EDS
A package for exterior differential systems

David Hartley
Physics and Mathematical Physics
University of Adelaide SA 5005
Australia
DHartley@physics.adelaide.edu.au

Version 2.1

May 3, 1999

Abstract

EDS is a REDUCE package for symbolic analysis of partial differential equations using the geometrical approach of exterior differential systems. The package implements much of exterior differential systems theory, including prolongation and involution analysis, and has been optimised for large, non-linear problems.

Contents

1 Introduction

Exterior differential systems give a geometrical framework for partial differential equations and more general differential geometric problems. The geometrical formulation has several advantages stemming from its coordinate-independence, including superior treatment of nonlinear and global problems. There is not sufficient space in this manual for an introduction to exterior differential systems beyond the scant details given in section 2, but there are a number of up-to-date texts on the subject (eg [2, 10]).

EDS provides a number of tools for setting up and manipulating exterior differential systems and implements many features of the theory. Its main strengths are the ability to use anholonomic or moving frames and the care taken with nonlinear problems.

There has long been interest in implementing the theory of exterior differential systems in a computer algebra system (eg [1, 3, 4]). The EDS package owes much to these earlier efforts, and also to related packages for PDE analysis (eg [6, 7, 9]), as well as to earlier versions of EDS produced at Lancaster university with R W Tucker and P A Tuckey. Finally, EDS uses the exterior calculus package EXCALC of E Schrüfer [8] and the exterior ideals package XIDEAL [5]. XIDEAL and EXCALC are loaded automatically with EDS.

This work has been supported by the Graduate College on Scientific Computing, University of Cologne and GMD St Augustin, funded by the DFG (Deutsche Forschungsgemeinschaft). I would like to express my thanks to R W Tucker, E Schrüfer, P A Tuckey, F W Hehl and R B Gardner for helpful and encouraging discussions.

2 EDS data structures and concepts

This section presents the various structures used for expressing exterior systems quantities in EDS. In addition, some the concepts used in EDS to aid computation are described.

2.1 Coframings

Within the context of EDS, a *coframing* means a real finite-dimensional differentiable manifold with a given global cobasis. The information about a coframing required by EDS is kept in a ⟨*coframing*⟩ object. The cobasis is the identifying element of an EDS ⟨*coframing*⟩: distinct cobases for the same differentiable manifold are treated as distinct ⟨*coframing*⟩ objects in EDS. The cobasis may be either holonomic or anholonomic, allowing some manifolds with non-trivial topology (eg. group manifolds) to be treated.

In addition to the cobasis, an EDS ⟨*coframing*⟩ can be given *coordinates*, *structure equations* and *restrictions*. The coordinates may be an incomplete or overcomplete set. The structure equations express the exterior derivative of the coordinates and cobasis elements as needed. All coordinate differentials must be expressed in terms of the given cobasis, but not all cobasis differentials need be known. The restrictions are a set of inequalities (at present using just $\neq$) describing point sets not in the manifold.

The ⟨*coframing*⟩ object is, of course, by no means a full description of a differentiable manifold. For example, there is no topology and there are no charts. However, the ⟨*coframing*⟩ object carries sufficient information about the underlying manifold to allow a range of exterior systems calculations to be carried out. As such, it is convenient to accept an abuse of language and think of the ⟨*coframing*⟩ object as a manifold.

A ⟨*coframing*⟩ is constructed or selected using the `coframing` operator.

Examples:

- $\mathbf{R}^3$ with cobasis $\{\mathrm{d}x, \mathrm{d}y, \mathrm{d}z\}$ and coordinates $\{x, y, z\}$.
- $\mathbf{R}^2\backslash\{0\}$ with cobasis $\{e^1, e^2\}$, a single coordinate $\{r\}$, "structure equations" $\{\mathrm{d}r = e^1$, $\mathrm{d}e^1 = 0$,$\mathrm{d}e^2 = e^1 \wedge e^2/r\}$ and restrictions $\{r \neq 0\}$.
- $\mathbf{R}^2\backslash\{0\}$ with cobasis $\{\mathrm{d}x, \mathrm{d}y\}$, coordinates $\{x, y\}$ and restrictions $\{x^2+$

$y^2 \neq 0\}$.

- S^1 with cobasis $\{\omega\}$ and structure equations $\{\mathrm{d}\omega = 0\}$.
- S^2 cannot be encapsulated by an EDS ⟨*coframing*⟩ since there is no global cobasis.

2.2 Exterior differential systems

A simple ⟨*EDS*⟩, or exterior differential system, is a triple (S, Ω, M), where M is a ⟨*coframing*⟩ (section 2.1), S is a ⟨*system*⟩ (section 2.3) on M, and Ω is an independence condition: either a decomposable ⟨*p-form*⟩ or a ⟨*system*⟩ of 1-forms on M (exterior differential systems without independence condition are not treated by EDS).

More generally, an ⟨*EDS*⟩ is a list of simple ⟨*EDS*⟩ objects where the various coframings are all disjoint. This last requirement in not enforced within EDS unless the `edsdisjoint` switch is `on` (section 8.4). These more general ⟨*EDS*⟩ objects are represented as a list of simple ⟨*EDS*⟩ objects. All operators which take an ⟨*EDS*⟩ argument accept both simple and compound types.

The trivial ⟨*EDS*⟩, describing an inconsistent problem with no solutions, is defined to be ({1},{},{}).

An ⟨*EDS*⟩ is represented by the `eds` operator (section 3.2), and can additionally be generated using the `contact` and `pde2eds` operators (sections 3.3, 3.4).

The solutions of (S, Ω, M) are integral manifolds, or immersions (cf section 2.7) on which S vanishes and the rank of Ω is preserved. Solutions at a single point are described by integral elements (section 2.5).

2.3 Systems

In EDS, the label ⟨*system*⟩ refers to a list

{⟨*p-form expr*⟩`,`···}

of differential forms. This is distinct from an ⟨*EDS*⟩ (section 2.2), which has additional structure. However, many EDS operators will accept either an ⟨*EDS*⟩ or a ⟨*system*⟩ as arguments. In the latter case, any extra information which is required is taken from the background coframing (section 2.4).

The $\langle$*system*$\rangle$ of an $\langle$*EDS*$\rangle$ can be obtained with the `system` operator (section 4.5).

2.4 Background coframing

The information encapsulated in a `coframing` operator is usually inactive. However, when operations are performed on a $\langle$*coframing*$\rangle$ or an $\langle$*EDS*$\rangle$ object (sections 2.1, 2.2), this information is activated for the duration of those operations. It is possible to activate the rules and orderings of a `coframing` operator globally, by making it the *background coframing.* All subsequent EXCALC operations will be governed by those rules. Operations on $\langle$*EDS*$\rangle$ objects are unaffected, since their coframings are still activated locally. The background coframing can be set and changed with the `set_coframing` command, and inspected using `coframing`.

2.5 Integral elements

An *integral element* of an exterior system (S, Ω, M) is a subspace $P \subset T_pM$ of the tangent space at some point $p \in M$ such that all forms in S vanish when evaluated on vectors from P. In addition, no non-zero vector in P may annul every form in Ω.

Alternatively, an integral element $P \subset T_pM$ can be represented by its annihilator $P^\perp \subset T_p^*M$, comprising those 1-forms at p which annul every vector in P. This can also be understood as a maximal set of 1-forms at p such that $S \simeq 0 \pmod{P^\perp}$ and the rank of Ω is preserved modulo $P^\perp$. This is the representation used by EDS. Further, the reference to the point p is omitted, so an $\langle$*integral element*$\rangle$ in EDS is a distribution of 1-forms on M, specified as a $\langle$*system*$\rangle$ of 1-forms.

In specifying an integral element for a particular $\langle$*EDS*$\rangle$, it is possible to omit the Pfaffian component of the $\langle$*EDS*$\rangle$, since these 1-forms must be part of any integral element.

Examples:

- With $M = \mathbf{R}^3 = \{(x, y, z)\}$, $S = \{\mathrm{d}x \wedge \mathrm{d}z\}$ and $\Omega = \{\mathrm{d}x, \mathrm{d}y\}$, the integral element $P = \{\partial_x + \partial_z, \partial_y\}$ is equally determined by its annihilator $P^\perp = \{\mathrm{d}z - \mathrm{d}x\}$.

- For $S = \{\mathrm{d}z - y\mathrm{d}x\}$ and $\Omega = \{\mathrm{d}x\}$, the integral element $P = \{\partial_x + y\partial_z\}$ can be specified simply as $\{\mathrm{d}y\}$.

2.6 Properties

For large problems, it can require a great deal of computation to establish whether, for example, a system is closed or not. In order to save recomputing such properties, an ⟨*EDS*⟩ object carries a list of ⟨*properties*⟩ of the form

{⟨*keyword*⟩ `=` ⟨*value*⟩`,`···}

where ⟨*keyword*⟩ is one of `closed`, `quasilinear`, `pfaffian` or `involutive`, and ⟨*value*⟩ is either `0` (false) or `1` (true). These properties are suppressed when an ⟨*EDS*⟩ is printed, unless the `nat` switch is `off`. They can be examined using the `properties` operator (section 4.7).

Properties are usually generated automatically by EDS as required, but may be explicitly checked using the operators in section 7. If a property is not yet present on the list, it is not yet known, and must be checked explicitly if required.

In addition to the properties just described, an ⟨*EDS*⟩ object carries a number of hidden properties which record the results of previous calculations, such as the closure or information about the prolongation of the system. These hidden properties speed up many operations which contain common sub-calculations. The hidden properties are stored using internal LISP data structures and so are not available for inspection.

Properties can be asserted when an ⟨*EDS*⟩ is constructed with the `eds` operator (section 3.2). Care is needed since such assertions are never checked. Properties can be erased using the `cleanup` operator (section 9.8).

2.7 Maps

Within EDS, a map $f : M \to N$ is given as a ⟨*map*⟩ object, a list

{⟨*coordinate*⟩ `=` ⟨*expr*⟩`,`···`,`⟨*expr*⟩ `neq` ⟨*expr*⟩`,`···}

of substitutions and restrictions. The substitutions express coordinates on the target manifold N in terms of those on the source manifold M. The restrictions describe point sets not contained in the source manifold M. The ordering of substitutions and restrictions in the list is unimportant. It is not necessary that the restrictions and right-hand sides of the substitutions be written entirely in M coordinates, but it must be possible by repeated substitution to produce expressions on M (see the examples below). Any denominators in the substitutions are automatically added to the list of restrictions. It is not necessary to include trivial equations for coordinates which are present on both M and N. Note that projections cannot be represented in this fashion (but see the `cross` operator, section 5.2).

Maps are applied using the `pullback` and `restrict` operators (sections 5.3, 5.4).

Examples:

- The map $\mathbf{R}^2\backslash\{0\} \to \mathbf{R}^3$, $(x, y) \mapsto (x, y, z = x^2 + y^2)$ is represented $\{z = x^2 + y^2, z \neq 0\}$.
- $\{x = u+v, y = u-v\}$ might represent the coordinate change $\mathbf{R}^3 \to \mathbf{R}^3$, $(u, v, z) \mapsto (x = u + v, y = u - v, z)$.
- $\{x = u + v, y = 2u - x\}$ is the same map again.
- $\{x = 2v + y, y = 2u - x\}$ is unacceptable since x and y cannot be eliminated from the right-hand sides by repeated substitution.

2.8 Cobasis transformations

A cobasis transformation is given in EDS by a $\langle$*transform*$\rangle$, a list

$\{\langle$*cobasis element*$\rangle$ `=` $\langle$*1-form expr*$\rangle$`,`$\cdots\}$

of substitutions. When applying a transformation to a $\langle$*p-form*$\rangle$ or $\langle$*system*$\rangle$, it is necessary to specify the *forward* transformation just as for a `sub` substitution. For $\langle$*EDS*$\rangle$ and $\langle$*coframing*$\rangle$ objects, it is also possible to specify the inverse of the desired substition: EDS will automatically invert the transformation as required. For a partial change of cobasis, it is not necessary to include trivial equalities. Cobasis transformations are applied by the `transform` operator (section 5.5).

Examples:

- $\{\omega^1 = x\mathrm{d}y - y\mathrm{d}x, \omega^2 = x\mathrm{d}x + y\mathrm{d}y\}$ gives a transformation between Cartesian and polar cobases on $\mathbf{R}^2\backslash\{0\}$.
- On $J^1(\mathbf{R}^2, \mathbf{R})$ with cobasis $\{\mathrm{d}u, \mathrm{d}p, \mathrm{d}q, \mathrm{d}r, \mathrm{d}s, \mathrm{d}t, \mathrm{d}x, \mathrm{d}y\}$, the list $\{\theta^1 = \mathrm{d}u - p\mathrm{d}x - q\mathrm{d}y, \theta^2 = \mathrm{d}p - r\mathrm{d}x - s\mathrm{d}y, \theta^3 = \mathrm{d}q - s\mathrm{d}x - t\mathrm{d}y\}$ specifies a new cobasis in which the contact system is simply $\{\theta^1, \theta^2, \theta^3\}$.

2.9 Tableaux

For a quasilinear Pfaffian exterior differential system $(\{\theta^a\}, \{\omega^i\}, M)$, the tableau $A = [\pi^a_i]$ is a matrix of 1-forms such that

$$\mathrm{d}\theta^a + \pi^a_i \wedge \omega^i \simeq 0 \quad (\mathrm{mod}\ \{\theta^a, \omega^i \wedge \omega^j\})$$

The π^a_i are not unique: if $\{\theta^a, \pi^\rho, \omega^i\}$ is a standard cobasis for the system (section 2.11), the EDS ⟨*tableau*⟩ is a matrix containing linear combinations of the π^ρ only. Zero rows are omitted.

The tableau of an ⟨*EDS*⟩ is generated by the `tableau` operator (section 6.12), or can be entered using the `mat` operator. The Cartan characters of a tableau are found using `characters` (section 6.3).

2.10 Normal form

Parts of the theory of exterior differential systems apply only at points on the underlying manifold where the system is in some sense non-singular. To ensure the theory applies, EDS automatically works all exterior systems (S, Ω, M) into a *normal form* in which

1. The Pfaffian (degree 1) component of S is in *solved* form, where each expression has a distinguished term with coefficient 1, unique to that expression.
2. The independence condition Ω is also in solved form.
3. The distinguished terms from the 1-forms in S have been eliminated from the rest of S and from Ω.
4. Any 1-forms in S which vanish modulo the independence condition are removed from the system and their coefficients are appended as 0-forms.

Conditions 1 and 2 ensure the 1-forms have constant rank, while 3 is convenient for many tests and calculations. In bringing the system into solved form, divisions will be made only by coefficients which are constants, parameters or functions which are nowhere zero on the manifold. The test for nowhere-zero functions uses the restrictions component of the ⟨*coframing*⟩ structure (cf section 2.1) and is still primitive: facts such as $x^2 + 1 \neq 0$ on a real manifold are overlooked. See also the switch `edssloppy` (section 8.3).

This "normal form" has, of course, nothing to do with the various normal forms (eg Goursat) into which some exterior systems may be brought by cobasis transformations and choices of generators.

Examples:

- On $M = \{(u, v, w) \in \mathbf{R}^3 \mid u \neq v\}$, the Pfaffian system

  $$\{u\mathrm{d}u + v\mathrm{d}v + \mathrm{d}w,\ (u^2 + u - v^2)\mathrm{d}u + u\mathrm{d}v + \mathrm{d}w\}$$

  has the solved form

  $$\{\mathrm{d}v + (u + v)\mathrm{d}u,\ \mathrm{d}w + (-uv + u - v)\mathrm{d}u\}.$$

- Since the independence condition is defined only modulo the system, the system

  $$S = \{\mathrm{d}u - \mathrm{d}x - u_y\mathrm{d}y\}, \quad \Omega = \mathrm{d}x \wedge \mathrm{d}y$$

  has an equivalent normal form

  $$S = \{\mathrm{d}x - \mathrm{d}u + u_y\mathrm{d}y\}, \quad \Omega = \mathrm{d}u \wedge \mathrm{d}y.$$

2.11 Standard cobasis

Given an ⟨*EDS*⟩ (S, Ω, M) in normal form (section 2.10), the cobasis of the ⟨*coframing*⟩ M can be decomposed into three sets: $\{\theta^a\}$, the distinguished terms from the 1-forms in S, $\{\omega^i\}$, the distinguished terms from the 1-forms in Ω, and the remainder $\{\pi^\rho\}$. Within EDS, $\{\theta^a, \pi^\rho, \omega^i\}$ is called the *standard cobasis*, and all expressions are ordered so that $\theta^a > \pi^\rho > \omega^i$. The ordering within the three sets is determined by the REDUCE ⟨*kernel*⟩ ordering.

Examples:

- For the system $S = \{\mathrm{d}u - \mathrm{d}x - u_y\mathrm{d}y\}$, $\Omega = \mathrm{d}x \wedge \mathrm{d}y$, the decomposed standard cobasis is $\{\mathrm{d}u\} \cup \{\mathrm{d}u_y\} \cup \{\mathrm{d}x, \mathrm{d}y\}$.
- For the contact system

$$S = \begin{cases} \mathrm{d}u - u_x\mathrm{d}x - u_y\mathrm{d}y \\ \mathrm{d}u_x - u_{xx}\mathrm{d}x - u_{xy}\mathrm{d}y \\ \mathrm{d}u_y - u_{xy}\mathrm{d}x - u_{yy}\mathrm{d}y, \end{cases}$$

  the standard cobasis is $\{\mathrm{d}u, \mathrm{d}u_x, \mathrm{d}u_y\} \cup \{\mathrm{d}u_{xx}, \mathrm{d}u_{xy}, \mathrm{d}u_{yy}\} \cup \{\mathrm{d}x, \mathrm{d}y\}$.

3 Constructing EDS objects

Before analysing an exterior system, it is necessary to enter it into EDS somehow. Several means are provided for this purpose, and are described in this section.

3.1 coframing

An EDS ⟨*coframing*⟩ is constructed using the `coframing` operator. There are several ways in which it can be used.

The simplest syntax

`coframing({`⟨*expr*⟩`,`$\cdots$`})`

examines the argument for 0-form and 1-form variables and deduces a full ⟨*coframing*⟩ object capable of supporting the given expressions. This includes recursively examining the exterior derivatives of the variables appearing explicitly in the argument, taking into account prevailing `let` rules. In this form, the ordering of the final cobasis elements follows the prevailing REDUCE ordering. Free indices in indexed expressions are expanded to a list of explicit indices using `index_expand` (section 9.4).

A more basic syntax is

`coframing(`⟨*cobasis*⟩ `[,`⟨*coordinates*⟩`]` `[,`⟨*restrictions*⟩`]`
`[,`⟨*structure equations*⟩`])`

where ⟨*cobasis*⟩ is a list of ⟨*kernel*⟩ 1-forms, ⟨*coordinates*⟩ is a list of ⟨*kernel*⟩ 0-forms, ⟨*restrictions*⟩ is a list of inequalities (using only $\neq$ at present), and

⟨*structure equations*⟩ is a list of rules giving the exterior derivatives of the coordinates and cobasis elements. All arguments except the cobasis are optional, and the order of arguments is unimportant. As in the first syntax, missing parts are deduced. The ordering of the final cobasis elements follows the ordering specified, rather than the prevailing REDUCE ordering.

Finally,

`coframing(`⟨*EDS*⟩`)`

returns the coframing argument of an ⟨*EDS*⟩, and

`coframing()`

returns the current background coframing (section 2.4).

Examples:

```
coframing {x,y,z};

   coframing({d x,d y,d z},{x,y,z},{},{})

coframing({e 1,e 2},{r},{r neq 0},
          {d r=>e 1,d e 1=>0,d e 2=>e 1^e 2/r});

                                                1   2
               1   2           1        2     e ^e           1
   coframing({e ,e },{r},{d e => 0,d e => -------,d r => e },
                                                 r
             {r neq 0})

coframing({e 2}) where {d r=e 1,d e 1=0,d e 2=e 1^e 2/r};

                                                1   2
               1   2           1        2     e ^e           1
   coframing({e ,e },{r},{d e => 0,d e => -------,d r => e },
                                                 r
             {r neq 0})
```

3.2 eds

A simple ⟨*EDS*⟩ is constructed using the `eds` operator.

`eds(`⟨*system*⟩`,`⟨*indep. condition*⟩ [`,`⟨*coframing*⟩] [`,`⟨*properties*⟩] `)`

(cf sections 2.3, 2.1, 2.6). The ⟨*indep. condition*⟩ can be either a decomposable ⟨*p-form*⟩ or a ⟨*system*⟩ of 1-forms. Free indices in indexed expressions are expanded to a list of explicit indices using `index_expand` (section 9.4).

The ⟨*coframing*⟩ argument can be omitted, in which case the expressions from the ⟨*system*⟩ and ⟨*indep. condition*⟩ are fed to the `coframing` operator (section 3.1) to construct a suitable working space.

The ⟨*properties*⟩ argument is optional, allowing the given properties to be asserted. This can save considerable time for large systems, but care is needed since the assertions are never checked.

The ⟨*EDS*⟩ is put into normal form (section 2.10) before being returned.

On output, only the ⟨*system*⟩ and ⟨*indep. condition*⟩ are displayed, unless the `nat` switch is off, in which case the ⟨*coframing*⟩ and ⟨*properties*⟩ are shown too. This is so that an ⟨*EDS*⟩ can be written out to a file and read back in.

The parts of an ⟨*EDS*⟩ are obtained with the operators `system`, `cobasis`, `independence` and `properties` (sections 4.5, 4.1, 4.6 and 4.7).

Examples:

```
pform {x,y,z,p,q}=0,{e(i),w(i,j)}=1;

indexrange {i,j,k}={1,2},{a,b,c}={3};

eds({d z - p*d x - q*d y, d p^d q},{d x,d y});

   EDS({d z - p*d x - q*d y,d p^d q},{d x,d y})

OMrules :=
   index_expand {d e(i)=>-w(i,-j)^e(j),w(i,-j)+w(j,-i)=>0}$

eds({e(a)},{e(i)}) where OMrules;
```

```
         3    1  2
   EDS({e },{e ,e })

coframing ws;
               3  2    1  2         1        2  2
   coframing({e ,w    ,e ,e },{},{d e  =>  - e ^w    ,
                   1                            1
                 2     1  2
               d e  => e ^w   },{})
                          1
```

3.3 contact

Many PDE problems are formulated as exterior systems using a jet bundle contact system. To facilitate construction of these systems, the `contact` operator is provided. The syntax is

`contact(`⟨*order*⟩`,`⟨*source manifold*⟩`,`⟨*target manifold*⟩`)`

where ⟨*order*⟩ is a non-negative integer, and the two remaining arguments are either ⟨*coframing*⟩ objects or lists of ⟨*p-form*⟩ expressions. In the latter case, the expressions are fed to the `coframing` operator (section 3.1). The contact system for the bundle $J^r(M,N)$ of r-jets of maps $M \to N$ is thus returned by `contact(r,M,N)`. Source and target spaces may have anholonomic cobases. Indexed names for the jet bundle fibre coordinates are constructed using the identifiers in the source and target cobases.

Examples:

```
pform {x,y,z,u,v}=0,{e i,w a}=1;
indexrange {i}={1,2},{a}=1;
contact(1,{x,y,z},{u,v});

   EDS({d u - u *d x - u *d y - u *d z,
               x        y        z
        d v - v *d x - v *d y - v *d z},{d x,d y,d z})
               x        y        z

contact(2,{e(i)},{w(a)})
```

```
where index_expand{d e(1)=>e(1)^e(2),d e(2)=>0,d w(a)=>0};

      1    1    1    1    2
EDS({w  - w   *e  - w   *e ,
           1         2
        1      1      1    1      2
    d w    - w     *e  - w     *e ,
          1      1 1         1 2
        1          1        1     1    1      2    1  2
    d w    + ( - w      + w   )*e  - w     *e },{e ,e })
          2          1 2      1          2 2
```

3.4 pde2eds

A PDE system can be encoded into an ⟨*EDS*⟩ using `pde2eds`. The syntax is

`pde2eds(`⟨*pde*⟩ [`,`⟨*dependent*⟩`,`⟨*independent*⟩] `)`

where ⟨*pde*⟩ is a list of equations or expressions (implicitly assumed to vanish) specifying the PDE system using either the standard REDUCE `df` operator, or the EXCALC `@` operator. If the optional variable lists ⟨*dependent*⟩ and ⟨*independent*⟩ are not given, `pde2eds` infers them from the expressions in ⟨*pde*⟩. The order of each dependent variable is determined automatically.

The result returned by `pde2eds` is an ⟨*EDS*⟩ based on the contact system of the relevant mixed-order jet bundle. Any of the ⟨*pde*⟩ members which is in solved form is used to pull back this contact system. Any remaining expressions or unresolved equations are simply appended as 0-forms: before many of the analysis tools (section 6) can be applied, it is necessary to convert this to a system generated in positive degree using the `lift` operator (section 5.6).

The automatic inference of dependent and independent variables is governed by the following rules. The independent variables are all those with respect to which derivatives appear. The dependent variables are those for which explicit derivatives appear, as well as any which have dependencies (as declared by `depend` or `fdomain`) or which are 0-forms. To exclude a variable from the dependent variable list (for example, because it is regarded as given) or to include extra independent variables, use the optional arguments to pde2eds.

One of the awkward points about pde2eds is that implicit dependence is changed globally. In order for the df and @ operators to be used to express the PDE, the ⟨*dependent*⟩ variables must depend (via depend or fdomain) on the ⟨*independent*⟩ variables. On the other hand, in the ⟨*EDS*⟩, these variables are all completely independent coordinates. The pde2eds operator thus removes the implicit dependence so that the ⟨*EDS*⟩ is correct upon return. This means that the ⟨*pde*⟩ will no longer evaluate properly until such time as the dependence is manually restored, whereupon the ⟨*EDS*⟩ will no longer be correct, and so on.

To assist with this difficulty, pde2eds saves a record of the dependencies it has removed in the shared variable dependencies. The operator mkdepend can be used to restore the initial state.

See also the operators pde2jet (section 9.5) and mkdepend (section 9.6).

Example:

```
depend u,x,y; depend v,x,y;
pde2eds({df(u,y,y)=df(v,x),df(v,y)=y*df(v,x)});

   EDS({d u - u *d x - u *d y,
               x        y

        d u  - u   *d x - u   *d y,
           x    x x         y x

        d u  - u   *d x - v *d y,
           y    y x         x

        d v - v *d x - v *y*d y},d x^d y)
               x        x

dependencies;

   {{u,y,x},{v,y,x}}
```

3.5 set_coframing

The background coframing (section 2.4) is set with `set_coframing`. The syntax is

`set_coframing` ⟨*arg*⟩

where ⟨*arg*⟩ is a ⟨*coframing*⟩ or an ⟨*EDS*⟩ and the previous background coframing is returned. All rules, orderings etc pertaining to the previous background coframing are removed and replaced by those for the new ⟨*coframing*⟩. The special form

`set_coframing()`

clears the background coframing entirely and returns the previous one.

4 Inspecting EDS objects

Given an ⟨*EDS*⟩ or some other EDS structure, it is often desirable to inspect or extract some part of it. The operators described in this section do just that. Many of them accept various types of arguments and return the relevant information in each case.

4.1 cobasis

`cobasis` ⟨*arg*⟩

returns the cobasis for ⟨*arg*⟩, which may be either a ⟨*coframing*⟩ or an ⟨*EDS*⟩ (sections 2.1, 2.2). The order of the items in the list gives the ⟨*kernel*⟩ ordering which applies when the ⟨*coframing*⟩ in ⟨*arg*⟩ is active.

4.2 coordinates

`coordinates` ⟨*arg*⟩

returns the coordinates for ⟨*arg*⟩, which may be either a ⟨*coframing*⟩, an ⟨*EDS*⟩, or a list of ⟨*expr*⟩ (sections 2.1, 2.2). The coordinates in a list of ⟨*expr*⟩ are defined to be those 0-form ⟨*kernels*⟩ with no implicit dependencies.

Examples:

```
coordinates contact(3,{x},{u});

   {x,u,u ,u   ,u     }
         x  x x  x x x

fdomain u=u(x);
coordinates {d u+d y};

   {x,y}
```

4.3 structure_equations

`structure_equations` ⟨*arg*⟩

returns the structure equations (cf section 2.1) for ⟨*arg*⟩, which may be either a ⟨*coframing*⟩, an ⟨*EDS*⟩, or a ⟨*transform*⟩ (sections 2.1, 2.2, 2.8). In the case of a ⟨*transform*⟩, it is assumed the exterior derivatives of the right-hand sides are known, and a list giving the exterior derivatives of the left-hand sides is returned. This requires inverting the transformation. In case this has already been done, and was time consuming, an alternative syntax

`structure_equations(`⟨*transform*⟩`,`⟨*inverse transform*⟩`)`

avoids recomputing the inverse.

Example:

```
structure_equations{e 1=d x/x,e 2=x*d y};

       1         2      1  2
   {d e  => 0,d e  => e ^e }
```

4.4 restrictions

`restrictions` ⟨*arg*⟩

returns the restrictions for ⟨*arg*⟩, which may be either a ⟨*coframing*⟩ or an ⟨*EDS*⟩ (sections 2.1, 2.2). The result is a list of inequalities.

4.5 system

`system` ⟨*EDS*⟩

returns the system component of an ⟨*EDS*⟩ (sections 2.2, 2.3) as a list of ⟨*p-form*⟩ expressions. (The PSL-based REDUCE command `system` operates as before: the syntax

`system "`⟨*command*⟩`"`

executes an operating system (eg UNIX) command.)

4.6 independence

`independence` ⟨*EDS*⟩

returns the independence condition of an ⟨*EDS*⟩ (section 2.2) as a list of ⟨*1-form*⟩ expressions.

4.7 properties

`properties` ⟨*EDS*⟩

returns the currently known properties of an ⟨*EDS*⟩ (sections 2.2, 2.6) as a list of equations of the form ⟨*keyword*⟩ `=` ⟨*value*⟩.

Example:

```
properties closure contact(1,{x},{u});

   {closed=1,pfaffian=1,quasilinear=1}
```

4.8 one_forms

`one_forms` ⟨*arg*⟩

returns the 1-forms in ⟨*arg*⟩, which may be either an ⟨*EDS*⟩ or a list of ⟨*expr*⟩ (sections 2.2, 2.3).

Example:

```
one_forms {5,x*y - u,d u - x*d y,d u^d x- x*d y^d x};

   {d u - d y*x}
```

4.9 zero_forms, nought_forms

    zero_forms ⟨*arg*⟩

returns the 0-forms in ⟨*arg*⟩, which may be either an ⟨*EDS*⟩ or a list of ⟨*expr*⟩ (sections 2.2, 2.3). The alternative syntax `nought_forms` does the same thing.

Example:

```
zero_forms {5,x*y - u,d u - x*d y,d u^d x- x*d y^d x};

   {5, - u + x*y}
```

5 Manipulating EDS objects

The abililty to change coordinates or cobasis, or to modify the system or coframing can make the difference between an intractible problem and a solvable one. The facilities described in this section form the low-level core of EDS functions.

Most of the operators in this section can be applied to both ⟨*EDS*⟩ and ⟨*coframing*⟩ objects. Where it makes sense (eg `pullback`, `restrict` and `transform`), they can be applied to a ⟨*system*⟩, or list of differential forms as well.

5.1 augment

    augment(⟨*EDS*⟩,⟨*system*⟩)

appends the extra forms in the second argument to the system part of the first. If the forms in the ⟨*system*⟩ do not live on the coframing of the ⟨*EDS*⟩, an error results. The original ⟨*EDS*⟩ is unchanged.

Example:

```
% Non-Pfaffian system for a Monge-Ampere equation
S := contact(1,{x,y},{z})$
S := augment(S,{d z(-x)^d z(-y)});

   s := EDS({d z - z *d x - z *d y,
                   x        y

             d z ^d z },{d x,d y})
                x    y
```

5.2 cross

The infix operator `cross` gives the direct product of ⟨*coframing*⟩ objects. The syntax is

⟨*arg1*⟩ `cross` ⟨*arg2*⟩ [`cross` ⋯]

The first argument may be either a ⟨*coframing*⟩ (section 2.1) or an ⟨*EDS*⟩ (section 2.2). The remaining arguments may be either ⟨*coframing*⟩ objects or any valid argument to the `coframing` operator (section 3.1), in which case the corresponding ⟨*coframing*⟩ is automatically inferred. The arguments may not contain any common coordinates or cobasis elements.

If the first argument is an ⟨*EDS*⟩, the result is the ⟨*EDS*⟩ lifted to the direct product space. In this way, it is possible to execute a pullback under a projection.

Example:

```
coordinates(contact(1,{x,y},{u}) cross {v});

   {x,y,u,u ,u ,v}
           x  y
```

5.3 pullback

Pullbacks with respect to an EDS ⟨*map*⟩ (section 2.7) have the syntax `pullback(`⟨*arg*⟩`,`⟨*map*⟩`)`

where ⟨*arg*⟩ can be any one of ⟨*EDS*⟩, ⟨*coframing*⟩, ⟨*system*⟩ or ⟨*p-form*⟩ expression (sections 2.2, 2.1, 2.3). The result is of the same type as ⟨*arg*⟩.

For an ⟨*EDS*⟩ or ⟨*coframing*⟩ with anholonomic cobasis, `pullback` calculates the pullbacks of the cobasis elements and chooses a cobasis for the source coframing itself. For a ⟨*system*⟩, any zeroes in the result are dropped from the list.

Examples:

```
pullback(contact(1,{x,y},{u}),{u(-y) = u*u(-x)});

   EDS({d u - u *d x - u *u*d y},{d x,d y})
               x        x

M := coframing({e 1,e 2},{r},{r neq 0},
               {d r=>e 1,d e 1=>0,d e 2=>e 1^e 2/r})$

pullback(M,{r=1/x});
                                     2
               2              2     e ^d x
   coframing({e ,d x},{x},{d e  => --------},{x neq 0})
                                      x

pullback(ws,{x=0});

   ***** Map image not within target coframing in pullback

pullback({y*d y,d y - d x},{y=x});

   {d x*x}
```

5.4 restrict

Restrictions with respect to an EDS ⟨*map*⟩ (section 2.7) have the syntax

`restrict(`⟨*arg*⟩`,`⟨*map*⟩`)`

where ⟨*arg*⟩ can be any one of ⟨*EDS*⟩, ⟨*coframing*⟩, ⟨*system*⟩ or ⟨*p-form*⟩ expression (sections 2.2, 2.1, 2.3). The result is of the same type as ⟨*arg*⟩. The action of `restrict` is similar to that of `pullback`, except that only scalar coefficients are affected: 1-form variables are unchanged.

Examples:

```
% Bring a system into normal form by restricting the coframing

S := eds({u*d v - v*d u},{d x});

   s := EDS({v*d u - u*d v},{d x})

restrict(S,{u neq 0});

               v
   EDS({d v - ---*d u},{d x})
               u

% Difference between restrict and pullback

pullback({x*d x - y*d y},{x=y,y=1});

   {}

restrict({x*d x - y*d y},{x=y,y=1});

   {d x - d y}
```

5.5 transform

A change of cobasis is made using the `transform` operator

`transform(`⟨*arg*⟩`,`⟨*transform*⟩`)`

where ⟨*arg*⟩ can be any one of ⟨*EDS*⟩, ⟨*coframing*⟩, ⟨*system*⟩ or ⟨*p-form*⟩ expression (sections 2.2, 2.1, 2.3) and ⟨*transform*⟩ is a list of substitutions (cf section 2.8). The result is of the same type as ⟨*arg*⟩.

For an ⟨*EDS*⟩ or ⟨*coframing*⟩, `transform` can detect whether the tranformation is given in the forward or reverse direction and invert accordingly. Structure equations are updated correctly. If an exact cobasis element is eliminated, its expression in terms of the new cobasis is added to the list of structure equations, since the corresponding coordinate may still be present elsewhere in the structure.

Example:

```
S := contact(1,{x},{u});

   s := EDS({d u - u *d x},{d x})
                    x

new := {e(1) = first system S,w(1) = d x};

            1               1
   new := {e =d u - d x*u  ,w =d x}
                          x

S := transform(S,new);

              1    1
   s := EDS({e },{w })

structure_equations s;

       1             1
   {d e  =>  - d u ^w  ,
                  x
       1
    d w  => 0,

            1       1
    d u => e  + u *w  ,
                 x
            1
    d x => w }
```

5.6 lift

Many of the analysis tools (section 6) cannot treat systems containing 0-forms. The `lift` operator

`lift` ⟨*EDS*⟩

solves the 0-forms in the system and uses the solution to pull back to a smaller manifold. This may generate new 0-form conditions (in the course of bringing the pulled-back system into normal form), in which case the process is repeated until the system is generated in positive degree. In non-linear problems, the solution space of the 0-forms may be a variety, in which case a compound ⟨*EDS*⟩ (section 2.2) will result. If `edsverbose` is on (section 8.1), the solutions are displayed as they are generated.

Example:

```
S := augment(contact(2,{x,y},{u}),{u(-y,-y)-u(-x,-x)})$
on edsverbose;
lift S;
   Solving 0-forms
   New equations:
   u   =u
    y y  x x

   EDS({d u - u *d x - u *d y,
               x        y

         d u  - u   *d x - u   *d y,
            x    x x         x y

         d u  - u   *d x - u   *d y},{d x,d y})
            y    x y         x x
```

6 Analysing exterior systems

This section describes higher level operators for extracting information about exterior systems. Many of them require a ⟨*EDS*⟩ in normal form (section 2.10) generated in positive degree as input, but some can also analyse a

⟨*system*⟩ (section 2.3) or a single ⟨*p-form*⟩. Only trivial examples are provided in this section, but many of these operators are used in the longer examples in the test file which accompanies this package.

6.1 cartan_system

The *Cartan system* of a form or system S is the smallest Pfaffian system C such that $\Lambda(C)$ contains a set I of forms algebraically equivalent to S. The Cartan system is also known as the *associated Pfaff system* or *retracting space*. An alternative characterisation is to note that the annihilator $C^\perp$ comprises all vectors V satisfying $i_V S \simeq 0 \pmod{S}$. Note this is a purely algebraic concept: S need not be closed under differentiation. See also `cauchy_system` (section 6.2).

The operator

`cartan_system` ⟨*arg*⟩

returns the Cartan system of ⟨*arg*⟩, which may be an ⟨*EDS*⟩, a ⟨*system*⟩ or a single ⟨*p-form*⟩ expression (sections 2.2, 2.3). For an ⟨*EDS*⟩, the result is a Pfaffian ⟨*EDS*⟩ on the same manifold, otherwise it is a ⟨*system*⟩. The argument must be generated in positive degree.

Example:

```
cartan_system{d u^d v + d v^d w + d x^d y};

   {d u - d w,d v,d x,d y}
```

6.2 cauchy_system

The *Cauchy system* C of a form or system S is the Cartan system or retracting space of its closure under exterior differentiation (section 6.1). The annihilator $C^\perp$ consists of the Cauchy vectors for the S.

The operator

`cauchy_system` ⟨*arg*⟩

returns the Cauchy system of ⟨*arg*⟩, which may be an ⟨*EDS*⟩, a ⟨*system*⟩ or a single ⟨*p-form*⟩ expression (sections 2.2, 2.3). For an ⟨*EDS*⟩, the result is a Pfaffian ⟨*EDS*⟩ on the same manifold, otherwise it is a ⟨*system*⟩. The argument must be generated in positive degree.

Example:

```
cauchy_system{u*d v + v*d w + x*d y};

   {d u,d v,d w,d x,d y}
```

6.3 characters

The Cartan characters $\{s_1, ..., s_n\}$ of an ⟨*EDS*⟩ or ⟨*tableau*⟩ (sections 2.2, 2.9) are obtained with

`characters` ⟨*EDS*⟩ *or* `characters` ⟨*tableau*⟩

The zeroth character s_0 is not returned, it is simply the number of 1-forms in the ⟨*EDS*⟩ (cf `one_forms`, section 4.8). The definition used for the last character: $s_n = (d - n) - (s_0 + s_1 + ... + s_{n-1})$, where d is the manifold dimension, allows Cartan's test to be used even when Cauchy characteristics are present.

For a nonlinear ⟨*EDS*⟩, the Cartan characters can vary from stratum to stratum of the Grassmann bundle variety of ordinary integral elements (cf `grassmann_variety` in section 6.14). Nonetheless, they are constant on each stratum, so it suffices to calculate them at one point (ie at one integral element). This is done using the syntax

`characters(`⟨*EDS*⟩`,`⟨*integral element*⟩`)`

where ⟨*integral element*⟩ is a list of 1-forms (cf section 2.5).

The Cartan characters are calculated from the reduced characters for a fixed flag of integral elements based on the 1-forms in the independence condition of an ⟨*EDS*⟩. This can lead to incorrect results if the flag is somehow singular, so two switches are provided to overcome this (section 8.5). First, `genpos` looks at a generic flag by using a general linear transformation to put the system in *general position*. This guarantees correct results, but can be too slow for practical purposes. Secondly, `ranpos` performs a linear transformation using a sparse matrix of random integers. In most cases, this is much faster than using general position, and a few repetitions give some confidence in the results.

Example:

```
S := pullback(contact(2,{x,y},{u}),{u(-x,-y)=0});

   s := EDS({d u - u *d x - u *d y,
                    x        y

             d u  - u   *d x,
                x    x x

             d u  - u   *d y},{d x,d y})
                y    y y

characters S;

   {1,1}

on ranpos; characters S;

   {2,0}
```

6.4 closure

`closure` ⟨*EDS*⟩

returns the closure of the ⟨*EDS*⟩ under exterior differentiation.

Owing to conflicts with the requirements of a normal form (section 2.10), `closure` cannot guarantee that the resulting system is closed if the ⟨*EDS*⟩ contains 0-forms.

6.5 derived_system

`derived_system` ⟨*arg*⟩

returns the first derived system of ⟨*arg*⟩, which must be a Pfaffian ⟨*EDS*⟩ or ⟨*system*⟩. Repeated use gives the derived flag leading to the maximal integrable subsystem.

Example:

```
pform {p,r,x,y,z}=0; korder z;
derived_system eds({d z - q*d y,d p - e**z*d y,
                    d r - e**z*p*d y,d x},{d y});

              z            z
   EDS({d p - e *d y,d r - e *p*d y,d x},{d y})

derived_system ws;

              1
   EDS({d p - ---*d r,d x},{d y})
              p

derived_system ws;

              1
   EDS({d p - ---*d r,d x},{d y})
              p
```

6.6 dim_grassmann_variety

`dim_grassmann_variety` ⟨*EDS*⟩

returns the dimension of the Grassmann bundle variety of ordinary integral elements for an ⟨*EDS*⟩ (cf `grassmann_variety`, section 6.14). This number is useful, for example, in Cartan's test. For a nonlinear ⟨*EDS*⟩, this can vary from stratum to stratum of the variety, so

`dim_grassmann_variety(`⟨*EDS*⟩`,`⟨*integral element*⟩`)`

returns the dimension of the stratum containing the ⟨*integral element*⟩ (cf section 2.5).

6.7 dim

`dim` ⟨*arg*⟩

returns the dimension of the manifold underlying ⟨*arg*⟩, which can be either an ⟨*EDS*⟩ or a ⟨*coframing*⟩ (sections 2.2, 2.1).

6.8 involution

`involution` ⟨*EDS*⟩

repeatedly prolongs an ⟨*EDS*⟩ until it reaches involution or inconsistency (cf `prolong`, section 6.11). The system must be in normal form (section 2.10) and generated in positive degree. For nonlinear problems, all branches of the prolongation tree are followed. The result is an ⟨*EDS*⟩ (usually a compound one for nonlinear problems, see section 2.2) giving the involutive prolongation. In case some variety couldn't be resolved during the process, the relevant branch is truncated at that point and represented by a system with 0-forms, as with `grassmann_variety` (section 6.14). The result of `involution` might then *not* be in involution.

If the `edsverbose` switch is on (section 8.1), a trace of the prolongations is produced. See the Janet problem in the test file for an example.

6.9 linearise, linearize

A nonlinear exterior system can be linearised at some point on the manifold with respect to any integral element, yielding a constant coefficient exterior system with the same Cartan characters. In EDS, reference to the point is omitted, so the result is an exterior system linearised with respect to a distribution of integral elements. The syntax is

`linearise(`⟨*EDS*⟩`,`⟨*integral element*⟩`)`

but `linearize` will work just as well. See the isometric embeddings example in the test file.

For a quasilinear ⟨*EDS*⟩ (cf section 7.4),

`linearise` ⟨*EDS*⟩

returns an equivalent exterior system containing only linear generators.

Example:

```
f := d u^d x + d v^d y$
S := eds({f,d v^f},{d x,d y});

   s := EDS({d u^d x + d v^d y,d u^d v^d x},{d x,d y})

linearise S;

   EDS({d u^d x + d v^d y},{d x,d y})
```

6.10 integral_element

`integral_element` ⟨*EDS*⟩

returns a random ⟨*integral element*⟩ of the ⟨*EDS*⟩ (section 2.5). The system must be in normal form (section 2.10) and generated in positive degree. This integral element is found using the method described by Wahlquist [11] (essentially the Cartan-Kähler construction filling in the free variables from each polar system with random integer values). This method can fail on non-involutive systems, or ⟨*EDS*⟩ objects whose independence conditions indicate a singular flag of integral elements (cf the discussion about Cartan characters, section 6.3).

See the isometric embedding problem in the test file for an example.

6.11 prolong

`prolong` ⟨*EDS*⟩

calculates the prolongation of the ⟨*EDS*⟩ to the Grassmann bundle variety of integral elements. The system must be in normal form (section 2.10) and generated in positive degree. The variety is decomposed using essentially the REDUCE `solve` operator. If no solutions can be found, the variety is empty, and the prolongation is the inconsistent ⟨*EDS*⟩ (section 2.2). Otherwise, the result is a list of variety components, which fall into three classes:

1. a submanifold of the Grassmann bundle which surjects onto the base manifold. The result in this case is the pullback of the Grassmann bundle contact ⟨*EDS*⟩ to this submanifold.

2. a submanifold of the Grassmann bundle which does not surject onto the base manifold (ie cannot be presented by solving for Grassmann bundle fibre coordinates). The result in this case is the pullback of the original ⟨*EDS*⟩ to the projection onto the base manifold. If 0-forms arise in bringing the pullback to normal form, these are solved recursively and the system pulled back again until the result is generated in positive degree (cf `lift`, section 5.6).
3. a component of the variety which `solve` was not able to resolve explicitly. The result in this case is the Grassmann bundle contact ⟨*EDS*⟩ augmented with the 0-forms which `solve` couldn't treat. This can be extracted from the result of `prolong` and manipulated further "by hand",

The result returned by `prolong` will, in general, be a compound ⟨*EDS*⟩ (section 2.2). If the switch `edsverbose` (section 8.1) is on, a trace of the prolongation is printed.

The ⟨*map*⟩s which are generated in a `prolong` call are available subsequently in the global variable `pullback_maps`. This facility is still very primitive and unstructured. It should be extended to the `involution` operator as well...

Example:

```
pde := {u(-y,-y)=u(-x,-x)**2/2,u(-x,-y)=u(-x,-x)};


                        2
                   (u    )
                     x x
   pde := {u    =---------,u    =u    }
            y y      2      x y  x x

S := pullback(contact(2,{x,y},{u}),pde)$
on edsverbose;
prolong S;
   Reduction using new equations:
   u    =1
    x x
   Prolongation using new equations:
   u      =0
```

```
    x x x
   u     =0
    x x y

   {EDS({d u - u *d x - u *d y,
                x        y

         d u  - d x - d y,
            x

                    1
         d u  - d x - ---*d y},{d x,d y}),
            y         2

    EDS({d u - u *d x - u *d y,
                x        y

         d u  - u   *d x - u   *d y,
            x    x x        x x
                                 2
                           (u   )
                             x x
         d u  - u   *d x - ---------*d y,
            y    x x           2

         d u   },{d x,d y})}
            x x
```

6.12 tableau

tableau ⟨*EDS*⟩

returns the ⟨*tableau*⟩ (section 2.9) of a quasilinear Pfaffian ⟨*EDS*⟩, which must be in normal form and generated in positive degree.

Example:

```
tableau contact(2,{x,y},{u});

   [d u     d u   ]
   [   x x     x y]
   [              ]
   [d u     d u   ]
   [   x y     y y]
```

6.13 torsion

For a semilinear Pfaffian exterior differential system, the torsion corresponds to first-order integrability conditions for the system. Specifically,

`torsion` ⟨*EDS*⟩

returns a list of 0-forms describing the projection of the Grassmann bundle variety of integral elements onto the base manifold. If the switch `edssloppy` (section 8.3) is on, quasilinear systems are treated as semilinear. A semilinear system is involutive if both the torsion is empty, and Cartan's test for the reduced characters is satisfied.

Example:

```
S := pullback(contact(2,{x,y},{u}),
              {u(-y,-y)=u(-x),u(-x,-y)=u});

   s := EDS({d u - u *d x - u *d y,
                    x        y

             d u  - u   *d x - u*d y,
                x    x x

             d u  - u*d x - u *d y},{d x,d y})
                y            x
torsion s;

   {u    - u }
     x x    y
```

6.14 grassmann_variety

Given an exterior system (S, Ω, M) with independence condition of rank n, the Grassmann bundle of n-planes over M contains a submanifold characterised by those n-planes compatible with the independence condition. All integral elements must lie in this submanifold. The operator

`grassmann_variety` $\langle EDS \rangle$

returns the contact system for this part of the Grassmann bundle augmented by the 0-forms specifying the variety of integral elements of S. In cases where `prolong` (section 6.11) is unable to decompose the variety automatically, `grassmann_variety` can be used in combination with `zero_forms` (section 4.9) to calculate the variety conditions. Any solutions found "by hand" can be incorporated using `pullback` (section 5.3).

Example: Using the system from the example in section 6.11:

```
zero_forms grassmann_variety S;

   { - u     *u    + u     ,
        x x x  x x    x x y

     - u      + u     }
        x x x    x x y

solve ws;

   Unknowns: {u     ,u     ,u   }
               x x x  x x y  x x

   {{u     =0,u     =0},
      x x y    x x x

    {u   =1,u     =u     }}
      x x    x x x  x x y
```

The second solution contains an integrability condition.

7 Testing exterior systems

The operators in this section allow various properties of an ⟨*EDS*⟩ to be checked. These checks are done automatically when required on entry to the routines in sections 5 and 6, but sometimes it is useful to know explicitly. The result is either a `1` (true) or a `0` (false), so the operators can be used in boolean expressions within `if` statements etc. Since checking these properties can be very time-consuming, the result of the first test is stored on the ⟨*properties*⟩ record of an ⟨*EDS*⟩ to avoid re-checking. This memory can be cleared using the `cleanup` operator.

7.1 closed

`closed` ⟨*arg*⟩

checks whether ⟨*arg*⟩, which may be an ⟨*EDS*⟩, a ⟨*system*⟩ or a single ⟨*p-form*⟩ is closed under exterior differentiation.

Examples:

```
closed(x*d x);

   1

closed {d u - p*d x,d p - p/y*d x};

   0
```

7.2 involutive

`involutive` ⟨*EDS*⟩

checks whether ⟨*EDS*⟩ is involutive, using Cartan's test. See the test file for examples.

7.3 pfaffian

`pfaffian` ⟨*EDS*⟩

checks whether $\langle EDS \rangle$ is a Pfaffian system: generated by a set of 1-forms and their exterior derivatives. The $\langle EDS \rangle$ must be in normal form (section 2.10) for this to succeed. Systems with 0-forms are non-Pfaffian by definition in EDS.

Examples:

```
pfaffian eds({d u - p*d x - q*d y,d p^d x+d q^d y},{d x,d y});

   1

pfaffian eds({d u - p*d x - q*d y,d p^d q},{d x,d y});

   0
```

7.4 quasilinear

An exterior system (S, Ω, M) is said to be *quasilinear* if, when written in the standard cobasis $\{\theta^a, \pi^\rho, \omega^i\}$ (section 2.11), its *closure* can be generated by a set of forms which are of combined total degree 1 in $\{\theta^a, \pi^\rho\}$. The operation

`quasilinear` $\langle EDS \rangle$

checks whether the *closure* of $\langle EDS \rangle$ is a quasilinear system. The $\langle EDS \rangle$ must be in normal form (section 2.10) for this to succeed. Systems with 0-forms are not quasilinear by definition in EDS.

Examples:

```
% A system where pi(rho)={d p,d q}, and which looks non-linear

S := eds({d u - p*d x - q*d y,d p^d q^d y},{d x,d y})$

quasilinear S;

   1

linearise closure S;
```

```
   EDS({d u - p*d x - q*d y,
         - d p^d x - d q^d y},{d x,d y})

% One which is really non-linear

quasilinear eds({d u - p*d x - q*d y,d p^d q},{d x,d y});

   0
```

7.5 semilinear

Let (S, Ω, M) be such that, written in the standard cobasis $\{\theta^a, \pi^\rho, \omega^i\}$ (section 2.11), its closure is explicitly quasilinear. If the coefficients of $\{\pi^\rho\}$ depend only on the independent variables, then the system is said to be *semilinear*. The operation

`semilinear` $\langle EDS \rangle$

checks whether *closure* of $\langle EDS \rangle$ is a semilinear system. The $\langle EDS \rangle$ must be in normal form (section 2.10) for this to succeed. Systems with 0-forms are not semilinear by definition in EDS.

For semilinear systems, the expressions determining the Grassmann bundle variety of integral elements will be linear in the Grassmann bundle fibre coordinates, with coefficients which depend only upon the independent variables. This allows alternative, faster algorithms to be used in analysis.

If the switch `edssloppy` is on (section 8.3), all quasilinear systems are treated as if they are semilinear.

Examples:

```
% A semilinear system: @(u,y) = y*@(u,x)
S := eds({d u - p*d x - p*y*d y},{d x,d y})$
semilinear S;

   1
% A quasilinear system: @(u,y) = u*@(u,x)
S := eds({d u - p*d x - p*u*d y},{d x,d y})$
```

```
quasilinear S;

   1
semilinear S;

   0
on edssloppy;
semilinear S;

   1
```

7.6 frobenius

`frobenius` ⟨*arg*⟩

checks whether ⟨*arg*⟩, which may be an ⟨*EDS*⟩ or a ⟨*system*⟩, is a completely integrable Pfaffian system.

Examples:

```
if frobenius eds({d u -p*(d x+d y)},d x^d y) then yes else no;

   no

if frobenius eds({d u -u*(d x+d y)},d x^d y) then yes else no;

   yes
```

7.7 equiv

⟨*EDS1*⟩ `equiv` ⟨*EDS2*⟩

checks whether ⟨*EDS1*⟩ and ⟨*EDS2*⟩ are algebraically equivalent as exterior systems (ie generate the same algebraic ideal).

Examples:

```
S1 := contact(2,{x,y},{u})$
S2 := augment(S1,foreach f in system S1 join {d f,d x^d f})$
```

```
if S1 equiv S2 then yes else no;

   no

if closure S1 equiv S2 then yes else no;

   yes
```

8 Switches

EDS provides several switches to govern the display of information and speed or reliability of the results.

8.1 edsverbose

If `edsverbose` is `on`, a number of operators (eg `prolong`, `involution`) will display additional information as the calculation progresses. For large problems, this can produce too much output to be useful, so `edsverbose` is `off` by default. This allows only warning (`***`) and error (`*****`) messages to be printed.

8.2 edsdebug

If `edsdebug` is `on`, EDS produces copious quantities of information, in addition to that produced with `edsverbose` on. This information is for debugging purposes, and may not make much sense without knowledge of the inner workings of EDS. `edsdebug` is `off` by default.

8.3 edssloppy

Normally, EDS will not divide by any expressions it does not know to be nowhere zero. If an ⟨*EDS*⟩ can be brought into normal form only by restricting away from the zeroes of some coefficients, then these restrictions should be made using the `restrict` operator (section 5.4). However, if `edssloppy` is `on`, then EDS will, as a last resort, divide by whatever is necessary to bring an ⟨*EDS*⟩ into normal form, invert a transformation, and so on. The relevant restrictions will be made automatically, so no inconsistency should arise. In addition, with `edssloppy on`, all quasilinear systems are treated as if they were semilinear (cf section 7.5). `edssloppy` is `off` by default.

8.4 edsdisjoint

When decomposing a variety into (something like) smooth components, EDS normally pays no attention to whether the components are disjoint. Turning `on` the switch `edsdisjoint` forces EDS to ensure the decomposition is a disjoint union (cf `disjoin`, section 9.7). For large problems this can lead to a proliferation of singular pieces. If some of these turn out to be uninteresting, EDS cannot re-join the remaining pieces into a smaller decomposition. `edsdisjoint` is `off` by default.

8.5 ranpos, genpos

When calculating Cartan characters (eg to check involution), EDS uses the independence condition of an $\langle EDS \rangle$ *as presented* to define a flag of integral elements. Depending on the cobasis and ordering, this flag may be singular, leading to incorrect Cartan characters. To overcome this problem, the switches `ranpos` and `genpos` provide a means to select other flags. With `ranpos on`, a flag defined by taking a random linear transformation of the 1-forms in the independence condition will be used. The results may still be incorrect, but the likelihood is much lower. With `genpos` on, a generic (upper triangular) transformation is used. this guarantees the correct Cartan characters, but reduces performance too much to be useful for large problems. Both switches are `off` by default, and switching one `on` automatically switches the other `off`. See section 6.3 for an example.

9 Auxiliary functions

This section describes various operators designed to ease working with exterior forms and exterior systems in REDUCE.

9.1 invert

`invert` ⟨*transform*⟩

returns a ⟨*transform*⟩ which is inverse to the given one (section 5.5). If the ⟨*transform*⟩ given is only partial, the 1-form ⟨*kernel*⟩s to eliminate are chosen based on the prevailing kernel ordering. If a background coframing (section 2.4) is active, and `edssloppy` (section 8.3) is `off`, `invert` will divide by nowhere-zero expressions only.

Examples:

```
set_coframing coframing{u,v,w,x,y,z}$
invert {d u = 3*d x - d y + 5*d z, d v = d x + 2*d z};

   {d x=d v - 2*d z,d y= - d u + 3*d v - d z}

% A y coefficient forces a different choice of inverse

invert {d u = 3*d x - y*d y + 5*d z, d v = d x + 2*d z};

   {d x=2*d u - 5*d v + 2*d y*y,d z= - d u + 3*d v - d y*y}
```

9.2 linear_divisors

`linear_divisors` ⟨*pform*⟩

returns a basis for the space of linear divisors (1-form factors) of a ⟨*p-form*⟩.

Example:

```
f := d p^d q^d u - d p^d q^d x*x + d p^d q^d z*y
     - d u^d v^d x*x + d u^d v^d z*y + d u^d x^d y
     + d x^d y^d z*y$
linear_divisors f;

   {d u - d x*x + d z*y}
```

9.3 exfactors

`exfactors` ⟨*pform*⟩

returns a list of factors for a ⟨*p-form*⟩, consisting of the linear divisors plus one more factor. The list is ordered such that the original expression is a product of the factors in this order.

Example:

```
f := d p^d q^d u - d p^d q^d x*x + d p^d q^d z*y
     - d u^d v^d x*x + d u^d v^d z*y + d u^d x^d y
     + d x^d y^d z*y$
exfactors f;

   {d p^d q - d v^d x*x + d v^d z*y + d x^d y,
    d u - d x*x + d z*y}

f - (part(ws,0) := ^);

   0
```

9.4 index_expand

EXCALC caters for indexed variables in which various index names have been assigned a specific set of values. Any expression with *paired* indices is expanded automatically to an explicit sum over the index set (provided the EXCALC command `nosum` has not been applied). The EDS operator `index_expand` is designed to expand an expression with *free* indices to an explicit list over the index set, taking some limited account of the possible index symmetries.

The syntax is

`index_expand` ⟨*arg*⟩

where ⟨*arg*⟩ can be an expression, a rule or equation or a boolean expression, or an arbitrarily nested list of these items. The result is a flattened list.

Examples:

```
indexrange {i,j,k}={1,2,3},{a,b}={x,y};
pform {e(i),o(a,b)}=1;
index_expand(e(i)^e(j));

     1   2   1   3   2   3
   {e ^e ,e ^e ,e ^e }

index_expand{o(-a,-b)+o(-b,-a) => 0};

   {2*o    => 0,o    + o    => 0, 2*o    => 0}
       x x       x y    y x          y y
```

9.5 pde2jet

A PDE system can be encoded into EDS jet variable notation using `pde2jet`. The syntax is as for `pde2eds`:

`pde2jet(`⟨*pde*⟩ [`,`⟨*dependent*⟩`,`⟨*independent*⟩] `)`

where ⟨*pde*⟩ is a list of equations or expressions (implicitly assumed to vanish) specifying the PDE system using either the standard REDUCE `df` operator, or the EXCALC `@` operator. If the optional variable lists ⟨*dependent*⟩ and ⟨*independent*⟩ are not given, `pde2jet` infers them from the expressions in ⟨*pde*⟩, using the same rules as `pde2eds` (section 3.4).

The result of `pde2jet` is the input ⟨*pde*⟩, with all derivatives of dependent variables replaced by indexed 0-form variables from the appropriate jet bundle. Unlike `pde2eds`, `pde2jet` does not disturb the variable dependencies.

Example:

```
depend u,x,y; depend v,x,y;
pde2jet({df(u,y,y)=df(v,x),df(v,y)=y*df(v,x)});

   {u    =v  ,
     y y   x

    v =v  *y}
     y   x
```

9.6 mkdepend

The `mkdepend` operator is intended for restoring the dependencies destroyed by a call to `pde2eds` (section 3.4). The syntax is

`mkdepend` {⟨*list of variables*⟩,···}

where the first variable in each list is declared to depend on the remaining ones.

9.7 disjoin

The `disjoin` operator takes a list of ⟨*maps*⟩ (section 2.7) describing a decomposition of a variety, and returns an equivalent list of ⟨*maps*⟩ such that the components are all disjoint. The background coframing (section 2.4) should be set appropriately before calling `disjoin`. The syntax is

`disjoin` {⟨*map*⟩,···}

Example:

```
set_coframing coframing {x,y};
disjoin {{x=0},{y=0}};

   {{y=0,x neq 0},{x=0,y neq 0},{y=0,x=0}}
```

9.8 cleanup

To avoid lengthy recomputations, EDS stores various properties (section 2.6) and also many intermediate results in a hidden list attached to each ⟨*EDS*⟩. When EDS detects a change in circumstances which could make the information innacurate, it is discarded and recomputed. Unfortunately, this mechanism is not perfect, and occasionally misses something which renders the results incorrect. In such a case, it is possible to discard all the properties and hidden information using the `cleanup` operator. The call

`cleanup` ⟨*arg*⟩

returns a copy of ⟨*arg*⟩, which may be a ⟨*coframing*⟩ or an ⟨*EDS*⟩ which has been stripped of this auxilliary information. Note that the original input (with possible innacuracies) is left undisturbed by this operation: the result of `cleanup` must be used.

Example:

```
% An erroneous property assertion
S := eds({d u - p*d x},{d x,d y},{closed = 1})$
closure S;

   EDS({d u - p*d x},{d x,d y});

S := cleanup S$
properties S;

   {}

closure S;

   EDS({d u - p*d x, - d p^d x},{d x,d y});
```

9.9 reorder

All operations with a ⟨*coframing*⟩ or ⟨*EDS*⟩ temporarily override the prevailing kernel order with their own. Thus the ordering of the cobasis elements in a ⟨*coframing*⟩ operator remains fixed, even when a REDUCE `korder` statement is issued. To enforce conformity to the prevailing kernel order, the `reorder` operator is available. The call

`reorder` ⟨*arg*⟩

returns a copy of ⟨*arg*⟩, which may be a ⟨*coframing*⟩ or an ⟨*EDS*⟩ which has been reordered. Note that the original input is left undisturbed by this operation: the result of `reorder` must be used.

Example:

```
M := coframing {x,y,z};

   m := coframing({d x,d y,d z},{x,y,z},{},{})
```

```
korder z,y,x;
reorder m;

   coframing({d z,d y,d x},{z,y,x},{},{})
```

10 Experimental facilities

This section describes various operators in EDS which either not algorithmically well-founded, or whose implementation is very unstable, or which have known bugs.

10.1 poincare

The `poincare` operator implements the homotopy integral found in the proof of Poincaré's lemma. The expansion point is the origin of the coordinates found in the input. The syntax is

`poincare` ⟨*p-form*⟩

If `f` is a p-form, then `poincare f` is a $(p-1)$-form, and `f - poincare d f` is an exact p-form.

Examples:

```
poincare(3*d x^d y^d z);

   d x^d y*z - d x^d z*y + d y^d z*x

d ws;

   3*d x^d y^d z

2*x*d y - poincare d(2*x*d y);

   d x*y + d y*x
```

10.2 invariants

The `invariants` operator implements the algorithm implicit in the inductive proof of the Frobenius theorem. The syntax is

`invariants(`⟨*system*⟩ [`,`⟨*list of coordinate*⟩] `)`

where ⟨*system*⟩ is a set of 1-forms satisfying the Frobenius condition. The optional second argument specifies the order in which the coordinates are projected away to get a trivially integrable system. The CRACK and ODESOLVE packages are used to solve the ODE systems which arise, so the limitations of these packages constrain the scope of this operator as well.

Examples:

```
invariants {d x*y + d y*x*z + d z*log(y)*x*y};


        z
   { - y *x}

invariants {d y*z**2 - d y*z + d z*y,d x*(1 - z) + d z*x};

       x     y*(z - 1)
   {-------,-----------}
     z - 1       z
```

10.3 symbol_relations

The `symbol_relations` operator finds the linear relations between the entries of the tableau matrix for a quasilinear system. The syntax is

`symbol_relations(`⟨*EDS*⟩`,`⟨*identifier*⟩`)`

where ⟨*EDS*⟩ is a quasilinear Pfaffian system and ⟨*identifier*⟩ is used to create a 2-indexed 1-form which will label the tableau entries.

Example:

```
S := pde2eds {df(u,y,y) = df(u,x,x)};
```

```
   s := EDS({d u - u *d x - u *d y,
                   x        y

             d u  - u   *d x - u   *d y,
                x    x x        x y

             d u  - u   *d x - u   *d y},d x^d y)
                y    x y        x x
```

```
symbol_relations(S,pi);

      1       2
   {pi    - pi   ,
        x       y
      1       2
    pi    - pi   }
        y       x
```

10.4 symbol_matrix

The `symbol_matrix` operator returns the symbol matrix for a quasilinear system in terms of a given variable. The syntax is

`symbol_matrix(`⟨*EDS*⟩`,`⟨*identifier*⟩`)`

where ⟨*EDS*⟩ is a quasilinear Pfaffian system and ⟨*identifier*⟩ is used to create an indexed 0-form which will parameterise the matrix.

Example:

```
% With the same system as for symbol_relations:

symbol_matrix(S,xi);

   [xi    - xi ]
   [  x       y]
   [           ]
   [xi    - xi ]
   [  y       x]
```

10.5 characteristic_variety

The `characteristic_variety` operator returns the equations specifying the characteristic variety for a quasilinear system in terms of a given variable. The syntax is

`characteristic_variety(`⟨*EDS*⟩`,`⟨*identifier*⟩`)`

where ⟨*EDS*⟩ is a quasilinear Pfaffian system and ⟨*identifier*⟩ is used to create an indexed 0-form variable. The result is a list of two lists: the first being the variety equations and the second the variables involved.

Example:

```
% With the same system as for symbol_relations:

characteristic_variety(S,xi);

         2        2
   {{(xi )  - (xi ) },
        x        y
    {xi ,xi }}
       x   y
```

A Command tables

The tables in this appendix summarise the commands available in EDS. More detailed descriptions of the syntax and function of each command are to be found in the earlier sections. In each case, examples of the command are given, whereby the argument variables have the following types (see section 2):

E, E'	$\langle$*EDS*$\rangle$
S	$\langle$*system*$\rangle$
M, N	$\langle$*coframing*$\rangle$, or a $\langle$*system*$\rangle$ specifying a $\langle$*coframing*$\rangle$
r	$\langle$*integer*$\rangle$
Ω	$\langle$*p-form*$\rangle$
f	$\langle$*map*$\rangle$
rsx	$\langle$*list of inequalities*$\rangle$
cob	$\langle$*list of 1-form variables*$\rangle$
crd, *dep*, *ind*	$\langle$*list of 0-form variables*$\rangle$
drv	$\langle$*list of rules for exterior derivatives*$\rangle$
pde	$\langle$*list of expressions or equations*$\rangle$
X	$\langle$*transform*$\rangle$
T	$\langle$*tableau*$\rangle$
P	$\langle$*integral element*$\rangle$

Command	**Function**
`coframing(`*cob*`,`*crd*`,`*rsx*`,`*drv*`)`	constructs a $\langle$*coframing*$\rangle$ with the given cobasis *cob*, coordinates *crd*, restrictions *rsx* and structure equations *drv*: *crd*, *rsx* and *drv* are optional
`coframing(`S`)`	constructs a $\langle$*coframing*$\rangle$ capable of supporting the given $\langle$*system*$\rangle$
`eds(`S`,`Ω`,`M`)`	constructs a simple $\langle$*EDS*$\rangle$ object with given system and independence condition: if M is not supplied, it is deduced from the rest
`contact(`r`,`M`,`N`)`	constructs the $\langle$*EDS*$\rangle$ for the contact system of the jet bundle $J^r(M,N)$
`pde2eds(`*pde*`,`*dep*`,`*ind*`)`	converts a PDE system to an EDS: dependent and independent variables are deduced if they are not specified (variable dependencies are removed)
`set_coframing(`M`)` `set_coframing(`E`)`	sets background coframing and returns previous one
`set_coframing()`	clears background coframing and returns previous one

Table 1: Commands for constructing EDS objects

Command	**Function**
`coframing(`E`)`	extracts the underlying ⟨*coframing*⟩
`coframing()`	returns the current background coframing

`cobasis(`M`)`
`cobasis(`E`)`
	extracts the underlying cobasis

`coordinates(`M`)`
`coordinates(`E`)`
	extracts the coordinates

`structure_equations(`M`)`
`structure_equations(`E`)`
	extracts the rules for exterior derivatives for cobasis and coordinates

`restrictions(`M`)`
`restrictions(`E`)`
	extracts the inequalities describing the restrictions in the ⟨*coframing*⟩

`system(`E`)`	extracts the ⟨*system*⟩ part of E
`independence(`E`)`	extracts the independence condition from E as a Pfaffian ⟨*system*⟩
`properties(`E`)`	returns the currently known properties of the ⟨*EDS*⟩ E as a list of equations ⟨*keyword*⟩=⟨*value*⟩

`one_forms(`E`)`
`one_forms(`S`)`
	selects the 1-forms from a system

`zero_forms(`E`)`
`zero_forms(`S`)`
	selects the 0-forms from a system

Table 2: Commands for inspecting EDS objects

Command	**Function**
`augment(`E`,`S`)`	appends the extra forms in S to the system in E
M `cross` N E `cross` N	forms the direct product of two coframings: an $\langle$*EDS*$\rangle$ E is lifted to the extended space

`pullback(`E`,`f`)`
`pullback(`S`,`f`)`
`pullback(`Ω`,`f`)`
	pulls back the first argument using the $\langle$*map*$\rangle$ f

`pullback(`M`,`f`)`	returns a $\langle$*coframing*$\rangle$ N suitable as the source for $f : N \to M$

`restrict(`E`,`f`)`
`restrict(`S`,`f`)`
`restrict(`Ω`,`f`)`
	restricts the first argument to the points specified by the $\langle$*map*$\rangle$ f

`restrict(`M`,`f`)`	adds the restrictions in f to M

`transform(`M`,`X`)`
`transform(`E`,`X`)`
`transform(`S`,`X`)`
`transform(`Ω`,`X`)`
	applies the change of cobasis X to the first argument: for a $\langle$*coframing*$\rangle$ M or an $\langle$*EDS*$\rangle$ E, X may be specified in either the forward or reverse direction

`lift(`E`)`	eliminates any 0-forms in E by solving and pulling back

Table 3: Commands for manipulating EDS objects

Command	**Function**

`cartan_system(`E`)`
`cartan_system(`S`)`
`cartan_system(`Ω`)`
	calculates the Cartan system (associated Pfaff system, retracting space): no differentiations are performed

`cauchy_system(`E`)`
`cauchy_system(`S`)`
`cauchy_system(`Ω`)`
	calculates the Cauchy system: the Cartan system of the closure under exterior differentiation

`characters(`E`)`
`characters(`T`)`
	calculates the (reduced) Cartan characters $\{s_1, ..., s_n\}$ (E quasilinear)

`characters(`E`,`P`)`	Cartan characters for a non-linear E at integral element P
`closure(`E`)`	calculates the closure of E under exterior differentiation

`derived_system(`E`)`
`derived_system(`S`)`
	calculates the first derived system of the Pfaffian system E or S

`dim_grassmann_variety(`E`)`
`dim_grassmann_variety(`E`,`P`)`
	dimension of the Grassman bundle variety of integral elements: for non-linear E, the base element P must be given

`dim(`M`)`
`dim(`E`)`
	returns the manifold dimension

`involution(`E`)`	repeatedly prolongs E to involution (or inconsistency)
`linearise(`E`,`P`)`	linearise the (non-linear) EDS E with respect to the integral element P
`integral_element(`E`)`	find a random ⟨*integral element*⟩ of E
`prolong(`E`)`	prolongs E, and projects back down to a subvariety of the original manifold if integrability conditions arise
`tableau(`E`)`	calculates the ⟨*tableau*⟩ of the quasilinear Pfaffian ⟨*EDS*⟩ E
`torsion(`E`)`	returns a ⟨*system*⟩ of 0-forms specifying the integrability conditions for the semilinear or quasilinear Pfaffian ⟨*EDS*⟩ E
`grassmann_variety(`E`)`	returns the contact ⟨*EDS*⟩ for the Grassmann bundle of n-planes over the manifold of E, augmented by the 0-forms specifying the variety of integral elements of E

Table 4: Commands for analysing exterior systems

Command	**Function**
`closed`(E) `closed`(S) `closed`(Ω)	checks for closure under exterior differentiation
`involutive`(E)	applies Cartan's test for involution
`pfaffian`(E)	checks if E is generated by 1-forms and their exterior derivatives
`quasilinear`(E)	tests if the *closure* of E can be generated by forms at most linear in the complement of the independence condition
`semilinear`(E)	tests if the *closure* of E is quasilinear and, in addition, the coefficients of the linear terms contain only independent variables or constants
E `equiv` E'	checks whether E and E' are algebraically equivalent

Table 5: Commands for testing exterior systems

Switch	**Function**
`edsverbose`	if `on`, displays additional information as calculations progress
`edsdebug`	if `on`, produces copious quantities of internal information, in addition to that produced by `edsverbose`
`edssloppy`	if `on`, allows EDS to divide by expressions not known to be non-zero and treats quasilinear systems as semilinear
`edsdisjoint`	if `on`, forces varieties to be decomposed into disjoint components
`ranpos` `genpos`	if `on`, uses a random or generic flag of integral elements when calculating Cartan characters: otherwise the independence condition as presented guides the choice of flag

Table 6: Switches (all `off` by default)

Command	**Function**
`coordinates(`S`)`	scans the expressions in S for coordinates
`invert(`X`)`	returns the inverse $\langle$*transform*$\rangle$ X^{-1}
`structure_equations(`X`)` `structure_equations(`X`,`X^{-1}`)`	returns exterior derivatives of lhs(X)
`linear_divisors(`Ω`)`	calculates a basis for the space of 1-form factors of Ω
`exfactors(`Ω`)`	as for `linear_divisors`, but with the additional (non-linear) factor
`index_expand(`*any*`)`	returns a list of copies of its argument, with free EXCALC indices replaced by successive values from the relevant index range
`pde2jet(`*pde*`,`*dep*`,`*ind*`)`	converts a PDE system into jet bundle notation, replacing derivatives by jet bundle coordinates (variable dependencies are not affected)
`mkdepend(`*list*`)`	restores variable dependencies destroyed by `pde2eds`
`disjoin(`$\{f, g, ...\}$`)`	decomposes the variety specified by the given $\langle$*map*$\rangle$ variables into a disjoint union
`cleanup(`E`)` `cleanup(`M`)`	returns a fresh copy of E or M with all properties and stored results removed
`reorder(`E`)` `reorder(`M`)`	returns a fresh copy of E or M, conforming to the prevailing REDUCE kernel order

Table 7: Auxilliary functions

Command	**Function**
`poincare(`Ω`)`	calculates the homotopy integral from the proof of Poincaré's lemma: if Ω is exact, then the result is a form whose exterior derivative gives back Ω

`invariants(`E`,`crd`)`
`invariants(`S`,`crd`)`
	calculates the invariants (first integrals) of a completely integrable Pfaffian system using the inductive proof of the Frobenius theorem: the optional second argument specifies the order in which the coordinates are to be projected away

`symbol_relations(`E`,`π`)`	returns relations between the entries of the tableau matrix, represented by 2-indexed ⟨*1-form*⟩ variables ${\pi^a}_i$
`symbol_matrix(`E`,`ξ`)`	returns the symbol matrix for a quasilinear ⟨*EDS*⟩ E as a function of ⟨*0-form*⟩ variables ξ_i
`characteristic_variety(`E`,`ξ`)`	returns equations describing the characteristic variety of E in terms of ⟨*0-form*⟩ variables ξ_i

Table 8: Experimental functions (unstable)

References

[1] E A Arais, V P Shapeev and N N Yanenko, Computer realization of Cartan's exterior calculus, *Soviet Math Dokl* **15** (1974) 203–205

[2] R L Byrant, S S Chern, R B Gardner, H L Goldschmidt and P A Griffiths, *Exterior Differential Systems* (Springer Verlag, New York, 1991)

[3] V G Ganzha, S V Meleshko, F A Murzin, V P Shapeev and N N Yanenko, Computer realization of an algorithm for investigating the compatibility of systems of partial differential equations, *Soviet Math Dokl* **24** (1981) 638–640

[4] D H Hartley and R W Tucker, A constructive implementation of the Cartan-Kähler theory of exterior differential systems, *J Symb Comp* **12** (1991) 655

[5] D Hartley and P A Tuckey, *XIDEAL, Gröbner Bases for Exterior Algebra* (REDUCE library package)

[6] E Mansfield and E D Fackerell, Differential Gröbner bases and involutivity of systems of non-linear partial differential equations, *submitted to Eur J Appl Math* 1993

[7] G J Reid, Algorithms for reducing a system of partial differential equations to standard form, determining the dimension of its solutions space and calculating its Taylor series solution, *Eur J Appl Math* **2** (1991) 293–318

[8] E Schrüfer, *EXCALC, a system for doing calculations in the calculus of modern differential geometry, User's manual* (Rand Corporation, Santa Monica, 1986)

[9] W M Seiler, *Applying AXIOM to partial differential equations* (Internal Report 95-17, Universität Karlsruhe, Fakultät für Informatik, 1995)

[10] M Spivak, *A comprehensive introduction to differential geometry* (Publish or Perish, Berkeley, 1979)

[11] HD Wahlquist, Monte Carlo calculation of Cartan characters: using the maximal-slicing, Ricci-flat ideal as an example, *Proc Aspects of General Relativity and Mathematical Physics, eds N Bretón, R Capovilla and T Matos*, (1993) 168–174