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

March 20, 2004

Contents

1 Online help

1.1 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

1.2 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
pd Plot the occurence of functions in equations
ss Find and print sub-systems
w Write equations into a file

1.3 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

1.4 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

1.5 Help to change data of equations

r Replace or add one equation
n Replace one inequality
d Delete one equation
c Change a flag or property of one pde

1.6 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

1.7 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

2 The purpose of Crack

The package CRACK attempts the solution of an overdetermined system of algebraic or 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. Applying the package CRACK to such problems directly may be successful, especially if these are ODEs, but the main type of application is to investigate qualitative properties of such DEs/systems of DEs and to solve the overdetermined PDE-systems that result in these investigations.

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.

3 Technical details

3.1 System requirements

The required system is REDUCE, version 3.6. or 3.7. (either the PSL version of REDUCE as distributed by the Konrad Zuse Institut / Berlin or the CSL version of REDUCE as distributed by CODEMIST Ltd). 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.

3.2 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$
```

3.3 Updates / web demos

The latest version of CRACK and related programs is available from `http://lie.math.brocku.ca/twolf/crack/`. Publications related to CRACK can be found under `http://lie.math.brocku.ca/twolf/home/publications.html#1`.

3.4 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.

3.5 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`.

3.6 The result

The result is a list of solutions

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

where each solution is a list of 4 lists:

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

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_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.

3.7 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 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.

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

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

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

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

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

`quick_integration` : integration of very specific short equations,

`full_integration` : integration of equations which 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,

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

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

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

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

`alg_length_reduction` : 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` : 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 for a system of equations, should have very slow priority if used at all,

`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.

`del_redundant_fc` : Drop redundant functions and constants. 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.

`point_trafo` : An interactive point transformation not to be used in automatic batch mode,

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

`del_redundant_de` : Delete redundant equations,

`idty_integration` : Integrate an identity (still experimental).

3.8 Performing long computations

3.8.1 The backup facility

If one does a long computation automatically then the computer or the link to it may go down and the computation may have to be started again. Even worse in a longer interactive session which is of an exploring nature, i.e. where every step may blow up the size of expressions or where a step (for example, decoupling, solving a subsystem, searching for a length-reduction, dropping redundant functions,...) may just take too long and where one would want to go back to the situation before this step and try something else. For these situations there is an interactive command for saving a bakup: `sb "file_name"` which saves all global variables + data into an ASCII file and a command `rb "file_name"` which reads these data from a file. The format is independent of the computer used and independent of the underlying LISP version. This has been used by the author to set up long and complex computations on a small

computer and to continue the same interactive session on larger computers later. To continue such a session one calls CRACK without data: `CRACK({},{},{},{})$` and loads the complete environment with `rb "file_name"`.

3.8.2 The history facility

Sometimes one does not only want to store an environment but also how one got there in an interactive session, to repeat the same steps or only some of them in a later session. In the global variable `history_` all interactive input during one call of CRACK is recorded and can be looked at during a CRACK run with the `pv` (print-variable) command `pv history_`. In order to save typing the same input in a later session the program CRACK always tries to read any expected input first from the global variable `old_history`. All that is needed is to do is typing
`lisp reverse history_;`
after a run of CRACK and to assign the result to the LISP variable `old_history`. The next run of CRACK will try to read any expected interactive input first from `old_history` and only if that is `nil` then read it from the keyboard.

3.9 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
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_

3.10 Global flags and parameters

The list below gives a selection of flags and global parameters that are available 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 (). 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`.

`!*batch_mode [x] (t)` : running crack in interactive mode (`!*batch_mode=nil`) or not (`!*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`.

`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.

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

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

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

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

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

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

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

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

`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

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

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

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

`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

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

`max_factor (400)` : maximal number of terms to be factorized

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

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

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

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

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

`max_red_len (50000)` : maximal product of lengths of two equations to be combined with length-reducing decoupling

`target_limit_* (nil)` : maximal product length(pde)*length(substituted expression) for PDEs in which substitutions are to be made, nil ==¿ no length limit, used with different values for different procedures `subst_level_*`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`dec_hist (0)` : length of pde history list during decoupling

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

`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

`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

`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 a procedure `algebraic procedure crack_out(eqns,assigns,freef,ineq)` which is currently empty in file `crmain.red` but which is called for each solution.

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

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

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

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

`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

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

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

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

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

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

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

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

4 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.

4.1 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 [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`.

4.2 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 [14].

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 $f_{,x}\,, ff_{,xxx}\,, f_{,xxy}$ 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

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

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

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

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

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

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 $x^i, 1 \le i \le n$, then a generalized version of the above algorithm can still provide a formal expression for the integral I (see [14]). The price consists of additional differential conditions, but they are equations in less variables than occur in the integrated equation. Integrating for example

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

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

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

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 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 [14].

4.3 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

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

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.

4.4 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:

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

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

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

and direct separation w.r.t. x:

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

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},{});
```

4.5 Solving standard ODEs

For solving standard ODEs the package ODESOLVE by Malcalm MacCallum and Francis Wright [16] 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 $f_{,xxy}$.

5 General hints

5.1 Problems involving $\sin, \cos$ 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 simpli-

fication 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.

5.2 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 methods 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.

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.

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

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

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

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

[18] 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)

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

[20] 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)