

UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques i Informàtica




Barcelona Supercomputing Center
Centro Nacional de Supercomputación

**DUAL DEGREE IN MATHEMATICS AND COMPUTER SCIENCE**

**Final degree thesis**

---

# NON-NEGATIVITY CERTIFICATES FOR POLYNOMIALS

---

**Author: Joel Hurtado Moreno**

**Supervisor:** Dr. Carlos D'Andrea
**Co-supervisor:** Dr. Sara Royuela Alcázar

**Carried out at:** Department of Mathematics and Computer Science

**Barcelona,** January 15, 2025

# Abstract

This study addresses questions in real algebraic geometry, including its intersection with computational aspects. In particular, the work focuses on characterizing the non-negativity of multivariate polynomials. The content is organized into four sections, followed by some conclusions.

The first chapter provides a detailed introduction to the fundamental concepts of the field, including basic definitions and the main theoretical tools necessary for understanding the subject. These concepts are also contextualized within their historical framework, with the aim of offering a comprehensive overview of the topic.

The second chapter presents a generalization of the work developed in "Rational certificates of non-negativity on semialgebraic subsets of cylinders" by Gabriela Jeronimo and Daniel Perrucci (see [9]), thereby extending the obtained results. Additionally, an explicit calculation of a question posed in the same article is included, with the objective of offering a more complete and detailed resolution.

The third chapter presents the implementation of three algorithms along with a set of examples of their executions. The first algorithm determines whether a multivariate polynomial is a sum of squares and, if so, provides a decomposition. The second algorithm allows us to determine whether a multivariate polynomial belongs to the quadratic module generated by a set of polynomials. Again, if this is the case, such a decomposition in the quadratic module is provided. The third and final algorithm checks whether a quadratic module is archimedean and, if so, provides a decomposition in the quadratic module. For archimedeanity only theoretical characterizations previously existed, and in this work, a computational version has been developed. Additionally, the code for the quadratic module algorithm has been optimized through parallelization to run on the MareNostrum 5 supercomputer at the Barcelona Supercomputing Center.

Finally, the fourth chapter focuses on the parallelization of the quadratic module algorithm and its performance evaluation. It details the methodology used to enhance efficiency and analyzes the resulting improvements in execution time and scalability. The assessment includes a comparison of sequential and parallel versions, highlighting the effectiveness and potential limitations of the parallelization strategy implemented.



2020 Mathematics Subject Classification. 13P25, 14P10, 14Q20, 14Q30, 68W30

# Resum

Aquest estudi aborda qüestions de geometria algebraica real, incloent la seva intersecció amb aspectes computacionals. En particular, el treball es centra en la caracterització de la no negativitat de polinomis en diverses variables. El contingut s'organitza en quatre blocs, seguits d'algunes conclusions.

El primer capítol proporciona una introducció detallada als conceptes fonamentals de l'àrea, incloent-hi les definicions bàsiques i les principals eines teòriques necessàries per comprendre la matèria. També es contextualitzen aquests conceptes dins el seu marc històric, amb l'objectiu d'oferir una visió global del tema.

En el segon capítol, es presenta una generalització del treball desenvolupat a "Rational certificates of non-negativity on semialgebraic subsets of cylinders" per Gabriela Jeronimo i Daniel Perrucci (veure [9]), ampliant així els resultats obtinguts. A més, s'inclou el càlcul explícit d'una qüestió plantejada en el mateix article, amb l'objectiu de proporcionar una resolució més completa i detallada.

El tercer capítol presenta la implementació de tres algorismes juntament amb un conjunt d'exemples de les seves execucions. El primer algorisme determina si un polinomi en diverses variables és una suma de quadrats i, en cas afirmatiu, permet obtenir una descomposició. El segon algorisme permet determinar si un polinomi en diverses variables pertany al mòdul quadràtic generat per un conjunt de polinomis. De nou, en cas afirmatiu, es proporciona tal descomposició en el mòdul quadràtic. El tercer i últim algorisme permet comprovar si un mòdul quadràtic és arquimedià i, en cas que ho sigui, proporciona una descomposició en el mòdul quadràtic. Per a l'arquimedianeitat només existien caracteritzacions teòriques, i en aquest treball s'ha desenvolupat una versió computacional. A més, s'ha dut a terme una optimització del codi de l'algorisme del mòdul quadràtic mitjançant paral·lelització, per tal d'executar-lo al superodinador MareNostrum 5 del Barcelona Supercomputing Center.

Finalment, el quart capítol es centra en la paral·lelització de l'algorisme del mòdul quadràtic i la seva avaluació de rendiment. Detalla la metodologia utilitzada per millorar l'eficiència i analitza les millores obtingudes en el temps d'execució així com l'escalabilitat. L'avaluació inclou una comparació entre les versions seqüencial i paral·lela, destacant l'efectivitat i les possibles limitacions de l'estratègia de paral·lelització implementada.

## Acknowledgments

First of all, I would like to thank Carlos for the time dedicated to addressing and discussing my doubts, and for always being willing to listen. Secondly, I would like to thank Sara for her invaluable help, which has been crucial in overcoming challenges and advancing the work. Finally, I want to express my gratitude to my family and friends for their support throughout this time, I would not have come this far without all of them.

# Contents


**1 Positive Polynomials and Sums of Squares** **1**
- 1.1 Basic Notation . . . 1
- 1.2 Historical Background and Preliminary Results . . . 2
- 1.3 Sums of Squares Representations and Semidefinite Optimization . . . 6
- 1.4 Between Non-negativity and Sums of Squares . . . 8
- 1.5 Positivity on Semialgebraic Sets . . . 9
  - 1.5.1 Semialgebraic Sets . . . 9
  - 1.5.2 Preorders, Quadratic Modules, and Archimedeanity . . . 10
  - 1.5.3 The Positivstellensatz . . . 12
- 1.6 Applications . . . 14
  - 1.6.1 Shape-Constrained Regression Problem . . . 14
  - 1.6.2 Difference of Convex (DC) Programming . . . 16

**2 Rational Certificates of Non-negativity on Semialgebraic Subsets of Cylinders** **19**
- 2.1 Contextual Frame and Further Insights . . . 19
- 2.2 Certificates for Generalized Cylinders . . . 20

**3 Algorithms and Implementation** **27**
- 3.1 Preliminaries . . . 27
- 3.2 Sums of Squares . . . 29
- 3.3 Quadratic Modules . . . 31
- 3.4 Archimedeanity . . . 34

**4 Parallelization and Performance Evaluation** **39**
- 4.1 Parallelization . . . 39
- 4.2 Performance Evaluation . . . 41

**5 Conclusions** **47**

**A Additional Proofs** **49**

# Chapter 1

# Positive Polynomials and Sums of Squares

## 1.1 Basic Notation

Throughout this text, we will primarily focus in the ring $\mathbb{R}[\bar{X}]$ of polynomials with real coefficients in $n \in \mathbb{N} \setminus \{0\}$ indeterminates $\bar{X} := (X_1, \ldots, X_n)$, but will also need to consider polynomials over subfields of $\mathbb{R}$. We will use the following monomial notation: For a multi-index $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}^n_{\geq 0}$, let $\bar{X}^\alpha$ denote $X_1^{\alpha_1} \ldots X_n^{\alpha_n}$. The degree of $\bar{X}^\alpha$ is $|\alpha| = \alpha_1 + \ldots + \alpha_n$, and the degree of a polynomial $f \in \mathbb{R}[\bar{X}]$, denoted deg $f$, is the maximum degree of the monomials in $f$ with nonzero coefficients.

Thus, we can express a polynomial in $\mathbb{R}[\bar{X}]$ as

$$f = \sum_{|\alpha| \leq m} a_\alpha \bar{X}^\alpha,$$

where deg $f$ is $m$, $a_\alpha \in \mathbb{R}$, and $a_\alpha \neq 0$ for some $\alpha$ with $|\alpha| = m$.

Also, for $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}^n_{\geq 0}$, the multinomial coefficient corresponding to $\alpha$ is

$$\binom{|\alpha|}{\alpha} = \frac{|\alpha|!}{\alpha_1! \ldots \alpha_n!},$$

whose name comes from the multinomial identity

$$(X_1 + \ldots + X_n)^m = \sum_{|\alpha| = m} \binom{m}{\alpha} \bar{X}^\alpha,$$

which holds for any $m \in \mathbb{N}$.

Given two real square matrices $A$ and $B$, we will use the notation $\langle A, B\rangle$ to denote the trace of their product, i.e., trace(AB). Additionally, we will use the symbols $C \succeq 0$ and $C \succ 0$ to indicate that a matrix $C$ is positive semidefinite (PSD) and positive definite (PD), respectively.

For a polynomial $f \in \mathbb{R}[\bar{X}]$ in several variables, we will define its $l_1$-norm as $||f||_1 = \sum |a_\alpha|$, where the sum runs over all coefficients $a_\alpha$ of $f$. We will write $f \geq 0$ (or PSD) if it is non-negative, i.e., if it is greater than or equal to 0 at all points in $\mathbb{R}^n$, and $f > 0$ (or PD) if it is strictly positive. We will also consider the vector $v_d(\bar{X}) = (\bar{X}^\alpha)_{|\alpha|\leq d} = (1, X_1, \ldots, X_n, X_1^2, X_1X_2, \ldots, X_{n-1}X_n, X_n^2, \ldots, X_1^d, \ldots, X_n^d)'$, which includes all monomials of degree at most $d$. It has dimension $s(d) \times 1$, where $s(d) := \binom{n+d}{d}$, and those monomials form the canonical basis of the vector space $\mathbb{R}[\bar{X}]_d$ of polynomials of degree at most $d$.

We will denote by $\sum \mathbb{R}[\bar{X}]^2 \subset \mathbb{R}[\bar{X}]$ the cone of sum of squares (SOS) polynomials, which are the polynomials that can be expressed as $g_1^2 + \ldots + g_k^2$, with $g_1, \ldots, g_k \in \mathbb{R}[\bar{X}]$ and $k \in \mathbb{Z}_+$, where this representation may not be unique. Similarly, we will denote by $\mathcal{H}_N[\bar{X}]_{2d}$ the cone of homogeneous and non-negative polynomials of degree $2d$ (for some $d \in \mathbb{Z}_+$), and by $\mathcal{H}_S[\bar{X}]_{2d}$ the cone of homogeneous SOS polynomials of the same degree. A complete definition of homogeneous polynomial, if needed, can be found in 1.4.

## 1.2 Historical Background and Preliminary Results

In real algebraic geometry, a key question is whether a non-negative polynomial can be represented in a way that explicitly demonstrates its non-negativity, such as by expressing it as a sum of squares. It is clear that a polynomial $f$ with real coefficients that can be written as a sum of squares of real polynomials will always be non-negative over $\mathbb{R}^n$ (an explicit decomposition of $f$ into a sum of squares immediately demonstrates this fact). This idea, and its generalizations, underpins a substantial body of theoretical and computational research related to positive polynomials and sums of squares.

The study of the relationship between non-negativity and sums of squares began with David Hilbert in 1888, who showed that every non-negative real polynomial can be expressed as a sum of squares only in one of the following three cases: univariate polynomials, quadratic polynomials, and bivariate polynomials of degree four. For all other cases, Hilbert demonstrated the existence of non-negative polynomials that are not sums of squares, but he did not explicitly provide a polynomial that is not a sum of squares.

The first explicit example appeared eighty years later and is due to Motzkin. Since then, many explicit examples of non-negative polynomials that are not sums of squares have

appeared.

**Proposition 1.1.** *Motzkin's polynomial* $m(x, y) = x^4y^2+x^2y^4+1-3x^2y^2$ *is non-negative.*

*Proof.* By the AM/GM inequality (inequality of arithmetic and geometric means) we have

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

so $m$ is actually non-negative. □

**Proposition 1.2.** *Motzkin's polynomial* $m(x, y) = x^4y^2 + x^2y^4 + 1 - 3x^2y^2$ *is not SOS.*

*Proof.* To show that is not SOS, we will need an auxiliary Lemma. Recall that the Newton Polytope of a polynomial $m$, $N(m)$, is the convex hull of the vectors of exponents of the monomials of $m$.

**Lemma 1.3.** *If* $p = \sum_i h_i^2$ *is an SOS polynomial then for every* $i$*, we have the polytope inclusion* $2N(h_i) \subseteq N(p)$.

Then, if $m$ were a sum of squares of polynomials, by the previous Lemma, the squares would be of the form $(ax^2y + bxy^2 + cxy + d)^2$, with $a, b, c, d \in \mathbb{R}$. However, none of which can have negative coefficients for $x^2y^2$, so we have a contradiction. □

In the case of polynomials in $\mathbb{Q}[\bar{X}]$, we cannot ensure that a polynomial being a sum of squares guarantees that the polynomials in this decomposition as a sum of squares live in $\mathbb{Q}[\bar{X}]$.

In order to keep going, we need the next definition.

**Definition 1.4.** *A form is a homogeneous polynomial, i.e., one for which all monomials have the same degree. Given* $f \in \mathbb{R}[\bar{X}]$ *of degree* $d$*, the degree* $d$ *homogenization of* $f$*, denoted* $\bar{f}$*, is a form in* $\mathbb{R}[X_1, \ldots, X_n, X_{n+1}]$ *of degree* $d$*, defined by*

$$\bar{f} := X_{n+1}^d \cdot f\left(\frac{X_1}{X_{n+1}}, \ldots, \frac{X_n}{X_{n+1}}\right).$$

*Similarly, if* $p \in \mathbb{R}[X_1, \ldots, X_n, X_{n+1}]$ *is a form of degree* $d$*, then the dehomogenization of* $p$ *is a polynomial of degree less than or equal to* $d$ *in* $\mathbb{R}[\bar{X}]$ *that results from evaluating* $X_{n+1} = 1$*, i.e.,* $p(X_1, \ldots, X_n, 1)$.

**Example 1.5.** Consider again the Motzkin polynomial $m(x,y) = x^4y^2 + x^2y^4 - 3x^2y^2 + 1 \in \mathbb{R}[x,y]$. The degree of $m$ is six and the decomposition of $m$ into forms would be $m = m_6 + m_4 + m_0 = (x^4y^2 + x^2y^4) + (-3x^2y^2) + 1$. Therefore, the degree six homogenization of $m$ would be $\bar{m}(x,y,z) = x^4y^2 + x^2y^4 - 3x^2y^2z^2 + z^6$, which is a form in $\mathbb{R}[x,y,z]$.

Then, going back to 1900, Hilbert proposed a list of twenty-three problems in the International Congress of Mathematicians held at Paris. Among them, there was the famous seventeenth problem which generalizes his previous results:

*Is it true that every PSD form must be a sum of squares of quotient forms? Dehomogenizing, given a PSD polynomial $f \in \mathbb{R}[\bar{X}]$, do there exist $g_1, \ldots, g_k, h_1, \ldots, h_k \in \mathbb{R}[\bar{X}]$ with $h_i \neq 0$ for all $i$, such that*

$$f = \frac{g_1^2}{h_1^2} + \ldots + \frac{g_k^2}{h_k^2}?$$

An equivalent formulation (by clearing the denominators) could be:

*Do there exist $g_1, \ldots, g_k, h \in \mathbb{R}[\bar{X}]$ with $h \neq 0$ such that*

$$h^2 f = g_1^2 + \ldots + g_k^2?$$

The answer turned out to be affirmative, and it was proven by Emil Artin in 1927, using the Artin-Schreier theory of real closed fields.

**Example 1.6.** We can represent the Motzkin polynomial as a sum of squares of rational functions:

$$X^4Y^2 + X^2Y^4 - 3X^2Y^2 + 1 = \frac{X^2Y^2(X^2+Y^2+1)(X^2+Y^2-2)^2 + (X^2-Y^2)^2}{(X^2+Y^2)^2}$$

Before moving on, it is important to take into account the next detail.

**Proposition 1.7.** *If $f \in \mathbb{R}[\bar{X}]$ is PSD, then the degree of $f$ must be even.*

*Proof.* Suppose, first, the case of $f$ being univariate. If deg $f$ is odd, then $f$ must have at least one real root with odd multiplicity. Since $f$ changes sign at this root, $f$ cannot be PSD.

Now suppose the multivariate case, where $f$ is in $\mathbb{R}[\bar{X}]$ and has degree $d$. Descomposing $f$ into forms we get $f = f_d + f_{d-1} + \ldots + f_0$, where $f_i$ is homogeneous of degree $i$. Suppose $d$ is odd. Fix $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and consider the univariate polynomial

$$g(X) := f(x_1X, x_2X, \ldots, x_nX) = \sum_{i=0}^{d} f_i(x)X^i.$$

Since $f$ is PSD, $g(t) = f(x \cdot t) \geq 0$ for any $t \in \mathbb{R}$, hence $g$ is PSD. Since $g$ is an univariate polynomial, it must have even degree, and since $d$ is odd, this implies that $f_d(x) = 0$. But that must be hold for any $x \in \mathbb{R}^n$, and therefore $f_d$ is the zero polynomial, which contradicts that deg $f$ = $d$. Thus the degree of $f$ must be even. □

**Remark 1.8.** The converse is not necessarily true, since a polynomial with even degree might not be PSD.

Now, taking a deeper look at the study of sum of squares for univariate polynomials, let us see some results.

**Proposition 1.9.** *An even degree polynomial $p(x) \in \mathbb{R}[x]$ is PSD if and only if it can be written as a sum of two squares, i.e., $p(x) = p_1(x)^2 + p_2(x)^2$, $p_1, p_2 \in \mathbb{R}[x]$.*

*Proof.* The implication from right to left is clear. Let us now consider the other direction. We first notice that every real root (if at all it exists) occurs with even multiplicity (it can be seen from 1.7). Hence, we may write:

$$p(x) = c\prod_i (x-\alpha_i)^{2n_i} \prod_j (x-\beta_j)^{m_j} \prod_j (x-\bar{\beta_j})^{m_j},$$

where $\alpha_i \in \mathbb{R}$, $\beta_j \notin \mathbb{R}$, and $c \in \mathbb{R}_{\geq 0}$ since $p$ is non-negative.

Now, let

$$h(x) = \prod_i (x-\alpha_i)^{n_i} \prod_j (x-\beta_j)^{m_j}.$$

We have $p(x) = c\,h(x)\bar{h}(x)$, and writing $h(x) = r(x) + is(x)$, with $r, s \in \mathbb{R}[x]$, we end up with $p(x) = c(r(x) + is(x))(r(x) - is(x)) = c(r(x)^2 + s(x)^2) = (\sqrt{c}r(x))^2 + (\sqrt{c}s(x))^2$. □

The scenario in $\mathbb{Q}[x]$ is of particular interest as well. For instance, take the polynomial $f(x) = x^2 + 3 \in \mathbb{Q}[x]$, which is PD.

**Proposition 1.10.** *The polynomial $f(x) = x^2 + 3 \in \mathbb{Q}[x]$ cannot be written as $f_1(x)^2 + f_2(x)^2$, with $f_1, f_2 \in \mathbb{Q}[x]$.*

*Proof.* Suppose that $x^2 + 3 = f_1(x)^2 + f_2(x)^2$, for some $f_1, f_2 \in \mathbb{Q}[x]$. Evaluating in $x = 0$ we get: $3 = f_1(0)^2 + f_2(0)^2 = \frac{A^2}{B^2} + \frac{C^2}{D^2}$, with $A, B, C, D \in \mathbb{Z}$. Or, equivalently, $3B^2D^2 = (AD)^2 + (BC)^2$. But this contradicts Fermat's theorem on sums of two squares, which states that an integer can be expressed as a sum of two squares if and only if each of its prime factors of the form $4k + 3$, $k \in \mathbb{Z}$, appears with an even exponent. □

However, we have that $f(x) = x^2 + 1^2 + 1^2 + 1^2$, so it is a sum of squares. Indeed, more than one hundred years ago Landau ([3]) proved that every polynomial in $\mathbb{Q}[x]$ which is non-negative is a sum of eight squares of rational polynomials. This result was later improved in [4] by Pourchet, who proved that five or fewer squares are enough. In [5], algorithms for finding such a decomposition are studied.

## 1.3 Sums of Squares Representations and Semidefinite Optimization

As said before, a sum of squares representation for a polynomial serves as an immediate proof of its positivity, which is why finding such representations is of great interest. The following important result, originally due to Choi, Lam, and Reznick [11], and later reformulated in a more convenient form by Powers and Wörmann [12], provides an algorithm for SOS decomposition. Following the notation from 1.1:

**Theorem 1.11.** *A polynomial $f \in \mathbb{R}[\bar{X}]_{2d}$ has a sum of squares decomposition (or is SOS) if and only if there exists a real symmetric and positive semidefinite matrix $Q \in \mathbb{R}^{s(d)\times s(d)}$ such that $f(\bar{X}) = v_d(\bar{X})'Qv_d(\bar{X})$, for all $\bar{X} \in \mathbb{R}^n$.*

*Proof.* Suppose, first, that $f$ has a SOS decomposition $f(\bar{X}) = \sum_{i=1}^{k} h_i(\bar{X})^2$ for some family $\{h_i : i = 1, \ldots, k\} \subset \mathbb{R}[\bar{X}]$. Since $f$ has degree $2d$, we know that the degree of each $h_i$ is bounded by $d$. Let $H_i$ be such that $h_i(\bar{X}) = H_i' v_d(\bar{X}), i = 1, \ldots, k$, i.e., the vector of coefficients of the polynomial $h_i$. Thus,

$$f(\bar{X}) = \sum_{i=1}^{k} v_d(\bar{X})' H_i H_i' v_d(\bar{X}) = v_d(\bar{X})' Q v_d(\bar{X}), \quad \forall \bar{X} \in \mathbb{R}^n,$$

with $Q \in \mathbb{R}^{s(d)\times s(d)}$, $Q := \sum_{i=1}^{k} H_i H_i' \succeq 0$, and the implication follows.

Suppose, now, that there exists a real symmetric $s(d) \times s(d)$ matrix $Q \succeq 0$ for which $f(\bar{X}) = v_d(\bar{X})'Qv_d(\bar{X})$, for all $\bar{X} \in \mathbb{R}^n$. Then $Q = GG'$ for some $s(d) \times k$ matrix $G$, and therefore,

$$f(\bar{X}) = v_d(\bar{X})' G G' v_d(\bar{X}) = \sum_{i=1}^{k} (G' v_d(\bar{X}))_i^2, \quad \forall \bar{X} \in \mathbb{R}^n.$$

Since $(G'v_d(\bar{X}))_i$ is a polynomial, then $f$ is expressed as a sum of squares of the polynomials $(G'v_d(\bar{X}))_i, i = 1, \ldots, k$, and the theorem is proved. □

On one hand, the method described by Choi, Lam, and Reznick yields a matrix that provides the form of the sum of squares decomposition. However, since the equations involved in constructing this matrix contain several variables, the resulting matrix is often

parametrized. To obtain an explicit SOS representation, one must determine the parameter values that ensure the matrix is PSD. A common approach is to compute the eigenvalues of the matrix. Nevertheless, finding eigenvalues of a matrix with parameter-dependent entries is impractical, except for very small cases.

Powers and Wörmann, on the other hand, exploit the structure of the equations by noting that each variable in the matrix construction appears only once per equation. They decompose the matrix as a sum of several matrices, each multiplied by a parameter. Consequently, they show that ensuring the initial matrix is PSD is equivalent to proving that a certain semialgebraic set is non-empty (a complete definition of semialgebraic set, if needed, can be found in 1.14). There are various algorithms available for determining whether a semialgebraic set is empty, such as those based on quantifier elimination. However, these algorithms are generally limited to handling "small" examples, making this approach more theoretical than practical.

Lasserre, Parrilo, and others have developed and studied a more practical approach based on semidefinite programming, which can be roughly described as linear programming over the cone of positive semidefinite matrices. Semidefinite programming generalizes linear programming but is not significantly more challenging to solve. Efficient numerical algorithms for solving semidefinite programs have been developed and implemented.

Following the notation from the theorem above they observed that, since the identity $f(\bar{X}) = v_d(\bar{X})'Qv_d(\bar{X})$ for all $\bar{X} \in \mathbb{R}^n$ provides linear equations that the coefficients of the matrix $Q$ must satisfy, writing

$$v_d(\bar{X})v_d(\bar{X})' = \sum_{\alpha \in \mathbb{N}^n} B_\alpha \bar{X}^\alpha,$$

for appropriate $s(d) \times s(d)$ real symmetric matrices with 0's and 1's, checking whether the polynomial $f(\bar{X}) = \sum_\alpha f_\alpha \bar{X}^\alpha$ is SOS reduces to solving the **semidefinite optimization problem**:

Find $Q \in \mathbb{R}^{s(d)\times s(d)}$ such that:

$$Q = Q', \quad Q \succeq 0, \quad \langle Q, B_\alpha \rangle = f_\alpha, \quad \forall \alpha \in \mathbb{N}^n.$$

This is a tractable convex optimization problem. Indeed, for any fixed arbitrary precision $\epsilon > 0$, a semidefinite program can be solved in computational time that is polynomial in the input size of the problem. Note that the size $s(d)$ of the semidefinite program is bounded by $n^d$. However, it is important to keep in mind that using semidefinite programs to obtain SOS representations in this manner involves numerical software, and therefore, there is no guarantee of an exact solution.

## 1.4 Between Non-negativity and Sums of Squares

Asking about the difference between the set of non-negative polynomials and the set of polynomials that are sums of squares is truly interesting. As we have seen in the previous section, there are efficient ways to check if a polynomial is a sum of squares, but checking if it is non-negative is not that easy. Therefore, it is relevant to understand to what extent we can gain information about non-negative polynomials by studying the polynomials that belong to the cone of sums of squares. There are two kinds of results for that comparison, depending on whether or not one keeps the degree fixed.

The first one, related to investigations done by Blekherman in the early 2000s, was obtained looking at the "gap" between these cones and quantifying the discrepancy. The second one, seeks closely approximating a PSD polynomial through SOS polynomials. This is the case of Lasserre, who also around 2005, showed that every PSD polynomial $f$ that attains a global minimum can be approximated with respect to the $l_1$ norm by SOS polynomials.

We will show the results, but without going into the proofs, which are more laborious and would take considerable time. However, more information about these can be found in [6], [7], [8] for interested readers.

Since homogenization preserves non-negativity and sum of squares representation, the next theorem works with forms. To compare both sets we need subsets of finite volume, so let $\mathscr{H}$ be the hyperplane $\{f \in \mathbb{R}[\bar{X}] : \int_{\mathbb{S}^{n-1}} f d\mu = 1\}$, where $\mu$ is the rotation invariant measure on the unit sphere $\mathbb{S}^{n-1} \subset \mathbb{R}^n$, and let $\hat{\mathcal{H}}_N[\bar{X}]_{2d} := \mathscr{H} \cap \mathcal{H}_N[\bar{X}]_{2d}$, $\hat{\mathcal{H}}_S[\bar{X}]_{2d} := \mathscr{H} \cap \mathcal{H}_S[\bar{X}]_{2d}$, where we have followed the notation from 1.1.

**Theorem 1.12.** *There exist constants $K_1, K_2 > 0$ depending only on $d$ and such that*

$$K_1 n^{(d/2-1)/2} \leq \frac{vol(\hat{\mathcal{H}}_N[\bar{X}]_{2d})}{vol(\hat{\mathcal{H}}_S[\bar{X}]_{2d})} \leq K_2 n^{(d/2-1)/2}.$$

Therefore, if we make $n \to \infty$ while keeping $d$ fixed, the gap between $\hat{\mathcal{H}}_N[\bar{X}]_{2d}$ and $\hat{\mathcal{H}}_S[\bar{X}]_{2d}$ can grow arbitrarily.

Now we will see that we can perturb a non-negative polynomial $f \in \mathbb{R}[\bar{X}]$ to make it a sum of squares. That said, we must keep in mind that there is a price to pay. In our case, the approximation of $f$ we need to consider does not have the same degree as $f$.

**Theorem 1.13.** *Given a polynomial $f \in \mathbb{R}[\bar{X}]$, arbitrary $r \in \mathbb{N}$, $\epsilon > 0$, and let $\Theta_r, \theta_r, f^1_{\epsilon r}, f^2_{\epsilon r} \in \mathbb{R}[\bar{X}]$ be the polynomials*

$$\Theta_r(\bar{x}) := 1 + \sum_{i=1}^{n} x_i^{2r}, \qquad \theta_r(\bar{x}) := \sum_{i=1}^{n} \sum_{k=0}^{r} \frac{x_i^{2k}}{k!}$$

$$f^1_{\epsilon r}(\bar{x}) := f(\bar{x}) + \epsilon\Theta_r(\bar{x}), \qquad f^2_{\epsilon r}(\bar{x}) := f(\bar{x}) + \epsilon\theta_r(\bar{x}).$$

*(i) If $f$ is non-negative on $[-1,1]^n$, then for every $\epsilon > 0$ there exists $r^1_\epsilon$ such that $f^1_{\epsilon r} \in \sum \mathbb{R}[\bar{X}]^2$ for all $r \geq r^1_\epsilon$ and $||f - f^1_{\epsilon r}||_1 \to 0$ as $\epsilon \to 0$ (and $r \geq r^1_\epsilon$).*

*(ii) If $f$ is non-negative, then for every $\epsilon > 0$ there exists $r^2_\epsilon$ such that $f^2_{\epsilon r} \in \sum \mathbb{R}[\bar{X}]^2$ for all $r \geq r^2_\epsilon$ and $||f - f^2_{\epsilon r}||_1 \to 0$ as $\epsilon \to 0$ (and $r \geq r^2_\epsilon$).*

As we can see, this theorem is a denseness result with respect to the $l_1$-norm of coefficients, and shows that a PSD polynomial can be approximated by SOS polynomials if the number of variables is fixed and the degree of the SOS polynomials is allowed to grow.

## 1.5 Positivity on Semialgebraic Sets

When we express a PSD polynomial $f \in \mathbb{R}[\bar{X}]$ as a sum of squares of polynomials, we obtain a certificate of positivity for $f$ in $\mathbb{R}^n$. However, more generally, we are interested in certificates of positivity or non-negativity on specific subsets $S \subseteq \mathbb{R}^n$. Typically, we focus on basic closed semialgebraic sets, which are subsets defined by a finite number of non-strict polynomial inequalities.

### 1.5.1 Semialgebraic Sets

At its core, algebraic geometry is the study of algebraic sets, which are the sets of common zeros of a collection of polynomials. Traditionally, this field examines subsets of $\mathbb{C}^n$ defined by polynomial equations, using methods rooted in algebra. Real algebraic geometry, on the other hand, focuses on finding solutions in $\mathbb{R}^n$ for polynomial equations defined over $\mathbb{R}$. In $\mathbb{C}$ there is no concept of "positivity" because it is not possible to define a relation $\geq$ that fulfills reasonable properties of positivity. However, in $\mathbb{R}$, the order relation $\geq$ exists, allowing for the study of semialgebraic sets, which are sets defined by polynomial inequalities.

There are multiple equivalent ways to define semialgebraic sets; we will present one.

**Definition 1.14.** *A semialgebraic set in $\mathbb{R}^n$ is a subset that can be expressed as*

$$\bigcup_{i=1}^{m} \bigcap_{j=1}^{k} \{\bar{x} \in \mathbb{R}^n | f_{i,j} *_{i,j} 0\},$$

*where* $f_{i,j} \in \mathbb{R}[\bar{X}]$ *and* $*_{i,j} \in \{<, =\}$*, for all* $i, j$*.*

Since for a polynomial $f \in \mathbb{R}[\bar{X}]$ we have

$$\{\bar{x} \in \mathbb{R}^n | f(\bar{x}) > 0\} = \{\bar{x} \in \mathbb{R}^n | -f(\bar{x}) < 0\},$$

$$\{\bar{x} \in \mathbb{R}^n | f(\bar{x}) \neq 0\} = \{\bar{x} \in \mathbb{R}^n | f(\bar{x}) < 0\} \cup \{\bar{x} \in \mathbb{R}^n | f(\bar{x}) > 0\},$$

$$\{\bar{x} \in \mathbb{R}^n | f(\bar{x}) \leq 0\} = \{\bar{x} \in \mathbb{R}^n | f(\bar{x}) < 0\} \cup \{\bar{x} \in \mathbb{R}^n | f(\bar{x}) = 0\},$$

$$\{\bar{x} \in \mathbb{R}^n | f(\bar{x}) \geq 0\} = \{\bar{x} \in \mathbb{R}^n | f(\bar{x}) > 0\} \cup \{\bar{x} \in \mathbb{R}^n | f(\bar{x}) = 0\},$$

then a semialgebraic set can be seen as a finite union of sets of solutions to systems of finitely many polynomial equations and inequalities $f * 0$, where $* \in \{<, >, =, \neq, \leq, \geq\}$.

**Example 1.15.** The set $\{(x, y) \in \mathbb{R}^2 | y = e^x\}$ is not semialgebraic. Since $e^x$ is a transcendental function, it cannot be expressed using polynomial equalities or inequalities.

**Example 1.16.** The set $\{(x, y) \in \mathbb{R}^2 | x^2 + y^2 \leq 1\}$ is semialgebraic.

**Definition 1.17.** *Given a finite family of polynomials* $g = \{g_1, \ldots, g_s\} \subseteq \mathbb{R}[\bar{X}]$*, the basic closed semialgebraic set generated by these polynomials is*

$$S(g) = \{\bar{x} \in \mathbb{R}^n | g_i(\bar{x}) \geq 0, \quad i = 1, \ldots, s\}.$$

Basic closed semialgebraic sets are crucial because many sets of interest in real algebraic geometry can be approximated arbitrarily well using finite unions and intersections of these sets.

### 1.5.2 Preorders, Quadratic Modules, and Archimedeanity

For the most part, certificates of positivity for polynomials on semialgebraic sets come from algebraic objects associated to the set. Two of these important objects are the preorders and quadratic modules.

Let $A$ be a commutative ring such that $\mathbb{Q} \subseteq A$, and let

$$\sum A^2 = \{a \in A | a = b_1^2 + \ldots + b_k^2\},$$

denote the set of sums of squares in $A$, for some $k \in \mathbb{N} \setminus \{0\}$ and $b_1, \ldots, b_k \in A$.

**Definition 1.18.** *A subset* $P$ *of* $A$ *is a preorder if*

$$P + P \subseteq P, \quad PP \subseteq P, \quad a^2 \in P \quad \forall a \in A.$$

**Definition 1.19.** *A subset* $M$ *of* $A$ *is a quadratic module if*

$$M + M \subseteq M, \quad a^2 M \subseteq M \quad \forall a \in A, \quad 1 \in M.$$

**Definition 1.20.** *Given a finite family of polynomials $g = \{g_1, \ldots, g_s\} \subseteq \mathbb{R}[\bar{X}]$, the preorder $P(g)$ generated by these polynomials is the smallest preorder containing $g$. Concretely, $P(g)$ consists of all the polynomials $f \in \mathbb{R}[\bar{X}]$ that can be expressed as*

$$f = \sum_{\epsilon=\{\epsilon_1,\ldots,\epsilon_s\}\in\{0,1\}^s} \sigma_\epsilon g_1^{\epsilon_1} \ldots g_s^{\epsilon_s},$$

*where each $\sigma_\epsilon \in \sum \mathbb{R}[\bar{X}]^2$.*

**Definition 1.21.** *Given a finite family of polynomials $g = \{g_1, \ldots, g_s\} \subseteq \mathbb{R}[\bar{X}]$, the quadratic module $M(g)$ generated by these polynomials is the smallest quadratic module containing $g$. Concretely, $M(g)$ consists of all the polynomials $f \in \mathbb{R}[\bar{X}]$ that can be expressed as*

$$f = \sigma_0 + \sigma_1 g_1 + \ldots + \sigma_s g_s,$$

*where $\sigma_0, \sigma_1, \ldots, \sigma_s \in \sum \mathbb{R}[\bar{X}]^2$.*

Note that a representation of a polynomial $f \in \mathbb{R}[\bar{X}]$ in the preorder $P(g)$ or the quadratic module $M(g)$ generated by $g_1, \ldots, g_s \in \mathbb{R}[\bar{X}]$, serves as a certificate of nonnegativity for $f$ on the basic closed semialgebraic set $S(g)$ defined by these polynomials.

There are more general ways to define archimedean quadratic modules from a commutative ring, taking into account preprimes and modules over preprimes. However, for the purposes of our study, we will define them over the ring $\mathbb{R}[\bar{X}]$.

**Definition 1.22.** *A quadratic module $M \subseteq \mathbb{R}[\bar{X}]$ is called archimedean if there exists $N \in \mathbb{R}_{>0}$ such that*

$$N - \sum_{i=1}^{n} X_i^2 \in M.$$

There are also alternative characterizations for the archimedeanity of quadratic modules in $\mathbb{R}[\bar{X}]$. For instance, a quadratic module $M \subseteq \mathbb{R}[\bar{X}]$ is archimedean if and only if there exists $N \in \mathbb{R}_{>0}$ such that $N \pm X_i \in M$ for $i = 1, \ldots, n$, or by the Jacobi-Prestel Criterion, which transfers this property to prime ideals and weakly isotropic regular parts from quadratic forms.

If a quadratic module $M(g)$ is archimedean, the basic closed semialgebraic set $S(g)$ will be bounded, and since $S(g)$ is closed, it will indeed be compact.

**Remark 1.23.** $S(g)$ being compact does not imply that $M(g)$ is archimedean.

We will provide some examples of this in chapter 3.

### 1.5.3 The Positivstellensatz

The Positivstellensatz is a key theorem in real algebraic geometry and serves as a representation theorem, involving denominators, for establishing positivity over any basic closed semialgebraic set. The term "Positivstellensatz", which translates from German as “positive-locus-theorem”, is generally used in real algebraic geometry for a theorem that characterizes polynomials that are strictly positive on a semialgebraic set. The name can be seen as the real analogue of Hilbert’s Nullstellensatz in complex algebraic geometry. Stengle published a version of the Positivstellensatz in 1974, although parts of the theorem were already noted by Dubois in 1969. It was later found that the core ideas were present in the work of the French mathematician Krivine as early as 1964. The theorem has multiple forms, and the next one is the most general version.

**Theorem 1.24.** *(The Positivstellensatz) Given $g = (g_1, \ldots, g_s)$ a finite subset of $\mathbb{R}[\bar{X}]$, and $S(g)$, $P(g)$ the respectives basic closed semialgebraic set and preorder. Then*

1. *$f > 0$ on $S(g) \iff$ there exist $p, q \in P(g)$ such that $pf = 1 + q$.*
2. *$f \geq 0$ on $S(g) \iff$ there exists an integer $m \geq 0$ and $p, q \in P(g)$ such that $pf = f^{2m} + q$.*
3. *$f = 0$ on $S(g) \iff$ there exists an integer $m \geq 0$ such that $-f^{2m} \in P(g)$.*
4. *$S(g) = \emptyset \iff -1 \in P(g)$.*

Observe that all of these provide certificates of positivity, though they include denominators, as in Artin’s Theorem. The proof of 1.24.1 result relies on many of the same concepts as the proof of Artin’s Theorem, such as properties of orderings on fields and the Tarski Transfer Principle. While we will not present the proof here, the interested reader can refer to Marshall’s book [13] for a detailed explanation. The proof that the four statements of the theorem are equivalent can be found in [2], Theorem 5.5.

In 1991, Schmüdgen demonstrated, as a corollary to a theorem on the multidimensional moment problem, that if a basic closed semialgebraic set $S(g)$ is compact, then every polynomial strictly positive on on $S(g)$ belongs to the preorder $P(g)$. In other words, certificates of positivity are guaranteed to exist for any polynomial strictly positive on a compact $S(g)$. This result, now widely known as Schmüdgen’s Positivstellensatz, is considered the first instance of a representation theorem without denominators for a broad class of semialgebraic sets. What makes this theorem particularly remarkable is that it ensures the existence of denominator-free certificates regardless of the specific polynomials $g_1, \ldots, g_s$ defining the semialgebraic set. A few years later, Putinar extended this result, showing that the preorder

$P(g)$ can be replaced with the quadratic module $M(g)$ as long as $M(g)$ is archimedean.

These certificates are highly significant because, under the necessary assumptions, verifying whether a polynomial is positive on a basic closed semialgebraic set reduces to checking whether the polynomial belongs to $P(g)$ or $M(g)$. We can implement algorithms capable of checking whether a polynomial lies in the preorder or quadratic module generated by a set of polynomials, so this problem can be solved.

Next, we present the main theorems along with a couple of additional results that provide bounds for the degree of the decomposition within the preorder and quadratic module, respectively. The proofs will not be included here, as they would considerably lengthen this work, but they can be found in [14], [15], [16], and [17].

**Theorem 1.25.** *(Schmüdgen's Positivstellensatz) Let $g = (g_1, \ldots, g_s)$ be a finite subset of $\mathbb{R}[\bar{X}]$ such that $S(g)$ is compact. Then for any $f \in \mathbb{R}[\bar{X}]$ such that $f > 0$ on $S(g)$, $f \in P(g)$.*

**Theorem 1.26.** *(Putinar's Positivstellensatz) Let $g = (g_1, \ldots, g_s)$ be a finite subset of $\mathbb{R}[\bar{X}]$ such that $M(g)$ is archimedean. Then for any $f \in \mathbb{R}[\bar{X}]$ such that $f > 0$ on $S(g)$, $f \in M(g)$.*

**Theorem 1.27.** *(Degree bound for Schmüdgen's Positivstellensatz) For all polynomials $g_1, \ldots, g_s \in \mathbb{R}[\bar{X}]$ defining a non-empty set $S(g) \subseteq (-r, r)^n$, there is some $c \in \mathbb{N}$ with the following property:*
*Every $f \in \mathbb{R}[\bar{X}]$ of degree d with $f^* := min\{f(\bar{x}) | \bar{x} \in S(g)\} > 0$ can be written as*

$$f = \sum_{\epsilon=\{\epsilon_1,\ldots,\epsilon_s\}\in\{0,1\}^s} \sigma_\epsilon g_1^{\epsilon_1} \ldots g_s^{\epsilon_s},$$

*where $\sigma_\epsilon \in \sum \mathbb{R}[\bar{X}]^2$ such that*

$$deg(\sigma_\epsilon g_1^{\epsilon_1} \ldots g_s^{\epsilon_s}) \leq cd^2 \left(1 + \left(d^2 n^d \frac{||f(r\bar{X})||}{f^*}\right)^c\right).$$

**Theorem 1.28.** *(Degree bound for Putinar's Positivstellensatz) For all polynomials $g_1, \ldots, g_s \in \mathbb{R}[\bar{X}]$ defining an archimedean quadratic module $M(g)$ and a non-empty set $S(g) \subseteq (-r, r)^n$, there is some $c \in \mathbb{R}_{>0}$ with the following property:*
*Every $f \in \mathbb{R}[\bar{X}]$ of degree d with $f^* := min\{f(\bar{x}) | \bar{x} \in S(g)\} > 0$ can be written as*

$$f = \sigma_0 g_0 + \sigma_1 g_1 + \ldots + \sigma_s g_s,$$

*where* $g_0 = 1$, *and* $\sigma_0, \sigma_1, \ldots, \sigma_s \in \sum \mathbb{R}[\bar{X}]^2$ *such that for all* $i = 0, \ldots, s$

$$deg(\sigma_i g_i) \leq c \, exp\left(\left(d^2 n^d \frac{||f(r\bar{X})||}{f^*}\right)^c\right).$$

## 1.6 Applications

The problem of recognizing non-negativity of multivariate polynomials and expressing them as sums of squares has numerous applications throughout mathematics and several fields such as chemistry or economics, among others. Recently, this area has garnered significant attention due to its wide-ranging uses in computational mathematics and the discovery that optimization over a notable subset of non-negative polynomials can be achieved through sum of squares techniques. By exploring the properties of polynomials and their representations, we can address questions from different areas and contexts.

A famous and important application of certificates of positivity is the problem of minimizing a polynomial on a compact basic closed semialgebraic set, which is a hard problem in general. However, since this topic has been extensively studied in the literature, we will shift our attention to other uses instead. Specifically, in this section, we will explore a couple of applications in machine learning (ML) where the theoretical framework of positive polynomials and sums of squares plays a crucial role. It should be emphasized that the theorems presented here will not be proven, as doing so would extend this section too much and the main objective is not to understand them in detail, but to illustrate why the problem tackled in this work is relevant nowadays. Nevertheless, all the theorems can be found in [18].

### 1.6.1 Shape-Constrained Regression Problem

To begin with, we introduce the Shape-Constrained Regression Problem. Suppose we have a set of data points and aim to fit a polynomial regressor to these points. While we could minimize the least squares, our main interest is to impose certain shape constraints on these polynomial regressors (e.g., monotonicity, convexity, concavity).

An example application: Imagine you have two cars that are identical in all aspects except for one feature, the age. An older car should logically be priced lower than a newer one, implying a monotonic relationship with that feature. This approach can also be applied to scenarios like determining interest rates for student loans, and other similar contexts.

How does this connect with optimizing over non-negative polynomials? We want a polynomial regressor to fit the data, and the derivatives of these polynomials will also be polynomials. Imposing monotonicity on a polynomial regressor over a range is the same as imposing the non-negativity of its partial derivatives over that range. Similarly, if we

want our polynomial regressor to be convex, imposing convexity is equivalent to requiring that the Hessian[1] of the polynomial be PSD, which, in turn, is equivalent to imposing the non-negativity of a polynomial (since the elements of the Hessian are polynomials):

$$H(x) \succeq 0 \quad \forall x \in \mathbb{R}^n \iff y^T H(x) y \geq 0 \quad \forall x, y \in \mathbb{R}^n.$$

Now, let us delve deeper into monotone regression. The setup is as follows: We have $N$ data points $(x_i, y_i)$, where $x_i \in \mathbb{R}^n$ represents the feature vector and $y_i \in \mathbb{R}$ is the response. Assume that there is an underlying function $f$ that generates these points $y_i$, with some added noise. That is, $y_i$ represents the noisy measurements of a monotone function:

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i > 0,$$

and also assume that our feature vector $x_i$ belongs to a box $B \subseteq \mathbb{R}^n$. In addition, we will assume that we have a so-called monotonicity profile:

$$\rho_j = \begin{cases} 1 & \text{if } f \text{ is monotonically increasing w.r.t. } x_j, \\ -1 & \text{if } f \text{ is monotonically decreasing w.r.t. } x_j, \\ 0 & \text{if no monotonicity requirements on } f \text{ w.r.t. } x_j, \end{cases} \qquad \text{for } j = 1, \ldots, n.$$

This means we know how our function $f$ depends on the features. We have a vector where each entry indicates how $f$ depends on a specific feature. The main goal is then to fit a polynomial to the data that maintains the monotonicity profile $\rho$ over $B$.

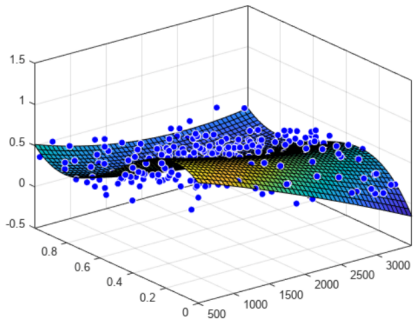



Figure 1.1: Polynomial regressor.

Our first result, however, is not very promising.

**Theorem 1.29.** *Given a cubic polynomial $p$, a box $B$, and a monotonicity profile $\rho$, it is NP-hard to test whether $p$ has profile $\rho$ over $B$.*

[1]The Hessian is a matrix where the entry of the i-th row and the j-th column is: $(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \, \partial x_j}$.

Nevertheless, we have an SOS relaxation for this problem. Suppose we want to enforce that our polynomial is increasing with respect to the variable $x_j$. What we want is that

$$\frac{\partial p(x)}{\partial x_j} \geq 0 \quad \forall x \in B,$$

with $B = [b_1^-, b_1^+] \times \ldots \times [b_n^-, b_n^+]$. If $p$ has an odd degree, we will relax this condition as follows:

$$\frac{\partial p(x)}{\partial x_j} = \sigma_0(x) + \sum_i \sigma_i(x)(b_i^+ - x_i)(x_i - b_i^-),$$

and if $p$ has an even degree, we will relax it this way:

$$\frac{\partial p(x)}{\partial x_j} = \sigma_0(x) + \sum_i \sigma_i(x)(b_i^+ - x_i) + \sum_i \tau_i(x)(x_i - b_i^-),$$

where $\sigma_i, \tau_i$ are SOS polynomials.

Obviously, these expressions ensure the positivity of the partial derivative of $p$ with respect to the variable of interest in the box $B$. Additionally, we have an approximation theorem that tells us how good our potential approximation is.

**Theorem 1.30.** *For any $\epsilon > 0$ and any $C^1$ function $f$ with monotonicity profile $\rho$, there exists a polynomial $p$ with the same profile $\rho$, such that*

$$\max_{x \in B} \quad |f(x) - p(x)| < \epsilon.$$

*Moreover, one can certify its monotonicity profile using SOS.*

The proof of this theorem uses results from approximation theory and Putinar's Positivstellensatz (1.26).

### 1.6.2 Difference of Convex (DC) Programming

On the other hand, there is another type of problem that focuses on polynomial optimization, known as Difference of Convex (DC) Programming. These are problems of the folowing form:

$$\begin{aligned} \min \quad & f_0(x) \\ s.t. \quad & f_i(x) \leq 0 \end{aligned}$$

where $f_i := g_i(x) - h_i(x)$, with $g_i, h_i$ convex.
The question we are interested in is the following: Given a polynomial $f$, can we find two

convex polynomials $g, h$ such that $f = g - h$?

First of all, we need to understand what SOS-convexity is. We know that a polynomial $f(x)$ is convex if and only if we impose the condition $y^T H_f(x) y \geq 0, \forall x, y \in \mathbb{R}^n$, where $H_f$ is the Hessian. Now, we can replace the non-negativity condition with a sum of squares condition, which is what we call SOS-convexity ($y^T H_f(x) y$ SOS). This is advantageous because it can be checked through an SDP program. Thus, the theorem we actually have is the following:

**Theorem 1.31.** *Any polynomial can be written as the difference of two SOS-convex polynomials.*

**Corollary 1.32.** *Any polynomial can be written as the difference of two convex polynomials.*

Then, we have seen that such a decomposition always exists, and in fact, we have shown that sum of squares decomposition plays a key role here.

Both of these types of problems appear in ML applications such as Sparse PCA, kernel selection, or feature selection in SVMs.

A concrete example of using Sparse PCA with shape constraints could be gene expression data analysis in the context of a disease like cancer. Suppose we have a dataset of gene expression, where each variable represents the expression of a specific gene in different cancer patient samples.

The goal of this analysis is to identify a subset of genes whose expression is strongly associated with the disease state, such as cancer progression. To achieve this, Sparse PCA could be used to reduce the dimensionality of the data and find linear combinations of genes that explain the most variability observed. This is useful because, generally, most genes are not relevant to the disease, and Sparse PCA helps identify only those genes that significantly impact cancer-related variability.

However, for the results to be biologically interpretable, the combinations of genes found need to follow certain structural constraints. For example, we might expect certain genes to have an increasing influence on cancer progression as the disease advances, while others might have a decreasing influence. This pattern could reflect the expected biological behavior in the disease process, making it necessary to apply shape constraints.

In this regard, a shape constraint could be added to force the principal components derived from Sparse PCA to respect an increasing or decreasing structure in the relationship

between the genes. This means that, when searching for the most relevant gene combinations, the model would not only seek those that maximize the data's variability, but also ensure that these combinations follow a trajectory consistent with the biological behavior observed in cancer. In this way, the combination of Sparse PCA with shape constraints enables the discovery of a more interpretable set of genes. Not only are the most influential genes in the data variability identified, but also their relationships follow an expected biological pattern.

# Chapter 2

# Rational Certificates of Non-negativity on Semialgebraic Subsets of Cylinders

In this chapter, we provide a generalization of the work presented in [9] (so all references to remarks or assumptions herein correspond to it). As we will see, this extension makes sense from a theoretical point of view, but not from a computational one. Nevertheless, even though it is not implementable (or at least easy to do), it is interesting to explore where it leads from a theoretical approach. We also address certain aspects that arise during the work, such as the explicit calculation of the bound in Remark 4.

## 2.1 Contextual Frame and Further Insights

As we have seen, the certificates of positivity and non-negativity have been moving towards sets with different characteristics, such as closed, compact, or Archimedean semialgebraic sets. There are also studies addressing the non-negativity of polynomials in non-compact sets. In this work, the authors study non-negativity in non-empty and possibly unbounded subsets of cylinders, based on a rational expression having as the numerator a polynomial in the quadratic module generated by $g_1, \ldots, g_s \in \mathbb{R}[\bar{X}, Y]$, and as the denominator a power of a non-constant polynomial $q \in \mathbb{R}[Y]$ that is positive on $\mathbb{R}$.

Other previous contributions in the literature have approached this type of set in different ways, as well as investigations that have explored the existence of rational certificates with the denominator being a power of a fixed particular polynomial. References to these can be found in the introductory section of [9].

Now, let us delve into some more concrete aspects of the study, showing the explicit calculation for the bound of $M$ in Remark 4.

We should keep in mind, first, from the expression of $h$ in Proposition 3, that $M$ and the $M_i$'s let us homogenize this polynomial in $(Y, Z)$ depending on the degree. Therefore, if $deg(h) = m$, $M$ would be 0 and we would use each $M_i$ to ensure homogeneity. Otherwise, if $deg(h) = r + (2k+1)(m_i + e_i)$ for some $i$, then we would use $M$ to homogenize the first part of $h$, and each of the remaining $M_i$'s for the homogenization of the second part.

Note that, in particular, for each $i$ the difference between $m$ and $r + (2k+1)(m_i + e_i)$ will always be a multiple of $m_0$: Since $r$ is the remainder of $m$ in the division by $m_0$, we can write $m = C_1 m_0 + r$, and since $m_0$ divides $m_i + e_i$ for each $i$, we can express $m_i + e_i = C_{2_i} m_0$, where $C_1, C_{2_i} \in \mathbb{Z}$.

Thus

$$m - (r + (2k+1)(m_i + e_i)) = C_1 m_0 + r - (r + (2k+1)C_{2_i} m_0) =$$
$$= C_1 m_0 - (2k+1)C_{2_i} m_0 = m_0(C_1 - (2k+1)C_{2_i}).$$

If $m \geq r + (2k+1)(m_i + e_i)$ for each $i$, then $C_1 - (2k+1)C_{2_i} \geq 0$ since $m_0 > 0$. The case where $m < r + (2k+1)(m_i + e_i)$ is analogous, and the same holds when computing the difference $r + (2k+1)(m_i + e_i) - (r + (2k+1)(m_j + e_j))$, with $i \neq j$.

Now, if we are in the case where $deg(h) = r + (2k+1)(m_i + e_i)$ for some $i$, then we want $m_0 M + m = \max_i(r + (2k+1)(m_i + e_i)) = r + (2k+1)\max_i(m_i + e_i)$. From the proof of Proposition 3 we know that $k \geq \frac{2\lambda\rho_2^r s}{ef^\bullet}$ and $\lambda \geq KD\frac{c}{\rho_1^r}(\frac{4K}{f^\bullet})^L$. Then,

$$m_0 M + m \geq r + (\frac{4\lambda\rho_2^r s}{ef^\bullet} + 1)\max_i(m_i + e_i) \geq$$

$$\geq r + (\frac{4\rho_2^r sKDc(4K)^L}{ef^\bullet\rho_1^r(f^\bullet)^L} + 1)\max_i(m_i + e_i) = r + (\frac{\rho_2^r sDc(4K)^{L+1}}{e\rho_1^r(f^\bullet)^{L+1}} + 1)\max_i(m_i + e_i).$$

Therefore,

$$M \geq (r + (\frac{\rho_2^r sDc(4K)^{L+1}}{e\rho_1^r(f^\bullet)^{L+1}} + 1)\max_i(m_i + e_i) - m)\frac{1}{m_0}.$$

Having checked Assumptions 1, 2 and 3, certifying that $f$ is non-negative on $S$ is equivalent to checking whether $q^M f$ belongs to $M(g)$, taking an $M$ as computed here. As we will see in the next chapter, we have a code to determine whether a polynomial lies in the quadratic module generated by a set of polynomials, so this task can be accomplished.

## 2.2 Certificates for Generalized Cylinders

Let us now consider $g_1, \ldots, g_s \in \mathbb{R}[\bar{X}, \bar{Y}] = \mathbb{R}[X_1, \ldots, X_n, Y_1, \ldots, Y_l]$,

$$S = \{(\bar{x}, \bar{y}) \in \mathbb{R}^{n+l} | g_1(\bar{x}, \bar{y}) \geq 0, \ldots, g_s(\bar{x}, \bar{y}) \geq 0\},$$

and the quadratic module generated by $g := (g_1, \ldots, g_s)$,

$$M(g) = \{\sigma_0 + \sigma_1 g_1 + \ldots + \sigma_s g_s | \sigma_0, \sigma_1, \ldots, \sigma_s \in \sum \mathbb{R}[\bar{X}, \bar{Y}]^2\}.$$

Let ***Assumption 1*** be the same: There exists $N \in \mathbb{R}_{>0}$ such that

$$N - \sum_{j=1}^{n} X_j^2 \in M(g).$$

Under this assumption $S \subset B \times \mathbb{R}^l$, where

$$B := \{\bar{x} \in \mathbb{R}^n | \sum_{j=1}^{n} X_j^2 \leq N\}.$$

Now, for $i = 1, \ldots, s$, we write

$$g_i(\bar{X}, \bar{Y}) = \sum_{|k| \leq m_i} g_{ik}(\bar{X})\bar{Y}^k = \sum_{|k| \leq m_i} g_{ik}(\bar{X}) Y_1^{k_1} \ldots Y_l^{k_l} \in \mathbb{R}[\bar{X}, \bar{Y}],$$

where $k = (k_1, \ldots, k_l) \in \mathbb{N}^l$, $|k| = k_1 + \ldots + k_l$, and $g_{ik}(\bar{X}) \neq 0$ when $|k| = m_i$.

Then, we write

$$\widetilde{g_i}(\bar{X}, \bar{Y}, Z) := Z^{m_i} g_i(\bar{X}, \bar{Y}/Z) = \sum_{|k| \leq m_i} g_{ik}(\bar{X})\bar{Y}^k Z^{m_i - |k|} \in \mathbb{R}[\bar{X}, \bar{Y}, Z]$$

for the homogenization of $g_i$ with respect to the variables $Y_1, \ldots, Y_l$.

We now make the following ***Assumption 2***:

(i) $S \neq \emptyset$.

(ii) If $|k| = m_i$, then $k_1, \ldots, k_l$ are even.

(iii) $S_\infty := \{\bar{x} \in \mathbb{R}^n | g_{ik}(\bar{x}) \geq 0, |k| = m_i\} \subset B$.

Let $q \in \mathbb{R}[\bar{Y}]$ be a non-constant polynomial which is positive on $\mathbb{R}^l$,

$$q(\bar{Y}) = \sum_{|k| \leq m_0} q_k \bar{Y}^k$$

with $m_0 > 0$ and $q_k \neq 0$ when $|k| = m_0$. The assumption of $q$ being positive on $\mathbb{R}^l$ implies $m_0$ is even. We write

$$\widetilde{q}(\bar{Y}, Z) := Z^{m_0} q(\bar{Y}/Z) = \sum_{|k| \leq m_0} q_k \bar{Y}^k Z^{m_0 - |k|} \in \mathbb{R}[\bar{Y}, Z].$$

Consider now the sets

$$C := \{(\bar{y}, z) \in \mathbb{R}^{l+1} | \widetilde{q}(\bar{y}, z) = 1\}$$

and

$$\widetilde{S} := \{(\bar{x}, \bar{y}, z) \in \mathbb{R}^{n+l+1} | \widetilde{g_1}(\bar{x}, \bar{y}, z) \geq 0, \ldots, \widetilde{g_s}(\bar{x}, \bar{y}, z) \geq 0, (\bar{y}, z) \in C\}.$$

Since we can write each element of $C$ as $u = \lambda(u)v(u) \in \mathbb{R}^{l+1}$, with $\lambda(u) = ||u||$ and $v(u) = \frac{u}{||u||} = (v_1, \ldots, v_l, v_{l+1}) \in \mathbb{S}^l$, to see that $C$ is compact it is enough to see that $\lambda(u) = ||u||$ is bounded.

We have

$$\widetilde{q}(||u||v(u)) = \sum_{|k| \leq m_0} q_k (||u||v_1)^{k_1} \ldots (||u||v_l)^{k_l} (||u||v_{l+1})^{m_0 - |k|} = ||u||^{m_0} \widetilde{q}(v(u)).$$

Then, enforcing $\widetilde{q}(||u||v(u)) = 1$, we get

$$||u||^{m_0} \widetilde{q}(v(u)) = 1 \Leftrightarrow ||u||^{m_0} = \frac{1}{\widetilde{q}(v(u))} \leq \max_{v(u) \in \mathbb{S}^l} \frac{1}{\widetilde{q}(v(u))} =: \widetilde{q}_0,$$

which exists since a continuous function on a compact set attains its minimum and maximum values.

From the last inequality, $||u||^{m_0} \leq \widetilde{q}_0 \Leftrightarrow ||u|| \leq (\widetilde{q}_0)^{\frac{1}{m_0}}$, that give us the bound and let us see that $C$ is compact.

Now, on one hand $\widetilde{S} \cap \{z \neq 0\} \subset B \times C$, since $\widetilde{g}_i(\bar{x}, \bar{y}, z) = z^{m_i} g_i(\bar{x}, \bar{y}/z)$ will be non-negative if $g_i(\bar{x}, \bar{y}/z)$ is non-negative (Assumption 2 ensures that $m_i$ is even, as $k_1, \ldots, k_l$ being all even implies that $m_i = k_1 + \ldots + k_l$ is also even), and $g_i$ is non-negative on S. But $S \subset B \times \mathbb{R}^l$ by Assumption 1, and by definition of $\widetilde{S}$, $(\bar{y}, z) \in C$. On the other hand, $\widetilde{S} \cap \{z = 0\} \subset B \times C$, since in this case we will have $\widetilde{g}_i(\bar{x}, \bar{y}, 0) = \sum_{|k|=m_i} g_{ik}(\bar{x}) y_1^{k_1} \ldots y_l^{k_l}$, which is non-negative by Assumption 2.

We conclude that $\widetilde{S} \subset B \times C$ and therefore $\widetilde{S}$ is compact.

Also, for a polynomial

$$f(\bar{X}, \bar{Y}) = \sum_{|k| \leq m} f_k(\bar{X}) \bar{Y}^k \in \mathbb{R}[\bar{X}, \bar{Y}]$$

with $f_k(\bar{X}) \neq 0$ when $|k| = m$, we write

$$\widetilde{f}(\bar{X}, \bar{Y}, Z) := Z^m f(\bar{X}, \bar{Y}/Z) = \sum_{|k| \leq m} f_k(\bar{X}) \bar{Y}^k Z^{m-|k|} \in \mathbb{R}[\bar{X}, \bar{Y}, Z]$$

for the homogenization of $f$ with respect to the variables $Y_1, \ldots, Y_l$.

We make the following assumptions on the polynomial $f$ as the ***Assumption 3***.
If $|k| = m$, then:

(i) $k_1, \ldots, k_l$ are even.

(ii) $f_k(\bar{x}) > 0$ on $S_\infty$.

Now, let us consider the same set $\Delta \subset \mathbb{R}^n$. Lemma 2 still works (see Appendix A), and Proposition 3 also holds taking

$$h(\bar{X}, \bar{Y}, Z) = \widetilde{q}(\bar{Y}, Z)^M \widetilde{f}(\bar{X}, \bar{Y}, Z) -$$
$$- \lambda(Y_1^2 + \ldots + Y_l^2 + Z^2)^{\frac{r}{2}} \sum_{i=1}^{s} \alpha_i (Y_1^2 + \ldots + Y_l^2 + Z^2)^{\frac{e_i}{2}} \widetilde{g}_i(\bar{X}, \bar{Y}, Z)$$
$$(\alpha_i (Y_1^2 + \ldots + Y_l^2 + Z^2)^{\frac{e_i}{2}} \widetilde{g}_i(\bar{X}, \bar{Y}, Z) - \widetilde{q}(\bar{Y}, Z)^{\frac{m_i + e_i}{m_0}})^{2k} \widetilde{q}(\bar{Y}, Z)^{M_i}.$$

Moreover, Lemmas 5 and 6 are still applicable since they only involve the variables $X_1, \ldots, X_n$.

Now, proceeding with the proof of the main theorem as in [9], we apply Lemma 5. Then, we will have

$$h(\bar{X}, \bar{Y}, Z) = \sum_{|\beta| \leq \kappa} b_\beta(\bar{Y}, Z) \bar{l}(\bar{X})^\beta,$$

with $b_\beta(\bar{Y}, Z) > 0$ for all $(\bar{Y}, Z) \in C$. Furthermore, for each $\beta$, since $h$ is homogeneous in $(\bar{Y}, Z)$ and $b_\beta$ is a linear combination of the coefficients of $h$ seen as a polynomial in $\bar{X}$, then $b_\beta$ is a homogeneous polynomial. Thus, $b_\beta(\bar{Y}, Z) > 0$ for every $(\bar{Y}, Z) \in \mathbb{R}^{l+1} \setminus \{0\}$, and applying Artin's Theorem related to Hilbert's seventeenth problem (explained in Section 1.2), there exist $\delta_\beta, \delta_{1_\beta}, \ldots, \delta_{r_\beta} \in \mathbb{R}[\bar{Y}, Z]$ with $\delta_\beta \neq 0$, and $r \geq 1$, such that

$$\delta_\beta^2 b_\beta = \delta_{1_\beta}^2 + \ldots + \delta_{r_\beta}^2.$$

Additionally, as proved by Reznick [10], since $f$ is PD the denominator $\delta_\beta$ can be chosen uniformly to be a power of $Y_1^2 + \ldots + Y_l^2 + Z^2$.

In other words, for each $b_\beta$ there exists $\delta_\beta \neq 0 \in \mathbb{R}[\bar{Y}, Z]$ such that $\delta_\beta^2(\bar{Y}, Z) b_\beta(\bar{Y}, Z)$ is SOS. In particular, $\delta_\beta^2(\bar{Y}, 1) b_\beta(\bar{Y}, 1)$ will be SOS.

Now, by the equality from Proposition 3,

$$\delta_{\beta_1}^2(\bar{Y}, 1) \ldots \delta_{\beta_t}^2(\bar{Y}, 1) h(\bar{X}, \bar{Y}, 1) = \delta_{\beta_1}^2(\bar{Y}, 1) \ldots \delta_{\beta_t}^2(\bar{Y}, 1) q(\bar{Y})^M f(\bar{X}, \bar{Y}) -$$
$$- \delta_{\beta_1}^2(\bar{Y}, 1) \ldots \delta_{\beta_t}^2(\bar{Y}, 1) \lambda \sum_{i=1}^{s} \alpha_i (Y_1^2 + \ldots + Y_l^2 + 1)^{\frac{r+e_i}{2}} g_i(\bar{X}, \bar{Y})$$
$$(\alpha_i (Y_1^2 + \ldots + Y_l^2 + 1)^{\frac{e_i}{2}} g_i(\bar{X}, \bar{Y}) - q(\bar{Y})^{\frac{m_i + e_i}{m_0}})^{2k} q(\bar{Y})^{M_i},$$

and therefore

$$\delta^2_{\beta_1}(\bar{Y},1)\dots\delta^2_{\beta_t}(\bar{Y},1)q(\bar{Y})^M f(\bar{X},\bar{Y}) = \delta^2_{\beta_1}(\bar{Y},1)\dots\delta^2_{\beta_t}(\bar{Y},1)\sum_{|\beta|\leq\kappa} b_\beta(\bar{Y},1)\bar{l}(\bar{X})^\beta +$$

$$+\,\delta^2_{\beta_1}(\bar{Y},1)\dots\delta^2_{\beta_t}(\bar{Y},1)\lambda\sum_{i=1}^{s}\alpha_i(Y_1^2+\ldots+Y_l^2+1)^{\frac{r+e_i}{2}}g_i(\bar{X},\bar{Y})$$

$$(\alpha_i(Y_1^2+\ldots+Y_l^2+1)^{\frac{e_i}{2}}g_i(\bar{X},\bar{Y}) - q(\bar{Y})^{\frac{m_i+e_i}{m_0}})^{2k}q(\bar{Y})^{M_i}.$$

Looking at the second term on the right-hand side of the last equation, since $\frac{r+e_i}{2}$ and $2k$ are even, if we make the assumption that $q(\bar{Y}) \in \mathbb{R}[\bar{Y}]$ is SOS, then this term is in $M(g)$. Let us now see that the first term on the right-hand side is also in $M(g)$.

We know that each term of the sum will be of the form:

$$\delta^2_{\beta_1}(\bar{Y},1)\dots\delta^2_{\beta_t}(\bar{Y},1)b_\beta(\bar{Y},1)\bar{l}(\bar{X})^\beta,$$

where $\delta^2_{\beta_i}(\bar{Y},1)b_\beta(\bar{Y},1)$ will be SOS, for some $1 \leq i \leq t$. Then, we will have an SOS polynomial in $(\bar{Y},1)$ multiplied by

$$\prod_{j\neq i}\delta^2_{\beta_j}(\bar{Y},1)\bar{l}(\bar{X})^\beta.$$

It is clear that the result of the SOS polynomial in $(\bar{Y},1)$ multiplied by the productory will still be SOS in $(\bar{Y},1)$, so we have to see that when multiplying by $\bar{l}(\bar{X})^\beta$ we still have an SOS polynomial.

By Lemma 6,

$$l_0(\bar{X}),\dots,l_n(\bar{X}) \in M(N-||\bar{X}||^2),$$

and since $M(N-||\bar{X}||^2)$ is closed under multiplication (it is generated by a single polynomial), the same holds for all the products $\bar{l}(\bar{X})^\beta = l_0(\bar{X})^{\beta_0}\dots l_n(\bar{X})^{\beta_n}$. Now, by Assumption 1 we deduce that $\delta^2_{\beta_1}(\bar{Y},1)\dots\delta^2_{\beta_t}(\bar{Y},1)b_\beta(\bar{Y},1)\bar{l}(\bar{X})^\beta \in M(g)$ for every $\beta$ with $|\beta| \leq \kappa$. We conclude that $\delta^2_{\beta_1}(\bar{Y},1)\dots\delta^2_{\beta_t}(\bar{Y},1)q(\bar{Y})^M f(\bar{X},\bar{Y}) \in M(g)$.

Thus, our generalized theorem is the following:

**Theorem 2.1.** *Let $g := g_1,\dots,g_s$ and $f$ be polynomials in $\mathbb{R}[\bar{X},\bar{Y}]$ such that $f > 0$ on $S$ and Assumptions 1, 2 and 3 hold. Let $q \in \mathbb{R}[\bar{Y}]$ be a non-constant polynomial which is a sum of squares. Then, there exist $M \in \mathbb{Z}_{\geq 0}$ and $p \in \mathbb{R}[\bar{Y}]$ such that $pq^M f \in M(g)$.*

Since $q$ is a sum of squares, multiplying on both sides by $q$ if necessary, we may assume that $M$ is even. Additionally, $p$ has been shown to be a product of squares of polynomials. Then, the identity

$$f = \frac{\sigma_0+\sigma_1 g_1+\ldots+\sigma_s g_s}{pq^M},$$

with $\sigma_0, \sigma_1, \ldots, \sigma_s \in \sum \mathbb{R}[\bar{X}, \bar{Y}]^2$, is a rational certificate of non-negativity of $f$ on $S$.

The infeasibility of implementing this generalization is mainly due to the polynomial $p$. This polynomial is constructed as a product of the squares of several polynomials, which are difficult to compute. First, we would need to determine the polynomials $b_\beta$ that appear when applying the version of Polya's Theorem, and then, we would also need to find certain bounds in order to compute the polynomial $\delta_\beta$ from $b_\beta$ for each $\beta$, using Reznick's result.

# Chapter 3

# Algorithms and Implementation

In this chapter we will explain the algorithms implemented and provide examples of their execution. Rather than going through the code line by line (as the code files are available in the repository[1]), we will focus on detailing all the relevant information needed to understand the workflow and logic behind the algorithms. Additionally, we will discuss the key decisions made during the implementation process and highlight any challenges encountered. This will provide a comprehensive overview of the algorithms structure and functionality. We recommend having the codes nearby while reading this chapter for better understanding.

## 3.1 Preliminaries

First of all, let us explain some details about how mathematical objects are represented in the code. The primary objects of study are polynomials, which are represented using lists of tuples. Each tuple in the list corresponds to a coefficient-monomial pair, where the monomial is represented as an $n$-tuple containing the exponents of the variables involved. In the case of univariate polynomials, the representation is given as a bivariate polynomial, with the second coordinate of the tuple always equal to 0.

For instance, if we want to represent the polynomial

$$f(x_1, x_2, x_3, x_4) = 2 + 9x_1^3x_2x_4^7 - 61x_2x_3 + x_1 \in \mathbb{Q}[x_1, x_2, x_3, x_4],$$

we would write

$$f = [(2, (0, 0, 0, 0)), (9, (3, 1, 0, 7)), (-61, (0, 1, 1, 0)), (1, (1, 0, 0, 0))].$$

Our algorithms follow Lasserre's approach described in Section 1.3, which requires the use of an SDP solver. In our case, we employ the SCS solver [19], though we have explored

[1] GitHub repository.

other options, as will be discussed below. This solver is accessed via a library that facilitates the construction of the problem by incorporating the necessary constraints and variables in a format compatible with the solver. Specifically, we use the CVXPY library [21]. CVXPY also supports other solvers to address diverse needs, but SCS stands out as one of the best suited to our requirements.

Apart from SCS, we have also tested the Mosek solver [20], with detailed tests provided in the Jupyter Notebooks accompanying the code files. Although Mosek is not free, it offers time-limited license options for students. Mosek appears to be significantly more accurate and faster than SCS, and has the notable advantage of supporting parallelization, making it an attractive option.

However, we have also observed that for the types of problems we submit to the solver, there are several instances where Mosek fails, and the reasons for these failures are unclear. In such cases, SCS continues to work but with reduced accuracy. As the complexity of the problem increases—measured by the size of $v_d$, which depends on the degree of the polynomials involved and the number of variables—the frequency of Mosek failures also rises. These failures do not appear to be random, as cases where a failure occurs will fail again upon re-execution. For this reason, we have chosen to use SCS. While SCS is less accurate, it seems to be more robust in this regard, and the level of accuracy it provides is sufficient for our purposes.

Solvers are general tools designed to handle a variety of problem types. Ideally, one could delve deeper into studying how solvers work internally, or even attempt to implement one from scratch. This would provide valuable insights into how they solve our specific type of problems and allow for a more accurate determination of thresholds when interpreting solver outputs compared to expected solutions. However, this approach would require significant time and effort, potentially constituting an entirely separate project.

It is important to keep in mind that working with numerical solvers yields only approximations. For this reason, considering the accuracy of the solver we are using, we must determine when an approximation is sufficiently accurate to be deemed correct and when it is not. By default, unless stated otherwise, we will consider a number to be zero if it is strictly smaller than 0.00001. In other words, the largest number treated as zero is $0.00000\overline{9}$. Despite not being perfect, this threshold seems to perform the best, and has been determined empirically, based on examples where we knew what the expected outcome should be.

After obtaining an approximation from the solver, we check its validity by comparing the coefficients of the input polynomial with those obtained through the algorithm. Specifically, for each monomial, we verify whether the difference between the real coefficient and the approximated coefficient is smaller than 0.00001. If this condition is not satisfied for at least one coefficient, the result is deemed incorrect.

This precision threshold can, of course, be adjusted: it can be relaxed if less precision

is acceptable or tightened if higher accuracy is required.

There are studies focused on obtaining exact representations through perturbations, such as [22], but since our primary objective was to construct an SOS decomposition algorithm as a step toward developing the quadratic module decomposition and, ultimately, the archimedeanity algorithm, we chose not to invest time in implementing an exact SOS representation algorithm. Clearly, the algorithms presented here could be enhanced with exact decomposition techniques, resulting in more precise solutions. However, this was not the main focus of our work and can be pursued in future research.

Through testing, we have observed that, in our context, when working with at most 5 polynomials generating the quadratic module, the solver we are using remains effective up to the point where the vectors $v_d$ contain approximately 50 elements. Beyond this point, the solver becomes too slow to be practical. This implies that the size of these vectors, given by $s(d) = \binom{n+d}{d}$, should not exceed this threshold. For example, with 2 variables, we can test up to approximately degree 9, meaning that the $\sigma$'s in the quadratic module decomposition could have a maximum degree of 18. Alternatively, with 3 variables, we could test up to approximately degree 5, meaning that the $\sigma$'s could have a maximum degree of 10.

Furthermore, if we were able to use the Mosek solver, which is more efficient, the size of the vectors $v_d$ could increase to approximately 250 elements. This would allow us to work with significantly higher degrees in the decomposition.

It is also worth emphasizing at this point that when we talk about degrees in the decomposition, we are usually referring to the maximum degree of the monomials in the vector $v_d$ used to decompose each $\sigma$. Therefore, it should be multiplied by 2 to obtain the actual degree of the $\sigma$ polynomials in the decomposition.

## 3.2 Sums of Squares

This algorithm takes a rational multivariate polynomial as input. If the polynomial can be expressed as a sum of squares of rational polynomials, the algorithm returns a possible decomposition. Otherwise, it outputs a message indicating that the polynomial cannot be expressed as a sum of squares.

This is a well-known algorithm and can be found in various libraries or software packages, such as [23] or [24]. However, we decided to build it from scratch to gain a deeper understanding of the logic behind it and, as mentioned earlier, to enable the construction of the quadratic module and archimedeanity algorithms later on. This algorithm is fully implemented in Python and can be found in the **SumsOfSquares.py** file, accompanied by thorough documentation.

To begin with, the algorithm calculates the vector $v_d(\bar{X})$ based on the degree $2d$ of the input polynomial. Then, following the notation introduced in the last part of Section 1.3, it generates a matrix $Q$ of variables and adds the constraint that $Q$ must be symmetric and PSD. Next, it computes the decomposition

$$v_d(\bar{X})v_d(\bar{X})' = \sum_{\alpha \in \mathbb{N}^n} B_\alpha \bar{X}^\alpha,$$

obtaining the matrices $B_\alpha$ corresponding to each monomial. The algorithm then iterates over the monomials in this decomposition and adds the constraints related to the trace of the product of the matrix associated with each monomial and the matrix of variables. Finally, the solver is called to solve the optimization problem defined by these variables and constraints. If the solver returns a solution matrix $Q$, a decomposition $Q = HH'$ is computed, and a sum of squares decomposition is immediately obtained, as outlined in the proof of Theorem 1.11. If no solution exists, a message is displayed indicating that such a decomposition cannot be constructed.

After completing this process, a verification method is invoked to ensure that the solution provided by the solver is sufficiently accurate.

**Example 3.1.** Let us consider the univariate polynomial $f(x) = x^4 + x^3 + x^2 + x + 1$. When we input this polynomial into our algorithm, it outputs the decomposition:

$$\begin{aligned} f(x) = &(-8.00752981 \times 10^{-1} - 6.34367731 \times 10^{-1}x - 8.00752981 \times 10^{-1}x^2)^2 + \\ &+ (5.96353066 \times 10^{-1} + 4.68164726 \times 10^{-17}x - 5.96353066 \times 10^{-1}x^2)^2 + \\ &+ (5.619344031 \times 10^{-2} - 1.41864293 \times 10^{-1}x + 5.61934403 \times 10^{-2}x^2)^2 \end{aligned}$$

**Example 3.2.** If we instead take the univariate polynomial $f(x) = x^4 + x^3 + x^2 + x - 1$, the algorithm outputs that this polynomial is not a sum of squares, which is evident since it is not non-negative.

**Example 3.3.** Let us now consider the multivariate polynomial $f(x_1, x_2) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$. When we input this polynomial into our algorithm, it outputs the decomposition:

$$\begin{aligned} f(x) = &(-0.77189296x_1^2 + 2.18272409x_2^2 - 0.18539422x_1x_2)^2 + \\ &+ (-1.11250237x_1^2 - 0.47509339x_2^2 - 0.9615459x_1x_2)^2 + \\ &+ (-0.40806831x_1^2 - 0.1000096x_2^2 + 0.52154646x_1x_2)^2 \end{aligned}$$

**Example 3.4.** If we instead take the multivariate polynomial $f(x_1, x_2) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4 - 1$, the algorithm outputs that this polynomial is not a sum of squares, which again is evident since it is not non-negative.

## 3.3 Quadratic Modules

This algorithm takes as input a set of rational multivariate polynomials $f, g_1, \ldots, g_s \in \mathbb{Q}[\bar{X}]$ and a degree bound $2d$. It checks whether $f$ can be expressed as

$$f = \sigma_0 + \sigma_1 g_1 + \ldots + \sigma_s g_s,$$

where each $\sigma_i \in \sum \mathbb{Q}[\bar{X}]^2$ has a degree less than or equal to $2d$. In other words, the algorithm determines if the polynomial $f$ lies in the quadratic module generated by $g_1, \ldots, g_s$ with the specified degree bound. If the answer is affirmative, the algorithm provides a decomposition in the quadratic module, i.e., it outputs the polynomials $\sigma_0, \ldots, \sigma_s$. If not, it displays a message indicating that such a decomposition does not exist.

The only material we have found regarding this algorithm is a pseudocode for computing an exact quadratic module representation, described in [22], but implementations in the literature—whether for exact decompositions or approximations—are rare. As before, we developed our own implementation. In this case, the logic is similar to that of the SOS algorithm but extends it to handle quadratic modules.

This algorithm is partially implemented in C++ and partially in Python, with its code divided between the files **QuadraticModule.cpp** and **QuadraticModule.py**. These two programming languages have been used because C++ enables more efficient parallelization, while Python allows for a more straightforward and simpler implementation, in addition to offering a wide range of available libraries.

Since we don't know which combination of degrees for $\sigma_0, \ldots, \sigma_s$ will yield the best approximation, the first step is to compute all the possible degree combinations for these polynomials. These degree combinations are, in fact, translated into the vectors $v_{d_i}(\bar{X})$, which are required to represent each $\sigma_i = v_{d_i}(\bar{X})' Q_i v_{d_i}(\bar{X})$ for all $i = 0, \ldots, s$. This is why we must consider all combinations of degrees from 0 to $d$ for each $\sigma_i$, resulting in a total cardinality of $(d+1)^{s+1}$. However, in some cases, it is not necessary to test all these combinations. The iteration space can be reduced by employing simple optimization. Let $k$ be the degree of $f$, and let $k_0, k_1, \ldots, k_s$ denote the degrees of $\sigma_0, \sigma_1 g_1, \ldots, \sigma_s g_s$, respectively. If $k_i > k$ for some $i$ and $k_i > k_j$ for all $j \neq i$, then this combination is not valid. This is because if $\sigma_i g_i$ generates a monomial with a degree greater than that of $f$ in the decomposition, such a monomial would need to be canceled out through the contributions of the other polynomials $\sigma_j g_j$ in the decomposition.

Note that increasing the bound on the degree of the polynomials in the decomposition within the quadratic module not only increases the number of degree combinations to check (iterations space), but also enlarges the vector $v_d$, thereby leading to more complex problems for the solver to handle.

Once the reduced iteration space is known, we attempt to express $f$ within the quadratic module generated by $g_1, \ldots, g_s$ for each of the degree combinations previously computed. This is where parallelization plays a key role. Parallelization allows us to execute multiple degree combinations for $\sigma_0, \ldots, \sigma_s$ simultaneously. If an accurate enough decomposition is found, the algorithm stops and outputs $\sigma_0, \ldots, \sigma_s$. If, instead, all combinations have been tested without finding an accurate decomposition, the algorithm outputs a message indicating that $f$ does not lie in $M(g)$, at least for the given range of degrees for the $\sigma$'s. The details of the parallelization strategy are explained in the next chapter.

The logic explained above is implemented in the C++ file, while the Python file contains the logic for verifying whether $f$ can be expressed in terms of $g_1, \ldots, g_s$ and some $\sigma_0, \ldots, \sigma_s$.

Let us now move on to this Python file. This file takes as input the polynomials $f, g_1 \ldots, g_s$, and a degree combination for $\sigma_0, \ldots, \sigma_s$ (i.e., the vector $v_{d_i}(\bar{X}), i = 0, \ldots, s$), and outputs a number indicating whether $f$ is in $M(g)$ for this combination. If such a decomposition exists and meets the required precision, the function returns 1. If a decomposition exists (the solver finds it) but does not meet the required precision, it returns 0. If no such decomposition exists (the solver cannot find one), it returns -1. Finally, if there is some internal error within the solver, the function returns -2.

The first step, as in the SOS algorithm, is to create the variable matrices $Q_i$, imposing them to be PSD as well as symmetric. Next, it computes $v_{d_i}(\bar{X})v_{d_i}(\bar{X})'$, and for each product, finds the decomposition into the matrices $B_{\alpha_i}$. If it is the first iteration ($\sigma_0$), it calculates the general expression of $v_{d_i}(\bar{X})'Q_i v_{d_i}(\bar{X})$ by computing $\langle Q_i B_{\alpha_i} \rangle$ for each $B_{\alpha_i}$ in the decomposition, and saves the resulting coefficient-monomial pairs. If it is not the first iteration ($\sigma_1, \ldots, \sigma_s$), it computes the general expression of $v_{d_i}(\bar{X})'Q_i v_{d_i}(\bar{X})g_i$ by similarly computing $\langle Q_i B_{\alpha_i} \rangle$ for each $B_{\alpha_i}$ in the decomposition, and then multiplying by $g_i$.

At the end of this process, we obtain a list where each element represents a polynomial $\sigma_i g_i, i = 0, \ldots, s$, with $g_0 = 1$. Each of these polynomials is represented as a list of tuples, where each tuple corresponds to a coefficient-monomial pair. The key idea is that we now have the expressions of these polynomials as functions of the variables in the matrices $Q_i$.

Since there may be monomials that appear in more than one of the polynomials $\sigma_i g_i$, the algorithm first loops over each of these polynomials and saves all the distinct monomials. After obtaining the set of distinct monomials from the decomposition, we loop over them and examine the monomials in the polynomial $f$. If a monomial from the decomposition appears in $f$, we save the corresponding coefficient from $f$. Otherwise, we save 0. This step is crucial to determine the value to which the coefficient of a monomial in the decom-

position should be set. Finally, the algorithm loops over all the distinct monomials in the decomposition and sums the coefficients that correspond to the monomial from the current iteration. This process groups the coefficients of the monomials that appear in more than one polynomial $\sigma_i g_i$. Once we have the full reduced decomposition, grouped by coefficient-monomial pairs, we can compare it to the list we previously set up with the coefficients of $f$, and impose the necessary constraints.

The algorithm then calls the solver with all the constraints. If a solution $Q_0, \ldots, Q_s$ is returned, the decompositions $Q_i = H_i H_i'$ are computed, and the verification method is called to check whether the solution is accurate enough. If no solution is found, a message is displayed.

**Example 3.5.** Let us consider the univariate polynomial $f(x) = 2x^6 + 2x^5 + x^4 + x^2$, and the quadratic module generated by the univariate polynomials $g_1(x) = x^4$ and $g_2(x) = x^2$. When we input these polynomials into our algorithm along with a degree bound of 2, it outputs a possible decomposition within the quadratic module:

$$f(x) = (0.70710679x)^2 + [(-0.70722971 - 1.41418626x)^2 + \\ + (-0.01757752 + 0.00879046x)^2]g_1(x) + [(0.70710679)^2 + (0.70676529x)^2]g_2(x)$$

**Example 3.6.** If we instead take the univariate polynomial $f(x) = 2x^6 + 2x^5 + x^4 + x^2 - 1$, and the quadratic module generated by the univariate polynomials $g_1(x) = x^4$ and $g_2(x) = x^2$, the algorithm outputs that a decomposition of $f$ within the quadratic module does not exist. This can be seen by noting that, if such a decomposition did exist, $f$ would need to be non-negative on $S(g)$, but $0 \in S(g)$ and $f(0) < 0$.

**Example 3.7.** Let us consider now the quadratic module generated by the multivariate polynomials $g_1(x_1, x_2, x_3) = x_1 x_3^2 + 1 - x_1^2 - x_2^2$ and $g_2(x_1, x_2, x_3) = -x_1 x_3^2 + 1$. Clearly, the multivariate polynomial $f(x_1, x_2, x_3) = 2 - x_1^2 - x_2^2$ can be represented within this quadratic module. When we input these polynomials into our algorithm along with a degree bound of 2, it outputs a possible decomposition within the quadratic module (the trivial):

$$f(x_1, x_2, x_3) = (-0.00032352)^2 + g_1(x_1, x_2, x_3) + [(1.00000002)^2]g_2(x_1, x_2, x_3)$$

**Example 3.8.** Take now the multivariate polynomial $f(x_1, x_2, x_3) = 5 - x_1^2 - x_2^2 - x_3^2$, and the non-compact (and thus non-archimedean) quadratic module generated by $g_1(x_1, x_2, x_3) = x_1$, $g_2(x_1, x_2, x_3) = x_2$, and $g_3(x_1, x_2, x_3) = x_3$. When we input these polynomials into our algorithm along with a degree bound of 6, it outputs that a decomposition of $f$ within the quadratic module does not exist, as expected. If a decomposition did exist, then $M(g)$ would be archimedean, which is not true.

## 3.4 Archimedeanity

For an arbitrary number of variables and polynomials generating the quadratic module, there are no implementable algorithms to verify archimedeanity; only theoretical characterizations exist. In 2005, *M. E. Canto Cabral* presented an algorithm to verify this property in the case $n = 2$ (polynomials in two variables) in her doctoral thesis, *Archimedean Quadratic Modules: A Decision Procedure in Dimension Two* [25], although it appears that it has not been implemented and, furthermore, the implementation is not straightforward.

Here, we present a result (and the implementation) for the general case that converts the problem of finding $N$ in Definition 1.22 into the problem of determining the radius $R$ (if it exists) of a ball that contains the semialgebraic set

$$S(g) := \{\bar{x} \in \mathbb{R}^n | g_1(\bar{x}) \geq 0, \ldots, g_s(\bar{x}) \geq 0\}.$$

Checking if a quadratic module is archimedean becomes a problem of finding a lower bound for $N$. It is clear that if a particular value of $N$ works, then any $M \in \mathbb{R}$ such that $M > N$ will also work, because we can express

$$M - \sum_{i=1}^{n} X_i^2 = N - \sum_{i=1}^{n} X_i^2 + (\sqrt{M-N})^2,$$

which clearly belongs to $M(g)$.

Assume that $S(g)$ is a compact set (otherwise, $M(g)$ would not be archimedean). Therefore, there exists $R > 0$ such that $S(g)$ is contained in the ball centered at $\mathbf{0} \in \mathbb{R}^{\mathbf{n}}$ with radius $R$. Define $g_R(\bar{x}) := R^2 - (x_1^2 + \ldots + x_n^2)$.

**Proposition 3.9.** *$M(g)$ is archimedean if and only if $g_R \in M(g)$.*

*Proof.* If $M(g)$ is archimedean, then $g_R \in M(g)$ since $g_R > 0$ on $S(g)$, and hence Putinar's Positivstellensatz applies to it.

To prove the converse, suppose $g_R \in M(g)$. Then, for any $M \in \mathbb{R}$ such that $M \geq R^2$, we have $M - (x_1^2 + \ldots + x_n^2) \in M(g)$. Consequently, $M(g)$ is archimedean by definition. □

This result gives an effective and sharp bound on the number $R$ to test whether a certain quadratic module $M(g)$ is archimedean or not if $S$ is a compact set.

The problem now turns into finding the radius (if it exists) of a ball containing $S$. This task can be accomplished using a quantifier elimination algorithm, for which software tools are available, such as Reduce-Redlog package [26]. The property that a semialgebraic set is

compact can be expressed in the first-order language of real closed fields. Both being closed and being bounded can be written using formulas with no free variables (all variables are quantified), and the quantifier elimination algorithm will output a statement that is either trivially true or trivially false. Moreover, if the boundedness condition involving the norm bound $R$ is not quantified, then after quantifier elimination, one obtains a semialgebraic set describing the half-line $[R, +\infty)$ of all possible bounds. In this case, the formula would look like:

$$\forall \bar{x}(\bar{x} \in S(g) \implies ||\bar{x}|| \leq R),$$

where there are $n + 1$ variables $(\bar{x}, R)$, and only $\bar{x}$ are quantified. Writing an equivalent formula in terms of the free variables (unquantified) results in a semialgebraic set in the real line, which corresponds to the half-line of upper bounds.

Once $R$ is determined, we only need to use our quadratic module algorithm to check whether the polynomial $R^2 - \sum_{i=1}^{n} X_i^2$ lies in $M(g)$.

That said, given the sequence $g_1, \ldots, g_s$, we can always add $g_{s+1} := R^2 - \sum_{i=1}^{n} X_i^2$ to the list to define $S(g)$ with an archimedean quadratic module. So, the fundamental question regarding archimedeanity relates to determining the compactness of semialgebraic sets. *Is there a way to certify if a given semialgebraic set is compact without using the quantifier elimination algorithm?* Moreover, can compactness be inferred directly from the coefficients or degrees of the polynomials generating the quadratic module?

The specific question we address by developing this algorithm is: *given a compact semialgebraic set, do we truly need to extend the set by adding an extra polynomial to ensure archimedeanity, or is the information ensuring archimedeanity already inherent within the set itself?* Our focus is always on finding the simplest way to define an archimedean quadratic module without introducing redundant information. It is evident that a more complex quadratic module increases the computational complexity when determining decompositions within it.

Then, this algorithm takes as input the polynomials $g_1, \ldots, g_s$ that generate the quadratic module and a degree bound $2d$ for the decomposition within it. The polynomial $f$ for the quadratic module algorithm is then defined as $f = N - \sum_{i=1}^{n} X_i^2$ once $N$ is determined. If the archimedean property is verified within the degree range imposed, then $\sigma_0, \ldots, \sigma_s$ are returned, confirming that the quadratic module is archimedean and that a decomposition exists within the specified degree range. On the other hand, if the property is not verified within this range for the $\sigma$'s, we can only conclude that no such decomposition exists within this range, and the archimedean property is not satisfied there. Beyond this range, the outcome is uncertain: it might be that a decomposition exists for higher degrees, or it might not. To address this uncertainty, one approach would be to set the degree bound as outlined in 1.28, ensuring that if no decomposition is found within this bound, then the quadratic

module is not archimedean. However, this theoretical bound is impractically large, making implementation infeasible.

**Example 3.10.** Let us consider the quadratic module generated by the multivariate polynomials $g_1(x, y) = x^3 - y^2$, and $g_2(x, y) = 1 - x$, the associated semialgebraic set $S(g) = \{(x, y) \in \mathbb{R}^2 | x^3 - y^2 \geq 0, 1 - x \geq 0\}$, and a degree bound of 10. Computing the radius with Reduce-Redlog we find that $R = \sqrt{2}$, confirming that $S(g)$ is compact. Let us take $R = 2$.

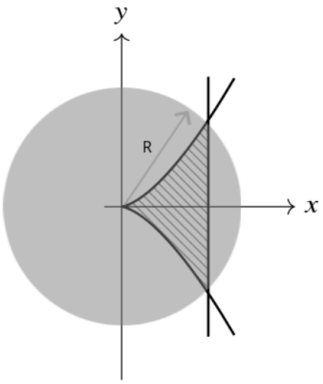



Figure 3.1: Ball containing the compact semialgebraic set $S(g)$.

Next, we need to determine whether $2 - X^2 - Y^2 \in M(g)$ for some polynomials $\sigma_0, \sigma_1, \sigma_2 \in \sum \mathbb{Q}[X, Y]^2$ of degree less than or equal to 10, using the quadratic module algorithm. The algorithm returns that a possible solution includes the polynomials:

$$\begin{aligned}
&\sigma_0(x,y) = (-6.08039805 \times 10^{-1} - 8.68254866 \times 10^{-17} y + 6.08038761 \times 10^{-1} x)^2 + \\
&+ (1.34755872 \times 10^{-15} - 7.50979301 \times 10^{-4} y + 1.34745380 \times 10^{-15} x)^2 + \\
&+ (3.76491031 \times 10^{-4} + 1.35104192 \times 10^{-15} y + 3.76491677 \times 10^{-4} x)^2, \\
&\sigma_1(x,y) = (-9.99999716 \times 10^{-1} + 9.30067140 \times 10^{-34} y - 5.07394190 \times 10^{-10} x)^2 + \\
&+ (-2.59459204 \times 10^{-19} + 1.20657274 \times 10^{-52} y + 5.11356132 \times 10^{-10} x)^2 + \\
&+ (1.53619622 \times 10^{-53} + 1.65170418 \times 10^{-20} y + 3.89728629 \times 10^{-63} x)^2, \\
&\sigma_2(x,y) = (-1.26380019 + 2.59763929 \times 10^{-17} y - 9.71597782 \times 10^{-1} x)^2 + \\
&+ (1.81924498 \times 10^{-1} - 1.35384393 \times 10^{-16} y - 2.36637238 \times 10^{-1} x)^2 + \\
&+ (-1.05834740 \times 10^{-20} - 4.75101281 \times 10^{-5} y + 1.23843747 \times 10^{-20} x)^2.
\end{aligned}$$

Now, we will examine a couple of additional examples of executions of our algorithm. The first example involves an archimedean quadratic module, while the second example

considers a non-archimedean quadratic module generated by a set of polynomials with a compact semialgebraic set associated. These correspond, respectively, to Examples 5.3.9 and 6.3.1 from [27], where the proofs of archimedeanity can be found. We will now verify that our algorithm produces the expected results.

**Example 3.11.** Consider the archimedean quadratic module generated by $g_1(x_1, x_2) = x_1$, $g_2(x_1, x_2) = x_2$, and $g_3(x_1, x_2) = 1 - x_1 - x_2$, along with the polynomial $f(x_1, x_2) = 1 - x_1^2 - x_2^2$, for which a decomposition exists. The radius $R = 1$ has been determined, again, using Reduce-Redlog. When these polynomials are provided to the algorithm with a degree bound of 12, it outputs the following polynomials, which provide a valid decomposition of $f$ within $M(g)$:

$$
\begin{aligned}
&\sigma_0(x_1, x_2) = (-0.00069383)^2,\\
&\sigma_1(x_1, x_2) = (-5.73953809 \times 10^{-1} - 8.34427487 \times 10^{-1} x_2 + 5.73936988 \times 10^{-1} x_1)^2 +\\
&+ (3.41056602 \times 10^{-1} - 4.69179305 \times 10^{-1} x_2 - 3.41057254 \times 10^{-1} x_1)^2 +\\
&+ (3.46955174 \times 10^{-4} - 3.82803085 \times 10^{-9} x_2 + 3.46959777 \times 10^{-4} x_1)^2,\\
&\sigma_2(x_1, x_2) = (-5.73953809 \times 10^{-1} + 5.73936988 \times 10^{-1} x_2 - 8.34427487 \times 10^{-1} x_1)^2 +\\
&+ (3.41056602 \times 10^{-1} - 3.41057254 \times 10^{-1} x_2 - 4.69179305 \times 10^{-1} x_1)^2 +\\
&+ (-3.46955174 \times 10^{-4} - 3.46959777 \times 10^{-4} x_2 + 3.82803085 \times 10^{-9} x_1)^2,\\
&\sigma_3(x_1, x_2) = (-9.83655974 \times 10^{-1} - 3.30746083 \times 10^{-1} x_2 - 3.30746083 \times 10^{-1} x_1)^2 +\\
&+ (3.73376214 \times 10^{-15} - 5.14430910 \times 10^{-1} x_2 + 5.14430910 \times 10^{-1} x_1)^2 +\\
&+ (-1.80059451 \times 10^{-1} + 2.67753065 \times 10^{-1} x_2 + 2.67753065 \times 10^{-1} x_1)^2.
\end{aligned}
$$

**Example 3.12.** Now, consider the multivariate polynomial $f(x_1, x_2) = 4 - x_1^2 - x_2^2$ and the non-archimedean quadratic module generated by the multivariate polynomials $g_1(x_1, x_2) = x_1 - \frac{1}{2}, g_2(x_1, x_2) = x_2 - \frac{1}{2}$, and $g_3(x_1, x_2) = 1 - x_1 x_2$. The set $S(g)$ is compact, and the radius $R = \sqrt{4.25}$ (though we have taken $R = 4$) is determined as in the previous examples. However, a decomposition of $f$ within this quadratic module does not exist.

Indeed, if we input these polynomials into the algorithm along with a degree bound, the algorithm confirms that $f$ cannot be expressed within $M(g)$.

To conclude this chapter, we would like to reflect on the following: a quadratic module could be archimedean while admitting different decompositions that "generate" this archimedeanity, i.e., different combinations of $\sigma$'s such that, for a fixed $N \in \mathbb{R}_{>0}$, $N - \sum_{i=1}^{n} X_i^2 = \sigma_0 + \sigma_1 g_1 + \ldots + \sigma_s g_s$. For this reason, as future work, we could study whether there is a relationship or pattern among the degrees of the different sets of $\sigma$'s that "generate" archimedeanity. In other words, we could investigate how the degrees of the $\sigma$'s that "generate" archimedean quadratic modules are distributed.

# Chapter 4

# Parallelization and Performance Evaluation

In this chapter, we will focus on understanding the motivation behind parallelization and delving into the tools and techniques used to implement it, describing their functionality and purpose in detail. Additionally, we will analyze the effectiveness of this parallelization strategy by evaluating its performance across various scenarios, providing insights into their strengths and limitations.

## 4.1 Parallelization

As explained earlier, our parallelization is implemented in the quadratic module algorithm, using different combinations of degrees for the $\sigma$'s in the representation within the quadratic module. For solvers supporting parallelization (like Mosek), a double layer of parallelization could be studied, combining a first layer with the degrees for the $\sigma$'s and a second layer implicitly managed by the solver, with benchmarking conducted to determine the optimal distribution of threads across both layers.

We have used OpenMP [28], a parallel programming model that allows the parallelization of code regions through directives also known as pragmas, and compiled the code using Clang/LLVM. OpenMP supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran. It is a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer. It provides a set of compiler directives, library routines, and environment variables that enable parallel programming. It is widely used for parallelizing loops, sections of code, and other regions in a program, making it easier to exploit multi-core processors and shared memory architectures.

One of the key features of OpenMP is its ease of use: it allows programmers to parallelize their code with minimal changes to the existing serial code by simply adding pragmas or directives. These directives are used to specify parallel regions, work-sharing constructs (such as loops), synchronization mechanisms (like critical sections, barriers, and atomic operations), and tasking constructs that enable fine-grained parallelism.

OpenMP is not limited to a specific hardware architecture, and it works on shared-memory systems, including multi-core and many-core processors, as well as on large-scale distributed systems, when used in combination with other tools like MPI (Message Passing Interface). This makes it highly versatile and effective for a wide range of computational problems, from simple multi-threaded tasks to large-scale simulations.

In summary, OpenMP is a powerful tool for parallel programming, providing a simple interface to harness the power of multi-core processors and shared-memory systems. Its ease of use, flexibility, and broad support across platforms make it an excellent choice for developing high-performance applications in scientific computing, engineering, and other fields requiring parallel processing.

In our implementation, we have used the directive #**pragma omp parallel for** to create a team of threads and run in parallel the loop that iterates over the combinations of degrees for the polynomials $\sigma_0, \ldots, \sigma_s$. We have also used the directives #**pragma omp cancel for** and #**pragma omp cancellation point for**, which allow us to stop the execution when an accurate enough solution is found. The first directive cancels the parallel loop, while the second allows us to define an explicit cancellation point within the loop, where it is verified if a cancellation of the loop has been requested. In other words, the first directive lets us cancel the loop, and the second one specifies where the loop's cancellation condition is checked by the threads.

A node of the Marenostrum V supercomputer includes 2x Intel Sapphire Rapids 8480+ at 2Ghz and 56 cores in each socket (for a total of 112 cores per node). Each core has a 2MB L2 and the whole node shares a 105MB L3. Keeping this in mind, we have used the **omp_num_threads** clause in the loop, so we can determine how many threads are created and used within it.

Also, when compiling, we have added a couple of environment variables that control specific aspects of the OpenMP runtime behavior. First, we set to true the variable **OMP_CANCELLATION** to enable thread cancellation, as explained earlier (by default, it is disabled to avoid a negative impact on performance when cancellations are not required). Secondly, we use the variable **OMP_PROC_BIND**, which is also set to true. This ensures that threads are fixed to specific processors and do not migrate during execution, potentially improving performance by reducing cache misses and enabling better memory locality.

## 4.2 Performance Evaluation

Next, we present the results of the benchmarking process, comparing the execution times of the sequential and the parallelized versions using different numbers of threads, across several examples seen in the previous chapter.

1. Consider, first, the archimedean quadratic module generated by $g_1(x_1, x_2) = x_1$, $g_2(x_1, x_2) = x_2$, and $g_3(x_1, x_2) = 1 - x_1 - x_2$, along with the polynomial $f(x_1, x_2) = 1 - x_1^2 - x_2^2$, for which a decomposition exists. We will now examine how long it takes to find a possible decomposition of $f$ within the quadratic module for several threads and degree bounds.

| Threads/Bound | deg 2 | deg 3 | deg 4 | deg 5 | deg 6 |
|---|---|---|---|---|---|
| **1 thread** | 5.757 sec | 7.234 sec | 8.247 sec | 9.195 sec | 10.353 sec |
| **25 threads** | 8.792 sec | 9.098 sec | 10.956 sec | 20.462 sec | 26.325 sec |
| **50 threads** | 5.673 sec | 10.080 sec | 11.815 sec | 22.355 sec | 29.608 sec |
| **75 threads** | 4.642 sec | 10.588 sec | 18.211 sec | 24.155 sec | 33.607 sec |
| **100 threads** | 5.511 sec | 11.594 sec | 15.916 sec | 23.840 sec | 37.289 sec |
| **112 threads** | 4.678 sec | 9.222 sec | 18.161 sec | 28.864 sec | 34.412 sec |

Table 4.1: Execution times first example.

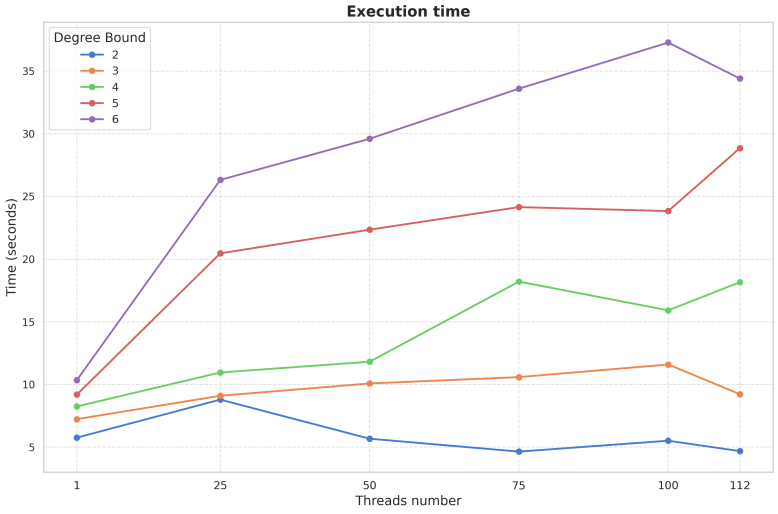



Figure 4.1: Plot execution times first example.

As we can observe, in this case, the sequential version appears to be the fastest. If we fix a degree bound and increase the number of threads, the execution time becomes slower. What happens here is that when using more than one thread, if one finds a valid solution, it must wait for all the other threads to finish their ongoing executions. In the sequential case, this does not happen, so the execution time is faster (especially if a solution is found in the early iterations of the sequential process, which is what is happening here). When we increase the number of threads while keeping the degree bound fixed, more executions need to finish once a valid solution has been found, which is why the execution time tends to increase. However, it is possible that, even with a fixed degree bound and an increasing number of threads, the time does not increase significantly or might even be slightly lower if the problems assigned to the remaining threads are resolved quickly (as is the case with a degree bound of value 2). This happens especially when working with small degree bounds.

On the other hand, if we fix the number of threads and increase the degree bound, the execution time will increase because higher-degree combinations are involved, and the problems sent to the solver from each thread are likely to be more computationally expensive. It could also happen that the first valid degree combination solution is not at the beginning of the combination vector. In such a case, the sequential version might take so long to find a solution that, even with the waiting time for all threads to finish in the multithreading cases, using multithreading would be faster.

2. Now, if we take the polynomial $f(x_1, x_2) = 4 - x_1^2 - x_2^2$ with the non-archimedean quadratic module generated by $g_1(x_1, x_2) = x_1 - \frac{1}{2}, g_2(x_1, x_2) = x_2 - \frac{1}{2}$, and $g_3(x_1, x_2) = 1 - x_1 x_2$, a decomposition of $f$ in this quadratic module does not exist. Therefore, we will examine how long it takes, using several threads, to check all possible combinations while varying the bounds for the degree of the polynomials in the decomposition.

| **Threads/Bound** | **deg 2** | **deg 3** | **deg 4** | **deg 5** | **deg 6** |
|---|---|---|---|---|---|
| **1 thread** | 14.188 sec | 51.681 sec | 215.500 sec | 594.932 sec | 1456.710 sec |
| **25 threads** | 2.586 sec | 7.417 sec | 28.509 sec | 53.847 sec | 154.680 sec |
| **50 threads** | 3.734 sec | 8.539 sec | 20.601 sec | 40.893 sec | 93.169 sec |
| **75 threads** | 2.555 sec | 8.091 sec | 18.091 sec | 34.749 sec | 69.054 sec |
| **100 threads** | 2.542 sec | 8.042 sec | 13.657 sec | 34.650 sec | 66.762 sec |
| **112 threads** | 3.610 sec | 7.930 sec | 14.893 sec | 25.989 sec | 60.081 sec |

Table 4.2: Execution times second example.

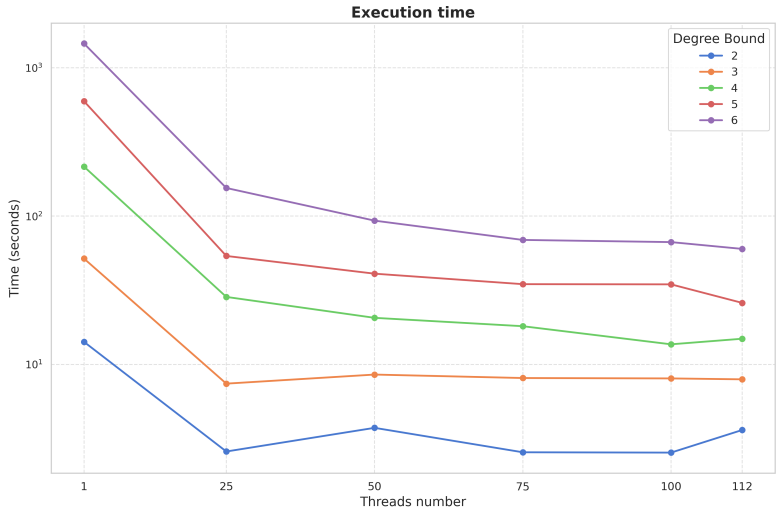



Figure 4.2: Plot execution times second example (logarithmic scale).

In this case, since the desired decomposition does not exist for any combination, all possible combinations up to the degree bound are tested, meaning the entire combination vector is traversed. Therefore, as can be observed, it is much more efficient to use the maximum number of threads possible to solve all the cases more quickly in parallel.

3. Let us consider the quadratic module generated by $g_1(x_1, x_2, x_3) = x_1x_3^2+1-x_1^2-x_2^2$ and $g_2(x_1, x_2, x_3) = -x_1x_3^2+1$. Clearly, the polynomial $f(x_1, x_2, x_3) = 2-x_1^2-x_2^2$ can be represented within this quadratic module. Let us examine how long it takes for the algorithm to find a possible decomposition of $f$ within it.

| **Threads/Bound** | **deg 2** | **deg 3** | **deg 4** | **deg 5** | **deg 6** |
|---|---|---|---|---|---|
| **1 thread** | 2.472 sec | 2.817 sec | 2.727 sec | 4.724 sec | 2.230 sec |
| **25 threads** | 3.299 sec | 11.094 sec | 28.813 sec | 102.136 sec | 210.550 sec |
| **50 threads** | 2.430 sec | 9.822 sec | 28.100 sec | 102.504 sec | 212.361 sec |
| **75 threads** | 2.446 sec | 10.925 sec | 28.876 sec | 101.665 sec | 211.171 sec |
| **100 threads** | 1.622 sec | 10.007 sec | 28.735 sec | 100.027 sec | 212.246 sec |
| **112 threads** | 2.309 sec | 8.737 sec | 32.399 sec | 101.717 sec | 211.413 sec |

Table 4.3: Execution times third example.

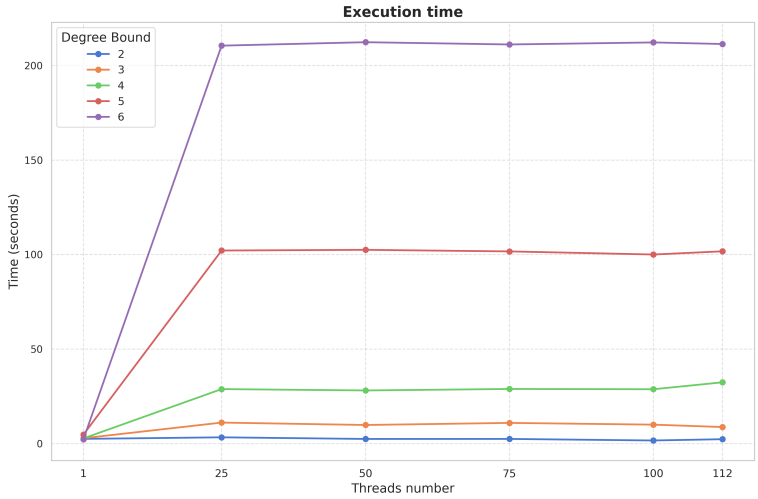



Figure 4.3: Plot execution times third example.

In this case, the situation is similar to what was explained in the first example. In fact, here the execution times increase less when fixing a degree bound and increasing the number of threads because, even though more threads are made available to the program, the iteration space after optimization is smaller than the number of threads provided. Therefore, fewer threads than those allocated are being used.

4. Finally, we will take the polynomial $f(x_1, x_2, x_3) = 5 - x_1^2 - x_2^2 - x_3^2$, and the non-compact (and thus non-archimedean) quadratic module generated by $g_1(x_1, x_2, x_3) = x_1$, $g_2(x_1, x_2, x_3) = x_2$, and $g_3(x_1, x_2, x_3) = x_3$. This setup allows us to observe how much time the algorithm requires to check all possible combinations up to a certain degree and verify that such a decomposition does not exist.

| **Threads/Bound** | **deg 2** | **deg 3** | **deg 4** |
|---|---|---|---|
| **1 thread** | 27.907 sec | 83.177 sec | 730.149 sec |
| **25 threads** | 6.627 sec | 9.833 sec | 166.875 sec |
| **50 threads** | 2.154 sec | 14.236 sec | 93.066 sec |
| **75 threads** | 2.654 sec | 16.265 sec | 64.641 sec |
| **100 threads** | 5.408 sec | 15.326 sec | 59.115 sec |
| **112 threads** | 5.935 sec | 15.026 sec | 61.475 sec |

Table 4.4: Execution times fourth example.

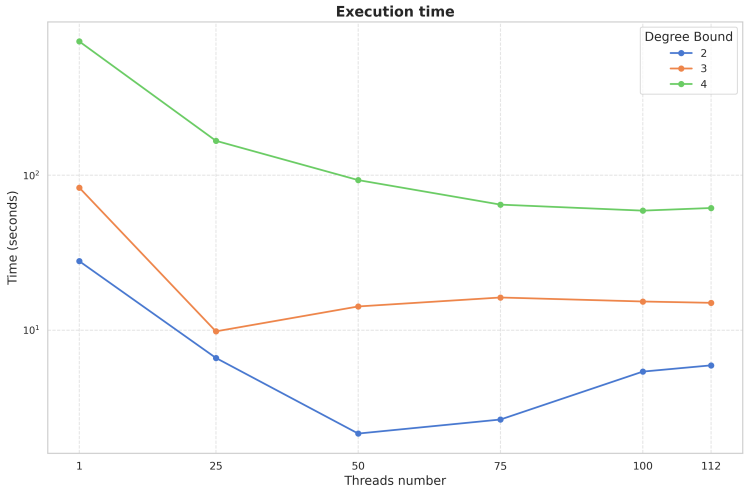



Figure 4.4: Plot execution times fourth example (logarithmic scale).

Finally, this is a case similar to the second example. Since the desired decomposition does not exist, all possible degree combinations in the vector are tested, making it more efficient to parallelize as much as possible.

Thus, after studying several examples, we can conclude that using the maximum number of threads possible (in our case, 112) is the best option.

When we cannot express a polynomial in the quadratic module generated by other polynomials for a given degree bound, and all combinations must be tested, it is always much better to fully exploit multithreading. In cases where a decomposition does exist, if it corresponds to a degree combination found early in the iteration space, fully parallelizing is slightly slower. However, the time lost in this scenario is minimal compared to the significant gains achieved by maximizing parallelization in cases where the decomposition does not exist. Conversely, if the combination that provides the decomposition is found later in the combination vector, then parallelization remains the more optimal choice.

Therefore, if we do not know in advance whether a given polynomial can be represented in the quadratic module generated by other polynomials for a certain degree bound, it seems that the best approach is to configure the program to use all available threads.

# Chapter 5

# Conclusions

It is worth noting that the area of study of these topics is an active field of research and truly fascinating. With this in mind, several conclusions can be drawn from the work conducted.

First, it would be worthwhile to delve deeper into the work presented in Chapter 2 to attempt the implementation of the main theorem by performing the necessary verifications of the assumptions. Although some verifications may not be trivial to implement, it could prove to be a valuable exercise.

Second, concerning the quadratic module algorithm and the bound on the representation for archimedean quadratic modules, it should be noted that the current theoretical bound for Putinar's Positivstellensatz, which would allow us to ensure non-archimedeanity by setting the bound at this value, is too high to be computable. Our algorithm can verify that a polynomial cannot be represented in the quadratic module up to a certain degree of the polynomials in the decomposition, but reaching the theoretically prescribed degree is currently not feasible. However, if a valid decomposition is found within the chosen degree range, we can indeed confirm that the quadratic module is archimedean.

Finally, regarding efforts to improve and optimize these algorithms, the semidefinite programming solver used internally by the quadratic module algorithm only functions up to a certain size of input vectors and matrices. As the number of variables or the degree of the polynomials in the decomposition increases, these vectors grow larger, causing the solver to become slow and incapable of solving the given problem. Thus, there is a limitation in this regard. This issue could be partially mitigated if a more powerful solver were available or if the code were executed on multiple GPUs simultaneously, leveraging greater computational capacity. It would be worthwhile to explore ways to reduce the iteration space in the combinations of degrees by deriving an even more optimal criterion. However, this improvement would be of limited utility if the issue with the solver is not addressed.

# Appendix A

# Additional Proofs

***Proof of Lemma 2:*** We want to see that, given a polynomial $f \in \mathbb{R}[\bar{X}, \bar{Y}]$, there is a constant $K > 0$ such that, for every $\xi_1, \xi_2 \in \Delta \times C$,

$$|\widetilde{f}(\xi_1) - \widetilde{f}(\xi_2)| \leq K||\xi_1 - \xi_2||.$$

Denoting by $D\widetilde{f}$ the derivative of $\widetilde{f}$, by the mean value theorem, it is enough to show that $|D\widetilde{f}(\xi)(e)| \leq K$ for all $\xi \in \Delta \times C$ and $e \in \mathbb{R}^{n+l+1}$ with $||e|| = 1$, where the expression of $K$ will be obtained later.

First of all, we are interested in bounding $\widetilde{f}$. We have

$$\widetilde{f}(\bar{X}, \bar{Y}, Z) = \sum_{|k| \leq m} f_k(\bar{X}) \bar{Y}^k Z^{m-|k|}$$

and, rewriting $f_k(\bar{X})$,

$$\widetilde{f}(\bar{X}, \bar{Y}, Z) = \sum_{|k| \leq m} (\sum_{\alpha} a_\alpha \binom{|\alpha|}{\alpha} \bar{X}^\alpha)_k \bar{Y}^k Z^{m-|k|}.$$

Then,

$$|\widetilde{f}(\bar{X}, \bar{Y}, Z)| = \left| \sum_{|k| \leq m} \left( \sum_{\alpha} a_\alpha \binom{|\alpha|}{\alpha} \bar{X}^\alpha \right)_k \bar{Y}^k Z^{m-|k|} \right| \leq$$

$$\leq \sum_{|k| \leq m} \left| \sum_{\alpha} a_\alpha \binom{|\alpha|}{\alpha} \bar{X}^\alpha \right|_k |\bar{Y}|^k |Z|^{m-|k|} \leq \sum_{|k| \leq m} ||f|| 2d(n\sqrt{N})^d |\bar{Y}|^k |Z|^{m-|k|},$$

since

$$\left|\sum_{\alpha} a_\alpha \binom{|\alpha|}{\alpha} \bar{X}^\alpha\right|_k \leq \left(\sum_{\alpha} |a_\alpha| \binom{|\alpha|}{\alpha} |X_1|^{\alpha_1} \cdots |X_n|^{\alpha_n}\right)_k \leq$$
$$\leq ||f_k|| \sum_{k=0}^{d_k} (n\sqrt{N})^k \leq ||f|| \sum_{k=0}^{d} (n\sqrt{N})^k \leq ||f||(d+1)(n\sqrt{N})^d \leq ||f||2d(n\sqrt{N})^d,$$

where $||f_k|| = \max_\alpha |a_\alpha|_k$, $||f|| = \max_k ||f_k||$, $d_k$ is the degree in $\bar{X}$ of $f_k$, $d$ is the maximum degree in $\bar{X}$ of all the $f_k$'s (i.e. $d = \max_k d_k$), and the multinomial identity and the fact that $\bar{X} \in \Delta$ have been used.

Moreover, since $C$ is compact, there exists $\rho \in \mathbb{R}$ which is the maximum value of $||(\bar{y}, z)||$ for $(\bar{y}, z) \in C$. Then,

$$|\widetilde{f}(\bar{X}, \bar{Y}, Z)| \leq \sum_{|k| \leq m} ||f||2d(n\sqrt{N})^d |Y_1|^{k_1} \dots |Y_l|^{k_l} |Z|^{m-|k|} \leq$$
$$\leq \sum_{|k| \leq m} ||f||2d(n\sqrt{N})^d \rho^{k_1} \dots \rho^{k_l} \rho^{m-|k|} = \sum_{|k| \leq m} ||f||2d(n\sqrt{N})^d \rho^m \leq$$
$$\leq \binom{l+m}{m} ||f||2d(n\sqrt{N})^d \rho^m,$$

where $m$ is the degree in the variables $Y_1, \dots, Y_l$ of $f$, and $\binom{l+m}{m}$ is the number of all the monomials $\bar{Y}^k$ of degree less than or equal to $m$.

Now that we have bounded $\widetilde{f}$, we are interested in bounding $D\widetilde{f}$. We know that

$$|\widetilde{f}(\bar{X}, \bar{Y}, Z)| \leq \sum_{|k| \leq m} ||f|| \sum_{i=0}^{d} (|X_1| + \dots + |X_n|)^i |\bar{Y}|^k |Z|^{m-|k|}.$$

Then

$$\left|\frac{\partial \widetilde{f}(\bar{X}, \bar{Y}, Z)}{\partial X_i}\right| \leq \sum_{|k| \leq m} ||f|| \sum_{i=1}^{d} i(|X_1| + \dots + |X_n|)^{i-1} |\bar{Y}|^k |Z|^{m-|k|} \leq$$
$$\leq \sum_{|k| \leq m} ||f|| d^2 (|X_1| + \dots + |X_n|)^{d-1} \rho^m \leq \binom{l+m}{m} ||f|| d^2 (n\sqrt{N})^{d-1} \rho^m,$$
$$\left|\frac{\partial \widetilde{f}(\bar{X}, \bar{Y}, Z)}{\partial Y_i}\right| \leq \sum_{|k| \leq m} ||f|| \sum_{i=0}^{d} (|X_1| + \dots + |X_n|)^i k_i |Y_i|^{k_i - 1} \prod_{j \neq i} |Y_j|^{k_j} |Z|^{m-|k|} \leq$$
$$\leq \sum_{|k| \leq m} ||f|| \sum_{i=0}^{d} (|X_1| + \dots + |X_n|)^i k_i \rho^{m-1} \leq \binom{l+m}{m} ||f||2d(n\sqrt{N})^d m \rho^{m-1},$$

and

$$\left|\frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial Z}\right| \leq \sum_{|k|\leq m} ||f|| \sum_{i=0}^{d} (|X_1|+\ldots+|X_n|)^i |\bar{Y}|^k (m-|k|)|Z|^{m-|k|-1} \leq$$
$$\leq \sum_{|k|\leq m} ||f|| 2d(n\sqrt{N})^d (m-|k|)\rho^{m-1} \leq \binom{l+m}{m} ||f|| 2d(n\sqrt{N})^d m\rho^{m-1}.$$

This implies that

$$|D\widetilde{f}(\bar{X},\bar{Y},Z)(e)| = \left|\sum_{i=1}^{n} \frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial X_i} e_i + \sum_{i=1}^{l} \frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial Y_i} e_{i+n} + \frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial Z} e_{n+l+1}\right| \leq$$
$$\leq \sum_{i=1}^{n} \left|\frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial X_i}\right| |e_i| + \sum_{i=1}^{l} \left|\frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial Y_i}\right| |e_{i+n}| + \left|\frac{\partial \widetilde{f}(\bar{X},\bar{Y},Z)}{\partial Z}\right| |e_{n+l+1}| \leq$$
$$\leq \binom{l+m}{m} ||f|| d^2 (n\sqrt{N})^{d-1} \rho^m \sqrt{n} + \binom{l+m}{m} ||f|| 2d(n\sqrt{N})^d m\rho^{m-1}\sqrt{l} +$$
$$+ \binom{l+m}{m} ||f|| 2d(n\sqrt{N})^d m\rho^{m-1} = \binom{l+m}{m} ||f|| d^2 (n\sqrt{N})^{d-1} \rho^m \sqrt{n} +$$
$$+ \binom{l+m}{m} ||f|| 2d(n\sqrt{N})^d m\rho^{m-1}(\sqrt{l}+1) =$$
$$= \binom{l+m}{m} ||f|| d(n\sqrt{N})^{d-1} \rho^{m-1}(d\rho\sqrt{n} + 2(n\sqrt{N})m(\sqrt{l}+1)),$$

since for a vector e on the unit sphere in $\mathbb{R}^{n+l+1}$, $\sum_{i=1}^{n} |e_i|$ and $\sum_{i=n+1}^{n+l} |e_i|$ can reach at most $\sqrt{n}$ and $\sqrt{l}$ respectively.

Then, we have shown that there exists a

$$K = \binom{l+m}{m} ||f|| d(n\sqrt{N})^{d-1} \rho^{m-1}(d\rho\sqrt{n} + 2(n\sqrt{N})m(\sqrt{l}+1))$$

computed in terms of $n$, $l$, the degrees in $\bar{X}$ and $\bar{Y}$ of $f$, the size of the coefficients of $f$, $N$, and $\rho$, such that the condition $|D\widetilde{f}(\xi)(e)| \leq K$ is satisfied, and thus the Lemma 2 is proved.

# References


[1] Lasserre, Jean Bernard. *Moments, positive polynomials and their applications.* Imperial College Press Optimization Series, 1. Imperial College Press, London, 2010.

[2] Powers, Victoria. *Certificates of positivity for real polynomials–theory, practice, and applications.* Developments in Mathematics, 69. Springer, Cham, 2021.

[3] Landau, Edmund. *Über die Darstellung definiter Funktionen durch Quadrate.* Math. Ann. 62 (1906), no. 2, 272–285.

[4] Pourchet, Y. *Sur la représentation en somme de carrés des polynômes à une indéterminée sur un corps de nombres algébriques.* Acta Arith. 19 (1971), 89–104.

[5] Victor Magron and Przemysław Koprowski and Tristan Vaccon. *Pourchet's theorem in action: decomposing univariate nonnegative polynomials as sums of five squares.* ISSAC: International Symposium on Symbolic and Algebraic Computation (2023).

[6] Blekherman, Grigoriy. *There are Significantly More Nonnegative Polynomials than Sums of Squares.* Israel J. Math. 153 (2006), 355–380.

[7] Lasserre, Jean Bernard. *A sum of squares approximation of non-negative polynomials.* SIAM Review, Vol. 49, ISS. 4 (2007).

[8] JB Lasserre, T Netzer. *SOS approximations of non-negative polynomials via simple high degree perturbations.* Springer. 256 (2007), 99–112.

[9] G. Jeronimo, D. Perrucci. *Rational certificates of non-negativity on semialgebraic subsets of cylinders.* Journal of Pure and Applied Algebra. 228, 6, (2024).

[10] Reznick, B. *Uniform denominators in Hilbert's seventeenth problem.* Math. Z. 220, 75–97 (1995).

[11] M. D. Choi, T. Y. Lam, and B. Reznick. *Sums of squares of real polynomials.* In Proc. Sympos. Pure Math., pages 103–126, 1995.

[12] V. Powers, T. Wörmann. *An algorithm for sums of squares of real polynomials.* Journal of Pure and Applied Algebra, 127:99–104, 1999.

[13] Marshall, M. *Positive Polynomials and Sums of Squares.* Mathematical Surveys and Monographs, vol. 146. American Mathematical Society, Providence (2008).

[14] Schmüdgen, K. *The K-moment problem for compact semi-algebraic sets.* Math. Ann. 289, 203–206 (1991).

[15] Putinar, M. *Positive polynomials on compact semi-algebraic sets.* Indiana Univ. Math. J. 43, 969–984 (1993).

[16] Schweighofer, M. *On the complexity of Schmüdgen's Positivstellensatz.* J. Complexity 20, 529–543 (2004).

[17] Nie, J., Schweighofer, M. *On the complexity of Putinar's Positivstellensatz.* J. Complexity 23, 135–150 (2007).

[18] Mihaela Curmei and Georgina Hall. *Shape-Constrained Regression using Sum of Squares Polynomials.* Operations Research 0 (0), (2023).

[19] Brendan O'Donoghue and Eric Chu and Neal Parikh and Stephen Boyd. *Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding.* Journal of Optimization Theory and Applications 169, 3, 1042-1068 (2016), http://stanford.edu/ boyd/papers/scs.html.

[20] MOSEK ApS. *Semidefinite optimization.* (2019), https://docs.mosek.com/modeling-cookbook/sdo.html.

[21] Steven Diamond, Stephen Boyd. *CVXPY: A Python-embedded modeling language for convex optimization.* Journal of Machine Learning Research 17, 83, 1–5 (2016).

[22] V. Magron and M. Safey El Din. *On Exact Polya and Putinar's Representations.* ISSAC: International Symposium on Symbolic and Algebraic Computation (2018).

[23] V. Magron and M. Safey El Din. *RealCertify: a Maple package for certifying non-negativity.* SIGSAM, ACM Communications in Computer Algebra, Vol. 52, No. 2, 34-37 (2018).

[24] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. *Sum of Squares Optimization Toolbox for MATLAB - User's Guide.* www.cds.caltech.edu/sostools/sostools.pdf.

[25] M. E. Canto Cabral. *Archimedean Quadratic Modules: A Decision Procedure in Dimension Two.* PhD thesis, Universität Konstanz, 2005.

[26] Andreas Dolzmann and Thomas Sturm. *REDLOG: computer algebra meets computer logic.* ACM SIGSAM Bulletin, 31, 2, 2 - 9 (1997).

[27] Alexander Prestel and Charles N. Delzell. *Positive Polynomials: From Hilbert's 17th Problem to Real Algebra.* Springer, 2001.

[28] OpenMP Architecture Review Board. *OpenMP Application Program Interface version 6.0.* (2024). https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf.