Computational and Algebraic Aspects of Sums of Nonnegative Circuit Polynomials

vorgelegt von
Henning Seidler

an der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 14. Februar 2022

Berlin, 2022

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 26. August 2022

. . . .
Unterschrift

Abstract

In this thesis we introduce a new method how to algorithmically determine lower bounds for multivariate polynomials. It is based on Circuit Polynomials, a class of sparse polynomials, whose nonnegativity is easily checked. The algorithm tries to decompose the given polynomial into a Sum of Nonnegative Circuit Polynomials (SONC) and a constant, therefore providing a lower bound in the majority of cases. Opposed to the established method Sums of Squares (SOS), the complexity of our method does not depend on the degree of the polynomial, but is mainly determined by the number of terms. In fact, our algorithm has a polynomial running time, even when polynomials are given in sparse notation. Hence, our approach is especially suited for sparse polynomials of high degree. Furthermore, we present two extensions of this method. The first one improves the lower bounds by performing a branch-and-bound over the signs of the variables. The running time is FPT in the number of variables, so it remains feasible for a moderate amount of them. In the second extension, we provide a method how, under some additional assumptions, the numerical results of our computations can be transformed into certificates in exact arithmetic.
All of our methods are implemented in the free software POEM in Python and extensively tested. We end with a detailed evaluation, in particular comparing running time and quality of the lower bounds with SOS.

Deutsche Zusammenfassung

In dieser Arbeit stellen wir eine neuartige Methode vor, um algorithmisch untere Schranken für multivariate Polynome zu bestimmen. Die Methode basiert auf Circuit-Polynomen, einer Klasse dünn besetzter Polynome, deren Nichtnegativität einfach überprüft werden kann. Der Algorithmus versucht, ein gegebenes Polynom in eine Summe nichtnegativer Circuit-Polynome und eine Konstante zu zerlegen, wodurch sich im Erfolgsfall eine untere Schranke für die Funktionswerte ergibt. Im Gegensatz zu der etablierten Methode Sums of Squares (SOS), hängt die Komplexität unserer Methode nicht vom Grad des Polynoms ab, sondern hauptsächlich von der Anzahl der Terme. Dadurch hat unser Algorithmus eine polynomielle Laufzeit, selbst wenn das Polynom in dünn besetzter Notation gegeben ist. Somit ist der Algorithmus besonders gut geeignet für dünn besetzte Polynome hohen Grades.
Weiterhin stellen wir zwei Erweiterungen dieser Methode vor. Die erste verbessert die unteren Schranken mittels eines branch-and-bound-Ansatzes, bei welchem die Vorzeichen der Variablen betrachtet werden. Die Laufzeit ist nicht mehr polynomiell, sondern FPT in der Anzahl der Variablen. Damit bleibt das Verfahren anwendbar für moderate Größenordnungen. Die zweite Erweiterung ist ein post-processing, welches unter gewissen zusätzlichen Annahmen in der Lage ist, die numerischen Lösungen umzuwandeln in nachweisbare untere Schranken in exakter Arithmetik.
Alle unsere Methoden sind in der freien Software POEM implementiert und ausgiebig getestet. Wir schließen die Arbeit mit einer detaillierten Evaluation dieser Software, wobei wir insbesondere die praktischen Laufzeiten, die Qualität der gefundenen Schranken und den Vergleich mit SOS untersuchen.

Contents

Chapter 1

Introduction

Finding the global minimum of a given multivariate polynomial is a well-known problem in optimisation. This problem has countless applications, e.g. the Closest-Vector-Problem in lattices, the maximum cut in a graph, or any NP-problem in general. For further concrete examples see Lasserre [Las10]. Closely connected is the decision problem, whether a given multivariate polynomial is nonnegative. This problem already is coNP-hard, as follows from a result by Murty and Kabadi [MK87, Theorem 3]. Therefore, a common approach to certify nonnegativity is to use some sufficient criterion. The most famous approach is sums of squares (SOS), which dates back to Hilbert [Hil88]. This approach has been widely applied with success in recent years; see, e.g. [BPT13, Lau09, Las10, Las15] for an overview.
However, the SOS approach has some serious drawbacks. In 2006, Blekherman proved that for fixed even degree $d \geq 4$ and $n \to \infty$ almost every nonnegative polynomial is not SOS [Ble06]. Furthermore, deciding whether an n-variate polynomial of degree d is SOS translates into a semidefinite programme (SDP) of size $\binom{n+d}{d}$, which quickly becomes infeasible even to state, let alone be solved. For sparse polynomials, i.e. where the support is significantly smaller than all $\binom{n+d}{d}$ possible monomials, this is particularly critical, as it presents an exponential blow-up. In this setting, deciding SOS actually is *harder* than nonnegativity, as for the latter Renegar [Ren88] presented an algorithm which runs in polynomial space and single exponential time. Even when considered in parametrised complexity, deciding SOS for sparse polynomials is in XP parametrised by either the degree d or the number of variables n, which is considered far from being efficient. There are several improvements, how the complexity for deciding SOS can be reduced, e.g. [Rez78, PY19, AM19], but so far all of them suffer from the above exponential blow-up.
We therefore propose an alternative certificate of nonnegativity for sparse polynomials, based on sums of nonnegative circuit polynomials (SONC), which were introduced by Iliman and de Wolff [IdW16a]. The fundamental idea is to decompose some given polynomial into circuit polynomials, whose nonnegativity we can easily verify. This is similar to SOS, where squares are the atomic functions, that are trivially known to be nonnegative. But opposed to SOS, we do not have to introduce new monomials but can instead restrict ourselves to the support of the input polynomial, as shown by Wang [Wan18]. This approach, therefore, is particularly suited for sparse, multivariate polynomials.
The idea of circuit polynomials builds on work by Reznick [Rez89], where he investigated

nonnegativity conditions based on the arithmetic-geometry-mean inequality. Ghasemi and Marshall [GM12, GM13] further pursued this course and introduced new methods to obtain lower bounds for polynomials via Geometric Programming, where they explicitly exploited sparsity of the input. However, these methods were still based on SOS. Hence, their bounds always are at most as good as the bounds obtained via the SDP (if the latter is practically feasible). Furthermore, the approach still requires a considerable amount of time and space for larger parameters. Based on these ideas, Iliman and de Wolff [IdW16b] used geometric programming (GP) to obtain lower bounds for a polynomial in the special case that the Newton polytope of that polynomial is a simplex. Joint with Dressler they extended this to some first examples for computing SONC certificates for polynomials with a non-simplex Newton polytope [DIdW19].
We introduce all these concepts in detail in our preliminaries in Chapter 2, and provide some results about the general complexity of problems related to nonnegativity in Chapter 3.
So, while previous research provided good criteria whether some given decomposition actually is a SONC, prior to our work there was no *algorithm* to actually *obtain* such a decomposition. More precisely, the approach had the following shortcomings.

1. Due to the non-existence of a software for the SONC approach, examples could only be computed with severe effort.
2. As a consequence, no large scale comparison with other approaches had been carried out; the existing examples had to be considered as preliminary data.
3. There existed no algorithm to compute SONC certificates for polynomials with non-simplex Newton polytope.

Our first major contribution is an algorithm, that partially solves these problems by applying some further relaxation and then continues to compute a lower bound that is verifiable with SONC. We present the details in Section 4.1. This algorithm became the start of our software Effective Methods for Polynomial Optimisation (POEM) [SdW19, Sei21b], which is open source and available at the following page.

`https://www.user.tu-berlin.de/henning.seidler/POEM/`

We implemented our algorithm for SONC and provide interfaces for some common packages for SOS. In addition, the software includes (to our knowledge) the first completely open source implementation for SOS.
With this software, we performed a first extensive experiment to compare SONC with SOS, also published in [SdW18]. We investigated the quality of the results, but also the running time in dependence of different input parameters. The main observations, which are also consistent with theoretical analysis, are the following.

- SONC can compute examples far beyond the point where SOS meets its limit due to RAM requirements.
- If SOS yields a result, then it mostly is better then the bound obtained via SONC.
- In every single one of our examples, SONC was the faster method.

However, while our initial algorithm gives highly promising results in these experiments, it has some major drawbacks.

1. The method can only find *some* lower bound and not even the best bound theoretically obtainable via SONC.
2. SONC inherently performs a relaxation, which possibly worsens the results.
3. It does not include a method to find any (local) minimiser, so we can not tell the optimality gap.

While for the third issue we could apply any numerical method for local minimisation, we can exploit the SONC decomposition we found. Based on ideas by de Wolff, this approach is investigated by Müller in [Mül18]. The result is a heuristic, that uses the decomposition to obtain a good starting point for further minimisation. We present this method in Section 4.2.
An important improvement on the first issue comes from Chandrasekaran and Shah, who published another type of certificate of nonnegativity for sparse polynomials, called sums of arithmetic-geometric-mean exponentials (SAGE) [CS16]. They show that we can find such a certificate via relative entropy programming (REP) and together with Murray provide an implementation. It was conjectured and later confirmed by Murray, Chandrasekaran and Wierman [MCW21], that both SONC and SAGE describe the same set of polynomials. As a consequence, the bound obtained from SAGE is at least as good as the bound obtained through our SONC approach, but the complexity is higher. We include SAGE in POEM and also experimentally confirm that our SONC relaxation is faster and can compute bounds for larger examples at the cost of lower quality of the bound. Papp [Pap19] further improved our work by developing an algorithm that finds the optimal SONC decomposition for a polynomial. For given circuits, it finds the decomposition by optimising via the generalised power cone, which is another type of convex optimisation. It continues by identifying an unused circuit, that most improves the solution. This iterative method eventually returns the optimal SONC bound.
However, SONC introduces a further relaxation, corresponding to a slight transformation of the polynomial, where we introduce new negative coefficients. Similarly, SAGE performs a substitution of variables, that comes with the same relaxation. As the second large contribution, we address the second of the above issues through a branch-and-bound framework that branches on the signs of the variables. This framework can be applied both with SONC and SAGE to find improved lower bounds for a polynomial. Alternatively, through a simple analysis of the exponents and the signs of the coefficients, we can easily identify orthants that are minimal in the sense that the infimum lies in one of them. As a drawback, the running time now has an exponential worst case due to the 2^n possible orthants. Still, this running time is feasible for moderate n and often we can significantly reduce this exponential factor. Section 4.3 describes these approaches in detail.
One should note that all methods we mentioned so far rely on the fact, that we can, under some mild assumptions, numerically solve convex optimisation problems in polynomial time. But this means that all constraints also are only satisfied up to some numerical error and it was not clear which impact a small error in the constraint can have on the bound of the polynomial. To overcome these problems, as our third big contribution, we present a post-processing method in Section 4.5, which, under some nondegeneracy assumption, transforms these numerical results for both SONC and SAGE into exact arithmetic. Furthermore, we analyse the error we obtain during this procedure.
All algorithms we present in this thesis are part of the software POEM and we performed

extensive experiments, that we evaluate in Chapter 5. We follow the above order, first evaluating the basic SONC algorithm and compare it to SOS in Section 5.3. Afterwards, we continue with the branch-and-bound framework, comparing our variants and investigating the optimality gap in Section 5.4. The last experiment studies our post-processing method in Section 5.5 with focus on running time and bit size. For each of our algorithms we present some chosen examples and examine the results of running our software on a large set of polynomials with a wide range of parameters.
We conclude the thesis in Chapter 6 with a summary on each of these methods, give some open research questions and present some ideas how to improve our algorithms or the software POEM.

Main Results

From the theoretical analysis and after experimental verification we draw the following conclusions.

- Sums of Nonnegative Circuit Polynomials are a viable alternative to certify nonnegativity of multivariate polynomials that significantly outperforms Sums of Squares already for moderate parameters.
- Our method to find a lower bound via SONC runs in polynomial time, even when the input polynomial is sparse. The running time mainly depends on the number of terms, i.e. the size of the support. It is practically independent of the degree or the number of variables.
- If SOS finds a lower bound it usually is better than the one currently found via SONC, although in general both approaches are incomparable. However, further improvements for SONC have already been made, so the quality of the results shifts in favour of SONC.
- For a moderate increase in the running time, we can further improve the lower bounds found via SONC by using a branch-and-bound approach.
- Under some mild assumptions, we can transform the numerical results into exact arithmetic with an error that we can determine a priori.
- With Effective Methods in Polynomial Optimisation (POEM) we present an easily usable software, that includes all of these algorithms.

Acknowledgements

First and foremost, I thank Meike Hatzel for her continuous support during the whole time. I also want to express my gratitude to Thorsten Koch, Dávid Papp and Jean-Pierre Seifert for reviewing this thesis. To the latter I am also grateful for allowing me to finish my thesis in his group SecT. Further thanks go to the groups MTV and LaS at TU Berlin for their hospitality and the warm welcome I experienced there.

Chapter 2

Preliminaries

In this section, we introduce the notation, which we use throughout the dissertation. Let $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ be the set of nonnegative real and positive real numbers, respectively. In the same manner, we define $\mathbb{Q}_{\geq 0}$ and $\mathbb{Q}_{>0}$. Furthermore, we use the convention $0 \in \mathbb{N}$. Throughout the work, we denote vectors with small bold letters and matrices with capital bold letters. For the entry of a matrix $M_{i,j}$, the first entry i denotes the row and j denotes the column. To denote the i-th row of a matrix M, we use $M^{(i)}$. For a given vector $\boldsymbol{x}$, we denote the j-th coordinate of $\boldsymbol{x}$ by $x_j \in \mathbb{R}$, and $\boldsymbol{x}_{\setminus j} \in \mathbb{R}^{n-1}$ as the vector obtained by $\boldsymbol{x}$ after removing x_j. On some occasions it is more natural, to index entries by vectors. Then for vector $\boldsymbol{\alpha}$, we denote the corresponding entry with $x_{\boldsymbol{\alpha}}$.
When analysing the complexity of a problem, for any object X, let $\|X\|$ denote the size of its encoding.

2.1 Sparse Polynomials and Nonnegativity

Let $\mathbb{R}[\boldsymbol{x}] = \mathbb{R}[x_1, \ldots, x_n]$ denote the ring of real n-variate polynomials. Likewise, define $\mathbb{Q}[\boldsymbol{x}]$. The problem, we wish to solve is the global (unconstrained) optimisation problem

Minimise polynomial MinPoly
Input: $p \in \mathbb{R}[\boldsymbol{x}]$
Task: compute $\inf \{p(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n\}$

Equivalently, we ask for the maximal constant term, we can subtract, such that we obtain a polynomial whose values are all nonnegative.

Minimise polynomial MinPoly
Input: $p \in \mathbb{R}[\boldsymbol{x}]$
Task: compute $\sup \{\gamma : \forall \boldsymbol{x} \in \mathbb{R}^n \,.\, p(\boldsymbol{x}) - \gamma \geq 0\}$

For the latter condition, we simply write $p - \gamma \geq 0$. To certify, that a value is a feasible solution, we have to verify nonnegativity of a polynomial. From a computational point of view, however, it is hard to work with real numbers, since we need a finite encoding of the input. Therefore, we restrict ourselves to rational coefficients. Note that the preimage of the negative numbers $\{\boldsymbol{x} \in \mathbb{R}^n : p(\boldsymbol{x}) < 0\}$ is an open set and the rational numbers lie dense in the real numbers. Thus, we have

$$\forall \boldsymbol{x} \in \mathbb{R}^n \,.\, p(\boldsymbol{x}) \geq 0 \iff \forall \boldsymbol{x} \in \mathbb{Q}^n \,.\, p(\boldsymbol{x}) \geq 0$$

So, we obtain the following decision problem.

Non-negativity NonNeg(p)
Input: polynomial $p \in \mathbb{Q}[x_1, \dots, x_n]$
Question: Is $p(x) \geq 0$ for all $x \in \mathbb{Q}^n$?

The set of nonnegative polynomials in n variables up to degree d we denote $\mathbb{P}_{n,d}$. For historical reasons, dating back to David Hilbert, this decision problem is often formulated in terms of homogeneous polynomials, i.e. polynomials where every term has the same degree. We can easily switch from one notation to the other. Given some nonhomogeneous polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ of degree d, we obtain its homogenisation by introducing a new variable x_0 and set

$$p_{\text{hom}}(x_0, \dots, x_n) := x_0^d \cdot p\left(\frac{1}{x_0} \cdot \boldsymbol{x}\right).$$

Conversely, given a homogeneous polynomial $p_{\text{hom}}(x_0, \dots, x_n)$, we obtain the original polynomial by assigning $x_0 = 1$. Nonnegativity of one polynomial implies nonnegativity of the other. Hence, for the decision problem, we can pick either notation.
In many problems from Computer Science, the optimisation problem and the decision problem are equivalent, because the optimisation problem can be solved by calling polynomially many instances of the decision problem.

2.1.1 Example. We illustrate this claim with the example of Max-Cut, which is a famous NP-complete problem [GJ79]. The problem is defined as follows:

Max-Cut(G)
Input: (undirected) graph $G = (V, E)$
Task: maximise $|\{\{u, v\} : u \in U, v \notin U\}|$ for all $U \subseteq V$

Max-Cut-Decision(G, k)
Input: (undirected) graph $G = (V, E)$, number $k \in \mathbb{N}$
Question: Is there some $U \subseteq V$ with $|\{\{u, v\} : u \in U, v \notin U\}| \geq k$?

The optimal value opt of the optimisation problem is a natural number between 1 and $|E|$, so we can determine this value with $\log(|E|)$ many calls of the decision problem, using binary search. ⬡

For MinPoly however, this is not the case, since we do not have a discrete search space. The solution might even be irrational, which raises the question, how to represent the solution. For example, $p = x^6 - 6x^2$, has the minimum $p(\sqrt{2}) = 8 - 6\sqrt{2}$. But it still allows us, to compute the optimum to arbitrary precision.
Note that we formulated the problem MinPoly with the infimum, since that value is not necessarily attained.

2.1.2 Example. Let $p = (x^2y - 1)^2 + x^2y^2$. Clearly, this polynomial is nonnegative. Furthermore,

$$\lim_{x\to 0} p\left(x, \frac{1}{x^2}\right) = \lim_{x\to 0} x = 0.$$

Therefore, $\inf p = 0$. But whenever $x^2y^2 = 0$, then $x^2y = 0$ and $(x^2y-1)^2 \neq 0$. So $p(x, y) > 0$ for all $x, y \in \mathbb{R}$, which means the infimum is not attained. $\hexagon$

Another problem, is the question, whether MinPoly has a finite solution. To answer this question, we have to solve the following decision problem.

Bounded(p)
Input: polynomial $p \in \mathbb{Q}[x_1, \dots, x_n]$
Question: Is $p(x)$ bounded below?

It turns out that deciding this problem is at least as hard as deciding NonNeg, see Section 3.3. Thus, we are interested in conditions that answer this question in either way. To investigate the complexity of the problem and our algorithms, we fix the following input format. Assume our polynomials are given in expanded form. For a polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ in n variables, we denote its *support* by $A(p) \subset \mathbb{N}^n$, or in case of unambiguity simply by A. For monomials we use the short notation $\boldsymbol{x}^{\boldsymbol{\alpha}} = \prod_{i=1}^n x_i^{\alpha_i}$. Then a polynomial has the form

$$p = \sum_{\boldsymbol{\alpha}\in A(p)} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}}$$

for some coefficient vector $\boldsymbol{b} \in \mathbb{R}^{A(p)}$. For convenience, we do not index the coefficient vector by numbers, but by the corresponding exponent vector. For a given degree d, there are $\binom{n+d}{d}$ possible monomials in n variables. But when writing down a polynomial, we leave out terms with zero coefficients. So we assume, our polynomials are sparse. Let t denote the number of terms, then sparsity means $t = |A| \ll \binom{n+d}{d}$. Therefore the input size heavily depends on the number of terms. When fixing an order on the monomials, e.g. the lexicographic order, we can encode p by a matrix $\boldsymbol{A} \in \mathbb{N}^{n\times t}$ and a vector $\boldsymbol{b} \in \mathbb{Q}^t$. Although this is a slight abuse of notation, it will be clear from the context, whether we use the set notation or the matrix notation. Then in either notation, our input size is bounded by

$$\|p\| \in \mathcal{O}\left(nt\log d + t \cdot \max\left\{\|b_{\boldsymbol{\alpha}}\| : \boldsymbol{\alpha} \in A(p)\right\}\right).$$

From the support of the polynomial, we get an important combinatorial object, called Newton polytope, which is defined as the convex hull of the exponents, i.e.

$$\mathsf{New}\,(p) = \operatorname{conv}\left(A\left(p\right)\right).$$

In Section 2.1.1, we derive some sufficient criteria to answer BOUNDED in either way. These strongly depend on the Newton polytope and the set of its vertices and faces

$$\begin{aligned}\mathsf{Vert}\,(p) &:= \{\boldsymbol{\alpha} \in A\,(p) : \boldsymbol{\alpha} \text{ is a vertex of } \mathsf{New}\,(p)\}\\ \mathsf{Faces}\,(p) &:= \{F \subseteq A\,(p) : F \text{ is face of } \mathsf{New}\,(p)\}\,.\end{aligned}$$

We indicate the elements of the support which are in the relative interior of $\mathsf{New}\,(p)$ by $\mathrm{int}\,(p) = A\,(p) \setminus \partial\mathsf{New}\,(p)$. Note that, beside being a vertex or in the interior, an exponent can also lie on some proper face of the Newton polytope. Finally, we distinguish between monomial squares and terms, which are potentially negative and split up the support accordingly into monomial squares $\mathrm{MoSq}\,(p) := \{\boldsymbol{\alpha} \in A\,(p) : \boldsymbol{\alpha} \in (2\mathbb{N})^n, b_{\boldsymbol{\alpha}} > 0\}$ and its complement in the support, $\mathrm{NoSq}\,(p) = A\,(p) \setminus \mathrm{MoSq}\,(p)$.
Both in these criteria and in our algorithm, we identify a class of exponents, which makes the problem harder to solve. We call these points degenerate points, which are formally defined as

$$\mathscr{D}\,(p) = \{\boldsymbol{\beta} \in \mathrm{NoSq}\,(p) : \exists F \in \mathsf{Faces}\,(p)\ .\, \boldsymbol{\beta} \in F, \mathbf{0} \notin F\}$$

In the respective sections, we describe in further detail the issues, these points cause.

2.1.1 Limits of Multivariate Polynomials

In this section, we investigate some sufficient criteria for a multivariate polynomial to be bounded or unbounded. To this end, we study limits of polynomials and the role of the Newton polytope. We regard the faces of the Newton polytope as directed, with the normal vector pointing to the outside. Throughout this section, let $p = \sum_{\boldsymbol{\alpha} \in A} p_{\boldsymbol{\alpha}} x^{\boldsymbol{\alpha}}$ be a sparse polynomial. Furthermore, let $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ denote the Euclidean scalar product and norm respectively. We can capture asymptotic behaviour of the polynomial in the direction $e^{\boldsymbol{y}} = (e^{y_1}, ..., e^{y_n})$ where $\boldsymbol{y} \in \mathbb{R}^n$ ($\boldsymbol{y}$ is also called the logarithmic coordinates) with the following lemma. The lemma is considered folklore, but for the reader's convenience, we provide a short proof.

2.1.3 Lemma. *Let $\boldsymbol{y} \in \mathbb{R}^n$ denote an arbitrary direction. Let $F \in \mathsf{Faces}\,(p)$ be a face of largest dimension to which $\boldsymbol{y}$ is normal, and let $\boldsymbol{v} \in F$. Then*

$$\lim_{t\to\infty} \frac{p(e^{t\boldsymbol{y}})}{e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle}} = \sum_{\boldsymbol{\alpha} \in A \cap F} b_{\boldsymbol{\alpha}}.$$

The limit does not depend on the choice of $\boldsymbol{v} \in F$.

Proof. Under the above assumptions, if $\boldsymbol{\alpha} \in F$, then $\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle = 0$. Furthermore, $\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle < 0$ for all $\boldsymbol{\alpha} \in A \setminus F$. Therefore, we have

$$\lim_{t\to\infty} \frac{p(e^{t\boldsymbol{y}})}{e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle}} = \lim_{t\to\infty} \sum_{\boldsymbol{\alpha} \in A \cap F} b_{\boldsymbol{\alpha}} e^{t\overbrace{\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle}^{=0}} + \sum_{\boldsymbol{\alpha} \in A \setminus F} b_{\boldsymbol{\alpha}} e^{t\overbrace{\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle}^{<0}} = \sum_{\boldsymbol{\alpha} \in A \cap F} b_{\boldsymbol{\alpha}}. \qquad \square$$

This statement generalises the univariate case, usually denoted by $\mathcal{O}(n^k)$, where the highest exponent determines the growth for $n \to \infty$. Additionally, for a negative exponent y, it catches the behaviour, that the lowest exponent dominates the growth for $n \to 0$. Since the faces of the Newton polytope play a major role in determining limits, for every face F of $\mathsf{New}\,(p)$ we define

$$p_F := \sum_{\boldsymbol{\alpha} \in A \cap F} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}}.$$

From Lemma 2.1.3, we can now derive a sufficient conditions, that some polynomial is unbounded.

2.1.4 Lemma. *If there is some $F \in \mathsf{Faces}\,(p)$ with $\mathbf{0} \notin F$ such that p_F attains a negative value, then p is unbounded below.*

Proof. Assume there is some face F and an argument $\boldsymbol{z} \in \mathbb{R}^n$ such that $p_F(\boldsymbol{z}) < 0$. Since the arguments with negative value form an open set, we can perturb $\boldsymbol{z}$ if necessary, so that we can assume $z_i \neq 0$ for $i = 1, \dots, n$. Then we define $p'(\boldsymbol{x}) := p\left(\langle \boldsymbol{z}, \boldsymbol{x} \rangle\right)$, which means, we apply some invertible linear transformation on $\boldsymbol{x}$. Thus, p is unbounded if and only if p' is unbounded. Furthermore, support and Newton polytope remain the same; only the coefficients change. Let $b'_{\boldsymbol{\alpha}}$ denote the corresponding coefficients of p'. Then

$$0 > p(\boldsymbol{z}) = p'(\mathbf{1}) = \sum_{\boldsymbol{\alpha} \in A \cap F} b'_{\boldsymbol{\alpha}}.$$

Let $\boldsymbol{y}$ be orthogonal to F, but not orthogonal to any higher dimensional face and let $\boldsymbol{v} \in F$ be arbitrary. Since F is a face, we have $\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v} \rangle \leq 0$ for all $\boldsymbol{\alpha} \in A$ with equality if and only if $\boldsymbol{\alpha} \in F$. By our assumption $\mathbf{0} \notin F$, we get $\langle \boldsymbol{y}, \boldsymbol{v} \rangle > 0$. With Lemma 2.1.3, this yields

$$\lim_{t \to \infty} p'\left(e^{t\boldsymbol{y}}\right) = \lim_{t \to \infty} e^{t \overbrace{\langle \boldsymbol{y}, \boldsymbol{v} \rangle}^{>0}} \cdot \overbrace{\sum_{\boldsymbol{\alpha} \in A \cap F} b_{\boldsymbol{\alpha}}}^{<0} \to -\infty,$$

which shows that also p does not have a lower bound. □

The importance of Lemma 2.1.4 lies in the fact, that it provides an easily checked certificate $(F, \boldsymbol{x})$, that some given polynomial is unbounded. In particular, it applies if a non-square forms a vertex of the Newton polytope. Unfortunately, this lemma does not cover all unbounded polynomials.

2.1.5 Example. The polynomial

$$p(x, y) = (x - 2y)^2 - y + 1$$

has a positive limit for every possible choice $\boldsymbol{y}, \boldsymbol{v}, F$ taken in Lemma 2.1.3, but it is unbounded since

$$\lim_{t \to \infty} p(2t, t) = -\infty.$$

This example highlights two problems of the above lemma. First, if p_F has a zero, then, actually, the lower degree terms control the value, even though these terms vanish in the quotient. We will further illustrate this problem later in this section. Second, Lemma 2.1.3 is only able to capture limits of a certain shape. The problem in the latter is, that for every $t \in \mathbb{R}_+$, we have to choose a different direction $\boldsymbol{y}_t = (1, 1 - \ln(2)/t)$. For each of these directions, the maximal orthogonal face is just the vertex $\{(0,2)\}$, but the limit vector $\boldsymbol{y} = (1,1)$ is orthogonal to the face $\{(2,0),(1,1),(0,2)\}$. So, only for the limit, we actually consider the negative term. $\hexagon$

As the next step, we want to develop a criterion that some polynomial is bounded. To this end, we will show that the values will be positive if the norm of the argument is sufficiently large. The limit statement can be rephrased as

$$\forall \varepsilon > 0 \,.\, \forall \boldsymbol{y} \in \mathbb{R}^n \,.\, \forall F \in \mathsf{Faces}\,(p)\,, \text{maximal with } F \perp \boldsymbol{y}\,.$$
$$\exists t_0 \in \mathbb{R}_+ \,.\, \forall t \geq t_0 \,.\, \left| \frac{p(e^{t\boldsymbol{y}})}{e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle}} - \sum_{\boldsymbol{\alpha} \in A \cap F} b_{\boldsymbol{\alpha}} \right| \leq \varepsilon.$$

Here, we would like to have a uniform t_0, that only depends on ε. As we are only interested in the direction given by $\boldsymbol{y}$, we can restrict ourselves to $\|\boldsymbol{y}\| = 1$. However, Example 2.1.5 also shows that

$$\{(\boldsymbol{y}, F) \in \mathbb{R}^n \times \mathsf{Faces}\,(p) : \|\boldsymbol{y}\| = 1, \boldsymbol{y} \perp F, F \text{ maximal}\}$$

is not a closed set. So, we cannot take the maximum of all $t_0 = t_0(\varepsilon, \cdot, \cdot)$. To overcome this issue, we modify the limit statement to also consider faces that are only "nearly" orthogonal to $\boldsymbol{y}$. First, we define

$$\langle \boldsymbol{y}, F\rangle := \max\{\langle \boldsymbol{y}, \boldsymbol{v}' - \boldsymbol{v}\rangle : \boldsymbol{v}', \boldsymbol{v} \in F, \|\boldsymbol{v}' - \boldsymbol{v}\| = 1\}\,. \qquad \text{(face-angle)}$$

This means, we regard the minimal angle between the face F and $\boldsymbol{y}$. Since F is compact, this maximum is well-defined. Based on dimension and degree there is some $\varepsilon = \varepsilon(n, d)$ such that $\langle \boldsymbol{y}, F\rangle \leq \varepsilon$ implies that there is a hyperplane orthogonal to $\boldsymbol{y}$ that separates F from the vertices not in F.

2.1.6 Lemma. *Assume $\langle \boldsymbol{y}, F\rangle \leq \varepsilon := (2n! \cdot d^{n+1})^{-1}$. Let $\boldsymbol{v}', \boldsymbol{v}$ be the vectors maximising (face-angle) with $\boldsymbol{v} \in A$. Then $\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle < 0$ for every $\boldsymbol{\alpha} \in A \setminus F$.*

Proof. Let $\boldsymbol{\alpha} \in A \setminus F$. We show that $\boldsymbol{\alpha}$ cannot lie too close to F. Let $\boldsymbol{\alpha}_F \in F$ be the projection of $\boldsymbol{\alpha}$ onto F. Then there is some $\boldsymbol{\eta} \perp F$ with $\boldsymbol{\alpha} = \boldsymbol{\alpha}_F + \boldsymbol{\eta}$ and $\boldsymbol{\eta} \neq \mathbf{0}$. Recall, that all entries in the vectors of A are bounded by d. So, $\boldsymbol{\eta}$ is the solution of an n-dimensional linear equation system with entries at most d. From Cramer's rule we get $\|\boldsymbol{\eta}\| \geq (n! \cdot d^n)^{-1}$. Additionally, we have the estimate

$$\|\boldsymbol{\alpha}_F - \boldsymbol{v}\| \leq \|\boldsymbol{\alpha} - \boldsymbol{v}\| \leq \|\boldsymbol{\alpha} - \boldsymbol{v}\|_1 \leq 2d,$$

since both vectors are exponents of p, which is of degree d. Furthermore, we decompose $\boldsymbol{y} = \boldsymbol{y}_\perp + \boldsymbol{y}_\parallel$, such that $\boldsymbol{y}_\perp \perp F$ and $\boldsymbol{y}_\parallel$ is parallel to F. Now, $\boldsymbol{y}_\perp$ points to the outside of the polytope, while $\boldsymbol{\alpha} - \boldsymbol{\alpha}_F$ points to the inside. Therefore, we have

$$\begin{aligned}\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle &= \left\langle \boldsymbol{y}_\perp + \boldsymbol{y}_\|, (\boldsymbol{\alpha} - \boldsymbol{\alpha}_F) + (\boldsymbol{\alpha}_F - \boldsymbol{v})\right\rangle \\ &= \langle \boldsymbol{y}_\perp, \boldsymbol{\alpha} - \boldsymbol{\alpha}_F\rangle + \underbrace{\langle \boldsymbol{y}_\perp, \boldsymbol{\alpha}_F - \boldsymbol{v}\rangle}_{=0} + \underbrace{\left\langle \boldsymbol{y}_\|, \boldsymbol{\alpha} - \boldsymbol{\alpha}_F\right\rangle}_{=0} + \left\langle \boldsymbol{y}_\|, \boldsymbol{\alpha}_F - \boldsymbol{v}\right\rangle \\ &\leq \|\boldsymbol{y}_\|\| \cdot \|\boldsymbol{\alpha}_F - \boldsymbol{v}\| - \|\boldsymbol{y}_\perp\| \cdot \|\boldsymbol{\alpha} - \boldsymbol{\alpha}_F\|.\end{aligned}$$

Our assumption $\langle \boldsymbol{y}, F\rangle \leq \varepsilon$ can be written as $\|\boldsymbol{y}_\|\| \leq \varepsilon$. For the orthogonal part, we simply use the estimate

$$\|\boldsymbol{y}_\perp\| = \sqrt{1 - \|\boldsymbol{y}_\|^2\|} \geq \sqrt{1-\varepsilon^2} > 1 - \varepsilon \geq \frac{1}{2}.$$

So, we have the strict inequality

$$\|\boldsymbol{y}_\|\| \cdot \|\boldsymbol{\alpha}_F - \boldsymbol{v}\| \leq \varepsilon d \leq \frac{1}{2}(n! \cdot d^n)^{-1} < \|\boldsymbol{y}_\perp\| \cdot \|\boldsymbol{\alpha} - \boldsymbol{\alpha}_F\|.$$

Together, this yields $\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle < 0$. □

We use these considerations to show a first criterion that guarantees a lower bound.

2.1.7 Lemma. *If for all faces F, we have*

$$\inf\left\{p_F(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n_{\geq 0}\right\} =: \varepsilon_F > 0$$

then p is bounded below on $\mathbb{R}^n_{\geq 0}$.

Proof. We show that outside some radius r, the polynomial only attains positive values in the positive orthant. Let $\|\cdot\|$ denote the maximum-norm and choose $\varepsilon := (2n! \cdot d^n)^{-2}$. For the limits, we regard the set

$$S := \{(\boldsymbol{y}, F) \in \mathbb{R}^n \times \mathsf{Faces}(p) : \|\boldsymbol{y}\| = 1, \langle \boldsymbol{y}, F\rangle \leq \varepsilon, \forall F' \supset F \,.\, \langle \boldsymbol{y}, F\rangle \geq \varepsilon\}.$$

This set is closed and bounded, hence compact. Intuitively, S describes that for each $\boldsymbol{y}$ we choose the largest nearly orthogonal face, but allow multiple faces for some $\boldsymbol{y}$, if $F \subset F'$ with $\langle \boldsymbol{y}, F\rangle = 0$ but $\langle \boldsymbol{y}, F'\rangle = \varepsilon$.
Now let $(\boldsymbol{y}, F) \in S$ and $\boldsymbol{v} \in F$. From Lemma 2.1.6 we get

$$\lim_{t\to\infty} \frac{p\left(e^{t\boldsymbol{y}}\right) - p_F\left(e^{t\boldsymbol{y}}\right)}{e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle}} = \sum_{\boldsymbol{\alpha} \in A \setminus F} b_{\boldsymbol{\alpha}} e^{t \overbrace{\langle \boldsymbol{y}, \boldsymbol{\alpha} - \boldsymbol{v}\rangle}^{<0}} = 0$$

due to our choice of ε. Expanding this limit statement yields

$$\forall \delta > 0 \,.\, \forall (\boldsymbol{y}, F) \in S \,.\, \forall \boldsymbol{v} \in F \,.\, \exists t_0 \in \mathbb{R}_{\geq 0} \,.\, \forall t \geq t_0 \,.\, \left|\frac{p\left(e^{t\boldsymbol{y}}\right) - p_F\left(e^{t\boldsymbol{y}}\right)}{e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle}}\right| \leq \delta.$$

Consider $\boldsymbol{v}$ such that $\langle \boldsymbol{y}, \boldsymbol{v}\rangle \leq 0$. This is possible as

$$\boldsymbol{v} = \operatorname{argmin}\{\langle \boldsymbol{y}, \boldsymbol{\alpha}\rangle : \boldsymbol{\alpha} \in A \cap F\}$$

has that property. Note that $t_0 = t_0(\delta, \boldsymbol{y}, F, \boldsymbol{v})$ depends on all these values. But since S is a compact set, and $\boldsymbol{v}$ functionally depends on $(\boldsymbol{y}, F)$, this function attains its maximum for every δ. In particular, this holds for $\delta = \frac{1}{2}\min\{\varepsilon_F : F \in \mathsf{Faces}(p)\}$. We denote this maximum by

$$T_0 := \max\{t_0(\delta, \boldsymbol{y}, F, \boldsymbol{v}(\boldsymbol{y}, F)) : (\boldsymbol{y}, F) \in S\}.$$

We define the radius as $r := e^{T_0}$. Take some arbitrary $\boldsymbol{x} \in \mathbb{R}^n_+$ with $\|\boldsymbol{x}\| \geq r$. Write $\boldsymbol{x} = e^{t\boldsymbol{y}}$ in logarithmic coordinates for some $t \in \mathbb{R}_+$ and $\boldsymbol{y} \in \mathbb{R}^n$ with $\|\boldsymbol{y}\| = 1$. By choosing a maximal face $F \in \mathsf{Faces}(p)$ with $\langle \boldsymbol{y}, F\rangle \leq \varepsilon$, we get $(\boldsymbol{y}, F) \in S$. Then

$$\left|\frac{p\left(e^{t\boldsymbol{y}}\right) - p_F\left(e^{t\boldsymbol{y}}\right)}{e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle}}\right| \leq \delta,$$

which in turn yields

$$p(\boldsymbol{x}) = p\left(e^{t\boldsymbol{y}}\right) \geq p_F\left(e^{t\boldsymbol{y}}\right) - \delta e^{t\langle \boldsymbol{y}, \boldsymbol{v}\rangle} \geq p_F\left(e^{t\boldsymbol{y}}\right) - \delta \geq \frac{\varepsilon_F}{2} > 0.$$

Therefore, $p(\boldsymbol{x}) > 0$ for $\|\boldsymbol{x}\| \geq r$. As p is also bounded in the ball $B_r(\boldsymbol{0})$, the claim follows. $\square$

From the above, we can derive a criterion that guarantees the existence of a global lower bound for p.

2.1.8 Lemma. *If for every $F \in \mathsf{Faces}(p)$ the polynomial p_F is strictly positive, then p is bounded below on $\mathbb{R}^n$.*

Proof. Each vector $\boldsymbol{s} \in \{-1, 1\}^n$ corresponds to an orthant, where the variables have exactly the signs given by $\boldsymbol{s}$. Put

$$p_{\boldsymbol{s}}(\boldsymbol{x}) = p(s_1x_1, \ldots, s_nx_n).$$

So, the values that $p_{\boldsymbol{s}}$ attains in $\mathbb{R}^n_{\geq 0}$ are exactly those that p attains over the orthant given by $\boldsymbol{s}$. As each $p_{\boldsymbol{s}}$ satisfies the condition of Lemma 2.1.7, p is bounded below on every orthant, which shows the claim. $\square$

While checking positivity of the restricted polynomials p_F may be feasible in practise, checking an exponential number of orthants usually is not. So, we are interested in a criterion, that is easier to check, but still guarantees a global lower bound. As immediate special case of Lemma 2.1.8, we get the following condition.

2.1.9 Corollary. *If all exponents of monomial non-squares lie in the interior of the Newton polytope, then p is bounded below.*

While Lemma 2.1.8 is stronger, Corollary 2.1.9 can be checked in polynomial time. Later in the thesis, we show an improved corollary, based on degenerate points.

Corollary (see Lemma 4.1.10). *If $\mathscr{D}(p) = \emptyset$, then p has a lower bound.*

This condition can also be checked in polynomial time. Although we might have an exponential number of faces, we only have to check those, which contain an element of $\mathrm{NoSq}(p)$. So their number is bounded by the input length.

Example (2.1.5 cont.). *The reason, that none of our lemmata could be applied to our example is, that we have some $F \in \mathsf{Faces}(p)$ such that p_F has a root, but still all restrictions to faces are nonnegative. More precisely, it is $p_F = (x - 2y)^2$ with the root $p_F(2, 1) = 0$. However, to have a root, a face has to contain a degenerate point; in this case it is $\boldsymbol{\alpha} = (1, 2)$.*

Conclusively, we say that degenerate points play a central role in the question, whether a polynomial is bounded. If, due to these points, we have a face F of the Newton polytope such that p_F has a root, then our criteria do not give an answer to the problem BOUNDED. But under standard assumptions in complexity theory, there just is no simple way to certify boundedness for every bounded polynomial. We discuss this in further detail in Section 3.3.

2.2 Complexity

In this section, we introduce the basic notions of complexity. At first, we present some computational models and discuss their merits and disadvantages. Then we introduce polynomial time reductions and the complexity classes P, NP and coNP.
Our definitions and notations follow Arora and Barak [AB09] and Papadimitriou [Pap03], except for Landau-notation.

2.2.1 Definition. Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions. Then we define the Landau-notation as

$$f \in \mathcal{O}(g) \iff \exists c, n_0 \in \mathbb{N} \,.\, \forall n \geq n_0 \,.\, f(n) \leq c \cdot g(n). \qquad \bigcirc$$

It describes that, up to constant factor, f grows at most as fast as g. In many sources (including [AB09]), you can find the alternative notation $f = \mathcal{O}(g)$.
As our basic model for computation, we use the Random Access Machine (RAM), following [Pap03, §2.6], because this model closely matches actual computers. Furthermore, it simplifies the use of a unit cost model. In the RAM, the data is stored in an unbounded number of registers, where each register can store an integer. A programme is a finite sequence of instruction $\pi = (\pi_1, \ldots, \pi_m)$. In addition, we have a process counter κ, that gives the index of the command that is executed next. If $\kappa = 0$ or $\kappa > m$, the programme stops. For each command that does not explicitly set κ, the value is implicitly increased by one.
If the integers in the registers can have arbitrary size, we call this the unit cost model. Otherwise, if we have a fixed bound B on the size, then storing some large number n requires $\lceil \log_B n \rceil$ registers. For actual computers, we usually have $B = 2^{32}$ or $B = 2^{64}$. Hence, the majority of applications outside cryptography are captured well by the unit cost model.
The running time of a programme π on input $\boldsymbol{x} = (x_1, \ldots, x_n)$ is denoted $T_\pi(n, \boldsymbol{x})$ and counts how many instructions were executed. For a fixed programme π, its worst case running time is

$$T(n) = \max \left\{ T_\pi(n, \boldsymbol{x}) : \boldsymbol{x} \in \mathbb{Z}^n \right\}.$$

The preference of RAM over Turing machines (TM) is of no importance. So far, every two models of computation have been shown to be equivalent. In particular, the RAM can be modelled by a Turing machine. More precisely, we have the following property.

2.2.2 Lemma ([Pap03, Theorem 2.5]). *If a boolean function is computable in time $T(n)$ in the RAM model, then it can be computed in time $\mathcal{O}(T(n)^3)$ by a 7-tape Turing machine.*

Even if we allow multiplication in the RAM, this would only change the exponent 3 to 6, which still is a polynomial dependence. This 7-tape machine can in turn be transformed into a single tape TM, with only a polynomial overhead. Therefore, if we do not distinguish between timings that lie within a polynomial factor of each other, we may freely exchange both models. So, when regarding nondeterminism, we do so via Turing machines.
The class P is defined as the class of all languages, that can be recognised by a deterministic Turing machine (or equivalently by a RAM), whose running time is bounded by a polynomial in the input size. Similarly, the class NP (nondeterministic P) is defined as the class of all languages, that can be recognised by a *non*deterministic Turing machine, whose running time is bounded by a polynomial in the input size. Equivalently, NP consists of all problems, where we can easily confirm membership in the language, if we are additionally given a certificate of polynomial size. Analogous to NP, there is the class $\mathsf{coNP} = \left\{L \subseteq \Sigma^* : \overline{L} \in \mathsf{NP}\right\}$. Here, the equivalent formulation means, that a problem A lies in coNP if there is a polynomial size certificate, such that we can easily disprove membership in A. We say a problem is efficiently solvable, if it lies in P.
One of the Millenium problems is to prove or disprove whether $\mathsf{P} = \mathsf{NP}$ [Coo06]. It is widely believed, that P is a strict subset of NP [Gas12], but the problem remains open. A weaker conjecture is $\mathsf{NP} \neq \mathsf{coNP}$, which also is a central open problem in complexity theory.
To better classify the hardness of problems, we use reductions. Informally, if we reduce A to B and we can (efficiently) solve B, then we can (efficiently) solve A.

2.2.3 Definition. Let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be two languages. We say A is polynomially-time reducible to B, written $A \leq_p B$ if there exists a polynomial time computable function $f : \Sigma^* \to \Gamma^*$, i.e. it maps instances of A to instances of B, such that $x \in A \iff f(x) \in B$ for all $x \in \Sigma^*$. ⬡

2.2.4 Definition. A problem A is NP-hard if for every $B \in \mathsf{NP}$ we have $B \leq_p A$. A problem is NP-complete (NPC) if it lies in NP and is NP-hard. In the same manner, we define coNP-hard and coNP-complete. ⬡

In this sense, NP-complete problems are the hardest problems in NP. Analogue, hardness and completeness exist for other complexity classes as well. We are particularly interested in the equivalent notions for the class coNP, because it is connected to the nonnegativity of multivariate polynomials.
We emphasise, that these definitions only cover decision problems. To compute functions, some register or tape is set as output. When the computation halts, the result is the content of this register or tape. As mentioned before, for classical problems like Travelling Salesman, Max-Cut, SAT and many more, the computational problem and the decision problem are equivalent, since an oracle for the decision problem can be used to find an

optimal/feasible solution via bisection. The reason behind this property is the discrete search space. This stands in stark contrast to minimisation of multivariate polynomials. We wish to emphasise this fact, as it is commonly overlooked in the literature.

> Nonnegativity and Minimisation of multivariate, real polynomials are *not* known to be equivalent.

2.2.1 Parametrised Complexity

While classical complexity captures the worst case running time for some input size, this can be a too coarse notion. This is particularly important, if the input consists of multiple parts, which are independent to some extent. In these cases, describing the running time in terms of several parameters leads to a more fine-grained analysis.

2.2.5 Definition. A parametrised problem is a pair (P, κ) such that $P \subseteq \Sigma^*$ is a language and $\kappa : P \to \mathbb{N}$, called the parameter is a function computable in polynomial time. ○

To comply with common notation we will usually give the function κ in textual form. Of practical relevance are problems, where the parameter only influences the running time by a constant factor.

2.2.6 Definition. The class FPT is the class of all parametrised problems (P, κ), where there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial p such that problem X can be decided in time $\mathcal{O}(f(\kappa(X)) \cdot p(\|X\|))$. ○

An example for a problem in FPT is the following.

Vertex Cover
Input: graph G, number $k \in \mathbb{N}$
Question: Does G have a vertex cover of size k?

This problem is known to be NP-complete [GJ79], but can be solved in time $\mathcal{O}\left(3^k \cdot \|G\|\right)$. Note, that increasing the parameter here only affects a factor of the running time, but not the exponent c. So for moderate values of the parameter, these problems can often be solved in practice.
Another common parametrised class is XP, defined as follows.

2.2.7 Definition. The class XP is the class of all parametrised problems (P, κ), where there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that problem X can be decided in time $\mathcal{O}(\|X\|^{f(\kappa(X))})$. ○

We have strict containment FPT $\subset$ XP, see, e.g. [FG06, Corollary 2.26]. In fact, there are more classes between FPT and XP, but since these do not occur in the context of this thesis, we refrain from further details.
For the established method sum-of-squares for polynomial optimisation we often found the running time described as "polynomial time for constant parameter". This exactly matches the class XP. In contrast, the algorithms we describe in this thesis run in polynomial time, or are fixed parameter tractable.

2.3 Logic

In this section, we introduce the basics of first-order logic. By formulating nonnegativity in this form, we can solve the problem using Tarski's quantifier elimination. Furthermore, we present the Existential Theory of the Reals and its connection to nonnegativity.

2.3.1 Structures

We start with introducing signatures and structures, which are the fundamental models in first-order logic.

2.3.1 Definition. A signature is a set of relation symbols and function symbols, each equipped with an arity. A function with arity 0 is also called a constant. ⬡

2.3.2 Definition. Let σ be a signature. A σ-structure $\mathcal{A}$ consists of

- a set A, called universe of $\mathcal{A}$,
- a relation $R^{\mathcal{A}} \subseteq A^k$ for each k-ary relation symbol in σ and
- a function $f^{\mathcal{A}} : A^k \to A$ for each k-ary function symbol in σ.

We write $\mathcal{A} = \left(A, f_1^{\mathcal{A}}, \dots, f_n^{\mathcal{A}}, R_1^{\mathcal{A}}, \dots, R_m^{\mathcal{A}}\right)$ if the order of the symbols is clear. ⬡

The two most important structures we regard are the arithmetic over the real numbers and over the rational numbers.

2.3.3 Example. Let $\sigma = \{+, -, \cdot, <, \leq\}$. For the remainder of this dissertation, we fix the two structures

$$\mathcal{R} = \left(\mathbb{R}, +^{\mathcal{R}}, -^{\mathcal{R}}, \cdot^{\mathcal{R}}, <^{\mathcal{R}}, \leq^{\mathcal{R}}\right)$$
$$\mathcal{Q} = \left(\mathbb{Q}, +^{\mathcal{Q}}, -^{\mathcal{Q}}, \cdot^{\mathcal{Q}}, <^{\mathcal{Q}}, \leq^{\mathcal{Q}}\right)$$

where all operations are interpreted in the usual way. By slight abuse of notation, we leave out the upper index in these cases, to favour readability. ⬡

As we will later see, we can equivalently restrict ourselves to the signature $\sigma = \{+, \cdot\}$, since the other operations can be expressed by these two in first-order logic.

2.3.2 First-Order Logic

Our formulas in first-order logic are defined over a set of variables $\mathsf{Var} = \{x_i : i \in \mathbb{N}\}$. Fixing a signature σ, we define the set of terms $\mathcal{T}_\sigma$ inductively via:

- $x_i \in \mathcal{T}_\sigma$ for all variables $x_i \in \mathsf{Var}$ and $c \in \mathcal{T}_\sigma$ for all constants $c \in \sigma$.
- If $f \in \sigma$ is a function symbol with arity k and $t_1, \dots, t_k \in \mathcal{T}_\sigma$, then $f(t_1, \dots, t_k) \in \mathcal{T}_\sigma$.

Having defined the terms, we now define the set of first-order formulas FO_σ as follows.

- For terms $t, t' \in \mathcal{T}_\sigma$ we have $t = t' \in \mathsf{FO}_\sigma$.

- If $R \in \sigma$ is a relation symbol with arity k and $t_1, \dots, t_k \in \mathcal{T}_\sigma$, then $R(t_1, \dots, t_k) \in \mathsf{FO}_\sigma$.
- If $\varphi, \psi \in \mathsf{FO}_\sigma$ and $x \in \mathsf{Var}$, then

$$\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi), \exists x \,.\, \varphi, \forall x \,.\, \varphi \in \mathsf{FO}_\sigma.$$

This means, equality is always included as a binary relation.
We use the natural interpretation of these formulas, see e.g. [vD94]. We write $\mathcal{A} \models \varphi$, if $\mathcal{A}$ satisfies φ, and say $\mathcal{A}$ is a model for φ. We say two formulas are equivalent, written $\varphi \equiv \psi$, if $\mathcal{A} \models \varphi \iff \mathcal{A} \models \psi$ for all σ-structures $\mathcal{A}$. In case, we require the equivalence only for a given class $\mathfrak{C}$ of σ-structures, we write $\varphi \equiv_{\mathfrak{C}} \psi$, to denote $\mathcal{A} \models \varphi \iff \mathcal{A} \models \psi$ for all $\mathcal{A} \in \mathfrak{C}$. If $\mathfrak{C} = \{\mathcal{A}\}$ is a singleton, we write $\varphi \equiv_{\mathcal{A}} \psi$ instead.
Mostly, we are interested in a formula only up to equivalence. Therefore, in favour of readability, we will leave out brackets in a sequence of conjunctions or in a sequence of disjunctions. Also we say that the unary operators $\neg, \exists, \forall$ bind stronger than the binary operators.

2.3.4 Example. Our original problem NonNeg can now be formulated in terms of first-order logic. Using $\sigma = \{+, -, \cdot, \leq\}$, we can interpret a polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ as a term in $\mathcal{T}_\sigma$. Then deciding NonNeg is equivalent to deciding $\mathcal{R} \models \forall x_1 \dots \forall x_n \,.\, 0 \leq p$. ○

In general, many computational problems in first-order logic are undecidable. A special case is Hilbert's tenth problem, where he asks for an algorithm, that decides satisfiability of a Diophantine Equation.

Diophantine Equation Dioph(p)
Input: $p \in \mathbb{Z}[\boldsymbol{x}]$, multivariate polynomial
Question: Is there $\boldsymbol{x} \in \mathbb{Z}^n$ such that $p(\boldsymbol{x}) = 0$?

Matiyasevich, based on work by Davis, Putnam and Robinson [Rob52, Dav53, DPR61], answered this problem in the negative, by showing that every semi-decidable set can be described by Diophantine Equations [Mat70]. As this includes the Halting Problem, the problem Dioph(p) is undecidable. Furthermore, $p(\boldsymbol{x})^2 > 0$ for all $\boldsymbol{x} \in \mathbb{Z}^n$ if and only if there is no $\boldsymbol{x} \in \mathbb{Z}^n$ with $p(\boldsymbol{x}) = 0$, so over the integers, we cannot decide $p > 0$. Finally, for integers, we have $p(\boldsymbol{x}) > 0$ if and only if $p(\boldsymbol{x}) - 1 \geq 0$, so the problem $p \geq 0$ is undecidable as well.
However, for real closed fields $\mathcal{K}$ over the signature $\sigma = \{+, \cdot\}$, Tarski presented an algorithm for quantifier elimination [Tar98]. This algorithm transforms any given formula $\varphi \in \mathsf{FO}_\sigma$ to a formula $\psi \in \mathsf{FO}_\sigma$ without quantifiers such that $\varphi \equiv_{\mathcal{K}} \psi$. In particular, this holds for the real numbers $\mathcal{R}$, so we can decide problems of the form $\mathcal{R} \models \varphi$.

2.3.3 Existential Theory of the Reals

We fix the extended signature $\sigma = \{+, -, \cdot, \leq, <\}$. Over the Reals, this has the same expressive power as the signature $\{+, \cdot\}$. Although Tarski's quantifier elimination gives an explicit algorithm to decide problems of the form $\mathcal{R} \models \varphi$ for $\varphi \in \mathsf{FO}_\sigma$, this method has a significant drawback. The running time is not an elementary function in the input size. In fact, the running time involves a power tower of height n [BPR06, §11].[1] Instead of using the whole expressional power of first-order logic, we restrict ourselves to a fragment, called the Existential Theory of the Reals. The existential fragment of FO is defined as

$$\exists\mathsf{FO}_\sigma = \{\varphi \in \mathsf{FO}_\sigma : \varphi = \exists x_1 \dots \exists x_n \,.\, \varphi_0, \text{ where } \varphi_0 \text{ is quantifier-free}\} \,.$$

Then we get the following corresponding decision problem.

Existential Theory of the Reals (ETR)
Input: sentence $\varphi \in \exists\mathsf{FO}_\sigma$
Question: Does $\mathcal{R} \models \varphi$ hold?

This problem is known to be decidable in polynomial space and single exponential time, as was shown by Renegar [Ren88]. Apparently, a polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ is nonnegative, if

$$\mathcal{R} \models \exists x_1 \dots \exists x_n \,.\, p(x_1, \dots, x_n) < 0$$

does *not* hold. So, for nonnegativity, the problem is even slightly simplified, as our quantifier-free formula φ_0 only consists of a single predicate.
Similar to classical complexity classes, we say problem A is called ETR-hard, if ETR $\leq_p$ A. If in addition A $\leq_p$ ETR, then we call A ETR-complete. With this notion, we investigate a slight variant of NonNeg, and ask for *strict* positivity of a polynomial.

Positivity Pos(p)
Input: polynomial $p \in \mathbb{Q}[x_1, \dots, x_n]$
Question: Do we have $p(x) > 0$ for all $x \in \mathbb{R}^n$?

It turns out that replacing the inequality by a strict inequality turns our problem into one, whose hardness we can characterise, while the hardness of NonNeg seems to be an open problem.

2.3.5 Theorem. *The problem* $\overline{\text{Pos}}$*, i.e. decide whether* $p(\boldsymbol{x}) \leq 0$ *for some* $\boldsymbol{x} \in \mathbb{R}^n$*, is* ETR*-complete.*

Proof. First observe, that the problem $\overline{\text{Pos}}$ is just a special case of the general ETR.
For the proof of the hardness, we execute some intermediate reductions. First, we construct an equisatisfiable formula, whose quantifier-free part is in 2-SAT. This construction is similar to the NP-hardness of 3-SAT, where we construct a clause for each subformula. Then we eliminate all relations other than equality and finally, we combine all of these polynomial equations into a single sum of squares.
Let $p = \exists \boldsymbol{x} \,.\, \varphi_0$ be our input for ETR. For each subformula ψ of φ_0, we introduce a variable x_ψ that gets assigned value 1 if and only if $\mathcal{R} \models \psi$, and gets assigned 0 otherwise.

[1] A note on stackexchance by Emil Jeřábek, from Czech Academy of Sciences claims time $\exp^{\mathcal{O}(n)}(\|p\|)$

To restrict the value to 0 and 1, we introduce the condition $x_\psi^2 = x_\psi$. Next, we define a clause C_ψ, according to its operator.

$$\begin{array}{lll}
\psi = \psi_1 \wedge \psi_2 & C_\psi := \; x_\psi = x_{\psi_1} x_{\psi_2} & \\
\psi = \psi_1 \vee \psi_2 & C_\psi := \; x_\psi = x_{\psi_1} + x_{\psi_2} - x_{\psi_1} x_{\psi_2} & \\
\psi = \neg\psi_1 & C_\psi := \; x_\psi = 1 - x_{\psi_1} & \\
\psi = \psi_1 \leftrightarrow \psi_2 & C_\psi := \; x_\psi = 1 - x_{\psi_1} - x_{\psi_2} + 2x_{\psi_1} x_{\psi_2} & \\
\psi = R(\boldsymbol{a}) & C_\psi := (x_\psi = 0 \vee R(\boldsymbol{a})) \wedge (x_\psi = 1 \vee \neg R(\boldsymbol{a})) & \text{(atomic formula)}
\end{array}$$

Note, that an iterated replacement $\psi_1 \leftrightarrow \psi_2 \equiv (\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \wedge \neg\psi_2)$ could have led to an exponential growth of the formula. For the remaining negations in the terms $\neg R(\boldsymbol{a})$, we adjust the relations $<, \leq, =, \neq$ accordingly. Now we define

$$\varphi' = \exists \boldsymbol{x} \,.\, \bigwedge_{\psi \in \mathrm{sub}(\varphi_0)} x_\psi^2 = x_\psi \wedge C_\psi$$

For each non-atomic subformula, we create two clauses of constant size. For each atomic subformula, we double the size, plus some constant. Hence the size of φ' is linear in the size of φ. In the next step, we can eliminate all relations other than equality.

$$\begin{array}{lll}
p \neq 0 & \equiv_{\mathcal{R}} & p^2 > 0 \\
p > 0 & \equiv_{\mathcal{R}} & \exists x \,.\, p \cdot x^2 = 1 \\
p \geq 0 & \equiv_{\mathcal{R}} & \exists x \,.\, p = x^2
\end{array}$$

Squaring a polynomial may square the size of the subformula. Next, we rewrite all polynomial equalities in the form $p_i(\boldsymbol{x}) = 0$. Then, we eliminate the remaining disjunctions $p_i = 0 \vee p_j = 0$ to $p_i \cdot p_j = 0$. After the above construction, we only have clauses of size two and p_i can only be of the form $p_i = x_\psi$ or $p_i = x_\psi - 1$ (while p_j can be an arbitrary predicate from the input). Hence, expanding these products can at most double the size of the formula. So we have a formula of the shape

$$\varphi'' = \exists \boldsymbol{x} \,.\, \bigwedge_{i=1}^{m} p_i(\boldsymbol{x}) = 0$$

with $\|\varphi''\| \in \mathcal{O}\left(\|\varphi\|^2\right)$. Finally, by adding all squares of these products, we obtain the final formula

$$\varphi''' = \left(\exists \boldsymbol{x} \,.\, \sum_{i=1}^{m} p_i(\boldsymbol{x})^2 = 0\right) \equiv_{\mathcal{R}} \left(\exists \boldsymbol{x} \,.\, \sum_{i=1}^{m} p_i(\boldsymbol{x})^2 \leq 0\right).$$

Expanding these squares may again square the size of the formula, which leads to $\|\varphi'''\| \in \mathcal{O}\left(\|\varphi\|^4\right)$. Therefore, we have polynomially reduced ETR to the question, whether some polynomial has a nonpositive value. Hence the latter question is ETR-complete. □

If we negate the question of NonNeg, we still get a problem in ETR. Similar to Example 2.3.4, we want to decide $\mathcal{R} \models \exists x_1 \ldots \exists x_n \,.\, p < 0$. However, whether this problem is ETR-complete or whether it lies in NP is an open problem. The difference between $\overline{\text{NonNeg}}$ and $\overline{\text{Pos}}$ is, that the preimage of negative values forms an open set in $\mathbb{R}^n$ and the rational numbers lie dense in there. Hence

$$\mathcal{R} \models \exists x_1 \dots \exists x_n \,.\, p < 0 \iff \mathcal{Q} \models \exists x_1 \dots \exists x_n \,.\, p < 0.$$

This correspondence to the rational numbers might allow some polynomial size certificate. The question is, whether we can impose a polynomial bound on enumerator and denominator. So the problem NonNeg could actually lie in coNP.

2.4 Convex Optimisation

In this section, we present the general concepts of convex optimisation. Afterwards, we discuss three special classes of convex optimisation problems in further detail. For this section, we follow Boyd and Vandenberghe [BV04], with a slight variation for problem reformulations. As discussed before the original problem we are interested in is hard to solve. In contrast, we can efficiently solve convex optimisation problems numerically, under some slight assumptions. Hence, we relax our infeasible problem to some convex problem to obtain an approximate solution.

2.4.1 Definition. A non-linear optimisation problem (NLP) has the form

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimise}} \quad & f_0(\boldsymbol{x}) & & \\ \text{subject to} \quad & f_i(\boldsymbol{x}) \le 0 & & \text{for all } i = 1, \dots, m \\ & h_j(\boldsymbol{x}) = 0 & & \text{for all } j = 1, \dots, \ell \end{aligned} \tag{NLP}$$

for $m, \ell \in \mathbb{N}$ and some functions $f_i, h_j : \mathbb{R}^n \to \mathbb{R}$. We use the following notation and nomenclature.

- f_0 is the objective function.
- The inequalities $f_i(\boldsymbol{x}) \le 0$ and the equalities $h_j(\boldsymbol{x}) = 0$ are called constraints.
- $\boldsymbol{x} \in \mathbb{R}^n$ is the decision variable.
- The feasible set is defined as

$$\mathcal{F} := \{\boldsymbol{x} \in \mathbb{R}^n : \forall i \in [m] \,.\, f_i(\boldsymbol{x}) \le 0, \forall j \in [\ell] \,.\, h_j(\boldsymbol{x}) = 0\}\,.$$

  We say $\boldsymbol{x}$ is feasible if $\boldsymbol{x} \in \mathcal{F}$.

- $p^* = \inf\{f_0(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{F}\} \in \mathbb{R} \cup \{\infty, -\infty\}$ is the optimal value.
- If $f(\boldsymbol{x}^*) = p^*$ for some $\boldsymbol{x}^* \in \mathcal{F}$, we say $\boldsymbol{x}$ is optimal or $\boldsymbol{x}^*$ solves (NLP).
- If f_0 is a constant function, then every $\boldsymbol{x} \in \mathcal{F}$ is optimal. The NLP then is called a feasibility problem.
- If $\boldsymbol{x} \in \mathcal{F}$ with $f(\boldsymbol{x}) \le p^* + \varepsilon$, we say $\boldsymbol{x}$ is ε-suboptimal. ⬡

Similar to decision problems, we define polynomial time reductions, to convert one problem (class) into another. Again this allows us to compare the hardness of problems, but here the focus lies on equivalent problems.

2.4.2 Definition (NLP-reduction). Given two classes of NLPs, $\mathfrak{P}$ and $\mathfrak{Q}$, we say, we can polynomially reduce $\mathfrak{P}$ to $\mathfrak{Q}$, written $\mathfrak{P} \leq_p \mathfrak{Q}$, if there are two functions

$$\begin{aligned} \varphi &: \mathfrak{P} \to \mathfrak{Q} \\ \psi &: \bigcup_{n \in \mathbb{N}} \mathbb{R}^n \to \bigcup_{n \in \mathbb{N}} \mathbb{R}^n \end{aligned}$$

such that both can be computed with a polynomial number of arithmetic (symbolic) operations and if $\boldsymbol{y}^*$ is an optimal solution for $\varphi(P)$, then $\psi(\boldsymbol{y}^*)$ is an optimal solution for P. If both $\mathfrak{P} \leq_p \mathfrak{Q}$ and $\mathfrak{Q} \leq_p \mathfrak{P}$, we say they are equivalent, written $\mathfrak{P} \sim \mathfrak{Q}$. ⬡

For a more intuitive view, we start with some problem instance P and translate the problem into $\mathfrak{Q}$. Then, we solve this new problem $\varphi(P)$ and translate back the solution. This captures the idea of reductions for decision problems, where $\mathfrak{P} \leq_p \mathfrak{Q}$ means "if we can solve $\mathfrak{Q}$, then we can solve $\mathfrak{P}$".
Our definition of equivalence extends the description of Boyd and Vandenberghe [BV04, §4.1.3], where they say

> "We call two problems equivalent if from a solution of one, a solution of the other is readily found, and vice versa."

On the one hand, it covers all examples given in the book, such as

- change of variables,
- transformation of objective and constraints,
- elimination/introduction of equalities, and
- optimisation over some variables.

On the other hand, it does not trivialise the concept of reduction. This is particularly important, because we are interested in convex problems, which are considered as essentially solvable in polynomial time, but we do not want to consider all of these as equivalent. We will give further details below.
An important simplification is the elimination of equalities, by replacing them with two inequalities, because it allows us to restrict ourselves to only inequalities.

2.4.3 Definition. We regard the setting of (NLP). If all functions f_i for $i = 0, \dots, m$ are convex and all functions h_j are affine for $j = 1, \dots, \ell$, then this NLP is called a convex optimisation problem. ⬡

The restriction to affine equality constraints is due to replacing them by $h_j(\boldsymbol{x}) \leq 0$ and $-h_j(\boldsymbol{x}) \leq 0$. Hence, both h_j and $-h_j$ must be convex, which means h_j is an affine function.

The advantage of convex optimisation lies in the property, that every locally optimal solution also is the global optimum. Furthermore, given some accuracy ε and a feasible starting point, we can solve a convex problem up to ε-suboptimality with running time polynomial in $-\log\varepsilon$ and the input size, using interior point methods [NN94].
At this point, we want to emphasise again a detail in our definition of reductions. As mentioned before, we do not want to consider all convex problem to be equivalent. Given a feasible starting point, they can be solved numerically up to some given accuracy in polynomial time. In contrast, in the reduction, we consider the number of symbolic operations working on exact values.
Another way to formulate convex optimisation problems is via some convex cone K. Then, a problem has the standard form

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimise}} \quad & \langle \boldsymbol{c}, \boldsymbol{x}\rangle \\ \text{subject to} \quad & \boldsymbol{A}\cdot\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x}\in K. \end{aligned} \tag{conic}$$

This allows for a concise representation, but for computational analysis, one needs further details, how the cone is given.
For the remainder of this work, we are interested in three particular types of convex optimisation problems: semidefinite programming, geometric programming and relative-entropy programming.

2.4.1 Semidefinite Programming (SDP)

A matrix $\boldsymbol{X}\in\mathbb{R}^{n\times n}$ is positive semi-definite (psd) if it has one of these (equivalent) properties.

- $\boldsymbol{v}^T\boldsymbol{X}\boldsymbol{v}\geq 0$ for all $\boldsymbol{v}\in\mathbb{R}^n$,
- all eigenvalues of $\boldsymbol{X}$ are nonnegative, or
- There is some matrix $\boldsymbol{Y}\in\mathbb{R}^{n\times n}$ such that $\boldsymbol{X}=\boldsymbol{Y}^T\boldsymbol{Y}$.

The set of all these matrices, which forms a convex cone, is denoted $\mathbb{S}^n_+$. We use the common notation $\boldsymbol{X}\succeq\boldsymbol{0}$ to say that $\boldsymbol{X}$ is psd. Based on this convex cone, we get the class of semi-definite programmes (SDP), which have the form

$$\begin{aligned} \underset{\boldsymbol{X}\in\mathbb{R}^{n\times n}}{\text{minimise}} \quad & \langle \boldsymbol{C}, \boldsymbol{X}\rangle \\ \text{subject to} \quad & \langle \boldsymbol{A}_i, \boldsymbol{X}\rangle = b_i \qquad \text{for all } i=1,\dots,\ell \\ & \boldsymbol{X}\succeq\boldsymbol{0}. \end{aligned} \tag{SDP}$$

for matrices $\boldsymbol{C},\boldsymbol{A}_i\in\mathbb{R}^{n\times n}$ and some $b_i\in\mathbb{R}$, where $\langle\boldsymbol{A},\boldsymbol{X}\rangle=\mathsf{trace}\,(\boldsymbol{A}\boldsymbol{X})$ denotes the scalar product of the vectorisation of the matrices.
Their complexity analysis is given by Nesterov and Nemirovskii [NN94, §6.4]. Given a feasible starting point, we can solve SDPs up to accuracy ε in $\mathcal{O}\left(n^{4.5}|\log\varepsilon|\right)$ arithmetic steps with the primal-dual-method.

2.4.2 Geometric Programming (GP)

A positive monomial is a function of the form $h(\boldsymbol{x}) = c\boldsymbol{x}^{\boldsymbol{\alpha}}$, where $c \in \mathbb{R}_{>0}$ and $\boldsymbol{\alpha} \in \mathbb{R}^n$. A posynomial is a sum of positive monomials, i.e. of the form

$$f(\boldsymbol{x}) = \sum_{k \in I} c_k \boldsymbol{x}^{\boldsymbol{\alpha}_k}$$

for some finite index set I with $c_k > 0$ for all k. Then a Geometric Programme (GP) in standard form is an optimisation problem of the form

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^n_{>0}}{\text{minimise}} \quad & p_0(\boldsymbol{x}) \\ \text{subject to} \quad & p_i(\boldsymbol{x}) \leq 1 \qquad \text{for all } i = 1, \dots, m \\ & m_j(\boldsymbol{x}) = 1 \qquad \text{for all } j = 1, \dots, \ell, \end{aligned} \tag{GP}$$

for some $m, \ell \in \mathbb{N}$ where the p_i are posynomials and the m_j are positive monomials. More generally, we also allow constraints $p_i(\boldsymbol{x}) \leq a_i$ and $m_j(\boldsymbol{x}) = b_j$ for $a_i, b_j > 0$. In standard form, GPs are not convex, but we can transform them into convex problems. We perform a transformation of variables $y_i = \log x_i$ and logarithmise each constraint as follows. Assume, the functions in the constraints have the form

$$p_i(\boldsymbol{x}) = \sum_{k=1}^{K_i} c_{i,k} \boldsymbol{x}^{\boldsymbol{\alpha}_{i,k}} \qquad\qquad m_j(\boldsymbol{x}) = c_j \boldsymbol{x}^{\boldsymbol{\beta}_j}.$$

Note, that the log-sum-exp function

$$\mathsf{LSE}\,(\boldsymbol{x}) = \log \sum_{i=1}^{n} e^{x_i}$$

is a convex function. So we can rewrite (GP) to obtain the Geometric Programme in convex form

$$\begin{aligned} \underset{\boldsymbol{y} \in \mathbb{R}^n}{\text{minimise}} \quad & \mathsf{LSE}\left((\langle \boldsymbol{\alpha}_{0,k}, \boldsymbol{y} \rangle + \log c_{0,k})_{k=1}^{K_i} \right) \\ \text{subject to} \quad & \mathsf{LSE}\left((\langle \boldsymbol{\alpha}_{i,k}, \boldsymbol{y} \rangle + \log c_{i,k})_{k=1}^{K_i} \right) \leq 0 \qquad \text{for all } i = 1, \dots, m \\ & \langle \boldsymbol{\beta}_j, \boldsymbol{y} \rangle + \log c_j = 0 \qquad \text{for all } j = 1, \dots, \ell, \end{aligned} \tag{GP-conv}$$

which is a convex optimisation problem. We can now use the linear equations to eliminate up to ℓ variables. So we transformed (GP) into a reduced problem with just m inequalities in $n - \ell$ variables.

$$\begin{aligned} \underset{\boldsymbol{y} \in \mathbb{R}^{n-\ell}}{\text{minimise}} \quad & f_0(\boldsymbol{y}) := \mathsf{LSE}\left(\left(\langle \boldsymbol{\alpha'}_{0,k}, \boldsymbol{y} \rangle + \log c'_{0,k} \right)_{k=1}^{K_i} \right) \\ \text{subject to} \quad & f_i(\boldsymbol{y}) := \mathsf{LSE}\left(\left(\langle \boldsymbol{\alpha'}_{i,k}, \boldsymbol{y} \rangle + \log c'_{i,k} \right)_{k=1}^{K_i} \right) \leq 0 \qquad \text{for all } i = 1, \dots, m \end{aligned} \tag{GP'}$$

To analyse the running time for solving a GP, we follow Nesterov and Nemirovskii [NN94, §6.3.1]. We need the further assumption, that our search space is contained in some ball $B_R(\mathbf{0})$ with $R \in \mathbb{R}_+$. In order to keep the analysis as simple as possible, we still use n as an upper bound for the number of variables. Furthermore, we define upper bounds for the values

$$\begin{aligned} a &:= \max\left\{(\alpha'_{i,k})_j : 1 = j, \ldots, n; i = 1, \ldots, m; k = 1, \ldots, K_i\right\} \\ c &:= \max\left\{c'_{i,k} : 1 = j, \ldots, \ell; i = 1, \ldots, m; k = 1, \ldots, K_i\right\}. \end{aligned}$$

Then our function values are bounded above by

$$f_i \leq \mathsf{LSE}\left((anR + \log c)_{i=1}^{K_i}\right) = \log \sum_{i=1}^{K_i} ce^{anR} = anR + \log(cK_i) =: V.$$

From this number, we define

$$V^+ = c + V \qquad\qquad V^\# = (1 + \max\{K_i : i \leq m\}) \cdot V^+.$$

Let k be the number of different values $\boldsymbol{\alpha}'_{i,k}$, which is bounded by the input size $\|(GP)\|$. Then the number of arithmetic operations needed to solve the problem (GP) up to accuracy ε is bounded by

$$\mathcal{O}\left(k(m+n)(n+k)(m+k)^{\frac{1}{2}} \cdot \ln\left(\frac{(m+k)V^\#}{\varepsilon}\right)\right).$$

Note, that ε and $V^\#$ only appear inside the logarithm. Furthermore, $\log V^\#$ is bounded by a polynomial in $\log a, \log c$ and the number of terms and inequalities. So the whole running time is polynomial in the input size.

2.4.3 Relative Entropy Programming (REP)

Relative Entropy Programmes were introduced by Chandrasekaran and Shah [CS17] and form a generalisation of GPs. The name comes from the relative entropy function $D : \mathbb{R}^n_{>0} \times \mathbb{R}^n_{>0} \to \mathbb{R}$, defined via

$$D(\boldsymbol{\nu}, \boldsymbol{\lambda}) = \sum_{i=1}^n \nu_i \cdot \log\left(\frac{\nu_i}{\lambda_i}\right),$$

which is an important measure in information theory and statistics. Here, we use, that this function is convex in both arguments. The relative entropy cone is defined as

$$\mathbb{RE}_n = \left\{(\boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\delta}) \in \mathbb{R}^n_{>0} \times \mathbb{R}^n_{>0} \times \mathbb{R}^n : \forall i \,.\, \nu_i \log\left(\frac{\nu_i}{\lambda_i}\right) \leq \delta_i\right\}.$$

This cone now gives rise to relative entropy programmes (REP), which are problems of the form

$$\begin{array}{cl} \underset{(\boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\delta}) \in \mathbb{R}^n_{>0} \times \mathbb{R}^n_{>0} \times \mathbb{R}^n}{\text{minimise}} & \langle \boldsymbol{c}, (\boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\delta}) \rangle \\ \text{subject to} & \boldsymbol{A}(\boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\delta})^T = \boldsymbol{b} \\ & (\boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\delta}) \in \mathbb{RE}_n \end{array} \tag{REP}$$

for some vectors $\boldsymbol{c} \in \mathbb{R}^{3n}, \boldsymbol{b} \in \mathbb{R}^m$ and some matrix $\boldsymbol{A} \in \mathbb{R}^{m \times 3n}$. An equivalent formulation is given via the exponential cone, defined as

$$\mathbb{EXP} = \left\{(x,y,z) \in \mathbb{R}^3 : y > 0, ye^{x/y} \leq z\right\} \cup \left\{(x,y,z) \in \mathbb{R}^3 : x \leq 0, y = 0, z \geq 0\right\}.$$

The equivalence is given via $(\nu, \lambda, \delta) \in \mathbb{RE}_1$ if and only if $(-\delta, \nu, \lambda) \in \operatorname{int}(\mathbb{EXP})$. So we can solve REPs to arbitrary precision ε by using the methods and solvers available for the exponential cone. The required time is polynomial in the input size and $-\log \varepsilon$, as shown by Nesterov and Nemirovskii [NN94].

2.5 Sums of Squares

Squares form a basic class of nonnegative expressions, due to the simple fact that $x^2 \geq 0$ for all $x \in \mathbb{R}$. Likewise, for polynomials we get $p^2 \geq 0$ for all $p \in \mathbb{R}[\boldsymbol{x}]$. Summing up such expressions, we define the cone of all sums of squares (SOS) in n variables as

$$\Sigma_n^{\mathsf{SOS}} = \left\{p \in \mathbb{R}[\boldsymbol{x}] : p = \sum_{\text{finite}} q_i^2 \text{ for some } q_i \in \mathbb{R}[\boldsymbol{x}]\right\}.$$

If we additionally restrict the degree, we get the cone

$$\Sigma_{n,2d}^{\mathsf{SOS}} = \left\{p \in \Sigma_n^{\mathsf{SOS}} : \deg(p) \leq 2d\right\}.$$

Both are, in fact, cones, since they are trivially closed under addition and scaling with any $c \geq 0$ yields

$$c \cdot \sum_{\text{finite}} q_i^2 = \sum_{\text{finite}} \left(\sqrt{c} \cdot q_i\right)^2.$$

First, we observe that $p \in \Sigma_n^{\mathsf{SOS}}$ implies $p \geq 0$. The question, when the converse holds, was most popularly investigated by David Hilbert in his seminal work "Über die Darstellung definiter Formen als Summe von Formenquadraten" [Hil88]. In that article, he shows the following theorem (although he states it for homogeneous polynomials).

2.5.1 Theorem (Hilbert, 1888). *Nonnegative polynomials and sums of squares coincide, i.e.* $\Sigma_{n,2d}^{\mathsf{SOS}} = \mathbb{P}_{n,2d}$*, if and only if we are in one of the following three cases*

- $n = 1$,
- $2d = 2$, *or*
- $(n, 2d) = (2, 4)$.

Although this shows the existence of counterexamples for all other cases, the first explicit one was found only in 1967 by Motzkin.

2.5.2 Example (Motzkin, 1967). The Motzkin polynomial is defined as $p = x^4y^2 + x^2y^4 + 1 - 3x^2y^2$. Using the arithmetic-geometric-mean inequality

$$x^2y^2 = \sqrt[3]{x^4y^2 \cdot x^2y^4 \cdot 1} \leq \frac{x^4y^2 + x^2y^4 + 1}{3}$$

we get $p(x,y) \geq 0$ for all $x, y \in \mathbb{R}$.

Now suppose $p = q_1^2 + \ldots + q_t^2$ with $q_i \in \mathbb{R}[x,y]$. Each q_i must be a polynomial of degree at most 3. If some q_i contains the monomial x^d for $d \geq 3$, then the leading coefficient of x^{2d} would be a square, hence positive. Likewise, none of the q_i can contain one of the monomials y^3, y^2, y, x^2, x. So they are generated by the monomials $x^2y, xy^2, xy, 1$. Thus, the only way, to obtain the monomial x^2y^2 is to square xy, which always leads to a nonnegative coefficient. Hence, f cannot be written as a sum of squares. $\bigcirc$

In Section 2.5.1 we briefly talk about Hilbert's 17th problem and how this problem can be avoided by using squares of rational functions. It turns out, that examples like the above become more frequent with growing number of variables. As Blekherman shows [Ble06], for every fixed degree $2d \geq 4$ and $n \to \infty$, almost no nonnegative polynomial is a sum of squares.
More recently, sums of squares received increased attention in the context of polynomial optimisation. As mentioned before in Section 2.1 (proven in Theorem 3.2.1), deciding Nonnegativity is coNP-hard. So Lasserre [Las01] and Parillo [Par00] proposed to use the SOS-"relaxation". Instead of computing

$$p^* = \max\left\{\gamma \in \mathbb{R} : p - \gamma \geq 0\right\},$$

we compute

$$p^*_{\text{SOS}} = \max\left\{\gamma \in \mathbb{R} : p - \gamma \in \Sigma_n^{\mathsf{SOS}}\right\}. \qquad \text{(SOS-bound)}$$

Trivially, we have $p^*_{\text{SOS}} \leq p^*$. Let $2d = \deg(p)$. We can theoretically compute p^*_{SOS} by translating (SOS-bound) into the following SDP

$$\begin{aligned} p^*_{\text{SOS}} = \quad & \underset{\boldsymbol{X} \in \mathbb{R}^{\binom{n+d}{d} \times \binom{n+d}{d}}}{\text{maximise}} && \gamma \\ & \text{subject to} && X_{0,0} = b_{\mathbf{0}} - \gamma \\ & && \sum_{\substack{\boldsymbol{\beta} \leq \boldsymbol{\alpha} \\ |\boldsymbol{\beta}| \leq d}} X_{\boldsymbol{\beta}, \boldsymbol{\alpha}-\boldsymbol{\beta}} = b_{\boldsymbol{\alpha}} \qquad \text{for all } \boldsymbol{\alpha} \in \mathbb{N}^d, 1 \leq |\boldsymbol{\alpha}| \leq 2d \\ & && \boldsymbol{X} \succeq 0, \end{aligned} \qquad \text{(SDP-SOS)}$$

where we say $b_{\boldsymbol{\alpha}} = 0$ for $\boldsymbol{\alpha} \notin A(p)$. However, this approach has a serious drawback, which is widely ignored in the literature: The matrix of variables has size $\binom{n+d}{d}^2$ and we have $\binom{n+2d}{2d}$ constraints, and even their length depends on the input. Recall, that we have input size

$$\mathcal{O}\left(nt \log d + t \cdot \max\left\{\log b_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in A(p)\right\}\right).$$

So the input is essentially linear in n and in $\log d$. We can estimate the size of (SOS-bound) as

$$\binom{n+d}{d} \geq \frac{d^n}{d!} \geq \frac{d^n}{d^d} = d^{n-d} = 2^{\log d \cdot (n - \log d)}$$

which is exponential in both n and $\log d$. While an SDP can essentially be solved in time polynomial in its problem size (see Section 2.4.1), this approach is computationally infeasible, because the problem size is *exponential* in the input size. The further relaxations DSOS and SDSOS [AM19] also do not solve this problem. While they replace the SDP with an easier solvable convex problem, their optimisation problems still have the exponential size $\binom{n+2d}{2d}$.
A slight improvement is due to Reznick [Rez78], where he presented a possible reduction of the problem size.

2.5.3 Theorem. *If $p = \sum_i q_i^2$, then $\mathsf{New}(g_i) \subseteq \frac{1}{2}\mathsf{New}(p)$ for all i.*

Therefore, in the SDP we only have to consider those points, which lie in the scaled Newton polytope $\frac{1}{2}\mathsf{New}(p)$. But in the worst case of the scaled standard simplex, this does not give any reduction at all. Even in the average case, this does not reduce the problem size enough to be practically feasible.
Nevertheless, many publications claim, that we can "efficiently decide, whether a polynomial is a sum of squares" or an equivalent phrasing, e.f. [Par00, p. 43], [Mar08, p. xii]. The problem is, that most of these publications neither provide a definition of their input format, nor what they mean by "efficiently".
The most likely interpretation is, that for the SOS approach, a polynomial is considered as its coefficient vector $\boldsymbol{b} \in \mathbb{R}^{\binom{n+2d}{2d}}$, i.e. every possible monomial is listed, even if the coefficient is zero. Given this input size, the problem (SDP-SOS) can be solved in polynomial time, which is the usual meaning of "efficient". However, Parillo [Par00, Def. 4.1] explicitly introduces a sparse notation, equivalent to this thesis.
Other publications say that the problem is solvable in polynomial time if n or d is fixed, but this is just the complexity class XP, which is usually considered far from being efficiently solvable.
To the best of our knowledge, every known algorithm to decide SOS requires at least exponential space. Considering, that nonnegativity can be decided in polynomial space and single exponential time, deciding SOS seems actually *harder* than deciding nonnegativity.

2.5.1 Hilbert's 17th Problem

As stated in Theorem 2.5.1, there are nonnegative polynomials, that cannot be written as a sum of squares of polynomials. However, a sum of squares of *rational functions* forms a certificate of nonnegativity as well. So in his famous list of 23 problems Hilbert asks, whether every nonnegative polynomial can be written as a sum of squares of rational functions. This was answered affirmatively by Artin in 1927 [Art27]. We briefly sketch the essential steps of the proof.
Let K be a field and $\sigma = \{+, \cdot\}$ be the signature of arithmetic. We define the expansion

$$\sigma[K] := \sigma \cup \{c_a : a \in K\}$$

where we add a constant symbol for every element in K. Then we naturally obtain a structure $\mathcal{K}$, where we interpret the arithmetic operations in the usual way and interpret every symbol c_a with a.
Furthermore, we need the following two theorems, which can be found, e.g. in [BCR13, §5.2]. We also follow that book for the proof of Hilbert's 17th problem.

2.5.4 Theorem (Tarski '31, Seidenberg '54). *Let K be a real field and $\varphi \in \mathsf{FO}_{\sigma[K]}$ with free variables $x_1, \dots, x_n$. Then there is a quantifier-free formula $\psi \in \mathsf{FO}_{\sigma[K]}$ with the same free variables such that for all real closed extensions $K \leq R$ and all $\boldsymbol{a} \in R^n$ we have*

$$\mathcal{R} \models \varphi(\boldsymbol{a}) \iff \mathcal{R} \models \psi(\boldsymbol{a}).$$

2.5.5 Theorem (Transfer Principle). *Let $K \leq R$ be an extension of real closed fields with corresponding $\sigma[K]$-structures $\mathcal{K}, \mathcal{R}$. Let $\varphi \in \mathsf{FO}_{\sigma[K]}$ be a sentence. Then $\mathcal{K} \models \varphi \iff \mathcal{R} \models \varphi$.*

Proof. Let ψ be a quantifier-free sentence for φ, as given by Theorem 2.5.4. Since we have no variables in ψ, the additional elements in the universe have no influence on the evaluation, so $\mathcal{K} \models \psi$ if and only if $\mathcal{R} \models \psi$. So by Theorem 2.5.4 we obtain

$$\mathcal{K} \models \varphi \iff \mathcal{K} \models \psi \iff \mathcal{R} \models \psi \iff \mathcal{R} \models \varphi. \qquad \square$$

With these two theorems, we can provide the proof of Hilbert's problem.

2.5.6 Theorem (Hilbert's 17th problem, Artin 1927). *Let $p \in \mathbb{R}[\boldsymbol{x}]$ be such that $p(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$. Then, p is a sum of squares of rational functions.*

Proof. Let $K = \mathbb{R}(x_1, \dots, x_n)$ be the field of rational functions. Suppose $p \notin \Sigma K^2$. Then

$$P' := \Sigma K^2[-p] = \left\{f - pg : f, g \in \Sigma K^2\right\}$$

is a proper cone of K. Therefore, it defines a partial order via

$$f \preceq g :\iff g - f \in P'.$$

By the Szpilrajn extension theorem [Mar30], every partial order can be extended to a total order, so let $\leq$ be such an extended total order on K. Let R be the real closure of $(K, \leq)$. Since $-p \in P'$, we thus have $-p > 0$ (both in K and in R), so there is some $z \in R$ such that $-p = z^2$. Consider the following statement in FO:

$$\varphi := \exists x_1 \dots \exists x_n \,.\, \exists z \,.\, p(x_1, \dots, x_n) + z^2 = 0 \land z \neq 0.$$

We know that φ holds over R, but it also is a statement over $\mathbb{R}$. By Theorem 2.5.5 we have

$$(\mathbb{R}, +, \cdot, c_a \mapsto a : a \in \mathbb{R}) \models \exists x_1 \dots \exists x_n \,.\, \exists z \,.\, p(x_1, \dots, x_n) + z^2 = 0 \land z \neq 0.$$

Therefore, we have some elements $\overline{x}_1, \dots, \overline{x}_n \in \mathbb{R}$ with $p(\overline{x}_1, \dots, \overline{x}_n) < 0$ which is a contradiction. $\square$

2.6 Sums of Nonnegative Circuit Polynomials

In this section, we introduce Nonnegative Circuit Polynomials and the cone they generate. Circuit polynomials were first introduced in [IdW16a]. They form a class of sparse polynomials whose nonnegativity can easily be verified. From these we build the cone sums of nonnegative circuit polynomials (SONC), which yields another type of certificate, that some polynomial is nonnegative.
In matroid theory, a minimal dependent set is called a circuit, see, e.g. [Oxl11]. In our context, a circuit is a minimal affinely dependent subset of $\mathbb{R}^n$. Circuit polynomials got their name, because their support is a circuit.

2.6.1 Definition. A *circuit polynomial* $p \in \mathbb{R}[x_1, \ldots, x_n]$ is a polynomial of the form

$$p(\boldsymbol{x}) = \sum_{j=0}^{r} b_{\boldsymbol{\alpha}(j)} \boldsymbol{x}^{\boldsymbol{\alpha}(j)} + b_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}}, \tag{2.1}$$

with $0 \leq r \leq n$, where for all j we have coefficients $b_{\boldsymbol{\alpha}(j)} \in \mathbb{R}_{>0}$, $b_{\boldsymbol{\beta}} \in \mathbb{R} \setminus \{0\}$, and exponents $\boldsymbol{\alpha}(j) \in (2\mathbb{N})^n$, $\boldsymbol{\beta} \in \mathbb{N}^n$, such that their Newton polytope is an r-dimensional simplex and $\boldsymbol{\beta} \in \operatorname{int}(\mathsf{New}(p))$, meaning that $\boldsymbol{\beta}$ lies in the relative interior of the Newton polytope. Therefore, we call $b_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}}$ the *inner term* of p. ⬡

These conditions imply that the support $A(p) = \{\boldsymbol{\alpha}(j), \boldsymbol{\beta} : j = 0, \ldots, r\}$ is a circuit. Furthermore, there exist unique, positive *barycentric coordinates* λ_j for $\boldsymbol{\beta}$ relative to the $\boldsymbol{\alpha}(j)$ with $j = 0, \ldots, r$ satisfying

$$\boldsymbol{\beta} = \sum_{j=0}^{r} \lambda_j \boldsymbol{\alpha}(j) \text{ and } \sum_{j=0}^{r} \lambda_j = 1. \tag{2.2}$$

In other words, $\boldsymbol{\lambda}$ describes the convex combination of $\boldsymbol{\beta}$ via the vectors $\boldsymbol{\alpha}(j)$. It is unique, because the support is a circuit and since $\boldsymbol{\beta}$ lies in the *strict* interior of the Newton polytope, we even have $\lambda_j > 0$ for all $j = 0, \ldots, r$.
For every circuit polynomial p we define the corresponding circuit number as

$$\Theta_p = \prod_{j=0}^{r} \left(\frac{b_{\boldsymbol{\alpha}(j)}}{\lambda_j} \right)^{\lambda_j}. \tag{2.3}$$

From our previous observations we know that the circuit number is well-defined.
The primary importance of the circuit number is that we can use it to determine whether some circuit polynomial is nonnegative. Thus, circuit polynomials are proper building blocks for nonnegativity certificates.

2.6.2 Theorem ([IdW16a], Theorem 1.1). *Let p be a circuit polynomial as in Definition 2.6.1. Then p is nonnegative if and only if:*

1. *p is a sum of monomial squares, or*

2. *the coefficient $b_{\boldsymbol{\beta}}$ of the inner term of p satisfies $|b_{\boldsymbol{\beta}}| \leq \Theta_p$.*

Furthermore, p has a root if and only if it is not a sum of monomial squares and $|b_{\boldsymbol{\beta}}| = \Theta_p$.

To illustrate these definitions and the theorem, we regard once more the Motzkin polynomial (see Example 2.5.2).

2.6.3 Example. Recall, that the Motzkin polynomial is $p = x^4y^2 + x^2y^4 + 1 - 3x^2y^2$. Since the dimension is $n = 2$, we can graphically illustrate the support as in figure 2.1.

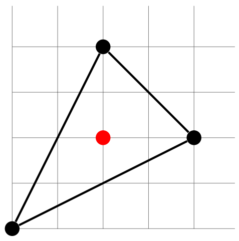

Figure 2.1: Black nodes mark the monomial squares, red the exponent $\boldsymbol{\beta}$.

As we can see, the Newton polytope is a triangle, and the last exponent lies in its strict interior. More explicitly, the exponents are $\boldsymbol{\alpha}(0) = (0, 0), \boldsymbol{\alpha}(1) = (2, 4), \boldsymbol{\alpha}(2) = (4, 2)$ and $\boldsymbol{\beta} = (2, 2)$. Now we can solve the linear equation system

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 4 & 2 \\ 0 & 2 & 4 \end{pmatrix} \boldsymbol{\lambda} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} \qquad \begin{array}{c} \text{all } 1 \\ \text{exponents of } x \\ \text{exponents of } y \end{array}$$

and get the solution $\boldsymbol{\lambda} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)^T$. So the circuit number is

$$\Theta_p = 3^{\frac{1}{3}} \cdot 3^{\frac{1}{3}} \cdot 3^{\frac{1}{3}} = 3.$$

Since we have $|b_{\boldsymbol{\beta}}| = 3 = \Theta_p$, we conclude from Theorem 2.6.2 that the Motzkin polynomial is nonnegative. ⬡

Furthermore, we can explicitly compute the minimum of a circuit polynomial. Generalising [IdW16a, Proposition 3.4], Müller [Mül18] provides the following theorem. For the reader's convenienve, we also present her proof here.

2.6.4 Theorem. *For a circuit polynomial p with constant term $\boldsymbol{\alpha}(0) = \mathbf{0}$ and $b_{\boldsymbol{\beta}} < 0$, let $\boldsymbol{s}$ be the vector satisfying the linear equation system*

$$\langle \boldsymbol{s}, \boldsymbol{\alpha}(j) - \boldsymbol{\beta} \rangle = \ln \left(-\frac{\lambda_j}{b_{\boldsymbol{\alpha}(j)}} \cdot b_{\boldsymbol{\beta}} \right) \qquad \textit{for all } 1 \leq j \leq n. \tag{2.4}$$

Then, $e^{\boldsymbol{s}}$ is the global minimiser of p.

Proof. Condition (2.4) implies $e^{\langle \boldsymbol{s}, \boldsymbol{\alpha}(j) \rangle} = -\frac{\lambda_j}{b_{\boldsymbol{\alpha}(j)}} b_{\boldsymbol{\beta}} \cdot e^{\langle \boldsymbol{s}, \boldsymbol{\beta} \rangle}$. Evaluating the partial derivative at $e^{\boldsymbol{s}}$ yields

$$\begin{aligned} \left(x_j \frac{\partial p}{\partial x_j} \right) (e^{\boldsymbol{s}}) &= \sum_{k=1}^{n} b_{\boldsymbol{\alpha}(k)} \boldsymbol{\alpha}(k)_j e^{\langle \boldsymbol{s}, \boldsymbol{\alpha}(k) \rangle} + b_{\boldsymbol{\beta}} \beta_j e^{\langle \boldsymbol{s}, \boldsymbol{\beta} \rangle} = -\sum_{k=1}^{n} \lambda_k b_{\boldsymbol{\beta}} \boldsymbol{\alpha}(k)_j e^{\langle \boldsymbol{s}, \boldsymbol{\beta} \rangle} + b_{\boldsymbol{\beta}} \beta_j e^{\langle \boldsymbol{s}, \boldsymbol{\beta} \rangle} \\ &= b_{\boldsymbol{\beta}} e^{\langle \boldsymbol{s}, \boldsymbol{\beta} \rangle} \left(-\sum_{k=1}^{n} \lambda_k \boldsymbol{\alpha}(k)_j + \beta_j \right) = 0. \end{aligned}$$

Note that the 0-th summand vanishes, since $\boldsymbol{\alpha}(0) = \mathbf{0}$ and the final sum vanishes, because

$$\boldsymbol{\beta} = \sum_{k=0}^{n} \lambda_k \boldsymbol{\alpha}(k) = \sum_{k=1}^{n} \lambda_k \boldsymbol{\alpha}(k).$$

Hence, $e^{\boldsymbol{s}}$ is a local minimiser of p. By [IdW16a, Proposition 3.3], it is the unique minimum in the positive orthant and since $b_{\boldsymbol{\beta}} < 0$, the global minimum is attained in the positive orthant. □

Since $\boldsymbol{\beta}$ lies in the interior of the Newton polytope, the vectors $\boldsymbol{\beta}, \boldsymbol{\alpha}(1), \ldots, \boldsymbol{\alpha}(n)$ span a simplex as well. Hence, the vectors $\boldsymbol{\alpha}(j) - \boldsymbol{\beta}$ are linearly independent. Therefore, (2.4) has a unique solution, so it is justified to speak of *the* solution $\boldsymbol{s}$.
For $b_{\boldsymbol{\beta}} > 0$, we have two cases. If the inner term is a monomial square, then the minimum of the circuit polynomial is $p(\mathbf{0})$. Otherwise, some entry β_i is odd. Then we put

$$\overline{p} = p(x_1, \ldots, x_{i-1}, -x_i, x_{i+1} \ldots, x_n). \tag{2.5}$$

This polynomial $\overline{p}$ has the same minimum as p and now satisfies the conditions of Theorem 2.6.4. So we can easily compute a minimiser for p.

2.6.1 The SONC Cone

We now naturally get the set of sums of nonnegative circuit polynomials (SONC). If we multipliy a nonnegative circuit polynomial with a positive scalar, we get another nonnegative circuit polynomial, so

$$\begin{aligned} \mathsf{SONC} :&= \left\{ \sum_{\text{finite}} q_i : q_i \text{ is a nonnegative circuit polynomial} \right\} \\ &= \left\{ \sum_{\text{finite}} c_i q_i : c_i > 0, q_i \text{ is a nonnegative circuit polynomial} \right\}. \end{aligned} \tag{2.6}$$

Observation. *The fundamental observation for our approach is*

If p is a SONC, then p is nonnegative.

However, in such a decomposition, every monomial non-square is treated as a possibly negative term, whose weight has to be cancelled out by monomial squares. This essentially corresponds to the relaxation

$$\overline{p} := \sum_{\boldsymbol{\alpha} \in \text{MoSq}(p)} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} - \sum_{\boldsymbol{\alpha} \in \text{NoSq}(p)} |b_{\boldsymbol{\alpha}}| \boldsymbol{x}^{\boldsymbol{\alpha}}. \tag{sign-relaxation}$$

which generalises (2.5). This also allows us to restrict ourselves to the positive orthant. We investigate this relaxation in more detail in Section 4.3.
The notation in (2.6) shows that SONC is a cone. Furthermore, it is a full-dimensional subset of the cone of nonnegative polynomials [DIdW17, Theorem 4.3]. Regarding only dimension n and degree d, this criterion is independent of the SOS-cone [IdW16a, Proposition 7.2], unless we have one of the trivial cases $n = 1$, $d = 2$ or $(n, d) = (2, 4)$. However, for special Newton polytopes, we have the additional relation.

2.6.5 Lemma ([IdW16a, Example 2.3, Corollary 5.3]). *If* $\mathsf{New}(p)$ *is the standard simplex and* p *is a nonnegative circuit polynomial, then* $p \in \Sigma^{\mathsf{SOS}}$.

Similar to SOS, there is also a converging hierarchy that can certify nonnegativity under constraints. For further details about SONC see [IdW16a, DIdW17].
Unfortunately, the complexity of finding such a SONC decomposition in exact arithmetic is unknown. On the positive side, we know that for any SONC polynomial, there is a SONC decomposition using the same support, as shown by Wang [Wan18, Theorem 5.5]. Furthermore, "short" certificates exist, as was observed by Papp [Pap19], whose idea we present here. We make use of a variant of Carathéodory's Theorem.

2.6.6 Theorem ([Roc70, Corollary 17.1.2]). *Let* $\{C_i : i \in I\}$ *for some index set* I *be an arbitrary collection of non-empty convex sets in* $\mathbb{R}^n$*, and let* K *be the convex cone generated by the union of the collection. Then every vector of* K *can be expressed as a non-negative linear combination of* n *or fewer linearly independent vectors, each belonging to a different* C_i.

Now we can make the notion of "shortness" more precise.

2.6.7 Lemma. *Let* $p \in \mathbb{R}[\boldsymbol{x}]$ *be a SONC. Then there is a SONC decomposition consisting of at most* $|A(p)|$ *circuit polynomials.*

Proof. We fix an order on the exponents and enumerate them as $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_t$. Then every circuit polynomial is identified by its coefficient vector in $\mathbb{R}^t$. For each circuit C, we define the set of coefficients that correspond to nonnegative circuit polynomials as

$$\mathsf{NC}(C) = \left\{ \boldsymbol{b} \in \mathbb{R}^t : \sum_{i \in C} b_i \boldsymbol{x}^{\boldsymbol{\alpha}_i} \geq 0, \forall i \notin C \,.\, b_i = 0 \right\}.$$

These sets are convex (as every positive scalar and every sum of nonnegative polynomials is nonnegative), so we can apply Theorem 2.6.6. The cone generated by their union is the SONC cone, so every SONC polynomial can be written as a sum of at most $t = |A(p)|$ nonnegative circuit polynomials. □

But even with the given support and a single negative term there can be an exponential number of candidates for the circuits.

2.6.8 Example. We fix the degree $d = 2n$ and put

$$A = \{0, d\} \times \Delta_d^{n-1} \qquad \text{and} \qquad \boldsymbol{\beta} = \left(\frac{d}{2}, 1, \ldots, 1\right).$$

So A consists of two copies of the scaled standard simplex. For each dimension $i \in \{2, \ldots, n\}$, we either pick vector $(d \cdot \boldsymbol{e}_i, 0)$ or $(d \cdot \boldsymbol{e}_i, d)$ and additionally $(\boldsymbol{0}, 0)$, $(\boldsymbol{0}, d)$ and $\boldsymbol{\beta}$. Now we have to show that each of these sets is a circuit with $\boldsymbol{\beta}$ in its interior. Regard the corresponding linear equation system

$$\begin{pmatrix} 1 & 1 & & \dots & 1 \\ 0 & d & c_2 & \dots & c_n \\ 0 & 0 & d & 0 & 0 \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & & 0 & d \end{pmatrix} \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \\ \lambda_n \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{d}{2} \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Immediately, we get $\lambda_i = \frac{1}{d}$ for $i \in \{2, \dots, n\}$. For the remaining entries, we have

$$\lambda_1 = \frac{1}{d}\left(\frac{d}{2} - \sum_{i=2}^{n} c_i \lambda_i\right)$$

for which we get the bounds

$$\frac{1}{2} = \frac{1}{d}\left(\frac{d}{2} - \sum_{i=2}^{n} 0 \cdot \lambda_i\right) \geq \lambda_1 \geq \frac{1}{d}\left(\frac{d}{2} - \sum_{i=2}^{n} d \cdot \lambda_i\right) = \frac{1}{2} - \frac{n-1}{2n} = \frac{1}{2n},$$

and finally, $\lambda_0 \geq 1 - \frac{1}{2} - \frac{n-1}{2n} = \frac{1}{2n}$. Therefore, for every choice of c_i, we have a solution $\boldsymbol{\lambda} > 0$, which shows that $\boldsymbol{\beta}$ is a strict convex combination of the chosen points. Additionally, all columns in the matrix are affinely independent, so our chosen set forms indeed a circuit with $\boldsymbol{\beta}$ in its interior.
Since there are 2^{n-1} possible choices for the c_i, we have a polynomial with $2n + 1$ terms, but the negative term is contained in 2^{n-1} circuits. ○

Having an exponential number of candidates of which we only need a small number gives rise to the following conjecture.

2.6.9 Conjecture. *Deciding SONC lies in* NP.

However, there are two open problems towards proving this conjecture. The first one lies in the bit size of the coefficients that are required in the decomposition. Second, even for a single circuit polynomial it is unknown whether we can check nonnegativity in polynomial time. We can compute the barycentric coordinates λ_j in polynomial time, but the circuit number involves exponents whose bit size is polynomial in the input size. Hence, checking for the conditions of Theorem 2.6.2 might actually be hard.
Note that this conjecture regards the complexity for solving the problem exactly. Some approaches, we present later, run in polynomial time, but these are numerical approaches that only solve the problem up to some accuracy.

2.7 Sums of Arithmetic-Geometric-Mean Exponentials

In [CS16] Chandrasekaran and Shah introduce another certificate for nonnegativity, based on "sums of arithmetic-geometric-mean exponentials" (SAGE). They also form a class of sparse polynomials whose nonnegativity can also easily be verified. Both computing this certificate and decomposing a polynomial into a SAGE (if possible) can

be done numerically by a relative entropy program. Like for SONC, the support of the certificate of nonnegativity is exactly the support of the input polynomial, which also makes this approach well-suited to obtain lower bounds for sparse polynomials. In fact, both approaches describe the same set of polynomials [MCW21].
A signomial is an expression of the form

$$p(\boldsymbol{y}) = \sum_{j=1}^{t} b_j \cdot \exp\left(\langle \boldsymbol{\alpha}(j), \boldsymbol{y}\rangle\right)$$

with $b_j \in \mathbb{R}$ and $\boldsymbol{\alpha}(j) \in \mathbb{N}^n$. Via logarithmic transformation $\boldsymbol{y} = \log(\boldsymbol{x})$, signomials correspond to polynomials, whose domain is restricted to $\mathbb{R}^n_+$.
An *arithmetic-geometric-mean-exponential (AGE)* is a nonnegative signomial with at most one negative coefficient. The name comes from the fact that its nonnegativity can be verified via arithmetic-geometric-mean inequality. The *sums of AGE polynomials (SAGE)* form a convex cone. Testing membership in this cone can be done by solving a relative entropy program (REP), which is a type of convex optimisation problem.
The *relative entropy* function is defined for $\boldsymbol{\lambda}, \boldsymbol{v} \in \mathbb{R}^t_+$ by $D(\boldsymbol{\lambda}, \boldsymbol{v}) := \sum_{j=1}^{t} \lambda_j \log \frac{\lambda_j}{v_j}$. Recall, that $\boldsymbol{v}_{\setminus i} \in \mathbb{R}^{n-1}$ denotes the vector derived from $\boldsymbol{v} \in \mathbb{R}^n$, where the entry at index i was removed and for a matrix $\boldsymbol{X}$ we denote the i-th row by $X^{(i)}$. To stay consistent with previous papers about SAGE, we may also use $\boldsymbol{\lambda}$ to denote a matrix instead of a vector. Then from [CS16, Proposition 2.4], we have the following characterisation.

2.7.1 Theorem. *A signomial $p(\boldsymbol{y}) = \sum_{j=1}^{t} b_j \exp\left(\langle \boldsymbol{\alpha}(j), \boldsymbol{y}\rangle\right)$ lies in SAGE if and only if there are $\boldsymbol{X}$, $\boldsymbol{\lambda} \in \mathbb{R}^{t\times t}$ satisfying the following conditions:*

$$\begin{aligned} &\sum_{i=1}^{t} \boldsymbol{X}^{(i)} = \boldsymbol{b}\,, \quad \sum_{j=1}^{t} \boldsymbol{\alpha}(j)\boldsymbol{\lambda}^{(i)}_j = \boldsymbol{0}\,, \quad -\boldsymbol{1}\cdot \boldsymbol{\lambda}^{(i)}_{\setminus i} = \lambda^{(i)}_i\,, \\ &\boldsymbol{X}^{(i)}_{\setminus i}, \boldsymbol{\lambda}^{(i)}_{\setminus i} \geq \boldsymbol{0}\,, \quad D\left(\boldsymbol{\lambda}^{(i)}_{\setminus i}, e\boldsymbol{X}^{(i)}_{\setminus i}\right) \leq X^{(i)}_i\,, \quad i = 1,\ldots,t\,. \end{aligned} \tag{SAGE-feas}$$

One way to obtain lower bounds of a signomial p is to solve the following REP:

$$p_{\mathrm{SAGE}} := \sup\{C \in \mathbb{R} : p - C \text{ is SAGE}\}. \tag{SAGE}$$

The constraints of (SAGE) correspond to (SAGE-feas), after replacing b_0 by $b_0 - C$.
The second block of constraints

$$\sum_{j=1}^{t} \boldsymbol{\alpha}(j)\boldsymbol{\lambda}^{(i)}_j = \boldsymbol{0} \qquad i = 1,\ldots,t$$

is in dimension n, butt by restricting to the ambient space, we may assume $n \leq t$. The sizes of the other constraints only depends on t. So the overall size of both the decision and the optimisation problem lies in $\mathcal{O}(t^2)$. Most notably, it is independent of the degree d. Recall, that we investigate sparse polynomials, which means $t \ll \binom{n+d}{d}$. So for some given signomial, we can decide in polynomial time, whether it is SAGE, see Section 2.4.3.
With the aforementioned logarithmic transformation, we can also use SAGE to certify nonnegativity of a polynomial $p(\boldsymbol{x})$ over $\mathbb{R}^n_+$, which means nonnegativity over $\mathbb{R}^n_{\geq 0}$. Using the relaxation (sign-relaxation) and showing $\overline{p} \in \mathsf{SAGE}$ yields global nonnegativity of p.
Since the cones for SONC and SAGE are the same, this also is a polynomial time method to determine, whether some polynomial is SONC (up to numerical errors). But it is yet unknown, how to obtain a SONC decomposition from the solution of the REP.

2.8 Exact Methods for Nonnegativity

In this section, we give a short overview on methods to exactly decide nonnegativity of multivariate polynomials.
The earliest method is based on Hilbert's 17th problem, that every nonnegative polynomial is a sum of squares of rational functions, see Theorem 2.5.6. In this form, however, the criterion is not suited for an algorithmic approach, as it allows for irrational coefficients in the decomposition. So we do not even have a discrete search space. But Artin, in his original paper actually provides a stronger statement.

2.8.1 Theorem ([Art27, Satz 6]). *Let R be a real number field. Then every nonnegative rational function $F \in R[\boldsymbol{x}]$ can be written as*

$$F = \sum_{\nu} w_\nu \cdot (\varphi_\nu(\boldsymbol{x}))^2$$

for nonnegative weights $w_\nu \in R$ and rational functions $\varphi_\nu \in R[\boldsymbol{x}]$.

This in particular holds for $R = \mathbb{Q}$. Hence, for every polynomial with rational coefficients, there is an SOS certificate with rational numbers. This shows, that the problem is semi-decidable. But as initially remarked, nonnegativity over $\mathbb{R}$ and over $\mathbb{Q}$ is equivalent. So the complement of NonNeg is semi-decidable as well, since we can enumerate all possible arguments $\boldsymbol{x} \in \mathbb{Q}^n$. Therefore, NonNeg is decidable. However, Artin's proof is not constructive, nor does it give a bound on the size of the certificate. Several improvements in the algorithmic approach have been made since then, e.g. by Delzell [Del84] and Reznick [Rez95]. But the best known bound on the number of squares still is 2^n as shown by Pfister [Pfi67]. Hence, there is no known algorithm that finds such a decomposition in less then exponential space.
Another early algorithm to decide NonNeg is Tarski's quantifier elimination, which we already mentioned. While it shows decidability of the problem more explicitly, it has nonelementary running time. So this method is practically infeasible. A significant improvement came in 1975 with the idea of cylindrical algebraic decomposition by Collins [Col75]. This approach solves the problem of quantifier elimination in time doubly exponential in the number of variables. Unfortunately, this behaviour is reached in the average and there are examples where even the output complexity is doubly exponential in n. But as we are only interested in the special case, where all quantifiers are existential quantifiers, we can use Renegar's approach for the Existential Theory of the Reals, which can solve the problem in polynomial space and running time $d^{\mathcal{O}(n)}$, see Section 2.3.3. As the problem NonNeg is coNP-hard, it is unlikely that any algorithm with sub-exponential time will be found.

Chapter 3

Complexity Results

In this section, we investigate the complexity of the problem related to unconstrained polynomial optimisation. Proofs, that the optimisation problem is NP-hard are easily found in the literature, e.g. [BR93, Lau09, Nes00]. However, showing that NonNeg(p) is a coNP-hard problem is a different question, since these problems are not equivalent, as mentioned in Section 2.1. One way to show the hardness of the decision problem is based on work by Murty and Kabadi [MK87], but this paper does not explicitly mention nonnegativity of polynomials. But their proof is quite technical, so we provide another proof, which is close to the proof for the optimisation problem.
For the reader's convenience, we provide a hardness proof of the optimisation problem, as given by Laurent [Lau09]. Additionally, the hardness proof already presents some ideas, how to show the hardness of nonnegativity. We reduce from the Subset-Sum problem, which is known to be NP-complete [GJ79].

Subset-Sum SSS($w_1, \dots, w_n, t$)
Input: vector $\boldsymbol{w} \in \mathbb{N}^n$, $t \in \mathbb{N}$
Question: Is there some $\boldsymbol{x} \in \{0,1\}^n$ such that $\sum_{i=1}^n x_i w_i = t$?

3.1 Hardness of Optimisation

3.1.1 Theorem. *The optimisation problem* MinPoly(p) *is NP-hard, even for degree 4.*

Proof. Let $(\boldsymbol{w}, t)$ be an instance of Subset-Sum. Define the polynomial

$$p := \left(t - \sum_{i=1}^n x_i w_i\right)^2 + \sum_{i=1}^n (x_i(x_i - 1))^2$$

Note that this polynomial has degree 4 and is a sum of squares. So, clearly, all function values are nonnegative.
If $(\boldsymbol{w}, t) \in$ Subset-Sum, then there is a certificate $\boldsymbol{x} \in \{0,1\}^n$ for this containment. By definition of Subset-Sum, we have $p(\boldsymbol{x}) = 0$, so the minimum of p is zero.
For the converse, assume $p(\boldsymbol{x}) = 0$ for some $\boldsymbol{x} \in \mathbb{R}^n$. Then every single square must be zero. On the one hand, this means $\sum_{i=1}^n x_i w_i = t$ and on the other hand $x_i \in \{0,1\}$ for

all i. Hence, $\boldsymbol{x}$ is a certificate for $(\boldsymbol{w}, t) \in$ SUBSET-SUM.
Therefore MINPOLY$(p) = 0$ if and only if $(\boldsymbol{w}, t) \in$ SUBSET-SUM. So the problem MINPOLY is NP-hard, even for degree 4. □

3.2 Hardness of Non-Negativity

For the optimisation problem, we constructed a sum of squares, so our polynomial is nonnegative by construction. The question is, whether we attain this minimum. In contrast, for the decision problem, we have to alter our construction in such a way, that the expression becomes negative if and only if the instance of SUBSET-SUM has a solution.

3.2.1 Theorem. *The decision problem* NONNEG *is* coNP*-hard, even for degree $d = 6$.*

We show this by reduction from the complement of SUBSET-SUM. First, we introduce the auxiliary function $f(x) = -2x^3 + 3x^2$. It has the properties

$$f(x) = x \qquad\qquad f'(x) = 0 \qquad\qquad \text{for } x \in \{0, 1\}.$$

Let $w_* = \max\{w_i : i = 1, \ldots, n\}$ denote the maximal weight. For an instance $(\boldsymbol{w}, t) \in \mathbb{N}^{n+1}$ define the "punishment value"

$$N := 32 w_* \left(\frac{w_*}{4} + t + \sum_{i=1}^{n} w_i \right)$$

as weight for violating the 0/1-constraints and put

$$p := \left(t - \sum_{i=1}^{n} f(x_i) w_i \right)^2 + N \cdot \sum_{i=1}^{n} x_i^2 (x_i - 1)^2 - 1.$$

3.2.2 Lemma. *The polynomial p attains its optimum in the cube $\left[-\frac{1}{2}, \frac{3}{2}\right]^n$.*

Proof. First note that $p(\mathbf{0}) = t^2 - 1$. Now assume $\boldsymbol{x}$ lies outside this cube. Then $x_i^2 (x_i - 1)^2 \geq \frac{9}{16} > \frac{1}{2}$. Hence,

$$p(\boldsymbol{x}) > N \cdot n \cdot \frac{1}{2} - 1 \geq t^2 - 1 = p(\mathbf{0})$$

by choice of N. Thus, a point outside this cube cannot be optimal. □

3.2.3 Lemma. *The polynomial p attains its optimum at a vertex of the hypercube, i.e. at some $\boldsymbol{x} \in \{0, 1\}^n$.*

Proof. For $\alpha \in \mathbb{R}$ let $[\alpha]$ denote the rounding function. Extend this to vector as

$$[\boldsymbol{x}]_k = (x_1, \ldots, [x_k], \ldots, x_n),$$

where the k-th entry is rounded, and write $[\boldsymbol{x}]$ for rounding every entry of the vector.

Now, we show that rounding and entry of the minimiser of p does not increase the value. This means, we change some entry by ε with $|\varepsilon| \leq \frac{1}{2}$. Fix some $x_k \in \{0,1\}$ with $[x_k] = x_k + \varepsilon$. For short, we denote the difference in f by

$$\begin{aligned}
\delta &:= f(x_k+\varepsilon) - f(x_k) = -2(3x_k^2\varepsilon + 3x_k\varepsilon^2 + \varepsilon^3) + 3(2x_k\varepsilon + \varepsilon^2) \\
&= \begin{cases} -2\varepsilon^3 + 3\varepsilon^2 & : x_k = 0 \\ -2\varepsilon^3 - 3\varepsilon^2 & : x_k = 1 \end{cases} \\
&= \pm 3\varepsilon^2 - 2\varepsilon^3
\end{aligned}$$

Assume p attains its minimum at some point $\boldsymbol{x}$. By Lemma 3.2.2 we know $\boldsymbol{x} \in \left[-\frac{1}{2}, \frac{3}{2}\right]^n$. Therefore, we have $[x_k] \in \{0,1\}$ and thus $x_k \in \{\pm\varepsilon, 1 \pm \varepsilon\}$. So, in every case,

$$[x_k]^2\left([x_k]-1\right)^2 = 0 \quad \text{and} \quad x_k^2(x_k-1)^2 = \varepsilon^2(1\pm\varepsilon)^2.$$

For short, we further write $\Sigma := t - \sum_{i=1}^n w_i f(x_i)$. Then

$$\begin{aligned}
p(\boldsymbol{x}) - p\left([\boldsymbol{x}]_k\right) &= \Sigma^2 + N\sum_{i=1}^n x_i^2(x_i-1)^2 - 1 \\
&\quad - \left((\Sigma + \delta \cdot w_k)^2 + N \sum_{i \neq k} x_i^2(1-x_i)^2 + N\,[x_k]^2\left(1-[x_k]\right)^2 - 1 \right) \\
&= \Sigma^2 - (\Sigma + \delta\cdot w_k)^2 + N\varepsilon^2(1\pm\varepsilon)^2 \\
&= -\delta^2 w_k^2 - 2\delta w_k \Sigma + N\varepsilon^2(1\pm\varepsilon)^2
\end{aligned}$$

Now we assume the worst case, where every unknown sign is negative, but $\varepsilon \geq 0$. Using $\varepsilon \leq \frac{1}{2}$ we get $(1-\varepsilon)^2 \geq \frac{1}{4}$ and $\delta \leq \frac{1}{2}$. This yields the bound

$$\begin{aligned}
p(\boldsymbol{x}) - p\left([\boldsymbol{x}]_k\right) &\geq -\delta^2 w_k^2 - 2\delta w_k\Sigma + N\varepsilon^2(1-\varepsilon)^2 \\
&\geq -(3\varepsilon^2 + 2\varepsilon^3) w_k \left(\frac{w_k}{2} + 2\Sigma\right) + \frac{1}{4}N\varepsilon^2 \\
&\geq -8\varepsilon^2 w_k \left(\frac{w_k}{4} + \Sigma\right) + \frac{1}{4}N\varepsilon^2 \\
&\geq 0
\end{aligned}$$

by choice of N. Therefore by iteratively rounding each entry, we get $p(\boldsymbol{x}) \geq p([\boldsymbol{x}])$. Since we started with some $\boldsymbol{x} \in \left[-\frac{1}{2}, \frac{3}{2}\right]^n$, the rounded vector $[\boldsymbol{x}]$ must be a vertex of the hypercube. $\square$

Proof of Theorem 3.2.1. It remains to show that $(w,t) \in$ SUBSET-SUM iff $p \notin$ NONNEG. If $(\boldsymbol{w},t) \in$ SUBSET-SUM, take $\boldsymbol{x} \in \{0,1\}^n$ such that $t = \sum_{i=1}^n x_i w_i$. Then $p(\boldsymbol{x}) = -1$, so $p \notin$ NONNEG.
Conversely, assume $p \notin$ NONNEG. By Lemma 3.2.3, we may choose a minimiser $\boldsymbol{x} \in \{0,1\}^n$ of p. Since p has integer coefficients, we have $p(\boldsymbol{x}) \in \mathbb{Z}$. Furthermore, $p+1$ is a sum of squares, in particular, it is nonnegative. Therefore, we have $0 > p(\boldsymbol{x}) \geq -1$, which means $p(\boldsymbol{x}) = -1$. Hence, each of the squares must be zero. Thus, $\boldsymbol{x} \in \{0,1\}^n$ and

$$t = \sum_{i=1}^n f(x_i) w_i = \sum_{i=1}^n x_i w_i$$

which means $\boldsymbol{x}$ is a certificate, that $(\boldsymbol{w},t) \in$ SUBSET-SUM. $\square$

In parametrised complexity, this also proves the following stronger statement.

3.2.4 Corollary. *Unless* $\mathsf{P} = \mathsf{NP}$*, there is no* XP*-algorithm to decide nonnegativity when parametrised by the degree.*

In fact, from [MK87, Theorem 2], we can obtain a slightly stronger result.

3.2.5 Theorem. *Given a matrix* $\boldsymbol{D} \in \mathbb{R}^{n\times n}$*, deciding, whether there is some real vector* $\boldsymbol{x} \geq 0$*, such that* $\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{x} < 0$ *is* NP*-complete.*

In other words, deciding copositivity of a matrix is coNP-hard. From this theorem, we immediately obtain the following result about deciding nonnegativity.

3.2.6 Corollary. *Deciding nonnegativity for polynomials is* coNP*-hard, even for homogeneous polynomials of degree 4.*

Proof. For some matrix $\boldsymbol{D} \in \mathbb{R}^{n\times n}$, define the homogeneous polynomial

$$p(\boldsymbol{x}) = (x_1^2, \dots, x_n^2)^T \boldsymbol{D} (x_1^2, \dots, x_n^2),$$

which has degree 4. Squaring all variables corresponds to restricting p to the positive orthant. Thus, $p(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$ if and only if $\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \geq 0$. So, by Theorem 3.2.5, deciding nonnegativity is coNP-hard, even for homogeneous polynomials of degree 4. $\square$

3.3 Hardness of Deciding the Existence of a Lower Bound

Another important question in polynomial optimisation is whether the infimum of the image is finite, which means deciding whether a given polynomial is bounded.

3.3.1 Theorem. *The problem* Bounded *is* coNP*-hard.*

Proof. For some $p \in \mathbb{Q}[x_1, \dots, x_n]$ let $p_{\text{hom}} \in \mathbb{Q}[x_0, \dots, x_n]$ denote its homogenisation.
If $p \in$ NonNeg, then p_{hom} is bounded.
If $p \notin$ NonNeg, say $p(x) < 0$ for some $x \in \mathbb{Q}^n$, then

$$\lim_{t\to\infty} p_{\text{hom}}(t(1,x)) = \lim_{t\to\infty} t^{\deg p + 1} \cdot p_{\text{hom}}(1,x) = -\infty$$

so p_{hom} is unbounded.
Therefore NonNeg $\leq_p$ Bounded. By Theorem 3.2.1 this shows that Bounded is coNP-hard. $\square$

On the positive side, the problem is decidable, using Tarski's quantifier elimination. On the negative side, using the formulation

$$\exists c\,.\,\forall x_1 \dots \forall x_n\,.\,p(x_1, \dots, x_n) \geq c,$$

the problem cannot even be solved using the method for the Existential Theory for the Reals. The quantifier flip suggests, that Bounded might even be harder than NonNeg. But it may also be possible, that we can eliminate the universal quantifier with only a polynomial blow-up. To the best of our knowledge, it is an open problem, which of these is the case.

Chapter 4

Algorithms

In this section, we describe the algorithms, which are implemented in the software "Effective Methods for Polynomial Optimisation" (POEM). The first part is a description of our algorithm to compute a lower bound for a polynomial using SONC in polynomial time. To the best of our knowledge, this is the first polynomial-time algorithm to compute lower bounds for *sparse* polynomials. In the second part, we describe, how the SONC decomposition can be used to obtain a candidate for the global minimum. Thus, we have an upper and a lower bound for the minimum of our polynomial and often enough they turn out to leave a small optimality gap. Afterwards, we describe two methods, how the lower bound can be further improved by increasing the running time by a factor of at most 2^n, so the running time is FPT in the number of variables. Finally, we present a post-processing method, which, on a large class of cases, converts the numerical results into lower bounds in exact arithmetic.
For each of these algorithms, we ran experiments, which we also discuss and evaluate in Chapter 5.

4.1 SONC-Algorithm

Let $p = \text{poly}(\boldsymbol{A}, \boldsymbol{b})$ be some polynomial with $\boldsymbol{A} \in \mathbb{N}^{n\times t}$ and $\boldsymbol{b} \in \mathbb{R}^t$. In this section, we describe how we can use SONC to compute a lower bound for p in polynomial time. However, this is not necessarily the optimal bound, that can be obtained via SONC. Also, due to the hardness of the problem Bounded, the algorithm may return the bound $-\infty$, although p is bounded. But under the assumption $\mathsf{P} \neq \mathsf{NP}$, there is no polynomial time algorithm, that completely avoids the latter problem. This section is an improved version of the approach described in [SdW18].
Recall, that a circuit polynomial has the shape

$$q = \sum_{j=0}^{r} b_{\boldsymbol{\alpha}(j)} \boldsymbol{x}^{\boldsymbol{\alpha}(j)} + b_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}}$$

for some $r \leq n$, where the terms $b_{\boldsymbol{\alpha}(j)} \boldsymbol{x}^{\boldsymbol{\alpha}(j)}$ are monomial squares, i.e. $b_{\boldsymbol{\alpha}(j)} > 0$ and $\boldsymbol{\alpha}(j) \in (2\mathbb{N})^n$. Furthermore, $\boldsymbol{\beta}$ lies in the relative interior of the simplex spanned by the points $\boldsymbol{\alpha}(j)$. Intuitively, the positive weight of the terms $b_{\boldsymbol{\alpha}(j)} \boldsymbol{x}^{\boldsymbol{\alpha}(j)}$ cancels out the potentially negative weight of $b_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}}$.

To find a lower bound for p, we search for some $\gamma \in \mathbb{R}$ such that $p - \gamma$ is a SONC, i.e. we want to find a decomposition

$$p - \gamma = \sum_{\text{finite}} q_i$$

where each q_i is a nonnegative circuit polynomial. Thus, γ is a lower bound Our approach works in two major steps. First, we determine the circuits involved in the decomposition. Each exponent of a non-square must occur as the inner point of some circuit polynomial and it can even appear in multiple circuits. This set of circuits, we call a cover, to emphasise that these circuits may overlap (in contrast to a triangulation).
After we fixed the circuits, we determine the actual decomposition of the polynomial by computing the distribution of the coefficients. However, as we consider the inner term of every circuit polynomial to be potentially negative, we implicitly perform the relaxation

$$\overline{p} := \sum_{\boldsymbol{\alpha} \in \mathrm{MoSq}(p)} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} - \sum_{\boldsymbol{\alpha} \in \mathrm{NoSq}(p)} |b_{\boldsymbol{\alpha}}| \, \boldsymbol{x}^{\boldsymbol{\alpha}}, \tag{sign-relaxation}$$

as mentioned in Section 2.6.1. More formally, we have

4.1.1 Theorem. *Let p be some polynomial. Then the lower bounds via SONC for p and $\overline{p}$ are the same. Furthermore, the lower bound of $\overline{p}$ is its lower bound on the nonnegative orthant.*

Proof. Assume $p - \gamma$ is SONC. Put

$$\Delta(\boldsymbol{\beta}) = \{\Delta \subseteq \mathbb{R}^n : \mathsf{Vert}(\Delta) \subseteq \mathrm{MoSq}(p), \boldsymbol{\beta} \in \mathrm{int}(\Delta), \Delta \text{ is a simplex}\}.$$

Investigation of the proofs of [Wan18, Theorem 3.9, Theorem 5.5] shows that there is a SONC decomposition

$$p - \gamma = \sum_{\boldsymbol{\beta} \in \mathrm{NoSq}(p)} \left(\sum_{\Delta \in \Delta(\boldsymbol{\beta})} \left(\sum_{\boldsymbol{\alpha} \in \mathsf{Vert}(\Delta)} X_{\boldsymbol{\alpha},\boldsymbol{\beta},\Delta} \boldsymbol{x}^{\boldsymbol{\alpha}} + b_{\boldsymbol{\beta},\Delta} \boldsymbol{x}^{\boldsymbol{\beta}} \right) \right),$$

i.e. the positive/negative points of the circuit polynomials are positive/negative points of p respectively. Since we do not regard monomials squares for $\boldsymbol{\beta}$, we are always in the second case of Theorem 2.6.2. Therefore, switching the sign of $b_{\boldsymbol{\beta},\Delta}$ does not change the nonnegativity of the circuit polynomial. Thus,

$$\overline{p} - \gamma = \sum_{\boldsymbol{\beta} \in \mathrm{NoSq}(p)} \left(\sum_{\Delta \in \Delta(\boldsymbol{\beta})} \left(\sum_{\boldsymbol{\alpha} \in \mathsf{Vert}(\Delta)} X_{\boldsymbol{\alpha},\boldsymbol{\beta},\Delta} \boldsymbol{x}^{\boldsymbol{\alpha}} - |b_{\boldsymbol{\beta},\Delta}| \, \boldsymbol{x}^{\boldsymbol{\beta}} \right) \right)$$

is a SONC decomposition for $\overline{p} - \gamma$.
Now, every summand that could be negative, becomes negative in the positive orthant. Hence, the infimum of $\overline{p}$ is given by its infimum over the positive orthant. Moreover, if we define the elementwise absolute value

$$|\boldsymbol{x}| := (|x_1|, \ldots, |x_n|),$$

then $p(\boldsymbol{x}) \geq \overline{p}(|\boldsymbol{x}|)$ for all $\boldsymbol{x} \in \mathbb{R}^n$. In particular, the infimum cannot increase by this relaxation, i.e.

$$\inf \{p(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n\} \geq \inf \{\overline{p}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n\} = \inf \left\{\overline{p}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n_{\geq 0}\right\}. \tag{4.1}$$

So, the bound for $\overline{p}$ is also a lower bound for p. □

For this relaxed polynomial we compute a lower bound via SONC. Due to (4.1), this also yields a lower bound for p. The advantage is, that we can restrict ourselves to the positive orthant. For the remainder of this section, for simplicity, we assume $p = \overline{p}$.

4.1.1 Computing the Covering

As the first step to compute a lower bound for p, we determine the circuits, which we use in our decomposition into circuit polynomials. In the original meaning, a circuit is a minimal affinely dependent set of exponent vectors. Furthermore, the $\boldsymbol{\alpha}(i)$ must be the vertices of a simplex (possibly not full dimensional) and $\boldsymbol{\beta}$ lies in the relative interior of this simplex. So $\boldsymbol{\beta}$ can be written as a convex combination of the vectors $\boldsymbol{\alpha}(i)$. We impose the further condition that the circuits have the form $C = \{\boldsymbol{\alpha}(0), \ldots, \boldsymbol{\alpha}(m), \boldsymbol{\beta}\}$ for some $m \leq n$, where $\boldsymbol{\alpha}(i) \in \operatorname{MoSq}(p)$ and $\boldsymbol{\beta} \in \operatorname{NoSq}(p)$. This gives rise to the following linear programme for each $\boldsymbol{\beta} \in \operatorname{NoSq}(p)$.

$$\begin{aligned} \sum_{\boldsymbol{\alpha} \in \operatorname{MoSq}(p)} \lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} \cdot \boldsymbol{\alpha} &= \boldsymbol{\beta} \\ \sum_{\boldsymbol{\alpha} \in \operatorname{MoSq}(p)} \lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} &= 1 \\ \lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} &\geq 0 \qquad \text{for all } \boldsymbol{\alpha} \in \operatorname{MoSq}(p) \end{aligned} \tag{LP}$$

If we solve (LP) with the Simplex Algorithm, each set $\mathsf{C}^{\boldsymbol{\beta}} := \left\{\boldsymbol{\alpha} \in \operatorname{MoSq}(p) : \boldsymbol{\lambda}_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} > 0\right\}$ for $\boldsymbol{\beta} \in \operatorname{NoSq}(p)$ describes a simplex (since the solution is a vertex of the feasible set). However, the Simplex Algorithm may have exponential running time. To have a guaranteed polynomial running time, we can solve (LP) with some polynomial time algorithm, see [Kha79, Kar84]. Then some of the sets $\mathsf{C}^{\boldsymbol{\beta}}$ can be affinely dependent. So we need additional post-processing of the solution $\boldsymbol{\lambda}$. This can be done with the following lemma. The statement is folklore, but for the reader's convenience, we provide a short proof.

4.1.2 Lemma. *For every non-extremal point $\boldsymbol{v} \in A(p) \setminus \mathsf{Vert}(p)$, we can efficiently compute affinely independent $\boldsymbol{v}_0, ..., \boldsymbol{v}_m \in \mathsf{Vert}(p)$ with $m \leq n$ such that $\boldsymbol{v} \in \operatorname{conv}(\{\boldsymbol{v}_0, ..., \boldsymbol{v}_m\})$.*

Proof. Solve (LP) to obtain a convex combination $\boldsymbol{v} = \sum_{i=0}^m \lambda_i \boldsymbol{v}_i$. Assume $m > n$, and let μ be a non-trivial element in

$$\ker \begin{pmatrix} 1 & \cdots & 1 \\ \boldsymbol{v}_0 & \cdots & \boldsymbol{v}_m \end{pmatrix}. \tag{4.2}$$

Assign $\eta = \min\left\{\frac{\lambda_i}{\mu_i} : i \leq m, \mu_i > 0\right\}$ and choose coefficients $\lambda_i' = \lambda_i - \eta\mu_i$. Then $\boldsymbol{v} = \sum_{i=0}^m \lambda_i' \boldsymbol{v}_i$ is a new convex combination but one of the coefficients vanishes, so we obtain a shorter convex combination of $\boldsymbol{v}$. We delete the corresponding column in the matrix (4.2) and iterate the process until we obtain a matrix with trivial kernel. Once this is the case the vectors $\boldsymbol{v}_i$ are affinely independent. This can only occur if $m \leq n$. $\square$

So finally, we have our solution matrix $\boldsymbol{\lambda} \in \mathbb{R}^{\mathrm{NoSq}(p) \times \mathrm{MoSq}(p)}$ with corresponding cover

$$\mathsf{C} = \left\{ \mathsf{C}^{\boldsymbol{\beta}} \cup \{\boldsymbol{\beta}\} : \boldsymbol{\beta} \in \mathrm{NoSq}\,(p) \right\}.$$

4.1.3 Algorithm. If a covering exists, we can compute one in polynomial time (and thus also in polynomial size) via the following algorithm. More precisely, the unit cost size of the covering is in $\mathcal{O}(tn)$.

Input: Polynomial p

Output: $(\mathsf{C}, \boldsymbol{\lambda})$: set of circuits and corresponding convex combinations

```
1: function COVER (p)
2:     for $\boldsymbol{\beta} \in \mathrm{NoSq}\,(p)$ do
3:         $\boldsymbol{\lambda}^{\boldsymbol{\beta}} \leftarrow$ feasible solution of (LP)
4:         $\mathsf{C}^{\boldsymbol{\beta}} \leftarrow \left\{ \boldsymbol{\alpha} \in \mathrm{MoSq}\,(p) : \lambda^{\boldsymbol{\beta}}_{\boldsymbol{\alpha}} > 0 \right\}$
5:         while $\mathsf{C}^{\boldsymbol{\beta}}$ is affinely dependent do
6:             apply Lemma 4.1.2
7:         append $C = \mathsf{C}^{\boldsymbol{\beta}} \cup \{\boldsymbol{\beta}\}$ to $\mathsf{C}$
8:     return $(\boldsymbol{\lambda}, \mathsf{C})$
```

Proof. To analyse the running time, first observe, that the for-loop has at most t iterations. Solving the LP in Line 3 runs in polynomial time. Finally, each iteration of Line 5 reduces the size of $\mathsf{C}^{\boldsymbol{\beta}}$ by one, so after at most t iterations, we are done. To apply Lemma 4.1.2, we solve a linear equation system. So all these operations together run in polynomial time.

If we write $\boldsymbol{\lambda}$ as a sparse matrix, and store the indices of the vectors (instead of the vectors themselves), then the cover has size

$$\|(\boldsymbol{\lambda}, \mathsf{C})\| \in \mathcal{O}\left(n \cdot |\,\mathrm{NoSq}\,(p)\,|\right) \subseteq \mathcal{O}(tn)$$

in the unit cost model. If we consider non-unit cost, then the entries of C have size at most $\log t$. The entries of $\boldsymbol{\lambda}$ are solutions of linear equation systems. By Cramer's rule, their enumerators and denominators are bounded by $n! \cdot d^n$. This yields an encoding size of

$$\|(\boldsymbol{\lambda}, \mathsf{C})\| \in \mathcal{O}\left(n \cdot |\,\mathrm{NoSq}\,(p)\,| \cdot (\log t + \log(n! \cdot d^n))\right)) \subseteq \mathcal{O}\left(nt(\log t + n \log d + n \log n)\right)$$

which is polynomial in $(n, t, \log d)$, which are the parameters of the input size.

The only step that might fail is Line 3, where we solve (LP). But if that step fails, then some $\boldsymbol{\beta} \in \mathrm{NoSq}\,(p)$ cannot be written as convex combination of monomial squares. Hence, this $\boldsymbol{\beta}$ (or another non-square) must be an extremal point. This implies both that the polynomial is unbounded due to Lemma 2.1.4 and that there is no cover. $\square$

4.1.2 Improving the Covering

While the covering, computed in Section 4.1.1, allows to compute a lower bound for p, the relaxation can be significantly strengthened. So far, not every positive point must occur in a circuit. As a result, we actually compute a lower bound for another polynomial. Let

$$A' = \bigcup_{\boldsymbol{\beta} \in \mathrm{NoSq}(p)} \mathsf{C}^{\boldsymbol{\beta}} \cup \{\boldsymbol{\beta}\} \qquad \text{and} \qquad p' = \sum_{\boldsymbol{\alpha} \in A'} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}}.$$

Then $p - p'$ is a sum of monomial squares, so $p' \leq p$. But our computations would effectively continue with p', thus possibly yielding a weaker lower bound for p.
To solve this problem, we keep track, which positive points we used and introduce an objective function for (LP), which we adjust accordingly. We initialise the objective with $\boldsymbol{w} = \mathbf{1} \in \mathbb{R}^{\mathrm{MoSq}(p)}$. Then our LP becomes

$$\begin{aligned} \underset{\boldsymbol{\lambda}^{\boldsymbol{\beta}} \in \mathbb{R}^{\mathrm{MoSq}(p)}}{\text{maximise}} \quad & \sum_{\boldsymbol{\alpha} \in \mathrm{MoSq}(p)} w_{\boldsymbol{\alpha}} \boldsymbol{\lambda}^{\boldsymbol{\beta}}_{\boldsymbol{\alpha}} \\ \text{subject to} \quad & \sum_{\boldsymbol{\alpha} \in \mathrm{MoSq}(p)} \lambda^{\boldsymbol{\beta}}_{\boldsymbol{\alpha}} \cdot \boldsymbol{\alpha} = \boldsymbol{\beta} \\ & \sum_{\boldsymbol{\alpha} \in \mathrm{MoSq}(p)} \lambda^{\boldsymbol{\beta}}_{\boldsymbol{\alpha}} = 1 \\ & \lambda^{\boldsymbol{\beta}}_{\boldsymbol{\alpha}} \geq 0 \qquad \text{for all } \boldsymbol{\alpha} \in \mathrm{MoSq}\,(p)\,. \end{aligned} \tag{LP-opt}$$

We determine our circuit C as before and update $w_{\boldsymbol{\alpha}} = 0$ for all $\boldsymbol{\alpha} \in C$. If there is an unused positive point, which can be used to cover the negative point $\boldsymbol{\beta}$, then (LP-opt) has a positive solution. So, the cover will afterwards contain an additional positive point. If we reach a state, where every negative point is covered, but there are still unused positive points, then we repeatedly iterate again over all $\boldsymbol{\beta} \in \mathrm{NoSq}\,(p)$ until $\boldsymbol{w} = \mathbf{0}$ or there is no positive solution for any negative point. If $\boldsymbol{w} = \mathbf{0}$, then every point of the support occurs in the covering. Otherwise, all non-squares lie in some lower dimensional subset of the Newton polytope. Then, the remaining points cannot be used in a covering. Without loss of generality, we assume to be in the first case.

4.1.4 Algorithm. We can, in polynomial time, compute a covering C, with $\bigcup_{C \in \mathsf{C}} C = A$ (or check that no such cover exists). The unit cost size of the covering is in $\mathcal{O}(tn)$.
Input: p -- polynomial
Output: $(\mathsf{C}, \boldsymbol{\lambda})$: set of circuits and corresponding convex combinations,

```
 1: function FULLCOVER (p)
 2:     $\boldsymbol{w} \leftarrow \mathbf{1} \in \mathbb{R}^{\mathrm{MoSq}(p)}$
 3:     uncovered $\leftarrow \mathrm{NoSq}\,(p)$
 4:     while $\boldsymbol{w} \neq \mathbf{0}$ or uncovered $\neq \emptyset$ do
 5:         $\boldsymbol{\beta} \leftarrow$ some element from uncovered (or from $\mathrm{NoSq}\,(p)$ if uncovered $= \emptyset$)
 6:         $\boldsymbol{\lambda}^{\boldsymbol{\beta}} \leftarrow$ optimal solution of (LP-opt)
 7:         $\mathsf{C}^{\boldsymbol{\beta}} \leftarrow \{\boldsymbol{\alpha} \in \mathrm{MoSq}\,(p) : \lambda^{\boldsymbol{\beta}}_{\boldsymbol{\alpha}} > 0\}$
 8:         while $\mathsf{C}^{\boldsymbol{\beta}}$ is affinely dependent do
 9:             apply Lemma 4.1.2
10:         uncovered $\leftarrow$ uncovered $\setminus \{\boldsymbol{\beta}\}$
11:         for $\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}$ do
12:             $w_{\boldsymbol{\alpha}} \leftarrow 0$
13:         append $\boldsymbol{\lambda}^{\boldsymbol{\beta}}$ to $\boldsymbol{\lambda}$ and $\mathsf{C}^{\boldsymbol{\beta}} \cup \{\boldsymbol{\beta}\}$ to $\mathsf{C}$
14:     return $(\boldsymbol{\lambda}, \mathsf{C})$
```

Proof. In every circuit, we cover an additional non-square $\boldsymbol{\beta}$, or use an additional monomial square $\boldsymbol{\alpha}$ and set $w_{\boldsymbol{\alpha}} = 0$. Therefore, we have at most $|\operatorname{MoSq}(p)| + |\operatorname{NoSq}(p)| = t$ circuits, each of size at most $n + 2$.
Once, we have uncovered $= \emptyset$, we might have to test all $\boldsymbol{\beta} \in \operatorname{NoSq}(p)$ to further improve $\boldsymbol{w}$. Hence, we may have to run Line 5 and the following computations $|\operatorname{NoSq}(p)| \leq t$ times. If then $\boldsymbol{w}$ was not changed, no such cover exists, and we may break. Otherwise, the computations from Line 5 to Line 13 are run at most t^2 times, and all of these computations run in polynomial time. $\square$

As a result of this extension, a non-square may occur in multiple circuit. Hence, we can no longer iterate over the non-squares. Instead, we iterate over the circuits in the covering and for each circuit C, we denote its (unique) non-square as $\boldsymbol{\beta}(C)$.
The next improvement is merely of computational nature. Whenever we found a circuit $\mathsf{C}^{\boldsymbol{\beta}}$, we can easily test, whether any other negative points lie in the interior of the described simplex. To do so, we solve $|\operatorname{NoSq}(p)|$ many linear equation systems of size n, which all have the same coefficient matrix. Using QR-factorisation, this runs in $\mathcal{O}(n^3 + n^2 \cdot |\operatorname{NoSq}(p)|)$. In contrast, (LP) is a linear programme of size $\mathcal{O}(n \cdot |\operatorname{MoSq}(p)|)$, which often takes longer to solve.

4.1.5 Example. Consider the polynomial

$$p = 2 + x^6 + y^4 + x^4y^4 - x^3y - xy^2 - x^4y^3.$$

For $\boldsymbol{\beta} = (4,3)^T$, we obtain the circuit

$$\begin{pmatrix} 0 & 6 & 4 & 4 \\ 0 & 0 & 4 & 3 \end{pmatrix},$$

but the corresponding simplex also contains $(3,1)^T$. Next, we still have to cover $\boldsymbol{\beta} = (1,2)^T$ and the monomial square $(0,4)^T$ is still unused. So we take the circuit

$$\begin{pmatrix} 0 & 6 & 0 & 1 \\ 0 & 0 & 4 & 2 \end{pmatrix}.$$

Again, the simplex also contains $(3,1)^T$, so we end up with 4 circuits. We can illustrate this decomposition as follows.

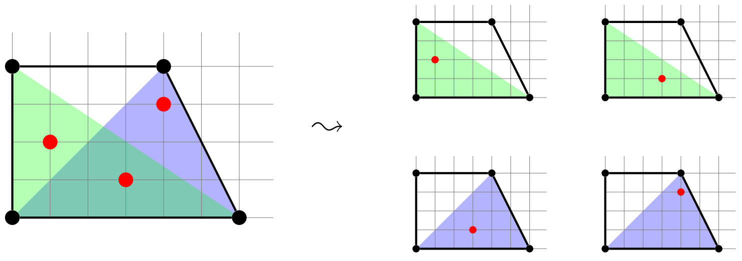

Figure 4.1: We use black points for monomial squares (outer points) and red points for the non-squares (inner points). The coloured triangles represent our circuits.

$\hexagon$

To further improve the circuit covering, research by Forsgård and de Wolff [FdW19] suggest a way to recognise redundant circuits.

4.1.6 Conjecture. *Let $\boldsymbol{\lambda}^i \in \mathbb{Q}^t$ denote the barycentric coordinates of our circuits C_i for $i = 0, \dots, k$. If there are $t_i > 0$ with $\boldsymbol{\lambda}^0 = \sum_{i=1}^{k} t_i \boldsymbol{\lambda}^i$, then the circuit C_0 is redundant, because every nonnegative polynomial supported on C_0 can be written as a sum of nonnegative polynomials supported on C_i for $i = 0, \dots, k$.*

However, to our knowledge, this statement is still unproven.

4.1.3 Computing a SONC Decomposition

In this section, we take a covering $(\boldsymbol{\lambda}, \mathsf{C})$, as computed in Section 4.1.2 and compute a SONC decomposition according to the given circuits. Unfortunately, if some negative points occur in multiple circuits, the optimisation problem is not a GP any more. So we introduce a further relaxation, by fixing the distribution of the negative coefficients. In Section 4.1.5, we present a way to improve this relaxation in case we do not have degenerate points.
Given some covering $(\boldsymbol{\lambda}, \mathsf{C})$, we have to find the coefficients of the circuit polynomials. In the end, we want to find a maximal shift $p_{\text{SONC}} \in \mathbb{R}$, such that we can find a decomposition

$$p - p_{\text{SONC}} = \sum_{C \in \mathsf{C}} \left(\sum_{\boldsymbol{\alpha} \in C \cap \text{MoSq}(p)} X_{C,\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} + X_{C,\boldsymbol{\beta}(C)} \boldsymbol{x}^{\boldsymbol{\beta}(C)} \right) \tag{SONC-decomp}$$

where the right hand side is a SONC and $\boldsymbol{X} \in \mathbb{R}^{|\mathsf{C}| \times t}$ is the distribution of the coefficients. Recall, that $\boldsymbol{\beta}(C)$ denotes the unique non-square of a circuit. The decomposition and the nonnegativity translate into the following (non-convex) optimisation problem.

$$\begin{aligned} p_{\text{SONC}} = \underset{\boldsymbol{X} \in \mathbb{R}^{|\mathsf{C}| \times |A(p)|}}{\text{maximise}} \quad & b_{\mathbf{0}} - \sum_{C \in \mathsf{C}} X_{C,\mathbf{0}} \\ \text{subject to} \quad & \sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\alpha}} = b_{\boldsymbol{\alpha}} && \text{for all } \boldsymbol{\alpha} \in \text{MoSq}(p), \boldsymbol{\alpha} \neq \mathbf{0} \\ & \sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\beta}} = -b_{\boldsymbol{\beta}} && \text{for all } \boldsymbol{\beta} \in \text{NoSq}(p) \\ & \prod_{\boldsymbol{\alpha} \in C \cap \text{MoSq}(p)} \left(\frac{X_{C,\boldsymbol{\alpha}}}{\lambda_{C,\boldsymbol{\alpha}}} \right)^{\lambda_{C,\boldsymbol{\alpha}}} = X_{C,\boldsymbol{\beta}(C)} && \text{for all } C \in \mathsf{C} \\ & X_{C,\boldsymbol{v}} \geq 0 && \text{for all } C \in \mathsf{C}, \boldsymbol{v} \in A(p). \end{aligned} \tag{SONC}$$

In the next step, we expand the feasible set, by replacing the equalities in the first two constraints by inequalities. We still obtain a valid bound for p if we use less positive weight, or if we use more negative weight. So we adjust the constraints to

$$\sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\alpha}} \leq b_{\boldsymbol{\alpha}} \qquad\qquad \sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\beta}} \geq -b_{\boldsymbol{\beta}}.$$

The first inequality already has the form required for (GP). But the second inequality does not match the shape. To circumvent this issue, we fix feasible values for the variables $X_{C,\boldsymbol{\beta}}$ for $\boldsymbol{\beta} \in \operatorname{NoSq}(p)$. Then the remaining problem is a Geometric Programme.

$$\begin{aligned}
p_{\text{SONC}} = \underset{\boldsymbol{X}}{\text{maximise}} \quad & b_{\mathbf{0}} - \sum_{C \in \mathsf{C}} X_{C,\mathbf{0}} \\
\text{subject to} \quad & \sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\alpha}} \leq b_{\boldsymbol{\alpha}} && \text{for all } \boldsymbol{\alpha} \in \operatorname{MoSq}(p), \boldsymbol{\alpha} \neq \mathbf{0} \\
& \prod_{\boldsymbol{\alpha} \in \mathsf{C} \cap \operatorname{MoSq}(p)} \left(\frac{X_{C,\boldsymbol{\alpha}}}{\lambda_{C,\boldsymbol{\alpha}}} \right)^{\lambda_{C,\boldsymbol{\alpha}}} = X_{C,\boldsymbol{\beta}(\boldsymbol{C})} && \text{for all } C \in \mathsf{C} \\
& X_{C,\boldsymbol{v}} \geq 0 && \text{for all } C \in \mathsf{C}, \boldsymbol{v} \in A(p).
\end{aligned} \tag{SONC-GP}$$

Intuitively, we distribute the coefficient $b_{\boldsymbol{\alpha}}$ into positive coefficients of the circuit polynomials $X_{C,\boldsymbol{\alpha}}$, such that each circuit polynomial exactly fulfils the condition of Theorem 2.6.2, i.e. the coefficient $X_{C,\boldsymbol{\beta}(C)}$ is exactly the circuit number.
In our basic approach, we use

$$X_{C,\boldsymbol{\beta}(C)} = \frac{-b_{\boldsymbol{\beta}(C)}}{|\{C' \in \mathsf{C} : \boldsymbol{\beta}(C) \in C'\}|} \quad \text{for all } C \in \mathsf{C}. \tag{Split}$$

The resulting GP, we can solve as described in Section 2.4.2. The optimal value of (SONC-GP) gives a lower bound for p, whereas the corresponding assignment for $\boldsymbol{X}$ yields the corresponding decomposition (SONC-decomp). In Section 4.1.5, we introduce a method to improve (Split).
To summarise our approach, we have the following.

4.1.7 Algorithm. The full algorithm to compute a lower bound for a polynomial via SONC is as follows.
Input: p -- polynomial
Output: p_{SONC} -- lower bound for p

```
1: function BOUND($p$)
2:     $(\boldsymbol{\lambda}, \mathsf{C}) \leftarrow$ fullcover $(p)$                          ▷ see Algorithm 4.1.4
3:     for $C \in \mathsf{C}$ do
4:         $X_{C,\boldsymbol{\beta}(C)} \leftarrow -b_{\boldsymbol{\beta}(C)}/|\{C' \in \mathsf{C} : \boldsymbol{\beta}(C) \in C'\}|$     ▷ see (Split)
5:     $\boldsymbol{X} \leftarrow$ solution of (SONC-GP)
6:     $p_{\text{SONC}} \leftarrow b_{\mathbf{0}} - \sum X_{C,\mathbf{0}}$
7:     return $p_{\text{SONC}}$
```

If required, this algorithm may also return $\boldsymbol{X}$ as a certificate for the bound. As (SONC-GP) allows for inequalities, we introduce additional variables $Y_{\boldsymbol{\alpha}} \in \mathbb{R}$, such that

$$\sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\alpha}} + Y_{\boldsymbol{\alpha}}^2 = b_{\boldsymbol{\alpha}}$$

for all $\mathbf{0} \neq \boldsymbol{\alpha} \in \mathrm{MoSq}\,(p)$. This yields the decomposition

$$p - p_{\mathrm{SONC}} = \sum_{C \in \mathsf{C}} \left(\sum_{\boldsymbol{\alpha} \in C \cap \mathrm{MoSq}(p)} X_{C,\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} - X_{C,\boldsymbol{\beta}(C)} \boldsymbol{x}^{\boldsymbol{\beta}(C)} \right) + \sum_{\boldsymbol{\alpha} \in \mathrm{MoSq}(p)} Y_{\boldsymbol{\alpha}}^2 \boldsymbol{x}^{\boldsymbol{\alpha}} \geq 0 \tag{SONC-solution}$$

which is a SONC and a sum of binomial squares, hence a certificate of nonnegativity. However, for simplicity we also call this a SONC decomposition, even is additional monomial squares occur. In Section 4.1.5 we give a criterion which ensures, that all inequalities are met with equality, i.e. we do not have monomial squares in the decomposition.
We continue our previous example to exhibit how the computation of the lower bounds works.

4.1.8 Example (Cont. of Example 4.1.5). We regard the polynomial

$$p = 2 + x^6 + y^4 + x^4y^4 - x^3y - xy^2 - x^4y^3.$$

For simpler notation, we enumerate the exponents

$$\begin{aligned} &\alpha_0 = (0,0)^T, && \alpha_1 = (6,0)^T, && \alpha_2 = (4,4)^T, && \alpha_3 = (0,4)^T, \\ &\beta_1 = (3,1)^T, && \beta_2 = (1,2)^T, && \beta_3 = (4,3)^T \end{aligned}$$

as well as the circuits

$$C_1 = \{\alpha_0, \alpha_1, \alpha_3, \beta_1\}, \quad C_2 = \{\alpha_0, \alpha_1, \alpha_3, \beta_2\}, \quad C_3 = \{\alpha_0, \alpha_1, \alpha_2, \beta_1\}, \quad C_4 = \{\alpha_0, \alpha_1, \alpha_2, \beta_3\}.$$

From these circuits, we get the following convex combinations.

$$\lambda_1 = \left(\frac{1}{4}, \frac{1}{2}, 0, \frac{1}{4}\right) \quad \lambda_2 = \left(\frac{1}{3}, \frac{1}{6}, 0, \frac{1}{2}\right) \quad \lambda_3 = \left(\frac{5}{12}, \frac{1}{3}, \frac{1}{4}, 0\right) \quad \lambda_4 = \left(\frac{1}{12}, \frac{1}{6}, \frac{3}{4}, 0\right)$$

In our circuit decomposition, the non-square $\beta_1 = (3,1)^T$ occurs in two circuits. So, we split the coefficient -1 evenly.

$$\begin{aligned} p_{\mathrm{SONC}} = \underset{\boldsymbol{X} > 0}{\text{maximise}} \quad & 2 - (X_{1,0} + X_{2,0} + X_{3,0} + X_{4,0}) \\ \text{subject to} \quad & X_{1,1} + X_{2,1} + X_{3,1} + X_{4,1} \leq 1 \\ & X_{1,3} + X_{2,3} \leq 1 \\ & X_{3,2} + X_{4,2} \leq 1 \\ & (4X_{1,0})^{\frac{1}{4}} \cdot (2X_{1,1})^{\frac{1}{2}} \cdot (4X_{1,3})^{\frac{1}{4}} = \frac{1}{2} \\ & (3X_{2,0})^{\frac{1}{3}} \cdot (6X_{2,1})^{\frac{1}{6}} \cdot (2X_{2,3})^{\frac{1}{2}} = 1 \\ & \left(\frac{12}{5}X_{3,0}\right)^{\frac{5}{12}} \cdot (3X_{3,1})^{\frac{1}{3}} \cdot (4X_{3,2})^{\frac{1}{4}} = \frac{1}{2} \\ & (12X_{4,0})^{\frac{1}{12}} \cdot (6X_{4,1})^{\frac{1}{6}} \cdot \left(\frac{4}{3}X_{4,2}\right)^{\frac{3}{4}} = 1 \end{aligned} \tag{SONC-Ex.}$$

After scaling the first and third equality, we obtain a GP in standard form. Solving this GP, we get the lower bound $p_{\mathrm{SONC}} \approx 1.667$. ○

Additionally, Algorithm 4.1.7 shows that SONC is in particular well-suited for polynomials of high degree.

4.1.9 Theorem. *The running time for SONC is independent of the degree in the sense that the following numbers are independent of the degree and depend only on the number of variables n and the size of the support t:*

- *the size of (LP) and the number of arithmetic operations in Algorithm 4.1.3,*
- *the size of (LP-opt) and the number of arithmetic operations in Algorithm 4.1.4*
- *the problem size of (SONC-GP).*

Proof. Recall, that t is independent of d. The size of both (LP) and (LP-opt) is in $\mathcal{O}(nt)$. We have to compute $(t-h)\cdot|\mathsf{C}| \leq t^2$ tuples of barycentric coordinates, each of size $n+1$. The largest GP, (SONC-GP), has size $\mathcal{O}(h(t-h)|\mathsf{C}|) \subseteq \mathcal{O}(t^3)$. So none of these depend on d. □

4.1.4 Degenerate Points

Under some circumstances, Algorithm 4.1.7 might fail to find a lower bound at all. This means, (SONC-GP) has no solution. One reason could be, that the given polynomial simply cannot be written as a SONC. In this case, the algorithm still yields the "correct" answer. However, another possible reason is that our cover in Algorithm 4.1.4 and our splitting (Split) was not chosen well. Recall, that we wish to distribute the weight of the positive coefficients among the circuits such that we obtain nonnegative circuit polynomials. For the constant term, we may use arbitrary weight for feasibility, but for every other monomial square, we may use at most the given weight $b_{\boldsymbol{\alpha}}$. This means, if we have some $C \in \mathsf{C}$ with $\mathbf{0} \notin C$ and $\boldsymbol{x} \in \mathbb{R}^n_+$ such that

$$\sum_{\boldsymbol{\alpha}\in \mathsf{C}\cap \mathrm{MoSq}(p)} b_{\boldsymbol{\alpha}}\boldsymbol{x}^{\boldsymbol{\alpha}} - X_{C,\boldsymbol{\beta}(C)}\boldsymbol{x}^{\boldsymbol{\beta}(C)} < 0, \tag{GP-infeas}$$

then there is no solution to (SONC-GP). But while (GP-infeas) is easily checked, it is only a sufficient, not a necessary criterion.
The above condition $\mathbf{0} \notin C$ also shows a problem, which of degenerate points may cause. Recall, that a degenerate point is an exponent of p, that lies on a face of the Newton polytope, which does not contain the origin. Therefore, every *non*-degenerate non-square can be covered with a circuit containing $\mathbf{0}$. If we meet condition (GP-infeas) for a non-degenerate point $\boldsymbol{\beta} := \boldsymbol{\beta}(C)$, we can also find a circuit C' with $\mathbf{0}, \boldsymbol{\beta} \in C'$. Then, we can redistribute the negative weight to $X_{C,\boldsymbol{\beta}} - \varepsilon$ and $X_{C',\boldsymbol{\beta}} + \varepsilon$ for some $\varepsilon > 0$ as necessary, or possibly even omitting C from the covering.
For degenerate points, however, this post-processing is not guaranteed to work. So, if we encounter the condition (GP-infeas) there, it is not clear, how to change the covering or the distribution of the negative coefficients.
In total, we have the following guarantee:

4.1.10 Lemma. *If $\mathscr{D}(p) = \emptyset$, then Algorithm 4.1.7 finds a lower bound.*

In the presence of degenerate points, the algorithm *may* fail, but often does not. We discuss experimental data in Section 5.3.

4.1.5 Improving the Coefficient Distribution

In this section, we describe how to extend the initial approach shown in Section 4.1.3 to obtain enhanced lower bounds for a fixed cover C. On a rough level the idea is an interior point algorithm, i.e. we start with some feasible solution and improve it by redistributing the coefficients of our SONC decomposition.
While Papp [Pap19] already described a numerical algorithm that finds the optimal SONC bound, our method provides two distinctive features. First, it does not depend on an external solver for which the problem would have to be transformed. As such, it nearly runs in place, since we need only $\mathcal{O}(|\mathsf{C}|)$ additional space. This is particularly important, because in our experiments the memory requirement usually determines the limit we could compute. Second, the ideas can be reused to generalise the post-processing in Section 4.5.
For simplicity, we first assume that every circuit contains $\mathbf{0}$, i.e. every circuit polynomial actually contributes to the absolute value. This is the same assumption we make in Section 4.5 when computing SONC certificates in exact arithmetic.
As mentioned before, we fix some cover C and search for the best SONC decomposition using this cover. We begin this section with a necessary condition for the optimal solution. Then, we use this property to improve the distribution of the negative coefficients across the circuit polynomials. In Section 4.1.3 fixing these coefficients was one of our relaxations, which we overcome here. Afterwards, we can solve (SONC-GP) again to improve the distribution of the positive coefficients. However, instead of calling a general purpose GP solver, we can also adapt the ideas from the negative coefficients. In the outlook, we describe how this assumption can be dropped both for the numerical computation and for certificates in exact arithmetic.
Suppose we are given some covering C. By our assumption, we have $\mathbf{0} \in C$ for all $C \in \mathsf{C}$. Hence, we can solve the constraints of the form

$$\prod_{\boldsymbol{\alpha} \in \mathsf{C}_+} \left(\frac{X_{C,\boldsymbol{\alpha}}}{\lambda_{C,\boldsymbol{\alpha}}} \right)^{\lambda_{C,\boldsymbol{\alpha}}} = X_{C,\boldsymbol{\beta}(C)} \qquad \text{for all } C \in \mathsf{C} \tag{4.3}$$

for the constant term $X_{C,\mathbf{0}}$ and replace it in the objective function. Also, for simplicity, we only consider the sum of the objective function. This yields the adjusted problem

$$\begin{aligned} p_{GP} = \underset{\boldsymbol{X} \in \mathbb{R}^{|\mathsf{C}| \times |A(p)|}}{\text{minimise}} \quad & f(\boldsymbol{X}) := \sum_{C \in \mathsf{C}} \lambda_{C,\mathbf{0}} \, X_{C,\boldsymbol{\beta}(C)}{}^{\frac{1}{\lambda_{C,\mathbf{0}}}} \cdot \prod_{\boldsymbol{\alpha} \in C_+^*} \left(\frac{\lambda_{C,\boldsymbol{\alpha}}}{X_{C,\boldsymbol{\alpha}}} \right)^{\frac{\lambda_{C,\boldsymbol{\alpha}}}{\lambda_{C,\mathbf{0}}}} \\ \text{subject to} \quad & \sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\alpha}} = b_{\boldsymbol{\alpha}} \qquad \text{for all } \boldsymbol{\alpha} \in \mathrm{MoSq}(p), \boldsymbol{\alpha} \neq \mathbf{0} \\ & \sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\beta}} = -b_{\boldsymbol{\beta}} \qquad \text{for all } \boldsymbol{\beta} \in \mathrm{NoSq}(p) \\ & X_{C,\boldsymbol{v}} \geq 0 \qquad \text{for all } C \in \mathsf{C}, \boldsymbol{v} \in A(p). \end{aligned} \tag{SONC2}$$

The optimal solution to (SONC2) yields the best SONC decomposition that can be obtained with the given cover C. We first derive some necessary conditions for this solution, which we later incorporate into our algorithm.

4.1.11 Lemma. *Let $\boldsymbol{X}^*$ be an optimal solution for (SONC2). Then*

1. $\frac{\partial f}{\partial X_{C,\boldsymbol{\alpha}}}(\boldsymbol{X}^*)$ *is a constant for each $\boldsymbol{\alpha} \in \operatorname{MoSq}(p)$, i.e. independent of C.*
2. $\frac{\partial f}{\partial X_{C,\boldsymbol{\beta}}}$ *is a constant for each $\boldsymbol{\beta} \in \operatorname{NoSq}(p)$, i.e. for any two circuits C_1 and C_2 with $\boldsymbol{\beta}(C_1) = \boldsymbol{\beta}(C_2)$ we have*

$$\frac{\partial f}{\partial X_{C_1,\boldsymbol{\beta}(C_1)}} = \frac{\partial f}{\partial X_{C_2,\boldsymbol{\beta}(C_2)}}.$$

The main idea is that if this is not the case we can shift a small bit of weight from one variable to another to obtain a better solution.

Proof. We only show the first statement, as the proof for the second is analogue. Assume we have some exponent $\boldsymbol{\alpha}$ and two circuit C_1, C_2 such that

$$\frac{\partial f}{\partial X_{C_1,\boldsymbol{\alpha}}} - \frac{\partial f}{\partial X_{C_2,\boldsymbol{\alpha}}} = \varepsilon > 0$$

Then, we define the new solution $\boldsymbol{X}'$ via

$$X'_{C_1,\boldsymbol{\alpha}} = X^*_{C,\boldsymbol{\alpha}} - \delta \qquad \text{and} \qquad X'_{C_2,\boldsymbol{\alpha}} = X^*_{C,\boldsymbol{\alpha}} + \delta$$

for some small δ keeping all other entries of $\boldsymbol{X}^*$. The constraints of (SONC2) are still satisfied, but the objective value changes to

$$f(\boldsymbol{X}') = f(\boldsymbol{X}^*) - \frac{\partial f}{\partial X_{C_1,\boldsymbol{\alpha}}} \cdot \delta + \frac{\partial f}{\partial X_{C_2,\boldsymbol{\alpha}}} \cdot \delta + \Theta = f(\boldsymbol{X}^*) - \varepsilon\delta + \Theta$$

for some $\Theta \in \mathcal{O}(\delta^2)$. Thus, for sufficiently small δ we have $f(\boldsymbol{X}') < f(\boldsymbol{X}^*)$, which means $\boldsymbol{X}^*$ was not optimal. □

Lemma 4.1.11 forms the basis for our new interior point algorithm.

Distributing the Negative Coefficients

Assume we are given some feasible solution $\boldsymbol{X}$ (that is not necessarily optimal). From this solution we adjust the values $X_{C,\boldsymbol{\beta}}$ to $X'_{C,\boldsymbol{\beta}}$, which satisfy Lemma 4.1.11 (2). We solve $\frac{\partial f}{\partial X_{C,\boldsymbol{\beta}}}$ for $X_{C,\boldsymbol{\beta}}$ and insert these expressions into the constraints

$$\sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\beta}} = -b_{\boldsymbol{\beta}} \qquad \text{for } \boldsymbol{\beta} \in \operatorname{NoSq}(p). \tag{NoSq}$$

This yields a polynomial with rational coefficients in one variable, which we numerically solve to obtain our new values $X'_{C,\boldsymbol{\beta}}$. Adjusting the constant terms $X_{C,\mathbf{0}}$ yields an improved feasible solution.

$$\begin{aligned}\mathrm{const}_{\boldsymbol{\beta}} = \frac{\partial f}{\partial X_{C,\boldsymbol{\beta}}} &= X'_{C,\boldsymbol{\beta}}{}^{\frac{1}{\lambda_{C,\mathbf{0}}}-1} \cdot \prod_{\boldsymbol{\alpha}\in C_+^*} \left(\frac{\lambda_{C,\boldsymbol{\alpha}}}{X_{C,\boldsymbol{\alpha}}}\right)^{\frac{\lambda_{C,\boldsymbol{\alpha}}}{\lambda_{C,\mathbf{0}}}} \\ &= X'_{C,\boldsymbol{\beta}}{}^{\frac{1}{\lambda_{C,\mathbf{0}}}-1} \cdot \left(\frac{X_{C,\mathbf{0}}}{\lambda_{C,\mathbf{0}}}\right) \cdot X_{C,\boldsymbol{\beta}(C)}{}^{-\frac{1}{\lambda_{C,\mathbf{0}}}}\end{aligned}$$

Solving this by $X'_{C,\boldsymbol{\beta}}$, we obtain

$$X'_{C,\boldsymbol{\beta}} = \left(\mathrm{const}_{\boldsymbol{\beta}} \cdot X_{C,\boldsymbol{\beta}(C)}{}^{\frac{1}{\lambda_{C,\mathbf{0}}}} \cdot \left(\frac{\lambda_{C,\mathbf{0}}}{X_{C,\mathbf{0}}}\right)\right)^{\frac{\lambda_{C,\mathbf{0}}}{1-\lambda_{C,\mathbf{0}}}}. \tag{4.4}$$

Summing them up over $\boldsymbol{\beta}$, the values still must satify (NoSq). Hence, we get

$$-b_{\boldsymbol{\beta}} = \sum_{C\in\mathsf{C}} X'_{C,\boldsymbol{\beta}} = \sum_{C\in\mathsf{C}} \left(\mathrm{const}_{\boldsymbol{\beta}} \cdot X_{C,\boldsymbol{\beta}(C)}{}^{\frac{1}{\lambda_{C,\mathbf{0}}}} \cdot \left(\frac{\lambda_{C,\mathbf{0}}}{X_{C,\mathbf{0}}}\right)\right)^{\frac{\lambda_{C,\mathbf{0}}}{1-\lambda_{C,\mathbf{0}}}}.$$

This is a univariate polynomial in $\mathrm{const}_{\boldsymbol{\beta}}$ which we can easily solve numerically. We may even exploit the property that every exponent and every coefficient is positive. Hence, the polynomial is strictly monotone, which allows faster solutions. Once we have $\mathrm{const}_{\boldsymbol{\beta}}$, we compute the updated coefficients via (4.4).

Distributing the Positive Coefficients

Using our updated solution, we can now fix the values $X_{C,\boldsymbol{\beta}}$ and solve (SONC-GP) again to obtain yet another improvement. Alternating between Section 4.1.5 and the GP, we may continue for some designated number of iterations or until the results numerically converges. However, practical tests have exposed some issues.

- On many instances the GP solver crashes after few iterations. This is most likely due to numerical problems.
- Using warm-starts, as we already had a good feasible solution, did not provide the expected speed-up.
- Our modeller for convex optimisation problems resulted in a significant increase in the required memory. Exploiting the structure of the problem, we can instead solve it nearly in place.

Our new approach to improve some given solution for (SONC2) by updating the values $X_{C,\boldsymbol{\alpha}}$ works similar as the one in Section 4.1.5, except we use Lemma 4.1.11 (1) and derive f by $X_{C,\boldsymbol{\alpha}}$.

$$
\begin{aligned}
\mathrm{const}_{\boldsymbol\alpha} &= \frac{\partial f}{\partial X_{C,\boldsymbol\alpha}} \\
&= \lambda_{C,\mathbf 0}\, X_{C,\boldsymbol\beta(C)}{}^{\frac{1}{\lambda_{C,\mathbf 0}}} \cdot \prod_{\substack{\boldsymbol\alpha'\in C_+^*\\ \boldsymbol\alpha'\neq\boldsymbol\alpha}} \left(\frac{\lambda_{C,\boldsymbol\alpha'}}{X_{C,\boldsymbol\alpha'}}\right)^{\frac{\lambda_{C,\boldsymbol\alpha'}}{\lambda_{C,\mathbf 0}}} \cdot \left(\frac{X'_{C,\boldsymbol\alpha}}{\lambda_{C,\boldsymbol\alpha}}\right)^{-\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}-1} \cdot \left(-\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}\cdot\frac{1}{\lambda_{C,\boldsymbol\alpha}}\right) \\
&= -\, X_{C,\boldsymbol\beta(C)}{}^{\frac{1}{\lambda_{C,\mathbf 0}}} \cdot \prod_{\substack{\boldsymbol\alpha'\in C_+^*\\ \boldsymbol\alpha'\neq\boldsymbol\alpha}} \left(\frac{\lambda_{C,\boldsymbol\alpha'}}{X_{C,\boldsymbol\alpha'}}\right)^{\frac{\lambda_{C,\boldsymbol\alpha'}}{\lambda_{C,\mathbf 0}}} \cdot \left(\frac{X'_{C,\boldsymbol\alpha}}{\lambda_{C,\boldsymbol\alpha}}\right)^{-\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}-1} \\
&= -\, X_{C,\boldsymbol\beta(C)}{}^{\frac{1}{\lambda_{C,\mathbf 0}}} \cdot \frac{X_{C,\mathbf 0}}{\lambda_{C,\mathbf 0}} \left(\frac{X_{C,\boldsymbol\alpha}}{\lambda_{C,\boldsymbol\alpha}}\right)^{\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}} \cdot \left(\frac{X'_{C,\boldsymbol\alpha}}{\lambda_{C,\boldsymbol\alpha}}\right)^{-\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}-1} \\
&= -\, X_{C,\boldsymbol\beta(C)}{}^{\frac{1}{\lambda_{C,\mathbf 0}}} \cdot \frac{X_{C,\mathbf 0}}{\lambda_{C,\mathbf 0}} \cdot \lambda_{C,\boldsymbol\alpha}\, X_{C,\boldsymbol\alpha}{}^{\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}} \cdot X'_{C,\boldsymbol\alpha}{}^{-\frac{\lambda_{C,\boldsymbol\alpha}}{\lambda_{C,\mathbf 0}}-1}
\end{aligned}
$$

Solving this equation for $X_{C,\boldsymbol\alpha}$ yields

$$
X'_{C,\boldsymbol\alpha} = \left(-\,\mathrm{const}_{\boldsymbol\alpha}^{-\lambda_{C,\mathbf 0}} \cdot X_{C,\boldsymbol\beta(C)} \cdot \left(\frac{X_{C,\mathbf 0}}{\lambda_{C,\mathbf 0}}\lambda_{C,\boldsymbol\alpha}\right)^{\lambda_{C,\mathbf 0}} \cdot X_{C,\boldsymbol\alpha}{}^{\lambda_{C,\boldsymbol\alpha}}\right)^{\frac{1}{\lambda_{C,\boldsymbol\alpha}+\lambda_{C,\mathbf 0}}} \tag{4.5}
$$

Summing up this equation over all circuits, we obtain

$$
b_{\boldsymbol\alpha} = \sum_{C\in\mathsf C} \left(-\,\mathrm{const}_{\boldsymbol\alpha}^{-\lambda_{C,\mathbf 0}} \cdot X_{C,\boldsymbol\beta(C)} \cdot \left(\frac{X_{C,\mathbf 0}}{\lambda_{C,\mathbf 0}}\lambda_{C,\boldsymbol\alpha}\right)^{\lambda_{C,\mathbf 0}} \cdot X_{C,\boldsymbol\alpha}{}^{\lambda_{C,\boldsymbol\alpha}}\right)^{\frac{1}{\lambda_{C,\boldsymbol\alpha}+\lambda_{C,\mathbf 0}}}
$$

Again, this is a polynomial in $\mathrm{const}_{\boldsymbol\alpha}$ with rational exponents, that we can easily solve numerically. Having computed $\mathrm{const}_{\boldsymbol\alpha}$, we get an improved solution $\boldsymbol X'$ via (4.5).
This procedure, we can repeat for different exponents, even alternating between monomial squares and non-squares. Each iteration results in an improvement, but as (SONC2) is bounded by $-\inf p$, the procedure converges.

Outlook for the General Case

Previously, we had to assume that every circuit contained the vector $\mathbf 0$, so we could solve the expression of the circuit number for the absolute value $X_{C,\mathbf 0}$ of the circuit polynomial. In this section, we describe how we can drop this precondition.
On a higher level, our iteration fixes some exponential $\boldsymbol\alpha$ or $\boldsymbol\beta$ and focuses on the values $X_{C,\boldsymbol\alpha}$ or $X_{C,\boldsymbol\beta}$ for all $C \in \mathsf C$. It then reallocates the weight among these variables and updates $X_{C,\mathbf 0}$ accordingly to obtain a better solution. If some circuit does not contain $\mathbf 0$, then simply adjusting $X_{C,\mathbf 0}$ to satisfy (4.3) is not possible. However, we may instead adjust some intermediate variables whose exponents lie in some sense closer to the origin and continue recursively, until we reach $\mathbf 0$.
Formally, for some cover $\mathsf C$ and some variable assignment $\boldsymbol X$, we introduce the cover graph $G(\mathsf C)$ as a weighted graph as follows.

- The vertices are the variables, i.e. $V(\mathsf{C}) = \{X_{C,\boldsymbol{v}} : C \in \mathsf{C}, \boldsymbol{v} \in A(p)\}$.
- The edges are
$$E(\mathsf{C}) = \bigcup_{\boldsymbol{v} \in A(p)} \{(X_{C_1,\boldsymbol{v}}, X_{C_2,\boldsymbol{v}}) : C_1, C_2 \in \mathsf{C}\} \cup \bigcup_{C \in \mathsf{C}} \{(X_{C,\boldsymbol{u}}, X_{C,\boldsymbol{v}}) : \boldsymbol{u}, \boldsymbol{v} \in C\}$$
Edges from the left union we call exponent edges, while edges from the right union we call circuit edges.
- Exponent edges have weight 0. For some circuit edge $(X_{C,\boldsymbol{u}}, X_{C,\boldsymbol{v}})$, we solve (4.3) for $X_{C,\boldsymbol{v}}$, derive it by $X_{C,\boldsymbol{u}}$, insert the current assignment for the variables and logarithmise the result.

The assignment $\boldsymbol{X}$ only affects the weights of the edges and not the topology of the graph. Having and edge weight $w(X, X')$ means if we decrease X by δ, then we have to increase X' by $\exp(\delta \cdot w(X, X'))$, to keep the corresponding constraint satisfied.
Using the graph, we can formalise the occurrence of monomial squares in our SONC decomposition, as mentioned in (SONC-solution).

4.1.12 Lemma. *If the cover graph is connected, then an optimal solution of (SONC-GP) satisfies all inequalities with equality.*

Proof. For each vertex, we find an (unweighted) shortest path to some $X_{C,\boldsymbol{0}}$. Since the graph is connected, such a path always exists. By construction, these paths have the form
$$(X_{C_0,\boldsymbol{\alpha}_1},) X_{C_1,\boldsymbol{\alpha}_1}, X_{C_1,\boldsymbol{\alpha}_2}, X_{C_2,\boldsymbol{\alpha}_2}, \ldots, X_{C_k,\boldsymbol{0}},$$
i.e. we alternatingly change the circuit and the exponent in the indices of our nodes. The first step here does not have to occur. Depending on which index changes, we call such a step a circuit step or an exponent step. We say k is the distance of $\boldsymbol{\alpha}_1$ to the origin $\boldsymbol{0}$.
Assume for some exponent $\boldsymbol{0} \neq \boldsymbol{\alpha}_1 \in \operatorname{MoSq}(p)$ we have $\sum_{C \in \mathsf{C}} X_{C,\boldsymbol{\alpha}_1} < b_{\boldsymbol{\alpha}}$. Choose $\boldsymbol{\alpha}_1$ with maximal distance to $\boldsymbol{0}$ such that the shortest path starts with a circuit step. For our updated solution $\boldsymbol{X}'$ we set
$$X'_{C_1,\boldsymbol{\alpha}_1} := b_{\boldsymbol{\alpha}} - \sum_{C \in \mathsf{C} \setminus \{C_1\}} X_{C,\boldsymbol{\alpha}_1}$$
and solve the constraint (4.3) for $X_{C_1,\boldsymbol{\alpha}_2}$ and update the value accordingly. Note that $X'_{C_1,\boldsymbol{\alpha}_2} < X_{C_1,\boldsymbol{\alpha}_2}$. All other values remain the same. Hence, we still have a feasible solution.
In total, in the problem (SONC-GP) we have one constraint more that is satisfied with equality, or the overall distance of vertices from the origin where (SONC-GP) is satisfied with strict inequality is reduced by one. Therefore, iterating this procedure terminates, which means we end up with a solution where every constraint (SONC-GP) is satisfied with equality. □

The above idea, to reroute weight of the variables along the edges we combine with our approach from Lemma 4.1.11 that our derivatives must be a constant for each exponent. In our generalised setting, we now require that for every exponent $\boldsymbol{v} \in A(p)$, all paths from $X_{\boldsymbol{0}}$ to $X_{\boldsymbol{v}}$ must have the same length. However, the details of this approach would require a considerable amount of additional work, so we leave it open as a possible avenue for future research.

4.2 Computing Minima

So far, in Section 4.1, we have computed a lower bound for the infimum of a given polynomial p. To judge the quality of these computations, we still need upper bounds for the infimum. As we have seen in Chapter 3, computing the global infimum is hard. Even worse, it is coNP-hard to decide, whether some given value is a local minimum of p [MK87]. But local search methods like gradient descent often converge to the local minimum. Even if they do not converge to a local minimum, we make use of the simple fact that any function value is an upper bound for the infimum. Since the quality of local searches heavily depends on the choice of the starting point, we describe a heuristic method how to find a reasonable choice by using SONC. As with Theorem 2.6.4, we follow the work of Helena Müller [Mül18].
Let p be some given polynomial. For the relaxed polynomial $\overline{p}$, we first compute a covering C and then the corresponding SONC decomposition

$$\overline{p} - p_{\text{SONC}} = \sum_{C \in \mathsf{C}} q_C$$

where q_C are circuit polynomials, as given in (SONC-decomp). With Theorem 2.6.4, we can explicitly compute the minimiser of each circuit polynomial q_C. Let $\boldsymbol{m}_C$ be the respective minima of the q_C. Then we use their barycentre

$$\boldsymbol{m} := \frac{1}{|\mathsf{C}|} \sum_{C \in \mathsf{C}} \boldsymbol{m}_C$$

as starting point for a gradient method to find a local minimum of $\overline{p}$. This point is then used as starting point for local minimisation of p. The expectation is, that $\boldsymbol{m}$ often lies sufficiently close to the global minimum of $\overline{p}$, and this minimum lies close to the global minimum of p. In these cases, the result will also be the global minimum of p. See Section 5.4 for the results how well the following heuristic performs.

4.2.1 Algorithm. The algorithm to numerically compute (local) minima via SONC in polynomial time works as follows.

```
Input: $p$ -- polynomial
Output: $\boldsymbol{m}$ -- local minimum of $p$
1: function SONC-Min(p)
2:     run SONC on $p$
3:     compute SONC-decomposition $p = q_1 + \ldots + q_{|\mathsf{C}|} + \mathsf{const}$
4:     for $C \in \mathsf{C}$ do
5:         set $\boldsymbol{s}_i$ as solution of (2.4) for $q_i$
6:         $\boldsymbol{m}_i \leftarrow e^{\boldsymbol{s}_i}$                    ▷ minimum of circuit-polynomial $q_i$
7:     $\boldsymbol{m}_{\mathsf{start}} \leftarrow \frac{1}{|\mathsf{C}|} \sum_{i=1}^{|\mathsf{C}|} \boldsymbol{m}_i$
8:     $\boldsymbol{m}_{\mathsf{relax}} \leftarrow$ LocalMin $(\overline{p}, \boldsymbol{m}_{\mathsf{start}})$      ▷ local minimum of $\overline{p}$, via gradient method
9:     return $\boldsymbol{m} \leftarrow$ LocalMin $(p, \boldsymbol{m}_{\mathsf{relax}})$     ▷ local minimum of $p$, via gradient method
```

Proof. As observed in Section 4.1.2, we have $|\mathsf{C}| \in \mathcal{O}(t^2)$, For each minimiser, we have to solve a linear equation system, which can be done in $\mathcal{O}(n^3)$. For the local minimum, we can use nonlinear gradient descent, which has quadratic convergence [FR64]. Hence, we have a polynomial running time. □

This heuristic for computing minima is a part of our branch-and-bound approach we describe in the following section. Furthermore it is implemented in our software POEM. We discuss some results in Section 5.4.

4.3 Branch-and-Bound

In our approach, we introduced a relaxation step (sign-relaxation), to account for the possibly negative sign of the terms that are not monomial squares. However, if we restrict ourselves to a single orthant, we know the exact sign of every term in our polynomial, which usually allows for a better bound. The downside of this idea is the additional factor 2^n in the running time, so we do not have polynomial running time any more. To reduce this computational effort, we instead use a branch-and-bound algorithm, where we fix the sign of one variable after the other. In the following section, we present this approach. While the running time may still be exponential, it is feasible for moderate parameters. We discuss our practical experiments in Section 5.4. The whole section closely follows our paper [Sei21a].
Recall, that the relaxation step (sign-relaxation) is given by

$$\overline{p} = \sum_{\boldsymbol{\alpha} \in \mathrm{MoSq}(p)} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} - \sum_{\boldsymbol{\beta} \in \mathrm{NoSq}(p)} |b_{\boldsymbol{\beta}}| \, x^{\boldsymbol{\beta}}$$

where we give every monomial non-square a negative sign and then restrict ourselves to the positive orthant. However, this is overly pessimistic already for a single variable, as can be seen in the following, simple example.

4.3.1 Example. Let $p = x^4 + x^3 - x + 1$, which has minimum ≈ 0.682. This polynomial is relaxed to $\overline{p} = x^4 - x^3 - x + 1$, which has minimum 0. ○

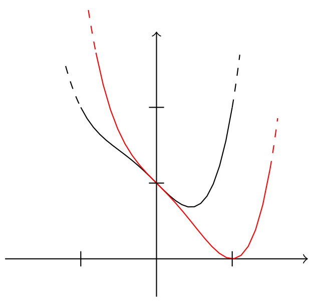

Figure 4.2: Graphs of p (black) and $\overline{p}$ (red)

To avoid this problem, we can use a branch-and-bound algorithm, where we branch over the sign of the variables. The high-level idea of SONC is to use the weight of the monomial squares (positive terms) to cancel out the negative weight of the non-squares (negative terms). By fixing the signs for some of the variables, we can identify additional terms as positive, so they can be treated like monomial squares. Hence, we do not have

to cancel out their negative weight, but in addition gain new positive weights to cancel out the remaining negative terms.
To denote our restrictions on the signs of the variables, we introduce sign cones.

4.3.2 Definition. Let $\boldsymbol{s} \in \{-1,0,1\}^n$. We call $\boldsymbol{s}$ a sign vector, where $-1,0,1$ represents negative/unknown/positive sign, respectively, and define the corresponding sign cone as

$$c_{\boldsymbol{s}} := \{\boldsymbol{x} \in \mathbb{R}^n : x_i \cdot s_i \geq 0 \text{ for all } i = 1,\dots,n\}\,.$$

For some sign vector $\boldsymbol{s}$, the positive points are given by

$$\mathsf{Pos}_{\boldsymbol{s}}(p) = \left\{\boldsymbol{\alpha} \in A(p) : \operatorname{sgn}(\boldsymbol{\alpha}) \cdot \prod_{i=1}^{n} s_i^{\boldsymbol{\alpha}_i \bmod 2} = 1\right\}$$

with the convention $0^0 = 1$. The negative points are $\mathsf{Neg}_{\boldsymbol{s}}(p) := A(p) \setminus \mathsf{Pos}_{\boldsymbol{s}}(p)$. The corresponding positive and negative terms are the terms $b_{\boldsymbol{\alpha}}\boldsymbol{x}^{\boldsymbol{\alpha}}$ for $\boldsymbol{\alpha} \in \mathsf{Pos}_{\boldsymbol{s}}(p)$ and $\boldsymbol{\alpha} \in \mathsf{Neg}_{\boldsymbol{s}}(p)$, respectively. The whole polynomial restricted to the domain $c_{\boldsymbol{s}}$ we denote by $p_{\boldsymbol{s}}$. ⬡

Clearly, $\mathrm{MoSq}(p) \subseteq \mathsf{Pos}_{\boldsymbol{s}}(p)$ for any sign vector $\boldsymbol{s}$. If it is a strict subset, we obtain the improved relaxation

$$\overline{p}_{\boldsymbol{s}} = \sum_{\boldsymbol{\alpha} \in \mathsf{Pos}_{\boldsymbol{s}}(p)} b_{\boldsymbol{\alpha}}\boldsymbol{x}^{\boldsymbol{\alpha}} - \sum_{\boldsymbol{\beta} \in \mathsf{Neg}_{\boldsymbol{s}}(p)} |b_{\boldsymbol{\beta}}|\, x^{\boldsymbol{\beta}}.$$

Note, that by fixing more signs, the set of positive terms may only increase.
Our branch-and-bound algorithm creates a binary search tree, where each node has a sign vector and a flag, whether it is active or not. Furthermore, we store the best lower bound and the lowest function value we found so far over the corresponding sign cone. For simplicity, we identify the nodes with their sign vectors and denote the lower bounds as $p_{\boldsymbol{s}}.\mathsf{low}$. The global lower bound is thus stored in $p_{\boldsymbol{0}}.\mathsf{low}$.
Initially, the tree consists only of the root node, which is active and has sign vector $\boldsymbol{s} = \boldsymbol{0}$. So it corresponds to $c_{\boldsymbol{s}} = \mathbb{R}^n$ as the domain and we compute bounds as in Section 4.1. In each iteration, we then pick some active node, which becomes inactive. If it satisfies any bounding criterion, we continue with the next iteration. Otherwise, we determine some index i with undetermined sign $s_i = 0$ and create two new child nodes for $\boldsymbol{s}$, where we update $\boldsymbol{s}$ with $s_i = \pm 1$. We compute lower bounds and minimisers for both nodes and mark them as active. Then we continue with the next iteration.

4.3.3 Algorithm. We have the following branch-and-bound blueprint.
Input: p -- polynomial
Output: $p_{\boldsymbol{0}}.\mathsf{low}$ -- lower bound of p

```
1: function BRANCH(p)
2:     run SONC, SAGE on $p$
3:     active := $\{p\}$
4:     while active $\neq \emptyset$ do
5:         pick some $p_{\boldsymbol{s}} \in$ active
6:         active = active $\setminus \{p_{\boldsymbol{s}}\}$
```

```
7:      if $p_{\boldsymbol{s}}$ does not satisfy bounding criterion then
8:          pick index $i$ with $s_i = 0$
9:          set $\boldsymbol{s}^+$ as $\boldsymbol{s}$ with $s_i = 1$ and $\boldsymbol{s}^-$ as $\boldsymbol{s}$ with $s_i = -1$
10:         compute SONC/SAGE bounds for $p_{\boldsymbol{s}^+}$ and $p_{\boldsymbol{s}^-}$
11:         $\mathsf{active} = \mathsf{active} \cup \{p_{\boldsymbol{s}^+}, p_{\boldsymbol{s}^-}\}$
```

At this point, the following steps have to be made more precise.

- Which node $\boldsymbol{s}$ do we pick in Line 5?
- What are our bounding criteria in Line 7?
- Which index i do we choose in Line 8?

We address these issues in the following subsections. First, we make some observations, that hold even in this general setting.
Each node has a bound at least as good as its parent. From the mathematical side, this is expected since we restrict the domain, and we achieve it algorithmically, because the certificate of the parent is also a certificate for the node itself.
The following statement provides our improved lower bound.

4.3.4 Lemma. *Let T be the search tree at any (possibly intermediate) state. A lower bound for the polynomial p is given by*

$$\inf p \geq \min \{p_{\boldsymbol{s}}.\mathsf{low} : \boldsymbol{s} \textit{ is leaf in } T\}.$$

Proof. The sign cones, represented by the leaves, cover the whole space $\mathbb{R}^n$, because in each step, we split the cone into $c_{\boldsymbol{s}} = c_{\boldsymbol{s}^+} \cup c_{\boldsymbol{s}^-}$. (Actually, it is a partition, up to cases where some variables are zero.) Put

$$M := \min \{p_{\boldsymbol{s}}.\mathsf{low} : \boldsymbol{s} \text{ is leaf in } T\}.$$

For any $\boldsymbol{x} \in \mathbb{R}^n$ choose some corresponding sign cone $\boldsymbol{x} \in c_{\boldsymbol{s}}$, where $\boldsymbol{s}$ is a leaf of T. So

$$p(x) = p_{\boldsymbol{s}}(x) \geq p_{\boldsymbol{s}}.\mathsf{low} \geq M.$$

Hence, globally we have $p \geq M$. □

4.3.1 Bounding Criteria

A crucial part in a branch-and-bound algorithm is to have efficient bounds. So we need some easily checkable criteria, which allow us to cut off a branch of the search tree.

4.3.5 Definition. Let $\boldsymbol{c}$ be our current node. We define the following criteria for cutting off branches.

Min If we have found some argument $\boldsymbol{x}$ such that $p(\boldsymbol{x}) \leq p_{\boldsymbol{c}}.\mathsf{low}$, then we cut off the branch at $\boldsymbol{c}$.

Leaf If there is some leaf $\boldsymbol{s} \in \{-1, 1\}^n$, i.e. leaf $\boldsymbol{s}$ lies at depth n, with $p_{\boldsymbol{s}}.\mathsf{low} \leq p_{\boldsymbol{c}}.\mathsf{low}$, then we cut off the branch at $\boldsymbol{c}$. ○

4.3.6 Lemma. *The above cut criteria are correct.*

Proof. Since both criteria are independent, we show their correctness separately.

Min According to Lemma 4.3.4, our lower bound is

$$M := \min \{p_{\boldsymbol{s}}.\mathsf{low} : \boldsymbol{s} \text{ is leaf in } T\}.$$

As observed before, in any state, the sign cones cover $\mathbb{R}^n$, so let $\boldsymbol{x} \in c_{\boldsymbol{s}}$ for some $\boldsymbol{s}$. Then, of course, $p_{\boldsymbol{s}}.\mathsf{low} \leq p(\boldsymbol{x})$. Hence, branching $\boldsymbol{c}$ any further, would only increase $p_{\boldsymbol{c}}.\mathsf{low}$ but not affect the minimum. So we can cut off the branch at $\boldsymbol{c}$.

Leaf Since $\boldsymbol{s} \in \{-1, 1\}^n$, we cannot branch it any further, so its bound will not improve. Therefore, $M \leq p_{\boldsymbol{s}}.\mathsf{low}$. Again, increasing $p_{\boldsymbol{c}}.\mathsf{low}$ will then not affect the minimum, so can cut off the branch at $\boldsymbol{c}$. □

Both criteria can easily be checked. We separately store the lowest value, we have found, and the lowest value of some leaf $p_{\boldsymbol{s}}.\mathsf{low}$ for $\boldsymbol{s} \in \{-1, 1\}^n$. (As long as we have not processed any such node, this value is ∞.) Then both Min and Leaf are single comparisons, which take $\mathcal{O}(1)$ time.

4.3.2 Choice of Branching Node

Another problem to be addressed is the choice of the node, on which to branch in Line 5. With regard to the quality of the solution, the best choice is to choose the node with the smallest lower bound. As we have seen in Lemma 4.3.4, the final bound is the smallest bound of any leaf. Hence, if we do not improve this worst bound, the final bound will not improve. The main disadvantage is, that the number of active leaves can grow exponentially, so our upper bound for the required space is exponential in n.
If memory is an issue, then the tree can also be traversed in a depth-first-search. With this strategy, we ensure $|\mathsf{active}| \leq n + 1$, so the computation runs in polynomial space. The significant disadvantage is the higher running time, because we compute more nodes than with the previous strategy.

4.3.3 Practical Improvements

If we choose the node with the worst bound for further branching, then the criterion Min never applies. However, for numerical computations, we use a relaxed version. Let $p(\boldsymbol{m})$ be the lowest function value we found so far and let ε be some given accuracy. If for the current node $\boldsymbol{c}$ we have $p_{\boldsymbol{c}}.\mathsf{low} \geq p(\boldsymbol{m}) + \varepsilon$, we stop the whole computation, because we already have solved the problem up to accuracy ε. To use this criterion, we integrate the computation of local minima via SONC-Min into the branch-and-bound method. The global minimum is also more likely to be found in the sign cone with the worst lower bound. So, whenever we compute bounds for a sign cone, we also run a local minimiser. However, the minima we compute for the circuit polynomials always lie in the positive orthant. Hence, our starting point $\boldsymbol{m}_{\mathsf{relax}}$ for the minimisation lies in the positive orthant as well. To comply with our orthant restriction, we flip some signs of our starting point and define the vector $\boldsymbol{m}'_{\mathsf{relax}}$ with

$$(m'_{\mathsf{relax}})_i = \begin{cases} -\left(m_{\mathsf{relax}}\right)_i & s_i = -1 \\ \left(m_{\mathsf{relax}}\right)_i & \text{else} \end{cases}.$$

Furthermore, we adjust Leaf as follows. Once we reach a node $\boldsymbol{s} \in \{-1, 1\}^n$, i.e. all signs are known, we stop the whole algorithm. We cannot improve this node by further branching, since by our choice of $\boldsymbol{s}$ and Lemma 4.3.4, we already have $p.\mathsf{low} = p_{\boldsymbol{s}}.\mathsf{low}$. Hence, we cannot improve the bound of p by our approach any more.
Concerning the space requirement, the largest part comes from solving the GP/REP, when computing the lower bound. But once we have this bound, we only store the bound itself, which is a small amount of data. Hence, for moderate choices of n, the space requirement for storing the search tree itself is negligible compared to the space requirement for solving the current node. Therefore, the potentially exponential growth of the search tree practically is not an issue.
All combined, this yields the following algorithm.

4.3.7 Algorithm. We have the following branch-and-bound algorithm, whose running time is fixed parameter tractable in the number of variables n.
Input: p -- polynomial
Output: $p.\mathsf{low}$ -- lower bound of p

```
 1: function TRAVERSE(p)
 2:     run SONC, SAGE on $p$
 3:     min ← SONC-Min($p$)
 4:     active ← $\{p\}$
 5:     while active $\neq \emptyset$ do
 6:         $\boldsymbol{s} \leftarrow \operatorname{argmin}\{p_{\boldsymbol{s}}.\mathsf{low} : \boldsymbol{s} \in \{-1,0,1\}^n, p_{\boldsymbol{s}} \in \mathsf{active}\}$      ▷ cone with worst bound
 7:         active ← active \ $\{p_{\boldsymbol{s}}\}$
 8:         if $p_{\boldsymbol{s}}$ satisfies Min or Leaf then                    ▷ see Section 4.3.1
 9:             return $p_{\boldsymbol{0}}.\mathsf{low}$
10:         compute SONC/SAGE for $p_{\boldsymbol{s}^+}$ and $p_{\boldsymbol{s}^-}$
11:         propagate new bound upwards
12:         if SONC-Min($p_{\boldsymbol{s}^+}$) < min or SONC-Min($p_{\boldsymbol{s}^-}$) < min then
13:             update min
14:         active ← active ∪ $\{p_{\boldsymbol{s}^+}, p_{\boldsymbol{s}^-}\}$
```

Proof. Our search tree is a binary tree of height at most n, so it has at most $2^{n+1} - 1$ nodes, which means at most $2^{n+1} - 1$ different polynomials are involved. Furthermore, in each state, exactly the leaves are in active and we never remove nodes from the tree. Therefore, each polynomial is chosen at most once in the loop in Line 6. □

As a final variation, it turned out that SAGE takes significantly longer than SONC, but for most sign cones the bound computed via SONC suffices. So, initially we only compute a lower bound via SONC for each $p_{\boldsymbol{s}}$. If some node is chosen for the first time, we then compute a lower bound via SAGE and the node remains active. Only if this node is chosen a second time, it becomes inactive and we branch into the two sub-cones.

4.4 Minimal Orthants

As alternative to the branch-and-bound algorithm, we can find a sufficient subset of the leaves and directly compute lower bounds for these polynomials.
As soon as we are given a concrete orthant, i.e. we know the sign of every variable, we can compute the effective sign of each term, i.e. we know whether it is positive or negative. To keep consistent with our previous notation for the relaxation, we also denote this polynomial as $p = (A(p), b(p))$. Now we define a partial order $b_1 \leq b_2$ on the effective coefficient vectors as elementwise $\leq$. This lifts to a partial order on the polynomials

$$\operatorname{poly}(A, b_1) \leq \operatorname{poly}(A, b_2) :\Leftrightarrow b_1 \leq b_2.$$

Going over all orthants yields 2^n polynomials. But the crucial observation is that we only need to compute bounds for the *minimal* polynomials.

4.4.1 Example. Consider the following polynomial with 3 variables.

$$\begin{aligned} p = 2.723 + 3.932 \cdot x_2^8 + 6.054 \cdot x_1^2 + 1.963 \cdot x_1^4 x_2^2 - 1.204 \cdot x_0 x_1 x_2^3 + 1.462 \cdot x_0 x_1^2 x_2 \\ + 1.766 \cdot x_0 x_1^2 x_2^2 + 0.841 \cdot x_0 x_1^2 x_2^4 - 0.329 \cdot x_0^2 x_1 x_2^2 + 7.57 \cdot x_0^2 x_1^2 x_2^4 + 2.428 \cdot x_0^4 x_2^2 \end{aligned}$$

Then the minimal orthants are given by the signs $(-,+,+)$, $(-,+,-)$ and $(-,-,+)$. So instead of solving $8 = 2^3$ instances, we only have to solve the three instances where we restrict p to each of the above orthants. ○

4.4.1 Computing Minimal Orthants

For convenience, we define the indicator function for strictly negative terms

$$\operatorname{neg}(x) = \begin{cases} 1 & : x < 0 \\ 0 & : x \geq 0 \end{cases}.$$

If called on a vector, the function is applied elementwise.

4.4.2 Algorithm. Computing the orthants with minimal coefficient vector is fixed parameter tractable in n, via the following algorithm.

```
Input: $p$ -- polynomial
Output: min -- set of orthants, where coefficients have minimal effective sign
  function MinimalOrthants(p)
      min = $\emptyset$
      for sign $\in \{0,1\}^n$ do                                  ▷ fork over all orthants
          $\boldsymbol{v}$ = (sign $\cdot \boldsymbol{A} + \operatorname{neg}(\boldsymbol{b})) \bmod 2$
          for $(e, s) \in$ min do
              if $e \leq v$ then
                  continue with next sign                            ▷ $v$ is not minimal
              if $v < e$ then
                  min.$remove(e, s)$                                 ▷ $e$ is not minimal
          min.$add(v,$ sign$)$                ▷ $v$ is minimal, if we reach the end of the for-loop
```

Proof. Let t' be the number of non-squares. The length of min is bounded by both 2^n and $\binom{t'}{t'/2}$, which is the length of the maximal antichain. Furthermore, each comparison runs in $\mathcal{O}(t')$. So the overall running time is

$$\mathcal{O}\left(t' \cdot 2^n \cdot \min\left(2^n, \binom{t'}{t'/2}\right)\right) \subseteq \mathcal{O}\left(t \cdot 4^n\right)$$

which is fixed parameter tractable in n. □

In particular, the proof shows that this approach is useful for polynomials with few non-squares. Possibly, we could improve the algorithm further by exploiting, that $v = \mathsf{sign} \cdot A + \text{neg}(b) \bmod 2$ is an affine map in $\mathbb{Z}_2$.
Experiments show that for $n = 10$ variables and $t' = 100$ non-squares this can be done in about 2 seconds. We consider the problem of determining the minimal orthants practically feasible for values $n \leq 15$.
Then we create polynomials

$$p_v = \text{poly}(A, v) \quad \text{for all } (v, \mathsf{sign}) \in \mathsf{min}$$

and optimise each over the positive orthant. Hence, the running time significantly depends on the number of minimal orthants. For polynomials with many monomial non-squares, we usually have $|\mathsf{min}| = 2^n$, but for instances with few monomial non-squares, we significantly reduce the running time by restricting ourselves to the minimal orthants. For the final lower bound, we then get

$$p.\mathsf{low} = \min\left\{p_v.\mathsf{low} : (v, \mathsf{sign}) \in \mathsf{min}\right\}.$$

The advantage of this approach, compared to the search tree, is its easy parallelisation. The major disadvantage is that a numerical failure in a single polynomial p_v already causes the trivial bound $p.\mathsf{low} = -\infty$. We discuss the quality of the results and the frequency of this problem in Section 5.4.

4.4.2 Reducing the Search Tree

The idea of this section also gives rise to a variant of the branch-and-bound approach from Section 4.3. First, we compute the minimal orthants min as described in Section 4.4.1. Then, we create a tree whose leaves are the elements of min and we branch the signs of the variables $x_1, \ldots, x_n$ in that order. Whenever we compute a node of the tree, that only has a single child, we further descend down the tree until we arrive at a node with two children, or a leaf. Otherwise we apply the same algorithm as in Section 4.3, including the criteria for cutting off a branch. We denote this algorithm by TRAVERSE-sparse.

4.5 Lower Bounds in Exact Arithmetic

So far, our algorithms contained optimisation problems, which we solved numerically. So our results were floating point values, which are subject to numerical errors. As a consequence, if a polynomial is certified as nonnegative, it might actually have small negative function values, no matter which of the methods SONC, SAGE or SOS we use. However, in some applications it is crucial to have a reliable lower bound. Examples are evaluation of functions in guaranteed bounds [CHJL11] or formal proofs [MAGW15], most prominently in the formal verification of the Kepler Conjecture [HAB+17].
In this section, we address this problem, closely following [MSdW19]. Assuming p has no degenerate points, we present two algorithms for converting a numerical solution for SONC and SAGE into a lower bound in exact arithmetic. For the case that we have degenerate points, we give a short outlook how this problem can be approached. Overall, our approach is similar to the rounding-projection procedure, suggested by Parrilo and Peyrl [PP08].
The analysis of the computational complexity now becomes more involved. For the numerical computations we could use the unit cost model. For exact arithmetic, however, this no longer is adequate, so we also consider the bit sizes of our coefficients.
We denote numerical solutions $\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{\lambda}}$ with a tilde, intermediate rational solutions $\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{\lambda}}$ with a hat, and our final rational solution with regular letters. To obtain rational values, we write $x' = \texttt{round}\,(x, \delta)$. Here, x can be a numerical value or an algebraic expression and δ denotes the precision, such that for the result $x' \in \mathbb{Q}$ we have $|x' - x| \leq \delta$. If we additionally require $x' \geq x$, we write $x' = \texttt{round-up}\,(x, \delta)$. When applied to a vector or matrix, the function is applied elementwise.

4.5.1 Symbolic Post-Processing for SONC

We assume that our given polynomial has no degenerate points and design an algorithm, called OPTSONC, to certify lower bounds of a given polynomial via SONC decompositions. We stay close to the approach given in Section 4.1, followed by a post-processing which transform the numerical solution into exact arithmetic. But unlike before, we already perform the calculations in Algorithm 4.1.3 in exact arithmetic when computing the cover and its corresponding coefficients. As we have no degenerate point, every non-square exponent lies in the strict interior of the Newton polytope. Therefore, each of them can be covered with a circuit that contains the origin $\mathbf{0}$. As a consequence, we can solve the constraints that occur for the constant terms of the circuit polynomials.
In the following, we describe and analyse our algorithm OPTSONC.

4.5.1 Algorithm. Assume, polynomial p has no degenerate points. The post-processing algorithm OPTSONC, to obtain a lower bound in exact arithmetic via SONC, based on a numerical solution proceeds as follows. The running time is polynomial in the *output* size.

Input:
$p = \sum_{\boldsymbol{\alpha} \in \boldsymbol{A}} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} \in \mathbb{Q}[\boldsymbol{x}]$,
$\widehat{\delta} \in \mathbb{Q}_{>0}$ -- rounding precision,
$\widetilde{\delta} \in \mathbb{Q}_{>0}$ -- precision parameter for the GP solver.
Output:
$\boldsymbol{X}$ -- matrix of rational numbers, coefficients of the decomposition,
$p.\mathsf{low} \in \mathbb{Q}$ -- lower bound of p on $\mathbb{R}^n$ in exact arithmetic.

```
1: function OPTSONC($p, \widehat{\delta}, \widetilde{\delta}$)
2:     $(\boldsymbol{\lambda}, \mathsf{C}) \leftarrow \texttt{cover}(p)$        ▷ exact solution of LP in Algorithm 4.1.3
3:     $\widetilde{\boldsymbol{X}} \leftarrow \texttt{SONC}\left(p, \widetilde{\delta}, \boldsymbol{\lambda}, \mathsf{C}\right)$        ▷ Solve (SONC) with accuracy $\widetilde{\delta}$
4:     $\widehat{\boldsymbol{X}} \leftarrow \texttt{round}\left(\widetilde{\boldsymbol{X}}, \widehat{\delta}\right)$        ▷ rounding step
5:     for $\boldsymbol{\alpha} \in \mathrm{MoSq}(p)$ and $\boldsymbol{\beta} \in \mathrm{NoSq}(p)$ do
6:         $X_{\boldsymbol{\beta},\boldsymbol{\alpha}} \leftarrow b_{\boldsymbol{\alpha}} \cdot \widehat{X}_{\boldsymbol{\beta},\boldsymbol{\alpha}} / \sum_{\boldsymbol{\beta'} \in \mathrm{NoSq}(p)} \widehat{X}_{\boldsymbol{\beta'},\boldsymbol{\alpha}}$        ▷ projection step
7:     for $\boldsymbol{\beta} \in \mathrm{NoSq}(p)$ do
8:         coeff $\leftarrow \lambda_{\boldsymbol{0}}^{\boldsymbol{\beta}} \cdot \left(-b_{\boldsymbol{\beta}} \cdot \prod_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} \left(\frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{X_{\boldsymbol{\beta},\boldsymbol{\alpha}}}\right)^{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}\right)^{\frac{1}{\lambda_{\boldsymbol{0}}^{\boldsymbol{\beta}}}}$
9:         $X_{\boldsymbol{\beta},\boldsymbol{0}} \leftarrow \texttt{round-up}\left(\mathrm{coeff}, \widehat{\delta}\right)$        ▷ adjust constant term
10:    $p.\mathsf{low} \leftarrow b_{\boldsymbol{0}} - \sum_{\boldsymbol{\beta} \in \mathrm{NoSq}(p)} X_{\boldsymbol{\beta},\boldsymbol{0}}$
11:    return $\boldsymbol{X}$, $p.\mathsf{low}$
```

Furthermore, if we want to achieve a multiplicative error of at most ε, then we can a priori compute $\widehat{\delta}$ and $\widetilde{\delta}$ accordingly, based on ε and the circuit cover, computed in Line 2.

Proof. In Line 3, the function `SONC` calls a GP solver, to solve (SONC) with accuracy $\widetilde{\delta}$, yielding a nearly feasible assignment $\widetilde{\boldsymbol{X}}$. This approximation is then rounded in Line 4 to a rational point $\widehat{\boldsymbol{X}}$ with a relative error of at most $\widehat{\delta}$. From this point on, we only use (exact) rational numbers and symbolic expressions. With the projection step from Line 6, we ensure that our equalities $\sum_{\boldsymbol{\beta}} X_{\boldsymbol{\beta},\boldsymbol{\alpha}} = b_{\boldsymbol{\alpha}}$ are satisfied for all $\boldsymbol{\alpha} \in \mathrm{MoSq}(p)$. In Line 9, we round the constant coefficient *up*, with relative error $\widehat{\delta}$, so that we have

$$X_{\boldsymbol{\beta},\boldsymbol{0}} \geq \lambda_{\boldsymbol{0}}^{\boldsymbol{\beta}} \cdot \left(-b_{\boldsymbol{\beta}} \cdot \prod_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} \left(\frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{X_{\boldsymbol{\beta},\boldsymbol{\alpha}}}\right)^{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}\right)^{\frac{1}{\lambda_{\boldsymbol{0}}^{\boldsymbol{\beta}}}}.$$

By Theorem 2.6.2, each polynomial

$$p_{\boldsymbol{\beta}} := \sum_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} X_{\boldsymbol{\beta},\boldsymbol{\alpha}} \cdot \boldsymbol{x}^{\boldsymbol{\alpha}} + b_{\beta} \boldsymbol{x}^{\boldsymbol{\beta}}$$

is a nonnegative circuit polynomial. Hence, C is a lower bound for p.
The vectors of the matrix $\boldsymbol{\lambda}$ are solutions of linear equation systems where the coefficients are chosen columns of $\boldsymbol{A}$ and thus are bounded by d. By Cramer's Rule, both numerator and denominator of the entries of $\boldsymbol{\lambda}$ are bounded by the maximal value of the determinant, which is given by

$$n! \cdot d^n \leq (dn)^n.$$

The encoding size of this number $n(\log d + \log n)$ is polynomial in the input size. As stated in Section 2.4.2, the problem in Line 3 can be solved in polynomial time, which also bounds the size of $\widetilde{\boldsymbol{X}}$. The crucial part is the rounding constant coefficient in Line 9, which we postpone for a moment.
Instead, we estimate the overall multiplicative error next. The overall error is the maximum of all errors $\varepsilon_{\boldsymbol{\beta}}$, occurring in Line 9. For simplicity, we consider higher order error terms to be negligible. So, for errors in the quotients, we use $(1-\varepsilon)^{-1} \approx 1+\varepsilon$. As we compute the coefficients $\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$ exactly, they do not contribute to the error. We denote

$$a \in b\ (1 \pm \varepsilon) :\Leftrightarrow b(1-\varepsilon) \leq a \leq b(1+\varepsilon)$$

for $a, b \in \mathbb{R}$ and extend this to elementwise comparison for vectors and matrices. Let $\boldsymbol{X}^*$ be the optimal solution. As we solve the GP with accuracy $\widetilde{\delta}$ we have $\widetilde{\boldsymbol{X}} \in \boldsymbol{X}^*(1 \pm \widetilde{\delta})$. Line 4 gives $\widehat{\boldsymbol{X}} \in \widetilde{\boldsymbol{X}}(1 \pm \widehat{\delta})$. Computing the projected, exact values in Line 6 yields

$$\frac{X_{\boldsymbol{\beta},\boldsymbol{\alpha}}}{\widehat{X}_{\boldsymbol{\beta},\boldsymbol{\alpha}}} = \frac{b_{\boldsymbol{\alpha}}}{\sum_{\boldsymbol{\beta}' \in \mathrm{NoSq}(p)} \widehat{X}_{\boldsymbol{\beta}',\boldsymbol{\alpha}}} \in \frac{b_{\boldsymbol{\alpha}}}{\sum_{\boldsymbol{\beta}' \in \mathrm{NoSq}(p)} \left(1 \pm \widehat{\delta}\right) X^*_{\boldsymbol{\beta}',\boldsymbol{\alpha}}} \approx \left(1 \pm \widehat{\delta}\right)$$

Considering the upper bound, this gives rise to the estimate of the overall error as

$$\begin{aligned} \frac{X_{\boldsymbol{\beta},\mathbf{0}}}{X^*_{\boldsymbol{\beta},\mathbf{0}}} &\leq \left(1+\widehat{\delta}\right) \cdot \prod_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} \left(\frac{X^*_{\boldsymbol{\beta},\boldsymbol{\alpha}}}{X_{\boldsymbol{\beta},\boldsymbol{\alpha}}}\right)^{\frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}}} \approx \left(1+\widehat{\delta}\right) \cdot \prod_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} \left(1 + 2\left(\widehat{\delta} + \widetilde{\delta}\right)\right)^{\frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}}} \\ &\approx 1 + \widehat{\delta} + \sum_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} \frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}} \cdot 2\left(\widehat{\delta} + \widetilde{\delta}\right). \end{aligned} \tag{Error}$$

So to achieve an error of ε, we can choose

$$\widehat{\delta} = \widetilde{\delta} = \varepsilon \left(4 \cdot \sum_{\boldsymbol{\alpha} \in \mathsf{C}^{\boldsymbol{\beta}}} \frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}}\right)^{-1}$$

which means we can a priori estimate the overall error.
As the values $\log \lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$ are polynomial in the input size, $\log \widehat{\delta}$ and $\log \widetilde{\delta}$ are as well. So by (Error) we can compute the relevant digits of the fractional part of $X_{\boldsymbol{\beta},\mathbf{0}}$ in time polynomial in the input size. The only problem is that the bit size of $X_{\boldsymbol{\beta},\mathbf{0}}$ contains the factor $\frac{\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}}$. So this bit size might be exponential in the input size, even when rounded to a full integer. But as this number is part of the output, the time for the other computations is polynomial in the input size, the overall running time is polynomial in the output size. □

Note that $\boldsymbol{\lambda}$ gives the convex combinations in the circuits, so $\lambda_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} \in (0,1)$ for all $\boldsymbol{\alpha} \in \mathrm{MoSq}\,(p)$ and $\boldsymbol{\beta} \in \mathrm{NoSq}\,(p)$. Our assumption that every circuit polynomial contains a constant term, is necessary to ensure that $\lambda_{\mathbf{0}}^{\boldsymbol{\beta}} \neq 0$ for all $\boldsymbol{\beta} \in \mathrm{NoSq}\,(p)$ in the computations above. However, as mentioned above, we can still encounter issues if some $\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}$ is very small. In this case, we cannot guarantee that the output of OPTSONC is polynomially bounded in the input size. But having a small value $\lambda_{\mathbf{0}}^{\boldsymbol{\beta}}$ means that the inner point of the circuit lies far away from the constant term. Therefore, the constant term influences the nonnegativity only with a small weight, hence it needs a high value. This happens in particular, when some inner point is close to a face of the newton polytope. So the instance is close to having a degenerate point. Regarding the condition of the problem, these instances would then be seen as ill-conditioned, hence also considered as problematic. For practical purposes, in Line 4, we computed rounded values such that their denominators are powers of two. Then the denominator of the sum in Line 6 is at most the maximal denominator occurring in $\widehat{\boldsymbol{X}}$. The function `round-up` can be computed via continued fractions, as they provide slightly smaller bit sizes. But the running time significantly increased, as it internally calls a larger number of roundings. Hence, we also round it to a fraction whose denominator is a power of two, i.e. we determine the binary digits according to our accuracy.

4.5.2 Symbolic Post-Processing for SAGE

Similarly to Algorithm 4.5.1, we design an algorithm, called OPTSAGE, which takes a given polynomial as input, obtains a numerical lower bound related to a SAGE decomposition computed via REP, and applies a post-processing to find a certified lower bound.

4.5.2 Algorithm. Assume, polynomial p has no degenerate points. The post-processing algorithm OPTSAGE, to obtain a lower bound in exact arithmetic via SAGE, based on a numerical solution proceeds as follows. The running time is polynomial in the *output* size, i.e. the size of the certificate.

```
Input:
    $p = \sum_{\boldsymbol{\alpha} \in A} b_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}} \in \mathbb{Q}[\boldsymbol{x}]$,
    $\widehat{\delta} \in \mathbb{Q}_{>0}$ -- rounding precision
    $\widetilde{\delta} \in \mathbb{Q}_{>0}$ -- precision parameter for the REP solver.
Output:
    $\boldsymbol{X}, \boldsymbol{\lambda} \in \mathbb{Q}^{t \times t}$ -- coefficients of the decomposition,
    $p.\mathsf{low} \in \mathbb{Q}$ -- certified lower bound of $p$ on $\mathbb{R}^n$.
1:  function OPTSAGE($p, \widehat{\delta}, \widetilde{\delta}$)
2:      $\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{\lambda}} \leftarrow \texttt{SAGE}\left(p, \widetilde{\delta}\right)$        ▷ Solve (SAGE) with accuracy $\widetilde{\delta}$
3:      $\widehat{\boldsymbol{X}} \leftarrow \texttt{round}\left(\widetilde{\boldsymbol{X}}, \widehat{\delta}\right)$        ▷ rounding step, elementwise
4:      for $k = 1, \ldots, t$ do
5:          LP $\leftarrow \left\{\boldsymbol{A} \cdot \boldsymbol{\lambda}^{(k)} = \boldsymbol{0}, \boldsymbol{1} \cdot \lambda^{(k)} = 0, \boldsymbol{\lambda}^{(k)}_{\setminus k} \geq \boldsymbol{0}, \|\boldsymbol{\lambda}^{(k)} - \widetilde{\boldsymbol{\lambda}}^{(k)}\|_\infty \leq \widehat{\delta}, \lambda^{(k)}_1 \geq \widehat{\delta}\right\}$
6:          $\boldsymbol{\lambda}^{(\boldsymbol{k})} \leftarrow$ some element from LP        ▷ projection step
7:          $\boldsymbol{X}^{(k)}_{\setminus k} \leftarrow \widehat{\boldsymbol{X}}^{(k)}_{\setminus k}$
8:          $X^{(k)}_k \leftarrow b_k - \boldsymbol{1} \cdot \boldsymbol{X}^{(k)}_{\setminus k}$
9:      for $k = 1, \ldots, t$ do
10:         $X^{(k)}_1 \leftarrow \texttt{round-up}\left(\left(\prod_{1<i\neq k}\left(\frac{\lambda^{(k)}_i}{eX^{(k)}_i} \cdot e^{-X^{(k)}_k}\right)^{\lambda^{(k)}_i}\right)^{\frac{1}{\lambda^{(k)}_1}} \cdot \frac{\lambda^{(k)}_1}{e}, \widehat{\delta}\right)$
11:     $p.\mathsf{low} \leftarrow b_1 - \sum_{j=1}^{t} X^{(j)}_1$
12:     return $\boldsymbol{X}$, $\boldsymbol{\lambda}$, $p.\mathsf{low}$
```

Opposed to Algorithm 4.5.1, the precision parameters are required for both additive and multiplicative error.

Proof. We start by numerically computing a lower bound for p with a corresponding SAGE certificate in Line 2. This can be done with precision $\widetilde{\delta}$, as REPs can be solved to arbitrary precisions, see Section 2.4.3. From this approximation, we round the coefficients $\widetilde{\boldsymbol{X}}$ to rational values $\widehat{\boldsymbol{X}}$ with a prescribed maximal error of $\widehat{\delta}$. For the matrix $\widetilde{\boldsymbol{\lambda}}$, the whole rounding is done in the LP (Lines 5 and 6). Next, we determine an exact feasible solution for this LP. Together with the assignment from Line 8, we ensure that we exactly satisfy the equality constraints

$$\boldsymbol{A} \cdot \boldsymbol{\lambda}^{(j)} = \boldsymbol{0} \qquad \boldsymbol{1} \cdot \lambda^{(j)} = 0 \qquad \text{for } 1 = j, \dots, t \qquad \sum_{j=1}^{t} \boldsymbol{X}^{(j)}_{\setminus 1} = \boldsymbol{b}_{\setminus 1}.$$

Similar to the assumption $\lambda_{\boldsymbol{0}}^{\boldsymbol{\beta}} > 0$ for SONC in Section 4.5.1, we require $\boldsymbol{\lambda}_1^{(j)} > 0$ to ensure that the computation in Line 10 is well-defined. Now, similar to SONC, for a large enough constant term, the relative entropy constraint

$$D\left(\boldsymbol{\lambda}^{(k)}_{\setminus k}, e\boldsymbol{X}^{(k)}_{\setminus k}\right) \leq X_k^{(k)} \tag{4.6}$$

is satisfied. Furthermore, in the optimal solution we must have equality, as $X_1^{(k)}$ does not occur in any other constraints and decreasing these coefficients of the constant term increases the left hand side. For $k = 2, \dots, t$, we solve (4.6) for $X_1^{(k)}$ and obtain

$$X_1^{(k)} = \left(\prod_{1 < i \neq k} \left(\frac{\lambda_i^{(k)}}{eX_i^{(k)}} \right)^{\lambda_i^{(k)}} \cdot e^{-X_k^{(k)}} \right)^{\frac{1}{\lambda_1^{(k)}}} \cdot \frac{\lambda_1^{(k)}}{e}.$$

By rounding up the symbolic expression, we assert that the constraint is still satisfied. Therefore, we have a SAGE decomposition with $\sum_{j=1}^{t} X_i^{(j)} = b_i$, for all $i > 1$ and $\sum_{j=1}^{t} X_1^{(j)} = b_1 - p.\mathsf{low}$, which certifies that $p \geq p.\mathsf{low}$ on $\mathbb{R}^n$. All parts, except Line 10 run in polynomial time. The bit size of $X_1^{(k)}$, however, might be exponential in the input size, but computing the value can be done in time polynomial in this bit size.
To estimate the overall multiplicative error, we assume that the respective parts are solved with the given bounds both as additive and multiplicative error. Similar to the proof of Algorithm 4.5.1, we ignore higher order error terms. More precisely, to estimate the error we use the approximations

$$(1+\varepsilon)^a \approx 1 + a\varepsilon, \qquad \frac{1}{1-\varepsilon} \approx 1 + \varepsilon, \qquad a^{\varepsilon} \approx 1 + \varepsilon \ln(a).$$

Let $\boldsymbol{\lambda}^*, \boldsymbol{X}^*$ denote the optimal solution. From the parameters, we get the estimates

$$|X_{k,i} - X^*_{k,i}| \leq \widehat{\delta} + \widetilde{\delta}, \qquad |\lambda_{k,i} - \lambda^*_{k,i}| \leq \widehat{\delta} + \widetilde{\delta} \qquad \text{for } 1 \neq i \neq k$$

and

$$|X_{k,k} - X^*_{k,k}| \leq t\left(\widetilde{\delta} + \widehat{\delta}\right), \qquad |\lambda_{k,k} - \lambda^*_{k,k}| \leq \widetilde{\delta} + \widehat{\delta} \qquad \text{for } k = 2, \dots, t.$$

Since $\widehat{\delta}$ and $\widetilde{\delta}$ only occur together in the error estimate, we unify them to $\delta := \widehat{\delta} + \widetilde{\delta}$. Then, we bound the overall multiplicative error

$$
\begin{aligned}
\frac{X_{k,1}}{X^*_{k,1}} &= \frac{\left(\prod_{1<i\neq k}\left(\frac{\lambda_{k,i}}{eX_{k,i}}\right)^{\lambda_{k,i}} \cdot e^{-X_{k,k}}\right)^{\frac{1}{\lambda_{k,1}}} \cdot \frac{\lambda_{k,1}}{e}}{\left(\prod_{1<i\neq k}\left(\frac{\lambda^*_{k,i}}{eX^*_{k,i}}\right)^{\lambda^*_{k,i}} \cdot e^{-X^*_{k,k}}\right)^{\frac{1}{\lambda^*_{k,1}}} \cdot \frac{\lambda^*_{k,1}}{e}} \\
&\leq \frac{\left(\prod_{1<i\neq k}\left(\frac{\lambda_{k,i}}{eX_{k,i}}\right)^{\lambda_{k,i}} \cdot e^{-X_{k,k}}\right)^{\frac{1}{\lambda_{k,1}}} \cdot \frac{\lambda_{k,1}}{e}}{\left(\prod_{1<i\neq k}\left(\frac{(1-\delta)\lambda^*_{k,i}}{e(1+\delta)X^*_{k,i}}\right)^{(1-\delta)\lambda^*_{k,i}} \cdot e^{-(1+t\delta)X^*_{k,k}}\right)^{\frac{1}{(1-\delta)\lambda^*_{k,1}}} \cdot \frac{(1-\delta)\lambda^*_{k,1}}{e}} \\
&\approx \left(\left(1 + 2\delta \cdot \sum_{1<i\neq k} \lambda_{k,i}\right) \prod_{1<i\neq k}\left(1 + \delta \ln \frac{\lambda_{k,i}}{eX_{k,i}}\right)(1+\delta)\right)^{\frac{1}{\lambda_{k,1}}} \\
&\quad \cdot \left(1 - t\delta X_{k,k} + \delta \cdot \sum_{1<i\neq k} \lambda_{k,i} \ln \frac{\lambda_{k,i}}{eX_{k,i}}\right)(1+\delta) \\
&\approx 1 + \delta\left(2\sum_{1<i\neq k}\lambda_{k,i} + \frac{1}{\lambda_{k,1}}\left(\sum_{1<i\neq k} \ln \frac{\lambda_{k,i}}{eX_{k,i}}\right) + \frac{1}{\lambda_{k,1}} + 1 - tX_{k,k} + \sum_{1<i\neq k} \lambda_{k,i} \ln \frac{\lambda_{k,i}}{eX_{k,i}}\right) \\
&\approx 1 + \delta\left(\frac{1}{\lambda_{k,1}}\left(3 - t - \lambda_{k,k} + \sum_{1<i\neq k} \ln \frac{\lambda_{k,i}}{X_{k,i}}\right) - tX_{k,k} + \sum_{1<i\neq k} \lambda_{k,i} \ln \frac{\lambda_{k,i}}{X_{k,i}}\right)
\end{aligned}
$$

Note, that $\lambda_{k,k}, X_{k,k} < 0$. As in Algorithm 4.5.1, we can thus compute the desired accuracy in polynomial time (in the input size), but the bit size of the number itself might be exponential in the input size. Hence, the overall time is polynomial in the output size. □

Opposed to Algorithm 4.5.1, we cannot a priori control this error, as $\boldsymbol{\lambda}$ and $\boldsymbol{X}$ are computed using δ, whereas for SONC the error depended only on $\boldsymbol{\lambda}$, which we computed exactly *before* choosing δ there.
For both OPTSONC and OPTSAGE we followed the idea that a slight perturbation in coefficients can be countered by a small adjustment of the constant terms in our decomposition. Therefore, our certificates in exact arithmetic will usually have a slightly worse lower bound than the numerical solution. Also, this adjustment is only possible, because we assume every circuit polynomial or AGE-polynomial in our decomposition involves the constant term. For a more detailed comparison see Section 5.5. We refer to Appendix A.1 for an example of an exact SAGE decomposition obtained with OPTSAGE.

Chapter 5

Experimental Comparisons

In this section, we discuss the actual physical running time, as well as the sizes of the certificates for the above algorithms. To this end, we generated a test set of 16923 instances. First, we describe the setup of our experiment and explain how our random instances were created. Then we go through each of the above sections and discuss a few selected examples which exhibit well the behaviour of the algorithms. Afterwards we shortly describe on which test cases we ran our implementation and in the end, we present how the programme behaved on this larger set of examples.

5.1 Experimental Setup

We give an overview of the experimental setup; in particular, which software and hardware we used.

Software The entire experiment was steered by our PYTHON 3.7 based software POEM (Effective Methods in Polynomial Optimisation) in the versions 0.2.1.0 [SdW19] and 0.3.0.0 [Sei21b], which we develop since July 2017. POEM is open source, under GNU public license, and available under the following URL.

`https://www.user.tu-berlin.de/henning.seidler/POEM/`

Version 0.2.1.0 implements Sections 4.1 and 4.5. Accordingly, we used it to obtain the experimental data we present in Sections 5.3 and 5.5. Version 0.3.0.0 extends the software by implementing Sections 4.2 to 4.4, the computation of minima and improving the bounds by regarding the signs of the variables. The corresponding experimental results we show in Section 5.4.

For our experiment, POEM calls a range of further software and solvers for computing both SONC, SAGE and SOS certificates.

In Python, we use CVXPY 1.0.24 [DB16, AVDB18], to model the convex optimisation problems. To solve the problems, we use ECOS 2.0.7 [DCB13], MOSEK 9 [ApS19] and CVXOPT 1.2.2 [AJV]. The symbolic computations were done in SYMPY 1.4 [JvMG12].

Since many established programmes for SOS are written in MATLAB, we also provide a simple MATLAB interface, together with direct calls to some of these packages. We use MATLAB R2017a and allow calls to YALMIP R20180209 [Löf04], GLOPTIPOLY 3 [HLL09] and SOSTOOLS 3.0.3 [PAV+13], the latter both with and without the "sparse" option, which attempts a reduction of the problem size via Theorem 2.5.3. Furthermore, we provide our own implementation of SONC and SOS, using CVX 2.1 [GB08]. For the solvers, behind these modellers, we use SDPT3 4.0 [TTT99] and SEDUMI 1.34 [Stu99]. However, after the first major experiment with POEM 0.1.0.0, we found out that Python/ECOS was significantly faster, so we aborted further development under MATLAB.

We also called SPARSEPOP 3.01 [WKK+08] on a couple of selected examples, but in each of these cases, YALMIP or SOSTOOLS was faster. Hence, we decided against running SPARSEPOP in full generality.

In addition, we attempted to solve the optimisation problem with SCS [OCPB17], but encountered several issues. The running times varied greatly, even when calling the same instance twice and SCS took significantly longer than ECOS for solving GPs. Furthermore, we often obtained results, that were actually infeasible. Thus, we decided not to use SCS in the experiment.

We observed that calling YALMIP via POEM and via MATLAB leads to slightly different running times. Similarly, we observed that calling YALMIP a second time both in POEM and in MATLAB can lead to reduced running times. In our large experiments, we call YALMIP via POEM once; for the selected examples in Section 5.3 we present the reduced running times given by a second call of YALMIP.

Investigated Data The experiment was carried out on a database containing 16923 polynomials with a wide range of variables, terms, degrees, and Newton polytopes. We created the database randomly using POEM. Further details can be found in Section 5.2; the full database of instances is available at the homepage cited above.

Hardware and System We used an `Intel Core i7-8550U` CPU with 1.8 GHz, 4 cores, 8 threads and 16 GB of RAM under Ubuntu 18.04 for our computations in Python. The computations in MATLAB were run on an `Intel Core i7-6700 CPU` with 3.4 GHz, 4 cores, 8 threads and 16 GB of RAM under openSUSE Leap 42.3.

Stopping Criteria For the accuracy of the solver and the precision of the rounding in PYTHON we use a tolerance of $\varepsilon = 2^{-23}$. For all computations in MATLAB we use the default accuracy $\varepsilon = 1.49\text{e-}8$ and 1.22e-4 for inaccurate solutions. To avoid memory errors, we call SOS only if the size of the matrix lies below a certain bound. For Python we check $\binom{n+d}{d} \leq 120$ and for MATLAB $\binom{n+d}{d} \leq 400$. If $400 < \binom{n+d}{d} \leq 1000$, we only call SOSTOOLS with the "sparse" option. To avoid excessive running times, we called SONC in MATLAB only if $t(n+1) < 3000$. We obtained all of these thresholds experimentally.

Running time and Memory The overall running time for all our instances was 29.9 days and created over 23 GB of data, mainly consisting of the certificates for SOS. Leaving out these certificates, and storing polynomials, call parameters, time and bound, we reduced this to 338 MB of data.

5.2 Generating Polynomials

When searching for existing benchmarks in unconstrained polynomial optimisation, we mostly found test cases of degree at most 6, e.g. in [WKK+08] (where the polynomials are even constructed as sums of squares). Presumably, this is because larger degrees are computationally intractable both with exact and SOS methods, already for moderate number of variables. Other papers only describe the general method how they generate their input, but do not provide the explicit instances [GM12, PS03]. Lastly, several collections regarded only integer problems or polynomial constraints. Hence, we decided to create our own benchmarks, whose generation we want to describe in further detail. Our instances are available on the POEM homepage, as well.
As parameters for the input size we define

n the number of variables, taking values $n = 2, 3, 4, 8, 10, 20, 30, 40$,

d the degree, taking values $d = 6, 8, 10, 20, 30, 40, 50, 60$,

t the number of monomials, taking values $t = 6, 9, 12, 20, 24, 30, 50, 100, 200, 300, 500$,

inner a lower bound for the number of summands whose exponent is not a vertex of the Newton polytope. In general, we use $\mathsf{inner} = \frac{k}{5}(t - n - 1)$ for $k = 1, \ldots, 4$. For the two special cases, where the Newton polytope is a simplex, this number is automatically given by $\mathsf{inner} = t - n - 1$.

The design goal was that within given parameters, the generated polynomial should have a significant probability to be bounded below, but otherwise be as general as possible. In particular, we do not impose any additional known structure on our polynomials and for given numbers, every polynomial with these parameters shall have a non-zero probability, to be constructed. One necessary condition to be bounded is that every vertex of the Newton polytope corresponds to a monomial square. Therefore, we first select points with even entries and then choose points from the interior of their convex hull. In the last step, we choose the coefficients of the polynomial, using a normal distribution. For vertices of the Newton polytope, we use the absolute value, to obtain strictly positive coefficients. Besides the above mentioned generality, we investigated two special cases of the Newton polytope, which were both easier to generate and easier to compute with.

Standard Simplex We have $\mathsf{Vert}(p) = \{\mathbf{0}, d\boldsymbol{e}_1, \ldots, d\boldsymbol{e}_n\}$ and $t - n - 1$ many elements in $\mathrm{int}(p)$, which we choose randomly using an even distribution over the lattice points of the interior of the scaled standard simplex.

This is similar to the approach chosen by Parrilo and Sturmfels in [PS03]. The interior corresponds to Δ^n_{d-n-1}, which we can easily enumerate. Hence, this case is especially simple to generate. Furthermore, when covering a non-square $\boldsymbol{\beta}$ with the vertices, we immediately get the convex combination $\lambda_j = \frac{\beta_j}{d}$ for $j = 1, \ldots, n$ and $\lambda_0 = 1 - \sum_{j=1}^n \lambda_j$. For SOS, on the one hand, these polynomials are particularly problematic, since Theorem 2.5.3 does not reduce the problem size at all. On the other hand, we have $p^*_{\mathrm{SONC}} \leq p^*_{\mathrm{SOS}}$, see Lemma 2.6.5. So, if the SDP can be solved, SOS will always yield a better bound than SONC.

Simplex We put $\boldsymbol{\alpha}_0 = \mathbf{0}$ and uniformly at random choose n points from the scaled standard simplex $\Delta^n_{d/2}$ and double their entries. These points form the vertex set $\mathsf{Vert}\,(p) = \{\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n\}$ and our initial support $A\,(p)$. Afterwards we choose $\boldsymbol{\lambda} \in [0,1]^{n+1}$ at random and normalise it to $|\boldsymbol{\lambda}|_1 = \sum_{i=0}^n \lambda_i = 1$. Then we compute $\boldsymbol{\beta}$ as elementwise rounding of $(\sum_{i=0}^n \lambda_i \boldsymbol{\alpha}_i)$ and check whether $\boldsymbol{\beta} \in \mathrm{int}(\mathrm{conv}\{\boldsymbol{\alpha}_0, \ldots, \boldsymbol{\alpha}_n\})$. If that is the case, then we add $\boldsymbol{\beta}$ to the set $A\,(p)$. We iterate this process until we reach $|A\,(p)\,| = t$ or some threshold of iterations (in this case, our generation fails).

General Case We put $\boldsymbol{\alpha}_0 = \mathbf{0}$ and uniformly at random choose $t - \mathsf{inner} - 1$ points from the scaled standard simplex $\Delta^n_{d/2}$ and double their entries. All further vertices are chosen as in the simplex case, only as convex combination of $\{\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{t-\mathsf{inner}-1}\}$ (and thus may fail as well, if it does not finish after a certain number of iterations). This way, we have ensured, that we have at least inner points in $A\,(p) \setminus \mathsf{Vert}\,(p)$.

Recall, if any vertex of the Newton polytope has a negative coefficient, then the polynomial is unbounded. Hence, we determine the convex hull or, more precisely, the extremal points of our set of chosen points, which is trivial in both simplex cases. For each of these we pick some random value from $\mathcal{N}\left(0, (t/n)^2\right)$, and take its absolute value as coefficient. For every other exponent vector, we choose a random value from $\mathcal{N}\,(0,1)$ as coefficient. We choose standard deviation $\frac{t}{n}$ to keep the lower bounds in a reasonable range.
For each combination of parameters (n, d, t) we ran the procedure with 10 different seeds and all three of the above shapes. For the general case, we additionally take $\mathsf{inner} = \frac{k}{5}(t - n - 1)$ for $k = 1, 2, 3, 4$. In the end, we created 16923 instances. Out of these, 501 are sums of monomial squares, which we consider trivial instances.
The current limit of our implementation lies in the range the random number generator can handle. In our setup, the methods failed, e.g. for $n = 40$ and $d = 60$. To pick evenly distributed vectors, we choose random numbers in the range of 0 to $\binom{40+30}{40}$, which becomes too large for some underlying data type to handle. The underlying optimisation methods can most likely handle even greater problem sizes.

5.3 SONC-Algorithm

In our initial experiment [SdW18], we provided implementations for SONC both in Python and in MATLAB. In that experiment, for every single instance, ECOS was the fastest method, unless it failed to find a solution at all (MOSEK was not able to solve GPs then). In this first experiment, failures occurred only in 195 instances, which is about 1.2% of the cases.
Since Python/ECOS dominated clearly in that first experiment, we decided to abandon further development in MATLAB at all, to reduce the overhead. Another major change since the first experiment is MOSEK 9 with its support for the exponential cone, i.e. GPs and REPs. So this solver is included in our experiments as well.
Before we present the outcome of the entire experiment of running Algorithm 4.1.7, we discuss a couple of chosen examples. In each table, "bound" denotes the lower bound p_{SONC} or p^*_{SOS}, according to the chosen strategy SONC or SOS respectively.
Since the polynomials discussed in the Examples 5.3.2 and 5.3.4 are very large, we do not explicitly state them in the thesis. All of the examples are available online at

`https://www.user.tu-berlin.de/henning.seidler/POEM/experimental_comparison_paper/`

5.3.1 Example (Improved bounds with Algorithm 4.1.7). We start with Example 5.5 from [DIdW19]:

$$p = 1 + 3 \cdot x_0^2 x_1^6 + 2 \cdot x_0^6 x_1^2 + 6 \cdot x_0^2 x_1^2 - x_0 x_1^2 - 2 \cdot x_0^2 x_1 - 3 \cdot x_0^3 x_1^3.$$

In this paper, the authors decompose the polynomial into several polynomials with simplex Newton polytope and put together their lower bounds. When evenly splitting the coefficients of the monomial squares over these polynomials, they achieve a lower bound of 0.5732 for p and a bound of 0.6583 with a different, arbitrarily chosen split. Using the same cover, as given in the paper, our Algorithm 4.1.7 yields the lower bound 0.693158. ⬡

5.3.2 Example (Range of running times of solvers). We consider a polynomial whose Newton polytope is the standard simplex with $n = 10$, $d = 30$, $t = 200$. Running time and results are shown in Table 5.1. All four solvers arrive at the same optimum, and Ecos and Mosek are significantly faster than the other two. Due to the problem size, we did not attempt to compute a bound using SOS. ⬡

language	strategy	solver	time (s)	bound
python	sonc	Ecos	1.22	-1109.45
python	sonc	Mosek	1.45	-1109.45
matlab	sonc	SeDuMi	27.24	-1109.45
matlab	sonc	SDPT3	153.09	-1109.45

Table 5.1: Standard simplex Newton polytope, $n = 10$, $d = 30$, $t = 200$

5.3.3 Example (Numerically stable via SONC but unstable via SOS). We consider the following randomly generated polynomial, whose Newton polytope is a non-standard simplex, with parameters $n = 5$, $d = 8$, $t = 10$.

$$\begin{aligned}
&4.8944034102934 + 1.2723031188849279 \cdot x_2^2 x_3^2 x_4^4 + 1.1845621900010301 \cdot x_1^2 x_3^6 \\
&+ 0.28148820906093597 \cdot x_0^2 x_2^4 x_4^2 + 2.237298866287131 \cdot x_0^2 x_1^4 x_4^2 + 0.5424841411187562 \cdot x_0^4 x_3^2 x_4^2 \\
&+ 0.3353216665073548 \cdot x_0 x_1 x_2 x_3 x_4^2 + 0.9042813721301197 \cdot x_0 x_1 x_2 x_3^3 x_4 \\
&- 2.1890122936443834 \cdot x_0 x_1 x_2 x_3 x_4 - 0.3902392990159973 \cdot x_0 x_1 x_2 x_3^2 x_4
\end{aligned}$$

The results of our computation are shown in Table 5.2. For SOS, numerical issues occur. Only Gloptipoly, Sostools and our own implementation with Sedumi return a solution. Furthermore, Gloptipoly's solution violates the constraints by more than $\varepsilon = 2^{-23}$, as denoted by -1 in the column "verify". Sostools returns an answer, which is far lower than the other two SOS bounds, likely caused by a numerical instability. These bounds, however, are all wrong, since there exists no finite SOS bound for this example, as certified, e.g. by Yalmip or Sostools with the sparse-flag. But the main observation is that SONC does not have any numerical issues and (therefore) yields a dramatically better bound than SOS, which we could obtain with three different solvers. Again, Ecos and Mosek are by far the fastest solvers. ⬡

language	strategy	modeller	solver	time (s)	bound	verify
matlab	sos	SOSTOOLS, sparse	SEDUMI	0.23	$-\infty$	-1
matlab	sos	SOSTOOLS	SEDUMI	7.94	-4633.91	1
matlab	sos	YALMIP	SEDUMI	0.76	$-\infty$	-1
matlab	sos	GLOPTIPOLY	SEDUMI	9.91	-68.4777	-1
matlab	sos	CVX	SEDUMI	19.70	-52.1625	1
matlab	sos	CVX	SDPT3	18.62	nan	-1
python	sos	CVXPY	CVXOPT	998.47	$-\infty$	-1
python	sos	CVXPY	MOSEK	515.23	$-\infty$	-1
matlab	sonc	CVX	SDPT3	2.12	4.24914	1
matlab	sonc	CVX	SEDUMI	0.99	4.24914	1
python	sonc	CVXPY	ECOS	0.08	4.24914	1
python	sonc	CVXPY	MOSEK	0.13	4.24914	1

Table 5.2: The Newton polytope is a simplex; parameters: $n = 5$, $d = 8$, $t = 10$.

5.3.4 Example (A polynomial with combinatorially challenging Newton polytope). The dwarfed cube is a polytope, that exhibits problematic behaviour from a combinatorial point of view, in the sense, that it causes many convex hull algorithms to perform badly [ABS97]. So we want to see, how our algorithms behave on a polynomial whose Newton polytope is the dwarfed cube. We scale the dwarfed cube by 4 and choose a full support for the polynomial in the sense that every lattice point of the polytope occurs in the support.
For SONC, we performed our computations with two different coverings and obtained slightly different lower bounds. Additionally, we computed a lower bound via SAGE. Solving the REP with ECOS failed, but MOSEK computed a solution, which is slightly better than the bounds we obtained via SONC, but it also takes significantly longer.
We show the results in Table 5.3. The bounds p^*_{SOS} and p_{SONC} differ by less than 1%, but again SONC with ECOS and MOSEK are the fastest methods, this time by about 45 % compared to the fastest SOS-method YALMIP. Furthermore, p^*_{SOS} is the actual lower bound, since we attain this value. ○

5.3.1 Evaluation of the Experiment

Now, we summarise our findings from running Algorithm 4.1.7, to compute lower bounds via SONC, on our data set. We focus on the running time with respect to our different parameters, and compare it with SOS.

The running time of SONC is independent of the degree As stated in the beginning, our input has size $\mathcal{O}(nt)$ in the unit cost model, so it only depends on the number of variables and terms. Hence, for a polynomial time algorithm, the running time may not increase with the degree d. That this holds in examples, was already observed in [DIdW19] (see also further reference there in). We provide a more formal proof in Theorem 4.1.9. However, this left open the possibility, that the degree influences the number of iterations in the LPs or GP. Here, we confirm experimentally that the running time for SONC is practically not affected by the degree. For our test cases with $n = 4$ and $t = 20$, we show the average running time

language	strategy	modeller	solver	time (s)	bound
matlab	sos	Gloptipoly	SeDuMi	8.76	28.3181
matlab	sos	Yalmip	SeDuMi	1.62	28.3181
matlab	sos	Sostools, sparse	SeDuMi	7.98	28.3181
matlab	sos	Sostools	SeDuMi	8.27	28.3181
matlab	sos	Cvx	SeDuMi	17.69	28.3181
matlab	sos	Cvx	SDPT3	11.87	28.3181
python	sos	Cvxpy	Cvxopt	390.89	28.3181
python	sos	Cvxpy	Mosek	152.64	28.3181
matlab	sonc	Cvx	SeDuMi	4.16	28.246
python	sonc	Cvxpy	Ecos	0.62	28.2779
python	sonc	Cvxpy	Ecos	0.91	28.2811
python	sonc	Cvxpy	Mosek	0.87	28.2811
python	sonc	Cvxpy	Mosek	0.46	28.2779
python	sage	Cvxpy	Ecos	4.81	$-\infty$
python	sage	Cvxpy	Mosek	4.10	28.2832

Table 5.3: Example 5.3.4: Polynomial supported over the full 7-dimensional dwarfed cube, scaled by a factor 4, as support. Corresponding parameters: $n = 7$, $d = 6$, $t = 113$, $\mathsf{inner} = 63$

.

degree	time	instances	degree	time	instances
58	0.080	13	60	0.085	25
38	0.081	12	20	0.088	39
50	0.083	39	10	0.094	40
40	0.083	25	6	0.122	32
28	0.084	6	8	0.128	38
30	0.085	34			

Table 5.4: Timing for $n = 4$, $t = 20$, ordered by time. We see that the running time is independent of the degree.

of Algorithm 4.1.7, together with the number of instances Table 5.4. Hence, for the following observations, we mainly regard dependencies on the number of variables and terms, and in some cases the shape of the Newton polytope.

SONC behaviour on the standard simplex The scaled standard simplex is the corresponding Newton polytope in the common approach to investigate (dense) polynomials in n variables of degree d. The results for running our algorithm on our instances with this Newton polytope are given in Table 5.5. We present the running time, depending on the number of variables and on the number of terms. Even for the largest cases, we can solve the problem in a few seconds. Note that we only consider instances with $d > n + 1$, since otherwise the interior of the Newton polytope contains no integer points.

Handling degenerate terms In the preliminaries, we introduced degenerate terms, noting that they can cause difficulties. We went into further detail, how they may

$t \setminus n$	2	3	4	8	10	20	30	40
6	0.043	0.047	0.048	-	-	-	-	-
9	0.058	0.057	0.061	-	-	-	-	-
12	0.070	0.070	0.065	0.072	0.075	-	-	-
20	0.096	0.099	0.095	0.091	0.099	-	-	-
24	0.108	0.113	0.103	0.102	0.119	0.145	-	-
30	0.137	0.134	0.123	0.122	0.133	0.173	-	-
50	0.179	0.197	0.191	0.184	0.200	0.244	0.283	0.314
100	0.352	0.324	0.367	0.374	0.439	0.511	0.570	0.615
200	0.950	1.578	1.184	1.300	1.293	1.408	1.500	1.570
300	2.761	5.587	3.313	3.678	3.665	3.995	3.910	4.332
500	10.503	12.100	12.831	13.665	13.725	13.962	14.168	14.825

Table 5.5: Average running time for SONC, where the Newton polytope is the standard simplex; depending on number of terms t and number of variables n. A "-" indicates that we have no such instance.

affect the algorithm in Section 4.1.4. Now we want to estimate, how often these problems occur. In our whole data set, 2415 instances contain degenerate terms. For 37 of these, we could certify, that they actually are unbounded by Lemma 2.1.4. In 2322 cases, which means about 96.1%, we could compute a bound via SONC, despite the presence of degenerate terms. Hence, for practical purposes, we conclude experimentally, that degenerate terms *might* cause problems for SONC, but in the clear majority of cases we still can compute a bound.

Dependence on the number of terms Here, we study an example case, how the number of terms influences the running times on SOS and SONC. We regard our instances with $n = 3$ and $d = 20$, and for each number of terms, we evaluate the average running time, as shown in figure 5.6. Most prominently, the running time of SONC displays a roughly quadratic growth, from 40.8ms for $t = 6$ up to 12.1s for $t = 500$. In contrast, SOS requires about 20s, even for few terms, and converges to a value around 40s for a higher number of terms. The figure also shows, that SOS already exploits sparsity, since polynomials with few terms usually have smaller Newton polytopes.

Qualitative running time comparison of SOS and SONC We are highly interested in comparing the actual physical running times of SOS and SONC, in relation to our chosen parameters. However, as we have seen, the time for SONC does not depend on the degree, whereas the running time for SOS is largely independent of the number of terms. To compare both, we therefore chose the following approach.

For each pair (n, d), we consider the largest value t, such that we have at least 10 instances with parameters (n, d, t). As we have several methods using SOS, we select the lowest time among the successful methods for each instance. For SONC, we select the running time of the variant, which gives the better bound. Note that for large degrees we can also expect large Newton polytopes and hence can have more terms in the instances. Thus, we consider our setup to be disadvantageous

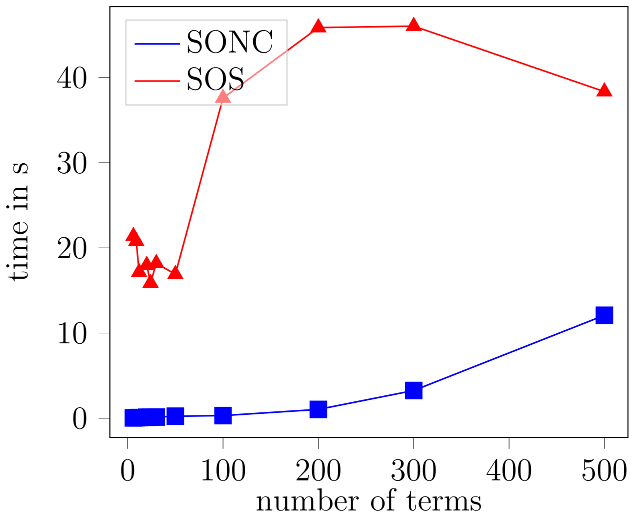

Figure 5.6: Average running time of SONC and SOS for instances with $n = 3$ and $d = 20$, depending on the number of terms.

for SONC especially in these cases. Under these choices, we present the average running times in Table 5.7. The upper entry shows the running time for SOS, while the lower entry is the time for SONC. Most notably, in every cell, SONC is the faster method. Already for 8 variables in degree 6 we observe a drastic increase in the running time of SOS. Additionally, the memory requirement for SOS grows extremely quickly. Therefore, we introduced some thresholds, depending on the method. With Cvxpy and Mosek/Cvxopt, we attempt it only, if the dimension of the psd-matrix is at most 120. In Matlab, we run all SOS methods up to a matrix of size 400×400. Additionally, for matrices of dimension up to 1000, we call Sostools with the "sparse" option. During our experiments we observed that problems beyond these bounds regularly lead to memory errors, because they require more than 16GB RAM. In our table, we mark these problem sizes with a "$\times$". Even if they do not exceed the memory, we easily have running times of several hours. Thus, we did not attempt to call SOS in these instances in our large test. The "-" in the table mean, we do not have at least 10 instances for these parameters.

Overall, we make the following two main observations:

- We can solve far larger instances with SONC, than we could solve with SOS.
- Even in the parameter range, where SOS was able to compute a solution, SONC usually is faster.

Quantitative comparison of SONC and SOS Here, we investigate the differences of the lower bounds and the ratios of the running time obtained via SOS and SONC. Similar to the previous case, for each instance and each of the two strategies, we use the best bound for the comparison. If several methods produced this bound, then we use the time of the fastest among these. In case some strategy failed, we treat the result as lower bound $-\infty$ and time ∞. We present the result in figure 5.8.

Overall, among our 16422 non-trivial instances, we found a SONC bound for 16122

$d \setminus n$	2	3	4	8	10	20	30	40
6	**0.075**	**0.152**	**0.209**	**0.368**	**1.980**	-	-	-
	0.194	0.338	0.343	54.371	1380.688			
8	**0.104**	**0.146**	**0.204**	**0.373**	**0.352**	-	-	-
	0.240	0.449	0.697	$\times$	$\times$			
10	**0.149**	**0.223**	**0.337**	**0.343**	**0.362**	**10.534**	-	-
	0.292	0.801	4.374	$\times$	$\times$	$\times$		
20	**0.342**	**12.085**	**12.417**	**13.510**	**13.789**	-	-	-
	1.066	37.961	$\times$	$\times$	$\times$			
30	**2.809**	**11.727**	**13.005**	**13.465**	**19.226**	**12.206**	-	-
	4.446	1249.596	$\times$	$\times$	$\times$	$\times$		
40	**10.582**	**12.177**	**12.882**	**22.170**	**21.900**	**15.800**	**13.616**	-
	17.475	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	
50	**10.462**	**11.360**	**12.830**	**16.581**	**13.449**	**11.619**	**15.067**	**16.038**
	46.075	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
60	**10.465**	**12.298**	**12.772**	**22.970**	**14.638**	**12.792**	**16.187**	**14.572**
	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

Table 5.7: Average running time of the fastest SOS method (normal font) compared to SONC (bold), depending on degree d and number of variables n. Note that for higher degree, we generally have more terms. A "$\times$" indicates, that solving was not attempted, to avoid memory errors.

of them, which is about 98.2%. There was no instance among our test cases, where SOS found a bound while SONC did not. This stands in stark contrast to 10993 instances, where SONC found a lower bound but SOS failed, or would have exceeded the memory, see Section 5.1.

In the left graphic of figure 5.8 we list the differences between the optima, obtained by SONC and SOS. When both algorithms found a bound, then in most cases SOS found the better bound. However, in 1078 cases we got similar bounds, differing by at most 0.001 and in 115 cases the bound obtained via SONC is better by more than 0.001. Of the 4578 instances, where SOS found a better bound than SONC, 1021 instances (22.3%) have the scaled standard simplex as Newton polytope. Since the scaled standard simplex is an H-simplex, we have by construction that $p_{\text{SONC}} \leq p^*_{\text{SOS}}$ in these cases, see [IdW16a, Section 5].

When comparing the running time, we see a clear advantage for SONC. An overview of the ratios of the running times between SOS and SONC is shown in the right graphic of figure 5.8. Considering only the examples, where both algorithms found a solution, the median of the running times is 8.4. However, there are instances with a speedup of more than 10,000, and the largest ratio is around 251,000.

In addition, there are 501 instances, which are sums of monomial squares and 238 instances, where both algorithms failed. Out of these at least 37 instances are unbounded. For 51 further instances, we managed to obtain a lower bounds via some of the other methods. For the remaining 150 the status is unknown, since checking whether a polynomial is bounded is coNP-hard on its own, see Theorem 3.3.1.

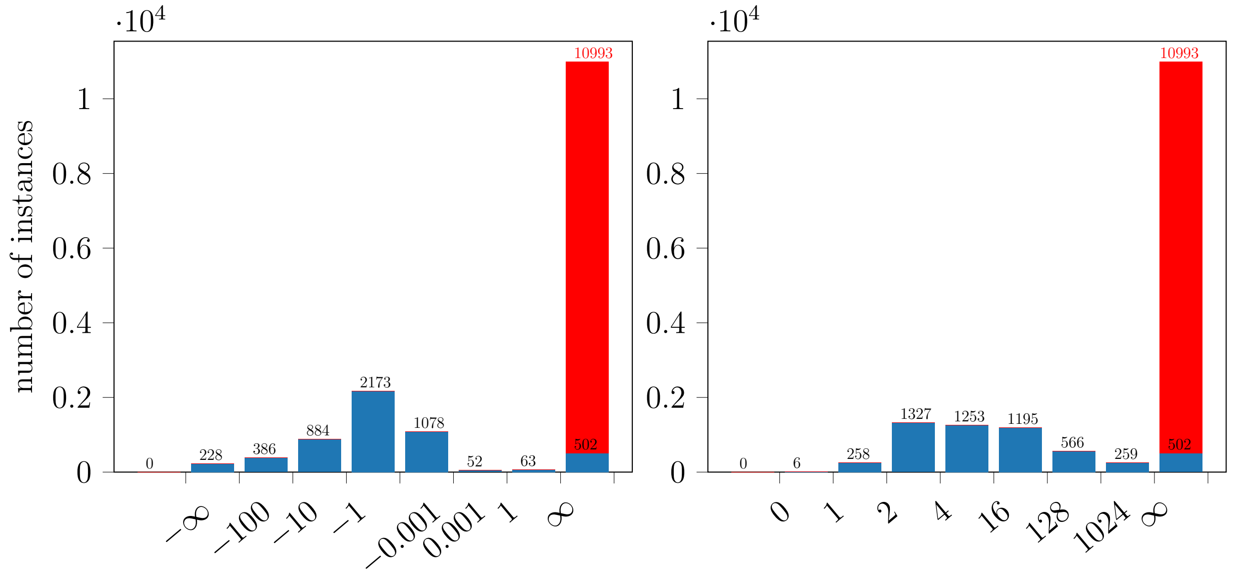

Figure 5.8: Number of instances where (left) the difference of the bounds $p^*_{\text{SOS}} - p_{\text{SONC}}$ lies in the interval given on the lower axis; (right) the ratio SOS / SONC of the running time lies in the interval given on the lower axis. The upper red bar represents instances, where the problem size for SOS lies above our threshold. Note, the labelling is asymmetric to better show the distribution.

Considering the gaps between the bounds for SOS and SONC, we would like to stress that this is only an early approach to obtain lower bounds via SONC. Further research has already been done, e.g. [Pap19]. Hence, we can expect that the results will significantly shift in favour of SONC.

5.4 Branch-and-Bound

As in the previous section, we start with discussing a few selected examples in detail. Afterwards, we summarise our findings from running the algorithm on a larger scale.

Stopping Criteria As before, we use the numerical accuracy $\varepsilon = 2^{-23}$. But now the running time grows exponentially in the number of variables n. Also, we observed that SAGE takes a significant amount of time for $t \geq 100$. Therefore, we run the large scale experiment only with polynomials having $n \leq 8$ and $t \leq 50$. Both thresholds were obtained experimentally.

5.4.1 Example. We consider a polynomial with $n = 4$ variables, degree $d = 16$ and $t = 50$ terms. Here we particularly see, how the branch-and-bound approach significantly improved the bound. The lowest value we found, is $\mathsf{p}_{\mathsf{min}} \approx 19.203$. The lower bounds we obtained, are given in Table 5.9.
The best bound was found by TRAVERSE. The approach by FORK failed, since for at least one of the orthants, both SONC and SAGE encountered numerical issues. Since the

overall bound is given by the worst bound on any of the orthants, we only obtain the trivial bound $-\infty$.
Next, we observe, that the sparse version of TRAVERSE here actually takes *longer* than the standard version. Both methods compute 23 out of 31 possible nodes of the search tree. So the sparse method does not have any advantage.
Furthermore, it computes a *worse* bound. The reason for the latter is, that we branch the variables in a different order. At some point, SAGE runs into numerical problems, and these issues arise at different nodes in the search tree. The remaining bounds are then computed with the weaker, but more stable SONC method. Therefore, the two versions of TRAVERSE can have different results. ○

lower bound	gap	time	strategy	options
11.992	7.211	0.17	SONC	cover via Algorithm 4.1.3
13.693	5.510	0.19	SONC	cover via Algorithm 4.1.4
14.458	4.745	3.09	SAGE	
18.769	0.434	53.98	TRAVERSE	
18.284	0.918	56.43	TRAVERSE,	sparse
$-\infty$	∞	2.26	FORK,	SONC only
$-\infty$	∞	9.97	FORK,	SONC and SAGE

Table 5.9: Comparison of time and quality of the lower bounds obtained by different approaches. Branch-and-bound yields the best bound. The forking algorithm numerically failed on some orthant, so its bound is $-\infty$.

5.4.2 Example. The next example is a polynomial with $n = 4$ variables, degree $d = 10$ and $t = 30$ terms, where we show the results in Table 5.10. Here, TRAVERSE-sparse computes the optimal bound $\mathsf{p}_{\mathsf{min}} \approx 4.08$ within half a minute, while all of our other methods have an optimality gap of at least 0.27. But it also shows, that the parallelised method FORK computed a good bound in significantly less time. ○

Together, Examples 5.4.1 and 5.4.2 show that TRAVERSE and TRAVERSE-sparse are in general incomparable both in their time requirement and in terms of their results.
Finally, we want to present an example, how the parallelised code behaves on a server

lower bound	gap	time	strategy	options
-58.80	62.88	0.12	SONC	cover via Algorithm 4.1.3
-51.84	55.92	0.20	SONC	cover via Algorithm 4.1.4
-25.65	29.73	1.46	SAGE	
3.8029	0.277	31.98	TRAVERSE	
4.0798	0	35.84	TRAVERSE,	sparse
0.3987	3.681	1.39	FORK,	SONC only
3.8029	0.277	4.29	FORK,	SONC and SAGE

Table 5.10: Comparison of time and quality of the lower bounds obtained by different approaches. Here, the sparse version of TRAVERSE computed the best bound but also took the longest time. All of our other methods computed worse bounds.

with many processors. We used a machine with `Intel Xeon CPU E7-4870 @ 2.40GHz` with 40 cores, 80 threads and 500 GB of RAM. The example polynomial has 8 variables, 50 terms and degree 6.[1] We show the results in Table 5.11. Out of the 256 orthants, 120 are minimal, so we only perform our computation on these. On each orthant, we attempt both cover strategies, so from the time on the positive orthant, we easily get a first estimate for the total running time, which is 88.8 seconds on the PC and 219.6 seconds on the server. To test these estimates, we ran FORK without parallelisation and found out, that the time was actually lower. The speedup due to parallelisation is 3.35 for the PC and 26.36 for the server. So increasing the number of cores by a factor 10 increased the speedup by a factor of nearly 8.

strategy	options	time PC	time Server	bound
SONC	cover via Algorithm 4.1.3	0.33	0.88	-3.499
SONC	cover via Algorithm 4.1.4	0.41	0.95	-311.435
FORK	SONC only	25.06	6.03	-0.068
FORK, single-core	SONC only	83.93	158.98	-0.068

Table 5.11: Comparing running time under high parallelisation. The server has slower single core performance, but 10 times the number of cores. The first two lines show the results for the positive orthant. The third line displays the time for the parallelised algorithm. For comparison, we also included the timing for serial computation.

5.4.1 Evaluation of the Experiment

In this section, we summarise our findings from running our experiment on 9639 instances.

The bound of Traverse is at least as good as the bound of Fork. Figure 5.12 shows the difference between the lower bounds obtained by TRAVERSE and FORK, where a positive value means that TRAVERSE gave a better bound. In the majority of cases the difference is numerically zero, but in some cases, TRAVERSE performs significantly better. In no case the differences goes below -10^{-5}.

Sparse Traverse is slightly faster than standard Traverse. The quotients of the running time for standard TRAVERSE divided by the time of sparse TRAVERSE range from 0.056 to 172.01 with a geometric mean of 1.141. So, on average, the standard version takes about 14.1% longer. The distribution of these quotients is shown with more detail in figure 5.13. In particular, in the majority of our cases, the running times differ by a factor of at most 2.

Failure of Fork is rare. As seen in Example 5.4.1, FORK may return the trivial bound $-\infty$. However, this only rarely happends. Among our test cases, there are only 102 instances, where FORK fails, but (at least one variant of) TRAVERSE finds a lower bound.

Running time of Traverse and Fork, depending on n, t: As expected, the running time increases with both n and t. In Table 5.14 and Table 5.15, we present the

[1]It is the polynomial with ID 13782 in our data base.

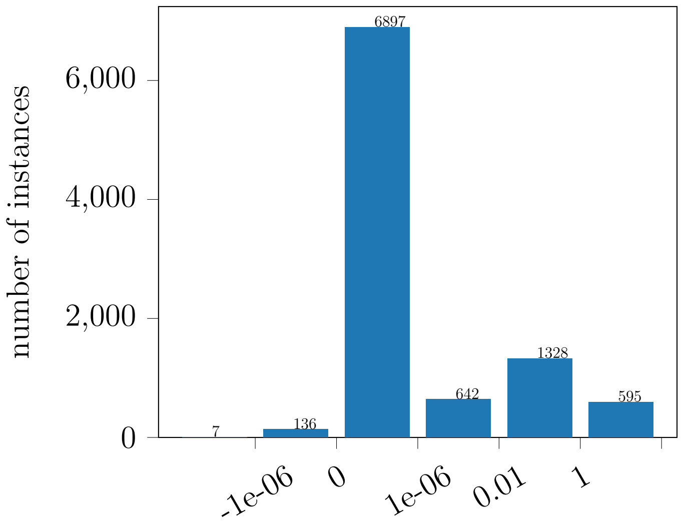

Figure 5.12: Difference between the bounds of TRAVERSE and FORK. Practically, the result of TRAVERSE is *always* as least as good as the one of FORK. In the majority of our examples the difference is numerically zero.

running times of TRAVERSE and FORK, depending on n and t. Since we observed in Section 2.7 and [SdW18], that the running time is independent of the degree, we average over the degree as well. We can see, that the growth in n is slower than $\Theta(2^n)$. Also note, that we have at least $n + 1$ monomial squares, so for few terms the ratio of non-squares decreases with growing n. Thus, in these cases, the running time even *decreases* with growing n.

Optimality Gap So far, we only compared the lower bounds with each other. As Algorithm 4.3.7 also computes local minima, we obtain an optimality gap and can discuss the quality of these lower bounds. In figure 5.16, we can see the distribution of the optimality gap among our instances, for how many instances the gap lies in the given interval. For the left bars (blue), we combined all of our new methods and took their best bound. The middle bars (orange) show the distribution of the optimality gap, when just using SAGE. Finally, we display the optimality gap for SONC in the third column (red). Similar to earlier experiments, we ran both of our variants, if feasible, and picked the better bound.

SAGE shows a higher bar in the rightmost column, which is due to a higher number of instances where the method failed, but SONC succeeded. Otherwise, the optimality gaps for SAGE are lower than for SONC. This trend continues with our branch-and-bound methods, where we see a clear shift to the left in the numbers of instances.

For 6438 instances, about 66.8% of our test cases, our improved methods yield a gap of at most 10^{-6}, which we consider numerically zero. Furthermore, we see a clear improvement compared to using only a single call of SAGE or SONC.

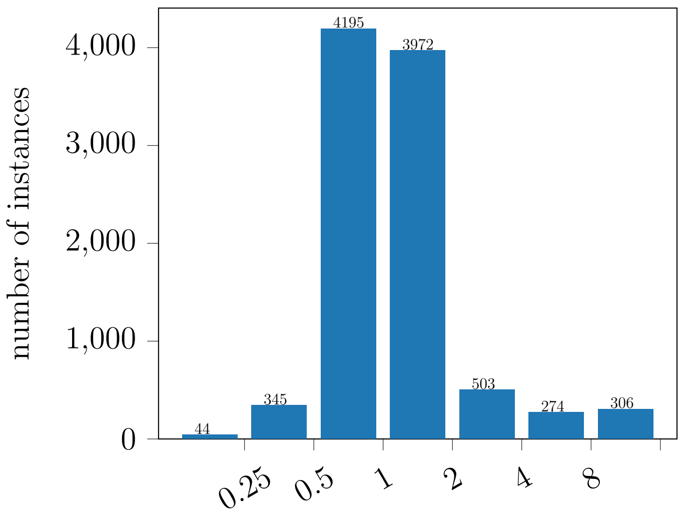

Figure 5.13: Quotient of running times of standard TRAVERSE divided by sparse TRAVERSE. In most cases the times differ by a factor of at most 2, but overall the sparse version has a slight advantage.

$n \setminus t$	6	9	12	20	24	30	50
2	2.53	3.33	4.98	6.79	7.72	9.41	14.44
3	1.55	1.93	2.73	5.27	7.15	10.18	22.38
4	2.97	2.71	3.83	8.96	11.63	16.05	37.45
7	×	0.03*	2.01*	50.84*	×	×	×
8	×	0.02	11.46	37.12	63.72	111.77	333.31

Table 5.14: Average running time of standard TRAVERSE, depending on n and t. A (*) marks parameters, where we have less than 10 instances.

5.5 Lower Bounds in Exact Arithmetic

Here, we describe the experimental evaluation of OPTSONC and OPTSAGE procedures, given by Algorithm 4.5.1 and Algorithm 4.5.2, as also presented in [MSdW19]. Again, we run these algorithms on a large set of our example instances. As we no longer get results in floating point arithmetic, we are also interested in the bit size of the solutions, besides the running time.

Stopping Criteria As before, we use the numerical accuracy $\varepsilon = 2^{-23}$. Additionally, we restrict ourselves to $t \leq 50$, since otherwise we already encounter problems in the numerical solution of (SAGE). In our previous article [MSdW19], we also had to impose a limit on the degree, because we encountered computational problems for $d \geq 30$. In the current version we solved these issues, so we dropped the restriction on the degree.

Exact LP solvers In contrast to the previous experiments, we do not only use numerical solver but also require solvers in exact arithmetic when we determine the coefficients $\boldsymbol{\lambda}$ within the circuits. To solve the occurring LP exactly, we use cdd [Fuk21] or SoPlex from the SCIP Optimisation Suite [BBC+21].

$n \setminus t$	6	9	12	20	24	30	50
2	0.22	0.31	0.42	0.76	0.98	1.40	3.33
3	0.21	0.30	0.44	1.00	1.39	2.06	5.45
4	0.18	0.28	0.42	1.49	2.15	3.48	9.48
7	$\times$	0.16*	0.31*	1.48*	$\times$	$\times$	$\times$
8	$\times$	0.06	0.28	1.01	5.05	18.39	95.76

Table 5.15: Average running time of FORK with SAGE, depending on n and t. A (*) marks parameters, where we have less than 10 instances.

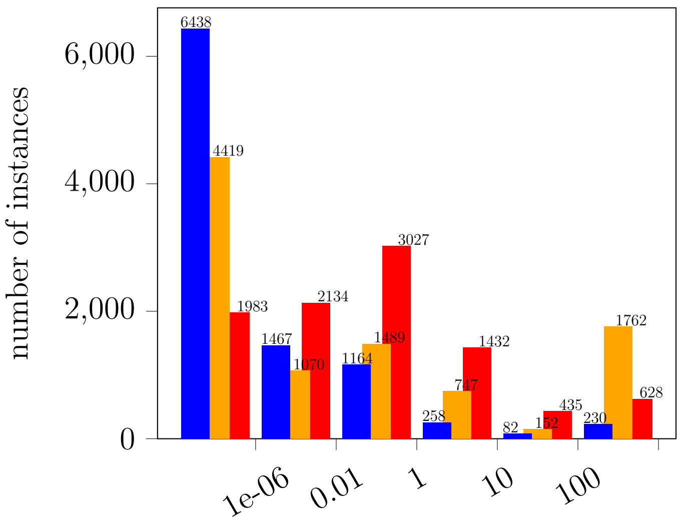

Figure 5.16: Distribution of the optimality gap, restricted to the instances where we called branch-and-bound; Left, in blue, we show the optimality gap for the combined methods standard-TRAVERSE, sparse-TRAVERSE and FORK with SAGE, taking their best bound. The middle bars, in orange, show the gap if we just use SAGE to compute a global lower bound. On the right, in red, we show the gap obtained with SONC.

In total, we ran 12867 examples in SAGE and 8460 examples for SONC. Both combined, our time for this part was 13.3 hours.

5.5.1 Evaluation of the Experiment

In this section we present and evaluate the results of our experiment and highlight our most important findings, when investigating the computational data. We focus on the results given by the procedure OPTSAGE (Algorithm 4.5.2) via SAGE decompositions and in the end give a comparison to OPTSONC.

Running time may *decrease* with growing number of variables. To test the influence of the number of variables, we increased this amount while all other parameters remained the same. The formulation of (SAGE) shows that the size of the problem only depends on the number of terms t, but in the SAGE decomposition, the number of AGE-summands is the number of monomial non-squares. In our setting, we assure that our instances always have full dimension and the vertices of

$d = 8$, $t = 20$

n	bit size	time
2	27882	3.86
3	23439	3.18
4	24912	3.44
8	9278	0.89
10	2166	0.31

$d = 10$, $t = 24$

n	bit size	time
2	37698	5.40
3	35092	4.94
4	30977	4.23
8	8302	1.10
10	3778	0.91

$d = 10$, $t = 30$

n	bit size	time
2	61969	9.73
3	57678	8.29
4	56139	7.79
8	13343	2.55
10	8507	1.13

Table 5.17: Dependency of the average bit size and the average running time of OPTSAGE, with the number of variables, for fixed values of degree d and number of terms t; For $n = 8$ we observe a drastic drop both in running time and bit size.

the Newton polytope are monomial squares. So, increasing the number of variables (with the same number of terms) reduces the number of non-squares, which in turn reduces the complexity of the problem.

Additionally, for $n \geq 8$ and $d \leq 10$, most exponents lie on faces of the Newton polytope and these faces include the origin. This leads to a simpler combinatorial structure, which we believe to result in lower bit sizes. Thus, we can solve the exact LPs from Line 2 of OPTSAGE and Line 6 in OPTSAGE faster. Next, we have more equality constraints in these LPs, which could also improve the running time. For some selected parameters, we exhibit that behaviour in Table 5.17.

Dependency of bit size and running time of degree and terms To illustrate how bit size and running time of OPTSAGE vary for different degrees and numbers of terms, we restrict ourselves to at most 4 variables. Our numbers from the previous point show, that in these cases bit size and time are similar for fixed (d, t), hence we may aggregate those instances. The results are shown in Table 5.18. We can see that running time and bit size roughly have a linear dependency. On the one hand, their growth is quadratic in the number of terms, which matches with the growth of the problem size in (SAGE). On the other hand, bit size and running time are basically unaffected by the degree. This shows that the bound $n(\log d + \log n)$, given in the worst case analysis in Algorithm 4.5.2, usually is not met.

Quality of the rounding-projection Our experiments verify that in the majority of cases the exact lower bound does not diverge far from the numerical bound. This holds both for SAGE and for SONC. The detailed distribution is shown in figure 5.19. There, we display for how many of our test cases the difference between numerical and exact lower bounds lie in the given interval, where the left bars in red show the number for SAGE and the right bars in blue show the numbers for SONC.

Most notably, in 159 instances of SAGE and 2443 instances of SONC, the exact lower bound is even *better* than the numerical bound. Within the other intervals, SONC and SAGE show a similar behaviour. In 77.6% of the instances, the exact bound differs by at most 0.001 from the numerical value. Only in 3613 instances the difference lies above 10. Thus, in the clear majority of examples, the lower bound in exact arithmetic does not differ much from the numerical bound.

Also, among the instances with large difference the numerical solution might actually

$t \setminus d$	6	8	10	16	18	20	24	26	28	30	32	34
6	885	991	1019	1148	1135	1052	1426	1017	888	×	1149	1293
	0.15	0.16	0.17	0.14	0.17	0.18	0.17	0.17	0.15		0.14	0.16
9	2927	2753	2858	3415	2658	2570	2756	3274	2306	×	3100	4054
	0.44	0.43	0.45	0.42	0.43	0.40	0.32	0.50	0.42		0.34	0.43
12	6177	6106	5744	8353	5687	5462	×	5955	4687	×	6631	7700
	0.91	0.87	0.84	0.89	0.82	0.80		0.90	0.72		0.70	0.82
20	×	24538	21740	×	21864	20431	×	24870	18969	×	24531	×
		3.35	2.91		2.79	2.71		3.37	2.55		2.40	
24	×	36847	33645	×	37382	29117	×	39462	27679	25010	×	33199
		5.64	4.69		5.38	3.89		5.59	3.33	2.19		3.35
30	×	57833	57372	×	62058	49160	×	60257	55492	34507	×	×
		8.20	8.20		9.05	6.49		7.91	7.48	3.28		
50	×	×	181062	×	×	143416	×	×	×	130294	×	169361
			25.87			18.43				12.69		17.63

$t \setminus d$	36	38	40	42	44	46	48	50	54	58	60
6	1242	863	1054	947	1161	981	909	1054	×	×	1115
	0.15	0.10	0.13	0.12	0.14	0.12	0.11	0.13			0.15
9	3742	2488	2968	3226	3220	2430	2646	3026	2524	2642	3360
	0.41	0.27	0.33	0.36	0.38	0.27	0.30	0.34	0.31	0.30	0.40
12	7186	5695	5645	7316	6844	5547	6005	5718	6053	5794	6887
	0.75	0.58	0.60	0.80	0.72	0.59	0.64	0.62	0.66	0.67	0.74
20	23355	20090	18572	×	23099	23433	23303	18860	×	19030	18957
	2.36	1.94	1.88		2.34	2.26	2.43	1.93		2.01	2.06
24	×	34434	27686	×	×	35150	33172	26295	×	31659	29596
		3.23	2.76			3.45	3.41	2.70		3.45	3.06
30	62010	58173	46274	×	×	×	×	44938	×	×	51228
	5.86	5.49	4.49					4.56			5.28
50	×	167099	142848	×	×	×	×	144628	×	×	128831
		17.03	14.74					14.91			13.59

Table 5.18: Bit size (upper part) and running time (lower part) of OPTSAGE in dependency of the degree d and the number of terms t for up to 4 variables; A "×" indicates, that we only have few or no instances with these parameters in our data set.

be slightly inaccurate. Even violating constraints within the bound ε can have a significantly larger impact on the lower bound. So it is unclear, whether a large difference is due to bad behaviour of the numerical solution, or a large error in the rounding algorithm.

Rounding time versus solving time In nearly every test case the post-processing in exact arithmetic takes longer than the numerical solving. Only in 41 instances, the rounding procedure was faster. The ratio of the rounding time to the total time ranges from 21.6% to 96.8%, with an average of 83.3%. However, one can implement this post-processing much closer to the hardware level, instead of working in Python. Thus, we expect that these ratios can be significantly improved. This would also lead to a large speedup in the overall running time.

Comparison between SONC and SAGE In their qualitative behaviour, OPTSONC and OPTSAGE are similar. However, as in the purely numerical case, OPTSONC runs faster and has smaller certificates than OPTSAGE, as shown in Table 5.20. But one should keep in mind that OPTSONC only computes *some* lower bound (not

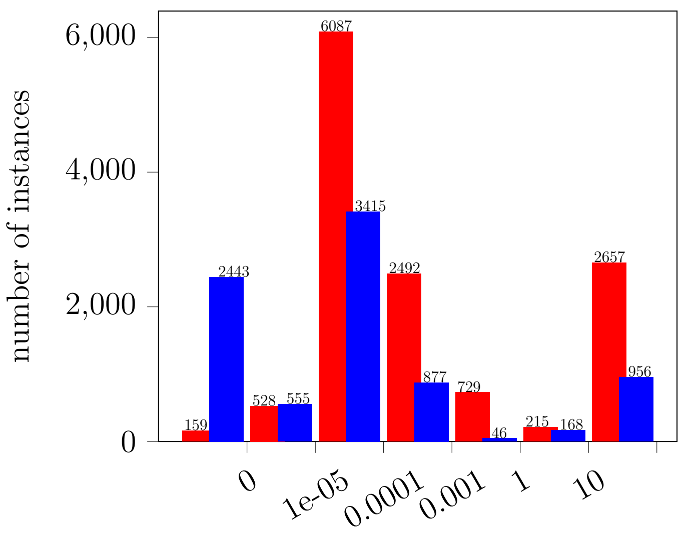

Figure 5.19: Number of instances where the difference of numerical lower bound and exact lower bound lies in the given interval; note that the exact bound sometimes is better. The left bar (red) show values for SAGE, while the right bar (blue) display the numbers for SONC.

terms	bits SONC	bits SAGE	time SONC	time SAGE
6	438	1022	0.09	0.15
9	830	2853	0.33	0.40
12	1302	5906	0.74	0.76
20	2866	20931	2.05	2.49
24	3561	30909	2.54	3.66
30	4979	52669	3.14	6.24
50	9236	156874	8.17	17.97

Table 5.20: Comparison of running time and bit size of the certificates between OPTSONC and OPTSAGE; OPTSONC runs faster and has significantly smaller certificates than OPTSAGE.

necessarily the optimal SONC-bound), whereas OPTSAGE computes the best bound that can be obtained via this approach. Still it shows that for very large instances SONC is the method of choice when other approaches fail due to the problem size.

Comparison with SOS For some polynomials lying in the interior of the SOS cone from [MSED21, Table 2], we performed preliminary experiments with OPTSONC and OPTSAGE, which are currently unable to provide nonnegativity certificates. For benchmarks from our database with $n \geq 8$ and $d \geq 10$ we ran REALCERTIFY [MSED18] which is a programme to provide SOS certificates in exact arithmetic. However, that method failed to solve our test cases. Hence, also for certificates in exact arithmetic, SOS and SONC/SAGE remain incomparable approaches.

Chapter 6

Conclusion and Outlook

We separately summarise for the basic SONC algorithm, the improvement via branch-and-bound and of the post-processing in exact arithmetic. At the end, we give an outlook on possible further developments, both in research and in possible extensions of the algorithms and the software.
The long-term goal was to provide a new method for constrained polynomial optimisation, using sums of nonnegative circuit polynomials. Currently, we can conclude that this objective is still beyond reach. But already the problem to efficiently find a *global* lower bound of a multivariate polynomial provides a fruitful field of research.

6.1 Conclusion on the SONC Algorithm

To our knowledge, POEM was the first programme, to compute lower bounds for sparse polynomials in polynomial time. In contrast, the widely researched method via sums of squares (SOS) requires exponential space to solve. This comes from the exponential blow-up, when translating the problem into a semi-definite programme. Even under the view of parametrised complexity, SOS only lies in the class XP when parametrised by either the degree or the number of variables. In practice, one is restricted to degree two, or degree four with a moderate number of variables.
Sums of nonnegative circuit polynomials have the huge advantage, that they only work on the support of the given polynomial. Furthermore, the degree of the input only affects numerical values and not the size of the underlying Geometric Programme (GP). Hence, SONC is especially well suited for polynomials of higher degree with relatively few terms. A similar approach is taken with sums of arithmetic-geometric-mean exponentials (SAGE). The optimal SAGE decomposition can be obtained by solving a Relative Entropy Programme (REP). As theoretical model, both SONC and SAGE describe the same cone of nonnegative polynomial. In contrast, our algorithms only compute *some* SONC decomposition. Hence, in theory, the bound given via SAGE will always be at least as good as the bound computed with SONC. But the running time of SAGE grows faster than for SONC, which makes SAGE practically infeasible for larger input sizes, where SONC still performs well.
We then went to test these observations in practice. We took some well-established implementations for SOS, and provided our own implementations for all three methods. Using these, we ran a large-scale experiment on thousands of test cases with a far range of

degrees, variables and numbers of terms to perform a detailed comparison of the practical behaviour of these methods. Based on these experiments, we draw the following key conclusions:

1. The running time for SONC certificates depends on the number of variables and the cardinality of the support only. It does not depend on the degree.

2. Among the tested GP solvers, Ecos and Mosek 9 are *by far* the best ones for computing SONC certificates. Between those two, Mosek is slightly faster and can solve a few more instances, whereas Ecos provides a completely open source software.

3. SONC certificates can be computed extremely fast. Even for very large instances, the running time remains in the realm of seconds. In a direct comparison of those cases where both SONC and SOS yield a bound, SOS is faster than SONC in only 0.1% of our cases.

4. For those test cases, where we ran both SOS and SONC, SOS yields better bounds than the current SONC algorithm in the clear majority (91.9%) of our cases. However, often the bounds do not differ very much, and there is a nontrivial amount of cases, when the SONC bound is better. See figure 5.8 for details.

5. SONC is widely applicable. Even for the majority of instances with degenerate terms, a potential source of problems for SONC, we were able to compute bounds. Moreover it uses vastly less memory than SOS. While in our experiments (for 16 GB RAM) SOS certificates can be computed up to corresponding SDPs with matrices of size roughly 400×400 (these are, e.g. instances of type $n = 5$, $d = 12$), SONC easily handles cases of 40 variables and 500 terms and arbitrary degree.

6. SAGE, despite its polynomial running time, takes much longer and fails to solve our larger instances. For 16 GB RAM, the limit lies between 50 and 100 terms. In contrast, SONC even allows some degree of control over the running time, by choosing the amount of circuits involved in the solution. In most cases, a number that is linear in the number of terms already gives good bounds.

6.2 Conclusion on the Branch-and-Bound Method

Both SONC and SAGE in their original form only work on the positive orthant. As a first relaxation, every term that is not a monomial square is assumed to be negative, which can significantly worsen the lower bound. So we propose a case distinction on the signs of the variables. Thus, additional terms can be identified as positive, which improves the lower bounds obtained via SONC and SAGE. We present two different approaches, how this can be achieved.
The first method is a branch-and-bound framework, where we branch over the signs of the variables one after the other. This is combined with the computation of local minima so that we also obtain an optimality gap. As an alternative, we regard all 2^n orthants and compute those that are minimal with respect to the signs of the terms of the given

polynomial. For each of these minimal orthants, we run SONC or SAGE. The primary advantage of this method is that it can easily be run in parallel.
In terms of theoretical complexity, both variants have an exponential running time in classical complexity. However, they are fixed parameter tractable when parametrised by the number of variables, so the problem is still practically feasible for moderate sizes.
We ran these methods on a larger number of test cases and draw the following conclusions.

1. To obtain the best result, the method of choice is TRAVERSE. However, between standard-TRAVERSE and sparse-TRAVERSE there is no clear favourite, which approach computes a better bound. Only with respect to the running time, there is a slight advantage in using the sparse version.

2. Especially for few terms, FORK runs significantly faster than TRAVERSE. This speedup partially comes from parallel computations, but also from our preprocessing, so we run fewer instances of SONC and SAGE.

3. Computing the minimal orthants for FORK runs fast, i.e. negligible compared to actually solving instances of SONC/SAGE. So it is possible to a priori get an estimate of the running time of FORK. If the gain from eliminating orthant is too small, we can simply run TRAVERSE instead.

In summary, making a case distinction on the signs of the variables significantly improves the quality of the lower bounds that can be computed via SONC and SAGE.

6.3 Conclusion on Symbolic Post-Processing

We propose a post-processing method which, under some additional assumption, transforms the numerical solutions of our previous methods into lower bounds in exact arithmetic. This method we run on a large number of our test cases. Based on these experiments we draw the following conclusions.

1. In the majority of cases, the exact solution lies close to the numerical solution with a difference of at most 0.001. For few instances the exact lower bound is even better than the numerical one.

2. When we compute a certified lower bound with OPTSONC we can a priori determine the multiplicative error after computing the circuit cover. In a lot of cases the certified bound of OPTSONC is even better than the numeric bound.

3. For OPTSAGE the running time and the bit size grow quadratically in the number of terms, which corresponds to the growth of the problem size.

4. For the investigated parameters increasing the degree or the number of variables does not increase the running time or the bit size. This also corresponds to the fact that the size of the REP is independent of the degree and the number of variables.

5. For very large instances SONC should be the first choice to obtain a certified bound since it runs significantly faster than the other methods. Furthermore, if we desire to achieve some given error bound, we may have to call multiple iterations of OPTSAGE, while for OPTSONC a single iteration suffices.

In total, we see this post-processing as a proof-of-concept which requires significant additional work. The primary issue is the necessary assumption that there are no degenerate points. For SONC this assumptions is even extended in that every circuit in our covering must contain the origin, i.e. every circuit polynomial must contain the constant term. Another issue which has to be addressed is the time required for the post-processing.
The incomparability to SOS is harder to solve. As SOS naturally is only used for small degree polynomials, these instances tend to have lots of degenerate points. So, both approaches will likely coexist for different classes of instances.

6.4 Outlook

As SONC is a rather young approach to certify nonnegativity of multivariate polynomial, there naturally are numerous ways for improvement. But also nonnegativity of polynomials in general has several open problems.

- The exact complexity classes of deciding nonnegativity and deciding the existence of a lower bound are unknown. Can we show ETR-hardness or containment in coNP for NonNeg? Can we reduce Bounded to NonNeg?
- Are there other simple, sufficient criteria to certify that a multivariate polynomial is (un)bounded?
- What is the complexity to decide membership in SONC (exactly, not numerically)?
- Is there a polynomial time algorithm to check nonnegativity for circuit polynomials?

SONC algorithm A tremendous progress was done by Dávid Papp in [Pap19] which improved our approach to find both the optimal circuits and the optimal distribution of the negative coefficients. Its only downside is the running time that grows cubic in the number of terms opposed to the quadratic growth we observed for our algorithm.

- Can we improve the quality of the bounds for our algorithm, while keeping the running time? In particular, how time-consuming is it to improve the choice of circuits and the distribution of the negative coefficients?
- Our heuristic for the local minimum in general has a starting point in the positive orthant which might not be optimal. Do we get better results by flipping the signs of some of the entries? If so, at which indices?
- When determining the circuit cover, we solve multiple linear programmes that all have the same constraints. Can we exploit this?
- Can we algorithmically find and replace redundant circuits in a given circuit decomposition?

Branch-and-Bound We want to emphasise, that both Traverse and Fork are just frameworks, which use SONC and SAGE. So any improvement for these, which could be quality of results, running time or numerical stability, results in an improvement for our algorithms presented in Section 4.3. Again, we see potential for improvement both on the theoretical side and in the implementation.

- Most importantly, can we find better cut criteria in the branch-and-bound approach? These might significantly improve the running time of our algorithm.
- What influence has the order in which we branch the variables? How can we find a good choice or even the best choice?
- Currently, Fork has the advantage of parallelisation. But the nodes in the search tree can be computed independently (unless one is an ancestor of the other). So we could run Traverse partially in parallel as well.
- Finding the minimal orthants primarily consists of bitwise operations, so it can easily be implemented in C for further speedup.
- Currently, failure of a single orthant in Fork causes the overall approach to fail. These orthants correspond to leaves in the search tree of Traverse. By moving up in the search tree and computing lower bounds for these nodes, we still obtain lower bounds for the original orthant.
- Different sign constraints on the variables can still lead to the same partition into positive and negative terms. So, if we encounter some partition a second time, we do not have to solve it anew. While this detection in the worst case needs exponential running time, we expect it to be practically faster than solving an additional instance of SAGE/SONC.

Exact Arithmetic Here, the main issue to overcome is the requirement not to have degenerate points. However, we can address this issue in a way similar to Section 4.1.5. The key observation is that reducing some coefficient in a circuit can be cancelled out by increasing another coefficient of the same circuit.

- We initially apply our post-processing method on each circuit polynomial of the numeric decomposition. Then, we identify exponents $\boldsymbol{\alpha}$ where we violate the constraint $\sum_C X_{C,\boldsymbol{\alpha}} \leq b_{\boldsymbol{\alpha}}$. We decrease some $X_{C,\boldsymbol{\alpha}}$ and increase some $X_{C,\boldsymbol{\alpha}'}$ for well-chosen $C, \boldsymbol{\alpha}$. If $\boldsymbol{\alpha}$ lies "closer" to the origin in some metric, then this method terminates and yields a certificate in exact arithmetic.
- Allowing degenerate points might also allow for better comparability with SOS certificates in exact arithmetic.
- For a number of instances we encountered computational problems when calling optsonc or optsage. So we would like to increase the robustness of our implementation.
- Regarding optsage, we remarked that we could have used the pseudo-inverse to obtain the projected values $\boldsymbol{\lambda} \in \mathbb{Q}^{t \times t}$. It would be interesting to see how this step compares to using the LP.

Software POEM The software does not include the latest research on SONC and SAGE, yet. Most prominently, it is missing Papp's algorithm to compute the optimal SONC decomposition. Also, POEM barely runs any parts in parallel so it can potentially be improved in this aspect.

Constrained Optimisation Finally, the whole framework we presented in this work only applies for unconstrained optimisation. The long-term goal behind our work was to find new methods for constrained optimisation. While there is a Positivstellensatz and a converging hierarchy for SONC, it is practically not applicable, yet, due to both the size of the search space and the size of the certificate. Further research needs to be done to efficiently compute the lower levels of the hierarchy.
Regarding our post-processing for SONC/SAGE certificates into exact arithmetic the situation is far better. Certifying nonnegativity over some semi-algebraic set requires certificates for global nonnegativity as subprocedures. Hence, we expect this framework to also generalise to constrained optimisation.
In total, we see SONC and SAGE as a fruitful field that allows a multitude of further research and we would like to see further growth of our software POEM.

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[ABS97] D. Avis, D. Bremner, and R. Seidel. How Good are Convex Hull Algorithms? *Computational Geometry*, 7(5-6):265--301, 1997.

[AJV] M. Andersen, J.Dahl, and L. Vandenberghe. CVXOPT - Python Software for Convex Optimization. `cvxopt.org`.

[AM19] A. A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebra and Geometry*, 3(2):193--230, 2019.

[ApS19] M. ApS. *MOSEK Optimizer API for Python 9.0.98*, 2019.

[Art27] E. Artin. Über die Zerlegung definiter Funktionen in Quadrate. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, volume 5, pages 100--115. Springer, 1927.

[AVDB18] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A Rewriting System for Convex Optimization Problems. *Journal of Control and Decision*, 5(1):42--60, 2018.

[BBC$^+$21] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig. The scip optimization suite 8.0. Technical Report 21-41, ZIB, Takustr. 7, 14195 Berlin, 2021.

[BCR13] J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry*, volume 36. Springer, 2013.

[Ble06] G. Blekherman. There are significantly more nonnegative polynomials than sums of squares. *Israel Journal of Mathematics*, 153(1):355--380, 2006.

[BPR06] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*, volume 10. Springer, 2006.

[BPT13] G. Blekherman, P. Parrilo, and R. Thomas. *Semidefinite Optimization and Convex Algebraic Geometry*, volume 13 of *MOS-SIAM Series on Optimization*. SIAM and the Mathematical Optimization Society, Philadelphia, 2013.

[BR93] M. Bellare and P. Rogaway. The Complexity of Approximating a Nonlinear Program. In *Complexity of Numerical Optimization*, pages 16--32. World Scientific, 1993.

[BV04] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004.

[CHJL11] S. Chevillard, J. Harrison, M. Joldeş, and C. Lauter. Efficient and accurate computation of upper bounds of approximation errors. *Theoretical Computer Science*, 412(16):1523--1543, 2011.

[Col75] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20--23, 1975*, pages 134--183. Springer, 1975.

[Coo06] S. Cook. The P versus NP problem. *The millennium prize problems*, pages 87--104, 2006.

[CS16] V. Chandrasekaran and P. Shah. Relative Entropy Relaxations for Signomial Optimization. *SIAM J. Optim.*, 26(2):1147--1173, 2016.

[CS17] V. Chandrasekaran and P. Shah. Relative Entropy Optimization and its Applications. *Mathematical Programming*, 161(1-2):1--32, 2017.

[Dav53] M. Davis. Arithmetical problems and recursively enumerable predicates 1. *The journal of symbolic logic*, 18(1):33--41, 1953.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research*, 17(83):1--5, 2016.

[DCB13] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071--3076, 2013.

[Del84] C. N. Delzell. A continuous, constructive solution to Hilbert's 17th problem. *Inventiones mathematicae*, 76(3):365--384, 1984.

[DIdW17] M. Dressler, S. Iliman, and T. de Wolff. A Positivstellensatz for Sums of Nonnegative Circuit Polynomials. *SIAM Journal of Applied Algebra Geometry*, 1(1):536--555, 2017.

[DIdW19] M. Dressler, S. Iliman, and T. de Wolff. An Approach to Constrained Polynomial Optimization via Nonnegative Circuit Polynomials and Geometric Programming. *Journal of Symbolic Computation*, 91:149--172, 2019.

[DPR61] M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, 74(3):425--436, 1961.

[FdW19] J. Forsgård and T. de Wolff. The Algebraic Boundary of the SONC Cone. *arXiv preprint arXiv:1905.04776*, 2019.

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[FR64] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149--154, 1964.

[Fuk21] K. Fukuda. *cddlib Reference Manual*, February 2021.

[Gas12] W. I. Gasarch. Guest column: The second P=? NP poll. *ACM SIGACT News*, 43(2):53--77, 2012.

[GB08] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95--110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/papers/graph_dcp.html`.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[GM12] M. Ghasemi and M. Marshall. Lower bounds for polynomials using geometric programming. *SIAM Journal on Optimization*, 22(2):460--473, 2012.

[GM13] M. Ghasemi and M. Marshall. Lower Bounds for a Polynomial on a Basic Closed Semialgebraic Set using Geometric Programming. *arXiv preprint arXiv:1311.3726*, 2013. Preprint, arxiv:1311.3726.

[HAB$^+$17] T. Hales, M. Adams, G. Bauer, D. T. Dat, J. Harrison, H. L. Truong, C. Kaliszyk, V. Magron, S. Mclaughlin, N. T. Thang, N. Q. Truong, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, T. T. H. An, T. N. Trung, T. T. Diep, J. Urban, V. K. Ky, and R. Zumkeller. A Formal Proof of the Kepler Conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[Hil88] D. Hilbert. Über die Darstellung definiter Formen als Summe von Formenquadraten. *Mathematische Annalen*, 32(3):342--350, 1888.

[HLL09] D. Henrion, J.-B. Lasserre, and J. Löfberg. GloptiPoly 3: moments, optimization and semidefinite programming. *Optimization Methods & Software*, 24(4-5):761--779, 2009.

[IdW16a] S. Iliman and T. de Wolff. Amoebas, nonnegative polynomials and sums of squares supported on circuits. *Research in the Mathematical Sciences*, 3, 2016.

[IdW16b] S. Iliman and T. de Wolff. Lower Bounds for Polynomials with Simplex Newton Polytopes Based on Geometric Programming. *SIAM Journal of Optimization*, 26(2):1128--1146, 2016.

[JvMG12] D. Joyner, O. Čertík, A. Meurer, and B. E. Granger. Open Source Computer Algebra Systems: SymPy. *ACM Commun. Comput. Algebra*, 45(3/4):225--234, January 2012.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302--311. ACM, 1984.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093--1096, 1979.

[Las01] J.-B. Lasserre. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization*, 11(3):796--817, 2001.

[Las10] J. Lasserre. *Moments, positive polynomials and their applications*, volume 1 of *Imperial College Press Optimization Series*. Imperial College Press, London, 2010.

[Las15] J. Lasserre. *An introduction to polynomial and semi-algebraic optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 2015.

[Lau09] M. Laurent. Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry*, volume 149 of *IMA Volumes in Mathematics and its Applications*, pages 157--270. Springer, New York, 2009.

[Löf04] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284--289. IEEE, 2004.

[MAGW15] V. Magron, X. Allamigeon, S. Gaubert, and B. Werner. Formal proofs for Nonlinear Optimization. *Journal of Formalized Reasoning*, 8(1):1--24, 2015.

[Mar30] E. Marczewski. Sur l'extension de l'ordre partiel. *Fundamenta Mathematicae*, 16:386--389, 1930.

[Mar08] M. Marshall. *Positive Polynomials and Sums of Squares.* Number 146. American Mathematical Soc., 2008.

[Mat70] Y. V. Matiyasevich. The Diophantineness of enumerable sets. In *Doklady Akademii Nauk*, volume 191, pages 279--282. Russian Academy of Sciences, 1970.

[MCW21] R. Murray, V. Chandrasekaran, and A. Wierman. Newton polytopes and relative entropy optimization. *Foundations of Computational Mathematics*, pages 1--35, 2021.

[MK87] K. G. Murty and S. N. Kabadi. Some NP-complete Problems in Quadratic and Nonlinear Programming. *Mathematical Programming*, 39(2):117--129, 1987.

[MSdW19] V. Magron, H. Seidler, and T. de Wolff. Exact Optimization via Sums of Nonnegative Circuits and Sums of Arithmetic-geometric-mean-exponentials. pages 291--298, 2019.

[MSED18] V. Magron and M. Safey El Din. RealCertify: a Maple package for certifying non-negativity. *ACM Communications in Computer Algebra*, 52(2):34--37, 2018.

[MSED21] V. Magron and M. Safey El Din. On Exact Polya, Hilbert-Artin and Putinar's Representations. *Journal of Symbolic Computation*, 2021.

[Mül18] H. Müller. Minima of Polynomials via SONC Decompositions. Bachelor's thesis, TU Berlin, Berlin, October 2018.

[Nes00] Y. Nesterov. Squared Functional Systems and Optimization Problems. In *High performance optimization*, pages 405--440. Springer, 2000.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming.* Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

[OCPB17] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting Conic Solver, version 2.0.2. `https://github.com/cvxgrp/scs`, November 2017.

[Oxl11] J. Oxley. *Matroid theory*, volume 21 of *Oxford Graduate Texts in Mathematics.* Oxford University Press, Oxford, second edition, 2011.

[Pap03] C. H. Papadimitriou. *Computational Complexity.* John Wiley and Sons Ltd., 2003.

[Pap19] D. Papp. Duality of sum of nonnegative circuit polynomials and optimal SONC bounds. *arXiv preprint arXiv:1912.04718*, 2019.

[Par00] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization.* PhD thesis, California Inst. Tech., 2000.

[PAV$^+$13] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB.* `http://arxiv.org/abs/1310.4716`, 2013. Available from `http://www.eng.ox.ac.uk/control/sostools`, `http://www.cds.caltech.edu/sostools` and `http://www.mit.edu/~parrilo/sostools`.

[Pfi67] A. Pfister. Zur Darstellung definiter Funktionen als Summe von Quadraten. *Inventiones mathematicae*, 4(4):229--237, 1967.

[PP08] H. Peyrl and P. Parrilo. Computing sum of squares decompositions with rational coefficients. *Theoretical Computer Science*, 409(2):269--281, 2008.

[PS03] P. A. Parrilo and B. Sturmfels. Minimizing Polynomial Functions. *Algorithmic and quantitative real algebraic geometry, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 60:83--99, 2003.

[PY19] D. Papp and S. Yildiz. Sum-of-Squares Optimization without Semidefinite Programming. *SIAM Journal on Optimization*, 29(1):822--851, 2019.

[Ren88] J. Renegar. A faster PSPACE algorithm for deciding the existential theory of the reals. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 291--295. IEEE, 1988.

[Rez78] B. Reznick. Extremal PSD forms with few terms. *Duke Mathematical Journal*, 45(2):363--374, 1978.

[Rez89] B. Reznick. Forms derived from the arithmetic-geometric inequality. *Mathematische Annalen*, 283(3):431--464, 1989.

[Rez95] B. Reznick. Uniform denominators in Hilbert's seventeenth problem. *Mathematische Zeitschrift*, 220(1):75--97, 1995.

[Rob52] J. Robinson. Existential definability in arithmetic. *Transactions of the American Mathematical Society*, 72(3):437--449, 1952.

[Roc70] R. T. Rockafellar. *Convex analysis.* Number 28. Princeton University Press, 1970.

[SdW18] H. Seidler and T. de Wolff. An Experimental Comparison of SONC and SOS Certificates for Unconstrained Optimization. *arXiv preprint arXiv:1808.08431*, 2018.

[SdW19] H. Seidler and T. de Wolff. POEM: Effective Methods in Polynomial Optimization, version 0.2.1.0. `http://www.user.tu-berlin.de/henning.seidler/POEM/`, july 2019.

[Sei21a] H. Seidler. Improved Lower Bounds for Global Polynomial Optimisation. *arXiv preprint arXiv:2105.14124*, 2021.

[Sei21b] H. Seidler. POEM: Effective Methods in Polynomial Optimization, version 0.3.0.0. `http://www.user.tu-berlin.de/henning.seidler/POEM/`, may 2021.

[Stu99] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625--653, 1999.

[Tar98] A. Tarski. A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition*, pages 24--84. Springer, 1998.

[TTT99] K. Toh, M. Todd, and R. Tutuncu. SDPT3 --- a Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11:545--581, 1999.

[vD94] D. van Dalen. *Logic and structure*, volume 3. Springer, 1994.

[Wan18] J. Wang. Nonnegative polynomials and circuit polynomials. *arXiv preprint arXiv:1804.09455*, 2018.

[WKK$^+$08] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto. Algorithm 883: SparsePOP --- A Sparse Semidefinite Programming Relaxation of Polynomial Optimization Problems. *ACM Transactions on Mathematical Software (TOMS)*, 35(2):15, 2008.

Appendix A

Appendix

A.1 An Example of Exact SAGE Decomposition

Let

$$\begin{aligned} f(\boldsymbol{x}) = {} & 277 - 1x_2^2 + 159x_2^2x_3^6 + 275x_2^4 - 112x_1^1x_2^1x_3^2 + 23x_1^1x_2^2x_3^3 + 338x_1^2x_3^4 + 166x_1^2x_2^1x_3^1 \\ & - 89x_1^2x_2^1x_3^2 - 19x_1^2x_2^2x_3^1 + 74x_1^2x_2^2x_3^2 + 268x_1^6x_3^2. \end{aligned}$$

Our optimisation algorithm OPTSAGE returns $(\boldsymbol{\nu}, \boldsymbol{c})$ corresponding to the following exact rational SAGE decomposition: $f(\boldsymbol{x}) = \sum_{j=1}^{12} f_j(\boldsymbol{x})$, where f_j is the polynomial with coefficient vector $\boldsymbol{c}^{(j)}$ for $j \in \{1, \ldots, 12\}$, $f_j = 0$ for $j \in \{1, 3, 4, 7, 11, 12\}$, and

$$\begin{aligned}
\boldsymbol{\nu}^{(2)} &= \left(\frac{1494563}{131072}, -\frac{1494563}{65536}, 0, \frac{1494563}{131072}, 0, 0, 0, 0, 0, 0, 0, 0\right) \\
f_2 &= \frac{7050}{5161} - \frac{470481700740758477768}{2063044386600414657}x_2^2 + \frac{51364821929347737990}{176335433100237218699413}x_2^2x_3^6 + \frac{1284035788022670567885}{13490375489389305583}x_2^4 + \\
&\frac{83062810507300624}{22200197261503152705103}x_1^1x_2^1x_3^2 + \frac{16345503627618}{597428155608650885}x_1^1x_2^2x_3^3 + \frac{151893109261090080}{12048313292258664398599}x_1^2x_3^4 + \\
&\frac{6887262866990276056}{3068799030848037634611567}x_1^2x_2^1x_3^1 + \frac{584984347065145672}{102911615960676281785795}x_1^2x_2^1x_3^2 + \frac{35180881401784}{5417152094682478713}x_1^2x_2^2x_3^1 + \\
&\frac{910066980468240}{55380305977850685287}x_1^2x_2^2x_3^2 + \frac{7052856072901897195}{672038441901303429467107}x_1^6x_3^2 \\
\boldsymbol{\nu}^{(5)} &= \left(\frac{2763713}{131072}, \frac{1404885}{65536}, \frac{40783}{131072}, \frac{2618913}{131072}, -\frac{1033291}{8192}, \frac{46873}{65536}, \frac{8086635}{131072}, \frac{2825}{16384}, \frac{16731}{65536}, \frac{5689}{32768}, \frac{4177}{16384}, \frac{3431}{65536}\right) \\
f_5 &= \frac{4895}{6532} + \frac{397583449413593513}{62516496563648929}x_2^2 + \frac{4282065321967986445380570}{176335433100237218699413}x_2^2x_3^6 + \frac{5340068537046034265265}{107923003915114444664}x_2^4 - \\
&\frac{797551428730610634080}{6323041088437240873}x_1^1x_2^1x_3^2 + \frac{624998409184656940}{119485631121730177}x_1^1x_2^2x_3^3 + \frac{2505598243831220279021280}{12048313292258664398599}x_1^2x_3^4 + \\
&\frac{317731662232293486548800}{3068799030848037634611567}x_1^2x_2^1x_3^1 + \frac{7986264919496237716952}{20582323192135256357159}x_1^2x_2^1x_3^2 + \frac{4894573585100619175}{16251456284047436139}x_1^2x_2^2x_3^1 + \\
&\frac{62000736112963044488}{55380305977850685287}x_1^2x_2^2x_3^2 + \frac{98753640749442855613080}{672038441901303429467107}x_1^6x_3^2 \\
\boldsymbol{\nu}^{(6)} &= \left(\frac{24235}{65536}, \frac{11469}{16384}, \frac{731111}{65536}, \frac{413881}{65536}, \frac{75499}{65536}, -\frac{2262417}{65536}, \frac{224607}{65536}, \frac{48937}{65536}, \frac{41999}{32768}, \frac{5895}{4096}, \frac{459131}{65536}, \frac{30411}{32768}\right) \\
f_6 &= \frac{11}{6124} + \frac{175751812330814062}{6189133159801243971}x_2^2 + \frac{20993411202657969962132460}{176335433100237218699413}x_2^2x_3^6 + \frac{230789345356723546855}{107923003915114444664}x_2^4 + \\
&\frac{3497123471444427592320}{22200197261503152705103}x_1^1x_2^1x_3^2 - \frac{20624391427747375598}{597428155608650885}x_1^1x_2^2x_3^3 + \frac{19031996317336046221320}{12048313292258664398599}x_1^2x_3^4 + \\
&\frac{188140322704886960215808}{3068799030848037634611567}x_1^2x_2^1x_3^1 + \frac{27412383525960721370432}{102911615960676281785795}x_1^2x_2^1x_3^2 + \frac{616440189731061280}{1805717364894159571}x_1^2x_2^2x_3^1 + \\
&\frac{232955118417446202960}{55380305977850685287}x_1^2x_2^2x_3^2 + \frac{239384185705575737209071}{672038441901303429467107}x_1^6x_3^2
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\nu}^{(8)} &= \left(\frac{5736955}{131072}, \frac{1718597}{65536}, \frac{2677}{65536}, \frac{283517}{16384}, \frac{298687}{65536}, \frac{5897}{32768}, \frac{1205677}{131072}, -\frac{1398845}{8192}, \frac{338517}{65536}, \frac{488327}{32768}, \frac{37365}{8192}, \frac{1469765}{32768}\right)\\
f_8 &= \frac{16553}{6389} + \frac{80194631513313539404}{6189133159801243971}x_2^2 + \frac{9360973661157175311112852}{176335433100237218699413}x_2^2x_3^6 + \frac{962838001240718755020}{13490375489389305583}x_2^4+\\
&\frac{168516279078159280107840}{2220019726150315270510 3}x_1^1x_2^1x_3^2 + \frac{261920871973041930}{119485631121730177}x_1^1x_2^2x_3^3 + \frac{6221905163318221681075 20}{1204831329225866439859 9}x_1^2x_3^4-\\
&\frac{174673447091258679729208576}{10229330102826792115371 89}x_1^2x_2^1x_3^1 + \frac{13456007831148218966679 52}{1029116159606762817857 95}x_1^2x_2^1x_3^2 + \frac{23324195037099031144 0}{5417152094682478713}x_1^2x_2^2x_3^1+\\
&\frac{184738012166616458823 0}{55380305977850685287}x_1^2x_2^2x_3^2 + \frac{1409203729090362281675896 50}{6720384419013034294671 07}x_1^6x_3^2\\
\boldsymbol{\nu}^{(9)} &= \left(\frac{942879}{131072}, \frac{66451}{8192}, \frac{13687}{131072}, \frac{291521}{32768}, \frac{487245}{65536}, \frac{29821}{65536}, \frac{2150473}{65536}, \frac{252191}{32768}, -\frac{3460417}{32768}, \frac{170395}{16384}, \frac{548667}{65536}, \frac{925733}{65536}\right)\\
f_9 &= \frac{181}{1076} + \frac{32665647808648050 56}{2063044386600414657}x_2^2 + \frac{945568751423587451936415}{176335433100237218699413}x_2^2x_3^6 + \frac{195552867021926712375}{13490375489389305583}x_2^4+\\
&\frac{10859834179375768514928 0}{2220019726150315270510 3}x_1^1x_2^1x_3^2 + \frac{261624143096996295}{119485631121730177}x_1^1x_2^2x_3^3 + \frac{876811143650234639492646}{1204831329225866439859 9}x_1^2x_3^4+\\
&\frac{933036235603612769694560 0}{3068799030848037634611567}x_1^2x_2^1x_3^1 - \frac{10867831413671203786608456}{1029116159606762817857 95}x_1^2x_2^1x_3^2 + \frac{192909634965734635520}{16251456284047436139}x_1^2x_2^2x_3^1+\\
&\frac{133955133103661412452 0}{55380305977850685287}x_1^2x_2^2x_3^2 + \frac{175320070736148747041373 30}{6720384419013034294671 07}x_1^6x_3^2\\
\boldsymbol{\nu}^{(10)} &= \left(\frac{732091}{262144}, \frac{167933}{32768}, \frac{14541}{131072}, \frac{7773365}{262144}, \frac{1327}{512}, \frac{29683}{65536}, \frac{272641}{131072}, \frac{293905}{65536}, \frac{214965}{65536}, -\frac{4886959}{65536}, \frac{144047}{32768}, \frac{1284635}{65536}\right)\\
f_{10} &= \frac{157}{2744} + \frac{5424537902241772912}{6189133159801243971}x_2^2 + \frac{880139855950527034906380}{176335433100237218699413}x_2^2x_3^6 + \frac{571069367996398273530}{13490375489389305583}x_2^4+\\
&\frac{3316914557864893313328 0}{2220019726150315270510 3}x_1^1x_2^1x_3^2 + \frac{228162076394260360}{119485631121730177}x_1^1x_2^2x_3^3 + \frac{48697840759706172793616}{1204831329225866439859 9}x_1^2x_3^4+\\
&\frac{1587820308255205569373114}{10229330102826792115371 89}x_1^2x_2^1x_3^1 + \frac{5915050338968076718077 7}{2058232319213525635715 9}x_1^2x_2^1x_3^2 - \frac{403951932103643744176}{5417152094682478713}x_1^2x_2^2x_3^1+\\
&\frac{616254425060782282800}{55380305977850685287}x_1^2x_2^2x_3^2 + \frac{213157775675871247307383 50}{6720384419013034294671 07}x_1^6x_3^2
\end{aligned}$$