CDE: A REDUCE PACKAGE FOR INTEGRABILITY OF PDES VERSION 1.0

R. VITOLO

Abstract. We describe CDE, a `Reduce` package devoted to differential-geometric computations on Differential Equations (DEs, for short). The package relies on the `Reduce` package CDIFF, whose development was carried out by P. Gragert, P.H.M. Kersten, G. Post and G. Roelofs at the University of Twente, The Netherlands.

The package is included in the official `Reduce` sources in Sourceforge [34] and 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 computations in the geometry of differential equations: higher symmetries, conservation laws, Hamiltonian operators and their Schouten bracket, recursion operators. All programs discussed here are shipped together with this manual and can be found in the `Reduce` sources or at the GDEQ website. The mathematical theory on which computations are based can be found in refs. [12, 19].

Contents

Date: 2014 October 01 – CDE version: 1.0.

2000 *Mathematics Subject Classification.* 37K05.

Key words and phrases. `Reduce`, Hamiltonian operators, symplectic operators, recursion operators, generalized symmetries, higher symmetries, conservation laws, nonlocal variables.

1. Why CDE?

This brief guide refers to using CDE, a Reduce package for differential-geometric computations for DEs. The package aims at defining differential operators in total derivatives and computing with them. Such operators are called $\mathcal{C}$*-differential operators* (see [12]). CDE runs in the computer algebra system Reduce and depends on the Reduce package CDIFF for constructing total derivatives. Recently, Reduce 3.8 became free software, and can be downloaded here [1]. This was an important motivation for making our computations accessible to a wider public, also through this user guide.

The development of the CDIFF package was started by Gragert and Kersten for symmetry computations in DEs. Then CDIFF was partly rewritten and extended by Roelofs and Post. The CDIFF package consists of 4 files, but only the main three files are documented [8, 9, 10]. This software and the related documentation can be found in both the Reduce sources and the Geometry of Differential Equations (GDEQ for short) web site [2].

There are already several software packages that may compute symmetries and conservation laws; many of them run on Mathematica or Maple. Those who run on Reduce were written by M.C. Nucci [28, 29], F. Oliveri (ReLie, [30]), F. Schwartz (SPDE, Reduce official distribution) T. Wolf (APPLYSYM and CONLAW in the official Reduce distribution, [35, 36, 37, 38]).

The development of CDE started from the idea that a computer algebra tool for the investigation of integrability-related structures of PDEs still does not exist in the public domain. We are only aware of a Mathematica package that may find recursion operators under quite restrictive hypotheses [13].

CDE is especially designed for computations of integrability-related structures (such as Hamiltonian, symplectic and recursion operators) for systems of differential equations with an arbitrary number of independent or dependent variables. On the other hand CDE is also capable of (generalized) symmetry and conservation laws computations. The aim of this manual is to introduce the reader to computations of integrability related structures using CDE.

The current version of CDE is able to define equations for Hamiltonian, symplectic and recursion operators. Such equations may be solved by different techniques; one of the possibilities is to use CRACK, a Reduce package for solving overdetermined systems of PDEs [39]. Moreover, CDE is able to compute Schouten brackets to check Hamiltonianity of operators. Very soon CDE will include simplecticity tests, hereditariety tests, computation of linearization (or Fréchet derivatives), adjoints of differential operators.

At the moment the papers [18, 21, 23, 32, 33] have been written by means of CDE, and more research by CDE on integrable systems is in progress.

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 this document.

Acknowledgements. My warmest thanks are for Paul H.M. Kersten, who explained to me how to use the original CDIFF package for several computations of interest in the Geometry of Differential Equations. I also would like to thank J.S. Krasil'shchik and A.M. Verbovetsky for constant support and stimulating discussions which led me to write this text. On the software side, I'd like to thank A.C. Norman for his unfailing support in my computer science 'troubles'. Moreover, I'd like to thank the developers of the `Reduce` mailing list for their prompt replies with solutions to my problems.

2. Installation

In order to use the CDE package it is enough to have a recent version of `Reduce` with both the CDE and the CDIFF packages installed.

We stress that *most of the technical difficulties related to installation and configuration are due to the lack of a `Reduce` installer. This problem should be solved in the near future.*

2.1. **Installation of Reduce.**

In order to install `Reduce` one can download a precompiled binary distribution from here [34]. However, please make sure that the version that you are downloading has been compiled *later* than 1st October 2014, or you will not get CDE in it. If you are ready to recompile `Reduce`, please consider the text [27] for instructions on how to do it in different operating systems.

In Linux you can also download `.deb` packages at the GetDeb website [3].

From now on we will assume that the binary executable of `Reduce` is in the path of the executables of your operating system. A typical location in Linux would be `/usr/local/bin`. You might put a link instead of the binary executable.

A `Reduce` program using CDE package 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), 01-Oct-14 ...

1:
```

At the prompt `1:` write `in "program.red";`. 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. In Linux, assuming that you are the user `user` and your program is in the subdirectory `Reduce/computations` of your home directory, you have something like

```
  in "/home/user/Reduce/computations/program.red";
```

In Windows, assuming that you are the user `user` and your program is in the subdirectory `Reduce\computations` of the Desktop folder, you would write

```
  in "C:\Documents and Settings\user\Desktop\Reduce
\computations\program.red";
```

Remember that each time you run `Reduce` from a command shell, `Reduce` inherits your current path from the shell 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. **Choice of an editor for writing Reduce programs.** Now, let us deal with the problem of writing `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 [5].

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 [4]. Similar tools in Linux are `kwrite` and `gedit`.

The IDE (Integrated Development Environment) of choice of the author 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. Another IDE which has `Reduce` capabilities is GNU TeXmacs, see `http://www.texmacs.org`.

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 [6] for more information. Assuming that `emacs` it is installed and working, the `Reduce` IDE for `emacs` can be found here [11]. We refer to their guide for the installation (the procedure is the same for both Linux and Windows). I tested the IDE with emacs 23.2.1 under Debian-based Linux systems (Debian Etch and Squeeze 32-bit and 64-bit, Ubuntu 11.04 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 LaTeX 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 CDE

All programs that we will discuss in this manual can be found inside the subfolder `examples` of the main directory of CDE. There are some conventions that I adopted on writing programs which use CDE.

- Test files have the following names:

  `equationname_typeofcomputation.red`

  where `equationname` stands for the shortened name of the equation (*e.g.* Korteweg–de Vries is always indicated by `kdv`), and `typeofcomputation` stands for the type of geometric object which is computed with the given file, for example symmetries, Hamiltonian operators, etc.. This string also includes a version number. The extension `.red` will tell `emacs` to load the reduce-ide mode (provided you made the installation steps described in the reduce-ide guides).
- 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. You may use either CSL or PSL `Reduce`: they are two different interpreters of the low-level programming language of `Reduce`, Standard Lisp. `Reduce` shows up the type of interpreter at startup, see 2.1. At the moment, tests by CDE computations show that the CSL interpreter is considerably faster than the PSL interpreter.

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.

NOTE: *at the time of writing the package CDE is being included into the main `Reduce` source tree. So, it is not likely that it will be contained in any old binary distribution. If this is the case, please put your program file in the main CDE directory is, in order to allow `Reduce` to find the main file `cde.red` and then all others.*

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";
on nat;
```

The above commands will write the content of `sym` into the file `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 tri and its own documentation.

Working remotely with Reduce is not difficult and it is highly recommended for big computations that a server can run more efficiently and without interruptions. A method of choice to do this is described by the following steps:

(1) login to the remote server with ssh;
(2) start emacs as a daemon on the server by the command `emacs --daemon` (only from version 23.1!);
(3) run `emacsclient -c file.red`. That program will connect to the emacs daemon and open the requested file.
(4) run Reduce (if you installed the reduce IDE everything is easier, otherwise you should open a shell within emacs and issue the command reduce);
(5) exit emacsclient normally (C-x C-c). This will not kill the daemon, that will keep your computation running until the end.
(6) login again when you wish to check the computation.

In next sections we will describe some examples of computations with CDE. The parts which are shared between all examples are described only once. We stress that all computations presented in this document can be downloaded at the GDEQ website [2], and that they are run in the Reduce environment by typing `in "program.red";` at the Reduce prompt, as explained above. Moreover, all examples can be run at once by the shell script cdiff.sh to test if the system is working properly and results are the same as obtained previously.

Each computation consists of two parts: setting up the jet space and the equation, and solving the problem using suitable ansatz for the unknown functions. We will emphasize this division only in the first example.

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

4. Higher symmetries

In this section we show the computation of (some) higher (or generalized, [31]) symmetries of Burgers'equation $B = u_t - u_{xx} + 2uu_x = 0$.

We provide two ways to solve the equations for higher symmetries. The first possibility is to use dimensional analysis. The idea is that one can use the scale symmetries of Burgers'equation to assign "gradings" to each variable appearing in the equation (in other words, one can use dimensional analisys). As a consequence, one could try different ansatz for symmetries with polynomial generating functions. 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 file for the above computation is `bur_hsy1.red` and the results of the computation are in `results/bur_hsy1_res.red`.

Another possibility to solve the equation for higher symmetries is to use a PDE solver that is especially devoted to overdetermined systems, which is the distinguishing feature of systems coming from the symmetry analysis of PDEs. This approach is described below. The file for the above computation is `bur_hsy2.red` and the results of the computation are in `results/bur_hsy2_res.red`.

4.1. **Setting up the jet space and the differential equation.**

The program that builds total derivatives restricted to the given equation has to be loaded in the beginning:

```
in "cde.red";
```

Then, CDE needs to know the variables, their scale degree and the maximal order of derivatives at which it will compute differential consequences of the given equation. The input is done in this way:

```
indep_var:={x,t}$
dep_var:={u}$
odd_var:={p}$
deg_indep_var:={-1,-2}$
deg_dep_var:={1}$
deg_odd_var:={0}$
total_order:=10$
```

Here

- `indep_var` is the list of independent variables;
- `dep_var` is the list of dependent variables;
- `odd_var` is the list of odd variables (not used in this computation – just a dummy variable);
- `deg_indep_var` is the list of scale degrees of the independent variables;
- `deg_dep_var` is the list of scale degrees of the dependent variables;
- `deg_odd_var` is the list of scale degrees of odd variables (not used in this computation);
- `total_order` is the maximal order of derivatives at which the program will compute differential consequences of the given equation;

Two more parameters can be set for convenience:

```
statename:="bur_hsy1_state.red"$
resname:="bur_hsy1_res.red"$
```

These are the name of the output file for recording the internal state of the program `cde.red`, including the total derivatives, and the name of the file containing results of the computation.

We now give the equation in the form of 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]. Parametric coordinates are coordinates on the equation manifold and its differential consequences, and principal coordinates can be deduced from the differential equation and its differential consequences. For scalar evolutionary equations with two independent variables parametric derivatives are of the type $(u, u_x, u_{xx}, \ldots)$. Note that the system must be in passive orthonomic form; this also means that there will be no nontrivial integrability conditions between parametric derivatives. (Lines beginning with `%` are comments for `Reduce`.)

```
% left-hand side of the differential equation
principal_der:={u_t}$
% right-hand side of the differential equation
de:={u_2x+2*u*u_x}$
% same construction for odd coordinates
principal_odd:={p_t}$
de_odd:={-p_2x+2*u*p_x}$
```

In this computation the odd equation will not have any role, but it must be present even for purely even computations. In order to speed up computations one could set `de_odd` to be zero.

The main routine in `cde.red` is called as follows:

```
cde({indep_var,dep_var,odd_var,total_order},
   {{principal_der,de},{principal_odd,de_odd}})$
```

The function `cde` defines total derivatives truncated at the order `total_order` and restricted on the (even and odd) equation; this means that total derivatives are tangent to the equation manifold. Their coordinate expressions are of the form

$$D_\lambda = \frac{\partial}{\partial x^\lambda} + \sum_{u^i_{\boldsymbol{\sigma}} \text{ parametric}} u^i_{\boldsymbol{\sigma}\lambda} \frac{\partial}{\partial u^i_{\boldsymbol{\sigma}}} + \sum_{p^i_{\boldsymbol{\sigma}} \text{ parametric}} p^i_{\boldsymbol{\sigma}\lambda} \frac{\partial}{\partial p^i_{\boldsymbol{\sigma}}}, \tag{1}$$

where $\boldsymbol{\sigma}$ is a multiindex. It can happen that $u^i_{\boldsymbol{\sigma}\lambda}$ (or $p^i_{\boldsymbol{\sigma}\lambda}$) is principal and must be replaced with differential consequences of the equation. Such differential consequences are called *primary differential consequences*, and are computed; in general they will depend on other, possibly new, differential consequences, and so on. Such newly appearing differential consequences are called *secondary differential consequences*. If the equation is in passive orthonomic form, the system of all differential consequences (up to the maximal order `total_order`) must be solvable in terms of parametric derivatives only. The function *cde* automatically computes all necessary and sufficient differential consequences which are needed to solve the system.

Note that when in total derivatives there is a coefficient of order higher than maximal this is replaced by the string `letop`. If such a string 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, just by increasing the number `total_order`. Later on we will describe a useful test to check the absence of `letop` from a computation.

[1]This terminology dates back to Riquier, see [25]

The output generated by the function `cde` is not a result of the computation, but it can be useful for debugging purposes or for storing intermediate computations to be reused later. It can be saved by the function:

```
save_cde_state(statename)$
```

4.2. **Solving the problem via dimensional analysis.** Higher symmetries of the given equation are functions `sym` depending on parametric coordinates up to some jet space order. We assume that they are graded polynomials of all parametric derivatives. In practice, we generate a linear combination of graded monomials with arbitrary coefficients, then we plug it in the equation of the problem and find conditions on the coefficients that fulfill the equation. To construct a good ansatz, it is required to make several attempts with different gradings, possibly including independent variables, etc.. For this reason, ansatz-constructing functions are especially verbose. In order to use such functions they must be initialized with the following command:

```
cde_grading(deg_indep_var,deg_dep_var,deg_odd_var)$
```

We need one operator `equ` whose components will be the equation of higher symmetries and its consequences. Moreover, we need an operator `c` which will play the role of a vector of constants, indexed by a counter `ctel`:

```
ctel:=0;
operator c,equ;
```

We prepare a list of variables ordered by scale degree:

```
graadlijst:=der_deg_ordering(0,all_parametric_der)$
```

The function `der_deg_ordering` is defined in `cde.red`. It produces the given list using the list `all_parametric_der` of all parametric derivatives of the given equation up to the order `total_order`. The first two parameters can assume the values 0 or 1 and say that we are considering even variables and that the variables are of parametric type.

Then, due to the fact that *all parametric variables have positive scale degree* then we prepare the list `ansatz` of all graded monomials of scale degree from 0 to 5

```
graadmon:=for i:=1:5 collect mkvarlist1(i,i)$
graadmon:={1} . graadmon$
ansatz:=for each el in graadmon join el$
```

More precisely, the command `mkvarlist1(i,i)` produces a list of monomials of degree `i` from the list of graded variables `graadlijst`; the second command adds the zero-degree monomial; and the last command produces a single list of all monomials.

Finally, we assume that the higher symmetry is a graded polynomial obtained from the above monomials (so, it is independent of x and t!)

```
sym:=(for each el in ansatz sum (c(ctel:=ctel+1)*el))$
```

Next, we define the equation $\ell_B(\texttt{sym}) = 0$. Here, ℓ_B stands for the *linearization*, or Fréchet derivative of the function B, where $B = 0$ is Burgers'equation. A function `sym` that fulfills the above equation is an higher symmetry.

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

In the above equation total derivatives with respect to x, t are `ddx`, `ddt`. The list of variables, to be passed to the equation solver:

```
vars:=append(indep_var,all_parametric_der);
```

The number of initial equation(s):

```
tel:=1;
```

Next command initializes the equation solver. It passes

- the equation vector `equ` togeher 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` on the first component of `equ` in order to obtain equations on coefficiens of each monomial.

```
splitvars 1;
```

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

```
put_equations_used tel;
```

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

```
for i:=2:tel do integrate_equation i;
;end;
```

The output is written in the result file by the commands

```
off echo$
off nat$
out <<resname>>;
sym:=sym;
write ";end;";
shut <<resname>>;
on nat$
on echo$
```

The command `off nat` turns off writing in natural notation; results in this form are better only for visualization, not for writing or for input into another computation. The command `<<resname>>` forces the evaluation of the variable `resname` to its string value. The commands `out` and `shut` are for file opening and closing. The command `sym:=sym` is evaluated only on the right-hand side.

One more example file is available; it concerns higher symmetries of the KdV equation. In order to deal with symmetries explicitely depending on x and t it is possible to use `Reduce` and CDE commands in order to have `sym = x*`(something of degree 3) `+ t*`(something of degree 5) + (something of degree 2); this yields scale symmetries. Or we could use `sym = x*`(something of degree 1) `+ t*`(something of degree 3) + (something of degree 0); this yields Galilean boosts.

4.3. **Solving the problem using CRACK.** CRACK is a PDE solver which is devoted mostly to the solution of overdetermined PDE systems [37, 39]. Several mathematical problems have been solved by the help of CRACK, like finding symmetries [36, 38] and conservation laws [35]. The aim of CDE is to provide a tool for computations with total derivatives, but it can be used to compute symmetries too. In this subsection we show how to interface CDE with CRACK in order to find higher (or generalized) symmetries for the Burgers'equation. To do that, after loading CDE and introducing the equation, the operator of linearization should be defined:

```
operator ell_b$
for all sym let ell_b(sym)=ddt(sym)-ddx(ddx(sym))-2*u*ddx(sym)-2*u_x*sym$
```

We introduce the new unknown function '`ansatz`'. We assume that the function depends on parametric variables of order not higher than 3. The variables are selected by the function `selectvars` of CDE as follows:

```
even_vars:=for i:=0:3 join selectvars(0,i,dep_var,all_parametric_der)$
```

In the arguments of `selectvars`, `0` means that we want even variables, `i` stands for the order of variables, `dep_var` stands for the dependent variables to be selected by the command (here we use all dependent variables), `all_parametric_der` is the set of variables where the function will extract the variables with the required properties. In the current example we wish to get all higher symmetries depending on parametric variables of order not higher than 3.

The dependency of `ansatz` from the variables is given with the standard `Reduce` command `depend`:

```
for each el in even_vars do depend(ansatz,el)$
```

The equation to be solved is the equation `ell_b(ansatz)=0`, hence we give the command

```
total_eq:=ell_b(ansatz)$
```

Another command is a safety measure agains the possibility that the application of the operator `ell_b` to the function `ansatz` yields a result which is of order higher than that of the current CDE jet space:

```
check_letop({total_eq})$
```

The above command will issue an error if the list {`total_eq`} depends on the flag variable `letop`. That variable appears when total derivatives shift the order of jet space coordinates our of the current CDE jet space.

The equation `ell_b(ansatz)=0` is polynomial with respect to the variables of order higher than those appearing in `ansatz`. For this reason, its coefficients can be put to zero independently. This is the reason why the PDEs that determine symmetries are overdetermined. To tell this to CRACK, we issue the command

```
split_vars:=diffset(all_parametric_der,even_vars)$
```

The list `split_vars` contains variables which are in the current CDE jet space but *not* in `even_vars`.

Then, we load the package CRACK and get results.

```
load_package crack;
crack_results:=crack(total_eq,{},{ansatz},split_vars);
```

The results are in the variable `crack_results`:

```
{{{},
{ansatz=(2*c_12*u_x + 2*c_13*u*u_x + c_13*u_2x + 6*c_8*u**2*u_x
 + 6*c_8*u*u_2x + 2*c_8*u_3x + 6*c_8*u_x**2)/2},
{c_8,c_13,c_12},
{}}}$
```

So, we have three symmetries; of course the generalized symmetry corresponds to `c_8`. Remember to check *always* the output of CRACK to see if any of the symbols `c_n` is indeed a free function depending on some of the variables, and not just a constant.

5. 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 [12].

The files containing this example are `kdv_lcl1`,`kdv_lcl2` and the corresponding results and debug files.

We suppose that the conservation law has the form $\omega = f_x dx + f_t dt$. Using the same `ansatz` as in the previous example we assume

```
fx:=(for each el in ansatz sum (c(ctel:=ctel+1)*el))$
ft:=(for each el in ansatz sum (c(ctel:=ctel+1)*el))$
```

Next we define the equation $\bar{d}(\omega) = 0$, where $\bar{d}$ is the total exterior derivative restricted to the equation.

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

After solving the equation as in the above example we get

```
fx := c(3)*u_x + c(2)*u + c(1)$
ft := (2*c(8) + 2*c(3)*u*u_x + 2*c(3)*u_3x + c(2)*u**2 +
2*c(2)*u_2x)/2$
```

Unfortunately it is clear that the conservation law corresponding to `c(3)` is trivial, because it is just the KdV equation. 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_lcl2`, 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. In order to do that we have to introduce two new counters:

```
operator cc,equ;
cctel:=0;
```

We make the following ansatz on `f0`:

```
f0:=(for each el in ansatz sum (cc(cctel:=cctel+1)*el))$
```

After solving the system, issuing the commands

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

we obtain

```
fxnontriv := c(2)*u + c(1)$
ftnontriv := (2*c(8) + c(2)*u**2 + 2*c(2)*u_2x)/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. However, we will present another approach to the computation of conservation laws in subsection 8.3.

6. Local Hamiltonian operators

In this section we will show how to compute local Hamiltonian operators for Korteweg–de Vries, Boussinesq and Kadomtsev–Petviashvili equations. It is interesting to note that we will adopt the same computational scheme for both equations, even if the latter is not in evolutionary form and it has more than two independent variables. This comes from a new mathematical theory which started in [19] for evolution equations and was later extended to general differential equations in [21].

6.1. Korteweg–de Vries equation.

Here we will find local Hamiltonian operators for the KdV equation $u_t = u_{xxx} + uu_x$. A necessary condition for an operator to be Hamiltonian is that it sends generating functions (or characteristics, according with [31]) of conservation laws to higher (or generalized) symmetries. As it is proved in [19], this amounts at solving $\bar{\ell}_{KdV}(\texttt{phi}) = 0$ 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, with the further condition that the shadows must be linear in the p-variables.

The file containing this example is `kdv_lho1`.

We stress that the linearization $\bar{\ell}_{KdV}(\texttt{phi}) = 0$ is the equation

```
ddt(phi)-u*ddx(phi)-u_x*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`.

At this point we should discuss how CDE treats odd variables. Externally they look just like even variables, and are indicated by a letter followed by a multiindex. Internally, they are components of an operator: `ext(1)`, `ext(2)`, `ext(3)`, ..., and they are endowed with a skew-symmetric product. There are CDE commands which translate expressions involving odd variables. Namely, to replace in the expression `f` odd variables with `ext` variables (for example, for computations with CDE), do

```
replace_oddext(f);
```

and do

```
replace_extodd(g);
```

if you wish to translate a result g of CDE computations, depending on skew-symmetric internal variables `ext`, in a more readable form in terms of odd variables.

The ansatz must be generalized to odd variables. To this aim we produce two lists: the list `graadlijst` of all even variables collected by their gradings and a similar list `graadlijst_odd` for odd variables:

```
graadlijst:=der_deg_ordering(0,all_parametric_der)$
graadlijst_odd:={1} . der_deg_ordering(1,all_parametric_odd)$
graadmon:=for i:=1:10 collect mkvarlist1(i,i)$
graadmon:={1} . graadmon$
```

In particular, the unknown must be linear in odd variables, so we need a list of graded monomials which are linear in odd variables. The function `mkalllinodd` produces all monomials which are linear with respect to the variables from `graadlijst_odd`, have (monomial) coefficients from the variables in `graadlijst`, and have total scale degrees from 1 to 6. Such monomials are then converted to the internal representation of odd variables.

```
linodd:=mkalllinodd(graadmon,graadlijst_odd,1,6)$
linext:=replace_oddext(linodd)$
```

Note that all odd variables have positive scale degrees thanks to our initial choice `deg_odd_var:=1;`. Finally, the ansatz for local Hamiltonian operators:

```
sym:=(for each el in linext sum (c(ctel:=ctel+1)*el))$
```

After having set

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

and having initialized the equation solver as before, we do `splitext`

```
splitext 1;
```

in order to split the polynomial equation with respect to the `ext` variables, then `splitvars`

```
tel1:=tel;
for i:=2:tel1 do begin splitvars i;equ i:=0;end;
```

in order to split the resulting polynomial equation in a list of equations on the coefficients of all monomials.

Now we are ready to solve all equations:

```
put_equations_used tel;
for i:=2:tel do integrate_equation i;
end;
```

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

The results are the two well-known Hamiltonian operators for the KdV. If we issue the command

```
sym_odd:=replace_extodd(sym)$
```

then the variable `sym_odd` will be expressed in the human-readable notation:

```
sym_odd := (c(5)*p*u_x + 2*c(5)*p_x*u + 3*c(5)*p_3x + 3*c(2)*p_x)/3$
```

Note the internal and external expressions of the result. Of course, the results correspond to the operators

$$p_x \to D_x,$$
$$\frac{1}{3}(3p_{3x} + 2up_x + u_xp) \to \frac{1}{3}(3D_{xxx} + 2uD_x + u_x)$$

Note that each operator is multiplied by one arbitrary real constant, `c(5)` and `c(2)`.

The same problem can be approached using CRACK, as follows (file `kdv_lho2.red`). An ansatz is constructed by the following instructions:

```
even_vars:=for i:=0:3 join selectvars(0,i,dep_var,all_parametric_der)$
odd_vars:=for i:=0:3 join selectvars(1,i,odd_var,all_parametric_odd)$
ext_vars:=replace_oddext(odd_vars)$

ctemp:=0$
ansatz:=for each el in ext_vars sum mkid(s,ctemp:=ctemp+1)*el$
```

Note that we have (after replacement of internal variables with odd variables):

```
ansatz_odd := p*s1 + p_2x*s3 + p_3x*s4 + p_x*s2$
```

Indeed, we are looking for a third-order operator whose coefficients depend on variables of order not higher than 3. This last property has to be introduced by

```
unk:=for i:=1:ctemp collect mkid(s,i)$
for each ell in unk do
 for each el in even_vars do depend ell,el$
```

Then, we introduce the linearization (lifted on the cotangent covering)

```
operator ell_f$
for all sym let ell_f(sym)=
   ddt(sym)-u*ddx(sym)-u_x*sym-ddx(ddx(ddx(sym)))$
```

and the equation to be solved, together with the usual test that checks for the nedd to enlarge the jet space:

```
total_eq:=ell_f(ansatz)$
check_letop({total_eq})$
```

Finally, we split the above equation by collecting all coefficients of odd variables:

```
coeff_ext:=operator_coeff(total_eq,ext)$
system_eq:=for i:=2:length(coeff_ext) collect second(part(coeff_ext,i))$
```

and we feed CRACK with the equations that consist in asking to the above coefficients to be zero:

```
load_package crack;
crack_results:=crack(system_eq,{},unk,
   diffset(all_parametric_der,even_vars));
```

The results are the same as in the previous section:

```
crack_results := {{{},
{s4=(3*c_17)/2,s3=0,s2=c_16 + c_17*u,s1=(c_17*u_x)/2},
{c_17,c_16},
{}}}$
```

6.2. **Boussinesq equation.** There is no conceptual difference when computing for systems of PDEs with respect to the previous computations for scalar equations. We will look for Hamiltonian structures for the following Boussinesq equation:

$$\begin{cases} u_t - u_x v - u v_x - \sigma v_{xxx} = 0 \\ v_t - u_x - v v_x = 0 \end{cases} \tag{2}$$

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

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

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

$$\ell_{\mathrm{Bou}} = \begin{pmatrix} D_t - vD_x - v_x & -u_x - uD_x - \sigma D_{xxx} \\ -D_x & D_t - v_x - vD_x \end{pmatrix}, \quad \ell^*_{\mathrm{Bou}} = \begin{pmatrix} -D_t + vD_x & D_x \\ uD_x + \sigma D_{xxx} & -D_t + vD_x \end{pmatrix}$$

and lead to the ℓ^*_{Bou} covering equation

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

We have to find shadows of symmetries on the above covering. At the level of source file (`bou_lho1_test`) the input data is:

```
indep_var:={x,t}$
dep_var:={u,v}$
odd_var:={p,q}$
deg_indep_var:={-1,-2}$
deg_dep_var:={2,1}$
deg_odd_var:={1,2}$
total_order:=8$
principal_der:={u_t,v_t}$
de:={u_x*v+u*v_x+sig*v_3x,u_x+v*v_x}$
principal_odd:={p_t,q_t}$
de_odd:={v*p_x+q_x,u*p_x+sig*p_3x+v*q_x}$
```

The ansatz for the components of the Hamiltonian operator, of scale degree between 1 and 6, is

```
linodd:=mkalllinodd(graadmon,graadlijst_odd,1,6)$
linext:=replace_oddext(linodd)$
phi1:=(for each el in linext sum (c(ctel:=ctel+1)*el))$
phi2:=(for each el in linext sum (c(ctel:=ctel+1)*el))$
```

and the equation for shadows of symmetries is

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

After the usual procedures for decomposing polynomials we obtain three local Hamiltonian operators:

```
phi1_odd := (2*c(31)*p*sig*v_3x + 2*c(31)*p*u*v_x + 2*c(31)*p*u_x*v + 6*c(31)*
p_2x*sig*v_x + 4*c(31)*p_3x*sig*v + 6*c(31)*p_x*sig*v_2x + 4*c(31)*p_x*u*v + 2*c
(31)*q*u_x + 4*c(31)*q_3x*sig + 4*c(31)*q_x*u + c(31)*q_x*v**2 + 2*c(16)*p*u_x +
 4*c(16)*p_3x*sig + 4*c(16)*p_x*u + 2*c(16)*q_x*v + 2*c(10)*q_x)/2$

phi2_odd := (2*c(31)*p*u_x + 2*c(31)*p*v*v_x + 4*c(31)*p_3x*sig + 4*c(31)*p_x*u
+ c(31)*p_x*v**2 + 2*c(31)*q*v_x + 4*c(31)*q_x*v + 2*c(16)*p*v_x + 2*c(16)*p_x*v
 + 4*c(16)*q_x + 2*c(10)*p_x)/2$
```

There is a whole hierarchy of nonlocal Hamiltonian operators [19].

6.3. **Kadomtsev–Petviashvili equation.** There is no conceptual difference in symbolic computations of Hamiltonian operators for PDEs in 2 independent variables and in more than 2 independent variables, regardless of the fact that the equation at hand is written in evolutionary form. As a model example, we consider the KP equation

$$u_{yy} = u_{tx} - u_x^2 - uu_{xx} - \frac{1}{12}u_{xxxx}. \tag{3}$$

Proceeding as in the above examples we input the following data:

```
indep_var:={t,x,y}$
dep_var:={u}$
odd_var:={p}$
deg_indep_var:={-3,-2,-1}$
deg_dep_var:={2}$
deg_odd_var:={1}$
total_order:=6$
principal_der:={u_2y}$
de:={u_tx-u_x**2-u*u_2x-(1/12)*u_4x}$
principal_odd:={p_2y}$
de_odd:={p_tx-u*p_2x-(1/12)*p_4x}$
```

and look for Hamiltonian operators of scale degree between 1 and 5:

```
linodd:=mkalllinodd(graadmon,graadlijst_odd,1,5)$
linext:=replace_oddext(linodd)$
phi:=(for each el in linext sum (c(ctel:=ctel+1)*el))$
```

After solving the equation for shadows of symmetries in the cotangent covering

```
equ 1:=ddy(ddy(phi))-ddt(ddx(phi))+2*u_x*ddx(phi)
+u_2x*phi+u*ddx(ddx(phi))+(1/12)*ddx(ddx(ddx(ddx(phi))))$
```

we get the only local Hamiltonian operator

```
phi_odd := c(13)*p_2x$
```

As far as we know there are no further local Hamiltonian operators.

Remark: the above Hamiltonian operator is already known in an evolutionary presentation of the KP equation [24]. Our mathematical theory of Hamiltonian operators for general differential equations [21] allows us to formulate and solve the problem for any presentation of the KP equation. Change of coordinate formulae could also be provided.

7. The Schouten bracket of local Hamiltonian operators

It is known [19] that the Schouten bracket between *local* candidates to Hamiltonian operators can be computed in terms of their equivalent linear functions on the cotangent covering. Such linear functions must be rewritten as bivectors, then the bracket can be computed; it will produce an equivalence class of three-vectors which are defined modulo total divergencies. So, to check that such a three-vector is zero we should apply to it the Euler operator and check if the result is zero.

More precisely, consider a differential equation $F = 0$ and its cotangent covering $\ell^*_F(p) = 0$, $F = 0$. Let H^i_1, H^i_2 be shadows of symmetries of the cotangent covering which are linear in p-variables:

$$H^i_j = a^{ik\,\sigma} p_{k\,\sigma}, \quad j = 1, 2,$$

where $a^{i\,\sigma}$ are functions defined on $F = 0$, *i.e.*, functions of the parametric coordinates. Then the corresponding bivectors are written as

$$H_j = a^{ik\,\sigma} p_{k\,\sigma} p_i.$$

Note that the product $p_{k\,\sigma}p_i$ is anticommutative since p's are odd variables. Then, the formula for the Schouten bracket is

$$[H_1, H_2] = \frac{\delta H_1}{\delta u^j}\frac{\delta H_2}{\delta p_j} + \frac{\delta H_2}{\delta u^j}\frac{\delta H_1}{\delta p_j}$$

The above formula produces a three-vector. If we would like to check that the three-vector is indeed a total divergence, we should apply the Euler operator, and check that it is zero:

$$\mathcal{E}([H_1, H_2]) = 0.$$

With the above formula one can check Hamiltonianity $[H_1, H_1] = 0$ and compatibility $[H_1, H_2] = 0$.

7.1. **Bi-Hamiltonian structure of the KdV equation.**

We can do the above computations using KdV equation as a test case (see the file `kdv_lho3.red`).

Let us load the above operators:

```
operator ham1;
for all psi1 let ham1(psi1)=ddx(psi1);
operator ham2;
for all psi2 let ham2(psi2)=(1/3)*u_x*psi2 + ddx(ddx(ddx(psi2)))
 + (2/3)*u*ddx(psi2);
```

We may convert the two operators into the corresponding generating functions. This amounts at evaluating the operators in the odd variable p:

```
sym1:=conv_cdiff2genfun(ham1);
sym2:=conv_cdiff2genfun(ham2);
```

The output of the above two command is, respectively,

```
sym1 := {p_x};
sym2 := {(1/3)*p*u_x + p_3x + (2/3)*p_x*u};
```

Then we shall convert the two generating functions into bivectors:

```
biv1 := conv_genfun2biv(sym1_odd);
biv2 := conv_genfun2biv(sym2_odd);
```

The output of the above commands is the internal notation for bivectors, which is hard to understand; in a short time a more readable form will be available.

Finally, the three Schouten brackets $[B_i, B_j]$ are computed, with $i, j = 1, 2$:

```
sb11 := schouten_bracket(biv1,biv1);
sb12 := schouten_bracket(biv1,biv2);
sb22 := schouten_bracket(biv2,biv2);
```

7.2. Bi-Hamiltonian structure of the WDVV equation.

This subsection refers to the the example file `wdvv_biham1.red`. The simplest nontrivial case of the WDVV equations is the third-order Monge–Ampère equation, $f_{ttt} = f_{xxt}^2 - f_{xxx}f_{xtt}$ [14]. This PDE can be transformed into hydrodynamic form,

$$a_t = b_x, \quad b_t = c_x, \quad c_t = (b^2 - ac)_x,$$

via the change of variables $a = f_{xxx}$, $b = f_{xxt}$, $c = f_{xtt}$. This system possesses two Hamiltonian formulations [17]:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}_t = A_i \begin{pmatrix} \delta H_i/\delta a \\ \delta H_i/\delta b \\ \delta H_i/\delta c \end{pmatrix}, \quad i = 1, 2$$

with the homogeneous first-order Hamiltonian operator

$$\hat{A}_1 = \begin{pmatrix} -\frac{3}{2}D_x & \frac{1}{2}D_x a & D_x b \\ \frac{1}{2}aD_x & \frac{1}{2}(D_x b + bD_x) & \frac{3}{2}cD_x + c_x \\ bD_x & \frac{3}{2}D_x c - c_x & (b^2 - ac)D_x + D_x(b^2 - ac) \end{pmatrix}$$

with the Hamiltonian $H_1 = \int c\,dx$, and the homogeneous third-order Hamiltonian operator

$$A_2 = D_x \begin{pmatrix} 0 & 0 & D_x \\ 0 & D_x & -D_x a \\ D_x & -aD_x & D_x b + bD_x + aD_x a \end{pmatrix} D_x,$$

with the nonlocal Hamiltonian

$$H_2 = -\int \left(\frac{1}{2} a \left(D_x{}^{-1} b \right)^2 + D_x{}^{-1} b D_x{}^{-1} c \right) dx.$$

Both operators are of Dubrovin–Novikov type [15, 16]. This means that the operators are homogeneous with respect to the grading $|D_x| = 1$. It follows that the operators are form-invariant under point transformations of the dependent variables, $u^i = u^i(\tilde{u}^j)$. Here and in what follows we will use the letters u^i to denote the dependent variables (a, b, c). Under such transformations, the coefficients of the operators transform as differential-geometric objects.

The operator A_1 has the general structure

$$A_1 = g_1^{ij} D_x + \Gamma_k^{ij} u_x^k$$

where the covariant metric $g_{1\,ij}$ is flat, $\Gamma^{ij}_k = g_1^{is}\Gamma^j_{sk}$ (here g_1^{ij} is the inverse matrix that represent the contravariant metric induced by $g_{1\,ij}$), and Γ^j_{sk} are the usual Christoffel symbols of $g_{1\,ij}$.

The operator A_2 has the general structure

$$A_2 = D_x\left(g_2^{ij}D_x + c^{ij}_k u^k_x\right)D_x, \tag{4}$$

where the inverse $g_{2\,ij}$ of the leading term transforms as a covariant pseudo-Riemannian metric. From now on we drop the subscript 2 for the metric of A_2. It was proved in [18] that, if we set $c_{ijk} = g_{iq}g_{jp}c^{pq}_k$, then

$$c_{ijk} = \frac{1}{3}(g_{ik,j} - g_{ij,k})$$

and the metric fulfills the following identity:

$$g_{mk,n} + g_{kn,m} + g_{mn,k} = 0. \tag{5}$$

This means that the metric is a Monge metric [18]. In particular, its coefficients are quadratic in the variables u^i. It is easy to input the two operators in CDE. Let us start by A_1: we may define its entries one by one as follows

```
operator a1;

for all psi let a1(1,1,psi) = - (3/2)*ddx(psi);
for all psi let a1(1,2,psi) = (1/2)*ddx(a*psi);
...
```

We could also use one specialized Reduce package for the computation of the Christoffel symbols, like `RedTen` or `GRG`. Assuming that the operators `gamma_hi(i,j,k)` have been defined equal to Γ^{ij}_k and computed in the system using the inverse matrix g_{ij} of the leading coefficient contravariant metric[2]

$$g^{ij} = \begin{pmatrix} -\frac{3}{2} & \frac{1}{2}a & b \\ \frac{1}{2}a & b & \frac{3}{2}c \\ b & \frac{3}{2}c & 2(b^2 - ac) \end{pmatrix}$$

then, provided we defined a list `dep_var` of the dependent variables, we could set

```
operator gamma_hi_con;
for all i,j let gamma_hi_con(i,j) =
(
 for k:=1:3 sum gamma_hi(i,j,k)*mkid(part(dep_var,k),!_x)
)$
```

and

```
operator a1$
for all i,j,psi let a1(i,j,psi) =
gu1(i,j)*ddx(ddx(psi))+(for k:=1:3 sum gamma_hi_con(i,j)*ddx(psi)
)$
```

[2]Indeed in the example file `wdvv_biham1.red` there are procedures for computing all those quantities.

The third order operator can be reconstructed as follows. Observe that the leading contravariant metric is

$$g^{ij} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & -a \\ 1 & -a & 2b + a^2 \end{pmatrix}$$

Introduce the above matrix in `Reduce` as `gu3`. Then set

```
gu3:=gl3**(-1)$
```

and define c_{ijk} as

```
operator c_lo$
for i:=1:3 do
 for j:=1:3 do
  for k:=1:3 do
  <<
   c_lo(i,j,k):=
    (1/3)*(df(gl3(k,i),part(dep_var,j)) - df(gl3(j,i),part(dep_var,k)))$
  >>$
```

Then define c^{ij}_k

```
templist:={}$
operator c_hi$
for i:=1:ncomp do
 for j:=1:ncomp do
  for k:=1:ncomp do
   c_hi(i,j,k):=
    <<
     templist:=
      for m:=1:ncomp join
       for n:=1:ncomp collect
        gu3(n,i)*gu3(m,j)*c_lo(m,n,k)$
     templist:=part(templist,0):=plus
    >>$
```

Introduce the contracted operator

```
operator c_hi_con$
for i:=1:ncomp do
 for j:=1:ncomp do
  c_hi_con(i,j):=
   <<
    templist:=for k:=1:ncomp collect
     c_hi(i,j,k)*mkid(part(dep_var,k),!_x)$
    templist:=part(templist,0):=plus
   >>$
```

Finally, define the operator A_2

```
operator aa2$
for all i,j,psi let aa2(i,j,psi) =
ddx(
gu3(i,j)*ddx(ddx(psi))+c_hi_con(i,j)*ddx(psi)
)$
```

Now, we can test the Hamiltonian property of A_1, A_2 and their compatibility:

```
sym1:=conv_cdiff2genfun(aa1)$
sym2:=conv_cdiff2genfun(aa2)$

biv1:=conv_genfun2biv(sym1)$
biv2:=conv_genfun2biv(sym2)$

schouten_bracket(biv1,biv1);
schouten_bracket(biv1,biv2);
schouten_bracket(biv2,biv2);
```

Needless to say, the result of the last three command is a list of zeroes.

We observe that the same software can be used to prove the bi-Hamiltonianity of a 6-component WDVV system [32].

More formulae are currently being implemented in the system, like symplecticity and Nijenhuis condition for recursion operators [20]. Interested readers are warmly invited to contact R. Vitolo for questions/feature requests.

8. Non-local operators

In this section we will show an experimental way to find nonlocal operators. The word 'experimental' comes from the lack of a comprehensive mathematical theory of nonlocal operators. In any case we will achieve the results by means of a covering of the cotangent covering. Indeed, it can be proved that there is a $1-1$ correspondence between (higher) symmetries of the initial equation and conservation laws on the cotangent covering. Such conservation laws provide new potential variables, hence a covering (see [12] for theoretical details on coverings).

In Section 8.3 we will also discuss a procedure for finding conservation laws from their generating functions that is of independent interest.

8.1. **Non-local Hamiltonian operators for the Korteweg–de Vries equation.**

Here we will compute some nonlocal Hamiltonian operators for the KdV equation. The result of the computation (without the details below) has been published in [19].

We have to solve equations of the type `ddx(ct)-ddt(cx)` as in 5. The main difference is that we will attempt a solution on the ℓ^*-covering (see Subsection 6). 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_nlcl1` and related results and debug files.

The conservation laws that we are looking for are in $1-1$ correspondence with symmetries of the initial equation [19]. We will look for conservatoin laws which correspond to Galilean boost, x-translation, t-translation at the same time. In the case of 2 independent variables and 1 dependent variable, one could prove that one component of such conservation laws can always be written as `sym*p_x0t0` as follows:

```
c1x_odd:=(t*u_x+1)*p$ % degree 1
c2x_odd:=u_x*p$ % degree 4
c3x_odd:=(u*u_x+u_3x)*p_x0t0$ % degree 6
```

Of course, we must pass to the internal representation:

```
c1x:=replace_oddext(c1x_odd)$
c2x:=replace_oddext(c2x_odd)$
c3x:=replace_oddext(c3x_odd)$
```

The second component must be found by solving an equation. To this aim we produce the ansatz

```
c1t_odd:=f1*p+f2*p_x+f3*p_2x$
c1t:=replace_oddext(c1t_odd)$
c2t:=(for each el in linext6 sum (c(ctel:=ctel+1)*el))$ % degree 6
c3t:=(for each el in linext8 sum (c(ctel:=ctel+1)*el))$ % degree 8
```

where we already introduced the sets `linext6` and `linext8` of 6-th and 8-th degree monomials which are linear in odd variables (see the source code). For the first conservation law solutions of the equation

```
equ 1:=ddx(c1t)-ddt(c1x);
```

are found by hand due to the presence of 't' in the symmetry:

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

We also have the equations

```
equ 2:=ddx(c2t)-ddt(c2x);
equ 3:=ddx(c3t)-ddt(c3x);
```

They are solved in the usual way (see the source code of the example and the results file `kdv_nlcl1_res`).

Now, we solve the equation for shadows of nonlocal symmetries in a covering of the ℓ^*-covering (source file `kdv_nlho1_test`). We can produce such a covering by introducing three new nonlocal (potential) variables `ra,rb,rc`. We are going to look for non-local Hamiltonian operators depending linearly on one of these variables. To this aim we modify the odd part of the equation to include the components of the above conservation laws as the derivatives of the new non-local variables `ra`, `rb`, `rc`:

The scale degree analysis of the local Hamiltonian operators of the KdV equation leads to the formulation of the ansatz

```
phi:=(for each el in linext sum (c(ctel:=ctel+1)*el))$
```

where `linext` is the list of graded mononials which are linear in odd variables and have degree 7 (see the source file). The equation for shadows of nonlocal symmetries in ℓ^*-covering

```
equ 1:=ddt(phi)-u*ddx(phi)-u_x*phi-ddx(ddx(ddx(phi)));
```

is solved in the usual way, obtaining (in odd variables notation):

```
phi_odd := (c(5)*(4*p*u*u_x + 3*p*u_3x + 18*p_2x*u_x + 12*p_3x*u
 + 9*p_5x + 4*p_x*u**2 + 12*p_x*u_2x - r2*u_x))/4$
```

Higher non-local Hamiltonian operators could also be found [19]. The CRACK approach also holds for non-local computations.

8.2. **Non-local recursion operator for the Korteweg–de Vries equation.** Following the ideas in [19], a differential operator that sends symmetries into symmetries can be found as a shadow of symmetry on the ℓ-covering of the KdV equation, with the further condition that the shadows must be linear in the covering q-variables. The tangent covering of the KdV equation is

$$\begin{cases} u_t = u_{xxx} + uu_x \\ q_t = u_x q + uq_x + q_{xxx} \end{cases}$$

and we have to solve the equation $\bar{\ell}_{KdV}(\texttt{phi}) = 0$, where $\bar{\ell}_{KdV}$ means that the linearization of the KdV equation is lifted over the tangent covering.

The file containing this example is `kdv_ro1.red`. The example closely follows the computational scheme presented in [23].

Usually, recursion operators are non-local: operators of the form D_x^{-1} appear in their expression. Geometrically we interpret this kind of operator as follows. We introduce a conservation law on the cotangent covering of the form

$$\omega = rt\,dx + rx\,dt$$

where $rt = uq + q_{xx}$ and $rx = q$. It has the remarkable feature of being linear with respect to q-variables. A non-local variable r can be introduced as a potential of ω, as $r_x = rx$, $r_t = rt$. A computation of shadows of symmetries on the system of PDEs

$$\begin{cases} u_t = u_{xxx} + uu_x \\ q_t = u_x q + uq_x + q_{xxx} \\ r_t = uq + q_{xx} \\ r_x = q \end{cases}$$

yields, analogously to the previous computations,

```
  2*c(5)*q*u + 3*c(5)*q_2x + c(5)*r*u_x + c(2)*q.
```

The operator q stands for the identity operator, which is (and must be!) always a solution; the other solution corresponds to the Lenard operator

$$3D_{xx} + 2u + u_x D_x^{-1}.$$

8.3. **Non-local Hamiltonian-recursion operators for Plebanski equation.** The Plebanski (or second Heavenly) equation

$$F = u_{tt}u_{xx} - u_{tx}^2 + u_{xz} + u_{ty} = 0 \tag{6}$$

is Lagrangian, hence it admits a trivial local Hamiltonian operator which is just the Noether map. Nonlocal Hamiltonian and recursion operators have been computed in an evolutionary presentation of the equation in [26]. We can recompute such operators in the above Lagrangian presentation as follows.

First of all, we remark that in a Lagrangian presentation symmetries and cosymmetries coincide, since the equation is self-adjoint. This means that the concept of

Hamiltonian, recursion, symplectic operators coincide. So, instead of trying to find a variety of operators we may focus on just one type of search.

Then, by introducing a suitable nonlocal variable on the cotangent covering. Namely, we compute a linear conservation law (with respect to p's) on the cotangent covering which corresponds with the u-translation symmetry (see [23] for a theoretical description). After guessing the generating function of the conservation law $\psi = (0, 1)$ from the generating function $\varphi = 1$ of the u-translation symmetry , we deduce that the equation

$$\bar{d}\omega = \ell^*_F(p) \tag{7}$$

should hold on the jet space. Here

$$\omega = ct\, dx \wedge dy \wedge dz + cx\, dt \wedge dy \wedge dz + cy\, dt \wedge dx \wedge dz + cz\, dt \wedge dx \wedge dy,$$

where ct, cx, cy, cz are linear functions of p's and its derivatives, with coefficients in u's[3], and

$$\bar{d}\omega = (D_t ct - D_x cx + D_y cy - D_z cz) dt \wedge dx \wedge dy \wedge dz,$$

where total derivatives are lifted on the jet space of even and odd coordinates.

Then, we try to find representatives of the above conservation law which have not more than two non-vanishing components. In particular we will solve the equation

$$D_t ct - D_x cx = 0 \tag{8}$$

in the cotangent covering. Such an equation cannot be solved in general, but it can be solved in this case. In order to solve it we perform dimensional analysis on (7) and we deduce the gradings of ct and cx. The result is determined up to trivial conservation laws, so that we have to remove them; at the end we remain with three 2-component conservation law. It can be proved that they are equivalent, *i.e.*, they differ each other by a trivial conservation law.

This allows us to introduce a new nonlocal *odd* variable r on the cotangent covering such that $r_x = ct$, $r_t = cx$. We obtain an Abelian covering of the cotangent covering:

$$\begin{cases} r_x = ct, \\ r_t = cx, \\ \ell^*_F(p) = 0, \\ F = 0. \end{cases}$$

A nonlocal Hamiltonian operator will be a shadow of symmetry of the above system with respect to the initial equation $F = 0$ with the property of being linear with respect to all (p's and r's) odd variables. With the above nonlocal variable we find a nonlocal Hamiltonian operator which, after changing coordinates to the evolutionary presentation of [26], coincides with one of the nonlocal Hamiltonian operators presented in that paper[4].

Let us describe the computation in detail. We start with the conservation law (see the file `ple_nlcl1.red`):

[3]In general, coefficients can explicitly depend on independent variables.

[4]We observe that in [26] also the trivial Hamiltonian operator is recovered in the evolutionary presentation; of course it has an apparently nontrivial expression.

```
indep_var:={t,x,y,z}$
dep_var:={u}$
odd_var:={p}$
deg_indep_var:={-1,-1,-4,-4}$
deg_dep_var:={1}$
deg_odd_var:={4}$
total_order:=6$
principal_der:={u_xz}$
de:={-u_ty+u_tx**2-u_2t*u_2x}$
principal_odd:={p_xz}$
de_odd:={-p_ty+2*u_tx*p_tx-u_2x*p_2t-u_2t*p_2x}$
```

Now we limit the computation to variables of jetspace order not greater than 4; this is done through the function `selectvars`, which takes four arguments. The first can be 0 for even variables or 1 for odd variables, the second argument is the specified order, the third argument is the subset of dependent variables that we wish to select (all dependent variables in our case) and the fourth is the set of derivative coordinates from which we wish to extract the variables.

```
v0_4:=for i:=0:4 join selectvars(0,i,dep_var,all_parametric_der)$
vo0_4:=for i:=0:4 join selectvars(1,i,odd_var,all_parametric_odd)$
```

We rearrange all variables by their scale degree, starting from variables of degree 1 (the degree is always chosen in such a way that grading of any even or odd derivative coordinates is positive):

```
graadlijst:=der_deg_ordering(0,v0_4)$
graadlijst_odd:={1} . der_deg_ordering(1,vo0_4)$
```

and we collect graded monomials of scale degree less than or equal 13:

```
graadmon:=for i:=1:13 collect mkvarlist1(i,i)$
graadmon:={1} . graadmon$
```

Then we have to make an ansatz for the conservation law: since the summands of `ellstarfp` have degree 9 we assume `[ct]=[cx]=8`

```
deg_cx:=8$
deg_ct:=deg_cx$
```

It would also be `[cy]=[cz]=5`, but in this computation we assume $cy = cz = 0$. Note that no simplification can be assumed like in the case of 2 independent variables: it is not true, in general, that one component of such conservation laws can always be written as `sym*p_x0t0`. The ansatz is constructed through the function `mklinodd`, which takes three arguments: the list of lists of graded monomials of degree 1, 2, . . . , the list of lists of graded odd variables of degree 1, 2, . . . , and the final degree of their products:

```
linoddt:=mklinodd(graadmon,graadlijst_odd,deg_ct)$
linoddx:=linoddt$
linextt:=replace_oddext(linoddt)$
linextx:=linextt$
% Ansatz:
ct:=(for each el in linextt sum (c(ctel:=ctel+1)*el))$
```

```
cx:=(for each el in linextx sum (c(ctel:=ctel+1)*el))$
```

The equation for conservation laws can be checked for the presence of `letop`. If an error is issued, the computation must be rerun with a higher value of `total_order`:

```
ct_t:=ddt(ct)$
cx_x:=ddx(cx)$
check_letop({ct_t,cx_x})$
```

Note that in the folder containing all examples there is also a shell script, `rrr.sh` (works only under `bash`, a GNU/Linux command interpreter) which can be used to run reduce on a given CDE program. If the function `check_letop` issues an error message then the script reruns the computation with a new value of `total_order` one unity higher than the previous one.

Finally we define the equation

```
equ 1:=ct_t-cx_x$
```

The equation admits a lot of solutions, almost all of which are trivial conservation laws (here they are expressed in odd variables):

```
ct_odd := (6*c(45)*p_4x + 6*c(44)*p_t3x + 6*c(43)*p_2t2x + 6*c(42)*p_3tx + 6*c(
34)*p_2x*u_t + 6*c(34)*p_t2x*u + 6*c(34)*p_y + 6*c(33)*p_2x*u_x + 6*c(33)*p_3x*u
 + 6*c(30)*p_t2x*u + 6*c(30)*p_tx*u_x + 6*c(27)*p_2t*u_x + 6*c(27)*p_2tx*u - 6*c
(24)*p_t2x*u + 6*c(24)*p_x*u_tx - 6*c(24)*p_y - 6*c(23)*p_3x*u + 6*c(23)*p_x*
u_2x + 3*c(21)*p_2x*u**2 + 6*c(21)*p_x*u*u_x + 6*c(18)*p_t*u_tx + 6*c(18)*p_tx*
u_t + 6*c(17)*p_t*u_2x - 6*c(17)*p_t2x*u + 6*c(15)*p_t*u*u_x + 3*c(15)*p_tx*u**2
 + 6*c(12)*p*u_2tx + 6*c(12)*p_x*u_2t + 6*c(11)*p*u_t2x + 6*c(11)*p_t2x*u + 6*c(
11)*p_y + 6*c(10)*p*u_3x + 6*c(10)*p_3x*u + 6*c(5)*p*u*u_tx + 6*c(5)*p*u_t*u_x +
 6*c(5)*p_x*u*u_t + 6*c(4)*p*u*u_2x + 6*c(4)*p*u_x**2 - 3*c(4)*p_2x*u**2 + 6*c(2
)*p*u**2*u_x + 2*c(2)*p_x*u**3)/6$

cx_odd := (6*c(45)*p_t3x + 6*c(44)*p_2t2x + 6*c(43)*p_3tx + 6*c(42)*p_4t - 6*c(
34)*p_2t*u_x + 6*c(34)*p_2tx*u + 12*c(34)*p_tx*u_t - 6*c(34)*p_z + 6*c(33)*p_2x*
u_t + 6*c(33)*p_t2x*u + 6*c(30)*p_2tx*u + 6*c(30)*p_tx*u_t + 6*c(27)*p_2t*u_t +
6*c(27)*p_3t*u + 6*c(24)*p_2t*u_x - 6*c(24)*p_2tx*u - 6*c(24)*p_tx*u_t + 6*c(24)
*p_x*u_2t + 6*c(24)*p_z - 6*c(23)*p_2x*u_t - 6*c(23)*p_t2x*u + 6*c(23)*p_tx*u_x
+ 6*c(23)*p_x*u_tx + 3*c(21)*p_tx*u**2 + 6*c(21)*p_x*u*u_t + 6*c(18)*p_2t*u_t +
6*c(18)*p_t*u_2t + 6*c(17)*p_2t*u_x - 6*c(17)*p_2tx*u + 6*c(17)*p_t*u_tx - 6*c(
17)*p_tx*u_t + 3*c(15)*p_2t*u**2 + 6*c(15)*p_t*u*u_t + 6*c(12)*p*u_3t + 6*c(12)*
p_t*u_2t + 6*c(11)*p*u_2tx - 6*c(11)*p_2t*u_x + 6*c(11)*p_2tx*u + 6*c(11)*p_t*
u_tx + 6*c(11)*p_tx*u_t - 6*c(11)*p_x*u_2t - 6*c(11)*p_z + 6*c(10)*p*u_t2x + 6*c
(10)*p_2x*u_t + 6*c(10)*p_t*u_2x + 6*c(10)*p_t2x*u - 6*c(10)*p_tx*u_x - 6*c(10)*
p_x*u_tx + 6*c(5)*p*u*u_2t + 6*c(5)*p*u_t**2 + 6*c(5)*p_t*u*u_t + 6*c(4)*p*u*
u_tx + 6*c(4)*p*u_t*u_x + 6*c(4)*p_t*u*u_x - 3*c(4)*p_tx*u**2 - 6*c(4)*p_x*u*u_t
 + 6*c(2)*p*u**2*u_t + 2*c(2)*p_t*u**3)/6$
```

We begin the removal of trivial conservation laws from the above solution. The idea is that a conservation law (*i.e.* a horizontal 3-form) is trivial if it is equal to the horizontal differential of a 2-form. Such two form will be chosen according with dimensional analysis. We introduce an operator and a counter that will parametrize trivial conservation laws:

```
operator cc$
cctel:=0$
```

Then we assume that the trivial conservation law has the form

$$tcl = tctxdt \wedge dx + tctydt \wedge dy + tctzdt \wedge dz + tcxydx \wedge dy + tcxzdx \wedge dz + tcyzdy \wedge dz$$

so that a conservation law will be trivial if and only if

$$\begin{aligned}\bar{d}(tcl) =&(D_z(tcxy) - D_y(tcxz) + D_x(tcyz))dx \wedge dy \wedge dz+\\ &(D_z(tcty) - D_y(tctz) + D_t(tcyz))dt \wedge dy \wedge dz+\\ &(D_z(tctx) - D_x(tctz) + D_t(tcxz))dt \wedge dx \wedge dz+\\ &(D_y(tctx) - D_x(tcty) + D_t(tcxy))dt \wedge dx \wedge dy\\ =&ctdx \wedge dy \wedge dz + cxdt \wedge dy \wedge dz + cydt \wedge dx \wedge dz + czdt \wedge dx \wedge dy\end{aligned}$$

Since in our case we are looking for a 2-component conservation law, we will assume a single potential of the form: $tcyzdy \wedge dz$:

```
deg_tcyz:=7$
linodd_tcyz:=mklinodd(graadmon,graadlijst_odd,deg_tcyz)$
linext_tcyz:=replace_oddext(linodd_tcyz)$
tcyz:=(for each el in linext_tcyz sum (cc(cctel:=cctel+1)*el))$
```

After clearing the previous equations, we set up the new equation

```
clear equ$
operator equ$

equ 1:=ddx(tcyz) - ct$
equ 2:=ddt(tcyz) - cx$
```

Note that in this case if the equation can be solved then the conservation law is trivial; only if the equation *cannot* be solved we found at least one nontrivial conservation law. Results can be written as follows:

```
write ctnontriv:=equ 1$
write cxnontriv:=equ 2$
```

they will be nonzero if a nontrivial conservation law remains in ct and cx.

Now, we look for nonlocal Hamiltonian operators in the cotangent covering using a new nonlocal odd variable r as follows (see `ple_nlho1.red`):

```
indep_var:={t,x,y,z}$
dep_var:={u}$
odd_var:={p,r}$
deg_indep_var:={-1,-1,-4,-4}$
deg_dep_var:={1}$
deg_odd_var:={1,4}$
total_order:=6$
principal_der:={u_xz}$
```

```
de:={-u_ty+u_tx**2-u_2t*u_2x}$
% rhs of the equations that define the nonlocal variable
rt:=p_2t*u_x - p_2tx*u - 2*p_tx*u_t + p_z$
rx:=- p_2x*u_t - p_t2x*u - p_y$
% We add conservation laws as new nonlocal odd variables;
principal_odd:={p_xz,r_x,r_t}$
%
de_odd:={-p_ty+2*u_tx*p_tx-u_2x*p_2t-u_2t*p_2x,rx,rt}$
```

We look for Hamiltonian operators which depend on r (which has scale degree 4); we produce the following ansatz for `phi`:

```
linodd:=mkalllinodd_e(graadmon,graadlijst_odd,1,4)$
linext:=replace_oddext_e(linodd)$
phi:=(for each el in linext sum (c(ctel:=ctel+1)*el))$
```

then we solve the equation of shadows of symmetries:

```
equ 1:=ddx(ddz(phi))+ddt(ddy(phi))-2*u_tx*ddt(ddx(phi))
+u_2x*ddt(ddt(phi))+u_2t*ddx(ddx(phi))$
```

The solution in odd coordinates is

```
phi_odd := c(20)*p_t*u_x - c(20)*p_tx*u - c(20)*p_x*u_t - c(20)*r + c(1)*p
```

hence we obtain the Noether map (the identity operator p) and the new nonlocal operator $u_x p_t - p_{tx} u - p_x u_t - r$. It can be proved that changing coordinates to the evolutionary presentation yields the local operator (which has a much more complex expression than the identity operator) and one of the nonlocal operators of [26]. More details on this computation can be found in [23].

References

[1] Obtaining Reduce: http://reduce-algebra.sourceforge.net/.
[2] Geometry of Differential Equations web site: http://gdeq.org.
[3] GetDeb website: http://archive.getdeb.net/getdeb/ubuntu/pool/apps/r/reduce-algebra/.
[4] notepad++: http://notepad-plus.sourceforge.net/
[5] List of text editors: http://en.wikipedia.org/wiki/List_of_text_editors
[6] 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
[7] How to install Reduce in Windows: http://reduce-algebra.sourceforge.net/windows.html
[8] 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.
[9] 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.
[10] 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.
[11] Reduce IDE for emacs: http://centaur.maths.qmul.ac.uk/Emacs/REDUCE_IDE/
[12] 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).

[13] D. Baldwin, W. Hereman, *A symbolic algorithm for computing recursion operators of nonlinear partial differential equations*, International Journal of Computer Mathematics, vol. 87 (5), pp. 1094-1119 (2010).

[14] B.A. Dubrovin, Geometry of 2D topological field theories, Lecture Notes in Math. 1620, Springer-Verlag (1996) 120–348.

[15] B.A. Dubrovin and S.P. Novikov, Hamiltonian formalism of one-dimensional systems of hydrodynamic type and the Bogolyubov-Whitham averaging method, Soviet Math. Dokl. **27** No. 3 (1983) 665–669.

[16] B.A. Dubrovin and S.P. Novikov, Poisson brackets of hydrodynamic type, Soviet Math. Dokl. **30** No. 3 (1984), 651–2654.

[17] E.V. Ferapontov, C.A.P. Galvao, O. Mokhov, Y. Nutku, Bi-Hamiltonian structure of equations of associativity in 2-d topological field theory, Comm. Math. Phys. **186** (1997) 649-669.

[18] E.V. Ferapontov, M.V. Pavlov, R.F. Vitolo, *Projective-geometric aspects of homogeneous third-order Hamiltonian operators*, J. Geom. Phys. **85** (2014) 16-28, DOI: 10.1016/j.geomphys.2014.05.027.

[19] 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.

[20] P.H.M. Kersten, I.S. Krasil'shchik, A.M. Verbovetsky, *A geometric study of the dispersionless Boussinesq equation*, Acta Appl. Math. **90** (2006), 143–178.

[21] P. Kersten, I. Krasil′shchik, A. Verbovetsky, and R. Vitolo, *Hamiltonian structures for general PDEs*, Differential equations: Geometry, Symmetries and Integrability. The Abel Symposium 2008 (B. Kruglikov, V. V. Lychagin, and E. Straume, eds.), Springer-Verlag, 2009, pp. 187–198, arXiv:0812.4895.

[22] I. Krasil′shchik and A. Verbovetsky, *Geometry of jet spaces and integrable systems*, J. Geom. Phys. (2011) doi:10.1016/j.geomphys.2010.10.012, arXiv:1002.0077.

[23] I. Krasil′shchik, A. Verbovetsky, R. Vitolo, *A unified approach to computation of integrable structures*, Acta Appl. Math. (2012).

[24] B. Kuperschmidt: *Geometric Hamiltonian forms for the Kadomtsev–Petviashvili and Zabolotskaya–Khokhlov equations*, in Geometry in Partial Differential Equations, A. Prastaro, Th.M. Rassias eds., World Scientific (1994), 155–172.

[25] M. Marvan, *Sufficient set of integrability conditions of an orthonomic system.* Foundations of Computational Mathematics **9** (2009), 651–674.

[26] F. Neyzi, Y. Nutku, and M.B. Sheftel, *Multi-Hamiltonian structure of Plebanski's second heavenly equation* J. Phys. A: Math. Gen. **38** (2005), 8473. arXiv:nlin/0505030v2.

[27] A.C. Norman, R. Vitolo, *Inside Reduce*, part of the official `Reduce` documentation included in the source code, see below.

[28] M.C. Nucci, *Interactive REDUCE programs for calculating classical, non-classical, and approximate symmetries of differential equations*, in Computational and Applied Mathematics II. Differential Equations, W.F. Ames, and P.J. Van der Houwen, Eds., Elsevier, Amsterdam (1992) pp. 345–350.

[29] M.C. Nucci, *Interactive REDUCE programs for calculating Lie point, non-classical, Lie-Bcklund, and approximate symmetries of differential equations: manual and floppy disk*, in CRC Handbook of Lie Group Analysis of Differential Equations. Vol. 3 N.H. Ibragimov, Ed., CRC Press, Boca Raton (1996) pp. 415–481.

[30] F. Oliveri, ReLie, `Reduce` software and user guide, http://mat521.unime.it/oliveri/.

[31] P. Olver, Applications of Lie Groups to Partial Differential Equations, 2nd ed, GTM Springer, 1992.

[32] M.V. Pavlov, R.F. Vitolo: *On the bi-Hamiltonian geometry of the WDVV equations*, http://arxiv.org/abs/1409.7647

[33] G. Saccomandi, R. Vitolo: *On the Mathematical and Geometrical Structure of the Determining Equations for Shear Waves in Nonlinear Isotropic Incompressible Elastodynamics*, J. Math. Phys. **55** (2014), 081502.

[34] `Reduce` official website: http://reduce-algebra.sourceforge.net/.

[35] T. Wolf, *A comparison of four approaches to the calculation of conservation laws*, Euro. Jnl of Applied Mathematics 13 part 2 (2002) 129-152.

[36] T. Wolf, *APPLYSYM - a package for the application of Lie-symmetries*, software distributed together with the computer algebra system REDUCE, (1995).

[37] T. Wolf, A. Brand, *Investigating DEs with CRACK and Related Programs*, SIGSAM Bulletin, Special Issue, (June 1995), p 1-8.

[38] T. Wolf, *An efficiency improved program LIEPDE for determining Lie-symmetries of PDEs*, Proc.of Modern Group Analysis: advanced analytical and computational methods in mathematical physics, Catania, Italy Oct.1992, Kluwer Acad.Publ. (1993) 377-385.

[39] T. Wolf, A. Brand: CRACK, user guide, examples and documentation http://lie.math.brocku.ca/Crack_demo.html. For applications, see also the publications of T. Wolf.

R. Vitolo, Dept. of Mathematics "E. De Giorgi", Università del Salento, via per Arnesano, 73100 Lecce, Italy

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