INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONE: 2240-1
CABLE: CENTRATOM - TELEX 460392 - I

SECOND SCHOOL ON ADVANCED TECHNIQUES
IN COMPUTATIONAL PHYSICS
(18 January - 12 February 1988)

SMR.282/ 17

COMPUTER ALGEBRA AND DIFFERENTIAL EQUATIONS

Eberhard Schrufer
GMD, St. Augustin, West Germany

Computer Algebra and Differential Equations

Eberhard Schrüfer
GMD, Institute F1-P, D5205 St. Augustin 1, W. Germany

January 28, 1988

1 Introduction

Investigations of differential equations are an active area of research in Computer Algebra. Besides the ultimate goal of finding exact solutions by some method, the more modest aim to assist a scientist or engineer in formulating his concrete problem for a numerical study is of great importance.
These lectures are therefore divided into two parts: the first is concerned with the application of Computer Algebra while preparing a problem for a numerical study. The second part is devoted to the study of differential equations themselves, which leads to insight into their structure and, in some cases, also to an explicit solution.
We can identify several crucial steps where Computer Algebra can be of help in the course of solving a problem.
First, there is a theory, which very often has its differential equations written in a concise 'symbolic' form. Examples are

$$div(\rho * \vec{v}) = 0$$
$$F^{\mu\nu}{}_{|\nu} = 4 * \pi / c * j^{\mu}$$
$$-1/2 * \#(e^a \wedge e^b \wedge e^c) \wedge R_{bc} = 8 * \pi * g * \#T^a$$

The structure of these equations can be studied on its own, and we will say more about that in the second part. In an application, however, these equations must be expressed with respect to a certain coordinate system. This can be quite a formidable task, especially if boundary conditions dictate the use of some untabulated, nonorthogonal coordinates. The process done by hand is very lengthy and thereby error-prone. However, with an appropriate Computer Algebra at hand it is no more than an easy exercise, as we will see.
Having performed this step, a decision has to be made on a method to proceed further towards a solution. If there is hope for a closed form solution we could try a method indicated in part 2 of these notes. Human intuition can lead here to quite wrong judgments. A seemingly simple problem may have no closed solution whatsoever, and a very complex problem in appearance might have a simple analytical solution. Unfortunately, the methods for finding closed form solutions are very costly, and the decision often must be

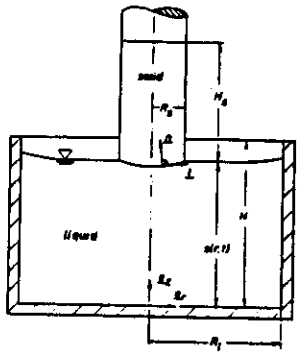
made on the basis of experience only.
If we are convinced that the solution can be obtained only by approximate methods, there is of course the choice between an analytic approximation method, like a truncated expansion into eigenfunctions, and numerical methods, like finite differences or finite element methods. It is obvious that in the case of an analytic approximation, Computer Algebra is a very viable resource, but it can also do a great deal for the numerical treatment. Computer Algebra systems usually can produce output in a language for numerical processing, such as Fortran. In the case of finite differences, the translation of the differential equations into appropriate finite difference equations can be performed automatically, and output can be generated in a form for immediate numerical processing. In the case of finite elements ,the various matrices needed can be generated and optimized by the Computer Algebra system.
Finally, the very tedious but very important task of investigating the stability of the numerical code by a local Fourier analysis can be performed with the help of a Computer Algebra system.

The plan for the lectures is to give some examples of the topics just mentioned. Some of them are quite simple, but they should provide enough information on how to proceed in more involved cases. The participants of the school should feel invited to redo the examples in these notes and to construct their own examples.

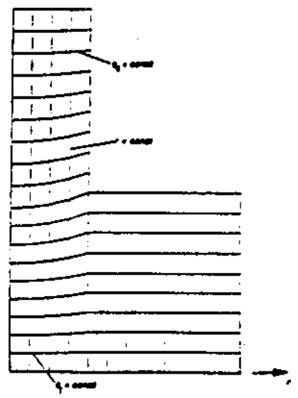
2 Preprocessing

2.1 Boundary fitted Coordinates

The need and usefulness for the transformation of the original equations to a coordinate system adapted to a problem can be very nicely demonstrated in the case of a free boundary. In general, for the formulation of boundary conditions (and also in view of an easy discretisation for a numerical study) the coordinate lines are profitably chosen in such a way that some of them form the boundary. However, in the case of a free boundary, only its initial shape is known. Coordinates fitted to the boundary therefore cannot be given explicitly and are part of the calculation itself.
To be more concrete, let's consider a problem from a simulation of a crystal growth experiment. In the beginning of the experiment, the melt is in a crucible with a flat free surface. A crystal is then grown by putting a cool piston into the melt and pulling out the solidifying melt on the piston. In the course of the crystal growth, the level of the melt surface lowers as more and more of the melt is converted into the crystal. The shape of the free surface of the melt depends, of course, on parameters like the angular velocity of the crucible, the surface tension of the melt, and the pulling rate of the crystal.



In the above figure we have indicated cylindrical coordinates. The height of the melt is a function of time and the radial distance. We can map the problem to a rectangular fixed region if we use a coordinate q, given by $q = H * z/s(r,t)$, instead of the z-coordinate. Here H is the initial height of the melt surface and $s(r,t)$ is a function describing its shape. With this coordinate transformation the free surface of the melt is always at $q = H$. In the interior, the coordinates form a nonorthogonal mesh varying with time.



We are thus left with the problem of expressing the equations governing the growth process in this coordinate system. It is not our aim to discuss these equations. Rather, we look at their form to see what is involved (for a discussion see [10]). In the following equations, $\vec{u}$ denotes the velocity, Φ the gravitational potential, β the thermal expansion coefficient, g the gravitational acceleration, T the temperature, ν the viscosity coefficient, κ the heat conductivity coefficient and w_p the pulling rate.

Navier-Stokes equation:

$$\partial_t \vec{u} = -\vec{u}\, grad\, \vec{u} - grad\, \Phi + \beta * g * (T_l - \bar{T}_l) * \vec{e}_z + \nu * \Delta \vec{u}$$

Mass conservation:

$$div\, \vec{u} = 0$$

Heat flow in the liquid (index l):

$$\partial_t T_l = -div(\vec{u} * T_l) + \kappa_l * \Delta T_l$$

Heat flow in the solid (index s):

$$\partial_t T_s = -(w_p(t) * \vec{e}_z) * grad\, T_s + \kappa_s * \Delta T_s$$

In addition to this, there are, of course, boundary conditions.
The transformation can easily be performed with the Computer Algebra system REDUCE [5] by using the package EXCALC [12] in the REDUCE library. EXCALC is an implementation of the exterior calculus, of which, however, we need at the moment very little to know. Let us recall that a reference frame in Euclidean or Riemannian geometry is specified by a set of basis vectors and a metric. For those not familiar with this notion, let us take the expression for the square of a line element

$$ds^2 = g_{ik} * dx^i * dx^k$$

The symmetric matrix g_{ik}, the metric tensor, together with the basis dx^l determine the frame of reference. Since it is possible to partition the expression for ds^2 in many different ways, we also write:

$$ds^2 = g_{ik} * e^i * e^k$$

The e^i's are now arbitrarily chosen basis elements (for example, one of them could be $e^\varphi = r * d\varphi$ in an orthogonal cylindrical frame).
In our case, we start from cylindrical coordinates $ds^2 = dr^2 + r^2 * d\varphi^2 + dz^2$.
To introduce the new coordinate $q = H * z/s(r,t)$ we simply express dz in the new coordinates. As a basis we choose dr, $d\varphi$ and dq.
Our frame of reference could now be defined in EXCALC by:

```
fdomain s=s(r,t);
pform z=0,q=0,s=0,r=0;

let z = q*s/h;

d z;

 d Q*S + d R*@ S*Q
               R
-------------------
         H
```

```
coframe e(r)   = d r,
        e(phi) = d phi,
        e(q)   = d q   .
 with metric
   g = e(r)**2 + r**2*e(phi)**2 +
       (e(q)*s/h + e(r)*q*@(s,r)/h)**2;
```

To calculate expressions for mass conservation, heat conduction etc., we need to know how to express operations of vector analysis in the language of exterior forms. Here is a table:

Vector product	$\vec{u} \times \vec{v}$	`#(u ^ v)`
Gradient	$grad\, f$	`d f`
Laplacian	Δf	`#d#d f`
Curl	$curl\, \vec{u}$	`d u`
Divergence	$div\, \vec{u}$	`#d# u`
C-derivative	$(\vec{u}\, grad)\, \vec{u}$	`U |_ u - 1/2*d(U _| u)`
Vector Laplacian	$\Delta \vec{u}$	`#d#d u - d#d# u`

All we have to do for the continuity equation is to define the velocity vector and declare the dependence of its components on r and q:

```
fdomain u=u(r,q);
pform u=1,u k=0;

u := u(k)*e(-k)$

 # d # u;

                        @ S*R + S
   R         Q      R   R
@ (U ) + @ (U ) + U *------------
 R         Q             R*S
```

and, equally simple, the advection and diffusion term of the heat conduction equation.

```
fdomain temp=temp(r,q);
pform temp=0;

#d# (u*temp);
```

```
                   R          R         Q            R
                  U *@ S*R + U *S + @ (U )*R*S + @ (U )*R*S
        Q             R              Q            R
@ TEMP*U  + TEMP*-------------------------------------------
 Q                               R*S

    R
 + U *@ TEMP
       R
```

```
#d#d temp;
```

```
                               2  2    2
                            @ S *Q  + H
                             R
 - ( - @   TEMP - @   TEMP*--------------- + @ TEMP*Q*
        R R        Q Q            2           Q
                                 S

                      2
     @   S*R*S - 2*@ S *R + @ S*S
      R R           R        R
    ------------------------------ +
                    2
                 R*S
```

```
     2*@ S*@   TEMP*Q*R - @ TEMP*S
        R   Q R            R
    --------------------------------)
                  R*S
```

To fully appreciate how much work has been automized this way, let us finally calculate the viscous term in the Navier Stokes equation, i.e. the vector Laplacian of the velocity field.

```
frame x;
pform visc(k)=0;

visc(k) := x(k) _| (d#d# u - #d#d u);
```

```
                                         2  2    2
                                      @ S *Q  + H
    R                 R          R     R
VISC  :=  - ( - @   (U ) - @   (U )*-------------- +
                 R R        Q Q             2
                                           S

                      R                   R
              2*@   (U )*@ S*Q      - @ (U )
                 Q R      R            R
             ------------------ + ----------- +
                     S                 R

                                        2
                       @   S*R*S - 2*@ S *R + @ S*S     R
                 R      R R           R        R       U
             @ (U )*Q*----------------------------- + ----)
              Q                       2                 2
                                   R*S                 R
```

```
                                         2  2    2
                                     @ S *Q  + H
     PHI          PHI          PHI    R
VISC     := @   (U   ) + @   (U   )*-------------- +
            R R          Q Q                2
                                          S


                        PHI                  PHI
              - 2*@   (U   )*@ S*Q     3*@ (U   )
                  Q R         R           R
           ----------------------- + ------------ +
                       S                 . R


                                           2
                        - @   S*R*S + 2*@ S *R - 3*@ S*S
               PHI          R R            R          R
           @ (U   )*Q*-----------------------------------)
            Q                            2
                                    R*S


                                   2
                     @   S*S - @ S
    Q          R      R R        R            R
VISC  := 2*@ (U )*Q*---------------- + 2*@ (U )*@ S*
             R               2             Q       R
                            S


                    2        2  2    2
           - @   S*Q *S + @ S *Q  + H
              R R          R                    Q          Q
         ------------------------------- + @   (U ) + @   (U )*
                     . 3                   R R        Q Q
                        S
```

```
    2  2    2                Q
 @ S *Q  + H        - 2*@    (U )*@ S*Q
  R                      Q R        R
-------------- + ---------------------- +
       2                   S
      S

        2*@ S*R + S                 @   S*R + @ S
    Q       R                 Q     R R         R
@ (U )*------------- - (@ (U )*Q)*---------------- +
 R          R*S           Q             R*S

 R              2  2                2              2
U *Q*(@      S*R *S  - 3*@   S*@ S*R *S + @   S*R*S  +
       R R R               R R    R          R R

     3  2      2            2    2  3
2*@ S *R  - @ S *R*S + @ S*S )/(R *S ) +
   R          R          R

    @   S*R + @ S
 Q   R R         R
U *---------------
         R*S
```

The obtained equations could now be used either to write a finite difference program or a finite element code to solve them.

2.2 Generation of Finite Difference Expressions and Local Fourier Stability Analysis

The translation of the differential equations into a finite difference approximation can be automated, or at least assisted, by a Computer Algebra system. Let us recall the basics of the finite difference method: first, the continuous domain of the variables is replaced by a discrete domain, a grid. If we have a function f depending on t, x, y, for example, one writes

$$f^n_{kl}$$

to denote the function at the point $f(n * \Delta t, k * \Delta x, l * \Delta y)$. Next, an approximation to the differential operators is constructed using values on the grid points. The so-called forward differences, for example, approximate a derivative by using values at the point of evaluation and a point lying one grid point ahead.

$$\begin{aligned}&(f(x+\Delta x) - f(x))/\Delta x\\ &= (f(x) + \acute{f}(x) * \Delta x - f(x))/\Delta x + O(\Delta x)\\ &= \acute{f}(x) + O(\Delta x)\end{aligned}$$

The two most common differences used are:
Forward difference:

$$(f(k+m) - f(k))/(m * \Delta)$$

Central difference:

$$(f(k+m/2) - f(k-m/2))/(m * \Delta)$$

It is easy to implement operators like this (see also [17] for a realization in MACSYMA). We do this here in REDUCE.
First, we associate with any coordinate its discrete analog and a corresponding increment by using operators IND and DELTA. If it is supposed that our differential equation involves coordinates t and x, the statements

```
operator ind,delta;
ind(tt) := n$
ind(x)  := k$
delta(tt) := dt$
delta(x) := dx$
```

set up the correspondence. All finite difference operators can be expressed by shifts in certain coordinate directions. Therefore we define a shift operator for this purpose:

```
algebraic procedure shift(u,x,k);
  %Shift expression u in direction x by k.
  sub(ind(x)=ind(x) + k,u);
```

Procedures for calculating forward and central differences now can be easily constructed

```
algebraic procedure fdiff(u,x,k);
  %Forward differencing in x by k steps.
  (shift(u,x,k) - u)/(k*delta(x));

algebraic procedure cdiff(u,x,k);
  %Central differencing in x by k steps.
  (shift(u,x,k/2) - shift(u,x,-k/2))/(k*delta(x));
```

Assuming we want to discretize the simple equation

$$\partial_t u = \partial_{x\,x}\, u^5$$

by forward differences in time and central differences in space, we could say:

```
pde := dif(u,tt) - dif(u**5,x,x)$
for all ex let dif(ex,tt) = fdiff(ex,tt,1);
for all ex let dif(ex,x,x) = cdiff(cdiff(ex,x,1),x,1);
operator u;
let u = u(n,k);
factor dt; on rat;
pde;
              5             5           5
  - U(N,K - 1)  - U(N,K + 1)  + 2*U(N,K)
-----------------------------------------
                      2
                    DX
```

```
       -1
 + DT    *(U(N + 1,K) - U(N,K))
```

We can solve for u at the advanced time level and output the expression in FORTRAN, which in turn could be used as part of a FORTRAN program to solve the differential equation.

```
solve(pde,u(n+1,k))$
on fort; off period;
u(n+1,k) := rhs first ws;

      U(N+1,K)=(DT*(U(N,K-1)**5+U(N,K+1)**5-2*U(N,K)**5))/
     . DX**2+U(N,K)5
```

Of course, this was a very simple problem, and a real appreciation of such a software tool is gained by more complex tasks like the one of the crystal growth.
It is obvious that we could write a complete program generator this way. Besides the simple switch ON FORT that caused the translation of output into FORTRAN syntax, a more advanced generator/translator package called GENTRAN [9] is available in REDUCE. The interested reader should consult the documentation.
Very often the expressions generated by a Computer Algebra system are very long and unstructured. It frequently happens that naively generated code is of a complexity not digestable by FORTRAN compilers. Therefore software tools for structuring and structure preserving [6] and code optimization [8,2] have been developed or are under construction. The effects of such programs can be quite stunning. Interesting examples can be found in the cited literature.
Having generated a finite difference code, it is very important to investigate its stability, depending on the choice of the spacings in the coordinates. Roughly speaking, we should expect from our finite difference equations that small perturbations should have a small effect on the solution.

We will demonstrate the method on the simple differential equation above. For the discretization, we again take forward differences in time; however, for the space derivatives we take weighted means of central differences at the actual time and the advanced time i.e.

```
for all ex let df2(ex,x,x)=(theta*shift(dfc2,tt,1) +
                             (1 - theta)*dfc2)
                        where dfc2 = cdiff(cdiff(ex,x,1),x,1);
pde;


                    5                          5
 - ((U(N + 1,K - 1) *THETA + U(N + 1,K + 1) *THETA

                     5                       5               5
     - 2*U(N + 1,K) *THETA - U(N,K - 1) *THETA + U(N,K - 1)

                  5                      5              5
     - U(N,K + 1) *THETA + U(N,K + 1)  + 2*U(N,K) *THETA

                 5    2       -1
     - 2*U(N,K) )/DX  + DT    *( - U(N + 1,K) + U(N,K)))
```

The equation obtained is nonlinear at the advanced time level, and we linearize in u(n+1,...).

```
for all k,r such that r>0
    let u(n+1,k)**r = r*u(n+1,k)*u(n,k)**(r-1)
                      -(r-1)*u(n,k)**r;

pde := pde$
```

Now we assume that there is a solution for u and study its behaviour under small perturbations, i.e. we add perturbation terms, keep them only to first order and subtract the zero order terms as they are assumed to be a solution:

```
for all k,l let u(k,l)=u0(k,l)+u1(k,l)*eps;
weight eps=1;
ppde1 := pde$
for all k,l clear u(k,l);
for all k,l let u0(k,l)=u(k,l);
ppde := ppde1 - pde$
ppde := ppde/eps$
```

We haven't shown the result for ppde, as it is somewhat messy. For the further analysis we replace the remaining unperturbed $u(n,k)$'s by some representative value u. At the same time we introduce a Fourier ansatz for the perturbation $u1(n,m) = e^{n*(i*kt*dt)} * e^{m*(i*kx*dx)} = lamb^n * kappa^n$.

```
clear u;
for all k,l let u(k,l)=u;
for all k,l let u1(k,l)=lamb**k*kappa**l;
ppde := ppde$
solve(ppde,lamb)$
on factor;
rhs first ws;


                                     2   4
   5*DT*(THETA - 1)*(KAPPA - 1) *U
-----------------------------------------
                  2   4           2
 5*DT*(KAPPA - 1) *U *THETA - DX *KAPPA

                     2
                   DX *KAPPA
 - ----------------------------------------
                   2   4           2
   5*DT*(KAPPA - 1) *U *THETA - DX *KAPPA
```

The difference equations are stable if the step sizes can be chosen such that there is no exponential growth in the perturbation as time goes on. This

means that the absolute value of LAMB must be bound by 1. In the case of THETA= 0:

```
let theta=0;
ws;

                   2   4
 5*DT*(KAPPA - 1) *U
--------------------- + 1
          2
        DX *KAPPA
```

we readily obtain that we must have

$$DT < DX^2/(10 * U^4)$$

I.e. there is a maximal allowable step size for the time, depending on the step size in space and the amplitude of the solution. The latter dependence on the amplitude is a characteristic of nonlinear problems. The analysis for arbitrary values of THETA is more involved and we leave this to the reader and only note that a good choice of THETA improves stability appreciably. This simple example should have demonstrated the usefulness of a Computer Algebra system for stability analysis. In the case of several dependent variables there is of course a system of equations for the perturbations, and the stability criterium comes out of the calculation of a characteristic polynomial, which can be quite a formidable undertaking, even with a Computer Algebra system at hand.

3 Intrinsic Study of Differential Equations

When a normal person attempts to solve a differential equation, very often the only solution methods used are 'tricks'. It is very rare that a systematic approach is taken. The reason for this is not because of the lack of theories. Rather, the existing systematic methods are considered to be so cumbersome in application that they seem impractical. With the availability of Computer Algebra systems, however, this argument no longer holds. Candidate theories are those of Lie, Cartan, Differential Galois theory and those which are generalizations of the methods used in integrating normal functions. In the following we use Cartan's theory of differential equations, which uses the calculus of differential forms. An implementation of this calculus is available in REDUCE through the EXCALC package. Related implementation efforts can be found in the references [7,4]. We can treat ordinary and partial differential equations at the same time as they can be formulated and attacked by the same machinery.

In Cartan's theory the differential equations are expressed as differential form expressions. A single first order ordinary differential equation, for example, is written as

$$\omega = d\,y - f(x,y) * d\,x.$$

Similiarly, a second order ordinary differential equation has the representation

$$\omega^1 = d\,y1 - f(x,y,y1) * d\,x$$

$$\omega^2 = d\,y - y1 * d\,x$$

and the partial differential equation $f(x,y,z,\partial_x z,\partial_y z)$ could be expressed as

$$\omega^1 = d\,z - p * d\,x - q * d\,y$$

$$\omega^2 = d\,f(x,y,z,p,q)$$

$$\omega^3 = d\,p \wedge d\,x + d\,q \wedge d\,y\,.$$

The equations appearing above are called exterior equations, or differential systems. Before proceeding we have to recall some basic facts from the calculus of differential forms, or the Exterior calculus as it is also called. There are excellent textbooks on this matter [1,3,14,16].

3.1 Differential Forms and Vectors

The basic objects of the Exterior Calculus are exterior forms and vectors. A prototype of an exterior form is the differential of a function

$$d\,f$$

In local coordinates x^i the differential of the function f can be expressed as

$$d\,f = \partial_{x^i} f * d\,x^i$$

We therefore call *df* also the gradient of f. It is customary to visualize forms by surfaces in space.

The above exterior form is a special form. If we adopt the $d\,x^i$ as a basis, the most general 1-form can be written as

$$\omega = \alpha_i * d\,x^i$$

where the α_i's are now general, not necessarily derivatives of a function.

It is useful to call a scalar function a 0-form.

A more common object is a vector with its visualization as an arrow. To give it a more precise meaning, consider a curve x^i with parametrization t and a function F. Taking the total derivative with respect to t gives

$$\frac{d\,F}{d\,t} = \frac{d\,x^i}{d\,t} * \partial_{x^i} F$$

This equation has the same form for any F, and we therefore write

$$\frac{d}{d\,t} = \frac{d\,x^i}{d\,t} * \partial_{x^i}$$

In the usual view $\frac{dx^i}{dt}$ are the components of the tangent to the curve, and we can view the symbol ∂_{x^i} as a basis for the tangent vector. This naturally leads us to write for a vector

$$u = w^i * \partial_{x^i}$$

in the coordinate system x.

In EXCALC we can introduce the exterior 1-form ω and the 0-form f by the declaration

```
    pform omega=1,f=0;
```

To have the function f depend on the coordinates x and y and to calculate the gradient, we simply write

```
    pform x=0,y=0;
    fdomain f=f(x,y);
```

and ask for $d\,f$:

```
    d f;

      @ f*d x + @ f*d y
       x         y
```

A vector u is introduced by the statement

```
    tvector u;
```

and its coordinate representation would be

```
    u := ux*@ x + uy*@ y;

      U := UX*@  + UY*@
               X       Y
```

In the older literature the 1-forms are called covariant vectors and the vectors above contravariant vectors.

3.2 Exterior Multiplication and p-forms

Consider an integral over a 2-dimensional surface element in cartesian coordinates

$$f(x,y) * d\,x * d\,y$$

To obtain the same integral in a different coordinate system u,v, say, it is not sufficient to replace the differentials by their transformed ones. Instead, one has to calculate the following

$$f(u,v) * \frac{\partial(x,y)}{\partial(u,v)} * d\,u * d\,v$$

where the factor before the differentials is the Jacobian. We can write this in the following way

$$f(u,v) * (\partial_u x * d\,u * \partial_v y * d\,v - \partial_v x * d\,v * \partial_u y * d\,u)$$

This suggests a definition of an antisymmetric product between the differentials called "exterior multiplication" which is usually denoted by $\wedge$ (wedge).

$$d\,x \wedge d\,y = -d\,y \wedge d\,x$$

With this, we can rewrite the integral in u,v coordinates as

$$f(u,v) * d\,x(u,v) \wedge d\,y(u,v),$$

thus leading to the correct expression by simply transforming the differentials.

The objects obtained by exterior multiplication of two 1-forms are called 2-forms, and those by exterior multiplication of three 1-forms are called 3-forms, and so on. They could be pictured by intersecting surfaces, i.e. tubes, honeycombs, etc., but this is of limited value.
We can ask how many different forms of degree p in an n-dimensional space can be formed. The value is

$$dim(p - forms\;in\;n - dimensions) = \binom{n}{p}$$

```
Exercise: Verify this formula by taking general 1-forms with
          coordinates x,y,z,t and multiply them to get the
          higher forms
```

Let α be a k-form and β be a l-form. Then with the above we have the following commutation law

$$\alpha \wedge \beta = (-1)^{k*l} * \beta \wedge \alpha$$

The exterior product gives an easily applicable criterium for linear dependence between 1-forms. It is obvious that the exterior product between two 1-forms which are a multiple of each other vanishes due to the antisymmetry of the product. By explicit calculation it is easy to show that a 1-form is linear dependent on a set of other 1-forms if the exterior product of this 1-form with all others vanishes.

3.3 Exterior derivative and Integrability Conditions for Differential Equations

We already used the symbol d to generate from a 0-form f the 1-form $d\,f$. The generalization of the exterior derivative to act on a form of arbitrary degree can be inductively defined by giving a rule for applying it to an exterior product

$$d\,(\alpha \wedge \beta) = (d\,\alpha) \wedge \beta + (-1)^{deg(\alpha)} * \alpha \wedge d\,\beta$$

and by defining the exterior derivative of an exterior derivative to yield zero "$d\,d = 0$".

That this is a reasonable definition can be seen by taking d of the gradient $d\,f$:

$$d\,d\,f = d(\partial_{x^i} f * d\,x^i) = d(\partial_{x^i} f) \wedge d\,x^i = \partial_{x^i\,x^j} f * d\,x^i \wedge d\,x^j = 0$$

This expresses the commutativity of the partial derivatives and can also be interpreted as the rule from vector analysis "curl grad = 0". These relations are the heart of integrability conditions of partial differential equations. Through the application of the exterior derivative to a set of exterior equations we obtain new equations, which, if not already contained in the original set have to be added to the set, as they constitute independent constraints on the functions involved. If no new equations are obtained, one says the differential system is closed.

There are, of course, no integrability conditions for ordinary differential equations, which can be seen from the fact that all differentials can be expressed by the differential of the independent variable and hence all exterior products vanish.

Example: Integrability conditions

Calculate the integrability conditions of the following 4 linear pdes:

$$\frac{\partial z_1}{\partial x} + a1 * z_1 + b1 * z_2 = c1$$
$$\frac{\partial z_1}{\partial y} + a2 * z_1 + b2 * z_2 = c2$$
$$\frac{\partial z_2}{\partial x} + f1 * z_1 + g1 * z_2 = h1$$

$$\frac{\partial z_2}{\partial y} + f2 * z_1 + g2 * z_2 = h2$$

```
pform w(k)=1,integ(k)=4,z(k)=0,x=0,y=0,a=1,b=1,c=1,f=1,g=1,h=1,
      a1=0,a2=0,b1=0,b2=0,c1=0,c2=0,f1=0,f2=0,g1=0,g2=0,h1=0,h2=(

fdomain  a1=a1(x,y),a2=a2(x,y),b1=b1(x,y),b2=b2(x,y),
         c1=c1(x,y),c2=c2(x,y),f1=f1(x,y),f2=f2(x,y),
         g1=g1(x,y),g2=g2(x,y),h1=h1(x,y),h2=h2(x,y);

a:=a1*d x+a2*d y$
b:=b1*d x+b2*d y$
c:=c1*d x+c2*d y$
f:=f1*d x+f2*d y$
g:=g1*d x+g2*d y$
h:=h1*d x+h2*d y$

%The equivalent exterior system:;
factor d;

w(1) := d z(-1) + z(-1)*a + z(-2)*b - c;

 1
W  := d Z  + d X*(Z *A1 + Z *B1 - C1) + d Y*(Z *A2 + Z *B2 - C2)
         1         1      2                   1      2

w(2) := d z(-2) + z(-2)*f + z(-1)*g - h;

 2
W  := d Z  + d X*(Z *G1 + Z *F1 - H1) + d Y*(Z *G2 + Z *F2 - H2)
         2         1      2                   1      2

indexrange 1,2;

factor z;
```

```
%The integrability conditions:;
integ(k) := d w(k) ^ w(1) ^ w(2);

     1
INTEG  := Z *d Z ^d Z ^d X^d Y*(@ A2 - @ A1 + G2*B1 - G1*B2) + Z *
          1    1    2            X      Y                        2

         d Z ^d Z ^d X^d Y*(@ B2 - @ B1 + F2*B1 - F1*B2 + B2*A1 -
            1    2              X      Y

             B1*A2) + d Z ^d Z ^d X^d Y*( - @ C2 + @ C1 - H2*B1 +
                         1    2                 X      Y

             H1*B2 - C2*A1 + C1*A2)

     2
INTEG  := Z *d Z ^d Z ^d X^d Y*(@ G2 - @ G1 + G2*F1 - G2*A1 - G1*
          1    1    2            X      Y

          F2 + G1*A2) + Z *d Z ^d Z ^d X^d Y*(@ F2 - @ F1 - G2*B1
                        2    1    2            X      Y

             + G1*B2) + d Z ^d Z ^d X^d Y*( - @ H2 + @ H1 - H2*F1
                           1    2                 X      Y

             + H1*F2 + G2*C1 - G1*C2)
```

3.4 The Inner Product between Vectors and Forms

The inner product is the familiar concept of a scalar product or the contraction of a contravariant and covariant quantity. The inner product between a vector and a 1-form yields a number. This number can be imagined as the number of surfaces that are pierced by the arrow of the vector. We use the symbol $\rfloor$ to denote the inner product.
The natural bases dx^i of forms and ∂_{x^i} of vectors are dual. This means

$$\partial_{x^i} \rfloor dx^j = \delta^j_i,$$

where δ is the Kronecker symbol.
The definition of the inner product can be extended to arbitrary degree forms by giving a rule for forming the inner product between a vector and an exterior product.
Let EXCALC print the rule for us:

```
tvector u;
pform x=k,y=1;

u _| (x^y);

                              K
   (U_|X)^Y + ( - 1) *X^(U_|Y)
```

The inner product between a vector and a 0-form is taken to be zero.

3.5 Characteristic Vectors and Formal Solutions of Ordinary Differential Equations

We can visualize a vector field as a smooth distribution of arrows in space. These arrows can be thought to be the tangents to space-filling curves (a congruence). The curves can be calculated by exponentiating the vector field:

$$x^j(t) = \{e^{t*u^i*\partial_{x^i}}\}\, x^j,$$

where t is a parametrization of the curve. It is natural to ask if we can construct a vector such that the generated curves are solutions to a given differential system, i.e. to a set of corresponding differential equations. To explain the principle of such a construction it suffices to consider a single first order differential equation, represented by $d\,y - f(y,x) * d\,x$. The vector field in question has the form $v = v^y * \partial_y + v^x * \partial_x$. If we now calculate $d\,y(t)$ and $d\,x(t)$ for small values of t, we get:

$$d\,y(t) = d\,t * (v \rfloor d\,y) + O(t)$$
$$d\,x(t) = d\,t * (v \rfloor d\,x) + O(t)$$

The condition for v to generate a solution follows from

$$d\,y(t) - f(y,x) * d\,x(t) = d\,t * (v \rfloor (d\,y - f * d\,x)) + O(t)$$

which in turn gives the condition

$$v \rfloor (d\,y - f(y,x) * d\,x) = 0$$

This vector field is called the characteristic vector field.
If there are several equations in the differential system, v has to annihilate all 1-forms and map all higher forms to forms already in the differential system. The determination of the characteristic vectors amounts to the solution of a system of linear algebraic equations. In general there need to be no solutions to these linear equations. However, as can be inferred from the calculation just carried out, they always exist for ordinary differential equations.
We can use this result to construct power series solutions to systems of ordinary differential equations.
As an example we solve the equation of motion for an electron in a constant perpendicular electromagnetic field of equal strength. The differential equations are:

$$\acute{u}^t = E_x * u^x$$
$$\acute{u}^x = E_x * u^t - u^z * B_y$$
$$\acute{u}^y = 0$$
$$\acute{u}^z = u^x * B_y$$

The corresponding differential system is:

$$d\,u^t - E_x * u^x * d\,s$$

$$d\,u^x - (E_x * u^t - u^z * B_y) * d\,s$$
$$d\,u^y = 0$$
$$d\,u^z - u^x * B_y * d\,s$$

The characteristic vector can be written down just by inspection. At this end we calculate with EXCALC the exponentiation of the vector:

```
algebraic procedure exponential(r,u,w,v0,n);
  %r is the parametrisation variable, u the vector,
  %w the name of the component sought,
  %v0 a list of initial values and n the number of
  %terms to be calculated.
    begin scalar y; integer k;
      k := 1;
      y := w;
      return sub(v0,w)+
             for m := 1:n sum
                 r**m*sub(v0,y := u _| d y)/(k := k*m)
    end$

pform ut=0,ux=0,uy=0,uz=0,s=0;

let ex=by;

v := ex*ux*@ ut + (ex*ut - uz*by)*@ ux + (ux*by)*@ uz + @ s$

exponential(tau,v,ut,{ut=ut0,ux=ux0,uy=uy0,uz=uz0,s=0},4);

    2   2
 TAU *BY *(UT0 - UZ0)
---------------------- + TAU*BY*UX0 + UT0
          2

exponential(tau,v,ux,{ut=ut0,ux=ux0,uy=uy0,uz=uz0,s=0},4);
```

```
TAU*BY*(UTO - UZO) + UXO

exponential(tau,v,uy,{ut=ut0,ux=ux0,uy=uy0,uz=uz0,s=0},4);

UYO

exponential(tau,v,uz,{ut=ut0,ux=ux0,uy=uy0,uz=uz0,s=0},4);

     2   2
 TAU *BY *(UTO - UZO)
---------------------- + TAU*BY*UXO + UZO
          2

exponential(tau,v,s,{ut=ut0,ux=ux0,uy=uy0,uz=uz0,s=0},4);

TAU
```

In the above example there are no higher terms than quadratic in TAU, even though we asked for terms up to order four. The reason is that the series indeed terminates with quadratic terms.
The simple procedure EXPONENTIAL can be used to calculate power series solutions to arbitrary systems of ordinary differential equations and can also be used for partial differential equations in the case when characteristic vectors exist, which is rare, however.

3.6 Lie Derivative and Symmetries of Differential Equations

As we have seen, a vector field maps, through its congruences, a space into itself.
It is possible to define a derivative telling us how much a form changes under the transformation given by the vector field. This derivative is called the Lie derivative.

$$u \lfloor \omega = \lim_{t\to 0}(\omega_i(x^j(t)) * d\,x^i(t) - \omega_i(x^j) * d\,x^i)/t$$

$$= (\partial_{x^j}(\omega_i) * u^j + \omega_j * \partial_{x^i}(u^j)) * d\,x^i$$

Expressing the above with the operations d and inner product, we get a formula which is valid also for forms of any degree.

$$u \lfloor \omega = d(u \rfloor \omega) + u \rfloor d\omega$$

If the Lie derivative vanishes, one says the form is Lie dragged by the vector field.
For a 0-form, the first term on the right-hand side vanishes, and Lie dragging a function means that the gradient of the function in the direction of the vector field is zero, or equivalently, that the function is constant along each individual curve of the congruence. This makes clear that the existence of vector fields which "Lie drag" an exterior form is connected to the symmetries of the exterior form.
Of special interest are transformations of the differential system that leave it unchanged. Infinitesimally, these transformations are generated by vector fields which are determined by the condition that the Lie derivative between the vectors and the differential system are forms in the differential system. Vector fields satisfying this condition are called isovectors.
The global transformation takes one solution into another one.
The study of the isovectors is of great importance for the understanding of the problem, for the reduction of the number of variables, conservation laws, etc..
The equations for the isovectors S

$$S \lfloor \mathfrak{I} \subset \mathfrak{I}$$

are linear first order pdes. Their derivation is in general quite currbersome, and Computer Algebra definitely is the right tool to generate the equations. To give an impression of the complexity: in the case of Maxwell's equation in vacuum one ends up with 176 equations.
There is the distinction of two classes of symmetries depending on the allowed domain of the coefficients of the isovector. If the coefficients of the vector-components in the direction of the dependent and independent var ables are allowed only to depend on the dependent and independent variables and not on the derivatives, one speaks of point symmetries, otherwise of generalized symmetries. We will first give an example of point symmetries of the heat equation and then generalized symmetries of the Kepler problem.
A differential system for the heat equation

$$\partial_t \Psi - \partial_{xx} \Psi = 0$$

can be generated by the exterior forms h0, h1 and h2 below, where *ps* denotes Ψ, *u* denotes $\partial_t \Psi$ and *ph* denotes $\partial_x \Psi$. It is easy to check that the system is closed (exercise!). Note the domains of the coefficients of the isovector, which are chosen to yield point symmetries.

```
pform ct=0,cx=0,cu=0,cps=0,cph=0,t=0,x=0,u=0,ps=0,ph=0;

h0 := d ps - u*d t - ph*d x$
h1 := u*d x^d t - d ph^d t$
h2 := d ph^d x + d u^d t$

fdomain cx=cx(x,t,ps),ct=ct(x,t,ps),cps=cps(x,t,ps),
        cu=cu(x,t,ps,u,ph),cph=cph(x,t,ps,u,ph);

s := cx*@ x+ct*@ t+cu*@ u+cps*@ ps+cph*@ ph;

S := @ *CT + @ *CU + @ *CX + @  *CPS + @  *CPH
     T       U       X       PS       PH

factor d;
```

```
lh0 := s |_ h0$

lh0 := sub(d ps=u*d t + ph*d x,lh0);

                                          2
LH0 :=  - (d T*(@ CT*U + @  CT*U  + @ CX*PH + @  CX*U*PH
                T        PS         T         PS

              - @ CPS - @  CPS*U + CU) + d X*(@ CT*U
                T       PS                    X

                                                   2
             + @  CT*U*PH + @ CX*PH + @  CX*PH  - @ CPS
               PS           X         PS          X

             - @  CPS*PH + CPH))
               PS

factor ^;

lh1 := s |_ h1$

lh1 := sub(d t^d ph=u*d t^d x,d x^d ph=d u^d t,
           d ps=u*d t + ph*d x, lh1);

                                                      2
LH1 :=  - (d T^d U*(@ CT - @ CPH) + d T^d X*(@  CT*U
                    X      U                 PS

             + @ CX*U + @  CX*U*PH - @ CPH - @  CPH*PH
               X        PS           X       PS

             - @  CPH*U + CU) - d T^d PH*@  CT*U
               PH                        PS
```

```
          - d X^d PH*@  CT*PH)
                      PS

lh2 := s |_ h2$

lh2 := sub(d t^d ph=u*d t^d x,d x^d ph=d u^d t,
           d ps=u*d t + ph*d x, lh2);

LH2 :=  - (d T^d U*(@ CT + @  CT*U - @ CX + @ CU - @  CPH)
                     T     PS        X      U      PH

          + d T^d X*(@ CX*U + @ CU + @  CU*PH + @  CU*U
                      T        X     PS          PH

             - @ CPH - @  CPH*U) + d T^d PH*@  CX*U
               T       PS                   PS

           - d U^d X*(@ CT + @  CT*PH + @ CPH)
                      X     PS         U

           + d X^d PH*@  CX*PH)
                      PS

eq1 := coeffn(lh0,d t,1)$
eq2 := coeffn(lh0,d x,1)$
eq1 := eq1 + cu$
eq2 := eq2 + cph$
let cu=eq1, cph=eq2;
lh1 := lh1$
lh2 := lh2$

eqs := {}$

for each j in {t,x,ps,u,ph} do
```

```
  for each k in {t,x,ps,u,ph} do
     (if ordp(j,k) and (j neq k) and (x neq 0)
         then eqs := x . eqs)
               where x = coeffn(lh1,d j^d k,1);

for each j in {t,x,ps,u,ph} do
  for each k in {t,x,ps,u,ph} do
     (if ordp(j,k) and (j neq k) and (x neq 0)
         then eqs := x . eqs)
               where x = coeffn(lh2,d j^d k,1);

eqs;

{ - @  CX*PH,
    PS

  - @  CX*U,
    PS

  - @  CX*PH,
    PS

        2
 @  CX*U ,
 PS

 @  CT*PH,
 PS

 @  CT*U,
 PS

  - (2*@ CT + @  CT*PH),
       X      PS
```

```
                                                        2
@ CT*U - @   CT*U - 2*@    CT*U*PH - @     CT*U*PH
 T         X X         X PS            PS PS

        2                                            2
 - @  CT*U  + @ CX*PH - @   CX*PH - 2*@    CX*PH
    PS        T          X X          X PS

                                  3
 - 2*@ CX*U - @     CX*PH  - 2*@  CX*U*PH - @ CPS
      X         PS PS             PS          T

                                          2
 + @   CPS + 2*@    CPS*PH + @     CPS*PH }
    X X          X PS           PS PS

let @(cx,ps)=0,@(ct,ps)=0,@(ct,x)=0;
factor ph,u;

eqs;

{0,

 0,

 0,

 0,

 0,

 0,

 0,
```

```
  2
PH *@     CPS + PH*(@ CX - @   CX + 2*@    CPS)
    PS PS            T      X X         X PS

 + U*(@ CT - 2*@ CX) - @ CPS + @   CPS}
      T         X       T       X X
```

Interactively the solution of the former equations can be found very easily:

```
S1 := @
       T

S2 := @
       X

S3 := PS*@   + PH*@   + U*@
          PS       PH      U

S4 := 2*T*@   + X*@   - PH*@    - 2*U*@
           T       X        PH         U

S5 := - 2*T*@ + X*PS*@   + (PS +X*PH)*@   + (2*PH + U*X)*@
             X        PS                PH                U
           2                          2
S6 := 2*T *@   + 2*X*T*@   - ((X   + 2*T)/2)*PS*@     -
            T           X                        PS
          2
      ((X *PH + 6*T*PH +2*X*PS)*@     -
                                 PH
          2
      ((X *U + 10*U*T + 2*PS + 4*X*PH)/2)*@
                                           U

S7 := F*@    + F*@   + @ F*@             %@ F - @   F = 0
         PS       PH    X   U            % T     X X
```

Most of the above symmetries would have been expected: time translation, space translation, phi scale change, t-x scale change, and Galilean transformation. S6 has no obvious meaning, whereas S7 expresses the linearity of the problem.

These equations and vectors can, of course, also be found using a different formulation. In REDUCE there is the package SPDE, realizing Lie's method for point symmetries [15].

Once isovectors are determined, one can look for so-called similarity solutions. These are solutions which are transformed by the isovector into itself. The solutions are generated by the original differential system augmented by the contractions of the isovector with all the forms in the differential system. One observation is essential: the augmented differential system is again closed and has the isovector as a *characteristic vector*.

As a final example we consider *generalized* symmetries for the Kepler problem. The symmetry given below generates the famous *Runge - Lenz vector*.

```
pform t=0,x=0,y=0,z=0,x1=0,y1=0,z1=0,x2=0,y2=0,z2=0,r=0;

fdomain r=r(x,y,z);
let @(r,x)=x/r,@(r,y)=y/r,@(r,z)=z/r;

eqx2 := d x1 + g*x/r**3*d t$
eqy2 := d y1 + g*y/r**3*d t$
eqz2 := d z1 + g*z/r**3*d t$

eqx := d x - x1*d t$
eqy := d y - y1*d t$
eqz := d z - z1*d t$

tvector v;
v := vx*@ x + vy*@ y + vz*@ z + vx1*@ x1 + vy1*@ y1 + vz1*@ z1$
```

```
pform vt=0,vx=0,vy=0,vz=0,vx1=0,vy1=0,vz1=0;

fdomain vx=vx(x,y,z,x1,y1,z1),
        vy=vy(x,y,z,x1,y1,z1),vz=vz(x,y,z,x1,y1,z1),
        vx1=vx1(x,y,z,x1,y1,z1),vy1=vy1(x,y,z,x1,y1,z1),
        vz1=vz1(x,y,z,x1,y1,z1);

sx2 := v |_ eqx2$
sy2 := v |_ eqy2$
sz2 := v |_ eqz2$

sx  := v |_ eqx$
sy  := v |_ eqy$
sz  := v |_ eqz$

let x2=-g*x/r**3,y2=-g*y/r**3,z2=-g*z/r**3;

sx2 := sub(d x=x1*d t,d y=y1*d t,d z=z1*d t,
           d x1=x2*d t,d y1=y2*d t,d z1=z2*d t,
           sx2);
                     5              5              5
SX2 := (d T*(@ VX1*R *X1 + @ VX1*R *Y1 + @ VX1*R *Z1
              X              Y              Z

                         2                2
             - @  VX1*G*R *X - @  VX1*G*R *Y
               X1                Y1

                         2         2            2
             - @  VX1*G*R *Z + G*R *VX - 3*G*X *VX
               Z1

                                                5
             - 3*G*X*Y*VY - 3*G*X*Z*VZ))/R
```

```
sy2 := sub(d x=x1*d t,d y=y1*d t,d z=z1*d t,
           d x1=x2*d t,d y1=y2*d t,d z1=z2*d t,
           sy2)$
sz2 := sub(d x=x1*d t,d y=y1*d t,d z=z1*d t,
           d x1=x2*d t,d y1=y2*d t,d z1=z2*d t,
           sz2)$

sx := sub(d x=x1*d t,d y=y1*d t,d z=z1*d t,
          d x1=x2*d t,d y1=y2*d t,d z1=z2*d t,
          sx);


                  3            3            3
SX := (d T*(@ VX*R *X1 + @ VX*R *Y1 + @ VX*R *Z1
             X            Y            Z


                                                    3
             - @  VX*G*X - @  VX*G*Y - @  VX*G*Z - R *VX1))
                X1          Y1          Z1


        3
      /R

sy := sub(d x=x1*d t,d y=y1*d t,d z=z1*d t,
          d x1=x2*d t,d y1=y2*d t,d z1=z2*d t,
          sy)$
sz := sub(d x=x1*d t,d y=y1*d t,d z=z1*d t,
          d x1=x2*d t,d y1=y2*d t,d z1=z2*d t,
          sz)$

factor @; on rat;

solve(sx/d t,vx1);
```

```
                                    - @  VX*G*X
                                       X1
{VX1=@ VX*X1 + @ VX*Y1 + @ VX*Z1 + ---------------
      X         Y         Z                 3
                                           R


          - @  VX*G*Y     - @  VX*G*Z
             Y1              Z1
      + -------------- + --------------}
               3               3
              R               R

sx2 := sub(ws,sx2);

                    2
SX2 := @   VX*d T*X1  + 2*@   VX*d T*X1*Y1
        X X                X Y

                                 - 2*@    VX*d T*G*X*X1
                                       X X1
        + 2*@   VX*d T*X1*Z1 + -------------------------
             X Z                           3
                                          R

            - 2*@    VX*d T*G*Y*X1
                 X Y1
        + ------------------------
                      3
                     R

            - 2*@    VX*d T*G*Z*X1     - @ VX*d T*G*X
                 X Z1                     X
        + ------------------------ + -----------------
                      3                     3
                     R                     R
               2
+ @   VX*d T*Y1  + 2*@   VX*d T*Y1*Z1
   Y Y                Y Z

   - 2*@    VX*d T*G*X*Y1
       Y X1
+ -------------------------
              3
             R

   - 2*@    VX*d T*G*Y*Y1
       Y Y1
+ -------------------------
              3
             R

   - 2*@    VX*d T*G*Z*Y1       - @ VX*d T*G*Y
       Y Z1                          Y
+ ------------------------- + -----------------
              3                        3
             R                        R

                    - 2*@    VX*d T*G*X*Z1
              2           Z X1
+ @   VX*d T*Z1  + -------------------------
   Z Z                           3
                                R

   - 2*@    VX*d T*G*Y*Z1
       Z Y1
+ -------------------------
              3
             R
```

```
     - 2*@    VX*d T*G*Z*Z1       - @ VX*d T*G*Z
         Z Z1                          Z
+ ------------------------- + -----------------
               3                        3
              R                        R

                      2  2                        2
    @      VX*d T*G  *X       2*@     VX*d T*G  *X*Y
     X1 X1                       X1 Y1
+ -------------------- + -----------------------
              6                        6
             R                        R

                        2
   2*@      VX*d T*G  *X*Z
      X1 Z1
+ ----------------------- + (@   VX*d T*G
               6                X1
              R

          2           2                                5
 *( - R  *X1 + 3*X  *X1 + 3*X*Y*Y1 + 3*X*Z*Z1))/R

                      2  2                        2
    @      VX*d T*G  *Y       2*@     VX*d T*G  *Y*Z
     Y1 Y1                       Y1 Z1
+ -------------------- + ----------------------- +
              6                        6
             R                        R

(@   VX*d T*G
  Y1

        2                       2                      5
 *( - R  *Y1 + 3*X*Y*X1 + 3*Y  *Y1 + 3*Y*Z*Z1))/R
```

```
                    2  2
 @      VX*d T*G *Z
  Z1 Z1
+ ------------------- + (@  VX*d T*G
           6             Z1
          R

       2                              2      5
 *( - R *Z1 + 3*X*Z*X1 + 3*Y*Z*Y1 + 3*Z *Z1))/R

         2          2
  d T*G*(R *VX - 3*X *VX - 3*X*Y*VY - 3*X*Z*VZ)
+ ----------------------------------------------
                         5
                        R
```

```
solve(sy/d t,vy1)$
sy2 := sub(ws,sy2)$
solve(sz/d t,vz1)$
sz2 := sub(ws,sz2)$

%A solution which generates the Runge-Lenz vector:

let vx=y*y1 + z*z1,vy=y*x1-2*x*y1,vz=z*x1-2*x*z1,
    vx1=y*y2+z*z2+y1**2+z1**2,vy1=x2*y-2*x*y2-x1*y1,
    vz1=x2*z-2*x*z2-x1*z1;
v;

@ *(Y*Y1 + Z*Z1) + @ *( - 2*X*Y1 + Y*X1)
 X                  Y

 + @ *( - 2*X*Z1 + Z*X1)
    Z
```

```
           2      2     3  2    3  2
 @  *( - G*Y  - G*Z  + R *Y1  + R *Z1 )
  X1
+ ----------------------------------------
                     3
                    R

             3                        3
 @  *(G*X*Y - R *X1*Y1)     @  *(G*X*Z - R *X1*Z1)
  Y1                         Z1
+ ------------------------ + ------------------------
           3                            3
          R                            R

sx2;
              2         2         2         2         3
 - (3*d T*G*(R *Y*Y1 + R *Z*Z1 - X *Y*Y1 - X *Z*Z1 - Y *Y1

                  2         2      3      5
             - Y *Z*Z1 - Y*Z *Y1 - Z *Z1))/R

sy2;
                 2    2    2    2
 3*d T*G*Y*X1*(R  - X  - Y  - Z )
----------------------------------
                5
               R
sz2;
                 2    2    2    2
 3*d T*G*Z*X1*(R  - X  - Y  - Z )
----------------------------------
                5
               R
```

3.7 Conservation laws, potentials and pseudopotentials

A conserved current is a form whose exterior derivative vanishes on a solution manifold. In the non-form notation the corresponding formula has the appearence of a continuity equation.
If there are closed forms in the exterior system, then any isovector generates a conserved current, which can be seen from rewriting the definition of the Lie derivative

$$d(S\rfloor\omega) = S\lfloor\omega - S\rfloor d\omega$$

I.e. the exterior form $S\rfloor\omega$ constitutes a conserved current.
In our heat equation example, the exterior form $h2$ of the differential system is closed and the isovector S3 gives the expected conservation of heat.
We can of course look for closed forms of the exterior system by solving for coefficients of a combination of the forms in the exterior system. Let's illustrate this with the following example, the famous Korteweg- de Vries equation.

$$\partial_t u + \partial_{xxx} u + 12 * u * \partial_x u = 0$$

With the abbreviations $z = \partial_x u$ and $p = \partial_{xx} u$ we can write for the equivalent exterior ideal

```
      a1 := d u^d t - z*d x^d t;
      a2 := d z^d t - p*d x^d t;
      a3 := d x^d u + d p^d t + 12*u*z*d x^d t;
```

```
Exercise: Show with EXCALC that this ideal represents the
          Korteweg-de Vries equation and that the ideal is
          closed.
```

To construct closed 2-forms from the forms a1,a2,a3 we make the Ansatz

```
      beta := f1*a1 + f2*a2 + f3*a3;
```

where

```
   fdomain f1=f1(t,x,u,z,p),f2=f2(t,x,u,z,p),f3=f3(t,x,u,z,p);
```

Since β is assumed to be closed, its exterior derivative has to vanish, i.e.$d\beta = 0$ yields the equations for the unknown functions f1,f2,f3.

```
Exercise: Find solutions to d beta = 0.
```

To proceed we select the special solution

```
      beta := - 12*u*a1 - a3;
```

The corresponding 1-form is given by

```
      omega := u*d x - (p + 6*u**2)*d t;
```

which is a conserved current for the Korteweg-de Vries equation.
It is important to note that we can add any exact 1-form to omega without changing its property of beeing a conserved current

```
      omega := d w + omega;
```

If we now regard w as an additional coordinate and add to our ideal the above equation, w has to satisfy the following relation on the solution manifold:

```
   fdomain w=w(t,x);

   omega;
                                    2
   (@ W + U)*d X + (@ W - P - 6*U )*d T
     X               T
```

I.e. the new coordinate has to obey the equation

$$\partial_t w + \partial_{xxx} w - 6 * (\partial_x w)^2 = 0$$

W is called a potential, and the process of enlarging the number of variables is called prolongation.
Of course, having obtained a solution for w, a solution of the Korteweg- de Vries equation follows immediately. Thus we have a tool to obtain solutions for an equation by solving a related, hopefully simpler, equation.
The systematic search for prolongations to a given equation is quite cumbersome and can be done reasonably only with the aid of Computer Algebra.
A very important generalization of the above method can be obtained by

allowing the prolongation variable to be present in the f's from the very beginning. The resulting equations for the f's are then nonlinear pde's. The prolongation variables are called in this case pseudopotentials. It turns out that the whole theory of pseudo-potentials can be stated very elegantly in terms of Lie-algebra valued forms, but this leads us too far away.

References

[1] Burke, W.L. (1985), *Applied Differential Geometry*, (Cambridge Univ. Press, Cambridge)

[2] Chang, T.Y.P, van Hulzen, J.A., Wang, P.S., *Code Generation and Optimization for Finite Element Analysis*, Proc. EUROSAM, LNCS 174 (Springer Berlin)

[3] Edelen, D.G.B. (1985), *Applied Exterior Calculus*, (Wiley, New York)

[4] Edelen, D.G.B. (1980), *Programs for Computer Implementation of the Exterior Calculus*, Comp. & Maths. with Appl. Vol. 6, pp. 415-424

[5] Hearn, A.C. (1987), *REDUCE User's Manual*, Rand Publication CP78, The Rand Corporation, Santa Monica, CA 90406

[6] Hearn, A.C. (1985), *Optimal Evaluation of Algebraic Expressions*, Proc. AAECC-3 Grenoble, LNCS 229 (Springer Berlin)

[7] Gragert, P.K.H., Kersten, P.H.M. (1982), *Implementation of Differential Geometric Objects and Functions*, LNCS 144 (Springer Berlin)

[8] van Hulzen, J.A, (1983), *Code Optimization of Multivariate Polynomial Schemes*, Proc. EUROCAL, LNCS 162 (Springer Berlin)

[9] Gates, B.L. (1985), *GENTRAN- an automatic code generation facility for REDUCE*, ACM Sigsam Bulletin, vol. 19, no. 3

[10] Kopetsch, H., (1987), *A numerical Method for the Time-dependent Stefan Problem in Czochralski Crystal Growth*, Journal of Crystal Growth

[11] MacCallum, M.A.H. (1986), *Algebraic Computing in Relativity*, TAU 86 - 04, Queen Mary College, University of London, (see also other reviews cited in there)

[12] Schrüfer, E. (1987), *EXCALC User's Manual*, Rand Corp. (appended to these notes)

[13] Schrüfer, E., Hehl, F.W.H., McCrea, J.D. (1987), *Exterior Calculus on the Computer: The REDUCE-Package EXCALC Applied to General Relativity and to the Poincare Gauge Theory*, General Relativity and Gravitation, vol. 19, nc. 2

[14] Schutz, B. (1980), *Geometrical Methods of Mathematical Physics*, (Cambridge University Press, Cambridge)

[15] Schrwarz, F. (1987), *SPDE User's Manual*, Rand Corp.

[16] Trautmann, A. (1984), *Differential Geometry for Physicists*, (Bibliopolis, Naples)

[17] Wirth, M.C. (1981), *Automatic Generation of Finite Differences and Fourier Stability Analysis*, (Proc. of the 1981 ACM Symp. on Symbolic and Algeb. Computation)