Proceedings of Institute of Mathematics of NAS of Ukraine 2000, Vol. 30, Part 1, 35–46.

Computer Package for Investigation of the Complete Integrability

A.G. MESHKOV

Oryol State University, 95 Komsomolskaya Str., Oryol, 302015, Russia
E-mail: meshkov@esc.private.oryol.su

The problems concerned with the complete integrability of the partial differential systems with two independent variables are considered. The algorithms and the Maple V procedures for the investigation of complete integrability and some examples are presented.

1 General information

This paper describes a new computer package for the investigation of partial differential systems with two independent variables. We called this package JET because of the jet space language is used. The package includes twenty eight basic procedures in Maple V language and fifteen auxiliary procedures. Our main aim was to create the full collection of the instruments for the investigation of completely integrable systems. But the package can be used for other purposes as well.

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

Here is the list of the basic procedures:

```
dif, INT, depend, DF, DN, ED, EU, ord, pot, defeq, SU, part, chn,
cho, L_E,  recursion, Frechet, Noether, INoether, implectic,
symplectic, com, Jac, evsub, struct, Cmetric, Killing, triada
```

We comment this list in the subsequent sections. And here we mention only that all procedures work in **interactive mode**. The automatic mode is impossible in view of two following reasons. First, the computation of the higher conserved densities or Lie–Bäcklund higher symmetries of nonlinear **systems** leads us to very cumbersome partial differential systems whose solutions are unknown to science. Second, these systems often contain dozens of thousands terms. Solving such systems is an art but not a mechanical process.

2 Differentiation and integration

Two first names in the previous list, **dif** and **INT**, are names of procedures for differentiation and integration. There are 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 computes Lie–Bäcklund symmetries or conserved densities knows that you are forced to deal with a lot of arbitrary functions. Equations arising in such problems often are very long. The procedure **depend** enables to omit all arguments of all functions. 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);
```

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

The first expression is 3–4 times shorter than the second one. It is very important if you deal with a long expression. The procedures `dif` and `INT` possess the same facilities as the built-in `diff` and `int`. Moreover, `INT` possesses many powerful facilities for operating with arbitrary functions. In continuation of the previous input dialog we give the next examples

```
> INT(a,u0);
```

$$\frac{\partial f}{\partial v0}\,\frac{\partial^3 f}{\partial u0^2 \partial v0} - \frac{1}{2}\left(\frac{\partial^2 f}{\partial u0 \partial v0}\right)^2$$

```
> INT(u0^2*dif(f,u0$3), u0);
```

$$u0^2\,\frac{\partial^2 f}{\partial u0^2} - 2\,u0\,\frac{\partial f}{\partial u0} + 2\,f$$

and so on. The built-in procedure `int` returns such integrals without having them evaluated.

The next procedure **DF** calculates the total derivative with respect to x on a jet space and **DN(f,n)** calculates the n-th total derivative of f:

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

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

The procedure **ED** computes the evolution derivative

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

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

$$u_t^\alpha = K^\alpha(u). \tag{1}$$

One has to input the vector field K beforehand as the list **sys** (`sys` is the global name). For example, if you deal with the KdV equation $u_t = u_{xxx} + 6\,u\,u_x$ you must enter the following commands:

```
> vard:=[u]: sys:=[u3+6*u0*u1]:
```

3 Symmetries and conservation laws

The determining equation for Lie–Bäcklund symmetries of the system $u_t = K$ takes the following form (see [1, 2] or [3] for instance):

$$(D_t - K')F = 0, \tag{2}$$

where the prime denotes the Fréchet derivative

$$(K')^\alpha_\beta = \frac{\partial K^\alpha}{\partial u^\beta_i} D^i. \tag{3}$$

Here and below the summation rule over the repeated indices is implied. The procedure **Frechet** calculates the Fréchet derivative in different forms for scalar and vector cases. Let us consider the examples.

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

$$array\left(0..3, \left[\ (0) = \frac{\partial f}{\partial u0}\ \ (1) = \frac{\partial f}{\partial u1}\ \ (2) = 0\ \ (3) = 1\ \right]\right)$$

Here we obtained a 1-dimensional array with scalar elements ($\partial F/\partial u_i$). But in the vector case the elements of this array are square matrices:

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

$$array\left(0..2, \left[\ (0) = \begin{bmatrix} \frac{\partial f}{\partial u0} & \frac{\partial f}{\partial v0} \\ \frac{\partial g}{\partial u0} & \frac{\partial g}{\partial v0} \end{bmatrix}\ \ (1) = \begin{bmatrix} \frac{\partial f}{\partial u1} & \frac{\partial f}{\partial v1} \\ \frac{\partial g}{\partial u1} & \frac{\partial g}{\partial v1} \end{bmatrix}\ \ (2) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\ \right]\right)$$

To obtain the left hand side of equation (2) you do not need to use the procedure `Frechet`. More simple way is provided by the procedure **defeq**. For example, in order to compute the third order Lie–Bäcklund symmetries for the KdV equation you must enter the following commands:

```
> vard:=[u]: depend(F(t,x,u0,u1,u2,u3) ):
> sys:=[u3+6*u0*u1]: flag:=0: a:=defeq([F],1);
```

$$a := ED(F) - 6\,u1\,F - 6\,u0\,D(F) - D^{(3)}(F)$$

Here **flag** is the control variables, $D(F)$ is `DF(F)` and $D^{(3)}(F)$ is `DN(F,3)`. If `flag=0` then the expressions `ED(F)`, `DF(F)` and `DN(F,n)` are not expanded. But if one assigns `flag:=1` or nothing (`flag:='flag':`) then all expressions will be expanded. Let us continue our example:

```
> flag:=1: a:=a: nops(");
```

62

This means that the expression a consists of 62 terms and there is no need to look through it. In order to know which variables this expression contains, we use the procedure **ord**:

```
> ord(a);
```

[5]

This means that the order of a is equal to 5, that is, expression a contains u5. For the systems ord(a) returns a list [m, n, ...]. If vard=[u,v] and ord(a)=[2,3] for example, then the expression a contains u2 and v3 and does not contain u3, u4, ..., or v4, v5, ...

More detailed information about the expression a can be obtained with the help of the built-in procedure **indets**. Let us mention that the obtained expression `a` is a polynomial with respect to the highest order variables u_i, and therefore the built-in procedure **degree** is useful as well. To extract the terms with u5 one can use the procedure **chn** (CHoose Name), but the better way is to use the following command:

```
> b:=factor(chn(a,u5));
```

$$b := -3\,u5\left(\frac{\partial^2 F}{\partial x\partial u3}+\frac{\partial^2 F}{\partial u3^2}\,u4+\frac{\partial^2 F}{\partial u3\partial u0}\,u1+\frac{\partial^2 F}{\partial u3\partial u1}\,u2+\frac{\partial^2 F}{\partial u3\partial u2}\,u3\right)$$

It is easy to see that $b = -3\,u5\,D(\partial F/\partial u3)$. Hence the equation $b = 0$ implies $\partial F/\partial u3 = f_1(t)$ or $F = f_1(t)\,u3 + f_2(t, x, u0, u1, u2)$. To continue the computation you must enter the following commands:

```
> depend(f1(t), f2(t,x,u0,u1,u2) ): F:=f1*u3+f2:
> a:=expand(eval(subs(Diff=dif,a))):
```

The last command is necessary for the recomputation of all derivatives because the procedure `dif` returns the result in the inert form, for example `dif(f,u0)` → `Diff(f,u0)`.

The next problem that we consider is the computation of conserved currents. The vector function (ρ, θ) on the jet space is called the conserved current if it solves the equation

$$D_t\,\rho = D\,\theta, \tag{4}$$

where D_t is the evolution derivative along the trajectories of the system $u_t = K(u)$. The function ρ is said to be the conserved density and θ is said to be the density current. The current $(D\,f, D_t\,f)$ is conserved for any system and it is called the trivial conserved current. A trivial current may be added to any conserved current and result will be the conserved current again.

Equation (4) can be investigated with the help of the Euler operator E

$$E_\alpha = (-D)^n\frac{\partial}{\partial u_n^\alpha} \tag{5}$$

that possesses an important property: $E\,f = 0$ if and only if $f = D(F)$ [4]. Applying the operator E to equation (4), we obtain the following equation for the conserved densities

$$E\,D_t\,\rho = 0. \tag{6}$$

The package JET contains the procedure **EU** that performs the computation according to formula (5). To obtain the left hand side of equation (6) you must call

```
> EU(ED(rho),k);
```

where $k = 1, 2, \ldots, m$, and m is the number of the dependent variable (or number of entry of the list `vard`). These equations can be solved by the same method as it was demonstrated above for Lie–Bäcklund symmetries.

Another way of the computation of the conserved currents is given by the procedure **pot** (potential) that calculates a function f if the function $\phi = D\,f$ is known: `pot`(ϕ)=f. Hence if $(\rho,\ \theta)$ is a conserved current then θ =pot(ED(ρ)). Let us consider the zero order conserved densities for the KdV equation:

```
> pot(ED(u0)),  pot(ED(u0^2));
```

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

Now let us take the expression $\rho = u0^3$ that is not a conserved density, of course:

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

$$Break,\ ord(rm) = [1]$$

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

This result means that ED($u0^3$)=DF(th) + rm, where rm=$3\,u1^3$. rm is the global name for a remainder when the pot is called.

When the zero order conserved density ρ exists, one can perform the following contact transformation $(t,\, x,\, u(t,\, x)) \to (t,\, y,\, U(t,\, y))$:

$$d\,y = \rho\,d\,x + \theta\,d\,t, \qquad U(t,\, y) = u(t,\, x). \tag{7}$$

This transformation is analogous to the transformation between Lagrange and Euler variables in the fluid dynamics. Therefore the procedure executing transformation (7) was called L_E. Let us transform the KdV equation, for example:

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

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

Here the second argument of L_E must be a list of new dependent variables. And besides the procedure L_E may be called with three arguments: L_E(ρ, θ,VARD), where (ρ, θ) is a conserved current and VARD is the list of new dependent variables. In this case the procedure works slightly faster because θ is entered but is not evaluated.

4 Canonical conserved densities

In the paper [5] the necessary conditions of the complete integrability for evolution systems were introduced. Later these conditions were explained and generalized in [6] for a wide class of systems with two independent variables. Let the system

$$F(u) = 0 \tag{8}$$

be transformable 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 and + is the symbol of the formal conjugation. Then let us consider the following system

$$\Phi(D_t + \theta,\, D_x + \rho)\psi = 0, \qquad (c, \psi) = 1, \tag{9}$$

where (c, ψ) is the Euclidean scalar product and c is an arbitrary constant vector. The main result is as follows.

If system (8) is integrable by the inverse spectral transform method then system (9) possesses a formal solution of the following form:

$$\rho = \sum_{i=-n}^{\infty} \rho_i \, k^i, \qquad \theta = \sum_{i=-n}^{\infty} \theta_i \, k^i, \qquad \psi = \sum_{i=0}^{\infty} \psi_i \, k^i, \tag{10}$$

where k is a parameter, $n > 0$, $\rho_{-n} \neq 0$ or $\theta_{-n} \neq 0$ and $(\rho_i, \, \theta_i)$, $i = -n, -n+1, \ldots$ are local or weakly nonlocal conserved currents of system (8).

System (9) and expansions (10) imply a recursion relation for ρ_i, θ_i and ψ_i. Therefore, the continuity equations $D_t \rho_i = D_x \theta_i$ give the constraints for system (8).

Let us consider the example

$$u_t = u_3 + f(u, u_1). \tag{11}$$

A simple calculation gives $F'^{+} = D_t - D^3 + f_0 - D\, f_1$, where $f_0 = \partial f / \partial u_0$, $f_1 = \partial f / \partial u_1$. Hence equation (9) takes the form $[\theta - (D + \rho)^3 + f_0 - (D + \rho)\, f_1]\, 1 = 0$, or

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

Setting

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

we obtain the required recursion formula

$$\begin{aligned} 3\, \rho_{i+2} = \theta_i - 3 \sum_{j=0}^{i+1} \rho_j \, \rho_{i-j+1} - \sum_{j,k=0}^{i} \rho_j \, \rho_k \, \rho_{i-j-k} - D^2(\rho_i) \\ - \frac{3}{2} \, D \left(2\, \rho_{i+1} + \sum_{j=0}^{i} \rho_j \, \rho_{i-j} \right) + (f_0 - D(f_1)) \delta_{i0} - f_1 \, \delta_{i,-1} - f_1 \, \rho_i, \end{aligned} \tag{12}$$

where $i = -2, -1, \ldots$ It is obvious that

$$\rho_0 = 0, \qquad \theta_0 = 0, \qquad \rho_1 = -\frac{1}{3} \, f_1, \quad \ldots$$

The conserved densities of system (8) produced by means of formula (9) are called canonical conserved densities. The canonical conserved densities of the KdV equation defined in (12) can be easily obtained, using the following program:

```
> 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;
> 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)-DF(dif(f,u1)))
>  *DLT(i,0)/3-dif(f,u1)*DLT(i,-1)/3-dif(f,u1)*r(i)/3
> end;
```

Here **DLT** is an 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 $i, j, k \geq n$ and besides $i + j + k = m$. Number of arguments of `SU` may be arbitrary, and arguments of `SU` may be under `DF` or `DN` operators. So expressions of the type `SU(A,DF(B),DN(C,p),n,m)` are admissible. Moreover we assume that the expressions θ_0, $\theta_1, \ldots$ must be saved under the names `th0,th1,...`

For systems of two and more equations canonical densities may consist of dozens of hundreds terms. The evolution derivatives of such long expressions consist of dozens thousands terms. Processing a large expression requires very long time. And, moreover, if the number of addends in an expression is more than 40000 then Maple finishes the computation and informs: *Object is too large*. To solve this problem we apply the procedure **part**. The command `z:=part(F,n)` returns the list `z` with n entries so that each entry contains a part of the expression `F` and `z[1]+z[2]+...+z[n] =F`. Then one can perform the required operations with each element `z[i]` separately and obtain the final result. Another method is based on using the procedure **cho** (CHoose Order). The call `cho(F, 3)` for example, collects and returns those terms from the expression `F` whose orders ≥ 3. As the terms with the greatest order are interesting almost always then the procedure `cho` is very useful.

5 Zero curvature representations

Let us consider the following linear overdetermined system

$$\Psi_x = U\,\Psi, \qquad \Psi_t = V\,\Psi, \tag{13}$$

where Ψ is a column, U and V are the square matrices depending on the jet space coordinates t, x, u_n^α and a parameter λ. System (13) is compatible if and only if the following equation holds

$$U_t - V_x + [\,U,\,V\,] = 0. \tag{14}$$

If equation (13) is satisfied on the solutions manifold of an evolution partial differential system (1) but not identically then it is said that system (1) possesses the zero curvature representation.

Systems (13) and (14) are covariant under the gauge transformation:

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

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

To investigate equation (14) in JET-package you must enter the following commands

```
> depend(U(...), V(...) ): matrices:={U,V}:
> z:=ED(U) - DF(V) + com(U, V);
```

and solve the equation $z = 0$. Here **matrices** is the global name of the set of symbolic matrices names, **com** is the name of a procedure for commutator. Arguments of `com` may be both symbolic matrices (names) and arrays. Procedure `com` knows all properties of commutators. For example,

```
> com(U, 2*U+3*V}, com(V,U), dif(com(U,V),u0);
```

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

Ordering is performed automatically in the alphabetical order. Integration of `com(A,B)` is possible only if `A` and `B` are constants, but it suffices the analysis of equations (14). The procedure **Jac** 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), Jac(z1,A,B,B);
```

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

`Jac` searches for the first nested commutator containing the 2nd, 3rd and 4th arguments of `Jac`, transforms it and returns the result. That is why different results are obtained. 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,yes,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 makes `Jac` to begin the search from the second addend. The 5th argument `yes` make `Jac` perform the transformation within the external commutator. The call `Jac(z1,A,B,C,yes)` makes `Jac` to perform the transformation within the external commutator in the first addend and the error message will be returned. The call `Jac(z1,A,B,C,2,yes)` is the mistake as well, both parameters 2 and `yes` will be ignored and the first term will be transformed in this case.

When equation (14) is solved, the next problem is to construct the Lie algebra. Let us consider the KdV equation as an example. After some simple calculations one can obtain the matrices $U = A_1\, u_0 + A_2$ and

$$V = A1\, u_2 - [A1,\, A2]\, u_1 + 3\, A1\, u_0^2 - 1/2\, u_0^2\, [A1,\, [A1,\, A2]] - [A2,\, [A1,\, A2]]\, u_0,$$

where A_i are constant matrices. Moreover the following equations are obtained

$$\begin{aligned} &[A1,\, [A2,\, A3]] + 2\, A3 = 0, \qquad [A1,\, A2] = A3, \\ &[A1,\, [A1,\, A3]] = 0, \qquad [A2,\, [A2,\, A3]] = 0. \end{aligned} \tag{15}$$

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 algebras and the better way is investigation of equations (15) in the spirit of the ideas by H.D. Wahlquist and F.B. Estabrook [7]. There is very useful procedure **struct** for obtaining the closed algebra in this approach . The procedure `struct` constructs the adjoint representation of a Lie algebra and returns the equations for unknown structural constants if the input algebra is not closed. In our example we can use `struct` at once, but it is necessary to enter the basis of the algebra beforehand. Let us assume that $A1$, $A2$ and $A3$ form the basis and enter the commands:

```
> s:={com(A1,A2)=A3}: bas:=[A1,A2,A3]:
> 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:=[[-x1 x6+x3 x4, x3 x1+x2 x6, x1+x5]]

Here the first parameter `bas` is the list of basis elements, the second parameter `s` must be a set of commutation relations and the third parameter of `struct` must be a name x so that *x1, x2, ...*, are free variables. These variables are used in the table of commutators:

$$EQ = \{[\,A1,\,A2\,] = A3, \qquad [\,A1,\,A3\,] = x1\,A1 + x2\,A2 + x3\,A3,$$
$$[\,A2,\,A3\,] = x4\,A1 + x5\,A2 + x6\,A3\}.$$

The names `C, EQ` and `S` are global. The obtained list z contains the left hand sides of the equations

$$-x1\,x6 + x3\,x4 = 0, \qquad x3\,x1 + x2\,x6 = 0, \qquad x1 + x5 = 0.$$

Solving these equations we obtain the closed table of commutators EQ. Setting for example $x1 = -x5 = 2$, $x2 = k$, $x3 = x4 = x6 = 0$, where k is a parameter we obtain the standard algebra $sl(2)$ for the KdV:

```
> EQ;
```

$$\{[\,A1,\,A2\,] = A3, \quad [\,A1,\,A3\,] = 2\,A1 + k\,A2, \quad [\,A2,\,A3\,] = -2\,A2\} \tag{16}$$

This algebra solves all equations (15). Constructing then a representation of the obtained Lie algebra we can find an explicit form of the matrices U and V.

For solving the problems considered in this section the following procedures are useful: **evsub, Cmetric** and **Killing**. The command `evsub(A)` is used for evaluation and simplification of the elements of 2-dimensional array `A`. The call `evsub(s,A)` where `s` is the set of substitutions is used for performing the substitutions into 2-dimensional array `A`. 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 $\langle A, B\rangle$=trace(ad A ad B) for the pair of elements A, B of a Lie algebra.

6 Recursion operators

The recursion operator Λ of evolution system (1) satisfies the following equation

$$[\,D_t - K',\,\Lambda\,] = 0 \tag{17}$$

by definition (see [1–3]). There are two procedures in JET for the computation of the recursion operator.

If you know the zero curvature representation for your system, try call the procedure **triada** that uses the algorithm published in [8, 9]. For example, the KdV equation possesses algebra (16) and we have

```
> triada(U,s);
```

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

Now you should transform this equation (or system in the vector case) to the following form $L\,g = k\,g$. Then $\Lambda = L^+$ [8, 9]. In our example this gives the well-known Lenard operator

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

When the matrices U and V are embedded in the Lie algebra of a small dimension then this approach is acceptable. Otherwise the equation $L(u,k)g = 0$ is too large object. In this case you can try calculate the recursion operator or Noether operators directly. If we set

$$\Lambda = \sum_{i=0}^{n} F_{n-i}(u)D^i + \Sigma(u)D^{-1}\Gamma(u), \tag{18}$$

then equation (17) implies that the columns of Σ are symmetries and the rows of Γ are gradients of conserved densities. That is, Σ satisfies equation (2) and Γ^T satisfies the adjoint equation

$$(D_t + K'^{+})\Gamma^T = 0.$$

The coefficients F_i, Σ and Γ satisfy a cumbersome system that can be obtained with the help of procedure **recursion**. It can be called with one or two input parameters:

```
> sys:=[u3+u0*u1]:
> recursion(0); recursion(1); recursion(1,2);
```

$$(F0\&*K3) - (K3\&*F0)$$

$$(F1\&*K3) - (K3\&*F1) + (F0\&*K2) - (K2\&*F0) + n\,(F0\&*D(K3))$$
$$-3\,(K3\&*D(F0)) - (rsys(3+n)\&*(\Sigma\&*\Gamma)) + (\Sigma\&*(\Gamma\&*rsys(3+n)))$$

$$(F1\&*K3) - (K3\&*F1) + (F0\&*K2) - (K2\&*F0) + n\,(F0\&*D(K3))$$
$$-3\,(K3\&*D(F0))$$

Here $\&*$ is a symbol of the matrix multiplication, $F0$, $F1$ etc., Σ and Γ are exactly the coefficients of operator (18), $K0$, $K1$ etc. are the coefficients of the operator

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

`rsys(i)`$=K_i$ if $0 \le i \le N$ and otherwise `rsys(i)`$=0$. The number N is determined by the list `sys` ($N = 3$ in our example) and n is a nonnegative parameter. The call `recursion(1,2)` means that we assume $n \ge 2$. Then the result is shorter. The number of equations returned by procedure `recursion` is $N + n + 1$. To solve the equations for $F0$, $F1, \ldots$ you must substitute there these matrices with undetermined coefficients and the matrices $K0$, $K1, \ldots$ that one can obtain with help of the procedure `Frechet`. Matrices Σ and Γ must be calculated beforehand. If you solve the first equation and find $F0$, try enter $n = 0$ or $n = 1$ and solve next equations. If such solution does not exist then you can call `recursion(i,2)`, $i = 0, 1, \ldots$ and solve these equations with arbitrary n (but $n \ge 2$). Then you can enter $n = 2$ and so on.

7 Noether operators

Let us consider a pair of operators Θ and J satisfying the following equations

$$(D_t - K')\Theta = \Theta(D_t + K'^{+}), \tag{19}$$

$$(D_t + K'^{+})J = J(D_t - K'). \tag{20}$$

The operator Θ is called a Noether operator and J is called the inverse Noether operator [10, 11]. Of course if Θ satisfies equation (19) then Θ^{-1} satisfies equation (20). But one cannot find Θ^{-1} or J^{-1} explicitly as a rule. 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. Sometimes system (1) admits Noether operator Θ and inverse Noether operator J ($\neq \Theta^{-1}$) then ΘJ is the recursion operator [11].

The most general form of the Noether and inverse Noether operators known today is

$$\Theta = \sum_{i=0}^{n} \theta_{n-i} D^i + A\,D^{-1}B, \tag{21}$$

$$J = \sum_{i=0}^{n} J_{n-i} D^i + G\,D^{-1}H. \tag{22}$$

Here the columns of A and rows of B are Lie–Bäcklund symmetries of system (1); the columns of G and rows of H are gradients of the conserved densities of system (1). It happens that $A = 0$ or $G = 0$ for some systems.

The procedure **Noether** returns the equations for the matrices θ_i, A and B of operator (21). The procedure **INoether** returns the equations for the matrices J_i, G and H of operator (22). Both procedures have the same syntax as the procedure `recursion`: `Noether(m)` or `Noether(m,k)`. Here m is a number of the returned equation, the second parameter k is used if you know that the order of Θ or J is greater than or equal to k.

The Noether operator of an integrable evolution system is an implectic operator and the inverse Noether operator is a symplectic operator as a rule.

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

$$\{f, g, h; \Theta\} + \{g, h, f; \Theta\} + \{h, f, g; \Theta\} = 0. \tag{23}$$

The operator J is called symplectic if it is antisymmetric ($J^{+} = -J$) and the bracket $[f, g, h; J] = \langle f,\, J'[g]h\rangle$ satisfies the Jacobi identity

$$[f, g, h; J] + [g, h, f; J] + [h, f, g; J] = 0. \tag{24}$$

The procedure **implectic** checks the identities $\Theta^{+} = -\Theta$ and (23). The syntax is `implectic(L,n)`. Here L=$[\theta_0, \theta_1, \ldots, \theta_n, A, B]$ is the list of the coefficients of operator (21), the second parameter n is the order of Θ.

The procedure **symplectic** checks the identities $J^{+} = -J$ and (24). The syntax is `symplectic(L,n)`. Here L=$[J_0, J_1, \ldots, J_n, G, H]$ is the list of the coefficients of operator (22), the second parameter n is the order of J.

Both procedures `implectic` and `symplectic` return the text information: "Antisymmetry – OK" or "Antisymmetry is not valid, reminder is saved as rm" Then these procedures simplify the left hand sides of identities (23) and (24) as much as possible and return them as the results.

Conclusion

We are going to prepare the help file for our package and place it in Internet.

References

[1] Ibragimov N.H., Transformation Groups in Mathematical Physics, Moscow, Nauka, 1983.

[2] Olver P.J., Applications of Lie Groups to Differential Equations, New York, Springer-Verlag, 1986.

[3] CRC Handbook of Lie Group Analysis of Differential Equations, ed. N.H. Ibragimov, London, Tokyo, CRC Press, 1994, 1995, etc.

[4] Galindo A. and Martinez L., *Lett. Math. Phys.*, 1978, V.2, N 5, 385–390.

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

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

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

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

[9] Meshkov A.G., Symmetries and Conservation Laws for Evolution Equations, VINITI, N 1511–85, Moscow, 1985.

[10] Fokas A.S. and Fuchssteiner B., *Lett. Nuovo Cimento*, 1980, V.28, 299.

[11] Fuchssteiner B. and Fokas A.S., *Physica D*, 1981, V.4, 47.