On the Integration of Differential Fractions

François Boulier
Université Lille 1
Villeneuve d'Ascq, France
Francois.Boulier@univ-lille1.fr

François Lemaire*
Université Lille 1
Villeneuve d'Ascq, France
Francois.Lemaire@lifl.fr

Georg Regensburger
RICAM
Linz, Austria
Georg.Regensburger@oeaw.ac.at

Markus Rosenkranz†
University of Kent
Canterbury, United Kingdom
M.Rosenkranz@kent.ac.uk

ABSTRACT

In this paper, we provide a differential algebra algorithm for integrating fractions of differential polynomials. It is not restricted to differential fractions that are the derivatives of other differential fractions. The algorithm leads to new techniques for representing differential fractions, which may help converting differential equations to integral equations (as for example used in parameter estimation).

Categories and Subject Descriptors

I.1.1 [**Symbolic and Algebraic Manipulation**]: Expressions and Their Representation—*simplification of expressions*; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*algebraic algorithms*

Keywords

Differential algebra, differential fractions, integration

1. INTRODUCTION

In this paper we present an algorithm that solves the following problem in differential algebra: Given a differential fraction F, i.e., a fraction P/Q where P and Q are differential polynomials, and a derivation δ, compute a finite sequence $F_1, F_2, F_3, \ldots, F_t$ of differential fractions such that

$$F \quad = \quad F_1 + \delta\, F_2 + \delta^2\, F_3 + \cdots + \delta^t\, F_t\,, \tag{1}$$

the differential fractions $\delta^\ell\, F_\ell$ have rank less than or equal to F (the result is thus ranking dependent) and the δ^ℓ differential operators are as much "factored out" as possible. In particular, if there exists some differential fraction G such that $F = \delta\, G$, then $F_1 = 0$: the algorithm recognizes first integrals.

This work originated from our attempts to generalize the construction of integro-differential polynomials [14] to the case of several independent and dependent variables. The results presented here constitute the most important step in this construction: the decomposition of an arbitrary differential polynomial—or differential fraction—into a total derivative and a remainder (see Example 1 with `iterated = false`). For ordinary differential polynomials, such a decomposition is described in [1]. See also [8, 2] and the recent dissertation [12] for further references.[1]

If the remainder is zero, the given differential fraction F is recognized to be the total derivative of another differential fraction G, so the latter appears as a *first integral* of F. Algorithms for determining such first integrals are known (see [15, 16]), and one such algorithm is implemented in Maple via the function `DEtools[firint]`.

Handling differential fractions rather than polynomials may be very important for the range of application of Algorithms 3 and 4 since differential equations may be much easier to integrate when multiplied, or divided, by an integrating factor. Beyond its theoretical interest, the algorithm presented in this paper may also be useful in practice:

1. Sometimes (though not always), a differential fraction is shorter in the representation (1) than in expanded form. Thus our algorithm provides new facilities for representing differential equations.

2. The representation (1) may also be more convenient than the expanded form if one wants to convert differential equations to integral equations. This may be a very important feature for the problem of estimating parameters of dynamical systems from their input-output behaviour (see Example 5).

The paper is organized as follows: In Section 2, we review some standard definitions for differential polynomials and generalize them to differential fractions. In Section 3, the main result of this paper is stated and proved (Algorithm 3 and Proposition 6). In Section 4, we describe an implementation along with a few worked-out examples.

*The first two authors acknowledge partial support by the French ANR-10-BLAN-0109 LEDA project.

†The fourth author acknowledges partial support by the EPSRC First Grant EP/I037474/1.

[1]The authors would like to thank the reviewers for pointing out these references.

ISSAC'13, June 26–29, 2013, Boston, Massachusetts, USA.

2. BASICS OF DIFFERENTIAL ALGEBRA

This paper is concerned with *differential fractions*, i.e., fractions of *differential polynomials*. A key problem with such fractions is to reduce them, which requires computing the gcd of multivariate polynomials, which is possible whenever the base field is computable.

The reference books are [13] and [11]. A *differential ring* $\mathscr{R}$ is a ring endowed with finitely many, say m, *derivations* $\delta_1, \ldots, \delta_m$, i.e., unary operations satisfying the following axioms, for all $a, b \in \mathscr{R}$:

$$\delta(a+b) = \delta(a) + \delta(b)\,, \quad \delta(a\,b) = \delta(a)\,b + a\,\delta(b)\,,$$

and which commute pairwise. To each derivation δ_i an *independent variable* x_i is associated such that $\delta_i\, x_j = 1$ if $i = j$ and 0 otherwise. The set of independent variables is denoted by $\mathscr{X} = \{x_1, \ldots, x_m\}$. The derivations generate a commutative monoid w.r.t. composition denoted by

$$\Theta = \{\delta_1^{a_1} \cdots \delta_m^{a_m} \mid a_1, \ldots, a_m \in \mathbb{N}\}\,,$$

where $\mathbb{N}$ stands for the nonnegative integers. The elements of Θ are called *derivation operators*. If $\theta = \delta_1^{a_1} \cdots \delta_m^{a_m}$ is a derivation operator then $\mathrm{ord}(\theta) = a_1 + \cdots + a_m$ denotes its *order*, with a_i being the order of θ w.r.t. derivation δ_i (or x_i).

In order to form differential polynomials, one introduces a set $\mathscr{U} = \{u_1, \ldots, u_n\}$ of n *differential indeterminates*. The monoid Θ acts on $\mathscr{U}$, giving the infinite set $\Theta\mathscr{U}$ of *derivatives*. For readability, we often index derivations by letters like δ_x and δ_y, denoting also the corresponding derivatives by these subscripts, so u_{xy} denotes $\delta_x\,\delta_y\,u$.

For applications, it is crucial that one can also handle *parametric* differential equations. Parameters are nothing but symbolic *constants*, i.e., symbols whose derivatives are zero. Let $\mathscr{C}$ denote the set of constants.

The differential fractions considered in this paper are ratios of differential polynomials taken from the differential ring $\mathscr{R} = \mathbb{Z}[\mathscr{X} \cup \mathscr{C}]\{\mathscr{U}\} = \mathbb{Z}[\mathscr{X} \cup \mathscr{C} \cup \Theta\mathscr{U}]$. A differential fraction is said to be *reduced* if its numerator and denominator do not have any common factor. A differential polynomial (respectively a differential fraction) is said to be *numeric* if it is an element of $\mathbb{Z}$ (resp. of $\mathbb{Q}$). It is said to be a *coefficient* if it is an element of $\mathbb{Z}[\mathscr{X} \cup \mathscr{C}]$ (resp. of $\mathbb{Q}(\mathscr{X} \cup \mathscr{C})$). The elements of $\mathscr{X} \cup \mathscr{C} \cup \Theta\mathscr{U}$ are called *variables*.

A *ranking* is a total ordering on $\Theta\mathscr{U}$ that satisfies the two following axioms:

1. $v \leq \theta v$ for every $v \in \Theta\mathscr{U}$ and $\theta \in \Theta$,
2. $v < w \Rightarrow \theta v < \theta w$ for every $v,\, w \in \Theta\mathscr{U}$ and $\theta \in \Theta$.

Rankings are well-orderings, i.e., every strictly decreasing sequence of elements of $\Theta\mathscr{U}$ is finite [11, §I.8]. Rankings such that $\mathrm{ord}(\theta) < \mathrm{ord}(\phi) \Rightarrow \theta u < \phi v$ for every $\theta,\, \phi \in \Theta$ and $u,\, v \in \mathscr{U}$ are called *orderly*. In this paper, it is convenient to extend rankings to the sets $\mathscr{X}$ and $\mathscr{C}$. For all rankings, we will assume that any element of $\mathscr{X} \cup \mathscr{C}$ is less than any element of $\Theta\mathscr{U}$; see Remark 5 for a brief discussion why we make this assumption.

Fix a ranking and consider some non-numeric differential polynomial P. The highest variable v w.r.t. the ranking such that $\deg(P, v) > 0$ is called the *leading variable* or *leading derivative* (though it may not be a derivative) of P. It is denoted by $\mathrm{ld}(P)$. The monomial $v^{\deg(P,v)}$ is called the *rank* of P. The leading coefficient of P w.r.t. v is called the *initial* of P. The differential polynomial $\partial P/\partial v$ is called the *separant* of P. More generally, if w is any variable, the differential polynomial $\partial P/\partial w$ is called the separant of P w.r.t. w and is denoted by $\mathrm{separant}(P, w)$.

2.1 Extension to differential fractions

In this section, some of the definitions introduced above are reformulated, to cover the case of differential fractions. For differential polynomials, these new definitions agree with the ones given before.

Remark 1. This section deals with differential fractions $F = P/Q$. However, the definitions stated below do not require F to be reduced, i.e., P and Q to be relatively prime.

Definition 1. Let F be a non-numeric differential fraction. The *leading variable* or *leading derivative* of F is defined as the highest variable v such that $\partial F/\partial v \neq 0$. It is denoted by $\mathrm{ld}(F)$.

Proposition 1. *Let $F = P/Q$ be a non-numeric differential fraction. If P and Q have distinct leading derivatives or, if P or Q is numeric, then $\mathrm{ld}(F)$ is the highest variable v such that $\deg(P, v) > 0$ or $\deg(Q, v) > 0$.*

Proof. First observe $\partial F/\partial w = 0$ for any $w > v$. It is thus sufficient to show that $\partial F/\partial v \neq 0$. Indeed, since v does not occur in both the numerator and the denominator of F, we have $\partial F/\partial v = (\partial P/\partial v)/Q$ or $\partial F/\partial v = -P\,(\partial Q/\partial v)/Q^2$. In each case, the corresponding fraction is nonzero. □

Definition 2. The *separant* of a non-numeric differential fraction F is defined as $\partial F/\partial v$, where $v = \mathrm{ld}(F)$.

Definition 3. Let $F = P/Q$ be a non-numeric differential fraction and write $v = \mathrm{ld}(F)$. The *degree* of F is defined as $\deg(F) = \deg(P, v) - \deg(Q, v)$. The *rank* of F is defined as the pair $(v, \deg(F))$.

Definition 4. A rank (v, d) is said to be lower than a rank (w, e) if $v < w$ or if $v = w$ and $d < e$.

The above definitions are a bit more complicated than in the polynomial case, because of differential fractions of degree 0. In particular, we wish to distinguish ranks $(v, 0)$ and $(w, 0)$, which allows us to state the following proposition.

Proposition 2. *If F is a non-numeric differential fraction, then the separant of F is either numeric or has lower rank than F.*

Proof. Assume the separant non-numeric. If its leading derivative is different from the leading derivative v of F, then it is lower than v and the Proposition is clear. Otherwise, the degree in v of the separant is less than or equal to $\deg(P, v) - \deg(Q, v) - 1$ and the Proposition is proved. □

Recall that if a polynomial P does not depend on a variable v, then $\mathrm{lcoeff}(P, v) = P$.

Definition 5. The *initial* of a non-numeric differential fraction $F = P/Q$ is defined as $\mathrm{lcoeff}(P, v)/\mathrm{lcoeff}(Q, v)$, where $v = \mathrm{ld}(F)$.

Proposition 3. *Let F be a differential fraction which is not a coefficient, $v = \mathrm{ld}(F)$ and δ be a derivation. Then the leading derivative of $\delta\,F$ is $\delta\,v$, this derivative occurs in the numerator of $\delta\,F$ only, with degree 1, and the initial of $\delta\,F$ is the separant of F.*

Proof. The first claim comes from the axioms of rankings. The two other ones are clear. □

3. MAIN RESULT

In this section, we write numer(F) and denom(F) for the numerator and denominator of a differential fraction F, both viewed as differential polynomials of the ring $\mathscr{R}$. Our result is Algorithm 3. It relies on two sub-algorithms (Algorithms 1 and 2), which are purely algebraic (i.e., they do not make use of derivations, in the sense of the differential algebra). These two sub-algorithms are related to the integration problem of rational fractions. They are either known or very close to known methods, such as those described in [7]. We state them in this paper, because our current implementation is actually based on them, the paper becomes self-contained, and the tools required by Algorithm 3 appear clearly.

Remark 2. In the three presented algorithms, all differential fractions are supposed to be reduced.

Algorithm 1 The prepareForIntegration algorithm

Require: F is a reduced differential fraction, v is a variable
Ensure: Three polynomials cont_F, N, B satisfying Prop. 4

1: **if** denom(F) is numeric **then**
2: $\quad \mathrm{cont}_F,\ N,\ B := \mathrm{denom}(F),\ \mathrm{numer}(F),\ 1$
3: **else**
4: $\quad \mathrm{cont}_F :=$ the gcd of all coeffs of denom(F) w.r.t. v { all gcd are of multivariate polynomials }
5: $\quad P_0 := \mathrm{numer}(F)$
6: $\quad Q_0 := \mathrm{denom}(F)/\mathrm{cont}_F$ { all divisions are exact }
7: $\quad A_0 := \gcd(Q_0, \mathrm{separant}(Q_0, v))$
8: $\quad B_0 := Q_0/A_0$
9: $\quad C_0 := \gcd(A_0, B_0)$
10: $\quad D_0 := B_0/C_0$
11: $\quad A_1 := \gcd(A_0, \mathrm{separant}(A_0, v))$
12: $\quad N,\ B := P_0 \cdot D_0 \cdot A_1,\ D_0 \cdot A_0$
13: **end if**
14: **return** cont_F, N, B

Proposition 4 (Specification of Algorithm 1). *Let $F = P_0/(\mathrm{cont}_F\, Q_0)$ be a differential fraction (P_0 and Q_0 being relatively prime, Q_0 primitive w.r.t. v, and cont_F denoting the content of* denom(F) *w.r.t. v). Then the polynomials returned by Algorithm 1 satisfy*

$$F = \frac{N}{\mathrm{cont}_F\, B^2}\,, \quad B = F_1\, F_2\, F_3^2 \cdots F_n^{n-1}\,,$$

where $Q_0 = F_1\, F_2^2\, F_3^3 \cdots F_n^n$ is the squarefree factorization of Q_0 w.r.t. v.

Proof. The case of F being a polynomial is clear. Assume F has a non-numeric denominator. We have $A_0 = F_2\, F_3^2 \cdots F_n^{n-1}$ at Line 8, $B_0 = F_1\, F_2\, F_3 \cdots F_n$ at Line 9, $C_0 = F_2\, F_3 \cdots F_n$ at Line 10, $D_0 = F_1$ at Line 11, $A_1 = F_3\, F_4^2 \cdots F_n^{n-2}$ at Line 12, $N = P_0\, F_1\, F_3\, F_4^2 \cdots F_n^{n-2}$ and $B = F_1\, F_2\, F_3^2 \cdots F_n^{n-1}$. $\square$

The following Lemma, which is easy to see, establishes the relationship between a well-known fact on the integration of rational fractions and Algorithm 1.

Lemma 1. *With the same notations as in Proposition 4, if there exists a reduced differential fraction R such that $F =$* separant(R, v)*, then $F_1 = 1$ and the denominator of R is $\mathrm{cont}_R\, B$, where cont_R has degree 0 in v.*

Algorithm 2 The integrateWithRemainder algorithm

Require: F_0 is a differential fraction, v is a variable
Ensure: Two differential fractions R and W such that

1. $F_0 = \mathrm{separant}(R, v) + W$
2. W is zero iff there exists R s.t. $F_0 = \mathrm{separant}(R, v)$

1: $R,\ W := 0,\ 0$
2: $F := F_0$
3: **while** $F \neq 0$ **do**
4: $\quad$ { invariant: $F_0 = F + \mathrm{separant}(R, v) + W$ }
5: $\quad \mathrm{cont}_F,\ N,\ B :=$ prepareForIntegration(F, v)
6: $\quad$ **if** $\deg(B, v) = 0$ **then**
7: $\quad\quad P :=$ the primitive of F w.r.t. v, with a 0 int. cst. { this amounts to integrate a polynomial }
8: $\quad\quad R := R + P$
9: $\quad\quad F := 0$
10: $\quad$ **else**
11: $\quad\quad$ { look for A such that $\mathrm{separant}(A/(\mathrm{cont}_R\, B), v) = N/(\mathrm{cont}_F\, B^2)$ }
12: $\quad\quad c_B,\ c_N := \mathrm{lcoeff}(B, v),\ \mathrm{lcoeff}(N, v)$
13: $\quad\quad d_B,\ d_N,\ \bar{d}_A := \deg(B, v),\ \deg(N, v),\ d_N - d_B + 1$
14: $\quad\quad$ **if** $d_N = 2\, d_B - 1$ or $\bar{d}_A < 0$ **then**
15: $\quad\quad\quad H := c_N\, v^{d_N}$
16: $\quad\quad\quad W := W + H/(\mathrm{cont}_F\, B^2)$
17: $\quad\quad\quad F := F - H/(\mathrm{cont}_F\, B^2)$
18: $\quad\quad$ **else**
19: $\quad\quad\quad R_2 := c_N\, v^{\bar{d}_A}/((\bar{d}_A - d_B)\, \mathrm{cont}_F\, c_B\, B)$
20: $\quad\quad\quad R := R + R_2$
21: $\quad\quad\quad F := F - \mathrm{separant}(R_2, v)$
22: $\quad\quad$ **end if**
23: $\quad$ **end if**
24: **end while**
25: **return** R, W

Remark 3. Algorithm 2 relies on Algorithm 1 for computing cont_F, N and B. However, it does not need N. It only needs its leading coefficient c_N and its degree d_N. Our formulation improves the readability of our algorithm.

Proposition 5. *Algorithm 2 is correct.*

Proof. First suppose Algorithm 2 terminates.

The loop invariant stated in the algorithm is clear, since it is satisfied at the beginning of the first loop and maintained in each of the three cases considered by the algorithm. Combined with the loop condition, this implies that the first ensured condition is satisfied at the end of the algorithm.

Let us now show that the second ensured condition is satisfied. We assume that

$$\exists\, R \text{ s.t.} \quad F \quad = \quad \mathrm{separant}(R, v)\,. \tag{2}$$

We prove that Lines 15–17 are not performed. This is sufficient to prove the second ensured condition since, if Lines 15–17 are not performed and (2) holds then, after one iteration, F is modified either at Line 9 or 21. In both cases, the new value of F satisfies (2) again.

Assume (2) holds. By Lemma 1, the denominator of R is $\mathrm{cont}_R\, B$. Let $A = c_A\, v^{d_A} + q_A$ be its numerator and $B = c_B\, v^{d_B} + q_B$ (we only need to consider the case $d_B > 0$). One can assume that $d_A \neq d_B$. Indeed, if $d_A = d_B$, one can take $\bar{R} = (A - c_A/c_B\, B)/B$ since $\mathrm{separant}(\bar{R}, v) =$

separant$(R, v) = F$ and $\deg(A - c_A/c_B\, B) < d_B$. Differentiating the fraction $A/(\mathrm{cont}_R\, B)$ w.r.t. v and identifying with $F = N/(\mathrm{cont}_F\, B^2)$ (Proposition 4), we see that $N = (d_A - d_B)\, c_A\, c_B\, \mathrm{cont}_F/\mathrm{cont}_R\, v^{d_A+d_B-1} + \cdots$ where the dots hide terms of degree, in v, less than $d_A + d_B - 1$. Since $d_A \neq d_B$, the degree of N satisfies $d_N = d_A + d_B - 1$, thus $d_N \neq 2\, d_B - 1$. Moreover, the variable $\bar{d}_A$ is equal to d_A since $d_N - d_B + 1 = d_A$. Summarizing, $d_N \neq 2\, d_B - 1$ and $\bar{d}_A \geq 0$, so Lines 15–17 are not performed.

Termination of Algorithm 2 follows from the fact that the degrees of N and B, in v, decrease (strictly, in the case of N). Indeed, at each iteration, one of the Lines 9, 17 or 21 is performed. If Line 9 is performed, then the algorithm stops immediately. The key observation is that, at Line 21, the fraction R_2 is chosen such that

$$\frac{\partial R_2}{\partial v} = \frac{c_N\, v^{d_N} + \cdots}{\mathrm{cont}_F\, B^2}\,,$$

where the dots hide terms of degree, in v, less than d_N. Thus, if Line 17 or 21 is performed, the algorithm subtracts from F a fraction which admits $\mathrm{cont}_F\, B^2$ as a denominator and the degree of N decreases strictly. If the numerator of the new fraction has a common factor with B, this can only decrease the degrees of both the numerator and the denominator. □

PROPOSITION 6. *Algorithm 3 is correct.*

PROOF. Termination is guaranteed, essentially, by the fact that rankings are well-orderings. Here are a few more details. The algorithm considers eight cases. In Cases 1, 2, 3 and 7, F is assigned 0. In Cases 4 and 8, the new value of F has lower leading derivative than the old one. In Case 5, the new value of F has lower rank than the old one. In Case 6, the rank of F does not necessarily change (if it does, it decreases) but, at the next iteration, another case than 6 will be entered.

The loop invariant stated in the algorithm is clear, since it is satisfied at the beginning of the loop and maintained in each of the eight cases. Combined with the loop condition, it implies that, at the end of the algorithm, the first ensured condition is satisfied.

Let us now address the second ensured condition. The implication from left to right is clear, so we must show that W is zero if F_0 is a total derivative. Since derivations commute with sums, it suffices to prove that W remains zero in each pass through the main loop as long as F is a total derivative. Hence assume $F = \delta_x\, G$ for a differential fraction G. If G is a coefficient, so is F, which is handled by Case 1; the ensured condition is then guaranteed by the properties of Algorithm 2 (Proposition 5). Now assume G is not a coefficient and let $v \in \Theta\mathscr{U}$ be its leading derivative. By the axioms of rankings, the variable $v_x = \delta_x\, v$ is the leading derivative of F, it has positive order w.r.t. x, degree 1, and we have

$$F = \frac{\partial G}{\partial v}\, v_x + \cdots\,,$$

where the dots hide terms that do not depend on v_x. The initial of F is thus the separant of G, and it depends on variables less than or equal to v. Writing $F = P/Q$, it is clear that v_x occurs only in P and so must coincide with v_N. Hence Cases 2–5 are excluded, and we have $F_2 = \partial G/\partial v$. Since the latter cannot involve variables greater than v, also

Algorithm 3 The integrate algorithm

Require: F_0 is a differential fraction, x is an independent variable

Ensure: Two differential fractions R and W such that

1. $F_0 = \delta_x\, R + W$, where δ_x is the derivation w.r.t. x
2. W is zero iff there exists R such that $F_0 = \delta_x\, R$
3. Unless F_0 is a coefficient, $\delta_x\, R$ and W have ranks lower than or equal to F_0

$R, W := 0, 0$
$F := F_0$
while $F \neq 0$ **do**
  { Invariant: $F_0 = F + \delta_x\, R + W$ }
  if F is a coefficient **then**
    $R_2, W_2 :=$ integrateWithRemainder(F, x)
    $R := R + R_2$ { Case 1 }
    $W := W + W_2$
    $F := 0$
  else if numer(F) is a coefficient **then**
    $W := W + F$ { Case 2 }
    $F := 0$
  else
    denote v_N the leading derivative of numer(F)
    denote v_B the one of denom(F) (if not a coefficient)
    if denom(F) is not a coefficient and $v_N \leq v_B$ **then**
      $W := W + F$ { Case 3 }
      $F := 0$
    else if v_N has order zero w.r.t. x **then**
      view numer(F) as a sum of monomials m_i
      denote H the sum of the m_i s.t. $\deg(m_i, v_N) > 0$
      $W := W + H/\mathrm{denom}(F)$ { Case 4 }
      $F := F - H/\mathrm{denom}(F)$
    else if $\deg(\mathrm{numer}(F), v_N) \geq 2$ **then**
      view numer(F) as a sum of monomials m_i
      denote H the sum of the m_i s.t. $\deg(m_i, v_N) \geq 2$
      $W := W + H/\mathrm{denom}(F)$ { Case 5 }
      $F := F - H/\mathrm{denom}(F)$
    else
      { we have: $\deg(\mathrm{numer}(F), v_N) = 1$ }
      let v be such that $\delta_x\, v = v_N$
      $F_2 := \mathrm{lcoeff}(\mathrm{numer}(F), v_N)/\mathrm{denom}(F)$
        {recall F_2 is supposed to be reduced}
      if $\exists\, w > v$ such that $\deg(\mathrm{numer}(F_2), w) > 0$ **then**
        view numer(F_2) as a sum of monomials m_i
        denote H the sum of the m_i such that for some $w > v$, we have $\deg(m_i, w) > 0$
        $W := W + (H/\mathrm{denom}(F_2)) \cdot v_N$ { Case 6 }
        $F := F - (H/\mathrm{denom}(F_2)) \cdot v_N$
      else if $\exists\, w > v$ s.t. $\deg(\mathrm{denom}(F_2), w) > 0$ **then**
        $W := W + F$ { Case 7 }
        $F := 0$
      else
        $R_2, W_2 :=$ integrateWithRemainder(F_2, v)
        $W := W + W_2 \cdot v_N$ { Case 8 }
        $R := R + R_2$
        $F := F - \delta_x\, R_2 - W_2 \cdot v_N$
      end if
    end if
  end if
end while
return W, R

Cases 6 and 7 are excluded. The remaining Case 8 is again handled by the properties of Algorithm 2 (Proposition 5).

Let us address the third ensured condition. Assume this condition is satisfied by R and W at the beginning of some loop. In Case 1, unless F_0 is a coefficient, R_2 and W_2 are assigned coefficients and have thus lower rank than F_0. The third condition is thus satisfied again. Cases 2–7 are clear. In Case 8, we show that the contribution $\delta_x R_2$ to the new value of F does not increase the rank of F. Recall that v_N, which is the leading derivative of numer(F), is also the leading derivative of F (by virtue of Case 3). Moreover, there exists a derivative v such that $\delta_x v = v_N$. An increase of the rank of F could only happen if F_2 involved derivatives w such that $\delta_x w > v_N$ and hence $w > v$ by the axioms of rankings. This situation is however impossible, due to Cases 6 and 7. Thus the third ensured condition is satisfied, and the Proposition is proved. $\square$

Remark 4. The second ensured condition of Algorithm 3 could be made stronger. Indeed, the algorithm does not only ensure that W is zero whenever it is possible, it also makes W as small as possible, storing in this variable at each iteration, a "small part" of F that cannot be integrated (Cases 5 and 6).

In the next section, we use an "iterated" version of Algorithm 3, stated in Algorithm 4.

Algorithm 4 The $\mathsf{integrate}$ algorithm (iterated version)

Require: F_0 is a differential fraction which is not a coefficient, x is an independent variable

Ensure: A possibly empty list $[W_0, W_1, \ldots, W_t]$ of differential fractions such that

1. W_t is nonzero
2. $F_0 = W_0 + \delta_x W_1 + \cdots + \delta_x^t W_t$
3. $W_0, W_1, \ldots, W_i$ are zero if, and only if there exists R such that $F_0 = \delta_x^{i+1} R$
4. The differential fractions $W_0, \delta_x W_1, \ldots, \delta_x^t W_t$ have ranks lower than or equal to F_0

$L :=$ the empty list
$R := F_0$
while R is not a coefficient **do**
  $W, R := \mathsf{integrate}(R, x)$
  append W at the end of L
end while
if $R \neq 0$ **then**
  append R at the end of L
end if
return L

Proposition 7. *Algorithm 4 is correct.*

Proof. The first ensured condition is clear from the code. The other ones follow the specifications of Algorithm 3. The number of iterations, t, is bounded by the total order of F_0. $\square$

4. IMPLEMENTATION AND EXAMPLES

A first version of Algorithm 3 was implemented in Maple. More recently, this algorithm was implemented in C, within the BLAD libraries, version 3.10. See [3, `bap` library, file `bap_rat_bilge_mpz.c`]. The following computations were performed by the C version, through a testing version[2] of the Maple `DifferentialAlgebra` package [5].

Example 1. The first example shows that Algorithm 3 permits to decide if a differential fraction F is the derivative of some other differential fraction G. Observe that this test was already implemented in Maple, by the `DEtools[firint]` function.

The variable *Ring* receives a description of the differential ranking. The variable G receives a differential fraction, F receives its derivative w.r.t. x.

```
> with (DifferentialAlgebra):
> with (Tools):
> integrate := DifferentialAlgebra0:-Integrate:
> Ring := DifferentialRing
            (derivations = [x,y],
             blocks = [[u,v],w]);
              Ring := differential_ring

> G := u[x]^2 + w[y]/w^2 + w[x,x,y];

                           2   w[y]
               G := u[x]  + ---- + w[x, x, y]
                                2
                               w
> F := Differentiate (G, x, Ring);
                         4                   4
F := (2 u[x, x] u[x] w  + w[x, x, x, y] w

             2                      /  4
     + w[x, y] w  - 2 w[x] w[y] w)  /  w
                                   /
```

Algorithm 3, implemented here using the name `integrate`, is applied to F and x. It returns the list $[W, R]$. We get $W = 0$, indicating that $F = \delta_x R$.

```
> L := integrate (F, x, Ring, iterated=false);
                     2  2                       2
                 u[x]  w  + w[y] + w[x, x, y] w
        L := [0, -------------------------------]
                               2
                              w

> normal (L[2] - G);
                          0
```

If the optional parameter `iterated` is left to its default value (true), Algorithm 4 is called.

```
> L := integrate (F, x, Ring);
                         2  2
                     u[x]  w  + w[y]
            L := [0, ---------------, 0, w[y]]
                            2
                           w
```

Example 2. This variant of the previous example shows the differences between `DEtools[firint]` and Algorithm 3. Since `DEtools[firint]` does not handle PDE, we switch to ODE and to the standard Maple `diff` notation.

[2]This testing version, called `DifferentialAlgebra0`, is available at [4].

```
> Ring := DifferentialRing
             (derivations = [x],
              blocks = [[u,v],w]);
               Ring := differential_ring

> G := diff (u(x),x)^2 + v(x)/w(x)^2 + diff(v(x),x,x);
                                            / 2      \
              /d      \2   v(x)    |d        |
         G := |-- u(x)|  + ----- + |--- v(x)|
              \dx     /        2   |  2      |
                           w(x)    \dx      /

>  F := diff(G,x):

> DEtools[firint](F, u(x));
                                / 2      \
      /d      \2   v(x)    |d        |
      |-- u(x)|  + ----- + |--- v(x)| + _C1 = 0
      \dx     /        2   |  2      |
                   w(x)    \dx      /
```

Both methods recognize that F is a total derivative.

```
> integrate (F, x, Ring, notation=diff,
                         iterated=false);
       / 2      \
       |d        |     2   /d      \2     2
       |--- v(x)| w(x)  + |-- u(x)|  w(x)  + v(x)
       |  2      |          \dx     /
       \dx      /
   [0, -----------------------------------------]
                                  2
                              w(x)
```

However, if one adds a term $u(x)$ to F, one gets a differential fraction which is not a total derivative (which is not *exact*, in `DEtools[firint]` terminology). Our algorithm produces a decomposition while `DEtools[firint]` gives up (which is its expected behaviour).

```
> DEtools[firint](F+u(x), u(x));
Error, (in ODEtools/firint) the given ODE is not exact
> integrate (F+u(x), x, Ring, notation=diff,
                              iterated=false);
         / 2      \
         |d        |     2   /d      \2     2
         |--- v(x)| w(x)  + |-- u(x)|  w(x)  + v(x)
         |  2      |          \dx     /
         \dx      /
  [u(x), -----------------------------------------]
                                    2
                                w(x)
```

Example 3. The following example illustrates Algorithm 4 on differential polynomials. This very simple example shows that the result of Algorithm 3 is ranking dependent. Any derivative of u is greater than any derivative of w. Thus we get

$$u_x\, w_x = \delta_x(u\, w_x) - u\, w_{xx}\,.$$

```
> integrate (u[x]*w[x], x, Ring);

                   [-u w[x, x], u w[x]]
```

However, the ranking between u and v is orderly, so that $v_{xx} > u_x$. For this reason, Algorithm 3 behaves differently.

```
> integrate (u[x]*v[x], x, Ring);

                       [u[x] v[x]]
```

Example 4. The following example shows that Algorithm 3 does not commute with sums. The issue is related to the existence of simple factors in the squarefree decompositions of denominators. For readability, results are displayed in factored form.

```
> F1 := u[x]/(u+1) - u[x]/(u+2)^2;
                       u[x]      u[x]
                F1 := ----- - --------
                       u + 1          2
                               (u + 2)

> F2 := -u[x]/(u+1) + u[x]/(u+3)^2;
                        u[x]      u[x]
              F2 := - ----- + --------
                        u + 1          2
                                (u + 3)

> F3 := F1+F2;
                        u[x]       u[x]
          F3 := - -------- + --------
                          2           2
                   (u + 2)    (u + 3)

> L1 := factor (
        integrate (F1, x, Ring, iterated=false));
                          3
            u[x] (u  + 2)          4 u + 3
     L1 := [-----------------, - ---------------]
                    2        2    (u + 2) (u + 1)
             (u + 2)  (u + 1)

> L2 := factor (
        integrate (F2, x, Ring, iterated=false));
                             3
                 u[x] u            12 u + 13
    L2 := [- -----------------, -----------------]
                     2        2  2 (u + 3) (u + 1)
              (u + 3)  (u + 1)

> L3 := factor (
        integrate (F3, x, Ring, iterated=false));
                                1
            L3 := [0, ---------------]
                       (u + 3) (u + 2)

> L12 := factor (L1+L2);
                 4      3      2
        u[x] (2 u  + 5 u  + 2 u  + 12 u + 18)
L12 := [-------------------------------------,
                     2         2         2
              (u + 3)  (u + 1)  (u + 2)


           2
        4 u  + 7 u + 8
    -------------------------]
    2 (u + 3) (u + 1) (u + 2)
```

Though $L_{12} \neq L_3$, it is possible to recover L_3 from L_{12} by applying Algorithm 3 once more.

```
> L := factor (
    integrate (L12[1], x, Ring, iterated=false));
                                 2
                              4 u  + 5 u + 6
         L := [0, - -------------------------]
                     2 (u + 3) (u + 1) (u + 2)

> factor (L3 - (L + [0, L12[2]]));

                            [0, 0]
```

Example 5. This example, inspired from [9], illustrates the usefulness of Algorithm 4 for simplifying equations produced by differential elimination methods. It features a compartmental model with two compartments, 1 and 2, with a same unit volume. There are two state variables x_1 and x_2 (one per compartment). State variable x_i represents the concentration of some drug in compartment i. The exchanges between the two compartments are supposed to follow linear laws (depending on parameters k_1 and k_2). The drug is supposed to exit from the model, from compartment 1, following a Michaelis-Menten type law (depending on parameters V_e, k_e). The output of the system, denoted y, is the state variable x_1.

```
> Ring := DifferentialRing (
     derivations = [t],
     blocks = [y,[x1,x2],[k1(),k2(),ke(),Ve()]]);

              Ring := differential_ring

> S := [ x1[t] = -k1*x1 + k2*x2 - (Ve*x1)/(ke+x1),
        x2[t] = k1*x1 - k2*x2,
            y = x1];

                                     Ve x1
S := [x1[t] = -k1 x1 + k2 x2 - -------,
                                    ke + x1

    x2[t] = k1 x1 - k2 x2, y = x1]
```

By means of differential elimination [6], one computes the so-called input-output equation of the system (though there is no input).

```
> ideal := RosenfeldGroebner
     (S, Ring,
      basefield =
          field (generators = [k1,k2,ke,Ve]));

        ideal := [regular_differential_chain]

> io_ideal := RosenfeldGroebner
     (ideal[1],
      blocks = [[x1,x2],y,[k1(),k2(),ke(),Ve()]]):
> io_eq := Equations
     (io_ideal, leader = derivative(y))[1];

                        2                             2
io_eq := y[t, t] y  + 2 y[t, t] y ke + y[t, t] ke

                2           2
     + y[t] y  k1 + y[t] y  k2 + 2 y[t] y k1 ke

                                        2            2
     + 2 y[t] y k2 ke + y[t] k1 ke  + y[t] k2 ke

                          2
     + y[t] ke Ve + y  k2 Ve + y k2 ke Ve
```

Using Algorithm 3, we obtain

```
> integrate (io_eq / Initial (io_eq, Ring), t, Ring);

           2       2           2        2
 y k2 Ve   y  k1 + y  k2 - k1 ke  - k2 ke  - ke Ve
[-------, ---------------------------------------, y]
 y + ke                    y + ke
```

This list can be translated into the following equation, whose structure is now much clearer:

$$\frac{y(t)\,k_2\,V_e}{y(t)+k_e} + \frac{\mathrm{d}}{\mathrm{d}t}\,\frac{(y(t)^2 - k_e^2)\,(k_1+k_2) - k_e\,V_e}{y(t)+k_e} + \frac{\mathrm{d}^2}{\mathrm{d}t^2}\,y(t)\,.$$

This example also shows that integrating differential fractions (as we did above), may yield better formulas than integrating differential polynomials. Indeed:

```
> integrate (io_eq, t, Ring);

          2             2       2
[-2 y[t]  y - 2 y[t]  ke + y  k2 Ve + y k2 ke Ve,

     3       3
    y  k1   y  k2    2           2                2
    ----- + ----- + y  k1 ke + y  k2 ke + y k1 ke
      3       3

                2                  3    2           2
     + y k2 ke  + y ke Ve, 1/3 y  + y  ke + y ke ]
```

While the fractional equation is simpler than that from an intuitive point of view, it would be interesting to investigate also their numerical properties from a more systematic viewpoint. In the case of differential-algebraic equations, the usual elimination methods are known to produce typically very lengthy polynomial differential equations whose numerical treatment may be more costly than that of the corresponding fractional differential equation.

Remark 5. The four parameters were placed at the bottom of the ranking, following the assumptions stated in Section 2. However, the `DifferentialAlgebra` package, which handles parameters as differential indeterminates constrained by implicit equations (their derivatives are zero), does not require parameters to lie at the bottom of rankings. With such rankings, the second ensured condition of Algorithm 3 would not hold anymore.

5. CONCLUSION

The algorithm presented in this paper may be extended in many different ways: to handle more complicated differential operators such as $\delta_x + \delta_y$, possibly with differential polynomials for coefficients, for instance. Some prototypes are currently under study.

In this paper, we have not addressed the interest of the third ensured condition of Algorithm 3. This topic will be covered in a further paper. It is related to the issue of characterizing the differential fractions which are returned "as is" by our method.

As mentioned in the Introduction, the decomposition of a differential polynomial or a differential fraction into a total derivative and a remainder is crucial for constructing the ring of integro-differential polynomials. Formalizing this idea in the general category of commutative integro-differential algebras over rings leads to the notion of quasi-antiderivative [10]. For ordinary differential polynomials in one indeterminate and for univariate rational functions, a quasi-antiderivative is exhibited in [10] but the case of partial differential polynomials or differential fractions remains open. While Algorithm 3 comes close to providing such a quasi-antiderivative, it falls short of being linear (Example 4). This question will be addressed in a future paper. It would allow us to make progress towards our original goal: defining and computing Gröbner bases for ideals of integro-differential polynomials of various kinds.

6. ACKNOWLEDGMENTS

The authors thank the referees for their detailed comments, suggestions, and references.

7. REFERENCES

[1] A. H. Bilge. A REDUCE program for the integration of differential polynomials. *Computer Physics Communications*, 71:263–268, 1992.

[2] G. W. Bluman and S. C. Anco. *Symmetry and Integration Methods for Differential Equations*. Springer Verlag, New York, 2002.

[3] F. Boulier. The BLAD libraries. `http://www.lifl.fr/~boulier/BLAD`, 2004.

[4] F. Boulier. The BMI library. `http://www.lifl.fr/~boulier/BMI`, 2010.

[5] F. Boulier and E. Cheb-Terrab. Help Pages of the DifferentialAlgebra Package. In MAPLE 14, 2010.

[6] F. Boulier, F. Lemaire, and M. Moreno Maza. Computing differential characteristic sets by change of ordering. *Journal of Symbolic Computation*, 45(1):124–149, 2010.

[7] M. Bronstein. *Symbolic Integration I*. Springer Verlag, Berlin, Heidelberg, New York, 1997.

[8] G. H. Campbell. Symbolic integration of expressions involving unspecified functions. *SIGSAM Bulletin*, 22(1):25–27, 1988.

[9] L. Denis-Vidal, G. Joly-Blanchard, and C. Noiret. System identifiability (symbolic computation) and parameter estimation (numerical computation). In *Numerical Algorithms*, volume 34, pages 282–292, 2003.

[10] L. Guo, G. Regensburger, and M. Rosenkranz. On integro-differential algebras. Preprint available at `http://arxiv.org/abs/1212.0266`., 2012.

[11] E. R. Kolchin. *Differential Algebra and Algebraic Groups*. Academic Press, New York, 1973.

[12] C. G. Raab. *Definite Integration in Differential Fields*. PhD thesis, University of Linz, 2012.

[13] J. F. Ritt. *Differential Algebra*, volume 33 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, New York, 1950.

[14] M. Rosenkranz and G. Regensburger. Integro-differential polynomials and operators. In *ISSAC'08: Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 261–268, New York, NY, USA, 2008. ACM.

[15] F. Schwarz. An algorithm for determining polynomial first integrals of autonomous systems of ordinary differential equations. *Journal of Symbolic Computation*, 1(2):229–233, 1985.

[16] W. Y. Sit. On Goldman's algorithm for solving first-order multinomial autonomous systems. In *Proceedings of the 6th International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, AAECC-6, pages 386–395, London, UK, 1989. Springer-Verlag.