Algebraic and Numerical Algorithms for Symmetric Tensor Decompositions

Subhayan Saha

▶ **To cite this version:**

Subhayan Saha. Algebraic and Numerical Algorithms for Symmetric Tensor Decompositions. Computational Complexity [cs.CC]. Ecole normale supérieure de lyon - ENS LYON, 2023. English. NNT : 2023ENSL0093 . tel-04379539

HAL Id: tel-04379539
https://theses.hal.science/tel-04379539

Submitted on 8 Jan 2024

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.

THÈSE
en vue de l'obtention du grade de Docteur, délivré par
l'ÉCOLE NORMALE SUPERIEURE DE LYON

École Doctorale N°512
InfoMaths

Discipline : Informatique

Soutenue publiquement le 08/12/2023, par :

Subhayan Saha

Algebraic and Numerical Algorithms for Symmetric Tensor Decompositions

Algorithmes algébriques et numériques pour la décomposition de tenseurs symétriques

---

Devant le jury composé de :

Beltrán Carlos, Professor, Universidad de Cantabria (Espagne)	Rapporteur
Srinivasan Srikanth, Associate Professor, Aarhus University (Danemark)	Rapporteur
Hubert Evelyne, Directrice de Recherche, Centre Inria d'Université Côte d'Azur	Examinatrice
Tavenas Sébastien, Chargé de Recherche CNRS, LAMA	Examinateur
Koiran Pascal, Professeur, ENS de Lyon	Directeur de thèse

Abstract

A symmetric tensor is a multi-dimensional array with entries that are invariant under all permutations of its indices and it is *equivalent* to a homogeneous polynomial with degree equal to the order of the tensor. In this thesis, we study in the decomposition of an order-d symmetric tensor T over $\mathbb{C}$ as a sum of symmetric rank-one tensors, that is, decompositions of the form $T = \sum_{i=1}^{r} u_i^{\otimes d}$ where $u_i \in \mathbb{C}^n$. In order to obtain efficient algorithms, it is necessary to add certain restrictions to these tensors. In most of our algorithms throughout this thesis, we treat the case where the u_i's are linearly independent and such a decomposition is *essentially unique*. This forces the number of summands, r to be at most n and if $r = n$, then the tensor is called *diagonalisable*. Given a tensor T, we are interested in the following two algorithmic questions: 1) is it diagonalisable? and 2) if it is diagonalisable, output a decomposition. We give an answer to the first question in the algebraic model of computation. More specifically, given oracle access to a blackbox for the degree-d homogeneous polynomial equivalent to an order-d tensor $T \in \mathbb{C}^n \otimes \ ... \ \otimes \mathbb{C}^n$, we can verify in polynomial (in n, d) time in the Blum-Shub-Smale model of computation, whether the tensor is diagonalisable or not. We also extend this to the case where the number of summands is strictly less than n. We also give a *numerically-stable* algorithm that solves the second question *approximately*. More formally, given an order-3 symmetric tensor T that is diagonalisable and a desired accuracy parameter ε, we give an algorithm that outputs a decomposition which is ε-close (in the l_2 norm) to the actual decomposition. It runs in linear time and requires polylogarithmic bits of precision when run on a finite precision machine.

Résumé en français

Un tenseur symétrique est un tableau multidimensionnel dont les entrées sont invariantes sous toutes les permutations de ses indices et il est *équivalent* à un polynôme homogène dont le degré est égal à l'ordre du tenseur. Dans cette thèse, nous étudions la décomposition d'un tenseur symétrique d'ordre d, noté T, sur $\mathbb{C}$ en tant que somme de tenseurs symétriques de rang un, c'est-à-dire des décompositions de la forme $T = \sum_{i=1}^{r} u_i^{\otimes d}$ où $u_i \in \mathbb{C}^n$. Afin d'obtenir des algorithmes efficaces, il est nécessaire d'ajouter certaines restrictions à ces tenseurs. Dans la plupart de nos algorithmes, nous traitons le cas où les u_i sont linéairement indépendants et une telle décomposition est *essentiellement unique*. Cela implique que le nombre de termes de la somme, r, soit au plus n, et si $r = n$, le tenseur est appelé *diagonalisable*. Étant donné un tenseur T, nous nous intéressons aux deux questions algorithmiques suivantes : 1) est-il diagonalisable ? et 2) s'il est diagonalisable, produire une décomposition. Nous répondons à la première question dans le cadre du modèle de calcul algébrique. Plus précisément, en ayant accès à un oracle pour une boîte noire représentant le polynôme homogène de degré d équivalent à un tenseur d'ordre d, $T \in \mathbb{C}^n \otimes \ldots \otimes \mathbb{C}^n$, nous pouvons vérifier en temps polynomial (en n et d) dans le modèle de calcul de Blum-Shub-Smale si le tenseur est diagonalisable ou non. Nous étendons également cela au cas où le nombre de termes de la somme est strictement inférieur à n. Nous donnons également un algorithme *numériquement stable* qui résout approximativement la deuxième question. Plus formellement, étant donné un tenseur symétrique d'ordre 3, T, qui est diagonalisable, et un paramètre de précision souhaité ε, nous donnons un algorithme qui produit une décomposition qui est ε-proche (au sens de la norme l_2) de la décomposition réelle. Il s'exécute en temps linéaire et nécessite un nombre de bits polylogarithmique de précision lorsqu'il est exécuté sur une machine à précision finie.

Contents

Chapter 1

Introduction

Tensors are important algebraic objects that appear in different branches of science such as mathematics, physics, computer science and chemistry. For most of our purposes, tensors can be viewed as a multi-dimensional array with entries from the underlying field $\mathbb{K}$. Following this, order-1 tensors are vectors and order-2 tensors are essentially matrices.

One fundamental question that people have been interested in is the decomposition of tensors into its *rank-one* components. In this thesis, we look at the following two algorithmic problems related to these decompostions:

- **Decision problem:** Given a tensor T and some fixed number r, does there exist a decomposition of the tensor T as a sum of r rank-one components?
- **Search problem:** If it is given that such a decomposition exists, find such a decomposition.

Tensor decompositions have generated significant interest in recent years due to their applications in different fields such as signal processing, computer vision, chemometrics, neuroscience and others (see [KB09] for a comprehensive survey on the applications and available software for this problem). In fact, a number of learning algorithms for certain models have been developed through the fundamental machinery of tensor decompositions. Pure topic models ([AHK12]), blind source separation and independent component analysis ([DLCC07]), Hidden Markov Models ([MR05],[HKZ09]), mixture of spherical gaussians ([HK13],[GHK15]), Latent Dirichlet Allocation ([AFH$^+$12]). Numerous algorithms have been devised for solving the tensor decomposition problem with different assumptions on the input tensor and different efficiency and accuracy bounds [Har70, LRA93, BCMT09, BGI11, GVX14, BCMV14, AGH$^+$14, GM15, HSS15, MSS16, KP20, BHKX22, DdL$^+$22].

In this thesis, we are interested in symmetric tensors that satisfy certain *genericity* conditions (we refer to these as *diagonalisable tensors*) and we will see in Section 4.3.1 that such a decomposition is *essentially unique*. We consider the decision problem from an algebraic complexity point of view - if an arbitrary symmetric tensor is given *succinctly*, we give an *algebraic* algorithm that runs in polynomial time to verify if it has a decomposition that satisfies the *genericity* conditions. Polynomials are the main objects of study in algebraic complexity theory - in Section 1.1.3 we explore their equivalence with tensors. We define the algebraic model of computation in Section 1.2.1 and discuss the relation of the *decision problem* with reconstruction algorithms and the problem of polynomial equivalence testing in algebraic complexity theory in Sections 1.5 and 1.5.2. We also consider the search problem but from the point of view of numerical algorithms (we discuss this in more detail in Section 1.3). We fix the standard finite-precision arithmetic to be the underlying model of computation. This model is explained in more detail in Section 1.2.2. In this setting, if the order-3

symmetric tensor given explicitly as input has a decomposition that satisfies some *genericity* conditions, then we give an algorithm that solves the search problem *approximately* - it outputs a solution which is *close* to the actual decomposition of the tensor.

1.1 Tensors and polynomials

1.1.1 Tensors

From a mathematical point of view, tensors describe a (multi)-linear relationship between products of vector spaces (and their duals). In linear algebra, a multilinear map is a function of several variables that is linear separately in each variable. Let $V_1, \cdots, V_d$ be vector spaces over the field $\mathbb{K}$. Formally, an order d tensor is an element of the set of all multilinear maps $\{\phi : V_1^* \times \cdots \times V_d^* \to \mathbb{K}\}$ where V_i^* denotes the dual of the vector space V_i. When each of these V_i's have a finite basis, then the tensors can be written as a multi-dimensional array with entries from the underlying field $\mathbb{K}$. Following this definition, order-1 tensors are vectors and order-2 tensors are essentially bilinear maps.

For the rest of this thesis, we will fix the underlying vector spaces to be either $\mathbb{R}^n$ or $\mathbb{C}^n$ for some n and then we can consider tensors to be multi-dimensional arrays. If $T \in \mathbb{C}^{I_1} \otimes \cdots \mathbb{C}^{I_d}$, then for a particular $k \in [d]$, the k-th mode is $\mathbb{C}^{I_k}$. The order of a tensor is then the number of dimensions (also referred to as *modes*). Let $a^{(1)}, \cdots, a^{(k)}$ be vectors such that $a^{(i)} \in \mathbb{C}^{I_k}$. Then their tensor product (also, sometimes referred to as the outer product) denoted by $T = a^{(1)} \otimes \cdots \otimes a^{(k)}$ is an element in the vector space $\mathbb{C}^{I_1} \otimes \cdots \mathbb{C}^{I_k}$. When written as a multi-dimensional array, for all $i_t \in [I_t]$ for $t \in [k]$,

$$T_{i_1,\cdots,i_k} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(k)}_{i_k}.$$

An order-d tensor $T \in \mathbb{C}^{I_1} \otimes \cdots \mathbb{C}^{I_d}$ is defined to be a *rank-one tensor* if there exist vectors $a^{(1)}, \cdots, a^{(d)} \neq 0$ where $a^{(k)} \in \mathbb{C}^{I_k}$ for all $k \in [d]$ such that

$$T = a^{(1)} \otimes \cdots \otimes a^{(d)}.$$

A tensor is called *cubical* if $T \in \mathbb{C}^{I \times \cdots I}$ for a particular I (that is, every mode has the same size.) A cubical tensor is called *symmetric* if its elements remain constant under any permutation of indices. For example, $T \in \mathbb{C}^I \otimes \mathbb{C}^I \otimes \mathbb{C}^I$ is symmetric if for all $i, j, k \in I$

$$T_{ijk} = T_{ikj} = T_{jik} = T_{jki} = T_{kij} = T_{kji}.$$

1.1.2 Polynomials

A degree d polynomial in $P \in \mathbb{F}[x_1, \cdots, x_n]$ is called *homogeneous* if all monomials have same total degree d, that is for $m = x_1^{e_1} \cdots x_n^{e_n}$, $e_1 + \cdots + e_n = d$. A polynomial $P \in \mathbb{F}[x_1, \cdots, x_n]$ is said to be *multilinear* if for all monomials $m = x_1^{e_1} ... x_n^{e_n}$ such that $\text{coeff}_m(f) \neq 0$, $e_i \leq 1$. A homogeneous polynomial $P \in \mathbb{F}[x_1, \cdots, x_n]$ of degree d is said to be *set-multilinear* if there exists a partition of the set of variables $\{x_1, \cdots, x_d\}$ into sets $S_1, \cdots, S_d$ such that every monomial with a non-zero coefficient in P contains exactly one variable from each S_i. More formally, every monomial in P has the form $\prod_{i=1}^d x^{(i)}$ where $x^{(i)} \in S_i$.

1.1.3 Tensor polynomial equivalence:

We can associate to a symmetric tensor T of order d the homogeneous polynomial

$$f(x_1, ..., x_n) = \sum_{i_1,\cdots,i_d=1}^{n} T_{i_1,\cdots,i_d} x_{i_1} x_{i_2} \cdots x_{i_d}.$$

This correspondence is bijective, and the symmetric tensor associated to a homogeneous polynomial f can be obtained from the following relation: for $i_1, \cdots, i_d \in [n]$, the $(i_1, \cdots, i_d)$-th entry of the tensor can be extracted from the partial derivative of the polynomial with respect to the monomial $x_{i_1}...x_{i_d}$.

To write the expression more formally, we will set up some notation. For a monomial $m = x_1^{e_1}...x_n^{e_n}$, we denote by $\bar{e} = (e_1, \cdots, e_n)$ the tuple of indices of the respective variables. Then we define $\bar{e}! = \frac{(\sum_{i=1}^{n} e_i)!}{e_1!...e_n!}$.

Let us assume we want to extract the $(i_1, \cdots, i_d)$-th entry of the tensor $T_{i_1,\cdots,i_d}$. Using this notation, we can write the corresponding monomial $x_{i_1}...x_{i_d}$ as $m = x_1^{e_1}...x_n^{e_n}$ for some appropriate $e_1, \cdots, e_n \geq 0$. Then we get that the corresponding entry of the tensor is given by the following relation

$$\frac{1}{\bar{e}!} \frac{\partial^d f}{\partial x_{i_1}...x_{i_d}} = T_{i_1,\cdots,i_d} \tag{1.1}$$

When the condition of symmetry is dropped from the tensor (often referred to as *ordinary tensors*), one can show similarly that an ordinary degree-d tensor is equivalent to a homogeneous set-multilinear polynomial (defined in Section 1.1.2).

1.2 Models of computation

In computer science, the computational complexity or simply complexity of an algorithm is the amount of resources (in terms of *time* taken and *memory* requirements) required to run the algorithm. Alan Turing in his seminal paper [Tur36] introduced the notion of Turing Machines which is a simple mathematical model that suffices for studying many questions about computational tasks and efficiently solving them.

More concretely, let f be a function that takes in a string of bits (that is, the input is in the set $\{0,1\}^*$) and outputs 0 or 1. An *algorithm* for computing f is a fixed set of instructions that on any input $x \in \{0,1\}^*$ computes $f(x)$. The Turing machine is a formal definition of the mechanical rules that any *algorithm* must consist of, that is, it can use only these fixed rules arbitrarily many number of times and nothing else. These rules are *elementary* like reading a bit of the input, writing something or reusing some already written symbols from the given working space or just stopping and giving as output the desired value. The running time of an algorithm is defined to be the number of times these elementary rules are used by the algorithm. (For a more formal definition of Turing machines, refer to [Sip13]). Throughout the rest of this thesis, this is what we will refer to as the *bit model of computation*.

1.2.1 Algebraic Models of Computations

An arithmetic circuit is a very natural and succinct way of representing polynomials. In fact, it captures exactly the number of arithmetic operations required to evaluate the polynomial on any input.

Definition 1.2.1. *An arithmetic circuit C over the field $\mathbb{F}$ with parameters $\mathcal{P} = \{\alpha_1, \cdots, \alpha_p\} \subseteq \mathbb{F}$ and set of variables $X = \{x_1, \cdots, x_n\}$ is a finite directed acyclic graph where each vertex (gate) is one of the following:*

- *A vertex with in-degree 0 labelled by some variable x_i or some element in $\mathcal{P}$. If the label is a variable, the vertex is called an input gate.*
- *A vertex with in-degree 2 labelled by either $+$ or $\times$.*
- *An output gate with out-degree zero; we assume there is exactly one output gate.*

C computes the polynomial in a natural way: every input gate computes a polynomial in $\mathbb{F} \cup X$. A vertex with label $\times$ (called the product gate) computes the product of the polynomials computed by its children and a vertex with label $+$ (called the sum gate) computes the sum of the polynomials computed by its children. The output gate outputs the polynomial computed by the circuit.

The size of a circuit is the number of vertices in the graph. For a fixed polynomial p, size(p) denotes the minimum size of a circuit computing the polynomial. There are other interesting models of computation as well such as arithmetic formulas (underlying acyclic graph in Definition 1.2.1 is a tree) and arithmetic branching programs (refer to [SY10] for a good exposition of the different arithmetic models of computation.)

The Blum-Shub-Smale Model:

The Blum-Shub-Smale model is a uniform model of computation due to [BSS89, BCSS98]. It constitutes of a generalization of Turing Machines that perform computation over some arbitrary ring $\mathcal{R}$. When $\mathcal{R} = \mathbb{F}_2$, then the BSS model is equivalent to the standard Turing machine model. Note that the definitions in this section are mostly taken from [Koi00].

An *algebraic circuit* over $\mathbb{C}$ is an arithmetic circuit (following Definition (1.2.1)) where in addition to the $\times$ and $+$ gates an equality test gate $=$ is allowed. A test gate takes in two elements $\alpha, \beta \in \mathbb{C}$ and outputs 1 if $\alpha = \beta$ and 0 otherwise. Size and depth of algebraic circuits are defined analogously. An *algebraic circuit* over $\mathbb{R}$ can be defined similarly by replacing the equality test gate $=$ by the order $\geq$ test gate. It takes in two elements $\alpha, \beta \in \mathbb{R}$ and outputs 1 if $\alpha \geq \beta$ and 0 otherwise. The following complexity classes can then be defined based on algebraic circuits

Complexity classes over $\mathbb{R}$ and $\mathbb{C}$: A problem is essentially a subset of $\mathbb{C}^\infty = \bigcup_{n\geq 1} \mathbb{C}^n$. We denote by $\mathbb{P}_\mathbb{C}$ the class of all non-uniform polynomial time problems over $\mathbb{C}$. More formally, a problem $X \subseteq \mathbb{C}^\infty$ is in $\mathbb{P}_\mathbb{C}$ if there exists a polynomial $p(n)$ and a family of algebraic circuits $(C_n)_{n\geq 1}$ with parameters $\alpha_1, \cdots, \alpha_p \in \mathbb{C}$ where C_n has $n+p$ inputs, size$(C_n) \leq p(n)$ and the following conditions are satisfied:

$$\forall x \in \mathbb{C}^n, x \in X \iff C_n(\alpha_1, \cdots, \alpha_p, x_1, \cdots, x_n) = 1. \tag{1.2}$$

A problem X is in the class $\mathrm{P}_\mathbb{C}$ of polynomial-time problems if $X \in \mathbb{P}_\mathbb{C}$ and the corresponding circuit family (C_n) in (1.2) is *uniform*. More formally, there exists a polynomial time Turing machine in the classical sense which on input n (in unary) constructs C_n. One can similarly define complexity classes $\mathrm{P}_\mathbb{R}$ and $\mathbb{P}_\mathbb{R}$ using algebraic circuits over $\mathbb{R}$.

Note: This complexity class can also be defined using a generalization of Turing Machines over $\mathbb{C}$ (with the added advantage that basic arithmetic operations (addition, subtraction, multiplication, division, and comparison) take a unit time step to

perform) [Poi95]. More generally, analogous complexity classes can be defined for any arbitrary *structures*. A structure M is a set equipped with a finite set of functions $f_i \in M_{n_i} \to M$ and relations $r_i \subseteq M_{m_i}$.

One can also similarly define corresponding non-deterministic classes $\mathrm{N}\mathbb{P}_{\mathbb{C}}$ and $\mathrm{NP}_{\mathbb{C}}$ (Refer to [Koi00] for more formal definitions).

Black-box model

For a lot of algebraic problems, it is necessary to give a polynomial as input. and one can study the different representations of these input polynomials. One simple way to give the polynomial as input is to give a list of the coefficients of the different monomials of the polynomial. But polynomials in n variables and degree d can have $\binom{n+d}{d}$ monomials (which is exponential in n, d).

Recall from Definition 1.2.1 that every polynomial can be represented by an arithmetic circuit. And this representation is succinct in the following sense i.e. there exist polynomials that have exponential in n, d monomials but have a circuit of size polynomial in n, d. For example, the polynomial

$$p(x_1, \cdots, x_n) = \prod_{i=1}^{n} (1 + x_i) \tag{1.3}$$

has 2^n monomials but it has an arithmetic circuit of size $2n$. So, often circuits computing a specific polynomial is given as input and this is referred to as the *white-box* model of computation.

Another way of giving the polynomial as input is by *black-box* access, which is defined as giving oracle access to the polynomial by evaluation at a certain point. This can be stated more formally in the following way: Let f be a polynomial in $\mathbb{C}[x_1, \cdots, x_n]$. For any query $(\alpha_1, \cdots, \alpha_n) \in \mathbb{C}^n$, the oracle returns $f(\alpha_1, \cdots, \alpha_n)$ in unit time.

The choice of models for representing the input polynomials can often affect the hardness of the problem. We will illustrate this using the *polynomial identity testing* (PIT) problem which is a fundamental problem in algebraic complexity theory. The algorithmic problem is the following: Given a multivariate polynomial $P \in \mathbb{C}[x_1, \cdots, x_n]$, determine whether $P \equiv 0$. If the input polynomial is given as a list of coefficients, then PIT becomes trivially linear (in the input size) because one just needs to check if the list has a non-zero element or not. For both the white-box and the black-box model, the problem becomes significantly harder - no deterministic subexponential time algorithms are known in the literature and solving this problem would have major consequences in complexity theory (refer to [SY10] for a detailed exposition).

Using the fact that the BSS model can be defined equivalently using generalizations of Turing machine model (refer to the note at the end of Section 1.2.1), one can also consider looking at algorithms based on this model (refer to the beginning of Section 1.2 for a discussion of algorithms and Turing machines). As discussed, they can perform basic arithmetic operations (addition, subtraction, multiplication, division, and comparison) as a unit time step irrespective of their size. In Chapter 2, we will fix the underlying model of computation to be the BSS model with oracle access to a black-box computing the polynomial and give certain algebraic algorithms based on this model.

An algorithm is said to run in *strongly polynomial* time in the BSS model of computation if the following properties are satisfied:

- The number of arithmetic operations is bounded by a polynomial in the size of the input instance
- The bit-size of the numbers on which these arithmetic operations are performed is also polynomial in the size of the input instance.

Any algorithm with these two properties can be converted to a polynomial time algorithm (by replacing the arithmetic operations by suitable algorithms for performing the arithmetic operations on a Turing machine) in the bit size model of computation (refer to the beginning of Section 1.2 for a discussion.) The following is an example of a situation where this is not satisfied: Given the integer 2^n as input (requires $O(n)$ bits of input size in the bit model), one only requires n multiplications to compute 2^{2^n} (via repeated squaring). But the size of the number of bits in this computation is $\log(2^{2^n}) = 2^n$ which is exponential in the input size and hence, this algorithm is not strongly polynomial time.

In the algorithms presented in Chapter 2, if the polynomial given as input has coefficients in $\mathbb{Q}$, we show that our algorithms indeed run in *strongly polynomial* time and by the previous discussion, we get polynomial time running bounds for our algorithms even in the standard Turing machine model of computation. This will be explained in more detail in Section 2.7.2 in Chapter 2.

1.2.2 Finite precision arithmetic

Another widely studied model of computation is the *finite precision arithmetic* model. In this model, real numbers are rounded to a fixed number of bits which may depend on the input size and accuracy. We use like [BGVKS22] the standard floating point axioms from [Hig02]. This model can yield actual Boolean complexity bounds and also helps to analyse the stability of the algorithm which we will see in more detail in Chapter 3.

We now elaborate on this model for completeness of the exposition. It is assumed that numbers are stored and manipulated up to some machine precision u which is a function of n, the size of the input and δ which is the desired accuracy parameter. This means that every number $x \in \mathbb{C}$ is stored as $\mathrm{fl}(x) = (1+\Delta)x$ for some adversarially chosen $\Delta \in \mathbb{C}$, satisfying $|\Delta| \leq u$ and each arithmetic operation $* \in \{+, -, \times, \div\}$ is guaranteed to yield an output satisfying

$$\mathrm{fl}(x * y) = (x * y)(1+\Delta) \text{ where } |\Delta| \leq u \tag{1.4}$$

It is also standard and convenient to assume that we can evaluate $\sqrt{x}$ and $x^{\frac{1}{3}}$ for any $x \in \mathbb{C}$, where again $\mathrm{fl}(\sqrt{x}) = \sqrt{x}(1+\Delta)$ and $\mathrm{fl}(x^{\frac{1}{3}}) = y(1+\Delta)$ for $|\Delta| \leq u$ where y is a cube root of x.

Thus, the outcomes of all operations are adversarially noisy due to roundoff. The bit lengths of numbers stored in this form remain fixed at $\log(\frac{1}{u})$. An iterative algorithm that can be implemented in finite precision (typically, polylogarithmic in the input size and desired accuracy) is called numerically stable. Note that in this model it is not even assumed that the input is stored exactly.

1.3 Numerical Algorithms and Condition Numbers

Recall that we had described the model of finite precision arithmetic in Section 1.2.2. An algorithm in this model cannot compute the desired function exactly, due to the error that can creep in the computation at every step. Algorithms that are

mathematically equivalent (that is, they are designed to compute the same function) can perform very differently in the finite precision model. In order to quantifiably characterize these "differences in performance", the notion of *numerical stability* of an algorithm was introduced. Without an attempt at being exhaustive, the following are the different notions of stability:

- **Forward stability:** On some input, the algorithm outputs a solution that is close to the exact solution on that input.
- **Backward stable:** We say $\tilde{f}$ is a *backward stable* algorithm for computing a function f, if on input x, it outputs $f(\tilde{x})$ where $\tilde{x}$ is some point close to x.

In this section we also provide more background on condition numbers in numerical computation. A book-length treatment of this subject can be found in [BC13]. There is no universally accepted definition of a "condition number" in numerical analysis, but a common one, is as follows. Suppose we wish to compute a map $f : X \to Y$. The condition number of f at an input x is a measure of the variation of the image $f(x)$ when x is perturbed by a small amount. This requires the choice of appropriate distances on the spaces X and Y. The condition number is therefore a quantitative measure of the continuity of f at x. In particular, it is independent of the choice of an algorithm for computing f. In finite arithmetic, we cannot hope to approximate $f(x)$ with a low precision algorithm at an input x with a high condition number since we do not even assume that the input is stored exactly. Moreover, designing algorithms that work in low precision at well-conditioned inputs is often a challenging task.

This can be elucidated nicely with an example from [BNV23]. Let $A \in \mathbb{R}^{m\times n}$ where $m \geq n$ be a left-invertible matrix and $b \in \mathbb{R}^m$. We consider the problem of finding $x \in \mathbb{R}^n$ such that $Ax = b$ (also known as the *over-determined least squares problem*). One algorithm for this is to transform the system into a system of normal linear equations which can be solved by Gaussian elimination with full pivoting. But the algorithm is said to be somewhat numerically unstable since the condition number of the Gram matrix A^TA (which governs the stability of this algorithm) in the intermediate step is $(\kappa(A))^2$. Here $\kappa(A)$ is the condition number of A, defined as $\kappa(A) = ||A||||A^\dagger||$ where $A^\dagger$ is the Moore-Penrose inverse of A. A more stable algorithm is obtained by computing a reduced singular value decomposition of $A = USV^T$, solving the diagonal system $Sy = U^Tb$ and computing $x = Vy$. [Bjö14]

Sometimes, the above continuity-based definition of condition numbers is not suitable. This is for instance the case for decision problems, where the map f is boolean-valued. A popular alternative is to use the inverse of the (normalized) distance to the set of ill-posed instances [BC13, chapter 6]. One can sometimes show that these two notions coincide [BC13, Section 1.3].

Another example that we would be looking at throughout this thesis is the problem of designing numerical algorithms for matrix diagonalisation (which we will explore in more detail in Section 3.3). Suppose for instance that we want to approximate the eigenvectors of a matrix. In order to estimate the condition number in the above sense, we need to understand how the eigenvectors evolve under a perturbation of the input matrix. This is a relatively standard task in perturbation theory, see for instance Appendix A of [BCMV14][1] or the proof of Proposition 1.1 in [BGVKS22]. However, until the recent breakthrough [BGVKS22] we did not have any efficient, low-precision algorithm for this task (see Theorem 3.3.4 in Section 3.3 for a precise statement of their result).

[1]This property is at the heart of their analysis of the robustness of Jennrich's algorithm.

One can ask the following question: Is the composition of *backward/forward* stable algorithms also a *backward/forward* stable algorithm for computing the composition of the respective functions? The following is an example borrowed from [BNV23] showing that it is not always true.

Example 1.3.1. *Define the functions* $g : \mathbb{R} \to \mathbb{R}$ *as* $g(x) = \frac{x}{3}$ *and* $h : \mathbb{R} \to [1, \infty)$ *as* $h(x) = x^2 + 1$. *Let* $\tilde{g}$ *and* $\tilde{h}$ *be the respective backward stable algorithms for computing each of these functions in finite precision arithmetic.*

Define the function $f : \mathbb{R} \to [\frac{1}{3}, \infty)$ *as* $f(x) = \frac{1}{3}(x^2 + 1)$. *Then it is easy to check that* f *can be written as a composition of these functions, that is,* $f = g \circ h$. *Let us assume the rounding in this model is by rounding downward. More formally, for any* $x \in \mathbb{R}$, *the corresponding element in the finite precision arithmetic denote by* $fl(x) \leq x$. *Since* $\tilde{g} \circ \tilde{h}(0) = fl(\frac{1}{3}) < \frac{1}{3}$, *there is no* $x \in \mathbb{R}$ *such that* $g \circ h(x) = fl(\frac{1}{3})$. *Hence the algorithm* $\tilde{g} \circ \tilde{h}$ *for computing* f *in finite precision is not backward stable.*

One can then attempt to characterize the exact conditions under which composition of stable algorithms yield stable algorithms. This question was studied for the notion of backward stability in [Bor07] (also refer to Section 2 of [BNV23] for a discussion). In [BNV23], the authors identified two sufficient conditions based on condition numbers for a stable composition of forward stable algorithms.

1.3.1 Contributions to Numerical Stability of Algorithms

In Chapter 5, we introduce the notion of *robust numerically stable* algorithms - given certain perturbations of the desired input, the algorithm outputs some solution close to the actual solution on the desired input. This is related to the notion of mixed stability in [BNV23]. We also show that if the function satisfies some *continuity* conditions (based on the *condition number* of the problem) and the algorithm computing it is *forward stable*, then the algorithm is also *robust numerically stable*. We apply this to certain simple intermediate functions that occur in the numerical algorithm for tensor decomposition (Algorithm 8).

We also define the notion of stable probabilistic algorithms for computing set-valued functions and create a framework for analysing the stability of the composition of these algorithms. More concretely, we show that if two functions satisfy some compatibility criteria and have stable probabilistic algorithms computing them, then the composition of those two algorithms will also be a stable probabilistic algorithm for the composition of the functions.

We create this framework to perform a streamlined analysis of the numerical algorithm for the tensor decomposition problem that we study in Chapter 4. We break the algorithm down into several smaller steps (algorithms), show that each of these steps are compatible and have stable probabilistic algorithms for computing them and hence, their composition is a stable probabilistic algorithm as well.

1.4 Tensor Decompositions

Recall that we had defined rank-one tensors in Section 1.1.1. Given an order-d tensor, we want to write it as a sum of rank-one tensors of order d. This is popularly referred to as the CP decomposition with its name originating from the psychometrics community. CP is short for CANDECOMP (canonical decomposition) introduced by Carroll and Chang [CC70] and PARAFAC (parallel factors) introduced by Harshman [Har70]. More formally, given an order-d tensor $T \in \mathbb{C}^{n_1} \otimes ... \mathbb{C}^{n_d}$ (similarly for $\mathbb{R}$),

we want to write it as

$$T = \sum_{i=1}^{r} a_i^{(1)} \otimes ... \otimes a_i^{(d)} \tag{1.5}$$

where r is some positive integer and $a_i^{(j)} \in \mathbb{C}^{n_j}$ for all $i \in [r], j \in [d]$. Firstly note, that a decomposition of the tensor of this form always exists. In fact, for any tensor $T \in \mathbb{C}^{n_1} \otimes ... \mathbb{C}^{n_d}$, it can be trivially written as

$$T = \sum_{i_j \in [n_j] \text{ for all } j \in [d]} T_{i_1,\cdots,i_d} e_{i_1}^{(1)} \otimes ... \otimes e_{i_d}^{(d)}. \tag{1.6}$$

where $e_j^{(k)}$ is the j-th standard basis vector in $\mathbb{C}^{n_k}$.

The *rank* of a tensor, denoted by $\text{rank}(T)$ is the smallest integer r such that T can be written as a sum of rank-one tensors as in (1.5). From 1.6, we can conclude that for any order-d cubical tensor $T \in (\mathbb{C}^n)^{\otimes d}$, $\text{rank}(T) \leq n^d$. With a slightly clever argument, one can show that for any such tensor T, $\text{rank}(T) \leq n^{d-1}$. (Lemma 16.9 in [Sap15]). A major open question in this field is to find an explicit tensor T of order-3 with super-linear rank.

Example 1.4.1 (Jan Draisma's talk in AG'23). *Let $T \in (\mathbb{R}^2)^{\otimes 3}$ such that $T_{111} = 1, T_{2,1,1} = 2, T_{2,2,1} = 6, T_{1,2,2} = -1, T_{ijk} = 0$ otherwise. Then trivially it can be written in the form of (1.5) using four summands*

$$T = 1.(e_1 \otimes e_1 \otimes e_1) + 2.(e_2 \otimes e_1 \otimes e_1) + 6.(e_2 \otimes e_2 \otimes e_1) + (-1)e_1 \otimes e_2 \otimes e_2.$$

But there is a smaller decomposition with two summands

$$T = (e_1 + 2e_2) \otimes (e_1 + 3e_2) \otimes e_1 + e_1 \otimes e_2 \otimes (-3e_1 - e_2).$$

Using the relation with polynomials, as described in Section 1.1.3, the the polynomial f_T corresponding to T can be equivalently written as

$$\begin{aligned} f_T &= x_1y_1z_1 + 2x_2y_1z_1 + 6x_2y_2z_1 - x_1y_2z_2 \\ &= x_1y_1z_1 + 2x_2y_1z_1 + 6x_2y_2z_1 + 3x_1y_2z_1 - 3x_1y_2z_1 - x_1y_2z_2 \\ &= (x_1 + 2x_2)(y_1 + 3y_2)x_1 + x_1y_2(-3z_1 - z_2). \end{aligned}$$

Again using the tensor polynomial equivalence, it follows that the tensor has the smaller decomposition with two summands.

The definition of tensor rank is the exact analogue of matrix rank. However, one key difference is that rank of a real-valued tensor can be different over $\mathbb{R}$ and $\mathbb{C}$. One example from [Kru77] will help illustrate this issue. Let us consider the following tensor $T \in (\mathbb{R}^2)^{\otimes 3}$ such that

$$T_{1,1,1} = T_{2,2,1} = T_{1,2,2} = 1 \text{ and } T_{2,1,2} = -1.$$

This tensor has rank 3 over $\mathbb{R}$ [tB91] but has rank 2 over $\mathbb{C}$.

Given an order-d tensor $T \in \mathbb{C}^{n_1} \otimes ... \otimes \mathbb{C}^{n_d}$, one can ask the following natural algorithmic questions:

1. *Decision problem:* Given r, does there exist sets of vectors $\{a_i^{(j)}\}_{i \in [r]} \in \mathbb{C}^{n_j}$ for all $j \in [d]$ such that T has a decomposition of the form (1.5)?

2. *Search problem:* If such a decomposition exists, find the decomposition.

The problem of determining the rank of a tensor T was shown to be NP-hard (even for order $d = 3$) by Håstad [Hås89].

1.4.1 Symmetric tensor decomposition

Recall that we had defined symmetric tensors in Section 1.1.1. One can also extend the notion of rank defined in Section 1.4 to the special case of symmetric tensors. The order-d rank-1 symmetric tensors are now of the form $T = u^{\otimes d}$ where $T_{i_1,\cdots,i_d} = \prod_{j\in[d]} u_{i_j}$. Let $T \in \mathbb{C}^n \otimes ... \otimes \mathbb{C}^n$ be an order-d symmetric tensor. Following (1.5), we can define the *symmetric tensor decomposition* analogously as

$$T = \sum_{i=1}^{r} u_i \otimes u_i \otimes ... \otimes u_i \text{ where } u_i \in \mathbb{C}^n. \tag{1.7}$$

It is slightly more non-trivial than the general case to show that every symmetric tensor has a decomposition of the form (1.7). We will include a proof sketch here for completeness of this exposition.

Lemma 1.4.2. *Every order-d symmetric tensor has a decomposition of the form 1.7.*

Proof. From the tensor-polynomial equivalence in Section 1.1.3, we get that every order-d symmetric tensor $T \in (\mathbb{C}^n)^{\otimes d}$ can be written as a homogeneous degree-d polynomial $f_T \in \mathbb{C}[x_1, \cdots, x_n]_d$. Hence, T has a decomposition of the form (1.7) iff $f_T = \sum_{i=1}^{s} \ell_i^d$ where ℓ_i are linear forms in $\mathbb{C}[x_1, \cdots, x_n]_1$. Let $\mathcal{L}$ be defined as $\text{span}_{\mathbb{C}}\{\ell^d | \ell \text{ is a linear form } \in \mathbb{C}[x_1, \cdots, x_n]_1\}$. We want to show that $\mathcal{L} = \mathbb{C}[x_1, \cdots, x_n]_d$.

The space $\mathcal{L}$ is trivially contained in $\mathbb{C}[x_1, \cdots, x_n]_d$. For the opposite direction, note that the set of monomials of degree-d given by $\{x_1^{e_1} \cdots x_n^{e_n} | e_1 + ... + e_n = d\}$ forms a basis of $\mathbb{C}[x_1, \cdots, x_n]_d$. From [Fis94], we get that every degree-d monomial can be written as a sum of d-th powers of linear forms. More formally, this follows from the following expression

$$y_1 ... y_t = \frac{1}{2^{t-1} t!} \sum_{e_2 \in \{0,1\}, ..., e_t \in \{0,1\}} (-1)^{e_2 + ... + e_t} \left(x_1 + \sum_{i \in \{2, ..., t\}} (-1)^{e_i} x_i \right)^t. \tag{1.8}$$

Hence, $\mathbb{C}[x_1, \cdots, x_n]_d = \text{span}_{\mathbb{C}}\{x_1^{e_1} \cdots x_n^{e_n} | e_1 + \cdots + e_n = d\} \subseteq \mathcal{L}$ and this gives us the desired result. □

We can then define the *symmetric rank* (over $\mathbb{C}$) for any symmetric tensor T similarly to be

$$\text{rank}_S(T) = \min \left\{ r : T = \sum_{i=1}^{r} a_i \otimes \cdots \otimes a_i, \text{ where } a_i \in \mathbb{C}^n \right\} \tag{1.9}$$

One can ask the following question: Is the symmetric rank of a tensor same as the rank of the tensor? This was famously known as the Comon's conjecture and was resolved negatively by Yaroslav Shitov in [Shi16]. One can define the *symmetric decision problem* and *symmetric search problem* in a similar way as in Section 1.4. The symmetric tensor rank of T is NP-hard to compute even for $d = 3$ [Shi16] and hence, the the *symmetric decision problem* is NP-hard as well.

1.4.2 Diagonalisable tensors:

We focus our attention on order-3 symmetric tensors. From (1.7), it has a decomposition of the form $T = \sum_{i=1}^{r} u_i^{\otimes 3}$ and in order to obtain efficient algorithms, one can impose an additional linear independence condition on the u_i. Note that such a decomposition is *essentially unique* if it exists (up to a permutation of the u_i's and scaling by cube roots of unity) [Kru77, Har70]. There is a traditional distinction between *undercomplete* decompositions, when the number of summands $r \leq n$ in (1.5), and *overcomplete* decompositions, where $r > n$. We consider only undercomplete decompositions because of the linear independence condition on the u_i. Moreover, we impose the additional condition that r is exactly equal to n, i.e., we focus on *complete decompositions*. We say that a tensor is *diagonalisable* if it satisfies these two conditions.

One can ask the following algorithmic question: Given a diagonalisable tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$, recover the vectors $u_1, \cdots, u_n \in \mathbb{C}^n$ such that $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. Note that, this is a special case of the search problem of tensor decomposition. One of the first algorithms to give provable guarantees for this problem was Jennrich's algorithm [Har70, LRA93]. This algorithm depends on the method of simultaneous diagonalisation and even works for more general settings such as ordinary tensors.

One can also study the *decision* version of the problem: Given an arbitrary order-d symmetric tensor T, is T diagonalisable? Using the relation between tensors and polynomials in Section 1.1.3, we can see that a homogeneous degree-d polynomial $f \in \mathbb{K}[x_1, \cdots, x_n]$ can be written as a sum of d-th powers of linear forms over $\mathbb{K}$ if and only if there exist $v_i \in \mathbb{K}$ such that the corresponding symmetric tensor T_f can be decomposed as $T_f = \sum_i v_i^{\otimes d}$. In a joint work with Pascal Koiran [KS22a, KS23] (which forms Chapter 2 of this thesis), we give a randomized polynomial-time algorithm for this problem in the algebraic model of computation (refer to the BSS model in Section 1.2.1) with oracle access to the *black-box* for the homogeneous degree-d polynomial T_f. We also extend this to the case of undercomplete decompositions ($r \leq n$) via a reduction to the complete case.

1.4.3 Approximate tensor decomposition

As explained above, an order-3 symmetric tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ is called diagonalisable if there exist linearly independent vectors $u_i \in \mathbb{C}^n$ such that $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. The objective of the ε-approximation problem for tensor decomposition is to find linearly independent vectors $u'_1, \cdots, u'_n$ such that there exists a permutation $\pi \in S_n$ where

$$||\omega_i u_{\pi(i)} - u'_i|| \leq \varepsilon$$

with ω_i a cube root of unity. Here ε is the desired accuracy parameter given as input. Hence the problem is essentially that of approximating the vectors u_i appearing in the decomposition of T. Note that this is a *forward approximation* in the sense of numerical analysis (refer to Section 1.3 for the discussion on different kinds of numerical algorithms and compare with Definitions 3.3.2 and 3.3.3).

1.4.4 Algorithms for Tensor Decompositions

One of the central spectral algorithms for computing tensor decompositions is Jennrich's Algorithm [Har70, LRA93, Moi18]. This algorithm, also referred to in the literature as the "simultaneous diagonalisation algorithm," was one of the first to give

provable guarantees for tensor decomposition. If an order-3 input tensor satisfies certain genericity conditions this algorithm returns the unique decomposition (up to permutation and scaling) almost surely. More formally, let $T \in \mathbb{C}^{n_1} \otimes \mathbb{C}^{n_2} \otimes \mathbb{C}^{n_3}$ be an order-3 tensor of the form (1.5) with the following additional conditions on the $u_i^{(j)}$

- The vectors $u_i^{(1)}$ are linearly independent.
- The vectors $u_i^{(2)}$ are linearly independent.
- The vectors $u_i^{(3)}$ are pairwise linearly independent. Following the definition of Kruskal rank from Section 4.3.1, this implies that k-rank$(\{u_i^{(3)}\}_{i\in[r]}) \geq 2$.

We will refer to these conditions as the *genericity conditions*. Note that if a tensor has such a decomposition, then it is unique (up to permutation and scaling by complex numbers across each mode of a summand such that their product is 1) (refer to the discussion in Section 4.3.1). Then Jennrich's algorithm returns such a decomposition exactly. This algorithm can also be generalized to higher order tensors which satisfy similar *genericity* conditions. Moreover, it is shown in [BCMV14] that the algorithm is robust to noise in the input. Namely, it was shown that for an input tensor $\tilde{T} = \sum_{i=1}^{n} v_i^{\otimes 3} + E$ where E is some arbitrary inverse-polynomial noise, Jennrich's algorithm can also be used to output a decomposition $\tilde{v}_i$ such that $||v_i - \tilde{v}_i|| \leq \varepsilon$. At the heart of the robustness analysis of the Jennrich's Algorithm in [BCMV14] (refer to their Appendix A) is the following statement about diagonalisability of perturbed matrices: Let M be a diagonalisable matrix that can be written as $M = UDU^{-1}$ where the condition number of U is bounded and let $\tilde{M}$ be another matrix such that $||M - \tilde{M}||$ is small. Then $\tilde{M}$ is also diagonalisable, has distinct eigenvalues and the eigenvectors of M and $\tilde{M}$ are close. This is similar in spirit to Proposition 1.1 in [BGVKS22] (Theorem 3.3.11 in this thesis.)

Note that the genericity conditions restrict the number of summands in the decomposition, r to be at most $\min\{n_1, n_2, n_3\}$. Another interesting line of research has focussed on exploring the *overcomplete* regime, that is $r > \min\{n_1, n_2, n_3\}$. For the rest of this section, to ease notation, we will restrict our attention to symmetric tensors which forces $r > n$.

One can drop the genericity condition entirely and attempt to decompose an arbitrary low-rank tensor given as input. For symmetric tensors with constant rank, such an algorithm can be found in [BSV21a]. This algorithm was recently extended to slightly superconstant rank in [PSV22].

One can also aim to decompose tensors under milder genericity conditions that still preserve the uniqueness of the decomposition. For order-4 tensors, when $T = \sum_{i\in[r]} u_i^{\otimes 4}$ and the u_i are in *general positions*, then the FOOBI (Fourth-Order-Only Blind Identification) algorithm from [DLCC07] can recover the decomposition where the number of summands r is at most $O(d^2)$. Still other algorithms for symmetric tensor decomposition can be found in the algebraic literature, see e.g. [BCMT09, BGI11]. These two papers do not provide any complexity analysis for their algorithms.

Furthermore, there are several algorithms for tensor decompositions which use optimization techniques. One of the most notable algorithm for computing the CP decomposition (refer to (1.5) for a definition) is the *Alternating Least Squares* (ALS) algorithm [CC70, Har70], often referred to as the "workhorse" algorithm for computing a CPD. The key idea is to solve the *least-squares* optimization problem on a mode of the tensor (refer to the beginning of Section 1.1.1 for a definition) while keeping

the other modes fixed and solving this optimization problem for all the modes. For a more formal explanation, refer to [KB09].

1.4.5 Contributions to tensor decomposition

Recall that an order-3 tensor $T \in (\mathbb{C}^n)^{\otimes 3}$ is called diagonalisable if there exist linearly independent vectors $u_1, \cdots, u_n \in \mathbb{C}^n$ such that T can be decomposed as in (1.5).

Definition 1.4.3 (Condition number of a diagonalisable symmetric tensor)**.** *Let T be a diagonalisable symmetric tensor over $\mathbb{C}$ such that $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. Let $U \in M_n(\mathbb{C})$ be the matrix with rows $u_1, \ldots, u_n$. We define the tensor decomposition condition number of T as: $\kappa(T) = ||U||_F^2 + ||U^{-1}||_F^2$.*

We will show in Section 4.3 that $\kappa(T)$ is well defined: for a diagonalisable tensor the condition number is independent of the choice of U. Note that when U is close to a singular matrix, the corresponding tensor is poorly conditioned, i.e., has a large condition number. This is not surprising since our goal is to find a decomposition where the vectors u_i are linearly independent.

Our main result is a randomized polynomial time algorithm in the finite precision model which on input a diagonalisable tensor, an estimate B for the condition number of the tensor and an accuracy parameter ε, returns a forward approximate solution to the tensor decomposition problem (following the definition in Section 1.4.3).

In the following, we denote by $T_{MM}(n)$ the number of arithmetic operations required to multiply two $n \times n$ matrices in a numerically stable manner. If ω denotes the exponent of matrix multiplication, it is known that $T_{MM}(n) = O(n^{\omega+\eta})$ for all $\eta > 0$ (see Section 3.1 for details).

Theorem 1.4.4 (Main Theorem)**.** *There is an algorithm which, given a diagonalisable tensor T, a desired accuracy parameter ε and some estimate $B \geq \kappa(T)$, outputs an ε-approximate solution to the tensor decomposition problem for T in*

$$O(n^3 + T_{MM}(n) \log^2 \frac{nB}{\varepsilon})$$

arithmetic operations on a floating point machine with

$$O(\log^{12}(\frac{nB}{\varepsilon}) \log n)$$

bits of precision, with probability at least $\left(1 - \frac{1}{n} - \frac{12}{n^2}\right)\left(1 - \frac{1}{\sqrt{2n}} - \frac{1}{n}\right)$.

The corresponding algorithm appears as Algorithm 8 in Section 4.3. A simplified version of this algorithm is presented in Section 4.1.1. The following are the important conclusions from the above theorem:

- The number of bits of precision required for this algorithm is polylogarithmic in n, B and $\frac{1}{\varepsilon}$.
- The running time as measured by the number of arithmetic operations is $O(n^3)$ for all $\varepsilon = \frac{1}{\text{poly(n)}}$, i.e., it is linear in the size of the input tensor. This requires the use of fast matrix multiplication. With standard matrix multiplication, the running time is quasilinear instead of linear (i.e., it is multiplied by a polylogarithmic factor). The bit complexity of the algorithm is also quasilinear.
- The algorithm can provide inverse exponential accuracy, i.e., it still runs in polynomial time even when the desired accuracy parameter is $\varepsilon = \frac{1}{\exp(n)}$.

In order to obtain this result we combine techniques from algorithm design and algorithm analysis; the main ideas are outlined in Sections 4.1.1, 4.1.3 and 4.3.4. To the best of our knowledge, this is the first tensor decomposition algorithm shown to work in polylogarithmic precision. Moreover, this algorithm is also the first to run in a linear number of arithmetic operations (i.e., prior to this work no linear time algorithm was known, *even in the exact arithmetic model*). This is based on joint work with Pascal Koiran [KS23, KS22b].

The bounds on the number of bits of precision can be improved up to $\log^4(\frac{nB}{\varepsilon})\log n$. A detailed proof of this can be found in [KS22b]. We lose out on some bits of precision to give a simpler and more streamlined error analysis of the algorithm. Refer to the discussions in Section 4.1.3 and Section 5.2 for more details.

Condition numbers for tensor decomposition

Recall that we had discussed the notion of numerical algorithms and the associated notion of condition numbers in Section 1.3.

For the purpose of this thesis, we have worked with the somewhat ad-hoc choice of $\kappa(T)$ (refer to Definition 1.4.3) as our condition number because this parameter controls the numerical precision needed for our main algorithm, as shown by Theorem 4.3.4[2]. In particular, we have found it more convenient to work with $\kappa(T)$ than with a quantity such as $||U||.||U^{-1}||$, commonly used as a condition number in numerical linear algebra.

The results presented in this thesis are in stark contrast with those of Beltrán et al. [BBV19]. That paper analyzes a class of tensor decomposition algorithms related to Jennrich's algorithm. Their conclusion is that all these "pencil-based algorithms" are numerically unstable.

A precise comparison of our results with the numerical instability result of [BBV19] is delicate because we do not work in the same setting. In particular, they work with ordinary instead of symmetric tensors; they do not work with the same condition number; and their result is obtained for undercomplete rather than complete decompositions. We believe that the main reason why we obtain a positive result is due to yet another difference, namely, the use of randomization in step (i) of our algorithm. In the setting of [BBV19] one would have to take two *fixed* linear combinations $T^{(a)}, T^{(b)}$ of the slices. Essentially, they show that for every fixed choice of a pair of linear combinations, there are input tensors for which this choice is bad; whereas we show that for every (well conditioned) input T, most choices of a and b are good.

Beltrán et al. conclude their paper with the following sentence: "We hope that these observations may (re)invigorate the search for numerically stable algorithms for computing CPDs."[3] The algorithm presented in this thesis answers their call, at least for the case of complete decomposition of symmetric tensors. We believe that our techniques can also be applied to decomposition of ordinary tensors. We have chosen to focus on symmetric tensors because this setting is somewhat simpler technically. In future work, we plan to extend this work to the case of undercomplete decompositions. Note that this will require a change in the definition of the condition number $\kappa(T)$; the role of U^{-1} will now be played by the Moore-Penrose pseudoinverse.

[2] $\kappa(T)$ also appears in the sublinear term for the arithmetic complexity of the algorithm.

[3] CPD stands for *Canonical Polyadic Decomposition, i.e., decomposition as a sum of rank-1 tensors.*"

1.5 Reconstruction Algorithms

Arithmetic circuit reconstruction is the following algorithmic problem: For an input polynomial f, typically given by a black box (refer to Section 1.2.1 for a definition), the goal is to find the smallest circuit computing f within some class $\mathcal{C}$ of arithmetic circuits. This problem can be divided in two subproblems: a decision problem (can f be computed by a circuit of size s from the class $\mathcal{C}$?) and the reconstruction problem *proper* (the actual construction of the smallest circuit for f). The *proper reconstruction problem* is an algebraic analogue of the exact learning problem for Boolean circuits.

1.5.1 Absolute reconstruction

An interesting subclass of reconstruction algorithms is the problem of *absolute reconstruction*, namely, in the case where $\mathcal{C}$ is a class of circuits over the field of complex numbers. The name is borrowed from *absolute factorization*, a well-studied problem in computer algebra (see e.g. [CG05, CL07, Gao03, Sha09]). Most of the existing reconstruction algorithms appeal to a polynomial factorization subroutine, see e.g. [GMKP17, GMKP18, KS09, Kay11, KNST18, KS19, Shp09]. This typically yields polynomial time algorithms over finite fields or the field of rational numbers. However, in standard models of computation such as the unit-cost RAM or the Turing machine this approach does *not* yield polynomial time algorithms for absolute reconstruction. This is true even for the decision version of this problem. In the Turing machine model, the difficulty is as follows. We are given an input polynomial f, say with rational coefficients, and want to decide if there is a small circuit $C \in \mathcal{C}$ for f, where C may have complex coefficients. After applying a polynomial factorization subroutine, a reconstruction algorithm will manipulate polynomials with coefficients in a field extension of $\mathbb{Q}$. If this extension is of exponential degree, the remainder of the algorithm will not run in polynomial time. This point is explained in more detail in [KS21] on the example of a reconstruction algorithm due to Neeraj Kayal [Kay11]. One way out of this difficulty is to work in a model where polynomial roots can be extracted at unit cost, as suggested in a footnote of [GGKS19]. We will work instead in more standard models, namely, the Turing machine model or the unit-cost RAM over $\mathbb{C}$ with arithmetic operations only (an appropriate formalization is provided by the Blum-Shub-Smale model of computation [BCSS98, BSS89], refer to Section 1.2.1).

1.5.2 Polynomial equivalence testing:

Definition 1.5.1. *Two polynomial $f, g \in \mathbb{K}[\boldsymbol{x}]$ (where $\boldsymbol{x} = \{x_1, \cdots, x_n\}$) are said to be equivalent over $\mathbb{K}$, if there exists an invertible linear transformation $A \in GL_n(\mathbb{K})$ such that $f(\boldsymbol{x}) = g(A\boldsymbol{x})$.*

Note that two polynomials can be equivalent over $\mathbb{K}$ but not over a subfield $K \subset \mathbb{K}$. The difference in the choice of fields can be illustrated by the following example:

Example 1.5.2. *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 $P_3(x_1, x_2) = x_1^3 + x_2^3$ over $\mathbb{R}$ and $\mathbb{C}$ but not over $\mathbb{Q}$.

An equivalence test for a family of polynomials $\mathcal{C} \subseteq K[x_1, \cdots, x_n]$ (where K is some underlying field) is defined to be the following algorithmic task: Given polynomial g, check if there exists some $f \in \mathcal{C}$ such that f and g are equivalent over $\mathbb{K}$. Note that this is a special case of the decision version of the reconstruction problem for the circuit class $\mathcal{C}$.

If the equivalence test succeeds, then another algorithmic question can be to output the circuit f and the corresponding linear transformation A. Note again this is a special case of the proper reconstruction problem.

The input polynomial g can be represented in two ways: either as a list of coefficients of the polynomial (referred to as the polynomial equivalence (PE) problem in [Gup22]) or as a *black-box* (refer to Section 1.2.1).

Previous works: We define a few polynomial families before..

- Det_n: The determinant polynomial takes in a matrix with variable entries $X = (x_{i,j})_{i,j\in[n]}$ and computes the determinant of the matrix. More formally,

$$\mathrm{Det}_n = \sum_{\sigma \in S_n} \mathrm{sign}(\pi) \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

- Perm_n: The permanent polynomial takes in a matrix with variable entries $X = (x_{i,j})_{i,j\in[n]}$ and computes the permanent of the matrix. More formally,

$$\mathrm{Perm}_n = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}. \tag{1.10}$$

- $\mathrm{IMM}_{n,d}$: The iterated matrix multiplication returns the $(1,1)$-th entry of the product of d matrices of variables of size $n \times n$ each. More formally, it returns the $(1,1)$-th entry of $(X_1 \cdots X_d)$, where for all $i \in [d]$, $X_i \in (\mathbb{K}[\bar{x}])^{n\times n}$.

In [Kay11], randomized equivalence tests for the families Perm_n and Det_n were given and [KNST18] gave a randomized equivalence test for the family of $\mathrm{IMM}_{n,d}$. These tests run in polynomial time in the standard Turing Machine model when the underlying field is $\mathbb{Q}$ ([KNST18] works over finite fields as well). But when the algorithm is considered over $\mathbb{C}$, they are not known to run in polynomial time in the standard BSS model of computation (refer to Section 1.2.1 and the discussion in Section 1.5.1). [KNST18] gave a randomized polynomial time equivalence test for the family of $\mathrm{IMM}_{n,d}$ over $\mathbb{Q}$ and finite fields. A polynomial time randomized equivalence test for Det_n over $\mathbb{Q}$ and finite fields was given in [GGKS19, KS19].

Sums of powers of linear forms

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

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{r} l_i(x_1, \ldots, x_n)^d \tag{1.11}$$

where the l_i are linear forms. Such a decomposition is sometimes called a Waring decomposition, or a symmetric tensor decomposition. The smallest possible value of r is the symmetric tensor rank of f, and it is NP-hard to compute already for $d = 3$ [Shi16]. One can nevertheless obtain polynomial time algorithms by restricting to a constant value of r [BSV21b]. In this thesis, we assume instead that the linear

forms l_i are linearly independent (hence $r \leq n$). This setting was already studied by Kayal [Kay11]. It turns out that such a decomposition is unique when it exists, up to a permutation of the l_i and multiplications by d-th roots of unity. This follows for instance from Kruskal's uniqueness theorem. For a more elementary proof, see [Kay11, Corollary 5.1] and [KS21, Section 3.1].

Under this assumption of linear independence, the case $r = n$ is of particular interest. In this case, f is *equivalent* to the sum of d-th powers polynomial

$$P_d(x) = x_1^d + x_2^d + \cdots + x_n^d \tag{1.12}$$

in the sense that $f(x) = P_d(Ax)$ where A is invertible. (refer to Section 1.5.2) A test of equivalence to P_d was provided in [Kay11]. The resulting algorithm provably runs in polynomial time over the field of rational numbers, but this is not the case over $\mathbb{C}$ due to the appeal to polynomial factorization. The first equivalence test to P_d running in polynomial time over the field complex numbers was given in [KS21] for $d = 3$. In a joint work with Pascal Koiran, we extend this result to arbitrary degree [KS22a, KS23] and this forms a major chunk of Chapter 2 of this thesis. In the general case $r \leq n$ we can first compute the number of essential variables of f [Car06, Kay11]. Then we can do a change of variables to obtain a polynomial depending only on its first r variables [Kay11, Theorem 4.1], and conclude with a test of equivalence to P_r (see [KS21, Proposition 44] for details).

Equivalence and reconstruction algorithms over $\mathbb{Q}$ are number-theoretic in nature in the sense that their behavior is highly sensitive to number-theoretic properties of the coefficients of the input polynomial. This point is clearly illustrated by Example 1.5.2

By contrast, equivalence and reconstruction algorithms over $\mathbb{R}$ and $\mathbb{C}$ are of a more geometric nature.

1.5.3 Contributions to Absolute Reconstruction

Our main contributions are as follows: Recall that P_d is the sum of d-th powers polynomials (1.12), and let us assume that the input $f \in \mathbb{C}[x_1, \ldots, x_n]$ is a homogeneous polynomial of degree d.

(i) For $d = 3$, we improve by a factor of n on the running time of the test of equivalence to P_3 from [KS21] presented in Section 2.1.1. The price to be paid for this improvement is that the algorithm now has two-sided error.

(ii) For $d > 3$, we provide the first blackbox algorithm for equivalence to P_d with running time polynomial in n and d (more specifically, $O(n^2 d)$ calls to the blackbox and $O(n^2 d \log^2(d) \log\log(d) + n^{\omega+1})$ arithmetic operations) where ω is the exponent of matrix multiplication, in an algebraic model where only arithmetic operations and equality tests are allowed (i.e., computation of polynomial roots is *not allowed*).

(iii) For $d > 3$, when f has rational coefficients this blackbox algorithm runs in polynomial time in the bit model of computation. More precisely, the running time is polynomial in n, d and the maximal bit size of any coefficient of f. This yields the first test of equivalence to P_d over $\mathbb{C}$ with polynomial running time in the bit model of computation.

As outlined in Section 1.5.2, these results have application to decomposition into sums of powers of linearly independent linear forms over $\mathbb{C}$. Namely, we can decide

whether the input polynomial admits such a decomposition, and if it does we can compute the number of terms r in such a decomposition. The resulting algorithm runs in polynomial time in the algebraic model of computation, as in item (ii) above; when the input has rational coefficients it runs in polynomial time in the bit model of computation, as in (iii) (refer to Appendix 2.7 for a detailed complexity analysis). This is the first algorithm with these properties. It can be viewed as an algebraic, high order, black box version of Jennrich's algorithm.

Using the relation to tensor decomposition problem mentioned in Section 1.4.2, if an order d-tensor $T \in K^{n\times\cdots\times n}$ is given as a blackbox, we give an algorithm that runs in time $\text{poly}(n, d)$ to check if there exist linearly independent vectors $v_i \in \mathbb{K}^n$ such that $T = \sum_{i=1}^{t} \alpha_i v_i^{\otimes d}$ for some $t \leq n$. Note here that $K \subseteq \mathbb{C}$ and $\mathbb{K} = \mathbb{C}$ or $\mathbb{R}$.

As an intermediate result, we obtain a new randomized algorithm for checking that k input matrices commute (for further details, see Lemma 2.1.3 and the discussion related to matrix commutativity in Section 2.1).

Finally, we show that our linear algebraic approach can be extended to the computation of the actual decomposition. For instance, when $f \in \mathbb{C}[x_1, \ldots, x_n]$ is equivalent to P_d, we can compute an invertible matrix A such that $f(x) = P_d(Ax)$. We emphasize that for this result we must step out of our usual algebraic model, and allow the computation of polynomial roots. The matrix A is indeed not computable from f with arithmetic operations only, as shown by the example in Section 1.5.2. We therefore obtain an alternative to the algorithm from [Kay11] for the computation of A. That algorithm relies on multivariate polynomial factorization, whereas our algorithm relies on matrix diagonalization (this is not an algebraic task since diagonalizing a matrix requires the computation of its eigenvalues).

Chapter 2

Absolute Reconstruction for Sums of Powers of Linear Forms

In this chapter, we look at the following decision problem: If an arbitrary homogeneous degree-d polynomial over $\mathbb{R}$ or $\mathbb{C}$ is given as blackbox, can it be written as sums of powers of linearly independent linear forms? This is related to the question of absolute reconstruction of the same family of polynomials (as explained in Section 1.5.1) and the connection of this question to tensor decompositions has been explored in detail in Section 1.4.2. This is based on joint work with Pascal Koiran and a preliminary version of the results of this chapter appear in [KS22a]. The full version appears in [KS23].

2.1 Introduction

2.1.1 Methods and proof strategies

Sums of cubes

For $d = 3$, the first test of equivalence to P_d running in polynomial time over $\mathbb{C}$ and over $\mathbb{R}$ was given in [KS21]. There, the problem was treated as a tensor decomposition problem which was then solved by methods from linear algebra. We briefly outline this approach since the present chapter improves on it and extends it to higher degree. Let $f \in \mathbb{K}[x_1, \ldots, x_n]$ be the input polynomial, where $\mathbb{K}$ is the field of real or complex numbers. We can form with the coefficients of f a symmetric tensor[1] of order three $T = (T_{ijk})_{1 \leq i,j,k \leq n}$ so that

$$f(x_1, \ldots, x_n) = \sum_{i,j,k=1}^{n} T_{ijk} x_i x_j x_k.$$

This tensor can be cut into n slices $T_1, \ldots, T_n$ where $T_k = (T_{ijk})_{1 \leq i,j \leq n}$. Each slice is a symmetric matrix of size n. By abuse of language we also say that $T_1, \ldots, T_n$ are the slices of f. The equivalence test to P_3 proposed in [KS21] works as follows.

1. On input $f \in \mathbb{K}[x_1, \ldots, x_n]$, pick a random matrix $R \in M_n(\mathbb{K})$ and set $h(x) = f(Rx)$.
2. Let $T_1, \ldots, T_n$ 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.

[1] Recall that a tensor of order d is symmetric it is invariant under all $d!$ permutations of its indices.

This simple randomized algorithm has one sided error: it can fail (with low probability) only when f is equivalent to P_3. Its analysis is based on the following characterization [KS21, Section 3.2]:

Theorem 2.1.1. *A degree 3 homogeneous polynomial $f \in \mathbb{K}[x_1, ..., x_n]$ is equivalent to P_3 iff its slices $T_1, ..., T_n$ span a non-singular matrix space and the slices are simultaneously diagonalisable by congruence, i.e., there exists an invertible matrix $Q \in M_n(\mathbb{K})$ such that $Q^T T_i Q$ is diagonal for all $i \in [n]$.*

Extension to higher degree:

In order to extend the approach of Section 2.1.1 to higher order, we associate to a homogeneous polynomial of degree d the (unique) symmetric tensor T of order d such that

$$f(x_1, \ldots, x_n) = \sum_{i_1,\ldots,i_d=1}^{n} T_{i_1 \ldots i_d} x_{i_1} x_{i_2} \ldots x_{i_d}.$$

A *slice* of T (or by abuse of language, a slice of f) is a matrix of size n obtained by fixing the values of $d-2$ indices. We show in Section 2.3.2 that Theorem 2.1.1 can be generalized as follows:

Theorem 2.1.2. *A degree d homogeneous polynomial $f \in \mathbb{C}[x_1, \ldots, x_n]$ is equivalent to $P_d = \sum_{i=1}^n x_i^d$ if and only if its slices span a nonsingular matrix space and the slices are simultaneously diagonalizable by congruence, i.e., there exists an invertible matrix $Q \in M_n(\mathbb{C})$ such that for every slice S of f, the matrix $Q^T S Q$ is diagonal.*

This characterization is satisfactory from a purely structural point of view, but not from an algorithmic point of view because the number of slices of a tensor of order d is exponential in d. Recall indeed that a tensor of size n and order d has $\frac{d(d-1)}{2} n^{d-2}$ slices: a slice is obtained by fixing the values of $d-2$ indices and hence, each slice is a matrix of size n. The tensors encountered in this chapter are all symmetric since they originate from homogeneous polynomials. Taking the symmetry constraints into consideration reduces the number of distinct slices to $\binom{n+d-3}{d-2}$ at most: this is the number of multisets of size $d-2$ in a set of n elements, or equivalently the number of monomials of degree $d-2$ in the variables $x_1, \ldots, x_n$. This number remains much too large to reach our goal of a complexity polynomial in n and d. This problem has a surprisingly simple solution: our equivalence algorithm needs to work with 3 slices only! This is true already for $d = 3$, and is the reason why we can save a factor of n compared to the algorithm of Section 2.1.1. More precisely, we can replace the loop at line 3 of that algorithm by the following test: check that $T_1' T_2$ is diagonalizable, and commutes with $T_1' T_3$ (recall that $T_1' = T_1^{-1}$). It may be surprising at first sight that we can work with the first 3 slices only of a tensor with n slices. To give some plausibility to this claim, note that T_1, T_2, T_3 are not slices of the input f, but slices of the polynomial $h(x) = f(Rx)$ obtained by a random change of variables. As a result, each slice of h contains some information on *all* of the n slices of f. The algorithm for order $d > 3$ is of a similar flavor, but one must be careful in the choice of the 3 slices from h.

Our algorithms are therefore quite simple (and the equivalence algorithm for $d = 3$ is even somewhat simpler than the algorithm from Section 2.1.1); but their analysis is not so simple and forms the bulk of this chapter. In fact, analysing the case of "negative" inputs, i.e. input polynomials that are not equivalent to any polynomial in $\mathcal{P}_d$, forms the bulk of this chapter. For $d > 3$, the notion of "weak-singularity" of matrices (Definition 2.3.7) will be introduced which along with the notions of

"commutativity property" and "diagonalisability property" helps us to give us another equivalent criterion for testing equivalence to a polynomial in $\mathcal{P}_d$ in Theorem 2.3.8. Finally, the crucial part of the proof (for $d > 3$, and already for $d = 3$) is to show that testing commutativity of two matrices and diagonalisability of one matrix is enough for testing these properties for any "symmetric family of symmetric matrices" (refer to Definition 2.3.12) with high probability.

Note here that an arbitrary slice of the polynomial is hard to compute, when the polynomial is given as blackbox (because that requires computing arbitrary degree-d partial derivatives using the blackbox). Hence, this particular choice of slices is crucial because they can be computed in polynomial time.

Real versus complex field. For $\mathbb{K} = \mathbb{R}$ and even degree there is obviously a difference between sums of d-th powers of linear forms and linear combinations of d-th powers. In this chapter we wish to allow arbitrary linear combinations. For this reason, in the treatment of the high order case ($d > 3$) we are not interested in equivalence to P_d only. Instead, we would like to know whether the input is equivalent to some polynomial of the form $\sum_{i=1}^{n} \alpha_i x_i^d$ with $\alpha_i \neq 0$ for all i. We denote by $\mathcal{P}_d$ this class of polynomials (one could even assume that $\alpha_i = \pm 1$ for all i). At first reading, there is no harm in assuming that $\mathbb{K} = \mathbb{C}$. In this case, one can assume without loss of generality that $\alpha_i = 1$ for all i. For $\mathbb{K} = \mathbb{R}$, having to deal with the whole of $\mathcal{P}_d$ slightly complicates notations, but the proofs are not significantly more complicated than for $\mathbb{K} = \mathbb{C}$. For this reason, in all of our results we give a unified treatment of the two cases $\mathbb{K} = \mathbb{C}$ and $\mathbb{K} = \mathbb{R}$.

Relation to matrix commutativity testing: As a byproduct of our analysis of the degree 3 case, we obtain a randomized algorithm for testing the commutativity of a family of matrices $A_1, \ldots, A_k$. The naive algorithm for this would check that $A_i A_j = A_j A_i$ for all $i \neq j$. Instead, we propose to test the commutativity of two random linear combinations of the A_i. The resulting algorithm has one sided-error, and its probability of error can be bounded as follows:

Lemma 2.1.3. *Let $A_1, ..., A_k \in M_n(\mathbb{K})$. We take two random linear combinations $A_\alpha = \sum_{i\in[k]} \alpha_i A_i$ and $A_\beta = \sum_{i\in[k]} \beta_i A_i$, where the α_i and β_i are picked independently and uniformly at random from a finite set $S \subset \mathbb{K}$. If $\{A_i\}_{i\in[k]}$ is not a commuting family, then the two matrices A_α, A_β commute with probability at most $\frac{2}{|S|}$.*

The resulting algorithm is so simple and natural that it may already be known to some readers, but we could not find in the literature on commutativity testing. Commutativity testing has been studied in particular in the setting of black box groups, in the classical [Pak12] and quantum models [MN07]. Pak's algorithm [Pak12] is based on the computation of random subproducts of the A_i. In its instantiation to matrix groups [Pak12, Theorem 1.5], Pak suggests as a speedup to apply Freivald's technique [Fre79] for the verification of matrix products. This can be done in the same manner for Lemma 2.1.3. We stress that Pak's algorithm applies only to groups rather than semigroups; in particular, for the application to commutativity of matrices this means that the M_i must be invertible.[2] Note that there is no such assumption in Lemma 2.1.3; compared to [Pak12] we therefore obtain a randomized algorithm for testing *matrix semigroup commutativity*. We also note that the idea of testing commutativity on random linear combinations is akin to the general technique for the verification of identities in [RS00]. However, in the case of commutativity testing that chapter does not obtain any improvement over the trivial deterministic algorithm

[2] Pak's result is definitely stated only for groups, and it appears that its correctness proof actually uses the invertibility hypothesis.

(see Theorem 3.1 in [RS00]). In order to analyze the higher order case $d > 3$, we will derive an appropriate generalization of Lemma 2.1.3 (to families of matrices satisfying certain symmetry properties).

2.1.2 Organization of this chapter

In Section 2.2, we present a faster algorithm for equivalence to sum of cubes. We give a detailed complexity analysis of our algorithm in Chapter 2.6 and compare it to that of [KS21]. In Section 2.3, we extend our ideas for the degree-3 case to the arbitrary degree-d case and give an algorithm for equivalence to sum of d-th powers (Algorithm 2). In fact our algorithm can test if the input polynomial is equivalent to some linear combination of d-th powers (As explained in Section 1.4.5, these notions are different over $\mathbb{R}$ when d is even). In Chapter 2.7.1, we give a detailed complexity analysis of Algorithm 2. In Section 2.7.2, we show that when the input polynomial has rational coefficients, Algorithm 2 runs in polynomial time in the bit model of computation, as well. In Section 2.4, we give an algorithm to check whether the input polynomial can be decomposed into a linear combination of d-th powers of at most n many linearly independent linear forms. In Section A.2, we compute the number of blackbox calls and arithmetic operations performed by this algorithm. In Section 2.5, we show how we can modify our decision algorithm to give an algorithm that actually computes the linear forms and their corresponding coefficients.

2.1.3 Notations

We work in a field $\mathbb{K}$ which may be the field of real numbers or the field of complex numbers. Some of our intermediate results (in particular, Lemma 2.1.3) apply to other fields as well. We denote by $\mathbb{K}[x_1, \ldots, x_n]_d$ the space of homogeneous polynomials of degree d in n variables with coefficients in $\mathbb{K}$. A homogeneous polynomial of degree d is also called a *degree-d form*. We denote by P_d the polynomial $\sum_{i=1}^n x_i^d$, and we say that a degree d form $f(x_1, \ldots, x_n)$ is equivalent to a sum of d-th powers if it is equivalent to P_d, i.e., if $f(x) = P_d(Ax)$ for some invertible matrix A. More generally, we denote by $\mathcal{P}_d$ the set of polynomials of the form $\sum_{i=1}^n \alpha_i x_i^d$ with $\alpha_i \neq 0$ for all i. As explained in Section 1.4.5, for $\mathbb{K} = \mathbb{R}$ we are not only interested in equivalence to P_d: we would like to know whether the input is equivalent to one of the elements of $\mathcal{P}_d$.

We denote by $M_n(\mathbb{K})$ the space of square matrices of size n with entries from $\mathbb{K}$. We denote by ω a feasible exponent for matrix multiplication, i.e., we assume that two matrices of $M_n(\mathbb{K})$ can be multiplied with $O(n^\omega)$ arithmetic operations in $\mathbb{K}$.

We denote by $M(d)$ the number of arithmetic operations required for multiplication of two polynomials of degree $\leq d$ and we will often refer to the $O(d \log d \log\log d)$ bounds given by [SS71] for polynomial multiplication to give concrete bounds for our algorithms.

Throughout the chapter, we will choose the entries r_{ij} of a matrix R independently and uniformly at random from a finite set $S \subset \mathbb{K}$. When we calculate the probability of some event E over the random choice of the r_{ij}, by abuse of notation instead of $\Pr_{r_{11},\ldots,r_{nn} \in S}[E]$ we simply write $\Pr_{R \in S}[E]$.

2.2 Faster algorithm for sums of cubes

In this section we present our fast algorithm for checking whether an input polynomial $f(x_1, \ldots, x_n)$ is equivalent to $P_3 = x_1^3 + \cdots + x_n^3$ (see Algorithm 1 below). As

explained in Section 1.5.2, this means that $f(x) = P_3(Ax)$ for some invertible matrix A. In Section 2.1.1 we saw that a degree 3 form in n variables can be viewed as an order 3 tensor, which we can be cut into n slices. All of our decomposition algorithms build on this approach.

Algorithm 1: Randomized algorithm to check equivalence to P_3

Input: A degree-3 homogeneous polynomial f
Let $R \in M_n(\mathbb{K})$ be a matrix such that its entries r_{ij} are picked uniformly and independently at random from a finite set S and set $h(x) = f(Rx)$
Let T_1, T_2, T_3 be the first 3 slices of h.
if T_1 *is singular* **then**
    reject
else
    compute $T_1' = T_1^{-1}$
    if $T_1'T_2$ *and* $T_1'T_3$ *commute and* $T_1'T_2$ *is diagonalisable over* $\mathbb{K}$ **then**
        accept
    else
        reject
    end
end

Recall from Section 2.1.1 that the equivalence algorithm from [KS21] needs to check that the n matrices $T_1'T_k$ commute and are diagonalisable, where $T_1, \ldots, T_n$ denote the slices of $h(x) = f(Rx)$. Algorithm 1 is faster because it only checks that $T_1'T_2$ and $T_1'T_3$ commute and that $T_1'T_2$ is diagonalisable. We do a detailed complexity analysis of the two algorithms in Chapter 2.6. It reveals that the cost of the diagonalisability tests dominates the cost of the commutativity tests for both algorithms. Since we have replaced n diagonalisability tests by a single test, it follows that Algorithm 1 is faster by a factor of n. More precisely, we show that the algorithm from [KS21] performs $O(n^{\omega+2})$ arithmetic operations when $\mathbb{K} = \mathbb{C}$, but Algorithm 1 performs only $O(n^{\omega+1})$ arithmetic operations.

The remainder of this section is devoted to a correctness proof for Algorithm 1, including an analysis of the probability of error. Our main result about this algorithm is as follows.

Theorem 2.2.1. *If an input $f \in \mathbb{K}[x_1, ..., x_n]_3$ is not equivalent to a sum of cubes, then f is rejected by the algorithm with high probability over the choice of the random matrix R. More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$ then the input will be rejected with probability $\geq 1 - \frac{2}{|S|}$.*

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. More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a set $S \subseteq \mathbb{K}$, then the input will be accepted with probability $\geq 1 - \frac{2n}{|S|}$.

The second part of Theorem 2.2.1 is the easier one, and it already follows from [KS21]. Indeed, the same probability of error $2n/|S|$ was already given for the randomized equivalence algorithm of [KS21], and any input accepted by that algorithm is also accepted by our faster equivalence algorithm. Nevertheless, we give a self-contained proof of this error bound in Section 2.2.2 as a preparation toward the case of higher degree.

One of the reasons why the analysis is simpler for positive inputs is that there is only one way for a polynomial to be equivalent to P_3: its slices must satisfy all

the properties of Theorem 2.2.8 (at the end of Section 2.2.1). By contrast, if a polynomial is not equivalent to P_3 this can happen in several ways depending on which property fails. We analyze failure of commutativity in Section 2.2.3 and failure of diagonalisability in Section 2.2.4. Then we tie everything together in Section 2.2.5.

2.2.1 Characterization of equivalence to P_3

Toward the proof of Theorem 2.2.1 we need some results from [KS21], which we recall in this section. We also give a complement in Theorem 2.2.7. First, let us recall how the slices of a polynomial evolve under a linear change of variables.

Theorem 2.2.2. *Let g be a degree-3 form with slices $S_1, ..., S_n$ and let $f(x) = g(Ax)$. The slices $T_1, ..., T_n$ of f are given by the formula:*

$$T_k = A^T D_k A$$

where $D_k = \sum_{i=1} a_{i,k} S_i$ and the $a_{i,k}$ are the entries of A. In particular, if $g = \sum_{i=1}^n \alpha_i x_i^3$ we have $D_k = diag\ (\alpha_1 a_{1,k}, ..., \alpha_n a_{n,k})$.

In Theorem 2.1.1 we gave a characterization of equivalence to P_3 based on simultaneous diagonalisation by congruence. This characterization follows from Theorem 2.2.2 and the next lemma. See [KS21, Section 3.2] for more details on Theorem 2.2.2, Lemma 2.2.3 and the connection to Theorem 2.1.1.

Lemma 2.2.3. *Let f be a degree 3 homogeneous polynomial such that $f(x) = P_3(Ax)$ for some non-singular A. Let $\mathcal{U}$ and $\mathcal{V}$ be the subspaces of $M_n(\mathbb{K})$ spanned by slices of f and P_3 respectively. Then the subspace $\mathcal{V}$ is the space of diagonal matrices and $\mathcal{U}$ is a non-singular subspace, i.e., it is not made of singular matrices only.*

Instead of diagonalisation by congruence, it is convenient to work with the more familiar notion of diagonalisation by similarity, where an invertible matrix A acts by $S \mapsto A^{-1}SA$ instead of $A^T SA$. We collect the necessary material in the remainder of this section (and we refer to diagonalisation by similarity simply as *diagonalisation*).

The two following properties play a fundamental role throughout this chapter.

Definition 2.2.4. *Let $\mathcal{V}$ be a non-singular space of matrices.*

- *We say that $\mathcal{V}$ satisfies the **Commutativity Property** if there exists an invertible matrix $A \in \mathcal{V}$ such that $A^{-1}\mathcal{V}$ is a commuting subspace i.e., $PQ = QP$ for any two matrices $P, Q \in A^{-1}\mathcal{V}$*
- *We say that $\mathcal{V}$ satisfies the **Diagonalisability Property** if there exists an invertible matrix $B \in \mathcal{V}$ such that all the matrices in the space $B^{-1}\mathcal{V}$ are diagonalisable.*

The next result can be found in [KS21, Section 2.2].

Theorem 2.2.5. *Let $\mathcal{V}$ be a non-singular subspace of matrices of $M_n(\mathbb{K})$. The following properties are equivalent.*

- *$\mathcal{V}$ satisfies the commutativity property.*
- *For all non-singular matrices $A \in \mathcal{V}$, $A^{-1}\mathcal{V}$ is a commuting subspace.*

Remark 2.2.6. *Let $\mathcal{V}$ be a non-singular subspace of matrices which satisfies the commutativity and diagonalisability properties. There exists an invertible matrix $B \in \mathcal{V}$ and an invertible matrix R which diagonalizes simultaneously all of $B^{-1}\mathcal{V}$ (i.e., $R^{-1}MR$ is diagonal for all $M \in B^{-1}\mathcal{V}$).*

Proof. Pick an invertible matrix $B \in \mathcal{V}$ such that $\mathcal{W} = B^{-1}\mathcal{V}$ is a space of diagonalizable matrices. By Theorem 2.2.5, $\mathcal{W}$ is a commuting subspace. It is well known that a finite collection of matrices is simultaneously diagonalisable if and only if they commute, and each matrix in the collection is diagonalisable. We conclude by applying this result to a basis of $\mathcal{W}$ (any matrix R which diagonalises a basis will diagonalise all of $\mathcal{W}$). □

We now give an analogue of Theorem 2.2.5 for the diagonalisability property.

Theorem 2.2.7. *Let $\mathcal{V}$ be a non-singular subspace of matrices which satisfies the commutativity property. The following properties are equivalent:*

- *$\mathcal{V}$ satisfies the diagonalisability property.*
- *For all non-singular matrices $A \in \mathcal{V}$, the matrices in $A^{-1}\mathcal{V}$ are simultaneously diagonalisable.*

Proof. Suppose that $\mathcal{V}$ satisfies the diagonalisability property. By the previous remark, we already know that there exists *some* invertible matrix $B \in \mathcal{V}$ such that the matrices in $B^{-1}\mathcal{V}$ are simultaneously diagonalisable by an invertible matrix R. We need to establish the same property for an arbitrary invertible matrix $A \in \mathcal{V}$. For any $M \in \mathcal{V}$, $A^{-1}M = (B^{-1}A)^{-1}(B^{-1}M)$. Hence $A^{-1}M$ is diagonalised by R since this matrix diagonalises both matrices $B^{-1}A$ and $B^{-1}M$. Since R is independent of the choice of $M \in \mathcal{V}$, we have shown that the matrices in $A^{-1}\mathcal{V}$ are simultaneously diagonalisable. □

The importance of the commutativity and diagonalisability properties stems from the fact that they provide a characterization of simultaneous diagonalisation by congruence, which in turn (as we have seen in Theorem 2.1.1) provides a characterization of equivalence to P_3:

Theorem 2.2.8. *Let $A_1, ..., A_k \in M_n(\mathbb{K})$ and assume that the subspace $\mathcal{V}$ spanned by these matrices is non-singular. There are diagonal matrices Λ_i and a non-singular matrix $R \in M_n(\mathbb{K})$ such that $A_i = R\Lambda_i R^T$ for all $i \in [k]$ if and only if $\mathcal{V}$ satisfies the Commutativity property and the Diagonalisability property.*

For a proof, see [KS21, Section 2.2] for $\mathbb{K} = \mathbb{C}$ and [KS21, Section 2.3] for $\mathbb{K} = \mathbb{R}$.

2.2.2 Analysis for positive inputs

In this section we analyze the behavior of Algorithm 1 on inputs that are equivalent to P_3. First, we recall the Schwartz-Zippel Lemma which we will be using throughout this chapter.

Lemma 2.2.9 ([DL78][Zip79][Sch80])**.** *Let $P \in \mathbb{K}[x_1, ..., x_n]$ be a non-zero polynomial of total degree $d \geq 0$ over a field $\mathbb{K}$. Let S be a finite subset of $\mathbb{K}$ and let $r_1, ..., r_n$ be picked uniformly and independently at random from a finite set S. Then*

$$Pr_{r_1,...,r_n \in S}[P(r_1, ..., r_n) = 0] \leq \frac{d}{|S|}.$$

Lemma 2.2.10. *Let f be a degree-3 form with slices $S_1, ..., S_n$ such that the subspace $\mathcal{V}$ spanned by the slices is non-singular. Let $h(x) = f(Rx)$ where the entries $r_{i,j}$ are*

chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$. *Let* $T_1, ..., T_n$ *be the slices of* h. *Then*

$$Pr_{R \in S}[T_1 \text{ is invertible}] \geq 1 - \frac{2n}{|S|}.$$

Proof. We can obtain the slices T_k of h from the slices S_k of f using Theorem 2.2.2 namely, we have $T_k = R^T D_k R$ where $D_k = \sum_{i \in [n]} r_{i,k} S_i$ and the $r_{i,k}$ are the entries of R.
Therefore T_1 is invertible iff R and D_1 are invertible. Applying the Lemma 2.2.9 to $\det(R)$ shows that R is singular with probability at most $n/|S|$. We will see that D_1 is singular also with probability at most $n/|S|$; the lemma then follows from the union bound. Matrix D_1 is not invertible iff $\det(D_1) = 0$. Since $D_1 = \sum_{i \in [n]} r_{i,1} S_i$ $\det(D_1) \in \mathbb{K}[r_{1,1}, ..., r_{n,1}]$ and $\deg(\det(D_1)) \leq n$. Since, $\mathcal{V}$ is non-singular, there exists some choice of $\alpha = (\alpha_1, ..., \alpha_n)$, such that $S = \sum_{i \in [n]} \alpha_i S_i$ is invertible. Hence $\det(D_1)$ is not identically zero, and it follows again from Lemma 2.2.9 that this polynomial vanishes with probability at most $\frac{n}{|S|}$. □

Lemma 2.2.11. *Given* $A \in M_n(\mathbb{K})$, *let* $T_1, ..., T_n$ *be the slices of* $h(x) = P_3(Ax)$. *If* T_1 *is invertible, define* $T_1' = (T_1)^{-1}$. *Then* $T_1' T_2$ *commutes with* $T_1' T_3$, *and* $T_1' T_2$ *is diagonalisable.*

Proof. By Theorem 2.2.2,

$$T_k = A^T \operatorname{diag}(A_{1k}, ..., A_{nk}) A = A^T D_k A.$$

If T_1 is invertible, the same is true of A and D_1. The inverse $(D_1)^{-1}$ is diagonal like D_1, hence $(D_1)^{-1} D_2$ and $(D_1)^{-1} D_3$ are both diagonal as well and must therefore commute. Now,

$$\begin{aligned} T_1' T_2 T_1' T_3 &= A^{-1}((D_1)^{-1} D_2 (D_1)^{-1} D_3) A \\ &= A^{-1}((D_1)^{-1} D_3 (D_1)^{-1} D_2) A \\ &= T_1' T_3 T_1' T_2. \end{aligned}$$

Finally, $T_1' T_2 = A^{-1}((D_1)^{-1} D_2) A$ so this matrix diagonalisable. □

In the above lemma we have essentially reproved the easier half of Theorem 2.2.8. We are now in position to prove the easier half of Theorem 2.2.1.

Proposition 2.2.12. *If an input* $f \in \mathbb{K}[x_1, ..., x_n]_3$ *is equivalent to a sum of* n *cubes then* f *will be accepted by Algorithm 1 with high probability over the choice of the random matrix* R. *More precisely, if the entries* $r_{i,j}$ *are chosen uniformly and independently at random from a set* $S \subseteq \mathbb{K}$, *then* f *will be accepted with probability at least* $1 - \frac{2n}{|S|}$.

Proof. Suppose that $f(x) = P_3(Bx)$ for some invertible matrix B. By Lemma 2.2.3, the space spanned by the slices of f is nonsingular. We can therefore apply Lemma 2.2.10: the first slice T_1 of $h(x) = f(Rx)$ is invertible with probability at least $1 - \frac{2n}{|S|}$. Moreover, when T_1 is invertible Lemma 2.2.11 shows that f will always be accepted (we can apply this lemma to h since $h(x) = P_3(BRx)$). □

2.2.3 Failure of commutativity

In this section we first give the proof of Lemma 2.1.3. This is required for the analysis of Algorithm 1, and moreover this simple lemma yields a new randomized algorithm

for commutativity testing as explained in Section 1.4.5. We restate the lemma here for the reader's convenience:

Lemma 2.2.13. *Let $A_1, ..., A_k \in M_n(\mathbb{K})$. We take two random linear combinations $A_\alpha = \sum_{i\in[k]} \alpha_i A_i$ and $A_\beta = \sum_{i\in[k]} \beta_i A_i$, where the α_i and β_i are picked independently and uniformly at random from a finite set $S \subset \mathbb{K}$. If $\{A_i\}_{i\in[k]}$ is not a commuting family, then the two matrices A_α, A_β commute with probability at most $\frac{2}{|S|}$.*

Proof. We want to bound the probability of error, i.e., $\Pr_{\alpha,\beta}[A_\alpha, A_\beta \text{ commute}]$. Let us define

$$\begin{aligned} P_{\text{comm}}(\alpha,\beta) &= A_\alpha A_\beta - A_\beta A_\alpha \\ &= \sum_{i,j\in[k]} \alpha_i\beta_j(A_iA_j - A_jA_i). \end{aligned}$$

By construction, A_α commutes with A_β if and only if $P_{\text{comm}}(\alpha,\beta) = 0$. Since $\{A_i\}_{i\in[k]}$ is not a commuting family, there exists $i, j \in [n]$ such that $A_iA_j - A_jA_i \neq 0$. Hence there exists some entry (r, s) such that

$$(A_iA_j - A_jA_i)_{r,s} \neq 0 \tag{2.1}$$

Let us define $P^{r,s}_{\text{comm}}(\alpha,\beta) = (A_\alpha A_\beta - A_\beta A_\alpha)_{r,s}$. From (2.1) we have

$$P^{r,s}_{\text{comm}}(e_i, e_j) \neq 0$$

where e_i is the vector with a 1 at the i-th position and 0's elsewhere. In particular, $P^{r,s}_{\text{comm}}$ is not identically zero. Since $\deg(P^{r,s}_{\text{comm}}) \leq 2$, it follows from the Lemma 2.2.9 that

$$\Pr_{\alpha,\beta\in S}[P^{r,s}_{\text{comm}}(\alpha,\beta) = 0] \leq \frac{2}{|S|}$$

and the same upper bound applies to $\Pr_{\alpha,\beta\in S}[P_{\text{comm}}(\alpha,\beta) = 0]$. □

The next result relies on the above lemma. Theorem 2.2.14 gives us a way to analyze the case when the slices of the input polynomial fail to satisfy the commutativity property (recall that this property is relevant due to Theorem 2.2.8):

Theorem 2.2.14. *Let $f \in \mathbb{K}[x_1, ..., x_n]_3$ be a degree 3 form such that the subspace $\mathcal{V}$ spanned by its n slices is non-singular and does not satisfy the commutativity property. Let $h(x) = f(Rx)$ where the entries $r_{i,j}$ of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$.Let $T_1, ..., T_n$ be the slices of h. If T_1 is invertible, define $T_1' = T_1^{-1}$. Then*

$$Pr[T_1 \text{ is invertible and } T_1'T_2, T_1'T_3 \text{ commute}] \leq \frac{2}{|S|}.$$

Proof. By Theorem 2.2.2 we know that $T_k = R^T(\sum_{i=1}^n r_{i,k}S_i)R$ where $S_1, \ldots, S_n$ are the slices of f. Let us define $D_1 = \sum_{i=1}^n r_{i,1}S_i$. Then we have:

$$T_1'T_2 = R^{-1}(D_1)^{-1}R^{-T}R^T(\sum_{i=1}^n r_{i,2}S_i)R$$
$$= R^{-1}(\sum_{i=1}^n r_{i,2}D_1^{-1}S_i)R.$$

Similarly, $T_1'T_3 = R^{-1}(\sum_{i=1}^n r_{i,3}D_1^{-1}S_i)R$. So $T_1'T_2$ commutes with $T_1'T_3$ iff R is invertible and $\sum_{i=1}^n r_{i,2}D_1^{-1}S_i$ commutes with $\sum_{i=1}^n r_{i,3}D_1^{-1}S_i$. Let E_1 be the event that $T_1'T_2$ commutes with $T_1'T_3$, and let E_1' be the event that $\sum_{i=1}^n r_{i,2}D_1^{-1}S_i$ commutes with $\sum_{i=1}^n r_{i,3}D_1^{-1}S_i$. Let E_2 be the event that $\{(D_1)^{-1}S_i\}_{i\in[n]}$ is not a commuting family. Since $\mathcal{V}$ does not satisfy the commutativity property, $(D_1)^{-1}\mathcal{V}$ is not a commuting subspace if D_1 is invertible. Hence the event that D_1 is invertible is the same as E_2. Setting $A_i = (D_1)^{-1}S_i$, $\alpha_i = r_{i,2}$, $\beta_i = r_{i,3}$ in Lemma 2.1.3 we obtain

$$\Pr_{R\in S}\left[E_1' \middle| E_2\right] \leq \frac{2}{|S|}.$$

Note here that D_1 depends only on the random variables $r_{i,1}$ for all $i \in [n]$ and therefore is independent of $r_{k,2}$ and $r_{l,3}$ for all $k, l \in [n]$, because we assume that the entries of R are all picked uniformly and independently at random.

Now we know that T_1 is invertible iff R and D_1 are invertible. Let E_3 be the event that T_1 is invertible, and E_4 the event that R is invertible. We have $E_3 = E_2 \cap E_4$, and we have seen that $E_1 = E_1' \cap E_4$. The probability of error can finally be bounded as follows:

$$\Pr_{R\in S}[E_1 \cap E_3] = \Pr_{R\in S}[E_1' \cap E_2 \cap E_4] \leq \Pr_{R\in S}[E_1'|E_2] \leq 2/|S|.$$

□

2.2.4 Failure of diagonalisability

Theorem 2.2.14 gives us a way to analyze the case when the slices of the input polynomial fail to satisfy the commutativity property. With the results in the present section we will be able to analyze the case where the commutativity property is satisfied, but the diagonalisability property fails (recall that these properties are relevant due to Theorem 2.2.8).

Proposition 2.2.15. *Let $\mathcal{U} \subseteq M_n(\mathbb{K})$ be a commuting subspace of matrices. We define*

$$\mathcal{M} := \left\{ M \middle| M \text{ is diagonalisable and } M \in \mathcal{U} \right\}.$$

Then $\mathcal{M}$ is a linear subspace of $\mathcal{U}$. In particular, if there exists $A \in \mathcal{U}$ such that A is not diagonalisable then $\mathcal{M}$ is a proper linear subspace of $\mathcal{U}$.

Proof. $\mathcal{M}$ is trivially closed under multiplication by scalars. Let $M, N \in \mathcal{M}$. These two matrices are diagonalisable by definition of $\mathcal{M}$, and they commute since $\mathcal{M} \subseteq \mathcal{U}$. Hence they are simultaneously diagonalisable. Thus $\mathcal{M}$ is closed under addition as well, which implies that it is a linear subspace of $\mathcal{U}$. □

Corollary 2.2.15.1. *Let $\{A_i\}_{i\in[n]}$ be a commuting family of matrices such that A_i is not diagonalisable for at least one index $i \in [n]$. Let $S \subset \mathbb{K}^n$ be a finite set. Then $D = \sum_i \alpha_i A_i$ is diagonalisable with probability at most $1/|S|$ when $\alpha_1, ..., \alpha_n$ are chosen uniformly and independently at random from S.*

Proof. We define $\mathcal{U} = \text{span}\{A_1, ..., A_n\}$ and

$$\mathcal{M} := \left\{ M \middle| M \text{ is diagonalisable and } M \in \mathcal{U} \right\}.$$

So the probability of error is $\text{Pr}_{\bar{\alpha}\in S}\left[D \in \mathcal{M}\right]$. By Proposition 2.2.15 and the hypothesis that there exists $A_i \in \mathcal{U} \setminus \mathcal{M}$, $\mathcal{M}$ is a proper linear subspace of $\mathcal{U}$. So $\mathcal{M}$ is an intersection of hyperplanes. Since $A_i \notin \mathcal{M}$, there exists a linear form $l_{\mathcal{M}}(X)$ corresponding to a hyperplane such that $l_{\mathcal{M}}(M) = 0$ for all $M \in \mathcal{M}$ and $l_{\mathcal{M}}(A_i) \neq 0$. This gives us that $l_{\mathcal{M}} \not\equiv 0$. We know that if D is diagonalisable then $l_{\mathcal{M}}(D) = 0$. By the Lemma 2.2.9 the probability of error satisfies:

$$\text{Pr}_{\bar{\alpha}\in S}\left[D \in \mathcal{M}\right] \leq \text{Pr}_{\bar{\alpha}\in S}\left[l_{\mathcal{M}}(D) = 0\right] \leq \frac{1}{|S|}$$

since $\deg(l_{\mathcal{M}}) = 1$. □

The last result of this section is an analogue of Theorem 2.2.14 for the diagonalisability property.

Theorem 2.2.16. *Let $f \in \mathbb{K}[x_1, ..., x_n]_3$ be a degree 3 form such that the subspace $\mathcal{V}$ spanned by its n slices is non-singular, satisfies the commutativity property but does not satisfy the diagonalisability property. Let $h(x) = f(Rx)$ where the entries $r_{i,j}$ of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$. Let $T_1, ..., T_n$ be the slices of h. If T_1 is invertible, define $T_1' = T_1^{-1}$. Then*

$$Pr_{R\in S}[T_1 \text{ is invertible and } T_1'T_2 \text{ is diagonalisable}] \leq \frac{1}{|S|}.$$

Proof. As in the proof of Theorem 2.2.14 we have

$$T_1'T_2 = R^{-1}(\sum_{i=1}^{n} r_{i,2} D_1^{-1} S_i)R$$

where $D_1 = \sum_{i=1}^n r_{i,1}S_i$. So $T_1'T_2$ is diagonalisable iff R is invertible and $M = \sum_{j\in[n]} r_{j,2}D_1^{-1}S_j$ is diagonalisable. We denote by E_1 be the event that $T_1'T_2$ is diagonalisable, and by E_1' the event that M is diagonalisable.

Let E_2 be the event that $\{(D_1)^{-1}S_i\}_{i\in[n]}$ is a commuting family, but there exists $i \in [n]$ such that $(D_1)^{-1}S_i$ is not diagonalisable. Since $\mathcal{V}$ satisfies the commutativity property and does not satisfy the diagonalisability property, by Theorem 2.2.7 the event that D_1 is invertible is the same event as E_2.

Setting $A_i = (D_1)^{-1}S_i$ and $\alpha_i = r_{i,2}$ in Corollary 2.2.15.1, we obtain

$$\text{Pr}_{R\in S}\left[E_1' \middle| E_2\right] \leq \frac{1}{|S|}.$$

Note here that D_1 depends only on the random variables $r_{i,1}$ for all $i \in [n]$ and therefore is independent of $r_{k,2}$ for all $k \in [n]$, because we assume that the entries of R are all picked uniformly and independently at random.

Now we know that T_1 is invertible iff R and D_1 is invertible. Let E_3 be the event that T_1 is invertible, and E_4 the event that R is invertible. We have $E_3 = E_2 \cap E_4$, and we have seen that $E_1 = E_1' \cap E_4$. The probability of error can finally be bounded as follows:

$$\Pr_{R\in S}[E_1 \cap E_3] = \Pr_{R\in S}[E_1' \cap E_2 \cap E_4] \leq \Pr_{R\in S}[E_1'|E_2] \leq \frac{1}{|S|}.$$

□

2.2.5 Analysis for negative inputs

In this section we complete the proof of Theorem 2.2.1. The case of positive inputs was treated in Section 2.2.2. It therefore remains to prove the following result.

Theorem 2.2.17. *If an input $f \in \mathbb{K}[x_1, ..., x_n]_3$ is not equivalent to a sum of cubes, then f is rejected by Algorithm 1 with high probability over the choice of the random matrix R. More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$ then the input will be rejected with probability at least $1 - \frac{2}{|S|}$.*

Proof. Let $S_1, ..., S_n$ be the slices of f and $\mathcal{V} = \text{span}\{S_1, ..., S_n\}$. From Theorem 2.1.1 and Theorem 2.2.8, we know that if $f \not\sim P_3$ there are three disjoint cases to consider:

(i) $\mathcal{V}$ is a singular subspace of matrices.

(ii) $\mathcal{V}$ is a non-singular subspace and does not satisfy the commutativity property.

(iii) $\mathcal{V}$ is a non-singular subspace, satisfies the commutativity property but does not satisfy the diagonalisability property.

We will upper bound the probability of error in each case. In case (i), $T_1 = \sum_{j\in[n]} r_{1,j} S_j \in \mathcal{V}$ is always singular for any choice of the $r_{1,j}$. So f is rejected by the algorithm with probability 1 in this case. In case (ii) we can upper bound the probability of error as follows:

$$\begin{aligned}
&\Pr_{R\in S}[f \text{ is accepted by the algorithm}] \\
&= \Pr_{R\in S}[T_1 \text{ is invertible, } T_1'T_2, T_1'T_3 \text{ commute, } T_1'T_2 \text{ is diagonalisable}] \\
&\leq \Pr_{R\in S}[T_1 \text{ is invertible, } T_1'T_2, T_1'T_3 \text{ commute}].
\end{aligned}$$

By Theorem 2.2.14, this occurs with probability $2/|S|$ at most. In case (iii) we have the following bound on the probability of error:

$$\begin{aligned}
&\Pr_{R\in S}[f \text{ is accepted by the algorithm}] \\
&= \Pr_{R\in S}[T_1 \text{ is invertible, } T_1'T_2, T_1'T_3 \text{ commute, } T_1'T_2 \text{ is diagonalisable}] \\
&\leq \Pr_{R\in S}[T_1 \text{ is invertible, } T_1'T_2 \text{ is diagonalisable}].
\end{aligned}$$

By Theorem 2.2.16 this occurs with probability $1/|S|$ at most. Therefore, in all three cases the algorithm rejects f with probability at least $1 - \frac{2}{|S|}$. □

2.3 Equivalence to a linear combination of d-th powers

We can associate to a symmetric tensor T of order d the homogeneous polynomial

$$f(x_1, ..., x_n) = \sum_{i_1,...,i_d \in [n]} T_{i_1...i_d} x_{i_1} ... x_{i_d}.$$

This correspondence is bijective, and the symmetric tensor associated to a homogeneous polynomial f can be obtained from the relation

$$T_{i_1...i_d} = \frac{1}{d!} \frac{\partial^d f}{\partial x_{i_1} ... \partial x_{i_d}}.$$

The $(i_1, ..., i_{d-2})$-th slice of T is the symmetric matrix $T_{i_1...i_{d-2}}$ with entries $(T_{i_1...i_{d-2}})_{i_{d-1},i_d} = T_{i_1...i_d}$

2.3.1 The Algorithm

Recall from Section 2.1.3, we denote by $\mathcal{P}_d$, the set of polynomials of the form $\sum_{i=1}^n \alpha_i x_i^d$ with $\alpha_i \neq 0$ for all $i \in [n]$. In this section we present a poly-time algorithm for checking whether an input degree d form in n variables f is equivalent to some polynomial in $\mathcal{P}_d$(see Algorithm 2 below). This means that $f(x) = P_d(Ax)$ for some $P_d \in \mathcal{P}_d$ such that A is invertible.

Recall from Section 2.2, that the equivalence algorithm for sum of cubes needs to check if $T_1'T_2$ commutes with $T_1'T_3$ and if $T_1'T_2$ is diagonalisable, where $T_1, ..., T_n$ are the slices of $h(x) = f(Rx)$. Now, we prove a surprising fact that even for the higher degree cases, checking commutativity of 2 matrices and the diagonalisability of 1 matrix is enough to check equivalence to sum of linear combination of d-th powers.

Recall the definition of the permanent polynomial from (1.10). Interestingly though, arbitrary slices of a degree-d polynomial f are as hard to compute as the permanent polynomial even if f has a small arithmetic circuit. This follows from the following observation in [Val79] that the coefficient of the monomial $y_1...y_n$ in the polynomial $\prod_{i=1}^n \sum_{j=1}^n x_{ij} y_j$ is the permanent of the $n \times n$ matrix $X = (x_{ij})$.

Let $\{T_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2} \in [n]}$ be the slices of $h(x) = f(Rx)$. We denote by $T_{\bar{i}}$, the corresponding slice $T_{i...i}$. Algorithm 2 checks if $T_{\bar{1}}'T_{\bar{2}}$ commutes with $T_{\bar{1}}'T_{\bar{3}}$ and if $T_{\bar{1}}'T_{\bar{3}}$ is diagonalisable. These particular slices are special because they can be computed using small number of calls to the blackbox and in small number of arithmetic operations (due to the fact that they are essentially repeated partial derivatives with respect to a single variable) and hence, help us give a polynomial time algorithm. More precisely, we show that if the polynomial is given as a blackbox, the algorithm requires only $O(n^2 d)$ calls to the blackbox and $O(n^2 M(d) \log d + n^{\omega+1})$ many arithmetic operations. We do a detailed complexity analysis of this algorithm in Appendix 2.7.

Algorithm 2: Randomized algorithm to check polynomial equivalence to $\mathcal{P}_d$

Input: A degree-d homogeneous polynomial f
Let $R \in M_n(\mathbb{K})$ be a matrix such that its entries r_{ij} are picked uniformly and independently at random from a finite set S and set $h(x) = f(Rx)$.
Let $\{T_{i_1 \dots i_{d-2}}\}_{i_1 \dots i_{d-2} \in [n]}$ be the slices of h.
We compute the slices $T_{\bar{1}}$,$T_{\bar{2}}$,$T_{\bar{3}}$.
if $T_{\bar{1}}$ *is singular* **then**
    reject
else
    compute $T'_{\bar{1}} = (T_{\bar{1}})^{-1}$
    if $T'_{\bar{1}}T_{\bar{2}}$ *and* $T'_{\bar{1}}T_{\bar{3}}$ *commute and* $T'_{\bar{1}}T_{\bar{2}}$ *is diagonalisable over* $\mathbb{K}$ **then**
        accept
    else
        reject
    end
end

The remainder of this section is devoted to a correctness proof for Algorithm 2, including an analysis of the probability of error. Our main result about this algorithm is as follows:

Theorem 2.3.1. *If an input $f \in \mathbb{F}[x_1, ..., x_n]_d$ is not equivalent to some polynomial $P_d \in \mathcal{P}_d$, then f is rejected by the algorithm with high probability over the choice of the random matrix R. More precisely, if the entries $r_{i,j}$ of R are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be rejected with probability $\geq (1 - \frac{2(d-2)}{|S|})$.*
Conversely, if f is equivalent to some polynomial $P_d \in \mathcal{P}_d$, then f will be accepted with high probability over the choice of the random matrix R. More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be accepted with probability $\geq (1 - \frac{n(d-1)}{|S|})$.

The proof structure of this theorem follows the one of Theorem 2.2.1. In Section 2.3.3, we give a proof of the second part of theorem i.e. the behavior of Algorithm 2 on the positive inputs. Here we require a stronger property of the subspace spanned by the slices of these positive inputs. For this we define the notion of "weak singularity" in Section 2.3.2 and prove an equivalence result related to it. On the negative inputs i.e if a polynomial is not equivalent to some polynomial in $\mathcal{P}_d$, this can again happen in several ways depending on which property fails. We analyze the failure of commutativity in Section 2.3.4 and failure of diagonalisability in Section 2.3.4. Then we collect everything together and prove the first part of the theorem in Section 2.3.4.

2.3.2 Characterisation of equivalence to $\mathcal{P}_d$

First, we show how the slices of a degree-d form evolve under a linear change of variables. This result is an extension of Theorem 2.2.2 to the higher degree case.

Theorem 2.3.2. *Let g be a degree-d form with slices $\{S_{i_1 \dots i_{d-2}}\}_{i_1, \dots, i_{d-2} \in [n]}$ and let $f(x) = g(Ax)$. Then the slices $T_{i_1 \dots i_{d-2}}$ of f, are given by $T_{i_1 \dots i_{d-2}} = A^T D_{i_1 \dots i_{d-2}} A$ where $D_{i_1 \dots i_{d-2}} = \sum_{j_1 \dots j_{d-2} \in [n]} a_{j_1 i_1} \dots a_{j_{d-2} i_{d-2}} S_{j_1 \dots j_{d-2}}$ and $a_{i,j}$ are the entries of A. If $g = \sum_{i=1}^n \alpha_i x_i^d$, we have $D_{i_1 \dots i_{d-2}} = diag(\alpha_1(\prod_{m=1}^{d-2} a_{1,i_m}), ..., \alpha_n(\prod_{m=1}^{d-2} a_{n,i_m}))$.*

Proof. By definition of the slices of a polynomial,

$$S_{i_1\dots i_{d-2}} = \frac{1}{d!} H_{\frac{\partial^{d-2} g}{\partial x_{i_1}\dots\partial x_{i_{d-2}}}}(x) \text{ and } T_{i_1\dots i_{d-2}} = \frac{1}{d!} H_{\frac{\partial^{d-2} f}{\partial x_{i_1}\dots\partial x_{i_{d-2}}}}(x)$$

where $H_f(x)$ is the Hessian matrix of f at point x. Since $f(x) = g(Ax)$, by differentiating d times, we get that $\frac{\partial^d f}{\partial x_{i_1}\dots\partial x_{i_d}}(x) = \sum_{j_1\dots j_d\in[n]} a_{j_1 i_1}\dots a_{j_d i_d} \frac{\partial^d g}{\partial x_{j_1}\dots\partial x_{j_d}}(Ax)$. Putting these equations in matrix form, and using the fact that $\frac{\partial^d g}{\partial x_{j_1}\dots\partial x_{j_d}}(Ax) = \frac{\partial^d g}{\partial x_{j_1}\dots\partial x_{j_d}}(x)$ we get the desired result. □

The next lemma uses Theorem 2.3.2 to reveal some crucial properties about the subspace spanned by the slices of any degree-d form which is equivalent to some $g \in \mathcal{P}_d$. It is an extension of Lemma 2.2.3 to the higher degree case.

Lemma 2.3.3. *Let $f(x_1, \dots, x_n)$ and $g(x_1, \dots, x_n)$ be two forms of degree d such that $f(x) = g(Ax)$ for some non-singular matrix A.*

1. *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 $\mathcal{U} = A^T\mathcal{V}A$.*
2. *$\mathcal{V}$ is non-singular iff $\mathcal{U}$ is non-singular.*
3. *In particular, for $g \in \mathcal{P}_d$ the subspace $\mathcal{V}$ is the space of diagonal matrices and $\mathcal{U}$ is a non-singular subspace, i.e., it is not made of singular matrices only.*

Proof. Theorem 2.3.2 shows that $\mathcal{U} \subseteq A^T\mathcal{V}A$. Now since, $g(x) = f(A^{-1}x)$, same argument shows that $\mathcal{V} \subseteq A^{-T}\mathcal{U}A^{-1}$. This gives us that $\mathcal{U} = A^T\mathcal{V}A$.

For the second part of the lemma, let us assume that $\mathcal{V}$ is non-singular and $M_{\mathcal{U}}$ be an arbitrary matrix in $\mathcal{U}$. Using the previous part of the lemma, we know that there exists $M_{\mathcal{V}} \in \mathcal{V}$ such that $M_{\mathcal{U}} = A^T M_{\mathcal{V}} A$. Since $\mathcal{V}$ is non-singular, $\det(M_{\mathcal{V}}) \neq 0$. Taking determinant on both sides, we get that $\det(M_{\mathcal{U}}) = \det(A)^2\det(M_{\mathcal{V}}) \neq 0$ (since A is invertible, $\det(A) \neq 0$). For the converse, assume that $\mathcal{U}$ is non-singular. Following a similar proof, it can be shown that $\det(M_{\mathcal{U}}) \neq 0$.

For the third part of the lemma, let $\{S_{i_1\dots i_{d-2}}\}_{i_1\dots i_{d-2}\in[n]}$ be the slices of g. If $g = \sum_{i\in[n]} \alpha_i x_i^d$, such that $\alpha_i \neq 0$ for all i, S_i has α_i in the (i,i)-th position and 0 everywhere. Also, $S_{i_1,\dots,i_{d-2}} = 0$, when the i_k's are not equal. Hence, $\mathcal{V}$ is the space of all diagonal matrices. Hence $\mathcal{V}$ is a non-singular space. Using the previous part of the lemma, we get that $\mathcal{U}$ is a non-singular space as well.

□

The next lemma is effectively a converse of the second part of Lemma 2.3.3. It shows that if the slices of f are diagonal matrices, then the fact that they effectively originate from a symmetric tensor forces them to be extremely special.

Lemma 2.3.4. *Let $f \in \mathbb{K}[x_1, \dots, x_n]_d$ be a degree-d form. If the slices of f are diagonal matrices, then $f = \sum_{i\in[n]} \alpha_i x_i^d$ for some $\alpha_1, \dots, \alpha_n \in \mathbb{K}$.*

Proof. Let $T_{i_1,\dots,i_{d-2}}$ be the slices of f. Let $I = \{(i_{\sigma(1)}, \dots, i_{\sigma(d)}) | \sigma \in S_d\}$. Now since they are slices of a polynomial, we know that

$$(T_{i_1\dots i_{d-2}})_{i_{d-1},i_d} = (T_{i_{\sigma(1)},\dots,i_{\sigma(d-2)}})_{i_{\sigma(d-1)},i_{\sigma(d)}}. \tag{2.2}$$

We want to show that $T_{i_1,\dots,i_d} \neq 0$ only if $i_1 = i_2 = \dots = i_d$. Using (2.2), it is sufficient to show that $(T_{i_1,\dots,i_{d-2}})_{i_{d-1},i_d} \neq 0$ only if $i_{d-1} = i_d$. This is true since $T_{i_1,\dots,i_{d-2}}$ are diagonal matrices. This gives us that $f = \sum_{i\in[n]} \alpha_i x_i^d$. □

Now we are finally ready to prove a theorem that characterizes exactly the set of degree-d homogeneous polynomials which are equivalent to some $g \in \mathcal{P}_d$. This is an extension of Theorem 2.1.1 to the degree-d case, and it already appears as Theorem 2.1.2 in the introduction. We restate it now for the reader's convenience.

Theorem 2.3.5. *A degree d form $f \in \mathbb{K}[x_1, ..., x_n]$ is equivalent to some polynomial $P_d \in \mathcal{P}_d$ if and only if its slices $\{T_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$ span a non-singular matrix space and the slices are simultaneously diagonalisable by congruence, i.e., there exists an invertible matrix $Q \in M_n(\mathbb{K})$ such that the matrices $Q^T T_{i_1...i_{d-2}} Q$ are diagonal for all $i_1, ..., i_{d-2} \in [n]$.*

Proof. Let $\mathcal{U}$ be the space spanned by $\{T_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$. If f is equivalent to P_d, Theorem 2.3.2 shows that the slices of f are simultaneously diagonalisable by congruence and Lemma 2.3.3 shows that $\mathcal{U}$ is non-singular.

Let us show the converse. Since the slices $\{T_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in [n]}$ are simultaneously diagonalisable, there are diagonal matrices $\Lambda_{i_1...i_{d-2}}$ and a non-singular matrix $R \in M_n(\mathbb{K})$ such that $T_{i_1...i_{d-2}} = R\Lambda_{i_1...i_{d-2}}R^T$ for all $i_1, ..., i_{d-2} \in [n]$. So now we consider $g(x) = f(R^{-T}x)$. Let $\{S_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$ be the slices of g. Using Theorem 2.3.2, we get that

$$\begin{aligned} S_{i_1...i_{d-2}} &= (R^{-1})(\sum_{j_1...j_{d-2}\in[n]} r_{j_1 i_1}...r_{j_{d-2}i_{d-2}} R\Lambda_{j_1...j_{d-2}}R^T)R^{-T} \\ &= \sum_{j_1...j_{d-2}\in[n]} r_{j_1 i_1}...r_{j_{d-2}i_{d-2}}\Lambda_{j_1...j_{d-2}}. \end{aligned}$$

This implies that $S_{j_1...j_{d-2}}$ are also diagonal matrices. By Lemma 2.3.4, $g = \sum_{i\in[n]} \alpha_i x_i^d$. It therefore remains to be shown that $\alpha_i \neq 0$, for all $i \in [n]$. Let $\mathcal{V}$ be the subspace spanned by the slices of g and the slices of f span a non-singular matrix space $\mathcal{U}$. Since, $\mathcal{U}$ is a non-singular subspace of matrices, using part (2) of Lemma 2.3.3, we get that $\mathcal{V}$ is a non-singular subspace of matrices.

But if some α_i vanishes, for all $A \in \mathcal{V}$, $A_{\bar{i}} = 0$. Hence $\mathcal{V}$ is a singular subspace, which is a contradiction. This gives us that $g = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$ for all i. Hence, $g \in \mathcal{P}_d$ and f is equivalent to g. □

Theorem 2.3.6. *Let $f \in \mathbb{K}[x_1, ..., x_n]$ be a degree-d form. f is equivalent to some polynomial $P_d \in \mathcal{P}_d$ iff the subspace $\mathcal{V}$ spanned by its slices $\{T_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$ is a non-singular subspace and $\mathcal{V}$ satisfies the Commutativity Property and the Diagonalisability Property.*

Proof. This follows from Theorem 2.3.5 and Theorem 2.2.8 for $k = n^{d-2}$ to get the result. □

We now introduce a weaker notion of singularity of a subspace spanned by a set of matrices and using that we prove a stronger version of Theorem 2.3.6. More formally we show that the characterization is valid even when the "non-singular subspace" criterion imposed on the subspace $\mathcal{V}$ spanned by the slices of the polynomial is replaced by the "not a weakly singular subspace" criterion.

Definition 2.3.7. *(Weak singularity)*
Let $\mathcal{V}$ be the space spanned by matrices $\{S_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$. $\mathcal{V}$ is weakly singular if for all $\alpha = (\alpha_1, ..., \alpha_n)$,

$$det(\sum_{i_1,...,i_{d-2}\in[n]} (\prod_{k\in[d-2]} \alpha_{i_k}) S_{i_1...i_{d-2}}) = 0.$$

Notice here that the notion of weak-singularity is entirely dependent on the generating set of matrices. So it is more of a property of the generating set. But by abuse of language, we will call the span of the matrices to be weakly singular. To put it in contrast, refer to Section 2.1.1 where the notion of singularity is a property of the subspace spanned by the matrices (irrespective of the generating set). It can be further observed that for all $n \geq 2$ and $d \geq 4$, non-singular families of matrices can be easily constructed which are weakly singular!

Theorem 2.3.8. *Let $f \in \mathbb{K}[x_1, ..., x_n]$ be a degree-d form. f is equivalent to some polynomial $P_d \in \mathcal{P}_d$ iff the subspace $\mathcal{V}$ spanned by its slices $\{T_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$ is not a weakly singular subspace, satisfies the Commutativity Property and the Diagonalisability Property.*

Proof. First we show that if $f = P_d(Ax)$ such that $P_d \in \mathcal{P}_d$ i.e. $P_d(x) = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$ for all $i \in [n]$ and A is invertible, then $\mathcal{V}$ is not a weakly singular subspace, satisfies the commutativity property and the diagonalisability property. Let $\{S_{i_1...i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$ be the slices of P_d. Then $S_{\bar{i}} = \alpha_i \text{diag}(e_i)$ where e_i is the i-th standard basis vector, and all other slices are 0. From Theorem 2.3.2,

$$T_{i_1...i_{d-2}} = A^T D_{i_1...i_{d-2}} A = A^T (\sum_{k\in[n]} a_{ki_1}...a_{ki_{d-2}} S_k) A.$$

Now we define

$$\begin{aligned} T(\bar{\beta}) &= \sum_{i_1,...,i_{d-2}\in[n]} (\prod_{k\in[d-2]} \beta_{i_k}) T_{i_1...i_{d-2}} \\ &= \sum_{i_1,...,i_{d-2}\in[n]} (\prod_{k\in[d-2]} \beta_{i_k}) A^T (\text{diag}(\alpha_1(\prod_{m\in[d-2]} a_{1i_m}), ..., \alpha_n(\prod_{m\in[d-2]} a_{ni_m}))) A \\ &= A^T \text{diag}(\alpha_1(\sum_{i_1,...,i_{d-2}\in[n]} (\prod_{k\in[d-2]} \beta_{i_k} a_{1i_k})), ..., \alpha_n(\sum_{i_1,...,i_{d-2}\in[n]} (\prod_{k\in[d-2]} \beta_{i_k} a_{ni_k}))) A. \end{aligned}$$

Taking determinant on both sides, $\det(T)(\bar{\beta}) = \det(A)^2 \prod_{m=1}^n T_m(\bar{\beta})$ where $T_m(\bar{\beta}) = \alpha_m(\sum_{i_1,...,i_{d-2}\in[n]}(\prod_{k\in[d-2]} \beta_{i_k} a_{mi_k}))$. Since, A is invertible, none of its rows are all 0. Hence for all $m_0 \in [n]$, there exists $j_0 \in [n]$, such that $a_{m_0 j_0} \neq 0$. Then $\text{coeff}_{\beta_{j_0}^{d-2}}(T_{m_0}) = a_{m_0 j_0}^{d-2} \neq 0$. Hence $T_{m_0} \not\equiv 0$ for all $m_0 \in [n]$ which implies that $\det(T) \not\equiv 0$. Therefore, there exists $\bar{\beta^0}$ such that $\det(T)(\bar{\beta^0}) \neq 0$. This proves that $\det(\sum_{i_1,...,i_{d-2}\in[n]}(\prod_{k\in[d-2]} \beta_{i_k}) T_{i_1...i_{d-2}}) \not\equiv 0$.

Hence, $\mathcal{V} = \text{span}\{T_{i_1...i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$ is not weakly singular. Theorem 2.3.6 gives us that the subspace spanned by the slices $\mathcal{V}$ satisfies the commutativity property and the diagonalisability property.

For the converse, if $\mathcal{V}$ is not a weakly singular subspace, then it is a non-singular subspace as well. And it satisfies the commutativity property and the diagonalisability property. By Theorem 2.3.6, we get that f is equivalent to some polynomial in $\mathcal{P}_d$. □

2.3.3 Analysis for positive inputs

In this section we analyze the behavior of Algorithm 2 on inputs that are equivalent to some polynomial in $\mathcal{P}_d$ (which we refer to as the positive inputs). We recall here again that by $T_{\bar{1}}$, we denote the slice $T_{11...1}$.

Lemma 2.3.9. *Let $f \in \mathbb{K}[x_1, ..., x_n]_d$ with slices $\{S_{i_1,...,i_{d-2}}\}_{i_1,...,i_{d-2}\in[n]}$, such that the subspace $\mathcal{V}$ spanned by the slices is not weakly singular. Let $h(x) = f(Rx)$ where*

the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$. Let $\{T_{i_1,\ldots,i_{d-2}}\}_{i_1,\ldots,i_{d-2}\in[n]}$ be the slices of h. Then

$$Pr_{R\in S}[T_{\bar{1}} \text{ is invertible}] \geq 1 - \frac{n(d-1)}{|S|}.$$

Proof. We can obtain the slices $T_{i_1\ldots i_{d-2}}$ of h from the slices $S_{i_1\ldots i_{d-2}}$ of f using Theorem 2.3.2. Namely, we have

$$T_{i_1\ldots i_{d-2}} = R^T D_{i_1\ldots i_{d-2}} R$$

where

$$D_{i_1\ldots i_{d-2}} = \sum_{j_1\ldots j_{d-2}\in[n]} \Big(\prod_{m\in[d-2]} r_{j_m,i_m}\Big) S_{j_1\ldots j_{d-2}}.$$

Therefore $T_{\bar{1}}$ is invertible iff R and $D_{\bar{1}}$ are invertible. Applying Lemma 2.2.9 to $\det(R)$ shows that R is singular with probability at most $\frac{n}{|S|}$. We will show that $D_{\bar{1}}$ is singular with probability at most $\frac{n(d-2)}{|S|}$. The lemma then follows from the union bound. Matrix $D_{\bar{1}}$ is not invertible iff $\det(D_{\bar{1}}) = 0$. Since,

$$D_{\bar{1}} = \sum_{j_1\ldots j_{d-2}\in[n]} \Big(\prod_{m\in[d-2]} r_{j_m,1}\Big) S_{j_1\ldots j_{d-2}},$$

$\det(D_{\bar{1}}) \in \mathbb{K}[r_{1,1},\ldots,r_{n,1}]$ and $\deg(\det(D_{\bar{1}})) \leq n(d-2)$. Since, $\mathcal{V}$ is not weakly singular, there exists some choice of $\alpha = (\alpha_1,\ldots,\alpha_n)$, such that

$$S = \sum_{i_1,\ldots,i_{d-2}\in[n]} \Big(\prod_{m\in[d-2]} \alpha_{i_m}\Big) S_{i_1\ldots i_{d-2}}$$

is invertible. Hence, $\det(S) \neq 0$. This gives us that $\det(D_{\bar{1}})(\alpha) \neq 0$. which gives us that $\det(D_{\bar{1}}) \not\equiv 0$. From the Lemma 2.2.9, it follows that

$$\Pr_{R\in S}[\det(D_{\bar{1}}) = 0] \leq \frac{n(d-2)}{|S|}.$$

□

Recall here from Section 2.1.3, we define by $\mathcal{P}_d$, the set of all polynomials of the form $\sum_{i=1}^n \alpha_i x_i^d$ such that $0 \neq \alpha_i \in \mathbb{K}$ for all $i \in [n]$.

Lemma 2.3.10. *Given $A \in M_n(\mathbb{K})$, let $\{T_{i_1,\ldots,i_{d-2}}\}_{i_1,\ldots,i_{d-2}\in[n]}$ be the slices of $h(x) = P_d(Ax)$ where $P_d \in \mathcal{P}_d$. If $T_{\bar{1}}$ is invertible, define $T'_{\bar{1}} = (T_{\bar{1}})^{-1}$. Then $T'_{\bar{1}}T_{\bar{2}}$ commutes with $T'_{\bar{1}}T_{\bar{3}}$ and $T'_{\bar{1}}T_{\bar{2}}$ is diagonalisable.*

Proof. Let $P_d = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$. By Theorem 2.3.2,

$$T_{i_1\ldots i_{d-2}} = A^T(\mathrm{diag}(\alpha_1(\prod_{m=1}^{d-2} a_{1,i_m}),\ldots,\alpha_n(\prod_{m=1}^{d-2} a_{n,i_m})))A = A^T D_{i_1\ldots i_{d-2}} A.$$

If $T_{\bar{1}}$ is invertible, the same is true of A and $D_{\bar{1}}$. The inverse $(D_{\bar{1}})^{-1}$ is diagonal like $D_{\bar{1}}$, hence $(D_{\bar{1}})^{-1}D_{\bar{2}}$ and $(D_{\bar{1}})^{-1}D_{\bar{3}}$ are both diagonal as well and must therefore commute. Now,

$$T'_{\bar{1}}T_{\bar{2}}T'_{\bar{1}}T_{\bar{3}} = A^{-1}((D_{\bar{1}})^{-1}D_{\bar{2}}(D_{\bar{1}})^{-1}D_{\bar{3}})A = A^{-1}((D_{\bar{1}})^{-1}D_{\bar{3}}(D_{\bar{1}})^{-1}D_{\bar{2}})A = T'_{\bar{1}}T_{\bar{3}}T'_{\bar{1}}T_{\bar{2}}.$$

Finally, $T'_{\bar{1}}T_{\bar{2}} = A^{-1}((D_{\bar{1}})^{-1}D_{\bar{2}})A$ so this matrix is diagonalisable. □

We are now in a position to prove the easier half of Theorem 2.3.1.

Theorem 2.3.11. *If an input $f \in \mathbb{K}[x_1, ..., x_n]_d$ is equivalent to some polynomial $P_d \in \mathcal{P}_d$ then f will be accepted by Algorithm 2 with high probability over the choice of the random matrix R. More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be accepted with probability $\geq (1 - \frac{n(d-1)}{|S|})$.*

Proof. We start by assuming that $f = P_d(Bx)$ for some $P_d \in \mathcal{P}_d$ where B is an invertible matrix. By Theorem 2.3.8, we know that the subspace spanned by the slices of f is not weakly singular. We can therefore apply Lemma 2.3.9, the first slice $T_{\bar{1}}$ of $h(x) = f(Rx)$ is invertible with probability at least $1 - \frac{n(d-1)}{|S|}$. Moreover if $T_{\bar{1}}$ is invertible, Lemma 2.3.10 shows that, f will always be accepted. (We can apply this lemma to h since $h = P_d(RBx)$). □

2.3.4 Analysis of negative inputs

In this section, we analyse the behaviour of Algorithm 2 on the inputs that are not equivalent to any polynomial in $\mathcal{P}_d$ (which we refer to as the negative inputs). The main goal is to show that the algorithm rejects negative inputs with high probability.

Failure of commutativity

Definition 2.3.12. *Let $\{S_{i_1,...,i_d}\}_{i_1,...,i_d \in [n]}$ be a family of matrices. We say that the matrices form a symmetric family of symmetric matrices if each matrix in the family is symmetric and for all permutations $\sigma \in S_d$, $S_{i_1,...,i_d} = S_{i_{\sigma(1)}...i_{\sigma(d)}}$.*

In the next lemma, we show that if a symmetric family of symmetric matrices (this family has size n^d) is not a commuting family, then two linear combinations of these matrices formed by picking just $2n$ elements at random also do not commute with high probability.

Lemma 2.3.13 (General commutativity lemma)**.** *Let $\{S_{i_1,...,i_d}\}_{i_1,...,i_d \in [n]}$ be a symmetric family of symmetric matrices in $M_n(\mathbb{K})$ that do not form a commuting family. Pick $\alpha = \{\alpha_1, ..., \alpha_n\}$ and $\alpha' = \{\alpha'_1, ..., \alpha'_n\}$ uniformly and independently at random from a finite set $S \subset \mathbb{K}$. We define*

$$M_\alpha = \sum_{i_1,...,i_d \in [n]} \Big(\prod_{m \in [d]} \alpha_{i_m}\Big) S_{i_1,...,i_d} \text{ and } M_{\alpha'} = \sum_{j_1,...,j_d \in [n]} \Big(\prod_{m \in [d]} \alpha'_{j_m}\Big) S_{j_1,...,j_d}.$$

Then, $Pr_{\alpha,\alpha' \in S}\Big[M_\alpha, M_{\alpha'} \text{ don't commute}\Big] \geq \Big(1 - \frac{2d}{|S|}\Big)$.

Proof. We want to bound the probability of error, i.e

$$\Pr_{\alpha,\alpha' \in S}\Big[M_\alpha M_{\alpha'} - M_{\alpha'} M_\alpha \neq 0\Big].$$

The expression $M_\alpha M_{\alpha'} - M_{\alpha'} M_\alpha$ can be written as

$$\sum_{\substack{i_1,...,i_d \in [n] \\ j_1,...,j_d \in [n]}} \Big(\prod_{m \in [d]} \alpha_{i_m} \alpha'_{j_m}\Big)(S_{i_1...i_d} S_{j_1...j_d} - S_{j_1...j_d} S_{i_1...i_d}).$$

For a fixed $r, s \in [n]$, we define the polynomial

$$P^{r,s}_{\text{comm}}(\alpha, \alpha') = \sum_{i_1,\ldots,i_d,j_1,\ldots,j_d \in [n]} \Big(\prod_{m\in[d]} \alpha_{i_m}\alpha'_{j_m}\Big) m^{r,s}_{i_1\ldots i_d j_1\ldots j_d}$$

where $m^{r,s}_{i_1\ldots i_d j_1\ldots j_d} = (S_{i_1\ldots i_d}S_{j_1\ldots j_d} - S_{j_1\ldots j_d}S_{i_1\ldots i_d})_{r,s}$.

First note that by construction M_α commutes with $M_{\alpha'}$ if and only if for all $r, s \in [n]$ such that $P^{r,s}_{\text{comm}}(\alpha, \alpha') = 0$. Since, $\{S_{i_1,\ldots,i_d}\}$ is not a commuting family, there exists $i^0_1, \ldots, i^0_d, j^0_1, \ldots, j^0_d \in [n]$, such that

$$S_{i^0_1\ldots i^0_d}S_{j^0_1\ldots j^0_d} - S_{j^0_1\ldots j^0_d}S_{i^0_1\ldots i^0_d} \neq 0.$$

Hence, there exists some entry (r_0, s_0) such that

$$(S_{i^0_1\ldots i^0_d}S_{j^0_1\ldots j^0_d} - S_{j^0_1\ldots j^0_d}S_{i^0_1\ldots i^0_d})_{r_0,s_0} \neq 0.$$

Now we claim that $P^{r_0,s_0}_{\text{comm}}(\alpha, \alpha') \not\equiv 0$. It is enough to show that the coefficient of $\alpha_{i^0_1}\ldots\alpha_{i^0_d}\alpha'_{j^0_1}\ldots\alpha'_{j^0_d}$ in $P^{r_0,s_0}_{\text{comm}}(\alpha, \alpha')$ is non-zero. Let

$$I_0 = \{(i^0_{\sigma(1)}, \ldots, i^0_{\sigma(d)}) | \sigma \in \ S_d\} \text{ and } J_0 = \{(j^0_{\sigma(1)}, \ldots, j^0_{\sigma(d)}) | \sigma \in S_d\}.$$

Then $\text{coeff}_{\alpha_{i^0_1}\ldots\alpha_{i^0_d}\alpha'_{j^0_1}\ldots\alpha'_{j^0_d}}(P^{r_0,s_0}_{\text{comm}}) = \sum_{\bar{i}\in I_0,\bar{j}\in J_0} m^{r_0 s_0}_{\bar{i}\bar{j}}$. The matrices $S_{i_1\ldots i_d}$ form a symmetric family in the sense of Definition 2.3.12. Therefore, for all $\bar{i} \in I_0, \bar{j} \in J_0$, $m^{r_0 s_0}_{\bar{i}\bar{j}}$ are equal. This gives us that

$$\text{coeff}_{\alpha_{i^0_1}\ldots\alpha_{i^0_d}\alpha'_{j^0_1}\ldots\alpha'_{j^0_d}}(P^{r_0,s_0}_{\text{comm}}) = |I_0||J_0|(m^{r_0,s_0}_{i^0_1\ldots i^0_d j^0_1\ldots j^0_d}) \neq 0.$$

Hence $P^{r_0,s_0}_{\text{comm}} \not\equiv 0$ and $\deg(P^{r_0,s_0}_{\text{comm}}) \leq 2d$ and using Lemma 2.2.9, we get that, $\Pr_{\alpha,\alpha'\in S}[P^{r_0,s_0}_{\text{comm}}(\alpha, \alpha') \neq 0] \geq 1 - \frac{2d}{|S|}$. Putting $r = r_0, s = s_0$, this gives us that

$$\Pr_{\alpha,\alpha'\in S}\Big[M_\alpha, M_{\alpha'} \text{ don't commute}\Big] \geq \Big(1 - \frac{2d}{|S|}\Big).$$

□

The next result relies on the above lemma. Theorem 2.3.14 gives us a way to analyze the case when the slices of the input polynomial fail to satisfy the commutativity property (recall that this property is relevant due to Theorem 2.3.8).

Theorem 2.3.14. *Let $f \in \mathbb{K}[x_1, \ldots, x_n]_d$ be a degree d form such that the subspace of matrices $\mathcal{V}$ spanned by its slices is not weakly singular and does not satisfy the commutativity property. Let $h(x) = f(Rx)$ where the entries $(r_{i,j})$ of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$. Let $\{T_{i_1\ldots i_{d-2}}\}_{i_1,\ldots,i_{d-2}\in[n]}$ be the slices of h. If $T_{\bar{1}}$ is invertible, define $T'_{\bar{1}} = (T_{\bar{1}})^{-1}$. Then $Pr[T_{\bar{1}}$ is invertible and $T'_{\bar{1}}T_{\bar{2}}, T'_{\bar{1}}T_{\bar{3}}$ commute $] \leq \frac{2(d-2)}{|S|}$.*

Proof. Let $\{S_{i_1\ldots i_{d-2}}\}_{i_1,\ldots,i_{d-2}\in[n]}$ be the slices of f. By Theorem 2.3.2, we know that

$$T_{i_1\ldots i_{d-2}} = R^T\Big(\sum_{j_1\ldots j_{d-2}\in[n]} \Big(\prod_{m\in[d-2]} r_{j_m,i_m}S_{j_1\ldots j_{d-2}}\Big)\Big)R.$$

Let us define $D_{i_1\dots i_{d-2}} = \sum_{j_1\dots j_{d-2}\in[n]}(\prod_{m\in[d-2]} r_{j_m,i_m})S_{j_1\dots j_{d-2}}$. Then we have for all $i \in \{2,\dots,n\}$:

$$T_{\bar{1}}'T_{\bar{i}} = R^{-1}(D_{\bar{1}})^{-1}(R)^{-T}R^T D_{\bar{i}}R = R^{-1}(\sum_{j_1\dots j_{d-2}\in[n]}(\prod_{m\in[d-2]} r_{j_m,i})(D_{\bar{1}})^{-1}S_{j_1\dots j_{d-2}})R. \tag{2.3}$$

So, if $T_{\bar{1}}$ is invertible, $T_{\bar{1}}'T_{\bar{2}}$ commutes with $T_{\bar{1}}'T_{\bar{3}}$ iff $(D_{\bar{1}})^{-1}D_{\bar{2}}$ commutes with $(D_{\bar{1}})^{-1}D_{\bar{3}}$.

Let E_1 be the event that $T_{\bar{1}}$ is invertible and $T_{\bar{1}}'T_{\bar{2}}$ commutes with $T_{\bar{1}}'T_{\bar{3}}$. Let E_1' be the event that $D_{\bar{1}}$ is invertible and $(D_{\bar{1}})^{-1}D_{\bar{2}}$ commutes with $(D_{\bar{1}})^{-1}D_{\bar{3}}$. Let E_4 be the event that R is invertible. Then we have that $E_1 = E_1' \cap E_4$.

Let E_2 be the event that $D_{\bar{1}}$ is invertible and $\{(D_{\bar{1}})^{-1}S_{i_1,\dots,i_{d-2}}\}_{i_1,\dots,i_{d-2}\in[n]}$ is not a commuting family. Since $\mathcal{V}$ does not satisfy the commutativity property, $(D_{\bar{1}})^{-1}\mathcal{V}$ is not a commuting subspace if $D_{\bar{1}}$ is invertible. Hence, the event that $D_{\bar{1}}$ is invertible is the same as the event E_2. This also implies that $E_1' \subseteq E_2$. Setting $A_{i_1\dots i_{d-2}} = (D_{\bar{1}})^{-1}S_{i_1\dots i_{d-2}}, \alpha_i = r_{i,2}, \alpha_i' = r_{i,3}$ and then using Lemma 2.3.13, we can conclude that $\Pr_{R\in S}\left[E_1'|E_2\right] \leq \frac{2(d-2)}{|S|}$.

Note here that $D_{\bar{1}}$ depends only on the random variables $r_{i,1}$ for all $i \in [n]$ and therefore is independent of $r_{k,2}$ and $r_{l,3}$ for all $k,l \in [n]$, because we assume that the entries of R are all picked uniformly and independently at random.
Let E_3 be the event that $T_{\bar{1}}$ is invertible. Now we know that $T_{\bar{1}}$ is invertible iff R and $D_{\bar{1}}$ are invertible. Then, we have $E_3 = E_2 \cap E_4$ Hence, the probability of error can be bounded as follows:

$$\Pr_{R\in S}[E_1] = \Pr_{R\in S}[E_1' \cap E_2 \cap E_4] \leq \Pr_{R\in S}[E_1'|E_2] \leq \frac{2(d-2)}{|S|}.$$

□

Failure of diagonalisability

Theorem 2.3.14 gives us a way to analyze the case when the slices of the input polynomial fail to satisfy the commutativity property. With the results in the present section we will be able to analyze the case where the commutativity property is satisfied, but the diagonalisability property fails (recall that these properties are relevant due to Theorem 2.3.8).

Lemma 2.3.15. *Let $\{A_{i_1\dots i_d}\}_{i_1,..,i_d\in[n]} \in M_n(\mathbb{K})$ be a commuting family of symmetric matrices. Let us assume that this family is symmetric in the sense of Definition 2.3.12 and there exists $i_1^0,\dots,i_d^0 \in [n]$ such that $A_{i_1^0\dots i_d^0}$ is not diagonalisable. Let $S \subset \mathbb{K}$ be a finite set. Then $D = \sum_{i_1,\dots,i_d=1}^n(\prod_{m\in[d]}\alpha_{i_m})A_{i_1\dots i_d}$ is diagonalisable with probability at most $\frac{d}{|S|}$ when $\alpha_1,\dots,\alpha_n$ are chosen uniformly and independently at random from S.*

Proof. We define $\mathcal{U} = \text{span}\{A_{i_1\dots i_d}\}_{i_1,..,i_d\in[n]}$. We also define the class of matrices

$$\mathcal{M} := \left\{M \middle| M \text{ is diagonalisable and } M \in \mathcal{U}\right\}.$$

So, we want to show that $\Pr_{\alpha\in S}\left[D \in \mathcal{M}\right] \leq \frac{d}{|S|}$.

Now using Proposition 2.2.15, and the hypothesis that there exists $A_{i_1^0\dots i_d^0} \in \mathcal{U} \setminus \mathcal{M}$, we get that $\mathcal{M}$ is a proper linear subspace of $\mathcal{U}$. So $\mathcal{M}$ is an intersection of

hyperplanes. Since $A_{i_1^0 \ldots i_d^0} \notin \mathcal{M}$, there exists a linear form

$$l_{\mathcal{M}}(X) = \sum_{i,j \in [n]} a_{ij} X_{ij}$$

corresponding to a hyperplane such that $l_{\mathcal{M}}(M) = 0$ for all $M \in \mathcal{M}$ and $l_{\mathcal{M}}(A_{i_1^0 \ldots i_d^0}) \neq 0$. We know that if D is diagonalisable, then $l_{\mathcal{M}}(D) \neq 0$. We compute the polynomial

$$l_{\mathcal{M}}(D)(\alpha) = \sum_{i_1, \ldots, i_d \in [n]} \Big(\prod_{m \in [d]} \alpha_{i_m} \Big) m_{i_1 \ldots i_d}$$

where $m_{i_1 \ldots i_d} = (\sum_{k,l \in [n]} a_{kl} (A_{i_1 \ldots i_d})_{k,l})$.

Now we claim that $l_{\mathcal{M}}(D) \not\equiv 0$. We show this by proving that the coefficient of $\alpha_{i_1^0} \ldots \alpha_{i_d^0}$ in $l_{\mathcal{M}}(D)(\alpha)$ is not equal to 0. Let $I_0 = \{(i^0_{\sigma(1)}, \ldots, i^0_{\sigma(d)}) | \sigma \in S_d\}$. Then $\text{coeff}_{\alpha_{i_1^0} \ldots \alpha_{i_d^0}}(D) = \sum_{\bar{i} \in I_0} m_{\bar{i}}$. Since the matrices $A_{i_1 \ldots i_d}$ form a symmetric family, the $m_{\bar{i}}$ are equal for all $\bar{i} \in I_0$. Also, since, $l_{\mathcal{M}}(A_{i_1^0 \ldots i_d^0}) \neq 0$, we get that

$$\sum_{k,l \in [n]} a_{k,l} (A_{i_1^0 \ldots i_d^0})_{k,l} \neq 0.$$

This gives us that $m_{i_1^0 \ldots i_d^0} \neq 0$. Hence, we get that

$$\text{coeff}_{\alpha_{i_1^0} \ldots \alpha_{i_d^0}}(D) = |I_0| m_{i_1^0 \ldots i_d^0} \neq 0.$$

Thus, $l_{\mathcal{M}}(D) \not\equiv 0$ and $\deg(l_{\mathcal{M}}(D)) \leq d$. From Lemma 2.2.9, we have

$$\text{Pr}_{\alpha \in S}\Big[D \in \mathcal{M}\Big] \leq \text{Pr}_{\alpha \in S}[l_{\mathcal{M}}(D)(\alpha) = 0] \leq \frac{d}{|S|}.$$

□

The last result for this section is an analogue of the Theorem 2.3.14 for the diagonalisability property.

Theorem 2.3.16. *Let $f \in \mathbb{K}[x_1, \ldots, x_n]_d$ be a degree-d form such that the subspace $\mathcal{V}$ spanned by its slices is a not weakly-singular subspace, satisfies the commutativity property, but does not satisfy the diagonalisability property. Let $h(x) = f(Rx)$ where the entries $r_{i,j}$ of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$, Let $\{T_{i_1 \ldots i_{d-2}}\}_{i_1, \ldots, i_{d-2} \in [n]}$ be the slices of h. If $T_{\bar{1}}$ is invertible, define $T'_{\bar{1}} = (T_{\bar{1}})^{-1}$. Then*

$$Pr[T_{\bar{1}} \text{ is invertible and } T'_{\bar{1}} T_{\bar{2}} \text{ is diagonalisable }] \leq \frac{d-2}{|S|}.$$

Proof. As in the proof of Theorem 2.3.14, we use the expression for $T'_{\bar{1}} T_{\bar{2}}$ which we obtain from the definition of the slices i.e.

$$R^{-1} \Big(\sum_{j_1 \ldots j_{d-2} \in [n]} \Big(\prod_{m \in [d-2]} r_{j_m,2} \Big) (D_{\bar{1}})^{-1} S_{j_1 \ldots j_{d-2}} \Big) R$$

where $D_{\bar{1}} = \sum_{j_1 \ldots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m,1}) S_{j_1 \ldots j_{d-2}}$. So if $T_{\bar{1}}$ is invertible, $T'_{\bar{1}} T_{\bar{2}}$ is diagonalisable iff $M = (\sum_{j_1 \ldots j_{d-2} \in [n]} (\prod_{m \in [d-2]} r_{j_m,2}) (D_{\bar{1}})^{-1} S_{j_1 \ldots j_{d-2}})$ is diagonalisable.

We denote by E_1 the event that $T_{\bar{1}}$ is invertible and $T'_{\bar{1}}T_{\bar{2}}$ is diagonalisable and by E'_1 the event that $D_{\bar{1}}$ is invertible and M is diagonalisable. Let E_4 be the event that R is invertible. Hence, $E_1 = E'_1 \cap E_4$.

Let E_2 be the event that $D_{\bar{1}}$ is invertible and $\{(D_{\bar{1}})^{-1}S_{i_1\dots i_{d-2}}\}_{i_1,\dots,i_{d-2}\in[n]}$ is a commuting family and there exists $j_1\dots j_{d-2} \in [n]$ such that $(D_{\bar{1}})^{-1}S_{j_1\dots j_{d-2}}$ is not diagonalisable. Since $\mathcal{V}$ satisfies the commutativity property and does not satisfy the diagonalisability property, by Theorem 2.2.7, the event that $D_{\bar{1}}$ is invertible is the event same as E_2. It can also be observed that $E'_1 \subseteq E_2$.

Setting $A_{i_1\dots i_{d-2}} = (D_{\bar{1}})^{-1}S_{i_1\dots i_{d-2}}$ and setting $\alpha_i = r_{i,2}$ for all $i \in [n]$ and using Lemma 2.3.15, we get that $\Pr_{R\in S}\left[E'_1 \middle| E_2\right] \leq \frac{d-2}{|S|}$. Now we know that $T_{\bar{1}}$ is invertible iff R and $D_{\bar{1}}$ is invertible. Let E_3 be the event that $T_{\bar{1}}$ is invertible. Then, we have $E_3 = E_2 \cap E_4$. The probability of error can finally be bounded as follows: $\Pr_{R\in S}[E_1 \cap E_3] = \Pr_{R\in S}[E'_1 \cap E_2 \cap E_4] \leq \Pr_{R\in S}[E'_1|E_2] \leq \frac{d-2}{|S|}$. □

Finishing the analysis for negative inputs

In this section we complete the proof of Theorem 2.3.1. The case of positive inputs was treated in Section 2.3.3. It therefore remains to prove the following result.

Theorem 2.3.17. *If an input $f \in \mathbb{K}[x_1, ..., x_n]_d$ is not equivalent to some polynomial $P_d \in \mathcal{P}_d$, then f is rejected by the algorithm with high probability over the choice of the random matrix R. More precisely, if the entries $r_{i,j}$ are chosen uniformly and independently at random from a finite set $S \subseteq \mathbb{K}$, then the input will be rejected with probability $\geq (1 - \frac{2(d-2)}{|S|})$.*

Proof. Let $\{S_{i_1,\dots,i_{d-2}}\}_{i_1,\dots,i_{d-2}\in[n]}$ be the slices of f and $\mathcal{V} = \text{span}\{S_{i_1,\dots,i_{d-2}}\}$. From Theorem 2.3.6 and Theorem 2.2.8, we know that if $f \not\sim P_d$, then there are three disjoint cases:

1. **Case 1:** $\mathcal{V}$ is a weakly singular subspace of matrices.
2. **Case 2:** $\mathcal{V}$ is not a weakly singular subspace and $\mathcal{V}$ does not satisfy the commutativity property.
3. **Case 3:** $\mathcal{V}$ is not a weakly singular subspace, $\mathcal{V}$ satisfies the commutativity property but does not satisfy the diagonalisability property.

Now we try to upper bound the probability of error in each case.
In case 1, $T_{\bar{1}} = R^T(\sum_{j_1\dots j_{d-2}\in[n]} r_{j_1,1}\dots r_{j_{d-2},1}S_{j_1\dots j_{d-2}})R \in R^T\mathcal{V}R$ is always singular for any choice of $r_{j,1}$. So f is rejected with probability 1 in this case.
In case 2, we can upper bound the probability of error as follows:

$$\begin{aligned}&\Pr_{R\in S}[f \text{ is accepted by the algorithm}]\\ &= \Pr_{R\in S}[T_{\bar{1}} \text{ is invertible}, T'_{\bar{1}}T_{\bar{2}}, T'_{\bar{1}}T_{\bar{3}} \text{ commute }, T'_{\bar{1}}T_{\bar{2}} \text{ is diagonalisable}]\\ &\leq \Pr_{R\in S}[T_{\bar{1}} \text{ is invertible}, T'_{\bar{1}}T_{\bar{2}}, T'_{\bar{1}}T_{\bar{3}} \text{ commute }].\end{aligned}$$

Using Theorem 2.3.14, we get that this occurs with probability at most $\frac{2(d-2)}{|S|}$. In Case 3, we have the following upper bound on the probability of error,

$$\begin{aligned}&\Pr_{R\in S}[f \text{ is accepted by the algorithm}]\\ &= \Pr_{R\in S}[T_{\bar{1}} \text{ is invertible}, T'_{\bar{1}}T_{\bar{2}}, T'_{\bar{1}}T_{\bar{3}} \text{ commute }, T'_{\bar{1}}T_{\bar{2}} \text{ is diagonalisable}]\\ &\leq \Pr_{R\in S}[T_{\bar{1}} \text{ is invertible}, T'_{\bar{1}}T_{\bar{2}} \text{ is diagonalisable}].\end{aligned}$$

By Theorem 2.3.16, this occurs with probability $\leq \frac{d-2}{|S|}$. Therefore in all these three cases, the algorithm rejects f with probability at least $1 - \frac{2(d-2)}{|S|}$. □

2.4 Variable Minimization

We first recall the notion of redundant and essential variables studied by Carlini [Car06] and Kayal [Kay11].

Definition 2.4.1. *A variable x_i in a polynomial $f(x_1, ..., x_n)$ is redundant if f does not depend on x_i , i.e., x_i does not appear in any monomial of f.*

Let $f \in \mathbb{K}[x_1, ..., x_n]$. The number of essential variables is the smallest number t such that there exists an invertible linear transformation $A \in \mathbb{K}^{n\times n}$ on the variables such that every monomial of $f(Ax)$ contains only the variables $x_1, ..., x_t$.

In this section we propose the following algorithm for variable minimization.

Algorithm 3: Randomized algorithm for $\leq n$ linearly independent linear forms

Input: A degree-d homogeneous polynomial P given by a blackbox
Pick $(\alpha_1, ..., \alpha_n)$ where $\alpha_j = (\alpha_j^{(1)}, ..., \alpha_j^{(n)})$ and $\alpha_j^{(i)}$ are picked uniformly and independently at random from a finite set $S \subset \mathbb{K}$
Compute $M = (\frac{\partial P}{\partial x_j}(\alpha_i))$, such that $i, j \in [n]$
Perform Gaussian elimination on M and define the basis of the kernel $B = \{v_1, ..., v_{n-t}\}$
Add vectors $u_1, ..., u_t$ to B to obtain a basis for $\mathbb{K}^n$
Define $n \times n$ matrix $A = (u_1, ..., u_t, v_1, ..., v_{n-t})$ where u_i and v_j are the columns of A
Let $f(x) = P(Ax)$
Run Algorithm 2 on $f(x_1, ..., x_t, 0, ..., 0)$
if *Algorithm 2 accepts* **then**
  accept
else
  reject
end

A randomized algorithm for minimizing the number of variables is given in ([Kay11], 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 that $t = \dim(\partial f)$ where $\partial(f)$ denotes the tuple of n first-order partial derivatives $\frac{\partial f}{\partial x_i}$ (and $\dim(\partial f)$ denotes the dimension of the spanned subspace). [Kay11] then uses Lemma 2.4.3 along with Theorem 2.4.4 to return the required invertible matrix.

We combine the algorithm for minimizing the number of variables along with Algorithm 2 to check if there exists a decomposition of the polynomial into linear combination of d-th powers of $\leq n$ many linearly independent linear forms. More formally, Algorithm 3 decides in polynomial time over $\mathbb{C}$ or $\mathbb{R}$ whether the input f which is given as blackbox can be written as $\sum_{i=1}^{t} \alpha_i l_i^d$ for some $t \leq n$ where l_i's are linearly independent linear forms and $\alpha_i \neq 0$ for all $i \in [t]$. We do a detailed complexity analysis of this algorithm in Appendix A.2.

Definition 2.4.2. *Let $\boldsymbol{f}(x) = (f_1(x), ..., f_m(x)) \in (\mathbb{K}[x])^m$ be a vector of polynomials over a field $\mathbb{K}$. The set of $\mathbb{K}$-linear dependencies in $\boldsymbol{f}$, denoted by $\boldsymbol{f}^{\perp}$, is the set of all*

vectors $\boldsymbol{v} \in \mathbb{K}^m$*, whose inner product with* $\mathbf{f}$ *is the zero polynomial i.e*

$$\boldsymbol{f}^{\perp} := \{(a_1, ..., a_m) \in \mathbb{K}^m | \sum_{i \in [m]} a_i f_i(x) = 0\}$$

This following lemma from [Kay11] gives a randomized algorithm to compute the basis of linear dependencies of a vector of polynomials. We restate it here for completeness and also calculate the probability bounds which we will need for our correctness proof of Algorithm 3.

Lemma 2.4.3. *Let* $f = (f_1(X), ..., f_m(X))$ *be a vector of* m *polynomial with* $deg(f_i) \leq d$ *such that* $rank(f^{\perp}) = t$*. Pick* $a_1, ..., a_m$ *where* $a_j = (a_j^{(1)}, ..., a_j^{(m)})$ *where the* $a_j^{(i)}$*'s are chosen uniformly and independently at random from a finite set* $S \subseteq \mathbb{K}$ *for all* $i, j \in [m]$*. Define matrix*

$$P(a_1, ..., a_m) = \begin{bmatrix} f_1(a_1) & f_2(a_1) & \dots & f_m(a_1) \\ f_1(a_2) & f_2(a_2) & \dots & f_m(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(a_m) & f_2(a_m) & \dots & f_m(a_m) \end{bmatrix}$$

Then

$$Pr_{a \in S}[rank(P(a_1, ..., a_m)) = m - t] \geq 1 - \frac{(m-t)d}{|S|}$$

Additionally, if $rank(P(a_1, ..., a_m)) = m - t$*, then* $ker(P(a_1, ..., a_m)) = f^{\perp}$*.*

Proof. Without loss of generality, we assume that the polynomial $f_1, ..., f_{m-t}$ are $\mathbb{K}$-linearly independent and the rest of the polynomials are $\mathbb{K}$ linear combinations of the first $m - t$ polynomials. So it is enough to prove that

$$Q(a_1, ..., a_m) = \begin{bmatrix} f_1(a_1) & f_2(a_1) & \dots & f_{m-t}(a_1) \\ f_1(a_2) & f_2(a_2) & \dots & f_m(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(a_{m-t}) & f_2(a_{m-t}) & \dots & f_m(a_{m-t}) \end{bmatrix}$$

has full rank, which is equivalent to proving that $\det(Q)(a_1, ..., a_m) \neq 0$.
Now $\deg(\det(Q)(x_1, ..., x_n)) \leq (m-t)d$. Claim 7 in [Kay11] shows that $\det(Q) \not\equiv 0$. Applying Lemma 2.2.9, we get that,

$$\Pr_{a \in S}[\text{rank}(P(a_1, ..., a_m)) = m - t] \geq 1 - \frac{(m-t)d}{|S|}$$

□

Recall that we define $\partial(f) = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n})$. Let $b_1, ..., b_{n-t}$ be a basis for $\partial(f)^{\perp}$. Now there exists t independent vectors $a_1, ..., a_t$ such that the vector space $\mathbb{K}^n$ is spanned by $a_1, ..., a_t, b_1, ..., b_{n-t}$. We define A_f to be the invertible matrix whose columns are $a_1, ..., a_t, b_1, ..., b_{n-t}$.

Theorem 2.4.4. *[Car06] The number of redundant variables in a polynomial* $f(x)$ *equals the dimension of* $\partial(f)^{\perp}$*. Furthermore, given a basis of* $\partial(f)^{\perp}$ *, the polynomial* $f(A_f X)$ *depends on only the first* $(n - dim(\partial(f)^{\perp})))$ *variables.*

Lemma 2.4.5. *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.*

Proof. Follows from Example 42 in [KS21]. □

We combine all the results in this section along with Theorem 2.3.1 to give a correctness proof for Algorithm 3.

Theorem 2.4.6. *If an input $P \in \mathbb{K}[x_1, ..., x_n]_d$ can not be written as $\sum_{i=1}^{t} \alpha_i l_i^d$, where l_i are linearly independent linear forms and $\alpha_i \neq 0$ for all $i \in [t]$, for any $t \leq n$, then P is rejected by Algorithm 3 with high probability over the choice of the random matrix R and the points $\alpha_1, ..., \alpha_n$. More formally, if the entries α_j^i and $r_{i,j}$ are chosen uniformly and independently at random from a finite set S, then the input will be rejected with probability $\geq (1 - \frac{2(d-2)}{|S|})(1 - \frac{n(d-1)}{|S|})$.*
Conversely, if $P \in \mathbb{K}[x_1, ..., x_n]_d$ can be written as $\sum_{i=1}^{t} \alpha_i l_i^d$ where l_i are linearly independent linear forms and $\alpha_i \neq 0$ for all $i \in [t]$, then P will be accepted by Algorithm 3 with high probability over the choice of the random matrix R and the points $a_1, ..., a_n$. More formally, if the entries $\alpha_j^{(i)}$ and $r_{i,j}$ are chosen uniformly and independently at random from a finite set S, then the input will be accepted with probability $\geq (1 - \frac{t(d-1)}{|S|})^2$.

Proof. Let us assume that P has t essential variables and we fix M as given by the algorithm. Then using Lemma 2.4.3 for $f_i = \frac{\partial P}{\partial x_i}$ and $a_i = \alpha_i$, we get that

$$\Pr_\alpha[v_1, ..., v_{n-t} \text{ is a basis for } \partial(P)^\perp] \geq 1 - \frac{t(d-1)}{|S|}.$$

The linear transformation A defined in the algorithm, satisfies the conditions of the linear transformation defined in Theorem 2.4.4 with $b_i = v_i$ and $a_i = u_i$ with probability $\geq 1 - \frac{t(d-1)}{|S|}$. Now, we define $f(x) = P(Ax)$. This gives us that

$$\begin{aligned}\Pr_\alpha[f(x) \text{ depends only on the first } t \text{ variables}] &\geq 1 - \frac{t(d-1)}{|S|} \\ &\geq 1 - \frac{n(d-1)}{|S|}.\end{aligned}$$

Using Theorem 2.3.1, we get that if f can not be written as a sum of t-many sum of d-th powers of linear forms for any $t \leq n$, then the algorithm rejects the polynomial with probability $\geq (1 - \frac{2(d-2)}{|S|})(1 - \frac{n(d-1)}{|S|})$.

For the converse, if f can be written as a sum of t-many sum of d-th powers of linear forms, using Lemma 2.4.5, then the polynomial has t essential variables. Then the algorithm accepts P with probability $\geq (1 - \frac{t(d-1)}{|S|})^2$. □

2.5 Reconstruction Algorithm for P_d

The general reconstruction problem for a special class of arithmetic circuits can be stated as follows: Given a homogeneous degree-d polynomial, output the smallest circuit that computes it. In this section, we look at the reconstruction problem for linear combination of d-th powers of linearly independent linear forms. Notice that Algorithm 2 already solves the decision version of this problem in polynomial time i.e. Given a homogeneous degree-d polynomial, can it be written as a linear combination of d-th powers of linearly independent linear forms?

In terms of the polynomial equivalence problem, we have already shown that given a homogeneous degree-d polynomial f, we can check in polynomial time if there exists an invertible matrix A and some $P_d \in \mathcal{P}_d$ such that $f(x) = P_d(Ax)$. We give an algorithm (see Algorithm 4 below) that uses Algorithm 2 to check the existence of such an A and then outputs the A and the corresponding P_d. When $f \in \mathbb{C}[x_1, ..., x_n]_d$, this algorithm runs in polynomial time, if we allow the computation of polynomial roots in our model.

Algorithm 4: Randomized Reconstruction Algorithm

Result: The algorithm checks if f is equivalent to some polynomial in $\mathcal{P}_d$ and outputs $\{(\alpha_1, l_1), ..., (\alpha_n, l_n)\}$ where $0 \neq \alpha_i \in \mathbb{K}$ and l_i's are linearly independent linear forms such that $f = \sum_{i=1}^{n} \alpha_i l_i^d$

Input: A degree-d homogeneous polynomial f given as blackbox
Let $R \in M_n(\mathbb{K})$ be a matrix such that its entries r_{ij} are picked uniformly and independently at random from a finite set S and set $h(x) = f(Rx)$
Let $T_{i_1,...,i_{d-2}}$ be the slices of h for all $i_1, ..., i_{d-2} \in [n]$
Compute $T_{\bar{1}}, T_{\bar{2}}, T_{\bar{3}}$
if $T_{\bar{1}}$ *is singular* **then**
    reject
else
    compute $T'_{\bar{1}} = T_{\bar{1}}^{-1}$
    if $T'_{\bar{1}}T_{\bar{2}}$ *and* $T'_{\bar{1}}T_{\bar{3}}$ *commute and* $T'_{\bar{1}}T_{\bar{2}}$ *is diagonalisable over* $\mathbb{K}$ **then**
        diagonalise $T'_{\bar{1}}T_{\bar{2}} = P\Lambda P^{-1}$
        Let l_i be the i-th row of $(R^{-1}P^{-1})$ and let $\alpha_i = f(PRx)(e_i)$ where $e_i \in \mathbb{K}^n$ is the i-th standard basis vector for all $i \in [n]$
        Output $\{(\alpha_1, l_1), ..., (\alpha_n, l_n)\}$
    else
        reject
    end
end

Note that this output is unique up to the permutation and scaling of the linear forms by a constant. If the linear form l_i is scaled by a constant c, then it is reflected in the α_i which becomes $\frac{\alpha_i}{c^d}$.

A natural question is to study the approximate version for this reconstruction problem i.e. if the input polynomial admits such a decomposition, the linear forms which are returned by the algorithm are "arbitrarily close" to the required linear forms. In this context, one should note that interestingly, the only non-algebraic step in this algorithm is the step of matrix diagonalization. All other steps can be computed exactly in polynomial time. Recently, [BGVKS22] gave a poly-time randomized algorithm for the approximate version of matrix diagonalization problem. Referring to that algorithm for our diagonalization step combined with our reconstruction algorithm should effectively give a polynomial time randomized algorithm for the approximate version of the reconstruction problem. A more precise analysis is left for future work.

For the next lemma, we assume that the input polynomial f is equivalent to some polynomial $P_d \in \mathcal{P}_d$ i.e. $f(x) = P_d(Ax)$ where A is an invertible matrix. Take a random change of variables and let $T_{\bar{1}}$ and $T_{\bar{2}}$ be two slices of the new polynomial. Then, the eigenvalues of $(T_{\bar{1}})^{-1}T_{\bar{2}}$ are distinct with high probability over the random change of variables. Lemma 2.5.1 along with Corollary 2.5.1.1 ensures that just diagonalization of a single $(T_{\bar{1}})^{-1}T_{\bar{2}}$ is enough to recover the matrix A uniquely (up to permutation and scaling of the rows).

Theorem 2.5.1. *Let $f(x) = P_d(Ax)$ for some $P_d \in \mathcal{P}_d$ where A is an invertible matrix. Let $h(x) = f(Rx)$ where the entries r_{ij} of R are chosen uniformly and independently at random from a finite set S. Let the slices of h be $\{T_{i_1\dots i_{d-2}}\}_{i_1,\dots,i_{d-2}\in[n]}$. If $T_{\bar{1}}$ is invertible, let $T'_{\bar{1}} = T_{\bar{1}}^{-1}$. Let $\lambda_1, \dots, \lambda_n$ be the eigenvalues of $T'_{\bar{1}}T_{\bar{2}}$. Then*

$$Pr_{R\in S}[\text{There exists } i,j \in [n] \text{ such that } \lambda_i = \lambda_j \text{ and } T_{\bar{1}} \text{ is invertible }] \leq \frac{2\binom{n}{2}(d-2)}{|S|}.$$

Proof. Let us assume that $P_d(x) = \sum_{i=1}^n \alpha_i x_i^d$ where $\alpha_i \neq 0$. We use the fact that $h(x) = P_d(RAx)$. Let $\{S_{i_1\dots i_{d-2}}\}_{i_1,\dots,i_{d-2}\in[n]}$ be the slices of P_d. Then we know from (2.3),

$$T'_{\bar{1}}T_{\bar{2}} = (RA)^{-1}\Big(\sum_{j_1\dots j_{d-2}\in[n]} \prod_{m\in[d-2]} (RA)_{j_m,2}(D_{\bar{1}})^{-1}S_{j_1\dots j_{d-2}}\Big)(RA)$$

where

$$D_{\bar{1}} = \sum_{j_1\dots j_{d-2}\in[n]} \Big(\prod_{m\in[d-2]} (RA)_{j_m,1}\Big)S_{j_1\dots j_{d-2}}.$$

Since $S_{i_1\dots i_{d-2}}$ are the slices of P_d, we know that

$$\begin{aligned} S_{i_1\dots i_{d-2}} &= \alpha_i \text{diag}(e_i) \text{ if } i_1 = i_2 = \dots = i_{d-2} = i \\ &= 0 \text{ otherwise.} \end{aligned}$$

Since $T_{\bar{1}}$ is invertible, then R and $D_{\bar{1}}$ are invertible. Now

$$D_{\bar{1}} = \text{diag}(\alpha_1((RA)_{1,1})^{d-2}, \dots, \alpha_n((RA)_{n,1})^{d-2}).$$

Since $D_{\bar{1}}$ is invertible, this gives us that $\alpha_i((RA)_{i,1})^{d-2} \neq 0$ for all $i \in [n]$ and we obtain

$$T'_{\bar{1}}T_{\bar{2}} = (RA)^{-1}\Big(\text{diag}\Big(\frac{((RA)_{1,2})^{d-2}}{((RA)_{1,1})^{d-2}}, \dots, \frac{((RA)_{n,2})^{d-2}}{((RA)_{n,1})^{d-2}}\Big)\Big)RA \tag{2.4}$$

This gives us that $\lambda_i = \frac{((RA)_{i,2})^{d-2}}{((RA)_{i,1})^{d-2}}$. We define

$$P_{i,j}(R) = ((RA)_{i,2}(RA)_{j,1})^{d-2} - ((RA)_{j,2}(RA)_{i,1})^{d-2}.$$

We can see that $\lambda_i \neq \lambda_j$ iff $P_{i,j}(R) \neq 0$. Also, $\deg(P_{ij}) \leq 2(d-2)$. We can choose R such that $(RA)_{i,2} = 1$, $(RA)_{j,1} = 1$, $(RA)_{i,1} = 0$, $(RA)_{j,2} = 0$, (If A is invertible, there exists R such that $RA = M$ for any matrix M). Hence, $P_{ij} \not\equiv 0$. Using Schwartz- Zippel Lemma, we get that

$$\text{Pr}_{R\in S}[P_{ij} = 0] \leq \frac{2(d-2)}{|S|}.$$

This gives us that

$$\text{Pr}_{R\in S}[T_{\bar{1}} \text{ is invertible and } \lambda_i = \lambda_j] \leq \text{Pr}_{R\in S}[P_{ij} = 0] \leq \frac{2(d-2)}{|S|}.$$

Taking the union bound over all possible pairs of $i, j \in [n]$, we get that

$$\Pr_{R\in S}[\text{There exists } i,j \in [n] \text{ such that } \lambda_i = \lambda_j \text{ and } T_{\bar{1}} \text{ is invertible }] \leq \frac{2\binom{n}{2}(d-2)}{|S|}.$$

□

Corollary 2.5.1.1. *Let $f(x)$ be a degree-d form which is equivalent to some polynomial in $\mathcal{P}_d$. Let $h(x) = f(Rx)$ where the entries r_{ij} of R are chosen uniformly and independently at random from a finite set $S \subset \mathbb{K}$. Let the slices of h be $\{T_{i_1,\ldots,i_{d-2}}\}_{i_1,\ldots,i_{d-2}\in[n]}$. If $T_{\bar{1}}$ is invertible, let $T'_{\bar{1}} = T_{\bar{1}}^{-1}$. Suppose $T'_{\bar{1}}T_{\bar{2}}$ can be diagonalised as $P\Lambda P^{-1}$. Let l_i be the rows of $R^{-1}P^{-1}$. Define $\alpha_i = f(PRx)(e_i)$ where $e_i \in \mathbb{K}^n$ is the i-th standard basis vector for all $i \in [n]$. Then*

$$Pr_{R\in S}[T_{\bar{1}} \text{ is invertible and } f(x) = \sum_{i=1}^{n} \alpha_i l_i^d] \geq 1 - \frac{2\binom{n}{2}(d-2)}{|S|}.$$

Proof. By Theorem 2.5.1, we get that

$$\Pr_{R\in S}[T_{\bar{1}} \text{ is invertible and } T'_{\bar{1}}T_{\bar{2}} \text{ has distinct eigenvalues}] \geq 1 - \frac{2\binom{n}{2}(d-2)}{|S|}.$$

We will now show that if $T_{\bar{1}}$ is invertible and $T'_{\bar{1}}T_{\bar{2}}$ has distinct eigenvalues, then $f(x) = \sum_{i=1}^{n} \alpha_i l_i^d$.

If the eigenvalues are distinct, then the rank of the eigenspaces corresponding to each eigenvalue is 1. Hence, the eigenvectors of $T'_{\bar{1}}T_{\bar{2}}$ are unique (up to a scaling factor). We already know that $h(x) = P_d(Bx)$ for some $P_d \in \mathcal{P}_d$ such that B is invertible. Then the columns of B^{-1} form the eigenvectors of $T'_{\bar{1}}T_{\bar{2}}$. We take the diagonalization of $T'_{\bar{1}}T_{\bar{2}}$ into $P\Lambda P^{-1}$. Note here that the columns of P form the eigenvectors for $T'_{\bar{1}}T_{\bar{2}}$. The uniqueness of eigenvectors of $T'_{\bar{1}}T_{\bar{2}}$ gives us that the set of columns of P are essentially the set of columns of B^{-1} upto a scaling factor. So this gives us that $h(x) = P'_d(P^{-1}x)$ for some $P'_d \in \mathcal{P}_d$. We know that $f(x) = h(R^{-1}x)$. This gives us that $f(x) = P'_d(R^{-1}P^{-1})x$. We define $A = R^{-1}P^{-1}$ and the i-th row of A as l_i. This fixes the set of linear forms of the decomposition of the input polynomial which are unique up to a scaling factor.
Taking $P_d = f(A^{-1}x)$ gives the corresponding polynomial in $\mathcal{P}_d$ such that f is equivalent to P_d. Now $P_d = \sum_{i=1}^{n} \alpha_i x_i^d$ where $\alpha_i \neq 0$. Evaluating P_d at $e_i \in \mathbb{K}^n$ where e_i is the i-th standard basis vector, returns the corresponding α_i. Hence,

$$\Pr_{R\in S}[f(x) = \sum_{i=1}^{n} \alpha_i l_i^d \text{ and } T_{\bar{1}} \text{ is invertible}] \geq 1 - \frac{2\binom{n}{2}(d-2)}{|S|}.$$

□

We combine all the results in this section along with Theorem 2.3.1 to give a correctness proof of Algorithm 4.

Theorem 2.5.2. *If an input $f \in \mathbb{K}[x_1, \ldots, x_n]_d$ is not equivalent to some polynomial in $\mathcal{P}_d$, then f is rejected by Algorithm 4 with high probability. More formally, if the entries $r_{i,j}$ of R are chosen uniformly and independently at random from a finite set S, then the input will be rejected with probability $\geq (1 - \frac{2(d-2)}{|S|})$.*

Conversely, if an input $f \in \mathbb{K}[x_1, ..., x_n]_d$ *is equivalent to some polynomial in* $\mathcal{P}_d$*, then Algorithm 4 outputs such a polynomial with high probability. More formally,if the entries of a matrix* R *are chosen uniformly and independently from a finite set* S*, then the algorithm outputs a set of linearly independent linear forms* l_i *and corresponding coefficients* $\alpha_i \neq 0$ *such that* $f = \sum_{i=1}^{n} \alpha_i l_i^d$ *with probability* $\geq 1 - (\frac{2\binom{n}{2}(d-2)}{|S|} + \frac{n(d-1)}{|S|})$.

Proof. From Theorem 2.3.17, we get that if f is not equivalent to any polynomial in $\mathcal{P}_d$, then f is rejected by Algorithm 4 with probability $\geq (1 - \frac{2(d-2)}{|S|})$.

For the converse, we start by assuming that f is equivalent to some polynomial in $\mathcal{P}_d$. We know that if the first slice $T_{\bar{1}}$ of $h(x) = f(Rx)$ is not invertible, the Algorithm always makes an error and rejects the input. From Theorem 2.3.8, we know that the subspace spanned by the slices of f is not weakly singular. We can therefore apply Lemma 2.3.9, we get that

$$\Pr_{R \in S}[T_{\bar{1}} \text{ is not invertible}] \leq \frac{n(d-1)}{|S|}. \tag{2.5}$$

Moreover if $T_{\bar{1}}$ is invertible, Lemma 2.3.10 shows that f will always be accepted. Let the output of the algorithm be $\{(\alpha_i, l_i)\}_{i \in [n]}$. So the only possible error is when $f \neq \sum_{i \in [n]} \alpha_i l_i^d$. From Corollary 2.5.1.1, we get that

$$\Pr_{R \in S}[\text{Algorithm makes an error and } T_{\bar{1}} \text{ is invertible}] \leq \frac{2\binom{n}{2}(d-2)}{|S|}. \tag{2.6}$$

Combining (2.5) and (2.6), we get that if f is equivalent to some polynomial in $\mathcal{P}_d$, then Algorithm 4 returns a set of linearly independent linear forms l_i and corresponding coefficients $\alpha_i \neq 0$ (which are unique up to scaling and permutation) such that $f = \sum_{i=1}^{n} \alpha_i l_i^d$ with probability $\geq 1 - (\frac{2\binom{n}{2}(d-2)}{|S|} + \frac{n(d-1)}{|S|})$. □

We can also replace the call to Algorithm 2 in Algorithm 3 by a call to Algorithm 4 to similarly get a reconstruction algorithm for linear combination of powers of at most n linearly independent linear forms. More specifically, given a polynomial f in blackbox, it will check if there exists a decomposition of $f = \sum_{i=1}^{t} \alpha_i l_i^d$ for some $t \leq n$ where l_i's are linearly independent and $\alpha_i \neq 0$ and outputs the decomposition, if it exists.

2.6 Complexity analysis for equivalence to a sum of cubes

We first explain how the diagonalizability of a matrix can be tested efficiently with an algebraic algorithm. This can be done thanks to the following classical result from linear algebra (see e.g. [HJ13], Corollary 3.3.8 for the case $\mathbb{K} = \mathbb{C}$).

Lemma 2.6.1. *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* $P_M = \frac{\chi_M}{gcd(\chi_M, \chi'_M)}$ *be the square-free part of* χ_M*. The matrix* M *is diagonalisable over* $\overline{\mathbb{K}}$ *iff* $P_M(M) = 0$*. Moreover, in this case* M *is diagonalisable over* $\mathbb{K}$ *iff all the roots of* P_M *lie in* $\mathbb{K}$*.*

We prove the following lemma which shows that each entry of the slices that we need to compute can be computed using $O(d)$ calls to the blackbox and $O(d \log^3 d)$ many arithmetic operations. This proof is motivated from the idea of polynomial

interpolation and the proof strategy of Lemma 4 in [FGS18].Their algorithm gives a poly(sd) runtime in our setting. In this section, we will require this lemma only for $d = 3$, but we prove the general form so that we can use it later in Chapter 2.7.

Lemma 2.6.2. *Let $f \in \mathbb{K}[x_1, ..., x_n]$ be a homogeneous polynomial of degree d where $|\mathbb{K}| > d$. If f is input as a blackbox C, then for some $i \in [n]$ can compute the coefficient of $x_i^{d-2}x_kx_j$ using $O(d)$ many oracle calls to the blackbox and $O(M(d)\log d)$ many arithmetic operations.*

Proof. Here we use the standard trick of polynomial interpolation. Without loss of generality, we assume that $i = 1$, that is we need to compute $\text{coeff}_{x_1^{d-2}x_jx_k}(f)$. So we can write,

$$C(x_1, ..., x_n) = \sum_{i=0}^{d} c_i x_1^i \text{ where } c_j \in \mathbb{K}[x_2, ..., x_n].$$

Now there are three cases:

- $j = k = 1$
- only one of j or $k = 1$
- $j, k \neq 1$.

Case 1: Evaluate the polynomial at the point $(1, 0, ..., 0) \in \mathbb{K}^n$. This gives us the coefficient of x_1^d in f.
Case 2: Exactly one of j or k is 1. Without loss of generality, we assume $j = 1$ and $k = 2$. So we want to compute $\text{coeff}_{x_1^{d-1}x_2}(f)$. We evaluate the polynomial f at the point $\bar{t} = (t, 1, 0, ..., 0) \in \mathbb{K}^n$. Now it's easy to check that $\text{coeff}_{t^{d-1}}(f(\bar{t})) = \text{coeff}_{x_1^{d-1}x_2}(f)$. So we need to only interpolate and calculate the coefficient of t^{d-1} in $f(\bar{t})$.

Case 3: $j, k \neq 1$.
Now in this, there are two cases :

- $j = k = 2$: Here we take a similar strategy as Case 2. We evaluate f at $\bar{t} = (t, 1, 0, ..., 0) \in \mathbb{K}^n$. Then $\text{coeff}_{t^{d-2}}(f(\bar{t})) = \text{coeff}_{x_1^{d-2}x_2^2}(f)$ So interpolate and compute the coefficient of t^{d-2} in $f(\bar{t})$.
- The final case is when the indices are all distinct. Let $j = 2$ and $k = 3$ without loss of generality. We evaluate the polynomial f at $\bar{t} = (t, 1, 1, 0, ..., 0) \in \mathbb{K}^n$. Now

  $$\text{coeff}_{t^{d-2}}(f(\bar{t})) = \text{coeff}_{x_1^{d-2}x_2^2}(f) + \text{coeff}_{x_1^{d-2}x_2x_3}(f) + \text{coeff}_{x_1^{d-2}x_3^2}(f)$$

  Now, using the previous case, we compute $\text{coeff}_{x_1^{d-2}x_2^2}(f)$ and $\text{coeff}_{x_1^{d-2}x_3^2}(f)$, subtract them from $\text{coeff}_{t^{d-2}}(f(\bar{t}))$ and return the answer.

Each case of this algorithm requires us to do univariate polynomial interpolation at most constantly many number of times and this can be done in time $O(M(d)\log d)$ [GG13] (Section 10.2) where $M(d)$ is the number of arithmetic operations for polynomial multiplication. Rest of the operations can be done in time $O(1)$. It also requires $O(d)$ many oracle calls to the blackbox.
So the algorithm uses $O(M(d)\log d)$ many arithmetic operations and $O(d)$ many oracle calls to the blackbox. □

Theorem 2.6.3. *If a degree 3 form $f \in \mathbb{K}[x_1, ..., x_n]$ is given in dense representation, Algorithm 1 runs in time $O(n^{\omega+1})$ where ω is the exponent of matrix multiplication. If the degree 3 form $f \in \mathbb{K}[x_1, ..., x_n]$ is given as a blackbox then the algorithm makes $O(n^2)$ many calls to the blackbox and $O(n^{\omega+1})$ many arithmetic operations.*

Proof. The following are the different stages of computation required in this algorithm:

1. Recall from Theorem 2.2.2, the slices T_i of $h = f(Rx)$ are given by the formula $T_k = R^T(\sum_{i\in[n]} r_{i,k}S_i)R$. If the polynomial is input in dense representation, then the elements of S_i can be computed from the coefficients of f. Then we take linear combinations of the S_i's and computing T_1, T_2, T_3 takes $O(n^3)$ many arithmetic operations.
   If the polynomial is given as a blackbox, we compute $x' = Rx$ and we call the blackbox on this input.

2. **Compute** T_1, T_2, T_3 We know
$$(T_k)_{ij} = \frac{1}{3!}\partial_{x_i x_j x_k}(h)$$
   So we can extract each entry of T_k using constant many calls to the blackbox and constantly many arithmetic operations using Lemma 2.6.2. There are in total $3n^2$ such entries that we need to compute. So the total number of calls to the blackbox is $O(n^2)$ and the number of arithmetic operations is $O(n^2)$.

3. **Check if T_1 is invertible. If invertible, compute** $T_1' = T_1^{-1}$.
   This can be done in time at most $O(n^3)$. (Faster algorithms exist [GG13] but this bound is enough since it is not the most expensive step of the algorithm.)

4. **Checking commutativity of** $T_1'T_2$ and $T_1'T_3$.
   Here we compute the product $T_1'T_2T_1'T_3$ and $T_1'T_3T_1'T_2$ and check if their difference is 0. This can be done in time $O(n^\omega)$.

5. **Checking the diagonalisability of** $T_1'T_2$:
   Here we use Lemma 2.6.1. Hence there are four steps:

   - Compute the characteristic polynomial of M i.e. χ_M. Owing to a recent breakthrough by [NP21], there is a deterministic algorithm for this problem that runs in time $O(n^\omega)$. A randomized algorithm for this problem with same running time was given by [PS07]. A more classical result is a deterministic algorithm for this problem due to [KG85] which runs in $O(n^\omega \log(n))$ number of arithmetic operations.
   - Compute $\gcd(\chi_M, \chi_M',)$. Since, $deg(\chi_M) \leq n$, this can be done in $O(n^2)$ using Euclidean Algorithm. [GG13]
   - Compute $P_M = \frac{\chi_M}{\gcd(\chi_M, \chi_M')}$. This is can be computed in $O(n^2)$ using the standard long-division algorithm.
   - Check if $P_M(M) = 0$. Using Horner's Method, we can evaluate the polynomial at M using n many matrix multiplications only. Hence, computing $P_M(M)$ takes $O(n^{\omega+1})$ time.

   Hence, we can conclude that the diagonalisability of $T_1'T_2$ can be checked in time $O(n^{\omega+1})$. Note that this is the most expensive step of the algorithm!

So we conclude that if the polynomial is given as an input in the dense representation model, then the algorithm runs in time $O(n^{\omega+1})$.
If the polynomial is given as a blackbox, then the algorithm makes $O(n^2)$ many oracle calls to the blackbox and takes $O(n^{\omega+1})$ many arithmetic operations. □

2.7 Complexity analysis for equivalence to some polynomial in $\mathcal{P}_d$

2.7.1 Complexity Analysis in the algebraic model

In this section, we provide a detailed complexity analysis of Algorithm 2. We show that if a degree d polynomial in n variables over $\mathbb{C}$ is given as a blackbox, the algorithm makes $\text{poly}(n,d)$ many calls to the blackbox and performs $\text{poly}(n,d)$ many arithmetic operations to decide if f is equivalent to some polynomial in $\mathcal{P}_d$.

Theorem 2.7.1. *If a degree-d form $f \in \mathbb{C}[x_1,...,x_n]$ is given as a blackbox, then Algorithm 2 makes $O(n^2 d)$ many calls to the blackbox and requires $O(n^2 d \log^2 d \log\log d + n^{\omega+1})$ many arithmetic operations.*

Proof. The following are the different stages of computation required in this algorithm:

1. If the polynomial is given as a blackbox, we compute $x' = Rx$. And we call the blackbox on this input.
2. Compute $T_{\bar{1}}, T_{\bar{2}}, T_{\bar{3}}$. We know that $(T_{\bar{k}})_{ij} = \frac{1}{d!}\partial_{x_i x_j x_k^{d-2}}(h)$.So we can extract each entry of $T_{\bar{i}}$ using $O(d)$ many oracle calls to C' and $O(M(d)\log d)$ many arithmetic operations using Lemma (2.6.2). There are in total $3n^2$ such entries that we need to compute. So this entire operation can be done using $O(n^2 d)$ many oracle calls to the blackbox and $O(n^2 M(d)\log d)$ many arithmetic operations.
3. **Check if $T_{\bar{1}}$ is invertible. If invertible, compute $T'_{\bar{1}} = (T_{\bar{1}})^{-1}$**
4. **Checking commutativity of $T'_{\bar{1}}T_{\bar{2}}$ and $T'_{\bar{1}}T_{\bar{3}}$.**
5. **Checking the diagonalisability of $T'_{\bar{1}}T_{\bar{2}}$:**

Steps (3), (4) and (5) are exactly the same as Theorem 2.6.3. This is because they don't require any assumptions on $T_{\bar{1}}, T_{\bar{2}}, T_{\bar{3}}$ except for the fact that they are $n \times n$ matrices. Thus, from the proof of Theorem 2.6.3, we get that these steps can be checked in $O(n^{\omega+1})$ many arithmetic operations.

So we conclude that if the polynomial is given as a blackbox, then the algorithm makes $O(n^2 d)$ many calls to the blackbox; the number of arithmetic operations required is $O(n^2 M(d)\log d + n^{\omega+1})$.
Since $M(d) = O(d \log d \log\log d)$, the number of arithmetic operations is $O(n^2 d \log^2 d \log\log d + n^{\omega+1})$.

2.7.2 Complexity analysis for the bit model

We look at the case where $f \in \mathbb{Q}[x_1,...,x_n]$ is a degree-d form and we want to check if it can be written as a linear combination of dth powers of linear forms over $\mathbb{R}$ or $\mathbb{C}$. 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). For a detailed discussion of how the previous algorithms fail to give a polynomial time algorithm for this problem in this model, refer to Section 1.1 from [KS21].

We try to estimate the complexity of each step of the algorithm in the standard bit model of computation, following the proof of Theorem 2.7.1. In Step (1), we take a matrix R such that its entries $r_{i,j}$ are picked uniformly and independently at random from a finite set S.Hence the bit size of the entries of R are bounded by $\log(|S|)+1$. We define $h = f(Rx)$. Recall from Theorem 2.3.2, that the slices $T_{\bar{i}}$ of h can be written as $R^T(\sum_{i_1,\ldots,i_{d-2}\in[n]}(\prod_{m\in[d]} r_{i_m,i})S_{i_1\ldots i_{d-2}})R$. The entries of the slices $S_{i_1\ldots i_{d-2}}$ are essentially the coefficients of f. So they are bounded by the bit size of the maximum coefficient in f which we define to be b_f. Therefore, the bit size of each element of $T_{\bar{i}}$ is $b := \text{poly}(\log(|S|), \log(n), d, b_f)$ for all i. Now the elements of $T_{\bar{1}}, T_{\bar{2}}, T_{\bar{3}}$ are computed using Lemma 2.6.2 that uses polynomial interpolation and hence, computing these matrices takes time $\text{poly}(n, d, b)$. In Step (3), we check if the slice $T_{\bar{1}}$ is invertible and if invertible, it is inverted. Since the bit-size of the inputs of $T_{\bar{1}}$ are bounded by b, the matrix can be inverted using Bareiss' Algorithm [Bar68] in time $\text{poly}(n, d, b)$. In Step (4), testing commutativity of $T'_{\bar{1}}T_{\bar{2}}$ and $T'_{\bar{1}}T_{\bar{3}}$ requires only matrix multiplication which does not blow up the entry of the matrices and hence, this step can be done in time $\text{poly}(n, d, b)$. In Step (5), we need to check the diagonalisability of $M = T'_{\bar{1}}T_{\bar{2}}$. Over the field of complex numbers it therefore suffices to check that $P_M(M) = 0$ which can be done in time $\text{poly}(n, d, b)$. For the discussion of how the same can be executed over $\mathbb{R}$, refer to Section 4 of [KS21].

So the total time required for the entire computation in the bit model of complexity is $\text{poly}(n, d, log(|S|), b_f)$ where b_f is the bit size of the maximum coefficient in f and S is the set from which the entries of R are picked uniformly and independently at random. □

Chapter 3

Numerical Linear Algebra

This chapter is dedicated to the study of different fundamental algorithms for linear algebra, albeit in the finite precision arithmetic model of computation. In Section 3.1.1, we further explore the model of finite precision arithmetic (which we had already described in Section 1.2.2) and in Section 3.1, we present the error guarantees for well-known fast algorithms for standard linear algebra problems like matrix multiplication and matrix inversion. Then, in Section 3.2, we give a linear time algorithm for computing the trace of the slices of a symmetric tensor after a *change of basis* operation. We also perform the error analysis of this algorithm in the finite precision arithmetic model of computation. In Section 3.3, we prove some interesting properties of the fast and *numerically stable* diagonalisation algorithm from [BGVKS22]. These are key components of the numerical algorithm for tensor decomposition that we will describe in Chapter 4.

3.1 Preliminaries: Fast and Stable Linear Algebra

In this section, we explore the computational model of finite precision arithmetic that has already been introduced in Section 1.2.2 in greater detail. We present the different estimates for various linear algebraic operations such as inner product of vectors, matrix multiplication and matrix inversion. This is the main content of Sections 3.1.1 and 3.1.2 and they have been taken from [Hig02] and [BGVKS22]. We include these for completeness of the exposition.

3.1.1 Finite precision arithmetic

Recall that we had described the finite precision arithmetic model of computation in Section 1.2.2. Every number $x \in \mathbb{C}$ is stored as $\mathrm{fl}(x) = (1+\Delta)x$ for some adversarially chosen $\Delta \in \mathbb{C}$, satisfying $|\Delta| \leq \mathbf{u}$ where $\mathbf{u}$ is the precision of the finite arithmetic machine.

Norms: We denote by $||x||$ the ℓ^2 (Hermitian) norm of a vector $x \in \mathbb{C}^n$. For $A \in M_n(\mathbb{C})$, we denote by $||A||$ its operator norm and by $||A||_F$ its Frobenius norm:

$$||A||_F^2 = \sum_{i,j=1}^{n} |A_{ij}|^2. \tag{3.1}$$

We always have $||A|| \leq ||A||_F$.

We'll need to compute the inner product of two vectors $x, y \in \mathbb{C}^n$. For this purpose, we will assume that

$$|x^T y - \mathrm{fl}(x^T y)| \leq \gamma_n ||x|| ||y|| \tag{3.2}$$

where $\mathbf{u}$ is the machine precision and $\gamma_n = \frac{n\mathbf{u}}{1-n\mathbf{u}}$. For a proof, refer to the discussion at the discussion in [Hig02], Section 3.1.

We will also assume similar guarantees for matrix-matrix addition and matrix-scalar multiplication. More specifically, if $A \in \mathbb{C}^{n\times n}$ is the exact output of such an operation, then its floating point representation $\text{fl}(A)$ will satisfy

$$\text{fl}(A) = A + A \circ \Delta \text{ where } |\Delta_{ij}| < \mathbf{u}.$$

Here $A \circ \Delta$ denotes the entry-wise product $A_{ij}\Delta_{ij}$. This multiplicative error can be converted into an additive form i.e.

$$||A \circ \Delta|| \leq \mathbf{u}\sqrt{n}||A||. \tag{3.3}$$

For more complicated linear algebraic operations like matrix multiplication and matrix inversion, we require more sophisticated error guarantees which we now explain.

3.1.2 Matrix Multiplication and Inversion

The definitions we state here are taken from [BGVKS22]

Definition 3.1.1. *A $\mu_{MM}(n)$-stable multiplication algorithm $MM(.,.)$ takes as input $A, B \in \mathbb{C}^{n\times n}$ and a precision $\boldsymbol{u} > 0$ and outputs $C = MM(A, B)$ satisfying*

$$||C - AB|| \leq \mu_{MM}(n) \cdot \boldsymbol{u}||A||||B||$$

on a floating point machine with precision $\boldsymbol{u}$, in $T_{MM}(n)$ arithmetic operations.

Definition 3.1.2. *A $(\mu_{INV}(n), c_{INV})$-stable inversion algorithm $INV(.)$ takes as input $A \in \mathbb{C}^{n\times n}$ and a precision $\boldsymbol{u}$ and outputs $C = INV(A)$ satisfying*

$$||C - A^{-1}|| \leq \mu_{INV}(n).\boldsymbol{u}.(\kappa(A))^{c_{INV}\log n}||A^{-1}||.$$

on a floating point machine with precision $\boldsymbol{u}$, in $T_{INV}(n)$ arithmetic operations.

The following theorem by [DDHK07] gives a numerically stable matrix multiplication algorithm which is used by [DDH07] to gives numerically stable algorithm for matrix inversion and a numerically stable algorithm for QR factorization of a given matrix. We use the presentation of these theorems from [BGVKS22].

Theorem 3.1.3. *1. If ω is the exponent of matrix multiplication, then for every $\eta > 0$, there is a $\mu_{MM}(n)$-stable matrix multiplication algorithm with $\mu_{MM}(n) = n^{c_\eta}$ and $T_{MM}(n) = O(n^{\omega+\eta})$, where c_η does not depend on n.*

2. Given an algorithm for matrix multiplication satisfying part (1), there is a $(\mu_{INV}(n), c_{INV})$-stable inversion algorithm with

$$\mu_{INV}(n) \leq O(\mu_{MM}(n)n^{\log 10}) \text{ and } c_{INV} \leq 8,$$

and $T_{INV}(n) = O(T_{MM})(n)$.

In particular, all of the running times above are bounded by $T_{MM}(n)$ for a $n \times n$ matrix.

Instead of the fast matrix multiplication algorithm, one can also consider the errors from the conventional computation. Let A, B be two matrices and let $C = AB$

computed on a floating point machine with machine precision $\mathbf{u}$. From (3.13) in [Hig02], we have that

$$||C - AB|| \leq 2n\mathbf{u}||A||_F||B||_F. \tag{3.4}$$

We will use this bound in the next section, where (in contrast to Section 3.3) fast matrix multiplication is not needed.

3.2 Slices after a change of basis

Given tensors $T, T' \in \mathbb{C}^{n\times n\times n}$, we say that there is a change of basis $A \in \mathrm{M}_n(\mathbb{C})$ that takes T to T' if $T' = (A \otimes A \otimes A).T$. Written in standard basis notation, the equality $T' = (A \otimes A \otimes A).T$ corresponds to the fact that for all $i_1, i_2, i_3 \in [n]$,

$$T'_{i_1 i_2 i_3} = \sum_{j_1,j_2,j_3\in[n]} A_{j_1 i_1} A_{j_2 i_2} A_{j_3 i_3} T_{j_1 j_2 j_3}. \tag{3.5}$$

Note that if $T = u^{\otimes 3}$ for some vector $u \in \mathbb{C}^n$, then $(A \otimes A \otimes A).T = (A^T u)^{\otimes 3}$.

The choice of making A act by multiplication by A^T rather than by multiplication by A is somewhat arbitrary, but it is natural from the point of view of the polynomial-tensor equivalence in Section 1.1.3. Indeed, from the polynomial point of view a change of basis corresponds to a linear change of variables. More precisely, if $f(x_1, \ldots, x_n)$ is the polynomial associated to T (refer to Section 1.1.3) and $f'(x_1, \ldots, x_n)$ is the polynomial associated to $T' = (A \otimes A \otimes A).T$, we have $f'(x) = f(Ax)$.

In the present section we give a fast and numerically stable algorithm for computing the trace of the slices after a change of basis. More formally, given a tensor T and a matrix V, it computes $Tr(S_1), ..., Tr(S_n)$ where $S_1, ..., S_n$ are the slices of the tensor $S = (V \otimes V \otimes V).T$ with small error in $O(n^3)$ many arithmetic operations. The following theorem was derived in Theorem 2.2.2 in the polynomial language of Section 1.1.3.

Theorem 3.2.1. *Let $T \in (\mathbb{C}^n)^{\otimes 3}$ be a tensor with slices $T_1, ..., T_n$ and let $S = (A \otimes A \otimes A).T$ where $A \in M_n(\mathbb{C})$. Then the slices $S_1, ..., S_n$ of S are given by the formula:*

$$S_k = A^T D_k A$$

where $D_k = \sum_{i=1} a_{i,k} T_i$ and $a_{i,k}$ are the entries of A.

Corollary 3.2.1.1. *Let $S = \sum_{i=1}^r a_i^{\otimes 3}$. Let A be the $r \times n$ matrix with rows $a_1, ..., a_r$. Then the slices S_k of S are given by the formula*

$$S_k = A^T D_k A \text{ where } D_k = diag(a_{1,k}, ..., a_{r,k}).$$

Definition 3.2.2 (Tensor Norm)**.** *Given a tensor $T \in (\mathbb{C}^n)^{\otimes 3}$, we define the Frobenius norm $||T||_F$ of T as*

$$||T||_F = \sqrt{\sum_{i,j,k=1}^{n} |T_{i,j,k}|^2}$$

Then if $T_1, ..., T_n$ are the slices of T, we also have that

$$\sum_{i=1}^{n} ||T_i||_F^2 = \sum_{j,k\in[n]} |(T_i)_{j,k}|^2 = ||T||_F^2. \tag{3.6}$$

Algorithm 5: Trace of the slices after a change of basis (TSCB)

Input: An order-3 symmetric tensor $T \in \mathbb{C}^{n\times n\times n}$, a matrix $V = (v_{ij}) \in \mathbb{C}^{n\times n}$.

Let $T_1, ..., T_n$ be the slices of T.

1 Compute $W = V^T V$ on a floating point machine.

2 Compute $x_{m,k} = (WT_m)_{k,k}$ on a floating point machine for all $m, k \in [n]$.

3 Compute $x_m = \sum_{k=1}^{n} x_{m,k}$ on a floating point machine for all $m \in [n]$.

4 Compute $\tilde{s}_i = \sum_{m=1}^{n} v_{m,i} x_m$ on a floating point machine for all $i \in [n]$.

Output $\tilde{s}_1, ..., \tilde{s}_n$

Wherever we mention that the computation is done on a floating point machine, we assume that there is an adversarial error associated with that computation. The following is the main theorem of this section.

Theorem 3.2.3. *Let us assume that a tensor $T \in (\mathbb{C}^n)^{\otimes 3}$ and a matrix $V \in M_n(\mathbb{C})$ are given as input to Algorithm 5. Set $S = (V \otimes V \otimes V).T$ following the definition in (3.5) and let $S_1, ..., S_n$ be the slices of S. Then the algorithm returns $\tilde{s}_1, ..., \tilde{s}_n$ such that*

$$|\tilde{s}_i - Tr(S_i)| \leq \mu_{CB}(n) \cdot \boldsymbol{u} \cdot ||V||_F^3 ||T||_F \tag{3.7}$$

where $\mu_{CB}(n) \leq 14n^{\frac{3}{2}}$. It performs $T_{CB}(n) = O(n^3)$ operations on a machine with precision $\boldsymbol{u} < \frac{1}{10n}$.

Proof. Let $S' \in \mathbb{C}^{n\times n\times n}$ be such that $S' = (V \otimes V \otimes V).T$. Let $S'_1, ..., S'_n$ be the slices of S'. We first claim that $\sum_{m=1}^{n} v_{mi} \left(\sum_{k=1}^{n} (V^T V T_m)_{k,k} \right) = Tr(S'_i)$. Using Theorem 3.2.1, we know that $S'_i = V^T D_i V$ where $D_i = \sum_{m=1}^{n} v_{m,i} T_m$. Now using the cyclic property and the linearity of the trace operator, we get that

$$\begin{aligned} Tr(S'_i) = Tr(V^T D_i V) = Tr(V^T V D_i) = Tr(V^T V (\sum_{m=1}^{n} v_{m,i} T_m)) \\ = \sum_{m=1}^{n} v_{mi} Tr(V^T V T_m) = \sum_{m=1}^{n} v_{mi} \left(\sum_{k=1}^{n} (V^T V T_m)_{k,k} \right). \end{aligned} \tag{3.8}$$

From this, we conclude that if Algorithm 5 is run in exact arithmetic, it computes exactly the trace of the slices S'_i of S'.

Running Time: We analyse the steps of the algorithm and deduce the number of arithmetic operations required to perform the algorithm. Note that only the numbered steps contribute to the complexity analysis.

1. Since $V \in M_n(\mathbb{C})$, Step 1 can be done in $O(n^3)$ operations with ordinary matrix multiplication.

2. In Step 2, for each $m, k \in [n]$, we compute the inner product of the k-th row of W with the k-th column of T_m. Computation of each inner product takes n arithmetic operations. There are n^2 such inner product computations. So this step requires n^3 arithmetic operations.

3. In Step 3, we compute each x_m by adding $x_{m,k}$ for all $k \in [n]$. Thus each x_m requires n arithmetic operations and hence, this step requires n^2 arithmetic operations.

4. In Step 4, we compute each $\tilde{s}_i$ by taking the inner product of the i-th column of V and $X = (X_1, ..., x_m)$. Each inner product requires n arithmetic operations and hence, this step requires n^2 arithmetic operations.

So, the total number of arithmetic operations required is $T_{CB}(n) = O(n^3)$.

Error Analysis: We denote by A_k the k-th row of any matrix A and by $A_{\_,k}$ we denote the k-th column of A.

We proceed step by step and analyse the errors at every step of the algorithm. At every step, we explain what the ideal output would be if the algorithm was run in exact arithmetic. And we show that the output in finite arithmetic at every stage is quite close to the ideal output.

Step 1: Let V be the matrix given as input. In this step, we want to compute a product of the matrices V^T and V. We use the standard matrix multiplication algorithm and the bounds from (3.4). Let $W = MM(V^T, V)$ be the output of Step 1 of this algorithm.

Using (3.4) and the fact that for any matrix V, $||V^T|| = ||V||$, we have:

$$||W - V^T V|| \leq 2n \cdot \mathbf{u} \cdot ||V||_F^2. \tag{3.9}$$

From (3.9) and the triangle inequality, we also have that

$$||W|| \leq ||V||_F^2 + 2n\mathbf{u}||V||_F^2 < 2||V||_F^2. \tag{3.10}$$

In the last inequality, we use the hypothesis that $2n\mathbf{u} < 1$.

Step 2: In this step, we take as input a matrix W and compute all the diagonal elements of the matrix WT_m. Let $x_{m,k} = (WT_m)_{k,k}$ be computed on a floating point machine. If the algorithm is run in exact arithmetic, the output at the end of Step 2 is $(V^T V T_m)_{k,k}$ for all $m, k \in [n]$.

Computationally, the k-th diagonal element can be computed as an inner product between the k-th row of W and the k-th column of T_m. Then using the error bounds of inner product computation in (3.2), we have that

$$|x_{m,k} - (WT_m)_{k,k}| \leq 2n\mathbf{u}||W_k||||(T_m)_{\_,k}|| \leq 2n\mathbf{u}||W||||(T_m)_{\_,k}||. \tag{3.11}$$

Also, from (3.9), we have that

$$\begin{aligned} |(WT_m)_{k,k} - (V^T V T_m)_{k,k}| &\leq |\langle (W - V^T V)_k, (T_m)_{\_,k} \rangle| \\ &\leq ||(W - V^T V)_k||||(T_m)_{\_,k}|| \\ &\leq 2n\mathbf{u}||V||_F^2||(T_m)_{\_,k}||. \end{aligned} \tag{3.12}$$

Combining (3.11) and (3.12), the triangle inequality and the bound from (3.10), we deduce that

$$|x_{m,k} - (V^T V T_m)_{k,k}| \leq 6n\mathbf{u}||V||_F^2||(T_m)_{\_,k}||. \tag{3.13}$$

We also want to give an upper bound for $|x_{m,k}|$. By (3.13) and the triangle inequality, we have

$$|x_{m,k}| \leq 6n\mathbf{u}||V||_F^2||(T_m)_{\_,k}|| + |(V^T V T_m)_{k,k}| \tag{3.14}$$

So we need to give an upper bound for $|(V^TVT_m)_{k,k}|$. Expanding along the definition and using the Cauchy-Schwarz inequality, we obtain

$$|(V^TVT_m)_{k,k}| = |\langle (V^TV)_k, (T_m)_{\_,k}\rangle| \leq ||(V^TV)_k||||(T_m)_{\_,k}||. \tag{3.15}$$

Putting this back in (3.14), we have that

$$|x_{m,k}| \leq \Big(6n\mathbf{u} + 1\Big)||V||_F^2||(T_m)_{\_,k}|| \leq 2||V||_F^2||(T_m)_{\_,k}||. \tag{3.16}$$

The final inequality uses the hypothesis that $\mathbf{u} < \frac{1}{6n}$.

Step 3: In this step, we take as input $x_{m,k}$ for all $m, k \in [n]$. We then compute $x_m = \sum_{k=1}^n x_{m,k}$ on a floating point machine for all $m \in [n]$. If the algorithm was run in exact arithmetic, the output at the end of this step would be $\sum_{k=1}^n (V^TVT_m)_{k,k}$ for all $m \in [n]$.

Computation of x_m can also be thought of as inner product between the all 1's vector $\bar{1}$ and the vector $(x_{m,1}, ..., x_{m,n})$. So, we can again use the bounds from (3.2). This gives us that

$$\begin{aligned} |x_m - \sum_{k=1}^n x_{m,k}| &\leq 2n^{\frac{3}{2}}\mathbf{u}\sqrt{\sum_{k=1}^n |x_{m,k}|^2} \\ &\leq 4n^{\frac{3}{2}}||V||_F^2\mathbf{u}\sqrt{\sum_{k=1}^n ||(T_m)_{\_,k}||^2}. \end{aligned} \tag{3.17}$$

The last equation uses (3.16) to bound the norm of $|x_{m,k}|$. Also, summing up (3.13) for all $k \in [n]$ and using the triangle inequality, we have that

$$\begin{aligned} |\sum_{k=1}^n x_{m,k} - \sum_{k=1}^n (V^TVT_m)_{k,k}| &\leq \mathbf{u}||V||_F^2(4n + \mu_{MM}(n))\Big(\sum_{k=1}^n ||(T_m)_{\_,k}||\Big) \\ &\leq \mathbf{u}||V||_F^2(6n^{\frac{3}{2}})\Big(\sum_{k=1}^n ||(T_m)_{\_,k}||^2\Big). \end{aligned} \tag{3.18}$$

The last inequality follows from the Cauchy-Schwarz inequality. Combining (3.17) and (3.18), we finally have the error at the end of that

$$\begin{aligned} |x_m - \sum_{k=1}^n (V^TVT_m)_{k,k}| &\leq \mathbf{u}||V||_F^2 10n^{\frac{3}{2}}\sqrt{\sum_{k=1}^n ||(T_m)_{\_,k}||^2} \\ &= \mathbf{u}||V||_F^2 10n^{\frac{3}{2}}||T_m||_F. \end{aligned} \tag{3.19}$$

In the last equality, we use the definition of the Frobenius norm of matrices from (3.1).

We also want to derive bounds for $|x_m|$. From the previous equation, by the triangle inequality we already get that

$$|x_m| \leq \mathbf{u}||V||_F^2 10n^{\frac{3}{2}}||T_m||_F + |\sum_{k=1}^n (V^TVT_m)_{k,k}|. \tag{3.20}$$

So it is enough to derive bounds for $|\sum_{k=1}^{n}(V^TVT_m)_{k,k}|$. Summing up (3.15) for all $m \in [n]$ and using the Cauchy-Schwarz inequality, we obtain:

$$\begin{aligned} |\sum_{k=1}^{n}(V^TVT_m)_{k,k}| &\leq \sqrt{n}\sqrt{\sum_{k=1}^{n}|(V^TVT_m)_{k,k}|^2} \\ &\leq \sqrt{n}||V||_F^2||T_m||_F. \end{aligned} \tag{3.21}$$

Putting this back in (3.20), we have that

$$|x_m| \leq 2\sqrt{n}||V||_F^2||T_m||_F. \tag{3.22}$$

Here in the last inequality, we use the hypothesis that $\mathbf{u} \leq \frac{1}{10n}$.

Step 4: In this step, we take as input x_m for all $m \in [n]$. We then compute $\tilde{s}_i = \sum_{m=1}^{n} v_{mi}x_m$ in floating point arithmetic. Recall that $S = (V \otimes V \otimes V).T$ and $S_1, ..., S_n$ are the slices of S. Ideally if the algorithm is run in exact arithmetic, the output at this stage is $Tr(S_i) = \sum_{m=1}^{n} v_{mi}\Big(\sum_{k=1}^{n}(V^TVT_m)_{k,k}\Big)$.

Using error bounds for the inner product operation (3.2) and using (3.22) to bound $|x_m|$ we have that

$$\begin{aligned} |\tilde{s}_i - \sum_{m=1}^{n} v_{mi}x_m| &\leq 2n\mathbf{u}||V_{\_,i}||\sqrt{\sum_{m=1}^{n}|x_m|^2} \\ &\leq 4n^{\frac{3}{2}}\mathbf{u}||V||_F^3\sqrt{\sum_{m=1}^{n}||T_m||_F^2}. \end{aligned} \tag{3.23}$$

Also, summing up (3.19) for all $m \in [n]$ and using the triangle inequality, we have:

$$\begin{aligned} |\sum_{m=1}^{n} v_{mi}\Big(x_m - \sum_{k=1}^{n}(V^TVT_m)_{k,k}\Big)| &\leq ||V_{\_,i}||\sqrt{\sum_{m=1}^{n}|x_m - \sum_{k=1}^{n}(V^TVT_m)_{k,k}|^2} \\ &\leq \mathbf{u}||V||_F^3(8n^{\frac{3}{2}} + \sqrt{n}\mu_{MM}(n))\sqrt{\sum_{m=1}^{n}||T_m||_F^2} \end{aligned} \tag{3.24}$$

Moreover, it follows from (3.6), that $\sum_{m=1}^{n}||T_m||_F^2 = ||T||_F^2$. Using this and combining (3.23) and (3.24) using triangle inequality, we have:

$$\begin{aligned} &||\tilde{s}_i - \sum_{m=1}^{n} v_{mi}\Big(\sum_{k=1}^{n}(V^TVT_m)_{k,k}\Big)| \\ &\leq \mathbf{u}||V||_F^3 14n^{\frac{3}{2}}\sqrt{\sum_{m=1}^{n}||T_m||_F^2} \\ &= \mu_{CB}(n) \cdot \mathbf{u} \cdot ||V||_F^3||T||_F, \end{aligned} \tag{3.25}$$

where $\mu_{CB}(n) \leq 14n^{\frac{3}{2}}$. □

3.3 Diagonalisation algorithm for diagonalisable matrices

In this section, we look at the algorithmic problem of matrix diagonalisation over the field of complex numbers.

Definition 3.3.1 (**Eigenpair and eigenproblem**). *[BGVKS22] An eigenpair of a matrix $A \in \mathbb{C}^{n\times n}$ is a tuple $(\lambda, v) \in \mathbb{C} \times \mathbb{C}^n$ such that $Av = \lambda v$ and $||v||_2 = 1$. The eigenproblem is the problem of finding a maximal set of linearly independent eigenpairs (λ_i, v_i) of a given matrix A. Note that an eigenvalue may appear more than once if it has geometric multiplicity greater than one. In the case when A is diagonalizable, the solution consists of exactly n eigenpairs, and if A has distinct eigenvalues, then the solution is unique, up to multiplication by phases of v_i. This is multiplication by complex numbers of modulus* 1.

Due to the Abel–Ruffini theorem, it is impossible to have a finite-time algorithm which solves the eigenproblem exactly using arithmetic operations and radicals. So one can only hope to find approximate eigenvalues and eigenvectors, up to a desired accuracy.

Definition 3.3.2 (δ-**forward approximation for the eigenproblem**). *Let (λ_i, v_i) be true eigenpairs for a diagonalizable matrix A. Given an accuracy parameter δ, the problem is to find pairs (λ'_i, v'_i) such that $||v_i - v'_i|| \leq \delta$ and $|\lambda_i - \lambda'_i| \leq \delta$ i.e., to find a solution close to the exact solution.*

Definition 3.3.3 (δ-**backward approximation for the eigenproblem**). *Given a diagonalizable matrix A and an accuracy parameter δ, find exact eigenpairs (λ'_i, v'_i) for a matrix A' such that $||A - A'|| \leq \delta$ i.e., find an exact solution to a nearby problem. Since diagonalizable matrices are dense in $\mathbb{C}^{n\times n}$, one can always find a complete set of eigenpairs for some nearby A'.*

The eigenproblem has been thoroughly studied in different models of computation. Without an attempt at being exhaustive, we will try to include a brief survey of the results. The problem of devising an algorithm for the general eigenproblem that was numerically stable remained open until the breakthrough by Armentano, Beltrán, Bürgisser, Cucker, and Shub [ABB+18]. They gave an algorithm which on input any matrix A with $||A|| \leq 1$, outputs a δ-backward approximation to the eigenproblem (refer to Definition 3.3.3) in $\frac{n^{10}}{\delta^2}$ arithmetic operations. Although the analysis of the algorithm in this paper has been performed for exact arithmetic, the authors argue informally that the homotopy continuation methods used in the algorithm are numerically stable and can be implemented in finite precision arithmetic. This result was further improved in [BGVKS22] who gave a numerically stable algorithm for matrix diagonalisation that also runs in nearly matrix multiplication time and the number of arithmetic operations has a polylogarithmic dependence on $\frac{1}{\delta}$ (refer to Theorem 3.3.4 for a precise statement) as compared to [ABB+18] which has dependence on $\frac{1}{\delta}$ in the finite precision arithmetic model. This is the algorithm whose properties we discuss in this section and this serves as one of the important building blocks of our tensor decomposition algorithm, which is described in Chapter 5. In comparison, the diagonalisation problem for Hermitian matrices has been well understood since the work of [Wil68] (refer to [BGVKS22] for a more detailed survey of the existing results.) In the model of rational arithmetic with bounded bit length a, [Cai94] gave an algorithm to compute a δ-forward approximation to the Jordan normal form in $\text{poly}(n, a, \log(\frac{1}{\delta}))$ arithmetic operations.

Contributions of this section to matrix diagonalisation: Given $A \in M_n(\mathbb{C})$ and $\delta > 0$, the algorithm for matrix diagonalisation in [BGVKS22] computes an invertible matrix V and a diagonal matrix D such that $||A - VDV^{-1}|| \leq \delta$. Moreover, V is guaranteed to be reasonably well-conditioned in the sense that $||V||.||V^{-1}|| = O(n^{2.5}/\delta)$ (refer to Theorem 3.3.4 for a precise statement). Note however that V might become arbitrarily poorly conditioned as δ goes to 0. The main question that we address in this section is the following: Can we have a better guarantee on V assuming that the input matrix A is diagonalisable? In Theorem 3.3.4, we show that this is indeed the case, under the additional assumption that the eigenvalues are distinct. The bounds in that theorem are expressed as a function of the condition number of the eigenproblem (3.29), already defined in [BGVKS22], and of the Frobenius eigenvector condition number (3.28).

Another issue that we address in Section 3.3 is the assumption $||A|| \leq 1$ on the input matrix. Relaxing this assumption in infinite precision arithmetic is very straightforward: given a bound $B \geq 1$ on $||A||$, one can simply divide A by B and this does not change the eigenvectors. In finite arithmetic, however, this simple scaling leads to round-off errors. In particular, the error analysis due to the scaling of A is worked out in the proof of Theorem 3.3.12.

Condition numbers. If A is diagonalizable, we define following [BGVKS22] its *eigenvector condition number*:

$$\kappa_V(A) = \inf_V ||V|| \cdot ||V^{-1}||, \tag{3.26}$$

where the infimum is over all invertible V such that $V^{-1}AV$ is diagonal. Its minimum eigenvalue gap is defined as

$$\text{gap}(A) := \min_{i \neq j} |\lambda_i(A) - \lambda_j(A)|, \tag{3.27}$$

where λ_i are the eigenvalues of A (with multiplicity). Instead of the eigenvector condition number, it is sometimes more convenient to work instead with the *Frobenius eigenvector condition number*

$$\kappa_V^F(A) = \inf_V (||V||_F^2 + ||V^{-1}||_F^2) = \inf_V \kappa_F(V), \tag{3.28}$$

where the infimum is taken over the same set of invertible matrices. We always have $\kappa_V^F(A) \geq 2\kappa_V(A)$. Following [BGVKS22], we define the condition number of the eigenproblem to be:

$$\kappa_{\text{eig}}(A) := \frac{\kappa_V(A)}{\text{gap}(A)} \in [0, \infty]. \tag{3.29}$$

For a given invertible matrix V, we define the *Frobenius condition number* to be

$$\kappa_F(V) = ||V||_F^2 + ||V^{-1}||_F^2. \tag{3.30}$$

The following is the main theorem of this section. As mentioned before, in this theorem, we give some properties of the diagonalisation algorithm *EIG* analyzed in [BGVKS22]. The first two are from their paper, and the third one provides an additional conditioning guarantee for V. It is especially useful for small values of δ.

Theorem 3.3.4. *There is a randomized algorithm EIG which on input a diagonalisable matrix $A \in \mathbb{C}^{n \times n}$ with $||A|| \leq 1$ and a desired accuracy parameter $\delta \in (0, \frac{1}{8\kappa_{eig}(A)})$ outputs a diagonal matrix D and an invertible matrix V. The following properties are satisfied by the output matrices:*

1. $||A - VDV^{-1}|| \leq \delta$ *and* $\kappa(V) \leq \frac{32n^{2.5}}{\delta}$.
2. $||v_i|| = 1 \pm n\boldsymbol{u}$ *where* v_i *are the columns of* V.
3. $\kappa(V) \leq \frac{\kappa_F(V)}{2} \leq \frac{1}{2}(\frac{9n}{4} + 9n^2(\kappa_V^F(A))^2)$.

The algorithm runs in

$$O(T_{MM}(n)\log^2\frac{n}{\delta})$$

arithmetic operations on a floating point machine with

$$\log(\frac{1}{u}) = O(\log^4(\frac{n}{\delta})\log n)$$

bits of precision with probability at least $1 - \frac{1}{n} - \frac{12}{n^2}$.

In the rest of this section, we give some definitions and prove some lemmas leading to the proof of this theorem.

Lemma 3.3.5. *Suppose that* A *has* n *distinct eigenvalues* $\lambda_1, \ldots, \lambda_n$, *with* $v_1, \ldots, v_n$ *the corresponding eigenvectors. Let* W *be the matrix with columns* $v_1, \ldots, v_n$; *let* $u_1, \ldots, u_n$ *be the left eigenvectors of* A, *i.e., the rows of* W^{-1}. *Then* $\kappa_V^F(A) = 2\sum_{i=1}^n ||u_i|| \cdot ||v_i||$, *and the infimum in (3.28) is reached for the matrix* V *obtained from* W *by multiplication of each column by* $\sqrt{||u_i||/||v_i||}$.

Proof. Since A has distinct eigenvalues, any matrix V that diagonalizes A is obtained from W by multiplication of each column by some nonzero scalar x_i. In matrix notation, we have $V = WD$ where $D = \mathrm{diag}(x_1, ..., x_n)$. We also have $V^{-1} = D^{-1}W^{-1}$, and the i-th row of V^{-1} is therefore equal to u_i/x_i. As a result,

$$||V||_F^2 + ||V^{-1}||_F^2 = \sum_{i=1}^n \left(x_i^2||v_i||^2 + \frac{||u_i||^2}{x_i^2} \right).$$

An elementary computation shows that the infimum is reached for $x_i = \sqrt{||u_i||/||v_i||}$. Here we have assumed that $x_i \in \mathbb{R}$ for all i. This is without loss of generality since multiplying each entry of V or V^{-1} by a complex number of modulus 1 does not change their Frobenius norms. □

If M is diagonalisable as VDV^{-1} over $\mathbb{C}$, let v_i be the columns of V and u_j^T be the rows of V^{-1}. Then M admits a spectral expansion of the form

$$M = \sum_{i=1}^n \lambda_i v_i u_i^*. \tag{3.31}$$

Definition 3.3.6. *For* $M \in M_n(\mathbb{C})$, *if* M *has distinct eigenvalues* $\lambda_1, ..., \lambda_n$ *and a spectral expansion as in (3.31), then we define the eigenvalue condition number of* λ_i *as*

$$\kappa(\lambda_i) := ||v_i u_i^*|| = ||v_i||||u_i||$$

Remark 3.3.7. *Note that* $\kappa(\lambda_i)$ *is independent of the choice of the* v_i*'s. This follows from the fact that* M *has distinct eigenvalues: if* v_i', u_i' *is another pair of vectors corresponding to* λ_i *in the spectral expansion, then* $v_i' = k_i v_i$ *and* $u_i' = l_i v_i$ *for some non-zero constants* k_i, l_i. *Using the fact that* $\langle v_i', u_i' \rangle = 1$, *we have* $k_i l_i = 1$. *Hence* $||u_i'||||v_i'|| = ||u_i||||v_i||$ *which proves that* $\kappa(\lambda_i)$ *is indeed independence of the choice of the* v_i*'s.*

Lemma 3.3.8. *Let M, M' be $n \times n$ matrices such that $||M||, ||M'|| \leq 1$ and $||M - M'|| \leq \delta$ where $\delta < \frac{1}{8\kappa_{eig}(M)}$. Let $\lambda_1, ..., \lambda_n$ be the distinct eigenvalues of M. Then*

1. *M' has distinct eigenvalues.*
2. *$|\kappa(\lambda_i) - \kappa(\lambda_i')| \leq 2\kappa_V(A)$.*
3. *$\sqrt{n \sum_{i\in[n]} \kappa(\lambda_i)^2} \leq n\kappa_V(A)$.*

Proof. Refer to the proof of Proposition 1.1 in [BGVKS22] for a proof of the first two properties. Towards the third one, we first show that $\kappa_V(A) \geq \kappa(\lambda_i)$ for all $i \in [n]$. Using the definition of $\kappa_V(A)$, we get that

$$\begin{aligned}\kappa_V(A) &= \inf_{V\in D(A)} ||V||||V^{-1}|| \\ &= \inf_{V\in D(A)} (\max_{x\in\mathbb{C}^n} \frac{||Vx||}{||x||})(\max_{x\in\mathbb{C}^n} \frac{||V^{-1}x||}{||x||}) \\ &\geq \inf_{V\in D(A)} ||v_i||||u_i||\end{aligned}$$

where v_i are the columns of V and u_i are the rows of V^{-1}. The inequality follows from the fact that $\max_{x\in\mathbb{C}^n} \frac{||Vx||}{||x||} \geq ||Ve_i|| = ||v_i||$. Since $||V|| = ||V^T||$ for all matrices V, $\max_{x\in\mathbb{C}^n} \frac{||V^{-1}x||}{||x||} = \max_{x\in\mathbb{C}^n} \frac{||V^{-T}x||}{||x||} \geq ||V^{-T}e_i|| = ||u_i||$. By Remark 3.3.7, $||v_i||||u_i||$ is equal for all $V \in D(A)$. As a result, $\kappa_V(A) \geq ||v_i||||u_i|| = \kappa(\lambda_i)$ for all $i \in [n]$. This gives us

$$\sqrt{n \sum_{i\in[n]} \kappa(\lambda_i)^2} \leq \sqrt{n \sum_{i\in[n]} \kappa_V(A)^2} = n\kappa_V(A).$$

□

Lemma 3.3.9. *Let $A, A' \in M_n(\mathbb{C})$ be such that A has n distinct eigenvalues and $||A - A'|| \leq \delta$ where $\delta < \frac{1}{8\kappa_{eig}(A)}$. Then*

$$\kappa_V^F(A') \leq 6n\kappa_V(A) \leq 3n\kappa_V^F(A).$$

Proof. We first explain the proof for $||A||, ||A'|| \leq 1$ and then modify it to deal with the general case. Suppose that $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A with corresponding eigenvectors $v_1, \ldots, v_n$. By Lemma 3.3.8, A' has distinct eigenvalues as well. Let $\lambda_1', \ldots, \lambda_n'$ be the eigenvalues of A' with corresponding eigenvectors $v_1', \ldots, v_n'$. Let W' be the matrix with columns $v_1', \ldots, v_n'$; let $u_1', \ldots, u_n'$ be the left eigenvectors of A', i.e., the rows of W^{-1}. Applying Lemma 3.3.5 to A', we know that

$$\kappa_V^F(A') = 2\sum_{i=1}^{n} ||u_i'|| \cdot ||v_i'||. \tag{3.32}$$

From Lemma 3.3.8, we know that

$$|\kappa(\lambda_i') - \kappa(\lambda_i)| \leq 2\kappa_V(A)$$

and hence

$$\kappa(\lambda_i') \leq \kappa(\lambda_i) + 2\kappa_V(A).$$

Adding this up for all $i = 1$ to n,

$$\sum_{i=1}^{n} \kappa(\lambda_i') \leq \sum_{i=1}^{n} \kappa(\lambda_i) + 2n\kappa_V(A).$$

Now using the definition of $\kappa(\lambda_i)$ as in Definition 3.3.6, we get that

$$\sum_{i=1}^{n} ||v_i'||||u_i'|| \leq \sum_{i=1}^{n} \kappa(\lambda_i) + 2n\kappa_V(A)$$

By the Cauchy-Schwarz inequality, we have

$$\sum_{i=1}^{n} ||v_i'||||u_i'|| \leq \sqrt{n \sum_{i\in[n]} \kappa(\lambda_i)^2} + 2n\kappa_V(A).$$

By (3.32) and Lemma 3.3.8, we get that

$$\kappa_V^F(A') \leq 6n\kappa_V(A) \leq 3n\kappa_V^F(A).$$

Let us now take A, A' to be arbitrary $n \times n$ matrices over $\mathbb{C}$ such that $||A - A'|| \leq \delta$ for some $\delta \in (0, \frac{1}{8\kappa_{\text{eig}}}(A))$. Let $N_0 := \max\{||A||, ||A'||\}$. Then we can define $C = \frac{C}{N_0}$ and $C' = \frac{A'}{N_0}$ where $||C||, ||C'|| \leq 1$. Also notice that $||C - C'|| \leq \delta' = \frac{\delta}{N_0}$ where $\delta' < \frac{1}{8N_0\kappa_{\text{eig}}(A)}$. Now, $\kappa_{\text{eig}}(C) = \frac{\kappa_V(C)}{\text{gap}(C)}$. Since $C = \frac{A}{N_0}$, we get that $\text{gap}(C) = \frac{\text{gap}(A)}{N_0}$ and $\kappa_V(C) = \kappa_V(A)$. Hence, $\kappa_{\text{eig}}(C) = N_0\kappa_{\text{eig}}(A)$ and this gives us $\delta' < \frac{1}{8\kappa_{\text{eig}}(C)}$. Using the previous argument, we have $\kappa_V^F(C') \leq 6n\kappa_V(C) \leq 3n\kappa_V^F(C)$. Since scaling of matrices preserves κ_V^F and κ_V, this gives us that $\kappa_V^F(A') \leq 6n\kappa_V(A) \leq 3n\kappa_V^F(A)$. □

Lemma 3.3.10. *Let $A \in M_n(\mathbb{C})$ be a diagonalisable matrix with distinct eigenvalues and let $A = VDV^{-1}$ such that for all $i \in [n]$, for each column v_i of V, $\Big|\,||v_i|| - 1\Big| \leq \delta$. Then $\kappa_F(V) \leq n(1+\delta)^2 + \frac{(\kappa_V^F(A))^2}{4(1-\delta)^2}$.*

Proof. By Lemma 3.3.5, if $U = V^{-1}$ and $u_1, ..., u_n$ are the rows of U, then $\kappa_V^F(A) = \sum_{i\in[n]} 2||u_i||||v_i||$. Since $|||v_i|| - 1| \leq \delta$ for all $i \in [n]$, we have that $(1-\delta)\sum_{i\in[n]} 2||u_i|| \leq \kappa_V^F(A) \leq (1+\delta)\sum_{i\in[n]} 2||u_i||$. From the definition of κ_F,

$$\begin{aligned}\kappa_F(V) = ||V||_F^2 + ||V^{-1}||_F^2 &= n(1+\delta)^2 + \sum_{i\in[n]} ||u_i||^2 \\ &\leq n(1+\delta)^2 + (\sum_{i\in[n]} ||u_i||)^2 \leq n(1+\delta)^2 + \frac{(\kappa_V^F(A))^2}{4(1-\delta)^2}.\end{aligned}$$

□

We are now ready to complete the proof of Theorem 3.3.4.

Proof of Theorem 3.3.4. The first two properties are from [BGVKS22] (see in particular Theorem 1.6 for the first one). From the second property and from Lemma 3.3.10 applied to $A' = VDV^{-1}$ for $\delta = n\mathbf{u}$, we have

$$\kappa(V) \leq \frac{\kappa_F(V)}{2} \leq \frac{1}{2}(n(1+n\mathbf{u})^2 + \frac{(\kappa_F(A'))^2}{4(1-n\mathbf{u})^2}).$$

Since $\delta < \frac{1}{8\kappa_{\text{eig}}(A)}$, it follows from Lemma 3.3.9 that $\kappa_V^F(A') \leq 3n\kappa_V^F(A)$ and this gives us that $\kappa(V) \leq \frac{\kappa_F(V)}{2} \leq \frac{1}{2}(n(1+n\mathbf{u})^2 + \frac{(9n^2\kappa_V^F(A))^2}{4(1-n\mathbf{u})^2})$. Since $n\mathbf{u} < \frac{1}{2}$, this implies that

$$\kappa(V) \leq \frac{\kappa_F(V)}{2} \leq \frac{1}{2}(\frac{9n}{4} + 9n^2(\kappa_V^F(A))^2).$$

□

In the remainder of this section, we relax the hypothesis $||A|| \leq 1$ on the input matrix.

Theorem 3.3.11. *[BGVKS22] If $||A||, ||A'|| \leq 1$, $||A - A'|| \leq \delta$, $\delta < \frac{gap(A)}{8\kappa_V(A)}$ and $\{(v_i, \lambda_i)\}_{i\in[n]}$, $\{(v_i', \lambda_i')\}_{i\in[n]}$ are eigenpairs of A, A' , then*

$$||v_i - v_i'|| \leq 6n\kappa_{eig}(A)\delta \text{ and } ||\lambda_i' - \lambda_i|| \leq \kappa_V(A)\delta \leq 2\kappa_{eig}(A)\delta \text{ for all } i \in [n],$$

where v_i's are given up to multiplication by phases.

Note here that by "phases" we mean complex numbers of norm 1.

Corollary 3.3.11.1. *For matrices $A, A' \in M_n(\mathbb{C})$, if $||A - A'|| \leq \delta$, $\{(v_i, \lambda_i)\}_{i\in[n]}$, $\{(v_i', \lambda_i')\}_{i\in[n]}$ are eigenpairs of A, A' respectively and $\delta < \frac{gap(A)}{8\kappa_V(A)}$, then*

$$||v_i - v_i'|| \leq 6n\kappa_{eig}(A)\delta \text{ and } |\lambda_i - \lambda_i'| \leq \kappa_V(A)\delta \text{ for all } i \in [n]$$

where the v_i's are given up to multiplication by phases.

Proof. Let $N_0 = \max\{||A||, ||A'||\}$. Let $C = \frac{A}{N_0}$ and $C' = \frac{A'}{N_0}$. Then $||C||, ||C'|| < 1$ and taking $\delta' = \frac{\delta}{N_0}$, we get that $||C - C'|| \leq \delta'$ where $\delta' < \frac{1}{8N_0\kappa_{\text{eig}}(A)} = \frac{1}{8\kappa_{\text{eig}}(C)}$. Applying Theorem 3.3.11, we have $||u_i - u_i'|| \leq 6n\kappa_{\text{eig}}(C)\delta'$ where the u_i are the eigenvectors of C after possibly multiplying u_i by phases. Using $\kappa_{\text{eig}}(C) = N_0\kappa_{\text{eig}}(A)$ gives us that $||u_i - u_i'|| \leq 6n\kappa_{\text{eig}}(A)\delta$. Since the eigenvectors remain unchanged after scaling the matrix by a constant, this implies that $||v_i - v_i'|| \leq 6n\kappa_{\text{eig}}(A)\delta$.

If $\mu_1, ..., \mu_n$ are the corresponding eigenvalues of C and $\mu_1', ..., \mu_n'$ are the corresponding eigenvalues of C', then we get that $|\mu_i - \mu_i'| < \kappa_V(C)\delta'$. Since $C = \frac{A}{N_0}$ and $C' = \frac{A'}{N_0}$, this implies that $\mu_i = \frac{\lambda_i}{N_0}$ and $\mu_i' = \frac{\lambda_i'}{N_0}$ for all $i \in [n]$. Hence, multiplying both sides by N_0 gives us that $N_0|\mu_i - \mu_i'| < \kappa_V(C)N_0\delta'$, hence $|\lambda_i - \lambda_i'| < \kappa_V(C)\delta$. Since $\kappa_V(C) = \kappa_V(A)$, we finally conclude that $|\lambda_i - \lambda_i'| < \kappa_V(A)\delta$. □

We now present the algorithm for computing a forward approximation to the eigenvectors of a diagonalisable matrix.

Algorithm 6: Forward approximation of the eigenvectors of a matrix (EIG-FWD)

Input: A diagonalisable matrix A with distinct eigenvalues, estimates $K_{\text{norm}} > \max\{||A||_F, 1\}$ and $K_{\text{eig}} > \kappa_{\text{eig}}(A)$, desired accuracy parameter δ.

1. Compute $B' = \frac{A}{2K_{\text{norm}}}$ on a floating point machine.
2. Compute $\delta' = \frac{\delta}{64nK_{\text{norm}}K_{\text{eig}}}$.
3. Let (W, D_0) be the output of $\text{EIG}(B', \delta')$.
4. Output the columns $w_1, ..., w_n$ of W.

Here we assume at step 2 that the parameter δ' is computed without any roundoff error. As in [BGVKS22], this will be done for simplicity throughout the thesis in computations whose size does not grow with n. In the next theorem, we state some properties of Algorithm 6.

Theorem 3.3.12. *Given a diagonalisable matrix $A \in M_n(\mathbb{C})$, a desired accuracy parameter $\delta \in (0, \frac{1}{2})$ and estimates $K_{norm} > \max\{||A||_F, 1\}$ and $K_{eig} > \kappa_{eig}(A)$ as input, Algorithm 6 outputs vectors $w_1, ..., w_n \in \mathbb{C}^n$ such that the following properties are satisfied with probability at least $1 - \frac{1}{n} - \frac{12}{n^2}$:*

- *If $v_1^{(0)}, ..., v_n^{(0)}$ are the true normalized eigenvectors of A, then we have $||v_i^{(0)} - w_i|| < \delta$ up to multiplication by phases.*
- *Let W be the matrix with columns $w_1, ..., w_n$. Then*

$$\kappa(W) \leq \frac{\kappa_F(W)}{2} \leq \frac{1}{2}(\frac{9n}{4} + 81n^4(\kappa_V^F(A))^2).$$

The algorithm requires

$$O(T_{MM}(n) \log^2 \frac{nK_{eig}K_{norm}}{\delta})$$

arithmetic operations on a floating point machine with

$$O(\log^4(\frac{nK_{eig}K_{norm}}{\delta}) \log n)$$

bits of precision.

Remark 3.3.13. *The proof of this theorem only incorporates the error due to scaling of the matrices and requires some relatively routine and technical calculations. Hence this has been relegated to Appendix B.*

Chapter 4

Numerical Algorithm for Tensor Decomposition

This chapter is dedicated to the following algorithmic problem: Given a diagonalisable tensor, find the unique decomposition of the tensor. As discussed in Section 1.4.5, we give a randomized linear-time and numerically stable algorithm to solve the problem *approximately*. In this section, we state the algorithm in the standard algebraic model of computation assuming infinite precision and give a correctness proof of that. We then present the algorithm in the finite-precision arithmetic model of computation. The correctness proof of the algorithm and its probabilistic analysis will be given in Chapters 5 and 6 respectively.

4.1 Introduction

4.1.1 Simplified Algorithm

Before giving a high-level presentation of our algorithm, we introduce a few notations. A symmetric tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ can be cut into n slices $T_1, \ldots, T_n$ where $T_k = (T_{ijk})_{1 \leq i,j \leq n}$. Each slice is a symmetric matrix of size n. In the algorithm below we also make use of a "change of basis" operation, which applies a linear map of the form $A \otimes A \otimes A$ to a tensor. Here, $A \in M_n(\mathbb{C})$ and we apply A to the 3 components of the input tensor. In particular, for rank-1 symmetric tensors we have

$$(A \otimes A \otimes A).(u \otimes u \otimes u) = (A^T u)^{\otimes 3}. \tag{4.1}$$

We give more details on this operation at the beginning of Section 3.2. The algorithm proceeds as follows.

(i) Pick vectors $a = (a_1, ..., a_n)$ and $b = (b_1, ..., b_n)$ at random from a finite set and compute two random linear combinations $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$ of the slices of T.

(ii) Diagonalise $(T^{(a)})^{-1} T^{(b)} = V D V^{-1}$. Let $v_1, ..., v_n$ be the columns of V.

(iii) Let $u_1, ..., u_n$ be the rows of V^{-1}.

(iv) Let $T' = (V \otimes V \otimes V).T$. Let $T'_1, ..., T'_n$ be the slices of T'. Define $\alpha_i = \mathrm{Tr}(T'_i)$. We will refer to the computation of $Tr(T'_i)$ as the trace of slices after a change of basis (TSCB).

(v) Output $(\alpha_1)^{\frac{1}{3}} u_1, ..., (\alpha_n)^{\frac{1}{3}} u_n$.

The above algorithm is a modified version of Jennrich's algorithm for symmetric tensors. In terms of algorithm design, our main contribution lies in step (iv). Previous

versions of Jennrich's algorithm have appealed instead to the resolution of a linear system of equations: see e.g. [BCMV14, Moi18] for the case of ordinary tensors. In the symmetric case, the algebraic algorithm in [Kay11] for decomposition of a polynomial as a sum of powers of linear forms also appeals to the resolution of a linear system for essentially the same purpose. Our trace-based version of step (iv) is more efficient, and this is crucial for the derivation of the complexity bounds in Theorem 4.3.4. Step (iv) is indeed the most expensive: it is responsible for the $O(n^3)$ term in the arithmetic complexity of the algorithm. We explain informally at the beginnining of Section 4.1.3 why our trace-based approach works.

4.1.2 Organization of the chapter

In Section 4.2, we state the above algorithm in more detail and show that if this algorithm is given a complete diagonalisable tensor exactly as input, it indeed returns the (unique) decomposition. In the underlying computational model assumed for the analysis in that section, all arithmetic operations can be done exactly and matrices can be diagonalised exactly. This is the algorithm that we will adapt to the finite arithmetic model in Section 4.3.

4.1.3 Ideas for algorithm design

Recall that we had defined in Section 1.4.2 the notion of diagonalisable tensors.

Definition 4.1.1. *We call a symmetric tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \times \mathbb{C}^n$ diagonalisable if $T = \sum_{i=1}^n u_i^{\otimes 3}$ where the u_i are linearly independent.*

In other words, a symmetric tensor T is diagonalisable if there exists an invertible change of basis U that takes the diagonal tensor $\sum_{i\in[n]} e_i^{\otimes 3}$ to T, i.e., $T = (U \otimes U \otimes U).(\sum_{i\in[n]} e_i^{\otimes 3})$ for some $U \in \mathrm{GL}_n(\mathbb{C})$.

In this section we outline the main ingredients required for the design of the algorithm and the proof strategy as well.

Trace of the slices after a change of basis

After step (iii) of the algorithm, we have determined matrix V^{-1} with rows $u_1, \ldots, u_n$ such that $T = \sum_{i=1}^n \alpha_i u_i^{\otimes 3}$. Here the α_i are unknown coefficients. As explained at the beginning of this chapter, the traditional approach is to find them by solving the corresponding linear system. One difficulty here is that this system is highly overdetermined: we have one equation for each entry of T, but only n unknowns. We show that the system *can* be solved quickly in a numerical stable way by exploiting some of its structural properties. Our approach relies on a change of basis (recall the definition from Section 3.2). More precisely, let $T' = (V \otimes V \otimes V).T$ be the tensor defined at the beginning of step (iv). We know that the tensor T can be written as $T = (V^{-1} \otimes V^{-1} \otimes V^{-1}).\left(\sum_{i=1}^n \alpha_i e_i^{\otimes 3}\right)$. Since $u_1, \ldots, u_n$ are the rows of V^{-1}, it follows that $T' = (V \otimes V \otimes V).T = \sum_{i=1}^n \alpha_i e_i^{\otimes 3}$ where e_i is the i-th standard basis vector. Therefore we can read off the α_i from the entries of T'. This observation is not sufficient to obtain the desired running time since it is not clear how to perform a change of basis in $O(n^3)$ arithmetic operations. Indeed, since a symmetric tensor of size n has $\Omega(n^3)$ coefficients, one would have to perform a constant number of operations per coefficient. A further observation is that we do not need to compute every entry of T': assuming that T is diagonalisable, we know in advance that all entries of T' except the diagonal ones will be equal to 0 (up to rounding errors). As

a result, α_i is approximately equal to the trace of T'_i, the i-the slice of T'.Recall that in Section 3.2 we have already given a fairly simple algorithm for the computation of these n traces in $O(n^3)$ arithmetic operations. For this we do not even need to assume that the input tensor is diagonalisable. We also analyse this algorithm in finite arithmetic in the same section. The correctness of our main algorithm in exact arithmetic (as presented at the beginning of this section) is established in Section 4.2 based on the results of Section 3.2.

4.2 Tensor decomposition for complete symmetric tensors in exact arithmetic

Let $T \in (\mathbb{C}^n)^{\otimes 3}$ be a diagonalisable tensor given as input. In this section we give an algorithm which returns the linearly independent u_i's in the decomposition of T, up to multiplication by cube roots of unity. This algorithm works in a computational model where all arithmetic operations can be done exactly and additionally, we can diagonalise a matrix exactly.

Algorithm 7: Tensor decomposition algorithm for complete symmetric tensors

Input: An order-3 diagonalisable symmetric tensor $T \in \mathbb{C}^{n\times n\times n}$.
Output: linearly independent vectors $l_1, ..., l_r \in \mathbb{C}^n$ such that $T = \sum_{i=1}^n l_i^{\otimes 3}$.
Pick $a_1, ..., a_n$ and $b_1, ..., b_n$ uniformly and independently at random from a finite set $S \subset \mathbb{C}$
Let $T_1, ..., T_n$ be the slices of T.
1 Set $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$.
2 Compute $T^{(a)'} = (T^{(a)})^{-1}$.
3 Compute $D = T^{(a)'} T^{(b)}$.
4 Compute the normalized eigenvectors $p_1, ..., p_n$ of D.
5 Let P be the matrix with $(p_1, ..., p_n)$ as columns and compute P^{-1}. Let v_i be the i-th row of P^{-1}.
6 Define $S = (P \otimes P \otimes P).T$ and let $S_1, ..., S_n$ be the slices of S. Compute $\alpha_i = \mathrm{Tr}(S_i)$.
7 Output $(\alpha_1)^{\frac{1}{3}} v_1, ..., (\alpha_n)^{\frac{1}{3}} v_n$.

Algorithm 7 is essentially the algorithm that was already presented in Section 4.1.3. As explained in that section, this is a symmetric version of Jennrich's algorithm where the coefficients α_i are determined in a novel way (as the traces of slices of a certain tensor). The algorithm will be analyzed in finite precision arithmetic in the following section and the final two chapters of this thesis.

The remainder of this section is devoted to a correctness proof for Algorithm 7 including an analysis of the probability of error. The main theorem of this section is Theorem 4.2.2. In that direction, we prove an intermediate lemma showing that if $a_1, ..., a_n$ and $b_1, ..., b_n$ are picked at random from a finite set, then $T^{(a)}$ is invertible and the eigenvalues of $(T^{(a)})^{-1} T^{(b)}$ are distinct with high probability.

Theorem 4.2.1. *Let $T = \sum_{i=1}^n u_i^{\otimes 3}$ where $u_i \in \mathbb{C}^n$ are linearly independent vectors. Let $T_1, ..., T_n$ be the slices of T. Set $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$ where $a_1, ..., a_n, b_1, ..., b_n$ are picked uniformly and independently at random from a finite set $S \subset \mathbb{K}$. If $T^{(a)}$ is invertible, let $T^{(a)'} = (T^{(a)})^{-1}$. Let $\lambda_1, ..., \lambda_n$ be the eigenvalues of*

$T^{(a)'}T^{(b)}$. *Then*

$$Pr_{a_1,...,a_n,b_1,...,b_n\in_r S}[T^{(a)} \text{ is invertible and } \lambda_i \neq \lambda_j \text{ for all } i \neq j] \geq 1 - (\frac{2\binom{n}{2}}{|S|} + \frac{n}{|S|}).$$

Proof. Let U be the matrix with columns $u_1, ..., u_n$. Since $T = \sum_{i=1}^n u_i^{\otimes 3}$, by Corollary 3.2.1.1, the slices $T_1, ..., T_n$ of T can be written as

$$T_i = U^T D_i U \text{ where } D_i = \text{diag}(u_{1,i}, ..., u_{n,i}).$$

Taking $a = (a_1, ..., a_n) \in \mathbb{C}^n$, this gives us that

$$T^{(a)} = U^T D^{(a)} U \text{ where } D^{(a)} = \text{diag}(\langle a, u_1\rangle, ..., \langle a, u_n\rangle).$$

Similarly, $T^{(b)} = U^T D^{(b)} U$ where $D^{(b)} = \text{diag}(\langle b, u_1\rangle, ..., \langle b, u_n\rangle)$. Now if $T^{(a)}$ is invertible, we can write

$$T^{(a)'}T^{(b)} = U^{-1}\left(\text{diag}(\frac{\langle b, u_1\rangle}{\langle a, u_1\rangle}, ..., \frac{\langle b, u_n\rangle}{\langle a, u_n\rangle})\right)U \tag{4.2}$$

Hence, the eigenvalues of $T^{(a)'}T^{(b)}$ are $\lambda_i = \frac{\langle b,u_i\rangle}{\langle a,u_i\rangle}$. For all $i \neq j \in [n]$, we define the polynomial

$$P_{ij}(x_1, ..., x_n, y_1, ..., y_n) = \langle y, u_i\rangle\langle x, u_j\rangle - \langle y, u_j\rangle\langle x, u_i\rangle = \sum_{k,l=1}^n y_k x_l (u_{ik}u_{jl} - u_{il}u_{jk})$$

where $y = (y_1, ..., y_n)$ and $x = (x_1, ..., x_n)$. Now $T^{(a)}$ is invertible iff $\langle a, u_i\rangle \neq 0$ for all $i \in [n]$. This gives us that

$$\text{Pr}_{a_1,...,a_n\in_r S}[T^{(a)} \text{ is invertible}] = \text{Pr}_{a_1,...,a_n\in_r S}[\langle a, u_i\rangle \neq 0 \text{ for all } i \in [n]]. \tag{4.3}$$

For all $i \in [n]$, there exists $k \in [n]$ such that $u_{ik} \neq 0$. Hence

$$\text{Pr}_{a_1,...,a_n\in_r S}[\langle a, u_i\rangle = 0] \leq \frac{1}{|S|}$$

by Lemma 2.2.9, and then

$$\text{Pr}_{a_1,...,a_n\in_r S}[T^{(a)} \text{ is invertible}] \geq 1 - \frac{n}{|S|} \tag{4.4}$$

by the union bound. Also, if $T^{(a)}$ is invertible, then $\lambda_i = \lambda_j$ if and only if $P_{ij}(a, b) = 0$. Written as a probabilistic statement, this gives us that

$$\begin{aligned}&\text{Pr}_{a,b\in_r S}[T^{(a)} \text{ is invertible and for all } i \neq j, \lambda_i \neq \lambda_j]\\ &= \text{Pr}_{a,b\in_r S}[T^{(a)} \text{ is invertible and for all } i \neq j, P_{ij}(a, b) \neq 0].\end{aligned} \tag{4.5}$$

Since U is an invertible matrix, its rows are pairwise linearly independent and for all $i, j \in [n]$, there must exist some k_0, l_0 such that $(u_{ik_0}u_{jl_0} - u_{il_0}u_{jk_0}) \neq 0$. Hence, taking $a = e_{k_0}$ and $b = e_{l_0}$ (where e_i denotes the vector with 1 at the i-th position and 0 otherwise), we get that $P_{ij}(e_{k_0}, e_{l_0}) \neq 0$. Hence, $P_{ij} \not\equiv 0$ and

$$\text{Pr}_{a,b\in_r S}[P_{ij}(a, b) \neq 0] \geq 1 - \frac{2}{|S|}$$

by Lemma 2.2.9 since $\deg(P_{ij}) \leq 2$.

Applying the union bound, we then have

$$\Pr_{a,b\in_r S}[\text{For all } i \neq j \in [n], P_{ij}(a,b) \neq 0] \geq 1 - \frac{2\binom{n}{2}}{|S|}.$$

Finally, using (4.4) and (4.5), we have that

$$\begin{aligned}
&\Pr_{a,b\in_r S}[T^{(a)} \text{ is invertible and } \lambda_i \neq \lambda_j \text{ for all } i \neq j] \\
&= \Pr_{a,b\in_r S}[T^{(a)} \text{ is invertible and for all } i \neq j, P_{ij}(a,b) \neq 0] \\
&\geq 1 - (\frac{2\binom{n}{2}}{|S|} + \frac{n}{|S|}).
\end{aligned}$$

□

Theorem 4.2.2. *If an input tensor $T \in (\mathbb{C}^n)^{\otimes 3}$ can be written as $T = \sum_{i=1}^n u_i^{\otimes 3}$ where the $u_i \in \mathbb{C}^n$ are linearly independent vectors, then Algorithm 7 succeeds with high probability. More formally, if $a_1, ..., a_n, b_1, ..., b_n$ are chosen uniformly and independently at random from a finite subset $S \subset \mathbb{C}$, then the algorithm returns linearly independent $l_1, ..., l_n \in \mathbb{C}^n$ such that $T = \sum_{i=1}^n l_i^{\otimes 3}$ with probability at least $1 - (\frac{2\binom{n}{2}}{|S|} + \frac{n}{|S|})$.*

Proof. First, using Theorem 4.2.1 we get that if $a_1, ..., a_n, b_1, ..., b_n$ are picked uniformly and independently at random from a finite subset $S \subset \mathbb{K}$, then $T^{(a)}$ is invertible and the eigenvalues of $T^{(a)'}T^{(b)}$ are distinct with probability at least $1 - (\frac{2\binom{n}{2}}{|S|} + \frac{n}{|S|})$.

Now we show that if $T^{(a)}$ is invertible and the eigenvalues of $T^{(a)'}T^{(b)}$ are distinct, then Algorithm 7 returns linearly independent vectors $l_1, ..., l_n \in \mathbb{C}^n$ such that $T = \sum_{i=1}^n (l_i)^{\otimes 3}$.

If the eigenvalues of a matrix are distinct, then the dimension of the eigenspaces corresponding to each eigenvalue is 1. Hence, the eigenvectors of $T^{(a)'}T^{(b)}$ are unique (up to a scaling factor). From (4.2), we get that the columns of U^{-1} are eigenvectors of $T^{(a)'}T^{(b)}$. Since the columns of P are also eigenvectors of $T^{(a)'}T^{(b)}$, this gives us the relation that there exists a permutation matrix P_π such that

$$P = U^{-1}P_\pi D \text{ where } D = \text{diag}(k_1, ..., k_n) \text{ and } k_i \neq 0. \tag{4.6}$$

Now we claim that the α_i's computed in Step 6 of Algorithm 7 are exactly equal to k_i^3. Let $S = (P \otimes P \otimes P).T$ and let $S_1, ..., S_n$ be the slices of S. Following (4.1), we know that the tensor T can be written as $T = \sum_{i=1}^n u_i^{\otimes 3} = \sum_{i=1}^n (U^T e_i)^{\otimes 3} = \sum_{i=1}^n ((P_\pi^T U)^T e_i)^{\otimes 3}$. Then

$$\begin{aligned}
\alpha_i = Tr(S_i) = \sum_{j=1}^n S_{i,j,j} &= \sum_{j=1}^n \left((P \otimes P \otimes P).(\sum_{t=1}^n (U^T P_\pi e_t)^{\otimes 3}) \right)_{i,j,j} \\
&= \sum_{j,t=1}^n \left((P^T U^T P_\pi e_t)^{\otimes 3} \right)_{i,j,j} \\
&= \sum_{j,t=1}^n ((D^T P_\pi^T U^{-T} U^T P_\pi e_t)^{\otimes 3})_{i,j,j} \\
&= \sum_{j,t=1}^n (D^T e_t)_i (D^T e_t)_j^2 = (D^T e_i)_i^3 = k_i^3.
\end{aligned} \tag{4.7}$$

Since U, D and P_π are all invertible, P is invertible as well and $P_\pi D P^{-1} = U$. Putting this in vector notation, if v_i are the rows of P^{-1}, then $u_i = k_{\pi(i)} v_{\pi(i)}$. As a result, for any cube root of unity ω_i,

$$T = \sum_{i=1}^{n} (u_i)^{\otimes 3} = \sum_{i=1}^{n} (\omega_i k_{\pi(i)} v_{\pi(i)})^{\otimes 3} = \sum_{i=1}^{n} ((\alpha_{\pi(i)})^{\frac{1}{3}} v_{\pi(i)})^{\otimes 3}.$$

We say that Algorithm 7 "succeeds" if the algorithm returns linearly independent $l_1, ..., l_n \in \mathbb{C}^n$ such that $T = \sum_{i=1}^{n} l_i^{\otimes 3}$. Hence, from the previous equation, we can see that the algorithm indeed returns the unique decomposition up to permutation and scaling by cube roots of unity. This finally gives us that

$\Pr_{a,b \in_r S}[\text{Algorithm 7 "succeeds"}]$

$\geq \Pr_{a,b \in_r S}[\text{Algorithm 7 "succeeds"}, T^{(a)} \text{ is invertible and the eigenvalues of } T^{(a)'} T^{(b)} \text{ are distinct}]$

$= \Pr_{a,b \in_r S}[T^{(a)} \text{ is invertible and the eigenvalues of } T^{(a)'} T^{(b)} \text{ are distinct}]$

$\geq 1 - (\frac{2\binom{n}{2}}{|S|} + \frac{n}{|S|})$.

□

4.3 Complete Decomposition of Symmetric Tensors in Finite Arithmetic

We claimed at the beginning of Section 1.4.5 that the condition number for symmetric tensor decomposition is well defined. In Section 4.3 we first justify that claim, then present our finite precision decomposition algorithm (Algorithm 8), and analyze its properties from Section 4.3.2 onward.

4.3.1 Uniqueness of Tensor Decompositions

One of the important properties of tensor decompositions that motivates its study and applications in spite of the hardness of the problem, is the fact that in most cases the decompositions are unique (also sometimes referred to as *identifiable*). In comparison to this, matrix decompositions are usually not unique (except in special settings, refer to [Gil20] for a nice exposition on uniqueness results for matrix factorizations.

First, we state a well-known result showing that the tensor decomposition is unique up to permutation if it satisfies certain generic conditions. Here we will state the result following the notation of [Moi18].

Definition 4.3.1. *We say that two sets of factors*

$$\{(u^{(i)}, v^{(i)}, w^{(i)})\}_{i=1}^{r} \text{ and } \{(\overline{u}^{(i)}, \overline{v}^{(i)}, \overline{w}^{(i)})\}_{i=1}^{r}$$

are equivalent if there is a permutation $\pi : [r] \to [r]$ such that for all i,

$$u^{(i)} \otimes v^{(i)} \otimes w^{(i)} = \overline{u}^{(\pi(i))} \otimes \overline{v}^{(\pi(i))} \otimes \overline{w}^{(\pi(i))}.$$

Theorem 4.3.2. *[Har70, Moi18] Suppose we are given a tensor of the form*

$$T = \sum_{i \in [r]} u^{(i)} \otimes v^{(i)} \otimes w^{(i)}$$

where the following conditions are met:

- *the vectors* $\{u^{(i)}\}_i$ *are linearly independent.*
- *the vectors* $\{v^{(i)}\}_i$ *are linearly independent.*
- *every pair of vectors in* $\{w^{(i)}\}_i$ *is linearly independent.*

Then $\{(u^{(i)}, v^{(i)}, w^{(i)})\}_{i=1}^r$ *and* $\{(\overline{u}^{(i)}, \overline{v}^{(i)}, \overline{w}^{(i)})\}_{i=1}^r$ *are equivalent factors.*

Note that this reduces to the notion of diagonalisablity in symmetric tensors. Applying this to the case of symmetric tensors, we get the following corollary.

Corollary 4.3.2.1. *Let* $T = \sum_{i\in[n]} u_i^{\otimes 3}$ *be a symmetric tensor where the vectors* $u_i \in \mathbb{C}^n$ *are linearly independent. For any other decomposition* $T = \sum_{i\in[n]} (u_i')^{\otimes 3}$*, the vectors* u_i' *must satisfy* $u_i' = \omega_i u_{\pi(i)}$ *where* ω_i *is a cube root of unity and* $\pi \in S_n$ *a permutation.*

The above result was also derived in [Kay11] by a different method (uniqueness of polynomial factorization). For order-3 tensors, a more general regarding uniqueness of tensor decompositions is due to Kruskal [Kru77].] For the next lemma, recall the definition of the condition number of a diagonalisable symmetric tensor from Definition 1.4.3.

Lemma 4.3.3. *Let* T *be a diagonalisable tensor. Then for all* $U \in M_n(\mathbb{C})$ *such that* U *diagonalises* T*, the condition numbers* $\kappa_F(U)$ *are equal.*

Proof. By Corollary 4.3.2.1, for all $U \in M_n(\mathbb{C})$ such that U diagonalises T, the rows of U are unique up to permutation and scaling by cube roots of unity. Writing this in matrix notation, if U and U' are two such distinct matrices that diagonalise T, there exists a permutation $\pi \in S_n$ and a diagonal matrix D with cube roots of unity along the diagonal entries, such that $U' = DP_\pi U$ where P_π is the permutation matrix corresponding to π.

Now, $||U'||_F = ||DP_\pi U||_F$. If $u_1, ..., u_n$ are the rows of U, then indeed the rows of U' can be written as $u_i' = \omega_i u_{\pi(i)}$ where ω_i are the cube roots of unity. Using the definition of $||.||_F$, we get that $||U'||_F^2 = \sum_{i\in[n]} ||u_i'||^2 = \sum_{i\in[n]} ||\omega_i u_{\pi(i)}||^2 = \sum_{i\in[n]} ||u_i||^2 = ||U||_F^2$. Similarly,

$$||(U')^{-1}||_F = ||(DP_\pi U)^{-1}||_F = ||U^{-1}(P_\pi)^T D^{-1}||_F.$$

Since $(P_\pi)^T$ is also a permutation matrix, multiplication by it on the right permutes the columns of U^{-1}. Also, inverse of cube roots of unity are cube roots of unity as well. Hence, if $v_1', ..., v_n'$ are the columns of U^{-1}, and $v_1, ..., v_n$ are the columns of U, this gives us that $v_i' = \omega_i' v_{\pi^{-1}(i)}$ where ω_i' is a cube root of unity. This gives us that $||(U')^{-1}||_F^2 = \sum_{i=1}^n ||v_i'||^2 = \sum_{i\in[n]} ||\omega_i' v_{\pi^{-1}(i)}||^2 = \sum_{i\in[n]} ||v_i||^2 = ||U^{-1}||_F^2$. This finally gives us that $\kappa_F(U') = ||U'||_F^2 + ||(U')^{-1}||_F^2 = ||U||_F^2 + ||U^{-1}||_F^2 = \kappa_F(U)$. □

4.3.2 Finite-precision Jennrich's Algorithm for Symmetric Tensors

Algorithm 8: Jennrich's Algorithm for Complete Decomposition of Symmetric Tensors.

Let $C_{\text{gap}}, C_\eta > 0$ and $c_F > 1$ be some absolute constants we will fix in (6.6).

Input: An order-3 symmetric diagonalisable tensor $T \in (\mathbb{C}^n)^{\otimes 3}$, an estimate B for the condition number of the tensor and an accuracy parameter $\varepsilon \leq 1$.

Output: A solution to the ε-approximation problem for the decomposition of T.

Set $k_{\text{gap}} := \frac{1}{C_{\text{gap}} n^6 B^3}$ and $k_F := c_F n^5 B^3$.

Pick $(a_1, ..., a_n, b_1, ..., b_n) \in G_\eta$ uniformly at random where $\eta := \frac{1}{C_\eta n^{\frac{17}{2}} B^4}$ is the grid size.

Let $T_1, ..., T_n$ be the slices of T.

1 Compute $S^{(a)} = \sum_{i=1}^n a_i T_i$ and $S^{(b)} = \sum_{i=1}^n b_i T_i$ on a floating point machine.

2 Compute $S^{(a)'} = INV(S^{(a)})$ on a floating point machine where INV is the stable matrix inversion algorithm in Theorem 3.1.3.

Let $\delta := \frac{1}{n^{c_4 \log^2(\frac{nB}{\varepsilon})}}$ where c_4 is a constant we will set in Section 5.6.3.

3 Compute $D = \text{MM}(S^{(a)'}, S^{(b)})$ on a floating point machine where MM is the stable matrix multiplication algorithm in Theorem 3.1.3.

4 Let $v_1^{(0)}, ..., v_n^{(0)}$ be the output of $EIG - FWD$ on the input $(D, \delta, \frac{B}{k_{\text{gap}}}, B^{\frac{3}{2}}\sqrt{nk_F})$ on a floating point machine.

Let $V^{(0)}$ be the matrix with $v_1^{(0)}, ..., v_n^{(0)}$ as columns.

5 Compute $C = \text{INV}(V^{(0)})$ on a floating point machine where INV is the matrix inversion algorithm in Theorem 3.1.3 and let u_i' be the rows of C.

6 Let $\alpha_1', ..., \alpha_n'$ be the output of $TSCB(T, V^{(0)})$ where $TSCB$ is the algorithm for computing the trace of the slices after a change of basis in Algorithm 5.

7 Output $\{l_1, ..., l_n\}$ where $l_i = (\alpha_i')^{\frac{1}{3}} u_i'$ is computed on a floating point machine for all $i \in [n]$. Note that by $(\alpha_i')^{\frac{1}{3}}$ we refer to any one of the cube roots of α_i'.

Recall that the condition number for tensor diagonalisation $\kappa(T)$ was defined in Definition 1.4.3, and the notion of ε-approximation for tensor decomposition was defined in Section 1.4.3. Our main result about Algorithm 8 below already appears as Theorem 4.3.4 in the introduction, and it is the central result of this chapter. We will build towards the proof of this theorem in the following chapters.

Theorem 4.3.4 (Main Theorem)**.** *There is an algorithm which, given a diagonalisable tensor T, a desired accuracy parameter ε and some estimate $B \geq \kappa(T)$, outputs an ε-approximate solution to the tensor decomposition problem for T in*

$$O(n^3 + T_{MM}(n) \log^2 \frac{nB}{\varepsilon})$$

arithmetic operations on a floating point machine with

$$O(\log^{12}(\frac{nB}{\varepsilon}) \log n)$$

bits of precision, with probability at least $\left(1 - \frac{1}{n} - \frac{12}{n^2}\right)\left(1 - \frac{1}{\sqrt{2n}} - \frac{1}{n}\right)$.

4.3.3 Proof Strategy of Theorem 4.3.4:

As we have seen in Section 4.2, if a diagonalisable tensor T is given as input, if each of the steps of Algorithm 8 are performed in exact arithmetic and if we can perform matrix diagonalisation exactly in Step 4, we will get a (unique) decomposition of the tensor, that is, we get vectors $v_1, ..., v_n \in \mathbb{C}^n$ such that $T = \sum_{i=1}^n v_i^{\otimes 3}$ exactly.

Let $T_1, ..., T_n$ be the slices of the tensor. In the algorithm, we pick a, b uniformly and independently at random from the finite grid $G_\eta = \{-1, -1+\eta, -1+2\eta, ..., 1-2\eta, 1-\eta\}^{2n}$, then define $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$.

In Chapter 5, we use certain techniques for analysing numerical algorithms to give a proof of Theorem 4.3.4, albeit under some extra assumptions. Namely, we will assume in Theorem 4.3.6 below that we have picked points a, b from the finite grid such that the Frobenius condition number of $T^{(a)}$ is "small" and the eigenvalue gap of $(T^{(a)})^{-1}T^{(b)}$ is "large". These are stated more formally in Definition 4.3.5.

Definition 4.3.5 (Input Conditions)**.** *We say that $x = (T, a, b)$ satisfies the (n, B)-**input conditions** with parameters k_F, k_{gap} if the following condition is true: Let $T \in (\mathbb{C}^n)^{\otimes 3}$ be a diagonalisable tensor such that $\kappa(T) \leq B$ with slices $T_1, ..., T_n$. Let $a, b \in (-1, 1]^n$ and let $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$. Moreover, $T^{(a)}$ is invertible, $\kappa_F(T^{(a)}) \leq k_F$ and $gap((T^{(a)})^{-1}T^{(b)}) \geq k_{gap}$ where k_F and k_{gap} are as defined in Algorithm 9.*

Theorem 4.3.6. *Let x be an input to the algorithm such that it satisfies the (n, B)-input conditions with parameters $k_F := c_F n^5 B^3, k_{gap} := \frac{1}{C_{gap} n^6 B^3}$ (from Definition 4.3.5) where the constants C_{gap}, c_F are set in (6.6). Let $\varepsilon \leq 1$ be the input accuracy parameter. Then on input (x, ε) Algorithm 8 outputs an ε-approximation to the tensor decomposition problem for T in*

$$O(n^3 + T_{MM}(n) \log^2 \frac{nB}{\varepsilon})$$

arithmetic operations on a floating point machine with

$$O(\log^{12}(\frac{nB}{\varepsilon}) \log n)$$

bits of precision, with probability at least $1 - \frac{1}{n} - \frac{12}{n^2}$.

We will then see in Chapter 6 that these conditions are satisfied with high probability over the choice of a and b and complete the proof of Theorem 4.3.4.

Proof ideas for probabilistic analysis in Chapter 6:

There are two sources of randomization in our algorithm: the diagonalisation algorithm from [BGVKS22] is randomized, and moreover our algorithm begins with the computation of two random linear combinations $T^{(a)}, T^{(b)}$ of slices of the input tensor (step (i) of the algorithm sketch at the beginning of the chapter). As it turns out, the error bounds from Section 4.3 are established under the hypothesis that the *Frobenius condition number* of $T^{(a)}$ is "small" and the eigenvalue gap of $(T^{(a)})^{-1}T^{(b)}$ is "large". We therefore need to show that this hypothesis is satisfied for most choices of the random vectors a, b. For this we assume that a, b are chosen uniformly at random from a discrete grid. Our analysis in Chapter 6 follows a two-stage process:

(i) First we assume that a and b are drawn from the uniform distribution on the hypercube $[-1, 1)^n$. This is analyzed with the Carbery-Wright inequality, a well-known anticoncentration inequality.

(ii) In a second stage, we round the (real valued) coordinates of a and b in order to obtain a point of the discrete grid. This is analysed with the multivariate Markov inequality.[1]

This two-stage process is inspired by the construction of "robust hitting sets" in [FS18]. However, the general bounds from [FS18, Theorem 3.6] are not sharp enough for our purpose: they would lead to an algorithm using polynomially many bits of precision, but we are aiming for polylogarithmic precision. As a result, we need to perform an ad hoc analysis for certain linear and quadratic polynomials connected to the Frobenius condition number of $T^{(a)}$ and to the eigenvalue gap of $(T^{(a)})^{-1}T^{(b)}$. These are essentially the polynomials occuring in [BCMV14] in their analysis of the stability of Jennrich's algorithm with respect to input noise; but in that paper they choose a, b to be (normalized) Gaussian vectors rather than points from a discrete grid.

4.3.4 Proof Ideas of Theorem 4.3.6

Matrix diagonalisation

For step (ii) of our algorithm we require a fast and numerically stable diagonalisation algorithm, which takes in a diagonalisable matrix and outputs a set of eigenvectors. For that, we use the fast and numerically stable algorithm for matrix diagonalisation that we had given in Theorem 3.3.12 which is a slight modification of the diagonalization algorithm of [BGVKS22]. Recall that we had also shown that the condition number of the matrix (denoted by $\kappa(V)$) which has these eigenvectors as columns can be expressed as function of the condition number of the eigenproblem. Note that the choice of $\kappa(T)$ as our tensor decomposition number arises from that analysis and we show in Chapter 6 that if $\kappa(T)$ is bounded, then the condition number of the eigenproblem is bounded with high probability and hence, $\kappa(V)$ is bounded as well. This facilitates the inversion of V in Step (iii) of the algorithm sketch mentioned at the beginning of this chapter in a fast and numerically stable way. We note that the diagonalization algorithm of [BGVKS22] is responsible for the number of bits of precision needed in our main result (Theorem 4.3.4).

Finite precision analysis of tensor decomposition:

The correctness of the infinite-precision version of our main algorithm is established in Section 4.2, and we proceed with its analysis in finite arithmetic in Chapter 5. The principle behind this analysis is relatively straightforward: we need to show that the output of each of the 7 steps does not deviate too much from the ideal, infinite-precision output. For each step, we have two sources of error:

(i) The input to that step might not be exact because of errors accumulated in previous steps.

(ii) The computation performed in that step (on an inexact input) is inexact as well.

Summing up these two contributions, we can upper bound the error for that step. Moreover, for each step we already have estimates for the error (ii) due to the inexact computation. In particular, for basic operations such as matrix multiplication and inversion there are well-known guarantees recalled in Section 3.1; for the change of

[1] This discretization stage could also be analyzed with [Koi95, Theorem 3], but we would not obtain a sharper bound in this case.

basis algorithm we have the guarantees from Section 3.2; and for diagonalisation we have the guarantees from Section 3.3 based on [BGVKS22]. Nevertheless, obtaining reasonably precise error bounds from this analysis requires rather long and technical developments. In Chapter 5, we give a general framework for analysis of numerical algorithms. We analyse the error in each step of the algorithm, show that each step satisfies some notion of *numerical stability* and show that the algorithm which is a composition of these different *numerically stable* steps is also *numerically stable*.

Chapter 5

Numerical Algorithms

This chapter is dedicated to the study of the composition of numerically stable algorithms. Recall that in Section 1.3, we had discussed the notions of *numerical stability* of algorithms and had elucidated via an example how the composition of numerically stable algorithms is not always numerically stable. We had also discussed the notion of condition number which is a measure of how much the image of a function varies on slight perturbation of the input. In this chapter, we give formal definitions for related notions of numerical stability of algorithms and prove theorems related to *numerical stability* of composition of *numerically stable* algorithms. As an application, we show that certain simple functions appearing in Algorithm 8 satisfy these conditions. This in turn helps us to prove the correctness of Algorithm 8, under the assumption that the input satisfies certain conditions.

5.1 Overview of the Chapter:

5.1.1 Analysis of numerical stability of algorithms:

In Section 5.2, we introduce the notion of (a, b)-continuous functions. Recall that we had discussed forward-stable algorithms in Section 1.3. In this chapter, we give a concrete mathematical definition of this. We then define an algorithm $\tilde{f}$ computing a function f to be a robustly numerically stable if when given a slightly perturbed input to $\tilde{f}$, it outputs a solution*close* to the value of the function f on the actual input. We finally show that a forward stable algorithm for computing a continuous function is also robustly numerically stable. In Section 5.4, we also define formally the notion of approximate computation by probabilistic algorithms and in Theorem 5.4.2 give sufficient conditions for which the composition of approximate probabilistic algorithms is also an approximate probabilistic algorithm.

5.1.2 Application to the analysis of Algorithm 8:

The goal of this section is to formally prove Theorem 4.3.6 which is one of the main steps for a correctness proof of Algorithm 8 as described in Section 4.3.3. In this theorem, we show that Algorithm 8 indeed outputs an ε-approximate solution to the tensor decomposition problem (refer to Section 1.4.3), if the input satisfies some *special conditions*. To prove this theorem, firstly we define *simple* functions corresponding to the steps of Algorithm 8. Recall that Algorithm 8 consists of seven steps. Using the machinery developed in Section 5.2, we show that the functions corresponding to Steps 1,2,3,5 and 6 are robustly numerically stable. We also prove certain similar guarantees for step 7 of the algorithm, albeit this deviates slightly from the definition of continuous functions. Step 4 in Algorithm 8 requires diagonalisation of a diagonalisable matrix. We can use the analysis from Section 3.3 to directly show

that our diagonalisation algorithm (Algorithm 6) indeed satisfies our definitions of approximate computation by probabilistic algorithms.

Finally, we show that Algorithm 8 can be rewritten as a composition of functions made of these simple functions. We also show that when the input to Algorithm 8 satisfies the *special conditions*, then we can use Theorem 5.4.2 to show that the composition at the end of each step is also a *probabilistic algorithm* (according to Definition 5.4.1).

5.2 Numerical Stability of Algorithms

For a function f on domain X, one can associate to it a parameter $\kappa_f : X \to \mathbb{R}$ which we will refer to as the condition number. The condition number can be chosen independently for every function with the following goal in mind : If a function is *continuous* (refer to Definition 5.2.1) on a certain subdomain, the condition number parameter indicates how *ill-conditioned* the problem is on certain inputs, that is how much the function value deviates on slight perturbation of the input values. A higher condition number on a certain input indicates that the function is more ill-conditioned on that particular input.

For a function f, we denote by $\text{dom}(f)$, the domain of the function.

Definition 5.2.1. *Let $f : S \subset \mathbb{C}^M \to \mathbb{C}^N$ with condition number κ_f and let $u \in \mathbb{R}_+$. Let $x \in S$ be an input for f such that $B(x,u) \subset S$. We call f an (a,b)-continuous function on subdomain S at scale u if for all $\tilde{x} \in B(x,u)$ such that $\tilde{x} \in dom(f)$,*

- $||f(\tilde{x}) - f(x)|| \leq ua\kappa_f(x)$.
- $\kappa_f(\tilde{x}) \leq b\kappa_f(x)$.

where $a, b \in \mathbb{R}_+$. We will see some concrete examples of functions satisfying these conditions in Section 5.3.

We had discussed the notion of *forward-stable* numerical algorithms in Section 1.3. Informally, it is an algorithm for computing a certain function that outputs a solution close to the actual output value on a certain desired input.

Definition 5.2.2. *Let $f : S \subset \mathbb{C}^M \to \mathbb{C}^N$ be a function with condition number κ_f. We say that $\tilde{f}$ is a (u,ψ)-forward stable algorithm for the function f on the domain S at machine precision $u \in \mathbb{R}_+$ if on any input $x \in S$,*

$$||\tilde{f}(x) - f(x)|| \leq u.\psi(\kappa_f(x))$$

Note: For forward-stable algorithms, u will usually be the precision of the machine on which the algorithm is executed.

Comparison to [BNV23]: The definition of forward stable algorithms in [BNV23] can be obtained from Definition 5.2.2 by adding the following restrictions that $u < \frac{1}{a(M)}.(1+\kappa(x))$ and $\psi(\kappa_f(x)) \leq a(M)(1+\kappa(x))$ where a is a univariate polynomial, M is the dimension of the input space and $\kappa(x)$ satisfies the definition of condition number (Definition 3.1) in [BNV23].

We introduce the notion of *robust numerical stability* of algorithms. An algorithm is said to be robustly numerically stable if the desired input to the algorithm is perturbed slightly, then the algorithm outputs some solution close to the actual solution on the desired input.

Definition 5.2.3. *Let $f : \mathbb{C}^M \to \mathbb{C}^N$ be a function with condition number κ_f. We say that $\tilde{f}$ is a (u, ψ')-robust numerically stable algorithm computing the function f on some set $S \subset \mathbb{C}^M$ with scale $u > 0$ if on any input $x \in S$ such that $B(x, u) \subset S$ and for all $\tilde{x} \in B(x, u)$,*

$$||\tilde{f}(\tilde{x}) - f(x)|| \leq u.\psi'(\kappa_f(x)).$$

We call ψ' to be the "condition number growth function" of $\tilde{f}$.

In the following lemma, we show that a "forward stable" algorithm for a "continuous" function is also "robustly numerically stable". As shown in the lemma, the function ψ' in Definition 5.2.3 witnessing the robust numerical stability of $\tilde{f}$ is a small modification of the function ψ of Definition 5.2.2 (witnessing the numerical stability of $\tilde{f}$).

Note: In Definition 5.2.3, the machine precision of $\tilde{f}$ and the scale of f have been assumed to be equal. This is just to make the analysis simpler and more streamlined. One can definitely take them to be different and this can often result in stricter error bounds. In fact, when this setup is applied to Algorithm 8, this is the central reason why we get a bound of $\log^{12}(\frac{nB}{\epsilon})$ bits of precision using this framework as compared to the analysis in [KS22a] which gives us a bound of $\log^4(\frac{nB}{\epsilon})$.

Lemma 5.2.4. *Let $\tilde{f}$ be a (u, ψ)-forward stable algorithm computing a function $f : \mathbb{C}^M \to \mathbb{C}^N$ with condition number κ_f which is (a, b)-continuous for scale u on domain $S \subset \mathbb{C}^M$ where ψ is a non-decreasing function. Then $\tilde{f}$ is also (u, ψ')-robust numerically stable on S where $\psi' = \psi \circ \phi_b + \phi_a$ where for any $c \in \mathbb{R}_+$, ϕ_c is defined as follows:*

$$\begin{aligned} \phi_c : \mathbb{R}_+ &\to \mathbb{R}_+ \\ t &\mapsto ct \end{aligned}$$

Proof. Let $x \in S$ be such that $B(x, u) \subset S$. Hence, any element $\tilde{x} \in B(x, u)$ lies in S as well. Since, f is a (a, b)-continuous function for scale u on the domain S, following Definition 5.2.1, we have that

$$||f(\tilde{x}) - f(x)|| \leq u.a.\kappa_f(x). \tag{5.1}$$

Following Definition 5.2.2, since $\tilde{f}$ is a (u, ψ)-forward stable algorithm on domain S and $\tilde{x} \in S$,

$$||\tilde{f}(\tilde{x}) - f(\tilde{x})|| \leq u \cdot \psi(\kappa_f(\tilde{x})) \tag{5.2}$$

We also have that $\kappa_f(\tilde{x}) \leq b\kappa_f(x)$ (this can concluded from Definition 5.2.1 for a continuous function f). Putting this back in (5.2) along with the fact that ψ is an non-decreasing function on its first coordinate, we have that

$$||\tilde{f}(\tilde{x}) - f(\tilde{x})|| \leq u \cdot \psi(b\kappa_f(x)). \tag{5.3}$$

Combining (5.1) and (5.3) using triangle inequality, we get the desired result. □

5.3 Defining functions and a robustness result

Recall that we had designed Algorithm 7 in Section 4.2 for computing the (unique) tensor decomposition of diagonalisable tensors. Notice that each step of the algorithm is a simple, linear algebraic function and in this section, we design functions $f_1, ..., f_7$

corresponding to the steps of Algorithm 7. (Since, Steps 2 and 5 are matrix inversion maps, hence, we will define $f_5 = f_2$.)

Moreover, in this section, we also define maps $\tilde{f}_i$ which can be essentially thought of as algorithms for computing f_i in finite precision arithmetic for all $i \in [7]$. We also define corresponding condition numbers κ_i for all $i \in \{1, 2, 3, 5, 6, 7\}$ (refer to the discussion at the beginning of Section 5.2).

The main conclusions of this section which is stated more formally in Theorem 5.3.11 are the following:

- Firstly, using the machinery developed in Section 5.2, we show that for some scale $\alpha_i > 0$, $\tilde{f}_i$ is a (α_i, ψ_i')-robust numerically stable algorithm for computing the function f_i on domain X_i (with input parameter n) for all $i \in \{1, 2, 3, 5, 6\}$ where ψ_i' is a function which is quasi-polynomial in n and κ_i.
- For f_7, we also show a similar result: For some $x \in X_7$ (with input parameter n), we get that on input some $\tilde{x}$ *close* to x, $\tilde{f}_7$ outputs some $\tilde{y}$ which is also close to some $y \in f_7(x)$.
- For f_4 (which corresponds to matrix diagonalisation), we also show that for some specific parameters p, $\tilde{f}_{4,p}$ is a *probabilistic algorithm* (refer to Definition 5.4.1) for computing f_4.

Organization: For better readability of this exposition, we keep the analysis for Steps 1,2 and 3 in the main text and defer the proofs for the rest to Appendix C.

5.3.1 Step 1:

Defining f_1: Let $X_1 := (\mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n) \times \mathbb{C}^n \times \mathbb{C}^n$ and $Y_1(n) = M_n(\mathbb{C}) \times M_n(\mathbb{C})$. Define function $f_1 : X_1 \to Y_1(n)$ as $f_1(T, a, b) = (T^{(a)}, T^{(b)})$ where for any $c = (c_1, ..., c_n) \in \mathbb{C}^n$, we define $T^{(c)} = \sum_{i=1}^n c_i T_i$. For any element, $x = (T, a, b) \in X_1$, we define $||x|| = \sqrt{||T||_F^2 + ||a||^2 + ||b||^2}$. We also define $\kappa_1(x) = \sqrt{||x||^2 + 2}$. For any $y = (A, B) \in Y_1$, we define $||y|| = \sqrt{||A||_F^2 + ||B||_F^2}$.

Lemma 5.3.1. *For any $\delta_0 \in (0, 1)$, f_1 is a $(2\sqrt{2}, \sqrt{2})$-continuous function at scale δ_0 on domain X_1.*

Proof. Let $x = (T, a, b) \in X_1$ such that

$$B(x, \delta_0) := \{x' : ||x - x'|| \leq \delta_0\} \subseteq X_1. \tag{5.4}$$

Let $\tilde{x} = (\tilde{T}, \tilde{a}, \tilde{b}) \in B(x, \delta_0)$. Then

$$||f_1(\tilde{x}) - f_1(x)|| = \sqrt{||\tilde{T^{(a)}} - T^{(a)}||_F^2 + ||\tilde{T^{(b)}} - T^{(b)}||_F^2} \tag{5.5}$$

Now, $\tilde{a} \leq ||a|| + \delta_0 \leq ||a|| + 1$. We define $T_{jk} = (T_{1,j,k}, ..., T_{n,j,k})$ for all $j, k \in [n]$. Then we have that

$$\begin{aligned}
||\tilde{T^{(a)}} - T^{(a)}||_F^2 &= \sum_{j,k=1}^{n} |\langle \tilde{a}, \tilde{T_{jk}} \rangle - \langle a, T_{jk} \rangle|^2 \\
&\leq 2 \sum_{j,k=1}^{n} \left(|\langle \tilde{a}, \tilde{T_{jk}} - T_{j,k} \rangle|^2 + |\langle \tilde{a} - a, T_{jk} \rangle|^2 \right) \\
&= 2||\tilde{a}||^2 \sum_{j,k=1}^{n} ||\tilde{T_{jk}} - T_{j,k}||^2 + 2||\tilde{a} - a||^2 \sum_{j,k=1}^{n} ||T_{jk}||^2 \\
&= 2(||a|| + 1)^2 ||\tilde{T} - T||_F^2 + 2||\tilde{a} - a||^2 ||T||_F^2 \\
&= 2\delta_0^2((||a|| + 1)^2 + ||T||_F^2)
\end{aligned}$$

Following a similar calculation for b and putting this back in (5.5), we have that

$$||f_1(\tilde{x}) - f_1(x)|| \leq 2\delta_0 \sqrt{(||a|| + 1)^2 + (||b|| + 1)^2 + 2||T||_F^2} \leq 2\sqrt{2}\delta_0 \sqrt{||x||^2 + 2}. \tag{5.6}$$

Now we want to bound the change in κ_1 on perturbation of input. Let $\tilde{x} \in B(x, \delta_0)$ and following the definition of κ_1, we have that

$$\kappa_1(\tilde{x}) = \sqrt{||\tilde{x}||^2 + 2} \leq \sqrt{(||x|| + \delta_0)^2 + 2} \leq \sqrt{2}\sqrt{||x||^2 + 2} = \sqrt{2}\kappa_1(x)$$

□

Defining $\tilde{f_1}$: Let $a, b \in \mathbb{C}^n$ and $T \in (\mathbb{C}^n)^{\otimes 3}$ be the inputs. Let $T_{jk} := (T_{1,j,k}, ..., T_{n,j,k})$. Then to compute (j,k)-th entry of $T^{(a)}$, the algorithm computes the inner product of a and $T_{j,k}$ on a floating point machine with suitable machine precision.

Lemma 5.3.2. *For any $0 \leq \alpha_0 \leq \frac{1}{2n}$, $\tilde{f_1}$ is an (α_0, ψ_1)-numerically stable algorithm for the function f on domain X_1 where $\psi_1(\kappa_1) = n\kappa_1^2$ when run on a floating point machine with machine precision α_0.*

Proof. Since $(T^{(a)})_{jk} = \sum_{i=1}^{n} a_i (T_i)_{j,k}$, it follows from (3.2) that

$$||(S^{(a)})_{j,k} - (T^{(a)})_{j,k}|| \leq \gamma_n ||a|| \sqrt{\sum_{i=1}^{n} |(T_i)_{j,k}|^2}.$$

Moreover, $\gamma_n \leq 2n\alpha_0$ since $n\alpha_0 \leq \frac{1}{2}$. Hence

$$\begin{aligned}
||S^{(a)} - T^{(a)}||_F = \sqrt{\sum_{j,k=1}^{n} ||(S^{(a)})_{j,k} - (T^{(a)})_{j,k}||^2} &\leq 2n\alpha_0 ||a|| \sqrt{\sum_{i,j,k=1}^{n} |(T_i)_{j,k}|^2} \\
&= 2n\alpha_0 ||a|| ||T||_F.
\end{aligned} \tag{5.7}$$

Similarly, for $S^{(b)}$, using the same computation, we also have that $||S^{(b)} - T^{(b)}||_F \leq 2n\alpha_0 ||b|| ||T||_F$. Combining them, we get that

$$\begin{aligned}
||\tilde{f_1}(x) - f_1(x)|| &= \sqrt{||S^{(a)} - T^{(a)}||_F^2 + ||S^{(b)} - T^{(b)}||_F^2} \\
&\leq 2n\alpha_0 ||T||_F \sqrt{||a||^2 + ||b||^2} \leq n\alpha_0 ||x||^2 \leq n\alpha_0 \kappa_1^2(x)
\end{aligned} \tag{5.8}$$

□

Theorem 5.3.3. *For any $\varepsilon_0 \in (0, \frac{1}{2n})$, $\tilde{f}_1$ is an (ε_0, ψ_1')-robust numerically stable algorithm for computing f_1 on domain X_1 where $\psi_1'(\kappa_1) = 2n\kappa_1^2 + 2\sqrt{2}\kappa_1$ when run on a floating point machine with machine precision ε_0.*

Proof. From Lemma 5.2.4, we get that

$$\begin{aligned} ||\tilde{f}_1(\tilde{x}_1) - f_1(x_1)|| &\leq \varepsilon_0.\psi_1'(\kappa_1(x)) \\ &= \varepsilon_0(\psi_1(\sqrt{2}\kappa_1(x)) + 2\sqrt{2}\kappa_1(x)) \\ &= \varepsilon_0\Big(2n(\kappa_1(x))^2 + 2\sqrt{2}(\kappa_1(x))\Big). \end{aligned}$$

□

5.3.2 Step 2:

Recall that we had defined the Frobenius condition number of matrices (denoted by κ_F) in (3.30).We first prove the following theorem: if A is a matrix with bounded κ_F and A' is another matrix which is *close* to A (in the Frobenius norm), then $(A')^{-1}$ is also *close* to A^{-1}.

Lemma 5.3.4. *Let $A \in M_n(\mathbb{C})$ be such that $\kappa_F(A) \leq K \leq \infty$. Define $A' \in M_n(\mathbb{C})$ as $A' = A + \Delta$ where $||\Delta||_F \leq M$ and $M\sqrt{K} \leq 1$. Then A' is invertible and*

$$||(A')^{-1} - A^{-1}||_F \leq \frac{MK}{1 - M\sqrt{K}}$$

Proof. We first use the fact that for any matrix $B \in M_n(\mathbb{C})$, if $||B|| < 1$, $I + B$ is invertible and

$$(I + B)^{-1} = \sum_{i=0}^{\infty}(-1)^i B^i. \tag{5.9}$$

Since $A' = A(I + A^{-1}\Delta)$ where $||\Delta|| \leq M$ and $||A^{-1}|| \leq ||A^{-1}||_F \leq \sqrt{\kappa_F(A)} \leq \sqrt{K}$, we have that $||A^{-1}\Delta|| \leq ||A^{-1}||||\Delta|| \leq M\sqrt{K} < 1$. This shows that A' is invertible, hence $(A')^{-1} = (I + A^{-1}\Delta)^{-1}A^{-1}$. Now, we can use (5.9) for $B = A^{-1}\Delta$ and apply the triangle inequality to get that

$$\begin{aligned} ||(A')^{-1} - A^{-1}|| &= ||(I + A^{-1}\Delta)^{-1}A^{-1} - A^{-1}|| \\ &\leq ||A^{-1}||||(I + A^{-1}\Delta)^{-1} - I|| \\ &\leq ||A^{-1}||||\Big(\sum_{i=1}^{\infty}||A^{-1}\Delta||^i\Big). \end{aligned}$$

Hence, we can finally conclude that

$$||(A')^{-1} - A^{-1}|| \leq \frac{MK}{1 - M\sqrt{K}}.$$

□

Defining f_2: Let function $f_2 : \mathrm{GL}_n(\mathbb{C}) \to \mathrm{GL}_n(\mathbb{C})$ be defined as the inversion of matrices i.e. $f_2(A) = A^{-1}$. We define the condition number for f_2 as $\kappa_2(A) = \kappa_F(A) = ||A||_F^2 + ||A^{-1}||_F^2$. We define $X_2 = \mathrm{GL}_n(\mathbb{C})$ to be the input space for f_2 and use the Frobenius norm of matrices as the metric on $\mathrm{GL}_n(\mathbb{C})$.

Lemma 5.3.5. *For any $\delta_1 > 0$, f_2 is a $(2,8)$-continuous function at scale δ_1 on domain $I_2(\delta_1) := \{A | A \in GL_n(\mathbb{C}), \delta_1\sqrt{\kappa_2(A)} \leq \frac{1}{2}\}$*

Proof. Let $A \in I_2(\delta_1)$ and let $\tilde{A} \in B(A, \delta_1)$. Using Lemma 5.3.4, we get that $\tilde{A}$ is invertible and hence, $\tilde{A} \in \text{dom}(f_2)$. Moreover,

$$||f_2(\tilde{A}) - f_2(A)|| = ||(\tilde{A})^{-1} - A^{-1}||_F \leq \frac{\delta_1\kappa_2(A)}{1 - \delta_1\sqrt{\kappa_2(A)}} \leq 2\delta_1\kappa_2(A).$$

Using the definition of the condition numbers, we also have that

$$\begin{aligned}\kappa_2(\tilde{A}) &= ||\tilde{A}||_F^2 + ||(\tilde{A})^{-1}||_F^2 \leq (\sqrt{\kappa_2(A)} + \delta_1)^2 + (\sqrt{\kappa_2(A)} + 2\delta_1\kappa_2(A))^2 \\ &\leq 4\kappa_2(A) + 4\kappa_2(A) = 8\kappa_2(A)\end{aligned}$$

The second last inequality uses the fact that $\delta_1 \leq \frac{1}{2\sqrt{\kappa_2(A)}} \leq \sqrt{\kappa_2(A)}$ since $\kappa_2(A) > 1$. □

Defining $\tilde{f}_2$: We fix a matrix multiplication algorithm as mentioned in Theorem 3.1.3 with a fixed $\eta > 0$ and define $\tilde{f}_2$ to be the numerically stable algorithm for computing matrix inversion on a floating point machine with some machine precision α_1.

Lemma 5.3.6. *For any $\alpha_1 > 0$, $\tilde{f}_2$ is a (α_1, ψ_2)-numerically stable algorithm for computing the function f_2 at machine precision α_1 on domain $GL_n(\mathbb{C})$ where $\psi_2(\kappa_2) = C_2 n^{c_\eta + \log(10)} \kappa_2^{8\log(n)+\frac{1}{2}}$ for some constant $C_2 > 0$.*

Proof. Let $A \in \text{GL}_n(\mathbb{C})$ be an input to $\tilde{f}_2$. Then

$$||\tilde{f}_2(A) - f_2(A)|| \leq \mu_{\text{INV}}(n).\alpha_1.(\kappa_2(A))^{c_{\text{INV}}\log(n)}||A^{-1}|| \leq C_2 n^{c_\eta+\log(10)}\kappa_2^{8\log(n)+\frac{1}{2}}$$

for some constant $C_2 > 0$. □

Now we combine the two above lemmas to show that $\tilde{f}_2$ is a robust numerically stable algorithm for computing f_2.

Theorem 5.3.7. *For any $\varepsilon_1 \in (0, 1]$, $\tilde{f}_2$ is an (ε_2, ψ_3')-robust numerically stable algorithm for computing f_2 on domain $I_2(\varepsilon_2) := \{A | A \in GL_n(\mathbb{C}), \varepsilon_2\sqrt{\kappa_2(A)} \leq \frac{1}{2}\}$. where $\psi_2'(\kappa_2) = \mu'_{INV}(n).(8\kappa_2(x_2))^{8\log(n)+\frac{1}{2}} + 2\kappa_2(x_2)$ and $\mu'_{INV}(n) = 2\sqrt{2}.\mu_{INV}(n).n^{8\log(8)}$.*

Proof. Let $x_2 \in I_2$ and $\tilde{x}_2 \in B(x_2, \varepsilon_1)$. Using Lemma 5.2.4, we get that

$$\begin{aligned}||\tilde{f}_2(\tilde{x}_2) - f_2(x_2)|| &\leq \varepsilon_1.\psi_2'(\kappa_2(x_2)) \\ &= \varepsilon_1(\psi_2(8\kappa_2(x_2)) + 2\kappa_2(x_2)) \\ &= \varepsilon_1(\mu_{INV}(n).(8\kappa_2(x_2))^{8\log(n)+\frac{1}{2}} + 2\kappa_2(x_2)) \\ &= \varepsilon_1(\mu'_{INV}(n).(8\kappa_2(x_2))^{8\log(n)+\frac{1}{2}} + 2\kappa_2(x_2))\end{aligned}$$

where $\mu'_{INV}(n) = 2\sqrt{2}.\mu_{INV}(n).n^{8\log(8)}$. □

5.3.3 Step 3:

Defining f_3: Let $X_3 := M_n(\mathbb{C}) \times M_n(\mathbb{C})$ be the input space. Then function $f_3 : X_3 \to M_n(\mathbb{C})$ be defined as the matrix multiplication map $f_3(A, B) = AB$. We define the condition number of f_3 as $\kappa_3(x) = \sqrt{2}||x|| + 1$. For any $x = (A, B) \in X_3$, we define the norm on the input space, $||x|| = \sqrt{||A||_F^2 + ||B||_F^2}$. We define the norm on the output space $M_n(\mathbb{C})$ to be the Frobenius norm.

Lemma 5.3.8. *Then for any $\delta_2 \in (0, 1]$, f_3 is a $(1, 2\sqrt{2})$-continuous function at scale δ_2 on domain X_3.*

Proof. Let $x = (A, B) \in X_3$ and $\tilde{x} = (\tilde{A}, \tilde{B}) \in B(x, \delta_2)$.Then

$$\begin{aligned} ||f_3(\tilde{x}) - f_3(x)|| &= ||\tilde{A}\tilde{B} - AB||_F \\ &\leq ||\tilde{A}\tilde{B} - A\tilde{B}||_F + ||A\tilde{B} - AB||_F \\ &\leq ||\tilde{A} - A||_F||\tilde{B}||_F + ||A||_F||\tilde{B} - B||_F \\ &\leq \delta_2(||\tilde{B}||_F + ||A||_F) \leq \delta_2(\sqrt{2}||x|| + 1) = \delta_2\kappa_3(x). \end{aligned}$$

The second-last inequality follows from the fact that $||\tilde{B}||_F \leq ||B||_F + \delta_2 \leq ||B||_F + 1$ and an application of the Cauchy-Schwarz inequality.

Let $x = (A, B) \in X_3$ and $\tilde{x} \in B(x, \delta_2)$. Following the definition of κ_3, we have that

$$\kappa_3(\tilde{x}) = \sqrt{2}||\tilde{x}|| + 1 \leq \sqrt{2}||x|| + \sqrt{2}\delta_2 + 1 \leq \sqrt{2}(||x|| + 2) \leq 2\sqrt{2}(\sqrt{2}||x|| + 1) = 2\sqrt{2}\kappa_3(x)$$

□

Defining $\tilde{f}_3$: Let $\tilde{f}_3$ be a fixed numerically stable algorithm for matrix multiplication with a fixed $\eta > 0$, as mentioned in Theorem 3.1.3 with machine precision $u = \alpha_1$.

Lemma 5.3.9. *For any $\alpha_2 > 0$, $\tilde{f}_3$ is a (α_2, ψ_3)-numerically stable algorithm for computing the function f_3 on domain X_3 where $\psi_3(\kappa_3) = \frac{1}{4}n^{c_\eta + \frac{1}{2}}\kappa_3^2$.*

Proof. Let $x = (A, B) \in X_3$ be the input to $\tilde{f}_3$. Then using the bounds from Theorem 3.1.3 (1), we can conclude that

$$\begin{aligned} ||\tilde{f}_3(x) - f_3(x)|| \leq n^{c_\eta + \frac{1}{2}}.\alpha_2.||A||||B|| &\leq \alpha_2 \cdot \frac{1}{2}n^{c_\eta + \frac{1}{2}}(||A||^2 + ||B||^2) \\ &\leq \alpha_2 \cdot \frac{1}{4}n^{c_\eta + \frac{1}{2}}(2||x||^2 + 1) \\ &\leq \alpha_2 \cdot \frac{1}{4}n^{c_\eta + \frac{1}{2}}(\sqrt{2}||x|| + 1)^2 \\ &= \alpha_2 \cdot \frac{1}{4}n^{c_\eta + \frac{1}{2}}(\kappa_3(x))^2. \end{aligned}$$

□

Now we combine the two above lemmas to show that $\tilde{f}_3$ is a robust numerically stable algorithm for computing f_3.

Theorem 5.3.10. *For any $\varepsilon_2 \in (0, 1]$, $\tilde{f}_3$ is an (ε_2, ψ_3')-robust numerically stable algorithm for computing f_3 on domain X_3 where $\psi_3'(\kappa_3) = 2n^{c_\eta + \frac{1}{2}}\kappa_3^2 + \kappa_3$.*

Proof. Let $x_3, \tilde{x}_3 \in X_3$ such that $\tilde{x}_3 \in B(x_3, \varepsilon_2)$. Then using Lemma 5.2.4, we get that

$$\begin{aligned} ||\tilde{f}_3(\tilde{x}_3) - f_3(x_3)|| &\leq \varepsilon_2.\psi_3'(\kappa_3(x)) \\ &= \varepsilon_2(\psi_3(2\sqrt{2}\kappa_3(x)) + \kappa_3(x)) \\ &= \varepsilon_2(2n^{c_\eta+\frac{1}{2}}\kappa_3^2 + \kappa_3). \end{aligned}$$

□

5.3.4 Conclusion:

In this section, we have already defined functions f_i for $i \in [3]$ and the corresponding numerically stable algorithms $\tilde{f}_i$ computing them in finite arithmetic. As explained previously, these correspond to Steps 1-3 of Algorithm 8. The functions f_i for $i \in \{5, 6, 7\}$ which are involved in Steps 5-7 of the algorithm and their corresponding numerically stable algorithms $\tilde{f}_i$ for $i \in \{5, 6, 7\}$ are defined in Appendices C.2.1-C.2.4.

The function corresponding to Step 4, denoted by f_4, takes in a diagonalisable matrix with distinct eigenvalues and returns the set of eigenvectors of the matrix. Further in Appendix C.2.1, we denote by $\tilde{f}_{4,p}$, the corresponding algorithm for computing f_4. It is essentially the algorithm for matrix diagonalisation (Algorithm 6) described in Section 3.3 run with parameters $p = (\delta, K_{\text{eig}}, K_{\text{norm}})$ as mentioned in the description of the algorithm.

The following is the main theorem of this section

Theorem 5.3.11. *For $i \in \{1, 3, 6\}$, $\tilde{f}_i$ is a (δ_i, ψ_i')-robust numerically stable algorithm for computing f_i on domain X_i where $\delta_i \in (0, \frac{1}{10n})$ and $\psi_i'(\kappa_i) = (n\kappa_i^{\log n})^{m_i}$ for some constant m_i.*

$\tilde{f}_2$ is a (δ_2, ψ_2')-robust numerically stable algorithm for computing f_2 on domain $I_2(\delta_2)$ where $\delta_2 \in (0, 1]$ and $\psi_2'(\kappa_2) = (n\kappa_2^{\log n})^{m_2}$ for some constant m_2.

Moreover, let $x, \tilde{x} \in X_7$ such that $||x - \tilde{x}|| \leq \delta \leq \frac{1}{216(||x||+1)}$. Then algorithm $\tilde{f}_7$ when run on input $\tilde{x}$ on a machine with precision δ, there exists $y \in f_7(x)$ such that

$$||\tilde{f}_7(\tilde{x}) - y|| \leq 2\boldsymbol{u}\Big(2n + (n||x||)^{\frac{2}{3}} + ||x||^2\Big).$$

where $\boldsymbol{u} := 6\delta^{\frac{1}{3}}(||x|| + 1)^{\frac{1}{3}}$.

Let $p = (\varepsilon_4, K_{eig}, K_{norm})$ be some parameters where $\varepsilon_4 \in (0, \frac{1}{2})$. Define $X_{4,p} := \{x \in X_4 | x \text{ satisfies parameter } p\}$ and

$$\boldsymbol{u}_p = \frac{1}{n^{C_4 \log^4(\frac{nK_{eig}K_{norm}}{\varepsilon_4})}} \text{ for some constant } C_4 > 0.$$

Then $\tilde{f}_{4,p}$ is a $(1 - \frac{1}{n} - \frac{12}{n^2}, \boldsymbol{u}_p, \varepsilon_4)$-algorithm for computing f_4 on subdomain $X_{4,p}$ when run on a finite precision machine with machine precision $\boldsymbol{u}_p$.

5.4 Composition Theorem

The main goal of this section is to create a general framework for the analysis of (probabilistic) algorithms that compute a function approximately on a subdomain of

the function. More formally, let f be a function which on input x returns a set of solutions Y.

One example of such function is the cube root function. For any $\alpha \in \mathbb{C}$, the output of the function is $\{\beta | \beta^3 = \alpha\}$. Another example is the function which takes in a rank-1 order-d tensor and outputs a decomposition. This is given by the map

$$u_1 \otimes ... \otimes u_d \mapsto \left\{ (w_1 u_1, ..., w_d u_d) | w_1 ... w_d = 1 \right\}$$

Another example we want to focus on is the function that takes in a diagonalisable matrix with distinct eigenvalues and outputs the set of all possible tuples of eigenvectors. Let $\mathcal{D}_n$ be the set of diagonalisable matrices with distinct eigenvalues. If a diagonalisable matrix $A \in \mathcal{D}_n$ is given as input, the function $\phi : \mathcal{D}_n \to \mathcal{P}((\mathbb{C}^n)^n)$ returns the following set

$$\phi(A) := \{(v_1, ..., v_n) \text{ is an ordered tuple of eigenvectors of } A\}$$

For a matrix $A \in \mathcal{D}_n$, its eigenvectors are unique up to scaling and permutations. As mentioned in [BBV19], a way that analysis of such set-valued functions can be often avoided is by quotienting the space by a suitable equivalence relation. In this specific case of ϕ, one can avoid dealing with set-valued functions by quotienting $\phi(\mathcal{D}_n)$ by the symmetric group on n elements, S_n and then treating the quotient space as a set of projective lines. But note that this is a very specific solution which can be applied specifically to this function whereas in this chapter, we give a general framework that does not assume any structure on the space of possible solutions.

We define a (probabilistic) algorithm to be a function which on input some $\tilde{x}$ close to x outputs $\tilde{y}$ close to some $y \in Y$ (with probability p). Recall that we had discussed in Example 1.3.1 in Section 1.3 how the composition of numerically stable algorithms might not always be numerically stable. In Theorem 5.4.2, we give sufficient conditions such that the composition of *probabilistic algorithms* is also a *probabilistic algorithm.*

Definition 5.4.1 (Probabilistic Algorithm)**.** *Let $f : \mathbb{C}^M \to \mathcal{P}(\mathbb{C}^N)$ be a function. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space where Ω is a set, $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ is a σ-algebra and $\mathbb{P} : \mathcal{F} \to [0,1]$ is a probability measure. Define $\tilde{F} := \{\tilde{f}_\omega : \mathbb{C}^M \to \mathbb{C}^N | \omega \in \Omega\}$ to be a family of functions. Let $u, \varepsilon \in \mathbb{R}_+$ and let $x \in S$ and $\tilde{x} \in B(x, u)$. Then define*

$$\mathcal{A}_{x,\tilde{x}} := \{\omega \in \Omega | \text{ there exists some } y \in f(x) \text{ such that } ||\tilde{f}_\omega(\tilde{x}) - y|| \leq \varepsilon\}.$$

We say that $\tilde{F}$ is a (p, u, ε)-algorithm on probability space $(\Omega, \mathcal{F}, \mathbb{P})$ computing f on domain $S \subset \mathbb{C}^M$ if $\mathbb{P}(\mathcal{A}_{x,\tilde{x}}) \geq p$.

We would refer to p as the probability parameter, u as the input scale and ε as the output scale of algorithm $\tilde{f}$.

When the algorithm is deterministic, that is, $\mathbb{P}(\mathcal{A}_{x,\tilde{x}}) = 1$ for all $x, \tilde{x}$, then $|\tilde{F}| = 1$ and for simpler notation, we will drop the probability parameter in that case. More formally, we would simply say that the algorithm $\tilde{F}$ is a (u, ε)-algorithm for computing f on some subdomain S.

We now extend the definition of composition of functions Let $f : \mathbb{C}^M \to \mathcal{P}(\mathbb{C}^N)$ and $g : \mathbb{C}^N \to \mathcal{P}(\mathbb{C}^P)$. We define the following special composition operation.

$$\begin{aligned} g \circ_s f : \mathbb{C}^M &\to \mathcal{P}(\mathbb{C}^P) \\ x &\mapsto \bigcup_{y \in f(x)} g(y). \end{aligned} \tag{5.10}$$

We denote by $\circ$ the usual composition map for functions.

For a function $f : \mathbb{C}^M \to \mathbb{C}^N$ and for any subset $S \subseteq \mathbb{C}^M$, $f(S) = \bigcup_{x \in S} f(x)$.

Theorem 5.4.2. *Let $\tilde{F}$ be a $(p_f, u_f, \varepsilon_f)$-algorithm on probability space $\mathcal{P}_f = (\Omega_f, \mathcal{F}_f, \mathbb{P}_f)$ computing a function $f : \mathbb{C}^M \to \mathcal{P}(\mathbb{C}^N)$ on a domain $S_f \subseteq \mathbb{C}^M$. Let $g : \mathbb{C}^N \to \mathcal{P}(\mathbb{C}^P)$ be another function and let $\tilde{G}$ be a $(p_g, \varepsilon_f, \varepsilon_g)$-algorithm on probability space $\mathcal{P}_g = (\Omega_g, \mathcal{F}_g, \mathbb{P}_g)$ computing g on domain $f(S_f) \subseteq \mathbb{C}^N$.*

Let $h = g \circ_s f$ and $\tilde{H} = \{\tilde{g} \circ \tilde{f} | \tilde{g} \in \tilde{G}, \tilde{f} \in \tilde{F}\}$. Then $\tilde{H}$ is a $(p_g p_f, u_f, \varepsilon_g)$-algorithm on probability space $\mathcal{P}_g \times \mathcal{P}_f$ for computing h on S_f.

Proof. Let $x \in S_f$ and let $\tilde{x} \in B(x, u_f)$. Then we define

$$A^f_{x,\tilde{x}} := \{\omega \in \Omega_f | \text{ there exists some } y_f \in f(x) \text{ such that } ||\tilde{f}_\omega(\tilde{x}) - y_f|| \leq \varepsilon_f\}.$$

Since $\tilde{F}$ is a $(p_f, u_f, \varepsilon_f)$-algorithm for computing f on $\mathcal{P}_f$, then

$$\mathbb{P}_f(A^f_{x,\tilde{x}}) \geq p_f. \tag{5.11}$$

For a particular $\omega \in A_{x,\tilde{x}}$, let $y_f^\omega \in f(x) \subseteq f(S_f)$ such that $||\tilde{f}_\omega(\tilde{x}) - y_f^\omega|| \leq \varepsilon_f$. Since, $\tilde{G}$ is a $(p_g, \varepsilon_f, \varepsilon_g)$-algorithm, then we can define the corresponding sets

$$A^g_{y_f^\omega, \tilde{f}_\omega(\tilde{x})} := \{\omega' \in \Omega_g | \text{ there exists some } y_g \in g(y_f^\omega) \text{ such that } ||\tilde{g}_{\omega'}(\tilde{f}_\omega(\tilde{x})) - y_g|| \leq \varepsilon_g\}. \tag{5.12}$$

Now for the function h, we can also define the following set

$$A^h_{x,\tilde{x}} := \{\omega \in \Omega_h | \text{ there exists some } y_h \in h(x) \text{ such that } ||\tilde{h}_\omega(\tilde{x}) - y_h|| \leq \varepsilon_g\} \tag{5.13}$$

Then, since for all $h \in \tilde{H}$, there exists $f \in \tilde{F}$ and $g \in \tilde{g}$ such that $h = g \circ f$, it is clear that

$$\bigcup_{\omega \in A^f_{x,\tilde{x}}} \left(\{\omega\} \times A^g_{y_f^\omega, \tilde{f}_\omega(\tilde{x})} \right) \subseteq A^h_{x,\tilde{x}}.$$

Let $\omega_1 \in A^f_{x,\tilde{x}}$ and let $y_f^{\omega_1} \in f(x)$ such that $||\tilde{f}_{\omega_1}(\tilde{x}) - y_f^{\omega_1}|| \leq \varepsilon_f$. Since $\tilde{g}$ is a $(p_g, \varepsilon_f, \varepsilon_g)$-algorithm for computing g on $\mathcal{P}_g$, then

$$\mathbb{P}_g(A^g_{y_f^{\omega_1}, \tilde{f}_{\omega_1}(\tilde{x})}) \geq p_g. \tag{5.14}$$

Then using this and (5.11), along with the law of total probabilities, we get that

$$\begin{aligned} \mathbb{P}_h(A^h_{x,\tilde{x}}) &\geq (\mathbb{P}_g \times \mathbb{P}_f)\Big(\bigcup_{\omega \in A^f_{x,\tilde{x}}} \left(\{\omega\} \times A^g_{y_f^\omega, \tilde{f}_\omega(\tilde{x})} \right) \Big) \\ &\geq p_g p_f. \end{aligned}$$

□

Corollary 5.4.2.1. *Let $\tilde{f}$ be a (u_f, ε_f)-algorithm computing a function $f : \mathbb{C}^M \to \mathcal{P}(\mathbb{C}^N)$ on a domain $S_f \subseteq \mathbb{C}^M$. Let $g : \mathbb{C}^N \to \mathcal{P}(\mathbb{C}^P)$ be another function and let $\tilde{g}$ be a $(\varepsilon_f, \varepsilon_g)$-algorithm computing g on domain $f(S_f) \subseteq \mathbb{C}^N$.*

Let $h = g \circ_s f$ and $\tilde{h} = \tilde{g} \circ \tilde{f}$. Then $\tilde{h}$ is a (u_f, ε_g)-algorithm for computing h on S_f.

This can also be extended to composition of multiple functions using a similar proof.

5.5 Algorithm 7 as a composition of simple functions

In this section, we define some simple functions using the functions defined in Section 5.3 and show that Algorithm 7 can be written as a composition of these functions.

Some standard functions: We define the function $\pi_i : X_1 \times ... \times X_n \to X_i$ to be the projection function on the i-th coordinate for all $i \in [n]$. For two functions $f_1 : X \to Y$ and $f_2 : X \to Z$, we define

$$\begin{aligned} f_1 \times f_2 : X &\to Y \times Z \\ x &\mapsto (f_1(x), f_2(x)). \end{aligned}$$

We define the map that takes in a matrix and returns a tuple of its columns

$$\begin{aligned} \psi_{\text{matrow}} : M_n(\mathbb{C}) &\to (\mathbb{C}^n)^n \\ A &\mapsto (a_1, ..., a_n) \text{ where } a_i \text{ is the } i\text{-th row of } A \end{aligned}$$

For any space X, Id_X denotes the identity map on that space. For our applications, we will drop the subscript as the space will be clear from the context.

Recall that X_i are the domains for functions f_i defined in Section 5.3. We define the following maps

$$\begin{aligned} g_1 &= f_1 \times \pi_1 : X_1 \to M_n(\mathbb{C}) \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \\ g_2 &= (f_2 \circ \pi_1) \times \pi_2 \times \pi_3 : X_2 \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \to M_n(\mathbb{C}) \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \\ g_3 &= (f_3 \circ \pi_1) \times \pi_2 : X_3 \times (\mathbb{C}^n)^{\otimes 3} \to M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \\ g_4 &= (f_4 \circ \pi_1) \times \pi_2 : X_4 \times (\mathbb{C}^n)^{\otimes 3} \to \mathcal{P}(M_n(\mathbb{C})) \times (\mathbb{C}^n)^{\otimes 3} \\ g_5 &= (\psi_{\text{matrow}} \circ f_2 \circ \pi_1) \times \text{Id} : X_5 \times (\mathbb{C}^n)^{\otimes 3} \to (\mathbb{C}^n)^n \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \\ g_6 &= (f_6 \circ \pi_2) \times \pi_1 : (\mathbb{C}^n)^n \times X_6 \to \mathbb{C}^n \times (\mathbb{C}^n)^n \\ g_7 &= f_7 : X_7 \to \mathcal{P}((\mathbb{C}^n)^n) \end{aligned} \tag{5.15}$$

Each g_i corresponds to the i-th step of Algorithm 7 and hence Theorem 4.2.2 can be rewritten in the following way to show that the composition of these functions $h = g_7 \circ_s ... \circ_s g_1$ where $\circ_s$ is the composition defined in (5.10) outputs the set of all possible decompositions of the input tensor T.

Note: The function definitions g_i always don't exactly resemble the i-th step of Algorithm 7 - in a few cases, it has some elements that are carried forward from the previous steps. For example, the function g_1 takes as input the diagonalisable tensor T given as input to the algorithm and vectors a and b and computes matrices $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$. The output of this step is the tuple $(T^{(a)}, T^{(b)}, T)$. It imitates Step 1 of Algorithm 7, except for the part that it carries forward the input tensor T as well. This is for book-keeping purposes and the computation at each step of the algorithm remains unchanged.

When $f : X \to \mathcal{P}(Y_1) \times Y_2$ and $g : Y_1 \times Y_2 \to \mathcal{P}(Z)$, then the corresponding map $\circ_s$ can be defined as

$$\begin{aligned} g \circ_s f : f &\to \mathcal{P}(Z) \\ x &\mapsto \bigcup_{y_1 \in \mathcal{Y}_1} g(y_1, y_2) \text{ where } f(x) = (\mathcal{Y}_1, y_2) \in \mathcal{P}(Y_1) \times Y_2 \end{aligned}$$

This is essentially an extension of the $\circ_s$ defined in (5.10) using the inclusion map

$\mathcal{P}(Y_1) \times Y_2 \hookrightarrow \mathcal{P}(Y_1 \times Y_2)$. Usually for our applications, it would be assumed that the range of f matches the domain of g. When the domains do not match, then the output of $g \circ_s f$ can be assumed to be an empty set.

Note: The maps $f_i : X_i \to Y_i(n)$ for all $i \in \{1,2,3,5,6\}$ defined in Section 5.3 can be written as a map $f_i' : X_i \to \mathcal{P}(Y_i(n))$ where for all $x \in X_i$, $f_i'(x) = \{f_i(x)\}$. In the rest of this section, by abuse of notation, we use the map f_i to denote the corresponding map f_i', wherever applicable. For all $i \in \{1,2,3,5,6\}$, $|g_i(y_i)| = 1$

Lemma 5.5.1. *Let $x = (T, a, b) \in X_1$ be such that the following conditions are satisfied:*

- *T is a diagonalisable tensor.*
- *Let $T_1, ..., T_n$ be the slices of T. Then $T^{(a)} = \sum_{i=1}^n a_i T_i$ is an invertible matrix and $(T^{(a)})^{-1}T^{(b)}$ has distinct eigenvalues.*

We define function $h = g_7 \circ_s ... \circ_s g_1$. Then

$$h(x) \subseteq \mathcal{D}(T) := \{(u_1, ..., u_n) | u_i \in \mathbb{C}^n \text{ are linearly independent and } T = \sum_{i=1}^n u_i^{\otimes 3}\}.$$

Proof. From the discussion above, we know that the functions g_i essentially imitates the different steps of Algorithm 7. Hence, by Theorem 4.2.2, we get that $h(x) \subseteq \mathcal{D}(T)$. □

The opposite direction is true as well - this follows from Corollary 4.3.2.1, that if $T = \sum_{i=1}^n u_i^{\otimes 3}$ where $u_i \in \mathbb{C}^n$ are linearly independent, then the vectors u_i are unique up to permutation and multiplication by cube roots of unity.

5.6 Error analysis of Algorithm 8:

5.6.1 Writing Algorithm 8 as a composition of functions

Defining $\tilde{g}_i$: Recall the definitions of $\tilde{f}_{4,p}$ from Section C.2.1 and $\tilde{f}_i$ for $i \in [7] \setminus \{4\}$ from Sections 5.3.1 - 5.3.3 and Appendices C.2.1 - C.2.4. The algorithm $\tilde{g}_i$ computing g_i is defined in the same way as in (5.15), just replacing f_4 by $\tilde{f}_{4,p}$ and for all $i \in [7] \setminus \{4\}$ by $\tilde{f}_i$ wherever it occurs. We define the following maps

$$\begin{aligned}
\tilde{g_1} &= \tilde{f_1} \times \pi_1 : X_1 \to M_n(\mathbb{C}) \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \\
\tilde{g_2} &= (\tilde{f_2} \circ \pi_1) \times \pi_2 \times \pi_3 : X_2 \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \to \Big(M_n(\mathbb{C}) \times M_n(\mathbb{C})\Big) \times (\mathbb{C}^n)^{\otimes 3} \\
\tilde{g_3} &= (\tilde{f_3} \circ \pi_1) \times \pi_2 : X_3 \times (\mathbb{C}^n)^{\otimes 3} \to M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \\
\tilde{g}_{4,p,\omega} &= (\tilde{f}_{4,p,\omega} \circ \pi_1) \times \pi_2 : X_4 \times (\mathbb{C}^n)^{\otimes 3} \to M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3} \text{ for fixed parameter } p \\
&\quad \text{and internal choices for randomness } \omega \in \Omega \text{ (refer to Section C.2.1)} \\
\tilde{g_5} &= (\psi_{\text{matrow}} \circ \tilde{f_2} \circ \pi_1) \times \text{Id} : X_2 \times (\mathbb{C}^n)^{\otimes 3} \to (\mathbb{C}^n)^n \times \Big(M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3}\Big) \\
\tilde{g_6} &= (\tilde{f_6} \circ \pi_2) \times \pi_1 : (\mathbb{C}^n)^n \times X_6 \to \mathbb{C}^n \times (\mathbb{C}^n)^n \\
\tilde{g_7} &= \tilde{f_7} : X_7 \to (\mathbb{C}^n)^n
\end{aligned} \tag{5.16}$$

We define the family of algorithm $\tilde{g}_{4,p}$ in a similar way as we had defined $\tilde{f}_{4,p}$ in (C.2).

$$\tilde{g}_{4,p} := \{\tilde{g}_{4,p,\omega} | \omega \in \Omega\} \tag{5.17}$$

We define the composition of these functions

$$\tilde{h}_{p,\omega} = \tilde{g}_7 \circ ...\tilde{g}_{4,p,\omega} \circ ... \circ \tilde{g}_1 : X_1 \to (\mathbb{C}^n)^n.$$

Consequently we can define the family of functions

$$\tilde{h_p} := \{\tilde{h}_{p,\omega} | \omega \in \Omega\}. \tag{5.18}$$

Remark 5.6.1. *Recall from Section C.2.1 that $\tilde{f}_{4,p}$ is the set of functions $\{\tilde{f}_{4,p,\omega} | \omega \in \Omega\}$ with the underlying probability space $\mathcal{P} = (\Omega, \mathcal{F}, \mathbb{P})$. Then the same probability distribution applies over $\tilde{g}_{4,p}$ and $\tilde{h}_p$ as well. This is the probability space $\mathcal{P}$ that we will be referring to throughout this section.*

Note that $\tilde{g}_i$ corresponds to Step i of Algorithm 9 and hence $\tilde{h}$ corresponds to Algorithm 9.

We define the error parameter ε_i (which corresponds to the error at every step of the algorithm) in the following way:

$$\varepsilon_i = \begin{cases} \frac{1}{n^{c_i \log^2(\frac{nB}{\varepsilon})}} & i \in \{4,5,6,7\} \\ \frac{1}{n^{c_i \log^{12}(\frac{nB}{\varepsilon})}} & i \in \{0,1,2,3\} \end{cases} \tag{5.19}$$

where $c_0 = 128 \times (24)^4, c_{i+1} = \frac{c_i}{2}$ for $i \in \{0,1,2,4,5\}, c_4 = (\frac{c_3}{2})^{\frac{1}{4}}$ and $c_7 = \frac{c_6}{6}$. Note that this value of c_0 is set in Section 5.6.3 and this implies that for all $i \in [7]$, $c_i \geq 1$, which we will use later. We show in this section that $\tilde{h}_p$ (as defined in (5.18))is an $((1 - \frac{1}{n} - \frac{12}{n^2}), \varepsilon_0, \varepsilon_7)$-algorithm (following Definition 5.4.1) on probability space $\mathcal{P}$ (refer to Remark 5.6.1) for computing h on some subdomain. The following is the formal statement of this theorem and we give a proof of this in Section 5.6.2

Theorem 5.6.2. *We define the space $\mathcal{D}$ to be the set of all $y \in X_1$ which satisfy the (n, B)-input conditions (according to Definition 4.3.5) with parameters k_F, k_{gap}. Let $\varepsilon \leq 1$ be the desired accuracy parameter and we set $p := (\frac{\varepsilon_4}{2}, K_{eig}, K_{norm})$ where $K_{eig} = \frac{B}{k_{gap}}$, $K_{norm} = \sqrt{nB^3 k_F}$ and ε_4 is set in (5.19). Then $\tilde{h}_p$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_0, \varepsilon_7)$-algorithm with probability space $\mathcal{P}$ for computing h on subdomain $\mathcal{D}$ when each individual $\tilde{g}_i$ is run on a finite precision machine with precision ε_{i-1} for all $i \in [7]$.*

Rewriting Algorithm 8:

We first rewrite Algorithm 8 in terms of the newly-defined $\tilde{g}_i$'s.

Algorithm 9: Jennrich's Algorithm for Complete Decomposition of Symmetric Tensors.

Let $C_{\text{gap}}, C_\eta > 0$ and $c_F > 1$ be some absolute constants we will fix in (6.6).
Input: An order-3 symmetric diagonalisable tensor $T \in (\mathbb{C}^n)^{\otimes 3}$, an estimate B for the condition number of the tensor and an accuracy parameter $\varepsilon \leq 1$.
Output: A solution to the ε-approximation problem for the decomposition of T.
Set $k_{\text{gap}} := \frac{1}{C_{\text{gap}} n^6 B^3}$ and $k_F := c_F n^5 B^3$.
Pick $(a_1, ..., a_n, b_1, ..., b_n) \in G_\eta$ uniformly at random where $\eta := \frac{1}{C_\eta n^{\frac{17}{2}} B^4}$ is the grid size.
Let $\delta := \frac{1}{n^{c_4 \log^2(\frac{nB}{\varepsilon})}}$ where c_4 is a constant we will set in Section 5.6.3 and define $p = (\delta, \frac{B}{k_{\text{gap}}}, B^{\frac{3}{2}}\sqrt{nk_F})$.

1 Let $\tilde{y}_1 = (\tilde{T}, \tilde{a}, \tilde{b})$. Compute $\tilde{g}_1(\tilde{y}_1) = (S^{(a)}, S^{(b)}, \tilde{T})$ where $(S^{(a)}, S^{(b)}) = \tilde{f}_1(\tilde{T}, \tilde{a}, \tilde{b})$
2 Let $\tilde{y}_2 = \tilde{g}_1(\tilde{y}_1)$. Compute $\tilde{g}_2(\tilde{y}_2) = (S^{(a)'}, S^{(b)}, \tilde{T})$ where $S^{(a)'} := \tilde{f}_2(S^{(a)})$.
3 Let $\tilde{y}_3 = \tilde{g}_2(\tilde{y}_2)$. Compute $\tilde{g}_3(\tilde{y}_3) = (D, \tilde{T})$ where $D := \tilde{f}_3(S^{(a)'}, S^{(b)})$.
4 Let $\tilde{y}_4 := \tilde{g}_3(\tilde{y}_3)$. Compute $\tilde{g}_{4,p,\omega}(\tilde{y}_4) = (V^{(0)}, \tilde{T})$ where $V^{(0)} := \tilde{f}_{4,p,\omega}(D)$ for some internal choice of randomness ω (refer to the discussion in Section C.2.1).
5 Let $\tilde{x_5} := \tilde{g}_{4,p}(\tilde{y}_4)$. Compute $\tilde{g}_5(\tilde{y}_5) = ((u'_1, ..., u'_n), V^{(0)}, \tilde{T})$ where $C := \tilde{f}_2(V^{(0)})$ and $u'_1, ..., u'_n$ be the rows of C.
6 Let $\tilde{x_6} := \tilde{g}_5(\tilde{y}_5)$. Compute $\tilde{g}_6(\tilde{y}_6) = (C, (\alpha'_1, ..., \alpha'_n))$ where $\alpha'_1, ..., \alpha'_n := \tilde{f}_6(V^{(0)}, \tilde{T})$.
7 Let $\tilde{y}_7 = \tilde{g}_6(\tilde{y}_6)$. Compute $\tilde{g}_7(\tilde{y}_7) = (l_1, ..., l_n)$ where $l_1, ..., l_n = \tilde{f}_7((\alpha'_1, ..., \alpha'_n), (u'_1, ..., u'_n))$.
Output $\{l_1, ..., l_n\}$.

5.6.2 Proof of Theorem 5.6.2:

In (5.16), we had defined $\tilde{g}_i$ to be the algorithm corresponding to Step i in Algorithm 9 for all $i \in [7]$. We start with y_1 and $\tilde{y}_1$ as the input for h and $\tilde{h}$ as defined in Theorem 5.6.2. We define the partial compositions

$$\begin{aligned} \tilde{h}_i &= \tilde{g}_i \circ ... \tilde{g}_1 \text{ for all } i \in [3] \\ \tilde{h}_{4,p} &= \tilde{g}_{4,p} \circ ... \circ \tilde{g}_1 \\ \tilde{h}_{i,p} &= \tilde{g}_i \circ ... \tilde{g}_{4,p} ... \tilde{g}_1 \text{ for all } i \in \{5,6,7\} \\ h_i &= g_i \circ ... g_1 \text{ for all } i \in [7]. \end{aligned} \tag{5.20}$$

Also, note that by this definition, $\tilde{h}_p = \tilde{h}_{7,p}$ and $h = h_7$. We define the spaces $\mathcal{D}_i(n)$ in the following way:

$$\mathcal{D}_i(n) = \begin{cases} \mathcal{D} & i = 1 \\ h_{i-1}(\mathcal{D}) & i \in \{2, ..., 7\} \end{cases} \tag{5.21}$$

Then we show the following statements:

- We start by showing in Section 5.6.2 that $\tilde{h}_1$ is an $(\varepsilon_0, \varepsilon_1)$ algorithm (following Definition 5.4.1) for computing h_1 on domain $\mathcal{D}_1$.

- Then in the following sections, for $i \in \{2,3,5,6,7\}$, we show that $\tilde{g}_i$ is an $(\varepsilon_{i-1}, \varepsilon_i)$ algorithm for computing g_i on domain $\mathcal{D}_i$. And we also show that for the parameters p set in the hypothesis of Theorem 4.3.6, $\tilde{g}_{4,p}$ is a $(1 - \frac{1}{n} - \frac{12}{n^2})$-algorithm on the probability space $\mathcal{P}$ (defined in the statement of Theorem 4.3.6) for computing g_4 on domain $\mathcal{D}_4$. Using Theorem 5.4.2 and Corollary 5.4.2.1 wherever applicable, we can then inductively show the following
  - $\tilde{h}_i$ is an $(\varepsilon_0, \varepsilon_i)$ algorithm for computing h_i on domain $\mathcal{D}_i$ for $i \in \{2,3\}$.
  - $\tilde{h}_{i,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_0, \varepsilon_i)$ algorithm on the probability space $\mathcal{P}$ for computing h_i on domain $\mathcal{D}_i$ for $i \in \{4,5,6,7\}$.

For better readability of this exposition, we keep the analysis for $\tilde{h}_1, \tilde{h}_2, \tilde{h}_3$ and $\tilde{h}_{4,p}$ in the main text and defer the proofs for the rest to Appendix C.4.

Starting with y_1:

The goal of this section is to show that $\tilde{g}_1 = \tilde{h}_1$ is an $(\varepsilon_0, \varepsilon_1)$-algorithm for computing $g_1 = h_1$ on domain $\mathcal{D}_1 = \mathcal{D}$.

Recall that we had defined the space $X_1 := (\mathbb{C}^n)^{\otimes 3} \times \mathbb{C}^n \times \mathbb{C}^n$ in Section 5.3.1. Let $y_1 \in \mathcal{D}_1$ and $\tilde{y}_1 \in B(y_1, \varepsilon_0)$. Then the inputs to functions $\tilde{g}_1$ and g_1 are $\tilde{y}_1$ and y_1 respectively. Following the definition of g_1 in (5.15) and $\tilde{g}_1$ in (5.16), consequently the inputs to $\tilde{f}_1$ and f_1 are set as $\tilde{x}_1 = \tilde{y}_1$ and $x_1 = y_1$ respectively.

Relation to Step 1 of the Algorithm: From the definition of $\mathcal{D}$, we know that $y_1 = x_1 = (T, a, b) \in \mathcal{D}_1 \subseteq X_1$. Following the definition of g_1 from (5.15), we get that it takes as input y_1 and outputs $(f_1(x_1), T)$. Following the definition of f_1 from Section 5.3.1 we get that $f_1(x_1) = (T^{(a)}, T^{(b)})$ where $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$.

We also have that

$$||x_1 - \tilde{x}_1|| = ||y_1 - \tilde{y}_1|| \leq \varepsilon_0. \tag{5.22}$$

First, we bound the norm of a diagonalisable tensor by a function of its condition number.

Lemma 5.6.3. *Let T be an order-3 diagonalisable tensor. Then $||T||_F \leq (\kappa(T))^{\frac{3}{2}}$.*

Proof. Let $T = \sum_{i=1}^n u_i^{\otimes 3}$ where the u_i's are linearly independent. Let $U \in \mathrm{GL}_n(\mathbb{C})$ be the matrix with rows $u_1, ..., u_n$. From Corollary 3.2.1.1, we get that the slices T_i of T can be written as $T_i = U^T D_i U$ where $D_i = \mathrm{diag}(u_{1,i}, ..., u_{n,i})$. Therefore,

$$\begin{aligned}
||T||_F^2 = \sum_{i=1}^n ||T_i||_F^2 &= \sum_{i=1}^n ||U^T D_i U||_F^2 \\
&\leq ||U||_F^4 \sum_{i=1}^n ||D_i||_F^2 \\
&= ||U||_F^4 (\sum_{i=1}^n (\sum_{k=1}^n |u_{k,i}|^2)) \\
&= ||U||_F^6 \leq \kappa(T)^3.
\end{aligned}$$

□

In Section 5.3.1, we have defined the condition number for f_1 as $\kappa_1(x) = \sqrt{||x||^2 + 2}$ for some $x \in X_1$ where $X_1 = (\mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n) \times \mathbb{C}^n \times \mathbb{C}^n$.

Since, $y_1 = (T, a, b) \in \mathcal{D}$, we already have that $\kappa(T) \leq B$ and $a, b \in [-1, 1]^n$. Using this along with Lemma 5.6.3 on T gives us that for all $x_1 = y_1 \in \mathcal{D}$,

$$\kappa_1(x_1) = \sqrt{||x_1||^2 + 2} = \sqrt{||T||_F^2 + ||a||^2 + ||b||^2 + 2} \leq \sqrt{2n + B^3 + 2}. \tag{5.23}$$

Putting this in Theorem 5.3.11 along with (5.22), we get that using Lemma C.1.1 that for large enough n and some appropriate constant m_1',

$$||\tilde{f}_1(\tilde{x}_1) - f_1(x_1)|| \leq \varepsilon_0.(n\Big(\kappa_1(x_1)\Big)^{\log n})^{m_1} \leq \varepsilon_0(nB)^{m_1' \log(n)} \leq \frac{\sqrt{\varepsilon_0}}{2} = \frac{\varepsilon_1}{2}. \tag{5.24}$$

Now following the definitions of g_i, h_i and $\tilde{g}_i, \tilde{h}_i$ from (5.15) and (5.20), we have

$$\begin{aligned} ||\tilde{h}_1(\tilde{y}_1) - h_1(y_1)|| &= ||\tilde{g}_1(\tilde{y}_1) - g_1(y_1)|| \\ &= \sqrt{||\tilde{f}_1(\tilde{x}_1) - f_1(x_1)||^2 + ||\pi_1(\tilde{x}_1) - \pi_1(x_1)||^2} \\ &\leq \sqrt{\frac{\varepsilon_1^2}{4} + \varepsilon_0^2} \leq \varepsilon_1. \end{aligned} \tag{5.25}$$

We can finally conclude that $\tilde{h}_1$ is an $(\varepsilon_0, \varepsilon_1)$ algorithm for computing h_1 on domain X_1.

Setting y_2:

The goal of this section is to show that $\tilde{g}_2$ is an $(\varepsilon_1, \varepsilon_2)$-algorithm for computing g_2 on domain $\mathcal{D}_2 = h_1(\mathcal{D}_1)$. We also use the conclusion of Section 5.6.2, that is, $\tilde{h}_1$ is an $(\varepsilon_0, \varepsilon_1)$-algorithm for computing h_1 on subdomain $\mathcal{D}_1$. Since, $h_2 = g_2 \circ_s h_1$ and $\tilde{h}_2 = \tilde{g}_2 \circ \tilde{h}_1$, using Corollary 5.4.2.1, we get that $\tilde{h}_2$ is an $(\varepsilon_0, \varepsilon_2)$-algorithm for computing h_2 on domain $\mathcal{D}_1$.

Recall that we had defined the space $X_2 := \mathrm{GL}_n(\mathbb{C})$ in Section 5.3.2. We pick some $y_2 \in \mathcal{D}_2 \subseteq X_2 \times M_n(\mathbb{C}) \times (\mathbb{C}^n)^{\otimes 3}$ (where $\mathcal{D}_2 = h_1(\mathcal{D})$ as defined in (5.21)) and $\tilde{y}_2 \in B(y_2, \varepsilon_1)$. The inputs to functions $\tilde{g}_2$ and g_2 will be $\tilde{y}_2$ and y_2 respectively. Following the definition of g_2 in (5.15) and $\tilde{g}_2$ in (5.16), consequently the inputs to $\tilde{f}_2$ and f_2 are set as $\tilde{x}_2 = \pi_1(\tilde{y}_2)$ and $x_2 = \pi_1(y_2)$ respectively.

Relation to Step 2 of the Algorithm: From the definition of h_1, the input to g_2, denoted by y_2 has the following structure: $y_2 = (T^{(a)}, T^{(b)}, T) \in \mathcal{D}_2$ (refer to Section 5.6.2 and Step 2 of Algorithm 9). Recall that f_2 is the function corresponding to matrix inversion defined in Section 5.3.2. Following the definition of $g_2 = (f_2 \circ \pi_1) \times \pi_2 \times \pi_3$, this consequently gives us that the input to f_2 is $x_2 = \pi_1(y_2) = T^{(a)} \in X_2$. Application of g_2 on y_2, performs matrix inversion on the first coordinate and leaves the rest of the coordinate unchanged. More formally,

$$g_2(y_2) = (f_2(x_2), T^{(b)}, T) = ((T^{(a)})^{-1}, T^{(b)}, T) \tag{5.26}$$

Since, $||\tilde{y}_2 - y_2|| \leq \varepsilon_1$, this in turn also implies that

$$||\tilde{x}_2 - x_2|| \leq \varepsilon_1. \tag{5.27}$$

Moreover, from the assumption that the input $y_1 \in \mathcal{D}$, more specifically, that it satisfies (n, B)-input Condition 4.3.5, we have that the Frobenius condition number (defined in Section 3.3) of $x_2 = T^{(a)}$ is bounded by $k_F := c_F n^5 B^3$. Recall that we had defined the condition number for f_2 in Section 5.3.2 as $\kappa_2 = \kappa_F$. Then this gives

us that

$$\kappa_2(x_2) = \kappa_F(T^{(a)}) \leq k_F := c_F n^5 B^3. \tag{5.28}$$

Following the definition of ε_i from (5.19) and using Lemma C.1.1, we already have that

$$\varepsilon_1.\sqrt{\kappa_2(x_2)} = \frac{1}{n^{c_1 \log^{12}(\frac{nB}{\varepsilon})}}.\sqrt{c_F} n^{\frac{5}{2}} B^{\frac{3}{2}} \leq \frac{\sqrt{\varepsilon_1}}{2} \leq \frac{1}{2} \tag{5.29}$$

Recall the definition of the space $I_2(\varepsilon_1)$ as defined in Section 5.3.2 (and Theorem 5.3.11) which is the subdomain of f_2 on which $\tilde{f}_2$ is a robust numerically stable algorithm for computing f_2. Using (5.29), we can thus conclude that $T^{(a)} \in I_2(\varepsilon_1)$. Putting this in Theorem 5.3.11 and using (5.27) and Lemma C.1.1, we get that for large enough n and for some appropriate constant m_2',

$$||\tilde{f}_2(\tilde{x}_2) - f_2(x_2)|| \leq \varepsilon_1.(n\Big(\kappa_2(x_2)\Big)^{\log n})^{m_2} \leq \varepsilon_1 (nB)^{m_2' \log(n)} \leq \frac{\sqrt{\varepsilon_1}}{2} = \frac{\varepsilon_2}{2}. \tag{5.30}$$

The final inequality follows from the definition of ε_i in (5.19).

Following the definition of $\tilde{g}_2$ and g_2 from (5.15), we can conclude that

$$\begin{aligned} &||\tilde{g}_2(\tilde{y}_2) - g_2(y_2)|| \\ =&\sqrt{||\tilde{f}_2(\pi_1(\tilde{y}_2)) - f_2(\pi_1(y_2))||^2 + ||\pi_2(\tilde{y}_2) - \pi_2(y_2)||^2 + ||\pi_3(\tilde{y}_2) - \pi_3(y_2)||^2} \\ \leq&\sqrt{\frac{\varepsilon_2^2}{4} + \varepsilon_1^2} \leq \varepsilon_2. \end{aligned} \tag{5.31}$$

This shows that $\tilde{g}_2$ is an $(\varepsilon_1, \varepsilon_2)$-algorithm for computing g_2 on the space $\mathcal{D}_2$ defined in (5.21).

For any $y \in h_2(\mathcal{D}_1)$, we also compute a bound on $||y||$ which we will require later in Section 5.6.2. Then, following the previous discussion, there exists some $y_1 = (T, a, b) \in \mathcal{D}_1$ such that $y = h_2(y_1) = ((T^{(a)})^{-1}, T^{(b}, T)$.

Lemma 5.6.4. *Let $U \in M_n(\mathbb{C})$ with rows $u_1, ..., u_k$ be such that $\kappa_F(U) \leq B$. Then, given $\boldsymbol{a} \in [-1, 1]^n$, $\sum_{k\in[n]} |\langle a, u_k\rangle|^2 \leq nB$.*

Proof. By the Cauchy-Schwarz inequality,

$$\sum_{k\in[n]} |\langle a, u_k\rangle|^2 \leq \sum_{k\in[n]} ||a||^2 ||u_k||^2 = ||a||^2 ||U||_F^2.$$

Since $\mathbf{a} \in [-1, 1]^n$, we know that $||a||^2 \leq n$. Hence $\sum_{k\in[n]} |\langle a, u_k\rangle|^2 \leq nB$. □

This gives us that $||T^{(b)}||_F \leq \sqrt{nB^3}$. Recall that since $y_1 = (T, a, b) \in \mathcal{D}$, we have that $\kappa(T) \leq B$ and using Lemma 5.6.3, this implies $||T||_F \leq B^{\frac{3}{2}}$. Using (5.29), we also get that $||(T^{(a)})^{-1}||_F^2 \leq \kappa_F(T^{(a)}) \leq c_F n^5 B^3$. Then we can finally conclude that

$$||y|| = \sqrt{||(T^{(a)})^{-1}||_F^2 + ||T^{(b)}||_F^2 + ||T||_F^2} \leq \sqrt{c_F n^5 B^3 + nB^3 + B^3} < \sqrt{3 c_F n^5 B^3}. \tag{5.32}$$

Setting y_3:

The goal of this section is to show that $\tilde{g}_3$ is an $(\varepsilon_2, \varepsilon_3)$-algorithm for computing g_3 on domain $\mathcal{D}_3 = h_2(\mathcal{D}_1)$. We also use the conclusion of Section 5.6.2, that is, $\tilde{h}_2$ is an $(\varepsilon_0, \varepsilon_2)$-algorithm for computing h_2 on subdomain $\mathcal{D}_1$. Since, $h_3 = g_3 \circ_s h_2$

and $\tilde{h}_3 = \tilde{g}_3 \circ \tilde{h}_2$, using Corollary 5.4.2.1, we get that $\tilde{h}_3$ is an $(\varepsilon_0, \varepsilon_3)$-algorithm for computing h_3 on domain $\mathcal{D}_1$.

Recall from Section 5.3.3 that we had defined $X_3 = M_n(\mathbb{C}) \times M_n(\mathbb{C})$. We denote by $y_3 \in \mathcal{D}_3 \subseteq X_3 \times (\mathbb{C}^n)^{\otimes 3}$ the input to g_3 and $\tilde{y}_3 \in B(y_3, \varepsilon_1)$ the input to $\tilde{g}_3$. Consequently the inputs to $\tilde{f}_3$ and f_3 are set as $\tilde{x}_3 = \pi_1(\tilde{y}_3)$ and $x_3 = \pi_1(y_3)$ respectively.

Relation to Step 3 of the Algorithm: From the definition of g_3 in (5.15), it takes in as input $y_3 = \left((T^{(a)})^{-1}, T^{(b)}), T\right) \in \mathcal{D}_3 \subseteq X_3 \times (\mathbb{C}^n)^{\otimes 3}$. Since $g_3 = (f_3 \circ \pi_1) \times \pi_2 \times \pi_3$, this consequently gives us that the input to f_3 is $x_3 = \pi_1(y_3) = ((T^{(a)})^{-1}, T^{(b)})$. Application of g_3 on y_3 performs matrix multiplication on the first coordinate and leaves the last coordinate unchanged. More formally, $g_3(y_3) = (f_3(x_3), T)$. Following the definition of f_3 in Section 5.3.3, we get that $g_3(y_3) = ((T^{(a)})^{-1}T^{(b)}, T)$. The following is the main result of this section.

Claim 5.6.5. *$\tilde{g}_3$ is an $(\varepsilon_2, \varepsilon_3)$-algorithm for computing g_3 on domain $\mathcal{D}_3$.*

Proof. Let $y_3 \in \mathcal{D}_3$ and $\tilde{y}_3 \in B(y_3.\varepsilon_2)$. We define $x_3 = \pi_1(y_3)$ and $\tilde{x}_3 = \pi_1(\tilde{y}_3)$ as the inputs to f_3 and $\tilde{f}_3$ respectively. Since, $||\tilde{y}_3 - y_3|| \leq \varepsilon_2$, it follows that

$$||\tilde{x}_3 - x_3|| \leq \varepsilon_2. \tag{5.33}$$

In Section 5.3.3, we had defined the condition number for Step 3 to be $\kappa_3(x) = \sqrt{2}||x|| + 1$.

Since, $x_3 = \pi_1(y_3)$ where $y_3 \in \mathcal{D}_3 = h_2(\mathcal{D}_1)$, using (5.32), we get that

$$\begin{aligned}\kappa_3(x_3) = \sqrt{2}||x_3|| + 1 &\leq \sqrt{2}||y_3|| + 1 \\ &\leq \sqrt{6c_F n^5 B^3} + 1.\end{aligned}$$

Putting this in Theorem 5.3.11 along with (5.33) and using Lemma C.1.1, we get that for large enough n and for some appropriate constant m_3',

$$||\tilde{f}_3(\tilde{x}_3) - f_3(x_3)|| \leq \varepsilon_2.(n\Big(\kappa_3(x_3)\Big)^{\log n})^{m_3} \leq \varepsilon_2(nB)^{m_3' \log(n)} \leq \frac{\sqrt{\varepsilon_2}}{2} = \frac{\varepsilon_3}{2}. \tag{5.34}$$

Following the definition of g_3 and $\tilde{g}_3$ from (5.15), this further gives us

$$\begin{aligned}||\tilde{g}_3(\tilde{y}_3) - g_3(y_3)|| &= \sqrt{||\tilde{f}_3(\pi_1(\tilde{y}_3)) - f_3(\pi_1(y_3))||^2 + ||\pi_2(\tilde{y}_3) - \pi_2(y_3)||^2} \\ &\leq \sqrt{\frac{\varepsilon_3^2}{4} + \varepsilon_2^2} \leq \varepsilon_3.\end{aligned} \tag{5.35}$$

This shows that $\tilde{g}_3$ is an $(\varepsilon_2, \varepsilon_3)$-algorithm for computing g_3 on the space $\mathcal{D}_3$ defined in (5.21). □

Setting y_4:

The goal of this section is to show that for the parameters p mentioned in the hypothesis of Theorem 4.3.6, $\tilde{g}_{4,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_3, \varepsilon_4)$-algorithm on probability space $\mathcal{P}$ for computing g_4 on domain $\mathcal{D}_4 = h_3(\mathcal{D}_1)$. We also use the conclusion of Section 5.6.2, that is, $\tilde{h}_3$ is an $(1, \varepsilon_0, \varepsilon_3)$-algorithm for computing h_3 on subdomain $\mathcal{D}_1$. Since, $h_4 = g_4 \circ_s h_3$ and $\tilde{h}_{4,p} = \tilde{g}_{4,p} \circ \tilde{h}_3$, using Corollary 5.4.2.1, we get that $\tilde{h}_{4,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_0, \varepsilon_4)$-algorithm on probability space $\mathcal{P}$ for computing h_4 on domain $\mathcal{D}_1$.

Recall from Section C.2.1 that we had defined X_4 to be set of all diagonalisable matrices and in (5.21), we had defined the subdomain $\mathcal{D}_4 = h_3(\mathcal{D})$.

Recall from the hypothesis of Theorem 4.3.6 that we had set the parameter $p = (\frac{\varepsilon_4}{2}, K_{\text{eig}}, K_{\text{norm}})$ where $K_{\text{eig}} = \frac{B}{k_{\text{gap}}}$ and $K_{\text{norm}} = \sqrt{nB^3k_F}$. Recall from (5.15) and the discussion in Remark 5.6.1, that $\tilde{g}_{4,p} := \{\tilde{g}_{4,p,\omega} | \omega \in \Omega\}$ with an associated probability space $\mathcal{P} = (\Omega, \mathcal{F}, \mathbb{P})$. The main goal of this subsection is the following result.

Claim 5.6.6. *For parameters $p = (\frac{\varepsilon_4}{2}, K_{eig}, K_{norm})$, $\tilde{g}_{4,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_3, \varepsilon_4)$-algorithm on probability space $\mathcal{P} = (\Omega, \mathcal{F}, \mathbb{P})$ (refer to Definition 5.4.1) for computing g_4 on domain $\mathcal{D}_4$.*

Relation to Step 4 of the algorithm: From the definition of h_3, it takes in $(T, a, b) \in \mathcal{D}$ and outputs $y_4 = \left((T^{(a)})^{-1}T^{(b)}, T\right) \in \mathcal{D}_4$ (refer to the Step 4 of Algorithm 9 for the details.) Let the first coordinate be denoted by $x_4 = (T^{(a)})^{-1}T^{(b)}$. Application of g_4 on y_4 performs matrix diagonalisation in the first coordinate and leaves the second coordinate unchanged. More formally, $g_4(y_4) = (V, T) = (f_4(x_4), T)$ such that for all columns v_i of V, $||v_i|| = 1$ where f_4 is the function for matrix diagonalisation defined in Section C.2.1.

Proof. Let $y_4 \in \mathcal{D}_4$ and $\tilde{y}_4 \in B(y_4, \varepsilon_3)$. By the definition of $\tilde{g}_{4,p}$ from (5.16) and the discussion in Remark 5.6.1, we get that for all $\omega \in \Omega$, $\tilde{g}_{4,p,\omega} = (\tilde{f}_{4,p,\omega} \circ \pi_1) \times \pi_2$. Then the corresponding inputs to $\tilde{f}_{4,p,\omega}$ and f_4 are set to be $\tilde{x}_4 := \pi_1(\tilde{y}_4)$ and $x_4 := \pi_1(y_4)$ respectively. This already implies that $\tilde{x}_4 \in B(x_4, \varepsilon_3)$.

From the hypothesis of Theorem 5.6.2, we already have that $y_1 = (T, a, b)$ satisfies the (n, B)-input Conditions 4.3.5 with parameters k_{gap} and k_F. Hence, $\text{gap}((T^{(a)})^{-1}T^{(b)}) \geq k_{\text{gap}}$, $||(T^{(a)})^{-1}||_F \leq k_F$ and $\kappa(T) \leq B$. Also, note that using Lemma 5.6.4, this implies that $||T^{(b)}||_F \leq \sqrt{nB^3}$.

Recall the definitions of κ_V^F and κ_{eig} from Section 3.3. Let U be a matrix that diagonalises the given tensor T. Firstly, from the proof of Theorem 4.2.1, we know that $(T^{(a)})^{-1}T^{(b)}$ is diagonalisable and the columns of U^{-1} form the eigenvectors of $(T^{(a)})^{-1}T^{(b)}$ as well. Hence,

$$\kappa_V^F(x_4) = \kappa_V^F((T^{(a)})^{-1}T^{(b)}) \leq \kappa_F(U^{-1}) = \kappa(T) \leq B. \tag{5.36}$$

Combining this with the fact $\text{gap}((T^{(a)})^{-1}T^{(b)}) \geq k_{\text{gap}} > 0$, we get that

$$\kappa_{\text{eig}}((T^{(a)})^{-1}T^{(b)}) = \frac{\kappa_V^F((T^{(a)})^{-1}T^{(b)})}{\text{gap}((T^{(a)})^{-1}T^{(b)})} \leq \frac{B}{k_{\text{gap}}} =: K_{\text{eig}}.$$

We also have that

$$||x_4|| = ||(T^{(a)})^{-1}T^{(b)}||_F \leq ||(T^{(a)})^{-1}||_F||T^{(b)}||_F \leq \sqrt{nB^3k_F} =: K_{\text{norm}}.$$

This gives us that x_4 satisfies parameters p as defined in Definition C.2.1. Using the fact that $\varepsilon \leq 1$ and $c_4 > 1$, we get that for large enough n,

$$\begin{aligned}\log^4\left(\frac{2nK_{\text{eig}}K_{\text{norm}}}{\varepsilon_4}\right) &= \log^4\left(2nK_{\text{eig}}K_{\text{norm}} \cdot n^{c_4 \log^2(\frac{nB}{\varepsilon})}\right) \\ &= \left(\log(2nK_{\text{eig}}K_{\text{norm}}) + c_4 \log^2(\frac{nB}{\varepsilon})\log(n)\right)^4 \\ &\leq 16c_4^4 \log^8(\frac{nB}{\varepsilon})\log^4(n) \\ &\leq 16c_4^4 \log^{12}(\frac{nB}{\varepsilon})\end{aligned} \tag{5.37}$$

The first inequality follows from the fact that for large enough n, $2nK_{\text{norm}}K_{\text{eig}} \leq (nB)^c$ for some appropriate constant $c > 1$.

Using this, we can conclude that the machine precision required by $\tilde{f}_{4,p}$ is also

$$\varepsilon_3 = \frac{1}{n^{c_3 \log^{12}(\frac{nB}{\varepsilon})}} = \frac{1}{n^{16c_4^4 \log^{12}(\frac{nB}{\varepsilon})}} < \frac{1}{n^{\log^4(\frac{nK_{\text{eig}}K_{\text{norm}}}{\frac{\varepsilon_4}{2}})}}. \tag{5.38}$$

Since $\tilde{x}_4 \in B(x_4, \varepsilon_3)$, we can define the set

$$\mathcal{A}_{x_4,\tilde{x}_4} := \{\omega \in \Omega | \text{there exists } y_4^{(f)} \in f_4(x_4) \text{ such that } ||\tilde{f}_{4,p,\omega}(\tilde{x}_4) - y_4^{(f)}|| \leq \frac{\varepsilon_4}{2}\}$$

where Ω is the set of internal random choices for Algorithm 6 (as explained in Section C.2.1). Since, we have the bound on ε_3 using (5.38), we can now use Theorem 5.3.11 to show that $\mathbb{P}(\mathcal{A}_{x_4,\tilde{x}_4}) \geq 1 - \frac{1}{n} - \frac{12}{n^2}$. Following the definition of g_4 and $\tilde{g}_{4,p,\omega}$ from (5.15), we get that for all $\omega \in \mathcal{A}_{x_4,\tilde{x}_4}$,

$$\begin{aligned}&||\tilde{g}_{4,p,\omega}(\tilde{y}_4) - y_4^{(g)}|| \\ =&\sqrt{||\tilde{f}_{4,p,\omega}(\pi_1(\tilde{y}_4)) - y_4^{(f)}||^2 + ||\pi_2(\tilde{y}_4) - \pi_2(y_4)||^2} \\ \leq&\sqrt{\frac{\varepsilon_4^2}{4} + \varepsilon_3^2} \leq \varepsilon_4.\end{aligned} \tag{5.39}$$

Now we can also similarly define the set

$$\mathcal{A}^{(g)}_{y_4,\tilde{y}_4} := \{\omega \in \Omega | \text{there exists } y_4^{(g)} \in g_4(y_4) \text{ such that } ||\tilde{g}_{4,p,\omega} - g_4^{(f)}|| \leq \varepsilon_4\}. \tag{5.40}$$

Using (5.39), $\mathcal{A}_{x_4,\tilde{x}_4} \subseteq \mathcal{A}^{(g)}_{y_4,\tilde{y}_4}$ and hence, $\mathbb{P}(\mathcal{A}^{(g)}_{y_4,\tilde{y}_4}) \geq 1 - \frac{1}{n} - \frac{12}{n^2}$. This implies that $\tilde{g}_{4,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_3, \varepsilon_4)$-algorithm on probability space $\mathcal{P}$ for computing g_4 on subdomain $\mathcal{D}_4$. □

Bounds on norms and condition numbers of outputs of g_4: For some $y \in g_4(\mathcal{D}_4)$, we want to bound $||y||$, which we will need later in Section C.4.2. From the previous discussion, we get that $y = (V, T)$ such that for all columns v_i of V, $||v_i|| = 1$. Since, $g_4(\mathcal{D}_4) = h_4(\mathcal{D}_1)$, there exists some $y_1 = (T, a, b) \in \mathcal{D}_1$ such that $y = h_4(y_1)$. Since $y_1 \in \mathcal{D}_1$ (following Definition 5.21), we already have that $\kappa(T) \leq B$. Using Lemma 5.6.3, we already have that $||T||_F^2 \leq (\kappa(T))^3 \leq B^3$. Using this, we can finally conclude that

$$||y|| = \sqrt{||V||_F^2 + ||T||_F^2} = \sqrt{n + B^3}. \tag{5.41}$$

Using (5.36), we also know that for all $x_4 \in \pi_1(\mathcal{D}_4)$, $\kappa_V^F(x_4) \leq B$. Following the definition of g_4, $\pi_1(g_4(\mathcal{D}_4)) = f_4(\pi_1(\mathcal{D}_4))$. Hence, from Lemma 3.3.10 for $\delta = 0$ for all $V \in \pi_1(g_4(\mathcal{D}_4))$, we can conclude that

$$\kappa_F(V) \leq n + \frac{B^2}{4}. \tag{5.42}$$

5.6.3 Finishing the proof of Theorem 4.3.6

We state the theorem here again for completeness.

Theorem 5.6.7. *Let x be an input to the algorithm such that it satisfies the (n, B)-input conditions with parameters $k_F := c_F n^5 B^3, k_{gap} := \frac{1}{C_{gap} n^6 B^3}$ (from Definition 4.3.5) where the constants C_{gap}, c_F are set in (6.6). Let $\varepsilon \leq 1$ be the input accuracy parameter. Then on input (x, ε) Algorithm 8 outputs an ε-approximation to the tensor decomposition problem for T in*

$$O(n^3 + T_{MM}(n) \log^2 \frac{nB}{\varepsilon})$$

arithmetic operations on a floating point machine with

$$O(\log^{12}(\frac{nB}{\varepsilon}) \log n)$$

bits of precision, with probability at least $1 - \frac{1}{n} - \frac{12}{n^2}$.

We have already seen from the definitions that $\tilde{h}$ corresponds to Algorithm 9.

Computing the error: Let x be an input to Algorithm 9 that satisfies the (n, B)-input conditions according to Definition 4.3.5. Let $\varepsilon \leq 1$ be the input desired accuracy parameter. We set $c_7 = 2$ and using (5.19), the rest of the c_i can be fixed as well. This gives us that $c_0 = 128 \times (24)^4$. Using this and the fact that $n, B > 1$, we can conclude that $\varepsilon_7 = \frac{1}{n^{2 \log^2(\frac{nB}{\varepsilon})}} \leq \varepsilon$. From Theorem 5.6.2, we get that on input $\tilde{x} \in B(x, \varepsilon_0)$, Algorithm 9 outputs an ε-approximation to the tensor decomposition problem for T.

Computing the machine precision: Let us assume that the algorithm is run with precision $\mathbf{u} := \frac{\varepsilon_0}{2n^{\frac{3}{2}}}$ where ε_0 is defined in (5.19). Let $x = (T, a, b)$ be the exact input to the algorithm that satisfies the (n, B)-input conditions with parameters k_F, k_{gap} as mentioned in the statement of Theorem 4.3.6. Following the definition of the model in Section 1.2.2, we know that if $\tilde{T}$ is the actual input to Algorithm 8 (subsequently Algorithm 9), such that for all $i, j, k \in [n]$, $||\tilde{T}_{i,j,k} - T_{i,j,k}|| \leq \mathbf{u}$. In Step 1, we pick $a, b \in [-1, 1]^n$ at random. Following the definition of the model, we get that the Step 1 of the algorithm actually gets some $\tilde{a}, \tilde{b} \in \mathbb{C}^n$ such that for all $i \in [n]$, $|\tilde{a}_i - a_i| \leq \mathbf{u}$ and $|\tilde{b}_i - b_i| \leq \mathbf{u}$. Then

$$||\tilde{T} - T||_F \leq \mathbf{u}.n^{\frac{3}{2}}. \tag{5.43}$$

This gives us that $||\tilde{a} - a|| \leq \mathbf{u}\sqrt{n}$ and $||\tilde{b} - b|| \leq \mathbf{u}\sqrt{n}$.

Then using (5.43), we have that $||\tilde{x} - x|| \leq \sqrt{||\tilde{T} - T||_F^2 + ||\tilde{a} - a||^2 + ||\tilde{b} - b||^2} \leq \sqrt{\mathbf{u}^2 n^3 + \mathbf{u}^2 n + \mathbf{u}^2 n} \leq \varepsilon_0$. Using this relation, we can fix the number of bits of precision required by the algorithm to

$$\log(\frac{1}{u}) = \log(2n^{c_0 \log^{12}(\frac{nB}{\varepsilon}) + \frac{3}{2}}) = O(\log^{12}(\frac{nB}{\varepsilon}) \log(n)).$$

Conclusion: Using Theorem 5.6.2, we get that if the algorithm is given as input $\tilde{x} \in B(x, \varepsilon_0)$, then the algorithm returns an ε-approximation to tensor decomposition problem for T (as defined in Section 1.4.3) with probability at least $(1 - \frac{1}{n} - \frac{12}{n^2})$ when run on a machine with $O(\log^{12}(\frac{nB}{\varepsilon}) \log(n))$ bits of precision.

Chapter 6

Probability Analysis of Condition Numbers and Gap

6.1 Introduction

The central theme of this chapter is to deduce anti-concentration inequalities about certain families of polynomials arising in the analysis of Algorithm 8. Compared to [BCMV14], an interesting novelty of these inequalities is that the underlying distribution for the random variables is discrete and that they are applicable to polynomials from $\mathbb{R}^n$ to $\mathbb{C}$. In Section 6.2, we first study some polynomial norms and then prove these results.

Let $T \in (\mathbb{C}^n)^{\otimes 3}$ be a diagonalisable tensor given as input to Algorithm 8 with $\kappa(T) < B$, and let $T_1, ..., T_n$ be the slices of T. In the algorithm, we pick $a_1, ..., a_n, b_1, ..., b_n$ uniformly and independently at random from a finite discrete grid $G_\eta \subset [-1, 1]^{2n}$ and define $T^{(a)} = \sum_{i=1}^n a_i T_i$, $T^{(b)} = \sum_{i=1}^n b_i T_i$. In this section we show that (T, a, b) indeed satisfy the (n, B)-input conditions from Definition 4.3.5. More formally, we show that $T^{(a)}$ is invertible, $\mathrm{gap}((T^{(a)})^{-1}T^{(b)}) \geq k_{\mathrm{gap}} := \frac{1}{c_{\mathrm{gap}} n^6 B^3}$ and $\kappa_F(T^{(a)}) \leq k_F := c_F n^5 B^3$ with high probability. This is the main result of Section 6.3. We also justify the choice of k_{gap} and k_F and choose appropriate values for c_{gap} and c_F in Section 6.3.1. As a consequence of this, a central theorem arising out of this section is Theorem 4.3.4 which concludes the probability analysis of Algorithm 8. We state it here again for completeness.

Theorem 6.1.1. *Given a diagonalisable tensor T, a desired accuracy parameter ε and some estimate $B \geq \kappa(T)$, Algorithm 8 outputs an ε-approximate solution to the tensor decomposition problem for T in*

$$O(T_{MM}(n) \log^2 \frac{nB}{\varepsilon})$$

arithmetic operations on a floating point machine with

$$O(\log^{12}(\frac{nB}{\varepsilon}) \log n)$$

bits of precision, with probability at least $(1 - \frac{1}{n} - \frac{12}{n^2})(1 - \frac{1}{\sqrt{2n}} - \frac{1}{n})$.

6.2 Some definitions and bounds on norms of polynomials

We define the norm of a polynomial following Forbes and Shpilka [FS18]. Recall from Section 4.1.3 that their goal was to construct so-called "robust hitting sets".

Definition 6.2.1. *(Norm of a complex-valued polynomial) For an n-variate polynomial $f(x) \in \mathbb{C}[\boldsymbol{x}]$, we denote*

$$||f||_2 := \left(\int_{[-1,1]^n} |f(\boldsymbol{x})|^2 d\mu(x)\right)^{\frac{1}{2}} \tag{6.1}$$

where $\mu(x)$ is the uniform probability measure on $[-1,1]^n$. We also denote

$$||f||_\infty = \max_{v \in [-1,1]^n} |f(v)|.$$

Lemma 6.2.2. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Then, for all $k \in [n]$, $\sum_{i\in[n]} |u_{ik}|^2 \geq \frac{1}{B}$.*

Proof. Since $\kappa_F(U) \leq B$, we already have $||U^{-1}||_F \leq \sqrt{B}$. Also, we know that $||U^{-1}|| \leq ||U^{-1}||_F$. Hence, $||U^{-1}|| \leq \sqrt{B}$. By definition of the matrix norm, $||U^{-1}x|| \leq \sqrt{B}||x||$ for all $x \in \mathbb{C}^n$. We define $x = Uy$ and this shows that $||Uy|| \geq \varepsilon||y||$ where $\varepsilon = \frac{1}{\sqrt{B}}$. Let u_k be the k-th column of U . Then $||u_k||^2 \geq \varepsilon^2$. Hence, $\sum_{i\in[n]} |u_{ik}|^2 \geq \varepsilon^2 = \frac{1}{B}$. □

Inequalities: One of the most popular applications of inequalities for probabilities typically uses the idea that (under certain assumptions) for a random variable, the probability that it belongs to an interval which is far away from its expected value is small. These are called concentration inequalities and some examples includes several well-known inequalities such as Chebyshev's inequality, Chernoff bounds, Hoeffding's inequality etc. In this section, we focus on inequalities which in principle, try to achieve the opposite. Usually the goal is to show that for a random variable, the probability that it belongs to an interval of small length is small, irrespective of the choice of the location of the interval. One of the first such anti-concentration type of inequality for linear combination of iid random variables (from specific distributions) was discovered by Littlewood and Offord [LO38] and this sparked a series of such results for different families of polynomials with applications to combinatorics and complexity theory (refer to [Vu17] for a quick summary of the results). One of the most general of such results is the Carbery-Wright inequality ([CW01], Theorem 8) which applies to all polynomials and log-concave probability measures which we use in this section.

The inequality in the form that we use states that if the l_2 norm of a polynomial is not too small, then on inputs picked uniformly and independently at random from $[-1,1)^n$, the value of the polynomial is not too close to zero with high probability. We use the following presentation of the theorem from [FS18].

Theorem 6.2.3 (Carbery-Wright)**.** *There exists an absolute constant C_{CW} such that if $f : \mathbb{R}^n \to \mathbb{R}$ is a polynomial of degree at most d, then for $\alpha > 0$, it holds that*

$$Pr_{\boldsymbol{v}\in_U[-1,1)^n}[|f(\boldsymbol{v})| \geq \alpha] \geq 1 - C_{CW}d\left(\frac{\alpha}{||f||_2}\right)^{\frac{1}{d}}.$$

Theorem 6.2.4 (Carbery-Wright for complex-valued polynomials)**.** *There exists an absolute constant C_{CW} such that if $f : \mathbb{R}^n \to \mathbb{C}$ is a polynomial of degree at most d, then for $\alpha > 0$, it holds that*

$$Pr_{\boldsymbol{v}\in_U[-1,1)^n}[|f(\boldsymbol{v})| \geq \alpha] \geq 1 - 2C_{CW}d\left(\frac{\alpha}{||f||_2}\right)^{\frac{1}{d}}.$$

Proof. Since $f : \mathbb{R}^n \to \mathbb{C}$, we can write $f = \mathfrak{R}(f) + \iota\mathfrak{I}(f)$ where $\mathfrak{R}(f), \mathfrak{I}(f) \in \mathbb{R}^n \to \mathbb{R}$ are real polynomials of degree $\leq d$. Then using Theorem 6.2.3, we get that

$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|\mathfrak{R}(f)(\mathbf{v})| \geq \alpha] \geq 1 - C_{CW}d\left(\frac{\alpha}{||\mathfrak{R}(f)||_2}\right)^{\frac{1}{d}}$$
$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|\mathfrak{I}(f)(\mathbf{v})| \geq \alpha] \geq 1 - C_{CW}d\left(\frac{\alpha}{||\mathfrak{I}(f)||_2}\right)^{\frac{1}{d}}.$$

This gives us that

$$\begin{aligned}
&\Pr_{\mathbf{v}\in_U[-1,1)^n}[|f(v)|^2 \leq \alpha^2||f||_2^2] \\
&= \Pr_{\mathbf{v}\in_U[-1,1)^n}[|\mathfrak{R}(f)(\mathbf{v})|^2 + |\mathfrak{I}(f)(\mathbf{v})|^2 \leq \alpha^2\left(||\mathfrak{R}(f)||_2^2 + |\mathfrak{I}(f)||_2^2\right)] \\
&\leq \Pr_{\mathbf{v}\in_U[-1,1)^n}[|\mathfrak{R}(f)(\mathbf{v})|^2 \leq \alpha^2||\mathfrak{R}(f)||_2^2 \bigcup |\mathfrak{I}(f)(\mathbf{v})|^2 \leq \alpha^2||\mathfrak{I}(f)||_2^2] \\
&\leq \Pr_{\mathbf{v}\in_U[-1,1)^n}[|\mathfrak{R}(f)(\mathbf{v})|^2 \leq \alpha^2||\mathfrak{R}(f)||_2^2] + \Pr_{\mathbf{v}\in_U[-1,1)^n}[|\mathfrak{I}(f)(\mathbf{v})|^2 \leq \alpha^2||\mathfrak{I}(f)||_2^2] \\
&\leq 2C_{CW}d\alpha^{\frac{1}{d}}.
\end{aligned}$$

As a result,

$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|f(v)|^2 \geq \alpha^2] \geq 1 - 2C_{CW}d\left(\frac{\alpha}{||f||_2}\right)^{\frac{1}{d}}.$$

□

The next two theorems are directed towards applying the Carbery-Wright Theorem to a special polynomial which we will require later in Theorem 6.3.2. More specifically, let $U = (u_{ij})_{i,j\in[n]}$ be a matrix with bounded κ_F. Consider the linear form $P^k(\mathbf{x}) = \sum_{i\in[n]} p_i x_i$ where $p_i = u_{ik}$. We will apply the Carbery-Wright theorem to P^k.

Theorem 6.2.5. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Let $P^k(\boldsymbol{x}) = \sum_{i\in[n]} p_i x_i$ where $p_i = u_{ik}$. Then*

$$Pr_{\boldsymbol{v}\in_U[-1,1)^n}[|f(\boldsymbol{v})| \geq \frac{\alpha}{\sqrt{3B}}] \geq 1 - 2C_{CW}\alpha.$$

Proof. Applying Theorem 6.2.4 to P^k for $d = 1$, we get that

$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|f(\mathbf{v})| \geq \alpha||P^k||_2] \geq 1 - 2C_{CW}\alpha. \tag{6.2}$$

Now we claim that $||P^k||_2^2 \geq \frac{1}{B}$. Expanding using the definition of ℓ_2-norm of polynomials from (6.1), we have:

$$\begin{aligned}
||P^k||_2^2 &= \int_{[-1,1]^n} |\sum_{i\in[n]} p_i x_i|^2 d\mu(\mathbf{x}) \\
&= \int_{[-1,1]^n} |\Big(\sum_{k\in[n]} p_k^{(r)} x_k\Big) + \iota\Big(\sum_{k\in[n]} p_k^{(i)} x_k\Big)|^2 d\mu(\mathbf{x}) \\
&= \int_{[-1,1]^n} \Big(\sum_{k\in[n]} p_k^{(r)} x_k\Big)^2 d\mu(\mathbf{x}) + \int_{[-1,1]^n} \Big(\sum_{k\in[n]} p_k^{(i)} x_k\Big)^2 d\mu(\mathbf{x}) \\
&= \sum_{k,l\in[n]} p_k^{(r)} p_l^{(r)} \Big(\int_{[-1,1]^n} x_k x_l d\mu(\mathbf{x})\Big) + \sum_{k,l\in[n]} p_k^{(i)} p_l^{(i)} \Big(\int_{[-1,1]^n} x_k x_l d\mu(\mathbf{x})\Big).
\end{aligned}$$

When $i \neq k$, $\int_{[-1,1]^n} x_i x_j d\mu(\mathbf{x}) = (\frac{1}{2}\int_{-1}^{1} x_i dx_i)^2 = 0$. Therefore,

$$||P^k||_2^2 = \sum_{k\in[n]} \frac{(p_k^{(r)})^2 + (p_k^{(i)})^2}{2} \Big(\int_{-1}^{1} x_i^2 dx_i\Big)$$

Using the fact that $\frac{1}{2}\int_{-1}^{1} x_i^2 dx_i = \frac{1}{3}$, we have $||P^k||_2^2 = \frac{1}{3}\sum_{i\in[n]} |p_i|^2$. Lemma 6.2.2 already implies that

$$||P^k||_2^2 \geq \frac{1}{3B}.$$

Using this in (6.2), we conclude that

$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|f(\mathbf{v})| \geq \frac{\alpha}{\sqrt{3B}}] \geq 1 - 2C_{CW}\alpha.$$

□

The next two theorems are directed towards applying the Carbery-Wright theorem to another special polynomial which we will require later in Theorem 6.3.1. More specifically, let $U = (u_{ij})_{i,j\in[n]}$ be a matrix with bounded κ_F. Let $P^{kl}(\mathbf{x},\mathbf{y}) = \sum_{i,j\in[n]} p_{ij}^{kl} x_i y_j$ be the quadratic polynomial defined for all $k,l \in [n]$ by its coefficients $p_{ij}^{kl} = u_{ik}u_{jl} - u_{il}u_{jk}$. We will apply the Carbery-Wright theorem to P^{kl}. First we give a lower bound for the l_2 norm of the polynomial.

Lemma 6.2.6. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Then, for all $k,l \in [n]$, $\sum_{i,j\in[n]} |u_{ik}u_{jl} - u_{il}u_{jk}|^2 \geq \frac{2}{B^2}$.*

Proof. We construct a submatrix $U_2 \in M_{n,2}(\mathbb{C})$ with the k-th and l-th columns of U. Let $k = 1$ and $l = 2$ without loss of generality. Since $\kappa_F(U) \leq B$, following the proof of Lemma 6.2.2, for all $y \in \mathbb{C}^n$, $||Uy|| \geq \varepsilon||y||$ where $\varepsilon = \frac{1}{\sqrt{B}}$. Then for all $y \in \mathbb{C}^2$, we have $||U_2 y|| \geq \varepsilon||y||$. This implies that

$$\begin{aligned}
||U_2 y||^2 &\geq \varepsilon^2 ||y||^2 \\
y^* U_2^* U_2 y &\geq \varepsilon^2 y^* y.
\end{aligned}$$

The minimum singular value $\sigma_{\min}$ of U_2 is defined as $\sigma_{\min}^2 = \min_{y\in\mathbb{C}^n, y\neq 0} \frac{y^* U_2^* U_2 y}{y^* y}$. Therefore, $\sigma_{\min}^2(U_2) \geq \varepsilon^2$. Since $U_2^* U_2$ is a Hermitian matrix, $\sigma_{\min}^2(U_2) = \lambda_{\min}(U_2^* U_2)$

where $\lambda^2_{\min}$ refers to the smallest eigenvalue. This gives us that

$$\lambda_{\min}(U_2^* U_2) \geq \varepsilon^2.$$

Let $a = (a_1, ..., a_n)$ and $b = (b_1, ..., b_n)$ be the columns of U_2. Then $U_2^* U_2 = \begin{pmatrix} ||a||^2 & a^*b \\ b^*a & ||b||^2 \end{pmatrix}$. Also, $\det(U_2^* U_2) \geq \lambda^2_{\min}(U_2^* U_2)$, i.e.,

$$||a||^2||b||^2 - |a^*b|^2 \geq \lambda^2_{\min}(U_2^* U_2) \geq \varepsilon^4.$$

Now from the complex form of Lagrange's identity, we know that

$$||a||^2||b||^2 - |a^*b|^2 = \frac{1}{2}\sum_{i,j=1}^{n} |a_i b_j - a_j b_i|^2.$$

As a result, $\sum_{i,j=1}^{n} |a_i b_j - a_j b_i|^2 \geq 2\varepsilon^4$. Choosing $\varepsilon = \frac{1}{\sqrt{B}}$, we finally conclude that for all $k, l \in [n]$, $\sum_{i,j\in[n]} |u_{ik}u_{jl} - u_{il}u_{jk}|^2 \geq \frac{2}{B^2}$. □

Theorem 6.2.7. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Let $P^{kl}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i,j\in[n]} p_{ij} x_i y_j$ where $p_{ij} = u_{ik}u_{jl} - u_{il}u_{jk}$. Then*

$$Pr_{\boldsymbol{v}\in_U[-1,1)^n}[|f(\boldsymbol{v})| \geq \frac{\sqrt{2}\alpha}{3B}] \geq 1 - 4C_{CW}\alpha^{\frac{1}{2}}.$$

Proof. Applying Theorem 6.2.4 to P^{kl} with $d = 2$ shows that

$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|f(\mathbf{v})| \geq \alpha||P^{kl}||_2] \geq 1 - 4C_{CW}\alpha^{\frac{1}{2}}. \tag{6.3}$$

Now we claim that $||P^{kl}||_2 \geq \frac{\sqrt{2}}{3B}$. Recall that

$$||P^{kl}||_2^2 = \int_{[-1,1]^{2n}} |P^{kl}(\mathbf{x}, \mathbf{y})|^2 d\mu(\mathbf{x}, \mathbf{y})$$

where $\mu(\mathbf{x}, \mathbf{y})$ is the uniform probability distribution on $[-1, 1]^{2n}$. Let us define $p_{ij}^{(r)}$ and $p_{ij}^{(i)}$ as the real and imaginary parts respectively of p_{ij}. We can estimate $|P^{kl}||_2^2$ as follows:

$$\int_{[-1,1]^{2n}} |\sum_{i,j\in[n]} p_{ij}x_i y_j|^2 d\mu(\mathbf{x}, \mathbf{y}) = \int_{[-1,1]^{2n}} |\Big(\sum_{i,j\in[n]} p_{ij}^{(r)} x_i y_j\Big) + \iota\Big(\sum_{i,j\in[n]} p_{ij}^{(i)} x_i y_j\Big)|^2 d\mu(\mathbf{x}, \mathbf{y})$$

$$= \Big(\sum_{i,j,k,l\in[n]} p_{ij}^{(r)} p_{kl}^{(r)} \Big(\int_{[-1,1]^{2n}} x_i y_j x_k y_l d\mu(\mathbf{xy})\Big)\Big) + \Big(\sum_{i,j,k,l\in[n]} p_{ij}^{(i)} p_{kl}^{(i)} \Big(\int_{[-1,1]^{2n}} x_i y_j x_k y_l d\mu(\mathbf{xy})\Big)\Big)$$

$$= \sum_{i,j,k,l\in[n]} (p_{ij}^{(r)} p_{kl}^{(r)} + p_{ij}^{(i)} p_{kl}^{(i)}) \Big(\int_{[-1,1]^n} x_i x_k d\mu(\mathbf{x})\Big)\Big(\int_{[-1,1]^n} y_j y_l d\mu(\mathbf{y})\Big).$$

When $i \neq k$, $\int_{[-1,1]^n} x_i x_k d\mu(\mathbf{x}) = (\int_{[-1,1]} x_i d\mu(x_i))^2 = 0$. Similarly $\int_{[-1,1]^n} y_j y_l d\mu(\mathbf{y}) = 0$ for $j \neq l$. This gives us that

$$||P^{kl}||_2^2 = \sum_{i,j\in[n]} \Big((p_{ij}^{(r)})^2 + (p_{ij}^{(i)})^2\Big) \Big(\int_{[-1,1]^n} x_i^2 d\mu(\mathbf{x})\Big)\Big(\int_{[-1,1]^n} y_j^2 d\mu(\mathbf{y})\Big)$$

Since $\int_{[-1,1]^n} x_i^2 d\mu(\mathbf{x}) = \frac{1}{2}\int_{-1}^{1} x_i^2 dx_i = \int_{[-1,1]^n} y_j^2 d\mu(\mathbf{y}) = \frac{1}{2}\int_{-1}^{1} y_j^2 dy_j = \frac{1}{3}$, we get that

$$||P^{kl}||_2^2 = \frac{1}{9}\sum_{i,j\in[n]} |p_{ij}|^2.$$

Now, from Lemma 6.2.6, it follows that $||P^{kl}||_2^2 \geq \frac{2}{9B^2}$. Using this in (6.3), we can conclude that

$$\Pr_{\mathbf{v}\in_U[-1,1)^n}[|f(\mathbf{v})| \geq \frac{\sqrt{2}\alpha}{3B}] \geq (1 - 4C_{CW}\alpha^{\frac{1}{2}}).$$

□

Our next goal is to show a similar probabilistic result for both families of polynomials (linear and quadratic), but replacing the previous continuous distribution over $[-1,1)^n$ by a distribution where the inputs are chosen uniformly and independently at random from a discrete grid. To formalise this distribution, we describe another equivalent random process of picking an element at random from $[-1,1)^n$ and rounding it to the nearest point on the grid.

Remark: η is chosen so that $\frac{1}{\eta}$ is an integer so that the intervals of the grid have the same length and this ensures that the "picking unifromly at random and rounding" process is equivalent to "picking uniformly at random from the grid".

Theorem 6.2.8 (Multivariate Markov's Theorem)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a homogeneous polynomial of degree r, that for every $\boldsymbol{v} \in [-1,1]^n$ satisfies $|f(\boldsymbol{v})| \leq 1$. Then, for every $||\boldsymbol{v}|| \leq 1$, it holds that $||\nabla(f)(\boldsymbol{v})|| \leq 2r^2$ where ∇ denotes the gradient of a function.*

Theorem 6.2.9. *Let $f : \mathbb{R}^{2n} \to \mathbb{C}$ be a homogeneous polynomial of degree at most d. Let $\eta > 0$ be such that $\frac{1}{\eta}$ is an integer. Let $(\boldsymbol{a}, \boldsymbol{b}) \in [-1,1)^{2n}$ and $(\boldsymbol{a}', \boldsymbol{b}') = g_\eta(\boldsymbol{a}, \boldsymbol{b})$ where the rounding function is chosen for $m = 2n$. Then $|f(\boldsymbol{a}, \boldsymbol{b}) - f(\boldsymbol{a}', \boldsymbol{b}')| \leq 4\eta\sqrt{n}||f||_\infty d^2$.*

Proof. We write $f = \Re(f) + \iota\Im(f)$ where $\Re(f), \Im(f) : \mathbb{R}^n \to \mathbb{R}$. By the mean value theorem, there exists a point $(\mathbf{a}_0, \mathbf{b}_0)$ on the line segment connecting $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{a}', \mathbf{b}')$, such that $|\Re(f)(\mathbf{a}, \mathbf{b}) - \Re(f)(\mathbf{a}', \mathbf{b}')| = ||(\mathbf{a}, \mathbf{b}) - (\mathbf{a}', \mathbf{b}')|| \cdot |(\Re(f))'(\mathbf{a}_0, \mathbf{b}_0)|$ where $(\Re(f))'(\mathbf{a}_0, \mathbf{b}_0)$ is the derivative of $\Re(f)$ in the direction $(\mathbf{a}, \mathbf{b}) - (\mathbf{a}', \mathbf{b}')$ evaluated at $\mathbf{a}_0, \mathbf{b}_0$. From Theorem 6.2.8, it follows that $|(\Re(f))'(\mathbf{a}_0, \mathbf{b}_0)| \leq 2||\Re(f)||_\infty d^2$. Similarly, we also get that $|(\Im(f))'(\mathbf{a}_0, \mathbf{b}_0)| \leq 2||\Im(f)||_\infty d^2$. This finally gives us that

$$\begin{aligned}
|f(\mathbf{a}, \mathbf{b}) - f(\mathbf{a}', \mathbf{b}')| &= |\Big(\Re(f)(\mathbf{a}, \mathbf{b}) - \Re(f)(\mathbf{a}', \mathbf{b}')\Big) + \iota\Big(\Im(f)(\mathbf{a}, \mathbf{b}) - \Im(f)(\mathbf{a}', \mathbf{b}')\Big)| \\
&= \sqrt{\Big(\Re(f)(\mathbf{a}, \mathbf{b}) - \Re(f)(\mathbf{a}', \mathbf{b}')\Big)^2 + \Big(\Im(f)(\mathbf{a}, \mathbf{b}) - \Im(f)(\mathbf{a}', \mathbf{b}')\Big)^2} \\
&\leq ||(\mathbf{a}, \mathbf{b}) - (\mathbf{a}', \mathbf{b}')|| \cdot \sqrt{4||\Re(f)||_\infty^2 d^4 + 4||\Im(f)||_\infty^2 d^4} \leq 4\eta\sqrt{n}||f||_\infty d^2.
\end{aligned}$$

The last inequality follows from the fact that $||\Re(f)||_\infty, ||\Im(f)||_\infty \leq ||f||_\infty$. □

Theorem 6.2.10. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Let $P^{kl}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i,j\in[n]} p_{ij}x_iy_j$ where $p_{ij} = u_{ik}u_{jl} - u_{il}u_{jk}$. Let C_{CW} be the absolute constant guaranteed by Theorem 6.2.3. Then*

$$Pr_{(\boldsymbol{a},\boldsymbol{b})\in_U G_\eta}[|P^{kl}(\boldsymbol{a},\boldsymbol{b})| \geq \frac{\sqrt{2}\alpha}{3B} - 16\eta n^{\frac{3}{2}}B] \geq 1 - 4C_{CW}\alpha^{\frac{1}{2}}.$$

Proof. Using Theorem 6.2.9 for $f = P^{kl}$ where $d = 2$, we already have that $|P^{kl}(\mathbf{a},\mathbf{b}) - P^{kl}(\mathbf{a}',\mathbf{b}')| \leq 16\eta\sqrt{n}||P^{kl}||_\infty$. Since $(\mathbf{a}',\mathbf{b}')$ is selected uniformly at random from $[-1,1]^{2n}$, by Theorem 6.2.7 we have $|P^{kl}(\mathbf{a}',\mathbf{b}')| \geq \frac{\sqrt{2}\alpha}{3B}$ with probability at least $(1 - 4C_{CW}\alpha^{\frac{1}{2}})$. This gives us that

$$|P^{kl}(\mathbf{a}',\mathbf{b}')| \geq \frac{\sqrt{2}\alpha}{3B} - 16\eta\sqrt{n}||P^{kl}||_\infty \tag{6.4}$$

Now we claim that $||P^{kl}||_\infty \leq Bn$. Indeed,

$$\begin{aligned}
||P^{kl}||_\infty &= \max_{v\in[-1,1]^{2n}} |P^{kl}(v)| \\
&\leq \sum_{i,j\in[n]} |u_{ik}u_{jl} - u_{jk}u_{il}| \\
&\leq \sum_{i,j\in[n]} (|u_{ik}u_{jl}| + |u_{jk}u_{il}|) \\
&\leq \sum_{i,j\in[n]} (\frac{|u_{ik}|^2 + |u_{jl}|^2}{2} + \frac{u_{jk}^2 + u_{il}^2}{2}) \\
&\leq n||U||_F^2 \leq Bn.
\end{aligned}$$

Putting this in (6.4), we can conclude that

$$\Pr_{(\mathbf{a},\mathbf{b})\in_U G_\eta}[|P^{kl}(\mathbf{a},\mathbf{b})| \geq \frac{\sqrt{2}\alpha}{3B} - 16\eta Bn^{\frac{3}{2}}] \geq 1 - 4C_{CW}\alpha^{\frac{1}{2}}.$$

□

Corollary 6.2.10.1. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Let $P^{kl}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i,j\in[n]} p_{ij}x_iy_j$ where $p_{ij} = u_{ik}u_{jl} - u_{il}u_{jk}$. Let C_{CW} be the absolute constant guaranteed by Theorem 6.2.3. Then*

$$Pr_{(\boldsymbol{a},\boldsymbol{b})\in_U G_\eta}[|P^{kl}(\boldsymbol{a},\boldsymbol{b})| \geq k] \geq (1 - 4C_{CW}\left(\frac{3B(k + 16\eta Bn^{\frac{3}{2}})}{\sqrt{2}}\right)^{\frac{1}{2}}).$$

In the next theorem, we give a similar result for the polynomial in Theorem 6.2.5. We will require this later in Theorem 6.3.2. First we give a lower bound for the l_2 norm of the polynomial.

Theorem 6.2.11. *Let $U = (u_{ij}) \in GL_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Let $P^k(\boldsymbol{x}) = \sum_{i\in[n]} p_ix_i$ where $p_i = u_{ik}$. Let C_{CW} be the absolute constant guaranteed in Theorem 6.2.3. Then*

$$Pr_{(\boldsymbol{a},\boldsymbol{b})\in_U G_\eta}[|P^k(\boldsymbol{a})| \geq \frac{\alpha}{\sqrt{3B}} - \eta\sqrt{nB}] \geq 1 - 2C_{CW}\alpha.$$

Proof. Using the fact that P^k is a linear polynomial and using the Cauchy-Schwarz inequality, we get that

$$\begin{aligned}
&||P^k(\mathbf{a}) - P^k(\mathbf{a}')||^2 \\
&= || \sum_{i\in[n]} p_i(a_i - a_i')||^2 \\
&\leq ||\mathbf{a} - \mathbf{a}'||^2 || \sum_{i=1}^{n} |p_i|^2 || \\
&\leq ||\mathbf{a} - \mathbf{a}'||^2 ||U||_F^2 \\
&\leq ||\mathbf{a} - \mathbf{a}'||^2 \kappa_F(U) < n\eta^2 B
\end{aligned}$$

This gives us that $||P^k(\mathbf{a}) - P^k(\mathbf{a}')|| \leq \eta\sqrt{nB}$. Since $\mathbf{a}$ is selected uniformly at random from $[-1,1]^n$, using Theorem 6.2.5, we have that $|P^k(\mathbf{a})| \geq \frac{\alpha}{\sqrt{3B}}$ with probability at least $(1 - 2C_{CW}\alpha)$. This gives us that

$$|P^k(\mathbf{a}')| \geq \frac{\alpha}{\sqrt{3B}} - \eta\sqrt{nB} \tag{6.5}$$

with probability at least $1 - 2C_{CW}\alpha$. □

Finally, we give a lemma that will be needed in Section 6.3.

Lemma 6.2.12. *Let $U = (u_{ij}) \in M_n(\mathbb{C})$ be such that $\kappa_F(U) \leq B$. Then, given $\boldsymbol{a} \in [-1,1]^n$, for all $k, l \in [n]$, $|(\sum_{i\in[n]} a_i u_{ki})(\sum_{j\in[n]} a_j u_{lj})| \leq \frac{nB}{2}$.*

Proof.

$$\begin{aligned}
&|(\sum_{i\in[n]} a_i u_{ki})(\sum_{j\in[n]} a_j u_{lj})| \\
&= |(\sum_{i\in[n]} a_i u_{ki})||(\sum_{j\in[n]} a_j u_{lj})| \\
&\leq ||\mathbf{a}||^2 ||u_k|| ||u_l||
\end{aligned}$$

where u_k and u_l are the k-th and l-th rows of U respectively. Now we get that

$$||u_k|| ||u_l|| \leq \frac{||u_k||^2 + ||u_l||^2}{2} \leq \frac{(\kappa_F(U))}{2} \leq \frac{B}{2}.$$

Since $\mathbf{a} \in [-1,1]^n$, then $||\mathbf{a}|| \leq \sqrt{n}$. Combining these inequalities yields the desired result. □

6.3 Towards a proof of Theorem 4.3.4

Let $T_1, ..., T_n$ be the slices of the tensor T given as input to Algorithm 8. Let k_{gap}, k_F be the parameters as set in Algorithm 8. Let $a_1, ..., a_n, b_1, ..., b_n$ be picked uniformly and independently at random from a finite grid $G_\eta \subset [-1,1]^{2n}$ (as defined in Definition **??**). Let $T^{(a)} = \sum_{i=1}^n a_i T_i$ and $T^{(b)} = \sum_{i=1}^n b_i T_i$. Recall from (3.27) that for a matrix A, $\text{gap}(A)$ is defined as the minimum distance between its eigenvalues. In this subsection, as claimed in Section 4.3.2, we show that $T^{(a)}$ is invertible, $\text{gap}((T^{(a)})^{-1}T^{(b)}) \geq k_{\text{gap}}$ and $\kappa_F(T^{(a)}) \leq k_F$ with high probability.

Theorem 6.3.1. *Let $T \in (\mathbb{C}^n)^{\otimes 3}$ be a diagonalisable order-3 symmetric tensor such that $\kappa(T) \leq B$. We denote by $T_1, ..., T_n$ the slices of T. Let $(a_1, ..., a_n, b_1, ..., b_n) \in \mathbb{R}^{2n}$ be picked from G_η uniformly at random and set $T^{(a)} := \sum_{i=1}^n a_i T_i$, $T^{(b)} := \sum_{i=1}^n b_i T_i$. If $T^{(a)}$ is invertible, let $T^{(a)'} = (T^{(a)})^{-1}$. Then for any $k_{gap} > 0$, we have that*

$$Pr_{(\boldsymbol{a},\boldsymbol{b})\in_U G_\eta}[T^{(a)} \text{ is invertible and } gap(T^{(a)'}T^{(b)}) \geq k_{gap}] \geq 1 - \left(4n^2 C_{CW}(\frac{3B\alpha_{gap}}{\sqrt{2}})^{\frac{1}{2}} + \frac{n\eta}{2}\right)$$

where $\alpha_{gap} = \frac{nBk_{gap}}{2} + 16\eta B n^{\frac{3}{2}}$.

Proof. Let U be the matrix with rows $u_1, ..., u_n$ such that $T = \sum_{i=1}^n u_i^{\otimes 3}$ and $\kappa(T) = \kappa_F(U) \leq B$. If $a_1, ..., a_n$ are picked independently and uniformly at random from a finite set S, from (4.4), we get that $T^{(a)}$ is invertible with probability at least $1 - \frac{n}{|S|}$. We use this for $S = \{-1, -1+\eta, ..., 1-2\eta, 1-\eta\} \subset [-1, 1]$. Since $|S| = \frac{2}{\eta}$, if a is picked uniformly and independently at random from G_η, $\langle a, u_k\rangle = 0$ with probability at most $\frac{\eta}{2}$. Recall the definition of $D^{(a)}$ in Theorem 3.2.1. It follows from the union bound that $\det(D^{(a)}) \neq 0$ with probability at least $1 - \frac{n\eta}{2}$.

If $T^{(a)}$ is invertible, let $\lambda_1, ..., \lambda_n$ be the eigenvalues of $T^{(a)'}T^{(b)}$. Then by Theorem 3.2.1 (more precisely the fact that $T^{(a)} = U^T D^{(a)} U$), we get that $\lambda_k = \frac{\langle b, u_k\rangle}{\langle a, u_k\rangle}$ where u_k are the rows of U and $\langle a, u_k\rangle \neq 0$. Hence

$$\begin{aligned} gap(T^{(a)'}T^{(b)}) &= \min_{k\neq l\in[n]} \left|\frac{\langle b, u_k\rangle}{\langle a, u_k\rangle} - \frac{\langle b, u_l\rangle}{\langle a, u_l\rangle}\right| \\ &= \min_{k\neq l\in[n]} \left|\frac{\langle b, u_k\rangle\langle a, u_l\rangle - \langle b, u_l\rangle\langle a, u_k\rangle}{\langle a, u_k\rangle\langle a, u_l\rangle}\right| \end{aligned}$$

By Corollary 6.2.10.1, if $\mathbf{a}$ is picked from G_η uniformly at random, then $|\langle b, u_k\rangle\langle a, u_l\rangle - \langle b, u_l\rangle\langle a, u_k\rangle| < t$ with probability at most $4C_{CW}(\frac{3B}{\sqrt{2}}(t + 16\eta B n^{\frac{3}{2}}))^{\frac{1}{2}}$. Combining these results with the union bound, we get that

$$\begin{aligned} &\Pr_{a,b\in G_\eta}[\exists k, l \in [n] | \langle b, u_k\rangle\langle a, u_l\rangle - \langle b, u_l\rangle\langle a, u_k\rangle| < t \cup T^{(a)} \text{ is not invertible}] \\ &\leq 4n^2 C_{CW}(\frac{3B}{\sqrt{2}}(t + 16\eta B n^{\frac{3}{2}}))^{\frac{1}{2}} + \frac{n\eta}{2}. \end{aligned}$$

This gives us that

$$\begin{aligned} &\Pr_{a,b\in G_\eta}[T^{(a)} \text{ is invertible and for all } k, l \in [n] | \langle b, u_k\rangle\langle a, u_l\rangle - \langle b, u_l\rangle\langle a, u_k\rangle| > t] \\ &\geq 1 - (4n^2 C_{CW}(\frac{3B}{\sqrt{2}}(t + 16\eta B n^{\frac{3}{2}}))^{\frac{1}{2}} + \frac{n\eta}{2}). \end{aligned}$$

Now if $|\langle b, u_k\rangle\langle a, u_l\rangle - \langle b, u_l\rangle\langle a, u_k\rangle| > t$, we have that

$$\text{gap}(T^{(a)'}T^{(b)}) > t \min_{k\neq l\in[n]} \frac{1}{|\langle a, u_k\rangle\langle a, u_l\rangle|}$$

By Lemma 6.2.12, since $\kappa_F(U) \leq B$ we have $|\langle a, u_k\rangle\langle a, u_l\rangle| \leq \frac{nB}{2}$ for all $\mathbf{a} \in G_\eta \subseteq [-1, 1]^n$. This implies that $\text{gap}(T^{(a)'}T^{(b)}) > \frac{2t}{nB}$. Finally, setting $t = \frac{nBk_{\text{gap}}}{2}$, we get the desired conclusion. □

Theorem 6.3.2. *Let $T \in \mathbb{C}^{n\times n\times n}$ be a diagonalisable degree-3 symmetric tensor such that $\kappa(T) \leq B$, where $T_1, ..., T_n$ are the slices of T. Let $a \in [-1, 1]^{2n}$ be picked from G_η uniformly at random and set $T^{(a)} := \sum_{i=1}^n a_i T_i$. If $T^{(a)}$ is invertible, let*

$T^{(a)'} = (T^{(a)})^{-1}$. *Then for all* $k_F > nB^3$, *we have that*

$$Pr_{\boldsymbol{a}\in_U G_\eta}[T^{(a)} \text{ is invertible and } \kappa_F(T^{(a)}) \leq k_F] \geq 1 - (2nC_{CW}\alpha_F + \frac{n\eta}{2})$$

where $\alpha_F = \sqrt{3B}(\sqrt{\frac{nB^2}{k_F - nB^3}} + \eta\sqrt{nB})$.

Proof. Let U be the matrix with rows $u_1, ..., u_n$ such that $T = \sum_{i=1}^n u_i^{\otimes 3}$ and $\kappa(T) = \kappa_F(U) \leq B$. Since a is picked uniformly and independently from G_η, following the proof of Theorem 6.3.1, $T^{(a)}$ is invertible with probability at least $(1 - \frac{n\eta}{2})$. If $T^{(a)}$ is invertible, using Theorem 3.2.1, and more precisely the fact that $T^{(a)} = U^T D^{(a)} U$, we have:

$$\begin{aligned} ||T^{(a)'}||_F &\leq ||U^{-1}||_F^2 ||(D^{(a)})^{-1}||_F \\ &\leq \kappa_F(U)||(D^{(a)})^{-1}||_F \\ &\leq B||(D^{(a)})^{-1}||_F. \end{aligned}$$

Now, $||(D^{(a)})^{-1}||_F^2 = \sum_{i=1}^n \frac{1}{|\langle a, u_i\rangle|^2}$. By Theorem 6.2.11, if $\mathbf{a}$ is picked from G_η uniformly at random, then $|\langle a, u_i\rangle| \geq k$ with probability at least $1 - 2C_{CW}(\sqrt{3B}(k + \eta\sqrt{nB}))$. This gives us that

$$\begin{aligned} &\Pr_{a\in G_\eta}[\exists m \in [n] |\langle a, u_m\rangle| \leq k \ \cup \ T^{(a)} \text{ is not invertible}] \\ &\leq \sum_{m=1}^{n} \Pr_{a\in G_\eta}[|\langle a, u_m\rangle| \leq k] + \Pr_{a\in G_\eta}[T^{(a)} \text{ is not invertible}] \\ &\leq 2nC_{CW}(\sqrt{3B}(k + \eta\sqrt{nB})) + \frac{n\eta}{2}. \end{aligned}$$

As a result,

$$\begin{aligned} &\Pr_{a\in G_\eta}[\text{for all } m \in [n] |\langle a, u_m\rangle| \geq k \text{ and } T^{(a)} \text{ is invertible}] \\ &\geq 1 - (2nC_{CW}(\sqrt{3B}(k + \eta\sqrt{nB})) + \frac{n\eta}{2}). \end{aligned}$$

By Lemma 5.6.4, $||D^{(a)}||^2 \leq nB$. This further implies that if $|\langle a, u_m\rangle| \geq k$ for all m, then $||(D^{(a)})^{-1}||_F^2 + ||D^{(a)}||_F^2 \leq \frac{n}{k^2} + nB$, which in turn implies that $\kappa_F(T^{(a)}) = ||T^{(a)'}||_F^2 + ||T^{(a)}||_F^2 \leq \frac{nB^2}{k^2} + nB^3$. Setting $k = \sqrt{\frac{nB^2}{k_F - nB^3}}$ gives the desired conclusion. □

Theorem 6.3.3. *Let* $T \in \mathbb{C}^{n\times n\times n}$ *be a diagonalisable degree-3 symmetric tensor such that* $\kappa(T) \leq B$. *Let* $T_1, ..., T_n$ *be the slices of* T *and given* a, b *picked uniformly and independently at random from* G_η, *set* $T^{(a)} := \sum_{i=1}^n a_i T_i$ *and* $T^{(b)} := \sum_{i=1}^n b_i T_i$. *If* $T^{(a)}$ *is invertible, let* $T^{(a)'} = (T^{(a)})^{-1}$. *We assume that* $l_1, ..., l_n$ *is the output returned by Algorithm 8 on input* T, B *and an accuracy parameter* ε. *Let* k_{gap} *and* k_F *be as defined in Theorem 4.3.6. Then there exist cube roots of unity* ω_i *such that* $||\omega_i u_i - l_i|| < \varepsilon$, $T^{(a)}$ *is invertible,* $gap(T^{(a)'}T^{(b)}) \geq k_{gap}$ *and* $\kappa_F(T^{(a)}) \leq k_F$ *with probability at least*

$$\left(1 - \frac{1}{n} - \frac{12}{n^2}\right)\left(1 - \left(nC_{CW}\alpha_F + 4n^2 C_{CW}(\frac{3B\alpha_{gap}}{\sqrt{2}})^{\frac{1}{2}} + n\eta\right)\right)$$

where $\alpha_{gap} = \frac{nBk_{gap}}{2} + 16\eta B n^{\frac{3}{2}}$ *and* $\alpha_F = \sqrt{3B}(\sqrt{\frac{nB^2}{k_F - nB^3}} + \eta\sqrt{nB})$.

Proof. Let E_1 be the event that there exist cube roots of unity ω_i with $||\omega_i u_i - l_i|| < \varepsilon$. Let E_2 be the event that $\text{gap}(T^{(a)'}T^{(b)}) \geq k_{\text{gap}}$. We define E_3 to be the event that $\kappa_F(T^{(a)}) \leq k_F$ and E_4 to be the event that $T^{(a)}$ is invertible. We want to bound

$$\begin{aligned}&\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_1 \cap E_2 \cap E_3 \cap E_4] \\ &= \Pr[E_1|E_2 \cap E_3 \cap E_4]\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_2 \cap E_3 \cap E_4].\end{aligned}$$

Note here the probability in the first line and the first factor in the second line is also with respect to the internal choice of randomness in the diagonalisation algorithm (Algorithm 6). We refrain from mentioning it at every step in order to make the equations more readable.

Using Theorem 4.3.6, we get that $\Pr[E_1|E_2 \cap E_3 \cap E_4] \geq 1 - \frac{1}{n} - \frac{12}{n^2}$. Using Theorem 6.3.1, we also have $\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_2, E_4] \geq 1 - (4n^2 C_{CW}(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}} + \frac{n\eta}{2})$ where $\alpha_{\text{gap}} = \frac{nBk_{\text{gap}}}{2} + 16\eta B n^{\frac{3}{2}}$. From Theorem 6.3.2, we already know that $\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_3, E_4] \geq 1 - (n(C_{CW}(\alpha_F)) + \frac{n\eta}{2})$ where $\alpha_F = \sqrt{3B}(\sqrt{\frac{nB^2}{k_F - nB^3}} + \eta\sqrt{nB})$. Combining these using the union bound shows that

$$\begin{aligned}\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_2, E_3, E_4] &\geq 1 - (n(C_{CW}(\alpha_F)) + \frac{n\eta}{2}) + (4n^2 C_{CW}(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}} + \frac{n\eta}{2}) \\ &= 1 - (n(C_{CW}(\alpha_F)) + (4n^2 C_{CW}(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}}) + n\eta).\end{aligned}$$

Multiplying this by $1 - \frac{1}{n} - \frac{12}{n^2}$ gives the desired result. □

6.3.1 Finishing the proof of Theorem 4.3.4

Let T be the diagonalisable symmetric tensor given as input and let $U \in \text{GL}_n(\mathbb{C})$ be such that U diagonalises T. Let B be an estimate for $\kappa(T) = \kappa_F(U)$. Let a, b be picked uniformly and independently at random from G_η and define $T^{(a)} = \sum_{i=1}^n a_i T_i$, $T^{(b)} = \sum_{i=1}^n b_i T_i$ to be two linear combination of the slices $T_1, ..., T_n$ of T. Let E_1 be the event that Algorithm 8 outputs an ε-approximate solution to the tensor decomposition problem, E_2 be the event that $\text{gap}(T^{(a)'}T^{(b)}) \geq k_{\text{gap}}$, E_3 be the event that $\kappa_F(T^{(a)}) \leq k_F$ and E_4 be the event that $T^{(a)}$ is invertible. By Theorem 6.3.3,

$$\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_1 \cap E_2 \cap E_3 \cap E_4] \geq \left(1 - \frac{1}{n} - \frac{12}{n^2}\right)\left(1 - nC_{CW}\alpha_F + 4n^2 C_{CW}(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}} + n\eta\right).$$

As promised in Algorithm 8, we define at last the constants C_{gap} and c_F. Namely, we set

$$C_{\text{gap}} := \frac{1}{48\sqrt{2}C_{CW}^2} \text{ and } c_F = 96C_{CW}^2 + 1. \tag{6.6}$$

Since in Algorithm 8, we set $k_{\text{gap}} = \frac{1}{48\sqrt{2}C_{CW}^2 n^6 B^3}$ and $\eta = \frac{1}{C_\eta n^{\frac{15}{2}} B^4}$, we have for large enough n,

$$\begin{aligned}\alpha_{\text{gap}} &= \frac{nBk_{\text{gap}}}{2} + 16\eta B n^{\frac{3}{2}} \\ &= \frac{1}{96\sqrt{2}C_{CW}^2 n^5 B^2} + \frac{1}{C_\eta n^{\frac{15}{2}} B^4} \\ &\leq \frac{1}{48\sqrt{2}C_{CW}^2 n^5 B^4}.\end{aligned}$$

This gives us that $(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}} \leq \frac{1}{4C_{CW}\sqrt{2n^5B}}$, hence

$$4n^2C_{CW}(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}} \leq \frac{1}{\sqrt{2nB}} \leq \frac{1}{\sqrt{2n}}. \tag{6.7}$$

The last inequality follows from the fact that $B > 1$. We also set $k_F = (96C_{CW}^2 + 1)n^5B^3$. Since $nB^3 < n^5B^3$, we have

$$\begin{aligned}
\alpha_F &= \sqrt{3B}(\sqrt{\frac{nB^2}{k_F - nB^3}} + \eta\sqrt{nB}) \\
&= \sqrt{3B}(\sqrt{\frac{1}{96C_{CW}^2n^4B}} + \frac{1}{C_\eta n^8 B^{\frac{7}{2}}}) \\
&\leq \frac{1}{8C_{CW}n^2} + \frac{\sqrt{3}}{C_\eta n^7 B^3} \\
&\leq \frac{1}{4C_{CW}n^2}
\end{aligned}$$

This gives us that $2nC_{CW}\alpha_F \leq \frac{1}{2n}$. Also, $\eta n = \frac{1}{C_\eta n^{\frac{13}{2}} B^4} \leq \frac{1}{2n}$. Combining these with (6.7) finally shows that

$$\begin{aligned}
\Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_1] &\geq \Pr_{\mathbf{a},\mathbf{b}\in G_\eta}[E_1 \cap E_2 \cap E_3 \cap E_4] \\
&\geq \left(1 - \frac{1}{n} - \frac{12}{n^2}\right)\left(1 - 2nC_{CW}\alpha_F + 4n^2C_{CW}(\frac{3B\alpha_{\text{gap}}}{\sqrt{2}})^{\frac{1}{2}} + n\eta\right) \\
&\geq \left(1 - \frac{1}{n} - \frac{12}{n^2}\right)\left(1 - \frac{1}{\sqrt{2n}} - \frac{1}{n}\right).
\end{aligned}$$

Bibliography

[ABB+18] Diego Armentano, Carlos Beltrán, Peter Bürgisser, Felipe Cucker, and Michael Shub. A stable, polynomial-time algorithm for the eigenpair problem. *Journal of the European Mathematical Society*, 20(6):1375–1437, Apr 2018. doi:10.4171/jems/789.

[AFH+12] Anima Anandkumar, Dean P Foster, Daniel J Hsu, Sham M Kakade, and Yi-kai Liu. A Spectral Algorithm for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems*, 2012. URL: https://proceedings.neurips.cc/paper/2012/file/15d4e891d784977cacbfcbb00c48f133-Paper.pdf.

[AGH+14] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor Decompositions for Learning Latent Variable Models. *J. Mach. Learn. Res.*, 15(1), 2014. URL: https://www.jmlr.org/papers/volume15/anandkumar14b/anandkumar14b.pdf.

[AHK12] Animashree Anandkumar, Daniel J. Hsu, and Sham M. Kakade. A Method of Moments for Mixture Models and Hidden Markov Models. In *COLT 2012 - The 25th Annual Conference on Learning Theory*. JMLR.org, 2012. URL: http://proceedings.mlr.press/v23/anandkumar12/anandkumar12.pdf.

[Bar68] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving gaussian elimination. *Mathematics of Computation*, 22(103):565–578, 1968. URL: http://www.jstor.org/stable/2004533.

[BBV19] 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), 2019. doi:10.1137/18M1200531.

[BC13] P. Bürgisser and F. Cucker. *Condition: The Geometry of Numerical Algorithms*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2013. URL: https://books.google.fr/books?id=d_SSnAEACAAJ.

[BCMT09] Jérôme Brachat, Pierre Comon, Bernard Mourrain, and Elias P. Tsigaridas. Symmetric tensor decomposition. In *17th European Signal Processing Conference, EUSIPCO*. IEEE, 2009. URL: https://ieeexplore.ieee.org/document/7077748/.

[BCMV14] Aditya Bhaskara, Moses Charikar, Ankur Moitra, and Aravindan Vijayaraghavan. Smoothed Analysis of Tensor Decompositions. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC, 2014. doi:10.1145/2591796.2591881.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[BGI11] Alessandra Bernardi, Alessandro Gimigliano, and Monica Idà. Computing symmetric rank for symmetric tensors. *Journal of Symbolic Computation*, 46(1), 2011. doi:10.1016/j.jsc.2010.08.001.

[BGVKS22] Jess Banks, Jorge Garza-Vargas, Archit Kulkarni, and Nikhil Srivastava. Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time. *Foundations of Computational Mathematics*, Aug 2022. doi:10.1007/s10208-022-09577-5.

[BHKX22] Mitali Bafna, Jun-Ting Hsieh, Pravesh K. Kothari, and Jeff Xu. Polynomial-Time Power-Sum Decomposition of Polynomials. *CoRR*, abs/2208.00122, 2022. arXiv:2208.00122.

[Bjö14] Å. Björck. *Numerical Methods in Matrix Computations*. Texts in Applied Mathematics. Springer International Publishing, 2014. URL: https://books.google.fr/books?id=joO8BAAAQBAJ.

[BNV23] Carlos Beltrán, Vanni Noferini, and Nick Vannieuwenhoven. When can forward stable algorithms be composed stably? *IMA Journal of Numerical Analysis*, page drad026, 05 2023. arXiv:https://academic.oup.com/imajna/advance-article-pdf/doi/10.1093/imanum/drad026/50479917/drad026.pdf, doi:10.1093/imanum/drad026.

[Bor07] Folkmar Bornemann. A model for understanding numerical stability. *IMA Journal of Numerical Analysis*, 27(2):219–231, apr 2007. URL: https://doi.org/10.1093%2Fimanum%2Fdrl037, doi:10.1093/imanum/drl037.

[BSS89] 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.

[BSV21a] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Reconstruction Algorithms for Low-Rank Tensors and Depth-3 Multilinear Circuits. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021. doi:10.1145/3406325.3451096.

[BSV21b] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Reconstruction algorithms for low-rank tensors and depth-3 multilinear circuits. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2021. URL: https://doi.org/10.1145/3406325.3451096.

[Cai94] Jin-Yi Cai. Computing jordan normal forms exactly for commuting matrices in polynomial time. *International Journal of Foundations of Computer Science*, 05(03n04):293–302, 1994. arXiv:https://doi.org/10.1142/S0129054194000165, doi:10.1142/S0129054194000165.

[Car06] Enrico Carlini. Reducing the number of variables of a polynomial. In *Algebraic geometry and geometric modeling*, Math. Vis., pages

237–247. Springer, Berlin, 2006. URL: https://doi.org/10.1007/978-3-540-33275-6_15.

[CC70] J. Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, Sep 1970. doi:10.1007/BF02310791.

[CG05] Guillaume Cheze and André Galligo. Four lectures on polynomial absolute factorization. In *Solving polynomial equations*, pages 339–392. Springer, 2005.

[CL07] Guillaume Chèze and Grégoire Lecerf. Lifting and recombination techniques for absolute factorization. *Journal of Complexity*, 23(3):380–420, 2007.

[CW01] Anthony Carbery and James Wright. Distributional and L-q norm inequalities for polynomials over convex bodies in R-n. *Mathematical Research Letters*, 8:233–248, 2001. URL: https://api.semanticscholar.org/CorpusID:59405379.

[DDH07] James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1), 2007. doi:10.1007/s00211-007-0114-x.

[DDHK07] James Demmel, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg. Fast matrix multiplication is stable. *Numerische Mathematik*, 106(2), 2007. doi:10.1007/s00211-007-0061-6.

[DdL+22] Jingqiu Ding, Tommaso d'Orsi, Chih-Hung Liu, David Steurer, and Stefan Tiegel. Fast algorithm for overcomplete order-3 tensor decomposition. In *Proceedings of Thirty Fifth Conference on Learning Theory*, 2022. URL: https://proceedings.mlr.press/v178/ding22a.html.

[DL78] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. URL: https://www.sciencedirect.com/science/article/pii/0020019078900674, doi:https://doi.org/10.1016/0020-0190(78)90067-4.

[DLCC07] Lieven De Lathauwer, Josphine Castaing, and Jean-Franois Cardoso. Fourth-order cumulant-based blind identification of underdetermined mixtures. *IEEE Transactions on Signal Processing*, 55(6):2965–2973, 2007. doi:10.1109/TSP.2007.893943.

[FGS18] Michael A. Forbes, Sumanta Ghosh, and Nitin Saxena. Towards Black-box Identity Testing of Log-Variate Circuits. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, 2018. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9058, doi:10.4230/LIPIcs.ICALP.2018.54.

[Fis94] Ismor Fischer. Sums of like powers of multivariate linear forms. *Mathematics Magazine*, 67(1):59–61, 1994. URL: http://www.jstor.org/stable/2690560.

[Fre79] Rūsiņš Freivalds. Fast probabilistic algorithms. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 57–69. Springer, 1979.

[FS18] Michael A. Forbes and Amir Shpilka. A PSPACE Construction of a Hitting Set for the Closure of Small Algebraic Circuits. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC, 2018. doi:10.1145/3188745.3188792.

[Gao03] Shuhong Gao. Factoring multivariate polynomials via partial differential equations. *Mathematics of computation*, 72(242):801–822, 2003.

[GG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, USA, 3rd edition, 2013.

[GGKS19] Ankit Garg, Nikhil Gupta, Neeraj Kayal, and Chandan Saha. Determinant Equivalence Test over Finite Fields and over Q. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, 2019. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10638.

[GHK15] Rong Ge, Qingqing Huang, and Sham M. Kakade. Learning Mixtures of Gaussians in High Dimensions. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC, 2015. doi:10.1145/2746539.2746616.

[Gil20] Nicolas Gillis. *Nonnegative Matrix Factorization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2020. doi:10.1137/1.9781611976410.

[GM15] Rong Ge and Tengyu Ma. Decomposing Overcomplete 3rd Order Tensors using Sum-of-Squares Algorithms. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.829.

[GMKP17] 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. URL: http://doi.acm.org/10.1145/3087604.3087605, doi:10.1145/3087604.3087605.

[GMKP18] 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.

[Gup22] Nikhil Gupta. *On symmetries of and equivalence tests for two polynomial families and a circuit class*. PhD thesis, CSA, IISc Bangalore, 2022.

[GVX14] Navin Goyal, Santosh Vempala, and Ying Xiao. Fourier PCA and Robust Tensor Decomposition. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC, 2014. doi:10.1145/2591796.2591875.

[Har70] RA Harshman. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-mode factor analysis. *UCLA Working Papers in Phonetics*, 1970. URL: https://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf.

[Hås89] Johan Håstad. Tensor rank is NP-complete. In *Automata, Languages and Programming*. Springer Berlin Heidelberg, 1989. URL: https://link.springer.com/chapter/10.1007/BFb0035776.

[Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, second edition, 2002. doi:10.1137/1.9780898718027.

[HJ13] Roger Horn and Charles Johnson. *Matrix Analysis*. Cambridge University Press (second edition), 2013.

[HK13] Daniel Hsu and Sham M. Kakade. Learning Mixtures of Spherical Gaussians: Moment Methods and Spectral Decompositions. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS, 2013. doi:10.1145/2422436.2422439.

[HKZ09] Daniel J. Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. In *COLT - The 22nd Conference on Learning Theory*, 2009. URL: http://www.cs.mcgill.ca/%7Ecolt2009/papers/011.pdf#page=1.

[HSS15] Samuel B. Hopkins, Jonathan Shi, and David Steurer. Tensor principal component analysis via sum-of-square proofs. In *Proceedings of The 28th Conference on Learning Theory*, 2015. URL: https://proceedings.mlr.press/v40/Hopkins15.html.

[Kay11] 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.

[KB09] Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3), 2009. doi:10.1137/07070111X.

[KG85] Walter Keller-Gehrig. Fast algorithms for the characteristics polynomial. *Theoretical Computer Science*, 36:309 – 317, 1985. URL: http://www.sciencedirect.com/science/article/pii/0304397585900490, doi:https://doi.org/10.1016/0304-3975(85)90049-0.

[KNST18] 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.

[Koi95] P. Koiran. Approximating the volume of definable sets. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995. doi:10.1109/SFCS.1995.492470.

[Koi00] Pascal Koiran. Circuits versus trees in algebraic complexity. In Horst Reichel and Sophie Tison, editors, *STACS 2000*, pages 35–52, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[KP20] Bohdan Kivva and Aaron Potechin. Exact nuclear norm, completion and decomposition for random overcomplete tensors via degree-4 SOS. *CoRR*, 2020. arXiv:2011.09416.

[Kru77] Joseph B. Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications*, 18(2), 1977. doi:https://doi.org/10.1016/0024-3795(77)90069-6.

[KS09] 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.

[KS19] 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.

[KS21] Pascal Koiran and Mateusz Skomra. Derandomization and Absolute Reconstruction for Sums of Powers of Linear Forms. *Theor. Comput. Sci.*, 887, 2021. doi:10.1016/j.tcs.2021.07.005.

[KS22a] Pascal Koiran and Subhayan Saha. Black Box Absolute Reconstruction for Sums of Powers of Linear Forms. In *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*, 2022. URL: https://drops.dagstuhl.de/opus/volltexte/2022/17416, doi:10.4230/LIPIcs.FSTTCS.2022.24.

[KS22b] Pascal Koiran and Subhayan Saha. Complete decomposition of symmetric tensors in linear time and polylogarithmic precision, 2022. arXiv:2211.07407.

[KS23] Pascal Koiran and Subhayan Saha. Absolute reconstruction for sums of powers of linear forms: degree 3 and beyond. *computational complexity*, 32(2), August 2023. doi:10.1007/s00037-023-00239-8.

[LO38] J. E. Littlewood and A. C. Offord. On the number of real roots of a random algebraic equation. *Journal of the London Mathematical Society*, s1-13(4):288–295, 1938. doi:https://doi.org/10.1112/jlms/s1-13.4.288.

[LRA93] S. E. Leurgans, R. T. Ross, and R. B. Abel. A Decomposition for Three-Way Arrays. *SIAM Journal on Matrix Analysis and Applications*, 14(4), 1993. doi:10.1137/0614071.

[MN07] Frédéric Magniez and Ashwin Nayak. Quantum complexity of testing group commutativity. *Algorithmica*, 48(3):221–232, 2007.

[Moi18] A. Moitra. *Algorithmic Aspects of Machine Learning*. Cambridge University Press, 2018. URL: https://books.google.fr/books?id=ruVqDwAAQBAJ.

[MR05] Elchanan Mossel and Sébastien Roch. Learning Nonsingular Phylogenies and Hidden Markov Models. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC, 2005. doi:10.1145/1060590.1060645.

[MSS16] Tengyu Ma, Jonathan Shi, and David Steurer. Polynomial-Time Tensor Decompositions with Sum-of-Squares. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, 2016. doi:10.1109/FOCS.2016.54.

[NP21] Vincent Neiger and Clément Pernet. Deterministic computation of the characteristic polynomial in the time of matrix multiplication. *Journal of Complexity*, 67:101572, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0885064X21000273, doi:https://doi.org/10.1016/j.jco.2021.101572.

[Pak12] Igor Pak. Testing commutativity of a group and the power of randomization. *LMS Journal of Computation and Mathematics*, 15:38–43, 2012.

[Poi95] B. Poizat. *Les petits cailloux: une approche modèle-théorique de l'algorithmie*. Nūr al-manṭiq wa-al-ma'rifah. Aléas, 1995. URL: https://books.google.fr/books?id=YeHuAAAAMAAJ.

[PS07] Clément Pernet and Arne Storjohann. Faster algorithms for the characteristic polynomial. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, page 307–314, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1277548.1277590.

[PSV22] Shir Peleg, Amir Shpilka, and Ben Lee Volk. Tensor reconstruction beyond constant rank, 2022. doi:10.48550/ARXIV.2209.04177.

[RS00] Sridhar Rajagopalan and Leonard J. Schulman. Verification of identities. *SIAM Journal on Computing*, 29(4):1155–1163, 2000. arXiv:https://doi.org/10.1137/S0097539797325387, doi:10.1137/S0097539797325387.

[Sap15] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. https://github.com/dasarpmar/lowerbounds-survey, 2015.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. doi:10.1145/322217.322225.

[Sha09] Hani Shaker. Topology and factorization of polynomials. *Mathematica Scandinavica*, pages 51–59, 2009.

[Shi16] Yaroslav Shitov. How hard is the tensor rank?, 2016. doi:10.48550/ARXIV.1611.01559.

[Shp09] Amir Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. *SIAM Journal on Computing*, 38(6):2130–2161, 2009.

[Sip13] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3):281–292, Sep 1971. doi:10.1007/BF02242355.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3–4):207–388, mar 2010. doi:10.1561/0400000039.

[tB91] Jos M F ten Berge. Kruskal's polynomial for 2×2×2 arrays and a generalization to 2×n×n arrays. *Psychometrika*, 56(4):631–636, December 1991.

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[Val79] L. G. Valiant. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, page 249–261, New York, NY, USA, 1979. Association for Computing Machinery. doi:10.1145/800135.804419.

[Vu17] Van Vu. *Anti-concentration Inequalities for Polynomials*, pages 801–810. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-44479-6_32.

[Wil68] J.H. Wilkinson. Global convergene of tridiagonal QR algorithm with origin shifts. *Linear Algebra and its Applications*, 1(3):409–420, 1968. doi:https://doi.org/10.1016/0024-3795(68)90017-7.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

Appendix A

Appendix to Chapter 2

A.1 Computing the complexity of the randomized algorithm in [KS21] and comparing it with our algorithm

Recall the [KS21] equivalence from Section 2.1.1. The algorithm proceeds as follows:

1. Pick a random matrix $R \in M_n(\mathbb{K})$ and set $h(x) = f(Rx)$.
2. Let $T_1, ..., T_n$ 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 diagonalisable over $\mathbb{K}$, accept. Otherwise, reject.

The following are the different stages of computation required in this algorithm:

1. If the polynomial is input in dense representation, here we have to compute all the slices $T_1, ..., T_n$ and following the proof of Theorem 2.6.3, this takes $O(n^4)$ many arithmetic operations
   If the polynomial is given as a blackbox, we compute $x' = Rx$. And we call the blackbox on this input.
2. **Compute** $T_1, T_2, ..., T_n$ We know
   $$(T_k)_{ij} = \frac{1}{3!}\partial_{x_i x_j x_k}(h).$$
   So we can extract each entry of T_k using constant many calls to the blackbox and constantly many arithemtic operations using Lemma 2.6.2. There are in total n^3 such entries that we need to compute. So the total number of calls to the blackbox is $O(n^3)$ and the number of arithmetic operations is $O(n^3)$
3. **Check if** T_1 **is invertible. If invertible, compute** $T_1' = T_1^{-1}$.
   This can be done in time at most $O(n^3)$.
4. **Checking pairwise commutativity of** $\{T_1'T_j\}_{j\in[n]}$
   Here we compute the product $T_1'T_iT_1'T_j$ and $T_1'T_jT_1'T_i$ and check if their difference is 0. For each pair, this can be done in time $O(n^\omega)$. Since there are $\binom{n}{2}$ many pairs, this can be done in time $O(n^{\omega+2})$.
5. **Checking the diagonalisability of** $T_1'T_j$ **for all** $j \in [n]$:
   As we showed that the diagonalisability of each $T_1'T_j$ can be checked in time $O(n^{\omega+1})$. So the total time taken for checking diagonalisability of n such matrices is $O(n^{\omega+2})$.

So we conclude that if the polynomial is given as an input in the dense representation model, then the algorithm runs in time $O(n^{\omega+2})$.
If the polynomial is given as a blackbox, then the algorithm makes $O(n^3)$ many calls to the blackbox and requires $O(n^{\omega+2})$ many arithmetic operations.

So we manage to shave a factor of n from [KS21] in both cases: when the polynomial is given in dense representation as well as when it is input as an arithmetic circuit.

A.2 Complexity analysis for variable minimization

In this section, we provide a detailed complexity analysis of Algorithm 3. We show that if a degree d polynomial in n variables over $\mathbb{C}$ is given as a blackbox, the algorithm performs $\text{poly}(n,d)$ many calls to the blackbox and performs $\text{poly}(n,d)$ many arithmetic operations to check if the polynomial can be written as a linear combination of t many linearly independent linear forms for some $t \leq n$ (see Theorem A.2.2).

In the next lemma, we show that given a blackbox computing the polynomial, we can compute the partial derivative with respect to a single variable at a given point in $O(d)$ many oracle calls and $\text{poly}(d)$ arithmetic operations.

Lemma A.2.1. *Given a blackbox computing $P \in \mathbb{K}[x_1,...,x_n]$ of degree at most d, and given points $\alpha_1,...,\alpha_n \in \mathbb{K}^n$, we can compute the matrix M such that $M_{ij} = \partial_{x_i}(P)(\alpha_j)$ using $O(n^2 d)$ many oracle calls to the blackbox and $O(n^2 M(d)\log d)$ many arithmetic operations.*

Proof. We assume $i = 1$. Given points $a_1,...,a_n \in \mathbb{K}$, $\partial_{x_1}(P)(a_1,...,a_n)$ can be computed using the following steps of computation:

- Compute the polynomial $P(x,a_2,...,a_n)$ explicitly using polynomial interpolation. This can be done using $O(M(d)\log d)$ many arithmetic operations and $d+1$ many calls to the blackbox. [GG13] (Section 10.2)
- Compute $\partial_x(P(x,a_2,...,a_n))$ using $O(d)$ many arithmetic operations.
- evaluate it at $x = a_1$, which requires $O(d)$ many arithmetic operations.

We do this for every $(a_1,...,a_n) = (\alpha_j^{(1)},...,\alpha_j^{(n)})$ for all $j \in [n]$. Thus each row of the matrix M can be computed using $O(nM(d)\log d)$ many operations. For each x_i such that $i \in [n]$, a similar algorithm works and the matrix M can be computed using $O(n^2 M(d)\log d)$ arithmetic operations and $O(n^2 d)$ many calls to the blackbox. □

Now we are in a position to analyse the complexity of Algorithm 3

Theorem A.2.2. *Given a blackbox computing a polynomial $P \in \mathbb{K}[x_1,...,x_n]_d$ with t essential variables, the algorithm makes $O(n^2 d)$ many calls to the blackbox and requires $O(n^\omega + n^2 M(d)\log d + t^{\omega+1})$ many arithmetic operations.*

Proof. The following are the steps of the algorithm:

- Using Lemma A.2.1, we know that the matrix M can be computed in time $O(n^2 M(d)\log d)$ many arithmetic operations and $O(n^2 d)$ many calls to the blackbox.
- The basis of $\ker(M)$ can be computed and the basis can be completed using $O(n^\omega)$ many arithmetic operations.

- For $n = t$, using Theorem 2.7.1 the algorithm checks equivalence using $O(t^2 d)$ many calls to the blackbox and $O(t^2 M(d) \log d + t^{\omega+1})$ many arithmetic operations.

So we conclude that if the polynomial is given as a blackbox, then the algorithm makes $O(n^2 d)$ many calls to the blackbox and the number of arithmetic operations required is $O(n^\omega + n^2 M(d) \log d + t^{\omega+1})$.

□

Using Section 2.7.2 and the fact that we just need to do polynomial interpolation and completion of the basis, we conclude that Algorithm 3 takes time $\text{poly}(n, d, \log(|S|), b_f)$ in the bit model of computation. Here b_f is the bit size of the maximum coefficient in f and S is the set from which the entries of R and $\alpha_j^{(i)}$s are picked uniformly and independently at random.

Appendix B

Appendix to Chapter 3

B.1 Proof of Theorem 3.3.12

Lemma B.1.1. *Let $A, A' \in M_n(\mathbb{C})$ be such that A has n distinct eigenvalues and $||A - A'|| \leq \delta$ where $\delta < \frac{1}{8\kappa_{eig}(A)}$. Then*

$$gap(A') \geq \frac{3gap(A)}{4}.$$

Proof. Refer to the proof of Proposition 1.1 in [BGVKS22]. □

Lemma B.1.2. *Let $\boldsymbol{u}$ be such that $\log(\frac{1}{u}) > \log^4(\frac{n}{\delta})\log n$ where $\delta < \frac{1}{2}$. Then for $n \geq 4$, $n^2\boldsymbol{u} < \delta$.*

Proof. From the hypothesis $\delta < \frac{1}{2}$ it follows that $\log(\frac{1}{\delta}) > 1$ and $\log(\frac{1}{\delta}) \leq \log^4(\frac{1}{\delta})$. Also, from $n \geq 4$ it follows that $\log^4(n) - 2 \geq 0$ and $\log(\frac{1}{\delta}) \leq \log^4(\frac{1}{\delta}) + \log^4(n) - 2$. Now we use the fact that $\log^4(\frac{1}{\delta}) + \log^4(n) \leq \log^4(\frac{n}{\delta})$. This implies that

$$\log(\frac{1}{\delta}) \leq (\log^4(\frac{n}{\delta}) - 2)\log n.$$

Thereefore $\log(\frac{n^2}{\delta}) < \log^4(\frac{n}{\delta})\log n \leq \log(\frac{1}{\mathbf{u}})$, and $n^2\mathbf{u} \leq \delta$. □

Proof of Theorem 3.3.12. Let $v_1^{(0)}, ..., v_n^{(0)}$ be the true normalized eigenvectors of A. By Theorem 3.3.4, we need $O(\log^4(\frac{n}{\delta'})\log n)$ bits of precision to run EIG in step 3. So, we assume that the number of bits of precision available for this algorithm is $\log(\frac{1}{\mathbf{u}}) := c\log^4(\frac{n}{\delta'})\log n$ for some constant $c > 1$. We define $B = \frac{A}{2K_{\text{norm}}}$. Then $||B|| \leq \frac{1}{2} < 1$. Following the notation of Section 1.2.2, let $B' = \text{fl}(\frac{A}{2K_{\text{norm}}})$. Using (3.3), we know that

$$||B - B'|| \leq \frac{\mathbf{u} \cdot \sqrt{n}}{2}. \tag{B.1}$$

Since $\mathbf{u} \cdot \sqrt{n} \leq 1$, then $||B'|| \leq \frac{1}{2} + \frac{\mathbf{u}\cdot\sqrt{n}}{2} < 1$. We first show that the conditions of Theorem 3.3.4 are satisfied when we run EIG on (B', δ'). For this, we have to show that $\delta' \leq \frac{1}{8\kappa_{\text{eig}}(B')}$.

Applying Lemma B.1.2 where $\log(\frac{1}{\mathbf{u}}) = c\log^4(\frac{n}{\delta'})\log n$, we get that for large enough n, $\mathbf{u}\sqrt{n} < n^2\mathbf{u} < \delta' < \frac{\delta}{4K_{\text{eig}}K_{\text{norm}}}$. This gives us:

$$\frac{\mathbf{u} \cdot \sqrt{n}}{2} \leq \frac{1}{8K_{\text{eig}}K_{\text{norm}}} < \frac{1}{8\kappa_{\text{eig}}(B)}.$$

Putting it back in (B.1), we also have that $||B - B'|| < \frac{1}{8\kappa_{\text{eig}}(B)}$. Now we can apply Lemma 3.3.9 to B, B' and we obtain the inequality

$$\kappa_V(B') \leq \frac{\kappa_V^F(B')}{2} \leq 3n\kappa_V(B) \leq \frac{3n\kappa_V^F(B)}{2}. \tag{B.2}$$

It from Lemma B.1.1 that $\text{gap}(B') \geq \frac{3\text{gap}(B)}{4}$. This gives us that

$$\frac{1}{K_{\text{eig}}} < \frac{1}{\kappa_{\text{eig}}(A)} = \frac{\text{gap}(A)}{\kappa_V(A)} = \frac{2K_{\text{norm}}\text{gap}(B)}{\kappa_V(B)} \leq \frac{8nK_{\text{norm}}\text{gap}(B')}{\kappa_V(B')}.$$

Hence,

$$\delta' = \frac{\delta}{64nK_{\text{eig}}K_{\text{norm}}} < \frac{\delta}{8\kappa_{\text{eig}}(B')} < \frac{1}{8\kappa_{\text{eig}}(B')},$$

and we can now run EIG on (B', δ'). By Theorem 3.3.4, the algorithm therefore outputs (W, D_0) such that

$$||B' - WD_0W^{-1}|| \leq \delta' \tag{B.3}$$

with probability at least $1 - \frac{1}{n} - \frac{12}{n^2}$. Applying the triangle inequality to (B.1) and (B.3) shows that $||B - WD_0W^{-1}|| \leq \frac{\mathbf{u}\cdot\sqrt{n}}{2} + \frac{\delta}{64nK_{\text{eig}}K_{\text{norm}}}$. Since $\mathbf{u}\sqrt{n} \leq n^2\mathbf{u}$, by Lemma B.1.2 and for large enough n we have $\frac{\mathbf{u}\cdot\sqrt{n}}{2} \leq \frac{\delta}{64nK_{\text{eig}}K_{\text{norm}}}$. This gives us that $||B - WD_0W^{-1}|| \leq \frac{\delta}{32nK_{\text{eig}}K_{\text{norm}}}$. Multiplying both sides by $2K_{\text{norm}}$, we obtain

$$||A - W(2K_{\text{norm}}D_0)W^{-1}|| \leq \frac{\delta}{16nK_{\text{eig}}}.$$

Let $A' = W(2K_{\text{norm}}D_0)W^{-1}$. We can now use Corollary 3.3.11.1 for A and A' since $\frac{\delta}{16nK_{\text{eig}}} < \frac{1}{8nK_{\text{eig}}} < \frac{1}{8\kappa_{\text{eig}}(A)}$. Let $v_1, ..., v_n$ be the eigenvectors of A'. Then there exists a phase ρ_i such that

$$||v_i^{(0)} - \rho_i v_i|| \leq 6n\kappa_{\text{eig}}(A)\frac{\delta}{16nK_{\text{eig}}} < \frac{\delta}{2}. \tag{B.4}$$

By Theorem 3.3.4, $||w_i|| = 1 \pm n\mathbf{u}$ since $w_1, ..., w_n$ are the columns of W. Note that the w_i's are the eigenvectors of A' as well. Since A' has distinct eigenvalues, the eigenvectors are unique up to scaling by complex numbers. This along with the fact that $||v_i|| = 1$ gives us that there exists phase ρ_i' such that

$$||v_i - (\rho_i')^{-1}w_i|| = |n\mathbf{u}|||v_i|| < \frac{\delta}{2}.$$

The final inequality comes from the fact that for $n > 2$, $n\mathbf{u} < \frac{n^2\mathbf{u}}{2}$ and we can therefore use Lemma B.1.2. Now, multiplying by $|\rho_i|$ on both sides, we have

$$||\rho_i v_i - \rho_i(\rho_i')^{-1}w_i|| < \frac{\delta}{2}. \tag{B.5}$$

Now, using the triangle inequality on (B.4) and (B.5),

$$||v_i^{(0)} - \rho_i(\rho_i')^{-1}w_i|| < \delta. \tag{B.6}$$

From Theorem 3.3.4, we get that $\kappa(W) \leq \frac{\kappa_F(W)}{2} \leq \frac{1}{2}(\frac{9n}{4} + 9n^2(\kappa_V^F(B'))^2)$. By (B.2) and the fact that $\kappa_V^F(A) = \kappa_V^F(B)$, we have

$$\frac{1}{2}(\frac{9n}{4} + 9n^2(\kappa_V^F(B'))^2) \leq \frac{1}{2}(\frac{9n}{4} + 81n^4(\kappa_V^F(B))^2) = \frac{1}{2}(\frac{9n}{4} + 81n^4(\kappa_V^F(A))^2).$$

□

Appendix C

Some omitted technical details from Chapter 5

C.1 Some technical lemmas

Lemma C.1.1. *Let* $\boldsymbol{u} = \frac{1}{n^{c\log^k(nB)}}$ *for some integer* $k \geq 2$*, for some constant* $c > 0$ *and* $n, B \geq 1$*. Then for any constant* $c' > 0$ *and large enough* n*,* $\boldsymbol{u}(nB)^{c'\log(n)} \leq \frac{\sqrt{\boldsymbol{u}}}{2}$.

Proof. Since, we have that $\log(\mathbf{u}) = -c\log^k(nB)\log(n)$, this gives us that

$$\log(\mathbf{u}) + c'\log(n)\log(nB) \leq (c'\log(nB) - c\log^k(nB))\log(n) \leq \log(2) - \frac{c}{2}\log^k(nB)\log(n).$$

Using this, we can conclude that $\mathbf{u}(nB)^{c'\log(n)} \leq \frac{1}{2n^{\frac{c}{2}\log^k(nB)}} = \frac{\sqrt{\mathbf{u}}}{2}$ □

Lemma C.1.2. *For constants* $c_{n_1}, c_{n_2}, c_{B_1}, c_{B_2}, c_\varepsilon, C > 0$*, if*

$$\log(\frac{1}{\boldsymbol{u}}) > c\log(\frac{nB}{\varepsilon})\log n,$$

where $c = 2\max\{c_{n_1} + c_{n_2}, c_{B_1} + c_{B_2}, c_\varepsilon, \log C\}$ *then*

$$C\frac{n^{c_{n_1}\log n + c_{n_2}} B^{c_{B_1}\log n + c_{B_2}}}{\varepsilon^{c_\varepsilon}} \leq \frac{1}{\boldsymbol{u}}.$$

Proof. Let $c = 2\max\{c_{n_1} + c_{n_2}, c_{B_1} + c_{B_2}, c_\varepsilon, \log C\}$. Then

$$\begin{aligned}
\log(\frac{1}{u}) &= c\log(\frac{nB}{\varepsilon})\log n \\
&= c\log(nB)\log n + \log(\frac{1}{\varepsilon})\log n \\
&\geq (c_{n_1}\log n + c_{n_2})\log(n) + (c_{B_1}\log n + c_{B_2})\log(B) + c_\varepsilon\log(\frac{1}{\varepsilon}) + \log(C) \\
&\geq \log\left(C\frac{n^{c_{n_1}\log n + c_{n_2}} B^{c_{B_1}\log n + c_{B_2}}}{\varepsilon^{c_\varepsilon}}\right).
\end{aligned}$$

□

C.2 Remaining proofs from Section 5.3

C.2.1 Step 4:

In this section, we also define functions f_4 corresponding to the matrix diagonalisation required in Step 4 of Algorithm 8 and then using results from Section 3.3 show that

there exist some algorithm which output some solution close to the actual solution of the diagonalisation function. **Defining f_4 and $\tilde{f}_4$:** Let X_4 be the set of all diagonalisable matrices in $M_n(\mathbb{C})$ with distinct eigenvalues. We define $f_4 : X_4 \to \mathcal{P}(\mathrm{GL}_n(\mathbb{C}))$ to be the diagonalisation map. More formally, for some $A \in X_4$, we define $f_4(A)$ to be the set of matrices

$$\{V \in \mathrm{GL}_n(\mathbb{C}) | \exists \text{ diagonal } D \text{ where } A = VDV^{-1}, \text{ columns of } V \text{ have norm } 1\}.$$

For any $x \in X_4$, we define $||x|| = ||A||$.

The forward-diagonalisation algorithm (EIG-FWD) defined in Algorithm 6 does the following: For any $A \in X_4$, the algorithm takes as some input $\tilde{A}$ close to A (along with parameters $K_{\text{eig}} \geq \kappa_{\text{eig}}(A)$, $K_{\text{norm}} \geq ||A||_F$ and some accuracy parameter ε) and outputs some matrix $\tilde{V}$ such that there exists $V \in f_4(A)$ for which $||\tilde{V} - V|| \leq \varepsilon$ with high probability on some internal choice of randomness of the algorithm. More formally, we denote by $\mathcal{P} = (\Omega, \mathcal{F}, \mathbb{P})$ to be the probability space of the internal choices of randomness of the algorithm EIG-FWD.

For all parameters $p = (\varepsilon, K_{\text{eig}}, K_{\text{norm}})$, define function

$$\begin{aligned} \tilde{f}_{4,p,\omega} : X_4 &\to \mathrm{GL}_n(\mathbb{C}) \\ x &\mapsto \text{EIG-FWD}(x, p, \omega) \end{aligned}$$

where EIG-FWD(x, p, ω) is the output of the algorithm EIG-FWD run with parameters $p = (\varepsilon, K_{\text{eig}}, K_{\text{norm}})$ and $\omega \in \Omega$ is the corresponding value for the internal random choices of the algorithm on input $x \in X_4$ on a floating point machine with machine precision $\mathbf{u}$ where

$$\log(\frac{1}{\mathbf{u}}) = C_4 \log^4(\frac{nK_{\text{eig}}K_{\text{norm}}}{\varepsilon}) \tag{C.1}$$

for some constant $C_4 > 0$.

Definition C.2.1. *We say that some $x \in X_4$ satisfies parameters $p = (\varepsilon, K_{eig}, K_{norm})$ if $K_{eig} \geq \kappa_{eig}(A)$ and $K_{norm} \geq max\{||A||_F, 1\}$.*

Note that if $\delta > 0$, EIG-FWD will always output a matrix and in fact, it will approximate f_4 when the parameters are satisfied. Rewriting Theorem 3.3.12 in this language, we get the following result. Define

$$\tilde{f}_{4,p} := \{\tilde{f}_{4,p,\omega} | \omega \in \Omega\}. \tag{C.2}$$

Lemma C.2.2. *Let $p = (\varepsilon_4, K_{eig}, K_{norm})$ be some parameters where $\varepsilon_4 \in (0, \frac{1}{2})$. Define $X_{4,p} := \{x \in X_4 | x \text{ satisfies parameter } p\}$ and*

$$\boldsymbol{u}_p = \frac{1}{n^{C_4 \log^4(\frac{nK_{eig}K_{norm}}{\varepsilon_4})}} \text{ for some constant } C_4 > 0.$$

Then $\tilde{f}_{4,p}$ is a $(1 - \frac{1}{n} - \frac{12}{n^2}, \boldsymbol{u}_p, \varepsilon_4)$-algorithm for computing f_4 on subdomain $X_{4,p}$ when run on a finite precision machine with machine precision $\boldsymbol{u}_p$.

Proof. Since, $\tilde{x} \in B(x, \mathbf{u})$ where $\mathbf{u} = \frac{1}{n^{C_4 \log^4(\frac{nK_{\text{eig}}K_{\text{norm}}}{\varepsilon_4})}}$ for some constant $C_4 > 0$, using the fact that $K_{\text{norm}} > 1$ and $\varepsilon < 1$, we get that for large enough n

$$||\tilde{x} - x|| \leq \mathbf{u} < \frac{1}{n^{c_4 \log(K_{\text{eig}})}} = \frac{1}{(K_{\text{eig}})^{C_4 \log(n)}} \leq \frac{1}{8K_{\text{eig}}} \leq \frac{1}{8\kappa_{\text{eig}}(x)}. \tag{C.3}$$

Using Lemma 3.3.8, this gives us that $\tilde{x}$ is diagonalisable and has distinct eigenvalues. Hence, $\tilde{x} \in X_4$.

Let us assume that Algorithm 6 is run with some choice of parameters p. Now, instead of some $x \in X_{4,p}$, it is given as input $\tilde{x}$ such that for all $i, j \in [n]$, $||\tilde{x} - x|| \leq ||\tilde{x}_{ij} - x_{ij}|| \leq \mathbf{u}_p$ where $\mathbf{u}_p$ is also the required machine precision of the algorithm. We follow the definition of the model of finite arithmetic from Section 1.2.2 and the explanation regarding the same at the end of Section 1.1.2 of [BGVKS22] "...since it is not even assumed that the input is stored exactly." In that case, we show that EIG-FWD is *robust*. More formally, we define

$$\mathcal{A}_{x,\tilde{x}} := \{\omega \in \Omega | \text{There exists } y \in f_4(x) \text{ such that } ||\tilde{f}_{4,p,\omega}(\tilde{x}) - y|| \leq \varepsilon_4\}.$$

Then using Theorem 3.3.12, we get that $\mathbb{P}(\mathcal{A}_{x,\tilde{x}}) \geq 1 - \frac{1}{n} - \frac{12}{n^2}$. This gives us the desired conclusion. $\tilde{V}$ such that there exists $V \in f_4(x)$ such that $||\tilde{V} - V|| \leq \varepsilon_4$.[1] □

C.2.2 Step 5:

We do not define a new function for Step 5 because it uses the matrix inversion function on a different input. So we reuse f_2 from Section 5.3.2 as the basic function for this step.

C.2.3 Step 6:

Written in standard basis notation, the equality $T' = (A \otimes A \otimes A).T$ corresponds to the fact that for all $i_1, i_2, i_3 \in [n]$,

$$T'_{i_1 i_2 i_3} = \sum_{j_1, j_2, j_3 \in [n]} A_{j_1 i_1} A_{j_2 i_2} A_{j_3 i_3} T_{j_1 j_2 j_3}. \tag{C.4}$$

Lemma C.2.3. *Let $a \in \mathbb{C}^m$ and $b \in \mathbb{C}^n$. Then $||a \otimes b||_F \leq ||a||||b||$.*

Proof. Following the definition of $||.||_F$ of tensors, we get that

$$||a \otimes b||_F^2 = \sum_{i \in [m], j \in [n]} |a_i b_j|^2 \leq (\sum_{i \in [m]} |a_i|^2)(\sum_{j \in [n]} |b_j|^2) = ||a||^2 ||b||^2.$$

□

Defining f_6: Let $X_6 := (\mathbb{C}^n)^{\otimes 3} \times M_n(\mathbb{C})$. Define function $f_6 : X_6 \to \mathbb{C}^n$ as the trace of slices after a change of basis. More formally, $f_6(T, V) = (\mathrm{Tr}(S_1), ..., \mathrm{Tr}(S_n))$ where $S_1, ..., S_n$ are the slices of the tensor $S = (V \otimes V \otimes V).T$. For any element $x = (V, T) \in X_6$, we define $||x|| = \sqrt{||V||_F^2 + ||T||_F^2}$. We define $\kappa_6(x) := (||x|| + 1)^3$. We also define the norm on the output space $\mathbb{C}^n$ as the l_2 norm on vectors.

Lemma C.2.4. *For any $\delta_5 \in (0, 1]$, f_6 is a $(4, 8)$-continuous function at scale δ_5 on the domain X_6.*

Proof. Let $x = (T, V) \in X_6$ and $\tilde{x} = (\tilde{T}, \tilde{V}) \in B((T, V), \delta_5)$. We want to give a bound on $||f_6(\tilde{T}, \tilde{V}) - f_6(T, V)||$. To do this, we bound $||f_6(\tilde{T}, \tilde{V}) - f_6(T, \tilde{V})||$ and $||f_6(T, \tilde{V}) - f_6(T, V)||$ independently and then use triangle inequality to give the required bound.

[1] One can assume that Algorithm 6 gets the desired input exactly. In Appendix C.3, we show that f_4 is continuous and combining this with the numerical stability of Algorithm 6, it can be shown that it is *robust*.

First, we give a bound on $||f_6(\tilde{T},\tilde{V}) - f_6(T,\tilde{V})||$. Let $v_1,...,v_n$ be the columns of V and $\tilde{v}_1,...,\tilde{v}_n$ be the columns of $\tilde{V}$. Let $\tilde{S} = (\tilde{V}\otimes\tilde{V}\otimes\tilde{V}).\tilde{T}$ with slices $\tilde{S}_i$ and $S' = (\tilde{V}\otimes\tilde{V}\otimes\tilde{V}).T$ with slices S'_i. Following the definition of change of basis from (3.5), we have that

$$\begin{aligned}
||f_6(\tilde{T},\tilde{V}) - f_6(T,\tilde{V})||^2 &= \sum_{i=1}^{n} |\mathrm{Tr}(\tilde{S}_i) - \mathrm{Tr}(S'_i)|^2 \\
&= \sum_{i=1}^{n} \Big| \sum_{j=1}^{n} \Big(\tilde{S}_{i,j,j} - S'_{i,j,j} \Big) \Big|^2 \\
&= \sum_{i=1}^{n} \Big| \sum_{j=1}^{n} \sum_{j_1,j_2,j_3=1}^{n} \tilde{V}_{j_1,i}\tilde{V}_{j_2,j}\tilde{V}_{j_3,j}(\tilde{T}_{j_1,j_2,j_3} - T_{j_1,j_2,j_3}) \Big|^2 \qquad \text{(C.5)} \\
&\leq ||\tilde{T} - T||_F^2 \Big(\sum_{i=1}^{n} || \sum_{j=1}^{n} \tilde{v}_i \otimes (\tilde{v}_j)^{\otimes 2} ||_F^2 \Big) \\
&\leq ||\tilde{T} - T||_F^2 \Big(\sum_{i,j=1}^{n} ||\tilde{v}_i \otimes \tilde{v}_j \otimes \tilde{v}_j||_F^2 \Big)
\end{aligned}$$

The second-last inequality follows from an application of the Cauchy-Schwarz inequality and the last inequality follows from triangle inequality. Using Lemma C.2.3 twice, we get that

$$\begin{aligned}
\sum_{i,j=1}^{n} ||\tilde{v}_i \otimes \tilde{v}_j \otimes \tilde{v}_j||_F^2 &\leq (\sum_{i=1}^{n} ||\tilde{v}_i||^2)(\sum_{j=1}^{n} ||\tilde{v}_j \otimes \tilde{v}_j||_F^2) \\
&\leq ||\tilde{V}||_F^2 (\sum_{j=1}^{n} ||\tilde{v}_j||^4) \leq ||\tilde{V}||_F^6.
\end{aligned} \qquad \text{(C.6)}$$

Putting this back in (C.5) and using the fact that $||\tilde{T} - T|| \leq \delta_5$ and $||\tilde{V}||_F \leq ||V||_F + \delta_5 \leq ||V||_F + 1$, we get that

$$||f_6(\tilde{T},\tilde{V}) - f_6(T,\tilde{V})|| \leq \delta_5 ||\tilde{V}||_F^3 \leq \delta_5 (||V||_F + 1)^3. \qquad \text{(C.7)}$$

We now want to give a bound on $||f_6(T,\tilde{V}) - f_6(T,V)||$. Let $S = (V\otimes V\otimes V).T$ and $S_1,...,S_n$ be the slices of S. Expanding along the definition, we get that

$$\begin{aligned}
||f_6(T,\tilde{V}) - f_6(T,V)|| &= \sqrt{\sum_{i=1}^{n} |Tr(S'_i) - Tr(S_i)|^2} \\
&\leq \sqrt{\sum_{i_1,i_2,i_3=1}^{n} |S'_{i_1,i_2,i_3} - S_{i_1,i_2,i_3})|^2} \\
&= \sqrt{\sum_{i_1,i_2,i_3=1}^{n} \Big| \sum_{j_1,j_2,j_3=1}^{n} (\tilde{V}_{j_1 i_1}\tilde{V}_{j_2 i_2}\tilde{V}_{j_3 i_3} - V_{j_1 i_1}V_{j_2 i_2}V_{j_3 i_3}) T_{j_1 j_2 j_3} \Big|^2} \\
&\leq ||T||_F \sqrt{\sum_{i_1,i_2,i_3,j_1,j_2,j_3=1}^{n} \Big| \tilde{V}_{j_1 i_1}\tilde{V}_{j_2 i_2}\tilde{V}_{j_3 i_3} - V_{j_1 i_1}V_{j_2 i_2}V_{j_3 i_3} \Big|^2} \\
&= ||T||_F ||\tilde{V}^{\otimes 3} - V^{\otimes 3}||_F.
\end{aligned} \qquad \text{(C.8)}$$

The last inequality follows again from an application of the Cauchy-Schwarz inequality.

Now we want to bound $||\tilde{V}^{\otimes 3} - V^{\otimes 3}||_F$. Let $E \in M_n(\mathbb{C})$ be such that $\tilde{V} = V + E$. Thus we already have that $||E||_F = ||\tilde{V} - V||_F \leq \delta_5$. Extending Lemma C.2.3 for matrices, we can get that for matrices V_1, V_2, $||V_1 \otimes V_2||_F \leq ||V_1||_F ||V_2||_F$. Applying this and expanding the tensor power, we get that

$$\begin{aligned}||\tilde{V}^{\otimes 3} - V^{\otimes 3}||_F = ||(V+E)^{\otimes 3} - V^{\otimes 3}||_F &\leq 3||V||_F^2||E||_F + 3||V||_F||E||_F^2 + ||E||_F^3 \\ &\leq 3\delta_5(||V||_F^2 + ||V||_F + \frac{1}{3}) \leq 3\delta_5(||V||_F + 1)^2.\end{aligned}$$

In the last step, we use the fact that $\delta_5 \leq 1$. Putting this back in (C.8),

$$||f_6(T, \tilde{V}) - f_6(T, V)|| \leq 3\delta_5(||V||_F + 1)^2||T||_F. \tag{C.9}$$

Combining (C.7) and (C.9), we have

$$||f_6(\tilde{T}, \tilde{V}) - f_6(T, V)|| \leq 3\delta_5(||V||_F + 1)^2||T||_F + \delta_5(||V||_F + 1)^3.$$

Thus, we can finally conclude that

$$||f_6(\tilde{x}) - f_6(x)|| \leq 4\delta_5(||x|| + 1)^3 = 4\delta_5\kappa_6(x). \tag{C.10}$$

Let $x \in X_6$ and $\tilde{x} \in B(x, \delta_5)$. Then

$$\kappa_6(\tilde{x}) = (||\tilde{x}|| + 1)^3 \leq 8(||x|| + 1)^3 = 8\kappa_6(x).$$

□

Defining $\tilde{f}_6$: We define $\tilde{f}_6$ as the numerically stable algorithm for computing a linear combination of the slices after a change of basis (Algorithm 5) with machine precision $u = \alpha_5$.

Lemma C.2.5. *For all $\alpha_5 \in (0, \frac{1}{10n})$, $\tilde{f}_6$ is a (α_5, ψ_5)-numerically stable algorithm for computing f_6 where $\psi_6(\kappa_6) = \frac{63}{8}n^2(\kappa_6)^{\frac{4}{3}}$.*

Proof. Let $x = (T, V)$ be the input to $\tilde{f}_6$ where $T \in (\mathbb{C}^n)^{\otimes 3}$ and $V \in M_n(\mathbb{C})$. Let $f_6(x) = (s_1, ..., s_n)$ and $\tilde{f}_6 = (\tilde{s}_1, ..., \tilde{s}_n)$. Using Theorem 3.2.3, we get that

$$\begin{aligned}||\tilde{f}_6(x) - f_6(x)|| = \sqrt{\sum_{i=1}^{n} ||s_i - \tilde{s}_i||^2} &\leq 14n^2\alpha_5||V||_F^3||T||_F \\ &\leq 14n^2\alpha_5\left(\frac{3||V||_F^2 + ||T||_F^2}{4}\right)^2 \\ &\leq \alpha_5(\frac{63}{8}n^2||x||^4) \leq \alpha_5(\frac{63}{8}n^2(\kappa_6(x))^{\frac{4}{3}}).\end{aligned} \tag{C.11}$$

The second-last inequality follows from an application of the weighted AM-GM inequality. □

Theorem C.2.6. *For any $\varepsilon_5 \in (0, \frac{1}{10n})$, $\tilde{f}_6$ is an (ε_5, ψ_6')-robust numerically stable algorithm for computing f_6 on domain where $\psi_6'(\kappa_6) = 126n^2\kappa_6^{\frac{4}{3}} + 4\kappa_6$.*

Proof. Let $x_6 \in X_6$ and $\tilde{x}_6 \in B(x_6, \varepsilon_5)$. From Lemma 5.2.4, we get that

$$\begin{aligned} ||\tilde{f}_6(\tilde{x}_6) - f_6(x_6)|| &\leq \varepsilon_5.\psi_6'(\kappa_6(x_6)) \\ &= \varepsilon_5(\psi_6(8\kappa_6(x_6)) + 4\kappa_6(x_6)) \\ &= \varepsilon_5\left(126n^2(\kappa_6(x_6))^{\frac{4}{3}} + 4\kappa_6(x_6)\right). \end{aligned}$$

□

C.2.4 Step 7:

In this section we first define the two simple functions $f_7^{(1)}$ (corresponding to computing cube root) and $f_7^{(2)}$ (corresponding to vector scaling) and their corresponding algorithms that are used in Step 7 of Algorithm 8. We define the map f_7 corresponding to Step 7 of the algorithm as a composition of these two simple functions and define the algorithm $\tilde{f}_7$ computing this function as a similar composition of the two corresponding algorithms. We show that $f_7^{(2)}$ is robustly numerically stable and use that to prove bounds on the error made by $\tilde{f}_7$ while computing f_7 on perturbed inputs.

Defining $f_7^{(1)}$: We define $f_7^{(1)}$ as the following cube root map

$$\begin{aligned} f_7^{(1)} : \mathbb{C} &\to \mathcal{P}(\mathbb{C}) \\ \alpha &\mapsto \{\beta | \beta^3 = \alpha\} \end{aligned}$$

Let $x, \tilde{x} \in \mathbb{C}$ such that $|\tilde{x} - x| \leq \delta \leq 1$. Then for a fixed $\tilde{y} \in f_7^{(1)}(\tilde{x})$,

$$|\tilde{x} - x| = \prod_{y \in f_7^{(1)}(x)} |\tilde{y} - y|.$$

This implies, that for all $\tilde{y} \in f_7^{(1)}(\tilde{x})$, there exists $y \in f_7^{(1)}(x)$ such that

$$|\tilde{y} - y| \leq |\tilde{x} - x|^{\frac{1}{3}} \leq \delta^{\frac{1}{3}}. \tag{C.12}$$

We also define $\tilde{f_7}^{(1)}$ as the algorithm for evaluation of cube roots in finite precision arithmetic as mentioned in Section 1.2.2. If the algorithm is executed on $\tilde{x} \in \mathbb{C}$ on a machine with finite precision δ, this gives us that there exists some $\tilde{y'} \in f_7^{(1)}(\tilde{x})$ such that

$$|\tilde{f_7}^{(1)}(\tilde{x}) - \tilde{y'}| \leq \delta|\tilde{y'}|. \tag{C.13}$$

Using (C.12), we also get that there exists $y' \in f_7^{(1)}(x)$ such that $|\tilde{y'} - y'| \leq \delta^{\frac{1}{3}}$. Combining this with (C.13) using triangle inequality, we get that there exists $y' \in f_7^{(1)}(x)$ such that

$$|\tilde{f_7}^{(1)}(\tilde{x}) - y'| \leq \delta|\tilde{y'}| + \delta^{\frac{1}{3}} \leq \delta(\delta^{\frac{1}{3}} + |y'|) + \delta^{\frac{1}{3}} \leq 2\delta^{\frac{1}{3}} + \delta|y'|. \tag{C.14}$$

Defining $f_7^{(2)}$: We define $f_7^{(2)}$ as the following map for vector scaling

$$\begin{aligned} f_7^{(2)} : \mathbb{C} \times \mathbb{C}^n &\to \mathbb{C}^n \\ (\alpha, v) &\mapsto \alpha v \end{aligned}$$

For some $x = (\alpha, v) \in \mathbb{C} \times \mathbb{C}^n$ in the input space, we define $||x|| = |\alpha|^2 + ||v||^2$ and the norm on the output space is the usual l_2 norm on vectors.

Let $x = (\alpha, v), \tilde{x} = (\tilde{\alpha}, \tilde{v}) \in \mathbb{C} \times \mathbb{C}^n$ such that $||\tilde{x} - x|| \leq \delta \leq 1$. Then

$$\begin{aligned} ||f_7^{(2)}(\tilde{x}) - f_7^{(2)}(x)||^2 &= \sum_{i=1}^{n} |\tilde{\alpha}\tilde{v}_i - \alpha v_i|^2 \\ &\leq 2\sum_{i=1}^{n} \Big(|\tilde{\alpha}|^2 |\tilde{v}_i - v_i|^2 + |v_i|^2 |\tilde{\alpha} - \alpha|^2 \Big) \\ &\leq 2\Big(|\tilde{\alpha}|^2 \delta^2 + \delta^2 \sum_{i=1}^{n} |v_i|^2 \Big) \\ &\leq 2\delta^2 \Big(2(|\alpha|^2 + 1) + \sum_{i=1}^{n} |v_i|^2 \Big) \\ &\leq 4\delta^2 (||x||^2 + 1). \end{aligned} \tag{C.15}$$

We also define $\tilde{f_7}^{(2)}$ as the algorithm for vector scaling in finite arithmetic. More formally, it takes in $x = (\alpha, v) \in \mathbb{C} \times \mathbb{C}^n$ and computes $y \in C^n$ such that $y_i = \text{fl}(\alpha \times v_i)$ as defined in Section 1.2.2.

If $\tilde{f_7}^{(2)}$ is executed on $\tilde{x} = (\tilde{\alpha}, \tilde{v}) \in \mathbb{C} \times \mathbb{C}^n$ on a machine with precision δ, this gives us that

$$||\tilde{f_7}^{(2)}(\tilde{x}) - f_7^{(2)}(\tilde{x})|| = \sqrt{\sum_{i=1}^{n} |\text{fl}(\tilde{\alpha}\tilde{v}_i) - \tilde{\alpha}\tilde{v}_i|^2} = \delta|\tilde{\alpha}| \sqrt{\sum_{i=1}^{n} ||\tilde{v}_i||^2} \leq \frac{\delta||\tilde{x}||^2}{2}. \tag{C.16}$$

Combining (C.15) and (C.16) along with the fact that $||\tilde{x}|| \leq ||x|| + \delta \leq ||x|| + 1$, we get that

$$\begin{aligned} ||\tilde{f_7}^{(2)}(\tilde{x}) - f_7^{(2)}(x)|| &\leq \delta \Big(\frac{||\tilde{x}||^2}{2} + 2\sqrt{(||x||^2 + 1)} \Big) \\ &\leq \delta \Big(||x||^2 + 1 + 2\sqrt{(||x||^2 + 1)} \Big) \\ &\leq \delta (\sqrt{||x||^2 + 1} + 1)^2 \leq 2\delta(||x||^2 + 2). \end{aligned} \tag{C.17}$$

Defining f_7: Let the input space to be $X_7 := \mathbb{C}^n \times (\mathbb{C}^n)^n$ and the output space to be $Y_7(n) := (C^n)^n$. Finally we define f_7 to be the following map

$$\begin{aligned} f_7 : X_7 &\to Y_7(n) \\ (\alpha_1, ..., \alpha_n), (v_1, ..., v_n) &\mapsto \{y | y_i = f_7^{(2)}(y_i^{(1)}, v_i) \text{ for all } y_i^{(1)} \in f_7^{(1)}(\alpha_i)\} \end{aligned}$$

We also define $\tilde{f_7}$ to be the following algorithm run with machine precision δ: On input $x = (\alpha_1, ..., \alpha_n), (v_1, ..., v_n) \in X_7$, $\tilde{f_7}$ performs the following computations:

- Computes $y_i^{(1)} = \tilde{f_7}^{(1)}(\alpha_i)$ for all $i \in [n]$ with precision δ .
- Computes $y_i = \tilde{f_7}^{(2)}(y_i^{(1)}, v_i)$ on machine with precision $\mathbf{u} := 6\delta^{\frac{1}{3}}(||x|| + 1)^{\frac{1}{3}}$.
- Outputs $y = (y_1, ..., y_n) \in Y_7(n)$.

The following theorem is the main theorem of this section.

Theorem C.2.7. *Let $x, \tilde{x} \in X_7$ such that $||x - \tilde{x}|| \leq \delta \leq \frac{1}{216(||x||+1)}$. If $\tilde{f}_7$ is run on a machine with precision δ, then there exists $y \in f_7(x)$ such that*

$$||\tilde{f}_7(\tilde{x}) - y|| \leq 2\boldsymbol{u}\Big(2n + (n||x||)^{\frac{2}{3}} + ||x||^2\Big).$$

Proof. Let $x = (\alpha_1, ..., \alpha_n), (v_1, ..., v_n) \in X_7$ and $\tilde{x} = (\tilde{\alpha_1}, ..., \tilde{\alpha_n}), (\tilde{v_1}, ..., \tilde{v_n})$. Since $||\tilde{x} - x|| \leq \delta$, it also follows that $|\tilde{\alpha}_i - \alpha_i| \leq \delta$. Then using (C.14), we get that there exists $y_i^{(1)} \in f_7^{(1)}(\alpha_i)$ such that

$$|\tilde{f}_7{}^{(1)}(\tilde{\alpha_i}) - y_i^{(1)}| \leq \delta|y_i^{(1)}| + 2\delta^{\frac{1}{3}}$$

for all $i \in [n]$. Notice that $\tilde{y}_i = (\tilde{f}_7{}^{(1)}(\tilde{\alpha_i}), \tilde{v}_i)$ and $y_i = (y_i^{(1)}, v_i)$ are the inputs to $\tilde{f}_7{}^{(2)}$ and $f_7^{(2)}$ respectively. Using the facts that $|y_i^{(1)}|^3 = |\alpha_i| \leq ||x||$ and $||\tilde{v}_i - v_i|| \leq ||\tilde{x} - x|| \leq \delta$, this implies that

$$\begin{aligned}
||\tilde{y}_i - y_i|| &= \sqrt{|\tilde{f}_7{}^{(1)}(\tilde{\alpha_i}) - y_i^{(1)}|^2 + ||\tilde{v}_i - v_i||^2} \\
&\leq \sqrt{(\delta|y_i^{(1)}| + 2\delta^{\frac{1}{3}})^2 + \delta^2} \\
&\leq 3\delta^{\frac{1}{3}}(||x||^{\frac{1}{3}} + 1) \\
&\leq 6\delta^{\frac{1}{3}}(||x|| + 1)^{\frac{1}{3}} =: \mathbf{u} \leq 1.
\end{aligned}$$

Running $\tilde{f}_7{}^{(2)}$ with machine precision $\mathbf{u}$ and using (C.17), we get that

$$||\tilde{f}_7{}^{(2)}(\tilde{y}_i) - f_7^{(2)}(y_i)|| \leq 2\mathbf{u}(||y_i||^2 + 2). \tag{C.18}$$

Since, $|y_i^{(1)}|^3 = |\alpha_i|$, using Jensen's inequality on the cube root function, we also get that

$$\frac{\sum_{i=1}^n |y_i^{(1)}|^2}{n} \leq \left(\frac{\sum_{i=1}^n |\alpha_i|^2}{n}\right)^{\frac{1}{3}} \leq \left(\frac{||x||^2}{n}\right)^{\frac{1}{3}} \tag{C.19}$$

Using this along with the fact that $\sum_{i=1}^n ||v_i||^2 \leq ||x||^2$ gives us that

$$\sum_{i=1}^n ||y_i||^2 = \sum_{i=1}^n \Big(|y_i^{(1)}|^2 + ||v_i||^2\Big) \leq (n||x||)^{\frac{2}{3}} + ||x||^2.$$

Putting this back in (C.18), we can conclude that

$$\begin{aligned}
||\tilde{f}_7(\tilde{x}) - f_7(x)|| &= \sqrt{\sum_{i=1}^n ||\tilde{f}_7{}^{(2)}(\tilde{y}_i) - f_7^{(2)}(y_i)||^2} \\
&\leq \sqrt{4\mathbf{u}^2 \sum_{i=1}^n \Big((||y_i||^2 + 2)\Big)^2} \\
&\leq 2\mathbf{u}\Big(2n + \sum_{i=1}^n ||y_i||^2\Big) \\
&\leq 2\mathbf{u}\Big(2n + (n||x||)^{\frac{2}{3}} + ||x||^2\Big).
\end{aligned}$$

□

C.3 Diagonalisation is a $(6n^{\frac{3}{2}}, 4n)$-continuous function

Recall from Section C.2.1 that we had defined f_4 to be the function corresponding to matrix diagonalisation. More formally, it takes as input a diagonalisable matrix A and outputs the set of eigenvectors of A.

Also, recall that in Definition 5.2.1 we had defined the notions of continuity of functions belonging to $\{f : \mathbb{C}^m \to \mathbb{C}^n\}$. This definition can be extended similarly to the case of functions belonging to $\{f : \mathbb{C}^m \to \mathcal{P}(\mathbb{C}^n)\}$. We mention it here for completeness.

Definition C.3.1. *Let $f : S \subset \mathbb{C}^M \to \mathcal{P}(\mathbb{C}^N)$ with condition number κ_f and let $u \in \mathbb{R}_+$. Let $x \in S$ be an input for f such that $B(x,u) \subset S$. We call f to be an (a,b)-continuous function on subdomain S at scale u if for all $\tilde{x} \in B(x,u)$ such that $\tilde{x} \in dom(f)$, $y \in f(x)$ and $\tilde{y} \in f(\tilde{x})$ such that*

- $||f(\tilde{x}) - y|| \leq ua\kappa_f(x)$.
- $\kappa_f(\tilde{x}) \leq b\kappa_f(x)$.

where $a, b \in \mathbb{R}_+$.

We finally show that f_4 is indeed a continuous function.

Claim C.3.2. *f_4 is a $(6n^{\frac{3}{2}}, 4n)$-continuous function at scale $\delta_3 > 0$ on domain $S_4 := \{A$ is diagonalisable and has distinct eigenvalues and $\kappa_{eig}(A) < \frac{1}{8\delta_3}\}$.*

Proof. Let $A \in S_4$ such that $B(A, \delta_3) \subseteq S_4$ let $\tilde{A} \in B(A, \delta_3)$. Using Lemma 3.3.9, $\tilde{A}$ is diagonalisable with distinct eigenvalues and hence $\tilde{A} \in \text{dom}(f_4)$. Then using Corollary 3.3.11.1, we get that there exist eigenvectors $y = (v_1, ..., v_n) \in f_4(A)$ and $\tilde{y} = (\tilde{v}_1, ..., \tilde{v}_n) \in f_4(\tilde{A})$ such that

$$||\tilde{y} - y|| = \sqrt{\sum_{i=1}^{n} ||v_i - \tilde{v_i}||^2} \leq 6n^{\frac{3}{2}}\kappa_{\text{eig}}(A)\varepsilon_3 \tag{C.20}$$

up to scaling by cube roots of unity.

Let $\lambda_1, ..., \lambda_n$ be the eigenvalues of A and $\lambda'_1, ..., \lambda'_n$ be the eigenvalues of $\tilde{A}$. Then using Corollary 3.3.11.1, we get that $|\lambda_i - \lambda'_i| \leq \kappa_V(A)\delta_3$. Therefore, we have that for all $i \neq j \in [n]$,

$$\begin{aligned}\text{gap}(A) &\leq |\lambda_i - \lambda_j| \\ &\leq |\lambda'_i - \lambda'_j| + |\lambda_i - \lambda'_i| + |\lambda_j - \lambda'_j| \\ &\leq |\lambda'_i - \lambda'_j| + 2\kappa_V(A)\delta_3\end{aligned}$$

Since, $\delta_3 < \frac{1}{8\kappa_{\text{eig}}(A)} = \frac{8\text{gap}(A)}{\kappa_V(A)}$, we get that $\text{gap}(\tilde{A}) \geq \frac{3\text{gap}(A)}{4}$. Using Lemma 3.3.9, we also get that $\kappa_V(\tilde{A}) \leq \frac{\kappa_V^F(\tilde{A})}{2} \leq 3n\kappa_V(A)$. Combining these, we finally get that

$$\kappa_4(\tilde{A}) = \kappa_{\text{eig}}(\tilde{A}) = \frac{\kappa_V(\tilde{A})}{\text{gap}(\tilde{A})} \leq \frac{3n\kappa_V(A)}{\frac{3\text{gap}(A)}{4}} = 4n\kappa_{\text{eig}}(A). \tag{C.21}$$

Combining (C.20) and (C.21), we get that f_4 is indeed a $(6n^{\frac{3}{2}}, 4n)$-continuous function on the domain S_4 at scale $\delta_3 > 0$. □

C.4 Appendix to Section 5.6.2

C.4.1 Setting y_5:

The goal of this section is to show that $\tilde{g}_5$ is an $(\varepsilon_4, \varepsilon_5)$-algorithm for computing g_5 on domain $\mathcal{D}_5 = h_4(\mathcal{D}_1)$ (refer to (5.20) for the definition of the functions). We also use the conclusion of Section 5.6.2, that is, for the parameter p mentioned in the hypothesis of Theorem 4.3.6, $\tilde{h}_{4,p}$ is an $((1 - \frac{1}{n} - \frac{12}{n^2}), \varepsilon_0, \varepsilon_4)$-algorithm on probability space $\mathcal{P}$ for computing h_4 on subdomain $\mathcal{D}_1$. Since, $h_5 = g_5 \circ_s h_4$ and $\tilde{h}_{5,p} = \tilde{g}_5 \circ \tilde{h}_{4,p}$, using Corollary 5.4.2.1, we get that $\tilde{h}_{5,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_0, \varepsilon_5)$-algorithm on probability space $\mathcal{P}$ for computing h_5 on domain $\mathcal{D}_1$.

Recall here from Section C.2.2 (and the definition of g_5 in (5.15)) that this corresponds to Step 5 of Algorithm 8 and here we will use the map designed for matrix inversion, i.e. f_2.

Relation to Step 5 of the Algorithm: From the definition of g_4 in (5.16), it takes in as input $y_5 = (V, T) \in \mathcal{D}_5 = h_4(\mathcal{D}) \subseteq X_2 \times (\mathbb{C}^n)^{\otimes 3}$. Since, $g_5 = (\psi_{\text{matrow}} \circ f_5 \circ \pi_1) \times \text{Id}$, the input to f_5 is $x_5 = \pi_1(y_5)$. Then the output at the end of the step is $g_5(y_5) = (\psi_{\text{matrow}}(f_2(x_5)), y_5)$. Since f_2 is the map for matrix inversion defined in Section 5.3.2, we have $g_5(V, T) = ((w_1, ..., w_n), (V, T))$ where $w_1, ..., w_n$ are the rows of V^{-1}. The following is the main result of this section.

Claim C.4.1. *$\tilde{g}_5$ is an $(\varepsilon_4, \varepsilon_5)$-algorithm for computing g_5 on domain $\mathcal{D}_5$.*

Proof idea: The proof strategy for this section is almost the same as Section 5.6.2. Recall that the condition number for f_5, denoted by κ_2 is defined in Section C.2.2. We want to show that $\kappa_2(x_5)$ is bounded and the rest of the proof idea remains the same.

Proof. Let $y_5 \in \mathcal{D}_5$ and $\tilde{y} \in B(y_5, \varepsilon_4)$. We define $x_5 = \pi_1(y_5)$ and $\tilde{x}_5 = \pi_1(\tilde{y}_5)$ as the inputs to f_5 and $\tilde{f}_5$ respectively. Since $||\tilde{y}_5 - y_5|| \leq \varepsilon_4$, this also implies that

$$||\tilde{x}_5 - x_5|| \leq \varepsilon_4. \tag{C.22}$$

In Section 5.3.2, for some $x \in X_2$, we had defined the condition number for f_2 to be $\kappa_2(x) = \kappa_F(x)$. Since $x_5 \in \pi_1(\mathcal{D}_5) = \pi_1(g_4(\mathcal{D}_4))$, using (5.42), we have that $\kappa_2(x_5) = \kappa_F(x_5) \leq n + \frac{B^2}{4}$. Then using this and applying Lemma C.1.1,

$$\varepsilon_4 . \sqrt{\kappa_2(x_5)} \leq \varepsilon_4 . \sqrt{n + \frac{B^2}{4}} \leq \frac{\sqrt{\varepsilon_4}}{2} \leq \frac{1}{8}, \tag{C.23}$$

we can also conclude that $x_5 \in I_2(\varepsilon_4)$. Putting this in Theorem 5.3.11, using (C.22) and Lemma C.1.1, we get that

$$||\tilde{f}_2(\tilde{x}_5) - f_2(x_5)|| \leq \varepsilon_4 . (n \Big(\kappa_2(x_5)\Big)^{\log n})^{\alpha_5} \leq \varepsilon_4 (nB)^{\alpha_5' \log(n)} \leq \frac{\sqrt{\varepsilon_4}}{2} = \frac{\varepsilon_5}{2}. \tag{C.24}$$

The final inequality follows from the definition of ε_i in (5.19). Following the definition of g_5 and $\tilde{g}_5$ from (5.15), we finally get that

$$\begin{aligned}
&||\tilde{g}_5(\tilde{y}_5) - g_5(y_5)|| \\
=&\sqrt{||\psi_{\text{matrow}}(\tilde{f}_2(\pi_1(\tilde{y}_5))) - \psi_{\text{matrow}}(f_2(\pi_1(y_5)))||^2 + ||\text{Id}(\tilde{y}_5) - \text{Id}(y_5)||^2} \\
=&\sqrt{||\tilde{f}_2(\pi_1(\tilde{y}_5)) - f_2(\pi_1(y_5))||^2 + ||\text{Id}(\tilde{y}_5) - \text{Id}(y_5)||^2} \\
\leq&\sqrt{\frac{\varepsilon_5^2}{4} + \varepsilon_4^2} \leq \varepsilon_5.
\end{aligned} \tag{C.25}$$

This shows that $\tilde{g}_5$ is an $(\varepsilon_4, \varepsilon_5)$-algorithm for computing g_5 on subdomain $\mathcal{D}_5$. □

Following the definition of g_i from (5.15), we get that $\pi_1(g_5(\mathcal{D}_5)) = \psi_{\text{matrow}}(f_2(\pi_1(\mathcal{D}_5)))$. For all $y_5 \in \pi_1(g_5(\mathcal{D}_5))$, we compute an upper bound on $||y_5||$, which we will require later in Section C.4.3. From the previous discussion, we know that for $y_5 = (V, T) \in \mathcal{D}_5$, $g_5(y_5) = ((w_1, ..., w_n), (V, T))$ where $w_1, ..., w_n$ are the rows of V^{-1}. Using (5.42),

$$||\pi_1(g_5(y_5))|| = ||f_2(\pi_1(y_5))|| = \sum_{i=1}^{n} ||w_i||^2 = ||V^{-1}||_F \leq \sqrt{\kappa_F(V)} = \sqrt{n + \frac{B^2}{4}}. \tag{C.26}$$

C.4.2 Setting y_6:

The goal of this section is to show that $\tilde{g}_6$ is an $(\varepsilon_5, \varepsilon_6)$-algorithm for computing g_6 on domain $\mathcal{D}_6 = h_5(\mathcal{D}_1)$ (refer to (5.20) for the definition of the functions). We also use the conclusion of Section C.4.1, that is, for the parameter p mentioned in the hypothesis of Theorem 4.3.6, $\tilde{h}_{5,p}$ is an $((1 - \frac{1}{n} - \frac{12}{n^2}), \varepsilon_0, \varepsilon_5)$-algorithm on probability space $\mathcal{P}$ for computing h_5 on subdomain $\mathcal{D}_1$. Since, $h_6 = g_6 \circ_s h_5$ and $\tilde{h}_{6,p} = \tilde{g}_6 \circ \tilde{h}_{5,p}$, using Corollary 5.4.2.1, we get that $\tilde{h}_{6,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_0, \varepsilon_6)$-algorithm on probability space $\mathcal{P}$ for computing h_6 on domain $\mathcal{D}_1$.

Relation to Step 6 of the Algorithm: Following the definition of g_6 from (5.15), g_6 takes as input $y_6 = \Big((w_1, ..., w_n), (V, T)\Big) \in \mathcal{D}_6 = h_5(\mathcal{D}) \subseteq (\mathbb{C}^n)^n \times X_6$. Since $g_6 = (f_6 \circ \pi_2) \times \pi_1$, the input to f_6 is $x_6 = \pi_2(y_6)$. Then the output at the end of the step is $g_6(y_6) = (f_6(x_6), (w_1, ..., w_n))$. Following the definition of f_6 in Section C.2.3 which computes the trace of the slices after a change of basis of the tensor T by the matrix V. More formally, $g_6(y_6) = ((\alpha_1, ..., \alpha_n), (w_1, ..., w_n))$ where $\alpha_i = Tr(S_i)$ for all $i \in [n]$ such that $S = (V \otimes V \otimes V).T$ and $S_1, ..., S_n$ are the slices of S .

Remark C.4.2. *Following the definition of g_5 and g_4 from (5.15), we have that $\pi_2 \circ g_5 = Id$ where Id is the identity function. Using this, we get that there exists some $y \in \mathcal{D}_4$ such that $x_6 = \pi_2(g_5(g_4(y))) = g_4(y) = (V, T)$. Moreover from the definition of g_4, we also get that since (V, T) is in the output space of g_4, for all columns v_i of V, $||v_i|| = 1$. This in turn, implies that $||V||_F \leq \sqrt{n}$.*

The following is the main result of this section.

Claim C.4.3. *$\tilde{g}_6$ is an $(\varepsilon_5, \varepsilon_6)$-algorithm for computing g_6 on domain $\mathcal{D}_6$.*

Proof. Let $y_6 \in \mathcal{D}_6$ and $\tilde{y}_6 \in B(y_6, \varepsilon_5)$ which are the inputs to g_6 and $\tilde{g}_6$ respectively. Then the inputs to $\tilde{f}_6$ and f_6 are set as $\tilde{x}_6 = \pi_2(\tilde{y}_6)$ and $x_6 = \pi_2(y_6) = (V, T)$

respectively. Since, $||\tilde{y}_6 - y_6|| \leq \varepsilon_5$, this in turn implies that

$$||\tilde{x}_6 - x_6|| \leq \varepsilon_5. \tag{C.27}$$

In Section C.2.3, we had defined the condition number for Step 6 to be $\kappa_6(x) = (||x|| + 1)^3$. Using Remark C.4.2 and (5.41), then we get that

$$\kappa_6(x_6) \leq (||x_6|| + 1)^3 \leq (\sqrt{n + B^3} + 1)^3.$$

Putting this in Theorem 5.3.11 and using (C.27) and Lemma C.1.1,it follows that for large enough n and for some appropriate constant m_6',

$$||\tilde{f}_6(\tilde{x}_6) - f_6(x_6)|| \leq \varepsilon_4.(n\Big(\kappa_6(x_6)\Big)^{\log n})^{m_6} \leq \varepsilon_5(nB)^{m_6' \log(n)} \leq \frac{\sqrt{\varepsilon_5}}{2} = \frac{\varepsilon_6}{2}. \tag{C.28}$$

Following the definition of g_6 and $\tilde{g}_6$ from (5.15), using this, we get that

$$\begin{aligned} &||\tilde{g}_6(\tilde{y}_6) - g_6(y_6)|| \\ =&\sqrt{||\pi_1(\tilde{y}_6)) - \pi_1(y_6)||^2 + ||\tilde{f}_6(\pi_2(\tilde{y}_6)) - f_6(\pi_2(y_6))||^2} \\ \leq&\sqrt{\varepsilon_5^2 + \frac{\varepsilon_6^2}{4}} \leq \varepsilon_6. \end{aligned} \tag{C.29}$$

This shows that $\tilde{g}_6$ is an $(\varepsilon_5, \varepsilon_6)$-algorithm for computing g_6 on subdomain $\mathcal{D}_6$. □

Now for all $y_6 \in \mathcal{D}_6$, we want to compute a bound on $||f_6(\pi_2(y_6))||$ which we will use later in Section C.4.3. From the discussion in this section, we already know that $y_6 = ((w_1, ..., w_n), (V, T))$ and $g_6(y_6) = ((\alpha_1, ..., \alpha_n), (w_1, ..., w_n))$. Moreover,since, $\mathcal{D}_6 = h_6(\mathcal{D}_1)$ as defined in (5.21), $y_6 = h_6(y_1)$ for some $y_1 \in \mathcal{D}_1$. From the assumption of Theorem 5.6.2, we get that $y_1 = (T, a, b)$ satisfies the (n, B)-input Conditions 4.3.5 which in turn implies that T is a diagonalisable tensor and $\kappa(T) \leq B$ (refer to Definitions 1.4.3 and 4.1.1 for the corresponding definitions). If U is a matrix that diagonalises the input tensor T, then $||U||_F \leq \sqrt{\kappa_F(U)} \leq \sqrt{B}$. From (4.6), we get that there exist scalars $k_1, ..., k_n \in \mathbb{C}$ such that $V = U^{-1}D$ where $D = \text{diag}(k_1, ..., k_n)$. Hence, $(UV)^T e_i = D^T e_i = k_i e_i$. Then, using (4.7, $|\alpha_i| = |k_i|^3 = ||(UV)^T e_i||^3$. Since using the previous discussion and Remark C.4.2 we have that $||UV||_F \leq ||U||_F ||V||_F \leq \sqrt{nB}$, we can finally conclude that

$$\begin{aligned} ||f_6(\pi_2(y_6))|| = \sqrt{\sum_{i=1}^{n} |\alpha_i|^2} &= \sqrt{\sum_{i=1}^{n} ||(UV)^T e_i||^6} \\ &\leq (\sum_{i=1}^{n} ||(UV)^T e_i||^2)^{\frac{3}{2}} \leq (nB)^{\frac{3}{2}}. \end{aligned} \tag{C.30}$$

C.4.3 Setting y_7:

The goal of this section is to show that $\tilde{g}_7$ is an $(\varepsilon_6, \varepsilon_7)$-algorithm for computing g_7 on domain $\mathcal{D}_7 = h_6(\mathcal{D}_1)$ (refer to (5.20) for the definition of the functions). We also use the conclusion of Section C.4.2, that is, for the parameter p mentioned in the hypothesis of Theorem 4.3.6, $\tilde{h}_{6,p}$ is an $((1 - \frac{1}{n} - \frac{12}{n^2}), \varepsilon_0, \varepsilon_6)$-algorithm on probability space $\mathcal{P}$ for computing h_6 on subdomain $\mathcal{D}_1$. Since, $h_7 = g_7 \circ_s h_6$ and $\tilde{h}_{7,p} = \tilde{g}_7 \circ \tilde{h}_{6,p}$, using Corollary 5.4.2.1, we finally get that $\tilde{h}_{7,p}$ is an $(1 - \frac{1}{n} - \frac{12}{n^2}, \varepsilon_0, \varepsilon_7)$-algorithm on probability space $\mathcal{P}$ for computing h_7 on domain $\mathcal{D}_1$.

Relation to Step 7 of the algorithm: Following the definition of g_7 from (5.15), we know that g_7 takes as input $y_7 = ((\alpha_1, ..., \alpha_n), (w_1, ..., w_n)) \in g_6(\mathcal{D}_6) \subseteq \mathbb{C}^n \times (\mathbb{C}^n)^n = X_7$ and computes $(l_1, ..., l_n)$ where $l_i = \beta_i w_i$ such that $\beta_i^3 = \alpha_i$.

The following is the main result of this section which will complete the proof of Theorem 5.6.2.

Claim C.4.4. *$\tilde{g}_7$ is an $(\varepsilon_6, \varepsilon_7)$-algorithm for computing g_7 on domain $\mathcal{D}_7$.*

Proof. We set $y_7 \in \mathcal{D}_7$ and $\tilde{y}_7 \in B(y_7, \varepsilon_6)$. Recall that X_7 is the domain of definition of f_7, defined in Section C.2.4. Since, $\mathcal{D}_i \subseteq X_7$, then the inputs to $\tilde{f}_7$ and f_7 are $\tilde{y}_7$ and y_7 as well, respectively.

First we compute a bound on $||y_7||$. By the definition of $\mathcal{D}_7$ from (5.21), $y_7 = g_6(y_6)$ for some $y_6 \in \mathcal{D}_6$. Following the definition of $g_6 = (f_6 \circ_s \pi_2) \times \pi_1$ from (5.15), using (C.26) and (C.30),

$$||y_7|| = \sqrt{||f_6 \circ \pi_2(y_6)||^2 + ||\pi_1(y_6)||^2} \leq \sqrt{n^3 B^3 + n + \frac{B^2}{4}} \leq 2n^{\frac{3}{2}} B^{\frac{3}{2}}.$$

Now, for large enough n, $\varepsilon_6 = \frac{1}{n^{c_6 \log^2(\frac{nB}{\varepsilon})}} \leq \frac{1}{216(2n^{\frac{3}{2}} B^{\frac{3}{2}} + 1)} \leq \frac{1}{216(||y_7||+1)}$. Then, we can apply Theorem 5.3.11 along with Lemma C.1.1 to get that there exists some $y_7^{(g)} \in g_7(y_7)$ such that

$$||\tilde{g}_7(\tilde{y}_7) - y_7^{(g)}|| \leq 12\varepsilon_6^{\frac{1}{3}}(\sqrt{n}||x_7|| + 1)^{\frac{1}{3}}(2n + ||x||_7^2 + n^{\frac{2}{3}}||x_7||^{\frac{2}{3}}) \leq 192\varepsilon_6^{\frac{1}{3}} n^{\frac{11}{3}} B^{\frac{7}{2}} \leq (\varepsilon_6)^{\frac{1}{6}} = \varepsilon_7. \tag{C.31}$$

Since $g_7 = f_7$ and $\tilde{g}_7 = \tilde{f}_7$ from (5.15), we finally conclude that $\tilde{g}_7$ is an $(\varepsilon_6, \varepsilon_7)$-algorithm for computing g_7 on subdomain $\mathcal{D}_7$. □