Derandomization and absolute reconstruction for sums of powers of linear forms

Pascal Koiran, Mateusz Skomra

▶ **To cite this version:**

Pascal Koiran, Mateusz Skomra. Derandomization and absolute reconstruction for sums of powers of linear forms. Theoretical Computer Science, 2021, 887, pp.63-84. 10.1016/j.tcs.2021.07.005 . hal-03457373

HAL Id: hal-03457373
https://laas.hal.science/hal-03457373

Submitted on 30 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Derandomization and Absolute Reconstruction for Sums of Powers of Linear Forms

Pascal Koiran* Mateusz Skomra†

July 15, 2021

Abstract

We study the decomposition of multivariate polynomials as sums of powers of linear forms. As one of our main results, we give a randomized algorithm for the following problem: given a homogeneous polynomial $f(x_1, \ldots, x_n)$ of degree 3, decide whether it can be written as a sum of cubes of linearly independent linear forms with complex coefficients. Compared to previous algorithms for the same problem, the two main novel features of this algorithm are:

(i) It is an algebraic algorithm, i.e., it performs only arithmetic operations and equality tests on the coefficients of the input polynomial f. In particular, it does not make any appeal to polynomial factorization.

(ii) For $f \in \mathbb{Q}[x_1, \ldots, x_n]$, the algorithm runs in polynomial time when implemented in the bit model of computation.

The algorithm relies on methods from linear and multilinear algebra (symmetric tensor decomposition by simultaneous diagonalization). We also give a version of our algorithm for decomposition over the field of real numbers. In this case, the algorithm performs arithmetic operations and comparisons on the input coefficients.

Finally we give several related derandomization results on black box polynomial identity testing, the minimization of the number of variables in a polynomial, the computation of Lie algebras and factorization into products of linear forms.

*Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France. Email: pascal.koiran@ens-lyon.fr.

† LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France. Email: mateusz.skomra@laas.fr.

1 Introduction

Let $f(x_1, \ldots, x_n)$ be a homogeneous polynomial of degree d, also called a *degree d form*. In this paper we study decompositions of the type:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{r} l_i(x_1, \ldots, x_n)^d \tag{1}$$

where the l_i are linear forms. Such a decomposition is sometimes called a Waring decomposition, or a symmetric tensor decomposition. We focus on the case where the linear forms l_i are linearly independent. This implies that the number r of terms in the decomposition is at most n. When $r = n$ we have $f(x) = P_d(Ax)$ where A is an invertible matrix of size n and

$$P_d(x_1, \ldots, x_n) = x_1^d + x_2^d + \cdots + x_n^d \tag{2}$$

is the "sum of d-th powers" polynomial. If f can be written in this way, we say that f is equivalent to a sum of n d-th powers. More generally, two polynomials f, g in n variables are said to be equivalent if they can be obtained from each other by an invertible change of variables, i.e., if $f(x) = g(Ax)$ where A is an invertible matrix of size n. As pointed out in [37], the case $d = 3$ (equivalence to a sum of n cubes) can be tackled with the decomposition algorithm for cubic forms in Saxena's thesis [48]. Equivalence to P_d for arbitrary d was studied by Kayal [37]. This paper also begins a study of equivalence to other specific polynomials such as the elementary symmetric polynomials; this study is continued in [20, 23, 24, 38, 39], in particular for the permanent and determinant polynomials. The contributions of the present paper are twofold:

(i) We give efficient tests for equivalence to a sum of n cubes over the fields of real and complex numbers. In particular, for an input polynomial with rational coefficients we give the first polynomial time algorithms in the standard Turing machine model of computation. As explained below in Section 1.1, this is not in contradiction with the polynomial time bounds from [37, 48] because we do not address exactly the same problem or work in the same computation model as these two papers.

More generally, we can test efficiently whether the input f can be written as in (1) as a sum of cubes of linearly independent linear forms. This follows easily from our equivalence tests and the algorithms from [37, 48] for the minimization of the number of variables in a polynomial.

(ii) Our first equivalence algorithm for the fields of real and complex numbers is randomized. We derandomize this algorithm in Section 5, and we continue with several related derandomization results on black box

polynomial identity testing, the minimization of the number of variables in a polynomial, the computation of Lie algebras and factorization into products of linear forms.

1.1 Equivalence to a sum of cubes

Our algorithm for equivalence to a sum of n cubes over $\mathbb{C}$ is *algebraic* in the sense that the input polynomial may have arbitrary complex coefficients and we manipulate them using only arithmetic operations and equality tests. Over the field of real numbers we also allow inequality tests. We therefore work in the "real RAM" model; an appropriate formalization is provided by the Blum-Shub-Smale model of computation [6, 7]. We can provide algebraic algorithms only because we are considering a *decision problem*: it is easy to see that if the input $f(x_1, \ldots, x_n)$ is equivalent to a sum of n cubes, the coefficients of the linear forms l_i in the corresponding decomposition need not be computable from those of f by arithmetic operations (see Examples 16 and 17 at the beginning of Section 3).

Polynomial factorization is an important subroutine in many if not most reconstruction algorithms for arithmetic circuits, see e.g. [18, 19, 36, 37, 39, 40, 51]. It may even seem unavoidable for some problems: reconstruction of $\Pi\Sigma$ circuits is nothing but the problem of factorization into products of linear forms, and reconstruction of $\Pi\Sigma\Pi$ circuits is factorization into products of sparse polynomials. Useful as it is, polynomial factorization is clearly not feasible with arithmetic operations only, even for polynomials of degree 2. We therefore depart from the aforementioned algorithms by avoiding all use of such a subroutine.

For an input polynomial f with rational coefficients, our algebraic algorithms run in polynomial time in the standard bit model of computation, i.e., they are "strongly polynomial" algorithms (this is not automatic due to the issue of coefficient growth during the computation). We emphasize that even for an input $f \in \mathbb{Q}[x_1, \ldots, x_n]$ we are still considering the problem of equivalence to a sum of n cubes over the real or complex numbers. Consider by contrast Kayal's equivalence algorithm [37], which appeals to a polynomial factorization subroutine. We can choose to factor polynomials over, say, the field of rational numbers. We can then run Kayal's algorithm without any difficulty on a probabilistic polynomial time Turing machine, but the algorithm will then reject the polynomial of Example 17 whereas our algorithm will accept it.[1] At first sight this difficulty seems to have a relatively simple solution: for an input with rational coefficients, instead of factoring

[1] Alternatively, one can run Kayal's algorithm in a computation model over $\mathbb{C}$ where, in addition to arithmetic operations over complex numbers, root finding of univariate polynomials is considered an atomic operation (as suggested in a footnote of [20]). The algorithm would then give the same answer as our algorithm, but it would not operate anymore within the Turing machine model (or within the BSS model).

polynomials in $\mathbb{Q}[X]$ we will factor in a field extension of $\mathbb{Q}$ containing the coefficients of the linear forms l_i (for instance in $\mathbb{Q}[\sqrt{2}]$ for Example 17). It is unfortunately not clear that this approach yields a polynomial time algorithm because it might lead to computations in a field extension of exponential degree. We explain this point in more detail in Section 3.1. For the same reason (reliance on a polynomial factorization subroutine) similar issues arise in the analysis of Saxena's decomposition algorithm. A complete analysis of these two algorithms for equivalence to a sum of powers over $\mathbb{C}$ in the Turing machine model would entail good control of coefficient growth and good bounds on the degrees of the field extensions involved. This has not been done yet to the best of our knowledge.

1.2 Derandomization

We give a deterministic black box identity testing algorithm for polynomials which can be represented as in (1) as a sum of powers of linearly independent linear forms. As we will see in Section 6.2, the problem is really to decide whether the (unknown) number of terms r in the decomposition is equal to 0. Indeed, for $r \geq 1$ such a polynomial can never be identically zero (and the PIT problem for this family of polynomials can therefore be solved by a trivial algorithm in the white box model). In contrast to our equivalence algorithms, this black box PIT applies to homogeneous polynomials of arbitrary degree.

There is already a significant amount of work on identity testing for sums of powers of linear forms. In particular, Saxena [47] gave a polynomial time algorithm in the white box model (where we have access to an arithmetic circuit computing the input polynomial). Subsequently, several algorithms were given for the black box model [1, 15, 14, 16] but they do not run in polynomial time. The current state of the art is in [16], with a black box algorithm running in time $s^{O(\log \log s)}$. We obtain here a polynomial running time under the assumption that the l_i are linearly independent. Without this assumption, designing a black box PIT algorithm running in polynomial time remains to the best of our knowledge an open problem.

In Section 7 we build on our black box PIT to derandomize Kayal's algorithm for the minimization of the number of variables in a polynomial [37]. Like our black box PIT, this result applies to polynomials that can be written as sums of powers of linearly independent linear forms. For such a polynomial, the minimal (or "essential") number of variables is just the number r of terms in the corresponding decomposition (1). We continue with the computation of Lie algebras of products of linear forms. Finally, our deterministic algorithm for this task is applied to the derandomization of a factorization algorithm from [41] and of Kayal's algorithm for equivalence to P_d [37].

1.3 Our approach

We obtain our equivalence algorithms by viewing the coefficients of the input polynomial $f(x_1, \ldots, x_n)$ as the coefficients of a symmetric tensor T of size n and order 3 (since f is of degree 3). Equivalence to a sum of n cubes then amounts to a kind of diagonalizability property of T. This approach is explained in detail in Section 3. It can be viewed as a continuation of previous work on orthogonal tensor decomposition [42] (the present paper is more algorithmic, is not limited to orthogonal decompositions and can be read independently from [42]).

We work on a tensor of size n by cutting it into n "slices"; each slice is a symmetric matrix of size n. We therefore rely on methods from linear algebra. This explains the presence of a section of preliminaries on simultaneous reduction by congruence (which is then applied to the slices of T). Despite these rather long preliminaries, the resulting randomized algorithm is remarkably simple: it is described in just 3 lines at the beginning of Section 4.

Our deterministic algorithms also rely on important insights from Kayal's paper [37]. In particular we rely on the factorization properties of the Hessian determinant of the input f, which we manage to use without appealing explicitly to a factorization subroutine (as explained in Section 1.1, this is ruled out in our approach). Our deterministic algorithm for the minimization of the number of variables is directly inspired by the randomized algorithm for this problem in the same paper.

1.4 Comparison with previous tensor decomposition algorithms

There is a vast literature on tensor decomposition algorithms, most of them numerical (see [4] for a recent paper showing that many of these algorithms are numerically unstable). From this literature, two papers by De Lathauwer et al. [12, 13] are closely related to the present work since they already recognized the importance of simultaneous diagonalization by congruence for tensor decomposition. Jennrich's algorithm is also closely related for the same reason. One can find a presentation of this algorithm in the recent book by Moitra [44] but it goes back much further to Harschman [27], where it is presented as a uniqueness result. There are nevertheless important differences between the settings of [12, 13, 44] and of the present paper. In particular, these three works do not phrase tensor decomposition as a decision problem but as an optimization problem which is solved by numerical means (one suggestion from [12, 13] is to perform the simultaneous diagonalization with the extended QZ iteration from [52]; Jennrich's algorithm as presented in [44] relies on pseudoinverse computations and eigendecompositions). All these numerical algorithms attempt to produce a decomposition

of a tensor $\tilde{T}$ which is close to the input tensor T. If one tries to adapt them to the setting of the present paper there is a fundamental difficulty: given $\tilde{T}$ and its decomposition, it is not clear how we can decide whether or not T admits an exact decomposition. This is the main reason why we need to design a new algorithm. As an alternative, one could attempt to run Jennrich's algorithm in symbolic mode. In particular, eigenvalues and the components of eigenvectors would be represented symbolically as elements of a field extension. This leads exactly to the same difficulty as with Kayal's algorithm: as explained in Section 1.1 and in more detail in Section 3, the resulting algorithm might not run in polynomial time because it might lead to computations in field extensions of exponential degree. Note finally that [12, 13, 44] deal with decompositions of ordinary rather than symmetric tensors. Algorithms for symmetric tensor decomposition can be found in the algebraic literature, see e.g. [8, 5]. Like [12, 13], these two papers do not provide any complexity analysis for their algorithms.

1.5 Future work

In the current literature there is apparently no polynomial time algorithm (deterministic or randomized) in the Turing machine model for the following problem: given a homogeneous polynomial $f(x_1, \ldots, x_n)$ of degree $d \geq 4$ with rational coefficients, decide whether it is equivalent to $x_1^d + \cdots + x_n^d$ over $\mathbb{C}$. It will be shown in a forthcoming paper that this can be done by extending the tensor-based approach of the present paper to higher degree. This results in a black box algorithm with running time polynomial in n and d [43]. Alternatively, one could try to modify Kayal's equivalence algorithm [37] or provide a better analysis of the existing algorithm. As we have argued in Section 1.1 this has not been done at present even for degree 3. One could also try an approach based on Harrison's work [26] like in Saxena's thesis [48].

More generally, we suggest to pay more attention to *absolute circuit reconstruction*, i.e., arithmetic circuit reconstruction over $\mathbb{C}$.[2] Circuit reconstruction over $\mathbb{Q}$ or over finite fields has a number-theoretic flavour, whereas circuit reconstruction over $\mathbb{R}$ or $\mathbb{C}$ is of a more geometric nature.

One goal could be to obtain algebraic decision algorithms; as we have explained, this requires the removal of all polynomial factorization subroutines. In principle, this is always possible since the set of polynomials computable by arithmetic circuits of a given shape and and size is definable by polynomial (in)equalities, i.e., it is a constructible set (over $\mathbb{C}$) or a semi-algebraic set (over $\mathbb{R}$).[3] Another goal would be to obtain good complexity bounds for the Turing machine model when they are not available in the existing

[2] The name is borrowed from absolute factorization, a well studied problem in computer algebra (see e.g. [10, 11, 17, 50]).

[3] This argument does not provide by itself an efficient decision algorithm.

literature.

1.6 Organization of the paper

Section 2 is devoted to preliminaries on simultaneous diagonalization, by similarity transformations and especially by congruence. In Section 3 we review Kayal's equivalence algorithm and explain why it does not yield a polynomial time bound in the bit model of computation. We also present our tensor-based approach to the equivalence problem. In Section 4 we give a polynomial time randomized algorithm for equivalence to a sum of cubes based on this approach. We derandomize this algorithm in Section 5. In Section 6 we give a PIT algorithm for polynomials that can be written as sums of d-th powers of linearly independent linear forms. As mentioned before, our algorithm runs in polynomial time in the black box model. We give a randomized algorithm to decide whether a polynomial can be expressed under that form in Section 7 (Proposition 41). The remainder of Section 7 is devoted to the derandomization of several algorithms from [37, 38] related to sums of powers of linear forms. We begin in Section 7.1 with the computation of linear dependencies between polynomials. Then we give applications to the minimization of the number of variables in sums of powers of linear forms (in Section 7.2), and to the computation of Lie algebras of products of linear forms (in Section 7.3). This leads to the derandomization of a factorization algorithm from [41] and of the equivalence algorithm by Kayal [37] described in Section 3.1. The resulting equivalence algorithm runs in polynomial time for every fixed value of the degree d. For the computation of the Lie algebras of products of linear forms, we give as an intermediate result (see Proposition 47 and Remark 48) an identity testing algorithm for a certain class of rational functions (rather than polynomials as is done usually). In the commutative setting, this is to the best of our knowledge the first result of this type.

2 Preliminaries

This section is devoted to preliminaries from linear algebra, and more specifically to simultaneous diagonalization by congruence. We begin with complex symmetric matrices in Section 2.1 and consider real symmetric matrices in Section 2.3. Section 2.2 is devoted to some refinements that are not strictly necessary for our main algorithms (they lead to an interesting connection with semidefinite programming, though; see Theorem 14 and the remarks following it). Upon first reading, if one wishes to understand only our results for the field of complex numbers it will therefore be sufficient to read Section 2.1 only (or even just the statement of Theorem 4; the corresponding result for the field of real numbers is Theorem 10).

A proof of the following lemma for the case of two matrices can be found in [41]. As shown in [42], the general case then follows easily. Note that this lemma is about the "usual" notion of diagonalization (by similarity) rather than by congruence (where one attempts to diagonalize a matrix A by a transformation of the form $A \mapsto R^T AR$). That is, we say that an invertible matrix T diagonalizes A if $T^{-1}AT$ is diagonal.

Lemma 1. *Let $A_1, \ldots, A_k \in M_n(\mathbb{K})$ be a tuple of simultaneously diagonalizable matrices with entries in a field $\mathbb{K}$, and let $S \subseteq \mathbb{K}$ be a finite set of size $|S| > n(n-1)/2$. Then there exist $\alpha_2, \ldots, \alpha_k$ in S such that any transition matrix which diagonalizes $A_1 + \alpha_2 A_2 + \ldots + \alpha_k A_k$ must also diagonalize all of the matrices $A_1, \ldots, A_k$.*

See Proposition 11 in Section 2.3 for an improvement of this lemma. In the next lemma we give an explicit description of a set of suitable choices for the tuple $(\alpha_2, \ldots, \alpha_k)$ in Lemma 1. This description will be needed for a derandomization result in Section 7.3.

Lemma 2. *Let $A_1, \ldots, A_k \in M_n(\mathbb{K})$ be a tuple of simultaneously diagonalizable matrices with entries in a field $\mathbb{K}$. There exists a set of at most $n(n-1)/2$ hyperplanes of $\mathbb{K}^{n-1}$ such that the following properties hold for any point $(\alpha_2, \ldots, \alpha_{n-1}) \in \mathbb{K}^{n-1}$ which does not belong to the union of the hyperplanes:*

(i) Any eigenvector of $A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k$ must also be an eigenvector of the k matrices $A_1, \ldots, A_k$.

(ii) Any transition matrix which diagonalizes $A_1 + \alpha_2 A_2 + \ldots + \alpha_k A_k$ must also diagonalize all of the matrices $A_1, \ldots, A_k$.

Proof. Since $A_1, \ldots, A_k$ are simultaneously diagonalizable, in order to establish (i) we may as well work in a basis where these matrices become diagonal. Let us therefore assume without loss of generality that $A_i = \mathrm{diag}(\lambda_{1i}, \ldots, \lambda_{ni})$. For any $\alpha_2, \ldots, \alpha_{n-1}$, the matrix $A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k$ is diagonal and its i-th diagonal entry is $\lambda_{i1} + \alpha_2 \lambda_{i2} + \cdots + \alpha_k \lambda_{ik}$. It therefore suffices to avoid the (proper) hyperplanes of $\mathbb{K}^{n-1}$ of the form:

$$\lambda_{i1} + \alpha_2 \lambda_{i2} + \cdots + \alpha_k \lambda_{ik} = \lambda_{j1} + \alpha_2 \lambda_{j2} + \cdots + \alpha_k \lambda_{jk}$$

where $(i, j) \in [n]^2$ ranges over all the pairs such that $(\lambda_{i1}, \ldots, \lambda_{ik}) \neq (\lambda_{j1}, \ldots, \lambda_{jk})$. Indeed, there are at most $n(n-1)/2$ hyperplanes of this form, and outside of their union there is no "collision of eigenvalues." More precisely, the eigenspace of $A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k$ associated to the eigenvalue $\lambda_{i1} + \alpha_2 \lambda_{i2} + \cdots + \alpha_k \lambda_{ik}$ is equal to $\bigcap_{j=1}^{k} E_{ij}$, where E_{ij} denotes the eigenspace of A_j associated to λ_{ij}. In particular, any eigenvector of $A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k$ is also an eigenvector of $A_1, \ldots, A_k$.

Finally, we show that any point $(\alpha_2, \ldots, \alpha_{n-1})$ which satisfies (i) also satisfies (ii). For any invertible matrix T, $T^{-1}(A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k)T$ is diagonal iff all the column vectors of T are eigenvectors of $A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k$. By (i) this implies that each column vector of T is also an eigenvector of $A_1, \ldots, A_k$. As a result, the k matrices $T^{-1}A_iT$ are all diagonal. □

We note that Lemma 1 directly follows from Lemma 2.(ii) via e.g. the Schwartz-Zippel lemma.

2.1 Simultaneous diagonalization by congruence

The following result is from Horn and Johnson [28]. The first part is just the statement of Theorem 4.5.17(b), and the additional properties in (ii) are established in the proof of that theorem (see [28] for details).

Theorem 3. *Let $A, B \in M_n(\mathbb{C})$ be two complex symmetric matrices of size n with A nonsingular, and let $C = A^{-1}B$.*

(i) C is diagonalizable if and only if there are complex diagonal matrices D and Δ and a nonsingular $R \in M_n(\mathbb{C})$ such that $A = RDR^T$ and $B = R\Delta R^T$.

(ii) Moreover, if $C = S\Lambda S^{-1}$ where S is nonsingular and Λ diagonal then the matrix R in (i) can be taken of the form $R = S^{-T}V^T$ where V is unitary and commutes with Λ.

The next result generalizes Theorem 3 and provides a solution to the second part of Problem 4.5.P4 in [28].

Theorem 4 (simultaneous diagonalization by congruence)**.** *Let $A_1, \ldots, A_k$ be complex symmetric matrices of size n and assume that A_1 is nonsingular. The $k-1$ matrices $A_1^{-1}A_i$ $(i = 2, \ldots, k)$ form a commuting family of diagonalizable matrices if and only if there are diagonal matrices Λ_i and a nonsingular matrix $R \in M_n(\mathbb{C})$ such that $A_i = R\Lambda_i R^T$ for all $i = 1, \ldots, k$.*

Proof. Suppose that $A_i = R\Lambda_i R^T$ where the Λ_i are diagonal and R nonsingular. Then the matrices $A_1^{-1}A_i = R^{-T}\Lambda_1^{-1}\Lambda_i R^T$ indeed form a commuting family of diagonalizable matrices. For the converse, assume that the matrices $A_1^{-1}A_i$ form such a family. Then these matrices are simultaneously diagonalizable, and by Lemma 1 there is a tuple $(\alpha_3, \ldots, \alpha_k)$ such that any transition matrix that diagonalizes the matrix

$$C = A_1^{-1}A_2 + \alpha_3 A_1^{-1}A_3 + \ldots + \alpha_k A_1^{-1}A_k$$

diagonalizes all of the $k-1$ matrices $C_i = A_1^{-1}A_i$. Now we apply Theorem 3 to $A = A_1$ and $B = A_2 + \alpha_3 A_3 + \ldots + \alpha_k A_k$. Write $C = S\Lambda S^{-1}$ where S is nonsingular and Λ diagonal. By part (ii) of Theorem 3 we can write

$A_1 = RDR^T$ and $B = R\Delta R^T$ where R is nonsingular, D and Δ are diagonal, $R = S^{-T}V^T$, V commutes with Λ and is unitary. By choice of the tuple α, we can write $C_i = S\Lambda_i S^{-1}$ where Λ_i is diagonal. We will show that $A_i = RD\Lambda_i R^T$ for $i \geq 2$, thereby completing the proof of the theorem. First, we note that V commutes with the Λ_i. Indeed, V and Λ are simultaneously diagonalizable since they commute and are diagonalizable (Λ is diagonal and V unitary). But any transition matrix which diagonalizes simultaneously V and Λ will diagonalize simultaneously V and the Λ_i (this follows from the choice of α and the relations $C_i = S\Lambda_i S^{-1}$, $C = S\Lambda S^{-1}$). These matrices must therefore commute. We can now complete the proof: for $i \geq 2$ we have

$$A_i = A_1 C_i = (S^{-T}V^T DVS^{-1})(S\Lambda_i S^{-1}) = S^{-T}V^T DV\Lambda_i S^{-1}.$$

Since V commutes with Λ_i, $A_i = S^{-T}V^T D\Lambda_i VS^{-1} = RD\Lambda_i R^T$ as announced. □

2.2 A refinement of Theorem 4

In this section we give a more "invariant" formulation of Theorem 4. Note indeed that this theorem assigns a special role to A_1. In Theorem 8 we give a formulation that depends only on the space spanned by the A_i and not on the choice of a specific spanning family $A_1, \ldots, A_k$. Some of the results in this section apply to $\mathbb{K} = \mathbb{R}$ as well as $\mathbb{K} = \mathbb{C}$.

The role of A_1 in Theorem 4 could of course be played by any other invertible matrix in the tuple. As it turns out, for our $k-1$ matrices the commutation property alone is also independent of the choice of the invertible matrix in the tuple. More precisely, we have:

Proposition 5. *Let $A_1, \ldots, A_k \in M_n(\mathbb{K})$; assume that A_1 and A_k are nonsingular. The $k-1$ matrices $A_k^{-1}A_i$ ($i = 1, \ldots, k-1$) commute if and only if the same is true of the $k-1$ matrices $A_1^{-1}A_i$ ($i = 2, \ldots, k$).*

The proof will use the following simple fact.

Lemma 6. *If A and B commute and A is invertible, then A^{-1} and B commute as well.*

Proof of Proposition 5. Suppose that the $A_k^{-1}A_i$ commute. We can write:

$$(A_1^{-1}A_i)(A_1^{-1}A_j) = (A_k^{-1}A_1)^{-1}(A_k^{-1}A_i)(A_k^{-1}A_1)^{-1}(A_k^{-1}A_j).$$

It follows from our hypothesis and from Lemma 6 that the four factors on the right hand side commute. We can therefore rewrite this equation as:

$$(A_1^{-1}A_i)(A_1^{-1}A_j) = (A_k^{-1}A_1)^{-1}(A_k^{-1}A_j)(A_k^{-1}A_1)^{-1}(A_k^{-1}A_i),$$

and now the right hand side is equal to $(A_1^{-1}A_j)(A_1^{-1}A_i)$. □

Let $\mathcal{V}$ be the space of matrices spanned by $A_1, \ldots, A_k$. The matrices $A_1^{-1}A_i$ commute if and only if $A_1^{-1}\mathcal{V}$ is a commuting subspace of $M_n(\mathbb{K})$. With Proposition 5 in hand, we can characterize this property in a way that is completely independent of the choice of a spanning family $A_1, \ldots, A_k$ for $\mathcal{V}$.

Theorem 7. *Let $\mathcal{V}$ be a nonsingular subspace of matrices of $M_n(\mathbb{K})$ (i.e., $\mathcal{V}$ does not contain singular matrices only). The two following properties are equivalent:*

(i) There exists a nonsingular matrix $A \in \mathcal{V}$ such that $A^{-1}\mathcal{V}$ is a commuting subspace.

(ii) For all nonsingular matrices $A \in \mathcal{V}$, $A^{-1}\mathcal{V}$ is a commuting subspace.

Proof. Since $\mathcal{V}$ is nonsingular, (ii) implies (i). For the converse, assume that $A_1^{-1}\mathcal{V}$ is a commuting subspace and that $A_1, \ldots, A_k$ is a spanning family of $\mathcal{V}$. Let $A \in \mathcal{V}$ be a nonsingular matrix. We can add A to our spanning family and apply Proposition 5 to $(A_1, \ldots, A_k, A)$, with A playing the role of A_k in that proposition. □

As a result, we can dissociate the commutativity test from the diagonalizability test in Theorem 4:

Theorem 8. *Let $A_1, \ldots, A_k$ be complex symmetric matrices of size n and assume that the subspace $\mathcal{V}$ spanned by these matrices is nonsingular. The two following properties are equivalent:*

(i) There are diagonal matrices Λ_i and a nonsingular matrix $R \in M_n(\mathbb{C})$ such that $A_i = R\Lambda_i R^T$ for all $i = 1, \ldots, k$.

(ii) $\mathcal{V}$ satisfies the two equivalent properties of Theorem 7, and there exists an invertible $B \in \mathcal{V}$ such that the matrices $B^{-1}A_i$ ($i = 1, \ldots, k$) are all diagonalizable.

Proof. Let $A \in \mathcal{V}$ be nonsingular, and suppose that there are diagonal matrices Λ_i and a nonsingular matrix $R \in M_n(\mathbb{C})$ such that $A_i = R\Lambda_i R^T$ for all i. Note that we have the same form for A, i.e., $A = R\Lambda R^T$ with Λ diagonal. As a result, we may assume without loss of generality that A is one of the matrices in the tuple $A_1, \ldots, A_k$ (we add it if necessary), and we may even assume that $A = A_1$. We may then take $B = A$ by Theorem 4.

Let us now prove the converse. We therefore assume that $\mathcal{V}$ satisfies the two properties of Theorem 7, and that $B \in \mathcal{V}$ is an invertible matrix such the matrices $B^{-1}A_i$ ($i = 2, \ldots, k$) are all diagonalizable. From property (ii) in Theorem 7 it follows that the matrices $B^{-1}A_i$ commute. We conclude by applying Theorem 4 to the tuple $(B, A_1, \ldots, A_k)$. □

2.3 Real matrices

Here we study the existence of decompositions similar to those of Theorem 3 and Theorem 4 for real matrices. We begin with a real version of Theorem 3.

Theorem 9. *Let $A, B \in M_n(\mathbb{R})$ be two real symmetric matrices of size n with A nonsingular, and let $C = A^{-1}B$.*

(i) C is diagonalizable and has real eigenvalues if and only if there are real diagonal matrices D and Δ and a nonsingular $R \in M_n(\mathbb{R})$ such that $A = RDR^T$ and $B = R\Delta R^T$.

(ii) Moreover, if $C = S\Lambda S^{-1}$ where S is a real nonsingular matrix and Λ diagonal then the matrix R in (i) can be taken of the form $R = S^{-T}V^T$ where V is orthogonal and commutes with Λ.

Proof. If a decomposition of the pair (A, B) as in (i) exists, it is clear that C must be diagonalizable with real eigenvalues since $C = R^{-T}D^{-1}\Delta R^T$. The converse and part (ii) can be obtained by a straightforward adaptation of the proof of Theorem 4.5.17(b) in [28]. □

The next result generalizes Theorem 9 and provides a real version of Theorem 4.

Theorem 10 (simultaneous diagonalization by congruence)**.** *Let $A_1, \ldots, A_k$ be real symmetric matrices of size n and assume that A_1 is nonsingular. The $k-1$ matrices $A_1^{-1}A_i$ $(i = 2, \ldots, k)$ form a commuting family of diagonalizable matrices with real eigenvalues if and only if there are real diagonal matrices Λ_i and a nonsingular matrix $R \in M_n(\mathbb{R})$ such that $A_i = R\Lambda_i R^T$ for all $i = 1, \ldots, k$.*

Proof. The proof of Theorem 4 applies almost verbatim: we just need to work everywhere with real matrices instead of complex matrices (and with real coefficients α_i), and appeal to Theorem 9 instead of Theorem 3. There is just one point in the proof where a little care is needed. Namely, in the proof of Theorem 4 we used the fact that Theorem 3.(ii) provides us with a unitary matrix V, and that unitary matrices are diagonalizable. In the real case we get an orthogonal matrix instead (as per Theorem 9.(ii)), and orthogonal matrices are not necessarily diagonalizable over $\mathbb{R}$. Nevertheless, real orthogonal matrices are diagonalizable over $\mathbb{C}$ since they are unitary. We can therefore conclude like in the proof of Theorem 4 that our real orthogonal matrix V commutes with the Λ_i. The remainder of the proof is unchanged. □

In the proofs of Theorems 4 and 10 we have used the fact that V is a unitary matrix. These arguments can be somewhat simplified at the expense of proving the following improvement to Lemma 1. First we recall that the

centralizer of a matrix M, denoted $Z(M)$, is the subspace of matrices that commute with M.

Proposition 11. *Let $A_1, \ldots, A_k$ and $\alpha_2, \ldots, \alpha_k$ be as in Lemma 1; let $A = A_1 + \alpha_2 A_2 + \ldots + \alpha_k A_k$. Then*

$$Z(A) = \bigcap_{i=1}^{k} Z(A_i).$$

This result applies to real as well as complex matrices. Before giving the proof, let us explain how it can be used in Theorems 4 and 10. Applying Proposition 11 to the tuple of simultaneously diagonalizable matrices $C_2, \ldots, C_k$, we see that

$$Z(C) = \bigcap_{i=2}^{k} Z(C_i).$$

Since $C_i = S\Lambda_i S^{-1}$ and $C = S\Lambda S^{-1}$, this implies $Z(\Lambda) = \bigcap_{i=2}^{k} Z(\Lambda_i)$. Since $V \in Z(\Lambda)$, we conclude that V commutes with the Λ_i. Therefore, we have established this commutation property without using the fact that V can be taken unitary.

Proof of Proposition 11. The inclusion from right to left obviously holds for any choice of the α_i. For the converse, let $B \in Z(A)$ and assume as a first step that B is diagonalizable. Since A and B commute and both matrices are diagonalizable, there exists a transition matrix T such that $T^{-1}AT$ and $T^{-1}BT$ are diagonal. By choice of the α_i, all of the matrices $T^{-1}A_iT$ are diagonal as well. We conclude that B and A_i commute since they are simultaneously diagonalizable. To complete the proof, we just need to observe that diagonalizable matrices are dense in $Z(A)$. This follows from the fact that A itself is diagonalizable (observe indeed that the centralizer of a diagonal matrix takes a block-diagonal from, and diagonalizable matrices are dense in each block). □

Like in the complex case, we can dissociate the commutativity test in Theorem 10 from the diagonalizability test:

Theorem 12. *Let $A_1, \ldots, A_k$ be real symmetric matrices of size n and assume that the subspace $\mathcal{V}$ spanned by these matrices is nonsingular. The two following properties are equivalent:*

(i) There are diagonal matrices Λ_i and a nonsingular matrix $R \in M_n(\mathbb{R})$ such that $A_i = R\Lambda_i R^T$ for all $i = 1, \ldots, k$.

(ii) $\mathcal{V}$ satisfies the two equivalent properties of Theorem 7, and there exists an invertible $B \in \mathcal{V}$ such that the matrices $B^{-1}A_i$ ($i = 2, \ldots, k$) are all diagonalizable with real eigenvalues.

The proof is identical to the proof of Theorem 8, except that we appeal to Theorem 10 instead of Theorem 4. The criterion in Theorem 12 takes a particularly simple form when $\mathcal{V}$ contains a positive definite matrix. Before explaining this, we recall the following lemma.

Lemma 13. *Let A and B be two real symmetric matrices with B positive definite. Then $B^{-1}A$ is diagonalizable with real eigenvalues.*

Proof. Since B is positive definite, we can write $B = HH^T$ where H is a real invertible matrix. Hence $B^{-1}A = H^{-T}H^{-1}A = H^{-T}(H^{-1}AH^{-T})H^T$. Since $H^{-1}AH^{-T}$ is real symmetric it is diagonalizable with real eigenvalues. This is true of $B^{-1}A$ as well since the two matrices are similar. □

As an immediate consequence of Lemma 13 and Theorem 12 we have:

Theorem 14 (simultaneous diagonalization by congruence, positive definite case)**.** *Let $A_1, \ldots, A_k$ be real symmetric matrices of size n and assume that the subspace $\mathcal{V}$ spanned by these matrices contains a positive definite matrix. The 3 following properties are equivalent:*

(i) There exists a nonsingular matrix $A \in \mathcal{V}$ such that $A^{-1}\mathcal{V}$ is a commuting subspace.

(ii) For all nonsingular matrices $A \in \mathcal{V}$, $A_1^{-1}\mathcal{V}$ is a commuting subspace.

(iii) There are real diagonal matrices Λ_i and a nonsingular matrix $R \in M_n(\mathbb{R})$ such that $A_i = R\Lambda_i R^T$ for all $i = 1, \ldots, k$.

A related characterization can be found in [33, Theorem 3.3]. As we will see at the end of Section 3, the significance of Theorem 14 is that when a polynomial is equivalent to a sum of n real cubes, the corresponding $\mathcal{V}$ always contains a positive definite matrix.

3 The equivalence problem

In this section we review Kayal's equivalence algorithm and present our tensor-based approach. We first recall the following definition from the introduction.

Definition 15. *A polynomial $f \in \mathbb{K}[x_1, \ldots, x_n]$ is said to be equivalent to a sum of n d-th powers if $f(x) = P_d(Ax)$ where $P_d(x_1, \ldots, x_n) = x_1^d + \ldots + x_n^d$ and $A \in M_n(\mathbb{K})$ is a nonsingular matrix.*

As explained before, our equivalence algorithms in Sections 4 and 5 deal only with the case $d = 3$ (equivalence to a sum of cubes). More generally, one could ask whether two forms of degree 3 given as input are equivalent (by an invertible change of variables as above). This problem is known to

be at least as hard as graph isomorphism [2, 37] and is "tensor isomorphism complete" [24]. By contrast, equivalence of quadratic forms is "easy": it is classically known from linear algebra that two real quadratic forms are equivalent iff they have the same rank and signature (this is "Sylvester's law of inertia") and two quadratic forms over $\mathbb{C}^n$ are equivalent iff they have the same rank.[4]

Note that when $\mathbb{K} = \mathbb{R}$, Definition 15 requires the changes of variables matrix A as well as the input polynomial f to be real. It also makes sense to ask if an input $f \in \mathbb{R}[x_1, \ldots, x_n]$ is equivalent to a sum of n cubes as a complex polynomial. The following example shows that these are two distinct notions of equivalence.

Example 16. *Consider the real polynomial*

$$f(x_1, x_2) = (x_1 + ix_2)^3 + (x_1 - ix_2)^3 = 2x_1^3 - 6x_1x_2^2.$$

This decomposition shows that as a complex polynomial, f is equivalent to a sum of two cubes. Moreover, there is no decomposition as a sum of 2 cubes of real *linear forms since the above decomposition is essentially the unique decomposition of f. This follows from Corollary 20 below: in any other decomposition $f = l_1^3 + l_2^3$, the linear forms l_1 and l_2 must be scalar multiples of $x_1 + ix_2$ and $x_1 - ix_2$.*[5] *Note that this is very different from the case of degree 2 forms: any real quadratic form in n variables can be written as a linear combinations of n real squares.*

A similar example shows that for a polynomial $f(x_1, x_2)$ with rational coefficients, equivalence to a sum of two cubes over $\mathbb{Q}$ or other $\mathbb{R}$ are distinct notions:

Example 17. *Consider the rational polynomial*

$$f(x_1, x_2) = (x_1 + \sqrt{2}x_2)^3 + (x_1 - \sqrt{2}x_2)^3 = 2x_1^3 + 12x_1x_2^2.$$

This polynomial is equivalent to a sum of two cubes over $\mathbb{R}$ but not over $\mathbb{Q}$.

3.1 Review of Kayal's equivalence algorithm

Let $f \in \mathbb{C}[x_1, \ldots, x_n]$ be a homogeneous polynomial of degree $d \geq 3$. Recall that the Hessian matrix of f is the symmetric matrix of size n with entries $\partial^2 f / \partial x_i \partial x_j$, and that the Hessian determinant of f, denoted H_f, is the determinant of this matrix. Kayal's equivalence algorithm is

[4] This paper is focused on the fields of real and complex numbers but equivalence of quadratic forms has been an active topic of study for other fields as well, especially from the point of view of number theory: see for instance [45, 49].

[5] More precisely one must have $l_1 = \alpha_1(x_1 + ix_2)$, $l_2 = \alpha_2(x_1 - ix_2)$ (or vice versa) where $\alpha_1^3 = \alpha_2^3 = 1$.

based on the factorization properties of the Hessian determinant. Note that $H_f = [d(d-1)]^n (x_1 \cdots x_n)^{d-2}$ for $f = P_d$. It is shown in [37] that we still have a factorization as a product of linear forms after an invertible change of variables:

Lemma 18. *Suppose that $f(x_1, \ldots, x_n) = \sum_{i=1}^n a_i l_i(x_1, \ldots, x_n)^d$ where the l_i are linear forms and $a_i \in \mathbb{C} \setminus \{0\}$. The Hessian determinant of f is of the form*

$$H_f(x_1 \ldots, x_n) = c \prod_{i=1}^n l_i(x_1, \ldots, x_n)^{d-2}$$

for some constant c. Moreover, $c \neq 0$ iff the l_i are linearly independent.

As a consequence we have the uniqueness result of Corollary 20 below, which generalizes Corollary 5.1 in [37]. First, we need the following lemma:

Lemma 19. *Suppose that f can be written as in (1) as a sum of powers of linearly independent linear forms. If $r \geq 1$ then f is not identically zero.*

Proof. We already know this for $r = n$: Lemma 18 shows that H_f is not identically 0. For a more direct proof of the result in this case, one can simply observe that f is equivalent to P_d but P_d is not equivalent to 0.

The general case can be reduced to the case $r = n$ by setting $n - r$ of the variables of f to 0. In this way, we obtain a sum of powers of r linear forms l_i' in r variables, Moreover, it is always possible to choose the variables of f that are set to 0 so that the l_i' remain linearly independent like the forms l_i in (1). Indeed, this follows from the fact that a $r \times n$ matrix of rank r must contain a $r \times r$ submatrix of rank r. □

Corollary 20. *Suppose that $f(x_1, \ldots, x_n) = \sum_{i=1}^r l_i(x_1, \ldots, x_n)^d$ where the l_i are linearly independent linear forms. For any other decomposition $f(x_1, \ldots, x_n) = \sum_{i=1}^r \ell_i(x_1, \ldots, x_n)^d$ the linear forms ℓ_i must satisfy $\ell_i = \omega_i l_{\pi(i)}$ where ω_i is a d-th root of unity and $\pi \in S_n$ a permutation.*

Proof. Consider first the case $r = n$. By Lemma 18 and uniqueness of factorization we must have $\ell_i = c_i l_{\pi(i)}$ for some constants c_i and some permutation π. Plugging this relation into the two decompositions of f shows that:

$$\sum_{i=1}^n l_i^d = \sum_{i=1}^n c_i^d l_{\pi(i)}^d.$$

Moving all terms to the left-hand side we obtain:

$$\sum_{i=1}^n (1 - c_{\pi^{-1}(i)}^d) l_i^d = 0,$$

and Lemma 19 then implies that $c_i^d = 1$ for all i. Assume now that $r < n$. Since the l_i are linearly independent, we can extend this family into a family

of n linearly independent linear forms $l_1, \ldots, l_n$. Our two decompositions of f yield two decompositions for the polynomial $g = f + l_{r+1}^d + \cdots + l_n^d$, and we can apply the result for the case $r = n$ to g. Another way to reduce to this case would be to decrease n by setting $n - r$ of the variables of f to 0 as in the proof of Lemma 19. □

Kayal's algorithm can be summarized by the 3 following steps. It takes as input a degree d form $f \in K[x_1, \ldots, x_n]$ and determines whether f is equivalent to P_d over $\mathbb{K}$, where $K \subseteq \mathbb{K}$ are two subfields of $\mathbb{C}$. If f is equivalent to P_d it determines linearly independent linear forms $\ell_i \in \mathbb{K}[x_1, \ldots, x_n]$ such that $f(x_1, \ldots, x_n) = \sum_{i=1}^n \ell_i(x_1, \ldots, x_n)^d$. This presentation generalizes slightly [37], which focuses on the case $K = \mathbb{K} = \mathbb{C}$.

1. Check that the Hessian determinant H_f is not identically 0 and can be factorized in $\mathbb{K}[x_1, \ldots, x_n]$ as $H_f(x_1 \ldots, x_n) = c \prod_{i=1}^n l_i(x_1, \ldots, x_n)^{d-2}$ where the l_i are linear forms and $c \in \mathbb{K}$. If this is not possible, reject.

2. Try to find constants $a_i \in \mathbb{K}$ such that

$$f(x_1, \ldots, x_n) = \sum_{i=1}^n a_i l_i(x_1, \ldots, x_n)^d.$$

   If this is not possible, reject.

3. Check that all the a_i have d-th roots in $\mathbb{K}$. If this is not the case, reject. Otherwise, declare that f is equivalent to P_d over $\mathbb{K}$ and output the linear forms $\ell_i = \alpha_i l_i$ where $\alpha_i^d = a_i$ and $\alpha_i \in \mathbb{K}$.

The correctness of the algorithm follows from Lemma 18 and Corollary 20. Note in particular that if the algorithm accepts, the forms l_i must be linearly independent (or else H_f would be identically 0 by Lemma 18, and the algorithm would have rejected at step 1); and the constants a_i at step 2 are unique if they exist.

For $d = 3$, or more generally for small degree, the constants a_i at step 2 can be found efficiently by dense linear algebra assuming an algebraic model of computation (for the Turing machine model, see the comments below). For large d we can instead evaluate f and the powers l_i^d at random points (see Section 7 on linear dependencies or [37] for details).

At step 1, the Hessian determinant can be factorized by Kaltofen's algorithm [35] for the factorization of arithmetic circuits as suggested in [37], or by the black box factorization algorithm of Kaltofen and Trager [34]. These two algorithms assume access to an algorithm for the factorization of univariate polynomials (one of the algorithms in [41] reduces instead to the closely related task of matrix diagonalization). For $K = \mathbb{K} = \mathbb{C}$ one can just assume the ability to factor univariate polynomials as part of our computation model. This yields a polynomial time algorithm, which is clearly not

designed to run on a Turing machine. Another option is to take $K = \mathbb{K} = \mathbb{Q}$, and we obtain a polynomial time algorithm for the Turing machine model.

Assume now that $K = \mathbb{Q}$, $\mathbb{K} = \mathbb{C}$ and that we wish to design again an algorithm for the Turing machine model. As mentioned in Section 1.1, a natural approach would be to factor H_f symbolically at step 1, i.e., to construct an extension K' of $\mathbb{Q}$ of finite degree where we can find the coefficients of the linear forms l_i. The linear algebra computations of step 2 would then be carried out symbolically in K'. It is not clear that this approach yields a polynomial time algorithm even for $d = 3$ because these computations could possibly take place in an extension of exponential degree (recall indeed that the splitting field of a univariate polynomial of degree r may be of degree as high as $r!$). We provide polynomial time algorithms for this problem (and for $\mathbb{K} = \mathbb{R}$) in Sections 4 and 5. In order to stay closer to Kayal's original algorithm, a plausible approach would be to stop his algorithm at step 1. Note indeed that step 3 is not necessary for $\mathbb{K} = \mathbb{C}$, and it is not immediately clear whether step 2 is necessary. Namely, it is not obvious whether there are polynomials that pass the factorization test of step 1 but fail at step 2. This led us to the following question:

Question 1. *Let $f \in \mathbb{C}[x_1, \ldots, x_n]$ be a homogeneous polynomial of degree $d \geq 3$. If the Hessian determinant of f is equal to $(x_1 x_2 \cdots x_n)^{d-2}$, must f be of the form $f(x_1, \ldots, x_n) = \alpha_1 x_1^d + \cdots + \alpha_n x_n^d$?*

A positive answer would yield a polynomial time decision algorithm for the equivalence problem since the *existence* of a suitable factorization at step 1 can be decided in polynomial time [41]. Representation of polynomials by Hessian determinants has proved to be a delicate topic: see [22] for a famous mistake by Hesse about his eponymous determinant. Hesse's mistake was about polynomials with vanishing Hessian, a topic that remains of interest to this day [29]. One of the authors of [29] came accross Question 1 in an earlier version of the present paper, and managed to obtain a negative answer for many cases of interest [53]: $n \geq 2$ and $d \geq 4$ even, or $n \geq 3$ and $3 \leq d \leq n$, or $n \geq 3$ and and $d = kn$ for some $k > 1$. The approach pursued in our paper, based on simultaneous diagonalization by congruence, therefore remains the only way of testing equivalence to a sum of cubes over $\mathbb{C}$ in polynomial time. We now present this approach in detail.

3.2 Equivalence by tensor decomposition

In the remainder of this section we explain our approach to the equivalence problem. Like in most of the paper, we work in a field $\mathbb{K}$ which is either the field of real or complex numbers. Recall that we can associate to a symmetric tensor T of order 3 the homogeneous polynomial $f(x_1, \ldots, x_n) = \sum_{i,j,k=1}^{n} T_{ijk} x_i x_j x_k$. This correspondence is bijective, and the symmetric tensor associated to a homogeneous polynomial f can be obtained from the

relation:

$$\frac{\partial^3 f}{\partial x_i \partial x_j \partial x_k} = 6T_{ijk}. \tag{3}$$

The i-th slice of T is the symmetric matrix T_i with entries $(T_i)_{jk} = T_{ijk}$. By abuse of language, we will also say that T_i is the i-th slice of f. Note that (3) is the analogue of the relation

$$\frac{\partial^2 q}{\partial x_i \partial x_j} = 2Q_{ij}$$

which connects the entries of a symmetric matrix Q to the partial derivatives of the quadratic from $q(x) = x^T Qx$. Comparing these two equations shows that the matrix of the quadratic form $\partial f/\partial x_k$ is equal to $3T_k$.

Remark 21. *The slices of a polynomial of the form*

$$g(x_1, \ldots, x_n) = \alpha_1 x_1^3 + \ldots + \alpha_n x_n^3 \tag{4}$$

are the diagonal matrices $\operatorname{diag}(\alpha_1, 0, \ldots, 0), \ldots, \operatorname{diag}(0, \ldots, 0, \alpha_n)$. *Conversely, if all the slices of a degree 3 homogeneous polynomial g are diagonal then g must be of the above form (in particular, such a g is equivalent to a sum of n cubes iff the coefficients α_i are all nonzero; this follows from the fact that for $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$, any element of $\mathbb{K}$ has a cube root in $\mathbb{K}$). Indeed, the presence of any other monomial in g would yield an off-diagonal term in some slice: for the monomial $m = x_i^2 x_j$ with $i \neq j$ we have $\partial m/\partial x_i = 2x_i x_j$ and for $m = x_i x_j x_k$ with all indices distinct we have $\partial m/\partial x_i = x_j x_k$.*

In light of Definition 15, it is important to understand how slices behave under a linear change of variables. This was done for symmetric and ordinary tensors in [42, Section 2.1 and Proposition 48]. In particular, for symmetric tensors the following result can be obtained from (3):

Proposition 22. *Let g be a degree 3 form with slices $S_1, \ldots, S_n$ and let $f(x) = g(Ax)$. The slices $T_1, \ldots, T_n$ of f are given by the formula: $T_k = A^T D_k A$ where $D_k = \sum_{i=1}^n a_{ik} S_i$ and the a_{ik} are the entries of A. In particular, if g is as in (4) we have $D_k = \operatorname{diag}(\alpha_1 a_{1k}, \ldots, \alpha_n a_{nk})$.*

A similar property appears in the analysis of Jennrich's algorithm [44, Lemma 3.3.3]. The action on slices given by the formula $T_k = A^T D_k A$ in this proposition seems at least superficially related to the action on tuples of symmetric (and antisymmetric) matrices studied by Ivanyos and Qiao [30]. They consider an action of GL_n sending a tuple $(S_1, \ldots, S_m)$ to the tuple $(T_1, \ldots, T_m)$ where $T_i = A^T S_i A$. Two tuples are said to be isometric if there exists an invertible matrix A realizing this transformation. Some of the main differences with our setting are:

(i) The number of elements in our matrix tuples is the same as the dimension n of the matrices, but in their setting m and n are unrelated.

(ii) The matrices in our tuples must come from a symmetric tensor but they allow arbitrary tuples of symmetric matrices.

(iii) They act independently on each component of a matrix tuple, whereas we "mix" components with the transformation $D_k = \sum_{i=1}^{n} a_{ik} S_i$. In spite of this difference, the actions on the space of matrices spanned by the tuple's components are the same, see Lemma 23 below.

Also we note that their algorithm for isometry testing is not algebraic since it requires the construction of field extensions as explained e.g. in the paragraph on the representation of fields and field extensions in [30].[6]

Lemma 23. *Let $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ be two forms of degree 3 such that $f(x) = g(Ax)$ for some nonsingular matrix A.*

(i) If $\mathcal{U}$ and $\mathcal{V}$ denote the subspaces of $M_n(\mathbb{K})$ spanned respectively by the slices of f and g, we have $\mathcal{U} = A^T \mathcal{V} A$.

(ii) In particular, for $g = P_3$ the subspace $\mathcal{V}$ is the space of diagonal matrices and $\mathcal{U}$ is a nonsingular subspace, i.e., it is not made of singular matrices only.

Proof. Proposition 22 shows that $\mathcal{U} \subseteq A^T \mathcal{V} A$. Since $g(x) = f(A^{-1}x)$, the same argument shows that $\mathcal{V} \subseteq A^{-T} \mathcal{U} A^{-1}$. The inclusion $\mathcal{U} \subseteq A^T \mathcal{V} A$ therefore cannot be strict. The second part of the lemma follows immediately from the first and from Remark 21. □

For the next theorem, we recall from the beginning of Section 3.2 that one may take either $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$.

Theorem 24. *A degree 3 form $f \in \mathbb{K}[X_1, \ldots, X_n]$ is equivalent to a sum of n cubes if and only if its slices $T_1, \ldots, T_n$ span a nonsingular matrix space and the slices are simultaneously diagonalizable by congruence, i.e., there exists an invertible matrix $Q \in M_n(\mathbb{K})$ such that the n matrices $Q^T T_i Q$ are diagonal.*

Proof. Let $\mathcal{U}$ be the space spanned by $T_1, \ldots, T_n$. If f is equivalent to a sum of n cubes, Proposition 22 shows that the slices of f are simultaneously diagonalizable by congruence and Lemma 23 shows that $\mathcal{U}$ is nonsingular.

Let us show the converse. Since the slices are simultaneously diagonalizable by congruence, there are diagonal matrices Λ_k and a nonsingular matrix

[6] An algebraic algorithm does not require the construction of field extensions since by definition all operations take place in the ground field. In [30] they only need to construct extensions of polynomially bounded degree. As explained in Section 3.1, this is not clear for Kayal's algorithm.

$R \in M_n(\mathbb{K})$ such that $T_k = R\Lambda_k R^T$ for all $k = 1, \ldots, n$. Let $g(x) = f(R^{-T}x)$. By Proposition 22 the slices of g are linear combinations of the Λ_k, i.e., they are all diagonal. By Remark 21, g must be as in (4). It therefore remains to show that the coefficients α_i are all nonzero. This must be the case due to the hypothesis on $\mathcal{U}$. Indeed, this hypothesis implies that the matrix space $\mathcal{V}$ spanned by the slices of g is nonsingular (apply again Lemma 23, this time in the other direction). But if some α_i vanishes, $\mathcal{V}$ is included in the space of diagonal matrices with a 0 in the i-th diagonal entry. □

Corollary 25. *Let f be a degree 3 form with slices $T_1, \ldots, T_n$ and assume that T_1 is nonsingular. Then f is equivalent to a sum of n cubes if and only if the $n-1$ matrices $T_1^{-1}T_k$ $(k = 2, \ldots, n)$ commute and are diagonalizable over $\mathbb{K}$.*

Proof. This follows from Theorem 24 as well as Theorem 4 for $\mathbb{K} = \mathbb{C}$ and Theorem 10 for $\mathbb{K} = \mathbb{R}$. □

We conclude this section with an alternative characterization of equivalence to a sum of cubes for the field of real numbers.

Theorem 26. *Let f be a real form of degree 3 and let $\mathcal{V}$ be the subspace of $M_n(\mathbb{R})$ spanned by the slices of f. The 3 following properties are equivalent:*

(i) f is equivalent as a real polynomial to a sum of n cubes.

(ii) There exist two invertible matrices $A, B \in \mathcal{V}$ such that $A^{-1}\mathcal{V}$ is a commuting subspace and B is positive definite.

(iii) $\mathcal{V}$ contains a positive definite matrix, and $A^{-1}\mathcal{V}$ is a commuting subspace for any invertible matrix $A \in \mathcal{V}$.

Proof. Suppose that f is equivalent to a sum of n cubes, i.e., $f(x) = P_3(Qx)$ where $Q \in M_n(\mathbb{R})$ is invertible. We have seen in Lemma 23 that the slices of P_3 span the space $\mathcal{D}$ of diagonal matrices, and that those of f span $Q^T\mathcal{D}Q$. The latter span contains the positive definite matrix $B = Q^TQ$. Moreover, according to Proposition 22 the slices of f are simultaneously diagonalizable by congruence. By Theorem 12, $A^{-1}\mathcal{V}$ is a commuting subspace for any invertible matrix $A \in \mathcal{V}$. Hence we have shown that (i) implies (iii). That (iii) implies (ii) is clear since $\mathcal{V}$ is nonsingular (by hypothesis, it contains a positive definite matrix). Finally, let us show that (ii) implies (i). By hypothesis, $\mathcal{V}$ contains a positive definite matrix B hence we can apply Theorem 14. It follows that the slices are simultaneously diagonalizable by congruence. By Theorem 24, f must be equivalent to a sum of n cubes. □

Compared to Theorem 24 or Corollary 25, Theorem 26 does not involve any diagonalizability test. One can check that $\mathcal{V}$ contains a positive definite matrix using semi-definite programming. Unfortunately, no efficient

algebraic algorithm is known for semi-definite programming (famously, this is already an open problem for linear programming). For this reason, the equivalence algorithms of this paper will be based on Corollary 25 rather than Theorem 26.

4 Randomized equivalence algorithm

As a test for equivalence to a sum of n cubes, Corollary 25 is not quite satisfactory due to the hypothesis on T_1 (note indeed that this hypothesis is not even satisfied by $f = P_3$). This restriction can be overcome by performing a random change of variables before applying Corollary 25. This yields the following simple randomized algorithm with one-sided error. The input is a degree 3 form $f \in \mathbb{K}[x_1, \dots, x_n]$. We recall from Section 3.2 that $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$ (except in Proposition 27 where any field of characteristic 0 is allowed).

1. Pick a random matrix $R \in M_n(\mathbb{K})$ and set $h(x) = f(Rx)$.
2. Let $T_1, \dots, T_n$ be the slices of h. If T_1 is singular, reject. Otherwise, compute $T_1' = T_1^{-1}$.
3. If the matrices $T_1' T_k$ commute and are all diagonalizable over $\mathbb{K}$, accept. Otherwise, reject.

Before proving the correctness of this algorithm, we explain how the diagonalizability test at step 3 can be implemented efficiently with an algebraic algorithm. This can be done thanks to the following classical result from linear algebra (see e.g. [28, Corollary 3.3.8] for the case $\mathbb{K} = \mathbb{C}$).

Proposition 27. *Let $\mathbb{K}$ be a field of characteristic 0 and let χ_M be the characteristic polynomial of a matrix $M \in M_n(\mathbb{K})$. Let $P_M = \chi_M / \gcd(\chi_M, \chi_M')$ be the squarefree part of χ_M. The matrix M is diagonalizable over $\overline{\mathbb{K}}$ iff $P_M(M) = 0$. Moreover, in this case M is diagonalizable over $\mathbb{K}$ iff all the roots of P_M lie in $\mathbb{K}$.*

Over the field of complex numbers it therefore suffices to check that $P_M(M) = 0$. Over $\mathbb{R}$, we need to check additionally that all the roots of P_M are real. This can be done for instance with the help of Sturm sequences, which can be used to compute the number of roots of a real polynomial on any real (possibly unbounded) interval. Alternatively, the number of real roots of a real polynomial can be obtained through Hurwitz determinants [46, Corollary 10.6.12], and is given by the signature of the Hermite quadratic form [3, Theorem 4.48]. The arithmetic cost of these methods is polynomially

bounded, and they can also be implemented to run in polynomial time in the bit model.[7]

Theorem 28. *If an input $f \in \mathbb{K}[X_1, \ldots, X_n]$ is accepted by a run of the above randomized algorithm then f must be equivalent to a sum of n cubes.*

Conversely, if f is equivalent to a sum of n cubes then f will be accepted with high probability over the choice of the random matrix R at step 1. More precisely, if the entries r_{ij} are chosen independently at random from a finite set S the input will be accepted with probability at least $1 - 2n/|S|$.

Proof. Assume that f is accepted for some choice of $R \in M_n(\mathbb{C})$. Since T_1 is invertible, it follows from Proposition 22 that R must be invertible as well. Moreover, h must be equivalent to a sum of n cubes by Corollary 25. The same is true of f since $f(x) = h(R^{-1}x)$.

For the converse, assume that f is equivalent to a sum of n cubes. We can obtain the slices T_k of h from the slices S_k of f by Proposition 22, namely, we have $T_k = R^T D_k R$ where $D_k = \sum_{i=1}^n r_{ik} S_i$ and the r_{ik} are the entries of R. Therefore T_1 is invertible iff R and D_1 are invertible. By Lemma 23.(ii) there is a way to choose the entries r_{i1} so that D_1 is invertible. In fact, D_1 will be invertible for most choices of these entries. This follows from the fact that as a polynomial in the entries $r_{11}, \ldots, r_{n1}$, $\det(D_1)$ is not identically zero. Therefore, by the Schwartz-Zippel lemma D_1 will fail to be invertible with probability at most $n/|S|$. Likewise, R will fail to be invertible with probability at most $n/|S|$ and the result follows from the union bound. □

Remark 29. *In Theorem 28 and in the corresponding algorithm, we can reduce the amount of randomness by picking random matrices R of the following special form: R is lower triangular with 1's on the diagonal (except possibly for r_{11}), $r_{11}, \ldots, r_{n1}$ are drawn independently and uniformly from S, and all the other entries are set to 0. The same analysis as before shows that D_1 will fail to be invertible with probability at most $n/|S|$. Moreover, R will fail to be invertible with probability at most $1/|S|$ since $\det(R) = r_{11}$. By the union bound, f will be accepted with probability at least $1 - (n+1)/|S|$.*

We will use a similar construction in the deterministic algorithm of the next section.

[7] For Sturm sequences this is not obvious because in a naive implementation, the bit size of the numbers involved may grow exponentially. There is however an efficient implementation based on subresultants [3]. The same issue of coefficient growth already occurs in the computation of the gcd of two polynomials, and can also be solved with subresultants.

5 Deterministic equivalence algorithm

In the analysis of our randomized algorithm we have invoked the Schwartz-Zippel lemma to argue that a polynomial of the form

$$H(r_1, \ldots, r_n) = \det(r_1 S_1 + \ldots + r_n S_n)$$

does not vanish for most of the random choices $r_1, \ldots, r_n$ (recall from the proof of Theorem 28 that $S_1, \ldots, S_n$ denoted the slices of f). In this section we will obtain obtain our deterministic equivalence algorithm by derandomizing this step. Namely, we will use the fact that we are not trying to solve an arbitrary instance of symbolic determinant identity testing: as it turns out, the polynomial H can be factored as a product of linear forms. This fact was already at the heart of Kayal's equivalence algorithm. Indeed, his algorithm is based on the factorization of the Hessian determinant of f [37, Lemma 5.2] and as pointed out in [42], the symbolic matrix $r_1 S_1 + \ldots + r_n S_n$ is a constant multiple of the Hessian. The point where we depart form Kayal's algorithm is that we do not explicitly factor H as a product of linear forms (recall indeed that this is not an algebraic step). Instead, we will use the *existence* of such a factorization to find deterministically a point where H does not vanish. We can then conclude as in the previous section.

First, we formally state this property of H as a lemma and for the sake of completeness we show that it follows from Proposition 22 (one can also make this argument in the opposite direction, see Section 2.1 of [42] for details).

Lemma 30. *Let f be a degree 3 form with slices $S_1, \ldots, S_n$ and let $H(x_1, \ldots, x_n) = \det(x_1 S_1 + \ldots + x_n S_n)$. If f is equivalent to a sum of n cubes then H is not identically 0 and can be factored as a product of n linear forms.*

Proof. Let A be the invertible matrix such that $f(x) = P_3(Ax)$. By Proposition 22, $H(x) = (\det A)^2 \det D(x)$ where

$$D(x) = \sum_{k=1}^{n} x_k D_k = \operatorname{diag}(a_{11}x_1 + \cdots + a_{1n}x_n, \ldots, a_{n1}x_1 + \cdots + a_{nn}x_n).$$

This gives the required factorization. In particular, H is nonzero since A is invertible. □

The non vanishing of H means that the slices span a nonsingular matrix space. We have given in Theorem 24 a slightly different proof of the fact that this space is indeed nonsingular when f is equivalent to a sum of n cubes. By Lemma 30, the zero set of H is a union of n hyperplanes. We can avoid the union of any finite number of hyperplanes by a standard construction involving the *moment curve* $\gamma(t) = (1, t, t^2, \ldots, t^{n-1})$.

Lemma 31. *Let $M \subseteq \mathbb{C}^n$ be a set of $(n-1)p+1$ points on the moment curve. For any set of p hyperplanes $H_1, \ldots, H_p \subseteq \mathbb{C}^n$ there is at least one point of M which does not belong to any of the H_i.*

Proof. Let $l_i(x_1, \ldots, x_n) = 0$ be the equation of H_i. The moment curve has at most $n-1$ intersections with H_i since $l_i(1, t, t^2, \ldots, t^{n-1})$ is a nonzero polynomial of degree $n-1$. For the p hyperplanes we therefore have a grand total of $p(n-1)$ intersection points at most. □

The size of M in this lemma is the smallest that can be achieved in such a blackbox construction. Indeed, for any set of $(n-1)p$ points one can always find a set of p hyperplanes which covers them all.

We can now describe our deterministic algorithm. As in Section 4 the input is a degree 3 form $f(x_1, \ldots, x_n)$ with slices $S_1, \ldots, S_n$.

1. Pick an arbitrary set M of $n(n-1)+1$ points on the moment curve.
2. Enumerate the elements of M to find a point $r = (1, r_2, \ldots, r_n) \in M$ such that the matrix $D_1 = S_1 + r_2 S_2 \ldots + r_n S_n$ is invertible. If there is no such point, reject.
3. Construct the following matrix $R \in M_n(\mathbb{K})$: R is lower triangular with 1's on the diagonal, $r_{21} = r_2, \ldots, r_{n1} = r_n$ and all the other entries are set to 0.
4. Compute $h(x) = f(Rx)$, the slices $T_1, \ldots, T_n$ of h and $T_1' = T_1^{-1}$.
5. If the matrices $T_1' T_k$ commute and are all diagonalizable, accept. Otherwise, reject.

Theorem 32. *A degree 3 form $f(x_1, \ldots, x_n)$ is accepted by the above algorithm if and only if f is equivalent to a sum of n cubes.*

Proof. As a preliminary observation, we note that if the algorithm reaches step 4 the matrix T_1' is well-defined since T_1 is invertible. Indeed, we have seen in the proof of Theorem 28 that $T_1 = R^T D_1 R$; moreover, D_1 is invertible since the algorithm has not failed at step 2 and R is clearly invertible as well.

Suppose now that an input $f(x_1, \ldots, x_n)$ is accepted by the algorithm. The same argument as in the proof of Theorem 28 shows that f is equivalent to a sum of n cubes. Namely, h must be equivalent to a sum of n cubes by Corollary 25. The same is true of f since R is an invertible matrix.

For the converse, suppose that an input $f(x_1, \ldots, x_n)$ is equivalent to a sum of n cubes. By Lemma 30 and Lemma 31, there exists a point $r \in M$ where the polynomial $H(r) = \det(r_1 S_1 + \ldots + r_n S_n)$ does not vanish. As a result, the algorithm will not reject at step 2. Since the matrix R constructed at step 3 is invertible, the polynomial h at step 4 is equivalent to a sum of n cubes and the algorithm will accept at step 5 by Corollary 25. □

Remark 33. *Some of the results in the paper by Ivanyos and Qiao [30] mentioned after Proposition 22 are motivated by an application to symbolic determinant identity testing (SDIT). In our setting we only need to consider very simple determinants (as explained at the beginning of this section, they factor as a product of linear forms). As a result we can use the simple black box solution provided by Lemma 31. More connections between group actions and SDIT can be found in [21, 31, 32].*

6 Polynomial Identity Testing

It is a basic fact that black box PIT for a class of polynomials $\mathcal{C}$ is equivalent to constructing a hitting set for $\mathcal{C}$, i.e., a set of points H such that every polynomial in $\mathcal{C}$ which vanishes on all points of H must vanish identically. Indeed, from a hitting set we obtain a black box PIT algorithm by querying the input polynomial f at all points of H. Conversely, for any black box PIT algorithm the set of points queried on the input $f \equiv 0$ must form a hitting set. Note that the validity of this simple argument depends on the hypothesis that $0 \in \mathcal{C}$ (otherwise we can declare that $f \not\equiv 0$ without making any query).

In this section we first consider the following scenario. An algorithm is provided with black box access to a polynomial f that is either identically 0 or equivalent to P_d, and must decide in which of these two categories its input falls (note that these are indeed two disjoint cases). This is equivalent to constructing a hitting set for the equivalence class of P_d, a task that we carry out in Section 6.1. Then in Section 6.2 we generalize this hitting set construction to a larger class of polynomials, namely, those that can be written as sums of d-th powers of linearly independent linear forms.

6.1 A hitting set for the equivalence class of P_d

Here we construct a polynomial size hitting set for the set of polynomials $f \in \mathbb{C}[X_1, \ldots, X_n]$ that are equivalent to P_d. Let $S = \{s_0, s_1, \ldots, s_d\}$ be a set of $d+1$ complex numbers with $s_0 = 0$. We denote by S_i the set of points $(x_1, \ldots, x_n) \in \mathbb{C}^n$ with $x_i \in S$ and all other coordinates equal to 0. Pick an arbitrary nonzero point $p \in \mathbb{C}^n$, and form the set $G_p = p + \bigcup_{i=1}^{n} S_i$, of size $nd + 1$.

Proposition 34. *For any $d \geq 2$ and any nonzero point $p \in \mathbb{C}^n$, G_p is a hitting set for the set of polynomials $f \in \mathbb{C}[X_1, \ldots, X_n]$ that are equivalent to P_d.*

Proof. Let $f(x) = P_d(Ax)$ where A is an invertible matrix. The proof is based on the fact that the gradient $\nabla f = (\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$ does not vanish anywhere except at $x = 0$. Indeed, this property clearly holds true

for P_d, and is preserved by an invertible change of variables since

$$\nabla f(x) = A^T \nabla P_d(Ax) \tag{5}$$

by the chain rule. In particular, $\nabla f(p) \neq 0$ since $p \neq 0$. This implies that f does not vanish on all of G_p. Indeed, if f vanishes on $p + S_i$ then f vanishes on the whole line going through $p + S_i$, hence $\frac{\partial f}{\partial x_i}(p) = 0$.

□

Let K be a subfield of $\mathbb{C}$. The same construction yields a hitting set for the set of polynomials in $K[x_1, \ldots, x_n]$ that are equivalent to a polynomial of the form $\sum_{i=1}^{n} a_i x_i^d$ where $a_i \in K \setminus \{0\}$. Such polynomials are indeed equivalent to P_d as complex polynomials.

6.2 Fewer powers

We now give a black box PIT algorithm for a bigger class of polynomials, namely, those that can be written as sums of d-th powers of linearly independent linear forms. These polynomials therefore admit decompositions as in (1) where the forms l_i are linearly independent and the number $r \leq n$ of forms in the decomposition is unknown. We will generalize the approach from Section 6.1. In particular, we will see that the gradient of f does not vanish outside of a certain (unknown) linear subspace V. We can find deterministically a point outside of V with the help of the moment curve like in Section 5. This leads to the construction of the following hitting set: for an arbitrary set M of n points on the moment curve, we construct the union of the G_p's as p ranges over M (recall that G_p is the hitting set of Section 6.1). This set $G \subseteq \mathbb{C}^n$ is of size $n(nd + 1)$.

Theorem 35. *For any $d \geq 2$, G is a hitting set for the set of polynomials $f \in \mathbb{C}[X_1, \ldots, X_n]$ that can be written as sums of d-th powers of linearly independent linear forms.*

Proof. Suppose that f can be written as a sum of r d-th powers for some $r \geq 1$. We have $f(x) = P_{d,r}(Ax)$ where A is an invertible matrix and

$$P_{d,r}(x_1, \ldots, x_n) = x_1^d + \cdots + x_r^d.$$

We will use the fact that the gradient of f vanishes only on a (proper) linear subspace V of dimension $n - r$. This property clearly holds true for $P_{d,r}$, and it is preserved for f since we now have

$$\nabla f(x) = A^T \nabla P_{d,r}(Ax)$$

instead of (5). By Lemma 31, M contains at least one point p lying outside of V. At this point $\nabla f(p) \neq 0$, and the same argument as in the proof of Proposition 34 shows that f does not vanish on G_p. □

7 Linear dependencies, essential variables and Lie algebras

In this section we build on the results from Section 6 to derandomize several algorithms from [37, 38]. We begin in Section 7.1 with the computation of linear dependencies between polynomials. Then we give applications to the minimization of the number of variables in sums of powers of linear forms (in Section 7.2), and to the computation of Lie algebras of products of linear forms (in Section 7.3). This leads to the derandomization of a factorization algorithm from [41] and of the equivalence algorithm by Kayal [37] described in Section 3.1.

7.1 From black box PIT to linear dependencies

We first recall from [37] the notion of *linear dependencies among polynomials*. It has found applications to the elimination of redundant variables [37], the computation of the Lie algebra of a polynomial [38], the reconstruction of random arithmetic formulas [25], full rank algebraic programs [39] and non-degenerate depth 3 circuits [40].

Definition 36. *Let* $\mathbf{f} = (f_1, \ldots, f_m)$ *be a tuple of* m *polynomials of* $K[X_1, \ldots, X_n]$. *The space of linear dependencies of* $\mathbf{f}$*, denoted* $\mathbf{f}^\perp$*, is the space of all vectors* $v = (v_1, \ldots, v_m) \in K^m$ *such that* $v_1 f_1 + \cdots + v_m f_m$ *is identically 0.*

As a computational problem, the POLYDEP problem consists of finding a basis of $\mathbf{f}^\perp$ for a tuple $\mathbf{f}$ given as input. If the f_i are verbosely given as sum of monomials, this is a simple problem of linear algebra. The problem becomes more interesting if the f_i are given by arithmetic circuits or black boxes. In Section 7.1 we present a simple and general relation between this problem and black box PIT.

A natural approach to POLYDEP consists of evaluating the f_j at certain points $a_1, \ldots, a_k$ of K^n to form a $k \times m$ matrix M with the $f_j(a_i)$ as entries. Note that $\mathbf{f}^\perp \subseteq \ker(M)$ for any choice of the evaluation points. We would like this inclusion to be an equality since this will allow to easily compute a basis of $\mathbf{f}^\perp$. This motivates the following definition.

Definition 37. *The points* $a_1, \ldots, a_k$ *form a* hitting set for the linear dependencies of $\mathbf{f}$ *if the above matrix* $M = (f_j(a_i))_{1 \le i \le k, 1 \le j \le m}$ *satisfies* $\mathbf{f}^\perp = \ker(M)$.

Kayal [37] showed (without using explicitly this terminology) that if $k = m$ and the a_i are chosen at random, a hitting set for the linear dependencies of $\mathbf{f}$ will be obtained with high probability.

Here we point out that constructing deterministically a hitting set for the linear dependencies of $\mathbf{f}$ is *equivalent* to solving black box PIT for the family of polynomials in $\mathrm{Span}(\mathbf{f})$ (the space of linear combinations of $f_1, \ldots, f_m$):

Proposition 38. *Let* $\mathbf{f} = (f_1, \ldots, f_m)$ *be a tuple of* m *polynomials of* $K[X_1, \ldots, X_n]$. *For any tuple* $(a_1, \ldots, a_k)$ *of* k *points of* K^n, *the two following properties are equivalent:*

(i) The points $a_1, \ldots, a_k$ *form a hitting set for the linear dependencies of* $\mathbf{f}$.

(ii) They form a hitting set for $\mathrm{Span}(\mathbf{f})$.

Proof. This is immediate from the definitions. Suppose indeed that (i) holds, and that some polynomial $f = v_1 f_1 + \ldots + v_m f_m$ of $\mathrm{Span}(\mathbf{f})$ vanishes at all of the a_i. This means that $v \in \ker M$, hence $v \in \mathbf{f}^\perp$ by (i). We conclude that f is identically 0 and (ii) holds.

To prove the converse we can take the same steps in reverse. Suppose that (ii) holds and that $v \in \ker M$. This means that $f = v_1 f_1 + \ldots + v_m f_m$ vanishes at all the a_i, hence f is identically 0 by (ii). We have shown that $v \in \mathbf{f}^\perp$, i.e., $\mathbf{f}^\perp = \ker M$. □

In Section 7.2 we will use this observation and the black box PIT algorithm of Section 6.2 to minimize the number of variables in sums of powers of linearly independent linear forms. In Section 7.3 we give an application to the computation of Lie algebras and factorization into products of linear forms.

7.2 Minimizing variables

We first recall the notion of *redundant* and *essential* variables studied by Carlini [9] and Kayal [37].

Definition 39. *A variable* x_i *in a polynomial* $f(x_1, \ldots, x_n)$ *is redundant if* f *does not depend on* x_i, *i.e.,* x_i *does not appear in any monomial of* f.

We say that f *has* t *essential variables if* t *is the smallest number for which there is an invertible matrix of size* n *such that* $f(Ax)$ *depends on* t *variables only.*

A randomized algorithm for minimizing the number of variables is given in [37, Theorem 4.1]. More precisely, if the input f has t essential variables the algorithm finds (with high probability) an invertible matrix A such that $f(Ax)$ depends on its first t variables only. It is based on the observation from [9, 37] that $t = n - \dim(\partial f)^\perp = \dim(\partial f)$ where ∂f denotes the tuple of n partial derivatives $\partial f / \partial x_i$ (and $\dim(\partial f)$ denotes the dimension of the spanned subspace). As recalled in Section 7.1, a basis of the space of linear dependencies $(\partial f)^\perp$ can be found by a randomized algorithm from [37]. Moreover, a suitable invertible matrix A is easily found from such a basis by completing it into a basis of the whole space K^n (see appendix B of [37] for details).

Example 40. *If f can be written as a sum of r powers of linearly independent linear forms then the number of essential variables of f is equal to r. This is clear for $f(x_1, \ldots, x_n) = x_1^d + \cdots + x_r^d$ since ∂f is spanned by $x_1^{d-1}, \ldots, x_r^{d-1}$. In the general case, f is equivalent to $x_1^d + \cdots + x_r^d$ and two equivalent polynomials have the same number of essential variables.*

The next proposition is a consequence of the above variable minimization algorithm. The input f to the algorithm of Proposition 41 can be described by an arithmetic circuit like in [37] or more generally by a black box. Here we assume (in contrast with Sections 4 and 5) that we have access to an oracle for the factorization of univariate polynomials. This is a prerequisite for running Kaltofen's factorization algorithms for the arithmetic circuit [35] and black box models [34].

Proposition 41. *There is a randomized polynomial time algorithm that decides whether a homogeneous polynomial $f(x_1, \ldots, x_n)$ can be written as in (1) as a sum of powers of linearly independent linear forms.*

Proof. First compute the number r of essential variables in f using the randomized algorithm from [37], and make the corresponding change of variables to obtain a polynomial $g(x_1, \ldots, x_r)$. Then test whether g is equivalent to $x_1^d + \cdots + x_r^d$ using the equivalence algorithm from [37]. To prove that this algorithm is correct, we show that f can be written as a sum of r powers of linearly independent linear forms if and only if g can be written in such a form. Let $A \in \mathbb{K}^{n\times n}$ be an invertible matrix such that $f(Ax) = g(x_1, \ldots, x_r)$ for all $x \in \mathbb{K}^n$. Suppose that $g(Bx) = x_1^d + \cdots + x_r^d$ for some invertible matrix $B \in \mathbb{K}^{r\times r}$. If we denote $C = \begin{pmatrix} B & 0 \\ 0 & I \end{pmatrix} \in \mathbb{K}^{n\times n}$ where $I \in \mathbb{K}^{(n-r)\times(n-r)}$ is the identity matrix, then the matrix AC is invertible and we have $f(ACx) = g\big(B(x_1, \ldots, x_r)^T\big) = x_1^d + \cdots + x_r^d$ for every $x \in \mathbb{K}^n$. Conversely, suppose that there exists an invertible matrix $B \in \mathbb{K}^{n\times n}$ such that $f(Bx) = x_1^d + \cdots + x_{r'}^d$. We have $r' = r$ by Example 40. Moreover, for all $x \in \mathbb{K}^n$ we have $x_1^d + \cdots + x_r^d = f(Bx) = f(AA^{-1}Bx) = g\big((A^{-1}Bx)_1, \ldots, (A^{-1}Bx)_r\big)$. Thus, by setting $x_{r+1} = \ldots = x_n = 0$, we get $x_1^d + \cdots + x_r^d = g(Cx)$ for all $x \in \mathbb{K}^r$, where $C \in \mathbb{K}^{r\times r}$ is the submatrix of $A^{-1}B$ obtained by taking the first r rows and columns. To show that C is invertible, suppose that this is not the case. Then, there exists an invertible matrix $D \in \mathbb{K}^{r\times r}$ such that $D(0, \ldots, 0, 1)^T \in \ker C$. Hence, for every $x \in \mathbb{K}^r$ we have $g(CD(x_1, \ldots, x_{r-1}, 0)^T) = g(CDx) = h(Dx)$, where $h(x) = x_1^d + \cdots + x_r^d$. In particular, h has less than r essential variables, which gives a contradiction with Example 40. Therefore, the matrix C is invertible. □

In this algorithm it is essential to compute the number of essential variables in f before calling the equivalence algorithm from [37]. Indeed, this

algorithm is based on the factorization of the Hessian determinant of f; but H_f is identically 0 for any polynomial with fewer than n essential variables. Hence looking at $\det H_f$ does not yield any useful information for $r < n$.

Remark 42. *We can minimize the number of variables of a degree 3 form $f(x_1, \ldots, x_n)$ in deterministic polynomial time using dense linear algebra. Indeed, as pointed out in Section 7.1 this is true more generally for the POLY-DEP problem with inputs that are verbosely given as sums of monomials.*[8] *Combining this observation with the deterministic equivalence algorithm from Section 5 we obtain , as in Proposition 41, a deterministic algorithm to decide whether a degree 3 form can be written as in (1) as a sum of cubes of linearly independent (real or complex) linear forms.*

The second result of this section is the following derandomization of Kayal's algorithm for finding the number of essential variables, under the assumption that the input polynomial is a sum of powers of independent linear forms:

Theorem 43. *Let $f(x_1, \ldots, x_n)$ be a homogoneous polynomial of degree d and let $\{a_1, \ldots, a_k\}$ be the hitting set of Theorem 35 corresponding to polynomials of degree $d-1$ in n variables (recall that it is of size $O(n^2 d)$). Let f_j be the partial derivative $\partial f/\partial x_j$. We consider like in Section 7.1 the matrix $M = (f_j(a_i))_{1 \leq i \leq k, 1 \leq j \leq n}$.*

If f can be written as in (1) as a sum of r powers of linearly independent linear forms then $\ker M = (\partial f)^{\perp}$. In particular, the number of essential variables of such an f can be computed deterministically from a black box for f by the formula: $r = n - \dim \ker M$.

Proof. We recall that a black box for f_j can be easily obtained from a black box for f by polynomial interpolation. It therefore remains to show that $\ker M = (\partial f)^{\perp}$. By Proposition 38 it suffices to show that $\{a_1, \ldots, a_k\}$ is a hitting set for $\mathrm{Span}(\partial f)$. This is clear from the definition of $\{a_1, \ldots, a_k\}$ since the elements of $\mathrm{Span}(\partial f)$ can be written as linear combinations of at most r $(d-1)$-th powers of linearly independent linear forms (namely, the same forms that appear in the decomposition of f). □

7.3 Lie algebras and polynomial factorization

One can associate to a polynomial $f \in K[X_1, \ldots, X_n]$ the group of invertible $n \times n$ matrices A that leave f invariant, i.e., such that $f(Ax) = f(x)$. One can in turn associate to this matrix group its Lie algebra. This is a linear subspace of $M_n(K)$, which we call simply "the Lie algebra of f." It turns out that elements of this Lie algebra correspond to linear dependencies between

[8] Variable minimization for forms of degree 3 is also studied in Saxena's thesis [48, Proposition 3.1]. He attributes the corresponding deterministic algorithm to Harrison [26].

the n^2 polynomials $x_j \frac{\partial f}{\partial x_i}$. A proof can be found in [38, Section 7.2], and we will take this characterization as our definition of the Lie algebra for the purpose of this paper:

Definition 44. *The Lie algebra of a polynomial $f \in K[x_1, \ldots, x_n]$ is the subspace of all matrices $C \in M_n(K)$ that satisfy the identity:*

$$\sum_{i,j \in [n]} c_{ij} x_j \frac{\partial f}{\partial x_i} = 0 \,.$$

A randomized algorithm for the computation of the Lie algebra was given in [38], with applications to the reconstruction of affine projections of polynomials. In this section we study the deterministic computation of Lie algebras of polynomials, a topic which has not been studied in the literature as far as we know.

The Lie algebra of a homogenous polynomial f consists of all matrices of $M_n(K)$ if and only if f is identically 0. This shows that one cannot hope to compute the Lie algebra in deterministic polynomial time without derandomizing Polynomial Identity Testing (and these two problems are in fact equivalent in the black box setting by Proposition 38). Nevertheless, it makes sense to search for deterministic algorithms for specific classes of polynomials. We take a first step in this direction in Theorem 49, for polynomials that factor as products of linear forms. Taking again our cue from Proposition 38, we will do this by constructing a hitting set for a related family of polynomials. As it turns out, it is convenient to first design a hitting set for a certain family of "simple" rational functions. Those are defined as follows:

Definition 45. *Let $p_1, \ldots, p_m \in \mathbb{C}^n, q_1, \ldots, q_m \in \mathbb{C}^n \setminus \{0\}$ be a collection of $2m$ vectors in $\mathbb{C}^n$. Furthermore, let $\mathcal{H} = \bigcup_{i=1}^m \{x \in \mathbb{C}^n \colon \langle q_i, x \rangle = 0\}$.*

We associate to this collection of $2m$ vectors an oracle which for any $x \in \mathbb{C}^n$ returns the value

$$f(x) = \begin{cases} \displaystyle\sum_{i=1}^{m} \frac{\langle p_i, x \rangle}{\langle q_i, x \rangle} & \text{if } x \notin \mathcal{H}, \\ \\ \text{NaN} & \text{otherwise.} \end{cases} \tag{6}$$

In the commutative setting, there does not seem to be a lot of literature on rational identity testing (there is however the deep result that rational identity testing can be done in deterministic polynomial time in the non-commutative setting [21]). For (commutative) arithmetic circuits with divisions, deterministic rational identity testing is easily seen to be equivalent to PIT for ordinary (division free) arithmetic circuits. Nevertheless, it makes sense to investigate it for specific families of rational functions such as those in Definition 45.

Remark 46. *According to the above definition, the oracle returns NaN when we evaluate f on a point x where $q_i(x) = 0$, and this remains true even if the corresponding vector p_i is equal to 0. This convention is useful for the proof of Proposition 47 below.*

For every $n, k \in \mathbb{N}$, let $\mathcal{P}(n,k) \subset \mathbb{C}^n$ be the set of $k(n-1)+1$ points defined as

$$\mathcal{P}(n,k) = \{(1,1,\ldots,1),(1,2,2^2,\ldots,2^{n-1}),\\ \ldots,(1,k(n-1)+1,(k(n-1)+1)^2,\ldots,(k(n-1)+1)^{n-1})\}\,.$$

Recall from Lemma 31 that these points cannot be all contained in a union of k hyperplanes since they lie on the moment curve. Moreover, for every $m, n \geq 1$ let

$$\Lambda(m,n) = \{u + \lambda v \colon v \in \mathcal{P}(n,m), u \in \mathcal{P}(n,m^2) \cup \{0\}, \lambda \in [2m+1]\}\,.$$

The next result shows that the set $\Lambda(m,n)$ is a hitting set for the rational functions of Definition 45.

Proposition 47. *The function $f(x)$ in (6) is equal to 0 for every $x \notin \mathcal{H}$ if and only if $f(x) \in \{0, \mathrm{NaN}\}$ for every $x \in \Lambda(m,n)$.*

Proof. The "only if" implication is trivial. To prove the other direction, suppose that $f(x) \in \{0, \mathrm{NaN}\}$ for every $x \in \Lambda(m,n)$. First we observe that it is enough to assume that $\{q_1,\ldots,q_m\}$ are pairwise linearly independent. Indeed, if $q_i = \mu q_j$ for some $i, j \in [m]$ and $\mu \in \mathbb{C} \setminus \{0\}$, then we can put $\tilde{p}_i = p_i + \mu p_j$, replace p_i by $\tilde{p}_i$ and forget p_j and q_j. This does not change the function f (in particular, by Remark 46 the domain of definition of f is unchanged). By repeating this procedure, we can write f in such a way that the denominators are pairwise linearly independent (and their number m does not increase).

From now on, we assume that $\{q_1,\ldots,q_m\}$ are pairwise linearly independent. By Lemma 31, there exists $v \in \mathcal{P}(n,m)$ such that $\langle q_i, v\rangle \neq 0$ for all $i \in [m]$. For every $i \in [m]$, denote $a_i = \langle p_i, v\rangle / \langle q_i, v\rangle \in \mathbb{C}$. We will show that $p_i = a_i q_i$ for all $i \in [m]$. To do so, suppose that there exists at least one i such that $p_i - a_i q_i \neq 0$. For every pair $(i,j) \in [m]^2$ such that $i \neq j$ let $d_{ij} = \langle q_i, v\rangle / \langle q_j, v\rangle$. Using Lemma 31 one more time, there exists $u \in \mathcal{P}(n,m^2)$ satisfying the following two conditions:

(i) $\langle p_i - a_i q_i, u\rangle \neq 0$ for every $i \in [m]$ such that $p_i - a_i q_i \neq 0$;

(ii) $\langle q_i - d_{ij} q_j, u\rangle \neq 0$ for every (i,j) such that $i \neq j$. (We note that $q_i - d_{ij} q_j \neq 0$ because the $\{q_i\}_i$ are pairwise linearly independent.)

Let $b_i = \langle p_i - a_i q_i, u\rangle$ for all $i \in [m]$ and consider the univariate function $g(\lambda)$ defined as $g(\lambda) = f(u+\lambda v)$. Note that for every $\lambda \in \mathbb{C}$ such that $u+\lambda v \notin \mathcal{H}$ we have

$$g(\lambda) = \sum_{i=1}^{m} \frac{\langle p_i, u+\lambda v\rangle}{\langle q_i, u+\lambda v\rangle} = \sum_{i=1}^{m} a_i + \sum_{i=1}^{m} \frac{b_i}{\langle q_i, u\rangle + \lambda\langle q_i, v\rangle}\,.$$

Observe that the function $g(\lambda)$ attains the value NaN for at most m values of λ. Furthermore, since we assumed that $\langle q_i - d_{ij} q_j, u\rangle \neq 0$, the functions $\lambda \mapsto \langle q_i, u\rangle + \lambda\langle q_i, v\rangle$ have distinct zeros. In particular, if $b_i \neq 0$, then $|g(\lambda)|$ approaches $+\infty$ as λ approaches $-\langle q_i, u\rangle/\langle q_i, v\rangle$. Since we assumed that at least one b_i is nonzero, it follows that the function $g(\lambda)$ attains some values not in $\{0, \mathrm{NaN}\}$. Moreover, $g(\lambda)$ has at most m zeroes, because it can be written in the form $g(\lambda) = P(\lambda)/Q(\lambda)$ where P, Q are nonzero polynomials of degree at most m. In particular, at least one of the values $g(1), \ldots, g(2m+1)$ does not belong to $\{0, \mathrm{NaN}\}$, contradicting our assumption. Therefore, we have $p_i = a_i q_i$ for all $i \in [m]$. In particular, for every $x \notin \mathcal{H}$ we have $f(x) = a_1 + \cdots + a_m$. To conclude, we observe that $f(v) = 0$ since $v \in \Lambda(m, n)$. It follows that $f(x) = a_1 + \cdots + a_m = 0$ for all $x \notin \mathcal{H}$. □

Remark 48. *One can derive from the above proof a syntactic characterization of the rational functions in Definition 45 that are identically 0. Namely, assuming that the q_i are pairwise linearly independent, the following condition is necessary and sufficient: there exist constants $a_1, \ldots, a_m \in \mathbb{C}$ such that $a_1 + \ldots + a_m = 0$ and $p_i = a_i q_i$ for all $i = 1, \ldots, m$.*

Let $P(x) \in \mathbb{C}[X_1, \ldots, X_n]$ be a polynomial that factors as a product of linear forms, i.e., $P(x) = \langle q_1, x\rangle\langle q_2, x\rangle \ldots \langle q_d, x\rangle$ for some vectors $q_1, \ldots, q_d$ in $\mathbb{C}^n$. From Proposition 47 we can derive the following characterization of the Lie algebra of P.

Theorem 49. *Let $P(x) \in \mathbb{C}[X_1, \ldots, X_n]$ be a polynomial of degree $d \geq 1$ that factors as a product of linear forms. Then, a matrix $C \in \mathbb{C}^{n\times n}$ belongs to the Lie algebra of P if and only if*

$$\sum_{i,j\in[n]} c_{ij} x_j \frac{\partial P}{\partial x_i}(x) = 0 \tag{7}$$

for every $x \in \Lambda(d, n)$. In particular, a basis of the Lie algebra can be computed deterministically in polynomial time with black box access to P.

Proof. The "only if" direction follows immediately from Definition 44. To prove the opposite implication, let us now assume that (7) holds for every $x \in \Lambda(d, n)$. We need to show that C belongs to the Lie algebra of f. Let

us write $P(x) = \langle q_1, x\rangle\langle q_2, x\rangle \dots \langle q_d, x\rangle$ where the q_i are nonzero vectors, and consider the function

$$f_C(x) = \begin{cases} \dfrac{1}{P(x)} \displaystyle\sum_{i,j\in[n]} c_{ij}x_j \frac{\partial P}{\partial x_i}(x) & \text{if } x \notin \mathcal{H}, \\ \text{NaN} & \text{otherwise.} \end{cases}$$

Here $\mathcal{H}$ denotes the union of the m hyperplanes $\{x \in \mathbb{C}^n \colon \langle q_i, x\rangle = 0\}$ as in Definition 45. Note that we have $f_C(x) \in \{0, \text{NaN}\}$ for every $x \in \Lambda(d, n)$. Furthermore, observe that for every $x \notin \mathcal{H}$ we have

$$\frac{1}{P(x)} \sum_{i,j\in[n]} c_{ij}x_j \frac{\partial P}{\partial x_i}(x) = \sum_{i,j\in[n]} c_{ij}x_j \sum_{k\in[d]} \frac{q_{ki}}{\langle q_k, x\rangle} = \sum_{k\in[d]} \frac{\sum_{i,j\in[n]} c_{ij}q_{ki}x_j}{\langle q_k, x\rangle}\,.$$

Since this rational fraction is of form (6), we can apply Proposition 47 and conclude that $f_C(x)$ is equal to zero for every $x \notin \mathcal{H}$. This implies that $\sum_{i,j\in[n]} c_{ij}x_j \frac{\partial P}{\partial x_i}(x) = 0$ for all $x \notin \mathcal{H}$. By continuity we obtain $\sum_{i,j\in[n]} c_{ij}x_j \frac{\partial P}{\partial x_i}(x) = 0$ for all $x \in \mathbb{C}^n$, which implies that C belongs to the Lie algebra of P.

Let us now turn to the second part of the theorem. It is well known that a black box for $\partial P/\partial x_i$ can be constructed from a black box for P (by interpolating P on a line). By the first part, the determination of the Lie algebra therefore boils down to the resolution of a system of $|\Lambda(d, n)|$ linear equations in n^2 variables. □

Remark 50. *We have stated Proposition 47 and Theorem 49 for the field of complex numbers only because the proof of Proposition 47 uses the absolute value. Nevertheless, it follows from general principles that these two results apply to any field K of characteristic 0. Indeed, K can be embedded in an algebraically closed field $\overline{K}$ which must satisfy the same first order formulas as $\mathbb{C}$.*

For our final derandomization results we will need to perform simultaneous diagonalization in polynomial time over $\mathbb{Q}$.

Proposition 51. *There is a polynomial time deterministic algorithm which takes as input a tuple $(A_1, \dots, A_k)$ of matrices of size n with rational entries, and:*

(i) decides whether $A_1, \dots, A_k$ are simultaneously diagonalizable over $\mathbb{Q}$;

(ii) if they are, constructs an invertible matrix $T \in M_n(\mathbb{Q})$ such that the k matrices $T^{-1}A_iT$ are all diagonal.

This result is not particularly surprising but we could not find it in the literature.

Proof. For $k = 1$ this is quite standard. We can for instance compute the characteristic polynomial of A_1, compute its roots (which should all be rational) and attempt to construct a basis of eigenvectors by solving the corresponding linear systems.

For $k > 1$, we can check that the matrices are simultaneously diagonalizable by checking that each matrix is diagonalizable, and that the A_i pairwise commute (Theorem 1.3.21 in [28]). In this case, in order to construct a transition matrix T which diagonalizes the A_i, we will use Lemma 2 to reduce to the case $k = 1$. Namely, we will construct a finite set S of points $(\alpha_2, \ldots, \alpha_n) \in \mathbb{Q}^{n-1}$, and for each $\alpha \in S$ we will:

1. Diagonalize $A_\alpha = A_1 + \alpha_2 A_2 + \cdots + \alpha_k A_k$ as: $A_\alpha = T_\alpha D_\alpha T_\alpha^{-1}$, where D_α is diagonal and T_α invertible.

2. Check whether $T_\alpha^{-1} A_i T_\alpha$ is diagonal for all $i = 1, \ldots, k$.

When the A_i are simultaneously diagonalizable, Lemma 2.(ii) guarantees that this test will will succeed for at least one $\alpha \in S$ if S is not included in a certain union of $n(n-1)/2$ hyperplanes. In order to avoid these hyperplanes we can proceed as in Section 5 and pick any set S of $1 + n(n-1)(n-2)/2$ points on the moment curve as per Lemma 31. □

An alternative to the above algorithm can possibly be extracted from the proof of Theorem 1.3.21 in [28].

Let $f(x_1, \ldots, x_n)$ be a polynomial that can be written as

$$f(x) = \lambda l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n} \tag{8}$$

where λ is a constant, the l_i are linearly independent linear forms and the exponents α_i are all nonzero. A randomized algorithm which finds such a factorization from black box access to f was proposed in [41, Section 4]. This algorithm appealed to randomization for the computation of the Lie algebra of f and also, following [38], for simultaneous diagonalization. In this paper we have given deterministic algorithms for these two tasks, in Theorem 49 and Proposition 51 respectively. This leads to the following polynomial time deterministic factorization algorithm, which we call the *derandomized Lie-algebraic factorization algorithm* (or `DerandLie` for short):

1. Compute a basis $B_1, \ldots, B_k$ of the Lie algebra of f.

2. Reject if $k \neq n - 1$, i.e., if the Lie algebra is not of dimension $n - 1$.

3. Check that the matrices $B_1, \ldots, B_{n-1}$ commute and are all diagonalizable over $\mathbb{Q}$. If this is not the case, reject. Otherwise, declare the existence of a factorization $f(x) = \lambda l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n}$ where the linear forms l_i are linearly independent and $\alpha_i \geq 1$ (λ, the l_i and α_i will be determined in the last 3 steps of the algorithm).

4. Perform a simultaneous diagonalization of the B_i's, i.e., find an invertible matrix A such that the $n-1$ matrices AB_iA^{-1} are diagonal.

5. At the previous step we have found a matrix A such that $g(x) = f(A^{-1}x)$ has a Lie algebra $\mathfrak{g}_g$ which is an $(n-1)$-dimensional subspace of the space of diagonal matrices. Then we compute the orthogonal of $\mathfrak{g}_g$, i.e., we find a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ such $\mathfrak{g}_g$ is the space of matrices $\mathrm{diag}(d_1, \ldots, d_n)$ satisfying $\sum_{i=1}^n \alpha_i d_i = 0$. We normalize α so that $\sum_{i=1}^n \alpha_i = d$.

6. We must have $g(x) = \lambda.m$ where $\lambda \in \mathbb{Q}^*$ and m is the monomial $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ (in particular, α must be a vector with integral entries). We therefore have $f(x) = \lambda.m(Ax)$ and we output this factorization.

Theorem 52. *Let $f(x_1, \ldots, x_n)$ be a polynomial that can be written as in (8) as a product of powers of linearly independent linear forms, where λ and the coefficients of the linear forms are in $\mathbb{Q}$. From a black box for f, the above* `DerandLie` *algorithm computes this factorization deterministically in polynomial time.*

See [41, Section 4] for a correctness proof. As mentioned above, in order to obtain a deterministic algorithm we appeal to Theorem 49 in Step 1 and to Proposition 51 in Step 4. In order to find the scaling factor λ at step 6, we evaluate f at a point x where $f(x) \neq 0$. From Lemma 31, we can find such a point deterministically by trying at most $1 + n(n-1)$ on the moment curve since we need to avoid the n hyperplanes $l_i(x) = 0$. Note that `DerandLie` may fail if f does not factor as a product of linear forms since this is a prerequisite of Theorem 49. The fact that `DerandLie` fails on some inputs may seem at first sight like a weakness of the algorithm, but this is in fact unavoidable for *any* deterministic polynomial-time black box algorithm (see [41, Section 1.5] for details).

Recall from Section 3.1 that Kayal's algorithm for equivalence to a sum of powers relies on factorization into products of linear forms. If this factorization is performed with the `DerandLie` algorithm, we obtain a deterministic version of Kayal's algorithm. Let us call `LieEquivalence` this deterministic equivalence algorithm. As our final result, we observe that `LieEquivalence` will work correctly on *all* inputs due to the presence of the verification step in Kayal's algorithm:

Theorem 53. *Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ be a homogeneous polynomial of degree d given verbosely as a sum of monomials. The* `LieEquivalence` *algorithm determines whether f is equivalent over $\mathbb{Q}$ to P_d, the "sum of d-th powers" polynomial from (2). If this is the case, it outputs an invertible matrix A with rational entries such that $f(x) = P_d(Ax)$. Moreover, for any fixed d the algorithm runs in polynomial time in the Turing machine model.*

Proof. Since we are interested in equivalence over the field of rational numbers, we will run the 3-step algorithm from Section 3.1 with $K = \mathbb{K} = \mathbb{Q}$. First we establish the correctness of `LieEquivalence`. If the algorithm accepts its input f, it explicitly finds at step 2 and step 3 linearly independent linear forms $\ell_i \in \mathbb{Q}[x_1, \ldots, x_n]$ such that $f = \sum_{i=1}^{n} \ell_i^d$. The algorithm's answer must therefore be correct in this case. Conversely, assume that such a decomposition exists. Then the Hessian determinant H_f factors as $H_f = c \prod_{i=1}^{n} \ell_i^{d-2}$ where c is a nonzero constant. Since the ℓ_i are linearly independent, we are in the situation where `DerandLie` works correctly. Therefore, by Theorem 52 we will find the ℓ_i (or actually constant multiples l_i of the ℓ_i) at step 1 of the algorithm of Section 3.1. Finally, the decomposition $f = \sum_{i=1}^{n} \ell_i^d$ is obtained at steps 2 and 3.

We now turn to the algorithm's complexity. Since `DerandLie` runs in polynomial time, the first step of the algorithm from Section 3.1 will also run in polynomial time. At step 2 we can afford to expand the powers l_i^d as sums of monomials (this takes polynomial time for constant d), and then we find the constants a_i by dense linear algebra. Finally, the extraction of d-th roots of rational numbers at step 3 also takes polynomial time. □

Acknowledgements

We would like to thank Roger Horn for pointing out Problem 4.5.P4 in [28] and Peter Bürgisser for references to the tensor literature. The anonymous referee suggested several improvements in the presentation of the paper, and a significant simplification in the hitting set constructions of Section 6.

References

[1] Manindra Agrawal, Chandan Saha, and Nitin Saxena. Quasi-polynomial hitting-set for set-depth-Δ formulas. In *Proc. 45th annual ACM symposium on Theory of Computing (STOC 2013)*, pages 321–330, 2013.

[2] Manindra Agrawal and Nitin Saxena. Equivalence of F-algebras and cubic forms. In *Proc. Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 115–126. Springer, 2006.

[3] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (second edition)*. Springer, 2006.

[4] Carlos Beltrán, Paul Breiding, and Nick Vannieuwenhoven. Pencil-based algorithms for tensor rank decomposition are not stable. *SIAM Journal on Matrix Analysis and Applications*, 40(2):739–773, 2019.

[5] Alessandra Bernardi, Alessandro Gimigliano, and Monica Ida. Computing symmetric rank for symmetric tensors. *Journal of Symbolic Computation*, 46(1):34–53, 2011.

[6] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[7] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.

[8] Jérôme Brachat, Pierre Comon, Bernard Mourrain, and Elias Tsigaridas. Symmetric tensor decomposition. *Linear Algebra and its Applications*, 433(11-12):1851–1872, 2010.

[9] Enrico Carlini. Reducing the number of variables of a polynomial. In *Algebraic geometry and geometric modeling*, Math. Vis., pages 237–247. Springer, Berlin, 2006.

[10] Guillaume Cheze and André Galligo. Four lectures on polynomial absolute factorization. In *Solving polynomial equations*, pages 339–392. Springer, 2005.

[11] Guillaume Chèze and Grégoire Lecerf. Lifting and recombination techniques for absolute factorization. *Journal of Complexity*, 23(3):380–420, 2007.

[12] Lieven De Lathauwer. A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization. *SIAM journal on Matrix Analysis and Applications*, 28(3):642–666, 2006.

[13] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. Computation of the canonical decomposition by means of a simultaneous generalized Schur decomposition. *SIAM journal on Matrix Analysis and Applications*, 26(2):295–327, 2004.

[14] Michael Forbes and Amir Shpilka. Explicit Noether normalization for simultaneous conjugation via polynomial identity testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (RANDOM 2013)*, pages 527–542. Springer, 2013.

[15] Michael Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *Proc. 54th Annual Symposium on Foundations of Computer Science (FOCS 2013)*, pages 243–252. IEEE, 2013.

[16] Michael A Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 867–875, 2014.

[17] Shuhong Gao. Factoring multivariate polynomials via partial differential equations. *Mathematics of computation*, 72(242):801–822, 2003.

[18] Ignacio García-Marco, Pascal Koiran, and Timothée Pecatte. Reconstruction algorithms for sums of affine powers. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 317–324, 2017.

[19] Ignacio García-Marco, Pascal Koiran, and Timothée Pecatte. Polynomial equivalence problems for sums of affine powers. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC)*, 2018.

[20] Ankit Garg, Nikhil Gupta, Neeraj Kayal, and Chandan Saha. Determinant equivalence test over finite fields and over $\mathbb{Q}$. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 26, page 42, 2019.

[21] Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. In *Proc. 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 109–117, 2016.

[22] Rodrigo Gondim and Francesco Russo. On cubic hypersurfaces with vanishing Hessian. *Journal of Pure and Applied Algebra*, 219(4):779–806, 2015.

[23] Joshua Grochow. Matrix isomorphism of matrix Lie algebras. In *27th IEEE Conference on Computational Complexity(CCC)*, pages 203–213, 2012.

[24] Joshua Grochow and Youming Qiao. Isomorphism problems for tensors, groups, and cubic forms: completeness and reductions. *arXiv:1907.00309*, 2019.

[25] Ankit Gupta, Neeraj Kayal, and Youming Qiao. Random arithmetic formulas can be reconstructed efficiently. *Computational Complexity*, 23(2):207–303, 2014.

[26] DK Harrison. A Grothendieck ring of higher degree forms. *Journal of Algebra*, 35(1-3):123–138, 1975.

[27] Richard Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA working papers in phonetics*, 1970.

[28] Roger Horn and Charles Johnson. *Matrix Analysis*. Cambridge University Press (second edition), 2013.

[29] Hang Huang, Mateusz Michałek, and Emanuele Ventura. Vanishing Hessian, wild forms and their border VSP. *arXiv preprint arXiv:1912.13174*, 2019.

[30] Gábor Ivanyos and Youming Qiao. Algorithms based on*-algebras, and their applications to isomorphism of polynomials with one secret, group isomorphism, and polynomial identity testing. *SIAM Journal on Computing*, 48(3):926–963, 2019.

[31] Gábor Ivanyos, Youming Qiao, and K Venkata Subrahmanyam. Constructive non-commutative rank computation is in deterministic polynomial time. In *Innovations in Theoretical Computer Science (ITCS)*, 2017.

[32] Gábor Ivanyos, Youming Qiao, and KV Subrahmanyam. Non-commutative Edmonds' problem and matrix semi-invariants. *Computational Complexity*, 26(3):717–763, 2017.

[33] Rujun Jiang and Duan Li. Simultaneous diagonalization of matrices and its applications in quadratically constrained quadratic programming. *SIAM Journal on Optimization*, 26(3):1649–1668, 2016.

[34] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

[35] Erich Kaltofen. Factorization of polynomials given by straight-line programs. In *Randomness and Computation*, pages 375–412. JAI Press, 1989.

[36] Zohar Karnin and Amir Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 274–285, 2009.

[37] Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, January 2011.

[38] Neeraj Kayal. Affine projections of polynomials. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 643–662, 2012.

[39] Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of full rank algebraic branching programs. *ACM Transactions on Computation Theory (TOCT)*, 11(1):2, 2018.

[40] Neeraj Kayal and Chandan Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In *Proc. 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–424, 2019.

[41] P. Koiran and N. Ressayre. Orbits of monomials and factorization into products of linear forms. *arXiv:1807.03663*, 2018.

[42] Pascal Koiran. Orthogonal tensor decomposition and orbit closures from a linear algebraic perspective. *Linear and Multilinear Algebra*, 2019. arXiv:1905.05094.

[43] Pascal Koiran and Subhayan Saha. Black box absolute reconstruction for sums of powers of linear forms. In preparation, 2021.

[44] Ankur Moitra. *Algorithmic aspects of machine learning*. Cambridge University Press, 2018.

[45] Onorato Timothy O'Meara. *Introduction to quadratic forms*. Springer, 1973.

[46] Qazi Ibadur Rahman and Gerhard Schmeisser. *Analytic theory of polynomials*. Number 26 in London Mathematical Society monographs (new series). Oxford University Press, 2002.

[47] N. Saxena. Diagonal circuit identity testing and lower bounds. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, LNCS 5125, pages 60–71. Springer, 2008.

[48] Nitin Saxena. *Morphisms of rings and application to complexity*. PhD thesis, IIT Kanpur, 2006.

[49] Winfried Scharlau. *Quadratic and Hermitian forms*. Springer, 1985.

[50] Hani Shaker. Topology and factorization of polynomials. *Mathematica Scandinavica*, pages 51–59, 2009.

[51] Amir Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. *SIAM Journal on Computing*, 38(6):2130–2161, 2009.

[52] A-J van der Veen and Arogyaswami Paulraj. An analytical constant modulus algorithm. *IEEE Transactions on Signal Processing*, 44(5):1136–1155, 1996.

[53] Emanuele Ventura. On a question of Koiran and Skomra. Personal communication, 2021.