The computer algebra package CRACK for solving over-determined systems of equations

Thomas Wolf
Department of Mathematics
Brock University
St.Catharines
Ontario, Canada L2S 3A1
twolf@brocku.ca

January 13, 2025

Contents

1 Introduction

1.1 Purpose

The package CRACK attempts the solution of an overdetermined system of algebraic, ordinary or partial differential equations (ODEs/PDEs) with at most polynomial nonlinearities.

Under 'normal circumstances' differential equations (DEs) which describe physical processes are not overdetermined, i.e. the number of DEs matches the number of unknown functions which are involved. Although CRACK may be successful in such cases (e.g. for characteristic ODE-systems of first order PDEs) this is not the typical application. It is rather the qualitative investigations of such differential equations, i.e. the investigation of their infinitesimal symmetries (with LIEPDE,APPLYSYM) and conservation laws (with CONLAW) which result in over-determined systems which are the main application area of CRACK.

1.2 Interactivity

The package was originally developed to run automatically and effort was taken for the program to decide which computational steps are to be done next with a choice among integrations, separations, substitutions and investigation of integrability conditions. It is known from hand computations that the right sequence of operations with exactly the right equations at the right

time is often crucial to avoid an explosion of the length of expressions. This statement keeps its truth for the computerized solution of systems of equations as they become more complex. As a consequence more and more interactive access has been provided to inspect data, to specify how to proceed with the computation and how to control it. This allows the human intervention in critical stages of the computations (see the switch *off batch_mode* below).

1.3 General Structure

A problem consists of a system of equations and a set of inequalities. With each equation are associated a short name and numerous data, like size, which functions, derivatives and variables occur but also which investigations have already been done with this equation and which not in order to avoid unnecessary duplication of work. These data are constantly updated if the equation is modified in any way.

A set of about 30 modules is available to integrate, substitute, decouple, ... equations. A complete list can be inspected in interactive mode with the command *p2*, each operation is listed with the number it is called. All modules can be called interactively or automatically. Automatic computation is organized by a priority list of modules (each represented by a number) where modules are invoked in the order they appear in the priority list, each module trying to find equations in the system it can be applied to. The priority list can be inspected with the command *p1*. If a module is not successful then the next module in the list is tried, if any one is successful then execution starts again at the beginning of the priority list. {prog_list_, default_proc_list_, full_proc_list_ }

Because each module has access to all the data, it is enough to call a module by its number. For example, the input of the number 2 in interactive mode will start the direct separation module (see below) to look for a directly separable equation and will split it.

2 Syntax

2.1 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`.

2.2 The result

The result is a list of solutions

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

where each solution is a list of 4 lists:

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

For example, in the case of a linear system, the input consists of 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 by 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 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. The elements of the fourth list are the expressions who have been assumed to be unequal zero in the derivation of this solution.

2.3 Automatic vs. interactive

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 to 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,
- add inequalities,

- inspect and change flags and parameters which govern individual modules as well as their interplay,
- pick a list of methods to be used out of about 30 different ones, and specify their priorities and in this way very easily compose an automatic solving strategy,
- or, for more interactive work, to specify how to proceed, i.e. which computational step to do and how often, like doing

  one automatic step,

  one specific step,

  a number of automatic steps,

  a specific step as often as possible or a specified number of times.

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 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. Exceptions to this rule may be specified. For example, some procedure, say P_1, requires after its execution another specific procedure, say P_2, to be executed, no matter whether P_2 is next according to `proc_list_` or not. This is managed by P_1 writing a task for procedure P_2 into a hot-list. Tasks listed in the global variable `'to_do_list'` are dealt with in the `'to_do'` step which should always come 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.

3 Modules

The following modules are represented by numbers in the priority list. Each module can appear with modifications under different numbers. For example, integration is available under *7*, *24* and *25*. Here *7* encodes an integrations of short equations $0 = \partial^n f/\partial x^n$. *7* has highest priority of the three integrations. *24* encodes the integration of an equation that leads to the substitution of a function and *25* refers to any integration and has lowest priority.

3.1 Integration and Separation

An early feature in the development of the package CRACK was the ability to integrate exact differential equations and some generalizations of them (see [22]). As a consequence of integrations *7*, *24*, *25* an increasing number of functions of fewer variables is introduced which sooner or later produces equations with some independent variables occuring only explicitly and not as variables in functions. Such equations are splitted by the separation module *2*. Another possibility are equations in which each independent variable occurs in at least one function in the equation but no function depends on all variables. In this case so-called indirect separations are appropriate: for linear problems *10 26* and for non-linear problems *48*.

3.2 Substitutions

Substitutions can have a dramatic effect on the size and complexity of systems. Therefore it is possible to have them not only done automatically but also controlled tightly, either by specifying exactly what equation should be used to substitute which unknown and where, or by picking a substitution out of a list of substitutions offered by the program {*cs*} . Substitutions to be performed automatically can be controlled with a number of filters, for example, by

- limiting the size of the equation to be used for substitution, {*length_limit*}
- limiting the size of equations in which the substitution is to be done, {*target_limit_*}
- allowing only linear equations to be used for substitutions, {*lin_subst*}
- allowing equations to increase in size only up to some factor in order for a substitution to be performed in that equation, {*cost_limit*}
- allowing a substitution for a function through an expression only if that expression involves exclusively functions of fewer variables, {*less_vars*}
- allowing substitutions only that do not lead to a case distinction coefficient = 0 or not,
- specifying whether extra effort should be spent to identify the substitution with the lowest bound on growth of the full system. {*min_growth*}

Substitution types are represented by different numbers (3-6,15-21) depending on the subset of the above filters to be used. If a substitution type is to be done automatically then from all possible substitutions passing all filters of this type that substitution is selected that leads to no sub-cases (if available) and that uses the shortest equation.

3.3 Factorization

It is very common that big algebraic systems contain equations that can be factorized. Factorizing an equation and setting the factors individually to zero simplifies the whole task because factors are simpler expressions than the whole equation and setting them to zero they may lead to substitutions and thereby further simplifications. The downside is that if problems with, say 100 unknowns, would need 40 case-distinctions in order to be able to solve automatically for the remaining 60 unknowns then this would require 2^{40} cases to be investigated which is impractical. The problem is to find the right balance, between delaying case-distinctions in order not to generate too many cases and on the other hand introducing case distinctions as early as necessary in order to simplify the system. This simplification may be necessary to

solve the system but in any case it will speed up its solution (although at the price of having to solve a simplified system at least twice, depending on the number of factors).

For large systems with many factorizable equations the careful selection of the next equation to be factorized is important to gain the most from each factorization and to succeed with as few as possible factorizations. Criteria which give factors and therefore equations a higher priority are

- the number of equations in which this factor occurs,
- if the factor is a single unknown function or constant, then the number of times this unknown turns up in the whole system,
- the total degree of the factor,
- the number of factors of an equation,
- and others.

It also matters in which order the factors are set to zero. For example, the equation $0 = ab$ can be used to split into the 2 cases: 1. $a = 0$, 2. $a \neq 0, b = 0$ or to split into the 2 cases 1. $b = 0$, 2. $b \neq 0, a = 0$. If one of the 2 factors, say b, involves functions which occur only linearly then this property is to be preserved and these functions should be substituted as such substitutions preserve their linearity. But to have many such substitutions available, it is useful to know of many non-linearly occuring functions to be non-zero as they occur as coefficients of the linearly occuring functions. In the above situation it is therefore better to do the first splitting 1. $a = 0$, 2. $a \neq 0, b = 0$ because $a \neq 0$ will be more useful for substitutions of linear functions than $b \neq 0$ would be.

An exception of this plausible rule occurs towards the end of all the substitutions of all the linearly occuring b_i when some b_i are an overall factor to many equations. If one would then set, say, $b_{22} = 0$ as the second case in a factorization, the first case would generate as subcases factorizations of other equations where $b_{22} = 0$ would be the second case again and so on. To avoid this one should investigate $b_{22} = 0$ as the first case in the first factorization.

The only purpose of that little thought experiment was to show that simple questions, like 'Which factored equation should be used first for case-distinctions and in which order to set factors to zero?' can already be difficult to answer in general.

CRACK currently offers two factorization steps: (8) and (47).

3.4 Elimination (Gröbner Basis) Steps

To increase safety and avoid excessive expression swell one can apart from the normal call (30) request to do Gröbner basis computation steps only if they are simplification steps replacing an equation by a shorter equation. (27)

In a different version only steps are performed in which equations are included which do not contain more than 3 unknowns. This helps to focus on steps which are more likely to solve small sub-systems with readily available simple results. (57)

Often the computationally cheapest way to obtain a consistent (involutive) system of equations implies to change the ordering during the computation. This is the case when substitutions of functions are performed which are not ranked highest in a lexicographical ordering of functions. But CRACK also offers an interactive way to

- change the lexicographical ordering of variables, $\{ov\}$
- change the lexicographical ordering of functions, $\{of\}$

- give the differential order of derivatives a higher or lower priority in the total ordering than the lexicographical ordering of functions, {*og*}
- give either the total differential order of a partial derivative a higher priority than the lexicographical ordering of the derivative of that function or to take the lexicographical ordering of derivatives as the only criterium. {*of*}

3.5 Solution of an under-determined differential equation

When solving an over-determined system of linear differential equations where the general solution involves free functions, then in the last computational step often a single equation for more than one function remains to be solved. Examples are the computation of symmetries and conservation laws of non-linear differential equations which are linearizable. In CRACK two procedures are available, one for under-determined linear ODEs (22) and one for linear PDEs, (23) both with non-constant coefficients.

3.6 Indirect Separation

Integrations introduce new functions of fewer variables. As equations are used to substitute functions of all variables it is only a question of time that equations are generated in which no function depends on all variables. If at least one variable occurs only explicitly then the equation can be splitted which we call direct separation. But sometimes all variables appear as variables to unknown functions, e.g. $0 = f(x) + g(y)$ although usually much more complicated with 10 or 20 independent variables and many functions that are depending on different combinations of these variables. Because no variable occurs only explicitly, direct separations mentioned above are not possible. Two different algorithms, one for linear indirectly separable equations (10), (26) and one for non-linear directly separable equations (48) provide systematic ways of dealing with such equations.

Indirectly separable equations always result when an equation is integrated with respect to different variables, like $0 = f_{xy}$ to $f = g(x) + h(y)$ and a function, here $f(x, y)$, is substituted.

3.7 Function and variable transformations

In the interactive mode one can specify a transformation of the whole problem with *pt* in which old functions and variables are expressed in a mix of new functions and variables.

3.8 Solution of first order linear PDE

If a system contains a single linear first order PDE for just one function then in an automatic step characteristic ODE-systems are generated, integrated if possible, a variable transformation for the whole system of equations is performed to have in the first order PDE only one single derivative and to make this PDE integrable for the integration modules. (39)

3.9 Length Reduction of Equations

An algorithm designed originally to length-reduce differential equations proved to be essential in length reducing systems of bi-linear algebraic equations or homogeneous equations which resulted from bi-linear equations during the solution process.

The aim of the method (11) is to find out whether one equation $0 = E_1$ can length reduce another one $0 = E_2$ by replacing E_2 through an appropriate linear combination $\alpha E_1 - \beta E_2, \quad \beta \neq$

0. To find α, β one can divide each term of E_2 through each term of E_1 and count how often each quotient occurs. If a quotient α/β occurs m times then $\alpha E_1 - \beta E_2$ will have $\leq n_1 + n_2 - 2m$ terms because $2m$ terms will cancel each other. A length reduction is found if $n_1 + n_2 - 2m \leq \max(n_1, n_2)$. The method becomes efficient after a few algorithmic refinements discussed in [21]. Length reduced equations

- are more likely to length reduce other equations,
- are much more likely to be factorizable,
- are more suited for substitutions as the substitution induces less growth of the whole systems and introduces fewer new occurences of functions in equations,
- are more likely to be integrable by being exact or being an ODE if the system consists of differential equations,
- involve on average fewer unknowns and make the whole system more sparse. This sparseness can be used to plan better a sequence of eliminations.

This concludes the listing of modules. Other aspects of CRACK follow.

4 Features

4.1 Flexible Process Control

Different types of over-determined systems are more or less suited for an automatic solution. With the currrent version (2002) it is relatively save to try solving large bi-linear algebraic problems automatically. Another well suited area concerns over-determined systems of linear PDEs. In contrast, non-linear systems of PDEs most likely require a more tight interactive control. Different modes of operation are possible. One can

- perform one $\{a\}$ or more computational steps $\{g\}$ automatically, where each step is trying modules in the order defined by the current priority list $\{p1\}$ until one module succeeds in its purpose,
- perform one module a specific number of times or as long as it is successful, $\{l\}$
- set a time limit until which the program should run automatically, {*time_limit, limit_time*}
- interrupt an on-going automatic computation and continue the computation interactively by copying the file `_stop_crack` into the directory where the ongoing computation was started and re-naming it `_stop_` (and by deleting `_stop_` to resume automatic computation),
- arrange the priority list of modules changes at a certain point in the computation when the system of equations has changed its character,
- induce a case distinction whether a user-given expression is zero or not, {*44*}
- have a module that changes the priority list *proc_list_* dynamically, depending essentially on the size and difficulty of the system but also on the success rate of previous steps. {*61, 62, 63*}

Apart from flexible control over what kind of steps to be done, the steps themselves can be controlled more or less too, e.g. whether equations are selected by the module or the user.

Highest priority in the priority list have so-called *to-do* steps. The list of to-do steps is usually empty but can be filled by any successful step if it requires another specific step to follow instantly. For example, if a very simple equation $0 = f_x$ is integrated then the substitution of f should follow straight away, even if substitutions would have a low priority according to the current priority list.

4.2 Total Data Control

To make wise decisions of how to continue the computation in an interactive session one needs tools to inspect large systems of equations. Helpful commands in CRACK print

- equations, inequalities, functions and variables, {*e, pi, f*}
- the occurence of all derivatives of selected functions in any equation, {v}
- a statistics summary of the equations of the system, {s}
- a matrix display of occurences of unknowns in all equations, {pd}
- the value of any LISP variable, {pv}
- the value of algebraic expressions that can be specified using equation names (e.g. `coeffn(e_5,df(f,x,y),2)`), {pe}
- not under-determined subsystems. {ss}

Inspecting a computation which already goes on for hours or a day and that has performed many thousand steps is time consuming. The task is made easier with the possibility to plot graphically as a function of time: the type of steps performed, the number of unknowns, the number of remainig equations, the number of terms in these equations and the memory usage. {ps}

When non-linear systems are considered and many case distinctions and sub-(sub-...)case distinctions are made in a long computation then one easily loses track. With one command one can list all cases that have been considered so far with their assumption, the number of steps made until they are solved or until the next sub-case distinction was made and the number of solutions contained in each completed case. {ls}

4.3 Safety

When working on large problems it may come to a stage where computational steps are necessary, like substitution, which are risky in the sense that they may simplify the problem but also complicate it by increasing its size. To avoid this risk a few safety features have been implemented.

- At any time during the computation one can save a backup of the complete current situation in a file and also load a backup. The command `sb "file_name"` saves all global variables and data into an ASCII file and the command `rb "file_name"` reads these data from a file. The format is independent of the computer used and independent of the underlying LISP version. Apart from reading in a backup file during an interactive computation with *rb* one also can start a computation with a backup file. After loading CRACK one makes in REDUCE the call `CRACKSHELL()$` followed by the file name of the backup.

- All key strokes are automatically recorded in a list which is available after each interactive step with *ph*, or when the computation has finished through `lisp reverse history_;`. This list can be fed into CRACK at the beginning of a new computation so that the same operations are performed automatically that were performed interactively before. The purpose is to be able to do an interactive exploration first and to repeat it afterwards automatically without having to note with pen or pencil all steps that had been done.

  By assigning this list to the LISP variable `old_history` before calling `Crack` with `off batch_mode` the same steps as in the previous run are performed first as CRACK is first reading input from `old_history` and only if that is `nil` then read from the keyboard.

- During an automatic computation the program might start a computational step which turns out to take far too long. It would be better to stop this computation and try something else instead. But in computer algebra with lots of global variables involved it is not straight forward to stop a computation in the middle of it. If one would use time as a criterion then it could happen that time is up during a garbage collection which to stop would be deadly for the session. CRACK allows to set a limit of garbage collections for any one of those computations that have the potential to last forever, like algebraic factorizations of large expressions. With such an arrangement an automatic computation can not get stuck anymore due to lengthy factorizations, searches for length reductions or elimination steps. {*max_gc_elimin, max_gc_fac, max_gc_red_len, max_gc_short, max_gc_ss*}

- Due to a recent (April 2002) initiative of Winfried Neun the parallel version of the computer algebra system REDUCE has been re-activated and is running on the Beowulf cluster at Brock University [20]. This allows conveniently (with {*pp*}) to duplicate the current status of a CRACK computation to another computer, to try out there different operations (e.g. risky ones) until a viable way to continue the computation is found without endangering the original session.

4.4 Managing Solutions

Non-linear problems can have many solutions. The number of solutions found by CRACK can even be higher because to make progress CRACK may have factorized an equation and considered the two cases $a = 0$ and $a \neq 0$ whereas solutions in both cases could be merged to only one solution without any restriction for a. This merging of solutions can be accomplished with a separate program `merge_sol()` after the computation.

Another form of post-processing is the production of a web page for each solution, like http://lie.math.brocku.ca/twolf/bl/v/v1l05o35-s1.html

If in the solution of over-determined differential equations the program performs integrations of equations before the differential Gröbner basis was computed then in the final solution there may be redundant constants or functions of integration. Redundant constants or functions in a solution are not an error but they make solutions appear unnecessarily complicated. In a postprocessing step these functions and constants can be eliminated. {*adjust_fnc, drop_const(), dropredundant()*}

4.5 Parallelization

The availability of a parallel version of CRACK was mentioned above allowing to try out different ways to continue an ongoing computation. A different possibility to make use of a cluster of computers with CRACK is, to export automatically the investigation of sub-cases and sub-sub-cases to different computers to be solved in parallel.

It was explained above how factorizations may be necessary to make any progress but also their potential of exploding the time requirements. By running the computation on a cluster and being able to solve many more cases one can give factorizations a higher priority and capitalize on the benefit of factorizations, i.e. the simplification of the problem.

4.6 Relationship to Gröbner Basis Algorithms

For systems of equations in which the unknown constants or functions turn up only polynomially a well known method is able to check the consistency of the system. For algebraic systems this is the Gröbner Basis Method and for systems of differential equations this is the differential Gröbner Basis method. To guarantee the method to terminate a total ordering of unknowns and their derivatives has to be introduced. This ordering determines which highest powers of unknowns are to be eliminated next or which highest order derivatives have to be eliminated next using integrability conditions. Often such eliminations lead to exponential growth of the generated equations. In the package CRACK such computations are executed with only a low priority. Operations have a higher priority which reduce the length of equations, irrespective of any orderings. Violating any ordering a finite number of times still guarantees a finite algorithm. The potential gain is large as described next.

4.7 Exploiting Bi-linearity

In bi-linear algebraic problems we have 2 sets of variables $a_1, .., a_m$ and $b_1, .., b_n$ such that all equations have the form $0 = \sum_{k=1}^{l} \gamma_k a_{i_k} b_{j_k},\ \gamma_k \in G$. Although the problem is linear in the a_i and linear in the b_j it still is a non-linear problem. A guideline which helps keeping the structure of the system during computation relatively simple is to preserve the linearity of either the a_i or the b_j as long as possible. In classification problems of integrable systems the ansatz for the symmetry/first integral usually involves more terms and therefore more constants (called b_j in applications of CRACK) than the ansatz for the integrable system (with constants a_i). A good strategy therefore is to keep the system linear in the b_j during the computation, i.e. to

- substitute only a b_j in terms of a_i, b_k, or an a_i in terms of an a_k but not an a_i in terms of any b_k,
- do elimination steps for any b_j or for an a_i if the involved equations do not contain any b_k.

The proposed measures are effective not only for algebraic problems but for ODEs/PDEs too (i.e. to preserve linearity of a sub-set of functions as long as possible). {*flin_*}

4.8 Occurrence of sin, cos or other special functions

If the equations to be solved involve special functions, like sin and cos then one is inclined to add `let`-rules for simplifying expressions. Before doing this the simplification rules at the end of the file `crinit.red` should be inspected such that new rules do not lead to cycles with existing rules. One possibility is to replace existing rules, for example to substitute the existing rule

```
trig1_:={sin(~x)**2 => 1-cos(x)**2}$
```

by the new rule

```
trig1_:={cos(~x)**2 => 1-sin(x)**2}$ .
```

These rules are switched off when integrations are performed in order not to interfere with the REDUCE Integrator.

Apart from an initial customization of let-rules to be used during the whole run one can also specify and clear let-rules during a computation using the interactive commands `lr,cr`.

4.9 Exchanging time for memory

The optimal order of applying different methods to the equations of a system is not fixed. It does depend, for example, on the distributions of unknown functions in the equations and on what the individual method would produce in the next step. For example, it is possible that the decoupling module which applies integrability conditions through cross differentiations of equations is going well up to a stage when it suddenly produces huge equations. They not only occupy much memory, they also are slow to handle. Right *before* this explosion started other methods should have been tried (shortening of equations, any integrations, solution of underdetermined ODEs if there are any,...). These alternative methods are normally comparatively slow or unfavourable as they introduce new functions but under the current circumstances they may be perfect to avoid any growth and to complete the calculation. How could one have known beforehand that some method will lead to an explosion? One does not know. But one can regularly make a backup with the interactive `sb` command and restart at this situation if necessary.

4.10 Customization

The addition of new modules to perform new specialized computations is easy. Only the input and output of any new module are fixed. The input consists of the system of equations, the list of inequalities and the list of unknowns to be computed. The output includes the new system of equations and new intermediate results. The module name has to be added to a list of all modules and a one line description has to be added to a list of descriptions. This makes it easy for users to add special techniques for the solution of systems with extra structure. A dummy template module $\{58\}$ is already added and has only to be filled with content.

4.11 Debugging

A feature, useful mainly for debugging is that in the middle of an ongoing interactive computation the program can be changed by loading a different version of CRACK procedures. Thus one could advance quickly close to the point in the execution where an error occurs, load a version of the faulty procedure that gives extensive output and watch how the fault happens before fixing it.

The possibility to interrupt REDUCE itself temporarily and to inspect the underlying LISP environment $\{br\}$ or to execute LISP commands and to continue with the CRACK session afterwards $\{pc\}$ led to a few improvements and fixes in REDUCE itself.

5 Technical issues

5.1 System requirements

The required system is REDUCE, version 3.6. or later. The PSL version is faster whereas the CSL version seems to be more stable under WINDOWS. Also it provides a portable compiled code.

Memory requirements depend crucially on the application. The `crack.rlg` file is produced from running `crack.tst` in a 4MB session running REDUCE, version 3.7 under LINUX. On the other hand it is not difficult to formulate problems that consume any amount of memory.

5.2 Availability

The package CRACK together with LIEPDE, CONLAW and APPLYSYM are included with REDUCE.

Publications related to CRACK itself and to applications based on it can be found under `http://lie.math.brocku.ca/twolf/home/publications.html#1`.

5.3 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` this manual.

6 Reference

6.1 Elements of proc_list_

The interactive command *p1* shows *proc_list_*. This list defines the order in which procedures are tried if a step is to be performed automatically. *p2* shows the complete list as it is shown below. To select any one procedure of the complete list interactively, one simply inputs the number shown in (). The numbering of procedures grew historically. Each number has only little or no connection with the priority of the procedure it is labeling.

`to_do (1):` hot list of steps to be taken next, should always come first,

`subst_level_? (3-6,15-21):` substitutions of functions by expressions, substitutions differ by their maximal allowed size and other properties, to find out which function has which properties one currently has to inspect the procedure definitions of `subst_level_?` in the file `crmain.red`.

`separation (2):` what is described as direct separation in the next section,

`gen_separation (26):` what is described as indirect separation in the next section, only to be used for linear problems,

`quick_gen_separation (10):` generalized separation of equations with an upper size limit,

`quick_integration (7):` integration of very specific short equations,

`full_integration (24):` integration of equations which lead to a substitution,

`integration (25):` any integration,

`factorize_to_substitute (8):` splitting the computation into the investigation of different subcases resulting from the algebraic factorization of an equation, only useful for non-linear problems, and applied only if each one of the factors, when individually set to zero, would enable the substitution of a function.

`factorize_any (47)`: splitting into sub-cases based on a factorization even if not all factors set to zero lead to substitutions.

`change_proc_list (37)`: reserved name of a procedure to be written by the user that does nothing else but changing `proc_list_` in a fixed manner. This is to be used if the computation splits naturally into different parts and if it is clear from the beginning what the computational methods (`proc_list_`) have to be.

`stop_batch (38)`: If the first steps to simplify or partially solve a system of equations are known and should be done automatically and afterwards CRACK should switch into interactive mode then `stop_batch` is added to `proc_list` with a priority just below the steps to be done automatically.

`drop_lin_dep (12)`: module to support solving big linear systems (still experimental),

`find_1_term_eqn (13)`: module to support solving big linear systems (still experimental),

`trian_lin_alg (14)`: module to support solving big linear systems (still experimental),

`undetlinode (22)`: parametric solution of single under determined linear ODE (with non-constant coefficients), only applicable for linear problems (Too many redundant functions resulting from integrations may prevent further integrations. If they are involved in single ODEs then the parametric solution of such ODEs treated as single underdetermined equations is useful. Danger: new generated equations become very big if the minimal order of any function in the ODE is high.),

`undetlinpde (23)`: parametric solution of single under determined linear PDE (with non-constant coefficients), only applicable for linear problems (still experimental),

`alg_length_reduction (11)`: length reduction by algebraic combination, only for linear problems, one has to be careful when combining it with decoupling as infinite loops may occur when shortening and lowering order reverse each other,

`diff_length_reduction (27)`: length reduction by differential reduction,

`decoupling (30)`: steps towards the computation of a differential Gröbner Basis,

`add_differentiated_pdes (31)`: only useful for non-linear differential equations with leading derivative occuring non-linearly,

`add_diff_ise (32)`: for the treatment of non-linear indirectly separable equations,

`multintfac (33)`: to find integrating factors for a system of equations, should have very slow priority if used at all,

`alg_solve_single (34)`: to be used for equations quadratic in the leading derivative,

`alg_solve_system (35)`: to be used if a (sub-)system of equations shall be solved for a set of functions or their derivatives algebraically,

`subst_derivative (9)`: substitution of a derivative of a function everywhere by a new function if such a derivative exists

`undo_subst_derivative (36)`: undo the above substitution.

`del_redundant_fc (40)`: Drop redundant functions and constants. For that an overdetermined PDE - system is formulated and solved to set redundant constants / functions of integration to zero. This may take longer if many functions occur.

`find_trafo (39)`: finding a first order linear PDE, by solving it the program finds a variable transformation that transforms the PDE to a single derivative and makes the PDE integrable for the integration modules. Because a variable transformation was performed the solution contains only new functions of integration which depend on single (new) variables and not on expressions of them, like sums of them. Therefore the result of the integration can be used for substitutions in other equations. if the transformation would not have been made then the solution of the PDE would involve arbitrary functions of expressions and could not be used for the other equations using the current modules of CRACK. A general transformation can be done interactively with the command *cp*.

`sub_problem (41)`: Solve a subset of equations first (still experimental),

`del_redundant_de (28)`: Delete redundant equations,

`idty_integration (29)`: Integrate an identity

`gen_separation2 (48)`: Indirect separation of a pde, this is a 2nd version for non-linear PDEs

`find_and_use_sub_systems12 (49)`: Find sub-systems of equations with at least as many equations as functions, in this case find systems with at most 2 functions, none of them a function of the set `flin_` (these are functions which occur initially only linearly in a non-linear problem, `flin_` is assigned initially by the user).

`find_and_use_sub_systems13 (50)`: like above only with at most 3 functions, none from `flin_`

`find_and_use_sub_systems14 (51)`: like above only with at most 4 functions, none from `flin_`

`find_and_use_sub_systems15 (52)`: like above only with at most 5 functions, none from `flin_`

`find_and_use_sub_systems22 (53)`: like above only with at most 2 functions, only `flin_` are considered, all others ignored

`find_and_use_sub_systems23 (54)`: like above only with at most 3 functions, only `flin_` are considered, all others ignored

`find_and_use_sub_systems24 (55)`: like above only with at most 4 functions, only `flin_` are considered, all others ignored

`find_and_use_sub_systems25 (56)`: like above only with at most 5 functions, only `flin_` are considered, all others ignored

`high_prio_decoupling (57)`: do a decoupling step with two equations that in total involve at most 3 different functions of all independent variables in these equations

`user_defined (58)`: This is an empty procedure which can be filled by the user with a very specific computational step that is needed in a special user application. Template:

```
symbolic procedure user_defined(arglist)$
begin % arglist is a lisp list {pdes,forg,vl_} where
      % pdes is the list of names of all equations
      % forg is the list of original functions + their values
      %      as far as known
      % vl_ the list of independent variables
 ...
 return if successful then list(pdes,forg)
                           % new pdes + functions and their value
                      else nil
end$
```

`alg_groebner (59)`: call of the Reduce procedure `groebnerf` trying to solve the whole system under the assumption that it is a completely algebraic polynomial system. All resulting solutions are considered individually further.

`solution_check (60)`: This procedure tests whether a solution that is defined in an external procedure sol_define() is still contained in the general solution of the system currently under investigation. This procedure is useful to find the place in a long computation where a special solution is either lost or added to the general solution of the system to be solved. Template:

```
algebraic procedure sol_define$
<< % This procedure contains the statements that specify a solution
  %Example: Test whether s=h_-y**2/t**2,  u=y/t is a solution
  %         where h_=h_(t)
  depend h_,t$
  % returned is a list of expressions that vanish for the solution
  % to be tested, in this example:
  {s-(h_-y**2/t**2),u-y/t}
>>$
```

6.2 Online help

The following commands and their one line descriptions appear in the same order as in the online help.

6.2.1 Help to help

hd Help to inspect data
hp Help to proceed
hf Help to change flags & parameters
hc Help to change data of equations
hi Help to work with identities
hb Help to trace and debug

6.2.2 Help to inspect data

e Print equations
eo Print overview of functions in equations
pi Print inequalities
f Print functions and variables
v Print all derivatives of all functions
s Print statistics
fc Print no of free cells
pe Print an algebraic expression
ph Print history of interactive input
pv Print value of any lisp variable
pf Print no of occurences of each function
pr Print active substitution rules
pd Plot the occurence of functions in equations
ps Plot a statistical history
lc List all case distinctions
ws Write statistical history in file
sn Show name of session
ss Find and print sub-systems
w Write equations into a file

6.2.3 Help to proceed

a Do one step automatically
g Go on for a number of steps automatically
t Toggle between automatic and user selection of equations (`expert_mode=nil/t`).
p1 Print a list of all modules in batch mode
p2 Print a complete list of all modules
Execute the module with the number '#' once
l Execute a specific module repeatedly
sb Save complete backup to file
rb Read backup from file
ep Enable parallelism
dp Disable parallelism
pp Start an identical parallel process
kp Kill a parallel process
x Exit interactive mode for good
q Quit current level or crack if in level 0

6.2.4 Help to change flags & parameters

pl Maximal length of an expression to be printed
pm Toggle to print more or less information about pdes (`print_more`)
pa Toggle to print all or not all information about the pdes (`print_all`)
cp Change the priorities of procedures
og Toggle ordering between 'lexicographical ordering of functions having a higher priority than any ordering of derivatives' and the opposite (`lex_fc=t`) resp. (`lex_fc=nil`)
od Toggle ordering between 'the total order of derivatives having a higher priority than lexicographical ordering' (`lex_df=nil`) or not (`lex_df=t`)

oi Interactive change of ordering on variables
or Reverse ordering on variables
om Mix randomly ordering on variables
of Interactive change of ordering on functions
op Print current ordering
ne Root of the name of new generated equations (default: e_)
nf Root of the name of new functions and constants (default: c_)
ni Root of the name of new identities (default: id_)
na Toggle for the NAT output switch (`!*nat`)
as Input of an assignment
kp Toggle for keeping a partitioned copy of each equation (`keep_parti`)
fi Toggle for allowing or not allowing integrations of equations which involve unresolved integrals (`freeint_`)
fa Toggle for allowing or not allowing solutions of ODEs involving the `abs` function (`freeabs_`)
cs Switch on/off the confirmation of intended substitutions and of the order of the investigation of subcases resulting in a factorization
fs Enforce direct separation
ll change of the line length
re Toggle for allowing to re-cycle equation names (`do_recycle_eqn`)
rf Toggle for allowing to re-cycle function names (`do_recycle_fnc`)
st Setting a CPU time limit for un-interrupted run
cm Adding a comment to the history_ list
lr Adding a LET-rule
cr Clearing a LET-rule

6.2.5 Help to change data of equations

r Replace or add one equation
rd Reduce an equation modulo LET rules
n Replace one inequality
de Delete one equation
di Delete one inequality
c Change a flag or property of one pde
pt Perform a transformation of functions and variables

6.2.6 Help to work with identities

i Print identities between equations
id Delete redundand equations
iw Write identities to a file
ir Remove list of identities
ia Add or replace an identity
ih Start recording histories and identities
ip Stop recording histories and identities
ii Integrate an identity
ic Check the consistency of identity data
iy Print the history of equations

6.2.7 Help to trace and debug

tm Toggle for tracing the main procedure (`tr_main`)
tg Toggle for tracing the generalized separation (`tr_gensep`)
ti Toggle for tracing the generalized integration (`tr_genint`)
td Toggle for tracing the decoupling process (`tr_decouple`)
tl Toggle for tracing the decoupling length reduction process (`tr_redlength`)
ts Toggle for tracing the algebraic length reduction process (`tr_short`)
to Toggle for tracing the ordering procedures process (`tr_orderings`)
tr Trace an arbitrary procedure
ut Untrace a procedure
br Lisp break
pc Do a function call
in Reading in a REDUCE file

6.3 Global variables

The following is a complete list of identifiers used as global lisp variables (to be precise symbolic fluid variables) within CRACK. Some are flags and parameters, others are glaboal variables, some of them can be accessed after the CRACK run.

`!*allowdfint_bak !*dfprint_bak !*exp_bak !*ezgcd_bak !*fullroots_bak !*gcd_bak !*mcd_bak !*nopowers_bak !*ratarg_bak !*rational_bak !*batch_mode abs_ adjust_fnc allflags_ batchcount_ backup_ collect_sol confirm_subst cont_ contradiction_ cost_limit5 current_dir default_proc_list_ do_recycle_eqn do_recycle_fnc done_trafo eqname_ expert_mode explog_ facint_ flin_ force_sep fname_ fnew_ freeabs_ freeint_ ftem_ full_proc_list_ gcfree!* genint_ glob_var global_list_integer global_list_ninteger global_list_number high_gensep homogen_ history_ idname_ idnties_ independence_ ineq_ inter_divint keep_parti last_steps length_inc level_ lex_df lex_fc limit_time lin_problem lin_test_const logoprint_ low_gensep max_gc_counter max_gc_elimin max_gc_fac max_gc_red_len max_gc_short max_gc_ss max_red_len maxalgsys_ mem_eff my_gc_counter nequ_ new_gensep nfct_ nid_ odesolve_ old_history orderings_ target_limit_0 target_limit_1 target_limit_2 target_limit_3 target_limit_4 poly_only potint_ print_ print_all print_more proc_list_ prop_list pvm_able quick_decoup record_hist recycle_eqns recycle_fcts recycle_ids reducefunctions_ repeat_mode safeint_ session_ simple_orderings size_hist size_watch sol_list solvealg_ stepcounter_ stop_ struc_dim struc_eqn subst_0 subst_1 subst_2 subst_3 subst_4 time_ time_limit to_do_list tr_decouple tr_genint tr_gensep tr_main tr_orderings tr_redlength tr_short trig1_ trig2_ trig3_ trig4_ trig5_ trig6_ trig7_ trig8_ userrules_ vl_`

6.4 Global flags and parameters

The list below gives a selection of flags and global parameters that are available, for example, to fine tune the performance according to specific needs of the system of equations that is studied. Usually they are not needed and very few are used regularly by the author. The interactive command that changes the flag/parameter is given in [ ], default values of the flags/parameters are given in (). All values can be changed interactively with the *as* command. The values of

the flags and parameters can either be set after loading CRACK and before starting CRACK with a lisp assignment, for example,
`lisp(print_:=8)$`
or after starting CRACK in interactive mode with specific commands, like `pl` to change specifically the print length determining parameter `print_`, or the command `as` to do an assignment. The values of parameters/flags can be inspected interactively using `pv` and changed with `as`.

`!*batch_mode [x] (t)` : running crack in interactive mode (`!*batch_mode=nil`) or automaticly (`!*batch_mode=t`). It can also be set in algebraic mode before starting CRACK by `ON/OFF BATCH_MODE`. Interactive mode can be left and automatic computation be started by the interactive commant `x`.

`!*iconic (nil)` : whether new processes in parallelization should appear as icons (t) or windows (nil)

`adjust_fnc (nil)` : if t then free constants/functions are scaled and redundant ones are dropped to simplify the result after the computation has been completed

`collect_sol (t)` : whether solutions found shall be collected and returned together at the end or not (to save memory), matters only for non-linear problems with very many special solutions. If a computation has to be performed with any solution that is found, then these commands can be put into an `algebraic procedure crack_out(eqns, assigns, freef, ineq)` which is currently empty in file `crmain.red` but which is called for each solution.

`confirm_subst [cs] (nil)` : whether substitutions have to be confirmed interactively

`cont_ (nil)` : interactive user control for integration or substitution of large expressions (enabled = t)

`cost_limit5 (100)` : maximal number of extra terms generated by a subst.

`do_recycle (nil)` : whether function names shall be recycled or not (saves memory but computation is less clear to follow)

`done_trafo (nil)` : an (algebraic mode) list of backtransformations that would invert done transformations, this list is useful after CRACK completed to invert transformations if needed

`eqname_ [ne] ('e_)` : name of new equations

`expert_mode [t] (nil)` : For `expert_mode=t` the equations that are involved in the next computational step are selected by CRACK, for `expert_mode=nil` the user is asked to select one or two equations which are to be worked with in the next computational step.

`facint_ (1000)` : if equal nil then no search for integrating factors otherwise equal the max product terms*kernels for searching an integrating factor

`flin_ (nil)` : a list of functions occuring only linearly in an otherwise non-linear problem. `flin_` has to be assigned before calling CRACK. During execution it is tried to preserve the linearity of these functions as long as possible.

`fname_ [nf] ('c_)` : name of new functions and constants (integration)

`force_sep (nil)` : whether direct separation should be forced even if functions occur in the supposed to be linear independent explicit expressions (for non-lin. prob.)

`freeabs_ [fi] (t)` : Do not use solutions of ODEs that involve the `abs` function

`freeint_ [fi] (t)` : Do only integrations if expl. part is integrable

`genint_ (15)` : if =nil then generalized integration disabled else equal the maximal number of new functions and extra equations due to the generalized integration of one equation

`high_gensep (300)` : min. size of expressions to separate in a generalized way by 'quick_gen_separation'

`homogen_ (nil)` : Test for homogeneity of each equation (for debugging)

`idname_ [ni] ('id_)` : name of new equations

`idnties_ (nil)` : list of identities resulting from reductions and integrability conditions

`independence_ (nil)` : interactive control of linear independence (enabled = t)

`inter_divint (nil)` : whether the integration of divergence identities with more than 2 differentiation variables shall be confirmed interactively

`keep_parti [kp] (nil)` : whether for each equation a copy in partitioned form is to be stored to speed up several simplifications but which needs more memory

`last_steps (nil)` : a list of the last steps generated and updated automatically in order to avoid cycles

`length_inc (1.0)` : factor by which the length of an expression may grow when performing `diff_length_reduction`

`lex_df [od] (nil)` : if t then use lexicographical instead of total degree ordering of derivatives

`lex_fc [og] (t)` : if t then lexicographical ordering of functions has higher priority than any ordering of derivatives

`limit_time (nil)` : = time() + how many more seconds allowed in batch mode

`logoprint_ (t)` : print logo after crack call

`low_gensep (6)` : max. size of expressions to be separated in a generalized way by 'quick_gen_separation'

`max_gc_counter (100000000)` : maximal total number of garbage collections

`max_gc_elimin (15)` : maximal number of garbage collections during elimination in decoupling

`max_gc_fac (15)` : maximal number of garbage collections during factorization

`max_gc_red_len (30)` : maximal number of garbage collections during length reduction

`max_gc_short (40)` : maximal number of garbage collections during shortening

`max_gc_ss (10)` : maximal number of garbage collections during search of sub_systems

`max_red_len (1000000)` : maximal product of lengths of two equations to be combined with length-reducing decoupling

`maxalgsys_ (20)` : max. number of equations to be solved in specialsol

`mem_eff (t)` : whether to be memory efficient even if slower

`my_gc_counter (0)` : initial value of my_gc_counter

`nequ_ (1)` : index of the next new equation

`new_gensep (nil)` : whether or not a newer (experimental) form of gensep should be used

`nfct_ (1)` : index of the next new function or constant

`nid_ (1)` : index of the next new identity

`odesolve_ (100)` : maximal length of a de (number of terms) to be integrated as ode

`old_history (nil)` : old_history is interactive input to be read from this list

`poly_only (nil)` : all equations are polynomials only

`potint_ (t)` : allowing 'potential integration'

`print_ [pl] (12)` : maximal length of an expression to be printed

`print_all [pa] (nil)` : Print all informations about the pdes

`print_more [pm] (t)` : Print more informations about the pdes

`quick_decoup (nil)` : whether decoupling should be done faster with less care for saving memory

`record_hist (nil)` : whether the history of equations is to be recorded

`safeint_ (t)` : uses only solutions of ODEs with non-vanishing denominator

`session_ (``bu''+random number+date)` : when loading CRACK or executing

`size_watch (nil)` : whether before each computational step the size of the system shall be recorded in the global variable size_hist

`solvealg_ (nil)` : Use SOLVE for algebraic equations

`struc_eqn (nil)` : whether the equations has the form of structural equations (an application are the Killing vector and Killing tensor computations)

`subst_*` : maximal length of an expression to be substituted, used with different values for different procedures `subst_level_*`

`target_limit_* (nil)` : maximum of product *length(pde)*length(substituted expression)* for a PDE which is to be used for a substitution, If `target_limit_* = nil` then no length limit, used with different values for different procedures `subst_level_*`

`time_ (nil)` : print the time needed for running crack

`time_limit (nil)` : whether a time limit is active after which batch-mode is interrupted to interactive mode

`tr_decouple [td] (nil)` : Trace decoupling process

`tr_genint [ti] (nil)` : Trace generalized integration

`tr_gensep [ts] (nil)` : Trace generalized separation

`tr_main [tm] (nil)` : Trace main procedure

`tr_orderings [to] (nil)` : Trace orderings stuff

`tr_redlength [tr] (nil)` : Trace length reduction

7 A more detailed description of some of the modules

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.

7.1 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 of both equations 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 [9, 5, 10, 6, 7, 8] and [11].

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`.

7.2 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 [22].

Stop:

Iteration stops if no term with any x^k-derivative of any f^i is left. If any $f^i(x^k)$ occurs in the remaining un-integrated terms 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 or not 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 three functionally independent expressions with respect 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 sums such that each of them 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

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

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

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

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 $x^i, 1 \le i \le n$, then a generalized version of the above algorithm can still provide a formal expression for the integral I (see [22]). The price consists of additional differential conditions, but they are equations in fewer variables than occur in the integrated equation. Integrating for example

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

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

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

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

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

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 nonlinearly increasing 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 fewer 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 [22].

7.3 Direct separation of PDEs

As a result of repeated integrations the functions in the remaining equations have fewer and fewer 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

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

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.

7.4 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. $z_{11} = y$:

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

    - 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:

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

- 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

$$\begin{aligned} y^1 : 0 &= f/c_1 - 1 - x^2 \quad &\Rightarrow \quad f' = 2c_1 x \\ y^0 : 0 &= c_2 f - \tfrac{1}{2}xf' - 1/c_1 \quad &\Rightarrow \quad 0 = c_2 c_1 (1+x^2) - c_1 x^2 - 1/c_1 \end{aligned}$$

  and direct separation w.r.t. x:

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

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},{});
```

7.5 Solving standard ODEs

For solving standard ODEs the package ODESolve by Malcalm MacCallum and Francis Wright [15] 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 $f_{,xxy}$.

Acknowledgement

Andreas Brand is the author of a number of core modules of Crack. The currently used data structure and program structure of the kernel of Crack are due to him. He contributed to the development of Crack until 1997.

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 contributed in supporting 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 Institute / Berlin through a fellowship of T.W.. Winfried Neun and Herbert Melenk are thanked for many discussions and constant support. Many of the low level control features have been provided by Winfried Neun. He ported Parallel Reduce to a Linux PC Beowulf cluster and helped in adapting Crack to it. Last not least he helped in enabling to encode PDF features of this document in LaTeX.

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] G. J. Reid, A. D. Wittkopf and A. Boulton, Reduction of systems of nonlinear partial differential equations to simplified involutive forms, European Journal of Applied Mathematics, Vol 7. (1996) 604-635.

[8] G. J. Reid, A. D. Wittkopf and P. Lin, Differential-Elimination Completion Algorithms for Differential Algebraic Equations and Partial Differential Algebraic Equations, to appear in Studies in Applied Mathematics (Submitted July 1995).

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

[10] 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.

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

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

[13] 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).

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

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

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

[17] 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)

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

[19] 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)

[20] Melenk, H., Neun, W., RR: Parallel Symbolic Algorithm Support for PSL Based REDUCE, ZIP preprint (2002).
`www.zib.de/Optimization/Software/Reduce/moredocs/parallel_reduce.ps`

[21] Wolf, T., Size reduction and partial decoupling of systems of equations, J. Symb. Comp. 33, no 3 (2002) 367-383.

[22] Wolf, T., The Symbolic Integration of Exact PDEs, J. Symb. Comp. **30** (No. 5) (2000), 619-629.

[23] Wolf, T., The integration of systems of linear PDEs using conservation laws of syzygies, J. of Symb. Comp. **35**, no 5 (2003) 499-526.

[24] Sokolov, V.V., Wolf, T., Classification of integrable polynomial vector evolution equations, J. Phys. A: Math. Gen. **34** (2001) 11139-11148.

[25] Euler, N., Wolf, T., Leach, PGL and Euler, M.: Linearizable Third Order ODEs and Generalised Sundman Transformations: The Case X"'=0, Acta Applicandae Mathematicae, Volume 76, (2003), Issue 1, 89-115.

[26] Wolf, T., Efimovskaya, O. V., Classification of integrable quadratic Hamiltonians on e(3), preprint, 11 pages (2003).