# CDE: A REDUCE PACKAGE FOR INTEGRABILITY OF PDES VERSION 2.0

R. VITOLO

Abstract. We describe CDE, a `Reduce` package devoted to differential-geometric computations on Differential Equations (DEs, for short). The package is included in the official `Reduce` sources in Sourceforge [36] 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. [11, 21].

## Contents



*Date*: 2015 October – CDE version: 2.0.

2010 *Mathematics Subject Classification.* 37Kxx.

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

## 1. Introduction: 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 [11]). 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 [7, 8, 9]. 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 [30, 31], F. Oliveri (ReLie, [32]), F. Schwartz (SPDE, `Reduce` official distribution) T. Wolf (APPLYSYM and CONLAW in the official `Reduce` distribution, [37, 38, 39, 40]).

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 [12].

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, 2.0, has the following features:

(1) is able to do standard computations in integrable systems like determining systems for generalized symmetries and conservation laws. However, CDE has not been programmed with this purpose in mind.
(2) CDE is able to compute linear overdetermined systems of partial differential equations whose solutions are Hamiltonian, symplectic or 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 [41].
(3) CDE can compute linearization (or Fréchet derivatives) of vector functions and adjoints of differential operators.
(4) CDE is able to compute Schouten brackets between multivectors. This can be used *eg* to check Hamiltonianity of an operator or to check their compatibility.

At the moment the papers [17, 18, 23, 25, 34, 35] have been written using 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.** I'd like to thank 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. When I started writing CDE I was substantially helped by A.C. Norman in understanding many features of Reduce which were deeply hidden in the source code and not well documented. This also led to writing a manual of Reduce's internals for programmers [29]. Moreover, I'd like to thank the developers of the `Reduce` mailing list for their prompt replies with solutions to my problems. On the mathematical side, I would like to thank J.S. Krasil'shchik and A.M. Verbovetsky for constant support and stimulating discussions which led me to write the software. Thanks are also due to B.A. Dubrovin, M. Casati, E.V. Ferapontov, P. Lorenzoni, M. Marvan, V. Novikov, A. Savoldi, A. Sergyeyev, M.V. Pavlov for many interesting discussions.

## 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. There is an experimental* `Reduce` *installer for Windows on Sourceforge, interested users might wish to try it.*

### 2.1. Installation of Reduce.

In order to install `Reduce` one can download the Windows installer or download a precompiled binary distribution from here [36]. However, please make sure that the version that you are downloading has been compiled *later* than October 2015, or you will not get CDE in it. If you are ready to recompile `Reduce`, please consider the text [29] for instructions on how to do it in different operating systems.

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-15 ...

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 [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.

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 [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 23.2.1 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 [11].

## 4. Jet space of even and odd variables, and total derivatives

The mathematical theory for jets of even (*ie* standard) variables and total derivatives can be found in [11, 33].

Let us consider the space $\mathbb{R}^n \times \mathbb{R}^m$, with coordinates $(x^\lambda, u^i)$, $1 \leq \lambda \leq n$, $1 \leq i \leq m$. We say $x^\lambda$ to be *independent variables* and $u^i$ to be *dependent variables*. Let us introduce the *jet space* $J^r(n, m)$. This is the space with coordinates $(x^\lambda, u^i_\sigma)$, where $u^i_\sigma$ is defined as follows. If $s\colon \mathbb{R}^n \to \mathbb{R}^m$ is a differentiable function, then

$$u^i_\sigma \circ s(x) = \frac{\partial^{|\sigma|}(u^i \circ s)}{(\partial x^1)^{\sigma_1} \cdots (\partial x^n)^{\sigma_n}}.$$

Here $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathbb{N}^n$ is a multiindex. We set $|\sigma| = \sigma_1 + \cdots + \sigma_n$. If $\sigma = (0, \ldots, 0)$ we set $u^i_\sigma = u^i$.

CDE is first of all a program which is able to create a *finite order jet space* inside `Reduce`. To this aim, issue the command

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

Then, CDE needs to know the variables and the maximal order of derivatives. The input can be organized as in the following example:

```
indep_var:={x,t}$
dep_var:={u,v}$
total_order:=10$
```

Here

- `indep_var` is the list of independent variables;
- `dep_var` is the list of dependent variables;
- `total_order` is the maximal order of derivatives.

Two more parameters can be set for convenience:

```
statename:="jetuv_state.red"$
resname:="jetuv_res.red"$
```

These are the name of the output file for recording the internal state of the program `cde.red` (and for debugging purposes), and the name of the file containing results of the computation.

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

```
cde({indep_var,dep_var,{},total_order},{})$
```

Here the two empty lists are placeholders; they are important for computations with odd variables/differential equations. The function `cde` defines derivative symbols of the type:

```
u_x,v_t,u_2xt,v_xt,v_2x3t,...
```

Note that the symbol `v_tx` does not exist in the jet space. Indeed, introducing all possible permutations of independent variables in indices would increase the complexity and slow down every computation.

Two lists generated by CDE can be useful: `all_der_id` and `all_odd_id`, which are, respectively, the lists of identifiers of all even and odd variables.

Other lists are generated by CDE, but they are accessible in `Reduce` symbolic mode only. Please check the file `global.txt` to know the names of the lists.

It can be useful to inspect the output generated by the function `cde` and the above lists in particular. All that data can be saved by the function:

```
save_cde_state(statename)$
```

CDE has a few procedures involving the jet space, namely:

- `jet_fiber_dim(jorder)` returns the number of derivative coordinates $u^i_\sigma$ with $|\sigma|$ equal to `jorder`;
- `jet_dim(jorder)` returns the number of derivative coordinates $u^i_\sigma$ with $0 \leq |\sigma|$ and $|\sigma|$ equal to `jorder`;
- `selectvars(par,orderofder,depvars,vars)` returns all derivative coordinates (even if `par=0`, odd if `par=1`) of order `orderofder` of the list of dependent variables `depvars` which belong to the set of derivative coordinates `vars`.

The function `cde` defines total derivatives truncated at the order `total_order`. Their coordinate expressions are of the form

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

where $\boldsymbol{\sigma}$ is a multiindex.

The total derivative of an argument $\varphi$ is invoked as follows:

```
td(phi,x,2);
td(phi,x,t,3);
```

the syntax closely follows `Reduce`'s syntax for standard derivatives `df`; the above expression translates to $D_xD_x\varphi$, or $D_{\{2,0\}}\varphi$ in multiindex notation.

When in total derivatives there is a coefficient of order higher than maximal this is replaced by the identifier `letop`, which is a function that depends on independent variables. If such a function (or its derivatives) appears during computations it is likely that we went too close to the highest order variables that we defined in the file. All results of computations are scanned for the presence of such variables by default, and if the presence of `letop` is detected the computation is stopped with an error message. This usually means that we need to extend the order of the jet space, just by increasing the number `total_order`.

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. When an error message about `letop` is issued the script reruns the computation with a new value of `total_order` one unity higher than the previous one.

The function that checks an expression for the presence of `letop` is `check_letop`. If you wish to switch off this kind of check in order to increase the speed, the switch `checkord` must be set off:

```
off checkord;
```

The computation of total derivatives of a huge expression can be extremely time and resources consuming. In some cases it is a good idea to disable the expansion of the total derivative and leave an expression of the type $D_\sigma\varphi$ as indicated. This is achieved by the command

```
noexpand_td();
```

If you wish to restore the default behaviour, do

```
expand_td();
```

CDE can also compute on jets of supermanifolds. The theory can be found in [20, 21, 24]. The input can be organized as follows:

```
indep_var:={x,t}$
dep_var:={u,v}$
odd_var:={p,q}
total_order:=10$
```

Here `off_var` is the list of odd variables. The call

```
cde({indep_var,dep_var,odd_var,total_order},{})$
```

will create the jet space of the supermanifold described by the independent variables and the even and odd dependent variables, up to the order `total_order`. Total derivatives truncated at the order `total_order` will also include odd derivatives:

$$D_\lambda = \frac{\partial}{\partial x^\lambda} + u^i_{\boldsymbol{\sigma}\lambda}\frac{\partial}{\partial u^i_{\boldsymbol{\sigma}}} + p^i_{\boldsymbol{\sigma}\lambda}\frac{\partial}{\partial p^i_{\boldsymbol{\sigma}}}, \tag{2}$$

where $\boldsymbol{\sigma}$ is a multiindex. The considerations on expansion and `letop` apply in this case too.

Odd variables can appear in anticommuting products; this is represented as

```
ext(p,p_2xt),ext(p_x,q_t,q_x2t),...
```

where `ext(p_2xt,p) = - ext(p,p_2xt)` and the variables are arranged in a unique way terms of an internal ordering. Indeed, the internal representation of odd variables and their products (not intended for normal users!) is

```
ext(3,23),ext(1,3,5),...
```

as all odd variables and their derivatives are indexed by integers. Note that `p` and `ext(p)` are just the same. The odd product of two expressions $\varphi$ and $\psi$ is achieved by the CDIFF function

```
super_product(phi,psi);
```

The derivative of an expression $\varphi$ with respect to an odd variable $p$ is achieved by

```
df_odd(phi,p);
```

## 5. Differential equations in even and odd variables

We now give the equation in the form of one or more derivatives equated to right-hand side expressions. The left-hand side derivatives are 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 are determined by 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`.) The input is formed as follows (Burger's equation).

[1]This terminology dates back to Riquier, see [27]

```
% 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}$
```

Systems of PDEs are input in the same way: of course, the above two lists must have the same length. See 11.2 for an example.

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

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

Here the three empty lists are placeholders; they are important for computations with odd variables. The function `cde` computes principal and parametric derivatives of even and odd variables, they are stored in the lists `all_parametric_der, all_principal_der, all_parametric_odd, all_principal_odd`.

The function `cde` also 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

$$D_\lambda = \frac{\partial}{\partial x^\lambda} + \sum_{u^i_{\boldsymbol{\sigma}} \text{ parametric}} u^i_{\boldsymbol{\sigma}\lambda} \frac{\partial}{\partial u^i_{\boldsymbol{\sigma}}} + \sum_{p^i_{\boldsymbol{\sigma}} \text{ parametric}} p^i_{\boldsymbol{\sigma}\lambda} \frac{\partial}{\partial p^i_{\boldsymbol{\sigma}}}, \tag{3}$$

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. The solved system is available in the form of `Reduce` let-rules in the variables `repprincparam_der` and `repprincparam_odd`.

The syntax and properties (expansion and `letop`) of total derivatives remain the same. For exmaple:

```
td(u,t);
```

returns

```
u_2x+2*u*u_x;
```

It is possible to deal with mixed systems on eve and odd variables. For example, in the case of Burgers equation we can input the linearized equation as a PDE on a new odd variable as follows (of course, in addition to what has been defined before):

```
odd_var:={q}$
principal_odd:={q_t}$
de_odd:={q_2x + 2*u_x*q + 2*u*q_x}$
```

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})$
```

## 6. Calculus of variations

CDE can compute variational derivatives of any function (usually a Lagrangian density) or superfunction $\mathcal{L}$. We have the following coordinate expression

$$\frac{\delta \mathcal{L}}{\delta u^i} = (-1)^{|\sigma|} D_\sigma \frac{\partial \mathcal{L}}{\partial u^i_\sigma}, \quad \frac{\delta \mathcal{L}}{\delta p^i} = (-1)^{|\sigma|} D_\sigma \frac{\partial \mathcal{L}}{\partial p^i_\sigma} \tag{4}$$

which translates into the CDE commands

```
pvar_df(0,lagrangian_dens,ui);
pvar_df(1,lagrangian_dens,pi);
```

where

- the first argument can be 0 or 1 and is the parity of the variable `ui` or `pi`;
- `lagrangian_dens` is $\mathcal{L}$;
- `ui` or `pi` are the given dependent variables.

The Euler operator computes variational derivatives with respect to all even and odd variables in the jet space, and arranges them in a list of two lists, the list of even variational derivatives and the list of odd variational derivatives. The command is

```
euler_df(lagrangian_dens);
```

All the above is used in the definition of Schouten brackets, as we will see in Subsection 7.2.

## 7. $\mathcal{C}$-differential operators

Linearizing (or taking the Fréchet derivative) of a vector function that defines a differential equation yields a differential operator in total derivatives. This operator can be restricted to the differential equation, which may be regarded as a differential constraint; the kernel of the restricted operator is the space of all symmetries (including higher or generalized symmetries) [11, 33].

The formal adjoint of the linearization operator yields by restriction to the corresponding differential equation a differential operator whose kernel contains all characteristic vectors or generating functions of conservation laws [11, 33].

Such operators are examples of $\mathcal{C}$-differential operators. The (still incomplete) `Reduce` implementation of the calculus of $\mathcal{C}$-differential operators is the subject of this section.

### 7.1. $\mathcal{C}$-differential operators.

Let us consider the spaces

$$P = \{\varphi \colon J^r(n,m) \to \mathbb{R}^k\}, \qquad Q = \{\psi \colon J^r(n,m) \to \mathbb{R}^s\}.$$

A *$\mathcal{C}$-differential operator* $\Delta \colon P \to Q$ is defined to be a map of the type

$$\Delta(\varphi) = (\sum_{\sigma,i} a_i^{\sigma j} D_\sigma \varphi^i), \tag{5}$$

where $a_i^{\sigma j}$ are differentiable functions on $J^r(n,m)$, $1 \le i \le k$, $1 \le j \le s$. The *order* of $\delta$ is the highest length of $\sigma$ in the above formula.

We may consider a generalization to *$k$-$\mathcal{C}$-differential operators* of the type

$$\Delta \colon P_1 \times \cdots \times P_h \to Q \tag{6}$$

$$\Delta(\varphi_1,\dots,\varphi_h) = (\sum_{\sigma_1,\dots,\sigma_h,i_1,\dots,i_h} a^{\sigma_1,\dots,\sigma_h,\ j}_{i_1\cdots i_h} D_{\sigma_1}\varphi_1^{i_1}\cdots D_{\sigma_h}\varphi_h^{i_h}),$$

where the enclosing parentheses mean that the value of the operator is a vector function in $Q$.

A $\mathcal{C}$-differential operator in CDE must be declared as follows:

```
mk_cdiffop(opname,num_arg,length_arg,length_target)
```

where

- `opname` is the name of the operator;
- `num_arg` is the number of arguments *eg* $k$ in (6);
- `length_arg` is the list of lengths of the arguments: *eg* the length of the single argument of $\Delta$ (5) is $k$, and the corresponding list is `{k}`, while in (6) one needs a list of $k$ items `{k_1,...,k_h}`, each corresponding to number of components of the vector functions to which the operator is applied;
- `length_target` is the numer of components of the image vector function.

The syntax for one component of the operator `opname` is

```
opname(j,i1,...,ih,phi1,...,phih)
```

The above operator will compute

$$\Delta(\varphi_1,\dots,\varphi_h) = \sum_{\sigma_1,\dots,\sigma_h} a^{\sigma_1,\dots,\sigma_h,\ j}_{i_1\cdots i_h} D_{\sigma_1}\varphi_1^{i_1}\cdots D_{\sigma_h}\varphi_h^{i_h}, \tag{7}$$

for fixed integer indices $i_1,\dots,i_h$ and $j$.

There are several operations which involve differential operators. Obviously they can be summed and multiplied by scalars.

An important example of $\mathcal{C}$-differential operator is that of *linearization*, or *Fréchet derivative*, of a vector function

$$F\colon J^r(n,m)\to\mathbb{R}^k.$$

This is the operator

$$\ell_F\colon \varkappa\to P,\quad \varphi\mapsto\sum_{\sigma,i}\frac{\partial F^k}{\partial u^i_\sigma}D_\sigma\varphi^i,$$

where $\varkappa=\{\varphi\colon J^r(n,m)\to\mathbb{R}^m\}$ is the space of *generalized vector fields on jets* [11, 33].

Linearization can be extended to an operation that, starting from a $k$-$\mathcal{C}$-differential operator, generates a $k+1$-$\mathcal{C}$-differential operator as follows:

$$\ell_\Delta(p_1,\dots,p_k,\varphi) = (\sum_{\sigma,\sigma_1,\dots,\sigma_k,i,i_1,\dots,i_k}\frac{\partial a^{\sigma_1,\dots,\sigma_k,\ j}_{i_1\cdots i_k}}{\partial u^i_\sigma}D_\sigma\varphi^i D_{\sigma_1}p_1^{i_1}\cdots D_{\sigma_k}p_k^{i_k})$$

(The above operation is also denoted by $\ell_{\Delta,p_1,\dots,p_k}(\varphi)$.)

At the moment, CDE is only able to compute the linearization of a vector function (Section 8).

Given a $\mathcal{C}$-differential operator $\Delta$ like in (5) we can define its *adjoint* as

$$\Delta^*((q_j)) = (\sum_{\sigma,i}(-1)^{|\sigma|}D_\sigma(a^{\sigma j}_i q_j)). \tag{8}$$

Note that the matrix of coefficients is transposed. Again, the coefficients of the adjoint operator can be found by computing $\Delta^*(x^\sigma e_j)$ for every basis vector $e_j$ and every count $x^\sigma$, where $|\sigma| \leq r$, and $r$ is the order of the operator. This operation can be generalized to $\mathcal{C}$-differential operators with $h$ arguments.

At the moment, CDE can compute the adjoint of an operator with one argument (Section 8).

Now, consider two operators $\Delta\colon P \to Q$ and $\nabla\colon Q \to R$. Then the composition $\nabla \circ \Delta$ is again a $\mathcal{C}$-differential operator. In particular, if

$$\Delta(p) = (\sum_{\sigma,i} a_i^{\sigma j} D_\sigma p^i), \quad \nabla(q) = (\sum_{\tau,j} b_j^{\tau k} D_\tau q^j),$$

then

$$\nabla \circ \Delta(p) = (\sum_{\tau,j} b_j^{\tau k} D_\tau (\sum_{\sigma,i} a_i^{\sigma j} D_\sigma p^i))$$

This operation can be generalized to $\mathcal{C}$-differential operators with $h$ arguments.

There is another important operation between $\mathcal{C}$-differential operators with $h$ arguments: the *Schouten bracket* [11]. We will discuss it in next Subsection, in the context of another formalism, where it takes an easier form [21].

7.2. **$\mathcal{C}$-differential operators as superfunctions.** In the papers [20, 21] (and independently in [19]) a scheme for dealing with (skew-adjoint) variational multivectors was devised. The idea was that operators of the type (6) could be represented by homogeneous vector superfunctions on a supermanifold, where odd coordinates $q^i_\sigma$ would correspond to total derivatives $D_\sigma \varphi^i$.

The isomorphism between the two languages is given by

$$(9)\quad \Big( \sum_{\sigma_1,\dots,\sigma_h,i_1,\dots,i_h} a_{i_1\cdots i_h}^{\sigma_1,\dots,\sigma_h,\ j} D_{\sigma_1}\varphi_1^{i_1} \cdots D_{\sigma_h}\varphi_h^{i_h} \Big) \longrightarrow \Big( \sum_{\sigma_1,\dots,\sigma_h,i_1,\dots,i_h} a_{i_1\cdots i_h}^{\sigma_1,\dots,\sigma_h,\ j} q_{\sigma_1}^{i_1} \cdots q_{\sigma_h}^{i_h} \Big)$$

where $q^i_\sigma$ is the derivative of an odd dependent variable (and an odd variable itself).

A superfunction in CDE must be declared as follows:

```
mk_superfun(sfname,num_arg,length_arg,length_target)
```

where

- `sfname` is the name of the superfunction;
- `num_arg` is the degree of the superfunction *eg* $h$ in (9);
- `length_arg` is the list of lengths of the arguments: *eg* the length of the single argument of $\Delta$ (5) is $k$, and the corresponding list is {`k`}, while in (6) one needs a list of $k$ items {`k_1,...,k_h`}, each corresponding to number of components of the vector functions to which the operator is applied;
- `length_target` is the numer of components of the image vector function.

The above parameters of the operator `opname` are stored in the property list[2] of the identifier `opname`. This means that if one would like to know how many arguments has the operator `opname` the answer will be the output of the command

```
get('cdnarg,cdiff_op);
```

[2]The property list is a lisp concept, see [29] for details.

and the same for the other parameters.

The syntax for one component of the superfunction `sfname` is

```
sfname(j)
```

CDE is able to deal with $\mathcal{C}$-differential operators in both formalisms, and provides conversion utilities:

- `conv_cdiff2superfun(cdop,superfun)`
- `conv_cdiff2superfun(superfun,cdop)`

where in the first case a $\mathcal{C}$-differential operator `cdop` is converted into a vector superfunction `superfun` with the same properties, and conversely.

7.3. **The Schouten bracket.** We are interested in the operation of Schouten bracket between *variational multivectors* [20]. These are differential operators with $h$ arguments in $\varkappa$ with values in densities, and whose image is defined up to total divergencies:

$$\Delta\colon \varkappa\times\cdots\times\varkappa\to\{J^r(n,m)\to\lambda^nT^*\mathbb{R}^\times\}/\bar{d}(\{J^r(n,m)\to\lambda^{n-1}T^*\mathbb{R}^\times\}) \tag{10}$$

It is known [19, 21] that the Schouten bracket between two variational multivectors $A_1$, $A_2$ can be computed in terms of their corresponding superfunction by the formula

$$[A_1,A_2]=\left[\frac{\delta A_2}{\delta u^j}\frac{\delta A_1}{\delta p_j}-(-1)^{(A_1+1)(A_2+1)}\frac{\delta A_1}{\delta u^j}\frac{\delta A_2}{\delta p_j}\right] \tag{11}$$

where $\delta/\delta u^i$, $\delta/\delta p_j$ are the variational derivatives and the square brackets at the right-hand side should be understood as the equivalence class up to total divergencies.

If the operators $A_1$, $A_2$ are compatible, *ie* $[A_1,A_2]=0$, the expression (11) must be a total derivative. This means that:

$$[A_1,A_2]=0\quad\Leftrightarrow\quad\mathcal{E}\left(\frac{\delta A_2}{\delta u^j}\frac{\delta A_1}{\delta p_j}-(-1)^{(A_1+1)(A_2+1)}\frac{\delta A_1}{\delta u^j}\frac{\delta A_2}{\delta p_j}\right)=0. \tag{12}$$

If $A_1$ is an $h$-vector and $A_2$ is a $k$-vector the formula (11) produces a $(h+k-1)$-vector, or a $\mathcal{C}$-differential operator with $h+k-1$ arguments. If we would like to check that this multivector is indeed a total divergence, we should apply the Euler operator, and check that it is zero. This procedure is considerably simpler than the analogue formula with operators (see for example [21]). All this is computed by CDE:

```
schouten_bracket(biv1,biv2,tv12);
```

where `biv1` and `biv2` are bivectors, or $\mathcal{C}$-differential operators with 2 arguments, and `tv12` is the result of the computation, which is a three-vector (it is automatically declared to be a superfunction). Examples of this computation are given in Section 12.

## 8. Computing linearization and its adjoint

Currently, CDE supports linearization of a vector function, or a $\mathcal{C}$-differential operator with 0 arguments. The computation is performed in odd coordinates.

Suppose that we would like to linearize the vector function that defines the (dispersionless) Boussinesq equation [22]:

$$\begin{cases} u_t-u_xv-uv_x-\sigma v_{xxx}=0\\ v_t-u_x-vv_x=0\end{cases} \tag{13}$$

where $\sigma$ is a constant. Then a jet space with independent variables `x,t`, dependent variables `u,v` and odd variables *in the same number as dependent variables* `p,q` must be created:

```
indep_var:={x,t}$
dep_var:={u,v}$
odd_var:={p,q}$
total_order:=8$
cde({indep_var,dep_var,odd_var,total_order},{})$
```

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

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

Let us introduces the vector function whose zeros are the Boussinesq equation:

```
f_bou:={u_t - (u_x*v + u*v_x + sig*v_3x),v_t - (u_x + v*v_x)};
```

The following command assigns to the identifier `lbou` the linearization $\mathcal{C}$-differential operator $\ell_{\mathrm{Bou}}$ of the vector function `f_bou`

```
ell_function(f_bou,lbou);
```

moreover, a superfunction `lbou_sf` is also defined as the vector superfunction corresponding to $\ell_{\mathrm{Bou}}$. Indeed, the following sequence of commands:

```
2: lbou_sf(1);

 - p*v_x + p_t - p_x*v - q*u_x - q_3x*sig - q_x*u

3: lbou_sf(2);

 - p_x - q*v_x + q_t - q_x*v
```

shows the vector superfunction corresponding to $\ell_{\mathrm{Bou}}$. To compute the value of the $(1,1)$ component of the matrix $\ell_{\mathrm{Bou}}$ applied to an argument `psi` do

```
lbou(1,1,psi);
```

In order to check that the result is correct one could define the linearization as a $\mathcal{C}$-differential operator and then check that the corresponding superfunctions are the same:

```
mk_cdiffop(lbou2,1,{2},2);
for all phi let lbou2(1,1,phi)=td(phi,t) - v*td(phi,x) - v_x*phi;
for all phi let lbou2(1,2,phi)= - u_x*phi - u*td(phi,x) - sig*td(phi,x,3);
for all phi let lbou2(2,1,phi)= - td(phi,x);
for all phi let lbou2(2,2,phi)=td(phi,t) - v*td(phi,x) - v_x*phi;

conv_cdiff2superfun(lbou2,lbou2_sf);
lbou2_sf(1) - lbou_sf(1);
lbou2_sf(2) - lbou_sf(2);
```

the result of the two last commands must be zero.

The formal adjoint of `lbou` can be computed and assigned to the identifier `lbou_star` by the command

```
adjoint_cdiffop(lbou,lbou_star);
```

Again, the associated vector superfunction `lbou_star_sf` is computed, with values

```
4: lbou_star_sf(1);

 - p_t + p_x*v + q_x

5: lbou_star_sf(2);

p_3x*sig + p_x*u - q_t + q_x*v
```

Again, the above operator can be checked for correctness.

Once the linearization and its ajdoint are computed, in order to do computations with symmetries and conservation laws such operator must be restricted to the corresponding equation. This can be achieved with the following steps:

(1) compute linearization of a PDE of the form $F = 0$ and its adjoint, and save them in the form of a vector superfunction;
(2) start a new computation with the given *even* PDE as a constraint on the (even) jet space;
(3) load the superfunctions of item 1;
(4) restrict them to the even PDE.

Only the last step needs to be explained. If we are considering, *eg* the Boussinesq equation, then $u_t$ and its differential consequences (*ie* the principal derivatives) are not automatically expanded to the right-hand side of the equation and its differential consequences. At the moment this step is not fully automatic. More precisely, only principal derivatives which appear as coefficients in total derivatives can be replaced by their expression. The lists of such derivatives with the corresponding expressions are `repprincparam_der` and `repprincparam_odd` (see Section 5). They are in the format of `Reduce`'s replacement list and can be used in let-rules. If the linearization or its adjoint happen to depend on another principal derivative this must be computed separately. A forthcoming release of `Reduce` will automatize this procedure.

However, note that for evolutionary equations this step is trivial, as the restriction of linearization and its adjoint on the given PDE will only affect total derivatives which are restricted by CDE to the PDE.

## 9. Higher symmetries

In this section we show the computation of (some) higher [11] (or generalized, [33]) 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`.

9.1. **Setting up the jet space and the differential equation.** After loading CDE:

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

Here the new lists are scale degrees:

- `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;

We now give the equation and call CDE:

```
principal_der:={u_t}$
de:={u_2x+2*u*u_x}$
cde({indep_var,dep_var,{},total_order},
   {principal_der,de,{},{}})$
```

9.2. **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,{})$
```

Note the empty list at the end; it playe a role only for computations involving odd variables.

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:

```
l_grad_var:=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

```
gradmon:=graded_mon(1,5,l_grad_var)$
gradmon:={1} . gradmon$
ansatz:=for each el in gradmon join el$
```

More precisely, the command `graded_mon` produces a list of monomials of degrees from `i` to `j`, formed from the list of graded variables `l_grad_var`; 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, $\ell_B$ stands for the linearization (Section 8). A function `sym` that fulfills the above equation, on account of $B = 0$, is an higher symmetry.

We **cannot** define the linearization as a $\mathcal{C}$-differential operator in this way:

```
bur:={u_t - (2*u*u_x+u_2x)};
ell_function(bur,lbur);
```

as the linearization is performed with respect to parametric derivatives only! This means that the linearization has to be computed beforehand in a free jet space, then it may be used here.

So, the right way to go is

```
mk_cdiffop(lbur,1,{1},1);
for all phi let lbur(1,1,phi)=td(phi,t)-td(phi,x,2)-2*u*td(phi,x)-2*u_x*phi;
```

Note that for evolutionary equations the restriction of the linearization to the equation is equivalent to just restricting total derivatives, which is automatic in CDE.

The equation becomes

```
equ 1:=lbur(1,1,sym);
```

At this point we initialize the equation solver. This is a part of the CDIFF package called `integrator.red` (see the original documentation inside the folder `packages/cdiff` in `Reduce`'s source code). In our case the above package will solve a large sparse linear system of algebraic equations on the coefficients of `sym`.

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_opequ` on the first component of `equ` in order to obtain equations on coefficiens of each monomial.

```
tel:=splitvars_opequ(equ,1,1,vars);
```

Note that `splitvars_opequ` needs to know the indices of the first and the last equation in `equ`, and here we have only one equation as `equ(1)`. The output `tel` is the final number of splitted equations, starting just after the initial equation `equ(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;
```

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.

9.3. **Solving the problem using CRACK.** CRACK is a PDE solver which is devoted mostly to the solution of overdetermined PDE systems [39, 41]. Several mathematical problems have been solved by the help of CRACK, like finding symmetries [38, 40] and conservation laws [37]. 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, we define the linearization of the equation `lbur`.

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 `lbur(ansatz)=0`, hence we give the command

```
total_eq:=lbur(1,1,ansatz)$
```

The above command will issue an error if the list {`total_eq`} depends on the flag variable `letop`. In this case the computation has to be redone within a jet space of higher order.

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.

## 10. 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 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:=td(fx,t)-td(ft,x)$
```

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-td(f0,x);
equ 2:=ft-td(f0,t);
```

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-td(f0,x);
ftnontriv := ft-td(f0,t);
```

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 13.3.

## 11. 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 all 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 [21] for evolution equations and was later extended to general differential equations in [23].

11.1. **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 [33]) of conservation laws to higher (or generalized) symmetries. As it is proved in [21], this amounts at solving $\bar{\ell}_{KdV}(\mathtt{phi}) = 0$ 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 $\ell^*$-covering of the KdV equation, with the further condition that the shadows must be linear in the $p$-variables. Note that the second equation (in odd variables!) is just the adjoint of the linearization of the KdV equation applied to an odd variable.

The file containing this example is `kdv_lho1`.

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

```
td(phi,t)-u*td(phi,x)-u_x*phi-td(phi,x,3)=0
```

but the total derivatives are lifted to the $\ell^*$ covering, hence they contain also derivatives with respect to $p$'s. We can define a linearization operator `lkdv` as usual.

In order to produce an ansatz which is a superfunction of one odd variable (or a linear function in odd variables) we produce two lists: the list `graadlijst` of all even variables collected by their gradings and a similar list `graadlijst_odd` for odd variables:

```
l_grad_var:=der_deg_ordering(0,all_parametric_der)$
l_grad_odd:={1} . der_deg_ordering(1,all_parametric_odd)$
gradmon:=graded_mon(1,10,l_grad_var)$
gradmon:={1} . gradmon$
```

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(gradmon,l_grad_odd,1,6)$
```

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:=lkdv(1,1,sym);
```

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

```
tel:=splitext_opequ(equ,1,1);
```

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

```
tel2:=splitvars_opequ(equ,2,tel,vars);
```

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. After integration the function `sym` becomes

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

Of course, the results correspond to the operators

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

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

```
ansatz := 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)=
   td(sym,t) - u*td(sym,x) - u_x*sym - td(sym,x,3)$
```

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)$
```

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

```
system_eq:=splitext_list({total_eq})$
```

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},
{}}}$
```

11.2. **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 dispersionless Boussinesq equation (13).

We will proceed by dimensional analysis. 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 odd coordinates. We have the $\ell^*_{\text{Bou}}$ covering equation

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

We have to find Hamiltonian operators as shadows of symmetries on the above covering. At the level of source file (`bou_lho1`) 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(gradmon,l_grad_odd,1,6)$
phi1:=(for each el in linodd sum (c(ctel:=ctel+1)*el))$
phi2:=(for each el in linodd sum (c(ctel:=ctel+1)*el))$
```

and the equation for shadows of symmetries is (`lbou2` is taken from Section 8)

```
equ 1:=lbou2(1,1,phi1) + lbou2(1,2,phi2);


equ 2:=lbou2(2,1,phi1) + lbou2(2,2,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 [21].

11.3. **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

$$u_{yy} = u_{tx} - u_x^2 - uu_{xx} - \frac{1}{12}u_{xxxx}. \tag{14}$$

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(gradmon,l_grad_odd,1,5)$
phi:=(for each el in linodd sum (c(ctel:=ctel+1)*el))$
```

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

```
equ 1:=td(phi,y,2) - td(phi,x,t) + 2*u_x*td(phi,x)
 + u_2x*phi + u*td(phi,x,2) + (1/12)*td(phi,x,4);
```

we get the only local Hamiltonian operator

```
phi := 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 [26]. Our mathematical theory of Hamiltonian operators for general differential equations [23] allows us to formulate and solve the problem for any presentation of the KP equation. Change of coordinate formulae could also be provided.

## 12. Examples of Schouten bracket of local Hamiltonian operators

Let $F = 0$ be a system of PDEs. Here $F \in P$, where $P$ is the module (in the algebraic sense) of vector functions $P = \{J^r(n,m) \to \mathbb{R}^k\}$.

The Hamiltonian operators which have been computed in the previous Section are differential operators sending generating functions of conservation laws into generating functions of symmetries for the above system of PDEs:

$$H \colon \hat{P} \to \varkappa \tag{15}$$

- $\hat{P} = \{J^r(n,m) \to (\mathbb{R}^k)^* \otimes \wedge^n T^*\mathbb{R}^n\}$ is the space of covector-valued densities,
- $\varkappa = \{J^r(n,m) \to \mathbb{R}^m\}$ is the space of generalized vector fields on jets; generating functions of higher symmetries of the system of PDEs are elements of this space.

As the operators are mainly used to define a bracket operation and a Lie algebra structure on conservation laws, two properties are required: skew-adjointness $H^* = -H$ (corresponding with skew-symmetry of the bracket) and $[H,H] = 0$ (corresponding with the Jacobi property of the bracket).

In order to compute the two properties we proceed as follows. Skew-adjointness is checked by computing the adjoint and verifying that the sum with the initial operator is zero.

In the case of evolutionary equations, $P = \varkappa$, and Hamiltonian operators (15) can also be interpreted as *variational bivectors, ie*

$$\hat{H} \colon \hat{\varkappa} \times \hat{\varkappa} \to \wedge^n T^*\mathbb{R}^n \tag{16}$$

where the correspondence is given by

$$H(\psi) = (a^{ij\sigma} D_\sigma \psi_j) \quad \to \quad \hat{H}(\psi_1, \psi_2) = (a^{ij\sigma} D_\sigma \psi_{1\ j} \psi_{2\ i}) \tag{17}$$

In terms of the corresponding superfunctions:

$$H = a^{ik\,\sigma} p_{k\,\sigma} \quad \to \quad \hat{H} = a^{ik\,\sigma} p_{k\,\sigma} p_i.$$

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

After that a $\mathcal{C}$-differential operator of the type of $H$ has been converted into a bivector it is possible to apply the formulae (11) and (12) in order to compute the Schouten bracket. This is what we will see in next section.

12.1. **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)=td(psi1,x);
operator ham2;
for all psi2 let ham2(psi2)=(1/3)*u_x*psi2 + td(psi2,x,3)
 + (2/3)*u*td(psi2,x);
```

We may convert the two operators into the corresponding superfunctions

```
conv_cdiff2superfun(ham1,sym1);
conv_cdiff2superfun(ham2,sym2);
```

The result of the conversion is

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

Skew-adjointness is checked at once:

```
adjoint_cdiffop(ham1,ham1_star);
adjoint_cdiffop(ham2,ham2_star);
ham1_star_sf(1)+sym1(1);
ham2_star_sf(1)+sym2(1);
```

and the result of the last two commands is zero.

Then we shall convert the two superfunctions into bivectors:

```
conv_genfun2biv(sym1_odd,biv1);
conv_genfun2biv(sym2_odd,biv2);
```

The output is:

```
biv1 :=  - ext(p,p_x);
biv2 := - (1/3)*( - 3*ext(p,p_3x) - 2*ext(p,p_x)*u);
```

Finally, the three Schouten brackets $[\hat{H}_i, \hat{H}_j]$ are computed, with $i, j = 1, 2$:

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

the result are well-known lists of zeros.

12.2. **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, $f_{ttt} = f_{xxt}^2 - f_{xxx}f_{xtt}$ [13]. 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 [16]:

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

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 [14, 15]. 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, $\Gamma_k^{ij} = g_1^{is}\Gamma_{sk}^j$ (here $g_1^{ij}$ is the inverse matrix that represent the contravariant metric induced by $g_{1\,ij}$), and $\Gamma_{sk}^j$ are the usual Christoffel symbols of $g_{1\,ij}$.

The operator $A_2$ has the general structure

$$A_2 = D_x \left( g_2^{ij} D_x + c_k^{ij} u_x^k \right) D_x, \tag{18}$$

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 [17] that, if we set $c_{ijk} = g_{iq} g_{jp} c_k^{pq}$, then

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

and the metric fulfills the following identity:

$$g_{mk,n} + g_{kn,m} + g_{mn,k} = 0. \tag{19}$$

This means that the metric is a Monge metric [17]. 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)*td(psi,x);
for all psi let a1(1,2,psi) = (1/2)*td(a*psi,x);
...
```

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 $\Gamma_k^{ij}$ and computed in the system using the inverse matrix $g_{ij}$ of the leading coefficient contravariant metric[3]

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

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)*td(psi,x)+(for k:=1:3 sum gamma_hi_con(i,j)*psi
)$
```

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 $c_k^{ij}$

```
templist:={}$
operator c_hi$
for i:=1:ncomp do
 for j:=1:ncomp do
  for k:=1:ncomp do
   c_hi(i,j,k):=
```

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

```
    <<
     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) =
td(
gu3(i,j)*td(psi,x,2)+c_hi_con(i,j)*td(psi,x)
,x)$
```

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

```
conv_cdiff2genfun(aa1,sym1)$
conv_cdiff2genfun(aa2,sym2)$

conv_genfun2biv(sym1,biv1)$
conv_genfun2biv(sym2,biv2)$

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

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 [34].

12.3. **Schouten bracket of multidimensional operators.** The formulae (11), (12) hold also in the case of multidimensional operators, *ie* operators with total derivatives in more than one independent variables. Here we give one Hamiltonian operator $H$ and we give two more variational bivectors $P_1$, $P_2$; all operators are of Dubrovin–Novikov type (homogeneous). We check the compatibility by computing $[H, P_1]$ and $[H, P_2]$. Such computations are standard for the problem of computing the Hamiltonian cohomology of $H$.

This example has been provided by M. Casati. The file of the computation is `dn2d_sb1.red`. The dependent variables are $p^1$, $p^2$.

Let us set

$$H = \begin{pmatrix} D_x & 0 \\ 0 & D_y \end{pmatrix} \tag{20}$$

(21)

$$P_1 = \begin{pmatrix} 2\frac{\partial g}{\partial p^1}p_y^2 D_x + \frac{\partial g}{\partial p^1}p_{xy}^2 + \frac{\partial g}{\partial p^1 \partial p^2}p_x^2 p_y^2 + \frac{\partial g}{\partial^2 p^1}p_x^1 p_y^2 \\ -fD_x^2 + gD_y^2 + \frac{\partial g}{\partial p^2}p_y^2 D_y - (\frac{\partial f}{\partial p^1}p_x^1 + 2\frac{\partial f}{\partial p^2}p_x^2)D_x - \frac{\partial f}{\partial^2 p^2}p_x^2 p_x^2 - \frac{\partial f}{\partial p^1 \partial p^2}p_x^1 p_x^2 - \frac{\partial f}{\partial p^2}p_2^2 x; \end{pmatrix}$$

(22)

$$\begin{pmatrix} fD_x^2 - gD_y^2 + \frac{\partial f}{\partial p^1}p_x^1 D_x - \Big(\frac{\partial g}{\partial p^2}p_y^2 + 2\frac{\partial g}{\partial p^1}p_y^1\Big)D_y - \frac{\partial g}{\partial^2 p^1}p_y^1 p_y^1 - \frac{\partial g}{\partial p^1 \partial p^2}p_y^1 p_y^2 - \frac{\partial g}{\partial p^1}p_{2y}^1; \\ 2\frac{\partial f}{\partial p^2}p_x^1 D_y + \frac{\partial f}{\partial p^2}p_x^1 y + \frac{\partial f}{\partial p^1 \partial p^2}p_x^1 p_y^1 + \frac{\partial f}{\partial^2 p^2}p_x^1 p_y^2; \end{pmatrix}$$

and let $P_2 = P_1^T$. This is implemented as follows:

```
mk_cdiffop(aa2,1,{2},2)$
for all psi let aa2(1,1,psi) = 2*df(g,p1)*p2_y*td(psi,x) +
  df(g,p1)*p2_xy*psi + df(g,p1,p2)*p2_x*p2_y*psi + df(g,p1,2)*p1_x*p2_y*psi;

for all psi let aa2(1,2,psi) = f*td(psi,x,2) - g*td(psi,y,2)
  + df(f,p1)*p1_x*td(psi,x) -
  (df(g,p2)*p2_y + 2*df(g,p1)*p1_y)*td(psi,y) -
  df(g,p1,2)*p1_y*p1_y*psi - df(g,p1,p2)*p1_y*p2_y*psi - df(g,p1)*p1_2y*psi;

for all psi let aa2(2,1,psi) = -f*td(psi,x,2)
  + g*td(psi,y,2) +
  df(g,p2)*p2_y*td(psi,y) -
  (df(f,p1)*p1_x+2*df(f,p2)*p2_x)*td(psi,x)
    - df(f,p2,2)*p2_x*p2_x*psi - df(f,p1,p2)*p1_x*p2_x*psi
      - df(f,p2)*p2_2x*psi;

for all psi let aa2(2,2,psi) = 2*df(f,p2)*p1_x*td(psi,y)
  + df(f,p2)*p1_xy*psi + df(f,p1,p2)*p1_x*p1_y*psi
    + df(f,p2,2)*p1_x*p2_y*psi;

mk_cdiffop(aa3,1,{2},2)$
for all psi let aa3(1,1,psi) = aa2(1,1,psi);
for all psi let aa3(1,2,psi) = aa2(2,1,psi);
for all psi let aa3(2,1,psi) = aa2(1,2,psi);
for all psi let aa3(2,2,psi) = aa2(2,2,psi);
```

Let us check the skew-adjointness of the above bivectors:

```
conv_cdiff2superfun(aa1,sym1)$
conv_cdiff2superfun(aa2,sym2)$
```

```
conv_cdiff2superfun(aa3,sym3)$

adjoint_cdiffop(aa1,aa1_star);
adjoint_cdiffop(aa2,aa2_star);
adjoint_cdiffop(aa3,aa3_star);

for i:=1:2 do write sym1(i) + aa1_star_sf(i);
for i:=1:2 do write sym2(i) + aa2_star_sf(i);
for i:=1:2 do write sym3(i) + aa3_star_sf(i);
```

Of course the last three commands produce two zeros each.

Let us compute Schouten brackets.

```
conv_cdiff2superfun(aa1,sym1)$
conv_cdiff2superfun(aa2,sym2)$
conv_cdiff2superfun(aa3,sym3)$

conv_genfun2biv(sym1,biv1)$
conv_genfun2biv(sym2,biv2)$
conv_genfun2biv(sym3,biv3)$

schouten_bracket(biv1,biv1,sb11);

schouten_bracket(biv1,biv2,sb12);

schouten_bracket(biv1,biv3,sb13);
```

`sb11` is trivially a list of zeros, while `sb12` is nonzero and `sb13` is again zero.

More formulae are currently being implemented in the system, like symplecticity and Nijenhuis condition for recursion operators [22]. Interested readers are warmly invited to contact R. Vitolo for questions/feature requests.

## 13. 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 particular, it is still missing a theoretical framework for Schouten brackets of nonlocal opeartors in the odd variable language.

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 [11] for theoretical details on coverings).

In Section 13.3 we will also discuss a procedure for finding conservation laws from their generating functions that is of independent interest.

### 13.1. 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 [21].

We have to solve equations of the type `ddx(ct)-ddt(cx)` as in 10. The main difference is that we will attempt a solution on the $\ell^*$-covering (see Subsection 11). 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 $\ell^*$-covering.

As a first step, let us compute conservation laws on the $\ell^*$-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 [21]. 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` as follows:

```
c1x:=(t*u_x+1)*p$ % degree 1
c2x:=u_x*p$ % degree 4
c3x:=(u*u_x+u_3x)*p$ % degree 6
```

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

```
c1t:=f1*p+f2*p_x+f3*p_2x$
c2t:=(for each el in linodd6 sum (c(ctel:=ctel+1)*el))$ % degree 6
c3t:=(for each el in linodd8 sum (c(ctel:=ctel+1)*el))$ % degree 8
```

where we already introduced the sets `linodd6` and `linodd8` 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:=td(c1t,x) - td(c1x,t);
```

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

```
f3:=t*u_x+1$
f2:=-td(f3,x)$
f1:=u*f3+td(f3,x,2)$
```

We also have the equations

```
equ 2:=td(c2t,x)-td(c2x,t);
equ 3:=td(c3t,x)-td(c3x,t);
```

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 $\ell^*$-covering (source file `kdv_nlho1`). 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 `r1`, `r2`, `r3`:

```
principal_odd:={p_t,r1_x,r1_t,r2_x,r2_t,r3_x,r3_t}$
de_odd:={u*p_x+p_3x,
p*(t*u_x + 1),
p*t*u*u_x + p*t*u_3x + p*u + p_2x*t*u_x + p_2x - p_x*t*u_2x,
```

```
p*u_x,
p*u*u_x + p*u_3x + p_2x*u_x - p_x*u_2x,
p*(u*u_x + u_3x),
p*u**2*u_x + 2*p*u*u_3x + 3*p*u_2x*u_x + p*u_5x + p_2x*u*u_x + p_2x*
u_3x - p_x*u*u_2x - p_x*u_4x - p_x*u_x**2}$
```

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 linodd 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 $\ell^*$-covering

```
equ 1:=td(phi,t)-u*td(phi,x)-u_x*phi-td(phi,x,3);
```

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

```
phi := (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 [21]. The CRACK approach also holds for non-local computations.

13.2. **Non-local recursion operator for the Korteweg–de Vries equation.** Following the ideas in [21], a differential operator that sends symmetries into symmetries can be found as a shadow of symmetry on the $\ell$-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 [25].

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 $\omega$, 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–Magri operator

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

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

$$F = u_{tt}u_{xx} - u_{tx}^2 + u_{xz} + u_{ty} = 0 \tag{23}$$

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 [28]. 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 [25] for a theoretical description). We can guess that the generating function of the conservation law is $\psi = (0, 1)$. This comes from the generating function $\varphi = 1$ of the $u$-translation symmetry by the correspondence that is described in [25]. Then we should find the corresponding conservation law by solving a system of PDEs. Luckily it is not difficult to realize that the above equation can be written in explicit conservative form as

$$\begin{aligned} p_{xz} + p_{ty} + u_{tt}p_{xx} + u_{xx}p_{tt} - 2u_{tx}p_{tx} \\ = D_x(p_z + u_{tt}p_x - u_{tx}p_t) + D_t(p_y + u_{xx}p_t - u_{tx}p_x) = 0, \end{aligned}$$

thus the corresponding conservation law is

$$\upsilon(1) = (p_y + u_{xx}p_t - u_{tx}p_x)\, dx \wedge dy \wedge dz + (u_{tx}p_t - p_z - u_{tt}p_x)\, dt \wedge dy \wedge dz. \tag{24}$$

We can introduce a potential $r$ for the above 2-component conservation law. Namely, we can assume that

$$r_x = p_y + u_{xx}p_t - u_{tx}p_x, \quad r_t = u_{tx}p_t - p_z - u_{tt}p_x. \tag{25}$$

This is a new nonlocal variable for the (co)tangent covering of the Plebanski equation. We can load the Plebanski equation together with its nonlocal variable $r$ as follows:

```
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_z - u_2t*p_x + u_tx*p_t$
rx:= p_y + u_2x*p_t - u_tx*p_x$
% 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 can easily verify that the integrability condition for the new nonlocal variable holds:

```
td(r,t,x) - td(r,x,t);
```

the result is 0.

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 [28], coincides with one of the nonlocal Hamiltonian operators presented in that paper[4]. Here follows the code.

```
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(gradmon,l_grad_odd,1,4)$
phi:=(for each el in linodd sum (c(ctel:=ctel+1)*el))$
```

[4]We observe that in [28] also the trivial Hamiltonian operator is recovered in the evolutionary presentation; of course it has an apparently nontrivial expression.

then we solve the equation of shadows of symmetries:

```
equ 1:=td(phi,x,z)+td(phi,t,y)-2*u_tx*td(phi,t,x)
+u_2x*td(phi,t,2)+u_2t*td(phi,x,2)$
```

The solution is

```
phi := c(28)*r + c(1)*p
```

hence we obtain the Noether map (the identity operator $p$) and the new nonlocal operator $r$. 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 [28]. More details on this computation can be found in [25].

## 14. Appendix: old versions of CDE

A short version history is provided here.

CDE 1.0. This version was published in October 2014. It was programmed in `Reduce`'s **algebraic mode**, so its capabilities were limited, and its speed was severely affected by the systematic use of the package `assist` for manipulating algebraic lists. Its features were:

(1) CDE 1.0 is able to do standard computations in integrable systems like determining systems for generalized symmetries and conservation laws.
(2) CDE 1.0 is able to compute linear overdetermined systems of partial differential equations whose solutions are Hamiltonian operators.
(3) CDE is able to compute Schouten brackets between bivectors. This can be used *eg* to check Hamiltonianity of an operator, or the compatibility of two operators.

CDE 1.0 has never ben included in the official `Reduce` distribution, and it is still available at [2].

## 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] 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).
[13] B.A. Dubrovin, Geometry of 2D topological field theories, Lecture Notes in Math. 1620, Springer-Verlag (1996) 120–348.
[14] 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.
[15] B.A. Dubrovin and S.P. Novikov, Poisson brackets of hydrodynamic type, Soviet Math. Dokl. **30** No. 3 (1984), 651–2654.
[16] 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.
[17] 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.
[18] E.V. Ferapontov, M.V. Pavlov, R.F. Vitolo, *Towards the classification of homogeneous third-order Hamiltonian operators*, http://arxiv.org/abs/1508.02752
[19] *E. Getzler*, A Darboux theorem for Hamiltonian operators in the formal calculus of variations, Duke J. Math. **111** (2002), 535-560.
[20] S. Igonin, A. Verbovetsky, R. Vitolo: *Variational Multivectors and Brackets in the Geometry of Jet Spaces*, V Int. Conf. on on Symmetry in Nonlinear Mathematical Physics, Kyiv 2003; Part 3 of Volume 50 of Proceedings of Institute of Mathematics of NAS of Ukraine, Editors A.G. Nikitin, V.M. Boyko, R.O. Popovych and I.A. Yehorchenko (2004), 1335–1342; http://www.imath.kiev.ua/~snmp2003/Proceedings/vitolo.pdf.
[21] P.H.M. Kersten, I.S. Krasil'shchik, A.M. Verbovetsky, *Hamiltonian operators and $\ell^*$-covering*, Journal of Geometry and Physics **50** (2004), 273–302.
[22] 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.
[23] 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.
[24] 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.
[25] I. Krasil′shchik, A. Verbovetsky, R. Vitolo, *A unified approach to computation of integrable structures*, Acta Appl. Math. (2012).
[26] 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.
[27] M. Marvan, *Sufficient set of integrability conditions of an orthonomic system*. Foundations of Computational Mathematics **9** (2009), 651–674.
[28] 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.
[29] A.C. Norman, R. Vitolo, *Inside Reduce*, part of the official `Reduce` documentation included in the source code, see below.
[30] 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.
[31] 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.

[32] F. Oliveri, ReLie, `Reduce` software and user guide, http://mat521.unime.it/oliveri/.
[33] P. Olver, Applications of Lie Groups to Partial Differential Equations, 2nd ed, GTM Springer, 1992.
[34] M.V. Pavlov, R.F. Vitolo: *On the bi-Hamiltonian geometry of the WDVV equations*, `http://arxiv.org/abs/1409.7647`
[35] 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.
[36] `Reduce` official website: http://reduce-algebra.sourceforge.net/.
[37] T. Wolf, *A comparison of four approaches to the calculation of conservation laws*, Euro. Jnl of Applied Mathematics 13 part 2 (2002) 129-152.
[38] T. Wolf, *APPLYSYM - a package for the application of Lie-symmetries*, software distributed together with the computer algebra system REDUCE, (1995).
[39] T. Wolf, A. Brand, *Investigating DEs with CRACK and Related Programs*, SIGSAM Bulletin, Special Issue, (June 1995), p 1-8.
[40] 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.
[41] 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 and Physics "E. De Giorgi", Università del Salento, via per Arnesano, 73100 Lecce, Italy
*E-mail address*: `raffaele.vitolo@unisalento.it`