# The computer algebra package Crack

Thomas Wolf
School of Mathematical Sciences
Queen Mary and Westfield College
University of London
London E1 4NS
T.Wolf@maths.qmw.ac.uk

Andreas Brand
Fakultät für Mathematik und Informatik
Friedrich Schiller Universität Jena
07740 Jena
Germany
maa@hpux.rz.uni-jena.de

May 3, 1999

## Contents

## 1 The purpose of Crack

The package CRACK attempts the solution of an overdetermined system of ordinary or partial differential equations (ODEs/PDEs) with at most polynomial nonlinearities. Under 'normal circumstances' the number of DEs which describe physical processes matches the number of unknown functions which are involved. Moreover none of those equations can be solved or integrated and integrability conditions yield only identities. Although applying the package CRACK to such problems directly will not be of much help usually, it is possible to investigate difficult DE-systems indirectly by studying analytic properties which would be useful for their solution. In this way overdetermined PDE-systems result.

Applications of CRACK include a program CONLAW for the computation of conservation laws of DEs, a program LIEPDE for the computation of infinitesimal symmetries of DEs and a program APPLYSYM for the computation of symmetry and similarity variables from infinitesimal symmetries.

## 2 Technical details

### 2.1 System requirements

The required system is REDUCE, version 3.6., strictly speaking the PSL version of REDUCE as distributed by the Konrad Zuse Institut / Berlin. Work on compatibility issues aims to provide a CSL REDUCE compatible version of CRACK in near future (by the end of 1998).

Memory requirements depend crucially on the application. The `crack.rlg` file is produced from running `crack.tst` in a 4MB session running REDUCE, version 3.6 under LINUX. On the other hand it is not difficult to formulate problems that consume any amount of memory.

### 2.2 Installation

In a running REDUCE session either do

```
    in "crack.red"$
```

or, in order to speed up computation, either compile it with `on comp$` before the above command, or, generate a fast-loading compiled file once with

```
    faslout "crack"$
    in "crack.red"$
    faslend$
```

and load that file to run CRACK with

```
    load crack$
```

### 2.3 Updates / web demos

A web demo under the address `http://cathode.maths.qmw.ac.uk/demo.html` that was created by Francis Wright and Arrigo Triulzi allows to run problems of restricted size. The latest version is available from

`ftp://ftp.maths.qmw.ac.uk/pub/tw/crack/`. Publications related to CRACK can be found under
`http://www.maths.qmw.ac.uk/~tw/public2.html#2`.

## 2.4 The files

The following files are provided with CRACK

- `crack.red` contains read-in statements of a number of files `cr*.red`.
- `crack.tst` contains test-examples.
- `crack.rlg` contains the output of `crack.tst`.
- `crack.tex` is this manual.

## 2.5 The call

CRACK is called by

`crack`($\{equ_1,\ equ_2,\ \ldots,\ equ_m\}$,
$\{ineq_1,\ ineq_2,\ \ldots,\ ineq_n\}$,
$\{fun_1,\ fun_2,\ \ldots,\ fun_p\}$,
$\{var_1,\ var_2,\ \ldots,\ var_q\}$);

$m, n, p, q$ are arbitrary.

- The $equ_i$ are identically vanishing partial differential expressions, i.e. they represent equations $0 = equ_i$, which are to be solved for the functions $fun_j$ as far as possible, thereby drawing only necessary conclusions and not restricting the general solution.
- The $ineq_i$ are algebraic or differential expressions which must not vanish identically for any solution to be determined, i.e. only such solutions are computed for which none of the expressions $ineq_i$ vanishes identically in all independent variables.
- The dependence of the (scalar) functions $fun_j$ on independent variables must be defined beforehand with `DEPEND` rather than declaring these functions as operators. Their arguments may themselves only be identifiers representing variables, not expressions. Also other unknown functions not in $fun_j$ must not be represented as operators but only using `DEPEND`.
- The functions $fun_j$ and their derivatives may only occur polynomially.
- The $var_k$ are further independent variables, which are not already arguments of any of the $fun_j$. If there are none then the fourth argument is the empty list $\{\}$, although it does no harm to include arguments of functions $fun_j$.
- The dependence of the $equ_i$ on the independent variables and on constants and functions other than $fun_j$ is arbitrary.

- CRACK can be run in automatic batch mode (by default) or interactively with the switch `OFF BATCH_MODE`.

## 2.6 The result

The result is a list of solutions

$$\{sol_1, \ldots\}$$

where each solution is a list of 4 lists:

$$\begin{array}{l}\{\{con_1,\ con_2, \ldots,\ con_q\},\\ \ \{fun_a = ex_a,\ \ fun_b = ex_b, \ldots,\ \ fun_p = ex_p\},\\ \ \{fun_c,\ \ fun_d, \ldots,\ \ fun_r\},\\ \ \{ineq_1,\ ineq_2,\ \ldots,\ ineq_s\}. \qquad\qquad\qquad \}\end{array}$$

For example, in the case of a linear system as input, there is at most one solution $sol_1$.

If CRACK finds a contradiction as e.g. $0 = 1$ then there exists no solution and it returns the empty list $\{\}$. If CRACK can factorize algebraically a non-linear equation then factors are set to zero individually and different sub-cases are studied through CRACK calling itself recursively. If during such a recursive call a contradiction results, then this sub-case will not have a solution but other sub-cases still may have solutions. The empty list is also returned if no solution exists which satisfies the inequalities $ineq_i \neq 0$.

The expressions $con_i$ (if there are any), are the remaining necessary and sufficient conditions for the functions $fun_c, \ldots, fun_r$ in the third list. Those functions can be original functions from the equations to be solved (of the second argument of the call of CRACK) or new functions or constants which arose from integrations. The dependence of new functions on variables is declared with `DEPEND` and to visualize this dependence the algebraic mode function `FARGS`($fun_{\mathtt{i}}$) can be used. If there are no $con_i$ then all equations are solved and the functions in the third list are unconstrained. The elements of the fourth list are the expressions who have been assumed to be unequal zero in the derivation of this solution.

The second list contains equations $fun_i = ex_i$ where each $fun_i$ is an original function and $ex_i$ is the computed expression for $fun_i$.

## 2.7 Interactive mode, flags, parameters and the list of procedures

Under normal circumstances one will try to have problems solved automatically by CRACK. An alternative is to input `OFF BATCH_MODE;` before calling CRACK and solve problems interactively. In interactive mode it is possible to

- inspect data, like equations and their properties, unknown functions, variables, identities, a statistics,
- save, change, add or drop equations,

- inspect and change flags and parameters which govern individual modules as well as their interplay,
- specify how to proceed, like doing
  - one automatic step,
  - one specific step,
  - a number of automatic steps,
  - a specific step as often as possible.

To get interactive help one enters 'h' or '?'.

Flags and parameters are stored as symbolic fluid variables which means that they can be accessed by `lisp( ... )`, like `lisp( print_:=5 );` before calling Crack. `print_`, for example, is a measure of the maximal length of expressions still to be printed on the screen (the number of factors in terms). A complete list of flags and parameters is given at the beginning of the file `crinit.red`.

One more parameter shall be mentioned, which is the list of modules/procedures called `proc_list_`. In interactive mode this list can be looked at with `'p'` or be changed with `'cp'`. This list defines in which order the different modules/procedures are tried whenever Crack has to decide of what to do next. There are exceptions to this rule possible. Some procedures, say $P_1$, require after their execution another specific procedure, say $P_2$, to be executed, independent of whether $P_2$ would be next according to `proc_list_`. This is managed by $P_1$ writing after its completion the procedure $P_2$ into a hot-list. This list is dealt with in the `'to_do'` step which comes always first in `proc_list_`. A way to have the convenience of running Crack automatically and still being able to break the fixed rhythm prescribed by `proc_list_` is to have the entry `stop_batch` in `proc_list_` and have Crack started in automatic batch mode. Then execution is continuing until none of the procedures which come before `stop_batch` are applicable any more so that `stop_batch` is executed next which will stop automatic execution and go into interactive mode. This allows either to continue the computation interactively, or to change the `proc_list_` with `'cp'` and to continue in automatic mode.

The default value of `proc_list_` does not include all possible modules because not all are suitable for any kind of overdetermined system to be solved. The complete list is shown in interactive mode under `'cp'`. A few basic modules are described in the following section. The efficiency of Crack in automatic mode is very much depending on the content of `proc_list_` and the sequence of its elements. Optimizing `proc_list_` for a given task needs experience which can not be formalized in a few simple rules and will therefore not be explained in more detail here. The following remarks are only guidelines.

`to_do` : hot list of steps to be taken next, should always come first,

`subst_level_?` : substitutions of functions by expressions differing by their maximal allowed size and other properties,

`separation` : what is described as direct separation in the next section, `gen_separation` : what is as indirect separation in the next section, only to be used for linear problems,

`quick_integration` : integration of very specific short equations,

`full_integration` : integration of equations which have the chance to lead to a substitution,

`integration` : any integration,

`factorization` : splitting the computation into the investigation of different subcases resulting from the algebraic factorization of an equation, only useful for non-linear problems,

`undetlinode` : parametric solution of single under determined linear ODE (with non-constant coefficients), only applicable for linear problems,

`length_reduction_1` : length reduction by algebraic combination, only for linear problems,

`length_reduction_2` : length reduction by differential reduction,

`decoupling` : steps towards the computation of a differential Gröbner Basis,

`add_differentiated_pdes` : only useful for non-linear differential equations with leading derivative occuring non-linearly,

`add_diff_star_pdes` : for the treatment of non-linear indirectly separable equations,

`multintfac` : to find integrating factors of for a system of equations (very slow),

`alg_solve_deriv` : to be used for equations quadratic in the leading derivative,

`alg_solve_system` : to be used if a (sub-)system of equations shall be solved for a set of functions or their derivatives algebraically,

`subst_derivative` : substitution of a derivative of a function everywhere by a new function if such a derivative exists

`undo_subst_derivative` : undo the above substitution.

## 3 Contents of the Crack package

The package Crack contains a number of modules. The basic ones are for computing a pseudo differential Gröbner Basis (using integrability conditions in a systematic way), integrating exact PDEs, separating PDEs, solving DEs containing functions of only a subset of all variables and solving standard ODEs (of Bernoulli or Euler type, linear, homogeneous and separable ODEs). These facilities will be described briefly together with examples. The test file `crack.tst` demonstrates these and others.

## 3.1 Pseudo Differential Gröbner Basis

This module (called 'decoupling' in `proc_list_`) reduces derivatives in equations by using other equations and it applies integrability conditions to formulate additional equations which are subsequently reduced, and so on.

A general algorithm to bring a system of PDEs into a standard form where all integrability conditions are satisfied by applying a finite number of additions, multiplications and differentiations is based on the general theory of involutive systems [1, 2, 3].

Essential to this theory is a total ordering of partial derivatives which allows assignment to each PDE of a *Leading Derivative* (LD) according to a chosen ordering of functions and derivatives. Examples for possible orderings are

- lex. order of functions $>$ lex. order of variables,
- lex. order of functions $>$ total differential order $>$ lex. order of variables,
- total order $>$ lex. order of functions $>$ lex. order of variables

or mixtures of them by giving weights to individual functions and variables. Above, the '$>$' indicate "before" in priority of criteria. The principle is then to

- take two equations at a time and differentiate them as often as necessary to get equal LDs,
- regard these two equations as algebraic equations in the common LD and calculate the remainder w.r.t. the LD, i.e. to generate an equation without the LD by the Euclidean algorithm, and
- add this equation to the system.

Usually pairs of equations are taken first, such that only one must be differentiated. If in such a generation step one of both equations is not differentiated then it is called a simplification step and this equation will be replaced by the new equation.

The algorithm ends if each combination of two equations yields only equations which simplify to an identity modulo the other equations. A more detailed description is given e.g. in [5, 6].

Other programs implementing this algorithm are described e.g. in [7, 5, 8, 6] and [9].

In the interactive mode of CRACK it is possible to change the lexicographical ordering of variables, of functions, to choose between 'total differential order' ordering of variables or lexicographical ordering of variables and to choose whether lexicographical ordering of functions should have a higher priority than the ordering of the variables in a derivative, or not.

An example of the computation of a differential Gröbner Basis is given in the test file `crack.tst`.

## 3.2 Integrating exact PDEs

The technical term 'exact' is adapted for PDEs from exterior calculus and is a small abuse of language but it is useful to characterize the kind of PDEs under consideration.

The purpose of the integration module in CRACK is to decide whether a given differential expression $D$ which involves unknown functions $f^i(x^j), \ \ 1 \leq i \leq m$ of independent variables $x^j, 1 \leq j \leq n$ is a total derivative of another expression $I$ w.r.t. some variable $x^k, 1 \leq k \leq n$

$$D(x^i,\ f^j,\ f^j{}_{,p}\,,\ f^j{}_{,pq}\,,\ldots) = \frac{dI(x^i,\ f^j,\ f^j{}_{,p}\,,\ f^j{}_{,pq}\,,\ldots)}{dx^k}.$$

The index $k$ is reserved in the following for the integration variable $x^k$. With an appropriate function of integration $c^r$, which depends on all variables except $x^k$ it is no loss of generality to replace $0 = D$ by $0 = I + c^r$ in a system of equations.

Of course there always exists a function $I$ with a total derivative equal to $D$ but the question is whether for arbitrary $f^i$ the integral $I$ is functionally dependent only on the $f^i$ and their derivatives, and not on integrals of $f^i$.

Preconditions:

$D$ is a polynomial in the $f^i$ and their derivatives. The number of functions and variables is free. For deciding the existence of $I$ only, the explicit occurrence of the variables $x^i$ is arbitrary. In order to actually calculate $I$ explicitly, $D$ must have the property that all terms in $D$ must either contain an unknown function of $x^k$ or must be formally integrable w.r.t. $x^k$. That means if $I$ exists then only a special explicit occurrence of $x^k$ can prevent the calculation of $I$ and furthermore only in those terms which do not contain any unknown function of $x^k$. If such terms occur in $D$ and $I$ exists then $I$ can still be expressed as a polynomial in the $f^i, f^i{}_{,j}\,,\ldots$ and terms containing indefinite integrals with integrands explicit in $x^k$.

Algorithm:

Successive partial integration of the term with the highest $x^k$-derivative of any $f^i$. By that the differential order w.r.t. $x^k$ is reduced successively. This procedure is always applicable because steps involve only differentiations and the polynomial integration ($\int h^n \frac{\partial h}{\partial x} dx = h^{n+1}/(n+1)$) where $h$ is a partial derivative of some function $f^i$. For a more detailed description see [12].

Stop:

Iteration stops if no term with any $x^k$-derivative of any $f^i$ is left. If in the remaining un-integrated terms any $f^i(x^k)$ itself occurs, then $I$ is not expressible with $f^i$ and its derivatives only. In case no $f^i(x^k)$ occurs then any remaining terms can contain $x^k$ only explicitly. Whether they can be integrated depends on their formal integrability. For their integration the REDUCE integrator is applied.

Speed up:

The partial integration as described above preserves derivatives with respect to other variables. For example, the three terms $f_{,x}\,, f f_{,xxx}\,, f_{,xxy}$ can not combine somehow to the same terms in the integral because if one ignores $x$-derivatives then it is clear that $f, f^2$ and $f_{,y}$ are like three completely different expressions from the point of view of $x$-integrations. This allows the following drastic speed up for large expressions. It is possible to partition the complete sum of terms into partial sum such that each of the partial sum has to be integrable on its own. That is managed by generating a label for each term and collecting terms with equal label into partial sums. The label is produced by dropping all $x$-derivatives from all functions to be computed and dropping all factors which are not powers of derivatives of functions to be computed.

The partitioning into partial sums has two effects. Firstly, if the integration of one partial sum fails then the remaining sums do not have to be tried for integration. Secondly, doing partial integration for each term means doing many subtractions. It is much faster to subtract terms from small sums than from large sums.

Example :
We apply the above algorithm to

$$D := 2f_{,y}\, g' + 2f_{,xy}\, g + gg'^3 + xg'^4 + 3xgg'^2 g'' = 0 \tag{1}$$

with $f = f(x,y)$, $g = g(x)$, $' \equiv d/dx$. Starting with terms containing $g$ and at first with the highest derivative $g_{,xx}$ , the steps are

$$\begin{aligned} \int 3xgg_{,x}^2\, g_{,xx}\, dx &= \int d(xgg_{,x}^3) - \int (\partial_x(xg)g_{,x}^3)\, dx \\ &= xgg_{,x}^3 - \int g_{,x}^3\,(g + xg_{,x})dx, \end{aligned}$$

$$I := I + xgg_{,x}^3$$

$$D := D - g_{,x}^3\,(g + xg_{,x}) - 3xgg_{,x}^2\, g_{,xx}$$

The new terms $-g_{,x}^3\,(g + xg_{,x})$ are of lower order than $g_{,xx}$ and so in the expression $D$ the maximal order of $x$-derivatives of $g$ is lowered. The conditions that $D$ is exact are the following.

- The leading derivative must occur linearly before each partial integration step.
- After the partial integration of the terms with first order $x$-derivatives of $f$ the remaining $D$ must not contain $f$ or other derivatives of $f$, because such terms cannot be integrated w.r.t. $x$ without specifying $f$.

The result of $x$- and $y$-integration in the above example is (remember $g = g(x)$)

$$0 = 2fg + xygg_{,x}^3 + c_1(x) + c_2(y) \;\; (= I). \tag{2}$$

Crack can now eliminate $f$ and substitute for it in all other equations.
Generalization:
If after applying the above basic algorithm, terms are left which contain functions of $x^k$ but each of these functions depends only on a subset of all $x^i, 1 \leq i \leq n$, then a generalized version of the above algorithm can still provide a formal expression for the integral $I$ (see [12]). The price consists of additional differential conditions, but they are equations in less variables than occur in the integrated equation. Integrating for example

$$\tilde{D} = D + g^2(y^2 + x \sin y + x^2 e^y) \tag{3}$$

by introducing as few new functions and additional conditions as possible gives as the integral $\tilde{I}$

$$\begin{aligned}\tilde{I} &= 2fg + xygg,_x^3 + c_1(x) + c_2(y) \\ &\quad + \frac{1}{3}y^3 c_3'' - \cos y(xc_3'' - c_3) + e^y(x^2 c_3'' - 2xc_3' + 2c_3)\end{aligned}$$

with $c_3 = c_3(x)$, $' \equiv d/dx$ and the single additional condition $g^2 = c_3'''$. The integration of the new terms of (3) is achieved by partial integration again, for example

$$\begin{aligned}\int g^2 x^2 dx &= x^2 \int g^2 dx - \int (2x \int g^2 dx) dx \\ &= x^2 \int g^2 dx - 2x \int\int g^2 dx + 2 \int\int\int g^2 dx \\ &= x^2 c_3'' - 2xc_3' + 2c_3.\end{aligned}$$

Characterization:
This algorithm is a decision algorithm which does not involve any heuristic. After integration the new equation is still a polynomial in $f^i$ and in the new constant or function of integration. Therefore the algorithms for bringing the system into standard form can still be applied to the PDE-system after the equation $D = 0$ is replaced by $I = 0$.

The complexity of algorithms for bringing a PDE-system into a standard form depends nonlinearly on the order of these equations because of the nonlinear increase of the number of different leading derivatives and by that the number of equations generated intermediately by such an algorithm. It therefore in general pays off to integrate equations during such a standard form algorithm.

If an $f^i$, which depends on all variables, can be eliminated after an integration, then depending on its length it is in general helpful to substitute $f^i$ in other equations and to reduce the number of equations and functions by one. This is especially profitable if the replaced expression is short and contains only functions of less variables than $f^i$.
Test:
The corresponding test input is

```
depend f,x,y;
depend g,x;
crack({2*df(f,y)*df(g,x)+2*df(f,x,y)*g+g*df(g,x)**3
       +x*df(g,x)**4+3*x*g*df(g,x)**2*df(g,x,2)
       +g**2*(y**2+x*sin y+x**2*e**y)},
      {},{f,g},{});
```

The meaning of the REDUCE command `depend` is to declare that $f$ depends in an unknown way on $x$ and $y$. For more details on the algorithm see [12].

## 3.3 Direct separation of PDEs

As a result of repeated integrations the functions in the remaining equations have less and less variables. It therefore may happen that after a substitution an equation

results where at least one variable occurs only explicitly and not as an argument of an unknown function. Consequently all coefficients of linearly independent expressions in this variable can be set to zero individually.
*Example:*
$f = f(x, y)$, $g = g(x)$, $x, y, z$ are independent variables. The equation is

$$0 = f_{,y} + z(f^2 + g_{,x}) + z^2(g_{,x} + yg^2) \qquad (4)$$

$x$-separation? $\rightarrow$ no
$y$-separation? $\rightarrow$ no
$z$-separation? $\rightarrow$ yes: $0 = f_{,y} = f^2 + g_{,x} = g_{,x} + yg^2$
$y$-separation? $\rightarrow$ yes: $0 = g_{,x} = g^2$ (from the third equation from the $z$-separation)

If $z^2$ had been replaced in (4) by a third function $h(z)$ then direct separation would not have been possible. The situation changes if $h$ is a parametric function which is assumed to be independently given and which should not be calculated, i.e. $f$ and $g$ should be calculated for any arbitrary given $h(z)$. Then the same separation could have been done with an extra treatment of the special case $h_{,zz} = 0$, i.e. $h$ linear in $z$. This different treatment of unknown functions makes it necessary to input explicitly the functions to be calculated as the third argument to CRACK. The input in this case would be

```
depend f,x,y;
depend g,x;
depend h,z;
crack({df(f,y)+z*f**2+(z+h)*df(g,x)+h*y*g**2},{},{f,g},{z});
```

The fourth parameter for CRACK is necessary to make clear that in addition to the variables of $f$ and $g$, $z$ is also an independent variable.

If the flag `independence_` is not `nil` then CRACK will stop if linear independence of the explicit expressions of the separation variable (in the example $1, z, z^2$) is not clear and ask interactively whether separation should be done or not.

## 3.4 Indirect separation of PDEs

For the above direct separation a precondition is that at least one variable occurs only explicitly or as an argument of parametric functions. The situation where each variable is an argument of at least one function but no function contains all independent variables of an equation needs a more elaborate treatment.

The steps are these

- A variable $x_a$ is chosen which occurs in as few functions as possible. This variable will be separated directly later which requires that all unknown functions $f_i$ containing $x_a$ are to be eliminated. Therefore, as long as $F := \{f_i\}$ is not empty do the following:
  - Choose the function $f_i(y_p)$ in $F$ with the smallest number of variables $y_p$ and with $z_{ij}$ as those variables on which $f_i$ does not depend.

  - Identify all different products $P_{ik}$ of powers of $f_i$-derivatives and of $f_i$ in the equation. Determine the $z_{ij}$-dependent factors $C_{ik}$ of the coefficients of $P_{ik}$ and store them in a list.
  - For each $C_{il}$ ($i$ fixed, $l = 1, \ldots$) choose a $z_{ij}$ and :
    * divide by $C_{il}$ the equation and all following elements $C_{im}$ with $m > l$ of this list, such that these elements are still the actual coefficients in the equation after the division,
    * differentiate the equation and the $C_{im}, m > l$ w.r.t. $z_{ij}$

- The resulting equation no longer contains any unknown function of $x_a$ and can be separated w.r.t. $x_a$ directly in case $x_a$ still occurs explicitly. In both cases the equation(s) is (are) free of $x_a$ afterwards and inverting the sequence of integration and multiplication of all those equations (in case of direct separability) will also result in an equation(s) free of $x_a$. More exactly, the steps are
  - multiplication of the equation(s) and the $C_{im}$ with $m < l$ by the elements of the $C_{ik}$-lists in exactly the inverse order,
  - integration of these exact PDEs and the $C_{im}$ w.r.t. $z_{ij}$.
- The equations originating that way are used to evaluate those functions which do not depend on $x_a$ and which survived in the above differentiations. Substituting these functions in the original equation, may enable direct separability w.r.t. variables on which the $f_i$ do not depend on.
- The whole procedure is repeated for another variable $x_b$ if the original DE could not be separated directly and still has the property that it contains only functions of a subset of all variables in the equation.

The additional bookkeeping of coefficients $C_{ik}$ and their updating by division, differentiation, integration and multiplication is done to use them as integrating factors for the backward integration. The following example makes this clearer. The equation is

$$0 = f(x)g(y) - \frac{1}{2}xf'(x) - g'(y) - (1 + x^2)y. \tag{5}$$

The steps are (equal levels of indentation in the example correspond to those in the algorithm given above)

- $x_1 := x$, $F = \{f\}$
  - Identify $f_1 := f, \quad y_1 := x, \quad z_{11} := y$
  - and $P_1 = \{f', f\}, \quad C_1 = \{1, g\}$
    * Divide $C_{12}$ and (5) by $C_{11} = 1$ and differentiate w.r.t. $z_{11} = y$ :

$$\begin{aligned} 0 &= fg' - g'' - (1 + x^2) \qquad (6) \\ C_{12} &= g' \end{aligned}$$

  - $*$ Divide (6) by $C_{12} = g'$ and differentiate w.r.t. $z_{11} = y$ :

$$0 = -(g''/g')' - (1+x^2)(1/g')'$$

- Direct separation w.r.t. $x$ and integration:

$$\begin{array}{rcllll} x^2 : 0 & = & (1/g')' & \Rightarrow\; c_1 g' = 1 & \Rightarrow\; g = y/c_1 + c_2 \\ x^0 : 0 & = & (g''/g')' & \Rightarrow\; c_3 g' = g'' & \Rightarrow\; c_3 = 0 \end{array}$$

- Substitution of $g$ in the original DE

$$0 = (y/c_1 + c_2)f - \frac{1}{2}xf' - 1/c_1 - (1+x^2)y$$

  provides a form which allows Crack standard methods to succeed by direct separation w.r.t. $y$

$$\begin{array}{rcll} y^1 : 0 & = & f/c_1 - 1 - x^2 & \Rightarrow\; f' = 2c_1 x \\ y^0 : 0 & = & c_2 f - \frac{1}{2}xf' - 1/c_1 & \Rightarrow\; 0 = c_2 c_1 (1+x^2) - c_1 x^2 - 1/c_1 \end{array}$$

  and direct separation w.r.t. $x$:

$$\begin{array}{rcl} x^0 : 0 & = & c_2 c_1 - c_1 \\ x^2 : 0 & = & c_2 c_1 - 1/c_1 \\ & \Rightarrow & 0 = c_1 - 1/c_1 \\ & \Rightarrow & c_1 = \pm 1 \Rightarrow c_2 = 1. \end{array}$$

We get the two solutions $f = 1 + x^2, g = 1 + y$ and $f = -1 - x^2, g = 1 - y$. The corresponding input to Crack would be

```
depend f,x;
depend g,y;
crack({f*g-x*df(f,x)/2-df(g,y)-(1+x**2)*y},{},{f,g},{});
```

## 3.5 Solving standard ODEs

For solving standard ODEs the package ODESolve by Malcalm MacCallum and Francis Wright [14] is applied. This package is distributed with Reduce and can be used independently of Crack. The syntax of ODESolve is quite similar to that of Crack

`depend` *function*, *variable*;

`odesolve(`ODE, *function*, *variable*);

In the present form (1998) it solves standard first order ODEs (Bernoulli and Euler type, with separable variables, ...) and linear higher order ODEs with constant coefficients. An improved version is currently under preparation by Francis Wright. The applicability of ODESolve is increased by a Crack-subroutine which recognizes such PDEs in which there is only one unknown function of all variables and all occurring derivatives of this function are only derivatives w.r.t. one variable of only one partial derivative. For example the PDE for $f(x,y)$

$$0 = f_{,xxy} + f_{,xxyy}$$

can be viewed as a first order ODE in $y$ for $f_{,xxy}$ .

## 4 Acknowledgement

Francis Wright contributed a module that provides simplifications of expressions involving symbolic derivatives and integrals. Also, CRACK makes extensive use of the REDUCE program ODESOLVE written by Malcolm MacCallum and Francis Wright.

Arrigo Triulzi provided a module to support the use of different total orderings of derivatives in doing pseudo differential Gröbner basis computations.

Work on this package has been supported by the Konrad Zuse Institut/Berlin through a fellowship of T.W.. Winfried Neun and Herbert Melenk are thanked for many discussions and constant support.

Anthony Hearn provided free copies of REDUCE to us as a REDUCE developers group which also is thankfully acknowledged.

## References

[1] C. Riquier, Les systèmes d'équations aux dérivées partielles, Gauthier–Villars, Paris (1910).

[2] J. Thomas, Differential Systems, AMS, Colloquium publications, v. 21, N.Y. (1937).

[3] M. Janet, Leçons sur les systèmes d'équations aux dérivées, Gauthier–Villars, Paris (1929).

[4] V.L. Topunov, Reducing Systems of Linear Differential Equations to a Passive Form, Acta Appl. Math. 16 (1989) 191–206.

[5] A.V. Bocharov and M.L. Bronstein, Efficiently Implementing Two Methods of the Geometrical Theory of Differential Equations: An Experience in Algorithm and Software Design, Acta. Appl. Math. 16 (1989) 143–166.

[6] G.J. Reid, A triangularization algorithm which determines the Lie symmetry algebra of any system of PDEs, J.Phys. A: Math. Gen. 23 (1990) L853-L859.

[7] F. Schwarz, Automatically Determining Symmetries of Partial Differential Equations, Computing 34, (1985) 91-106.

[8] W.I. Fushchich and V.V. Kornyak, Computer Algebra Application for Determining Lie and Lie–Bäcklund Symmetries of Differential Equations, J. Symb. Comp. 7, (1989) 611–619.

[9] E.L. Mansfield, The differential algebra package diffgrob2, Mapletech 3, (1996) 33-37 .

[10] E. Kamke, Differentialgleichungen, Lösungsmethoden und Lösungen, Band 1, Gewöhnliche Differentialgleichungen, Chelsea Publishing Company, New York, 1959.

[11] T. Wolf, An Analytic Algorithm for Decoupling and Integrating systems of Nonlinear Partial Differential Equations, J. Comp. Phys., no. 3, 60 (1985) 437-446 and, Zur analytischen Untersuchung und exakten Lösung von Differentialgleichungen mit Computeralgebrasystemen, Dissertation B, Jena (1989).

[12] T. Wolf, The Symbolic Integration of Exact PDEs, preprint, (1991).

[13] M.A.H. MacCallum, F.J. Wright, Algebraic Computing with REDUCE, Clarendon Press, Oxford (1991).

[14] M.A.H. MacCallum, An Ordinary Differential Equation Solver for REDUCE, Proc. ISAAC'88, Springer Lect. Notes in Comp Sci. 358, 196–205.

[15] H. Stephani, Differential equations, Their solution using symmetries, Cambridge University Press (1989).

[16] T. Wolf, An efficiency improved program LiePDE for determining Lie - symmetries of PDEs, Proceedings of the workshop on Modern group theory methods in Acireale (Sicily) Nov. (1992)

[17] V.I. Karpman, Phys. Lett. A 136, 216 (1989)

[18] B. Champagne, W. Hereman and P. Winternitz, The computer calculation of Lie point symmetries of large systems of differential equation, Comp. Phys. Comm. 66, 319-340 (1991)