Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

THOMAS WOLF [1]

The Parametric Solution of Underdetermined linear ODEs

[1]Department of Mathematics, Brock University, 500 Glenridge Avenue, St.Catharines, Ontario, Canada L2S 3A1, email: twolf@brocku.ca

ZIB-Report 08-15 (March 2008)

The Parametric Solution of Underdetermined linear ODEs

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

March 10, 2008

Abstract

The purpose of this paper is twofold. An immediate practical use of the presented algorithm is its applicability to the parametric solution of underdetermined linear ordinary differential equations (ODEs) with coefficients that are arbitrary analytic functions in the independent variable. A second conceptual aim is to present an algorithm that is in some sense dual to the fundamental Euclids algorithm, and thus an alternative to the special case of a Gröbner basis algorithm as it is used for solving linear ODE-systems. In the paper Euclids algorithm and the new 'dual version' are compared and their complementary strengths are analysed on the task of solving underdetermined ODEs. An implementation of the described algorithm is interactively accessible under [7].

1 Introduction

Underdetermined ordinary and partial differential equations (ODEs/PDEs) are typical objects of investigation in control theory but also overdetermined systems of equations where originally the number of equations is larger than the number of functions may turn into underdetermined problems in the course of their solution. Examples are the conditions for Lie-symmetries and conservation laws of linearizable differential equations. These overdetermined systems of conditions have solutions that involve arbitrary functions of one or more variables, i.e. in the process of solving these systems towards its conclusion, underdetermined problems often occur.

Algorithms for solving underdetermined linear equations and systems (ODEs, PDEs, multidimensional discrete,..) are known ([1],[2],[3]). There also exists efficient software by Daniel Robertz et al. [4], [5]. The purpose of this paper is to describe an alternative algorithm that is elegant, efficient and in some sense complementary to the fundamental Euclids algorithm which is the basis for the Gröbner basis algorithms that are used in other implementations. An open question is whether the new algorithm can be generalized to partial differential equations (PDEs) and thus not only be a complimentary algorithm for the non-commutative differential Gröbner basis algorithm for solving ODEs but also for solving PDEs.

An implementation of algorithms described in this paper is accessible online under [7] and is part of the package CRACK ([8]) for solving overdetermined systems. As we will describe, a new feature of these programs are flags which allow to prevent denominators or that allow to reduce the number of terms in the solution. This can be achieved by absorbing explicit x-dependent factors into new functions that are introduced during the computation.

We consider a single linear ODE

$$\begin{aligned} 0 &= \sum_i^r D_i f_i + c_{00}(x), \qquad r > 1, && (1)\\ D_i &= \sum_{j=0}^{n_i} c_{ij}(x) \left(\frac{d}{dx}\right)^j, \qquad n_i > 0 \end{aligned}$$

for functions $f_1 \ldots f_r$ of the independent variable x. The equation may be homogeneous ($c_{00} = 0$) or inhomogeneous ($c_{00} \neq 0$). With the requirement that the ODE is underdetermined we only assume that at least two functions f_i are involved, i.e. $r > 1$. The differential order of each f_i is n_i. For simplicity we assume that all coefficients c_{ij} are sufficiently often differentiable, at least $\sum_i n_i$ times. An upper index in round brackets indicates the number of differentiations, i.e. $\frac{d^n f_i}{dx^n} = f_i^{(n)}$ and low order derivatives are denoted by apostrophe, e.g. $\frac{d^2 f_i}{dx^2} = f_i''$.

The task is to find the general solution of the ODE (1) in the form of explicit differential expressions F_i

$$f_i = F_i(x, h_1(x), \ldots, h_{r-1}(x)), \quad i = 1, \ldots, r \qquad (2)$$

in terms of parametric functions $h_1(x), \ldots, h_{r-1}(x)$ which are either all free or one of them, say $h_1(x)$, having to satisfy an ODE and the other $h_j(x)$ being free.

For example, the general solution of the ODE

$$0 = f''x^2 + g''x - g'x^2 + f + 3x$$

for $f = f(x),\ g = g(x)$ can be written in the form

$$\begin{aligned} f &= \frac{x}{x^8 - 2x^6 + 7x^4 - 6x^2 + 9}\Big((x^5 - x^3 + 3x)h'' - (x^6 + x^4 + 3x^2 - 6)h' && (3)\\ &\quad +(3x^5 + 3x^3 + 17x)h - 3x^8 + 3x^6 - 16x^4 + 9x^2\Big), \\ g &= \frac{x}{2(x^8 - 2x^6 + 7x^4 - 6x^2 + 9)}\Big((-2x^6 + 2x^4 - 6x^2)h'' && (4)\\ &\quad +(8x^5 - 4x^3)h' - (14x^4 + 14x^2 + 6)h + 4x^7 + x^5 + 3x^3 - 27x\Big) && (5) \end{aligned}$$

where $h = h(x)$ is an arbitrary function. The form of the solution is not unique. Because $h(x)$ is arbitrary, replacing h by a differential expression in one or more arbitrary functions would give a solution too, but a solution with expressions of higher differential order in more arbitrary functions without the solution being more general. One naturally seeks a general solution which involves only as few as possible arbitrary functions and lowest order derivatives of them. But even this requirement does not give a unique solution. One might want to minimize the highest order of all parametric functions or, for example, the sum of all orders of all parametric functions.

Furthermore, the occurring function(s) could be scaled to modify the form of the solution, for example, to make it denominator free. In the above case replacing $h(x)$ by $(x^8 - 2x^6 + 7x^4 - 6x^2 + 9)^3 p(x)$ with an arbitrary function $p(x)$ would make the solution polynomial but increase its size, i.e. the total number of terms in the coefficients of the parametric functions on the right hand sides.

In the following section the algorithm is given, followed by comments on its characteristics. In section 3 and 4 we formulate the new algorithm and Euclids algorithm in vector notation in order to compare them in detail in section (5). We show that both are essentially different but also that they compliment each other in the sense that one can give a criterion under which circumstances which of both is better suited. Another option is to combine both algorithms in a hybrid version.

Finally, in section (7) an application is described which arose from a classification of hyperbolic evolutionary vector PDEs.

2 The Algorithm

2.1 Outline

The essence of the algorithm is

- to partition the ODE into a total derivative and an algebraic remainder,
- to introduce a new function f_{r+1} such that the total derivative part is df_{r+1}/dx and thus to write the ODE as a system of 2 equations: one equation defining f_{r+1} and one re-formulating the ODE in terms of f_{r+1},
- to use the re-formulated ODE to eliminate and substitute another function and thus to arrive again at a single ODE for the same number of functions which in some sense is closer to be solvable algebraically than the ODE before.

These steps are repeated until either the ODE involves only one function and thus is not underdetermined anymore, or until one function occurs purely algebraically and thus allowing the ODE to be solved for that function. The following is a more detailed pseudo code description.

2.2 Pseudocode

Input • list of functions $f_1, \ldots, f_r$ of x,
• linear ODE $0 = \omega$ in f_i (like (1))

Body

$L := \{\}$ % L will be a list of substitutions
$s := r$
while (the ODE $0 = \omega$ involves at least two f_i) **and**
(differential orders $n_i > 0\ \forall f_i$) **do**

- Factor out $\frac{d}{dx}$ from ω once as far as possible
  by introducing a new function $f_{s+1}(x)$ and
  by computing expressions b_i:

$$0 = \omega = {f_{s+1}}' + \sum_{i=1}^{s} f_i b_i + a_{00} \tag{6}$$

where f_{s+1}, b_i are given through:

$$f_{s+1} = \sum_{i=1}^{s} \sum_{j=0}^{n_i-1} f_i^{(n_i-1-j)} \sum_{k=0}^{j} (-1)^{(j+k)} a_{i,n_i-k}^{(j-k)} \tag{7}$$

$$b_i = \sum_{k=0}^{n_i} (-1)^{(n_i+k)} a_{i,n_i-k}^{(n_i-k)} \tag{8}$$

- **if** $b_i = 0$ for all i **then**
  the ODE $0 = \omega$ is exact (apart from a_{00}), i.e.
  consider new ODE $0 = \hat{\omega} := f_{s+1} + \int a_{00}\, dx$
  where f_{s+1} is only an abbreviation defined in (7)

  else

  ∘ regard f_{s+1} as a new unknown function,
  ∘ solve (6) for one function f_j of $f_1, .., f_s$ that
  − has a non-vanishing coefficient b_i in (6), and
  − is of lowest possible order in ω and thus in (7),

$$f_j = -\frac{1}{b_j}\left({f_{s+1}}' + \sum_{i \neq j} f_i b_i + a_{00}\right) \tag{9}$$

  ∘ use (9) to substitute f_j in (7) to get a new ODE
  $0 = \hat{\omega} := -f_{s+1} + \sum_{i=1}^{s} \sum_{j=0}^{n_i-1} \ldots$
  for functions $f_1, .., f_{j-1}, f_{j+1}, .., f_s, f_{s+1}$
  ∘ update $\omega := \hat{\omega}$, $s := s + 1$, $L := \{f_j = ..\} \cup L$

end
if the ODE involves a function f_j purely algebraically **then**
solve for f_j and add it to L : $L := \{f_j = ..\} \cup L$

$h(x) := f_s(x)$ % to remember the last introduced function
$P := L$ % to remember the complete list of substitutions L
% next substitutions each as stored in the first
% element of P are performed in the rest of P
while $s > r$ **do**
$P := rest(P)|_{f_s = \ldots}$ % as given in $first(P)$
$s := s - 1$
end
end   % of Body

Output • l % list of new functions
• L % the complete list of substitutions
• P % All initial functions $f_1, .., f_r$ are either
% parametric, or are given in P in terms of h
• $0 = \omega(x, h(x))$ % only if the first while loop
% terminates due to ω not
% being underdetermined anymore

2.3 Comments

Before comparing the algorithm with Euclids algorithm a few comments are necessary.

- All steps involve only algebra or differentiations with only one exception: if the homogeneous part of the ODE is exact then the integral of the inhomogeneous part is taken. But this integral does not have to be evaluated, i.e. to be expressed in terms of elementary functions. It can stay in a symbolic unevaluated form for the algorithm to continue.

- The coefficients a_{ij} can be arbitrary explicit functions of x, and involve, for example, sin or log with the only condition that it must be decidable whether expressions involving these functions and their derivatives are zero or not.

- The transformation of the ODE in each step of the first while loop is reversible, i.e. the new ODE is equivalent to the previous one. Therefore the obtained solution is the general one.

- The algorithm terminates.
  Proof:
  We consider how the total sum $\sum_{i=1}^{s} n_i$ of differential orders n_i of all functions f_i changes during execution. After substitution of f_j with (9) in (7) the new differential orders $\hat{n}_i$ of functions f_i that occur in the new ODE are:

$$\hat{n}_i \quad \begin{cases} = & n_j & \text{for } i = s+1 \\ \leq & \max(n_j - 1, n_i - 1) = n_i - 1 & \text{for } i \leq s,\ b_i \neq 0 \\ = & n_i - 1 & \text{for } i \leq s,\ b_i = 0. \end{cases}$$

  The function f_j of order n_j gets replaced by a new function f_{s+1} which then occurs with same order n_j, but the order of all other functions is lowered by at least one because we choose j such that $n_j = \min(n_k)$ ($\forall k$ with $b_k \neq 0$), i.e. we have $\hat{n}_i \leq n_i - 1$ also for the f_i with $b_i \neq 0$. Because the ODE has at least two functions, the total sum of derivatives is decreasing. The algorithm is therefore finite.

- The algorithm results in an ODE for a single function iff the differential operators D_i in (1) have a common differential factor. The remaining ODE is of the same order as the common factor, i.e. its order can not be higher than the order of the original ODE (1).

- The algorithm naturally splits into two parts, a part A) establishing a list L of substitutions (9) in the first while loop and part B) performing the substitutions in the second while loop to obtain an explicit solution. The first part is executed very fast (see next section for more details), the second may take longer for higher order ODEs because expressions typically grow with each substitution, often exponentially. An example is given in the appendix.

For many applications the list L of substitutions may even be of higher practical value than the explicit formulas $f_i = F_i(x, h_1(x), \ldots, h_{r-1}(x)), \quad i = 1, \ldots, r$ resulting from B). In general, the list L is a much shorter representation of the parametric solution of the ODE than the explicit solution itself and thus is more useful as a solution, like in the following scenario.

Let us assume that we have a large algebraic expression in terms of the original functions $f_1, .., f_r$ that has to be evaluated modulo the underdetermined ODE (1). Instead of replacing the f_i directly by their large explicit expressions as given in the list P it is usually much better to perform successively the substitutions stored in L in the order they were derived, allowing cancellations to happen after each individual substitution. Also, if numerical computations are to be done, it is much faster to compute the sequence of substitutions than the explicit expressions.

3 A vector representation

To perform the iteration process efficiently, i.e. to replace functions by algebraic combinations of other functions (9) in the ODE (7) without needing to perform any differentiations, the ODE has to be represented in a form where $D := \frac{d}{dx}$ is factored out as far as possible. As we will see further down, this is also the appropriate representation for using the Euclidean algorithm to solve the underdetermined ODE. We write the ODE in the form

$$\begin{aligned} 0 &= \sum_i^r A_i f_i \, + \, a_{00}(x) && (10) \\ A_i &= D\tilde{A}_i + a_{i0}(x) && (11) \\ \tilde{A}_i &= D^{n_i - 1} a_{in_i}(x) + \ldots + D a_{i2}(x) + a_{i1}(x) && (12) \end{aligned}$$

Replacing, for example, $f_1 = w(x)\hat{f}_1$ would only require multiplications $\hat{a}_{1k} := w a_{1k}$ to update this representation and no differentiations.

In this notation a single step in the first while loop of the new method consists of

- introducing a new function f_{r+1} through

$$f_{r+1} = \sum_i \tilde{A}_i f_i \tag{13}$$

  giving the ODE the form

$$0 = D f_{r+1} + \sum_i a_{i0}(x) f_i + a_{00}. \tag{14}$$

- regarding the defining relation (13) for the new function as the new ODE and using the old ODE (14) for a substitution of an 'old' function. For that we choose one a_{j0} of the non-vanishing a_{i0} in (11) for which the order n_j of the corresponding f_j is minimal. Thus, performing the substitution

$$f_j = -\frac{1}{a_{j0}} \left(D f_{r+1} + \sum_{i \neq j} a_{i0} f_i + a_{00} \right) \tag{15}$$

  in the definition (13) gives the new ODE

$$0 = -f_{r+1} + \sum_{i \neq j} \tilde{A}_i f_i - \sum_{i \neq j} \tilde{A}_j \left(\frac{a_{i0}}{a_{j0}} f_i \right) - \tilde{A}_j \left(\frac{1}{a_{j0}} D f_{r+1} \right) - \tilde{A}_j \left(\frac{a_{00}}{a_{j0}} \right). \tag{16}$$

In vector notation the above two steps change the differential operators (assuming for simplicity of notation $j = 1$) according to:

$$\begin{pmatrix} A_1 \\ \\ A_2 \\ \vdots \\ A_r \end{pmatrix} \rightarrow \begin{pmatrix} A_{r+1} \\ \\ A_2 \\ \vdots \\ A_r \end{pmatrix} = \begin{pmatrix} -1 - \tilde{A}_1 \frac{1}{a_{10}} D \\ \tilde{A}_2 - \tilde{A}_1 \frac{a_{20}}{a_{10}} \\ \vdots \\ \tilde{A}_r - \tilde{A}_1 \frac{a_{r0}}{a_{10}} \end{pmatrix} \tag{17}$$

$$a_{00} \rightarrow -\tilde{A}_1 \left(\frac{a_{00}}{a_{10}} \right) \tag{18}$$

and add (15) to the list of substitutions. Updating components $2 \ldots r$ in (17) is done purely by multiplications and additions, only updating the first component takes differentiations when factoring out D.

A more conventional algorithm solving underdetermined linear ODEs tries to lower the order of the ODE by performing Euclids algorithm until the ODE is algebraic for one function and thus can be solved algebraically. In the following section we formulate such an Euclidean step to be able to compare it with the new algorithm above.

4 The Euclid version in vector notation

For applying the 'right' Euclid algorithm the ODE is given as well in a form with D completely factored out:

$$0 = \sum_i^r A_i f_i + a_{00}(x) \tag{19}$$

$$A_i = D^{n_i} a_{in_i}(x) + \ldots + D a_{i1}(x) + a_{i0}(x). \tag{20}$$

One iteration step is performed by

- choosing two functions f_i, f_j (w.l.o.g. $n_i \geq n_j$) and introducing a new function f_{r+1} through
$$f_j = f_{r+1} - D^{n_i - n_j} \left(\frac{a_{in_i}}{a_{jn_j}} f_i \right) \tag{21}$$
- and performing this substitution in the ODE (19).

The substitution is chosen such that the differential order of f_i is lowered by at least one. This iteration process stops like in the new algorithm when the first function appears purely algebraically or when the ODE involves only one function. Therefore, in order to minimize the number of steps one would choose f_j as one of the lowest order functions and f_i as one of the lowest order functions from the remaining ones. (From all these choices it is recommendable to choose a pair so that multiplication is minimized and A_i, A_j have as few as possible terms.) For $j = 1, i = 2$ the vector notation gives

$$\begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ A_r \end{pmatrix} \rightarrow \begin{pmatrix} A_{r+1} \\ A_2 \\ A_3 \\ \vdots \\ A_r \end{pmatrix} = \begin{pmatrix} A_1 \\ A_2 - A_1 D^{n_2 - n_1} \frac{a_{2n_2}}{a_{1n_1}} \\ A_3 \\ \vdots \\ A_r \end{pmatrix} \tag{22}$$

$$a_{00} \rightarrow a_{00}. \tag{23}$$

The update of the 2$^{\text{nd}}$ component in (22) requires differentiations when D is factored out as far as possible.

5 Relations between both algorithms

5.1 Differences

In this section we want to justify our claim that both algorithms differ significantly from each other and that they are not merely variations of one and the same procedure.

Both algorithms differ conceptually in that Euclids algorithm pairs two differential operators to get a new differential operator of lower order whereas in the new algorithm the relation that defines a new function becomes the ODE and the old ODE is used for substituting one of the 'old' functions.

Another difference is that Euclids algorithm lowers in one step the order of only one differential operator whereas the new algorithm lowers the order of all but one differential operators. Both types of steps are performed at about the same cost as for the new method only the update of the first component in (17) is potentially size increasing and for Euclids method it is only the update of the second component of (22). Using Euclids algorithm one can of course take one of the lowest order differential operators and decrease the order of all others but that would take $r-1$ computations, each potentially size increasing. This difference does usually not affect the number of necessary iterations (until one function occurs purely algebraically), because Euclids method can operate just on two of the lowest order operators and ignore all others. So, by applying Euclids method to lower the order of the two lowest order operators A_i with each other, all other higher order operators keep their high order. Moreover, when the algorithm stops because the final equation contains one function algebraically, then back substitutions start (part B of the algorithm) which even increase the highest orders.

Differently with the new method where the differential order of all functions but one gets lowered in one step. Thus the obtained parametric solution is typically of lower order than the solution obtained by Euclids algorithm if this is executed by pairing only lowest order derivatives.

For example, the size of the parametric solution of the ODE

$$0 = x^3 f_{ax} + (x-1) g_{bx} + h_{5x}, \quad \text{for } f, g, h \text{ of } x \tag{24}$$

and the differential order of parametric functions in it depend strongly on the differential orders a, b of f and g in the ODE and the method that is used. As a, b increase from $a = b = 1$ to $a = b = 3$ one can see the following trends:

- In Euclids solution the differential order of parametric functions increases from 5 to 7 whereas in the solution of the new algorithm the order decreases from 4 to 3 in the expression for g, from 4 to 2 in the expression for f and only increases from 0 to 1 in the expression for h.

- The size of expressions in Euclids solution steadily increases and in the solution of the new method it decreases.

  For $a = b = 1$ the solutions have comparable size:
  - When explicit x-dependent factors are not absorbed (for absorbing factors see section 5.3) then h is parametric and the rational expressions for f, g have the form (5 terms)/(2 terms) in Euclids solution and in the solution of the new method $f =$ (4 terms)/(2 terms), $g =$ (3 terms)/(2 terms).

  For $a = b = 3$:
  - When factors are not absorbed then in Euclids solution h is parametric, $f =$ (85 terms)/(15 terms), $g =$ 13 terms and in the solution of the new algorithm we have $f =$ 1 term, $g =$ 7 terms, $h =$ 4 terms.

- When factors are absorbed then Euclids solution gives $f = 27$ terms, $g = 16$ terms and in the solution of the new algorithms we have $f = 1$ term, $g = 7$ terms, $h = 4$ terms.

Not only the solutions of both algorithms may differ significantly but also the number of steps to reach them. Starting, for example, with

$$0 = f' + a(x)h^{(n)} = 0$$

to be solved for $f = f(x), h = h(x)$ this takes only one iteration step with the new algorithm giving, for example for $n = 5$, a solution of the form $f = (27 \text{ terms})/(1 \text{ term})$, $h = (1 \text{ term})/(1 \text{ term})$, whereas it takes n steps for Euclids algorithm giving, for example for $n = 5$, a more spacious solution of the form $f = (40 \text{ terms})/(1 \text{ term})$, $h = (2 \text{ terms})/(1 \text{ term})$.

If the ODE is inhomogeneous then another difference between both algorithms becomes apparent. With non-zero inhomogeneity a_{00} in the ODE (10) the transformation (15) becomes inhomogeneous and changes a_{00} in (18) whereas Euclids algorithm does not change a_{00} in the homogeneous transformation (21).

Two ODEs that highlight the duality between both algorithms are

$$0 = f' + f + g' + (ah)^{(20)} \qquad (25)$$
$$0 = f' + f + g' + a(h)^{(20)} \qquad (26)$$

where $a = a(x)$ is a given function of x and the ODE has to be solved for f, g, h. As shown in table 1 for the ODE (25) the new algorithm is to be preferred whereas for ODE (26) Euclids algorithm gives a shorter solution.

algorithm	equation (25)		equation (26)	
new	f=1 term	g=22 terms	f=2 terms	g=23 terms
Euclid	f=22 terms	g=23 terms	f=2 terms	g=3 terms

Table 1. The size of solutions given by both algorithms (both executed without absorbing of factors).

An explanation of the above behaviour comes from the fact that Euclids method is better suited when factors $\frac{a_{in_i}}{a_{jn_j}}$ are small, i.e. when the coefficients of the leading derivatives in the representation (10) - (12) effectively cancel each other in a quotient, and less suited when these factors are large prime expressions in x. Differently, the new method works best if factors $\frac{a_{i0}}{a_{j0}}$ are small, i.e. when the coefficients of the algebraic terms effectively cancel each other, and less suited when these factors are large prime expressions in x.

Although both ODEs look similar, they are rather different as $(ah)^{(20)}$ has many terms if the product rule of differentiation is fully applied and $a(h)^{(20)}$ has many terms if $\frac{d}{dx}$ is factored out as far as possible. Thus both algorithms differ in their suitability for both ODEs.

The above computations and any other tests can easily be performed online (see [7]), where access to the computer algebra system REDUCE and to the procedures `uode` and `print_uode_solution` is provided.

5.2 Hybridization

The advantage of having these two, in some sense, complementary algorithms lies in the fact that both operate on the same data structure, i.e. both their input and output consists of an ODE in the form (10) - (12) and both add one substitution to a list of substitutions of functions in terms of newly introduced functions. Consequently, both algorithms are interchangeable, i.e. one step performed with one algorithm could be followed by another step performed with the other algorithm, in order to minimize the size of x-dependent factors and thus to minimize growth. A different strategy could be to perform each step with both algorithms in parallel, to compare the size of the ODE and/or size of derived substitution that both methods give and to choose which one to adopt for this step. This strategy does at most double the amount of computation but more likely will lead to savings due to working with smaller expressions.

Any hybrid algorithm performing a mixture of Euclid steps and new algorithm steps is still finite because in any such step the sum of differential orders of all functions is decreasing.

5.3 Absorbing factors

The following efficiency improving measures work for both methods. A representation (10) - (12) (and identically (19) - (20)) of the ODE where D is maximally factored out has the advantage that any change of functions $f_i \rightarrow h(x)\bar{f}_i$ does require only multiplications and is done very easily in both algorithms. This freedom of efficiently multiplying functions with x-dependent factors can be used for different purposes.

If in a newly generated ODE all coefficients of a function f_i have a non-trivial GCD $\hat{c}(x) := GCD(a_{in_i}, \ldots, a_{i0}) \neq 1$ then this can be absorbed into a new function f_{r+1} and the ODE be simplified by performing the substitution

$$f_i = f_{r+1}/\hat{c} \tag{27}$$

and adding it to the accumulating list of substitutions. Such non-trivial GCDs occur relatively frequently as the GCD is taken only from the few coefficients a_{ij} of any single one operator A_i that changed in an iteration step.

A different purpose of introducing new functions multiplied with an explicit x-depending factor is to avoid a denominator (den). This would arise in the first component of (17) and it can be prevented by introducing another new function f_{r+2} and by performing the substitution

$$f_{r+1} = a_{10}^2 f_{r+2}. \tag{28}$$

Similarly, in the other components of (17) one can scale

$$f_i = \operatorname{den}\left(\frac{a_{i0}}{a_{10}}\right) f_{r+i}. \tag{29}$$

In Euclids algorithm the appearance of denominators in the second component of (22) can be prevented by introducing the scaling

$$f_2 = \operatorname{den}\left(\frac{a_{2n_2}}{a_{1n_1}}\right) f_{r+1}. \tag{30}$$

Another situation where a dominator occurs is at the end of both algorithms when one function occurs purely algebraically

$$0 = a_{i0} f_i + \sum_{j \neq i} A_j f_j \; + \; a_{00}(x). \tag{31}$$

To avoid the denominator a_{i0} one can scale all f_j depending on their differential order and on their coefficients in A_j. Avoiding this denominator is especially helpful, as $f_i = \ldots$ is the last substitution and thus substituted successively in all previous substitutions leading easily to a substantial growth of denominators. By avoiding denominators in the above way both methods produce denominator free solutions if the ODE is homogeneous.

5.4 Embedded ODEs

The case that an underdetermined ODE factorizes, i.e. that the differential operators A_i in (10) have a non-trivial GCD, or in other words, that the ODE can be written in form of two nested ODEs $0 = \Omega(x, \omega(x, f_i))$ is discovered by both algorithms. The inner ODE $0 = \omega$ is only determined up to a linear change $\omega = \alpha(x)\hat{\omega}$ and both algorithms will usually find Ω's that differ by some $\alpha(x)$. A slight advantage of the new algorithm is that the D^0 part of the ODE is used within the algorithm, thus it is automatically recognized if this vanishes, i.e. if the ODE is exact (up to an inhomogeneity).

6 ODE-systems

The introduced methods of solving a single underdetermined ODE (or converting it to an ODE for a single function) can be used to convert an ODE system into an equivalent set of fully decoupled ODEs, each for a single function.

After treating any one equation of the original ODE system the original functions in it are expressed in terms of fewer functions which are either all free or at most one has to satisfy a single ODE. All functions can be replaced in the remaining ODEs. This can go on as long as we have equations of at least two functions. When the procedure stops we have solved the system or got ODEs, each containing only one function. If more than one ODE contain only one and the same single function, then these ODEs form an over-determined subsystem which can be treated by a Gröbner bases computation and result either in the explicit solution for this function or a single ODE for this function of an order not higher than the lowest order of the ODEs for this function. This whole procedure terminates with either the explicit parametric solution of the original system or a decoupled set of ODEs, each ODE for a single function.

7 An application

A class of applications where underdetermined linear ODEs occur frequently is the classification of hyperbolic evolutionary PDE systems. The aim of such an investigation is to find integrable systems of PDEs by determining those systems which have a higher order symmetry (see below). More information about the mathematical background is given in [6] where a classification of hyperbolic *vector* PDEs is discussed.

Let us look at two hyperbolic scalar PDEs for functions $u(x,t,\tau), v(x,t,\tau)$ (where x, t are the usual independent variables and τ is a symmetry parameter). The following ansatz for the system and symmetry is generated based on homogeneity considerations. By requiring the same homogeneity weights as the potential nonlinear Schrödinger equation we get for the system the ansatz

$$\begin{aligned} u_{tx} &= a_{10}u_{3x} + a_6 u_x v_x^2 u^2 + a_8 u_{2x} u_x v + a_9 u_{2x} v_x u + a_1 u_x^2 v_x + a_3 u_x^3 v^2 + a_7 u_x v_{2x} u \\ v_{tx} &= a_{19}v_{3x} + a_{15} u_x^2 v_x v^2 + a_{18} v_{2x} v_x u + a_{17} u_x v_{2x} v + a_{13} u_x v_x^2 + a_{16} v_x^3 u^2. \end{aligned} \tag{32}$$

Assuming the same differential order for the symmetry we get the ansatz

$$u_\tau = b_{10}u_{3x} + \ldots, \qquad v_\tau = b_{19}v_{3x} + \ldots, \tag{33}$$

with right hand sides identical to those of (32), only with coefficients b_i instead of a_i. All coefficients a_i, b_i are undetermined functions of the product uv.[1]

[1]For hyperbolic systems the homogeneity weights include negative values, for example here

The relations (33) are considered to be a symmetry of the system (32) if the symmetry conditions

$$\partial_\tau(u_{tx}) - \partial_t\partial_x(u_\tau) = 0, \qquad \partial_\tau(v_{tx}) - \partial_t\partial_x(v_\tau) = 0 \tag{34}$$

are fulfilled identically in u, v and derivatives of u and v both modulo substitutions based on (32) and (33). In performing the differentiations in (34), doing repeatedly substitutions (32), (33) and finally setting all coefficients of different products of powers of derivatives of u, v individually to zero gives 27 ODEs with a total of 1334 terms for 26 functions a_i, b_j of $z := uv$. The length of equations ranges from 2 to 266 terms. In the course of solving this system the program CRACK performs integrations, substitutions, splittings (i.e. separations when z occurs only explicitly) and a number of case distinctions. In one of the sub cases the resulting conditions can be integrated successively up to the underdetermined linear ODE

$$0 = 3b_{13}'z - 6b_{15}'z^2 - 2b_{17}''z^2 + b_{17}'z - 6b_{15}z + 2b_{17}. \tag{35}$$

Step 1: To start, partial integration gives

$$0 = (3b_{13}z - 6b_{15}z^2 - 2b_{17}'z^2 + 5b_{17}z)' - 3b_{13} + 6b_{15}z - 3b_{17}. \tag{36}$$

which can be written as

$$0 = c_1' - 3b_{13} + 6b_{15}z - 3b_{17} \tag{37}$$

by introducing $c_1(z)$ through

$$c_1 = 3b_{13}z - 6b_{15}z^2 - 2b_{17}'z^2 + 5b_{17}z. \tag{38}$$

From the functions that occur only algebraically in (37) (i.e. b_{13}, b_{15}, b_{17}) the ones that have lowest derivatives in (35) are b_{13}, b_{15}. Eliminating one of them from (37), say b_{13} and substituting it in (38) gives

$$0 = 2b_{17}'z^2 - 2b_{17}z - c_1'z + c_1 \tag{39}$$

which is not algebraic in any function yet, but already of first order, so one more step has to be performed.

Step 2: Partial integration of (39) and introduction of

$$c_2 = 2b_{17}z^2 - c_1z \tag{40}$$

results in

$$0 = c_2' - 6b_{17}z + 2c_1 \tag{41}$$

which allows to solve for c_1 and replace it in (40) giving

$$0 = \frac{1}{2}c_2'z - c_2 - b_{17}z^2. \tag{42}$$

This condition is purely algebraic for one function, b_{17}, and therefore the algorithm stops.

Cleanup: We need explicit solutions of (35), so what remains to be done are back substitutions: (42) provides

$$b_{17} = \frac{1}{2z}c_2' - \frac{1}{z^2}c_2. \tag{43}$$

---

weight(u)=1, weight(v)=-1, so that uv has weight zero and therefore we have no limitation on the degree of powers of uv and thus all unknown coefficients are arbitrary functions of uv.

The second substitution expressing b_{13} in terms of the parametric function c_2 is obtained after backward substituting c_1 from (41) into (37) and b_{17} from (43) in (37) to get the explicit solution consisting of (43) and

$$b_{13} = \frac{1}{3}{c_2}'' - \frac{3}{2z}{c_2}' + 2b_{15}z + \frac{2}{z^2}c_2 \tag{44}$$

involving the free function $c_2(z)$.

Another underdetermined equation resulting in this integrability problem is

$$0 = 3{b_1}'z - 9{b_3}'z^2 + 3{b_6}'z^2 - 4{b_8}''z^2 - 12{b_8}'z - 12b_3z + 6b_6z - 3b_8 \tag{45}$$

which has the solution

$$\begin{aligned} b_1 &= (-8{b_8}'z^2 + 3{c_3}'z + 6b_6z^2 - 5b_8z - 2c_3)/(3z) \\ b_3 &= (-4{b_8}'z^2 + {c_3}'z + 3b_6z^2 - 3b_8z - c_3)/(3z^2). \end{aligned}$$

We finally obtain as a system with higher order symmetries:

$$\begin{aligned} u_{tx} &= \frac{a_1}{2uv^3}\left((uv)_x^{\ 3}v - uv(uv)_x^{\ 2}v_x\right) \\ v_{tx} &= \frac{a_{13}}{2uv}(uv)_x^{\ 2}v_x. \end{aligned}$$

8 Appendix

The following example illustrates the comments made in the last paragraphs of section 2.3. It shows two possible representations of a solution to an underdetermined ODE. For the equation

$$(x-1)^3f_1^{(5)} + 3f_1^{(3)} + xf_1'' + (1-x^2)f_1' + f_1 - (x-2)(x-3)f_2'' - xf_2' = 0$$

the generated list L of substitutions is given through

$$\begin{aligned} f_6 &= (7289/9f_8'x^6 - 39400/3f_8'x^5 + 725945/9f_8'x^4 - 691667/3f_8'x^3 + 2665016/9f_8'x^2 \\ &\quad -463541/3f_8'x + 40582f_8' - 82543/9f_8x^5 + 393212/3f_8x^4 - 1997777/3f_8x^3 \\ &\quad +1404610f_8x^2 - 9064364/9f_8x + 863648/3f_8)/(x^12 - 27x^11 + 339x^10 - 2551x^9 \\ &\quad +12566x^8 - 43294x^7 + 111667x^6 - 221121x^5 + 332143x^4 - 447477x^3 + 625912x^2 \\ &\quad -422746x + 117948) \\ f_7 &= (-197/419f_8' - 2227/6704f_6x^6 + 40655/6704f_6x^5 - 282747/6704f_6x^4 \\ &\quad +937545/6704f_6x^3 - 737203/3352f_6x^2 + 236215/1676f_6x - 15636/419f_6)/ \\ &\quad (x^5 - 5988/419x^4 + 30423/419x^3 - 64170/419x^2 + 46012/419x - 13152/419) \\ f_1 &= (-288/197f_7' - 450/197f_6)/(x^5 - 2988/197x^4 + 15683/197x^3 - 32571/197x^2 \\ &\quad +18956/197x - 6093/197) \\ f_5 &= -9/16f_6' + 37/32f_1x^4 - 415/32f_1x^3 + 347/8f_1x^2 - 815/32f_1x + 261/32f_1 \\ f_4 &= -4/9f_5' - 17/9f_1x^3 + 124/9f_1x^2 - 176/9f_1x + 11f_1 \\ f_3 &= -1/4f_4' + 5/4f_1x^2 - 17/4f_1x - 11/4f_1 \\ f_2 &= f_3' + 2f_1x + f_1 \end{aligned}$$

which is a much shorter representation of the solution than the 10 page explicit form which results from substituting $f_6, f_7, f_1, f_5, f_4, f_3$ in this order into each other and takes the form f_1 =(42 terms)/(25 terms), f_2 =(304 terms)/(61 terms) involving up to 29-digit integers. The list of 7 substitutions is not only shorter than the list of 2 substitutions for f_1, f_2, it also is much faster to derive and more useful if a differential expression is to be simplified modulo the solution of the above ODE by substituting $f_2, f_3, f_4, f_5, f_1, f_7, f_6$ in this order.

The difference in size of both solution representations can be arbitrarily amplified by having an input ODE of higher order and higher degree polynomials as coefficients.

9 Summary

We present an algorithm for the solution of underdetermined linear ODE that is compatible but structurally different from the (right) Euclidean algorithm. Because both algorithms operate on the same data structure and because both complement each other in the sense that each one is most efficient for ODEs of different form, the combination of both algorithms is superior to each individual one.

Acknowledgements

I would like to thank Sergey Tsarev for many comments. Daniel Robertz is thanked for comparative runs with the OreModules package.

References

[1] Chyzak, F., Quadrat, A., Robertz, D. (2005). "Effective algorithms for parametrizing linear control systems over Ore algebras", Rapport de Recherche INRIA , Applicable Algebra in Engineering, Communications and Computing **16**, no 5, 319-376.

[2] Pommaret, J.-F., Quadrat, A. (2004). "A differential operator approach to multidimensional optimal control", International Journal of Control **77**, 821-836.

[3] Fröhler, S., Oberst, U. (1998). "Continuous time-varying linear systems", Systems & Control Letters **35**, 97-110.

[4] Chyzak, F., Quadrat, A., Robertz, D. OreModules project `http://wwwb.math.rwth-aachen.de/OreModules`

[5] Chyzak, F., Quadrat, A., Robertz, D. "OreModules: A symbolic package for the study of multidimensional linear systems" in: J. Chiasson, J.-J. Loiseau, "Applications of Time-Delay Systems", Springer, to appear.

[6] Anco, S. and Wolf, T. (2005). "Some symmetry classifications of hyperbolic vector evolution equations", JNMP, Volume 12, Supplement 1, p 13-31 (also nlin.SI/0412015).

[7] Wolf, T. (2007). Online demo for solving underdetermined ODEs. `http://lie.math.brocku.ca/crack/uode`

[8] Wolf, T.: Applications of Crack in the Classification of Integrable Systems, CRM Proceedings and Lecture Notes, vol 37 (2004) pp. 283-300. (arXiv nlin.SI/0301032) and online under `http://lie.math.brocku.ca/crack/demo`