

CDIFF: REDUCE PACKAGES FOR COMPUTATIONS IN GEOMETRY OF PDES

R. VITOLO

ABSTRACT. We describe CDIFF, a symbolic computation package for the geometry of Differential Equations (DEs, for short) and developed by P. Gragert, P.H.M. Kersten, G. Post and G. Roelofs at the University of Twente, The Netherlands.

The package is part of the official REDUCE distribution at Sourceforge [1], but it is also distributed on the Geometry of Differential Equations web site <http://gdeq.org> (GDEQ for short).

We start from an installation guide for Linux and Windows. Then we focus on concrete usage recipes for the computation of higher symmetries, conservation laws, Hamiltonian and recursion operators for polynomial differential equations. All programs discussed here are shipped together with this manual and can be found at the GDEQ website. The mathematical theory on which computations are based can be found in refs. [11, 12].

1. INTRODUCTION

This brief guide refers to using CDIFF, a set of symbolic computation packages devoted to computations in the geometry of DEs and developed by P. Gragert, P.H.M. Kersten, G. Post and G. Roelofs at the University of Twente, The Netherlands.

Initially, the development of the CDIFF packages was started by Gragert and Kersten for symmetry computations in DEs, then they have been partly rewritten and extended by Roelofs and Post. The CDIFF packages consist of 3 program files plus a utility file; only the main three files are documented [7, 8, 9]. The CDIFF packages, as well as a copy of the documentation (including this manual) and several example programs, can be found both at Sourceforge in the sources of REDUCE [1] and in the Geometry of Differential Equations (GDEQ for short) web site [2]. The name of the packages, CDIFF, comes from the fact that the package is aimed at defining differential operators in total derivatives and do computations involving them. Such operators are called *C-differential operators* (see [11]).

The main motivation for writing this manual was that REDUCE 3.8 recently became free software, and can be downloaded here [1]. For this reason, we are able to make our computations accessible to a wider public, also thanks to the inclusion of CDIFF in the official REDUCE distribution. The readers are warmly invited to send questions, comments, etc., both on the computations and on the technical aspects of installation and configuration of REDUCE, to the author of the present manual.

Date: 2010 July 22.

2000 Mathematics Subject Classification. 37K05.

Key words and phrases. REDUCE, Hamiltonian operators, generalized symmetries, higher symmetries, conservation laws, nonlocal variables.

Acknowledgements. My warmest thanks are for Paul H.M. Kersten, who explained to me how to use the CDIFF packages for several computations of interest in the Geometry of Differential Equations. I also would like to thank I.S. Krasil'shchik and A.M. Verbovetsky for constant support and stimulating discussions which led me to write this text.

2. INSTALLATION

In order to use CDIFF packages you should be able to write REDUCE programs using CDIFF and run them in the REDUCE interactive shell. So, you need two programs: REDUCE and a text editor which is preferably oriented to program development.

2.1. Installation of REDUCE. In order to install REDUCE it is enough to download from here [1] a precompiled binary for your operating system (*e.g.*, 32-bit or 64-bit Debian-based Linux like Debian itself or Ubuntu, 32-bit Windows) and uncompress it in your computer in a location of your choice.

For the moment CDIFF packages have been tested under Linux (both 32bit and 64bit) and Windows XP; please contact the author of this guide if you tested the packages with positive results under Mac or other versions of Windows like Vista or Windows 7.

A REDUCE program using CDIFF packages can be written with any text editor; it is customary to use the extension `.red` for REDUCE programs, like `program.red`. If you wish to run your program, just run the REDUCE executable. After starting REDUCE, you would see something like

```
Reduce (Free CSL version), 14-Apr-09 ...
```

```
1:
```

Assume that you wrote the program `program.red`, using CDIFF macros. The program must contain the line

```
load_package cdiff;
```

just before the first macro from the package CDIFF. Then you may run the program with the following instruction at the REDUCE prompt:

```
1: in "program.red";
```

Note that in what follows we will omit the prompt in REDUCE commands.

Of course, if the program file `program.red` is *not* in the place where the REDUCE executable is, you should indicate the full path of the program, and this depends on your system. Remember that each time you run REDUCE the path at the REDUCE prompt is always the path of the REDUCE executable, unless you use an absolute path as above. However, if you start REDUCE with the graphical interface (see below) you can always use the leftmost menu item `File>Open...` in order to avoid to write down the whole absolute path.

2.2. Installation of an editor for writing REDUCE programs. Now, let us deal with the problem of editing REDUCE programs.

Generally speaking, any text editor can be used to write a REDUCE program. A more suitable choice is an editor for programming languages. Such editors exist in Linux and Windows, a list can be found here [4].

A suggested text editor in Windows is **notepad++**. This editor is easy to install, it has support for many programming languages (but *not* for REDUCE!), and has a GPL free license, see [3]. Similar tools in Linux are **kwrite** and **gedit**.

However, the only IDE (Integrated Development Environment) for developing programs and running them inside the editor itself exists for the great text editor **emacs**, which runs in all operating systems, and in particular Linux and Windows. We stress that an IDE makes the developing-running-debugging cycle much faster because every step is performed in the same environment.

Installation of **emacs** in Linux is quite smooth, although it depends on the Linux distribution; usually it is enough to select the package **emacs** in your favourite package management tool, like **aptitude**, **synaptic**, or **kpackage**. In order to install **emacs** on Windows one has to work a little bit more. See here [5] for more information. Assuming that **emacs** it is installed and working, the REDUCE IDE for **emacs** can be found here [10]. We refer to their guide for the installation (the procedure is the same for both Linux and Windows). I tested the IDE with emacs 23.1 under Debian-based Linux systems (Debian Etch and Squeeze 32-bit and 64-bit, Ubuntu 10.04 64-bit) and Windows XP and it works fine for me.

Suppose you have **emacs** and its REDUCE IDE installed, then there is a last configuration step that will make **emacs** and REDUCE work together. Namely, when opening for the first time a REDUCE program file with **emacs**, go to the REDUCE>Customize... menu item and locate the ‘REDUCE run Program’ item. This item contains the command which is issued by **emacs** from the REDUCE IDE when the menu item Run REDUCE>Run REDUCE is selected. Change the command to:

- under Linux (user and location as above):
`reduce -w`
- under Windows (user and locations as above):
`reduce.exe`

This setting will run REDUCE inside **emacs**. If you prefer the (slower) graphical interface to REDUCE, remove ‘-w’. Note that the graphical interface will produce L^AT_EX output, making it much more readable. This behaviour can be turned off in the graphical interface by issuing the command `off fancy;`.

3. WORKING WITH CDIFF

All programs that we will discuss in this manual can be found inside the subfolder **examples** in the folder which contains this manual. There are some conventions that I adopted on writing programs which use CDIFF.

- Program files have the extension **.red**. This will load automatically the reduce-ide mode in emacs (provided you made the installation steps described in the reduce-ide guides).

- Program files have the following names:

`equationname_typeofcomputation_version.red`

where `equationname` stands for the shortened name of the equation (*e.g.* Korteweg–de Vries is always indicated by KdV), `typeofcomputation` stands for the type of geometric object which is computed with the given file, for example symmetries, Hamiltonian operators, etc., `version` is a version number.

- More specific information, like the date and more details on the computation done in each version, are included as comment lines at the very beginning of each file.

If you use a generic editor, as soon as you are finished writing a program, you may run it from within REDUCE by following the instructions in the previous section.

In `emacs` with REDUCE IDE it is easier: issuing the command `M-x run-reduce` (or choosing the menu item `Run REDUCE>Run REDUCE`) will split the window in two halves and start REDUCE in the bottom half. If you are running PSL REDUCE you must first issue the command `lisp set_bndstk_size 1000000`; from within REDUCE, in order to avoid memory problems. If you are running CSL REDUCE there is no need of that instruction. Then you may load the program file that you were editing (suppose that its name is `program.red`) by issuing `in "program.red"`; at the REDUCE prompt. In fact, `emacs` lets REDUCE assume as its working directory the directory of the file that you were editing.

Results of a computation consist of the values of one or more unknown. Suppose that the unknown's name is `sym`, and assume that, after a computation, you wish to save the values of `sym`, possibly for future use from within REDUCE. Issue the following REDUCE commands (of course, after you finish your computations!):

```
off nat;
out file_res.red;
sym:=sym;
shut file_res.red;
```

The above commands will write the content of `sym` into a file whose name is `file_res.red`, where `file` stands for a filename which follows the above convention. The command `off nat`; is needed in order to save the variable in a format which could be imported in future REDUCE sessions. If you wish to translate your results in \LaTeX , see the package `rlfi` and its own documentation.

4. COMPUTING WITH CDIFF

Here we describe some examples of computations with CDIFF. The parts of examples which are shared between all examples are described only once. We stress that all computations presented in this document are included in the official REDUCE distribution and can be also downloaded at the GDEQ website [2]. The examples can be run with REDUCE by typing `in "program.red"`; at the REDUCE prompt, as explained above.

Remark. The mathematical theories on which the computations are based can be found in [11, 12].

4.1. Higher symmetries. In this section we show the computation of (some) higher symmetries of Burgers' equation $B = u_t - u_{xx} + 2uu_x = 0$. The corresponding file is `Burg_hsym_1.red` and the results of the computation are in `Burg_hsym_1.res.red`.

The idea underlying this computation is that one can use the scale symmetries of Burgers' equation to assign "gradings" to each variable appearing in the equation. As a consequence, one could try different ansatz for symmetries with polynomial generating function. For example, it is possible to require that they are sum of monomials of given degrees. This ansatz yields a simplification of the equations for symmetries, because it is possible to solve them in a "graded" way, *i.e.*, it is possible to split them into several equations made by the homogeneous components of the equation for symmetries with respect to gradings.

In particular, Burgers' equation translates into the following dimensional equation:

$$[u_t] = [u_{xx}], \quad [u_{xx} = 2uu_x].$$

By the rules $[u_z] = [u] - [z]$ and $[uv] = [u] + [v]$, and choosing $[x] = -1$, we have $[u] = 1$ and $[t] = -2$. This will be used to generate the list of homogeneous monomials of given grading to be used in the ansatz about the structure of the generating function of the symmetries.

The following instructions initialize the total derivatives. The first string is the name of the vector field, the second item is the list of even variables (note that `u1, u2, ...` are u_x, u_{xx}, \dots), the third item is the list of odd (non-commuting) variables ('ext' stands for 'external' like in external (wedge) product). Note that in this example odd variables are not strictly needed, but it is better to insert some of them for syntax reasons.

```
super_vectorfield(ddx,{x,t,u,u1,u2,u3,u4,u5,u6,u7,
u8,u9,u10,u11,u12,u13,u14,u15,u16,u17},
{ext 1,ext 2,ext 3,ext 4,ext 5,ext 6,ext 7,ext 8,ext 9,ext 10,ext
11,ext 12,ext 13,ext 14,ext 15,ext 16,ext 17,ext 18,ext 19,ext 20,ext
21,ext 22,ext 23,ext 24,ext 25,ext 26,ext 27,ext 28,ext 29,ext 30,
ext 31,ext 32,ext 33,ext 34,ext 35,ext 36,ext 37,ext 38,ext 39,ext 40,
ext 41,ext 42,ext 43,ext 44,ext 45,ext 46,ext 47,ext 48,ext 49,ext 50,
ext 51,ext 52,ext 53,ext 54,ext 55,ext 56,ext 57,ext 58,ext 59,ext 60,
ext 61,ext 62,ext 63,ext 64,ext 65,ext 66,ext 67,ext 68,ext 69,ext 70,
ext 71,ext 72,ext 73,ext 74,ext 75,ext 76,ext 77,ext 78,ext 79,ext 80
});

super_vectorfield(ddt,{x,t,u,u1,u2,u3,u4,u5,u6,u7,
u8,u9,u10,u11,u12,u13,u14,u15,u16,u17},
{ext 1,ext 2,ext 3,ext 4,ext 5,ext 6,ext 7,ext 8,ext 9,ext 10,ext
11,ext 12,ext 13,ext 14,ext 15,ext 16,ext 17,ext 18,ext 19,ext 20,ext
21,ext 22,ext 23,ext 24,ext 25,ext 26,ext 27,ext 28,ext 29,ext 30,
ext 31,ext 32,ext 33,ext 34,ext 35,ext 36,ext 37,ext 38,ext 39,ext 40,
ext 41,ext 42,ext 43,ext 44,ext 45,ext 46,ext 47,ext 48,ext 49,ext 50,
ext 51,ext 52,ext 53,ext 54,ext 55,ext 56,ext 57,ext 58,ext 59,ext 60,
ext 61,ext 62,ext 63,ext 64,ext 65,ext 66,ext 67,ext 68,ext 69,ext 70,
ext 71,ext 72,ext 73,ext 74,ext 75,ext 76,ext 77,ext 78,ext 79,ext 80
});
```

Specification of the vectorfield `ddx`. The meaning of the first index is the parity of variables. In particular here we have just even variables. The second index parametrizes the second item (list) in the `super_vectorfield` declaration. More precisely, `ddx(0,1)` stands for $\partial/\partial x$, `ddx(0,2)` stands for $\partial/\partial t$, `ddx(0,3)` stands for $\partial/\partial u$, `ddx(0,4)` stands for $\partial/\partial u_x$, ..., and all coordinates x, t, u_x, \dots , are treated as even coordinates. Note that ‘\$’ suppresses the output.

```
ddx(0,1):=1$
ddx(0,2):=0$
ddx(0,3):=u1$
ddx(0,4):=u2$
ddx(0,5):=u3$
ddx(0,6):=u4$
ddx(0,7):=u5$
ddx(0,8):=u6$
ddx(0,9):=u7$
ddx(0,10):=u8$
ddx(0,11):=u9$
ddx(0,12):=u10$
ddx(0,13):=u11$
ddx(0,14):=u12$
ddx(0,15):=u13$
ddx(0,16):=u14$
ddx(0,17):=u15$
ddx(0,18):=u16$
ddx(0,19):=u17$
ddx(0,20):=letop$
```

The string `letop` is treated as a variable; if it appears during computations it is likely that we went too close to the highest order variables that we defined in the file. This could mean that we need to extend the operators and variable list. In case of large output, one can search in it the string `letop` to check whether errors occurred.

Specification of the vectorfield `ddt`. In the evolutionary case we never have more than one time derivative, other derivatives are $u_{txxx\dots}$.

```
ddt(0,1):=0$
ddt(0,2):=1$
ddt(0,3):=ut$
ddt(0,4):=ut1$
ddt(0,5):=ut2$
ddt(0,6):=ut3$
ddt(0,7):=ut4$
ddt(0,8):=ut5$
ddt(0,9):=ut6$
ddt(0,10):=ut7$
ddt(0,11):=ut8$
ddt(0,12):=ut9$
ddt(0,13):=ut10$
```

```

ddt(0,14):=ut11$
ddt(0,15):=ut12$
ddt(0,16):=ut13$
ddt(0,17):=ut14$
ddt(0,18):=letop$
ddt(0,19):=letop$
sddt(0,20):=letop$

```

We now give the equation in the form one of the derivatives equated to a right-hand side expression. The left-hand side derivative is called *principal*, and the remaining derivatives are called *parametric*¹. For scalar evolutionary equations with two independent variables internal variables are of the type $(t, x, u, u_x, u_{xx}, \dots)$.

```

ut:=u2+2*u*u1;
ut1:=ddx ut;
ut2:=ddx ut1;
ut3:=ddx ut2;
ut4:=ddx ut3;
ut5:=ddx ut4;
ut6:=ddx ut5;
ut7:=ddx ut6;
ut8:=ddx ut7;
ut9:=ddx ut8;
ut10:=ddx ut9;
ut11:=ddx ut10;
ut12:=ddx ut11;
ut13:=ddx ut12;
ut14:=ddx ut13;

```

Test for verifying the commutation of total derivatives. Highest order defined terms may yield some `letop`.

```

for i:=1:17 do write ev(0,i):=ddt(ddx(0,i))-ddx(ddt(0,i));

```

This is the list of variables with respect to their grading, starting from degree *one*.

```

graadlijst:={{u},{u1},{u2},{u3},{u4},{u5},
{u6},{u7},{u8},{u9},{u10},{u11},{u12},{u13},{u14},{u15},{u16},{u17}};

```

This is the list of all monomials of degree 0, 1, 2, ... which can be constructed from the above list of elementary variables with their grading.

```

grd0:={1};
grd1:= mkvarlist1(1,1)$
grd2:= mkvarlist1(2,2)$
grd3:= mkvarlist1(3,3)$
grd4:= mkvarlist1(4,4)$
grd5:= mkvarlist1(5,5)$
grd6:= mkvarlist1(6,6)$
grd7:= mkvarlist1(7,7)$

```

¹This terminology dates back to Riquier, see [13]


```

grd8:= mkvarlist1(8,8)$
grd9:= mkvarlist1(9,9)$
grd10:= mkvarlist1(10,10)$
grd11:= mkvarlist1(11,11)$
grd12:= mkvarlist1(12,12)$
grd13:= mkvarlist1(13,13)$
grd14:= mkvarlist1(14,14)$
grd15:= mkvarlist1(15,15)$
grd16:= mkvarlist1(16,16)$

```

Initialize a counter for the vector of arbitrary constants

```
ctel:=0;
```

We assume a generating function **sym**, *independent of x and t* , of degree ≤ 5 .

```

sym:=
(for each el in grd0 sum (c(ctel:=ctel+1)*el))+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))+
(for each el in grd3 sum (c(ctel:=ctel+1)*el))+
(for each el in grd4 sum (c(ctel:=ctel+1)*el))+
(for each el in grd5 sum (c(ctel:=ctel+1)*el))$

```

This is the equation $\bar{\ell}_B(\text{sym}) = 0$, where $B = 0$ is Burgers' equation and **sym** is the generating function. From now on all equations are arranged in a single vector whose name is **equ**.

```
equ 1:=ddt(sym)-ddx(ddx(sym))-2*u*ddx(sym)-2*u1*sym ;
```

This is the list of variables, to be passed to the equation solver.

```
vars:={x,t,u,u1,u2,u3,u4,u5,u6,u7,u8,u9,u10,u11,
u12,u13,u14,u15,u16,u17};
```

This is the number of initial equation(s)

```
tel:=1;
```

The following procedure uses **multi_coeff** (from the package **tools**). It gets all coefficients of monomials appearing in the initial equation(s). The coefficients are put into the vector **equ** after the initial equations.

```

procedure splitvars i;
begin;
ll:=multi_coeff(equ i,vars);
equ(tel:=tel+1):=first ll;
ll:=rest ll;
for each el in ll do equ(tel:=tel+1):=second el;
end;

```

This command initializes the equation solver. It passes

- the equation vector **equ** together with its length **tel** (*i.e.*, the total number of equations);

- the list of variables with respect to which the system *must not* split the equations, *i.e.*, variables with respect to which the unknowns are not polynomial. In this case this list is just `{}`;
- the constants' vector `c`, its length `ctel`, and the number of negative indexes if any; just 0 in our example;
- the vector of free functions `f` that may appear in computations. Note that in `{f,0,0}` the second 0 stands for the length of the vector of free functions. In this example there are no free functions, but the command needs the presence of at least a dummy argument, `f` in this case. There is also a last zero which is the negative length of the vector `f`, just as for constants.

```
initialize_equations(equ,tel,{},{c,ctel,0},{f,0,0});
```

Run the procedure `splitvars` in order to obtain equations on coefficients of each monomial.

```
splitvars 1;
```

Next command tells the solver the total number of equations obtained after running `splitvars`.

```
pte tel;
```

It is worth to write down the equations for the coefficients.

```
for i:=2:tel do write equ i;
```

This command solves the equations for the coefficients. Note that we have to skip the initial equations!

```
for i:=2:te do es i;
;end;
```

In the folder `computations/NewTests/Higher_symmetries` it is possible to find the following files:

Burg_hsym_1.red: The above file, together with its results file.

KdV_hsym_1.red: Higher symmetries of KdV, with the ansatz: $\deg(\text{sym}) \leq 5$.

KdV_hsym_2.red: Higher symmetries of KdV, with the ansatz:

$$\text{sym} = x^*(\text{something of degree 3}) + t^*(\text{something of degree 5}) \\ + (\text{something of degree 2}).$$

This yields scale symmetries.

KdV_hsym_3.red: Higher symmetries of KdV, with the ansatz:

$$\text{sym} = x^*(\text{something of degree 1}) + t^*(\text{something of degree 3}) \\ + (\text{something of degree 0}).$$

This yields Galilean boosts.

4.2. Local conservation laws. In this section we will find (some) local conservation laws for the KdV equation $F = u_t - u_{xxx} + uu_x = 0$. Concretely, we have to find non-trivial 1-forms $f = f_x dx + f_t dt$ on $F = 0$ such that $\bar{d}f = 0$ on $F = 0$. “Triviality” of conservation laws is a delicate matter, for which we invite the reader to have a look in [11].

The files containing this example is `KdV_loc-cl.1.red`, `KdV_loc-cl.2.red` and the corresponding results files.

We make use of `ddx` and `ddt`, which in the even part are the same as in the previous example (subsection 4.1). After defining the total derivatives we prepare the list of graded variables (recall that in KdV u is of degree 2):

```
graadlijst:={ {}, {u}, {u1}, {u2}, {u3}, {u4}, {u5},
{u6}, {u7}, {u8}, {u9}, {u10}, {u11}, {u12}, {u13}, {u14}, {u15}, {u16}, {u17} };
```

We make the ansatz

```
fx:=
(for each el in grd0 sum (c(ctel:=ctel+1)*el))+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))+
(for each el in grd3 sum (c(ctel:=ctel+1)*el))$
ft:=
(for each el in grd2 sum (c(ctel:=ctel+1)*el))+
(for each el in grd3 sum (c(ctel:=ctel+1)*el))+
(for each el in grd4 sum (c(ctel:=ctel+1)*el))+
(for each el in grd5 sum (c(ctel:=ctel+1)*el))$
```

for the components of the conservation law. We have to solve the equation

```
equ 1:=ddt(fx)-ddx(ft);
```

the fact that `ddx` and `ddt` are expressed in internal coordinates on the equation means that the objects that we consider are already restricted to the equation.

We shall split the equation in its graded summands with the procedure `splitvars`, then solve it

```
initialize_equations(equ,tel,{}, {c,ctel,0}, {f,0,0});
splitvars 1;
pte tel;
for i:=2:tel do es i;
end;
```

As a result we get

```
fx := c(3)*u1 + c(2)*u + c(1)$
ft := (2*c(3)*u*u1 + 2*c(3)*u3 + c(2)*u**2 + 2*c(2)*u2)/2$
```

Unfortunately it is clear that the conservation law corresponding to $c(3)$ is trivial, because it is the total x -derivative of F ; its restriction on the infinite prolongation of the KdV is zero. Here this fact is evident; how to get rid of less evident trivialities by an ‘automatic’ mechanism? We considered this problem in the file `KdV_loc-cl_2.red`, where we solved the equation

```
equ 1:=fx-ddx(f0);
equ 2:=ft-ddt(f0);
```

after having loaded the values `fx` and `ft` found by the previous program. We make the following ansatz on `f0`:

```
f0:=
(for each el in grd0 sum (cc(cctel:=cctel+1)*el))+
(for each el in grd1 sum (cc(cctel:=cctel+1)*el))+
(for each el in grd2 sum (cc(cctel:=cctel+1)*el))+
```

```
(for each el in grd3 sum (cc(cctel:=cctel+1)*el))$
```

Note that this gives a grading which is compatible with the gradings of `fx` and `ft`. After solving the system

```
initialize_equations(equ,tel,{},{cc,cctel,0},{f,0,0});
for i:=1:2 do begin splitvars i;end;
pte tel;
for i:=3:tel do es i;
end;
```

issuing the commands

```
fxnontriv := fx-ddx(f0);
ftnontriv := ft-ddt(f0);
```

we obtain

```
fxnontriv := c(2)*u + c(1)$
ftnontriv := (c(2)*(u**2 + 2*u2))/2$
```

This mechanism can be easily generalized to situations in which the conservation laws which are found by the program are difficult to treat by pen and paper.

4.3. Local Hamiltonian operators. In this section we will find local Hamiltonian operators for the KdV equation $u_t = u_{xxx} + uu_x$. Concretely, we have to solve $\bar{\ell}_{KdV}(\mathbf{phi}) = 0$ over the equation

$$\begin{cases} u_t = u_{xxx} + uu_x \\ p_t = p_{xxx} + up_x \end{cases}$$

or, in geometric terminology, find the shadows of symmetries on the ℓ^* -covering of the KdV equation. The reference paper for this type of computations is [12].

The file containing this example is `KdV_Ham_1.red`.

We make use of `ddx` and `ddt`, which in the even part are the same as in the previous example (subsection 4.1). We stress that the linearization $\bar{\ell}_{KdV}(\mathbf{phi}) = 0$ is the equation `ddt(phi)-u*ddx(phi)-u1*phi-ddx(ddx(ddx(phi)))=0`

but the total derivatives are lifted to the ℓ^* covering, hence they must contain also derivatives with respect to p 's. This will be achieved by treating p variables as odd and introducing the odd parts of `ddx` and `ddt`,

```
ddx(1,1):=0$
ddx(1,2):=0$
ddx(1,3):=ext 4$
ddx(1,4):=ext 5$
ddx(1,5):=ext 6$
ddx(1,6):=ext 7$
ddx(1,7):=ext 8$
ddx(1,8):=ext 9$
ddx(1,9):=ext 10$
ddx(1,10):=ext 11$
ddx(1,11):=ext 12$
ddx(1,12):=ext 13$
ddx(1,13):=ext 14$
```

```

ddx(1,14):=ext 15$
ddx(1,15):=ext 16$
ddx(1,16):=ext 17$
ddx(1,17):=ext 18$
ddx(1,18):=ext 19$
ddx(1,19):=ext 20$
ddx(1,20):=letop$

```

In the above definition the first index ‘1’ says that we are dealing with odd variables, `ext` indicates anticommuting variables. Here, `ext 3` is p_0 , `ext 4` is p_x , `ext 5` is p_{xx} , ...so `ddx(1,3):=ext 4` indicates $p_x \partial / \partial p$, etc..

Now, remembering that the additional equation is again evolutionary, we can get rid of p_t by letting it be equal to `ext 6 + u*ext 4`, as follows:

```

ddt(1,1):=0$
ddt(1,2):=0$
ddt(1,3):=ext 6 + u*ext 4$
ddt(1,4):=ddx(ddt(1,3))$
ddt(1,5):=ddx(ddt(1,4))$
ddt(1,6):=ddx(ddt(1,5))$
ddt(1,7):=ddx(ddt(1,6))$
ddt(1,8):=ddx(ddt(1,7))$
ddt(1,9):=ddx(ddt(1,8))$
ddt(1,10):=ddx(ddt(1,9))$
ddt(1,11):=ddx(ddt(1,10))$
ddt(1,12):=ddx(ddt(1,11))$
ddt(1,13):=ddx(ddt(1,12))$
ddt(1,14):=ddx(ddt(1,13))$
ddt(1,15):=ddx(ddt(1,14))$
ddt(1,16):=ddx(ddt(1,15))$
ddt(1,17):=ddx(ddt(1,16))$
ddt(1,18):=letop$
ddt(1,19):=letop$
ddt(1,20):=letop$

```

Let us make the following ansatz about the Hamiltonian operators:

```

phi:=
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd3 sum (c(ctel:=ctel+1)*el))*ext 3+

(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 4+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 4+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 4+

(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 5+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 5+

```

```
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 6
$
```

Note that we are looking for generating functions of shadows which are *linear* with respect to p 's. Moreover, having set $[p] = -2$ we will look for solutions of maximal possible degree $+1$.

After having set

```
equ 1:=ddt(phi)-u*ddx(phi)-u1*phi-ddx(ddx(ddx(phi)));
vars:={x,t,u,u1,u2,u3,u4,u5,u6,u7,u8,u9,u10,u11,u12,u13,u14,u15,u16,u17};
tel:=1;
```

we define the procedures `splitvars` as in subsection 4.1 and `splitext` as follows:

```
procedure splitext i;
begin;
ll:=operator_coeff(equ i,ext);
equ(tel:=tel+1):=first ll;
ll:=rest ll;
for each el in ll do equ(tel:=tel+1):=second el;
end;
```

Then we initialize the equations:

```
initialize_equations(equ,tel,{},{c,ctel,0},{f,0,0});
do splitext
splitext 1;
then splitvars
tel1:=tel;
for i:=2:tel1 do begin splitvars i;equ i:=0;end;
```

Now we are ready to solve all equations:

```
pte tel;
for i:=2:tel do write equ i:=equ i;
pause;
for i:=2:tel do es i;
end;
```

Note that we want *all* equations to be solved!

The results are the two well-known Hamiltonian operators for the KdV:

```
phi := c(4)*ext(4) + 3*c(3)*ext(6) + 2*c(3)*ext(4)*u + c(3)*ext(3)*u1$
```

Of course, the results correspond to the operators

$$\begin{aligned} \text{ext}(4) &\rightarrow D_x, \\ 3*c(3)*\text{ext}(6) + 2*c(3)*\text{ext}(4)*u + c(3)*\text{ext}(3)*u_1 &\rightarrow 3D_{xxx} + 2uD_x + u_x \end{aligned}$$

Note that each operator is multiplied by one arbitrary real constant, $c(4)$ and $c(3)$.

4.4. Non-local Hamiltonian operators. In this section we will show an experimental way to find nonlocal Hamiltonian operators for the KdV equation. The word ‘experimental’ comes from the lack of a consistent mathematical theory. The result of the computation (without the details below) has been published in [12].

We have to solve equations of the type $\text{ddx}(\mathbf{f}\mathbf{t})-\text{ddt}(\mathbf{f}\mathbf{x})$ as in 4.2. The main difference is that we will attempt a solution on the ℓ^* -covering (see Subsection 4.3). For this reason, first of all we have to determine covering variables with the usual mechanism of introducing them through conservation laws, this time on the ℓ^* -covering.

As a first step, let us compute conservation laws on the ℓ^* -covering whose components are linear in the p ’s. This computation can be found in the file `KdV_nloc-cl_1.red` and related results file. When specifying odd variables in `ddx` and `ddt`, we have something like

```

ddx(1,1):=0$
ddx(1,2):=0$
ddx(1,3):=ext 4$
ddx(1,4):=ext 5$
ddx(1,5):=ext 6$
ddx(1,6):=ext 7$
ddx(1,7):=ext 8$
ddx(1,8):=ext 9$
ddx(1,9):=ext 10$
ddx(1,10):=ext 11$
ddx(1,11):=ext 12$
ddx(1,12):=ext 13$
ddx(1,13):=ext 14$
ddx(1,14):=ext 15$
ddx(1,15):=ext 16$
ddx(1,16):=ext 17$
ddx(1,17):=ext 18$
ddx(1,18):=ext 19$
ddx(1,19):=ext 20$
ddx(1,20):=letop$
ddx(1,50):=(t*u1+1)*ext 3$ % degree -2
ddx(1,51):=u1*ext 3$ % degree +1
ddx(1,52):=(u*u1+u3)*ext 3$ % degree +3
and
ddt(1,1):=0$
ddt(1,2):=0$
ddt(1,3):=ext 6 + u*ext 4$
ddt(1,4):=ddx(ddt(1,3))$
ddt(1,5):=ddx(ddt(1,4))$
ddt(1,6):=ddx(ddt(1,5))$
ddt(1,7):=ddx(ddt(1,6))$
ddt(1,8):=ddx(ddt(1,7))$
ddt(1,9):=ddx(ddt(1,8))$

```

```

ddt(1,10):=ddx(ddt(1,9))$
ddt(1,11):=ddx(ddt(1,10))$
ddt(1,12):=ddx(ddt(1,11))$
ddt(1,13):=ddx(ddt(1,12))$
ddt(1,14):=ddx(ddt(1,13))$
ddt(1,15):=ddx(ddt(1,14))$
ddt(1,16):=ddx(ddt(1,15))$
ddt(1,17):=ddx(ddt(1,16))$
ddt(1,18):=letop$
ddt(1,19):=letop$
ddt(1,20):=letop$
ddt(1,50):=f1*ext 3+f2*ext 4+f3*ext 5$
ddt(1,51):=f4*ext 3+f5*ext 4+f6*ext 5$
ddt(1,52):=f7*ext 3+f8*ext 4+f9*ext 5$

```

The variables corresponding to the numbers 50,51,52 here play a dummy role, the coefficients of the corresponding vector are the unknown generating functions of conservation laws on the ℓ^* -covering. More precisely, we look for conservation laws of the form

```

fx= phi*ext 3
ft= f1*ext3+f2*ext4+f3*ext5

```

The ansatz is chosen because, first of all, `ext 4` and `ext 5` can be removed from `fx` by adding a suitable total divergence (trivial conservation law); moreover it can be proved that `phi` is a symmetry of KdV. We can write down the equations

```

equ 1:=ddx(ddt(1,50))-ddt(ddx(1,50));
equ 2:=ddx(ddt(1,51))-ddt(ddx(1,51));
equ 3:=ddx(ddt(1,52))-ddt(ddx(1,52));

```

However, the above choices make use of a symmetry which contains ‘`t`’ in the generator. This would make automatic computations more tricky, but still possible. In this case the solution of `equ 1` has been found by hand and passed to the program:

```

f3:=t*u1+1$
f1:=u*f3+ddx(ddx(f3))$
f2:=-ddx(f3)$

```

together with the ansatz on the coefficients for the other equations

```

f4:=(for each el in grd5 sum (c(ctel:=ctel+1)*el))$
f5:=(for each el in grd4 sum (c(ctel:=ctel+1)*el))$
f6:=(for each el in grd3 sum (c(ctel:=ctel+1)*el))$

```

```

f7:=(for each el in grd7 sum (c(ctel:=ctel+1)*el))$
f8:=(for each el in grd6 sum (c(ctel:=ctel+1)*el))$
f9:=(for each el in grd5 sum (c(ctel:=ctel+1)*el))$

```

The previous ansatz keep into account the grading of the starting symmetry in `phi*ext 3`. The resulting equations are solved in the usual way (see the example file).

Now, we solve the equation for shadows of nonlocal symmetries in a covering of the ℓ^* -covering. We can choose between three new nonlocal variables `ra,rb,rc`. We are

going to look for non-local Hamiltonian operators depending linearly on one of these variables. Higher non-local Hamiltonian operators could be found by introducing total derivatives of the r 's. As usual, the new variables are specified through the components of the previously found conservation laws according with the rule

$ra_x=fx$, $ra_t=ft$,

and analogously for the others. We define

```
ddx(1,50):=(t*u1+1)*ext 3$ % degree -2
ddx(1,51):=u1*ext 3$ % degree +1
ddx(1,52):=(u*u1+u3)*ext 3$ % degree +3
```

and

```
ddt(1,50) := ext(5)*t*u1 + ext(5) - ext(4)*t*u2 + ext(3)*t*u*u1 +
ext(3)*t*u3 + ext(3)*u$
ddt(1,51) := ext(5)*u1 - ext(4)*u2 + ext(3)*u*u1 + ext(3)*u3$
ddt(1,52) := ext(5)*u*u1 + ext(5)*u3 - ext(4)*u*u2 - ext(4)*u1**2 -
ext(4)*u4 + ext(3)*u**2*u1 + 2*ext(3)*u*u3 + 3*ext(3)*u1*u2 + ext(3)*u5$
```

as it results from the computation of the conservation laws. The following ansatz for the nonlocal Hamiltonian operator comes from the fact that local Hamiltonian operators have gradings -1 and $+1$ when written in terms of p 's. So we are looking for a nonlocal Hamiltonian operator of degree 3.

```
phi:=
(for each el in grd6 sum (c(ctel:=ctel+1)*el))*ext 50+
(for each el in grd3 sum (c(ctel:=ctel+1)*el))*ext 51+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 52+

(for each el in grd5 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd4 sum (c(ctel:=ctel+1)*el))*ext 4+
(for each el in grd3 sum (c(ctel:=ctel+1)*el))*ext 5+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 6+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 7+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 8
$
```

As a solution, we obtain

```
phi := c(1)*(ext(51)*u1 - 9*ext(8) - 12*ext(6)*u - 18*ext(5)*u1 -
4*ext(4)*u**2 - 12*ext(4)*u2 - 4*ext(3)*u*u1 - 3*ext(3)*u3)$
```

where $ext51$ stands for the nonlocal variable rb fulfilling

```
rb_x:=u1*ext 3$
rb_t:=ext(5)*u1 - ext(4)*u2 + ext(3)*u*u1 + ext(3)*u3$
```

Remark. In the file `KdV_nloc-Ham_2.red` it is possible to find another ansatz for a non-local Hamiltonian operator of degree $+5$.

4.5. Computations for systems of PDEs. There is no conceptual difference when computing for systems of PDEs. We will look for Hamiltonian structures for the following Boussinesq equation:

$$(1) \quad \begin{cases} u_t - u_x v - u v_x - \sigma v_{xxx} = 0 \\ v_t - u_x - v v_x = 0 \end{cases}$$

where σ is a constant. This example also shows how to deal with jet spaces with more than one dependent variable. Here gradings can be taken as

$$[t] = -2, \quad [x] = -1, \quad [v] = 1, \quad [u] = 2, \quad [p] = [\frac{\partial}{\partial u}] = -2, \quad [q] = [\frac{\partial}{\partial v}] = -1$$

where p, q are the two coordinates in the space of generating functions of conservation laws.

The linearization of the above system and its adjoint are, respectively

$$\ell_{\text{Bou}} = \begin{pmatrix} D_t - v D_x - v_x & -u_x - u D_x - \sigma D_{xxx} \\ -D_x & D_t - v_x - v D_x \end{pmatrix}, \quad \ell_{\text{Bou}}^* = \begin{pmatrix} -D_t + v D_x & D_x \\ u D_x + \sigma D_{xxx} & -D_t + v D_x \end{pmatrix}$$

and lead to the ℓ_{Bou}^* covering equation

$$\begin{cases} -p_t + v p_x + q_x = 0 \\ u p_x + \sigma p_{xxx} - q_t + v q_x = 0 \\ u_t - u_x v - u v_x - \sigma v_{xxx} = 0 \\ v_t - u_x - v v_x = 0 \end{cases}$$

We have to find shadows of symmetries on the above covering. Total derivatives must be defined as follows:

```
super_vectorfield(ddx,{x,t,u,v,u1,v1,u2,v2,u3,v3,u4,v4,u5,v5,u6,v6,u7,
v7,u8,v8,u9,v9,u10,v10,u11,v11,u12,v12,u13,v13,u14,v14,u15,v15,
u16,v16,u17,v17},
{ext 1,ext 2,ext 3,ext 4,ext 5,ext 6,ext 7,ext 8,ext 9,ext 10,ext
11,ext 12,ext 13,ext 14,ext 15,ext 16,ext 17,ext 18,ext 19,ext 20,ext
21,ext 22,ext 23,ext 24,ext 25,ext 26,ext 27,ext 28,ext 29,ext 30,
ext 31,ext 32,ext 33,ext 34,ext 35,ext 36,ext 37,ext 38,ext 39,ext 40,
ext 41,ext 42,ext 43,ext 44,ext 45,ext 46,ext 47,ext 48,ext 49,ext 50,
ext 51,ext 52,ext 53,ext 54,ext 55,ext 56,ext 57,ext 58,ext 59,ext 60,
ext 61,ext 62,ext 63,ext 64,ext 65,ext 66,ext 67,ext 68,ext 69,ext 70,
ext 71,ext 72,ext 73,ext 74,ext 75,ext 76,ext 77,ext 78,ext 79,ext 80
});
```

```
super_vectorfield(ddt,{x,t,u,v,u1,v1,u2,v2,u3,v3,u4,v4,u5,v5,u6,v6,u7,
v7,u8,v8,u9,v9,u10,v10,u11,v11,u12,v12,u13,v13,u14,v14,u15,v15,
u16,v16,u17,v17},
{ext 1,ext 2,ext 3,ext 4,ext 5,ext 6,ext 7,ext 8,ext 9,ext 10,ext
11,ext 12,ext 13,ext 14,ext 15,ext 16,ext 17,ext 18,ext 19,ext 20,ext
21,ext 22,ext 23,ext 24,ext 25,ext 26,ext 27,ext 28,ext 29,ext 30,
ext 31,ext 32,ext 33,ext 34,ext 35,ext 36,ext 37,ext 38,ext 39,ext 40,
ext 41,ext 42,ext 43,ext 44,ext 45,ext 46,ext 47,ext 48,ext 49,ext 50,
ext 51,ext 52,ext 53,ext 54,ext 55,ext 56,ext 57,ext 58,ext 59,ext 60,
```

```
ext 61,ext 62,ext 63,ext 64,ext 65,ext 66,ext 67,ext 68,ext 69,ext 70,
ext 71,ext 72,ext 73,ext 74,ext 75,ext 76,ext 77,ext 78,ext 79,ext 80
});
```

In the list of coordinates we alternate derivatives of u and derivatives of v . The same must be done for coefficients; for example,

```
ddx(0,1):=1$
ddx(0,2):=0$
ddx(0,3):=u1$
ddx(0,4):=v1$
ddx(0,5):=u2$
ddx(0,6):=v2$
...
```

After specifying the equation

```
ut:=u1*v+u*v1+sig*v3;
vt:=u1+v*v1;
```

we define the (already introduced) time derivatives:

```
ut1:=ddx ut;
ut2:=ddx ut1;
ut3:=ddx ut2;
...
vt1:=ddx vt;
vt2:=ddx vt1;
vt3:=ddx vt2;
...
```

up to the required order (here the order can be stopped at 15). Odd variables p and q must be specified with an appropriate length (here it is OK to stop at `ddx(1,36)`). Recall to replace p_t , q_t with the internal coordinates of the covering:

```
ddt(1,1):=0$
ddt(1,2):=0$
ddt(1,3):=+v*ext 5+ext 6$
ddt(1,4):=u*ext 5+sig*ext 9+v*ext 6$
ddt(1,5):=ddx(ddt(1,3))$
...
```

The list of graded variables:

```
graadlijst:={{v},{u,v1},{u1,v2},{u2,v3},{u3,v4},{u4,v5},
{u5,v6},{u6,v7},{u7,v8},{u8,v9},{u9,v10},{u10,v11},{u11,v12},{u12,v13},
{u13,v14},{u14,v15},{u15,v16},{u16,v17},{u17}};
```

The ansatz for the components of the Hamiltonian operator is

```
phi1:=
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 5+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 4+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 6
```

\$

```
phi2:=
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 5+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 4
$
```

and the equation for shadows of symmetries is

```
equ 1:=ddt(phi1)-v*ddx(phi1)-v1*phi1-u1*phi2-u*ddx(phi2)
-sig*ddx(ddx(ddx(phi2)));
equ 2:=-ddx(phi1)-v*ddx(phi2)-v1*phi2+ddt(phi2);
```

After the usual procedures for decomposing polynomials we obtain the following result:

```
phi1 := c(6)*ext(6)$
phi2 := c(6)*ext(5)$
```

which corresponds to the vector (D_x, D_x) . Extending the ansatz to

```
phi1:=
(for each el in grd3 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 5+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 7+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 9+
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 4+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 6+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 8
$
```

```
phi2:=
(for each el in grd2 sum (c(ctel:=ctel+1)*el))*ext 3+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 5+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 7+
(for each el in grd1 sum (c(ctel:=ctel+1)*el))*ext 4+
(for each el in grd0 sum (c(ctel:=ctel+1)*el))*ext 6
$
```

allows us to find a second (local) Hamiltonian operator

```
phi1 := (c(3)*(2*ext(9)*sig + ext(6)*v + 2*ext(5)*u + ext(3)*u1))/2$
phi2 := (c(3)*(2*ext(6) + ext(5)*v + ext(3)*v1))/2$
```

There is one more higher local Hamiltonian operator, and a whole hierarchy of nonlocal Hamiltonian operators [12].

4.6. Explosion of denominators and how to avoid it. Here we propose the computation of the repeated total derivative of a denominator. This computation fills up the whole memory after some time, and can be used as a kind of speed test for the system. The file is `KdV_denom_1.red`.

After having defined total derivatives on the KdV equation, run the following iteration:

```

phi:=1/(u3+u*u1)$
for i:=1:100 do begin
    phi:=ddx(phi)$
    write i;
end;

```

The program shows the iteration number. At the 18th iteration the program uses about 600MB of RAM, as shown by `top` run from another shell, and 100% of one processor.

There is a simple way to avoid denominator explosion. The file is `KdV_denom_2.red`.

After having defined total derivatives with respect to x (on the KdV equation, for example) consider in the same `ddx` a component with a sufficiently high index **immediately after ‘letop’** (otherwise `super_vectorfield` does not work!), say `ddx(0,21)`, and think of it as being the coefficient to a vector of the type

```
aa21:=1/(u3+u*u1);
```

In this case, its coefficient must be

```
ddx(0,21):=-aa21**2*(u4+u1**2+u*u2)$
```

More particularly, here follows the detailed definition of `ddx`

```

ddx(0,1):=1$
ddx(0,2):=0$
ddx(0,3):=u1$
ddx(0,4):=u2$
ddx(0,5):=u3$
ddx(0,6):=u4$
ddx(0,7):=u5$
ddx(0,8):=u6$
ddx(0,9):=u7$
ddx(0,10):=u8$
ddx(0,11):=u9$
ddx(0,12):=u10$
ddx(0,13):=u11$
ddx(0,14):=u12$
ddx(0,15):=u13$
ddx(0,16):=u14$
ddx(0,17):=u15$
ddx(0,18):=u16$
ddx(0,19):=u17$
ddx(0,20):=letop$
ddx(0,21):=-aa21**2*(u4+u1**2+u*u2)$

```

Now, suppose that we want to compute the 5th total derivative of `phi`. Write the following code:

```

phi:=aa30;
for i:=1:5 do begin
    phi:=ddx(phi)$
    write i;
end;

```

The result is then a polynomial in the additional ‘denominator’ variable

```
phi := aa21**2*( - 120*aa21**4*u**5*u2**5 - 600*aa21**4*u**4*u1**2*u2**4 - 600*
aa21**4*u**4*u2**4*u4 - 1200*aa21**4*u**3*u1**4*u2**3 - 2400*aa21**4*u**3*u1**2*
u2**3*u4 - 1200*aa21**4*u**3*u2**3*u4**2 - 1200*aa21**4*u**2*u1**6*u2**2 - 3600*
aa21**4*u**2*u1**4*u2**2*u4 - 3600*aa21**4*u**2*u1**2*u2**2*u4**2 - 1200*aa21**4
*u**2*u2**2*u4**3 - 600*aa21**4*u*u1**8*u2 - 2400*aa21**4*u*u1**6*u2*u4 - 3600*
aa21**4*u*u1**4*u2*u4**2 - 2400*aa21**4*u*u1**2*u2*u4**3 - 600*aa21**4*u*u2*u4**
4 - 120*aa21**4*u1**10 - 600*aa21**4*u1**8*u4 - 1200*aa21**4*u1**6*u4**2 - 1200*
aa21**4*u1**4*u4**3 - 600*aa21**4*u1**2*u4**4 - 120*aa21**4*u4**5 + 240*aa21**3*
u**4*u2**3*u3 + 720*aa21**3*u**3*u1**2*u2**2*u3 + 720*aa21**3*u**3*u1*u2**4 +
240*aa21**3*u**3*u2**3*u5 + 720*aa21**3*u**3*u2**2*u3*u4 + 720*aa21**3*u**2*u1**
4*u2*u3 + 2160*aa21**3*u**2*u1**3*u2**3 + 720*aa21**3*u**2*u1**2*u2**2*u5 + 1440
*aa21**3*u**2*u1**2*u2*u3*u4 + 2160*aa21**3*u**2*u1*u2**3*u4 + 720*aa21**3*u**2*
u2**2*u4*u5 + 720*aa21**3*u**2*u2*u3*u4**2 + 240*aa21**3*u*u1**6*u3 + 2160*aa21
**3*u*u1**5*u2**2 + 720*aa21**3*u*u1**4*u2*u5 + 720*aa21**3*u*u1**4*u3*u4 + 4320
*aa21**3*u*u1**3*u2**2*u4 + 1440*aa21**3*u*u1**2*u2*u4*u5 + 720*aa21**3*u*u1**2*
u3*u4**2 + 2160*aa21**3*u*u1*u2**2*u4**2 + 720*aa21**3*u*u2*u4**2*u5 + 240*aa21
**3*u*u3*u4**3 + 720*aa21**3*u1**7*u2 + 240*aa21**3*u1**6*u5 + 2160*aa21**3*u1**
5*u2*u4 + 720*aa21**3*u1**4*u4*u5 + 2160*aa21**3*u1**3*u2*u4**2 + 720*aa21**3*u1
**2*u4**2*u5 + 720*aa21**3*u1*u2*u4**3 + 240*aa21**3*u4**3*u5 - 60*aa21**2*u**3*
u2**2*u4 - 90*aa21**2*u**3*u2*u3**2 - 120*aa21**2*u**2*u1**2*u2*u4 - 90*aa21**2*
u**2*u1**2*u3**2 - 780*aa21**2*u**2*u1*u2**2*u3 - 180*aa21**2*u**2*u2**4 - 60*
aa21**2*u**2*u2**2*u6 - 180*aa21**2*u**2*u2*u3*u5 - 120*aa21**2*u**2*u2*u4**2 -
90*aa21**2*u**2*u3**2*u4 - 60*aa21**2*u*u1**4*u4 - 1020*aa21**2*u*u1**3*u2*u3 -
1170*aa21**2*u*u1**2*u2**3 - 120*aa21**2*u*u1**2*u2*u6 - 180*aa21**2*u*u1**2*u3*
u5 - 120*aa21**2*u*u1**2*u4**2 - 540*aa21**2*u*u1*u2**2*u5 - 1020*aa21**2*u*u1*
u2*u3*u4 - 360*aa21**2*u*u2**3*u4 - 120*aa21**2*u*u2*u4*u6 - 90*aa21**2*u*u2*u5
**2 - 180*aa21**2*u*u3*u4*u5 - 60*aa21**2*u*u4**3 - 240*aa21**2*u1**5*u3 - 990*
aa21**2*u1**4*u2**2 - 60*aa21**2*u1**4*u6 - 540*aa21**2*u1**3*u2*u5 - 480*aa21**
2*u1**3*u3*u4 - 1170*aa21**2*u1**2*u2**2*u4 - 120*aa21**2*u1**2*u4*u6 - 90*aa21
**2*u1**2*u5**2 - 540*aa21**2*u1*u2*u4*u5 - 240*aa21**2*u1*u3*u4**2 - 180*aa21**
2*u2**2*u4**2 - 60*aa21**2*u4**2*u6 - 90*aa21**2*u4*u5**2 + 10*aa21*u**2*u2*u5 +
20*aa21*u**2*u3*u4 + 10*aa21*u*u1**2*u5 + 110*aa21*u*u1*u2*u4 + 80*aa21*u*u1*u3
**2 + 160*aa21*u*u2**2*u3 + 10*aa21*u*u2*u7 + 20*aa21*u*u3*u6 + 30*aa21*u*u4*u5
+ 50*aa21*u1**3*u4 + 340*aa21*u1**2*u2*u3 + 10*aa21*u1**2*u7 + 180*aa21*u1*u2**3
+ 60*aa21*u1*u2*u6 + 80*aa21*u1*u3*u5 + 50*aa21*u1*u4**2 + 60*aa21*u2**2*u5 +
100*aa21*u2*u3*u4 + 10*aa21*u4*u7 + 20*aa21*u5*u6 - u*u6 - 6*u1*u5 - 15*u2*u4 -
10*u3**2 - u8)$
```

where the value of aa21 can be replaced back in the expression.

REFERENCES

- [1] Obtaining REDUCE: <http://reduce-algebra.sourceforge.net/>.
- [2] Geometry of Differential Equations web site: <http://gdeq.org>.
- [3] notepad++: <http://notepad-plus.sourceforge.net/>
- [4] List of text editors: http://en.wikipedia.org/wiki/List_of_text_editors

- [5] How to install emacs in Windows: <http://www.cmc.edu/math/alee/emacs/emacs.html>. See also <http://www.gnu.org/software/emacs/windows/ntemacs.html>
- [6] How to install REDUCE in Windows: <http://reduce-algebra.sourceforge.net/windows.html>
- [7] G.H.M. ROELOFS, The SUPER VECTORFIELD package for REDUCE. Version 1.0, Memorandum 1099, Dept. Appl. Math., University of Twente, 1992. Available at <http://gdeq.org>.
- [8] G.H.M. ROELOFS, The INTEGRATOR package for REDUCE. Version 1.0, Memorandum 1100, Dept. Appl. Math., University of Twente, 1992. Available at <http://gdeq.org>.
- [9] G.F. POST, A manual for the package TOOLS 2.1, Memorandum 1331, Dept. Appl. Math., University of Twente, 1996. Available at <http://gdeq.org>.
- [10] REDUCE IDE for emacs: http://centaur.maths.qmul.ac.uk/Emacs/REDUCE_IDE/
- [11] A. V. BOCHAROV, V. N. CHETVERIKOV, S. V. DUZHIN, N. G. KHOR'KOVA, I. S. KRASIL'SHCHIK, A. V. SAMOKHIN, YU. N. TORKHOV, A. M. VERBOVETSKY AND A. M. VINOGRADOV: Symmetries and Conservation Laws for Differential Equations of Mathematical Physics, I. S. Krasil'shchik and A. M. Vinogradov eds., Translations of Math. Monographs **182**, Amer. Math. Soc. (1999).
- [12] P.H.M. KERSTEN, I.S. KRASIL'SHCHIK, A.M. VERBOVETSKY, *Hamiltonian operators and ℓ^* -covering*, Journal of Geometry and Physics **50** (2004), 273–302.
- [13] M. MARVAN, *Sufficient set of integrability conditions of an orthonomic system*. Foundations of Computational Mathematics **9** (2009) 651–674.

R. VITOLO, DEPT. OF MATHEMATICS “E. DE GIORGI”, UNIVERSITÀ DEL SALENTO, VIA PER ARNESANO, 73100 LECCE, ITALY

E-mail address: raffaele.vitolo@unisalento.it