Finding First Integrals Using Normal Forms Modulo Differential Regular Chains

François Boulier and François Lemaire

Univ. Lille, CRIStAL, UMR 9189, 59650 Villeneuve d'Ascq, France
{francois.boulier,francois.lemaire}@univ-lille1.fr

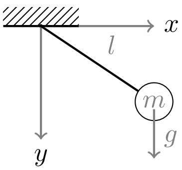
Abstract. This paper introduces a definition of polynomial first integrals in the differential algebra context and an algorithm for computing them. The method has been coded in the Maple computer algebra system and is illustrated on the pendulum and the Lotka-Volterra equations. Our algorithm amounts to finding linear dependences of rational fractions, which is solved by evaluation techniques.

Keywords: First integral, linear algebra, differential algebra, nonlinear system.

1 Introduction

This paper deals with the computation of polynomial first integrals of systems of ODEs, where the independent variable is t (for time). A first integral is a function whose value is constant over time along every solution of a system of ODEs. First integrals are useful to understand the structure of systems of ODEs. A well known example of first integral is the energy of a mechanical conservative system, as shown by Example 1.

Example 1. Using a Lagrange multiplier $\lambda(t)$, a pendulum of fixed length l, with a centered mass m submitted to gravity g can be coded by:



$$\Sigma \begin{cases} m\,\ddot{x}(t) & = \lambda(t)\,x(t) \\ m\,\ddot{y}(t) & = \lambda(t)\,y(t) + mg \\ x(t)^2 + y(t)^2 & = l^2. \end{cases} \quad (1)$$

A trivial first integral is $x(t)^2 + y(t)^2$ since $x(t)^2 + y(t)^2 = l^2$ on any solution. A less trivial first integral is $\frac{m}{2}\left(\dot{x}(t)^2 + \dot{y}(t)^2\right) - mg\,y(t)$ which corresponds to the total energy of the system (kinetic energy + potential energy).

When no physical considerations can be used, one needs alternative methods. Assume we want to compute a polynomial first integral of a system of ODEs. Our algorithm findAllFirstIntegrals, which has been coded in Maple, relies on the following strategy. We choose a certain number of monomials $\mu_1, \mu_2, \ldots, \mu_e$ built

V.P. Gerdt et al. (Eds.): CASC 2015, LNCS 9301, pp. 101–118, 2015.
DOI: 10.1007/978-3-319-24021-3_8

over t, the unknown functions and their derivatives (namely t, $x(t)$, $y(t)$, $\dot{x}(t)$, $\dot{y}(t)$, ... on Example 1), and look for a candidate of the form $q = \alpha_1\mu_1 + \cdots + \alpha_e\mu_e$ satisfying $\frac{\mathrm{d}q(t)}{\mathrm{d}t} = 0$ for all solutions, where the α_i are in some field $\mathbb{K}$. If the α_i are constant (i.e. $\frac{\mathrm{d}\alpha_i}{\mathrm{d}t} = 0$ for each α_i), then our problem amounts to finding α_i such that $\frac{\mathrm{d}q(t)}{\mathrm{d}t} = \alpha_1\frac{\mathrm{d}\mu_1}{\mathrm{d}t} + \cdots + \alpha_e\frac{\mathrm{d}\mu_e}{\mathrm{d}t}$ is zero for all solutions. On Example 1, μ_1, μ_2 and μ_3 could be the monomials $\dot{x}(t)^2$, $\dot{y}(t)^2$ and $y(t)$ and α_1, α_2 and α_3 could be $m/2$, $m/2$ and $-mg$.

Anticipating Section 4, differential algebra techniques combined with the Nullstellensatz Theorem permit to check that a polynomial p vanishes on all solutions of a radical ideal $\mathfrak{A}$ by checking that p belongs to $\mathfrak{A}$. Moreover, in the case where the ideal $\mathfrak{A}$ is represented[1] by a list of differential regular chains $M = [C_1, \ldots, C_f]$, checking that $p \in \mathfrak{A}$ can be done by checking that the normal form (see Section 4) of p modulo the list of differential regular chains M, denoted $\mathsf{NF}(p, M)$, is zero. Since the normal form is $\mathbb{K}$-linear (see Section 4), finding first integrals can be solved by finding a linear dependence between $\mathsf{NF}(\frac{\mathrm{d}\mu_1}{\mathrm{d}t}, M)$, $\mathsf{NF}(\frac{\mathrm{d}\mu_2}{\mathrm{d}t}, M)$, ..., $\mathsf{NF}(\frac{\mathrm{d}\mu_e}{\mathrm{d}t}, M)$.

This process, which is in the spirit of [3,8], encounters a difficulty: the normal forms usually involve fractions. As a consequence, we need a method for finding linear dependences between rational functions. A possible approch consists in reducing all fractions to the same denominator, and solving a linear system based on the monomial structure of the numerators. However, this has disadvantages. First, the sizes of the numerators might grow if the denominators are large. Second, the linear system built above is completely modified if one considers a new fraction. Finding linear dependences between rational functions is addressed in Sections 2 and 3 by evaluating the variables occurring on the denominators, thus transforming the fractions into polynomials. The main difficulty with this process is to choose adequate evaluations to obtain deterministic (i.e. non probabilistic) and terminating algorithms.

This paper is structured as follows. Section 2 presents some lemmas around evaluation and Algorithm findKernelBasis (and its variant findKernelBasisMatrix) which computes a basis of the linear dependences of some given rational functions. Section 3 presents Algorithm incrementalFindDependence (and its variant incrementalFindDependenceLU) which sequentially treats the fractions and stops when a linear dependence is detected. Section 4 recalls some differential algebra techniques and presents Algorithm findAllFirstIntegrals which computes a basis of first integrals which are $\mathbb{K}$-linear combinations of a predefined set of monomials.

In Sections 2 and 3, $\mathbf{k}$ is a commutative field of characteristic zero containing $\mathbb{R}$, $\mathbb{A}$ is an integral domain with unit, a commutative $\mathbf{k}$-algebra and also a $\mathbf{k}$-vector space equipped with a norm $\|\cdot\|_{\mathbb{A}}$. Moreover, $\mathbb{K}$ is the total field of fractions of $\mathbb{A}$. In Section 4, we will require, moreover, that $\mathbb{K}$ is a field of constants. In applications, $\mathbf{k}$ is usually $\mathbb{R}$, $\mathbb{A}$ is usually a commutative ring of multivariate polynomials over $\mathbf{k}$ (i.e. $\mathbb{A} = \mathbf{k}[z_1, \ldots, z_s]$) and $\mathbb{K}$ is the field of fractions of $\mathbb{A}$ (i.e. the field of rational functions $\mathbf{k}(z_1, \ldots, z_s)$).

[1] More precisely $\mathfrak{A} = [C_1] : H_{C_1}^{\infty} \cap \cdots \cap [C_f] : H_{C_f}^{\infty}$.

2 Basis of Linear Dependences of Rational Functions

This section presents Algorithms findKernelBasis and findKernelBasisMatrix which compute a basis of the linear dependences over $\mathbb{K}$ of e multivariate rational functions denoted by $q_1, \ldots, q_e$. More precisely, they compute a $\mathbb{K}$-basis of the vectors $(\alpha_1, \ldots, \alpha_e) \in \mathbb{K}^e$ satisfying $\sum_{i=1}^{e} \alpha_i q_i = 0$.

Those algorithms are mainly given for pedagogical reasons, in order to focus on the difficulties related to the evaluations (especially Lemma 2). The rational functions are taken in the ring $\mathbb{K}(Y)[X]$ where $Y = \{y_1, \ldots, y_m\}$ is a finite set of indeterminates and X is another (possibly infinite) set of indeterminates such that $Y \cap X = \emptyset$. The idea consists in evaluating the variables Y, thus reducing our problem to finding the linear dependences over $\mathbb{K}$ of polynomials in $\mathbb{K}[X]$, which can be easily solved for instance with linear algebra techniques. If enough evaluations are made, the linear dependences of the evaluated fractions coincide with the linear dependences of the non evaluated fractions.

Even if it is based on evaluations, our algorithm is not probabilistic. A possible alternative is to use [12] by writing each rational function into a (infinite) basis of multivariate rational functions. However, we chose not to use that technique because it relies on multivariate polynomial factorization into irreducible factors, and because of a possible expression swell.

2.1 Preliminary Results

This section introduces two basic definitions as well as three lemmas needed for proving Algorithm findKernelBasis.

Definition 1 (Evaluation). *Let $D = \{g_1, \ldots, g_e\}$ be a set of polynomials of $\mathbb{K}[Y]$ where $Y = \{y_1, \ldots, y_m\}$. Let $y^0 = (y_1^0, y_2^0, \ldots, y_m^0)$ be an element of $\mathbb{K}^m$, such that none of the polynomials of D vanish on y^0. One denotes σ_{y^0} the ring homomorphism from $\mathbb{K}[X, Y, g_1^{-1}, \ldots, g_e^{-1}]$ to $\mathbb{K}[X]$ defined by $\sigma_{y^0}(y_j) = y_j^0$ for $1 \le j \le m$ and $\sigma_{y^0}(x) = x$ for $x \in X$. Roughly speaking, the ring homomorphism σ_{y^0} evaluates at $y = y_0$ the rational functions whose denominator divides a product of powers of g_i.*

Definition 2 (Linear combination application). *Let E be a $\mathbb{K}$-vector space. For any vector $v = (v_1, v_2, \ldots, v_e)$ of E^e, Φ_v denotes the linear application from $\mathbb{K}^e$ to E defined by $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_e) \to \Phi_v(\alpha) = \sum_{i=1}^{e} \alpha_i v_i$.*

The notation Φ_v defined above is handy: if $q = (q_1, \ldots, q_e)$ is (for example) a vector of rational functions, then the set of the vectors $\alpha = (\alpha_1, \ldots, \alpha_n)$ in $\mathbb{K}^e$ satisfying $\sum_{i=1}^{e} \alpha_i q_i = 0$ is simply $\ker \Phi_q$.

The following Lemma 1 is basically a generalization to several variables of the classical following result: *a polynomial of degree d over an integral domain admitting $d+1$ roots is necessarily the zero polynomial.*

Lemma 1. *Let $p \in \mathbb{K}[Y]$ where $Y = \{y_1, \ldots, y_m\}$. Assume* $\deg_{y_i} p \le d$ *for all $1 \le i \le m$. Let $S_1, S_2, \ldots, S_m$ be m sets of $d+1$ points in $\mathbb{Q}$. If $p(y^0) = 0$ for all $y^0 \in S_1 \times S_2 \times \ldots \times S_m$, then p is the zero polynomial.*

Proof. By induction on m. When $m = 1$, p has $d+1$ distinct roots and a degree at most d. Since p is a polynomial over a field $\mathbb{K}$, p is the zero polynomial.

Suppose the lemma is true for m. One shows it is true for $m+1$. Seeing p as an element of $\mathbb{K}(y_1, \ldots, y_m)[y_{m+1}]$ and using the Lagrange polynomial interpolation formula [9, page 101] over the ring $\mathbb{K}(y_1, \ldots, y_m)[y_{m+1}]$, one has $p = \sum_{i=1}^{d+1} \left(p(y_1, \ldots, y_m, s_i) \prod_{j=1, j\neq i}^{d+1} \frac{y_{m+1}-s_j}{s_i - s_j} \right)$ where $S_{m+1} = \{s_1, \ldots, s_{d+1}\}$. For each $1 \leq i \leq d+1$, $p(y_1, \ldots, y_m, s_i)$ vanishes on all points of $S_1 \times S_2 \times \ldots \times S_m$. By induction, all $p(y_1, \ldots, y_m, s_i)$ are thus zero, and p is the zero polynomial. □

The following lemma is quite intuitive. If one takes a nonzero polynomial g in $\mathbb{K}[Y]$, then it is possible to find an arbitrary large "grid" of points in $\mathbb{N}^m$ where g does not vanish. This lemma will be applied later when g is the product of some denominators that we do not want to cancel.

Lemma 2. *Let g be a nonzero polynomial in $\mathbb{K}[Y]$ and d be a positive integer. There exist m sets $S_1, S_2, \ldots, S_m$, each one containing $d+1$ consecutive nonnegative integers, such that $g(y^0) \neq 0$ for all y^0 in $S = S_1 \times S_2 \times \cdots \times S_m$.*

The proof of Lemma 2 is the only one explicitly using the norm $\|\cdot\|_{\mathbb{A}}$. The proof is a bit technical but the underlying idea is simple and deserves a rough explanation. If g is nonzero, then its homogeneous part of highest degree, denoted by h, must be nonzero at some point $\bar{y}$, hence on a neighborhood of $\bar{y}$. Since g "behaves like" h at infinity (the use of $\|\cdot\|_{\mathbb{A}}$ will make precise this statement), one can scale the neighborhood to prove the lemma.

Proof. Since $\mathbb{K}$ is the field of fraction of $\mathbb{A}$, one can assume with no loss of generality that g is in $\mathbb{A}[Y]$.

Denote $g = h + p$ where h is the homogeneous part of g of (total) degree $e = \deg g$. Since, g is nonzero, so is h. By Lemma 1, there exists a point $\bar{y} \in \mathbb{R}^m_{>0}$, such that $h(\bar{y}) \neq 0$, where $\mathbb{R}_{>0}$ denotes the positive reals. Without loss of generality, one can assume that $\|\bar{y}\|_\infty = 1$ since h is homogeneous. There exists an open neighborhood V of $\bar{y}$ such that $V \subset \mathbb{R}^m_{>0}$ and $0 \notin h(V)$. Moreover, this open neighborhood V also contains a closed ball B centered at $\bar{y}$ for some positive real ϵ i.e. $B = \{y \in \mathbb{R}^m_{>0} \mid \ \|y - \bar{y}\|_\infty \leq \epsilon\} \subset V$.

Since B is a compact set and the functions h and t are continuous, the two following numbers are well-defined and finite: $m = \min_{y \in B} \|h(y)\|_{\mathbb{A}}$ and $M = \max_{y\in B}(\sum_{i\in I} \|a_i\|_{\mathbb{A}}\, m_i(y))$ (where $p = \sum_{i \in I} a_i m_i$ with $a_i \in \mathbb{A}$ and m_i is a monomial in Y). Moreover, $m > 0$ (since by a compactness argument there exists $y \in B$ such that $h(y) = m$).

Take $y \in B$ and $s > 1 \in \mathbb{R}$. Then $g(sy) = h(sy) + p(sy) = s^e h(y) + p(sy)$. By the reverse triangular inequality and the homogeneity of h, one has $\|g(sy)\|_{\mathbb{A}} \geq s^e \|h(y)\|_{\mathbb{A}} - \|p(sy)\|_{\mathbb{A}}$. Moreover

$$\|p(sy)\|_{\mathbb{A}} \leq \sum_{i\in I} \|a_i\|_{\mathbb{A}}\, m_i(sy) \leq \sum_{i \in I} \|a_i\|_{\mathbb{A}}\, s^{e-1} m_i(y) \leq s^{e-1} M\,.$$

Consequently, $\|g(sy)\|_{\mathbb{A}} \geq s^e m - s^{e-1}M$. If one takes s sufficiently large to ensure $s^e m - s^{e-1}M > 0$ and $s\epsilon \geq (d+1)/2$, the ball $\bar{B}$ obtained by uniformly scaling B

by a factor s contains no root of g. Since the width of the box $\bar{B}$ is at least $d+1$, the existence of the expected S_i is guaranteed. □

Roughly speaking, the following lemma ensures that if a fraction q in $\mathbb{K}(Y)[X]$ evaluates to zero for well chosen evaluations of Y, then q is necessarily zero.

Lemma 3. *Take an element q in $\mathbb{K}(Y)[X]$. Then, there exist an integer d, and m sets $S_1, S_2, \ldots, S_m$, each one containing $d+1$ nonnegative consecutive integers, such that $\sigma_{y^0}(q)$ is well-defined for all $y^0 \in S = S_1 \times S_2 \times \cdots \times S_m$. Moreover, if $\sigma_{y^0}(q) = 0$ for all y^0 in S, then $q = 0$.*

Proof. Let us write $q = p/g$ where $p \in \mathbb{K}[X, Y]$ and $g \in \mathbb{K}[Y]$. Consider $d = \max_{1 \le i \le m} \deg_{y_i}(p)$. By Lemma 2, there exist m sets $S_1, \ldots, S_m$ of $d+1$ consecutive nonnegative integers such that $g(y^0) \neq 0$ for all y^0 in $S = S_1 \times S_2 \times \cdots \times S_m$. Consequently, $\sigma_{y^0}(q)$ is well-defined for all y^0 in S. Let us assume that $\sigma_{y^0}(q) = 0$ for all y^0 in S. As a consequence, one has $\sigma_{y^0}(p) = 0$ for any y^0 in S. By Lemma 1, one has $p = 0$, hence $q = 0$. □

2.2 Algorithm findKernelBasis

We rely on the notion of *iterator*, which is a tool in computer programming that enables to enumerate all values of a structure (such as a list, a vector, ...). An iterator can be *finite* (if it becomes empty after enumerating a finite number of values) or *infinite* (if it never gets empty).

In order to enumerate evaluation points (that we take in $\mathbb{N}^m$ for simplicity), we use two basic functions to generate one by one all the tuples, which do not cancel any element of D, where D is a list of nonzero polynomials of $\mathbb{K}[Y]$. The first one is called newTupleIteratorNotCancelling(Y, D). It builds an iterator I to be used with the second function getNewTuple(I). Each time one calls the function getNewTuple(I), one retrieves a new tuple which does not cancel any element of D. The only constraint we require for getNewTuple is that any tuple of $\mathbb{N}^m$ (which does not cancel any element of D) should be output after a finite number of calls to getNewTuple. To ensure that, one can for example enumerate all integer tuples by increasing order (where the order is the sum of the entries of the tuple) refined by the lexicographic order for tuples of the same order. Without this constraint on getNewTuple, Algorithm findKernelBasis may not terminate.

Example 2. Take $\mathbb{K} = \mathbb{Q}(z)$, $Y = \{a\}$ and $X = \{x\}$. Consider $q = (q_1, q_2, q_3) = (\frac{ax+2}{a+1}, \frac{z(a-1-ax)}{1+a}, 1)$. One can show that the only (up to a constant) $\mathbb{K}$-linear dependence between the q_i is $-q_1 - (1/z)q_2 + q_3 = 0$.

Apply Algorithm findKernelBasis. One has $D = [a+1]$ at Line 3, so one can evaluate the indeterminate a on any nonnegative integer. At Line 7, $\bar{S}$ contains the tuple $s_1 = (0)$, corresponding to the evaluation $a = 0$. One builds the vectors v_1, v_2 and v_3 at Line 7 by evaluating q_1, q_2 and q_3 on $a = 0$. One obtains polynomials in $\mathbb{K}[x]$. Thus $v_1 = (2)$, $v_2 = (-z)$ and $v_3 = (1)$. A basis of $\ker \Phi_v$ computed at Line 10 could be $B = \{(-1/2, 0, 1), (z/2, 1, 0)\}$. Note that it is normal that both $v = (v_1, v_2, v_3)$ and B involve z since one performs linear

Input: Two lists of variables Y and X
Input: A vector $q = (q_1, \ldots, q_e)$ of e rational fractions in $\mathbb{K}(Y)[X]$
Output: A basis of $\ker \Phi_q$

1 **begin**
2   For each $1 \leq i \leq e$, denote the reduced fraction q_i as f_i/g_i with $f_i \in \mathbb{K}[Y, X]$, $g_i \in \mathbb{K}[Y]$;
3     $D \leftarrow [g_1, \ldots, g_e]$;
4     $I \leftarrow$ newTupleIteratorNotCancelling(Y, D);
5     $\bar{S} \leftarrow$ [getNewTuple(I)];
6   **while** *true* **do**
7   For each $1 \leq i \leq e$, denote v_i the vector $(\sigma_{s_1}(q_i), \sigma_{s_2}(q_i), \ldots, \sigma_{s_r}(q_i))$ where $\bar{S} = [s_1, s_2, \ldots, s_r]$;
8   // each v_i is a vector of $r = |\bar{S}|$ elements of $\mathbb{K}[X]$, obtained by evaluating q_i on all points of $\bar{S}$
9   Denote $v = (v_1, \ldots, v_e)$;
10   Compute a basis B of the kernel of Φ_v using linear algebra;
11   // if $\ker \Phi_q \supset \ker \Phi_v$, one returns B
12   **if** $\sum_{i=1}^{e} b_i q_i = 0$ for all $b = (b_1, \ldots, b_e) \in B$ **then return** B;
13   Append to $\bar{S}$ the new evaluation getNewTuple(I);

Algorithm 1. findKernelBasis

algebra over $\mathbb{K}$, which contains z. The test at Line 12 fails because the vector $b_1 = (-1/2, 0, 1)$ of B does not yield a linear dependence over the q_i. Indeed, $-(1/2)q_1 + 0q_2 + q_3 \neq 0$.

Consequently, $\bar{S}$ is enriched with the new tuple $s_1 = (1)$ at Line 13 and one performs a second iteration. One builds the vectors v_1, v_2 and v_3 at Line 7 by evaluating q_1, q_2 and q_3 on $a = 0$ and $a = 1$. Thus $v_1 = (2, 1 + x/2)$, $v_2 = (-z, -xz/2)$ and $v_3 = (1, 1)$. A basis B computed at Line 10 could be $(-1, -1/z, 1)$. This time, the test at Line 12 succeeds since $-q_1 - (1/z)q_2 + q_3 = 0$. The algorithm stops by returning the basis $(-1, -1/z, 1)$.

Proof (Algorithm findKernelBasis*).* **Correctness.** If the algorithm returns, then $\ker \Phi_q \supset \ker \Phi_v$. Moreover, at each loop, one has $\ker \Phi_q \subset \ker \Phi_v$ since $\sum \alpha_i q_i = 0$ implies $\sum \alpha_i \sigma(q_i) = 0$ for any evaluation which does not cancel any denominator. Consequently, if the algorithm stops, $\ker \Phi_q = \ker \Phi_v$, and B is a basis of $\ker \Phi_q$. **Termination.** Assume the algorithm does not terminate. The vector space $\ker \Phi_v$ is smaller at each step since the set $\bar{S}$ grows. By a classical dimension argument, the vector space $\ker \Phi_v$ admits a limit denoted by E, and reaches it after a finite number of iterations. From $\ker \Phi_q \subset \ker \Phi_v$, one has $\ker \Phi_q \subset E$. Since the test at Line 12 always fails (the algorithm does not terminate), $\ker \Phi_q \subsetneq E$.

Take $\alpha \in E \setminus \ker \Phi_q$ and consider the set S obtained by applying Lemma 3 with $\bar{q} = \sum_{i=1}^{e} \alpha_i q_i$. Since the algorithm does not terminate, $\bar{S}$ will eventually contain S. By construction of v, one has $\sum_{i=1}^{e} \alpha_i \sigma_{y^0}(q_i) = 0 = \sigma_{y^0}(\bar{q})$ for all y^0 in S. By Lemma 3, one has $\bar{q} = \sum_{i=1}^{e} \alpha_i q_i = 0$ which proves that $\alpha \in \ker \Phi_q$. Contradiction since $\alpha \notin \ker \Phi_q$. □

Input: Two lists of variables Y and X
Input: A vector $q = (q_1, \ldots, q_e)$ of e rational fractions in $\mathbb{K}(Y)[X]$
Output: A basis of $\ker \Phi_q$

begin
  For each $1 \leq i \leq e$, denote the reduced fraction q_i as f_i/g_i with $f_i \in \mathbb{K}[Y, X]$, $g_i \in \mathbb{K}[Y]$;
  $D \leftarrow [g_1, \ldots, g_e]$;
  $I \leftarrow$ newTupleIteratorNotCancelling(Y, D);
  $M \leftarrow$ the $0 \times e$ matrix;
  while *true* **do**
    $y^0 \leftarrow$ getNewTuple(I);
    // evaluate the vector q at y^0
    $\bar{q} \leftarrow \sigma_{y^0}(q)$;
    build $L = [\omega_1, \ldots, \omega_l]$ the list of monomials involved in $\bar{q}$;
    build the $l \times e$ matrix $N = (N_{ij})$ where N_{ij} is the coefficient of $\bar{q}_j$ in the monomial ω_i ;
    $M \leftarrow \begin{pmatrix} M \\ \hline N \end{pmatrix}$;
    Compute a basis B of the right kernel of M;
    if $\sum_{i=1}^{e} b_i q_i = 0$ for all $b = (b_1, \ldots, b_e) \in B$ **then** **return** B;

Algorithm 2. findKernelBasisMatrix

2.3 A Variant of findKernelBasis

A variant of algorithm findKernelBasis consists in incrementally building a matrix M with values in $\mathbb{K}$, encoding all the evaluations. Each column of M corresponds to a certain q_j, and each line corresponds to a couple (s, m), where s is an evaluation point, and m is a monomial of $\mathbb{K}[X]$. This yields Algorithm findKernelBasisMatrix.

Example 3. Apply Algorithm findKernelBasisMatrix on Example 2. The only difference with Algorithm findKernelBasis is that the vectors v_i are stored vertically in a matrix M that grows at each iteration. At the end of the first iteration of the while loop, one has $y^0 = (0)$, $\bar{q} = (2, -z, 1)$, $L = [1]$, and $M = N = \begin{pmatrix} 2 & -z & 1 \end{pmatrix}$, and $B = \{(-1/2, 0, 1), (z/2, 1, 0)\}$. Another iteration is needed since $-(1/2)q_1 + 0q_2 + q_3 \neq 0$.

At the end of the second iteration, one has $y^0 = (1)$, $\bar{q} = (1 + x/2, -xz/2, 1)$, $L = [1, x]$, $N = \begin{pmatrix} 1 & 0 & 1 \\ 1/2 & -z/2 & 0 \end{pmatrix}$, and $M = \begin{pmatrix} 2 & -z & 1 \\ 1 & 0 & 1 \\ 1/2 & -z/2 & 0 \end{pmatrix}$. A basis B is $(-1, -1/z, 1)$ and the algorithm stops since $-q_1 - (1/z)q_2 + q_3 = 0$.

Proof (Algorithm findKernelBasisMatrix). The proof is identical to the proof of findKernelBasis. Indeed, the vector v_i in findKernelBasis is encoded vertically in the i^{th} column of the M matrix in findKernelBasisMatrix. Thus, the kernel of Φ_v in findKernelBasis and the kernel of M in findKernelBasisMatrix coincide. □

In terms of efficiency, Algorithm findKernelBasisMatrix is far from optimal for many reasons. First, the number of lines of the matrix M grows excessively. Second, a basis B has to be computed at each step of the loop. Third, the algorithm needs to know all the rational functions in advance, which forbids an incremental approach. Next section addresses those issues.

3 Incremental Computation of Linear Dependences

The main idea for building incrementally the linear dependences is presented in Algorithm incrementalFindDependence. It is a sub-algorithm of Algorithms findFirstDependence and findAllFirstIntegrals. Assume we have e linearly independent rational functions $q_1, \ldots, q_e$ and a new rational function q_{e+1}: either the $q_1, \ldots, q_{e+1}$ are also linearly independent, or there exists $(\alpha_1, \ldots, \alpha_{e+1}) \in \mathbb{K}^{e+1}$ such that $\sum_{i=1}^{e+1} \alpha_i q_i = 0$ with $\alpha_{e+1} \neq 0$. It suffices to iterate this idea by increasing the index e until a linear dependence is found. When such a dependence has been found, one can either stop, or continue to get further dependences.

3.1 Algorithm incrementalFindDependence

The main point is to detect and store the property that the $q_1, \ldots, q_e$ are linearly independent, hence the following definition. Moreover, for efficiency reasons, one should be able to update this property at least cost when a new polynomial q_{e+1} is considered.

Definition 3 (Evaluation matrix). *Consider e rational functions $q_1, \ldots, q_e$ in $\mathbb{K}(Y)[X]$, and e couples $(s_1, \omega_1), \ldots, (s_e, \omega_e)$ where each s_i is taken in $\mathbb{N}^m$ and each ω_i is a monomial in X. Assume that none of the s_i cancels any denominator of the q_i. Consider the $e \times e$ matrix M with coefficients in $\mathbb{K}$, defined by $M_{ij} = \mathsf{coeff}(\sigma_{s_i}(q_j), \omega_i)$ where $\mathsf{coeff}(p, m)$ is the coefficient of the monomial m in the polynomial p. The matrix M is called the* evaluation matrix *of $q_1, \ldots, q_e$ w.r.t $(s_1, \omega_1), \ldots, (s_e, \omega_e)$.*

Example 4. Recall $q_1 = \frac{ax+2}{a+1}$ and $q_2 = \frac{z(a-1-ax)}{1+a}$ from Example 2. Consider $(s_1, \omega_1) = (0, 1)$ and $(s_2, \omega_2) = (1, 1)$. Thus, $\sigma_{s_1}(q_1) = 2$, $\sigma_{s_1}(q_2) = -z$, $\sigma_{s_2}(q_1) = x/2 + 1$ and $\sigma_{s_2}(q_2) = -zx/2$. Then, the evaluation matrix for $(s_1, \omega_1), (s_2, \omega_2)$ is $M = \begin{pmatrix} 2 & -z \\ 1 & 0 \end{pmatrix} \cdot$ If w_2 were the monomial x instead of 1, the evaluation matrix would be $M = \begin{pmatrix} 2 & -z \\ 1/2 & -z/2 \end{pmatrix} \cdot$ In both cases, the matrix M is invertible.

The evaluation matrices computed in Algorithms incrementalFindDependence, findFirstDependence and findAllFirstIntegrals will be kept invertible, to ensure that some fractions are linearly independent (see Proposition 1 below).

Proposition 1. *Keeping the same notations as in Definition 3, if the matrix M is invertible, then the rational functions $q_1, \ldots, q_e$ are linearly independent.*

Input: Two lists of variables Y and X, and a list D of elements of $\mathbb{K}[Y]$
Input: A list $Q = [q_1, \ldots, q_e]$ of e rational functions in $\mathbb{K}(Y)[X]$
Input: A list $E = [(s_1, \omega_1), \ldots, (s_e, \omega_e)]$, with $s_i \in \mathbb{N}^m$ and ω_i a monomial in X
Input: M an invertible eval. matrix of $q_1, \ldots, q_e$ w.r.t. $(s_1, \omega_1), \ldots, (s_e, \omega_e)$
Input: q_{e+1} a rational function
Assumption: denote $q_i = f_i/g_i$ with $f_i \in \mathbb{K}[X, Y]$ and $g_i \in \mathbb{K}[Y]$ for $1 \le i \le e+1$. Each g_i divides a power of elements of D. Moreover, $\sigma_{s_i}(d_j) \neq 0$ for any s_i and $d_j \in D$.
Output: Case 1 : $(\alpha_1, \ldots, \alpha_e, 1)$ s.t. $q_{e+1} + \sum_{i=1}^{e} \alpha_i q_i = 0$
Output: Case 2 : $M', (s_{e+1}, \omega_{e+1})$ such that M' is the evaluation matrix of $q_1, \ldots, q_{e+1}$ w.r.t. $(s_1, \omega_1), \ldots, (s_{e+1}, \omega_{e+1})$, with M' invertible

begin
  $b \leftarrow (\ldots, \mathsf{coeff}(\sigma_{s_j}(q_{e+1}), \omega_j), \ldots)_{1 \le j \le e}$;
  solve $M\alpha = -b$ in the unknown $\alpha = (\alpha_1, \ldots, \alpha_e)$;
  $h \leftarrow q_{e+1} + \sum_{i=1}^{e} \alpha_i q_i$;
  if $h = 0$ **then**
    return $(\alpha_1, \ldots, \alpha_e, 1)$; // Case 1: a linear dependence has been found
  else
    $I \leftarrow \mathsf{newTupleIteratorNotCancelling}(Y, D)$;
    repeat $s_{e+1} \leftarrow \mathsf{getNewTuple}(I)$ **until** $\sigma_{s_{e+1}}(h) \neq 0$;
    choose a monomial ω_{e+1} such that $\mathsf{coeff}(\sigma_{s_{e+1}}(h), \omega_{e+1}) \neq 0$;
    $l \leftarrow (\mathsf{coeff}(\sigma_{s_{e+1}}(q_1), \omega_{e+1}) \quad \cdots \quad \mathsf{coeff}(\sigma_{s_{e+1}}(q_{e+1}), \omega_{e+1}))$;
    $M' \leftarrow \begin{pmatrix} M \mid b \\ l \end{pmatrix}$;
    return $M', (s_{e+1}, \omega_{e+1})$ // Case 2: $q_1, \ldots, q_{e+1}$ are linearly independent

Algorithm 3. incrementalFindDependence

Proof. Consider $\alpha = (\alpha_1, \ldots, \alpha_e)$ such that $\sum_{i=1}^{e} \alpha_i q_i = 0$. One proves that $\alpha = 0$. For each $1 \le j \le e$, one has $\sum_{i=1}^{e} \alpha_i \sigma_{s_j}(q_i) = 0$, and consequently $\sum_{i=1}^{e} \alpha_i \mathsf{coeff}(\sigma_{s_j}(q_i), \omega_j) = 0$. This can be rewritten as $M\alpha = 0$, which implies $\alpha = 0$ since M is invertible. □

By Proposition 1, each evaluation matrix of Example 4 proves that q_1 and q_2 are linearly independent. In some sense, an invertible evaluation matrix can be viewed as a certificate (in the the computational complexity theory terminology) that some fractions are linearly independent.

Proposition 2. *Take the same notations as in Definition 3 and assume the evaluation matrix M is invertible. Consider a new rational function q_{e+1}. If the rational functions $q_1, \ldots, q_{e+1}$ are linearly independent then one necessarily has $q_{e+1} + \sum_{i=1}^{e} \alpha_i q_i = 0$ where α is the unique solution of $M\alpha = -b$, with $b = (\ldots, \mathsf{coeff}(\sigma_{s_j}(q_{e+1}), \omega_j), \ldots)_{1 \le j \le e}$.*

Proof. Since M is invertible and by Proposition 1, any linear dependence involves q_{e+1} with a nonzero coefficient, assumed to be 1. Assume $q_{e+1} + \sum_{i=1}^{e} \alpha_i q_i = 0$. Then for each j, one has $\sum_{i=1}^{e} \alpha_i \mathsf{coeff}(\sigma_{s_j}(q_i), \omega_j) = -\mathsf{coeff}(\sigma_{s_j}(q_{e+1}), \omega_j)$ which can be rewritten as $M\alpha = -b$. □

Example 5. Consider the q_i of Example 2. Take $D = [a+1]$, and $(s_1, \omega_1) = (0, 1)$, and the 1×1 matrix $M = (2)$ which is the evaluation matrix of q_1 w.r.t. (s_1, ω_1). Apply algorithm incrementalFindDependence on Y, X, D, $[q_1]$, $[(s_1, \omega_1)]$, M and q_2. The vector b at Line 2 equals $(-z)$ since $q_2 = z(a - ax - 1)/(a+1)$ evaluates to $-z$ when $a = 0$. Solving $M\alpha = -b$ yields $\alpha = (z/2)$. Then $h = q_2 + (z/2)q_1 = \frac{az(2-x)}{2(a+1)} \neq 0$. One then iterates the *repeat until* loop until h evaluates to a non zero polynomial. The value $a = 0$ is skipped, and the *repeat until* loop stops with $s_2 = 1$. Choosing the monomial $w_2 = 1$ yields the matrix $M' = \begin{pmatrix} 2 & -z \\ 1 & 0 \end{pmatrix}$.

Proof (Algorithm incrementalFindDependence*).* **Correctness.** Assume the fractions $q_1, \ldots, q_{e+1}$ are not linearly independent. Then Proposition 2 ensures that h must be zero and Case 1 is correct. It the $q_1, \ldots, q_{e+1}$ are linearly independent, then necessarily h is non zero. Assume the *repeat until* loop terminates (see proof below). Since $\sigma_{s_{e+1}}(h) \neq 0$, a monomial ω_{e+1} such that $\mathsf{coeff}(\sigma_{s_{e+1}}(h), \omega_{e+1}) \neq 0$ can be chosen. By construction, the matrix M' is the evaluation matrix of $q_1, \ldots, q_{e+1}$ w.r.t $(s_1, \omega_1), \ldots, (s_{e+1}, \omega_{e+1})$. One just needs to prove that M' is invertible to end the correctness proof. Assume $M'v = 0$ with a non zero vector $v = (\alpha_1, \ldots, \alpha_e, \beta)$. If $\beta = 0$, then $M\alpha = 0$ where $\alpha = (\alpha_1, \ldots, \alpha_e)$ and $\alpha \neq 0$ since $v \neq 0$ and $\beta = 0$. Since M is invertible, $\alpha = 0$ hence a contradiction. If $\beta \neq 0$, then one can assume $\beta = 1$. The e first lines of $M'v = 0$ imply $M\alpha = -b$ where α is the vector computed in the algorithm. The last line of $M'v = 0$ implies that $l(\alpha_1, \ldots, \alpha_e, 1) = 0$, which implies $\mathsf{coeff}(\sigma_{s_{e+1}}(h, \omega_{e+1})) = 0$ and contradicts the choice of ω_{e+1}.

Termination. One only needs to show that the *repeat until* loop terminates. This follows from Lemma 3 in the case of the single fraction q_{e+1}. □

3.2 Improvement Using a LU-decomposition

In order to optimize the solving of $M\alpha = -b$ in incrementalFindDependence, one can require a LU-decomposition of the evaluation matrix M. The specification of Algorithm incrementalFindDependence can be slightly adapted by passing a LU-decomposition of $M = LU$ (with L lower triangular with a diagonal of 1, and U upper triangular), and by returning a LU-decomposition of $M' = L'U'$ in Case 2. Note that a PLU-decomposition (where P is a permutation matrix) is not needed in our case as shown by Proposition 4 below.

Proposition 3 (Solving α). *Solving $M\alpha = -b$ is equivalent to solving the two triangular systems $Ly = -b$, then $U\alpha = y$.*

Proposition 4 (The LU-decomposition of M'). *The LU-decomposition of M' can be obtained by:*

- *solve $\gamma U = l_{1:e}$ (in the variable γ), where $l_{1:e}$ denotes the e first components of the line vector l computed in* findFirstDependence
- *solve $Ly = -b$ (in the variable y)*
- $L' \leftarrow \left(\begin{array}{c|c} L & 0_{e\times 1} \\ \hline \gamma & 1 \end{array}\right)$ *and* $U' \leftarrow \left(\begin{array}{c|c} U & -y \\ \hline 0_{1\times e} & l_{e+1} + \gamma y \end{array}\right)$

Input: Two lists of variables Y and X
Input: A list D of elements of $\mathbb{K}[Y]$
Input: A finite iterator J which outputs rational functions $q_1, q_2, \ldots$ in $\mathbb{K}(Y)[X]$
Assumption: Denote the reduced fraction $q_i = f_i/g_i$ with $f_i \in \mathbb{K}[X, Y]$ and $g_i \in \mathbb{K}[Y]$. Each g_i divides a power of elements of D.
Output: Case 1: a shortest linear dependence i.e. a vector $(\alpha_1, \ldots, \alpha_e)$ and a list $[q_1, \ldots, q_e]$ with $\sum_{i=1}^{e} \alpha_i q_i = 0$ and e the smallest possible.
Output: Case 2: `FAIL` if no linear dependence exists

begin
  $M \leftarrow$ the 0×0 matrix ; // M is an evaluation matrix
  $Q \leftarrow$ the empty list [] ; // Q is the list of the q_i
  // E is a list of (s, ω), where s is an evaluation and w is a monomial
  $E \leftarrow$ the empty list [] ;
  bool $\leftarrow$ false;
  while (*bool*=false) and (the iterator J is not empty) **do**
    $q \leftarrow$ getNewRationalFunction(J);
    $r \leftarrow$ incrementalFindDependence(Y, X, D, Q, E, M, q);
    append q at the end of Q;
    if r is a linear dependence **then** $\alpha \leftarrow r$; *bool* $\leftarrow$ true ;
    else $M', (s, \omega) \leftarrow r$; $M \leftarrow M'$; append (s, ω) at the end of E;
  if *bool*=true **then** **return** (α, Q) **else** **return** `FAIL` ;

Algorithm 4. findFirstDependence

3.3 Finding the First Linear Dependence

The main advantage of Algorithm incrementalFindDependence is to limit the number of computations if one does not know in advance the number of rational fractions needed for having a linear dependence. Imagine the rational functions are provided by a finite iterator (i.e. a iterator that outputs a finite number of fractions), one can build the algorithm findFirstDependence which terminates on the first linear dependence it finds, or fails if no linear dependence exists.

Example 6. Let us apply Algorithm findFirstDependence on Example 2. The first fraction q to be considered is $\frac{ax+2}{a+1}$. The call to incrementalFindDependence returns the 1×1 matrix (2) and the couple $(s_1, \omega_1) = (0, 1)$. One can check that q evaluated at $a = 0$ yields 2 and that (2) is indeed the evaluation matrix for the couple $(s_1, \omega_1) = (0, 1)$.

Continue with the second iteration. One now considers $q = \frac{z(a-ax-1)}{a+1}$. Example 5 shows that the call to incrementalFindDependence returns the 2×2 invertible matrix $\begin{pmatrix} 2 & -z \\ 1 & 0 \end{pmatrix}$ and the couple $(s_2, \omega_2) = (1, 1)$.

Finally, the third iteration makes a call to incrementalFindDependence. Line 2 builds the vector $b = (1, 1)$. Line 3 solves $M\alpha = -b$, yielding $\alpha = (-1, -1/z)$. Line 4 builds $h = q_3 - q_1 - (1/z)q_2$ which is zero, so incrementalFindDependence returns at Line 6. As a consequence, findFirstDependence detects that a linear dependence has been found at Line 11 and returns $((-1, -1/z), [q_1, q_2, q_3])$ at Line 13.

Proof (Algorithm findFirstDependence*).* **Termination.** The algorithm obviously terminates since the number of loops is bounded by the number of elements output by the iterator J.

Correctness. One first proves the following loop invariant: M is invertible, and M is the evaluation matrix of Q w.r.t. E. The invariant is true when first entering the loop (even if the case is a bit degenerated since M, Q and E are all empty). Assume the invariant is true at some point. The call to incrementalFindDependence either detects a linear dependence α, or returns an evaluation matrix M' and a couple (s, ω). In the first case, M, Q and E are left unmodified so the invariant remains true. In the second case, M, Q and E are modified to incorporate the new fraction q, and the invariant is still true thanks to the specification of incrementalFindDependence. The invariant is thus proven.

When the algorithm terminates, it returns either (α, Q) or `FAIL`. If it returns (α, Q), this means that the variable *bool* has changed to true at some point. Consequently a linear dependence α has been found, and the algorithm returns (α, Q). The dependence is necessarily the one with the smallest e because of the invariant (ensuring M is invertible) and Proposition 1. This proves the Case 1 of the specification.

If the algorithms returns `FAIL`, the iterator has been emptied, and the variable *bool* is still false. Consequently, all elements of the iterator have been stored in Q, and because of the invariant and Proposition 1, the elements of Q are linearly independent. This proves the Case 2 of the specification. □

Remark 1. Please note that Algorithm findFirstDependence can be used with an infinite iterator (i.e. an iterator that never gets empty). However, Algorithm findFirstDependence becomes a semi-algorithm since it will find the first linear dependence if it exists, but will never terminates if no such dependence exists.

3.4 Complexity of the Linear Algebra

When findFirstDependence terminates, it has solved at most e square systems with increasing sizes from 1 to e. Assuming the solving of each system $M\alpha = b$ has a complexity of $O(n^\omega)$ [7] arithmetic operations, where n is the size of the matrix and ω is the exponent of linear algebra, the total number of arithmetic operations for the linear algebra is $O(e^{\omega+1})$. If the LU-decomposition variant is used in Algorithm incrementalFindDependence, then the complexity of drops to $O(e^3)$, since solving a triangular system of size n can be made in $O(n^2)$ arithmetic operations. As for the space complexity of algorithm findFirstDependence, it is $O(e^2)$, whether using the LU-decomposition variant or not.

4 Application to Finding First Integrals

In this section, we look for first integrals for ODE systems. Roughly speaking, a first integral is an expression which is constant over time along any solution of the ODE system. This is a difficult problem and we will make several simplifying hypotheses. First, we work in the context of differential algebra, and assume that the ODE system is given by polynomial differential equations. Second, we will only look for polynomial first integrals.

4.1 Basic Differential Algebra

This section is mostly borrowed from [6] and [2]. It has been simplified in the case of a single derivative. The reference books are [11] and [10]. A *differential ring* R is a ring endowed with an[2] abstract *derivation* δ i.e. a unary operation which satisfies the axioms $\delta(a+b) = \delta(a) + \delta(b)$ and $\delta(a\,b) = \delta(a)\,b + a\delta(b)$ for all $a,\, b \in R$. This paper considers a differential polynomial ring R in n *differential indeterminates* $u_1, \ldots, u_n$ with coefficients in the field $\mathbb{K}$. Moreover, we assume that $\mathbb{K}$ is a field of constants (i.e. $\delta k = 0$ for any $k \in \mathbb{K}$). Letting $U = \{u_1, \ldots, u_n\}$, one denotes $R = \mathbb{K}\{U\}$, following Ritt and Kolchin. The derivation δ generates a monoid w.r.t. the composition operation. It is denoted: $\Theta = \{\delta^i, i \in \mathbb{N}\}$ where $\mathbb{N}$ stands for the set of the nonnegative integers. The elements of Θ are the *derivation operators.* The monoid Θ acts multiplicatively on U, giving the infinite set ΘU of the *derivatives.*

If A is a finite subset of R, one denotes (A) the smallest ideal containing A w.r.t. the inclusion relation and $[A]$ the smallest differential ideal containing A. Let $\mathfrak{A}$ be an ideal and $S = \{s_1, \ldots, s_t\}$ be a finite subset of R, not containing zero. Then $\mathfrak{A} : S^\infty = \{p \in R \mid \exists\ a_1, \ldots, a_t \in \mathbb{N},\ s_1^{a_1} \cdots s_t^{a_t}\, p \in \mathfrak{A}\}$ is called the *saturation* of $\mathfrak{A}$ by the multiplicative family generated by S. The saturation of a (differential) ideal is a (differential) ideal [10, chapter I, Corollary to Lemma 1].

Fix a *ranking*, i.e. a total ordering over ΘU satisfying some properties [10, chapter I, section 8]. Consider some differential polynomial $p \notin \mathbb{K}$. The highest derivative v w.r.t. the ranking such that $\deg(p, v) > 0$ is called the *leading derivative* of p. It is denoted $\mathsf{ld}\, p$. 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. If C is a finite subset of $R \setminus \mathbb{K}$ then I_C denotes its set of initials, S_C denotes its set of separants and $H_C = I_C \cup S_C$.

A differential polynomial q is said to be *partially reduced* w.r.t. p if it does not depend on any proper derivative of the leading derivative v of p. It is said to be *reduced* w.r.t. p if it is partially reduced w.r.t. p and $\deg(q, v) < \deg(p, v)$. A set of differential polynomials of $R \setminus \mathbb{K}$ is said to be *autoreduced* if its elements are pairwise reduced. Autoreduced sets are necessarily finite [10, chapter I, section 9]. To each autoreduced set C, one may associate the set $L = \mathsf{ld}\, C$ of the leading derivatives of C and the set $N = \Theta U \setminus \Theta L$ of the derivatives which are not derivatives of any element of L (the derivatives "under the stairs" defined by C).

In this paper, we need not recall the (rather technical) definition of differential regular chains (see [2, Definition 3.1]). We only need to know that a differential regular chain C is a particular case of an autoreduced set and that membership to the ideal $[C] : H_C^\infty$ can be decided by means of normal form computations, as explained below.

---

[2] In the general setting, differential ring are endowed with finitely many derivations.

4.2 Normal Form Modulo a Differential Regular Chain

All the results of this section are borrowed from [2] and [6]. Let C be a regular differential chain of R, defining a differential ideal $\mathfrak{A} = [C]:H_C^\infty$. Let $L = \mathsf{ld}\, C$ and $N = \Theta U \setminus \Theta L$. The normal form of a rational differential fraction is introduced in [2, Definition 5.1 and Proposition 5.2], recalled below.

Definition 4. *Let a/b be a rational differential fraction with b regular modulo $\mathfrak{A}$. A* normal form *of a/b modulo C is any rational differential fraction f/g such that*

1 *f is reduced with respect to C,*
2 *g belongs to $\mathbb{K}[N]$ (and is thus regular modulo $\mathfrak{A}$),*
3 *a/b and f/g are equivalent modulo $\mathfrak{A}$ (in the sense that $a\,g - b\,f \in \mathfrak{A}$).*

Proposition 5. *Let a/b be a rational differential fraction, with b regular modulo $\mathfrak{A}$. The normal form f/g of a/b exists and is unique. The normal form is a $\mathbb{K}$-linear operation. Moreover*

4 *a belongs to $\mathfrak{A}$ if and only if its normal form is zero,*
5 *f/g is a canonical representative of the residue class of a/b in the total fraction ring of $R/\mathfrak{A}$,*
6 *each irreducible factor of g divides the denominator of an inverse of b, or of some initial or separant of C .*

The interest of [6] is that it provides a normal form algorithm which always succeeds (while [2] provides an algorithm which fails when splittings occur).

Recall that the normal form algorithm relies on the computation of inverses of differential polynomials, defined below.

Definition 5. *Let f be a nonzero differential polynomial of R. An* inverse *of f modulo C is any fraction p/q of nonzero differential polynomials such that $p \in \mathbb{K}[N \cup L]$ and $q \in \mathbb{K}[N]$ and $f\,p \equiv q \mod \mathfrak{A}$.*

4.3 Normal Form Modulo a Decomposition

This subsection introduces a new definition which in practice is useful for performing computations modulo a radical ideal $\sqrt{[\Sigma]}$ expressed as an intersection of differential regular chains (i.e. $\sqrt{[\Sigma]} = [C_1] : H_{C_1}^\infty \cap \cdots \cap [C_f] : H_{C_f}^\infty$). Such a decomposition can be computed with the RosenfeldGroebner algorithm [1,4].

Definition 6 (Normal form modulo a decomposition). *Let Σ be a set of differential polynomials, such that $\sqrt{[\Sigma]}$ is a proper ideal. Consider a decomposition of $\sqrt{[\Sigma]}$ into differential regular chains $C_1, \ldots, C_f$ for some ranking, that is differential regular chains satisfying $\sqrt{[\Sigma]} = [C_1] : H_{C_1}^\infty \cap \ldots \cap [C_f] : H_{C_f}^\infty$. For any differential fraction a/b with b regular modulo each $[C_i] : H_{C_i}^\infty$, one defines the normal form of a/b w.r.t. to the list $[C_1, \ldots, C_f]$ by the list*

$$[\mathsf{NF}(a/b, C_1), \ldots, \mathsf{NF}(a/b, C_f)]\,.$$

It is simply denoted by $\mathsf{NF}(a/b, [C_1, \ldots, C_f])$.

Since it is an open problem to compute a canonical (e.g. minimal) decomposition of the radical of a differential ideal, the normal form of Definition 6 depends on the decomposition and not on the ideal.

Proposition 6. *With the same notations as in Definition 6, for any polynomial* $p \in R$*, one has* $p \in \sqrt{[\Sigma]} \iff \mathsf{NF}(p, [C_1, \ldots, C_f]) = [0, \ldots, 0]$*. Moreover, the normal form modulo a decomposition is* $\mathbb{K}$*-linear.*

4.4 First Integrals in Differential Algebra

Definition 7 (First integral modulo an ideal). *Let* p *be a differential polynomial and* $\mathfrak{A}$ *be an ideal. One says* p *is a first integral modulo* $\mathfrak{A}$ *if* $\delta p \in \mathfrak{A}$*.*

For any ideal $\mathfrak{A}$, the set of the first integrals modulo $\mathfrak{A}$ contains the ideal $\mathfrak{A}$. If $\mathfrak{A}$ is a proper ideal, the inclusion is strict since any element of $\mathbb{K}$ is a first integral. In practice, the first integrals taken in $\mathbb{K}$ are obviously useless.

Example 7 (Pendulum). Take $\mathbb{K} = \mathbb{Q}(m, l, g)$. Consider the ranking $\cdots > \ddot{l} > \ddot{x} > \ddot{y} > \dot{l} > \dot{x} > \dot{y} > l > x > y$. Recall the pendulum equations Σ in Equations (1). Algorithm RosenfeldGroebner [4] shows that $\sqrt{[\Sigma]} = [C_1] : H_{C_1}^{\infty} \cap [C2] : H_{C2}^{\infty}$ where C_1 and C_2 are given by:

- $C_1 = [\dot{\lambda} = -3\,\frac{\dot{y}gm}{l^2}, \dot{y}^2 = -\frac{-\lambda y^2 l^2 + \lambda l^4 - y^3 gm + ygml^2}{ml^2}, x^2 = -y^2 + l^2]$;
- $C_2 = [\lambda = -\frac{ygm}{l^2}, x = 0, y^2 = l^2]$.

Remark that the differential regular chain C_2 corresponds to a degenerate case, where the pendulum is vertical since $x = 0$. Further computations show that $\mathsf{NF}(\delta(\frac{m}{2}(\dot{x}^2 + \dot{y}^2) - mg\,y), [C_1, C_2]) = [0,\ 0])$, proving that $p = \frac{m}{2}(\dot{x}^2 + \dot{y}^2) - mg\,y$ is a first integral modulo $\sqrt{[\Sigma]}$. Remark that $x^2 + y^2$ is also a first integral. This is immediate since $\delta(x^2 + y^2) = \delta(x^2 + y^2 - 1) \in \sqrt{[\Sigma]}$.

Definition 7 is new to our knowledge. It is expressed in a differential algebra context. The end of Section 4.4 makes the link with the definition of first integral in a analysis context, through analytic solutions using results from [5].

Definition 8 (Formal power solution of an ideal). *Consider a* n*-uple* $\bar{u} = (\bar{u}_1(t),\ldots,\bar{u}_n(t))$ *of formal power series in* t *over* $\mathbb{K}$*. For any differential polynomial, one defines* $p(\bar{u})$ *as the formal power series in* t *obtained by replacing each* u_i *by* $\bar{u}_i$ *and interpreting the derivation* δ *as the usual derivation on formal power series. The* n*-uple* $\bar{u} = (\bar{u}_1, \ldots, \bar{u}_n)$ *is called a solution of an ideal* $\mathfrak{A}$ *if* $p(\bar{u}) = 0$ *for all* $p \in \mathfrak{A}$*.*

Lemma 4. *Take a differential polynomial* p *and* n*-uple* $\bar{u} = (\bar{u}_1(t),\ldots,\bar{u}_n(t))$ *of formal power series. Then* $(\delta p)(\bar{u}) = \frac{\mathrm{d}p(\bar{u})}{\mathrm{d}t}$*. If* p *is a first integral modulo an ideal* $\mathfrak{A}$ *and* $\bar{u}$ *is a solution of* $\mathfrak{A}$*, then the formal power series* $p(\bar{u})$ *satisfies* $\frac{\mathrm{d}p(\bar{u})}{\mathrm{d}t} = 0$*.*

Proof. Since δ is a derivation, $(\delta p)(\bar{u}) = \frac{\mathrm{d}p(\bar{u})}{\mathrm{d}t}$ is proved if one proves it when p equals any u_i. Assume that $p = u_i$. Then $(\delta p)(\bar{u}_i) = (\delta u_i)(\bar{u}_i) = \frac{\mathrm{d}p(\bar{u})}{\mathrm{d}t}$. Assume p is a first integral modulo $\mathfrak{A}$ and $\bar{u}$ is a solution of $\mathfrak{A}$. Then $\delta p \in \mathfrak{A}$ and $(\delta p)(\bar{u}) = 0$. Using $(\delta p)(\bar{u}) = \frac{\mathrm{d}p(\bar{u})}{\mathrm{d}t}$, one has $\frac{\mathrm{d}p(\bar{u})}{\mathrm{d}t} = 0$ □

Take a system of differential polynomials Σ. By [5, Theorem and definition 3], a differential polynomial p in R vanishes on all analytic solutions (over some open set with coordinates in the algebraic closure of the base field) of Σ if and only if $p \in \sqrt{[\Sigma]}$. Applying this statement to $p = \delta q$ for some first integral q w.r.t. $\sqrt{[\Sigma]}$, then δq vanishes on all analytic solutions of $\sqrt{[\Sigma]}$ so p is a first integral in the context of analysis, if one excludes the non analytic solutions.

4.5 Algorithm findAllFirstIntegrals

In this section, one considers a proper ideal $\mathfrak{A}$ given as $\mathfrak{A} = [C_1] : H_{C_1}^\infty \cap \cdots \cap [C_f] : H_{C_f}^\infty$ where the C_i are differential regular chains. Denote $M = [C_1, \ldots, C_f]$. Take

Input: A list of differential regular chains $M = [C_1, \ldots, C_f]$
Input: A finite iterator J which outputs differential monomials
Output: A $\mathbb{K}$-basis of the linear combinations of momomials of J which are first integrals w.r.t. $[C_1] : H_{C_1}^\infty \cap \cdots \cap [C_f] : H_{C_f}^\infty$

begin
  result $\leftarrow$ the empty list [] ; $D \leftarrow$ the empty list [] ;
  for $i = 1$ **to** f **do**
    $I \leftarrow$ the inverses of the initials and separants of C_i modulo C_i ;
    append to D the denominators of the elements of I ;
  $Y \leftarrow$ the list of derivatives occurring in D ;
  $X \leftarrow$ the list of dummy variables $[d_1, \ldots, d_f]$;
  $M \leftarrow$ the 0×0 matrix ; $Q \leftarrow$ the empty list [] ;
  $E \leftarrow$ the empty list [] ; $F \leftarrow$ the empty list [];
  while the iterator J is not empty **do**
    $\mu \leftarrow$ getNewDerivativeMonomial(J);
    $q \leftarrow \sum_{i=1}^{f} d_i \mathsf{NF}(\delta\mu, C_i)$;
    append to X the variables of the numerator of q,
      which are neither in X nor Y ;
    $r \leftarrow$ incrementalFindDependence(Y, X, D, Q, E, M, q);
    if r is a linear dependence **then**
      // A new first integral has been found
      $(\alpha_1, \ldots, \alpha_e, 1) \leftarrow r$;
      append $\alpha_1\mu_1 + \cdots + \alpha_e\mu_e + w$ to *result*, where $F = (\mu_1, \ldots, \mu_e)$;
    else
      append μ to the end of F ;   append q to the end of Q ;
      $M', (s, \omega) \leftarrow r$; $M \leftarrow M'$;   append (s, ω) to the end of E;
  return *result* ;

Algorithm 5. findAllFirstIntegrals

a first integral modulo $\mathfrak{A}$ of the form $p = \sum \alpha_i \mu_i$, where the α_i's are in $\mathbb{K}$ and the μ_i's are monomials in the derivatives. Computing on lists componentwise, we have $0 = \mathsf{NF}(\delta p, M) = \sum \alpha_i \mathsf{NF}(\delta \mu_i, M)$. Consequently, a candidate $\sum \alpha_i \mu_i$ is a first integral modulo $\mathfrak{A}$ if and only if the $\mathsf{NF}(\delta \mu_i, M)$ are linearly dependent over $\mathbb{K}$.

Since the μ_i have no denominators, every irreducible factor of the denominator of any $\mathsf{NF}(w_i, C_j)$ necessarily divides (by Proposition 5) the denominator of the inverse of a separant or initial of C_j. As a consequence, the algorithms presented in the previous sections can be applied since we can precompute factors which should not be cancelled.

Algorithm findAllFirstIntegrals is very close to Algorithm findFirstDependence and only requires a few remarks. Instead of stopping at the first found dependence, it continues until the iterator has been emptied and stores all first dependences encountered. It starts by precomputing a safe set D for avoiding cancelling the denominators of any $\mathsf{NF}(\delta w_i, C_j)$. The algorithm introduces some dummy variables $d_1, \ldots, d_f$ for storing the normal form $\mathsf{NF}(\delta\mu, M)$, which is by definition a list, as the polynomial $d_1\mathsf{NF}(\delta\mu, C_1) + \cdots + d_f\mathsf{NF}(\delta\mu, C_f)$. This alternative storage allows us to directly reuse Algorithm incrementalFindDependence which expects a polynomial.

Example 8 (Pendulum). Take the same notations as in Example 7. Take an iterator J enumerating the monomials 1, y, x, $\dot{y}$, $\dot{x}$, y^2, xy, y^2, $\dot{y}y$, $\dot{y}x$, $\dot{y}^2$, $\dot{x}y$, $\dot{x}x$, $\dot{x}\dot{y}$ and $\dot{x}^2$. Then Algorithm findAllFirstIntegrals returns the list $[1, x^2 + y^2, \dot{y}y + \dot{x}x, -2g\,y + \dot{x}^2 + \dot{y}^2]$. Note the presence of $\dot{y}y + \dot{x}x$ which is in the ideal since it is the derivative of $(x^2 + y^2 - 1)/2$.

The intermediate computations are too big be displayed here. As an illustration, one gives the normal forms of δy, $\delta(\dot{y}^2)$, $\delta(\dot{x}\dot{y})$ and $\delta(\dot{x}^2)$ modulo $[C_1, C_2]$ which are respectively

$$[\dot{y}, 0]\,, \left[\frac{2(\lambda y\dot{y} + mg\dot{y})}{m}, 0\right], \left[\frac{x\dot{y}(\lambda(2y^2 - l^2) + mgy)}{m(y^2 - l^2)}, 0\right] \text{ and } \left[-\frac{2\lambda y\dot{y}}{m}, 0\right].$$

When increasing the degree bound, one finds more and more spurious first integrals like $\dot{y}\,y^2 + \dot{x}\,x\,y$ (which is in the ideal) or some powers of the first integral $-2g\,y + \dot{x}^2 + \dot{y}^2$.

Example 9 (Lotka-Volterra equations).

$$C \begin{cases} \dot{x}(t) & = a\,x(t) - b\,x(t)\,y(t) \qquad\qquad x(t)\,\dot{u}(t) = \dot{x}(t) \\ \dot{y}(t) & = -c\,y(t) + d\,x(t)\,y(t) \qquad\quad y(t)\,\dot{v}(t) = \dot{y}(t) \end{cases} \tag{2}$$

Take $\mathbb{K} = \mathbb{Q}(a, b, c, d)$ and the ranking $\cdots > \dot{u} > \dot{v} > \dot{x} > \dot{y} > u > v > x > y$. One can show that C is a differential regular chain for the chosen ranking. The two leftmost equations of C corresponds to the classical Lotka-Volterra equations, and the two rightmost ones encode the logarithms of $x(t)$ and $y(t)$ in a polynomial way. A call to findAllFirstIntegrals with the monomials of degree at most 1 built over x, y, u, v yields $[1, -\frac{av}{d} - \frac{cu}{d} + \frac{by}{d} + x]$ which corresponds to the usual first integral $-a\ln(y(t)) - c\ln(x(t) + by(t) + dx(t)$.

Remark 2. The choice of the degree bounds and the candidate monomials in the first integrals is left to the user, through the use of the iterator J. This makes our algorithm very flexible especially if the user has some extra knowledge on the first integrals or is looking for specific ones. Finally, this avoids the difficult problem of estimating degree bounds, which can be quite high. Indeed, the simple system $\dot{x} = x, \dot{y} = -ny$, where n is any positive integer, admits $x^n y$ as a first integral, which is minimal in terms of degrees.

4.6 Complexity

The complexity for the linear algebra part of Algorithm findAllFirstIntegrals is the same as for Algorithm findFirstDependence: it is $O(e^3)$, where e is the cardinal of the iterator J, if one uses the LU-decomposition variant. However, e can be quite large in practice. For example, if one considers the monomials of total degree at most d involving s derivatives, then e is equal to $\binom{s+d}{s}$.

References

1. Boulier, F.: The BLAD libraries (2004). `http://cristal.univ-lille.fr/~boulier/BLAD`
2. Boulier, F., Lemaire, F.: A normal form algorithm for regular differential chains. Mathematics in Computer Science **4**(2–3), 185–201 (2010)
3. Boulier, F.: Efficient computation of regular differential systems by change of rankings using Kähler differentials. Technical report, Université Lille I, 59655, Villeneuve d'Ascq, France (November 1999) Ref. LIFL 1999–14, presented at the MEGA 2000 conference. `http://hal.archives-ouvertes.fr/hal-00139738`
4. Boulier, F., Lazard, D., Ollivier, F., Petitot, M.: Computing representations for radicals of finitely generated differential ideals. Applicable Algebra in Engineering, Communication and Computing 20(1), 73–121 (2009); (1997 Techrep. IT306 of the LIFL)
5. Boulier, F., Lemaire, F.: A computer scientist point of view on Hilbert's differential theorem of zeros. Submitted to Applicable Algebra in Engineering, Communication and Computing (2007)
6. Boulier, F., Lemaire, F., Sedoglavic, A.: On the Regularity Property of Differential Polynomials Modulo Regular Differential Chains. In: Gerdt, V.P., Koepf, W., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2011. LNCS, vol. 6885, pp. 61–72. Springer, Heidelberg (2011)
7. Bunch, J.R., Hopcroft, J.E.: Triangular factorization and inversion by fast matrix multiplication. Mathematics of Computation **28**(125), 231–236 (1974)
8. Faugère, J.C., Gianni, P., Lazard, D., Mora, T.: Efficient computation of Gröbner bases by change of orderings. Journal of Symbolic Computation **16**, 329–344 (1993)
9. Gathen, J.V.Z., Gerhard, J.: Modern Computer Algebra, 3rd edn. Cambridge University Press, New York (2013)
10. Kolchin, E.R.: Differential Algebra and Algebraic Groups. Academic Press, New York (1973)
11. Ritt, J.F.: Differential Algebra. Dover Publications Inc., New York (1950)
12. Stoutemyer, D.R.: Multivariate partial fraction expansion. ACM Commun. Comput. Algebra **42**(4), 206–210 (2009)