Lin–Reisner–Tsien equation is given.

1. Introduction to Symbolic Computations

By means of some examples, which are chosen such that they are still computable by hand, we shall give a sample of the interactive symbolic computations in REDUCE. All functions not explicitly described in this section are already available in the basic REDUCE system. We recommend that the reader without experience consult the *REDUCE Users' Manual* [11]. Moreover, one has to be informed how to start a REDUCE session.

(A) TAYLOR SERIES

Our first example will be a *PROCEDURE* for finite Taylor series expansions. Here, two special system functions are used, one for differentiation and one for substitution.

```
df(function,variable)
```

This command differentiates 'function' with respect to 'variable'.

```
sub(x=pt,expression)
```

Every 'x' occurring in 'expression' is substituted by 'pt'.

Note: Only the simplest form of DF and SUB are introduced here.

In order to illustrate the Taylor series expansions for various functions, we write a *PROCEDURE*:

```
*procedure taylor(fn,x,pt,n);begin scalar fakt;
*     fakt:=1;
*     return
*     sub(x=pt,fn)+
*     for i:=1:n sum
*     sub(x=pt,fn:=df(fn,x))*
*     (fakt:=fakt*(x-pt)/i)
*end taylor;
TAYLOR
```

The four parameters of the PROCEDURE TAYLOR have the following meaning: FN has to be a function, X represents the variable with respect to which the function FN is expanded about the point PT up to order N.

EXAMPLE 1.1. The Taylor series expansion $T(X)$ of a rational function $F(X)$ with two parameters a and b:

$$F(X) = \frac{2X^2 + X + 1}{bX^2 + aX + 2}$$

about 0 up to order three is constructed. Moreover, we give the rest $R(X)$, i.e., $R(X) = F(X) - T(X)$.

```
*operator f,t,r;

*f x:=(2*x^2+x+1)/(b*x^2+a*x+2);

                2                 2
F(X) := (2*X  + X + 1)/(X *B + X*A + 2)

*t x:=taylor(f x,x,0,3);

              3            3       3  3       3  2       3
T(X) := (4*X *B*A - 4*X *B - X *A  + 2*X *A  - 8*X *A - 4*

           2         2  2       2          2
          X *B + 2*X *A  - 4*X *A + 16*X  - 4*X*A + 8*X + 8)/

         16

*r x:=f x-t x;

            4           2           2         3          2
R(X) := (X *( - 4*X*B *A + 4*X*B  + X*B*A  - 2*X*B*A  + 8*X*

                        2          2                      4       3
               B*A + 4*B  - 6*B*A  + 8*B*A - 16*B + A  - 2*A

                     2          2
              + 8*A  ))/(16*(X *B + X*A + 2))
```

□

The following example illustrates that the expressions may be expanded with respect to different variables:

EXAMPLE 1.2.

```
*taylor(sin(x+2*a),x,0,3);

                3                                  2
( - COS(2*A)*X  + 6*COS(2*A)*X - 3*SIN(2*A)*X  + 6*SIN(2*A))/6

*taylor(sin(x+2*a),a,0,3);

                3                                2
( - 4*COS(X)*A  + 6*COS(X)*A - 6*SIN(X)*A  + 3*SIN(X))/3
```

□

(B) INTEGRATION

The integration package of REDUCE is called by

```
int(function,variable)
```

and its application is demonstrated in the following example

EXAMPLE 1.3.

```
*h:=2*x*atan x/(1+x^2)^2;
                              4       2
H := (2*ATAN(X)*X)/(X  + 2*X  + 1)
```

H is now a function of x.

```
*hh:=int(h,x);

                    2                              2
HH := (ATAN(X)*X  - ATAN(X) + X)/(2*(X  + 1))

*h-df(hh,x);

0

*int(x*e^(x^2),x);

   2
 (X )
E    /2

*int(e^(-x^2),x);

    2                  2                2
  (X )          2    (X )           (X )
(E    *INT((2*X )/E     ,X) + X)/E

*on div;
```

DIV is an output switch: each term gets its denominator. The last algebraic result is always stored under the name !*ANS.

```
*!*ans;
       2                          2
 ( - X )                  2    (X )
E        *X + INT((2*X )/E     ,X)
```

□

(C) MATRIX CALCULATIONS

Matrix operations will be illustrated by calculations involving two different matrices.

EXAMPLE 1.4. The first matrix represents the adjoint matrix

$$\mathrm{ad}(Ah + Bx + Cy)$$

of the Lie algebra $\mathrm{sl}(2)$ with respect to the standard basis h, x, y, satisfying the commutator relations

$$[h, x] = 2x, \qquad [h, y] = -2y, \qquad [x, y] = h.$$

$Ah + Bx + Cy$ represents a 'general' element of $\mathrm{sl}(2)$. For convenience, the matrix

$\mathrm{ad}(Ah + Bx + Cy)$ is called ADSL2 and equals:

$$\begin{bmatrix} 0 & -C & B \\ -2{*}B & 2{*}A & 0 \\ 2{*}C & 0 & -2{*}A \end{bmatrix}$$

```
*matrix adsl2(3,3);

*adsl2(1,1):=0$    adsl2(1,2):=-C$  adsl2(1,3):=B$
*adsl2(2,1):=-2*B$ adsl2(2,2):=2*A$ adsl2(2,3):=0$
*adsl2(3,1):=2*C$  adsl2(3,2):=0$   adsl2(3,3):=-2*A$
```

Let us calculate the determinant and the characteristic polynomial in the variable T of adsl 2:

```
*det adsl2;
0
```

We construct the 3×3-identity matrix $I3$ as follows:

```
*matrix i3(3,3);
*for i:=1:3 do i3(i,i):=1;
```

Then the characteristic polynomial equals:

```
*det(adsl2 - t*i3);
                  2    2
T*(4*B*C + 4*A  - T )
```

This agrees with the fact that ADSL2 has to be singular and we find two roots with opposite signs if $BC + A^2 \neq 0$.

To show a second matrix operation, taking the inverse, we compute $(\mathrm{ADSl}\,2 - T{*}I3)^{-1}$. Moreover, to shorten the output we introduce the name DETSL2 for the determinant of this matrix.

```
*let t*(4*b*c+4*a^2-t^2)=detsl2;

*(adsl2-t*i3)^(-1);

                        2     2
MAT(1,1) := ( - 4*A  + T )/DETSL2

MAT(1,2) := (C*( - 2*A - T))/DETSL2

MAT(1,3) := (B*( - 2*A + T))/DETSL2

MAT(2,1) := (2*B*( - 2*A - T))/DETSL2

                                    2
MAT(2,2) := ( - 2*B*C + 2*A*T + T )/DETSL2

                      2
MAT(2,3) := ( - 2*B )/DETSL2

MAT(3,1) := (2*C*( - 2*A + T))/DETSL2

                      2
MAT(3,2) := ( - 2*C )/DETSL2

                                    2
MAT(3,3) := ( - 2*B*C - 2*A*T + T )/DETSL2
```

□

Computations with matrix-polynomials are possible too and, of complication, comparable with the given examples.

EXAMPLE 1.5. The second matrix represents the matrix of the Killing form of the Lie algebra with the following commutator table. Here the Lie product $[x_i, x_j]$ is represented by LIE(i, j) and only nonzero commutators are printed.

This Lie algebra emerges in the determination of a 11-dimensional prolongation algebra of the Korteweg–de Vries equation [4, 8].

lie(1, 3) := x^5
lie(1, 4) := − x^6*M − x^5*L
lie(1, 5) := (x^4 + x^3*L − x^2)/M
lie(1, 6) := − x^4
lie(2, 3) := − x^6 − x^5*M
lie(2, 4) := x^6*L
lie(2, 5) := − x^3*L + x^2
lie(2, 6) := (x^4*L + x^3*L^2 − x^2*L)/M
lie(3, 4) := x^6
lie(3, 5) := − x^3
lie(3, 6) := (x^4 + x^3*L − x^2 − x*M)/M
lie(4, 5) := x^2 + x*M
lie(4, 6) := − x^2*M + x*(− M^2 + L)
lie(8, 11) := − x^7
lie(9, 10) := x^7

The matrix C of the Killing form of the 6-dimensional sub-algebra spanned by $x_1, \ldots, x_6$ equals:

```
*matrix c(6,6);
```

After a declaration of a matrix, all elements are initialized to zero, therefore we have to give only the nonzero elements of C.

```
*c(1,1):=2*m$                    c(1,2):=c(2,1)-4*1$
*c(2,2):=2*m*1$                  c(2,3):=c(3,2):=2*m$
*c(1,3):=c(3,1):=-4$             c(3,4):=c(4,3):=-2*m$
*c(4,4):=2*m*(-m^2+3*1)$         c(5,5):=4$
*c(5,6):=c(6,5):=-2*M$           c(6,6) := 2*(m^2- 2*1)$
```

We compute the determinant of C:

```
*det c;

     2   6         4           2  2         3
64*M *(M  - 12*M *L + 48*M *L  - 64*L )
```

This result shows that this subalgebra is semisimple for almost all values of the parameters L and M. □

2. Vector and Form Manipulations in REDUCE

Using the concepts of differential geometry in applied mathematics, we have to manipulate the vector fields and differential forms, which are 'living' on a differentiable manifold. All manipulations are essentially simple and straightforward, but doing computations by hand, in particular on high-dimensional spaces, the amount of labour grows rapidly and becomes rather tedious. Moreover, there is every chance that errors are introduced. For instance, a slip of the pen or a wrong sign is very likely. Manipulations with vector fields and forms are so systematic that they are very well suited for symbolic computations. In doing so only input, which is usually very short, has to be checked carefully.

In this section, almost all the computations are carried out with respect to a local coordinate system for an n-dimensional manifold M. So, in fact, we assume that M is a domain U of the Euclidean space R^n. All objects are assumed to be infinitely differentiable.

0-FORMS

The ring of all functions on U will be denoted by $\mathscr{A}^0(U)$. 0-forms on U are identified with elements of $\mathscr{A}^0(U)$.

The *algebraic* mode of REDUCE already contains all tools to manipulate within the ring $\mathscr{A}^0(U)$. In practice, the first problem to solve is to introduce local coordinates in a flexible way. The coordinates should be introduced such that indicial calculation is allowed and also such that a user may give them convenient names. This problem is solved by using the *algebraic operator* VNAT (*nat*ural *v*ariables), where VNAT(1), VNAT(2), ..., VNAT(n) represents a coordinate system of U. The name of the ith coordinate may be changed into another, more suitable name by: VNAT(i) := 'name';

EXAMPLE 2.1. The coordinates ought to be x, y, z and u_1, u_2. This is introduced to REDUCE by:

```
*d!@dif:=5$
*vnat 1:=x$ vnat 2:=y$ vnat 3:=z$
*operator u; vnat 4:=u 1; vnat 5:=u 2$

VNAT(4) := U(1)
```

Note: $U(i)$ is used instead of u_i. □

Now let us define two functions $f(1)$ and $f(2)$, add and substract them, and let $f(3)$ be the product of $f(1)$ and $f(2)$.

EXAMPLE 2.2

```
*operator f;
*f 1:=for i:=1:5 sum i*vnat i;
```

```
F(1) := 4*U(1) + 5*U(2) + X + 2*Y + 3*Z

*f 2:=sin(vnat 1+vnat 4)+3*(1+vnat(1)^2);
                                    2
F(2) := SIN(U(1) + X) + 3*X  + 3
```

An alternative way of input for $f(2)$ is:

```
*f 2:=sin(x+u 1)+3*(1+x^2)$
```

Any 'mixed form' of input is allowed too, and yields the same result.

```
*f 1+f 2;
                                                    2
4*U(1) + 5*U(2) + SIN(U(1) + X) + 3*X  + X + 2*Y + 3*Z + 3

*f 1-f 2;
                                                    2
4*U(1) + 5*U(2) - SIN(U(1) + X) - 3*X  + X + 2*Y + 3*Z - 3

*f 3:=f 1*f 2;
                                                        2
F(3) := 4*U(1)*SIN(U(1) + X) + 12*U(1)*X  + 12*U(1) + 5*U(2)

          *SIN(U(1) + X) + 15*U(2)*X  + 15*U(2) + SIN(U(1) + X
                                   2

          )*X + 2*SIN(U(1) + X)*Y + 3*SIN(U(1) + X)*Z + 3*X
                                                          3

           + 6*X *Y + 9*X *Z + 3*X + 6*Y + 9*Z
                2         2
```

Here the construction of $f(1)$ shows the advantage of indicial calculations. □

Note: Observe that the examples show different ways of giving names.

1-FORMS

The set of 1-forms on the domain U is denoted by $\mathscr{A}^1(U)$. If $x_1, x_2, \ldots, x_n$ is a coordinate system of U, then:

$$BF(1) = \{ \mathrm{d}x_1, \mathrm{d}x_2, \ldots, \mathrm{d}x_n \}$$

is a basis for the $\mathscr{A}^0(U)$-module $\mathscr{A}^1(U)$.

A general 1-form can be written in the form:

$$u = a_1\, \mathrm{d}x_1 + a_2\, \mathrm{d}x_2 + \cdots + a_n\, \mathrm{d}x_n,$$

where $a_1, a_2, \ldots, a_n$ are elements of $\mathscr{A}^0(U)$.

In REDUCE, we represent the basis 1-forms by WEDGE(i), where WEDGE has to be declared an algebraic operator.

EXAMPLE 2.2. Fixing the dimension of U to 3 and introducing coordinates $X1$, $X2$ and $X3$, a general 1-form in REDUCE can be constructed with, e.g., an

algebraic operator A as follows:

```
*u:=for i:=1:3 sum a(i,x1,x2,x3)*wedge i;

U := A(1,X1,X2,X3)*WEDGE(1) + A(2,X1,X2,X3)*WEDGE(2) + A(3,
     X1,X2,X3)*WEDGE(3)
```

A more concrete 1 – form is:

```
*v:=x2*wedge(1)-wedge(3);

V := X2*WEDGE(1) - WEDGE(3)
```

□

MULTI-INDEX NOTATION

To introduce differential forms of a higher order, a multi-index notation will be used.

Let

$$I = (i_1 i_2 \cdots i_m)$$

be a list of m-ordered integers, a multi-index of order m. Then $\mathrm{d}x_I$ is an abbreviation of the following m-form:

$$\mathrm{d}x_I = \mathrm{d}x_{i_1} \wedge \mathrm{d}x_{i_2} \wedge \cdots \wedge \mathrm{d}x_{i_m}$$

Let

$$II(n, m) = \{(i_1 i_2 \cdots i_m) \mid 1 \leqslant i_1 < i_2 \cdots < i_m \leqslant n\}$$

be the index set of all strictly-increasing ordered multi-indices of order $m(\leqslant n)$. This notation is used in the following.

FORMS OF HIGHER ORDER

A basis for the $\mathscr{A}^0(U)$-module of m-forms of $\mathscr{A}^m(U)$ is

$$BF(m) = \{\mathrm{d}x_I \mid I \in II(n, m)\}$$

where the dimension of U equals n.

An arbitrary m-form u can be written uniquely as:

$$u = \sum_{I \in II(n,m)} a_I \, \mathrm{d}x_I.$$

As usual $\mathscr{A}^m(U) = 0$ for $m > n$, by definition.

In REDUCE $\mathrm{d}x_I$ with $I = (i_1 i_2 \cdots i_m)$ becomes:

$$\mathrm{WEDGE}(i_1, i_2, \ldots, i_m).$$

EXAMPLE 2.3. Let the coordinate system of U be $X(1)$, $X(2)$, $X(3)$, $X(4)$ when X is

an algebraic operator, then two concrete forms are:

```
*f1 :=x(1)*wedge(2,3)-x(3)^2*wedge(1,4);
                                  2
F1 := X(1)*WEDGE(2,3) - X(3) *WEDGE(1,4)

*f2 := a(1,x(1))*wedge(1,2,4) + wedge(1,2,3);

F2 := A(1,X(1))*WEDGE(1,2,4) + WEDGE(1,2,3)
```

$F1$ is a 2-form and $F2$ is a 3-form.

□

THE WEDGE PRODUCT OF FORMS

The $\mathscr{A}^0(U)$-module of all differential forms on U equals:

$$\mathscr{A}(U) = \mathscr{A}^0(U) \oplus \cdots \oplus \mathscr{A}^n(U)$$

It becomes an algebra if the usual wedge product

$$\wedge : \mathscr{A}^r(U) x \mathscr{A}^s(U) \to \mathscr{A}^{r+s}(U) \quad (r, s \geqslant 0)$$

is introduced. We recall that $\wedge$ is bilinear and alternating. Therefore, it suffices to define $\wedge$ on basis forms in $BF(r)$ and $BF(s)$.
In order to implement this idea we have to decompose forms:

$$\sum_{I \neq 0} a_I \, \mathrm{d}x_I + a_0$$

into a_I, $\mathrm{d}x_I$ and a_0 for all occurring multi-indices I with $a_I \neq 0$.
Actually, such a differential form is written as

$$\sum_{i=0}^{N} K(i)^* \text{ WEDGE (arg}(i)) + \text{REST},$$

where $K(i)$ corresponds to an $a_I (\neq 0)$, WEDGE(arg(i)) to some $\mathrm{d}x_I$ and REST to a_0.

This decomposition is performed by the function OPCOEFF, which depends on three parameters: OPCOEFF (EX, OP, AR), where EX is the expression that has to be decomposed with respect to the operator OP. EX also has to be linear with respect to OP, WEDGE in this case. The third parameter is used to retrieve the $K(i)$ and WEDGE (arg (i)). REST will be the value of a global variable !@RESTOP-COEFF. The value of OPCOEFF is the number (-1) of terms $K(\cdots)$*OP$(\cdots)$ in the expression EX. AR becomes a function of two parameters: AR$(0, i)$ with value one of the $\mathrm{d}x_I$ written as WEDGE$(\cdots)$ and AR$(1, i)$ has the corresponding coefficient $K(i)$ as its value.

Internally, the numerator of EX is checked for kernels with operator OP by an auxiliary function GETFIRSTOP. If successful, the corresponding coefficient of

this kernel is determined and intermediate results are stored in a list. The rest is examined in the same way. A second auxiliary function M!@A2D makes a function from AR which operates on the list of intermediate results, acting as described above.

Summarizing, the expression EX can be reconstructed as follows:

```
!@restopcoeff + for i:=0:n sum ar(1,I)*ar(0,I) .
```

Let us decompose $f1$, introduced in Example 2.3.

EXAMPLE 2.4.

```
*d1:=opcoeff(f1,wedge,cc);

D1 := 1
```

One way to show the effect of opcoeff:

```
*for i:=0:d1 do write cc(0,i),"   ",cc(1,i);
                      2
WEDGE(1,4)   - X(3)

WEDGE(2,3)   X(1)
```

A second way to show the effect of opcoeff:

```
*!@restopcoeff+for i:=0:d1 sum cc(0,i)*cc(1,i);
                           2
X(1)*WEDGE(2,3) - X(3) *WEDGE(1,4)

*!*ans-f1;

0
```

This is again a check of the result. □

Now we can deal with the basic problem, how to compute

$$dx_I \wedge dx_J, (dx_I \in BF(r), dx_J \in BF(s))$$

because differential forms in our implementation in REDUCE are essentially represented by a list of strictly increasing integers, the multi-index. The algebraic operator WEDGE serves only as a carrier and a medium for input and output purposes. Indeed, the implementation allows any convenient different name for WEDGE.

It suffices to construct a PROCEDURE which merges two multi-indices into a new multi-index and which, at the same time, gives the sign of the resulting product or zero, if at least one integer occurs twice in the concatenated multi-index of I and J. Therefore, a function is written to check whether at least two elements in a list occur twice.

There exists a system function in REDUCE, which orders a list of integers in decreasing order and another that reverses the order of a list.

Finally, a system function is used, which establishes the permutation class of two equally long lists of integers. All these ingredients are joined together in the procedure

MULFORM (FORM1, FORM2), which effectuates the wedge product on $\mathscr{A}(U) \times \mathscr{A}(U)$, e.g., if

$$f1 = \sum a_I dx_I + a_0, \qquad f2 = \sum b_J dx_J + b_0$$

are two forms in $\mathscr{A}(U)$ then in order to compute $f1 \wedge f2$ one has to convert $f1$ and $f2$ in our notation, as shown in the preceding examples. $f1 \wedge f2$ then becomes: mulform $(f1, f2)$.

Note: Because the manipulation of lists of integers is very effective and 'cheap', this implementation is faster compared with other implementations [3] of the wedge product of differential forms.

EXAMPLE 2.5.

```
*f1:=x 1*wedge(2,3)-x 3^2*wedge(1,4);

                                    2
F1 := X(1)*WEDGE(2,3) - X(3) *WEDGE(1,4)

*f2:=a(1,x 1)*wedge(1,2,4)+wedge(1,2,3);

F2 := A(1,X(1))*WEDGE(1,2,4) + WEDGE(1,2,3)

*mulform(f1,f2);

0

*mulform(f1,wedge 5);

                          2
 - WEDGE(1,4,5)*X(3)  + WEDGE(2,3,5)*X(1)

*mulform(f2,wedge 5);

WEDGE(1,2,3,5) + WEDGE(1,2,4,5)*A(1,X(1))

*mulform(wedge 5,f2);

 - (WEDGE(1,2,3,5) + WEDGE(1,2,4,5)*A(1,X(1)))
```

□

EXTERIOR DIFFERENTIATION OF FORMS

Comparing exterior differentiation with the wedge product, there remains only one computational problem, the computation of total derivatives of 0-forms. Fortunately, this is very easy to achieve in our approach: For example, let G be an element of $\mathscr{A}^0(U)$, then dG is computed by:

```
dg :=for i:=1:d!@d!f sum df(f,vnat(i))*wedge(i)
```

We call the function of exterior differentiation EXDF according to the system function DF.

EXAMPLE 2.6.

```
*d!@dif:=4;

D@DIF := 4

*for i:=1:4 do vnat i:=x i;

*f1:=x 1*wedge(2,4)-sin(x 3)*wedge 4+2*x 4$
*exdf f1;

2*WEDGE(4) + WEDGE(1,2,4) - COS(X(3))*WEDGE(3,4)
```

□

VECTOR FIELDS

The $\mathscr{A}^0(U)$-module of vector fields on $U \subset R^n$ is denoted by $\mathscr{X}(U)$.

$$\text{BVF} = \left\{ \frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n} \right\}$$

is a basis with respect to the coordinate system $x_1, x_2, \ldots, x_n$.
In fact it is the dual basis of BF(1), i.e.

$$\mathrm{d}x_i\left(\frac{\partial}{\partial x_j}\right) = \delta_{ij}$$

where δ_{ij} is the Kronecker symbol.

A vector field V on U may be written as

$$V = \sum_{i=1}^{n} a_i \frac{\partial}{\partial x_i} \quad (a_i \in \mathscr{A}^0(U)).$$

To implement the vector fields in REDUCE, we prefer to use the already-existing system functions. If we represent vector fields as polynomials, applying the linear isomorphism given by

$$\sum a_i \frac{\partial}{\partial x_i} \leftrightarrow \sum a_i D^i,$$

then the components a_i can be determined easily by the system function COEFF.

Note: There will be no confusion with this vector field notation, if we do not use the letter D in function names.

EXAMPLE 2.7. Let $n = 3$ and $x1, x2, x3$ be a coordinate system of U. Then, for instance, the vector field

$$V = x1\frac{\partial}{\partial x_i} - x2^3\frac{\partial}{\partial x_2} + \sin(x1\,x2)\frac{\partial}{\partial x_3}$$

becomes in REDUCE:

```
*v:=x1*d-x2^3*d^2+sin(x1*x2)*d^3$
```

To determine the coefficients of V and printing them, you may type the following commands:

```
*array aa 3; dim:=coeff(v,d,aa);

DIM := 3

*for i:=1:3 do write aa i:=aa i;

AA(1) := X1
                  3
AA(2) :=  - X2
AA(3) := SIN(X1*X2)
```

□

Note: Due to this approach, we can use the input and output facilities of REDUCE for vector fields. The ordinary operations with vector fields, addition and multiplication with functions in $\mathscr{A}^0(U)$ are therefore standard operations with polynomials in REDUCE.

If $V1$, $V2$ are vector fields belonging to $\mathscr{X}(U)$,

$$V1 = \sum_{i=1}^{n} a_i \frac{\partial}{\partial x_i}, \qquad V2 = \sum_{i=1}^{n} b_i \frac{\partial}{\partial x_i}$$

then the commutator $[V1, V2]$ of $V1$ and $V2$ is defined by

$$[V1, V2] = \sum_{i=1}^{n} \left(a_i \sum_{j=1}^{n} \left(\frac{\partial}{\partial x_i} b_j \right) \frac{\partial}{\partial x_j} - b_i \sum_{j=1}^{n} \left(\frac{\partial}{\partial x_i} a_j \right) \frac{\partial}{\partial x_j} \right)$$

This formula can be implemented directly, where the COEFF function is used to determine the a_i, b_i $(i = 1, 2, \ldots, n)$. Obviously, the summations can be effected by FOR-loops. The commutator function is called COM and will be demonstrated by

EXAMPLE 2.8.

```
*fload opcoef,com;     % the necessary programs are loaded

EXECUTE: INICOM(DIMENSION OF PROBLEM,NAME OF VARIABLES) !!

*inicom(3,x);          % as we are asked to do

D!@DIF = 3

THE SPECIAL VECTOR FIELDS VF(I) ARE INITIALIZED TO 0
1 <= I <= 3
AND SHOULD CHANGED BY: CH(CHANGE,N),
THAT MEANS, CH(CHANGE,N) HAS AS EFFECT: VF(N):= VF(N)+CHANGE;
PLUS OTHER SIDE EFFECTS

TO USE THE SPECIAL VECTOR FIELDS, USE ONLY THE INDEX
OF THEM AS PARAMETER TO COM
```

```
THE LOCAL COORDINATES ARE X1,...,X3

INICOMEND

*v1:=x3*d$  v2:=x1*d^2-d^3$  ch(v1,1);  ch(v2,2)$

CHEND
```

Now two special vector fields are initialized.

```
*on test;

368 MS
```

By this switch, execution times are printed too, and differences in execution time can be observed.

```
*com(1,2);

 2
D *X3 + D

30 MS

*com(v1,v2);              % the same commutator again

 2
D *X3 + D

47 MS

*com(1,x1*d-sin x2*d^2+x3*cos x1*d^3);

     3    2
 - D *X3 *SIN(X1) + D*X3*( - COS(X1) + 1)

104 MS
```

□

INNER PRODUCT OF VECTOR FIELDS AND FORMS

The inner product of vector fields and differential forms is a $\mathscr{A}^0(U)$-bilinear mapping:

$$\mathrm{IP}: \mathscr{X}(U) \times \mathscr{A}^m(U) \to \mathscr{A}^{m-1}(U)$$

where $\mathscr{A}^{-1}(U) = 0$ by definition.

It suffices to define the action of IP on basis elements BVF and BF(i), i.e., to define

$$\mathrm{IP}\left(\frac{\partial}{\partial x_i}, \mathrm{d}x_I\right).$$

In our implementation in REDUCE this action can be described as follows:

$$\mathrm{IP}\left(\frac{\partial}{\partial x_i}, \mathrm{d}x_I\right) = 0,$$

if i does not occur in the multi-index I. Otherwise, let

$$I = (i_1 \cdots i \cdots i_m)$$
$$J = (i i_1 \cdots \hat{i} \cdots i_m)$$
$$K = (i_1 \cdots \hat{i} \cdots i_m),$$

where, as usual,"^"over an index denotes the delation of this index. Because

$$dx_I = \pm dx_J,$$

with the sign depending on the permutation class of I and J, we have

$$\mathrm{IP}\left(\frac{\partial}{\partial x_i}, dx_I\right) = IP\left(\frac{\partial}{\partial x_i}, \pm dx_J\right) = \pm dx_K.$$

This manipulation with an index i and a multi-index I is implemented as a function B!@IP. Now the inner product IP is obtained by a simple composition of COEFF and OPCOEFF, to determine the coefficients of forms and vector fields, and B!@IP.

EXAMPLE 2.9.

```
*ip(d,wedge(1,2));

WEDGE(2)

*ip(d^2,wedge(1,2));

 - WEDGE(1)

*ip(x1*d^3-sin(x2)*d^2,x1*x3*wedge(1,3,4)+wedge(2)+x1);
      2
 - (X1 *X3*WEDGE(1,4) + SIN(X2))
```

□

LIE DERIVATIVES OF FORMS

To implement the Lie derivative of a differential form F with respect to a vector field V, one may write the following procedure:

```
*algebraic procedure liedf(v,f);
*exdf(ip(v,f))+ip(v,exdf(f));

LIEDF
```

As an illustration we give

EXAMPLE 2.10.
The local coordinates are $x1, \ldots, x5$.

```
*liedf(d-d^2,x1*x2*x3*wedge(1,2,3)-cos(x2)*wedge(1));

- WEDGE(1)*SIN(X2) - X1*X3*WEDGE(1,2,3) + X2*X3*WEDGE(1,2,3)
```

□

Note: One may obtain a somewhat faster procedure, if the existing functions EXDF and IP are not used as above, but instead, a procedure for the Lie derivative is constructed directly.

DISCUSSION OF DIFFERENT IMPLEMENTATIONS

In this section we have described in some detail an implementation of the exterior differential calculus in REDUCE. There are a few other implementations known, which are based on different designs.

First we mention an implementation by Edelen [3], based only on LET-statements. Experience with this system on a DEC-10 computer indicates that the system suffers from frequently-occurring garbage collections and is therefore substantially more time-consuming, compared with our implementation.

Second, even on a micro-computer, exterior calculus can be done. There exists a program package CARTAN for an 'apple'-computer designed by Wahlquist [18].

Third, one may want an implementation allowing a coordinate-free exterior differential calculus and with output coming closer to the usual notation.

Up to the now, we have only partly succeeded in fulfilling these wishes. This development will be illustrated by the following

EXAMPLE 2.11. Let us assume that we have 1-forms A, 2-forms B and 3-forms C, i.e., we may use A, $A(1)$ or $A(2)$ as 1-forms, etc.

```
*fload difnew; precedence mulformgen,cons;
*off raise;
```

From now on, small and capital letters are distinguished and in the remaining part of this example we deviate from the convention of writing input in small letters, because we want to indicate differentials by a small letter d.

```
*INITDF(3,X,Y,Z)$
```

The dimension of U is 3, coordinates are X, Y, Z.

```
*COMPOPDF A,B,2,C,3;
```

This command declares A, B, C as forms of the required degree. Here'^' is used as the infix operator for the wedge product

```
*H:=(A 1-X *A 2)^(A 4+A 2);

H :=  - A(2)^A(4)*X + A(1)^A(4) + A(1)^A(2)
```

The letter D is used as a function for exterior derivatives.

```
*D H;

dA(1)^A(2) - A(1)^dA(2) + dA(1)^A(4) - A(1)^dA(4) -
```

```
dX^A(2)^A(4) - dA(2)^A(4)*X + A(2)^dA(4)*X
*H2:=(B+Z*B 1)^(B+555*A);
H2 := B(1)^B*Z + 555*A^B(1)*Z + 555*A^B
*D H2;
555*dZ^A^B(1) + 555*dA^B(1)*Z - 555*A^dB(1)*Z + dZ^B(1)^B +
dB(1)^B*Z + dB^B(1)*Z + 555*dA^B + 555*dB^A
```

□

Note: On account of its experimental state, the details of this package are not included in the Appendix. We hope to report on this in future work.

3. Infinitesimal Symmetries of Exterior Differential Systems.

As an application of the above-developed software, we treat infinitesimal symmetries of partial differential equations by means of differential geometric methods. Following Cartan, partial differential equations can be described geometrically by exterior differential systems [2, 7, 10].

We consider an exterior differential system I in a domain U of a n-dimensional manifold M, and suppose that I is generated by the finite set of forms

$$\alpha(1), \ldots, \alpha(m). \tag{3.1}$$

By definition, infinitesimal symmetries are vector fields V such that

$$\mathscr{L}_V I \subset I \tag{3.2}$$

where $\mathscr{L}_V$ denotes the Lie derivative by the vector field V [7, 10]. In physical literature, infinitesimal symmetries are also called isovectors.

Note: In the case that I is generated by k-forms $\beta(1), \ldots, \beta(r)$ and their exterior derivatives, it suffices to require (3.2) to hold for these k-forms $\beta(1), \ldots, \beta(r)$ only, due to the fact that exterior derivative and Lie derivative commute

$$\mathrm{d}\mathscr{L}_V = \mathscr{L}_V \mathrm{d}. \tag{3.3}$$

The relation (3.2) is equivalent to

$$\mathscr{L}_V \alpha(i) - \sum_{j=1}^{m} \gamma(i,j) \wedge \alpha(j) = 0 \tag{3.4}$$

with $\gamma(i,j)$ suitably-chosen differential forms.

If $X(1), \ldots, X(n)$ are local coordinates in U and if we describe forms and fields locally in these coordinates

$$V = \sum_{i=1}^{n} V^i \frac{\partial}{\partial X(i)} \tag{3.5}$$

$$\alpha(i) = \sum_{i_1, \ldots, i_p} A_{i_1 \ldots i_p}(X)\, \mathrm{d}X(i_1) \wedge \cdots \wedge \mathrm{d}X(i_p),$$

then (3.4) yields the conditions on V in order to be an infinitesimal symmetry. This leads to an overdetermined system of partial differential equations for the functions

$$V^i \quad (i = 1, \ldots, n).$$

EXAMPLE 3.1. In this example we shall derive the first set of partial differential equations for the determination of the infinitesimal symmetries of Burgers' equation,

$$U_T = U_{XX} + 2UU_X.$$

The ideal I in $R^4 = \{(X(1), X(2), X(3), X(4))\} = \{(X, T, U, U_X)\}$ is generated by

$$\alpha(1) = -\mathrm{d}T \wedge \mathrm{d}U - U_X\,\mathrm{d}X \wedge \mathrm{d}T$$
$$\alpha(2) = -\mathrm{d}T \wedge \mathrm{d}U_X - \mathrm{d}X \wedge \mathrm{d}U + 2UU_X\,\mathrm{d}X \wedge \mathrm{d}T$$

First we call the necessary programs and declare some algebraic operators.

```
*fload opcoef,diffor; operator x,v,alfa,aa,ver;

GIVE VALUES TO D!@DIF AND VNAT(I) I=3,...,D!@DIF
 (VNAT 1=X, VNAT 2=T)

*in tools.red,opl.red,ip.red,bevat.red,lie.red$

IP
EXECUTE: ARRAY A!@IP(MAXIMAL DIMENSION);
LIEDF
LE


*d!@dif:=4;

D@DIF := 4

*array a!@ip 4; for i:=1:4 do vnat i:=x i;
*vec:=for i:=1:4 sum v(i,var)*d^i;

        4               3              2
VEC := D *V(4,VAR) + D *V(3,VAR) + D *V(2,VAR) + D*V(1,VAR)
```

We introduce functions $V(I, \mathrm{VAR})$, meaning functions dependent on the variables $X(1), \ldots, X(4)$, generating the vector field VEC (V in (3.5))

```
*for all i,j let df(v(i,var),x j)=v(i,x j);
```

Due to the introduction of VAR in the functions $V(I, \mathrm{VAR})$ ($I = 1, 2, 3, 4$) we are able to 'create' partial derivatives $V(I, X(J))$ by this LET-statement, although $V(I, \mathrm{VAR})$ does not explicitly depend on $X(1), \ldots, X(4)$.

```
*alfa 1:=-wedge(2,3)-x 4*wedge(1,2)$
*alfa 2:=-wedge(1,3)+2*x 3*x 4*wedge(1,2)-wedge(2,4)$
*la:=liedf(vec,alfa 1)-a0*alfa 1-a1*alfa 2;

LA := WEDGE(3,4)*V(2,X(4)) + WEDGE(2,4)*(A1 + V(1,X(4))*X(4) - V

          (3,X(4))) + WEDGE(2,3)*(V(1,X(3))*X(4) - V(3,X(3)) +
```

```
A0 - V(2,X(2))) - WEDGE(1,4)*X(4)*V(2,X(4)) +
WEDGE(1,3)*( - V(2,X(3))*X(4) + A1 - V(2,X(1))) +
WEDGE(1,2)*( - V(1,X(1))*X(4) + V(3,X(1)) + A0*X(4) -
2*A1*X(3)*X(4) - X(4)*V(2,X(2)) - V(4,VAR))
```

LA = 0 is just requirement (3.4) for ALFA(1).

Using OPCOEFF we calculate the coefficients of the basis-forms, WEDGE (3, 4), ..., which have to be zero, and print them.

```
*aant:=opcoeff(la,wedge,cc);
AANT := 5
*for i:=0:5 do write aa(1,i):=cc(1,i);
AA(1,0) := V(2,X(4))
AA(1,1) :=  - X(4)*V(2,X(4))
AA(1,2) := A1 + V(1,X(4))*X(4) - V(3,X(4))
AA(1,3) :=  - V(2,X(3))*X(4) + A1 - V(2,X(1))
AA(1,4) :=  - V(1,X(1))*X(4) + V(3,X(1)) + A0*X(4) - 2*A1*X(
           3)*X(4) - X(4)*V(2,X(2)) - V(4,VAR)
AA(1,5) := V(1,X(3))*X(4) - V(3,X(3)) + A0 - V(2,X(2))
```

$AA(1, J)$ are the coefficients of the basis-forms and we now eliminate $A0, A1$, by the use of OPL and retain the remaining equations.

```
*opl(aa(1,2),a1)$ a1:=a1;
A1 :=  - V(1,X(4))*X(4) + V(3,X(4))
*opl(aa(1,5),a0)$
*ver 1:=aa(1,0)$ ver 2:=aa(1,1)$ ver 3:=aa(1,3)$
*ver 4:=aa(1,4)$
*for i:=1:4 do write ver i:=ver i;
VER(1) := V(2,X(4))
VER(2) :=  - X(4)*V(2,X(4))
VER(3) :=  - V(2,X(3))*X(4) - V(1,X(4))*X(4) + V(3,X(4))
           - V(2,X(1))
VER(4) :=  - V(1,X(1))*X(4) + V(3,X(1)) - V(1,X(3))*X(4)^2 +
          V(3,X(3))*X(4) + 2*V(1,X(4))*X(3)*X(4)^2
          - 2*V(3,X(4))*X(3)*X(4) - V(4,VAR)
```

□

The process of computing the whole system of partial differential equations for the functions V^i $(i = 1, \ldots, n)$ runs along similar lines, as in Example 3.1, and is done automatically by the procedure INFSYM(VOR, $N1$, N) in the appendix, where VOR is the name of the differential forms generating I (3.1), $N1$ is the number of differential forms, and N is the dimension of R^n.

Once the system of partial differential equations has been derived, we have to solve it for the functions $V(I, \text{VAR})$. In general, it is not possible to solve them automatically; but it turns out that in the most interesting cases, e.g., nonlinear evolution equations such as Korteweg–de Vries and Burgers', and field equations such as Maxwell and Dirac equations, solutions are of polynomial type and, therefore, we are able to compute them semi-automatically [9, 12].

The process of automization of solving these overdetermined systems is still in progress, and we shall discuss two frequently occurring cases.

(A) Usually there are equations in this system of type

$$\text{VER}(J) := V(I, X(K))$$

for some I, K; from which we derive

$V(I, \text{VAR})$ is independent of $X(K)$.

Procedure ONE( ) searches for such equations and assigns zero to the function $V(I, X(K))$.

(B) In most cases, the system of partial differential equations is linear in $V(I, X(J))$, $V(K, \text{VAR})$ (we shall denote these by $V(I, *)$) with coefficients which are polynomials in some of the variables $X(1), \ldots, X(N)$.

If an equation contains only terms $V(I, *)$ independent of $X(I_1), \ldots, X(I_r)$, (results, which can be obtained by procedure ONE( )) and if the coefficients are polynomials in $X(I_1), \ldots, X(I_r)$, we consider this equation as a polynomial in $X(I_1), \ldots, X(I_r)$ and split up this equation into a set of smaller equations, due to the fact that the coefficients of this polynomial have to be zero.

This analysis is carried out by SPLIT1(K), which searches for a polynomial structure in VER(K), or SPLITS( ), which deals with the whole system.

SCHEM( ) is a procedure that creates lists !@$A(L)$ of numbers representing partial derivatives which are zero. !@$A(2) = (1, 4)$ means $V(2, X(1)) = 0$; $V(2, X(4)) = 0$
FIG( ) prints a diagram of partial derivatives being zero.

EXAMPLE 3.2. First we load some necessary programs.

```
*fload opcoef;
*in tools.red,opl.red,split.red$

TEL1 := 0

*in mlie$
EXECUTE : ARRAY !@A(D!@DIF)$

EXECUTE : ARRAY A!@FIG(D!@DIF)$
```

We read a file which contains a system of five equations (TOTAAL = 4), while the number of variables is 7 (D!@DIF = 7); so $V(I, \text{VAR})$ is dependent on $X(1), \ldots, X(7)$.

```
*in test$

VER(0) := V(1,X(2)) + X(3)*V(1,X(4))

VER(1) := V(1,X(4))

VER(2) := V(1,X(2))*X(1) + 3*X(4)*V(1,X(3))

                  2
VER(3) := X(2) *V(1,X(7)) + X(3)*V(1,X(6))

VER(4) := X(4)*V(2,X(5)) + V(1,X(6))

TOTAAL := 4

D@DIF := 7

*array !@a d!@dif,a!@fig d!@dif;
```

These are the initiations required by the system:

```
*one()$

V(1,X(4))

0

V(1,X(2))

0

V(1,X(3))

0

                  2
VER(3) := X(2) *V(1,X(7)) + X(3)*V(1,X(6))


VER(4) := X(4)*V(2,X(5)) + V(1,X(6))
```

ONE sets $V(1, X(4))$, $V(1, X(2))$, $V(1, X(3))$ equal to zero and prints the remaining set of nontrivial equations.

```
*fig()$
VIJ   1   2   3   4   5   6   7
  1   *   *   *   *   *   *   *
  2   0   *   *   *   *   *   *
  3   0   *   *   *   *   *   *
  4   0   *   *   *   *   *   *
  5   *   *   *   *   *   *   *
  6   *   *   *   *   *   *   *
  7   *   *   *   *   *   *   *
```

From FIG we see that $V(1, X(2)) = V(1, X(3)) = V(1, X(4)) = 0$.

```
*zerodelete('ver)$
```

This is a procedure that deletes trivial equations, i.e.,

$$\mathrm{VER}(I) := 0$$

from the system.

```
*splits()$

0-TH EQUATION : YES!!(2 3 4)
1-TH EQUATION : NO!!
TOTAAL:=2
```

From the message of the procedure SPLITS we see that equation 0 (i.e., VER(3) above, because ZERODELETE has renumbered the equations) is a polynomial in $X(2)$, $X(3)$ and $X(4)$. The coefficients of this polynomial, which have to be zero, generate a new and larger set of equations.

```
*for i:=0:totaal do write ver i:=ver i;

VER(0) := X(4)*V(2,X(5)) + V(1,X(6))

VER(1) := V(1,X(6))

VER(2) := V(1,X(7))


VER(1),VER(2) are just the coefficients of VER(3) above.

*one()$

V(1,X(6))

0

V(1,X(7))

0

V(2,X(5))

0
```

Now all equations are trivial, so there is no remaining set to be printed.

```
*fig()$
VIJ  1  2  3  4  5  6  7
  1  *  *  *  *  *  *  *
  2  0  *  *  *  *  *  *
  3  0  *  *  *  *  *  *
  4  0  *  *  *  *  *  *
  5  *  0  *  *  *  *  *
  6  0  *  *  *  *  *  *
  7  0  *  *  *  *  *  *
```

□

We applied this to the computation of the infinitesimal symmetries of the Lin–Reisner–Tsien equation (unsteady transonic gas motion)

$$-u_x u_{xx} - 2u_{xt} + u_{yy} = 0.$$

The ideal I in $R^{12} = \{(x(1), \ldots, x(12))\} = \{(x, t, y, u, u_x, u_t, u_y, u_{xx}, u_{xt}, u_{xy}, u_{tt}, u_{ty})\}$ is generated by the 1-forms

```
ALFA(1):=WEDGE(4)-x(5) *WEDGE(1)-x(6) *WEDGE(2)-x(7)  *WEDGE(3)
ALFA(2):=WEDGE(5)-x(8) *WEDGE(1)-x(9) *WEDGE(2)-x(10) *WEDGE(3)
ALFA(3):=WEDGE(6)-x(9) *WEDGE(1)-x(11)*WEDGE(2)-x(12) *WEDGE(3)
ALFA(4):=WEDGE(7)-x(10)*WEDGE(1)-x(12)*WEDGE(2)-PSI(1)*WEDGE(3)
PSI(1):= 2*X(9)+ X(5)*X(8)
```

and their exterior derivatives.

Due to (3.4) we shall omit EXDF(ALFA(I))($I := 1:4$). We calculated the following infinitesimal generators of the group of symmetries

$$
\begin{aligned}
V^x &= V(1, \mathrm{VAR}) = xF'(t) + y^2F''(t) + yG'(t) + H(t) + 2Ax. \\
V^t &= V(2, \mathrm{VAR}) = 3F(t) \\
V^y &= V(3, \mathrm{VAR}) = 2yF'(t) + G(t) + Ay \\
V^u &= V(4, \mathrm{VAR}) = -uF'(t) + x^2F''(t) + 2xy^2F'''(t) + \\
&\qquad + 1/3y^4F''''(t) + 2xyG''(t) + 2/3\,y^3G'''(t) + \\
&\qquad + 2xH'(t) + 2y^2H''(t) + y\sigma(t) + \tau(t) + 4Au
\end{aligned}
\tag{3.6}
$$

where F, G, H, σ, τ are arbitrary functions of t, A is constant, and denotes differentiation. For simplicity, we omit the functions $V^{u_x}, \ldots$, which can be derived from (3.6) by prolongation. This result is in agreement with that of Anderson and Ibragimov [1, p. 73], apart from the generator arising from A.

4. Conclusions

In principal, anything which can be achieved by symbolic calculations can be achieved by hand and vice versa. By using the computer, however much valuable time can often be saved.

We invite the reader to do symbolic calculations in REDUCE himself and to experience with the material presented in this paper. We are aware that various improvements can be made, and so further suggestions will be most welcome.

If a reader is interested in the functions in the Appendix, it may be of help to get a copy on magnetic tape. This can be arranged by sending a blank tape to one of the authors.

Appendix

```
%OPCOEF.RED 10-FEB-81$
SCALAR !@RESTOPCOEFF$
LISP OPERATOR OPCOEFF$

LISP PROCEDURE GETFIRSTOP(FORM,L,OP);
%C GETFIRSTOP.i;
IF ATOM FORM THEN L
%C ii;
```

```
ELSE BEGIN SCALAR X!@GETFIRSTOP;
%C II.1;
    WHILE (NOT ATOM FORM) AND NULL L  DO <<
%C ii.2;
    L:=GETFIRSTOP(LC FORM,
        IF (NOT ATOM (X!@GETFIRSTOP:=MVAR FORM))
            AND CAR X!@GETFIRSTOP=OP
            THEN X!@GETFIRSTOP ELSE L,OP);
%C ii.3;
    FORM:=RED FORM>>;
%C ii.4;
    RETURN L
END GETFIRSTOP;

LISP PROCEDURE OPCOEFF(EX,OP,AR);
%C OPCOEFF.i;
IF NOT(IDP OP AND IDP AR) THEN
REDERR("OPCOEFF: SECOND OR THIRD PARAMETER  NO ID")
%C ii;
ELSE IF EX = 0 THEN << !@RESTOPCOEFF:=0;
    M!@A2D(AR,'(NIL NIL));-1 >>
%C iii;
ELSE BEGIN SCALAR EL,Y,XX,DENEX,LKERN,LCOEFF;
%C iii.1;
    EX:=SIMP!* EX;
    Y:=NUMR EX;DENEX:=DENR EX;
%R    IF GETFIRSTOP(DENEX,NIL,OP) THEN REDERR
%R ("OPCOEFF: NOT A LINEAR OP-ELEMENT (WRONG DENOMINATOR)");
    XX:=KORD!*;
%C iii.2;
    WHILE EL:=GETFIRSTOP(Y,NIL,OP) DO BEGIN
%C iii.3;
        KORD!*:=(LIST(EL:=!*A2K EL));
        Y:=FORMOP Y;
%C iii.4;
        IF  NOT LDEG Y=1 THEN
        BEGIN KORD!*:=XX;REDERR(
    "OPCOEFF: NOT A LINEAR OP-ELEMENT (POWER UNEQUAL 1)")
        END;
%C iii.5;
%R    IF GETFIRSTOP(LC Y,NIL,OP) THEN BEGIN KORD!*:=XX;REDERR
%R("OPCOEFF: NOT A LINEAR OP-ELEMENT (WRONG COEFFICIENT)")
%R     END;
    LKERN:=EL . LKERN;
    LCOEFF:=MK!*SQ1 CANCEL(LC Y ./ DENEX) . LCOEFF;
%C iii.6;
    Y:=RED Y; END LOOP L;
%C iii.7;
    !@RESTOPCOEFF:=MK!*SQ1 CANCEL( Y  ./ DENEX);
    M!@A2D(AR,LIST(LKERN,LCOEFF));
    KORD!*:=XX;
%C iii.8;
    RETURN IF LKERN THEN LENGTH(LKERN)-1 ELSE -1
END OPCOEFF;

LISP PROCEDURE M!@A2D(NAME,L);
```

```
%C M!@A2D.i;
BEGIN IF NULL GET(NAME,'!@OPELEMENT) THEN
%C ii;
    <<FLAG(LIST(NAME),'OPFN);
        PUTD(NAME,'EXPR,LIST('LAMBDA,'(I0 I1),
        LIST('NTH,
        LIST('NTH,LIST('GET, MKQUOTE NAME,
        MKQUOTE '!@OPELEMENT),
        '(ADD1 I0)),'(ADD1 I1))) )>>;
%C iii;
    PUT(NAME,'!@OPELEMENT,L);
END M!@A2D;
%DIFFOR.RED 6-APR-1981;
SCALAR !@RESTOPCOEFF,D!@DIF,!@UIT;
OPERATOR VNAT,UIT;
LISP OPERATOR EXDF,MULFORM,NORMDIF$

%INITIALIZATIONS;
WRITE(
"GIVE VALUES TO D!@DIF AND VNAT(I) I=3,...,D!@DIF
(VNAT 1=X, VNAT 2= T)")$
FACTOR UIT$OFF ALLFAC$
VNAT(1)d:=X;VNAT(2):=T;
LISP !@UIT:=WEDGE$

LISP PROCEDURE !@REPEATSONCE X;
%C !@REPEATSONCE.i;
IF NULL X THEN NIL
%C ii;
ELSE IF CAR X MEMBER CDR X THEN T
%C iii;
ELSE !@REPEATSONCE CDR X;

LISP PROCEDURE MERGESTRICT(LA,LB);
%C MERGESTRICT.i;
BEGIN SCALAR LALB,RES;
%C ii;
RETURN IF !@REPEATSONCE (LALB:=APPEND(LA,LB)) THEN '(0)
%C iii;
ELSE IF PERMP(LALB,RES:=REVERSIP ORDN LALB)
THEN  1 . RES
%C iv;
ELSE (-1) . RES
END MERGESTRICT;

LISP PROCEDURE MULFORM(A,B);
%C MULFORM.i;
BEGIN INTEGER F,D1,D2;
    SCALAR H3,AMUL1I,RES,AREST,BREST,ELA,ELB,L1;
%C ii;
    D1:=OPCOEFF(A,!@UIT,'!@AMUL);AREST:=!@RESTOPCOEFF;
    D2:=OPCOEFF(B,!@UIT,'!@BMUL);BREST:=!@RESTOPCOEFF;
%C iii;
%C iii.1;
    RES:=AEVAL LIST('PLUS, LIST('TIMES,AREST,B),
        LIST('TIMES,BREST,AEVAL LIST('DIFFERENCE,A,AREST)));
```

```
%C iii.2;
%C 2.a;
    FOR I:=0:D1 DO
%C 2.b;
    BEGIN AMUL1I:=!@AMUL(1,I);ELA:=CDR !@AMUL(0,I);
%C 2.c;
    FOR J:=0:D2 DO BEGIN
%C 2.d;
            L1:=MERGESTRICT(ELA,CDR !@BMUL(0,J));
%C 2.e;
                IF (F:=CAR L1) NEQ 0 THEN
            RES:=AEVAL LIST(IF F=1 THEN 'PLUS ELSE
            'DIFFERENCE,RES,LIST('TIMES,AMUL1I,
                !@BMUL(1,J),!@UIT . CDR L1)) ;
    END J;END I;
%C iv;
    RETURN RES
END MULFORM;

LISP PROCEDURE EXDF(EXPRESSIE);
%C EXDF.i;
BEGIN SCALAR H1,H2,H3,L1,DRES,COI;
    INTEGER D1,F;
%C ii;
    D1:=OPCOEFF(EXPRESSIE,!@UIT,'A!@EXDF);
%C iii;
    FOR I:=0:D1 DO
%C iv;
%C iv.1;
    BEGIN COI:=A!@EXDF(1,I);H1:=CDR A!@EXDF(0,I);
%C iv.2;
        FOR II:=1:D!@DIF DO <<
%C iv.3;
            IF NOT(II MEMBER H1) THEN
%C iv.4;
            BEGIN H2:=AEVAL LIST('DF,COI,LIST('VNAT,II));
%C iv.5;
                IF H2 NEQ 0 THEN
%C iv.6;
%C 6.I;
                BEGIN L1:=MERGESTRICT(LIST II,H1);
%C 6.II;
                    DRES:=AEVAL
                        LIST(IF (CAR L1)=1 THEN 'PLUS ELSE
                        'DIFFERENCE,DRES,
                        LIST('TIMES,H2,!@UIT . CDR L1 ))
                END
            END >>
    END;
%C v;
    RETURN IF !@RESTOPCOEFF NEQ 0 THEN
%C vi;
        AEVAL LIST('PLUS,DRES,
        FOR II:=1:D!@DIF SUM AEVAL
            << H2:=AEVAL LIST('DF,!@RESTOPCOEFF,
                LIST('VNAT,II));IF 0=H2 THEN 0 ELSE
```

```
                        LIST('TIMES,H2,LIST(!@UIT,II))>>)
%C vii;
        ELSE DRES
END EXDF;

LISP OPERATOR NORMDIF;
LISP PROCEDURE NORMDIF(X);
%C NORMDIF.i;
BEGIN INTEGER D;SCALAR L;
%C ii;
    D:=OPCOEFF(X,!@UIT,'OP!@NORMDIF);
%C iii;
    RETURN AEVAL LIST('PLUS,!@RESTOPCOEFF,
    FOR I:=0:D SUM
        IF CAR(L:=MERGESTRICT(CDR OP!@NORMDIF(0,I),NIL)) = 0
        THEN 0 ELSE LIST('TIMES,CAR L,OP!@NORMDIF(1,I),
        !@UIT . CDR L))
END NORMDIF;
%COM.RED 7-JUL-81;
ARRAY A!@COM(1,1);
SCALAR D!@DIF,D!@DIFP1,D!@DIFP2,!@VECVAR;
OPERATOR VNAT,D!@COM,VF;
LISP OPERATOR NIEUWCOEFF,FILVNAT$

WRITE("EXECUTE: INICOM(DIMENSION OF PROBLEM,",
"NAME OF VARIABLES) !!")$
LISP !@VECVAR:='D$
OFF ALLFAC;FACTOR D;

ALGEBRAIC PROCEDURE CH(CHANGE,N);
IF N<1 OR N>D!@DIF THEN WRITE(
"***** CH: SECOND PARAMETER OUT OF RANGE: 1,...,",D!@DIF)
ELSE BEGIN VF(N):=VF(N)+CHANGE;
    D!@COM(N):=NIEUWCOEFF(VF N,!@VECVAR,A!@COM,N);
    IF A!@COM(N,0) NEQ 0 THEN WRITE(
"*** THE VECTOR FIELD CONTAINS A CONSTANT TERM,
WHICH  SHOULD BE ELIMINATED
NO FURTHER ACTION TAKEN");
RETURN CHEND
END CH$

LISP PROCEDURE FILVNAT(N,NAM);
FOR I:=1:N DO
SETK(LIST('VNAT,I), INTERN COMPRESS APPEND(NAM, EXPLODE I))$

PROCEDURE INICOM(N,NAME);BEGIN CLEAR A!@COM;
    WRITE("D!@DIF = ",N);
    WRITE(
"THE SPECIAL VECTOR FIELDS VF(I) ARE INITIALIZED TO 0
1 <= I <= ",N,"
AND SHOULD BE CHANGED BY: CH(CHANGE,N),
THAT MEANS, CH(CHANGE,N) HAS AS EFFECT:VF(N):= VF(N)+CHANGE;
PLUS OTHER SIDE EFFECTS");
    WRITE(
"TO USE THE SPECIAL VECTOR FIELDS, USE ONLY THE INDEX
OF THEM AS PARAMETER TO COM");
    FOR I:=1:N DO VF(I):=D!@COM(I):=0;
```

```
    D!@DIF:=N;D!@DIFP1:=N+1;D!@DIFP2:=N+2;
    FILVNAT(N,NAME);ARRAY A!@COM(N+2,N);
    WRITE(
"THE LOCAL COORDINATES ARE ",VNAT(1),",...,",VNAT(N));
    RETURN INICOMEND
END INICOM$

LISP PROCEDURE NIEUWCOEFF(A,XX,NAM,I)$
COEFF(A,XX,LIST(NAM,I,'TIMES))$
ALGEBRAIC PROCEDURE COM(I,J);
%C COM.i;
BEGIN INTEGER DI,DJ;
%C ii;
    IF NOT(NUMBERP I AND I 0 AND I =D!@DIF)
%C iii;
         THEN << DI:=NIEUWCOEFF(I,!@VECVAR,A!@COM,D!@DIFP1);
                 I:=D!@DIFP1 >>
%C iv;
    ELSE DI:=D!@COM(I);
%C v;
    IF NOT(NUMBERP J AND J>0 AND J<=D!@DIF)
%C vi;
         THEN << DJ:=NIEUWCOEFF(J,!@VECVAR,A!@COM,D!@DIFP2);
                 J:=D!@DIFP2 >>
%C vii;
    ELSE DJ:=D!@COM(J);
%C viii;
    RETURN FOR II:=1:DI SUM IF A!@COM(I,II) NEQ 0 THEN
    FOR JJ:=1:DJ SUM IF A!@COM(J,JJ)NEQ 0 THEN
    (A!@COM(I,II)*!@VECVAR^JJ*DF(A!@COM(J,JJ),VNAT(II))-
    A!@COM(J,JJ)*!@VECVAR^II*DF(A!@COM(I,II),VNAT(JJ)))
    ELSE 0 ELSE 0
END COM$
%IP.RED 25-MAR-81;
ARRAY A!@IP(0);
SCALAR !@UIT,!@VECVAR$
LISP OPERATOR REDERR,B!@IP$
OPERATOR UIT$
LISP !@VECVAR:='D$ LISP !@UIT:='WEDGE$

LISP PROCEDURE B!@IP(IND,BUIT);
%C B!@IP.I;
BEGIN SCALAR H,FAC;
%C ii;
    RETURN IF IND MEMBER (H:=CDR BUIT) THEN
%C iii;
    IF LENGTH H=1 THEN 1
%C iv;
    ELSE<<FAC:=1;
        WHILE (IND NEQ CAR H) DO<<FAC:=-FAC;H:=CDR H>>;
%C v;
        AEVAL LIST('TIMES,FAC,DELETE(IND,BUIT))>>
%C vi;
    ELSE 0
END B!@IP;
```

```
PROCEDURE IP(X,ALFA);
%C IP.i;
BEGIN SCALAR CO,IND;INTEGER DX,DA;
%C ii;
    DX:=COEFF(X,!@VECVAR,A!@IP);
%C iii;
    IF A!@IP(0) NEQ 0 THEN
    REDERR("IP: NO GOOD VECTORFIELD GIVEN");
%C iv;
    DA:=OPCOEFF(
%R NORMDIF
        ALFA,!@UIT,UIT!@IP);
%C v;
    RETURN FOR I:=1:DX SUM
%C vi;
%C vi.1;
    IF (CO:=A!@IP(I)) = 0 THEN 0
%C vi.2;
        ELSE << FOR J:=0:DA SUM
%C vi.3;
        ( B!@IP(I,UIT!@IP(0,J))*CO*UIT!@IP(1,J) )>>
END IP;

CLEAR A!@IP;
WRITE("EXECUTE: ARRAY A!@IP(MAXIMAL DIMENSION);");
%OPL.RED 15-APR-81$
OFF ECHO$
ARRAY A!@OPL(1)$
LISP OPERATOR OPL,D!@MAX$

LISP PROCEDURE OPL(WAT,WAARNA)$
IF D!@MAX(WAT,WAARNA)=1 THEN
BEGIN COEFF(WAT,WAARNA,'A!@OPL)$
    RETURN SETK(WAARNA,AEVAL LIST('MINUS
    ,LIST('QUOTIENT,GETEL LIST('A!@OPL,0),
    GETEL LIST('A!@OPL,1))))END
ELSE WRITE(WAARNA," NOT A KERNEL
OR EXPRESSION NOT LINEAR WITH RESPECT TO ",WAARNA);

LISP PROCEDURE D!@MAX(EPR,VARI)$
MAXDEGREE(NUMR SIMP EPR,VARI)$

LISP PROCEDURE MAXDEGREE(FORM,VARI)$
IF DOMAINP FORM THEN 0
ELSE IF MVAR FORM = VARI THEN LDEG FORM
ELSE MAX2(MAXDEGREE(LC FORM,VARI),MAXDEGREE(RED FORM,VARI))$
ON ECHO$
SCALAR TOTAAL,D!@DIF$
LISP OPERATOR CLEARKVALUE,LE;
LISP PROCEDURE CLEARKVALUE(OP);
<<REMPROP(OP,'KVALUE);0>>;

OPERATOR V,CO,B,LA,VER,O!@INF;
PROCEDURE INFSYM(VOR,N1,N);
BEGIN INTEGER M,M1, JJ,I,J,K;
    ARRAY RA N1;
    FOR I:=1:N1 DO B I:=NORMDIF(VOR I);
```

```
    M:=1;M1:=0;
    VEC:=(FOR I:=1:N  SUM V(I,VAR)*D^I);
    FOR ALL X,II LET DF(V(II,VAR),X)=V(II,X);
    FOR JJ:=1:N1 DO
    BEGIN AA:=!@BEVATOP(B(JJ),UIT);
        RA(JJ):=LE AA;
    END;
    FOR I:=1:N1 DO
    BEGIN
        LA I:=LIEDF(VEC,B I)-(FOR J:=1:N1 SUM
          MULFORM(GEFORM(J,RA I-RA J,N),B J));
        AANT:=OPCOEFF(LA I,UIT,C);
        FOR K:=0:AANT DO
        BEGIN L1!@INF:=OPCOEFF(C(1,K),CO,O!@INF);
              S!@INF:=0;
              WHILE S!@INF   L1!@INF+1 DO
              IF (O!@INF(1,S!@INF)^2)=1 THEN
              BEGIN OPL(C(1,K),O!@INF(0,S!@INF));
                    S!@INF:=L1!@INF+1;
              END ELSE S!@INF:=S!@INF+1;

        END;
        FOR K:=0:AANT DO
        BEGIN L:=!@BEVATOP(C(1,K),CO);
            IF L NEQ 0 THEN BEGIN
                        WRITE "YOU ARE CREATING DENOMINATORS!!";
                              OPL(C(1,K),L)
                         END
            ELSE IF C(1,K) NEQ 0 THEN BEGIN VER M:=C(1,K);
                                            VER M:=VER M*DEN VER M;
                                            M:=M+1;
                                       END;
        END;
        TOTAAL:=M-1;%ONETERMSOL(VER,TOTAAL);
        WRITE "THERE EXIST ",M-1-M1," EQUATIONS FOR THE ",I,"-TH
        M1:=M-1;CLEARKVALUE(CO);
    END;
    VER 0:=0;
    FOR JJJ:=0:M-1 DO WRITE VER JJJ:=VER JJJ;
    RETURN M-1;
END;
    PROCEDURE GEFORM(NR,J,N);
    IF J=0 THEN  CO(NR,J) ELSE
    IF J=1 THEN  (FOR I:=1:N SUM CO(NR,J,I)*UIT I) ELSE
    IF J=2 THEN   (FOR I:=1:N SUM
    (FOR K:=I+1:N SUM CO(NR,J,I,K)*UIT(I,K))) ELSE
    IF J=3 THEN  (FOR I:=1:N SUM
    (FOR K:=I+1:N SUM
    (FOR L:=K+1:N SUM CO(NR,J,I,K,L)*UIT(I,K,L)))) ELSE 0

    PROCEDURE PSP(N,M,K)$
    FOR I:=M:K DO IF N I NEQ 0 THEN WRITE N I:=N I$

    PROCEDURE ONETERMSOL(NAME,AANT)$
    BEGIN M:=0$
          FOR I:=0:AANT DO IF VER I NEQ 0 THEN
                           IF OPCOEFF(VER I,V,!@CC)=0
                             AND !@RESTOPCOEFF=0
                             THEN BEGIN WRITE !@CC(0,0)$
                                        M:=M+1$
                                        !@B:=OPL(VER I,!@CC(0,0)
                                        WRITE !@B$
                                  END$
```

```
      IF M 0 THEN ONETERMSOL(NAME,AANT)$
      END$

PROCEDURE ONE()$
BEGIN ONETERMSOL(VER,TOTAAL)$
      PSP(VER,0,TOTAAL)$
END$
PROCEDURE PS()$
PSP(VER,0,TOTAAL)$
PROCEDURE LPO(AANT1,AANT2)$
BEGIN OPL(VER AANT1,V(AANT2,VAR))$
      VER AANT1:=0$
END$

ARRAY !@A(0)$
LISP OPERATOR SCHEM,LX,LYSTX,TEST,TEST1$
LISP PROCEDURE LYSTX(L)$
BEGIN LX:=NIL$
       WHILE L DO BEGIN LX:=LIST('X,CAR L).LX$
                        L:=CDR L
                  END$
      RETURN LX$
END LYSTX$
LISP PROCEDURE SCHEM()$
BEGIN FOR J:=1:D!@DIF DO !@A(J):=NIL$
      FOR I:=1:D!@DIF DO
      FOR K:=1:D!@DIF DO
      IF REVAL LIST('V,I,LIST('X,K)) EQ 0 THEN !@A(K):=I.!@A(K)$
END$

ARRAY A!@FIG(0)$
LISP OPERATOR FIG$
LISP PROCEDURE FIG()$
BEGIN SCALAR H$
      H:=NIL$
      SCHEM()$
      FOR I:=D!@DIF STEP -1 UNTIL 1 DO
      IF I 10 THEN H:='! .'! .I.H
              ELSE H:='! .APPEND(EXPLODEC I,H)$
      H:='V.'I.'J.H$
      SETEL('(A!@FIG 0),H)$
      FOR J:=1:D!@DIF DO
      BEGIN H:=NIL$
            FOR I:=D!@DIF STEP -1 UNTIL 1 DO
            IF I MEMBER !@A(J) THEN H:='! .'! .'O.H
                               ELSE H:='! .'! .'!*.H$
            IF J 10 THEN H:='! .'! .J.H
                    ELSE H:='! .APPEND(EXPLODEC J,H)$
            SETEL(LIST('A!@FIG,J),H)$
      END$
      FOR J:=0:D!@DIF DO
      <<MAPCAR(GETEL LIST('A!@FIG,J),FUNCTION PRINC)$
         TERPRI()>>
END FIG $
WRITE "EXECUTE : ARRAY !@A(D!@DIF)$"$
WRITE "EXECUTE : ARRAY A!@FIG(D!@DIF)$"$
ARRAY !@B 0$

LISP OPERATOR SPLITS,SPLIT1;

LISP PROCEDURE SPLITS()$
BEGIN SCALAR RY,RYTJE,RES,TEL$
      RY:=NIL$TEL:=0$
```

```
     FOR II:=1:D!@DIF DO RY:=II.RY$
     SCHEM()$
     FOR JJ:=0:TOTAAL DO
     BEGIN RYTJE:=NIL$
           LB:=OPCOEFF(REVAL LIST('VER,JJ),'V,'!@CO)$
           FOR EACH EL IN RY DO
           BEGIN RES:=NIL$
                 FOR KK:=0:LB DO
                 IF CADR !@CO(0,KK) MEMBER !@A(EL)
                 THEN NIL
                 ELSE BEGIN KK:=LB+1$
                            RES:=T$
                      END$
                 IF RES THEN NIL
                        ELSE RYTJE:=EL.RYTJE$
           END$
           IF RYTJE=NIL THEN
             BEGIN SETK(LIST('VER,TEL),REVAL LIST('VER,JJ))$
                                       TEL:=TEL+1$
                                       WRITE JJ,"-TH EQUATION : NO!!"$
                                 END
                           ELSE BEGIN
                           MLIETAB(REVAL LIST('VER,JJ),LYSTX RYTJE)$
                                  SETK(LIST('VER,JJ),0)$
                                  WRITE JJ,"-TH EQUATION :",RYTJE$
                                 END$
      END$
      FOR LL:=TEL:TEL+TEL1-1 DO
      SETK(LIST('VER,LL),REVAL LIST('PETER,LL-TEL+1))$
      TOTAAL:=TEL+TEL1-1$
      CLEARKVALUE('PETER)$
      TEL1:=0$
      WRITE "TOTAAL:=",TOTAAL$
END SPLITS $

LISP OPERATOR ZERODELETE;
SCALAR L!@ZERODELETE;
LISP PROCEDURE ZERODELETE NAME;BEGIN SCALAR L1,L2,TEL;
    TEL:=0;L!@ZERODELETE:=NIL;
    FOR EACH XX IN GET(NAME,'KVALUE) DO
    IF REVAL(L2:=CADR XX)=0 THEN L!@ZERODELETE:=(CAR XX .
    '(0)) . L!@ZERODELETE
    ELSE L1:=(LIST(NAME,TEL:=TEL+1). LIST L2 ).L1;
    IF L1 THEN PUT(NAME,'KVALUE,REVERSIP L1) ELSE
    RETURN TEL
END;

LISP PROCEDURE SPLIT1(JJ);
BEGIN SCALAR RY,RYTJE,RES;
      RY:=RYTJE:=NIL;
      FOR II:=1:D!@DIF DO RY:=II.RY;
      SCHEM();
      LB:=OPCOEFF(REVAL LIST('VER,JJ),'V,'!@CO);
      FOR EACH EL IN RY DO
      BEGIN RES:=NIL;
            FOR KK:=0:LB DO
            IF CADR !@CO(0,KK) MEMBER !@A(EL)
            THEN NIL
            ELSE BEGIN KK:=LB+1;
                       RES:=T;
                 END;
            IF RES THEN NIL
                   ELSE RYTJE:=EL.RYTJE;
      END;
```

```
IF RYTJE=NIL THEN
BEGIN WRITE JJ,"-TH EQUATION : NO!!";
      RETURN 0
END
      ELSE BEGIN
                  MLIETAB(REVAL LIST('VER,JJ),LYSTX RYTJE);
                  SETK(LIST('VER,JJ),0);
                  WRITE JJ,"-TH EQUATION :",RYTJE;
           END;
      FOR LL:=TOTAAL+1:TOTAAL+TEL1 DO
      SETK(LIST('VER,LL),REVAL LIST('PETER,LL-TOTAAL);
      TOTAAL:=TOTAAL+TEL1;
      CLEARKVALUE('PETER);
      TEL1:=0;
      WRITE "TOTAAL:= ",TOTAAL;
END SPLIT1;
```

References

1. Anderson, R. L. and Ibragimov, N. H.: *Lie-Bäcklund Transformations in Applications*, SIAM Studies in Applied Mathematics, Philadelphia, 1979.
2. Cartan, E.: *Les systèmes différentiels et leurs applications* géométriques, Hermann, Paris, 1945.
3. Edelen, D. G. B.: *Isovector Methods for Equations of Balance*, Sijthoff and Noordhoff, Alphen a/d Rijn, The Netherlands, 1981.
4. Wahlquist, H. D. and Estabrook, F. B.: *J. Math Phys.* **16** (1975), 1–7.
5. Estabrook, F. B. and Wahlquist, H. D.: *J. Math Phys.* **17** (1976), 1293–1297.
6. Frick, I.: *SHEEP's Users' Manual*, University of Stockholm, 1977.
7. Estabrook, F. B.: 'Differential geometry as a tool for applied mathematicians, in R. Martini (ed.), *Geometric Approaches to Differential Equations* (Lecture Notes in Mathematics, Vol. 810), Springer-Verlag, Heidelberg, New York, 1979.
8. Gragert, P. K. H.: 'Symbolic computations in prolongation theory', PhD thesis, Twente University of Technology, 1981.
9. Gragert, P. K. H. and Kersten, P. H. M.: 'Implementation of differential geometric objects and functions with an application to extended Maxwell equations', in J. Calmet (ed.), *Proceedings Eurocam 1982* (Lecture Notes in Computer Science, Vol. 144), Springer-Verlag, Heidelberg, New York, 1982, pp. 181–187.
10. Harrison, B. K. and Estabrook, F. B.: *J. Math. Phys.* **12** (1971), 653–666.
11. Hearn, A. C.: *Reduce 2 Users' Manual*, 2nd edition, University of Utah, 1973.
12. Kersten, P. H. M.: 'Infinitesimal symmetries and conserved currents for nonlinear Dirac equations', to appear in *J. Math. Phys.*
13. Lewis, V. E. (ed.): *MACSYMA Users' Conference Proceedings, 1979*, Washington D.C., MIT Laboratory of Computer Science, Cambridge, Mass., 1979.
14. McCarthy, J. *et al.*: *LIPS 1.5 Programmers' Manual*, MIT Press, Cambridge, Mass., 1962.
15. Molino, P.: 'Exterior differential systems and partial differential equations', Notes of invited lectures at the University of Amsterdam, 1982.
16. *REDUCE Newsletters*, 1 (1978), 2 (1978), 3 (1978), 4 (1978), 5 (1979), 6 (1979), 7 (1979), Symbolic Computation Group, University of Utah.
17. Risch, R. H.: *Trans. Am. Math. Soc.* **139** (1969), 167–189.
18. Wahlquist, H. D.: 'Exterior calculus on an Apple: the program "CARTAN", preprint, 1981.