UNIVERSITY OF PADUA
Department of Mathematics
"Tullio Levi Civita"

Master Degree in Mathematics

Polynomial Programming with DIGS: a Second Order Cone implementation

Supervisor:
Professor Francesco Rinaldi

Candidate:
Davide Angioni
Student Number:
1180005

21 February 2020 - A. Y. 2019/2020

Contents

Chapter 1

Introduction

Polynomials are well known, powerful tools used for centuries in Mathematics. We do not only find them in theoretical Maths, but they are also used to model problems in Sciences, Economy and Finance, Machine Learning, Engineering, Computer Sciences and many other fields. Their strength is that they are intuitive and yet very useful to describe non-linear problems. Indeed, polynomials are the most famous non-linear functions that one can think of.

During the last 20 years the interest towards non-linear programming increased drastically. This is due to many factors, including the better understanding of linear programming, new mathematical tools and also the advancement of computers, now capable of solving big problems in a reasonable amount of time.
It is not surprising then, that during this period of time many new advances where achieved also in polynomial programming.

Indeed, nonetheless polynomials were studied for many centuries, only during last century we were able to solve some complicated problems related to them. For example how to describe non-negative polynomials using sum of squares, the famous Hilbert problem [Hil88]. Or, closely related to this one, how to describe non-negative polynomials in a semialgebraic set.
The latter is the problem we are more interested in, since it is strictly connected to solving polynomial programs. Shor [Sho87], in facts, showed that solving a polynomial problem can be traced back to characterizing non-negative polynomials over the set described by the constraints of the program, that is a semi-algebraic set.
Unfortunately this tunrned out to be $\mathcal{NP}$-hard, but thanks to the work of Parrilo [Par00], Parrilo and Strumfels [PS01], and Lasserre [Las01], and using algebraic results known as Positivstellensatz from Putinar [Put93], Stengle [Ste74] and others, we were able to relax it. These relaxations

are a sequence of semidefinite programs (a linear program with a further constraint of positive semidefiniteness for a matrix). In each step of this sequence, the degree of Sum of Squares (SOS) polynomials involved is increased, and increasing the degree improves the relaxation. This sequence is also proved to converge to the solution of the polynomial program taken in consideration. We call it a Lasserre Hierarchy, and we use the degree of polynomials involved to specify which step of the sequence we are considering.

This approach is also used nowadays but, unfortunately, needs a lot of computational power to be implemented, since the complexity of the problem for a fixed number of variables increases exponentially with the degree.
In order to solve this problem, many methods were implemented. For example exploiting the sparsity to reduce the number of computations needed [Fuk+01], or to relax the problem even more. The latter was done by Ahmadi and Majumdar [AM17a], and further developed also by Hall [AH15]. It was achieved using a further relaxation of sum of squares polynomials, namely Scaled Diagonal dominant Sum of Squares (SDSOS), that can be written using Scaled Diagonally Dominant (sdd) matrices. This approach, even if relaxes the problem even more (and thus give weaker bounds to the optimal solution of the original polynomial program) has the advantage of being very fast since the semidefiniteness contraint is applied only to 2 by 2 matrices, namely it uses Second Order Cone constraints. This allows to increase the size of problems taken in consideration.

It's in this setting that our work is collocated. Indeed, in a recent article of Ghaddar, Vera and Anjos [GVA16], a new algorithm for solving polynomial problems using SOS polynomials was implemented. This algorithm, called Dynamical Inequalities Generation Scheme (DIGS), instead of increasing the degree of polynomials involved, generates new inequalities to improve the description of the semialgebraic set in which the optimization takes place. Since the degree is fixed, the dimension of the problem does not increase at each step. The method was shown to give very good bounds and sometimes also to converge to the optimal value using less time and memory compared to Lasserre's approach.
In this thesis we focused on extending the algorithm to SDSOS polynomials, moved by the increasing attention that SOCP are getting.
After writing down the new formulation for the algorithm and implementing it in Matlab, we were able to test it on some examples.

To further increase its efficiency we also implemented a method to exploit the sparsity that this kind of problems usually have, following the approach of [SY19], checking that this method (thought for quadratic problems) could be used also for our problem.

Finally we explained a way to strengthen our relaxation using valid inequal-

ities.

Before presenting our work, we would like to clarify the notation used in this thesis:

- We use the symbol $\mathbb{N}$ for the set $\{0, 1, 2, ...\}$, that is the set of natural numbers including 0. We will write $\mathbb{N}_+$ whenever we want to exclude the 0.
- $\mathbb{R}$ is the set of real numbers and $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$ is the set of non-negative reals.
- Given a set A, we use the convention $A^n = A \times A \times ... \times A$ n-times and $x \in A^n$ is a vector with n entries in A.
- Given a vector $x \in \mathbb{R}^n$, we denote with $\mathbb{R}[x]$ the ring of polynomials in $(x_1, x_2, ..., x_n)$. We use $\mathbb{R}_d[x]$ for the ring of polynomials of degree at most d.
- Given two vectors $x, y \in \mathbb{R}^n$, we use (x, y) for the scalar product $(x, y) = x_1y_1 + x_2y_2 + ... + x_ny_n$.

Chapter 2

Polynomial Programming

First of all, we need to understand what a polynomial programming problem is and what is the state of the art method to optimize over it.
We start by defining what a multivariate polynomial is, and introducing some notation to work with them.
Most of the notation, definitions and theorems are taken from [GVA16], [Gha11] and references therein.

Given a vector $x = (x_1, ..., x_n) \in \mathbb{R}^n$ and a second vector $a = (a_1, ..., a_n) \in \mathbb{N}^n$, we will use the convention

$$x^a = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$$

We will also say that $|a| = \sum_{i=1}^{n} a_i$.

Definition 2.1 (Multivariate Polynomial)**.** Given a vector $x = (x_1, ..., x_n)$, we say that $p(x) \in \mathbb{R}[x]$ is a *multivariate polynomial* if

$$p(x) = \sum_{a \in \mathbb{N}^n} c_a x^a$$

where $a = (a_1, ..., a_n)$ and $c_a \in \mathbb{R},\ \forall a \in \mathbb{N}^n$. We will also say $p(x) \in \mathbb{R}_d[x]$ is a multivariate polynomial of degree at most d if

$$p(x) = \sum_{|a| \leq d} c_a x^a$$

Example 2.1. For example, if we take $x \in \mathbb{R}^2$ and

$$p(x) = x_1^2 + 3x_1x_2 - \frac{1}{2}x_2^2$$

in $\mathbb{R}_2[x]$, we have that

$$a = (2,0) \Rightarrow c_{(2,0)} = 1;$$
$$a = (1,1) \Rightarrow c_{(1,1)} = 3;$$
$$a = (0,2) \Rightarrow c_{(0,2)} = -\frac{1}{2};$$

Definition 2.2 (Polynomial Programming)**.** We call *polynomial programming* problem, any optimization problem of the form

$$\textbf{(PP-P)} \quad \rho = \inf_{x \in \mathbb{R}^n} f(x) \tag{2.1}$$
$$\text{s.t.} \quad g_i(x) \geq 0 \qquad i = 1, ..., m \tag{2.2}$$

where $f(x)$ and $g_i(x)$ are multivariate polynomials in n variables.

For example, we can regard linear programming as a special case of polynomial programming, where each polynomial has degree at most 1.
The main problem with Polynomial Programming is that the feasible region

$$S = \left\{ x \in \mathbb{R}^n \mid g_i(x) \geq 0 \quad \forall i \in \{1, ..., m\} \right\}$$

is not always convex and the objective function $f(x)$ is not linear and many time it is not even convex.
We call a set like S a *semialgebraic* set.

To address the problem of optimizing over S a non-linear function we need to relax the polynomial program in a way such that it can be cast as a *conic program*.
Conic programming is a subcategory of convex programming [BV04] and is a broadly studied field. We know how to optimize non-linear functions over a *cone* using, for example, Interior Point methods.

2.1 Conic Programming

A conic program is a program in which the feasible set is given as an intersection between a *cone* and a polytope.

Definition 2.3 (Cone)**.** Given a set $\mathcal{C} \subset \mathbb{R}^n$, we say that $\mathcal{C}$ is a *cone* if, taken $y \in \mathcal{C},\ \lambda \geq 0$, then $\lambda y \in \mathcal{C}$.
We say that $\mathcal{C}$ is a pointed cone if $\mathcal{C} \cap -\mathcal{C} = \{0\}$.

Example 2.2. We give here some examples of famous cones

- $\mathbb{R}^n$ is a cone for all $1 \leq n \in \mathbb{N}$

- $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n | x_i \geq 0 \ i = 1, ..., n\}$ is a cone for all $1 \leq n \in \mathbb{N}$
- The set $C = \left\{x \in \mathbb{R}^{n+1} \mid x_0 \geq \sqrt{\sum_{i=1}^n x_i^2}, \ x_0 \geq 0\right\}$ is the so-called Second Order cone.

We will see more examples of cones through the chapter.

Definition 2.4 (Dual)**.** Given a set $A \subset \mathbb{R}^n$, we say that

$$A^* = \{y \in \mathbb{R}^n \mid (x, y) \geq 0 \quad \forall x \in A\}$$

is the *dual* of A.

We can check that $(\mathbb{R}^n)^* = \{0\}$ and $(\mathbb{R}^n_+)^* = \mathbb{R}^n_+$.

Definition 2.5 (Convex Set)**.** A set A is said to be *convex* if

$$\lambda y_1 + (1 - \lambda)y_2 \in A \quad \forall y_1, y_2 \in A, \ \lambda \in [0, 1]$$

Intuitively a set is convex whenever, taken two arbitrary points in the set, the segment between those 2 point lies on the set.

It is then straightforward to define a convex cone and the dual cone. We say that a cone is self-dual if $\mathcal{C}^{**} = \mathcal{C}$.
Notice that both examples of cones given above are convex and self dual.

Lemma 2.3 (Dual of a cone is a convex cone)**.** Given a cone $\mathcal{C}$, then $\mathcal{C}^*$ is a convex cone.

Proof. First, let's prove that $\mathcal{C}^*$ is a cone. If $\mathcal{C}^*$ is non-empty, then taken $y \in \mathcal{C}^*$, we have that $(x, y) \geq 0$ for all $x \in \mathcal{C}$. Then, if we take λy with $\lambda \geq 0$, we have

$$(x, \lambda y) = \lambda(x, y) \geq 0$$

since it is a product of two positive quantities. Then $\lambda y \in \mathcal{C}^*$.
Now we prove that it is convex. Let $y_1, y_2 \in \mathcal{C}^*$ and $\lambda \in [0, 1]$. Then we have

$$(x, \lambda y_1 + (1 - \lambda)y_2) = \lambda(x, y_1) + (1 - \lambda)(x, y_2) \geq 0$$

since it is the sum of positive terms. Then $\lambda y_1 + (1 - \lambda)y_2 \in \mathcal{C}^*$ □

We say that a set is closed, if it is closed for the topology one is considering. In our case, the Euclidean topology.

Definition 2.6 (Conic Programming)**.** A program of the form

$$\begin{aligned} (\text{CP} - \text{P}) \qquad & \min c^T x \\ & \text{s.t. } Ax = b \\ & x \in \mathcal{C} \end{aligned} \tag{2.3}$$

is a *conic program*, whenever $\mathcal{C}$ is a pointed closed convex cone, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$

From this definition, we can derive the dual problem using a Lagrangian multiplier y. Let

$$L(x, y) = c^T x + y^T (b - Ax)$$

be the Lagrangian of the system. Then we get [BV04]:

$$\begin{aligned} \min_{x \in \mathcal{C}} \left\{ c^T x \mid Ax = b \right\} &= \min_{x \in \mathcal{C}} \max_{y \in \mathbb{R}^n} c^T x + y^T (b - Ax) \\ &\geq \max_{y \in \mathbb{R}^n} \min_{x \in \mathcal{C}} c^T x + y^T (b - Ax) \qquad (2.4) \\ &= \max_{y \in \mathbb{R}^n} \left(b^T y + \min_{x \in \mathcal{C}} (c - A^T y)^T x \right) \\ &= \max_{y \in \mathbb{R}^n} \left\{ b^T y \mid c - A^T y \in \mathcal{C}^* \right\}. \end{aligned}$$

So we have the following:

Definition 2.7 (Conic Dual Program)**.** We call *dual* of (2.3) the program

$$\begin{aligned} (\text{CP} - \text{D}) \qquad & \max_{y \in \mathbb{R}^n} b^T y \\ & \text{s.t. } c - A^T y \in \mathcal{C}^* \end{aligned}$$

Like in linear programming, we say that a point $x \in \mathcal{C}$ is feasible if $Ax = b$, and that $y \in \mathbb{R}^n$ is dual feasible if $c - A^T y \in \mathcal{C}^*$.
Moreover if x is a feasible optimal solution of (CP-P) and y of (CP-D), then we can define the duality gap as

$$c^T x - b^T y.$$

The following theorem holds

Theorem 2.4 (Duality Gap)**.** *Let x be a feasible solution of (CP-P) and y a dual feasible solution. Then*

$$c^T x - b^T y \geq 0$$

Proof. It comes directly from (2.4) in the derivation of the dual problem using Lagrange Multipliers given above. □

If $c^T x - b^T y = 0$, we say that *strong duality* holds.

Definition 2.8 (Slater's Condition)**.** We say that (CP-P) satisfies Slater's condition if there exist a point $\hat{x}$ such that $\hat{x}$ is feasible for (CP-P) and $\hat{x} \in \text{int}(\mathcal{C})$.
The same way, we say that (CP-D) satisfies Slater's condition if there exist a point $\hat{y}$ such that $\hat{y}$ is feasible for (CP-D) and $c - A^T \hat{y} \in \text{int}(\mathcal{C}^*)$.

Slater's condition is a very powerful tool to prove strong duality, since we have the following

Theorem 2.5. *Let p^* be a feasible solution for (CP-P) and d^* a feasible solution for (CP-D). Then:*

- *If (CP-P) satisfies Slater's condition and p^* is finite, then $p^* = d^*$ and p^* is attained for (CP-P).*
- *If (CP-D) satisfies Slater's condition and d^* is finite, then $p^* = d^*$ and d^* is attained for (CP-D)*
- *If both (CP-P) and (CP-D) satisfy Slater's condition, then $p^* = d^*$ and this value is attained for both problems.*

Strong duality is a very useful property, since solving only one between the primal and the dual problem, we get solutions for both. This means that we can chose to solve the easier of the two.
Moreover Slater's condition is a necessary condition for most Interior Point algorithms, and this is why we always need to be sure our problems satisfy it to be able to solve them.

2.2 From Polynomial to Conic

We have seen what a polynomial program is. Now we show hot to cast it as a conic program.
We start noticing that if λ^* is the optimal value of (PP-P), i.e.

$$\lambda^* = \min_{x \in S} f(x)$$

where S was the feasible region $S = \left\{ x \in \mathbb{R}^n \mid g_i(x) \geq 0 \quad \forall i \in \{1, ..., m\} \right\}$, then $f(x) - \lambda^* \geq 0 \quad \forall x \in S$.
But this means $f(x) - \lambda^*$ is a polynomial that is positive for all x in S. Then, to find the value λ^* we can solve the following

$$\begin{aligned} (PP - D) \quad & \sup_{\lambda} \lambda \\ & \text{s.t. } f(x) - \lambda \geq 0 \quad \forall x \in S. \end{aligned} \tag{2.5}$$

In conclusion our main problem is to understand what's the set of non-negative polynomials with variables in the semialgebraic set S.
We will now use the notation

$$\mathcal{P}(S) = \{ f(x) \in \mathbb{R}[x] \mid f(x) \geq 0 \ \forall x \in S \}$$
$$\mathcal{P}_d(S) = \{ f(x) \in \mathbb{R}_d[x] \mid f(x) \geq 0 \ \forall x \in S \}$$

where $\mathbb{R}[x]$ is the ring of polynomials, and $\mathbb{R}_d[x]$ of polynomials of degree at most d.
In the end (2.5) becomes

$$(PP-D) \quad \sup_{\lambda} \lambda \\ \text{s.t. } f(x) - \lambda \in \mathcal{P}(S) \tag{2.6}$$

It is easy to verify that $\mathcal{P}(S)$ is a convex cone. Unfortunately, it is not self-dual and optimizing over it is a $\mathcal{NP}$-hard problem. Indeed there is no way to characterize it yet. So we need a way to relax request (2.6).

2.2.1 Lasserre Hierarchies and Semidefinite Programming

The study of non-negative polynomials over a set S is a classic problem. For example, sum of square polynomials, that are polynomials of the form

$$p(x) = \sum_{i=1}^{K} q_i(x)^2$$

for some $K \in \mathbb{N}_+$, with q_i polynomials in n variables of degree at most r, are trivially non-negative polynomials. And the set of all sum of squares polynomials is also a cone. We will call it $SOS_r(n)$ $r, n \in \mathbb{N}$ (we will write $SOS(n)$ if $r = +\infty$).

Lemma 2.6 ($SOS_r(n)$ is a convex cone)**.** Let

$$SOS_r(n) = \left\{ p \in \mathbb{R}[x] \mid \ p(x) = \sum_{i=1}^{K} q_i(x)^2, \ \exists q_i \in \mathbb{R}_r[x] \ \forall i = 1, ..., K \right\}$$

Then $SOS_r(n)$ is a convex cone $\forall r \in \mathbb{N}_+$.

Proof. Let $p(x) \in SOS_r(n)$ for some $r \in \mathbb{N}_+$. We need to show that given a scalar $\lambda \geq 0$, then $\lambda p(x) \in SOS_r(n)$.
But

$$\lambda p(x) = \lambda \sum_{i=1}^{K} q_i(x)^2 = \sum_{i=1}^{K} \lambda q_i(x)^2 = \sum_{i=1}^{K} \left(\sqrt{\lambda} q_i(x) \right)^2$$

that is well defined since $\lambda \geq 0$. We prove that it is also convex. Given now $\lambda \in [0, 1]$, $p(x), s(x) \in SOS_r(n)$, is $\lambda p(x) + (1-\lambda)s(x) \in SOS_r(n)$?
We have

$$p(x) = \sum_{i=1}^{K} q_i(x)^2 \quad \text{and} \quad s(x) = \sum_{i=1}^{M} h_i(x)^2$$

since they are SOS. Then

$$\begin{aligned}\lambda p(x) + (1-\lambda)s(x) &= \lambda \sum_{i=1}^{K} q_i(x)^2 + (1-\lambda)\sum_{i=1}^{M} h_i(x)^2 \\ &= \sum_{i=1}^{K} \left(\sqrt{\lambda} q_i(x)\right)^2 + \sum_{i=1}^{M} \left(\sqrt{(1-\lambda)} h_i(x)\right)^2 \\ &= \sum_{i=1}^{K} l_i(x)^2\end{aligned}$$

where

$$l_i(x) = \begin{cases} \sqrt{\lambda} q_i(x), & i = 1, ..., K \\ \sqrt{(1-\lambda)} h_{i-K}(x), & i = K+1, ..., K+M \end{cases}$$

Moreover the degree of $p(x)$ can not exceed $2r$ since the degree of the sum of two polynomials is less or equal to the maximum degree of every polynomial. □

Is it true that all positive polynomials can be expressed as sum of squares? [Hil88]
Unfortunately not, for example the famous Motzkin [Mot67] polynomial $x_1^4 x_2^2 + x_1^2 x_2^4 - 3x_1^2 x_2^2 + 1$ in $\mathbb{R}[x_1, x_2]$ can not. Then we usually have

$$SOS_r(n) \subseteq \mathcal{P}_{2r}(\mathbb{R}^n)$$

But Hilbert was able to prove equality when $n = 1$ or $r = 1$ or $(n, r) = (2, 2)$.

This means that, if we want to describe polynomials in $\mathcal{P}_{2r}(S)$ with S semialgebraic set, we can not just restrict our search to $SOS_r(n) \cap S$.
Nonetheless it is true that SOS polynomials are always non-negative. But in general, it is very hard to find polynomials that are guaranteed to be non-negative over a semialgebraic set. This is why many partial results characterizing when a polynomial is non-negative were found. They come with the name of Positivstellensatz. Putinar's Positivstellensatz is the one we need, but we remand the interested reader to [Pò74], [Ste74], [Sch91].

The power of Putinar's result is that it tells us that every positive polynomial on a semialgebraic set S can be written exactly only using SOS polynomials and the polynomials appearing in the description of S. The hypothesis underlying this theorem are not restrictive either, it states the follwing:

Theorem 2.7 (Putinar's Positivstellensatz [Put93])**.** *With the notation used above, let* $g_0(x) = 1$ *and* $G = \{g_i \mid i = 0, ..., m\}$*, we define*

$$K(G) = \left\{ p(x) = \sum_{i=0}^{m} g_i(x) s_i(x) \mid s_i(x) \in SOS(n), g_i \in G \right\} \tag{2.7}$$

If there exists L such that $S \subseteq \{x \in \mathbb{R}^n | \sum_{i=1}^n x_i^2 \leq L\}$ and $\forall x \in S,\ p(x) > 0$, then $p(x) \in K(G)$.

$K(G)$ is usually called a *quadratic module* generated by G. This means that positive polynomials over S can be written as a finite sum of products between SOS polynomials and the constraints that define S.
This is a very useful result but has two drawbacks that we need to notice. The first one is that it characterizes only positive polynomials, while we would like to describe non-negative ones; the second one is that it poses no limit on the degree of the polynomials involved in the description of $K(G)$. Indeed it could happen that to describe $p(x)$ of small degree, one needs $s_i \in SOS_r(n),\ i = 1, ...m$ with r very large.
That S needs to be contained in a ball of some radius $\sqrt{L}$ is not a demanding request, since it naturally happens many times in practical application or it can be added as a constraints to the definition of S for L large enough. It can be proved that this request is also equivalent to a hypothesis of compactness of S. One might find other equivalent requests in the literature.

The unlimited degree in Putinar's Positivstelelnsatz is the biggest limit to practical use of the theorem. It was then Lasserre that, trying to overcome it, noticed the following [Las01]:

Theorem 2.8 (Lasserre's Theorem)**.** *Let $g_0(x) = 1$ and*

$$K_r(G) = \left\{ p(x) = \sum_{i=0}^{m} g_i(x) s_i(x) \mid s_i(x) \in SOS(n), \deg(g_i(x)s_i(x)) \leq 2r \right\} \tag{2.8}$$

Then

$$K_0(G) \subseteq K_1(G) \subseteq ... \subseteq K_r(G) \subseteq ... \subseteq \mathcal{P}(S)$$

Also we get $\{p \in \mathbb{R}[x] \mid p(x) > 0\ \forall x \in S\} \subseteq \bigcup_{r=0}^{+\infty} K_r(G)$.

Moreover, called

$$\begin{aligned} &\lambda_r^* = \sup_{\lambda} \lambda \\ &\text{s.t. } f(x) - \lambda \in K_r(G) \end{aligned} \tag{2.9}$$

then

$$\lambda_0^* \leq \lambda_1^* \leq ... \leq \lambda_r^* \leq ... \leq \lambda^*$$

with λ^* optimal value of (PP-D).

(2.9) are called Lasserre Hierarchies of order $2r$ (since the total degree of polynomials involved is $2r$).

Given a polynomial $p(x)$, let $d_p = \left\lceil \frac{\deg(p(x))}{2} \right\rceil$. Then notice that $\deg(g_i(x)s_i(x)) \leq 2r$ can be reformulated as

$$s_i \in SOS_{r-d_{g_i}}(n)$$

This means that we can approximate the solution of (PP-D) using SOS polynomials of fixed degree and that, the higher the degree, the better the approximation. We will see, again in practice, that many times the optimal value is reached already for small degree relaxations.

Remark 2.9. $K_r(G)$ is the truncation of order r of the quadratic module introduced previously. It is sometimes called *truncated module* of order r.

Motivated by this result, we need to better understand how to describe SOS polynomials and how to optimize over the cone $SOS_r(n)$.

To better work with SOS polynomials, we need a better description of them. That's why we will now study the relationship between SOS polynomials and *positive semidefinite* matrices. This will allow us to cast (2.9) as a *semidefinite* program, i.e. a conic program where $\mathcal{C}$ is the cone of positive semidefinite matrices.

Definition 2.9 (Positive Semidefinite Matrix)**.** Given a matrix $Q \in \mathbb{R}^n \times \mathbb{R}^n$, we say that Q is positive semidefinite (SDP) if

$$z^T Q z \geq 0, \quad \forall z \in \mathbb{R}^n$$

Lemma 2.10 (SDP matrices form a convex cone)**.** The set of all SDP matrices is a convex cone.

Proof. Let Q and z be as in the definition above. Then

$$z^T (\lambda Q) z = \lambda \left(z^T Q z\right) \geq 0, \quad \forall \lambda \geq 0$$

This prove the set of SDP matrices is a cone. We prove it is also convex. Let's take $Q, T \in \mathbb{R}^n \times \mathbb{R}^n$ SDP matrices, $z \in \mathbb{R}^n$, $\lambda \in [0, 1]$. We have

$$z^T (\lambda Q + (1-\lambda)T) z = \lambda z^T Q z + (1-\lambda) z^T T z \geq 0$$

since it is a sum of non-negative terms. □

This shows us that semidefinite programming is a subclass of conic programming, and so we can use interior point methods to solve this kind of problems.

Theorem 2.11. *A polynomial $p(x)$ is in $SOS_r(n)$ if and only if $p(x) = z(x)^T Q z(x)$, where Q is SDP and $z(x)$ is the vector of all monomials in x of degree at most r.*
That is $z = (x^a)$ with $|a| \leq r$. Then $Q \in \mathbb{R}^{N \times N}$ with $N = \binom{n+r}{r}$.

Proof. Assume $p(x)$ is SOS. Then $p(x) = \sum_{i=1}^{K} q_i(x)^2$. But each $q_i(x)$ can be written as $q_i(x) = \delta_i^T z(x)$, where $\delta_i \in \mathbb{R}^N$. Then we get

$$p(x) = \sum_{i=1}^{K} q_i(x)^2 = \sum_{i=1}^{K} \left(\delta_i^T z(x)\right)^T \delta_i^T z(x) = z(x)^T \left(\sum_{i=1}^{K} \delta_i \delta_i^T\right) z(x) \quad (2.10)$$

Then we say that $Q = \sum_{i=1}^{K} \delta_i \delta_i^T$.
Now let $p(x) = z(x)^T Q z(x)$ where Q is SDP. Then we can factorize Q as $Q = \sum_{i=1}^{K} \delta_i \delta_i^T$ and read (2.10) from right to left. □

So, unlike $\mathcal{P}(\mathbb{R}^n)$, $SOS_r(n)$ is easily described using SDP matrices. This means that (2.9) can be written as

$$f(x) - \lambda = \sum_{i=0}^{m} g_i(x) s_i(x) \quad (2.11)$$
$$s_i \in SOS_{r-d_{(g_i)}}(n)$$

that is

$$f(x) - \lambda = \sum_{i=1}^{m} g_i(x) \left(z(x)^T Q_i z(x)\right) \quad (2.12)$$
$$Q_i \succeq 0 \quad (2.13)$$

(2.12) can be formulated as linear constraints on $f(x) - \lambda$ and $g_i(x)$ coefficients and Q_i entries, while (2.13) are SDP constraints. This is our formulation as a conic program.

But also solving semidefinite programs can be very costly, since to check positive semidefiniteness for a $N \times N$ matrix one would need to check its eigenvalues, that is known to be an hard problem in computational mathematics.
This is why we can think to relax the problem even more, considering only a subclass of SOS polynomial, namely SDSOS polynomials.

2.2.2 From SOS to SDSOS

Ahmadi and Majumdar [AM17a], realizing that *SOS* relaxation could be very costly for mid to big sized problems, tried to relax (2.12) and (2.13) even more using a particular subclass of SDP matrices, and therefore a subclass of *SOS* polynomials.

Definition 2.10 (Sdd matrix)**.** A symmetric matrix $Q = (q_{ij})$ is diagonally dominant if $q_{ii} \geq \sum_{j \neq i}^{N} q_{ij}$ for all $j = 1, ..., N$.
Q is scaled diagonally dominant (sdd) if there exist a diagonal matrix D such that DQD is diagonally dominant.

Definition 2.11 (SDSOS polynomial)**.** We say that a polynomial $p(x)$ is $SDSOS_r(n)$ if

$$p(x) = z(x)^T Q z(x)$$

with Q sdd-matrix and z is the vector of all monomials of degree at most r.

SDSOS polynomials can also be seen as polynomial of the form

$$p(x) = \sum_{i=1}^{N} \alpha_i m_i^2(x) + \sum_{i,j=1}^{N} \left(\beta_{ij}^+ m_i(x) + \gamma_{ij}^+ m_j(x)\right) + \left(\beta_{ij}^- m_i(x) - \gamma_{ij}^- m_j(x)\right)$$

where m_i, m_j are monomials in n variables of degree at most r, α_i, β_{ij}^+, β_{ij}^-, γ_{ij}^+, γ_{ij}^- are scalars and $\alpha_i \geq 0$.

But usually we prefer working with sdd matrices instead, due to the following result that can be found in [AM17a]

Lemma 2.12. A symmetric matrix Q is sdd if and only if it can be expressed as

$$Q = \sum_{i<j} M^{ij} \tag{2.14}$$

where M^{ij} is an $N \times N$ matrix with all zeros entries except for $\left(M^{ij}\right)_{ii}$, $\left(M^{ij}\right)_{ij}$, $\left(M^{ij}\right)_{ji}$ and $\left(M^{ij}\right)_{jj}$ and $\begin{pmatrix} \left(M^{ij}\right)_{ii} & \left(M^{ij}\right)_{ij} \\ \left(M^{ij}\right)_{ji} & \left(M^{ij}\right)_{jj} \end{pmatrix}$ is SDP.

Checking if that 2×2 matrix is SDP is equivalent to checking

$$\left(M^{ij}\right)_{ii} + \left(M^{ij}\right)_{jj} \geq 0, \quad \left\| \begin{pmatrix} 2\left(M^{ij}\right)_{ij} \\ \left(M^{ij}\right)_{ii} - \left(M^{ij}\right)_{jj} \end{pmatrix} \right\| \leq \left(M^{ij}\right)_{ii} + \left(M^{ij}\right)_{jj} \tag{2.15}$$

This is also called a rotated second-order cone (SOC) constraints and they are much faster than SDP constraints. A program in which the conic constraints are all SOC constraints, is called a SOCP (where P stands for Program).

Then (2.6) can be relaxed also with

$$f(x) - \lambda = \sum_{k=1}^{m} g_i(x) \left(z(x)^T Q_k z(x)\right) \tag{2.16}$$

$$Q_k = \sum_{i<j} M_k^{ij} \tag{2.17}$$

$$\left(M_k^{ij}\right)_{ii} + \left(M_k^{ij}\right)_{jj} \geq 0, \quad \left\| \begin{pmatrix} 2\left(M_k^{ij}\right)_{ij} \\ \left(M_k^{ij}\right)_{ii} - \left(M_k^{ij}\right)_{jj} \end{pmatrix} \right\| \leq \left(M_k^{ij}\right)_{ii} + \left(M_k^{ij}\right)_{jj} \tag{2.18}$$

and once again (2.16) + (2.17) can be imposed using linear constraints and (2.18) are SOC constraints that are also solvable with interior points methods in a more efficient way than SDP constraints.

If we call

$$H_r(G) = \left\{ p(x) = \sum_{i=0}^{m} g_i(x) s_i(x) \mid s_i(x) \in SDSOS_{r-d_{g_i}}(n) \right\}$$

we can also rewrite the relaxation as

$$\begin{aligned} \lambda_r^* &= \sup_{\lambda} \lambda \\ &\text{s.t. } f(x) - \lambda \in H_r(G) \end{aligned} \tag{2.19}$$

The drawback of this method comes from the fact that

$$SDSOS_r(n) \subseteq SOS_r(n)$$

and, even when we reach convergence using the SOS relaxation, it might be that the $SDSOS$ one needs many more iterations to reach it. How good it might be to use $SDSOS$ instead of SOS highly depends on the objective function and the constraints of our problem.

Notice that, a hierarchy of order r has:

- Around $3m\binom{n+r}{r}^2$ variables (three for every M_k^{ij})
- $\binom{n+r}{r}$ linear constraints
- $\binom{n+r}{r}^2$ SOCP constraints (i.e. to check if those 2×2 matrices are PSD.

Then, increasing r means increasing the problem complexity a lot. Already for small values of n, having a big r leads to very costly problems.
This is what motivated the research for a new approach, namely the DIGS algorithm, where the value of r is fixed in advance.
We will study it in more detail from next chapter.

2.3 Moment Approach

Lasserre Hierarchy actually comes from using the so called *moment representation* for polynomials [Las01], [Las15], and was proved this way.
The moment approach is very useful for understanding polynomial programs and the relationship between dual and primal problems. Moreover we will introduce a new notation that will be useful later on and that's very common in the literature.

Let's suppose we are working on $\mathbb{R}^n$ with the usual *Borel σ-algebra* $\mathcal{B}(\mathbb{R}^n)$, and let μ be a Borel measure of probability, i.e. $\mu(\mathbb{R}^n) = 1$ and $\mu(A) \geq 0$, $\forall A \in \mathcal{B}(\mathbb{R}^n)$.

Definition 2.12 (Moment sequence)**.** Given a measure μ on $\mathbb{R}^n$, we call *moment of order a* the quantity

$$y_a = \int_{\mathbb{R}^n} x^a d\mu(x)$$

for $a \in \mathbb{N}^n$. We call *sequence of moments of μ* the sequence $(y_a)_{a\in\mathbb{N}^n}$ and *truncated sequence of order r* the sequence $(y_a)_{|a|\leq r}$, with $r \in \mathbb{N}$.

For example, given the Dirac measure δ_x, the sequence of moments of δ_x is $y_a = x^a,\ \forall a \in \mathbb{N}^n$.

2.3.1 Moment Matrices

Given a sequence $(y_a)_{a\in\mathbb{N}^n}$, we can define its moment matrix $M(y)$ as the infinite matrix indexed in $\mathbb{N}^n \times \mathbb{N}^n$, such that $M(y)_{(a,b)} = y_{a+b}$. The truncated matrix of order r is the submatrix of $M(y)_{(a,b)}$ given by the rows and columns in which $a \leq r, b \leq r$. We will denote in with $M_r(y)$.
We can now define a new sequence, that links our moment approach to polynomials.

Definition 2.13 (Shifted Vector)**.** Given $g \in \mathbb{R}[x], y \in \mathbb{R}^n$, we call *shifted vector* the sequence

$$gy := M(y)g \in \mathbb{R}^{\mathbb{N}^n}$$

Namely, $(gy)_a = \sum_b g_b y_{a+b}$.

The same definition holds if y is a truncated sequence, with $M_r(y)$ instead of $M(y)$.

Example 2.13. If $n = r = 2$, and $y = (y_{00}, y_{10}, y_{01}, y_{20}, y_{11}, y_{02}, ...)$ sequence associated to monomials in 2 variables, we get

$$M_2(y) = \begin{pmatrix} y_{00} & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\ y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\ y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\ y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\ y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\ y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04} \end{pmatrix}$$

that is the truncated matrix.
If we now consider $n = 2$, $d = 1$ and $g(x) = 5 - x_1^2 - x_2^2$, we have

$$M_1(gy) = \begin{pmatrix} 5y_{00} - y_{20} - y_{02} & 5y_{10} - y_{30} - y_{12} & 5y_{01} - y_{21} - y_{03} \\ 5y_{10} - y_{30} - y_{12} & 5y_{20} - y_{40} - y_{22} & 5y_{11} - y_{31} - y_{13} \\ 5y_{01} - y_{21} - y_{30} & 5y_{11} - y_{31} - y_{13} & 5y_{02} - y_{22} - y_{04} \end{pmatrix}$$

is the shifted moment matrix.

We have seen already that we can write a polynomial $p(x)$ as

$$p(x) = \sum_a p_a x^a$$

Indeed we can identify $p(x)$ with the vector of his coefficients $(p_a)_a$, indexed in $\mathbb{N}^n$ (or in $\mathbb{N}^n_r$). This way we can identify $\mathbb{R}[x]$ with a vectorial space. Then we can define linear forms on $\mathbb{R}[x]$.
Given a polynomial $f(x)$ and a sequence y, we are interested in the following linear form

$$L_y(f) = y^T f = \sum_a y_a f_a \tag{2.20}$$

$L_y(f)$ is called *Riesz linear functional*.

With next lemma we are going to show what's the link between moment matrices and this form.

Lemma 2.14. Let $y \in \mathbb{R}^{\mathbb{N}^n}, L_y$ the linear form discussed above and $f, g \in \mathbb{R}[x]$.
Then $L_y(fg) = f^T M(y) g$

Proof. $f(x)g(x) = \sum_c \left(\sum_{a+b=c} f_a g_b\right) x^c$. Then

$$L_y(fg) = \sum_c \left(\sum_{a+b=c} f_a g_b \right) y_c = \sum_a \sum_b f_a g_b y_{a+b} = f^T M(y) g.$$

□

Remark 2.15. From previous result, we get

- $L_y(f^2) = f^T M(y) f$
- $L_y(f) = \mathrm{vec}(1)^T M(y) f$

Remark 2.16 (Bilinear Form)**.** Lemma 2.14 gives us a method to extend L_y to the bilinear form

$$B_y(f, g) := L_y(fg)$$

and to the quadratic form

$$Q_y(f) := L_y(f^2)$$

We want now to understand when, given a sequence $y \in \mathbb{R}^{\mathbb{N}^n_{2r}}$, this is sequence a sequence of moments of some measure μ.
We leave the following important theorem without proof, since it would need advanced results in functional analysis.

Theorem 2.17 (Riesz-Haviland Theorem)**.** *Let $(y_a)_{a\in\mathbb{N}^n}$ be a sequence, $S \subseteq \mathbb{R}^n$ be closed. There exists a probability measure μ on S such that*

$$\int_S x^a d\mu(x) = y_a, \quad \forall a \in \mathbb{N}^n$$

if and only if $L_y(f) \geq 0$ for all polynomials $f \in \mathbb{R}[x]$ non-negative on S.

Of course this result is the link between non-negative polynomials and linear functionals.

Milder but very useful for practical implementation results are also the following:

Lemma 2.18. Let $y \in \mathbb{R}^{\mathbb{N}^n_{2r}}, g \in \mathbb{R}[x]$ and $d_g = \left\lceil \frac{\deg(g)}{2} \right\rceil$.

(i) If y is the sequence of moments of a measure μ, then $M_t(y) \succeq 0$.

(ii) If $t \geq d_g$ is the sequence of moments of a measure μ with support in $S = \{x \in \mathbb{R}^n \mid g(x) \geq 0\}$, then $M_{r-d_g}(gy) \succeq 0$.

Proof. (i) Let $p(x) \in \mathbb{R}_t[x]$. Then

$$\begin{aligned} p^T M_t(y) p &= \sum_{|a|\leq r} \sum_{|b|\leq r} p_a p_b y_{a+b} \\ &= \sum_{|a|\leq r} \sum_{|b|\leq r} p_a p_b \int_{\mathbb{R}^n} x^{a+b} d\mu(x) \\ &= \int_{\mathbb{R}^n} p(x)^2 d\mu(x) \geq 0 \end{aligned}$$

(ii) Let $p(x) \in \mathbb{R}_{r-d_g}[x]$. Then

$$\begin{aligned} p^T M_t(gy) p &= \sum_{|a|\leq r-d_g} \sum_{|b|\leq r-d_g} p_a p_b (gy)_{a+b} \\ &= \sum_{|a|\leq r-d_g} \sum_{|b|\leq r-d_g} \sum_{c\in\mathbb{N}^n} p_a p_b g_c y_{a+b+c} = \\ &= \int_S g(x) p(x)^2 d\mu(x) \geq 0. \end{aligned}$$

□

We can even extend point (ii) to $S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i = 1, ..., m\}$ using $d_S = \max_{i=1,...,m}\{d_{g_i}\}$ and $r \geq d_S$, and we get

$$M_r(y) \succeq 0, \quad M_{r-d_{g_i}}(g_i y) \succeq 0,\ i = 1, ..., m$$

2.3.2 Moment relaxation of a Polynomial Program

We can easily notice that

$$\lambda^* = \inf_{x \in S} p(x) = \inf_{\mu} \int_S p(x) d\mu(x)$$

Indeed, for $\bar{x} \in S$, $p(\bar{x}) = \int_S p(x) d\delta_{\bar{x}}(x)$, showing that $\lambda^* \geq \inf_\mu \int_S p(x) d\mu(x)$. Conversely, $p(x) \geq \lambda^*$ for all $x \in S$. Then

$$p(x) \geq \lambda^* \Rightarrow \int_S p(x) d\mu(x) \geq \int_K \lambda^* d\mu(x) = \lambda^*$$

since μ is a probability measure.
Moreover, since $p(x)$ is a polynomial, we get

$$\int_S p(x) d\mu(x) = \int_S \sum_a p_a x^a d\mu(x) = \sum_a p_a \int_S x^a d\mu(x) = p^T y$$

with y sequence of moments of μ.

We can then rewrite $(PP - P)$ as

$$\begin{aligned} \lambda^* = & \inf p^T y \\ & \text{s.t. } y_0 = 1 \\ & y \text{ has a representing measure on } S \end{aligned}$$

Using Lemma 2.18, we can relax it to

$$\lambda_r^{mom} = \inf_{y \in \mathbb{R}^{\mathbb{N}^n_{2r}}} p^T y \tag{2.21}$$

$$\text{s.t } y_0 = 1 \tag{2.22}$$

$$M_r(y) \succeq 0 \tag{2.23}$$

$$M_{r-d_{g_i}}(g_i y) \succeq 0, \ i = 1, ..., m \tag{2.24}$$

Or, equivalently using the linear forms,

$$\begin{aligned} \lambda_r^{mom} = & \inf_{L \in (\mathbb{R}[x])^*} L(p) \\ & \text{s.t. } L(1) = 1 \\ & L(f) \geq 0 \ \forall f \in K_r(G) \end{aligned}$$

and $K_r(G)$ is the set defined in (2.8).

Finally, if S has non-empty interior, one can show that $\lambda_r^* = \lambda_r^{mom}$. We have already seen that we attain the supremum for λ_r^* (and therefore for λ_r^{mom}) whenever S is inside a ball of radius big enough.

2.3.3 Frobenius Inner Product

We end the section introducing another way to write conic programs. This approach is largely used nowadays but was developed recently.

Definition 2.14 (Frobenius Inner Product)**.** Given two matrices $A, B \in \mathbb{R}^n \times \mathbb{R}^m$, we define the product

$$A \cdot B := \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij} B_{ij} = \mathrm{tr}(A^T B)$$

Example 2.19. For example, given a polynomial $p(x1, x2) \in \mathbb{R}_2[x1, x2]$, such that

$$p(x) = p_{00} + p_{10}x_1 + p_{01}x_2 + p_{20}x_1^2 + p_{11}x_1x_2 + p_{02}x_2^2$$

and y sequence of moments of x, we can define

$$P = \begin{pmatrix} p_{00} & \frac{p_{10}}{2} & \frac{p_{01}}{2} \\ \frac{p_{10}}{2} & p_{20} & \frac{p_{11}}{2} \\ \frac{p_{01}}{2} & \frac{p_{11}}{2} & p_{02} \end{pmatrix} \quad Y = \begin{pmatrix} y_{00} & y_{10} & y_{01} \\ y_{10} & y_{20} & y_{11} \\ y_{01} & y_{11} & y_{02} \end{pmatrix}$$

Then

$$P \cdot Y = p_{00}y_{00} + p_{10}y_{10} + p_{01}y_{01} + p_{20}y_{20} + p_{11}y_{11} + p_{02}y_{02} = p^T y$$

Looking at Example 2.19, we understand that we can write (2.21) as

$$\lambda_r^{mom} = \inf_Y P \cdot Y$$

where Y is exactly $M_r(y)$, and so (2.22) and (2.23) become

$$\begin{aligned} H \cdot Y &= 1 \\ Y &\succeq 0 \end{aligned}$$

where H is a matrix with a 1 in entry $(0, 0)$ and 0 otherwise.
Lastly, (2.24) involves the use of shifted moment matrices. Calling $Y_{g_i} := M_{r_0 - d_{g_i}}(g_i y)$, we get

$$Y_{g_i} \succeq 0$$

And then the full problem can be cast as

$$\begin{aligned} \lambda_r^{mom} = \inf_Y \ & P \cdot Y \\ & s.t.\ H \cdot Y = 1 \\ & Y \succeq 0 \\ & Y_{g_i} \succeq 0 \end{aligned}$$

With SDSOS

In the SOS case, we get that both in the primal and dual formulation we need matrices to be SDP. This is because the dual space of SDP matrices is the space of SDP matrices itself. This doesn't happen with sdd matrices. The dual space of sdd matrices is instead given by

$$\mathbb{T}_+^n := \left\{X \in \mathbb{R}^n \times \mathbb{R}^n \mid X = X^T \text{ and } X^{ij} \succeq 0\right\}$$

where X^{ij} is the 2×2 minor specified by (i, j).
Understanding the proof of this would need to introduce a better description of what a cone is, we remand the interested reader to [AH15].
So, (2.23) and (2.24) become

$$\begin{aligned} &Y \in \mathbb{T}_+^N \\ &Y_{g_i} \in \mathbb{T}_+^{N-d_{g_i}} \end{aligned}$$

and so the full problem is

$$\begin{aligned} \lambda_r^{mom} = &\inf_Y P \cdot Y \\ &s.t.\ H \cdot Y = 1 \\ &Y \in \mathbb{T}_+^N \\ &Y_{g_i} \in \mathbb{T}_+^{N(d_{g_i})} \end{aligned} \tag{2.25}$$

where $N = \binom{n+r_0}{r_0}$ and $N(d_{g_i}) = \binom{n+r_0-d_{g_i}}{n}$.

Before ending the chapter, we want to give also the formulation for (2.25) in the case of a quadratic program with $r = 2$, since it will be used later. We call $Q_i,\ i = 1, ..., m$ the matrices of quadratic constraints. Then we have:

$$\begin{aligned} \lambda_r^{mom} = &\inf_Y P \cdot Y \\ &s.t.\ H \cdot Y = 1 \\ &Y \in \mathbb{T}_+^N \end{aligned} \tag{2.26}$$

$$Q_i \cdot Y \geq 0 \tag{2.27}$$

where (2.27) comes from the fact that we are considering a matrix of moment truncated at 0, that is exactly $g_i(x)$ or in matrix form $Q_i \cdot X$.

2.4 Interior Point algorithms

Interior Point algorithms are the tool we use to optimize conic programs. There are many algorithms under this name, but the way they work is very

similar. We present only one example of Interior Point algorithm to give a hint to the reader on how they work in general. We follow the work of [NT98].

Let's take in consideration the couple of conic programs presented previously (CP-P) and (CP-D). We recall $\mathcal{C}$ is a pointed closed convex cone. We also assume that Slater's condition holds for both the primal and dual program (i.e. we have strong duality).
We introduce the function

$$F : \text{int}(\mathcal{C}) \longrightarrow \mathbb{R}$$

and we say that F is a barrier function if

- F is strictly convex
- Taken $(x_k)_{k\in\mathbb{N}} \in \text{int}(\mathcal{C})$ sequence of points in the interior of $\mathcal{C}$, such that $\lim_{k\to+\infty} x_k = \bar{x} \in \partial\mathcal{C}$, then $\lim_{k\to+\infty} F(x_k) = +\infty$.

Let F_* be a barrier function for the dual problem. We chose a value μ and we introduce the following *Barrier programs.*

$$(BP_\mu) \quad \min_x \; c^T x + \mu F(x) \qquad \text{s.t. } Ax = b$$

$$(BD_\mu) \quad \max_{y,s} \; b^T y - \mu F_*(s) \qquad \text{s.t. } A^T y + s = c$$

(s is a slack variable that we use to rewrite the conic dual program in a more readable way. It is easy to check that this is the same formulation we gave previously) Those programs have solutions $x(\mu)$ and $y(\mu), s(\mu)$ respectively, that change smoothly with μ. The trajectories they form are called *central paths.* This is why this kind of algorithm is usually also called central path algorithm.
Of course now we need to characterize the barrier function F. We start by asking for some properties we'd like it to have [NN94].
We say that F is ν-self-concordant (SCB) for $\mathcal{C}$ if

- $F \in C^3(\mathcal{C}; \mathbb{R})$
- For all $x \in \text{int}(\mathcal{C})$, $\frac{\partial^2 F}{\partial x^2}$ is positive definite.
- For all $x \in \text{int}(\mathcal{C})$ and $d \in \mathbb{R}^n$ the following holds:
  - $\frac{\partial^3 F}{\partial x^3}(d,d,d) \leq 2\left(\frac{\partial^2 F}{\partial x^2}(d,d)\right)^{3/2}$
  - $\frac{\partial F}{\partial x} d \leq \sqrt{\nu}\left(\frac{\partial^2 F}{\partial x^2}(d,d)\right)^{1/2}$

We also say it is ν-logarithmically homogeneous (LHSCB) if, for all $x \in \text{int}(\mathcal{C})$, $\tau > 0$, it holds $F(\tau x) = F(x) - \nu \ln \tau$.
If F is ν-LHSCB, we deine F_* as

$$F_*(s) := \sup -s^T x - F(x)$$

and we have that F_* is ν-LHSCB for $\mathcal{C}^*$.

Example 2.20. We give here some examples of barrier functions we are interested in.

- If $\mathcal{C}$ is the cone of SDP matrices in $\mathbb{R}^n$, then $F(X) = -\log(\det(X))$ and it is ν-LHSCB with $\nu = n$.
- If $\mathcal{C}$ is the cone $\mathbb{T}_+^{n+1}$, then $F(X) = -\ln(x_0^2 - ||\bar{x}||_2^2)$ with $\nu = 2$ and $\bar{x} = (x_1, ..., x_n)$

Once we create an F as above, the following holds

Lemma 2.21. x is optimal for (BP_μ) and y, s for (BD_μ) if and only if they solve the following set of equations

$$\begin{aligned} &A^T y + s = c, \quad s \in \text{int}(\mathcal{C}^*) \\ &Ax = b, \quad x \in \text{int}(\mathcal{C}) \\ &\mu F'(x) + s = 0 \end{aligned}$$

The idea of Interior Point methods is then that, using Newton method, we can solve these equations and find $x(\mu), y(\mu), s(\mu)$ that, for μ going towards 0, converge to the optimal solution of (CP-P) and (CP-D).

Now we have all the tools to talk about DIGS, namely the new polynomial optimization algorithm that we will present in next chapter.

Chapter 3

The algorithm DIGS

In the previous chapter we have seen how Lasserre Hierarchies work and what are the main problem about them, namely the fast growth of the size of the problem when increasing the degree of the hierarchy.

Here we want to introduce a new approach, presented for the first time in [Gha11], and then outline the scope of this work, that is adapt this method to the use of SOCP programming instead of SDP.

3.1 Dynamic Inequality Generation Scheme

We recall that in the previous chapter we wanted to solve the problem

$$\begin{aligned} \textbf{(PP-P)} \quad & \rho = \inf_{x \in \mathbb{R}^n} f(x) \\ & \text{s.t.} \quad g_i(x) \geq 0 \qquad i = 1, ..., m \end{aligned}$$

and that we were left with the following relaxation

$$\begin{aligned} & \lambda_r^* = \sup_{\lambda} \lambda \\ & \text{s.t. } f(x) - \lambda \in K_r(G) \end{aligned}$$

where

$$K_r(G) = \left\{ p(x) = \sum_{i=0}^{m} g_i(x) s_i(x) \mid s_i(x) \in SOS(n), \deg(g_i(x) s_i(x)) \leq 2r \right\}$$

and $G = \{g_i \mid i = 1, ..., m\}$.
We also showed that, increasing r, these relaxations converge to the actual optimum of (PP-P) but that this approach led to large size problems.

With the approach we present in this chapter, we fix the degree of $K_r(G)$ to a small value, for example $r_0 = \max\{d_{g_i}, d_f\}$ and, instead of improving the relaxation increasing the degree of it, we want to show that we can improve it by adding constraints.
But not every constraint works fine, so our problem will be how to generate a new constraint $p(x)$ such that $K_{r_0}(G \cup \{p(x)\})$ gives a better approximation than $K_{r_0}(G)$.

3.1.1 Improving Inequalities

We notice that, if $p(x) \in K_{r_0}(G)$, then $K_{r_0}(G \cup \{p(x)\}) = K_{r_0}(G)$. Then, if we want to improve our relaxation, our first request for $p(x)$ is not to be in $K_{r_0}(G)$ already. If that happens, we have

$$K_{r_0}(G) \subset K_{r_0}(G \cup \{p(x)\}) \subseteq \mathcal{P}_{2r_0}(S)$$

The second problem is that we want a $p(x)$ such that $p(x) \geq 0 \ \forall x \in S$. So we want $p(x)$ to be non-negative over S, namely $p(x) \in \mathcal{P}_{2r_0}(S)$.
This means we need to find $p(x) \in \mathcal{P}_d(S) \setminus K_{r_0}(G)$ with $\deg(p) \leq r_0$.

3.1.2 The algorithm

The algorithm will work this way: at every iteration we solve a Lasserre Hierarchy of fixed degree. Using its solution, we look for an improving inequality and we update the hierarchy with it. We stop as soon as we can not find new inequalities to add.

Algorithm 1 is what we call master problem. We call subproblem the problem we need to solve in order to generate improving inequalities.

First of all, notice that in Algorithm 1 we have to deal with the description of $\mathcal{P}_{2r_0}(S)$ once again. And, since we don't have a good way to represent it, we are going to approximate $\mathcal{P}_{2r_0}(S)$ with $K_{r_0+1}(S)$. Since $K_{r_0}(S) \subset K_{r_0+1}(S)$, we know that $K_{r_0+1}(S) \setminus K_{r_0}(S)$ is not empty.

Now that we have a description for $(PP - M_k)$, we can also write its dual. We get

$$\begin{aligned} &\inf_Y \ (f, Y) \\ &\text{s.t. } (1, Y) = 1 \\ &\quad Y \in (K_{r_0}(G))^* \end{aligned}$$

This comes from the description using moments that we outlined in Chapter 2.
Why is it important to study the dual of $(PP - M_s)$?
Because thanks to the property of the dual cone, we will be able to extract

Algorithm 1 Master Problem for DIGS with SOS polynomials

Require: $G \subseteq \mathbb{R}_d[x]$ description of S and $2r_0 \geq d$
$s \leftarrow 0, G_0 \leftarrow G$
loop
Let

$$(PP - M_s)\ \nu_s = \sup_{\lambda} \lambda \quad \text{s.t.} f(x) - \lambda \in K_{r_0}(G_s)$$

Subproblem: Look for an improving inequality

$$p_s(x) \in \mathcal{P}_{2r_0}(S) \setminus K_{r_0}(G_s)$$

if Improving equality doesn't exist **then**
STOP
else
$G_{s+1} \leftarrow G_s \cup \{p_s(x)\}$
$s \leftarrow s + 1$
end if
end loop

improving equality using the solution of the dual problem. Indeed we recall that

$$(K_{r_0}(G))^* = \left\{Y \in \mathbb{R}^N \mid (p, Y) \geq 0,\ \forall p \in K_{r_0}(G)\right\}$$

This means that, given a feasible solution f for $(PP - M_k)$, let Y be its dual solution. We can understand when a polynomial $p(x)$ is not in $K_{r_0}(G)$, i.e. whenever $(p, Y) < 0$.
To recap, if we find $p(x)$ in $K_{r_0+1} \cap \mathbb{R}_{2r_0}[x]$ such that $(p, Y) < 0$, we have found an improving equality for $(PP - M_s)$.
We are now ready to describe the full algorithm in Algorithm 2.

Algorithm 2 DIGS Algorithm using SOS polynomials

Require: G description of S, $2r_0 \geq d$ and $\epsilon \geq 0$
  $s \leftarrow 0, G_0 \leftarrow G$
  loop
    Let

$$(PP - M_s)\ \nu_s = \sup_{\lambda} \lambda \quad \text{s.t.} f(x) - \lambda \in K_{r_0}(G_s)$$

    $Y_s \leftarrow$dual optimal solution of $(PP - M_s)$
    Subproblem: Let $p_s(x) \in \mathbb{R}_{2r_0}[x]$ be an optimal solution of

$$(PP - Sub_s)\ \alpha_s = \min_{p}(p, Y_s) \quad \text{s.t. } p \in K_{r_0+1}(G_s) \cap \mathbb{R}_{2r_0}[x] \quad ||p|| = 1$$

    if $\alpha_s < -\epsilon$ **then**
      $G_{s+1} \leftarrow G_s \cup \{p_s(x)\}$
      $s \leftarrow s + 1$
    else
      STOP
    end if
  end loop

There are a few things to point out:

- We are here looking for the minimum of (p, Y_s). This is not strictly necessary, as we only care that $(p, Y_s) \leq 0$. But this kind of problem is usually referred as *pricing problem*, and heuristically the best bound is achieved using the minimum.
  Indeed this problem can be relaxed and stopped before the minimum is reached. Also, further studies can be done to understand if there is a way to extract stronger inequalities.

- We are forcing $||p|| = 1$. This is necessary since we know that, for every $C > 0$, constraints $p(x)$ and $Cp(x)$ are the same but $||Cp|| = C||p||$. So imposing norm of p to be equal to one is a way to force uniqueness of the representation of equal constraints.

- We haven't specified any norm yet. We have yet to study how different norms influence the performances of the algorithm.
  Commonly used norms are:

- $||p||_2 = \left(\sum_{|a|=2r_0} p_a^2 \right)^{\frac{1}{2}}$
- $||p||_1 = \sum_{|a|=2r_0} |p_a|$

- We stop the iterations whenever $\alpha_s \geq -\epsilon$ for ϵ very small instead of 0, since we are working in finite arithmetic.

Now that we have a formulation of DIGS using SOS polynomials, we are ready to give the same formulation using $SDSOS$.

3.1.3 DIGS with SDSOS

The formulation of the problem is almost the same, but we have SDSOS polynomials instead. The full algorithm in this case would be

Algorithm 3 DIGS Algorithm using SDSOS polynomials

Require: G description of S, $2r_0 \geq d$ and $\epsilon \geq 0$
$s \leftarrow 0, G_0 \leftarrow G$
loop
Let

$$(PP - M_s)\ \nu_s = \sup_{\lambda} \lambda \quad \text{s.t.} f(x) - \lambda \in H_{r_0}(G_s)$$

$Y_s \leftarrow$dual optimal solution of $(PP - M_s)$
Subproblem: Let $p_s(x) \in \mathbb{R}_{2r_0}[x]$ be an optimal solution of

$$(PP - Sub_s)\ \alpha_s = \min_{p}(p, Y_s) \quad \text{s.t. } p \in H_{r_0+1}(G_s) \cap \mathbb{R}_{2r_0}[x] \quad ||p|| = 1$$

if $\alpha_s < -\epsilon$ **then**
$G_{s+1} \leftarrow G_s \cup \{p_s(x)\}$
$s \leftarrow s + 1$
else
STOP
end if
end loop

Notice that the only difference is that we have $H_{r_0}(G)$ and $H_{r_0+1}(G)$ instead of $K_{r_0}(G)$ and $K_{r_0+1}(G)$ respectively.

3.2 Analysis of dimensionality

We compare now Lasserre Hierarchy approach with DIGS, in both SOS and SDSOS case, for a problem of order r, with n variables and m constraints. To simplify the notation, we use

$$N_{n,r} = \binom{n+r}{n}$$

Notice that $N_{n,r} = O(n^r)$

3.2.1 Lasserre Hierarchy with SOS polynomials

Since we have n variables and the maximum degree of polynomials involved is $2r$, we have a constraint for every possible monomial. The number of monomials that can appear in a polynomial of degree $2r$ in n variables is $N_{n,2r}$. Indeed these constraints are equality constraints between the coefficients of $f - \lambda$ and $\sum_{i=0}^{m} s_i(x)g_i(x)$.
Since we write s_i as matrices that we force to be SDP, we also have $m+1$ conic constraints imposing $Q_i \succeq 0$.
The amount of variables involved is given by the entries of Q_i matrices. Those matrices are symmetric and, since in our constraints we have

$$\deg\Big(g_i(x) \left[z(x)^T Q_i z(x) \right] \Big)$$

Q_i is a $N_{n,r-d_{g_i}} \times N_{n,r-d_{g_i}}$ matrix.
This means that the amount of distinct entries in Q_i are

$$\left[N_{n,r-d_{g_i}} + 1 \right] \frac{N_{n,r-d_{g_i}}}{2}$$

For Q_0 it is $O(n^r)$ (since $d_{g_0} = 0$) and, when increasing the value of r, this dominates the dimensionality of the problem.

So, once again, there are

- A $O(n^{2r}) \times O(mn^r)$ matrix of linear constraints
- $m+1$ Semidefiniteness constraints on $O(n^{r-d_{g_i}}) \times O(n^{r-d_{g_i}})$

3.2.2 Lasserre Hierarchy with SDSOS polynomials

We can make almost the same considerations of before, this time we have that each Q_i is decomposed in the sum of matrices that are sdd. Indeed we have

$$Q_k = \sum_{i<j} M_k^{ij}$$

Here $i = 1, ..., N_{n,r-d_{g_i}}$; $j = i, ..., N_{n,r-d_{g_i}}$. This means we have a total of

$$\frac{3}{2}\left[N_{n,r-d_{g_i}} - 1\right] N_{n,r-d_{g_i}}$$

linear constraints. Here 3 are the numbers of non-zero entries of every M_k^{ij}. Each one of these matrices has to satisfy a $SOCP$ constraint, so we have

$$\left[N_{n,r-d_{g_i}} - 1\right] \frac{N_{n,r-d_{g_i}}}{2}$$

SOCP constraints. In the end, we have:

- A $O(n^{2r}) \times O(mn^{2r})$ matrix for linear constraints
- $O(mn^{2r})$ SOCP constraints on 2×2 matrices

Increasing the degree of Lasserre Hierarchy is usually a much stronger method than adding an improving inequality, but is prohibitive due to the size increasing very fast as soon as r becomes bigger.

3.2.3 Dimensionality of DIGS

We see now that, if we set $r_0 = d$, the dimension of DIGS is dominated by the dimension of Lasserre Hierarchy of order $2r_0+2$, due to the Subproblem. For example, if we want to solve a quadratic problem, we can set $r_0 = 1$, and the dimension of the DIGS problem will be approximately the same of a Lasserre Hierarchy of degree $2r_0 + 2 = 2 + 2 = 4$.

But, a new iteration of DIGS doesn't improve the dimensionality of the whole problem that much since it only adds a constraint $p_s(x)$, that is

- For SOS: it adds $O(n^{r_0+1-d_{p_s}})$ variables and a semidefiniteness constraint on the constraint matrix
- For SOCP: it adds $O(n^{2(r_0+1-d_{p_s})})$ variables and $O(n^{2(r_0+1-d_{p_s})})$ SOCP constraints on 2×2 matrices

So, while Lasserre increases exponentially, here with each iteration the growth it is only polynomial.

Unfortunately, it is very hard to study convergence rate for this kind of problem, and right now there is close to no literature about them. But there are already many promising results about DIGS, showing that for some problems it can converge to the optimal solution faster than Lasserre Hierarchies, or reach a better bound than Lasserre's approach (since many times Lasserre Hierarchy can not run due to time or memory limitations).

More about DIGS-SDSOS

It is very hard to compare the two approaches for DIGS from a theoretical point of view. But we will see, experimentally, that many times SDSOS bounds are way weaker that SOS ones. So why should we focus our attention on the SDSOS approach?
The fact is that, the biggest problem right now for polynomial optimization is to have efficient tools to solve mid-large sized polynomial problems. Semidefinite programming scales very badly, since imposing positive semidefiniteness to a large matrix is usually very expensive in terms of time. That's why many optimization software nowadays don't implement an SDP solver anymore, and only implement SOCP ones.
SDSOS approach, even if right now is weaker, has better chances to improve in the future. Indeed DIGS approach is very similar to a column generation scheme by Ahmadi et al. [ADH15]. Also there the author both implemented the SOS and SDSOS approach. The latter showed worse performances than SOS approach in small instances but the ability to run for large size problems.

During our work we also focused on testing DIGS-SDSOS on high instances, to understand how we could improve its performances already.

3.3 Equalities, exploiting sparsity, adding inequalities and other improvements

One of the strength of this approach is that whenever we find a way to improve Lasserre Hierarchy, those improvements can be applied to DIGS as well. And since DIGS order is fixed to a small value, they are usually very effective.

3.3.1 Dealing with equalities

For example, in $(PP - P)$ we considered only constraints of the form

$$g_i(x) \geq 0$$

and never equality constraints. This is because every constraint of the form $g_i(x) = 0$ can be rewritten as a couple of constraints

$$g_i(x) \geq 0$$
$$g_i(x) \leq 0$$

But this also means that we are increasing the number of constraints needed to solve an equality constraint. We would like to be able to deal with

equalities directly.
In this case we can think at S as the set

$$S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i = 1, ..., m_1,\ h_j(x) = 0,\ j = 1, ..., m_2\}$$

and so we get

$$G = \{g_i(x), h_j(x) \mid i = 1, ..., m_1,\ j = 1, ..., m_2\}$$

Then, since $h_j(x) = 0\ \forall x \in S$, this means

$$h_j(x)q(x) = 0,\ \forall x \in \mathbb{R}^n,\ q(x) \in \mathbb{R}[x]$$

From this we get that

$$K_r(G) = \left\{ p(x) = \sum_{i=0}^{m_1} g_i(x)s_i(x) + \sum_{j=1}^{m_2} h_j(x)\delta_j(x) \mid \begin{matrix} s_i(x) \in SOS_{r-d_i}(n), \\ \delta_j(x) \in \mathbb{R}_{2r-deg(\delta_j)}[x] \end{matrix} \right\}$$

This is a huge improvement, especially when there are many equalities involved, since this doesn't just reduce the number of constraints, but also constraints on δ_js are only linear and they don't involve any SDP constraint.
A more detailed proof of this can be found in [PVZ08].
In this article the author uses the following hypothesis:

i $K_r(G) \subseteq K_{r+1}(G) \subseteq \mathcal{P}(S),\ r = 0, 1, ...$

ii $\mathcal{P}_+(S) \subseteq \cup_{r=0}^{\infty} K_r(G) \subseteq \mathcal{P}(S)$

that are always verified for Lasserre Hierarchies thanks to Putinar's theorem.

The same way we can heuristically apply it to $H_r(G)$ with $SDSOS$ instead of SOS and avoid many $SOCP$ constraints. Unfortunately though, while hypothesis (i) holds true even for $H_r(G)$, the same is not always true for hypothesis (ii) (see [AM17b]). That those hypothesis are true should be checked whenever someone wants to use this method, since we weren't able to prove or find proofs of the same result for $SOCP$ hierarchies.

A good example in which this improvement is very effective are binary programs, since in there we have an equality $x_i(x_i - 1) = 0$ for every x_i binary.

3.3.2 Exploiting Sparsity

Another thing to notice is that, given every polynomial constraint $g_i(x)$ has to be multiplied by $s_i(x)$ SOS-polynomial in (2.11) to get

$$f(x) - \lambda = \sum_{i=0}^{m} g_i(x)s_i(x)$$

Many times it happens that some monomials in the result of $g_i(x)s_i(x)$ don't have any real effect in the solution of the problem, since they don't appear in $f(x)$ or in any $g(x)$, but they only appear due to the multiplication of g_i with s_i.
We need to formalize this, and find a way to describe those monomials and prove that they are not needed in our optimization framework.

This was done for the SOS case by Fukuda [Fuk+01] already in 2001, using chordal graphs to detect unnecessary monomials. It was proven that for quadratic problems, the sparse formulation (the formulation given by getting rid of those monomials) gave the same result as the dense formulation. For problems of degree 3 or more, the sparse formulation is a relaxation of the dense problem. But has the pro that it greatly improves time and memory usage for big problems.
There is also a toolbox for Matlab called SparseCOlO [Zhe+16], [Zhe+] that implements this method.

Recently a new sparsity exploitation method was published by Sheen and Yamashita [SY19], regarding $SOCP$ relaxation (Lasserre Hierarchy of order 2) of quadratic programs. In it, the authors proven equality between their sparse formulation and the full formulation of the problem.
We have seen already that, using matrix notation, an $SOCP$ relaxation of order 2 for a quadratic problem can be written as (check [KK01])

$$\begin{aligned} \min_{Y} \; & P \cdot Y \\ s.t. \; & Q_k \cdot Y \geq 0, \quad k = 1, ..., m \\ & H_0 \cdot Y = 1 \\ & Y \in \mathbb{T}_+^N \end{aligned} \tag{3.1}$$

Now, if we denote with E the $N \times N$ matrix such that

$$E_{ij} = \begin{cases} 1 \text{ if } (Q_k)_{ij} \neq 0 \text{ forsome } k \text{ or } P_{ij} \neq 0 \text{ or } i = j \\ 0 \text{ otherwise} \end{cases}$$

we can create the set $J_E = \{(i,j) \in N \times N \mid E_{ij} = 1\}$.
This was needed to introduce 2 more sets:

$$\mathbb{T}_+^N(X, E, ?) := \left\{X \in sym(\mathbb{R}^N) \mid \exists \bar{X} \in \mathbb{T}_+^N \text{ s.t. } X_{ij} = \bar{X}_{ij} \; \forall (i,j) \in J_E\right\}$$
$$\mathbb{T}_+^N(X, E, 0) := \left\{X \in \mathbb{T}_+^N(X, E, ?) \mid \bar{X}_{ij} = 0 \; \forall (i,j) \notin J_E\right\}$$

$\mathbb{T}_+^N(X, E, ?)$ is the set of all matrices that can be completed to a matrix in $\mathbb{T}_+^N$ by changing elements in $N \times N \setminus J_E$, and $\mathbb{T}_+^N(X, E, 0)$ the sets of matrices that are in $\mathbb{T}_+^N$ if elements outside J_E are put to 0.

Now we also introduce the set

$$\bar{\mathbb{T}}_+^N(E) := \left\{X \in sym(\mathbb{R}^N) \mid X^{ij} \succeq 0 \forall (i,j) \in J_E\right\}$$

Notice that, since $(i,i) \in J_E \ \forall i = 1,...,N$, every diagonal element of $X \in \bar{\mathbb{T}}_+^N(E)$ is non-negative.
We want to prove that $\mathbb{T}_+^N(X,E,?)$ is equivalent to $\bar{\mathbb{T}}_+^N(E)$. For $X \in \bar{\mathbb{T}}_+^N(E)$, we define

$$R_{ij}(X) := \left[-\sqrt{X_{ii}X_{jj}}, \sqrt{X_{ii}X_{jj}}\right]$$

and we call this value *range* (i,j) of X.
There is a last set we need to introduce:

$$\hat{T}_+^N(E,X) := \left\{\bar{X} \in sym(R^N) \mid \begin{matrix}\bar{X}_{ij} = X_{ij} \forall (i,j) \in J_E, \\ \bar{X}_{ij} \in R_{ij}(X) \forall (i,j) \notin J_E\end{matrix}\right\}$$

Lemma 3.1. It holds that $\mathbb{T}_+^N(E,?) = \bar{\mathbb{T}}_+^N(E)$

Proof. $\mathbb{T}_+^N(E,?) \subseteq \bar{\mathbb{T}}_+^N(E)$:
Fix $X \in \mathbb{T}_+^N(E,?)$. Then there exists $\bar{X} \in \mathbb{T}_+^N$ such that $\bar{X}_{ij} = X_{ij}$ for $(i,j) \in J_E$. We have that

$$\bar{X}^{ij} = \begin{pmatrix}\bar{X}_{ii} & \bar{X}_{ij} \\ \bar{X}_{ji} & \bar{X}_{jj}\end{pmatrix} \succeq 0$$

for $(i,j) \in J_E$, and so also $X^{ij} \succeq 0$, since they are equal for $(i,j) \in J_E$. Notice that this means that also the diagonal of X is non-negative. Then this means $X \in \bar{\mathbb{T}}_+^N(E)$.
$\bar{\mathbb{T}}_+^N(E) \subseteq \mathbb{T}_+^N(E,?)$:
Fix $X \in \bar{\mathbb{T}}_+^N(E)$ this time, and take a matrix $\bar{X} \in \hat{\mathbb{T}}_+^N(E,X)$. Then we can show that $\bar{X} \in \mathbb{T}_+^N$. This is obvious for $(i,j) \in J_E$. For $(i,j) \notin J_E$, we have

$$\begin{pmatrix}\bar{X}_{ii} & \bar{X}_{ij} \\ \bar{X}_{ji} & \bar{X}_{jj}\end{pmatrix} = \begin{pmatrix}X_{ii} & \bar{X}_{ij} \\ \bar{X}_{ji} & X_{jj}\end{pmatrix} \succeq 0$$

since $-\sqrt{X_{ii}X_{jj}} \leq \bar{X}_{i,j} \leq \sqrt{X_{ii}X_{jj}}$ □

From Lemma 3.1, we notice that, given $\bar{X} \in \bar{\mathbb{T}}_+^N(E)$, we can complete $\bar{X}$ to a matrix $\hat{X} \in \hat{\mathbb{T}}_+^N(E,\bar{X})$ without changing the elements specified by J_E. Moreover, $\hat{\mathbb{T}}_+^N(E,\bar{X})$ covers all possible completion matrices in $\mathbb{T}_+^N$ for $\bar{X}$.

As a result, we can modify (3.1) to (3.2)

$$\begin{aligned}\min_Y \ & P \cdot Y \\ s.t. \ & Q_k \cdot Y \geq 0, \quad k = 1,...,m \\ & H_0 \cdot Y = 1 \\ & Y \in \bar{\mathbb{T}}_+^N(E)\end{aligned} \tag{3.2}$$

Remark 3.2. We are considering here only quadratic problems. More in general, we would have to deal also with $SOCP$ constraints on shifted moment matrices. Notice that the reasoning we have done so far can be repeated for those matrices as well. Now we will prove that the objective of (3.1) and (3.2) are the same. This only holds for quadratic problems instead.

Let $\bar{X}$ be a solution of (3.2). Let ζ and $\bar{\zeta}$ be optimal values of (3.1) and (3.2) respectively. Since $\mathbb{T}_+^N \subseteq \bar{\mathbb{T}}_+^N(E)$, in general we have $\zeta \geq \bar{\zeta}$, i.e. (3.2) is a relaxation of (3.1).

Theorem 3.3. *We have $\zeta = \bar{\zeta}$. Moreover, if X^* is an optimal solution of (3.1), then it is also an optimal solution for (3.2) and $\hat{\mathbb{T}}_+^N(E, \bar{X})$ is included in the set of solution of (3.1).*

Proof. We show that any $\hat{X} \in \hat{\mathbb{T}}_+^N(E, \bar{X})$ is an optimal solution of (3.1).
Indeed $P \cdot \hat{X} = P \cdot \bar{X} = \bar{\zeta}$, $Q_k \cdot \hat{X} = Q_k \cdot \bar{X}, k = 1, ..., m$ and $H_0 \cdot \hat{X} = H_0 \cdot \bar{X}$. But, since $\hat{X} \in \mathbb{T}_+^N$ (see Lemma 3.1), $\hat{X}$ is a feasible solution of (3.1). Then $\bar{\zeta} \geq \zeta$. But we already had $\zeta \geq \bar{\zeta}$, and then equality holds and $\hat{X}$ is an optimal solution of (3.1).
Finally, since $\mathbb{T}_+^N \subseteq \bar{\mathbb{T}}_+^N(E)$, X^* is also a feasible solution for (3.2), and it is optimal because $\zeta = \bar{\zeta}$. □

The completion method comes from [Fuk+01] and it uses the maximum determinant property to show that the completion matrix is constructed by putting 0 in entries that are not in J_E. We will not show the proof, but we want to point out that this is a way easier method than the one needed for the SOS case, that uses chordal graphs instead and maximal cliques. This is another reason to prefer $SDSOS$ over SOS.

Since we are more used to the dual formulation of (3.1), we end the chapter giving the dual formulation of (3.2), that is the one we implemented on Matlab.

$$
\begin{aligned}
&\max \xi \\
&s.t.\ P + \sum_{k=1}^{m} Q_k y_k - \xi = \sum_{(i,j)\in J_E} W^{ij} \\
&\quad y \in \mathbb{R}_+^m, \xi \in \mathbb{R} \\
&\quad W \in SDSOS_1(N) \text{ and } W_{ij} = 0 \text{ for } (i,j) \notin J_E
\end{aligned}
$$

Notice that we have $W^{i,j} \in SDSOS_1(N)$ since we are considering first order relaxation and thus we have $\sum_{k=1}^{m} Q_k y_k$ with $y \in \mathbb{R}_+^m$ since this is the same

of asking $y_k \in SDSOS_0(N),\ k = 1, ..., m$.

3.3.3 Adding Inequalities

DIGS approach consists in adding inequalities at each iteration in order to better describe the feasible set. The fact that more inequalities can bring a better level of detail was already known and what DIGS does is generating them.
In literature, instead, we can find this method of manually adding valid inequalities to a relaxation to improve its description. Most of the times these inequalities are added to linearized binary problems [BEL08] or MILP [DL13], or they come from analysis of a specific problem.

In [GVA11], for example, they studied valid inequalities for SOCP relaxations of binary programs. Indeed they showed that adding redundant inequalities like

$$(x_i - 1)(x_j - 1) \geq 0$$
$$(x_i - 1)(x_i + 1) \geq 0$$
$$(x_i + 1)(x_j + 1) \geq 0$$

can improve the bound of a SOCP relaxation. Since the Master problem of DIGS-SDSOS algorithm is exactly a SOCP relaxation, we can use this method to improve it whenever we deal with binary programs.
Also, in order to save the sparsity of the problem, we can add them only for the couples $(i, j) \in J_E$.

Now we can test how this algorithm works on some examples, and also improve it using the methods explained above.

Chapter 4

Numerical Results

In this chapter we will at first compare how DIGS and Lasserre Hierarchy work in both SOS and SDSOS case. This will be done for small to medium size instances, due to the dimensionality problems already outlined in the previous chapters.
Then we will start comparing DIGS-SOS and DIGS-SDSOS also for bigger problems, and we will do a detailed analysis on how they work.
We will also implement sparsity for DIGS-SDSOS to see how the algorithm performs in this case. Then we enrich our relaxation with valid inequalities to show how this can affect the algorithm.

All the experiments are implemented in Matlab and solved using SeDuMi [Stu99] as SDP solver and Mosek [ApS19] as SOCP solver. The hardware is a Lenovo Thinkstation P300 with 32 gigabyte of RAM with Intel Xeon CPU 2.40 GHz.

4.1 Continuous Quadratic Problems

The first three problems we try to optimize are taken from [GVA16].

4.1.1 Example 5: a 5 variables instance

The first problem is the following

$$\begin{aligned}
\max_{x\in\mathbb{R}^5} \; & -2x_1 + x_2 - x_3 + 2x_4 + 2x_5 \\
\text{s.t. } & (x_1-2)^2 - x_2^2 - (x_3-1)^2 - (x_5-1)^2 \geq 0 \\
& x_1x_3 - x_4x_5 + x_1^2 \geq 1 \\
& x_3 - x_2^2 - x4^2 \geq 1 \\
& x_1x_5 - x_2x_3 \geq 2 \\
& x_1 + x_2 + x_3 + x_4 + x_5 \leq 14 \\
& x_i \geq 0, \quad i = 1, ..., 5
\end{aligned}$$

It is a rather low dimensional problem but it is very hard to optimize over it due to its structure. Indeed we show here the amount of time needed to optimize over it using Lasserre Hierarchy with SOS and SDSOS.

Relaxation	Degree	Bound	Time
SOS	2	25	0.3
	4	6.006	0.82
	6	2.399	7.09
	8	1.567∗	157.7
SDSOS	2	25	0.12
	4	6.71	0.35
	6	4.55	1.55
	8	2.81	15.0

Table 4.1: Comparing Lasserre Hierarchies with SOS and SDSOS formulations in the 5 variable instance. A $*$ means that it is the optimal value.

We can see in Table 4.1 that we need a degree 8 hierarchy to reach optimality with SOS, even if the problem is only quadratic. We want also to point out how much the time needed to solve the problem increased when we increased the order of the relaxation, and that SOS grows very quickly compared to SDSOS.
The SDSOS hierarchy, instead, was not able to reach the optimal bound with degree 8.

Now let's look at Table 4.2 to see how DIGS performs on the same example.

One can clearly see that DIGS-SOS is very fast to reach the optimal value, even if we are only using order 2 for DIGS. The same does not hold for

Relaxation	Bound	Time	Iterations
SOS	1.568	58.45	59
SDSOS	1.731	4000+	1225

Table 4.2: DIGS results on the 5 variables instance.

SDSOS though. It takes a lot of time to reach a value that is not even optimal.

Let's see if we get the same results using a 10 variables example.

4.1.2 Example 10: a 10 variable instance

The second problem we try to optimize is

$$\begin{aligned}
\max_{x\in\mathbb{R}^{10}} \; & x_1 + x_2 - x_3 + 2x_4 + x_5 - x_6 - x_7 + x_8 - x_9 + 2x_{10} \\
\text{s.t. } & (x_3 - 2)^2 - (x_5 - 1)^2 - 2x_6 + x_8^2 - (x_9 - 2)^2 \geq -4 \\
& - x_2^2 + x_3 x_{10} - x_4^2 - x_5^2 + x_6 x_7 \geq 1 \\
& x_1 x_8 - x_2 x_3 + x_4 x_7 - x_5 x_{10} \geq 2 \\
& \sum_{i=1}^{10} x_i \leq 5 \\
& x_i \geq 0, \quad i = 1, ..., 10
\end{aligned}$$

Once again we start testing Lasserre Hierarchies.

Relaxation	Degree	Bound	Time
SOS	2	10	0.2
	4	7.758	9.9
	6	5.183∗	3613.8
SDSOS	2	10	0.04
	4	7.758	0.80
	6	5.183∗	18.58

Table 4.3: Comparing Lasserre Hierarchies with SOS and SDSOS formulations in the 10 variable instance. A $*$ means that it is the optimal value.

We see that here the order needed to reach optimality is 6 and that it takes a lot of time to compute that using SOS. The SDSOS formulation instead is able to reach the same results in a smaller amount of time. This is a good example that shows why SDSOS popularity is growing.

Let's see if this time DIGS gives good results.

Relaxation	Bound	Time	Iterations
SOS	5.183∗	301.5	30
SDSOS	5.877	4000+	410

Table 4.4: DIGS results on the 10 variables instance. A $*$ means that it is the optimal value.

And once again, looking at 4.4, DIGS-SOS looks very promising, reaching optimality very quickly if compared to Lasserre's method. SDSOS instead still did not converge in the time limit even if it was getting closer.
Let's see if, even in the last problem, the situation is the same.

4.1.3 Example 15: a 15 variables instance

The last of these examples is a 15 variables instance

$$\begin{aligned}
\max_{x\in\mathbb{R}^{15}} \; & -x_1+x_2-x_3+x_4+x_5-x_6-x_7+x_8-x_9+x_{10}-x_{11} \\
& +x_{12}-x_{13}+x_{14}-x_{15} \\
\text{s.t. } & (x_1-2)^2-x_2^2+(x_3-2)^2-(x_4-1)^2-(x_5-1)^2+(x_6-1)^2 \\
& -(x_7-2)^2-x_8^2-(x_9-2)^2-(x_{10}-1)^2+x_{11}^2-x_{12}^2 \\
& +(x_{13}-2)^2+x_{14}^2-(x_{15}-1)^2\geq 0 \\
& -x_1x_7-x_4x_5-x_{13}^2+x_6x_9+x_{10}x_{12}\geq 3 \\
& x_2x_3-x_8x_{11}-x_{14}^2+x_5x_{15}\geq 3 \\
& \sum_{i=1}^{15} x_i \leq 10 \\
& x_i\geq 0, \quad i=1,...,15
\end{aligned}$$

The results are the following:

Relaxation	Degree	Bound	Time
SOS	2	10	0.3
	4	8.059	790.2
SDSOS	2	10	0.02
	4	8.29	3.60

Table 4.5: Comparing Lasserre Hierarchies with SOS and SDSOS formulations in the 15 variable instance.

This time we were not able to compute Lasserre Hierarchies of degree greater than 4 due to memory limitation. We start to see why Lasserre Hierarchies

can not be used for problems of larger scale. Let's see if we can achieve anything better with DIGS.

Relaxation	Bound	Time	Iterations
SOS	10	4000+	4
SDSOS	9.98	4000+	125

Table 4.6: DIGS results on the 15 variables instance.

Unfortunately this time DIGS was not able to outperform Lasserre. This is also because our time limit of 4000 seconds is relatively small, and we know from [GVA16] that given more time DIGS-SOS is able to reach the optimal value for this problem in around 10000 seconds.
Meanwhile we can also notice that DIGS-SDSOS was able to get a better bound than its SOS counterpart. Once again it is a sign of the scalability that SDSOS formulation has.

4.2 Lattice Problem

From the last section, one could think that SDSOS relaxations are not strong enough to accurately solve polynomial problems. With the next example we show why this is not true. Moreover this example will prove the sparse formulation to be essential to be able to solve the problem. The example is taken from [SY19].

Given $n \in \mathbb{N}$, we call a Lattice graph a graph in which the nodes are arranged in a $n \times n$ square and such that there is an edge between nodes i and j if and only if i is the closest node on top, bottom, left or right of j. See Figure 4.1.

In our example, each variable is a node of the graph, and the function and constraints are quadratic. Quadratic terms involving both nodes i and j appear only if there is an edge between i and j.
More formally, we can write it as

$$\begin{aligned} \min_{x \in \mathbb{R}^{n^2}} \ & x^T P_0 x \\ \text{s.t. } & x^T P_k X + r_k \leq 0, i = 1, ..., m \end{aligned}$$

and introduce

$$\begin{aligned} J_E = \{((i-1)n + j, (i-1)n + j + 1) \mid i = 1, ..., n, \ j = 1, ..., n-1\} \\ \cup \{((i-1)n + j, in + j) \mid i = 1, ..., n-1, \ j = 1, ..., n\} \\ \cup \{(i, i) \mid i = 1, ..., n\} \end{aligned}$$

(Notice that the symmetry of matrices let us define J_E as above).

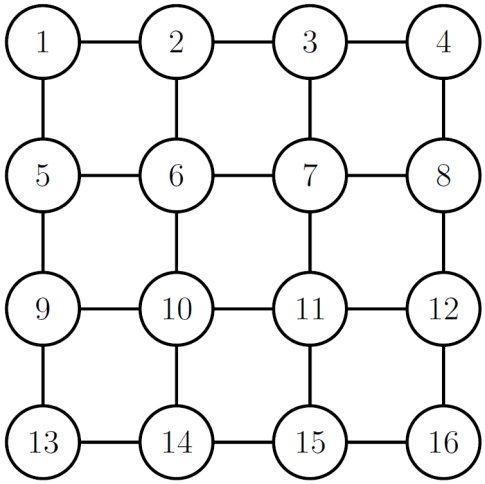

Figure 4.1: A lattice graph for $n = 4$. The image is taken from [SY19]

With the same notation used in Chapter 3, we have $E = 0 \Rightarrow P_l = 0$, $l = 0, ..., m$. For our example we will use:

- for $l = 0, 2, 3, ..., m$, $(P_l)_{ij}$ is a random number in $[-1, 0]$ if $(i, j) \in J_E$, $i \neq j$.
- for $l = 0, 2, 3, ..., m$, $(P_l)_{ii}$ is a random number in $[-1, 1]$
- P_1 is a diagonal matrix with random values in $(0, 1]$

These choices were made to satisfy the hypothesis of Theorem 3.4 and 3.5 of [KK02]. This will grant us that the SDSOS relaxation is able to solve the problem at optimality.

We used n going from 20 to 35, than means the number of variables goes from 400 to 1225. In this setting not only the SOS version, but also the dense one of SDSOS wouldn't be able to solve these problems in reasonable time. Let's see how the SDSOS-sparse one performs.

We can see in Table 4.7 that it takes very little time to solve every instance. From the theory we know that it reaches optimality due to its structure and since it's a quadratic problem, that means that dense and sparse formulations are guaranteed to give the same result. So exploiting sparsity in this case has given an enormous advantage in terms of time without sacrificing accuracy.

n, (variables)	Optimization time (s)
20, (400)	0.82
21, (441)	0.85
22, (484)	0.87
23, (529)	0.89
24, (576)	1.04
25, (625)	0.85
26, (676)	1.12
27, (729)	0.96
28, (784)	1.00
29, (841)	1.05
30, (900)	1.12
31, (961)	1.33
32, (1024)	1.57
33, (1089)	1.30
34, (1156)	1.34
35, (1225)	1.92

Table 4.7: Results on the Lattice problem. We can see that thanks to sparsity, even if the problem has more than 1000 variables, the optimization process still takes little time.

4.3 Quadratic Knapsack

Quadratic Knapsack (QKP) is a problem that models many real-life applications. This is why it is very important to be able to solve it.
Quadratic Knapsack is a generalization of its linear version.
Suppose you have to fill your knapsack with some items: each item has a value but it also occupies space, increases the weight of the knapsack, or anyway has to fit in some constraints. Then your aim is to carry those items that give you the maximum value but satisfy the constraints.

Then, if you have n items, you can introduce variable $x_i,\ i = 1, ..., n$ such that

$$x_i = \begin{cases} 1 & \text{if you carry item } i \\ -1 & \text{otherwise} \end{cases}$$

Then if your profit (the value you get from items) is linear, you have a linear knapsack problem. This means your profit is, given $c = (c_1, ..., c_n)$ vector of values, $c^T x$. We will focus on linear constraints, i.e. constraints of the form

$w^T x \leq M$. Then the full problem with only one constraint will be:

$$\begin{aligned} \max_x \ & c^T x \\ \text{s.t. } & w^T x \leq M \\ & x_i \in \{-1, 1\} \end{aligned}$$

The same way, if your profit is quadratic in the items you choose to carry, your profit will be of the form

$$\begin{pmatrix} 1 & x^T \end{pmatrix} Q \begin{pmatrix} 1 \\ x \end{pmatrix}$$

and then the full problem will look like this

$$\begin{aligned} \max_x \ & \begin{pmatrix} 1 & x^T \end{pmatrix} Q \begin{pmatrix} 1 \\ x \end{pmatrix} \\ \text{s.t. } & w^T x \leq M \\ & x_i \in \{-1, 1\} \end{aligned}$$

Then this is a quadratic problem with binary variables. While on the linear case one needs to find a way to deal with binary constraints using methods like Branch and Bound or Gomory cuts, here we can write those constraints as $(x_i - 1)(x_i + 1) = 0,\ i = 1, ..., n$. These are still polynomial constraints. We tested our algorithm also on instances of this problem.

Since this kind of problem is usually very sparse, this time we compared the dense algorithms with the sparse one. Compared to the previous 3 examples, QKP is usually easier to solve.
We created our problems using a generator that takes in input the amount of variables and the density of the problem and gives as output a matrix Q with specified density and a vector of weights w. So we were able to test our algorithm on problem of the desired size and sparsity.

We list in Table 4.8 bounds and time for every instance we tested the problem on.

Test with valid inequalities

The same sparse formulation was then enriched with the inequalities discussed in Chapter 3. We tested in on the same instances. The results are once again listen in Table 4.8.
To show why we choose to focus our attention on DIGS-SDSOS, we also tested only the master problem (a Lassere Hierarchy of degree 2) on QKP instances of 10 to 150 variables and density 70.

From Table 4.9 we can notice that the time needed for optimization in the SOCP case does not even reach 1 second on the biggest instance, while SOS time is more than 10000 seconds. The fact that SDSOS is so fast allows us to use valid inequalities to improve the bound without affecting much the time needed for optimization. By doing that, we have relaxations so strong that they give stronger bounds than the SOS ones.

Form Table 4.8, instead, we can clearly see that, once again dense SDSOS can not compete with SOS, but its sparse implementation is very fast. Sometimes it is able to converge to the same bound of SOS in less time (especially when the dimensions are big and SOS can not iterate). And once again, when improved using valid inequalities, even if the time needed to construct the subproblem increases slightly, the better starting bound gives us convergence in fewer iterations to the same optimal of the sparse formulation.
We can see that this does not happen in Figure 4.3. This is due the fact that we are taking in consideration also the time needed to create the subproblem, that was not optimized for the sparse case. If one would only look at the time needed for optimization, it is clear that the sparse formulation with valid inequalities is usually preferable. For example, in Figure 4.5, we can also notice that the starting bound given by the Sparse+ formulation is already better than the final bound of the Sparse one. Finally, in Figure 4.4, we see that Sparse and Sparse+ reach the same objective. But a problem of DIGS-SDSOS is the fact that its first iteration usually gives a very weak bound and then it needs many iterations to improve from it. Sparse+ formulation instead starts already with a very good bound and it needs few iteration to converge.

4.4 Considerations on the results

In general we have seen that DIGS-SDSOS iterations are faster than DIGS-SOS one, but also weaker. Usually we see that the SOS version converges in less time than the SDSOS anyway, especially for small-size problem. We have also seen that this does not always hold true, for example for the instance with 15 variables, when the problem is so big that each iteration for the SOS is very expensive.
The Lattice example showed us that the performances of SDSOS relaxations highly depends on the structure of the problem, and how sparsity can improve performances and let us solve very large problems.
From the QKP instances we can clearly see that the exploitation of sparsity works very well, especially when the problem is very sparse. The fact that we want to keep that sparsity also in the new constraints that we generate is also making use of more information that can lead to stronger inequality. So, for problems like QKP, the sparse formulation is outperforming both

SOS and SDSOS.
Finally, when adding valid inequalities, we got convergence to the same optimal value but usually in a lesser time.
This is an example of how DIGS-SDSOS can give a good balance between time, scalability and bound accuracy. And also that this algorithm can be easily adapted to specific problems.

A weak point right now is that, after the first iterations, the inequalities generated by the subproblem do not give good improvements on the bound, and the convergence rate slows down. This causes the algorithm to keep running for many iterations without really improving its bound. The best example for this behavior is Figure 4.6.
This weakness might also be the reason why we almost never reached optimality when using SDSOS.

4.5 Tables and Figures

n	d	Bound				Time				Iterations				Construction Time			
		SOS	SDSOS	SPARSE	SPARSE+	SOS	SDSOS	SPARSE	SPARSE+	SOS	SDSOS	SPARSE	SPARSE+	SOS	SDSOS	SPARSE	SPARSE+
10	20	625.05	610.33	620.78	620.78	1.85	3427.65	7.05	6.17	0	395	12	5	0.98	4.00	5.77	5.03
	30	806.22	766.00	766.00	766.00	1.04	430.66	4.42	6.41	0	138	21	4	0.97	4.12	2.02	5.34
	70	1614.23	1452.88	1510.86	1503.90	1.17	4000+	29.41	20.42	0	446	25	11	1.11	3.62	2.70	11.81
20	20	1648.87	1649.01	1647.00	1647.00	1.30	4000+	18.02	47.86	0	30	7	4	1.16	95.78	16.22	45.63
	30	2273.83	2271.72	2255.41	2246.42	1.78	4000+	29.18	81.33	0	30	37	12	1.60	99.02	16.87	66.64
	70	6662.97	6636.76	6529.06	6524.81	2.58	4000+	303.46	394.86	0	38	224	17	2.43	94.30	30.56	262.34
30	20	4810.89	4933.71	4777.38	4777.37	3.07	4000+	1637.66	747.63	0	7	636	19	2.45	1622.39	121.57	647.06
	30	6916.26	7131.40	6770.05	6769.36	3.30	4000+	1843.36	1374.97	0	8	720	39	2.86	1647.98	143.04	1023.31
	70	14164.65	14299.42	14051.78	14072.25	5.78	4000+	4000+	4000+	0	8	500	3	5.17	1644.56	380.91	3872.92
40	20	1078.70	2131.47	844.22	830.26	4.76	0.55	4000+	4000+	0	0	727	24	3.06	0.53	612.13	3562.63
	30	1550.98	3066.29	1259.47	1209.86	6.71	0.55	4000+	0.56	0	0	622	0	4.56	0.53	657.23	0.53
	70	3769.90	7708.50	4080.94	2627.47	11.44	0.55	4000+	0.587	0	0	72	0	8.79	0.53	3018.74	0.53

Table 4.8: Table comparing DIGS algorithm in the SOS, SDSOS, sparse-SDSOS and sparse with valid inequalities for QKP programs of different sizes and sparsity. Here Time refers to the sum of Construction time and the time needed for optimization. We can see that, except for small instances, where dense SDSOS gives the best bound, Sparse+ is the one with the best results also in terms of time. We notice that the time needed to construct the problem in the Sparse+ case can be significantly bigger than the sparse one, but this is compensated by better bounds and faster convergence rate.

		$n=10$	$n=20$	$n=30$	$n=40$	$n=50$	$n=60$	$n=70$	$n=80$	$n=90$	$n=100$	$n=110$	$n=120$	$n=130$	$n=140$	$n=150$
SOS	Bound	1614.23	6662.97	14164.65	3769.90	16488.84	49909.67	19789.48	63711.41	132869.32	149549.40	117689.73	152594.59	244023.2256	301807.86	306923.30
	Time	0.087	0.170	0.442	1.970	5.81	15.145	73.605	201.794	503.149	1017.957	1351.947	3507.135	5223.860	7870.034	11370.926
SDSOS	Bound	1742.08	6929.21	14890.91	7708.5	23229.3	55811	34831.31	78927.69	137583.22	161291.64	147286.24	185913.38	262317.07	317227.86	339699.86
	Time	0.015	0.016	0.019	0.028	0.029	0.039	0.049	0.059	0.078	0.083	0.103	0.116	0.151	0.154	0.184
Sparse	Bound	1742.08	6929.21	14890.91	7708.5	23229.3	55811	34831.31	78927.69	137583.22	161291.64	147286.24	185913.38	262317.08	317227.86	339699.86
	Time	0.014	0.016	0.021	0.027	0.030	0.039	0.048	0.057	0.073	0.081	0.101	0.113	0.142	0.149	0.179
Sparse+	Bound	1557.68	6560.48	14120.68	2627.47	13989.81	47737.69	15456.73	55995.87	130983.53	145084.28	103284.05	137086.68	233321.98	292719.07	290106.06
	Time	0.013	0.019	0.036	0.057	0.063	0.100	0.107	0.148	0.265	0.299	0.269	0.361	0.397	0.619	0.744

Table 4.9: Table comparing the first iteration for the master problem of SOS, SDSOS and sparse-SDSOS case for QKP programs of different sizes and sparsity 70. The time reported is only the time needed for optimization, since the time for constructing the problems for degree 2 Lasserre Hierarchies is similar for all the relaxations.
Notice how quickly the time needed to SOS increases with bigger instances, while the optimization time for SOCP formulations always stays below 1 second. Moreover, the bound given by Sparse+, is always the best one.

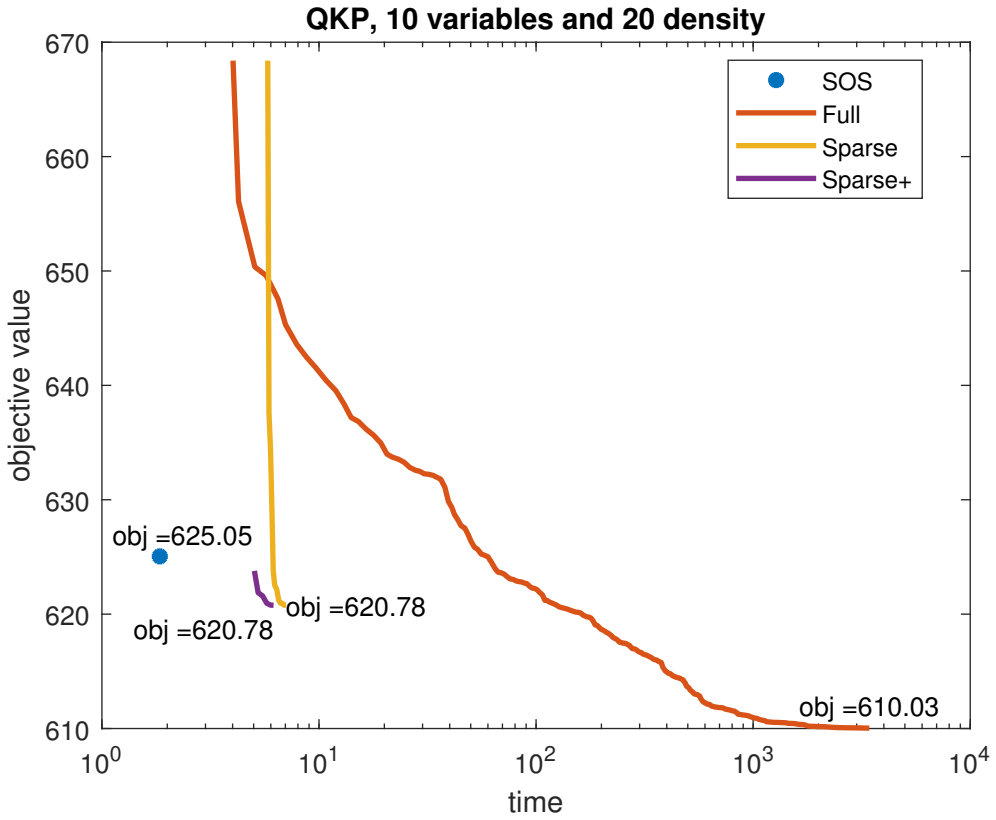

Figure 4.2: A quadratic knapsack with 10 variables and high sparsity. Because of the limited size, we can see that the dense formulation reaches lower bounds than the sparse one. We also notice that Sparse+ starts from an already good bound (better than the SOS one) and reaches the final bound in few iterations.

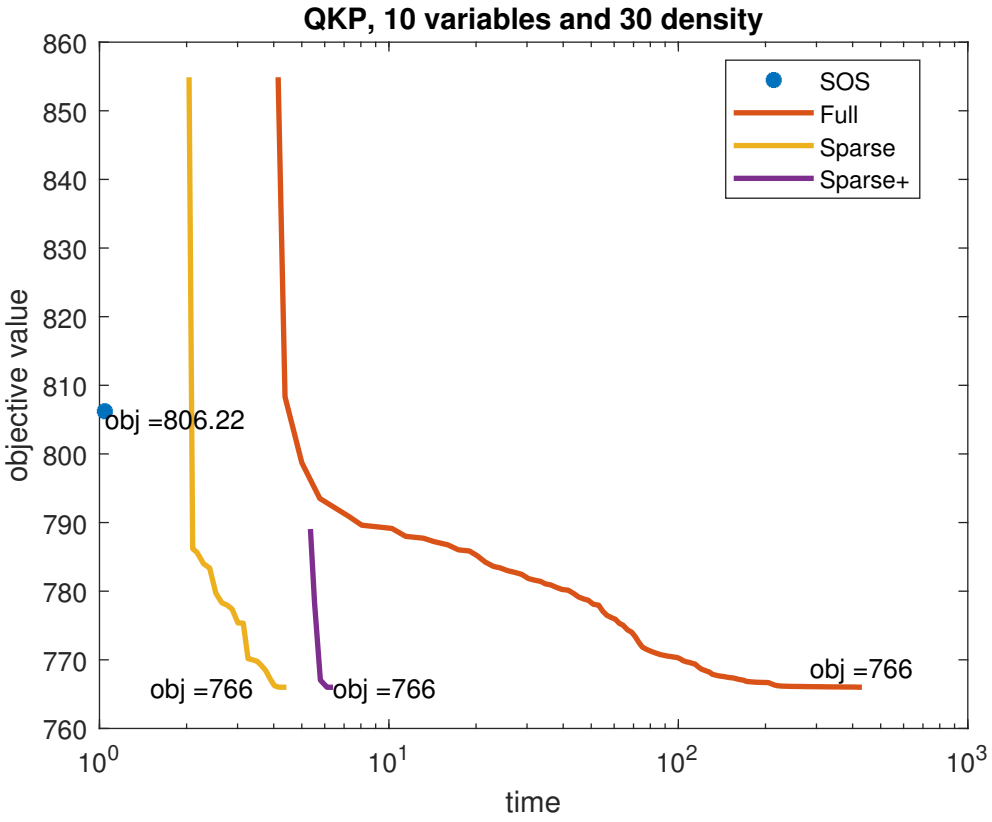

Figure 4.3: An example of QKP in which both the sparse and dense formulation reach the same objective. It is clear how much time the sparse formulation saves. Here the time needed to create the problem in the Sparse+ case is higher than the time needed to the Sparse formulation to reach convergence. This is not usual, and it is probably because the dimensionality is low.

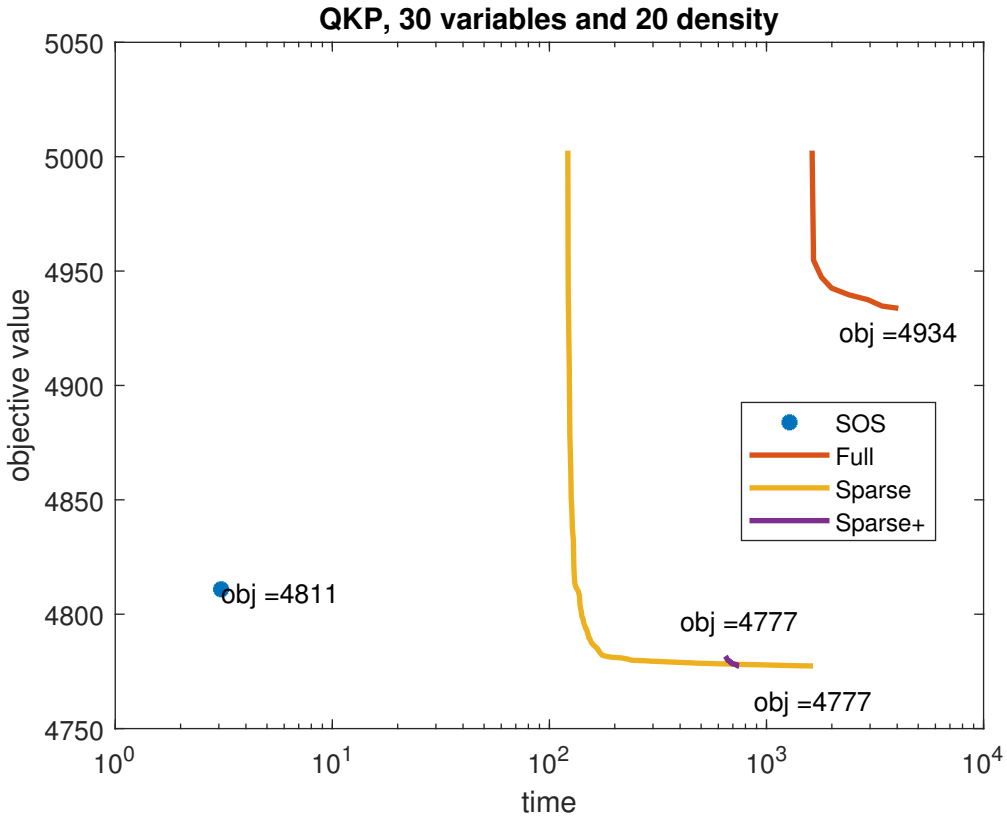

Figure 4.4: Increasing the size makes the sparse formulation even more effective. It takes less time to reach a better bound. Also notice that most of the time needed to Sparse+ is construction time, while the convergence happens very quickly.

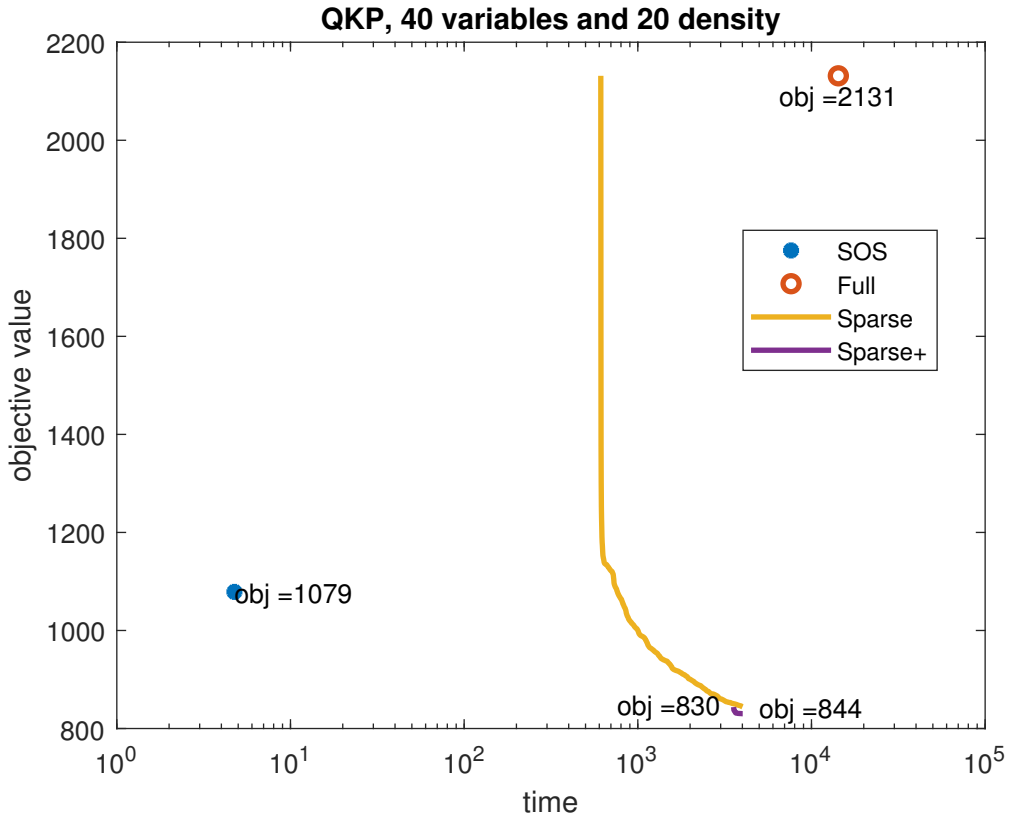

Figure 4.5: Things are even better for the Sparse formulations when the size of the problem is so big that the dense one can not compute the subproblem in the time limit anymore. Once again Sparse+ needs more time to be built but provides better starting bounds. Here it is already better than what the Sparse formulation can reach in the time limit.

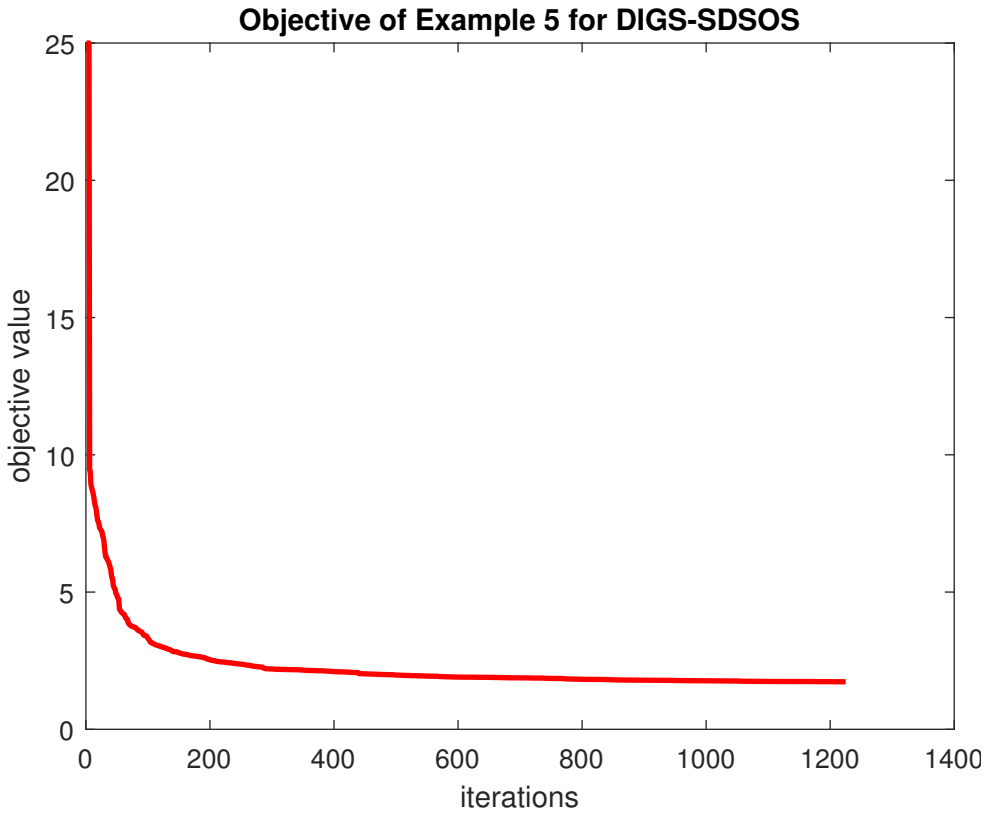

Figure 4.6: This plot shows that inequalities generated by DIGS-SDSOS can be very weak and lead to a slow convergence rate. In this case the final bound is not much better than the objective reached after 400 iterations.

Chapter 5

Conclusions

In this thesis we implemented a new version of the DIGS algorithm for solving polynomial programs. This version made use of the SOC relaxation of Lasserre Hierarchies.
We showed that for some problems it is not able to give good bounds or that needs a lot of iteration to reach convergence due to the weakness of SDSOS compared to SOS polynomials. But we were also able to show that with other problems, for example QKP, the performances are good and we get convergence in reasonable time, sometimes to the same bound we can get with SOS.
Performances can improve drastically when the sparse formulation is taken into account. This formulation is also very easy to implement compared to the SOS case, and this is a great advantage of the SOC formulation.
We also discussed about the possibility of strenghtening the formulation adding special inequalities for some specific problems. This is more convenient when using the SDSOS formulation, since the dimension of the problem increases more slowly than in the SOS case.

In general we have shown the advantages of implementing the SDSOS version of the algorithm, also due to the increasing attention that SOCP is gaining recently and the fact that most optimization software nowadays have a SOC optimizer. On the other hand, Semidefinite Programming fortune is decreasing because of the drawback of its complexity increasing very fast with the dimensionality of a program.

Further studies can focus on how to create stronger inequalities at each iterations, for example using other norms or different convergence conditions. Moreover we focused on a general framework, but we have seen that DIGS can easily adapt to specific problems. This was shown to improve its performances. In the same way, a future direction could be how to adapt the inequality generation scheme to specific problems.

Bibliography

[ADH15] Amir Ahmadi, Sanjeeb Dash, and Georgina Hall. "Optimization over Structured Subsets of Positive Semidefinite Matrices via Column Generation". In: *Discrete Optimization* 24 (Dec. 2015). DOI: 10.1016/j.disopt.2016.04.004.

[AH15] Amir Ali Ahmadi and Georgina Hall. *Sum of Squares Basis Pursuit with Linear and Second Order Cone Programming*. 2015. arXiv: 1510.01597 [math.OC].

[AM17a] Amir Ali Ahmadi and Anirudha Majumdar. *DSOS and SDSOS Optimization: More Tractable Alternatives to Sum of Squares and Semidefinite Optimization*. 2017. arXiv: 1706.02586 [math.OC].

[AM17b] Amir Ali Ahmadi and Anirudha Majumdar. *Response to "Counterexample to global convergence of DSOS and SDSOS hierarchies"*. 2017. arXiv: 1710.02901 [math.OC].

[ApS19] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* 2019. URL: http://docs.mosek.com/9.0/toolbox/index.html.

[BEL08] Alain Billionnet, Sourour Elloumi, and Amélie Lambert. "Linear Reformulations of Integer Quadratic Programs". In: vol. 14. Sept. 2008, pp. 43–51. DOI: 10.1007/978-3-540-87477-5_5.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787.

[DL13] Hongbo Dong and Jeff Linderoth. "On Valid Inequalities for Quadratic Programming with Continuous Variables and Binary Indicators". In: *Integer Programming and Combinatorial Optimization*. Ed. by Michel Goemans and José Correa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 169–180.

[Fuk+01] Mituhiro. Fukuda et al. "Exploiting Sparsity in Semidefinite Programming via Matrix Completion I: General Framework". In: *SIAM Journal on Optimization* 11.3 (2001), pp. 647–674. DOI: 10.1137/S1052623400366218. eprint: https://doi.org/10.1137/S1052623400366218. URL: https://doi.org/10.1137/S1052623400366218.

[Gha11] Bissan Ghaddar. "New Conic Optimization Techniques for Solving Binary Polynomial Programming Problems". PhD thesis. University of Waterloo, 2011.

[GVA11] Bissan Ghaddar, Juan Vera, and Miguel Anjos. "Second-Order Cone Relaxations for Binary Quadratic Polynomial Programs". In: *SIAM Journal on Optimization* 21 (Jan. 2011), pp. 391–414. DOI: 10.1137/100802190.

[GVA16] Bissan Ghaddar, Juan C. Vera, and Miguel F. Anjos. "A dynamic inequality generation scheme for polynomial programming". In: *Mathematical Programming* 156 (2016), pp. 21–57.

[Hil88] David Hilbert. "Ueber die Darstellung definiter Formen als Summe von Formenquadraten". In: *Mathematische Annalen* 32.3 (Sept. 1888), pp. 342–350. ISSN: 1432-1807. DOI: 10.1007/BF01443605. URL: https://doi.org/10.1007/BF01443605.

[KK01] Sunyonga Kim and Masakazu Kojima. "Second order cone programming relaxation of nonconvex quadratic optimization problems". In: *Optimization Methods and Software* 15.3-4 (2001), pp. 201–224. DOI: 10.1080/10556780108805819. eprint: https://doi.org/10.1080/10556780108805819. URL: https://doi.org/10.1080/10556780108805819.

[KK02] Sunyoung Kim and Masakazu Kojima. "Exact Solutions of Some Nonconvex Quadratic Optimization Problems via SDP and SOCP Relaxations". In: *Computational Optimization and Applications* 26 (Feb. 2002). DOI: 10.1023/A:1025794313696.

[Las01] Jean B. Lasserre. "Global Optimization with Polynomials and the Problem of Moments". In: *SIAM Journal on Optimization* 11.3 (2001), pp. 796–817. DOI: 10.1137/S1052623400366802. eprint: https://doi.org/10.1137/S1052623400366802. URL: https://doi.org/10.1137/S1052623400366802.

[Las15] Jean Bernard Lasserre. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2015. DOI: 10.1017/CBO9781107447226.

[Mot67] Theodore Samuel Motzkin. "The arithmetic-geometric inequality". In: *O. Shisha, Inequalities* (1967), pp. 205–224.

[NN94] Yurii. Nesterov and Arkadii. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994. DOI: 10.1137/1.9781611970791. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970791. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970791.

[NT98] Yu Nesterov and Michael Todd. "Primal-Dual Interior-Point Methods for Self-Scaled Cones". In: *SIAM Journal on Optimization* 8 (Feb. 1998), pp. 324–364. DOI: 10.1137/S1052623495290209.

[Pò74] Georg Pòlya. "Uber positive darstellung von polynomen vierteljschr". In: *Natur-forsch. Ges. Zurich, 73: 141–145, 1928, in: R.P. Boas (Ed.), Collected Papers* 2 (1974), pp. 309–313.

[Par00] Pablo Parrilo. "Structured Semidenite Programs and Semialgebraic Geometry Methods in Robustness and Optimization". In: *PhD thesis* (Aug. 2000).

[PS01] Pablo A. Parrilo and Bernd Sturmfels. *Minimizing Polynomial Functions*. 2001. arXiv: `math/0103170 [math.OC]`.

[Put93] Mihai Putinar. "Positive Polynomials on Compact Semi-algebraic Sets". In: *Indiana University Mathematics Journal* 42.3 (1993), pp. 969–984. ISSN: 00222518, 19435258. URL: `http://www.jstor.org/stable/24897130`.

[PVZ08] Javier Pena, Juan Vera, and Luis Zuluaga. "Exploiting equalities in polynomial programming". In: *Operations Research Letters* 36 (Mar. 2008), pp. 223–228. DOI: `10.1016/j.orl.2007.05.011`.

[Sch91] Konrad Schmudgen. "The K-moment problem for compact semi-algebraic sets." In: *Mathematische Annalen* 289.2 (1991), pp. 203–206. URL: `http://eudml.org/doc/164777`.

[Sho87] Naum Z. Shor. "Class of global minimum bounds of polynomial functions". In: *Cybernetics* 23 (1987), pp. 731–734.

[Ste74] Gilbert Stengle. "A Nullstellensatz and a Positivstellensatz in Semialgebraic Geometry." In: *Mathematische Annalen* 207 (1974), pp. 87–98. URL: `http://eudml.org/doc/162533`.

[Stu99] Jos F. Sturm. "Using SeDuMi 1.02, A Matlab toolbox for optimization over symmetric cones". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 625–653. DOI: `10.1080/10556789908805766`. eprint: `https://doi.org/10.1080/10556789908805766`. URL: `https://doi.org/10.1080/10556789908805766`.

[SY19] Heejune Sheen and Makoto Yamashita. *Exploiting Aggregate Sparsity in Second Order Cone Relaxations for Quadratic Constrained Quadratic Programming Problems*. 2019. arXiv: `1911.02188 [math.OC]`.

[Zhe+] Yang Zheng et al. "Chordal decomposition in operator-splitting methods for sparse semidefinite programs". In: (). arXiv: `1707.05058 [math-OC]`.

[Zhe+16] Yang Zheng et al. *CDCS: Cone Decomposition Conic Solver, version 1.1*. `https://github.com/giofantuzzi/CDCS`. Sept. 2016.