The Symbolic Integration of Exact PDEs

Thomas Wolf

School of Mathematical Sciences

Queen Mary and Westfield College

Mile End Road

London E1 4NS

England

September 8, 2000

Abstract

An algorithm is described which decides if a given polynomial differential expression Δ of multivariate functions is exact, i.e. whether there exists a first integral P such that $D_xP = \Delta$ for any one x of a set of n variables and to provide the integral P. A generalization is given to allow integration in the case that the exactness is prevented by terms which contain only functions of less than n independent variables.

1 Motivation

The common way to deal with problems that involve the solution of non-linear differential equations is to try different ansätze which are either geometrically motivated or just chosen to simplify computations. Typical examples are the investigation of infinitesimal symmetries, the search for classes of integrating factors and related first integrals/conservation laws or the search for a variational principle equivalent to a given system of equations. In all these cases over-determined systems of partial differential equations (PDEs) have to be solved. If their solution could be automated then not only would erroneous hand calculations be speeded up but also less over-determined PDE-systems could be solved, i.e. less restrictive ansätze be investigated. One commonly used technique is to apply integrability conditions systematically by computing a characteristic system of the differential ideal [4] or a differential Gröbner basis (or a Pseudo differential Gröbner basis) [5, 6, 7]. Other related programs are listed in [3].

The algorithms to be described in this paper aim to extend such algorithms by enabling the integration of classes of equations either in parallel to the computation of integrability conditions or after a basis for the differential ideal/differential Gröbner basis has been computed. It appears to the author that the problem of adding integrations splits naturally into two parts answering the following questions.

1. What is a good trade off between the generality of the class of equations to be integrated and the practicality of the corresponding integration algorithm? In other words, for which class of equations is an effective and finite algorithm available? At which computational price can such an algorithm be generalized? Are there fast necessary tests available for the integration algorithm to be chosen?

2. How can any integration method of some class of equations be incorporated into the Gröbner basis calculation? What are the conditions which keep the extended Gröbner basis computation finite? With which priority should integrations be performed within such a combined machinery? Which one of a set of integrable equations should be integrated first and if an equation can be integrated, should it be integrated as often as possible? Could the Gröbner basis algorithm be modified to cover the same functionality without having to perform integrations explicitly? Which other techniques, like separation (splitting/fragmenting an equation) become necessary, once integrations are performed? Should any equation which could be integrated also be used for substitutions of unknown functions? Can one give general rules for the order in which the different techniques (computing integrability conditions, integrations, separations) should be applied? Do integrations which integrate an equation if possible at least once and integrations -

which integrate an equation only if it can be integrated often enough to enable a substitution - have different priorities? Which data structures should be used to represent differential equations in order to organize the application of different modules efficiently?

In this paper we are going to answer the first set of questions. To deal with the second set of questions properly one has to discuss no less than a complete package of modules aiming at the explicit solution of over-determined PDE-systems. This will be done elsewhere [10].

In the following section we will outline alternative integration problems before in the subsequent section the algorithm is explained. In the concluding sections extensions are described.

2 Choosing an integration task

There is a wide variety of integration related tasks from which we choose one that is to be addressed by the algorithm described in this paper. Our aim will be to choose a type of problem that is as general as possible but that can still be answered algorithmically and efficiently.

On the complex end of possible problems we have the computation of conservation laws, i.e. relations

$$\begin{aligned} \mathrm{Div}\, P &= D_{x^1}P^1 + \ldots + D_{x^p}P^p && (1)\\ &= \sum_{\nu,J} Q^J_\nu D_J \Delta_\nu \quad \text{(identically in } \textit{all}\ x^i, u^\alpha_J) && (2) \end{aligned}$$

where $x^1, \ldots, x^p$ are the independent variables, $u^1, \ldots, u^q$ are the dependent variables, $0 = \Delta_\nu,\ \nu = 1, \ldots, r$ are the given differential equations, D_{x^i}, D_J are total differential operators with $_J$ being a multi-index, standing, for example, for $_{1223}$, i.e. standing for a derivative with respect to x^1, x^2, x^2, x^3. The unknowns in equation (2) that are to be determined are the components of a conserved current P^i and coefficients Q^J_ν, all being expressions in x^i, u^α_J. We will keep this notation for the rest of the paper.

Partial integration of the right hand side of relation (2) and transferring divergences to the left hand side can always reach a form

$$\mathrm{Div}\, \tilde{P} = \sum_{\nu} \tilde{Q}_{\nu} \Delta_{\nu}. \tag{3}$$

A conservation law could be used to express P by the curl of new potentials that are introduced. In two recent papers [12, 13] it is shown that there are very good chances of solving the resulting over-determined systems by computer algebra but the programs have to perform integrations as part of their solution which is what we want to do in this paper. What one would gain from such an involved computation is an invariant statement. The fact whether or not a divergence $\mathrm{Div}\, P$ is contained in the differential ideal of a given system of equations is not altered by a Gröbner basis computation which does not change the differential ideal. (What may change during a Gröbner basis computation is the order of u-derivatives that the Q_{ν}^{J} involve.)

In order to perform integrations without having to *solve* a system of over-determined equations we will make more restrictive assumptions in this paper. We assume all but one P^i to vanish. Restricting ourselves to the integration of a single equation $0 = \Delta$ we assume the single integrating factor Q to be 1. Further we assume the equations $0 = \Delta_{\nu}$ to be at most polynomially non-linear in the dependent variables u and their derivatives. A few minor generalizations will be added later.

Even with this specification it appears that we are more general than most other programs that are listed in [3] which aim at solving over-determined systems. Two strong programs to be mentioned are Dimsym of James Sherring, Geoff Prince and Michael Jerie and Lie of Alan Head. The integration techniques used in these programs are partially described in [2], [8] and [9] but to the best of the author's knowledge not in the published literature.

Other programs either do not integrate or their integration capabilities are restricted to PDEs of the form

$$0 = \partial_{x^1} u(x^i) - g(x^j)$$

where u is an unknown function and g is a differential expression which involves x^1 only explicitly and polynomially and not as an argument of another unknown function, or their integration algorithm is a translation (see [1]) from the one in the package Crack with contributions from the package Lie. The implementation in Crack is based on the algorithm to be described in this paper. The implementation in [1] does not contain the generalization given in section 4.

The basic form of the algorithms discussed here will be to determine for a given equation $0 = \Delta$ any independent variable x^i and differential expression $P(x^i, u^\alpha)$ which is functionally dependent only on u^α and their partial derivatives such that

$$D_{x^i} P = \Delta.$$

In doing that we lose invariance with respect to steps of a Gröbner Basis computation, as is seen in the following short example. The equation $\partial_x^2 u = 0$ satisfies our integration criterium but adding the two equations $0 = \partial_x \partial_y u - \frac{1}{x}\partial_y u, \;\; 0 = \partial_y^2 u - u$ and computing the Gröbner basis of all three as $0 = x\partial_x u - u, \;\; 0 = \partial_y^2 u - u$ we lose this property.

Although the algorithm to be described is not the most general possible, we will see that it is fast and still frequently applicable. The faster an integration algorithm is, the lower is the risk of wasting computational resources in unsuccessful tries.

3 The algorithm for exact DEs

A skeleton program

In the following we consider integration with respect to only one variable x. In the presence of different variables the computation would have to be repeated for each of them. An algorithm for adding constants/functions of integration will be given further below.

The expression Δ which is to be integrated is assumed to be a polynomial in u^α and their derivatives. In the following algorithm % denotes the beginning of a comment ending at the end of a line, `highpow`(A, B) denotes the highest power of B in the expression A, `coeff`(A, B) denotes the coefficient of B in A and `LD`(Δ, u^α) denotes the leading derivative of the function u^α in the expression Δ with respect to some lexicographic total ordering $>_{lex}$ of variables in which the integration variable x has highest priority. We further denote by u^α_{J-x} a derivative of u^α that has one less $x-$

derivative than u^α_J. The lower multi-index $_K$ is reserved to annotate derivatives u^α_K which do not contain derivatives with respect to the integration variable x.

1 **Algorithm** EXACT
2 **Input** Δ, x % Δ is a differential expression, x is an independent variable
3 **Output** P % with $D_x P = \Delta$, if Δ is exact
4 $\quad$ *nil* % if Δ is not exact
5 **Body**
6 $\quad P := 0$
7 $\quad$ **while** Δ contains an unknown function $u^\alpha(x)$ or its derivatives **do**
8 $\qquad u^\alpha_J := \mathrm{LD}(\Delta, u^\alpha)$
9 $\qquad$ **while** u^α_J involves x-derivatives **do**
10 $\qquad\quad$ **if** highpow$(\Delta, u^\alpha_J) \neq 1$ **then return** *nil* % Δ is not exact
11 $\qquad\quad a := \mathrm{coeff}(\Delta, u^\alpha_J)$
12 $\qquad\quad b := \int a\, d(u^\alpha_{J-x})$
13 $\qquad\quad P := P + b$
14 $\qquad\quad \Delta := \Delta - D_x b$
15 $\qquad\quad u^\alpha_{\tilde{J}} := \mathrm{LD}(\Delta, u^\alpha)$
16 $\qquad\quad$ **if** $u^\alpha_{\tilde{J}} >_{lex} u^\alpha_J$ **then return** *nil* % Δ is not exact
17 $\qquad\quad u^\alpha_J := u^\alpha_{\tilde{J}}$
18 $\qquad$ **if** Δ involves u^α **then return** *nil* % Δ is not exact
19 $\quad P := P + \int \Delta\, dx$ % by now Δ contains x only explicitly
20 $\quad$ **return** P

Scope of applicability

The algorithm is applicable for expressions Δ that are polynomially non-linear in the u^α but also for non-polynomially non-linear Δ provided the integral in line 12 can be computed and the simplification of Δ in line 14 is effective. This means, the expression for Δ computed in line 14 must be free of u_J.

If Δ involves terms that do not involve unknowns $u^\alpha(x)$ but are completely explicit in x then in line 19 an integral of them is added to P. Whether that integral can be expressed in terms of elementary functions or not does depend on these terms but this "integrability" is irrelevant for the exactness of Δ.

The overall structure

The overall structure of the algorithm is to integrate successively highest x-derivatives of any function of x. The highest $x-$ derivatives must occur linearly which is tested in line 10. After integrating all terms that include an x-derivative of some function $u^\alpha(x)$, this function u^α and any derivatives of u^α other than x-derivatives must not occur any more in the so far unintegrated terms Δ (line 18). Otherwise the remaining terms Δ and consequently the original Δ are not exact with respect to x. The reason for this is, an x-integral of the original Δ would have to involve an x-integral of u^α or an x-integral of (non x-) derivatives of u^α which is not what we want as an integral for Δ.

Example: For $g = g(x), \;\; ' = d/dx, \;\; \Delta = gg'^3 + xg'^4 + 3xgg'^2g''$ the computation is completed by going through the inner loop (steps 8-17) only once:

$$\begin{aligned}
u^\alpha_J &:= g'' \\
\mathrm{highpow}(\Delta, g'') &= 1 \\
a &:= 3xgg'^2 \\
u^\alpha_{J-x} &= g' \\
b &:= \int 3xgg'^2\, d(g') = xgg'^3 \\
P &:= xgg'^3 \\
\Delta &:= \Delta - (xgg'^3)' = 0.
\end{aligned}$$

The integral is $P := xgg'^3$. The generation of constants of integration is described further below.

The non-integrability tests

The test in line 16 is motivated with the following simple example. We assume, the single term $\Delta = u_{xxy}u_{xyy}$ with $u = u(x,y)$ is to be x-integrated. The lines 11-14 would perform the partial integration $\int u_{xxy}u_{xyy}\, dx = u_{xy}u_{xyy} - \int u_{xy}u_{xxyy}\, dx$. Going through these lines the second time to compute $\int u_{xy}u_{xxyy}\, dx$ would reverse this partial integration and so on. The purpose of the test in line 16 is to prevent an infinite loop.

In this example the steps are:

$$
\begin{aligned}
u^\alpha_J &:= u_{xxy} \\
\mathrm{highpow}(\Delta, u_{xxy}) &= 1 \\
a &:= u_{xyy} \\
u^\alpha_{J-x} &= u_{xy} \\
b &:= u_{xyy}\, d(u_{xy}) = u_{xyy} u_{xy} \\
P &:= u_{xyy} u_{xy} \\
\Delta &:= \Delta - (u_{xyy} u_{xy})_x = -u_{xxyy} u_{xy} \\
u^\alpha_{\tilde{J}} &:= u_{xxyy} \\
u_{xxyy} &>_{lex} u_{xxy} \;\Rightarrow \Delta \text{ is not exact.}
\end{aligned}
$$

Proof for the test in line 16 to be a necessary integrability criterion:

If an expression Δ has an integral P with $D_x P = \Delta$ then for a leading derivative LD based on a lexicographical ordering with x having highest priority we have

$$D_x(\mathrm{LD}(P, u^\alpha)) = \mathrm{LD}(D_x P, u^\alpha) = \mathrm{LD}(\Delta, u^\alpha) = u^\alpha_J. \tag{4}$$

This argument justifies the test in line 16 to be a necessary criterium for integrability. The case $u^\alpha_{\tilde{J}} = u^\alpha_J$ can theoretically not occur. (It can occur in practice if expressions Δ are input that are not polynomial in the u^α so that simplification routines in the computer algebra system used are not effective for expressions occurring in Δ and consequently in line 14 the terms with u^α_J do not cancel. In that case it is appropriate to stop in line 14 too but without concluding non-integrability.)

Finally, if a test in lines 10,16,18 gives `true`, i.e. if the current value of Δ is not a total x-derivative then the original input Δ cannot be a total x-derivative because the two differ by total derivatives (the repeatedly subtracted $D_x b$ in line 14).

The termination of the algorithm

The algorithm as shown above is finite.

Proof:

1. Assume the leading derivative u^α_J as determined in line 8 (or determined in line 15,17) is $u^\alpha_J = \partial^n_x u^\alpha_{K_i}$ where n is the highest x-derivative of u^α in Δ and $u^\alpha_{K_i}$

does not involve x-derivatives. Then the test in line 16 guarantees that the new leading derivative $u^\alpha_{\tilde{J}} = \partial_x^{\tilde{n}} u^\alpha_{\tilde{K}_i}$ determined in line 15 has either

(a) $\tilde{n} < n$, or

(b) $\tilde{n} = n$, $u^\alpha_{\tilde{K}_i} <_{lex} u^\alpha_{K_i}$.

2. There are only finitely many different derivatives $u^\alpha_{K_m}$ (not involving x derivatives) which either turn up themselves in Δ or x-derivatives of them turn up in Δ. Their number does not increase as only x-derivatives are taken in the algorithm. Therefore case (1b) can happen only a finite number of times before in case (1a) the order in x must be lowered. This in turn can also happen only a finite number of times before u^α appears without x-derivatives (end of inner loop) or does not occur anymore in Δ (end of outer loop).

Methods of speeding up the algorithm

In this paragraph methods are described which proved useful in the author's REDUCE implementation of the algorithm.

1. The proof given above for the test in line 16 to be a necessary criterion also extends to the test

$$\textbf{if } \mathrm{LD}(a, u^\alpha) >_{lex} u^\alpha_{J-x} \textbf{ then return } \mathit{nil} \qquad \% \ \Delta \text{ is not exact}$$

put between lines 11 and 12. It allows us to drop the test in line 16. If this new test shows $\mathrm{LD}(a, u^\alpha) >_{lex} u^\alpha_{J-x}$ then x-differentiation would yield a contradiction to condition (4) and therefore non-integrability. In the previous example $\Delta = u_{xxy}u_{xyy}$ the test would give:

$$\begin{aligned} u^\alpha_J &:= u_{xxy} \\ \mathrm{highpow}(\Delta, u_{xxy}) &= 1 \\ a &:= u_{xyy} \\ LD(a, u) &= u_{xyy} \\ u^\alpha_{J-x} &= u_{xy} \\ u_{xyy} &>_{lex} u_{xy} \Rightarrow \Delta \text{ is not exact.} \end{aligned}$$

If this extra test shows non-integrability then it saves the computationally more expensive steps in lines 13, 14 and 15. If it does not find a contradiction then the effort in determining $\mathrm{LD}(a, u^\alpha)$ was wasted because $\mathrm{LD}(\Delta, u^\alpha)$ has to be determined in line 15 anyway as it is needed in line 9 when going through the inner loop the next time. This extra but quick test pays off especially if expressions Δ are big and are non-integrable.

2. According to line 7 in the algorithm EXACT at first all terms containing some function u^α are integrated, then all terms containing some other functions u^β and so on. This requirement can be relaxed as long as it is guaranteed that later partial integrations do not reverse earlier integration steps. One way to ensure this would be to determine in line 8 not the leading derivative $\mathrm{LD}(\Delta, u^\alpha)$ of some given u^α but the leading derivative $\mathrm{LD}(\Delta)$ with respect to any u^α. Then the outer **while** loop in line 7 can be dropped. The effect would be to integrate more x-derivatives of different functions u^α before the test in line 18 is carried out with a chance to find non-integrability. Therefore the overall effect would be a *slowdown*. To reach a *speed up* the opposite measure could be applied. Instead of partially integrating x-derivatives of any function, only x-derivatives of one special non-x-derivative $u^\alpha_{K_i}$ of one function are integrated in the inner loop. This can be accomplished, for example, by a preprocessor step in which all different non-x-derivatives $u^\alpha_{K_i}$ of each function u^α would be renamed as different functions $U^\beta := u^\alpha_{K_i}$ so that Δ involves only x-derivatives of an enlarged set of functions U^β. For example, the expression $\Delta = xu_{xxy} + u_{xy} + {u_{xy}}^2 + x^2 u_{xyy}$ would be rewritten as $\Delta = x{U^1}_{xx} + {U^1}_x + ({U^1}_x)^2 + x^2 {U^2}_x$ by substituting $U^1 := u_y, \;\; U^2 := u_{yy}$.

   This renaming does not affect integrability because only x-differentiations and x-integrations are performed and non-x-derivatives therefore do not change. As a result the test in line 18 is already made after all x-derivatives of one U^β have been integrated. Non-integrability is therefore recognized earlier.

3. The above measure can be refined further for the case that Δ is not linear but polynomially non-linear in the U^β and their x-derivatives. Differentiations with

respect to x do not change the power of any U^α in a monomial. This allows us to partition Δ even further into smaller partial sums which all have to be exact if Δ is to be exact. This more fine-grained partitioning is accomplished by associating with each monomial M in Δ a monomial $\tilde{M}$ obtained by dropping all x-derivatives from all functions U^β in M but keeping the U^β themselves and replacing any U^β-independent coefficients by 1. Any two monomials in Δ with the same $\tilde{M}$ belong to the same partial sum. For example, if P contains the monomial $xU^1_x(U^2)^2$ then all monomials in its x-derivative $U^1_x(U^2)^2 + xU^1_{xx}(U^2)^2 + 2xU^1_x U^2 U^2_x$ are associated to the same $\tilde{M} = U^1(U^2)^2$ as is the monomial $xU^1_x(U^2)^2$ in P.

For the example from the previous speed up method this would mean that Δ splits not into two but three parts:

$$
\begin{aligned}
\Delta \;=\; & x{U^1}_{xx} + {U^1}_x && (=\text{: first partial sum } s_1 \text{ with } \tilde{M} = U^1) \\
& +({U^1}_x)^2 && (=\text{: second partial sum } s_2 \text{ with } \tilde{M} = (U^1)^2) \\
& +x^2 {U^2}_x && (=\text{: third partial sum } s_3 \text{ with } \tilde{M} = U^2)
\end{aligned}
$$

The speed up has the following sources.

(a) By splitting up Δ into smaller partial sums, smaller expressions have to be handled within the inner loop.

(b) Instead of having to integrate all x-derivatives of U^1 in $s_1 + s_2$ before detecting non-integrability, this would be found quicker if s_2 is tried before s_1. If s_1 is tried before s_2 then no significant difference from trying both at once would occur. So on average a speed up results.

(c) The partial sums s_i could be sorted by their number of terms before trying their integrability beginning with the shortest s_i.

One more remark concerning this efficiency improvement. The overall gain in speed does depend on the overhead spent on partitioning. The easy and quick access to low level data in the symbolic mode of the computer algebra system REDUCE makes the partitioning effort low compared with the integration itself. This is different in MAPLE, where the strength lies in handling relatively efficiently the very large expressions for which the partitioning method is supposed to be especially useful. For these two reasons the partitioning method turned out to be more useful in a REDUCE implementation than in a MAPLE implementation.

Constants and functions of integration

To complete an integration a constant (or x-independent function if x is not the only independent variable) has to be added to P. If Δ is to be integrated more than once then it is better to add all constants/functions after all integrations have been completed. Assuming Δ is integrated k_i–times with respect to each x_i, then we have to add to the integral P an expression P_c as computed in the following procedure.

Procedure INTCONST

Input $r, m, k_1, \ldots, k_m$ % r is the next index to be used to create new names
% $c_r, c_{r+1} \ldots$ for constants/functions of integration

Output P_c, r

$I_c := 0$

for $i := 1 \ldots m$ **do**

  $s := 0$

  for $j := 1 \ldots k_i$ **do**

    $r := r + 1$

    $s := s \cdot x_i + c_r(x_0, .., x_{i-1}, x_{i+1}, .., x_m)$

  $P_c := P_c + s$

return P_c, r

If P is a linear algebraic equation for one of the u^α and one intends to eliminate and substitute this u^α in other equations, then the choice of the c_r's can be refined. By replacing the original c_r by a product of a new $\tilde{c}_r$ with all those nonzero factors of coeff(P, u^α) which only depend on variables of c_r, these factors will vanish after substitution. For example, for $u = u(x, y)$ the x-integration of

$$0 = yu_x + 2x$$

to

$$0 = yu + x^2 + y\tilde{c}(y)$$

provides simpler expressions after substitution of u than

$$0 = yu + x^2 + c(y).$$

4 A generalization

The generalization of the above integration algorithm to be discussed in this section is not mathematically as clearly defined as the integration of exact differential expressions. Nevertheless this generalization proves to be very useful when the complete integration module is used as part of a program to solve (usually over-determined) systems of differential equations.

If integrations and substitutions are performed in a system of equations then gradually more and more functions occur that depend on few variables. The following generalization concerns equations which are exact up to terms that contain only unknown functions of fewer variables than occur in that equation. The idea is to reach integrability of an equation in n variables at the price of extra conditions (differential equations) in less than n variables. The algorithm EXACT is to be modified as follows

line 7: take only functions u^α depending on all independent variables

line 19: $\int \Delta \, dx$ now includes as well the integration of terms involving only functions of fewer variables than occurred in the original Δ. If this "generalized integration" of all terms involving any unknown function of x as explained below is not possible then Δ cannot be integrated, i.e. *nil* is to be returned.

The method

The remaining expression Δ to be integrated in line 19 is regarded as a sum of products (if necessary as a single term of one or more factors). Each term T_i is written as a product $T_i = V_i W_i$ of a product V_i of x–independent factors and a product W_i of x–dependent factors. The integral is $\int T_i \, dx = \int V_i W_i \, dx = V_i c_r$ with the extra condition $D_x c_r = W_i$ and a new function c_r that depends on exactly all independent variables occurring in W_i. The generalized integration is not successful if any W_i does depend on all variables and does contain an unknown function of x.

Example: If the original Δ has x, y as independent variables and involves a term

$\sin(xy)\, g$ with $g = g(x)$ depending only on x then this term cannot be factorized into an x–independent factor V and an x–dependent factor W which does not depend on all variables. The generalized integration is therefore not successful. If one would introduce a new function c_r with $D_x c_r = \sin(xy)\, g$ then c_r would have to depend on all variables x and y which would make this integration useless as no information is gained.

Improvements

In order to reduce the number of new functions and extra conditions that are added and that have to be solved afterwards, the following refinements have been implemented in the program CRACK.

1. Instead of determining V_i to be the product of x–independent factors, it now can contain powers of x, i.e. it can be a polynomial of x.

2. Terms $T_i, T_j, \ldots$ with equal W–factor are added to $T := (V_i + V_j + \ldots)W$.

3. For the integration of T only one new function c which depends on all variables occurring in W has to be introduced in the following way. We assume we want to integrate $(v_0 + v_1 x + \ldots + v_n x^n)W$. Only for simplicity in notation in the following equation we use $\partial_x^k c = c^{(k)}$:

$$
\begin{aligned}
\int W\, dx &= c^{(n-1)}, \\
\int xW\, dx &= -c^{(n-2)} + xc^{(n-1)}, \\
\int x^2 W\, dx &= c^{(n-3)} - 2xc^{(n-2)} + x^2 c^{(n-1)}, \\
&\ldots \\
\int x^k W\, dx &= (-1)^k k! c^{(n-k-1)} + \ldots - kx^{k-1} c^{(n-2)} + x^k c^{(n-1)} \\
&= \sum_{m=0}^{k} (-1)^m \frac{k!}{(k-m)!} x^{k-m} c^{(n-m-1)} \\
&\ldots \\
\int x^n W\, dx &= \sum_{m=0}^{n} (-1)^m \frac{n!}{(n-m)!} x^{n-m} c^{(n-m-1)}
\end{aligned}
$$

5 Implementations

The first implementation known to the author was written in the computer algebra system FORMAC in 1987 [11]. A later version was implemented by A. Brand and the author in the computer algebra system REDUCE. This version is currently (1999) used in the package CRACK for the solution of over-determined PDE systems. In autumn 1998 a MAPLE version was implemented by M. White in collaboration with the author. Khai Vu independently translated the integration module of CRACK into MAPLE (see [1]) but without the part of section 4.

Acknowledgement

The author thanks Andreas Brand for years of collaboration on algorithmic and implementation issues regarding the package CRACK. Malcolm MacCallum is thanked for comments on the manuscript.

References

[1] J. Carminati and Vu K. Symbolic computation and differential equations: Lie symmetries. to appear in J. of Symb. Comp.

[2] A. Head. Lie and BigLie, programs to find lie symmetries of differential equations. `http://www.cmst.csiro.au/LIE/LIE.htm`.

[3] W. Hereman. Symbolic software for lie symmetry analysis. In N.H. Ibragimov, editor, *CRC Handbook of Lie Group Analysis of Differential Equations, Vol. 3: New trends in Theoretical Developments and Computational Methods*, chapter 13, pages 367–413. CRC Press, Boca Raton, Florida, 1996.

[4] E. Hubert. Essential components of algebraic differential equations. To appear in the Journal of Symbolic Computation, 2000.

[5] E.L. Mansfield. The differential algebra package diffgrob2. *Mapletech*, 3:33–37, 1996.

[6] 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*, 7:604–635, 1996.

[7] 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, 1995.

[8] J. Sherring. *Symmetry and computer algebra techniques for differential equations*. PhD thesis, La Trobe University, 1993.

[9] J. Sherring, G. Prince, and M. Jerie. Symmetry determination and linear differential equation package. `http://www.latrobe.edu.au/www/mathstats/Maths/Dimsym`.

[10] T. Wolf. Combining Gröbner basis computations and integrations. in preparation.

[11] T. Wolf. A package for the analytic investigation and exact solution of differential equations. In *Proceedings of EUROCAL 87, Lect. Notes of Comp. Sci.*, volume 378, pages 479–91, 1989.

[12] T. Wolf. A comparison of four approaches to the calculation of conservation laws. accepted for publication in the Europ. J. of Appl. Math., 1998.

[13] T. Wolf, A. Brand, and M. Mohammadzadeh. Computer algebra algorithms and routines for the computation of conservation laws and fixing of gauge in differential expressions. *J. Symb. Comp.*, 27:221–238, 1999.