DIFFERENTIAL EQUATIONS
AND
CONTROL PROCESSES
N 1, 2002
Electronic Journal,
reg. N P23275 at 07.03.97

http://www.neva.ru/journal
e-mail: diff@osipenko.stu.neva.ru

Computer software
for the investigation of differential equations,
dynamical systems, and control processes

TOOLS for SYMMETRY ANALYSIS of PDEs

A. G. MESHKOV

Oryol State University
95 Komsomolskaya str., 302015, Oryol, Russia
e-mail: meshkov@orel.ru

Abstract

The package of routines JET and help files for it are presented. There are html- and mws-files for help. The package contains more than 30 Maple routines for the symmetry analysis of evolution partial differential systems with two independent variables and related problems: computation of Lie-Bäcklund symmetries and conserved densities, computation of canonical densities and investigation of the complete integrability, computation of differential substitutions and Bäcklund transformations, computation of recursion operators and Noether operators; several computations in matrix Lie algebras, the check of the antisymmetry, symplectic and cosymplectic conditions of linear integrodifferential operators and some others. All routines and algorithms are described in the article. It can be used as the user's guide.

Contents

1 Introduction

Package JET is free. The unique restriction is to cite each use of the package. The author will be grateful everyone for any critical remarks and offers.

We wrote and tested our routines during 5 years approximately [1], [2]. The package is had written for investigation of partial differential systems with two independent variables. We called this package JET because the jet space language is used. The package includes more than thirty procedures for Maple 6[1]. The main our intent was to create the full collection of tools for investigations of the completely integrable evolution systems. But the package can be used for other purposes as well. We use the same notions and notations as in the known books [3], [4], [5], [6].

For the independent variables we used the fixed (global) names: t is the temporal variable and x is the spatial one. For the dependent variables one may use any names but they must be specified in the global list **vard**. For example, if you deal with the jet space $J^\infty(R, R^2)$ with the local coordinates $x,\ u_i,\ v_i$, then you must assign **vard:=[u,v]**. Then all input and output expressions will depend on $t,\ x,\ u0,\ v0,\ u1,\ v1$ and so on.

[1]Maple V.5 also can read and compile these routines.

Let us mention that all procedures work in the **interactive mode** as a rule. The **automatic mode** is undesirable in the real problems in view of the following reasons.

1. Equations for the higher conserved densities or Lie-Bäcklund higher symmetries of nonlinear **systems** is reduced to very cumbersome partial differential systems. Solutions of that huge systems with dozens thousands terms in the automatic mode would require a huge RAM of a computer and very long time for computations. Solution of such systems is an art but not the mechanical process.

2. Probably any software contains the bags. Maple is one of the best pakages, but it contains the bags too. Here is only two examples:

```
>  int(x^n,x);
```

$$\frac{x^{n+1}}{n+1}$$

Here the function `int` ignores the branch $n = -1$.

```
>  z:=diff(f(x),x)-1/x*f(x)+2*f(x)^2;
```

$$z := \frac{\partial f(x)}{\partial x} - \frac{f(x)}{x} + 2\, f(x)^2$$

```
>  dsolve(z,{f(x)});
```

$$f(x) = \frac{x}{x^2 + \_C1}$$

So, the solution $f = 0$ is lost. Of course, everybody see that the limit $\_C1 \to \infty$ is possible here. But using the automatic mode you can not know which equations were solved and how much solutions were lost.

2 Differentiation and Integration

Our procedures for differentiation and integration are called **dif** and **INT**. There are the built-in procedures **diff** and **int** for differentiation and integration in Maple. Nevertheless we wrote our own procedures in order to make all expressions more compact. Everybody who computed Lie-Bäcklund symmetries or conserved densities knows that you are forced to deal with a lot of arbitrary functions. Equations arising in such problems are often very long. The procedure **depend** gives you possibility to hide all arguments of all functions. It creates two sets of substitutions: **var**={f1=f1(x1, x2), … , fn=fn(y1, … ,yk)} and **rav**={f1(x1, x2)=f1, … , fn(y1, … ,yk)=fn}. The names **var** and **rav** are global. Substitutions **var** are performed before the differentiation or integration of an expression. Substitutions **rav** are performed after the differentiation or integration of an expression to make the output shorter. **depend** protects the names of its arguments f1, … , fn and returns the set **NamesOfdeps**={f1, … , fn} that is used in other routines.

ATTENTION: Nothing will work in JET until **vard** and **depend** are entered.

When we enter the command dif(f,x) the substitutions **var** are performed and the built-in procedure **diff** is called. Therefore the procedures `dif` possesses the same facilities as the built-in `diff`. The following example shows the difference between the built-in and our procedures:

```
> vard:=[u, v]: depend(f(u0,v0,u1,v1)):
> a:=dif(f,v0)*dif(f,u0$3,v0),
> b:=diff(f(u0,v0,u1,v1),v0)*diff(f(u0,v0,u1,v1),u0$3,v0);
```

$$a := \frac{\partial f}{\partial v0}\,\frac{\partial^4 f}{\partial u0^3 \partial v0},\quad b := \frac{\partial f(u0,v0,u1,v1)}{\partial v0}\,\frac{\partial^4 f(u0,v0,u1,v1)}{\partial u0^3 \partial v0}$$

The first expression is a 3-4 times shorter then the second one. It is very important if you deal with a long expression. The derivatives of arbitrary functions are returned in the **inert** form: `a:=Diff(f,v0)*Diff(f,v0,‘$‘(u0,3))`, for instance. If you wish to assign `f:=u0^2+v0^2` for example, you must enter `depend(...)` without f beforehand to unprotect the name f:

```
> depend(g(x) ):f:=u0^2+v0^2:
> a;
```

$$\left(\frac{\partial}{\partial v0}(u0^2+v0^2)\right)\left(\frac{\partial^4}{\partial u0^3 \partial v0}(u0^2+v0^2)\right)$$

To compute this expression the substitution `Diff=dif` is necessary:

```
> a:=expand(eval(subs(Diff=dif,a)));
```

$$a := 0$$

The procedure `dif` admits the functional assignment and it is very useful property.

```
> dif(f,x):=f^3+f-1;
> a:=dif(x^2*f,x);
```

$$\mathrm{dif}(f,x) := f^3+f-1$$
$$a := 2\,xf + x^2\frac{\partial f}{\partial x}$$

```
> a:=expand(eval(subs(Diff=dif,a)));
```

$$a := 2\,xf + x^2f^3 + x^2f - x^2$$

The procedure `INT` possesses not only the same facilities as the built-in `int` but many others for operating with arbitrary functions using the integration by parts. Examples:

```
> f:='f': depend(f(x,y,z)):
> InT(dif(f,y)*dif(f,x$3,y),x)=INT(dif(f,y)*dif(f,x$3,y),x);
```

$$\int \frac{\partial f}{\partial y}\,\frac{\partial^4 f}{\partial x^3 \partial y}\,dx = \frac{\partial f}{\partial y}\,\frac{\partial^3 f}{\partial x^2 \partial y} - \frac{1}{2}\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2$$

```
> InT(x^2*dif(f,x$3,y),x)=INT(x^2*dif(f,x$3,y), x);
```

$$\int x^2\,\frac{\partial^4 f}{\partial x^3 \partial y}\,dx = x^2\,\frac{\partial^3 f}{\partial x^2 \partial y} - 2\,x\,\frac{\partial^2 f}{\partial x \partial y} + 2\,\frac{\partial f}{\partial y}$$

```
> InT(x^2*dif(f,x),x)=INT(x^2*dif(f,x), x);
```

$$\int x^2\,\frac{\partial f}{\partial x}\,dx = x^2\,f - 2\,x\int f\,dx$$

The built-in procedure `int` returns integrals with arbitrary functions without computation. In these examples `InT` is the inert version of the integration procedure. We wrote the new inert version because the built-in `Int` possesses one undesirable property:

```
> expand(Int(f,x));
```

$$f\int 1\,dx$$

The next procedure **DF** computes the total derivative with respect to x on a jet space and **DN** computes the n-fold total derivative. The syntax is `DF(F)` and `DN(F,n)` where `F` is an expression, `n` is an integer or a name:

```
> vard:=[u]: depend(f(x,u0,u1), g(u0) ):
> DF(f), DF(g), DN(g,2);
```

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial u0}\,u1 + \frac{\partial f}{\partial u1}\,u2,\quad \frac{\partial g}{\partial u0}\,u1,\quad \frac{\partial g}{\partial u0}\,u2 + \frac{\partial^2 g}{\partial u0^2}\,u1^2$$

DN(g,n) is computed as DN(DF(g), n-1) and DN(F,0)=F, DN(F,1)=DF(F); DN(F,-1) is the inverse to DF operator. If the control variable **flag** is zero then the minimal computations are performed (in the previous example `flag` was not assigned):

```
> flag:=0:DF(2*f-k*g), DN(g^2-c*f,2),DF(u3),DN(u2,3),DN(u3,-2);
```

$$2\,DF(f) - DF(k\,g), DN(g^2,2) - DN(c\,f,2),\ u4,\ u5,\ u1$$

To compute DF(u3) or DN(u2,3) the routines DF and DN create the global set **_Var** $= \{u_0, u_1, \ldots, u_n\}$, where the number n=`maxord` may be entered by user. If `maxord` is not entered then it is set 50 by default.To simplify the expressions with parameters one can enter the global set **const**:

```
> const:={c,k}: DF(2*f-k*g), DN(g^2-c*f,2);
```

$$2\,DF(f) - k\,DF(g),\ DN(g^2,2) - c\,DN(f,2)$$

There are three more control variables **fldf**, **fldn** and **flpt**. If one of them is assigned 1 when `flag=0` then `DF` and `DN` acquire the new facilities:

```
> fldf:=1: DF(2*f*g),DF(g^2*f);
```

$$2\,DF(f)\,g+2\,f\,DF(g),\ 2\,g\,DF(g)\,f+g^2\,DF(f)$$

```
> fldn:=1: DN(f*g,3); DN(g^2*f,2); DN(u3*u0,-1),DN(u4*u0,-1);
```

$$f\,DN(g,3)+3\,DF(f)\,DN(g,2)+3\,DN(f,2)\,DF(g)+DN(f,3)\,g$$

$$g^2\,DN(f,2)+4\,g\,DF(g)\,DF(f)+(2\,DF(g)^2+2\,g\,DN(g,2))\,f$$

$$u0\,u2-1/2\,u1^2,\ u0\,u3-u2\,u1+DN(u2^2,-1)$$

```
> fldn:=0:flpt:=1:DN(DN(f,3)*g,-1);DN(DN(f,5)*g,-1);
```

$$DN(f,2)\,g-DN(DN(f,2)\,DF(g),-1)$$

$$g\,DN(f,4)-DF(g)\,DN(f,3)+DN(DN(f,3)\,DN(g,2),-1)$$

The **simultaneous usage `fldn=1` and `flpt=1` is undesirable** in large expressions because of too large number of levels of the recursion. One can apply the call DN(F,-1) when flag=1 too:

```
> fldn:='fldn':flpt:='flpt':flag:=1:vard:=[u]:depend(g(u0)):DN(DN(g,3),-1);
```

$$\frac{\partial^2 g}{\partial u0^2}\,u1^2+\frac{\partial g}{\partial u0}\,u2$$

Argument of dif, INT, DF, DN and the next routine ED may be any algebraic expression, *array, Vector set, equation* or *list.* The procedure **ED** computes the evolution derivative

$$ED(F)\to D_t(F)=\frac{\partial F}{\partial t}+\sum_{i,\alpha}\frac{\partial F}{\partial u_i^\alpha}D^iK^\alpha,$$

where D is the total derivative with respect to x and K^α are the right hand sides of an evolution system

$$u_t^\alpha=K^\alpha(u).$$

One has to enter the vector field K beforehand as the list **sys** (`sys` is the global name). For example, if you deal with the system $u_t=F,\ v_t=G$ you must enter the following commands:

```
> vard:=[u, v]:  depend(F(u0,v0,...), G(u0,v0,...) ):  sys:=[F, G]:
```

If `flag:=0` then the minimal simplifications are executed:

```
> flag:=0:  ED(f1=c*f2), ED(DN(g,k)), ED(DF(g));
```

$$ED(f1)=c\,ED(f2),\ DN(ED(g),k),\ DF(ED(g))$$

Here c is the element of the set `const`. When `flag:=1` or `flag:='flag'` all expressions `DF(f)`, `DN(f,n)`, and `ED(f)` will be computed completely. There is one more control variables **fled**. If `fled=1` then ED(f) will be computed completely, but expanding DF(f) and DN(f,k) depend on their control variables:

```
> vard:=[u]: depend(F(u0,u1),G(u0,u1),f(u0,u1),g(u0,u1)):
> flag:=0:fldf:='fldf':fldn:='fldn':flpt:='flpt':fled:=1:
> sys:=[F, G]: ED(f);
```

$$\frac{\partial f}{\partial u0}F+\frac{\partial f}{\partial u1}DF(F)$$

Arguments of DF, DN and ED may depend on nonlocal variables $w_i = D_x^{-1}\rho_i$, where ρ_i are conserved densities and θ_i are the conserved fluxes of an evolution system under consideration: $D_t\rho_i = D_x\theta_i$. Such variables w_i are called the weakly nonlocal one, they were heuristically introduced in [7] and more punctually in [8]. The nonlocal variables w_i, densities ρ_i and fluxes θ_i must be specified in the global lists **nlo, densities** and **fluxes** respectively. For example, the KdV equation $u_t = u_{xxx} + 6\,u\,u_x$ admits the following conserved densities u and u^2:

$$D_t(u_0) = D(u_2 + 3\,u_0^2),\quad D_t(u_0^2) = D(2\,u_0\,u_2 - u_1^2 + 4\,u_0^3).$$

Therefore the new variables $w1 = D^{-1}u_0$ and $w2 = D^{-1}u_0^2$ may be introduced:

```
> nlo:=[w1,w2]: densities:=[u0,u0^2]:
> fluxes:=[u2+3*u0^2,2*u0*u2-u1^2+4*u0^3]:
> depend(f(x,u0,u1,w1,w2) ): vard:=[u]: sys:=[u3+6*u0*u1]:
```

Now DF, DN and ED are prolonged on $w1$ and $w2$:

```
> flag:=1:DF(f);ED(f);
```

$$\frac{\partial f}{\partial x}+\frac{\partial f}{\partial u0}\,u1+\frac{\partial f}{\partial u1}\,u2+\frac{\partial f}{\partial w1}\,u0+\frac{\partial f}{\partial w2}\,u0^2$$

$$\frac{\partial f}{\partial u0}\,(u3+6\,u0\,u1)+\frac{\partial f}{\partial u1}\,(u4+6\,u1^2+6\,u0\,u2)+\frac{\partial f}{\partial w1}\,(u2+3\,u0^2)$$
$$+\frac{\partial f}{\partial w2}\,(2\,u0\,u2-u1^2+4\,u0^3)$$

3 Lie-Bäcklund Symmetries

Lie-Bäcklund symmetries for the evolution system $u_t = K(u)$ satisfy the following determining equation (see [3], [4] or [5] for instance):

$$(D_t - K')\,F = 0, \tag{1}$$

where the prime means the Frechet derivative

$$(K')^\alpha_\beta = \frac{\partial K^\alpha}{\partial u_i^\beta}D^i. \tag{2}$$

For example, for the KdV equation $u_t = u_{xxx} + 6\,u\,u_x$ the determining equation (1) takes the following form

$$(D_t - 6\,u_1 - 6\,u_0\,D - D^3)\,F = 0.$$

The procedure **LBsymm** computes the left hand side of the equation (1) and splits it with respect to higher order variables. The syntax is `LBsymm(F)` or `LBsymm(F,dialog)`, where F is a list (a symmetry) and `dialog` is the keyword. The routine performs a preliminary splitting of the equations (1) and calls the package **diffalg** for additional splitting. But sometimes diffalg works very long time and the keyword `dialog` was introduced to prevent the call of diffalg. The output is a set or a list of sets of split equations. If a list arise then you have the branching. The initial (unsplit) equations are stored in the global set **zero**. One can use it to check whether the solution is true or not. The check is necessary because the split system may be incomplete.

For example, in order to calculate the first order Lie-Bäcklund symmetries for the nonlinear Schrödinger like system you must enter the following commands:

```
> vard:=[u,v]:
> depend(F1(t,x,u0,u1,v0,v1), F2(t,x,u0,u1,v0,v1),
  f(u0,v0), g(u0,v0) ):
> sys:=[u2+f, -v2+g]: a1:=LBsymm([F1,F2]);
```

$$a1 := \left\{ \frac{\partial^2 F2}{\partial v1 \partial x} = 0,\ \frac{\partial^2 F1}{\partial u1 \partial x} = 0,\ \frac{\partial^2 F1}{\partial u1^2} = 0,\ \frac{\partial F1}{\partial v0} = 0,\ \frac{\partial F1}{\partial v1} = 0,\right.$$
$$\left. \frac{\partial^2 F2}{\partial v0 \partial v1} = 0,\ \frac{\partial^2 F2}{\partial v1^2} = 0,\ \frac{\partial F2}{\partial u0} = 0,\ \frac{\partial F2}{\partial u1} = 0,\ \frac{\partial^2 F1}{\partial u0 \partial u1} = 0 \right\}$$

Solving this system we find

$$F1 = u1\, h1(t) + g1(t,x,u0), \quad F2 = v1\, h2(t) + g2(t,x,v0)$$

and call `LBsymm` again.

```
> depend(h1(t),h2(t),g1(t,x,u0),g2(t,x,v0), f(u0,v0),g(u0,v0) ):
> a1:=LBsymm([u1*h1+g1,v1*h2+g2]);
```

Common subsystem

$$\left\{ \frac{\partial h2}{\partial t} = -2\,\frac{\partial^2 g2}{\partial x \partial v0},\ \frac{\partial^2 g2}{\partial v0^2} = 0,\ \frac{\partial^3 g2}{\partial x^2 \partial v0} = 0,\ \frac{\partial^2 g1}{\partial u0^2} = 0 \right\}$$

for the branches:

$$a1 := \left[\left\{ \frac{\partial h1}{\partial t} = 2\,\frac{\partial^2 g1}{\partial u0 \partial x},\ \frac{\partial^2 g2}{\partial v0^2} = 0,\ \frac{\partial^2 g1}{\partial u0^2} = 0,\ \frac{\partial f}{\partial v0} = 0,\right.\right.$$
$$\left. \frac{\partial g}{\partial u0} = 0,\ \frac{\partial^3 g1}{\partial x \partial u0 \partial x} = 0,\ \frac{\partial h2}{\partial t} = -2\,\frac{\partial^2 g2}{\partial v0 \partial x},\ \frac{\partial^3 g2}{\partial x \partial v0 \partial x} = 0 \right\},$$
$$\left\{ h1 = h2,\ \frac{\partial^2 g2}{\partial v0^2} = 0,\ \frac{\partial^2 g1}{\partial u0^2} = 0,\ \frac{\partial h2}{\partial t} = -2\,\frac{\partial^2 g2}{\partial v0 \partial x},\right.$$
$$\left.\left. \frac{\partial^2 g1}{\partial u0 \partial x} = -\frac{\partial^2 g2}{\partial v0 \partial x},\ \frac{\partial^3 g2}{\partial x \partial v0 \partial x} = 0 \right\} \right]$$

These two cases are obtained from the following system:

```
> zero=0;
```

$$\left\{v1\,\frac{\partial h2}{\partial t}+\frac{\partial g2}{\partial t}+\frac{\partial g2}{\partial v0}g+h2\,\frac{\partial g}{\partial u0}u1-\frac{\partial g}{\partial u0}u1\,h1-\frac{\partial g}{\partial u0}g1-\frac{\partial g}{\partial v0}g2+\right.$$
$$+\frac{\partial^2 g2}{\partial x^2}+2\,\frac{\partial^2 g2}{\partial v0\partial x}v1+\frac{\partial^2 g2}{\partial v0^2}v1^2,\;\; u1\,\frac{\partial h1}{\partial t}+\frac{\partial g1}{\partial t}+\frac{\partial g1}{\partial u0}f+h1\,\frac{\partial f}{\partial v0}v1-$$
$$\left.-\frac{\partial f}{\partial u0}g1-\frac{\partial^2 g1}{\partial x^2}-2\,\frac{\partial^2 g1}{\partial u0\partial x}u1-\frac{\partial^2 g1}{\partial u0^2}u1^2-\frac{\partial f}{\partial v0}v1\,h2-\frac{\partial f}{\partial v0}g2\right\}=0$$

The name `zero` is global. When the object `zero` is large the automatic splitting of it requires a long time. In such situations one can use the dialogue mode. To do this call the routine with additional argument **dialog**:

```
> vard:=[u,v]:
> depend(F1(t,x,u0,u1,v0,v1,u2,v2,u3,v3,u4,v4,u5,v5),
   F2(t,x,u0,u1,v0,v1,u2,v2,u3,v3,u4,v4,u5,v5),f(u0,v0),g(u0,v0) ):
> sys:=[u2+f, -v2+g]:
> a:=LBsymm([F1,F2],dialog):
> a1:=a[1]:
> nops(a1), ord(a1);
```

$$248,\quad [7, 6]$$

Here **ord** is the routine for computing of the order of an expression. It begin the search of the variables u_i in an expression from the highest order **maxord** to zero. The name `maxord` is global, if it is not entered by user, then the routines `ord, DF` or `DN` stand `maxord=50` by default.The previous output means that the expression `a1` consists of 248 terms and contains u7,u6, ... ,v6,v5, ... , but does not contain u8, u9, ... , v7,v8, ... In general case the call `ord(a)` returns a list [m, n, ...], with partial orders of the expression `a`. That is, m is the order with respect to variable `vard[1]`, n is the order with respect to variable `vard[2]`, etc. More detail information about any expression you can obtain with the help of the built-in procedure **indets**. Let us remember that the obtained expression `a1` is a polynomial with respect to the highest order variables u7, u6, and v6 (but others are contained in F), therefore the built-in procedure **degree** is useful as well. To extract the terms with u7 one can use the procedure **chn** (**ch**oose **n**ame):

```
> b1:=chn(a1,u7);
```

$$b1 := 2\,u7\,\frac{\partial F2}{\partial u5}$$

The syntax of `chn` is `chn(expr, a,b,...,z)` where `expr` is any expression, `a,b,...,z` are any names or powers of names or functions. When type of `expr` is not '+' then `chn` returns `expr` if all the objects `a,b,...,z` are contained in the `expr`, else `chn` returns zero. If type of `expr` is '+' then `chn` returns the sum of those terms what contain all the objects. The initial expression `expr` is not changed by `chn`.

It is easy to see that $F2$ does not depend on u5 in our example. And the terms with u6 give rise that $F2$ does not depend on u4. And moreover it follows from `zero[2]` that $F1$ does not depend on v5 and v4. To continue the calculation you must enter the following commands:

```
> depend(F1(t,x,u0,u1,v0,v1,u2,v2,u3,v3,u4,u5),
   F2(t,x,u0,u1,v0,v1,u2,v2,u3,v3,v4,v5),f(u0,v0),g(u0,v0)):
> a1:=expand(eval(subs(Diff=dif,a1))): nops(a1), ord(a1);
```

$$165, \ \ [5,\ 6]$$

The substitution `Diff=dif` is necessary for recalculation of all derivatives because the command `dif(F,x)` returns the result in the inert form `Diff(F,x)` if F is an indefinite function as it was mentioned above.

One can also use in such calculations the procedure `cho` (**ch**oose **o**rder) instead of `chn`:

```
> depend(f(t,x,u0,u1,u2) ):
> A:=2*u1+f+u3+u4:
> chn(A,u4), cho(A,4);
```

$$u4, \ \ u4$$

But

```
> chn(A,u3), cho(A,3);
```

$$u3, \ \ u3 + u4$$

The call `cho(A, n)` collects and returns those terms from the expression `A` whose orders $\geqslant n$. The both `cho` and `chn` can find the implicit variables as well:

```
> chn(A,u2), cho(A,2);
```

$$f, \ \ f + u3 + u4$$

The routine `cho` works faster than `chn`.

4 Conserved Densities and Covariants

The vector function $(\rho,\ \theta)$ on the jet space is called the conserved current for a system $u_t = K(u)$ if it solves the equation

$$D_t\,\rho = D\,\theta, \tag{3}$$

where D_t is the evolution derivative along the trajectories of the system and D is the total derivative with respect to x. The function ρ is said to be the conserved density and θ is said to be the flux. The current $(D\,f,\ D_t\,f)$ is the conserved one for any system and it is called the trivial conserved current. If one add a trivial current to any other conserved current then the sum will be the conserved current too. The transformation $(\rho,\ \theta) \longrightarrow (\rho + D\,f,\ \theta + D_t\,f)$ is called the equivalence transformation.

One can investigate equation (3) with the help of the Euler operator E

$$E_\alpha = \sum_{n=0}^{\infty}(-D)^n \frac{\partial}{\partial u_n^\alpha},\ \alpha = 1, \ldots, m, \tag{4}$$

where m is number of dependent variables (m=nops(vard)). Operator E possesses an important property: $E\,f = 0$ if and only if $f = D(F)$ [10]. Applying the operator E to the equation (3) we obtain the following equation for the conserved densities

$$E\,(D_t\,\rho) = 0. \tag{5}$$

JET-package contains the procedure **EU** that computes the variational derivatives according to the formula (4):

$$EU(F, k) \longrightarrow E_k\,F.$$

So, to obtain the left hand side of the equation (5) you must call `EU(ED(rho),k)`, where $k = 1, 2, \ldots, m$, and m is the number of the dependent variables. These equations can be solved by splitting in higher order variables. But if the order of the density ρ is large ($\geqslant 3$) the object $E\,(D_t\,\rho)$ may be too large for your computer.

Another way for solving the equation (3) is to apply the procedure **pot** (potential). The command `pot(ED(rho))` integrates the equation (3) and computes the flux θ. Let us denote $\varphi = D_t\,\rho$ and suppose that the order of φ is $n+1$, then the order of θ is n and the equation under consideration takes the following form

$$D\,\theta \equiv \frac{\partial\theta}{\partial u_n^\alpha}\,u_{n+1}^\alpha + \frac{\partial\theta}{\partial u_{n-1}^\alpha}\,u_n^\alpha + \cdots = \varphi. \tag{6}$$

Here and below the summation rule over the repeated indices is implied. If the function φ is nonlinear with respect to u_{n+1}^α the equation (6) is impossible. Hence one must check vanishing of all second derivatives

$$\frac{\partial^2\varphi}{\partial u_{n+1}^\alpha \partial u_{n+1}^\beta} = 0. \tag{7}$$

If this is true and φ is linear, $\varphi = f_\alpha(u_n)\,u_{n+1}^\alpha + g(u_n)$, then

$$\frac{\partial\theta}{\partial u_n^\alpha} = f_\alpha(u_n) \tag{8}$$

and the compatibility conditions

$$\frac{\partial f_\alpha}{\partial u_n^\beta} = \frac{\partial f_\beta}{\partial u_n^\alpha} \tag{9}$$

must be satisfied. Setting

$$F_1 = \int f_1\,du_n^1, \quad \theta = \theta_1 + F_1,$$

we obtain from (8)

$$\frac{\partial\theta_1}{\partial u_n^1} = 0, \quad \frac{\partial\theta_1}{\partial u_n^\alpha} = \tilde{f}_\alpha \equiv f_\alpha - \frac{\partial F_1}{\partial u_n^\alpha}, \;\; \alpha > 1. \tag{10}$$

Now the compatibility conditions for (10) take the form

$$\frac{\partial\tilde{f}_\alpha}{\partial u_n^1} = 0, \quad \frac{\partial\tilde{f}_\alpha}{\partial u_n^\beta} = \frac{\partial\tilde{f}_\beta}{\partial u_n^\alpha}, \;\; \alpha, \beta > 1. \tag{11}$$

We can represent this process in the following form

$$\begin{gathered}\theta \to \theta_1 = \theta - F_1, \quad \varphi \to \varphi_1 = \varphi - DF_1, \\ D\theta_1 = \varphi_1 = f_\alpha\, u^\alpha_{n+1} + g - D\,F_1 = \tilde{f}_2\, u^2_{n+1} + \cdots + \tilde{f}_m\, u^m_{n+1} + \tilde{g},\end{gathered} \tag{12}$$

where the functions

$$\tilde{f}_\alpha = f_\alpha - \frac{\partial F}{\partial u^\alpha_n}, \quad \tilde{g} = g - D\,F + \frac{\partial F}{\partial u^\alpha_n}\, u^\alpha_{n+1}$$

do not depend on the variables u^α_{n+1}. If the first of the equations (11) is satisfied we can prolong the chain (12)

$$\begin{gathered}\theta_1 \to \theta_2 = \theta_1 - F_2, \quad \varphi_1 \to \varphi_2 = \varphi_1 - DF_2, \\ D\theta_2 = \varphi_2 = \tilde{f}_2\, u^2_{n+1} + \cdots + \tilde{f}_m\, u^m_{n+1} + \tilde{g} - D\,F_2 = \hat{f}_3\, u^3_{n+1} + \cdots + \hat{f}_m\, u^m_{n+1} + \hat{g},\end{gathered}$$

where

$$F_2 = \int \tilde{f}_2\, du^2_n,$$

and so on. After m steps we obtain $D\theta_m = \varphi_m,$ $\mathrm{ord}\varphi_m \leqslant n$ and the process is repeated.

When the command `pot(phi)` is entered the procedure **pot** checks all conditions (7), (9), (11), ... and performs the integrations and all equivalence transformations

$$th := 0: \;\; th := th + F: \;\; \varphi := \varphi - DF(F):$$

If some condition is broken the process is stopped, the reminder $\varphi - D\,F_1 - D\,F_2 - \cdots - D\,F_k$ is saved under the global name **rm**, the message "Break, rm contains, expr" is printed and the result $F_1 + F_2 + \cdots + F_k$ is returned. The expression "exp" may be ${u^1_3}^2$ or $u^1_2 \& u^2_3$ and all that. The sense of such messages is that the reminder `rm` is nonlinear function with respect to u^1_3 or u^1_2 and u^2_3 (it may be arbitrary function but not the second power) and some compatibility condition is not satisfied. If rm$\neq 0$ the output G of the command `pot(phi)` means that $\varphi = D(G) + rm$ and rm is not the total derivative. Notice that the call DN(phi,–1) returns the same result in another form G+DN(rm,–1). When all compatibility conditions are satisfied we obtain an output and the message "Finish, rm=0".

Let us consider, for example, the conserved densities for the KdV equation:

```
> vard:=[u]: depend( ): sys:=[u3 + 6*u0*u1]:
> pot(ED(u0^2));
```

Finish, rm=0

$$2\,u0\,u2 - u1^2 + 4\,u0^3$$

And more example

```
> th:=pot(ED(u0^3)):
```

Break, rm contains, $u1^2$

```
> th:=th; rm:=rm;
```

$$th := 3\,u0^2\,u2 - 3\,u0\,u1^2 + \frac{9}{2}u0^4$$

$$rm := 3\,u1^3$$

Hence $D_t\,u0^3 = D\,th + 3\,u1^3$ and $u0^3$ is not a conserved density.

Notice that there is more short version of `pot` with the name `pt`. It works without any text messages. This quality is necessary to call the routine from another one.

The function `pot` may be useful for computing of nonlocal conserved densities too. Let us consider the example.

```
> vard:=[u]: sys:=[u3+6*u0*u1]:
> nlo:=[w1,w2]: densities:=[u0,u0^2]:
> fluxes:=[u2+3*u0^2,2*u0*u2-u1^2+4*u0^3]:
> depend(f(x,u0,u1,w1,w2) ):
> a1:=DF(f);
```

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial u0}u1 + \frac{\partial f}{\partial u1}u2 + \frac{\partial f}{\partial w1}u0 + \frac{\partial f}{\partial w2}u0^2$$

```
> pot(a1);
```

$$Finish, rm = 0$$

$$f$$

The result will be just so for any arbitrary function and for some concrete functions:

```
> a2:=DF(u1^2+u0*u1*w1+w2^2*u2);
```

$$a2 := u1^2 w1 + 2\,u2\,u1 + u2\,u0\,w1 + w2^2 u3 + u0^2 u1 + 2\,w2\,u2\,u0^2$$

```
> pot(a2};
```

$$Finish, rm = 0$$

$$u1^2 + u0\,u1\,w1 + w2^2\,u2$$

But often it is not so

```
> a3:=DF(u1^2+u0*u1*w1+w2^2);
```

$$a3 := u1^2 w1 + 2\,u2\,u1 + u2\,u0\,w1 + u0^2 u1 + 2\,w2\,u0^2$$

```
> pot(a3}; rm;
```

$$See\ rm,\ ord(rm) =,\ [0]$$

$$u1^2 + u0\,u1\,w1$$

$$2\,w2\,u0^2$$

There is another way to construct the conserved densities. Let ρ be a conserved density of the evolution system (1). Then the covector field $\gamma = E\rho$, where E is the Euler operator, satisfies the following equation [11]

$$(D_t + K'^{+})\gamma = 0, \tag{13}$$

where K'^{+} is the adjoint to K' operator.

If a covector field γ is differentiable in a starlike neighbourhood of the point $u = 0 \in J^{\infty}(\mathbb{R}, \mathbb{R}^m)$ and

$$\gamma' = \gamma'^{+} \tag{14}$$

in this neighbourhood then $\gamma = E\tilde{\rho}$ [9], [10], where

$$\tilde{\rho} = \int_0^1 \gamma_\alpha(\xi u)\, u^\alpha \, d\xi. \tag{15}$$

Hence if a solution of (13) satisfies to the condition (14) then the formula (15) gives the conserved density of the evolution system.

Solutions of the equation (13) are called the conserved covariants. They applied not only for computation the conserved densities but for constructing the recursion and the inverse Noether operators. There is the routine **covariant** for solving the equation (13). The syntax is the same as for `LBsymm`: `covariant(G)` or `covariant(G,dialog)`, where `G` is a list of expressions and `dialog` is the keyword. The routine performs the preliminary splitting of equations (13) and call the package diffalg for additional splitting. Sometimes diffalg works very long time and to prevent the call of diffalg the keyword `dialog` is introduced. The initial (unsplit) equations are stored in the global set **zero**. One can use it to check whether the solution is true or not. The check is necessary because the split system may be incomplete.

Let us consider the Schrödinger system for example.

```
> vard:=[u,v]: sys:=[u2+u0^2*v0,-v2-v0^2*u0]:
> depend(f(u0,v0,u1,v1), g(u0,v0,u1,v1) ):
> a:=covariant([f,g]);
```

$$a := \left\{ \frac{\partial^2 g}{\partial u0 \partial u1} = 0,\ \frac{\partial^2 g}{\partial u1^2} = 0,\ \frac{\partial g}{\partial v0} = 0,\ \frac{\partial g}{\partial v1} = 0, \right.$$
$$\left. \frac{\partial^2 f}{\partial v0 \partial v1} = 0,\ \frac{\partial^2 f}{\partial v1^2} = 0,\ \frac{\partial f}{\partial u0} = 0,\ \frac{\partial f}{\partial u1} = 0 \right\}$$

Solving these equations call the routine again:

```
> depend(f1(v0), g1(u0) ):
> a:=covariant([f1+c1*v1,g1+c2*u1]);
```

$$a := \left\{ \frac{\partial^2 f1}{\partial v0^2} = 0,\ c1 = -c2,\ \frac{\partial^2 g1}{\partial u0^2} = 0 \right\}$$

```
> a:=covariant([c1*v1+c3*v0+c4,-c1*u1+c5*u0+c6]);
```

$$a := \{-2\,c5\,u0 + 2\,u0\,c3 - 2\,c6,\ -u0\,(-u0\,c3 + c5\,u0 + 2\,c6),\\ -v0\,(-c3\,v0 + v0\,c5 - 2\,c4),\ -2\,c5\,u0\,v0 + 2\,u0\,c3\,v0 + 2\,u0\,c4 - 2\,v0\,c6,\\ -2\,v0\,c5 + 2\,c3\,v0 + 2\,c4,\ -2\,c5 + 2\,c3\}$$

We have $c5 = c3,\ c4 = c6 = 0$ obviously:

```
> covariant([c1*v1+c3*v0,-c1*u1+c3*u0]),  zero;
```

$$\{\,\},\ \{\,\}$$

So, we obtain two covariants for the Schrödinger system

```
> gamma1=<v1,-u1>, gamma2=<v0,u0>;
```

$$\gamma_1 = \begin{bmatrix} v1 \\ -u1 \end{bmatrix}, \gamma_2 = \begin{bmatrix} v0 \\ u0 \end{bmatrix} \tag{16}$$

and two conserved densities

$$\rho_1 = \frac{1}{2}\,(u_0\,v_1 - u_1\,v_0), \quad \rho_2 = u_0\,v_0$$

according to the formula (15).

5 Canonical conserved densities

In the articles [12] were introduced the canonical conserved densities for completely integrable systems and the necessary conditions of the complete integrability of an evolution system was formulated. Then this approach was developed in [13], [6] and many others articles. In cited papers and books they applied a powerful but difficult for computations operator technique for obtaining the canonical densities. Another easy way for obtaining the *same* canonical densities was presented in [14]. Later this ("Chinese") technique was explained and generalized on a wide class systems with two independent variables in [15]. Below we follow this article. Let the system

$$F(u) = 0 \tag{17}$$

can be transformed to the Cauchy-Kowalewski normal form with the help of transformation of independent variables. Let us denote $\Phi(D_t,\, D_x) = F'$, where D_t and D_x are the total differentiation operators. Then let us consider the following system

$$\Phi(D_t + \theta,\, D_x + \rho)\psi\Big|_{F=0} = 0, \quad (c, \psi) = 1, \tag{18}$$

where (c, ψ) is the Euclidean scalar product and c is a constant vector. The main result is

If the system (17) is integrable by the inverse spectral transform method then the system (18) possesses a formal solution in the following form

$$\rho = \sum_{i=-n}^{\infty} \rho_i \, k^i, \quad \theta = \sum_{i=-m}^{\infty} \theta_i \, k^i, \quad \psi = \sum_{i=0}^{\infty} a_i \, k^i, \tag{19}$$

where k is a parameter, $n > 0$, $\rho_{-n} \neq 0$ or $m > 0$, $\theta_{-m} \neq 0$ and (ρ_i, θ_i), $i \geqslant \min(-m, -n)$ are local or weakly nonlocal conserved currents of the system (17). The conserved densities ρ_i obtained from the equations (18) coincide with the canonical densities constructed by means of the original algorithm developed in [6], [12]. But the "Chinese" technique is simpler from the computational viewpoint because it deals with the commutative series (19) but not with the operator series. There is a wide class of the evolution systems for which $n = 1$ in the expansions (19). We call these systems as *regular*. There is an example of the *irregular* (but nonintegrable) system when $n > 1$ [15].

Let us consider the example

$$u_t = u_3 + f(u, u_1). \tag{20}$$

A simple calculation gives $F' = D^3 + f_1 D + f_0 - D_t$, where $f_k = \partial f / \partial u_k$. Hence the equation (18) takes the following form $[-D_t - \theta + (D+\rho)^3 + f_0 - (D+\rho)\, f_1] \cdot 1 = 0$, or

$$\rho^3 - \theta + (D^2 + f_1)\, \rho + \frac{3}{2}\, D\rho^2 + f_0 = 0.$$

Setting

$$\rho = k^{-1} + \sum_{i=0}^{\infty} \rho_i \, k^i, \quad \theta = k^{-3} + \sum_{i=0}^{\infty} \theta_i \, k^i,$$

we obtain the required recursion formula

$$\begin{aligned} \rho_{i+2} = \frac{1}{3}\, \theta_i - \sum_{j=0}^{i+1} \rho_j \, \rho_{i-j+1} - \frac{1}{3} \sum_{j,k=0}^{i} \rho_j \, \rho_k \, \rho_{i-j-k} \\ - \frac{1}{3}\, (D^2 + f_1)\rho_i - D\left(\rho_{i+1} + \frac{1}{2} \sum_{j=0}^{i} \rho_j \, \rho_{i-j}\right) - \frac{1}{3}\, f_0 \, \delta_{i0}, \end{aligned} \tag{21}$$

where $i \geqslant 0$, $\rho_0 = 0$, $\theta_0 = 0$, $\rho_1 = -\frac{1}{3}\, f_1$ and δ_{ij} is the Kronecker δ-symbol. The recursion formula (21) can be easily coded in Maple:

```
> r:=proc(n)
  local i;
    i:=n-2; if n <= 0 then RETURN(0) fi;
    if n = 1 then RETURN(-1/3*dif(f,u1)) fi;
      cat(th,i)/3-SU(r,r,0,i+1)-1/3*SU(r,r,r,0,i)
      - 'DF'(r(i+1)) -1/2*'DF'(SU(r,r,0,i))-'DN'(r(i),2)/3
      - dif(f,u0)*DLT(i,0)/3-dif(f,u1)*r(i)/3
  end;
```

Here **DLT** is the auxiliary procedure for the Kronecker δ-symbol and **SU** is the procedure for the multiple sums. For example, the call `SU(A,B,C,n,m)` returns the sum of the monomials `A(i)*B(j)*C(k)` where $i, j, k \geqslant n$ and besides $i + j + k = m$. Number of arguments of `SU` may be any and arguments of `SU` may be under `DF` or `DN` operators. That is the expressions of the type `SU(A,DF(B),DN(C,p),n,m)` are admissible. Moreover we assume here that the fluxes $\theta_0,\ \theta_1, \ldots$ will be saved under the names `th0,th1, ...` .

It was proved in the papers [6], [12] that there is another series of conserved densities for any integrable evolution system. We call them *adjoint* densities for brevity because they are obtained from the adjoint to (18) equation. Let us denote the adjoint densities as $\tilde{\rho}_k$, then $\tilde{\rho}_i - \rho_i \in \operatorname{Im} D$ as it was proved in the cited works. This condition is stronger than $D_t \rho_i \in \operatorname{Im} D$ obviously. That is why the both ρ_i and $\tilde{\rho}_i$ are applied in the symmetry analysis of integrable systems.

Let us consider the quasi-linear evolution system with m equations

$$u_t = A(u)\, u_n + F(u_0, \ldots, u_{n-1}), \quad \partial A/\partial u_n = 0.$$

If all eigenvalues of the main matrix A are different then one can obtain explicit recursion formulas for m sequences of ρ_i and m sequences of $\tilde{\rho}_i$ choosing different normalization conditions $(c, \psi) = 1$: $\psi = (1, \psi_2, \ldots, \psi_m)$ or $\psi = (\psi_1, 1, \psi_3, \ldots, \psi_m)$ and so on. In the case of multiple eigenvalues explicit recursion formula is impossible. And instead of it some differential equations arise for ρ_i and a_i [15].

The package JET contains two routines that compute the canonical and adjoint canonical densities for Nth order **regular** evolution systems, $N > 1$. They are **cd** and **acd** respectively. The both `cd` and `acd` use the following algorithm. The routine substitutes the finite series (19) into system (18) (or its adjoint), collect the obtained polynomials with respect to powers of k and extract coefficients of the polynomials. These coefficients must be zeroes and they are stored under the global name **EQS**. Then the routines, solve the obtained systems with respect to ρ_i and a_i (or $\tilde{\rho}_i$ and $\tilde{a}_i$) and return results. If it is impossible to solve the system for ρ_i, a_i (because of branches for example) then the routines suggest you to enter some values or to choose a branch. It is important remember in this moment that the scale transformation $k \to k\,\lambda$ is admitted in (19). Therefore when $\rho_{-1} = const$ one may choose $\rho_{-1} = 1$ or -1 without loss of generality.

The call `cd(n,m)` returns m canonical densities from nth sequence as a rule. But for some systems the routines may ask you to choose the constant `C` and the function $\rho_{[-1]} = r$ and repeat the call with the additional argument `cd(n,m,[C,r])`. One may also use several substitutions (such as $a_{[n,m]} = value$ for example) with the help of another optional argument: `cd(n,m,[C,r],S)` or `cd(n,m,S,[C,r])` or `cd(n,m,S)`, where S is a **set** of substitutions. The both routines assign `flag:=0` in order to obtain the shortest expressions for the densities and save the auxiliary functions a[i,k] (or b[i,k]) in the global set **SOL**. The syntax for `acd` is the same.

Let us consider examples. The single equation:

```
> vard:=[u]: depend(f(u0,u1,u2)): sys:=[u3 + f];
```

$$sys := [u3 + f]$$

```
> cd(1,2);
```

$$\left[\rho_0 = -\frac{1}{3}\frac{\partial f}{\partial u2},\ \rho_1 = \frac{1}{9}\left(\frac{\partial f}{\partial u2}\right)^2 + \frac{1}{3}DF\left(\frac{\partial f}{\partial u2}\right) - \frac{1}{3}\frac{\partial f}{\partial u1}\right]$$

The Schrödinger like system:

```
> vard:=[u,v]:  depend(f(u0,u1,v0,v1),g(u0,u1,v0,v1)):
> sys:=[u2+f, -v2-g];
```

$$sys := [u2 + f, -v2 - g]$$

```
> cd(2,2);
```

$$\left[\rho_{[2,0]} = -\frac{1}{2}\frac{\partial g}{\partial v1},\ \rho_{[2,1]} = \frac{1}{8}\left(\frac{\partial g}{\partial v1}\right)^2 - \frac{1}{2}th_{[2,0]} + \frac{1}{4}\frac{\partial f}{\partial v1}\frac{\partial g}{\partial u1} - \frac{1}{2}\frac{\partial g}{\partial v0} + \frac{1}{4}DF\left(\frac{\partial g}{\partial v1}\right)\right]$$

This output is differ from the previous. For a single differential equation we have a single sequence of canonical densities $\rho_n, n = 0, 1, \ldots$ and for the m-component system we have m sequences $\rho_{[1,j]}, \rho_{[2,j]}, \ldots, \rho_{[m,j]}$.

The nonlinear diffusion system as a case of the multiple roots:

```
> vard:=[u,v]: > depend(f(u0,u1,v0,v1), g(u0,u1,v0,v1)):
> sys:=[u2+f, v2-g];
```

$$sys := [u2 + f, v2 - g]$$

```
> cd(1,1);
```

$$ED(rho[i,k] = DF(th[i,k])$$

$$SOL = \left\{a2_{1,0} = \ \mathrm{RootOf}\left(2\,DF(\_Z) - \_Z\,\frac{\partial f}{\partial u1} - \frac{\partial f}{\partial v1}\,\_Z^2 - \frac{\partial g}{\partial v1}\,\_Z - \frac{\partial g}{\partial u1}\right)\right\}$$

$$\mathit{Be\ careful,\ multiple\ roots}$$

$$\left[\rho_{1,0} = -\frac{1}{2}\frac{\partial f}{\partial u1} - \frac{1}{2}\frac{\partial f}{\partial v1}\mathrm{RootOf}\left(2\,DF(\_Z) - \_Z\,\frac{\partial f}{\partial u1} - \frac{\partial f}{\partial v1}\_Z^2 - \_Z\,\frac{\partial g}{\partial v1} - \frac{\partial g}{\partial u1}\right)\right]$$

So, $\rho_{[1,0]}$ and $a2_{[1,0]}$ are nonlocal functions in general case.

For the systems with two and more equations the canonical densities may consist of the dozen hundred terms. The evolution derivative of such long expression consists of dozen thousand terms. Processing a large expression requires very long time. And moreover if the number of addends in an expression is more then 40000 then Maple V.4 or Maple V.5 finish computation and inform: "Object too large". In Maple 6 no restriction on the size of the object probably but the time of an computation exponentially depend on the size of the object. To solve this problem we apply the procedure **entry**. The command `z:=entry(F,N)` returns the list `z` so that each entry of `z` contains N addends from `F`; number of addends in the last entry of `z` will be less or equal to N. After obtaining the list `z` one can perform the required operations with each element `z[i]` separately and obtain the final result. This approach requires less time and memory than the direct computation. Another method is based on using the procedure **cho**. As the terms with the greatest order are most interesting then the procedure `cho` is very useful.

6 Differential Substitutions and Bäcklund transformations

Let us consider a pair of an evolution differential systems

$$u_t = K(u) \tag{22}$$

and

$$v_t = H(v), \tag{23}$$

where $u : \mathbb{R}^2 \longrightarrow \mathbb{R}^m,\ v : \mathbb{R}^2 \longrightarrow \mathbb{R}^m$ and $K,\ H$ are some smooth vector differential operators. Let we also have a smooth vector differential operator $f : \mathbb{C}^\infty(\mathbb{R}^2,\, \mathbb{R}^m) \longrightarrow \mathbb{C}^\infty(\mathbb{R}^2,\, \mathbb{R}^m)$. If for any solution v of the system (23) the function u defined by the formula

$$u = f(v) \tag{24}$$

satisfies the system (22) then the equation (24) is called the differential substitution. The order of the operator f is called the order of the differential substitution.

The first order differential substitutions are considered as a rule, the well-known example is the Miura substitution. But the higher order substitutions exist as well [16].

Let the operator K be known. To compute the substitution (24) one has to differentiate the equation (24) with respect to t

$$K(u) - \frac{\partial f}{\partial v_n} D^n H(v) = 0 \tag{25}$$

and substitute $u = f(v),\ u_1 = Df(v),\ u_2 = D^2 f(v), \ldots$ into this equation. As a result we obtain a system containing only variables v_i. After spiting this system with respect to the higher order variables v_i we obtain sufficiently many differential equations to find f and H.

Bäcklund transformations can be considered as the implicit differential substitution. The first order Bäcklund transformation between systems (22) and (23) takes the following general form $F(u, u_1, v, v_1) = 0$. If $\partial F/\partial u_1 = 0$ we have the simple differential substitution (24). And if $\partial F/\partial u_1 \neq 0$ we can solve the equation $F = 0$ with respect to u_1 and write the Bäcklund transformation in the following form

$$u_1 = f(u, v, v_1). \tag{26}$$

To find the explicit form of the function f one must to differentiate the equation (26) with respect to t. This gives

$$DK(u) - \frac{\partial f}{\partial u} K(u) - \frac{\partial f}{\partial v_n^\alpha} D^n H(v) = 0. \tag{27}$$

Substituting into this equation $u_1 = f(u, v, v_1),\ u_2 = Df(u, v, v_1), \ldots$, we obtain a system containing variables v_i and u. This system is much more cumbersome than that obtained from (25) but algorithm is the same. Therefore the both (25) and (27) can be obtained with the help of one and the same procedure **difsub**. The syntax is `difsub({u=f})` or `difsub({u=f},dialog)` for a differential substitution and `difsub({u1=f})` or `difsub({u1=f},dialog)` for a Bäcklund transformation. Here `f` is a vector function on the jet space, `dialog` is the keyword. If `difsub` is called with one argument then the equation (25) or (27) is preliminary split by means of the differentiation and then the package **diffalg** is called. But this require very long time especially for computation the Bäcklund transformations. When `difsub` is called with two argument then the preliminary split system is returned. In the both these cases the original system is stored in the global set **zero**. For the scalar equation (22) the first argument may be equation: `difsub(u=f,dialog)`.

Let us consider the Bäcklund transformation for the nonlinear Schrödinger system.

```
> vard:=[u,v,U,V]:
> depend(f1(u0,v0,U0,V0,U1),f2(u0,v0,U0,V0,V1) }:
> sys:=[u2+u0^2*v0,-v2-u0*v0^2,U2+U0^2*V0,-V2-U0*V0^2]:
> a:=difsub({u1=f1,v1=f2},dialog): ord(%);
```

$$[0, 0, 2, 2]$$

If one do not use *dialog* then the routine works a very long time.

```
> dif(a,U2); dif(a,V2);
```

$$\left\{0, -2\,\frac{\partial^2 f2}{\partial U1 \partial V1}, 2\,\frac{\partial^2 f1}{\partial U1 \partial V1}, -2\,\frac{\partial^2 f2}{\partial U1^2}, 2\,\frac{\partial^2 f1}{\partial U1^2}\right\},$$

$$\left\{0, 2\,\frac{\partial^2 f1}{\partial V1^2}, -2\,\frac{\partial^2 f2}{\partial U1 \partial V1}, 2\,\frac{\partial^2 f1}{\partial U1 \partial V1}, -2\,\frac{\partial^2 f2}{\partial V1^2}\right\}$$

So, the functions $f1$ and $f2$ are linear with respect to $U1$, $V1$. To abridge the example let us follow to [17] and set

$$f1 = U1 + (u0 + U0)\,h + k\,(u0 - U0), \quad f2 = V1 + (v0 + V0)\,h - k\,(v0 - V0),$$

where the function h depend on $(u0 - U0)\,(v0 - V0)$ only and k is a constant.

```
> depend(h(u0,v0,U0,V0)):
> dif(h,U0):=-dif(h,u0): dif(h,V0):=-dif(h,v0):
> a:=difsub({u1=U1+(u0+U0)*h+k*(u0-U0),
   v1=V1+(v0+V0)*h-k*(v0-V0)},dialog):
> a:=factor(eval(subs(Diff=dif,a))):ord(%);
```

$$[0, 0, 0, 0]$$

```
> a[2], a[5];
```

$$(u0 + U0)\left(u0 + 4\,h\frac{\partial h}{\partial v0} - U0\right), -(v0 + V0)\left(v0 + 4\,h\frac{\partial h}{\partial u0} - V0\right)$$

These equations give rise the solution $h = \sqrt{c - \frac{1}{2}\,(u0 - U0)\,(v0 - V0)}$ where c is a constant. If one substitutes this result into the set *zero* then $\{0\}$ is returned.

The next example will be a first order differential substitution for the mKdV equation $u_t = u_3 + 2u_0^2\,u_1$. Let another equation be unknown $v_t = F(v_0, v_1, v_2, v_3)$:

```
> vard:=[u,v]: depend(f(v0,v1),F(v0,v1,v2,v3) ):
> sys:=[u3+2*u0^2*u1, F];
```

$$sys := [u3 + 2\,u0^2\,u1, F]$$

```
> a:=difsub(u0=f);
```

There is a branching

$$a := \left[\left\{\frac{\partial f}{\partial v1} = 0\right\},\ \left\{\frac{\partial F}{\partial v3} = 1\right\}\right]$$

As $\partial f/\partial v1 \neq 0$ then $F = v3 + G(v0, v1, v2)$ and we enter the new command.

```
> depend(f(v0,v1),G(v0,v1,v2) ): sys:=[u3+2*u0^2*u1, v3+G]:
> a:=difsub(u0=f, dialog);
```

$$\left\{3\,\frac{\partial^2 f}{\partial v0 \partial v1}v1 + 3\,\frac{\partial^2 f}{\partial v1^2}v2 - \frac{\partial f}{\partial v1}\,\frac{\partial G}{\partial v2}\right\}$$

If one do not use *dialog* here then the time of computation is around one second but too much branches occur. Integrating the obtained equation one can define more precisely the input parameter and repeat the call, etc.

There is another type of substitutions. When the zero order conserved density ρ exists for an evolution system one can perform the following nonlocal contact transformation [18], [19] $(t,\, x,\, u(t,\, x)) \rightarrow (\tau,\, y,\, U(\tau,\, y))$

$$\tau = t, \quad d\,y = \rho\,d\,x + \theta\,d\,t, \quad U(\tau,\, y) = u(t,\, x), \tag{28}$$

where θ is the flux corresponding to the density ρ. This means that

$$\frac{\partial y}{\partial x} = \rho, \quad \frac{\partial y}{\partial t} = \theta, \quad D_x u = \rho D_y U, \quad D_t u = (D_\tau + \theta D_y)U$$

This transformation is analogous to the transformation between Lagrange and Euler variables in the fluid dynamics. Therefore the procedure executing the transformation (28) was called as `L_E`. The syntax is `L_E(rho,theta,VARD)`, where `rho`$=\rho$ is a conserved density, `theta`$=\theta$ (optional) is the flux corresponding to the density ρ and `VARD` is a list of new dependent variables. If vard=[u, v], then you may choose VARD=[U, V] for instance. Let us transform the KdV equation taking $\rho = u0$, for example:

```
> L_E(u0,[U]);
```

$$[U\_\tau = U0^3\,U3 + 3\,U0^2\,U1\,U2 + 3\,U0^2\,U1]$$

If one call the procedure `L_E` with the three parameters then it works slightly faster because the θ is entered but is not computed.

7 Zero curvature representations

Let us consider the following linear matrix system

$$\Psi_x = U\,\Psi, \quad \Psi_t = V\,\Psi, \tag{29}$$

where Ψ is a column, U and V are the square matrices depending on the jet space coordinates t, x, u^α_n and a parameter λ. The system (29) is compatible if and only if the following equation is valid

$$U_t - V_x + [\,U,\,V\,] = 0. \tag{30}$$

If the equation (30) is satisfied on the solutions manifold of an evolution partial differential system

$$u_t = K(u), \tag{31}$$

but not identically then they say the system (31) possesses the zero curvature representation.

The systems (29) and (30) are covariant under the gauge transformation:

$$\Psi \to \tilde\Psi = S\Psi,\; U \to \tilde U = S\,U\,S^{-1} + S_x S^{-1},\; V \to \tilde V = S\,V\,S^{-1} + S_t S^{-1}.$$

This transformation may be used for simplification the matrices U and V.

There are several functions in the package JET for manipulating with commutators. The first of them is **com** (commutator). Arguments of `com` may be both symbolic matrices (names) and arrays. If F and G are linear forms of some symbolic matrices then the call `com(F,G)` returns the expanded expression, but the symbolic matrices must be specified in the global set **matrices**. If F and G are arrays then the explicit value of the commutator $[F, G]$ will be returned as array. Procedure `com` knows all properties of commutators. For example,

```
> matrices:={A,B,C,E,U,V}:
> a:=com(c1*A+2*B, c2*A+3*B+U);
```

$$a := 3\,c1\,'[A,B]' + c1\,'[A,U]' - 2\,c2\,'[A,B]' + 2\,'[B,U]'$$

The strange primes appeared here because of the undesirable property of lists in the recent versions of Maple (V.5 or 6):

```
> 2*[A,B];
```

$$[2\,A,\,2\,B]$$

Therefore we were forced introduce the additional primes in the auxiliary procedure `‘print/com‘`:

```
> ‘print/com‘:=proc(a,b) ’’[a,b]’’ end:
```

But you can remove this procedure from the package if you wish. We omit all primes below for brevity. Let us continue our examples.

```
> vard:=[u]:depend(U(u0), V(u0)):
> com(U, 2*U+3*V}, com(V,U), dif(com(U,V),u0);
```

$$3\,[U,V],\quad -[U,V],\quad \left[\frac{\partial U}{\partial u0},V\right]+\left[U,\frac{\partial V}{\partial u0}\right]$$

`com` orders its arguments in the alphabetical order. Integration of `com(A,B)` is possible only if `A` and `B` are constants, but it is sufficient for analysis of the equations (30). Let us consider now operations with the arrays.

```
> a:=com(A,B)+C; A:=array(1..2,1..2,[[1,0],[0,-1]]);
  B:=array(1..2,1..2,[[c1,c2],[c3,c4]]);
```

$$a := [A,B]+C$$

$$A := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$B := \begin{bmatrix} c1 & c2 \\ c3 & c4 \end{bmatrix}$$

```
> a;
```

$$\begin{bmatrix} 0 & 2\,c2 \\ -2\,c3 & 0 \end{bmatrix} + C$$

```
> evalm(%);
```

$$\begin{bmatrix} C & 2\,c2 \\ -2\,c3 & C \end{bmatrix}$$

That's why all arrays must be substituted simultaneously:

```
> C:=array(1..2,1..2,[[a1,0],[0,a2]]): evalm(a);
```

$$\begin{bmatrix} a1 & 2\,c2 \\ -2\,c3 & a2 \end{bmatrix}$$

The better way is to create the set of substitutions

```
> A:='A': B:='B': C:='C':
> s:={A=array(1..2,1..2,[[1,0],[0,-1]]),
   B:=array(1..2,1..2,[[c1,c2],[c3,c4]]),
   C:=array(1..2,1..2,[[a1,0],[0,a2]])}:evalm(eval(subs(s,a)));
```

$$\begin{bmatrix} a1 & 2\,c2 \\ -2\,c3 & a2 \end{bmatrix}$$

It is possible to abbreviate the last command:

```
> evsub(s,a);
```

$$\begin{bmatrix} a1 & 2\,c2 \\ -2\,c3 & a2 \end{bmatrix}$$

The procedure of JET `evsub` have the following syntax `evsub(expr)` or `evsub(s,expr)`, where `expr` is an algebraic expression or square matrix, `s` is an equation or a set of equations. The first calling sequence is applied for computation the expression or matrix `expr`. The second calling sequence is applied for two purposes. The first purpose is the substitution of the matrices into the expression `expr` as in our example and the second one is to substitute some parameters into the matrix `expr`.

Let (31) be the KdV equation. Let us consider how one can solve the equation (30).

```
> vard:=[u]:depend(U(u0), V(u0,u1,u2) ):
> matrices:={U,V}: sys:=[u3+6*u0*u1]:
> z:=ED(U) - DF(V) + com(U, V);
```

$$z := \frac{\partial U}{\partial u0}\,(u3 + 6\,u0\,u1) - \frac{\partial V}{\partial u0}\,u1 - \frac{\partial V}{\partial u1}\,u2 - \frac{\partial V}{\partial u2}\,u3 + [U, V]$$

Here all the flags are unassigned or flag=1. Splitting this equation with respect to u_3 we obtain

```
> dif(z,u3), F1+ INT(dif(z,u3),u2);
```

$$\frac{\partial U}{\partial u0} - \frac{\partial V}{\partial u2},\; F1 + \frac{\partial U}{\partial u0}u2 - V$$

```
> depend(U(u0), V(u0,u1,u2), F1(u0,u1) ):
> matrices:={U,V,F1}: s:={V= dif(U,u0)*u2+F1};
  z:=expand(eval(subs(s,Diff=dif,z)));
```

$$s := \left\{V = \frac{\partial U}{\partial u0}\,u2 + F1\right\}$$

$$z := 6\,\frac{\partial U}{\partial u0}u0\,u1 - u1\,\frac{\partial^2 U}{\partial u0^2}u2 - u1\,\frac{\partial F1}{\partial u0} - \frac{\partial F1}{\partial u1}\,u2 - u2\,\left[\frac{\partial U}{\partial u0}, U\right] - [F1, U]$$

Finding now $F1$ as we find V

$$F_1 = F_2 - \frac{1}{2}\,u_1^2\,\frac{\partial^2 U}{\partial u0^2} - u_1\left[\frac{\partial U}{\partial u_0}, U\right],$$

substitute it into the expression z:

```
> depend(U(u0), V(u0,u1,u2), F1(u0,u1), F2(u0) ):
> matrices:={U,V,F1,F2}:
> s:={V= dif(U,u0)*u2+F1,
   F1=F2-1/2*u1^2*dif(U,u0$2)-com(dif(U,u0),U)*u1}:
> z:=expand(eval(subs(s,Diff=dif,z)));
```

$$z := 6\,\frac{\partial U}{\partial u0}u0\,u1 - u1\,\frac{\partial F2}{\partial u0} + \frac{1}{2}\,u1^3\,\frac{\partial^3 U}{\partial u0^3} + \frac{3}{2}\,u1^2\left[\frac{\partial^2 U}{\partial u0^2}, U\right]$$
$$-[F2, U] + u1\left[\left[\frac{\partial U}{\partial u0}, U\right], U\right]$$

It is obvious from this equation that U is the second degree polynomial of u_0. But it is known that U is the first degree polynomial of u_0. Substituting $U = A_1 u_0 + A_2$ into z one can find

$$U = A_1\,u_0 + A_2,\ V\ = A_1\,u_2 + A_3 + 3\,A_1\,u_0^2 - \tfrac{1}{2}\,u_0^2\,[A_1, [A_1, A_2]]$$
$$-[A_2, [A_1, A_2]]\,u_0 - [A_1, A_2]\,u_1$$

where the constant matrices A_i satisfy the following equations

$$\begin{aligned}&2\,[[A2, [A1, A2]], A1] - 6\,[A1, A2] - [A2, [A1, [A1, A2]]] = 0,\\&[A1, [A1, [A1, A2]]] = 0, [A2, A3] = 0,\\&[A1, A3] + [[A2, [A1, A2]], A2] = 0.\end{aligned} \qquad (32)$$

To simplify such equations one may denote some of the commutators as new matrices ($[A_1, A_2] = A_4$, for example) and use the Jacoby identity. There is the procedure **Jac** that transforms the nested commutators according to the Jacobi identity:

```
> matrices:={U,V,A,B,C,E}:
> z1:=com(A, com(B, E)) + com(C, com(A, B));
```

$$z1 := [A, [B, E]] + [C, [A, B]]$$

```
> Jac(z1,A,B,C);
```

$$[A, [B, E]] - [A, [B, C]] + [B, [A, C]]$$

```
> Jac(z1,A,B,B);
```

$$[C, [A, B]] + [B, [A, E]] - [E, [A, B]]$$

`Jac` searches the first nested commutator containing A, B and C in z1, transforms it and return the result, but the arguments may coincide. In the first example A, B and C are contained in the second term and it was transformed. In the second example A and B are contained in the first term that's why the different results are obtained. The procedure `Jac` has two optional arguments:

```
   Jac(expr,A,B,C,N),  Jac(expr,A,B,C,nested),

   Jac(expr,A,B,C,nested,N)=Jac(expr,A,B,C,N,nested),
```

where `expr` is an algebraic expression, A, B, C are matrices, N is a number of the position in `expr` from which the search is begun, `nested` is the keyword. If the 5th or 6th argument is `nested` then `Jac` transforms the internal commutator. Here is one more example

```
> z2:=com(A,com(B,C)) + com(E,com(C,com(A,B)));
```

$$z2 := [A,[B,C]] + [E,[C,[A,B]]]$$

```
> Jac(z2,A,B,C,2); Jac(z1,A,B,C,nested,2);
```

$$[A,[B,C]] + [C,[E,[A,B]]] + [[A,B],[C,E]]$$
$$[A,[B,C]] - [E,[A,[B,C]]] + [E,[B,[A,C]]]$$

The 5th argument 2 in the first case make `Jac` begin the search from the second addend. If one calls `Jac(z1,A,B,C,nested)` then `Jac` try to transform the internal commutator [B, C] in the first addend and the information on the error will be returned. The call `Jac(z1,A,B,C,2,nested)` gives the same result as `Jac(z1,A,B,C,nested,2)`.

When the equation (30) is solved the next problem is to construct Lie algebra. Let us consider for example the equations (32), where $[A_1, A_2] = A_4$:

$$\begin{aligned} &[A_1,[A_2,A_4]] + 2\,A_4 = 0, \quad [A_1,A_2] = A_4, \\ &[A_1,[A_1,A_4]] = 0, \quad [A_2,[A_2,A_4]] = [A_1,A_3], \quad [A_2,A_3] = 0. \end{aligned} \tag{33}$$

There are different ways to solve this system. For example, one can choose one of the matrices in the Jordan normal form and try to solve the equations directly. But this way is difficult for large matrix dimensions and the better way is to investigate the equations (33) in the spirit of the ideas by H.D. Wahlquist and F.B. Estabrook [21]. They suggested to introduce the new matrices and reduce the problem to the Lie algebra. Introducing in our example matrices

$$A_5 = [A_2,\ A_4], \quad A_6 = [A_1,\ A_4], \quad A_7 = [A_1,\ A_3]$$

we obtain 7 matrices and 8 commutator relations. This is unclosed Lie algebra as 7-dimensional algebra has 21 independent commutator relations. The main idea is: if a commutator $[A_i,\ A_j]$ is unknown then

or $[A_i,\ A_j]$ is a linear combination of known elements of the Lie algebra,

or $[A_i,\ A_j]$ is a new element linear independent with the previous elements.

So, for (33) the following suggestions are possible:

or $A_1, \ldots, A_7$ are linear dependent,

or $A_1, \ldots, A_7$ are linear independent and form the basis,

or $A_1, \ldots, A_7$ are linear independent but do not form the basis.

Investigating these possibilities step by step with the help of the Jacobi identity one can close the algebra. To investigate the second step when we know the basis the following algorithm is useful. To obtain the closed table of commutators

$$[A_i,\, A_j] = C^k_{ij}\, A_k \tag{34}$$

let us define some of commutators $[A_i,\, A_j] = x^1_{ij}\, A_1 + \cdots + x^n_{ij}\, A_n$, where x^k_{ij} are the indefinite coefficients. Then we can construct the adjoint representation $A_i \to C_i$. It is known that the matrices C_i satisfy the same relations (34). Therefore substituting $A_i = C_i$ into the commutator relations (34) we can obtain the complete set of equations for x^k_{ij} and can find the structural constants C^i_{jk}. This algorithm is realized in the procedure **struct**. The syntax is: `struct(bas, s, x)`, where `bas` is the list of the basis elements A_i of an algebra, `s` is the set of equations (34) and `x` is a free name. The procedure creates the global array **C**, global set **EQ** for the closed table (34) and the global set **S** for substitutions $bas_i = C_i$. If a system for the indefinite coefficients x_i is satisfied identically or if the algebra is closed then the empty set is returned. If the set of equations contains the explicit contradictions then the message `Contradiction, 1=0` is returned. In all other cases the system for x_i is returned.

Let us consider the following simple example:

```
> s:={com(A1,A2)=A4}: bas:=[A1,A2,A4]:
> z:=struct(bas,s,x);
```

Structural constants are given by array C[i][kj]=C^k_{ij}
*Table of commutators [e_i,e_n]=C^k_{in}*e_k is given by set EQ*
Substitutions bas[i]=C[i] are set S, and constraints are:

$$z := [[[-x_4x_3 + x_6x_1, x_1x_3 + x_6x_2, x_1^2 + x_2x_4, x_1 + x_5],\ plex(x_5, x_2, x_1, x_4, x_3, x_6), \{\}],$$
$$[[x_6, x_3, x_1 + x_5], plex(x_5, x_2, x_1, x_4, x_3, x_6), \{x_1^2 + x_2x_4\}]]$$

```
> EQ;
```

$$\{[A1, A4] = x_1A1 + x_2A2 + x_3A4,\ [A2, A4] = x_4A1 + x_5A2 + x_6A4,\ [A1, A2] = A4\}$$

The obtained list z contains two variants:

$$(a)\ x_1\, x_6 - x_3\, x_4 = 0,\ x_3\, x_1 + x_2\, x_6 = 0,\ x_1^2 + x_2x_4 = 0,\ x_5 = -x_1,$$

$$(b)\ x_5 = -x_1,\ x_3 = x_6 = 0,\ x_1^2 + x_2x_4 \neq 0.$$

Setting in (b) $x_1 = -x_5 = 2,\ x_2 = 0$ for example, we find the sl(2) algebra

$$[A_1, A_2] = A_4, \quad [A_1, A_4] = 2A_1, \quad [A_2, A_4] = x_4A_1 - 2A_2. \tag{35}$$

These equations and $A_3 = -x_4A_2$ solve the equations (33). Constructing then a representation of the obtained Lie algebra we can find an explicit form of the matrices U and V.

For solving the considered in this section problems the following two procedures **Cmetric** and **Killing** are useful. The call `Cmetric()` returns the Cartan metric tensor g_{ij} of a Lie algebra. And the call `Cmetric(y)` returns the quadratic form $g_{ij}\, y^i\, y^j$. The command `Killing(A,B)` returns the value of the Killing form $< A, B >$=trace(ad A ad B) of the pair elements A, B of a Lie algebra. In order to these routines `Cmetric` and `Killing` can work the program `struct` must be run beforehand.

8 Recursion operators

The recursion operator R for the evolution system $u_t = K(u)$ (31) satisfies the following equation

$$[\,D_t - K',\, R\,] = 0 \tag{36}$$

by definition (see [3]–[5]). There are three algorithms for constructing the recursion operators of the evolution systems at least. The first one is the direct solving of the equation (36). The second approach uses the equation for the gradient of the spectral parameter [22], it was late developed in [23] and independently in [24]. The third algorithm [25] is based on the perfectly other ideas. It is more powerful but it is not realized in routines now.

Let us remind the results of the gradient algorithm. If an evolution system (31) possesses the zero curvature representation (30) then the system (29) and its adjoint

$$h_x = -h\,U, \quad h_t = -h\,V \tag{37}$$

are compatible. In view of these equations the function

$$g_\alpha = \frac{\delta}{\delta u^\alpha} < h,\, U\psi >, \;\; \alpha = 1, \ldots, m, \tag{38}$$

where the angle brackets denote the Euclidean scalar product, satisfies the equation for conserved covariants

$$(D_t + K'^{+})\, g = 0.$$

Here + denotes the formal conjugation. If ord$U \leqslant 1$ then we have the following explicit formula

$$g_\alpha = \left\langle h, \left(\left[U, \frac{\partial U}{\partial u_1^\alpha} \right] + \frac{\delta U}{\delta u^\alpha} \right) \psi \right\rangle .$$

If U depend on higher order variables the analogous formulas exist. We adopt that the matrix U is embedded into some Lie algebra

$$U = \sum_{i=1}^{n} f_i(u, \lambda)\, A_i, \tag{39}$$

where

$$[A_i,\, A_j] = C_{ij}^k\, A_k, \quad i, j, k = 1, 2, \ldots, N \geqslant n. \tag{40}$$

and λ is a parameter. It is obvious from these formulas that the function (38) is a linear form on the Lie algebra

$$g_\alpha = \sum_{i=1}^{N} G_i(u) e_i, \quad e_i =< h,\, A_i\psi > . \tag{41}$$

Using (29), (37) and (39) one can easily to see that the functions e_i satisfy the following equations

$$De_i =< h, [A_i, U]\psi >= \sum_{j,k=1}^{N} C_{ij}^k\, f_j(u, \lambda) e_k. \tag{42}$$

So, the vector function (41) and all its derivatives may be expressed in the terms of e_i. And vice versa, differentiating g_α in the form (41) sufficiently many times one can express the functions e_i in the terms of derivatives $D^k g_\alpha$, $\alpha = 1, \ldots, m,\ k = 0, 1, \ldots, M < \infty$. Continuing the process of differentiating and substituting e_i into obtained equations one can find a linear differential system for the function g:

$$D^p g_\alpha + \sum_{\substack{0 \leqslant i \leqslant p-1 \\ 0 \leqslant \beta \leqslant m}} A^{\beta}_{i,\alpha} D^i g_\beta = 0$$

As the matrix U depend on a parameter λ then the matrices A_i depend on λ too. It is remarkable that very often the previous equation for $g = 0$ can be reduced into the form

$$L\, g = \mu(\lambda) g, \tag{43}$$

where L is an independent on λ integrodifferential operator and μ is a scalar function. And moreover the recursion operator R takes the form $R = L^{+}$. This fact gives a possibility to find the recursion operator with the help of the equation (43) for the function (38).

There are two procedures in JET for finding the recursion operator. If you know the zero curvature representation for your system (31) then you can try to exploit the eigenvalue equation (43) for the function (38). To do this one must expand the matrix U with respect to the basis $[A_i]$ of a Lie algebra and find the closed table of commutators (40). Then try to call the procedure **triada** that use the second algorithm described above. The syntax is `triada(U,s)` or `triada(U,s,p)`, where U is the matrix $\mathbf{U}$ in the *symbolic* form (39), `s` is the set of equations (40) and `p` is integer positive number, number of steps executing by the routine. The third parameter is optional. The name **U** is fixed in the auxiliary routine `df` that is called from `triada` therefore you must use this name only for the first matrix (39) of the zero curvature representation.The routine `triada` computes the functions (38) in the form (41), differentiates them according to (42) and expresses e_i as linear forms of g_α: $e_i = \mathcal{L}_i(g)$. These equations are stored in the pair of global sets `SUB` and `SUB1`. The independent on e_i linear system for g_α is returned.

Let us consider, for example, the KdV equation possessing the algebra (35). If we set there $x_4 = -4\lambda$, $A_2 = A_2' - \lambda A_1$ and $A_3' = A_4$ then $U = A_1(u0 - \lambda) + A_2'$ and $[A_1, A_2', A_3]$ is the Cartan-Weyl basis:

$$[A_1, A_2'] = A_3', \quad [A_1, A_3'] = 2A_1, \quad [A_2', A_3'] = -2A_2', \tag{44}$$

We shall use this basis now:

```
> vard:=[u]: depend( ):
> U:=A1*(u0-k)+A2: matrices:={A1,A2,A3}:
> s:={com(A1, A2) = A3,com(A1, A3)=2*A1, com(A2, A3) = -2*A2}:
> triada(U,s);
```

$$\left[\frac{\partial^3 g}{\partial x^3} + 4\, u0\, \frac{\partial g}{\partial x} - k\, \frac{\partial g}{\partial x} + 2\, u1\, g\right]$$

Transforming the obtained equation (or system in the vector case) to the form (43) we find R^{+} and

$$R = D^2 + 4\, u_0 + 2\, u_1\, D^{-1}. \tag{45}$$

This is the well-known Lenart operator.

When the matrices U and V are embedded in the Lie algebra of a small dimension then the gradient algorithm works satisfactory. But when the matrix U is embedded in a Lie algebra with dimension $\geqslant 15$ then the equation $L(u,k)g = 0$ is too large object. In this case you can try use the third argument of `triada`, but it will be better to compute the recursion operator solving the equation (36) directly.

It is known (see [25] for instance) that the recursion operators for evolution systems take the following form

$$R = \sum_{i=0}^{n} F_{n-i}(u)D^i + S(u)D^{-1}Gt(u), \tag{46}$$

where F_{n-i} are square matrices, the matrices S and Gt may be rectangular. In particular it is possible that S is a column and Gt is a row. For a single evolution equation S and Gt are vectors in general case. One can prove that the columns of S are Lie-Bäcklund symmetries and the rows of Gt are conserved covariants. That is,

$$(D_t - K')\,S = 0, \quad (D_t + K'^{+})\,Gt^T = 0 \tag{47}$$

and these equations can be solved beforehand. Let us mention that the integral term in (46) may be transformed according to the formula

$$S(u)D^{-1}Gt(u) = (S(u)C)D^{-1}(C^{-1}Gt(u)) \equiv \tilde{S}(u)D^{-1}\tilde{G}t(u),$$

where C is any non-singular constant matrix. This means that one may take the basis symmetries for columns of S then the rows of Gt will be the linear combinations of the basis covariants. And vice versa rows of the Gt may be the basis covariants then columns of the S are the linear combinations of the basis symmetries.

To obtain from (36) equations for the coefficients F_i, S and Gt of the recursion operator the following formula for integration by parts is used

$$D^{-1}FD^k = \sum_{s=1}^{k}(-1)^{s-1}(D^{s-1}F)D^{k-s} + (-1)^k D^{-1}(D^k F). \tag{48}$$

The equations for the coefficients F_i, S and Gt are coded in our procedure `recursion`. One can call it with one or two input parameters, the call `recursion(m)` is possible if m=0 only, **m** is the number of the calling equation. If the order **n** of the operator (46) is known then another calling sequence is possible `recursion(m,n)`, in this case the complete sequence of values m=0,1, ... , n+N are allowed where N=ord(K). If one assume $n \geqslant k$ then the third calling sequence is useful `recursion(m,C>=k)`, where C is a free name. The first parameter m may be in this case 0,1, ... ,k.

Let us consider the KdV equation for example:

```
> vard:=[u]: sys:=[u3+6*u0*u1]: depend():
> a:=recursion(0);
```

$$a := 0$$

```
> b:=recursion(1);
```

Error, (in recursion) First argument is too large

```
> b:=recursion(1,n>=1);  c:=recursion(2,n>=2);
```

$$b := -3\,K3\,DF(F0) + n\,F0\,DF(K3)$$

$$c := -2\,K2\,DF(F0) - 3\,K3\,DN(F0,2) + nF0\,DF(K2) - 3\,K3\,DF(F1)$$

$$+(n-1)F1\,DF(K3) + \frac{1}{2}\,n(n-1)F0\,DN(K3,2)$$

Here n is the order of the recursion operator, $F0,\ F1,\ldots$ are exactly the coefficients of the operator (46). The name **n** is global. $K0,\ K1,\ldots$ are the coefficients of the operator

$$K' = \sum_{i=0}^{N} K_i\,D^i,$$

The number N is determined from the list `sys` (N=3 in our example). To solve the equations for $F0$, $F1,\ldots$ you must create and perform the following set ot substitutions

```
> s:={seq(cat(K,i)=Frechet(sys)[i],i=0..3)};
```

$$s := \{K2 = 0, K0 = 6u1, K1 = 6u0, K3 = 1\}$$

where `Frechet` is the routine for the Frechet derivative of the scalar and vector fields on the jet space. The syntax is `Frechet(F)` where `F` is an algebraic expression or a list of expressions.

ATTENTION: All substitutions are allowed when flag=0 only! The routines `recursion`, `Noether` and `INoether`, stand flag=0 automatically.

Continuing the previous dialog we find

```
> b:=expand(eval(subs(s,b)));
```

$$b := -3\,DF(F0)$$

This means that $F0 = F0(t)$. Let us assume that $F0 = 1$ and $n = 2$, then

```
> s:=s union {F0=1}:
> c:=recursion(2,2): c:=expand(eval(subs(s,c)));
```

$$c := -3\,DF(F1)$$

Assuming $F1 = 0$, one can find $F2 = 4\,u0$ from the third equation, and the fourth is

```
> s:=s union {F1=0,F2=4*u0}:
> a:=recursion(4,2): a:=expand(eval(subs(s,a/3)));
```

$$a := 12\,DF(u1) - 6\,DN(u0,2) - 3\,(DF(\_S)\& * \_Gt) - 3\,(\_S\& * DF(\_Gt))$$

To solve such equations one must find $\_S$ and $\_Gt$ from the equations $(D_t - K')\_S = 0$, $(D_t + K'^{+})\_Gt^T = 0$. But in our example the solution is obvious $\_S = 2\,u1$, $\_Gt = 1$ and we obtain Lenart's operator (45).

Let us consider now the Kupershmidt equation

$$u_t = u_5 + 5\,u_1\,u_3 + 5\,u_2^2 - 5\,u_0^2\,u_3 - 20\,u_0\,u_1\,u_2 - 5\,u_1^3 + 5\,u_0^4\,u_1. \tag{49}$$

This equation possesses the six order recursion operator in the form (46) where $\_S$ and $\_Gt$ are two-dimensional vectors [29], [30].

```
> vard:=[u]: depend(F1(x),F2(x) ):
> sys:=[u5+5*u1*u3+5*u2^2-5*u0^2*u3-20*u0*u1*u2-5*u1^3+5*u0^4*u1]:
> s:={seq(cat(K,i)=Frechet(sys)[i],i=0..5),F0=1,F1=0}:
```

If one use the shown substitution s, then the equations $0-2$ are satisfied and the next is

```
>  a:=recursion(3,6): a:=expand(eval(subs(s,a)));
```

$$6\,DF(u1) - 6\,DF(u0^2) - DF(F2)$$

```
>  pot(%);
```

$$Finish, rm = 0$$
$$6\,u1 - F2 - 6\,u0^2$$

```
> solve(%,{F2});
```

$$\{F2 = 6\,u1 - 6\,u0^2\}$$

```
> s:=s union %:
```

Notice that the routine `pot` stands flag=1 and restores the previous value of flag before returning the result. Therefore one must declare all functions in `depend` before calling `pot`. One can find $F3, \ldots, F6$ by the same way and the next commands are

```
>  depend(_Gt(x),_S(x)):
>  a:=recursion(8,6): a:=expand(eval(subs(s,a))):
> flag:=1:  a:=expand(eval(subs(s,a/5)));
```

$$\begin{aligned} a := &- 30\,u2\,u0\,u3 - 10\,u0\,u1\,u4 + 10\,u0^3u4 + 50\,u0^2u2^2 + 20\,u1^4 - 20\,u1^2u3 \\ &- 30\,u1\,u2^2 + 160\,u0\,u1^2u2 - 60\,u0^4u1^2 + 80\,u0^2u1\,u3 - 2\,u4\,u2 - 2\,u0\,u6 \\ &- 12\,u0^5u2 - \_S\& * \frac{\partial \_Gt}{\partial x} - \frac{\partial \_S}{\partial x}\& * \_Gt - 4\,u5\,u1 \end{aligned}$$

ATTENTION: It will be the error if you enter flag=1 immediately after `a:=recursion(8,6)`.

```
> b:=pot(a);
```

$$\textit{Can not integrate}$$

$$b := -2\,u0\,u5 - 2\,u1\,u4 - 10\,u3\,u0\,u1 + 10\,u3\,u0^3$$
$$-10\,u0\,u2^2 - 10\,u1^2 u2 + 50\,u0^2 u1\,u2 + 20\,u0\,u1^3 - 12\,u0^5 u1$$

```
> rm;
```

$$-\_S\& * \frac{\partial \_Gt}{\partial x} - \frac{\partial \_S}{\partial x}\& * \_Gt$$

The routine `pot` can not integrate rm, but everybody can do it by hand and the result is $< \_S, \_Gt >= b$, where the angle brackets denote the scalar product $< \_S, \_Gt >= \_S_1\,\_Gt_1 + \_S_2\,\_Gt_2$. Equation (49) is time independent and it possesses the symmetry $\_S_1 = u_t = sys[1] = u_5 + \ldots$ and u_0 is the conserved covariant. Hence one can try set $\_S_1 = sys[1]$, $\_Gt_1 = -2\,u_0$:

```
> c:=factor(b+2*u0*sys[1]);
```

$$c := -2\,u1\,\left(-5\,u0\,u1^2 + 5\,u1\,u2 + u4 - 5\,u0^2 u2 + u0^5\right)$$

The field u_1 is the symmetry with respect to translations along the axis X and the expression in the brackets is the conserved covariant as it can be easily checked with the help of the routine `covariant`. If one add to the set s two more equations

```
> s:=s union {_S=<-2*op(sys),-2*u1>,
     _Gt=<u0,u4-5*u0^2*u2-5*u0*u1^2+5*u1*u2+u0^5>}:
```

then this set contains the solution of all equations for the recursion operator.

Our next example is the Schrödinger system

$$u_t = u_2 + u_0^2\,v_0, \quad v_t = -v_2 - v_0^2\,u_0. \tag{50}$$

To find the recursion operator one must enter the following commands:

```
> vard:=[u,v]: depend( ):
> sys:=[u2+u0^2*v0,-v2-v0^2*u0]:
> s:={K0=Frechet(sys)[0],K1=Frechet(sys)[1],K2=Frechet(sys)[2]};
```

$$K0 = \begin{bmatrix} 2\,u0\,v0 & u0^2 \\ -v0^2 & -2\,u0\,v0 \end{bmatrix}, \quad K1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad K2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Let us call now the routine `recursion`.

```
>  a:=recursion(0);
```

$$a := (F0\& * K2) - (K2\& * F0)$$

As $K2$ is the diagonal matrix with the different elements then $F0$ is diagonal too. Follow to [31] we set $n = 1$, $F0 = K2$, $F1 = 0$, then $a = 0$ and the next equation is

```
> const:={F0,K2}:
> s:=s union {F0=Frechet(sys)[2],F1=0}:
> a:=recursion(2,1):a:=eval(subs(F1=0,K1=0,a));
```

$$a := (F0\& * K0) - (K0\& * F0) - \& * (K2, \_S, \_Gt) + \& * (\_S, \_Gt, K2)$$

One can prove that if one set

```
> {_S=array(1..2,1..2,[[u0,0],[-v0,0]]),
   _Gt=array(1..2,1..2,[[v0,u0],[0,0]])};
```

$$\left\{ \_S = \begin{bmatrix} u0 & 0 \\ -v0 & 0 \end{bmatrix},\ \_Gt = \begin{bmatrix} v0 & u0 \\ 0 & 0 \end{bmatrix} \right\}$$

then all equatons are satisfied and the well known operator is obtained.

Recursion operators for many important equations are hereditary. Operator R is called hereditary one if the bilinear form $\Phi(f,g) = R'[Rf]g - R\,R'[f]g$ is symmetric [26] – [28]. The routine **hered** computes and simplifies the expression $\Phi(f,g) - \Phi(g,f)$ using the differentiation rules and the formula (48). If zero is obtained then true is returned else false is returned and the simplified expression is stored under the name **rm**.The syntax is `hered(L,k)` where $L = [F_0, F_1, \ldots, F_n, S, Gt]$ is the list of the coefficients of the operator in the form (46), k is the matrix dimension of the operator. Let us consider the examples.

For the Lenart operator (45) one has to enter the following commands:

```
> vard:=[u]:hered([1,0,4*u0,2*u1,1],1);
```

$$true$$

One more example is the recursion operator for the potential Sawada-Kotera equation [29]

$$\begin{aligned} R = {} & D^6 + 6u_1D^4 + 3u_2D^3 + (8u_3 + 9u_1^2)D^2 + (2u_4 + 3u_2u_1)D \\ & + 3u_5 + 13u_3u_1 + 3u_2^2 + 4u_1^3 - 2u_1D^{-1}(u_4 + u_2u_1) \\ & - 2D^{-1}(u_6 + 3u_4u_1 + 6u_3u_2 + 2u_2u_1^2) \end{aligned}$$

```
> vard:=[u]: L:=[1,0,6*u1,3*u2,8*u3+9*u1^2,2*u4+3*u2*u1, 3*u5+13u3*u1
  +3*u2^2+4*u1^3,<-2*u1,-2>,<u4+u2*u1,u6_3*u4*u1+6*u3*u2+2*u2*u1^2>]:
> hered(L,1);
```

$$true$$

It was the difficult test, to perform it the Intel processor at 400 Mhz worked 67 s.

If the routine returns `false` the transformed expression $\Phi(f,g)-\Phi(g,f)$ is stored under the name **rm** for the visual control and moreover the integral terms in the form $D^{-1}FD^{-1}G$ or $D^{-1}F$ are stored separately under the name **zero**.

The next example is the recursion operator for the nonlinear Schrödinger system:

$$R=\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} D + 2 \begin{bmatrix} u0 & 0 \\ -v0 & 0 \end{bmatrix} D^{-1} \begin{bmatrix} v0 & u0 \\ 0 & 0 \end{bmatrix}$$

```
> L:=[matrix([[1,0],[0,-1]]),0,2*matrix([[u0,0],[-v0,0]]),
  matrix([[v0,u0],[0,0]])];
> vard:=[u,v]:hered(L,2);
```

$$true$$

9 Noether operators

The Noether operator Θ maps the set of conserved covariants of an evolution system

$$u_t = K(u) \tag{51}$$

into the set of its Lie-Bäcklund symmetries. The operator J performing the inverse map is called the inverse Noether operator.

The Noether operator Θ and the inverse Noether operator J satisfy the following equations

$$(D_t - K')\Theta = \Theta(D_t + K'^{+}), \tag{52}$$
$$(D_t + K'^{+})J = J(D_t - K'). \tag{53}$$

Of course, if Θ satisfies the equation (52) and Θ^{-1} exists then it satisfies the equation (53). But one can not find Θ^{-1} (or J^{-1}) explicitly as a rule.

Let us consider the following operators

$$\Theta_s = \frac{1}{2}(\Theta+\Theta^{+}), \quad \Theta_a = \frac{1}{2}(\Theta-\Theta^{+}),$$
$$J_s = \frac{1}{2}(J+J^{+}), \quad J_a = \frac{1}{2}(J-J^{+}).$$

Θ_s (J_s) and Θ_a (J_a) are called the symmetric and antisymmetric parts of the operator Θ (J) respectively. It follows from the definitions (52) and (53) that the both Θ_s and Θ_a are the Noether operators; J_s and J_a are the inverse Noether operators. There is the observation that Noether and inverse Noether operators are antisymmetric for the systems possessing the zero curvature representations. Nevertheless we consider below the operators Θ and J in general form.

If an evolution system admits two Noether operators Θ_1 and Θ_2 and Θ_2 is invertible then $\Theta_1\,\Theta_2^{-1}$ is the recursion operator. If two inverse Noether operators J_1 and J_2 exist and J_2 is invertible then $J_2^{-1}J_1$ is the recursion operator. Some of the evolution systems possess the

Noether operator Θ and the inverse Noether operator J ($\neq \Theta^{-1}$) then ΘJ is the recursion operator [35].

The most general form of the Noether operators known today is

$$\Theta = \sum_{i=0}^{n} F_{n-i} D^i + S\, D^{-1} St, \tag{54}$$

$$J = \sum_{i=0}^{n} J_{n-i} D^i + G\, D^{-1} Gt. \tag{55}$$

where F_i, J_i, S, St, G and Gt are matrices depending on the jet space variables. The columns of S and rows of St are Lie-Bäcklund symmetries of the evolution system; the columns of G and rows of Gt are conserved covariants of the system. This means that for the system writing in the form (51) the following equations must be satisfied

$$(D_t - K')S = 0, \qquad (D_t - K')St^T = 0, \tag{56}$$

$$(D_t + K'^{+})G = 0, \qquad (D_t + K'^{+})Gt^T = 0, \tag{57}$$

where T is the transposition symbol. It's happened sometimes that $S = 0$ or $G = 0$ in (54) and (55).

Let us mention that the integral term in (54) (and (55)) may be transformed according to the formula

$$S\, D^{-1} St = (SC)\, D^{-1} (C^{-1} St) \equiv \tilde{S}\, D^{-1} \tilde{S}t,$$

where C is any non-singular constant matrix. This means that one may take the basis symmetries for columns of S then the rows of St are the linear combinations of the basis symmetries. And vice versa rows of the St may be the basis symmetries then the columns of the S are the linear combinations of the basis symmetries. By analogy one may take for the columns of G or for the rows of Gt the basis conserved covariants.

There are two procedures in JET for finding the operators (54) and (55). The procedure **Noether** returns the equations for the matrices F_i of the operator (54). The equations (56) must be solved beforehand. The procedure **INoether** returns the equations for the matrices J_i of the operator (55). The equations (57) must be solved beforehand. The both procedures **Noether** and **INoether** have the same syntax as the procedure `recursion`. They may be called with one or two parameters, but the call `Noether(m)` is possible for m=0 only; m is the number of returned equation. If you know that the order n of Θ is greater or equal to k you can call `Noether(m,C>=k)`, where C is any free name. In this case $m = 0, 1, \ldots, k$ and output depend on parameter **n** that is the global variable. When you know exactly that the order of Θ is n the call `Noether(m,n)` is possible for m=0,1, ... , n+N, N=ord(sys) and gives the complete system of equations (52).

The coefficients F_i and J_i of the operators (54) and (55) take in our routines the same names F_i. The coefficients S, St, G and Gt are denoted as $\_S$, $\_St$, $\_G$ and $\_Gt$ respectively, these coefficients are declared as the matrices in the global set `matrices` automatically.The coefficients F_i of Θ and J are square matrices and S, St, G, Gt may be rectangular. But in the scalar case F_i are scalar functions, $\_S$, $\_St$, $\_G$ and $\_Gt$ may be scalars or vectors. The coefficients of the operators $K' = \sum K_i D^i$ and $K'^{+} = \sum (-D)^i K_i^T$ are coded as K0, K1, ... and tK0, tK1, ... respectively. That is, tK0 is the transposed of K0, etc.

Let us consider examples. The first will be the KdV equation:

```
> vard:=[u]: depend( ): sys:=[u3+6*u0*u1]:
> s:={seq(cat(K,i)=Frechet(sys)[i],i=0..3)}; a:=Noether(0);
```

$$s := K0 = 6\,u1, K1 = 6\,u0, K2 = 0, K3 = 1$$
$$a := 0$$

```
> a:=Noether(1,C>=1);
```

$$a := 2\,F0\,K2 - (n+3)\,F0\,DF(K3) + 3\,K3\,DF(F0)$$

```
> a:=expand(eval(subs(s,a)));
```

$$3\,DF(F0)$$

Let us take $F0 = 1$ and $n = 3$, then

```
> s:=s union {F0=1}:
> a:=Noether(2,3): a:=expand(eval(subs(s,a)));
```

$$a := 3\,DF(F1)$$

Assuming that operator Noether does not depend on t we obtain $F1 = c1 = const$,

```
> s:={K0=6*u1,K1=6*u0,K2=0,K3=1,F0=1,F1=c1}: const:={c1}:
> a:=Noether(3,3): a:=expand(eval(subs(s,a)));
```

$$a := 12\,u1 - 24\,DF(u0) + 3\,DF(F2)$$

To find $F2$ one can apply `pot`:

```
> depend(F2(u0,u1),F3(u0,u1) ):
> pot(-a/3);
```

$$Finish,\ rm = 0$$
$$-F2 + 4\,u0$$

This means that $F2 = 4\,u0 + c2$.

```
> s:=s union {F2=4*u0+c2}: const:={c1,c2}:
> a:=Noether(4,3): a:=expand(eval(subs(s,a)));
```

$$a := 12\,u1\,c1 + 18\,DF(u1) - 24\,DN(u0,2) - 18\,c1\,DF(u0) + 3\,DF(F3) + 2\,\_S\& * \_St$$

Assuming $\_S = 0$ one can find as above $F3 = 2\,u1 + 2\,c1\,u0 + c3$. The 5th equation is satisfied identically and the 6th gives $c1 = c3 = 0$. So, we obtained the well known operator

$$\Theta = D^3 + (4\,u_0 + c_2)\,D + 2\,u_1 = D^3 + 2\,(u_0\,D + D\,u_0) + c_2\,D, \qquad (58)$$

where c_2 is arbitrary parameter.

Let us consider now the Kupershmidt equation.

$$u_t = u_5 + 5\,u_1\,u_3 + 5\,u_2^2 - 5\,u_0^2\,u_3 - 20\,u_0\,u_1\,u_2 - 5\,u_1^3 + 5\,u_0^4\,u_1. \qquad (59)$$

This equation admits Noether operator $\Theta = D$ and the inverse Noether operator J. Operator J takes the form (55) where **n**=5, G and Gt are vectors. The following commands are necessary

```
> depend(F1(u0,u1),F2(u0,u1) ): vard:=[u];
> sys:=[u5+5*u1*u3+5*u2^2-5*u0^2*u3-20*u0*u1*u2-5*u1^3+5*u0^4*u1]:
> s:={seq(cat(K,i)=Frechet(sys)[i],i=0..5)}: a:=INoether(0);
```

$$0$$

The first and the second equations give $F0 = 1$ and $F1 = c1$ respectively. To find the third equation one must enter the commands

```
> s:=s union {F0=1,F1=c1}: const:={c1,F0,F1,K5}:
> a:=INoether(3,5): a:=expand(eval(subs(s,a)));
```

$$a := 20\,u2 - 40\,u0\,u1 + 10\,DF(u1) - 10\,DF(u0^2) - 5\,DF(F2)$$

To integrate this and the next equations enter the command

```
> depend(F2(u0),F3(u0),F4(u0),F5(u0)):
```

Then

```
> solve(c2+pt(a/5),{F2}); s:=s union %:
> const:=const union {c2}:
```

$$\{F2 = c2 + 6\,u1 - 6\,u0^2\}$$

One can find by the same way $F3$, $F4$ and $F5$ from the 4th, 5th and 6th equations respectively. These functions contain arbitrary constants $c3$, $c4$ and $c5$. The 7th equation contains the term $DF(G\& * Gt)$ and the order of this equation is equal 5. Hence the covariants G and Gt have the fourth order. Applying the routine `covariant` one can find that the general fourth order covariant takes the following form

$$g = a_1\big(u4 + 5(u_1 - u_0^2)\,u_2 - 5u_0\,u_1^2 + u_0^5\big) + a_2\,u_0 + a_3, \tag{60}$$

where a_i are constants. So, one can set

```
> {_G=<u4+5*(u1-u0^2)*u2-5*u0*u1^2+u0^5,u0,1>, _Gt=<g1,g2,g3>};
```

$$\left\{\_G = \begin{bmatrix} u4 + 5\,\left(u1 - u0^2\right) u2 - 5\,u0\,u1^2 + u0^5 \\ u0 \\ 1 \end{bmatrix}, \_Gt = \begin{bmatrix} g1 \\ g2 \\ g3 \end{bmatrix}\right\}$$

where g_i are the covariants in the general form (60). Including this set in the set s, we found from the seventh and eighth equations all previous constants $c_i = 0$ and the covariants g_i:

$$g_1 = -2u0,\quad g_2 = -2(u4 + 5(u1 - u0^2)u2 - 5u0\,u1^2 + u0^5),\quad g_3 = const$$

The operator D is the Noether operator for the Kupershmidt equation and D^{-1} is the inverse Noether operator. Therefore one may set $g_3 = 0$ as this constant gives the trivial term $g_3\,D^{-1}$

in the operator J. Extracting the coefficients F_i from the set s we obtain

$$\begin{aligned} J = {} & D^5 + 6(u_1 - u_0^2)D^3 + 9(u_2 - 2u_0\,u_1)D^2 \\ & + (5\,u_3 - 22\,u_0\,u_2 - 13\,u_1^2 - 6\,u_1\,u_0^2 + 9\,u_0^4)D \\ & + u_4 - 8\,u_0\,u_3 - 15\,u_1\,u_2 - 3\,u_0^2\,u_2 - 6\,u_0\,u_1^2 + 18\,u_0^3\,u_1 \quad (61) \\ & - 2(u_4 + 5\,(u_1 - u_0^2)\,u_2 - 5\,u_0\,u_1^2 + u_0^5)D^{-1}u_0 \\ & - 2u_0D^{-1}(u_4 + 5\,(u_1 - u_0^2)\,u_2 - 5\,u_0\,u_1^2 + u_0^5). \end{aligned}$$

The computed operator coincides with the operator J from [29].

One more example is the Drinfeld-Sokolov-Hirota-Satsuma system [32], [33]

$$u_t = \frac{1}{2}u_3 + 3uu_1 - 6vv_1, \quad v_t = -v_3 - 3uv_1. \tag{62}$$

This system is often cited in the West as Hirota-Satsuma system. It possesses the first order inverse Noether operator (see [29], for example).

```
> vard:=[u,v]: sys:=[1/2*u3+3*u0*u1-6*v0*v1,-v3-3*u0*v1]:
> depend(F1(u0),F2(u0),F3(u0),F4(u0),F5(u0),F6(u0),_Gt(x),_G(x)):
> s:={}:for i from 0 to 3 do s:=s union {cat(K,i)=Frechet(sys)[i],
   cat(tK,i)=linalg[transpose](Frechet(sys)[i])} od:
> a:=INoether(0);
```

$$a := \& * (F0, K3) - \& * (tK3, F0)$$

As K_3 is the diagonal matrix and $tK_3 = K_3 = \text{diag}(1/2, -1)$ then $F0$ is also diagonal. We assume that $F0$ is a constant matrix:

```
> s:=s union {F0=matrix([[c1,0],[0,c2]])}:
> const:={F0,K3,tK3,c1,c2}:
> a:=INoether(1,C>=1);
```

$$a := \& * (F0, K2) + \& * (tK2, F0) + \& * (F1, K3) - \& * (tK3, F1) - 3DF(\& * (tK3, F0))$$

As this is the algebraic equation then no need to declare any functions

```
> s:=s union {F1=matrix([[c3,c4],[c5,c6]])}:
> a:=evsub(s,a);
```

$$a := \begin{bmatrix} 0 & -3/2\,c4 \\ 3/2\,c5 & 0 \end{bmatrix}$$

```
> s:=s minus {F1=array(1..2,1..2,[[c3,c4],[c5,c6]])} union
   {F1=array(1..2,1..2,[[c3,0],[0,c6]])}:
```

But the command `minus` does not work here. There are two ways to solve this problem: or copy the output and remove the old expression for $F1$ by hand or add to s two equations $c4 = 0, c5 = 0$. The next equation contains the covariants G and Gt therefore one must find them now. Very simple computation gives that $g = [c1u0 + c2, -2c1v0]$ is the general first order covariant:

```
> covariant([c1*u0+c2,-2*c1*v0]), zero;
```

$$\{\,\},\ \{\,\}$$

Let us try to set

```
> {_G=matrix([[u0,1],[-2*v0,0]]),
  _Gt=matrix([[k1*u0+k2,-2*k1*v0],[k3*u0+k4,-2*k3*v0]])};
> s:=s union %: const:=const union {c3,c6,k1,k2,k3,k4}:
```

$$\left\{\_G = \begin{bmatrix} u0 & 1 \\ -2\,v0 & 0 \end{bmatrix},\ \_Gt = \begin{bmatrix} k1\,u0 + k2 & -2\,k1\,v0 \\ k3\,u0 + k4 & -2\,k3\,v0 \end{bmatrix}\right\}$$

Then we obtain the following equations

```
> a:=INoether(2,1); a:=evsub(s,a);
```

$$\begin{aligned} a := -\&*(tK3,_{G}\,,_{G}\,t) + \&*(F0, K1) - \&*(tK1, F0) + \&*(F1, K2) \\ +\&*(tK2, F1) + \&*(F0, DF(K2)) + 2DF(\&*(tK2, F0)) \\ -3DF(\&*(tK3, F1)) - 3DN(\&*(tK3, F0), 2) + \&*(\_G, \_Gt, K3) \end{aligned}$$

$$a := \begin{bmatrix} 0 & 3\,u0\,k1\,v0 + 3\,k3\,v0 - 6\,c1\,v0 \\ -3\,u0\,k1\,v0 - 3\,v0\,k2 + 6\,c1\,v0 & 0 \end{bmatrix}$$

```
> s1:={k1=0,k2=2*c1,k3=2*c1}:
> s:=s union s1: a:=evsub(s,a);
```

$$a := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> a:=INoether(3,1): a:=evsub(s,a):flag:=1:a:=evsub(s,a);
```

$$a := \begin{bmatrix} 0 & -12\,c1\,v1 - 3\,c2\,v1 - 6\,c3\,v0 \\ -12\,c1\,v1 - 3\,c2\,v1 + 6\,c3\,v0 & 0 \end{bmatrix}$$

```
> s1:=s1 union {c3=0,c2=-4*c1}:
> s:=s union {c3=0,c2=-4*c1}: a:=evsub(s,a);
```

$$a := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> a:=INoether(4,1): a:=evsub(s,a):flag:=1:a:=evsub(s,a);
```

$$\begin{bmatrix} 0 & -3\,c6\,v1 - 6\,v0\,k4 \\ -3\,c6\,v1 + 6\,v0\,k4 & 3\,c6\,u1 \end{bmatrix}$$

```
> s1:=s1 union {c6=0,k4=0}:
> s:=s union {c6=0,k4=0}:
> a:=evsub(s,a);
```

$$a := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

It was the last equation. Let us extract now the solution from the set s.

```
> z:={F0,F1,_G,_Gt}:S:={}:
 for i in s do if has(z,lhs(i)) then S:=S union {i} fi od;
> S:=evsub(s1,S);
```

$$S := \left\{ \_G = \begin{bmatrix} u0 & 1 \\ -2\,v0 & 0 \end{bmatrix}, F1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, F0 = \begin{bmatrix} c1 & 0 \\ 0 & -4\,c1 \end{bmatrix}, \right.$$

$$\left. \_Gt = \begin{bmatrix} 2\,c1 & 0 \\ 2\,c1\,u0 & -4\,c1\,v0 \end{bmatrix} \right\}$$

If we set $c_1 = 1/2$ this solution coincides with one that is presented in [29]:

$$J = \begin{bmatrix} 1/2 & 0 \\ 0 & -2 \end{bmatrix} D + \begin{bmatrix} u0 & 1 \\ -2\,v0 & 0 \end{bmatrix} D^{-1} \begin{bmatrix} 1 & 0 \\ u0 & -2\,v0 \end{bmatrix} \tag{63}$$

Often it is important to know whether the Noether or inverse Noether operator antisymmetric. The routine **asymm** checks the antisymmetry condition $(L + L^{+})F = 0$, $\forall F$ for the operators taking the following form

$$L = L_0 D^n + L_1 D^{n-1} + \cdots + L_n + A D^{-1} B, \tag{64}$$

where L_i are $n \times n$ matrices, A and B are such matrices that AB is $n \times n$ matrix. When $n = 1$ A and B may be scalars or vectors. The syntax is `asymm(L,n)`, where L is the list $L = [L_0, L_1, \ldots, L_n, A, B]$, and n is the matrix dimension of the coefficients L_i. If the operator is antisymmetric then **true** is returned, else is returned **false**. The routine stands flag=1 and restore the previous value of flag before returning the result. Let us consider the examples. If we have

$$L = D^3 + uD + Du = D^3 + 2uD + u_1,$$

then

```
> vard:=[u]: asymm([1,0,2*u0,u1,0,0],1);
```

$$true$$

Let us add the integral terms $uD^{-1}u + D^{-1}$ or $uD^{-1} + D^{-1}u$:

```
> asymm([1,0,2*u0,u1,<u0,1>,<u0,1>],1);
```

$$true$$

```
> asymm([1,0,2*u0,u1,<u0,1>,<1,u0>],1);
```

$$true$$

But adding D^2, we obtain

```
> asymm([1,1,2*u0,u1,0,0],1);
```

$$false$$

In such cases the expression $(L + L^+)F$ is stored under the global name **rm**

```
> rm;
```

$$2\,\frac{\partial^2 F01}{\partial x^2}$$

for the visual control.

Let us consider the matrix operator (63):

```
> L:=[matrix([[1/2,0],[0,-2]]),matrix([[0,0],[0,0]]),
  matrix([[u0,1],[-2*v0,0]]),matrix([[1,0],[u0,-2*v0]])]:
> vard:=[u,v]: asymm(L,2);
```

$$true$$

Noether operator of an integrable evolution system is the implectic (Hamiltonian, cosymplectic) operator as a rule and the inverse Noether operator is the symplectic operator as a rule. We follow below to the definitions from [34], [35].

The operator Θ is called the implectic one if it is antisymmetric $\Theta^+ = -\Theta$ and the bracket $\{f, g, h; \Theta\} =< f,\ \Theta'[\Theta\, g]h >$ satisfies the Jacobi identity

$$\{f, g, h; \Theta\} + \{g, h, f; \Theta\} + \{h, f, g; \Theta\} \in \mathrm{Im}D. \qquad (65)$$

The operator J is called the symplectic one if it is antisymmetric $J^+ = -J$ and the bracket $[f, g, h; J] =< f,\ J'[g]h >$ satisfies the Jacobi identity

$$[f, g, h; J] + [g, h, f; J] + [h, f, g; J] \in \mathrm{Im}D. \qquad (66)$$

The angle brackets denote here the Euclidean scalar product and $f,\ g,\ h$ are arbitrary functions on the jet space. But it is sufficient to consider that $f,\ g,\ h$ depend on x only. Let us mention that there exists the powerful technique of the functional multivectors [4] for the check the identity (65). This technique is very useful for the hand computation, but it was simpler for us to code the direct reduction of (65) and (66) with the help of the integration by parts.

The procedure **implectic** checks the identity (65). The syntax is the same as for `asymm`: `implectic(L,n)`, L=$[L_0, L_1, \ldots, L_n, A, B]$ is the list of the coefficients of the operator (64), the second parameter n is the matrix dimension of the operator L. But if $n \neq$ `nops(vard)`, then the message on the error is returned. The parameter n was introduced for the control of correctness.

The routine **symplectic** checks on the identity (66). The syntax is `symplectic(L,n)` with the same parameters as for `implectic`. The output is **true** or **false**, for the both `implectic` and `symplectic`. If false is returned the reminder of integration is stored under the global name **rm** for additional control.

Let us consider, for example, the Noether operator (58). It is antisymmetric as it was shown above. Let us check the identity (65).

```
> vard:=[u]: implectic([1,0,4*u0+c2,2*u1,0,0],1);
```

$$true$$

The both identities (65) and (66) are satisfied for any linear operator with constant coefficients. For example, for the operators D and D^3 we have:

```
> implectic([1,0,0,0],1),implectic([1,0,0,0,0,0],1);
```

$$true,\ true$$

The potential Sawada-Kotera equation $u_t = u_5 + 5u_1\,u_3 + (5/3)\,u_1^3$ admits the following implectic Noether operator $\Theta = D + 2(u_1 D^{-1} + D^{-1} u_1)$ (see [29]). Let us check this

```
> implectic([1,0,2*<u1,1>,2*<1,u1>],1);
```

$$true$$

The DSHS system (62) admits the following Noether operator (see [29])

$$\Theta = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big(\frac{1}{2}\,D^3 + 2uD + u_1\Big) + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} (2vD + v_1).$$

Let us check is it implectic or not:

```
> vard:=[u,v]: L:=[1/2*matrix([[1,0],[0,1]]), 0, 2*matrix([[u0,v0],[v0,u0]]),
  matrix([[u1,v1],[v1,u1]]),0,0]:
> implectic(L,2);
```

$$true$$

The nonlinear Schrödinger system (50) admits the following Noether operator

$$\Theta = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} D + \begin{bmatrix} u & 0 \\ -v & 0 \end{bmatrix} D^{-1} \begin{bmatrix} u & -v \\ 0 & 0 \end{bmatrix}$$

```
> vard:=[u,v]: L:=[matrix([[0,1],[1,0]]),0,matrix([[u0,0],[-v0,0]]),
  matrix([[u0,-v0],[0,0]])]:
> implectic(L,2);
```

$$true$$

Let us consider now the examples with inverse Noether operators. The Kupershmidt equation (59) possesses the inverse Noether operator (61). The check of the symplecticness:

```
> L:=[1,0,6*u1-6*u0^2,9*u2-18*u0*u1,
   5*u3-22*u0*u2-13*u1^2-6*u1*u0^2+9*u0^4,
   u4-8*u0*u3-15*u1*u2-3*u0^2*u2-6*u0*u1^2+18*u0^3*u1,
   <u4+5*(u1-u0^2)*u2-5*u0*u1^2+u0^5,u0>,
   -2*<u0,u4+5*(u1-u0^2)*u2-5*u0*u1^2+u0^5>]:
> vard:=[u]: symplectic(L,1);
```

$$true$$

The Sawada-Kotera equation

$$u_t = u_5 + 5u\,u_3 + 5u_1\,u_2 + 5u^2\,u_1$$

possesses the following inverse Noether operator

$$J = D^3 + 2u\,D + u_1 + \Big(u_2 + \frac{1}{2}\,u^2\Big)D^{-1} + D^{-1}\Big(u_2 + \frac{1}{2}\,u^2\Big).$$

The check of the symplecticness:

```
> L:=[1, 0,2*u0,u1,<u2+u0^2/2,1>,<1,u2+u0^2/2>]:
> vard:=[u]: symplectic(L,1);
```

$$true$$

ATTENTION: We coded in `implectic` and `symplectic` integration of the most typical expressions. If the integral terms in an operator are cumbersome then we do not sure that integration will be completely performed. Therefore in such cases the message **"Probably false", see rm** is typed and simplified expression (65) or (66) is stored in **rm**. In such cases you have to look at the reminder `rm` and investigate it by hand.

Let we have, for example

$$J = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} D^3 + \begin{bmatrix} u0 & v0 \\ v0 & u0 \end{bmatrix} D + 1/2 \begin{bmatrix} u1 & v1 \\ v1 & u1 \end{bmatrix}$$

$$- \begin{bmatrix} 1 & u0^2 + v0^2 \\ 0 & ku0\,v0 \end{bmatrix} D^{-1} \begin{bmatrix} u0^2 + v0^2 & ku0\,v0 \\ 1 & 0 \end{bmatrix}$$

then entering `vard:=[u,v]:  L:=[...]:` we have

```
> asymm(L,2); symplectic(L,2);
```

$$true$$
$$Probably\ false,\ see\ rm$$

```
> factor(rm);
```

$$v0\,(k-2)\left(\frac{\partial G01}{\partial x}F02\,H01-\frac{\partial G01}{\partial x}F01\,H02+\frac{\partial F01}{\partial x}H02\,G01\right.$$
$$\left.+\frac{\partial H01}{\partial x}G02\,F01-\frac{\partial F01}{\partial x}H01\,G02-\frac{\partial H01}{\partial x}G01\,F02\right)$$

It is obvious that J is symplectic if and only if $k=2$.

10 Auxiliary Routines

The routine `Desol` is useful for solving the ordinary differential equations or systems in the JET notations:

```
> vard:=[u,v]: depend(f(x,u0,v0) );
> a:=dif(f,x$2)+f;
```

$$a:=\frac{\partial^2 f}{\partial x^2}+f$$

```
> Desol(a,{f(x)});
```

$$f=\_C1\sin(x)+\_C2\cos(x)$$

`Desol` transforms an input equation into standard Maple form, calls the built-in function `dsolve` and returns the result in the notations of the package JET. The first parameter of `Desol` is an equation or set of equations, the second parameter is the set of functions.

The routine `entry` transform the sums into lists:

```
> a:=b-c+d+e-f+g*x^2+k*y-r*sin(x):
> entry(a,3);
```

$$[b+d-c,\ -f+g\,x^2+e,\ k\,y-r\sin(x)]$$

The first parameter of `entry` is an expression, the second parameter is an integer number, number of addends of elements in the output list.

The routine `vrd` returns the sequences of the dependent variables:

```
> vard:=[u]:vrd(4);
```

$$u0,\ u1,\ u2,\ u3,\ u4$$

```
> vard:=[u,v]:vrd(4);
```

$$u0,\ v0,\ u1,\ v1,\ u2,\ v2,\ u3,\ v3,\ u4,\ v4$$

```
> vard:=[u,v]:vrd(4,2);
```

$$v0,\ v1,\ v2,\ v3,\ v4$$

The routine `DLT` returns unit or zero:

```
> DLT(1,2),DLT(1,-1),DLT(-2,-2),DLT(1,1);
```

$$0,\ 0,\ 1,\ 1$$

The routine `InT` is the inert form of the integration routine. It differs from the built-in `Int`:

```
> expand(InT(f^2,x)), expand(Int(f^2,x));
```

$$\int f^2\,dx,\ f^2\int 1\,dx$$

The routine `‘print/InT‘` provides output of the integrals in the mathematical form.

```
> InT(sin(x),x)=INT(sin(x),x);
```

$$\int \sin(x)\,dx = -\cos(x)$$

The routine `‘print/com‘` provides output of the commutators in the mathematical form and the routine `‘diff/com‘` introduces the rule of differentiation of the commutators (see p. 31).

The routine `pr` can multiply two matrices, or two vectors, or matrix and vector, or scalar and vector, or scalar and matrix, or two scalars. It processes the both arrays and symbolic matrices or vectors and symbolic vectors. It is called from the routines `recursion`, `Noether` and `INoether`. The routine `‘print/pr‘` provides output as the noncommutative product: $A\&*B$.

The routine `binom` computes the binomial coefficients with the both numerical and symbolic arguments:

```
> binom(5,0), binom(5,1), binom(k,2);
```

$$1,\ 5,\ \mathrm{binom}(k,2)$$

```
> notneg:={k,n}: binom(k,2), binom(n,3);
```

$$1/2\,k\,(k-1),\ 1/6\,n\,(n-1)\,(n-2)$$

The name **notneg** is global. If $k \in notneg$ then $k \geqslant 0$. The name **n** is global for `recursion`, `Noether`, `Inoether` and `binom`, it is assumed $n \geqslant 0$ always. There is the third optional argument, the calls `binom(n,m,k)` or `binom(m,n,k)` means that $n \geqslant k$, where k is a non-negative integer number:

```
> notneg:={k,n}: binom(k,k-2), binom(n-1,3), binom(n-1,3,2);
```

$$1/2\,k, (k-1), \mathrm{binom}(n-1,3), 1/6\,(n-1)\,(n-2)\,(n-3)$$

The routine `rec` computes the coefficients F_i of recursion or Noether or inverse Noether operators.The call `rec(i)` returns `cat(F,i)` if $0 \leqslant i \leqslant n$ or 0 in all other cases. Here n is the order of the operator.

The routine `rsys` computes the coefficients K_i of the operator $K' = K_0 + K_1 D + \cdots + K_N\, D^N$, where K is the list `sys`. The call `rsys(i)` returns `cat(K,i)` if $0 \leqslant i \leqslant N$ or 0 in all other cases.

The routine `trsys` computes the coefficients tK_i of the operator $K'^+ = K_0^T - DK_1^T + \cdots + (-D)^N K_N^T$, $(tK_i = K_i^T)$. The call `trsys(i)` returns `cat(K,i)` for a single evolution equation and `cat(tK,i)` for a system if $0 \leqslant i \leqslant N$ or 0 in all other cases.

The routine `newmatr` helps to enter the square matrices with arbitrary elements:

```
> A:=newmatr(3,k);
```

$$A := \begin{bmatrix} k1 & k2 & k3 \\ k4 & k5 & k6 \\ k7 & k8 & k9 \end{bmatrix}$$

The routine `eqord` is used for sorting the equations $x[i] = H_i$ with respect to the index i of the indexed variable $x[i]$:

```
> sort([x[2]=0,x[4]=f,x[1]=1,x[3]=g],eqord);
```

$$[x[1] = 1,\ x[2] = 0,\ x[3] = g,\ x[4] = f]$$

The routine `eqord2` is used for sorting the equations $x[i,j] = H_{ij}$ with respect to the second index j of $x[i,j]$. The both `eqord` and `eqord2` are called from `cd` and `acd`.

The routine `numdif` computes the differential order of a monomial with respect to indicated variable:

```
> depend(f(x,y),g(x,y)): a:=2*f*dif(f,x,y$3)*dif(g,x$2,y):
> numdif(a,x),numdif(a,y);
```

$$2,\ 3$$

This routine is called from `INT`, `implectic` and `symplectic`.

The routine `moddif` computes the differential polynomial by modulo $\partial/\partial x$ (x is the global name):

```
> depend(f(x),g(x),h(x)):
> a:=5*f*dif(g,x$4)*dif(h,x);
```

$$a := 5\,f\,\frac{\partial h}{\partial x}\,\frac{\partial^4 g}{\partial x^4}$$

```
> b:=moddif(a,x); b[2]+dif(b[1],x);
```

$$b := \left[5\,f\,\frac{\partial h}{\partial x}\frac{\partial^3 g}{\partial x^3} - 5\,\frac{\partial h}{\partial x}\frac{\partial f}{\partial x}\frac{\partial^2 g}{\partial x^2} - 5\,f\,\frac{\partial^2 h}{\partial x^2}\frac{\partial^2 g}{\partial x^2},\ 10\,\frac{\partial^2 h}{\partial x^2}\frac{\partial f}{\partial x}\frac{\partial^2 g}{\partial x^2} + 5\,\frac{\partial h}{\partial x}\frac{\partial^2 f}{\partial x^2}\frac{\partial^2 g}{\partial x^2} + 5\,f\,\frac{\partial^3 h}{\partial x^3}\frac{\partial^2 g}{\partial x^2}\right]$$

$$5\,f\,\frac{\partial h}{\partial x}\,\frac{\partial^4 g}{\partial x^4}$$

It is obvious from this example that the initial expression a is equivalent to $b[2]$. The routine is called from `implectic` and `symplectic`.

The routine `df` performs the differentiation according to the formula (42) where we identify the functions e_i and elements of the basis A_i of lie algebra.

```
> df(f,0), df(f,1), df(f,2);
```

$$f,\ DF(f),\ DN(f,2)$$

```
> matrices:={A1,A2,U}:
> df(A1,1),df(U,1),df(com(A1,A2),1),df(A1,2);
```

$$[A1,U],\ 0,\ [[A1,A2],U],\ [[A1,U],U]$$

This routine is called from `triada`. The name U is global for `df` (U is one of the matrices of the zero curvature representation).

The routine `opt` optimizes the process of solving the linear algebraic system so that large denominators do not arise. It is called from `triada`.

Acknowledgements

Some of the routines were written in collaboration with I. Kulemin and D. Demskoy. Author is grateful to them for the help.

Author is also grateful to Dr. Jan Sanders for hospitality and stimulating discussions during February 2001.

References

[1] Meshkov A.G., Kulemin I.V. *Package JET for computation the conserved densities and symmetries.* Algebraic and Analytic Methods in the Differential Equations Theory. Proc. Int. Conf. Orel, 14-19 November 1996. Orel, 1996, 99–103 [in Russian].

[2] Meshkov A.G. *Computer Package for Investigation of the Complete Integrability.* Proc. of the Third Int. Conf. Symmetry in Nonlinear Physics. Kyiv, 12-18 July 1999. Part 1. Kyiv, 2000, p.35-46.

[3] Ibragimov N.H. Transformation Groups in Mathematical Physics. Nauka, Moskow, 1983 [in Russian].

[4] Olver P.J. Applications of Lie Groups to Differential Equations. Springer-Verlag, 1986. (In Russian Mir, Moskow, 1989).

[5] CRC Handbook of Lie Group Analysis of differential equations. Ed. Ibragimov N. H. CRC Press, London, Tokio, 1994, 1995.

[6] Mikhailov A. V., Shabat A. B. and Sokolov V. V. *The symmetry approach to the classification of integrable equations* in: *Integrability and kinetic equations for solitons*, 1990, 213–279, Naukova Dumka, Kiev, [in Russian]; in English see *What is Integrability ?/* Springer-Verlag (Springer Series in Nonlinear Dynamics), 1991, 115–184.

[7] Akhatov I. S., Gazizov R. K. and Ibragimov N H. *Nonlocal Symmetries. Heuristic Approach.* In Sovrem. Probl. Math. 1989, V. 34, Moscow: VINITI, P. 3. (In Russian).

[8] Sokolov V. V. and Svinolupov S. I. Weak nonlocalities in evolution equations. *Mat. Zametki*, 1990, V. 48, no. 6, 91–97 (in Russian); *translation in Math. Notes*, 1990, **48**, no.**5-6**, 1234–1239, (1991).

[9] Volterra V. Theory of Functionals and Integrodifferential equations. London, 1929 (reprinted in 1959 by Dover).

[10] Galindo A., Martinez L. Kernels and ranges in the variational formalism. *Lett. Math. Phys.*, 1978, V.2, no.5, 385–390.

[11] Lax P.D. Integrals of nonlinear equations of evolution and solitary waves. *Commun. Pure and Appl. Math.* 1968, V.21, 467–490.

[12] N. H. Ibragimov and A. B. Shabat. Evolution equations with nontrivial Lie-Bäcklund groups. *Functs. Anal.*, 1980, V.14, no.1, 25–36 [in Russian]; N. H. Ibragimov and A. B. Shabat. On the infinite Lie-Bäcklund algebras. *Functs. Anal.*, 1980, V.14, no.4, 79–80 [in Russian].

[13] V. V. Sokolov and A. B. Shabat. Classification of integrable evolution equations. *Soviet Sci. Rev. C*, 1984, V.4, 221–280. Harwood Academic Publ.

[14] Chen H.H., Lee Y.C., Liu C.S. Integrability of nonlinear Hamiltonian systems by inverse scattering transform. *Phys. scr.*, 1979, V.20, N 3, 490–492.

[15] Meshkov A.G. Necessary conditions of the integrability. *Inverse Problems*, 1994, V.10, 635–653.

[16] V. V. Sokolov. Pseudosymmetries and differential substitutions. *Functs. Anal. i ego Pril.*, 1988, V.22, no.2, 47–56 [in Russian]; *translation in Functional Anal. Appl.*, 1988, **22**, no.**2**, 121–129.

[17] Lamb G. L., Jr. Bäcklund transformations for certain nonlinear evolution equations. *J. Math. Phys.*, 1974, V.15, no. 12, 2157–2165.

[18] Rogers C. Application of reciprocal Bäcklund transformations to a class of nonlinear boundary value problems. *J. Phys. A.*,1983, V.16, no. 14, L493–L495.

[19] Holm D. D., Kupershmidt B. A., Levermore C. D. Canonical maps between Poisson brackets in Eulerian and Lagrangian descriptions of continuum mechanics. P̂hys. Lett., 1983, V. A98, no. 8-9, 389–395.

[20] Mikhailov A. V., Shabat A. B. and Yamilov R. I. Extension of the module of invertible transformations. *Dokl. AN SSSR*, 1987, V.295, no.2, 288–291 [in Russian]; Extension of the module of invertible transformations. Classification of integrable systems. *Commun. Math. Phys.*, 1988, V.115, no.1, 1–19.

[21] Wahlquist H.D., Estabrook F.B. Prolongation structures of nonlinear evolution equations. *J. Math. Phys.*, 1975, V.16, 1–7.

[22] Fokas A.S., Anderson R.L. On the use of isospectral eigenvalue problems for obtaining hereditary symmetries for Hamiltonian systems. *J. Math. Phys.*, 1982, V.23, N 6, 1066–1073.

[23] Meshkov A.G. Symmetries and Conservation Laws for Evolution Equations. VINITI, no. 1511–85, Moskow, 1985 [in Russian].

[24] Prikarpatsky A.K. Gradient algorithm for constructing the criteria of integrability of nonlinear dynamical systems. *Dokl. AN SSSR*, 1986, V.287, no.4, 827–832 [in Russian].

[25] Gurses M., Karasu A. and Sokolov V. V. On constructing of recursion operator from Lax representation, *J. Math. Phys.*, 1999. Gürses M. and Sokolov V. V. On constructing of recursion operator from Lax representation. *ibid.* 2000.

[26] Asano N., Kato Y. Spectrum method for a general evolution equation. *Progr. Theor. Phys.*, 1977, V. 58, no. 1, 161–174.

[27] Fuchssteiner B. Application of hereditary symmetries to nonlinear evolution equations. *Nonlinear Anal. Theor. Meth. Appl.*, 1979, V. 3, no. 6, 849–862.

[28] Fuchssteiner B. The Lie algebra structure of nonlinear evolution equations admitting infinite dimensional Abelian symmetry groups. *Progr. Theor. Phys.*, 1981, V. 65, no. 3, 861–876.

[29] Wang Jing Ping. *Symmetries and Conservation Laws of Evolution Equations.* PhD thesis, de Vrije Univrsiteit te Amsterdam, 1998.

[30] Fordy A. P., Gibbons J. Integrable nonlinear Klein-Gordon equations and Toda lattice. *Commun. Math. Phys.*,1980, V.77, 21–30.

[31] Oevel W. *Rekursionsmechanismen für Symmetrien und Erhaltungssätze in Integrablen Systemen.* PhD thesis, Universität-Gesamthochschule Paderborn, 1984.

[32] Drinfeld V. G. and Sokolov V. V. *New evolution equations having* **(L-A)***-pairs*, Trudy Sem. S. L. Soboleva, Inst. Mat. Novosibirsk, 1981, **2**, 5-9 [in Russian].

[33] Hirota R., Satsuma J. Soliton solutions of a coupled Korteweg-de Vries equations. *Phys. Lett.* 1981, V. A85, no. 8, 407–408.

[34] Focas A.S., Fuchssteiner B. On the structure of symplectic operators and hereditary symmetries. *Lett. Nuovo Cimento*, 1980, v.28, no.8, 299-303.

[35] Fuchssteiner B., Focas A.S. Symplectic structures, their Bäcklund transformations and hereditary symmetries. *Physica,* 1981, V. D4, no.1, 47–66.

Index