Czech Technical University in Prague
Faculty of Electrical Engineering

DSOS and SDSOS Optimization for Strategic Games

Master's thesis of
Tomáš Votroubek

Supervised by
doc. Ing. Tomáš Kroupa, Ph.D.

August 21, 2020

Abstract

We apply Scaled-Diagonally-Dominant-Sum-of-Squares (SDSOS) optimization to the problem of two-player zero-sum polynomial games. Our baseline is a version of a Sum-of-Squares (SOS) Program by Parrilo extended to solve polynomial games over semialgebraic sets. SDSOS optimization (via a change-of-basis method) promises to alleviate the large problem sizes inherent to SOS optimization without a significant loss of accuracy. Unfortunately, the results are overly conservative to be useful for our problem. We show the results on examples extracted from the literature, and on games with prescribed solutions generated using an equation by Gale and Gross. In addition, we extend the Double Oracle algorithm to solve semialgebraic games and demonstrate its speed of convergence. All our code is available online on Github and Gitlab. We explain all of the fundamental parts of the source code in the appendix of this thesis.

Abstrakt

Aplikujeme optimalizaci Scaled-Diagonally-Dominant-Sum-of-Squares (SDSOS) k řešení polynomiálních her s nulovým součtem o dvou hráčích. Náš algoritmus vychází z programu Sum-of-Squares (SOS) formulovaného Parrilem, který jsme rozšířili pro řešení her se semialgebraickými prostory strategií. I přestože je optimalizace SDSOS schopna řešit větší problémy než optimalizace SOS, pro náš problém jsou její výsledky příliš konzervativní. Problém předvedeme na příkladech z literatury, a také na hrách se zadaným ekvilibriem vytvořených pomocí Galovi a Grossovi rovnice. Představíme také rozšíření algoritmu Double Oracle pro řešení semialgebraických her a demonstrujeme rychlost jeho konvergence. Veškerý zdrojový kód je dostupný na Github a Gitlab. Důležité úryvky kódu vysvětlíme na konci práce.

Acknowledgments

I would like to extend my sincere thanks to doc. Ing. Tomáš Kroupa, Ph.D. for supervising my thesis through the difficult times of COVID-19.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Datum	Tomáš Votroubek

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Votroubek Tomáš** Personal ID number: **440927**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

DSOS and SDSOS Optimization for Strategic Games

Master's thesis title in Czech:

DSOS a SDSOS optimalizace pro strategické hry

Guidelines:

1. Familiarize yourself with the computational procedures for strategic games [3] and polynomial games [1] based on linear programming and semidefinite programming (SDP), respectively.
2. Diagonally dominant sum of squares (DSOS) and scaled diagonally dominant sum of squares (SDSOS) optimization have been recently introduced in [2] as more tractable alternatives to SDP. Identify the classes of polynomial games amenable to the application of DSOS/SDSOS methods.
3. Develop a solver for polynomial games implementing DSOS and SDSOS optimization using YALMIP (a MATLAB toolbox for optimization modeling). Compare its effectivity with the semidefinite procedure formulated by P. Parillo [1].

Bibliography / sources:

[1] P. Parrilo. Polynomial games and sum of squares optimization. In Decision and Control, 2006 45th IEEE Conference on, pages 2855–2860, 2006.
[2] A. A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization. SIAM Journal on Applied Algebra and Geometry, 3(2):193–230, 2019.
[3] Y. Shoham and K. Leyton-Brown. Multiagent systems: Algorithmic, game-theoretic, and logical founda- tions. Cambridge University Press, 2008.

Name and workplace of master's thesis supervisor:

doc. Ing. Tomáš Kroupa, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **21.02.2020** Deadline for master's thesis submission: **14.08.2020**

Assignment valid until: **19.02.2022**

doc. Ing. Tomáš Kroupa, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Contents

Chapter 1

Introduction

In 2006, Parrilo presented a considerable contribution to the two-person zero-sum games with infinite sets of strategies. In his paper (Parrilo, 2007), he illustrates a constructive method for computing the optimal polynomial games' optimal strategies by solving a single semidefinite programming problem.

While his paper deals with one-dimensional strategy spaces only, the underlying techniques are based on a much more general framework of sum-of-squares optimization and extend naturally to *semialgebraic games*, that is, games with a polynomial payoff, and basic semialgebraic strategy sets. Unfortunately, while his original technique is exact, the semialgebraic extension is not. The problem of determining the value of a Semialgebraic game is NP-hard since its fundamental problems of determining polynomial nonnegativity and the recognition of valid moment sequences are hard problems. Nonetheless, both the optimal value and the moment problem can still be approximated in polynomial time by a hierarchy of semidefinite relaxations, in the spirit of the moment approach developed in Lasserre (2004). Furthermore, techniques based on sums of squares often need only a few relaxations for good approximations (and possibly finite convergence), and they might, in some sense, even be optimal.

Even then, Sum-Of-Squares (SOS) optimization itself is dependent on semidefinite programming, which A. Ahmadi and Majumdar classify as its weakness. Compared to linear or second-order-cone programs, Semidefinite Programming (SDP) is relatively expensive, and SDP solvers are slow. This is further amplified by the fact that sum-of-squares problems are large by default — a problem undeniably shared by the quickly growing Lasserre hierarchies. As a result, scalability is a significant challenge for sum-of-squares optimization. A. Ahmadi and Majumdar (2017) propose a more tractable alternative in their recent work. They suggest techniques based on diagonally-dominant matrices as a way to inner-approximate a general semidefinite program by a sequence of linear programs. Their approach has already been successfully applied to several areas. In this thesis, we explore how it can be used to solve semialgebraic games.

Chapter 2

Background on Two-Player Zero-Sum Games

Two-Player Zero-sum games are games where one player wins what the other loses. They are defined by a strategy space $\mathcal{X}$ for player 1, a strategy space $\mathcal{Y}$ for player 2, and a single function which determines the payoff of player 1 (and accordingly, the loss of player 2). The strategy spaces are classically expected to be convex, but as we will see later, this limitation can be overcome.

2.1 Finite games

A matrix game is a two-person zero-sum game with finite strategy sets for both players. Without loss of generality, let $X = \{1, \ldots, n\}$ be the strategy set of player 1, and $Y = \{1, \ldots, m\}$ the strategy set of player 2. Any matrix $A \in R^{n \times m}$ is called a payoff matrix and its entry a_{ij} is the payoff of player 1 (the loss incurred by player 2) when the strategy profile (i, j) is selected. The elements of the sets X and Y are called *pure* strategies.

A mixed strategy of player 1 is a probability distribution $\mu = [\mu_1, \ldots, \mu_n] \in \Delta^n$, where Δ^n is a probability simplex[1]. An analogous definition applies to player 2. When players use the mixed strategy profile (μ, ν), the expected payoff to player 1 is

$$p(\mu, \nu) = \sum_{i=1}^{n} \sum_{j=1}^{m} \mu_i a_{ij} \nu_j \quad \text{or} \quad \mu A \nu^\intercal,$$

for the payoff matrix

$$A = \begin{array}{c} \\ 1 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccc} 1 & \cdots & m \end{array} \\ \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \end{array}$$

A trivial example of a game resulting in mixed strategies is rock-paper-scissors, where the optimal strategy is to pick all options randomly.

[1] The standard simplex in $\mathbb{R}^n$ is $\Delta^n = \{x \in \mathbb{R}^n_+ \mid \sum_{i=1}^n x_i = 1\}$

Optimal strategies

Every finite game has a point in $\Delta^n \times \Delta^m$ called a *Nash equilibrium*, that is, a choice of concrete strategies of all players, such that no player can improve their payoff by deviating from it. Remember that the expected payoff of player 1 is given by $p(\mu, \nu) = \mu A \nu^\intercal$ and player 1 wants to maximize it. Equivalently, since one players win is another players loss, player 2 wants to minimize it. Then $(\mu^\star, \nu^\star)$ is a Nash equilibrium if it is a saddle point in the sense

$$\max_\mu p(\mu, \nu^\intercal) = p(\mu^\intercal, \nu^\intercal) = \min_\nu p(\mu^\intercal, \nu),$$

which we can rewrite specifically for matrix games as

$$\max_\mu \min_\nu \ \mu A \nu^\intercal = \mu^\star A \nu^{\star\intercal} = \min_\nu \max_\mu \ \mu A \nu^\intercal.$$

This follows directly from the *Minimax theorem*. First, note for any function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ the ineqality

$$\sup_{x \in \mathcal{X}} \inf_{y \in \mathcal{Y}} f(x, y) \leq \inf_{y \in \mathcal{Y}} \sup_{x \in \mathcal{X}} f(x, y)$$

always holds. The *Minimax theorem* classifies the situations, in which the relation is an *equality*.

Theorem 1 (Minimax Theorem (Kjeldsen, 2001))**.** *Let $\mathcal{X} \subset \mathbb{R}^m$ and $\mathcal{Y} \subset \mathbb{R}^n$ be convex and compact sets, and f a continuous function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, which is quasiconcave in x and quasiconvex in y, then:*

$$\max_{x \in \mathcal{X}} \min_{y \in \mathcal{Y}} f(x, y) = \min_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} f(x, y)$$

In our case, the sets $\mathcal{X}$ and $\mathcal{Y}$ are standard simplices, which are convex and compact, and the function p is a bilinear form.

Finding optimal strategies using linear programming

We can find a Nash equilibrium using linear programming. First, assume that player 2 is forced to play first and reveal his strategy y. player 1 can obviously do no better, than to respond with a strategy with the maximal payoff, and has no incentive to randomize, even if there are multiple strategies with the same payoff. This means that

$$\max_x \ x^\intercal A y = \max_i \ e_i^\intercal A y = \max_i \ A[i, :] \cdot y,$$

where $A[i, :]$ is the i-th row of A, and e_i is a standard basis vector. With this, we can rewrite the saddle point condition as

$$\max_x \min_j \ x \cdot A[:, j] = x^{\star\intercal} A y^\star = \min_y \max_i \ A[i, :] \cdot y.$$

The left and the right sides if the equation can be written as linear programs. Taking first the left side $\max_x \min_j \ x \cdot A[:, j]$, we formulate it as the linear program

$$\begin{aligned} \underset{\gamma, x}{\text{maximize}} \quad & \gamma \\ \text{such that} \quad & \gamma - x \cdot A[:, j] \leq 0 \quad \forall j \in 1, \ldots, m \\ & \textstyle\sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0 \quad \forall i \in 1, \ldots, n \end{aligned} \tag{2.1}$$

where γ is the value of the game.

Instead of formulating the other side of the equation, we can dualize this one. The first line of m "nonpositivity" constraints generates m nonnegative dual variables $y \in \mathbb{R}^n_+$. The one equality constraint generates one free variable $\lambda \in \mathbb{R}$. The one free primal variable in the objective 1γ creates the constraint $= 1$. Finally, the n x_i's become:

$$\lambda + \sum_{j=1}^{n} -A[i, j] y_i \geq 0 \quad \forall i \in 1, \ldots, n,$$

or $\lambda - A[i, :] \cdot y \geq 0$, giving the dual formulation

$$\begin{aligned} \underset{\lambda, y}{\text{minimize}} \quad & \lambda \\ \text{such that} \quad & \lambda - A[i, :] \cdot y \geq 0 \quad \forall i \in 1, \ldots, n \\ & \textstyle\sum_{j=1}^{m} y_j = 1 \\ & y_j \geq 0 \quad \forall j \in 1, \ldots, m \end{aligned} \tag{2.2}$$

which exactly corresponds to the right hand side $\min_y \max_i \ A[i, :] \cdot y$. Source code implementing this formulation is in the appendix of this thesis.

2.2 Infinite games

A natural extension to finite games are games with infinite-dimensional strategy spaces. Similarly to finite games, infinite games are defined by a strategy space $\mathcal{X}$ for player 1, a strategy space $\mathcal{Y}$ for player 2, and a single payoff function. Likewise, the concept of a solution, or optimal strategies, is generally given by the Nash equilibrium. For the same reason, the equation

$$\max_x \min_y f(x, y) = \min_y \max_x f(x, y)$$

is expected to hold. At this point, if the strategy spaces were convex and compact, and $f(x, y)$ were quasiconcave in x and quasiconvex in y, then we could simply apply the minimax theorem and this would not be a problem. In general, we do not have such guarantees.

Assume for a moment, that the condition holds. We can then define the infinite analogs to the payoffs and strategies. Consider a game given by sets X, Y, and a continuous function f. Then let $\mathcal{X}$ and $\mathcal{Y}$ to be sets of all probability measures supported on X and Y respectively. Given a $\mu \in \mathcal{X}$ and $\nu \in \mathcal{Y}$,

$$E_{\mu,\nu}[f] = \int_X \int_Y f \, d\mu \, d\nu$$

is the expected payoff, provided that the payoff function f is measurable. The elements of X and Y can be interpreted as sets of Dirac measures δ_x, centered on x. The measures in X are analogous to the pure strategies of finite games, and the measures in $\mathcal{X}$ are the mixed analogues.

Polynomial games

A polynomial game is a separable, continuous game, whose payoff function is defined as

$$p(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x^i y^j.$$

In a simplified setting, polynomial games are played on a unit square. In that case, there exists an exact numerical approach to solving them. In a more general sense, polynomial games can be multi-dimensional, and their strategy spaces can be solution sets of polynomial equations and inequalities, that is, semialgebraic sets. Due to their polynomial nature (and their separability), polynomial games have some *nice* properties, which make them easier to analyze and solve compared to other types of continuous games.

However, even our basic requirement of

$$\max_x \min_y f(x, y) = \min_y \max_x f(x, y)$$

does not have an obvious answer. The requirements for the applicability of the Minimax theorem were: the strategy spaces must be convex and compact, and $f(x, y)$ must be quasiconcave in x and quasiconvex in y. We will satisfy the requirements on sets by projecting them into higher dimensional sets. At the same time the requirement for f's convexity will be satisfied by exploiting its separability.

Chapter 3

Computation of Equilibria via SOS Optimization

The goal in a two-player polynomial game is to maximize the value gained from the other player. As the outcome ultimately depends on the actions of all players, this corresponds to finding strategies whose worst-case is as favourable as possible. Each player is doing the same, and due to the *Minimax theorem* we know that the worst-cases are exactly the equilibria they settle on. This is analogous to the finite problem, except the strategy space now has infinite dimensions. Linear programming can not solve this *exactly*, but semidefinite programming can. Finding a strategy with the best worst-case are actually two sides of polynomial optimization.

3.1 Polynomial Optimization

We are interested in the following polynomial optimization problem: Given a polynomial p, find the constrained global minimum

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & p(x) \\ \text{such that} \quad & x \in \mathcal{X} \end{aligned}$$

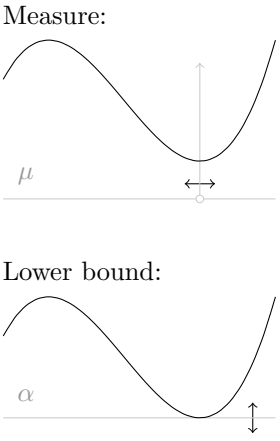
where $\mathcal{X}$ is a compact, but not necessarily convex set, defined by a system of polynomial equations and inequalities (Semialgebraic set). When possible, we are also interested in finding the global minimizers $x^\star$ of p[1].

Sum of Squares optimization transforms this nonconvex finite-dimensional polynomial optimization problem into a linear infinite-dimensional one. Very loosely, to find the optimal $y^\star = f(x^\star)$, we can either try to guess the y or the x's. Finding all the optimal x's is essentially a problem of finding an appropriate probability measure. Conversely, the $y^\star$ is unique, and it is the functions lower-bound. The implicit assumption behind both formulations being that it is easy to enumerate probability measures, and to check whether something is a lower-bound.

[1] Multivariate polynomials which are bounded below do not necessarily achieve a global minimum. A famous example is $x^2 + (1 - xy)^2$, which is clearly nonnegative and can get arbitrarily close to zero, but can never achieve it. Since we are interested in solving the problem numerically, we will make no accommodation for this fact.

More formally, minimizing a polynomial $p(x)$ over a compact semi-algebraic set $\mathcal{X}$ is equivalent to either: finding any probability measure supported on p's global minimizers; or finding the largest lower-bound α, which makes $p(x) - \alpha$ nonnegative on $\mathcal{X}$. These problems are mutually dual.

$$\begin{array}{ll|ll} \underset{\mu}{\text{minimize}} & \int p \, d\mu & \underset{\alpha}{\text{maximize}} & \alpha \\ \text{such that} & \mu(\mathcal{X}) = 1 & \text{such that} & p(x) - \alpha \geq 0, \quad \forall x \in \mathcal{X} \\ & \mu \in \mathscr{M}_+(\mathcal{X}) & & \end{array} \tag{3.1}$$



where $\mathscr{M}_+(\mathcal{X})$ is a convex cone of all measures on[2] $\mathcal{X}$.

An example of an optimal solution in the first case is when μ is a Dirac measure $\delta_{x^\star}$, and $x^\star$ is a global minimizer of $p(x)$, since by definition

$$\min_x \; p(x) = p(x^\star) = \int p \, d\delta_{x^\star}.$$

In the second case, the largest lower-bound α is a global minimum on $\mathcal{X}$. The expression $p(x) - \alpha$ is then nonnegative on $\mathcal{X}$ and p's global minimizers are its zeros.

These problems are linear, and their formulations are straightforward. Unfortunately, there exist no computationally efficient representations of neither $\mathscr{M}_+$ nor of polynomials nonnegative on $\mathcal{X}$. Semidefinite Programming has a numerical answer to both of these problems — Sum-of-Squares optimization. Rather than checking membership in the set of all nonnegative polynomials, the space is restricted to sums of squares of polynomials of bounded degrees. An SOS decomposition $p(x) = \sum_{i=0}^{n} s_i(x)^2$ syntactically certifies nonnegativity. Similarly, truncated sequences of moments replace nonnegative measures. These bounded degree SOS decompositions and the dual truncated moment-problems make up the Lasserre hierarchies (Lasserre, 2004).

Relaxations for Nonnegative Polynomials

Checking if a polynomial is globally-nonnegative is NP-hard, but checking a sufficient condition for nonnegativity is possible in polynomial time. One such certificate is a Sum-of-Squares decomposition, giving an easily verifiable answer, albeit approximate in general.

The Gram-matrix Method (Choi et al., 1994) A polynomial $p(x)$ of an even degree $2d$ with n variables is *Sum-of-Squares* if it can be written as a sum of finitely many squared polynomials: $p(x) = \sum_{i=0}^{m} q_i^2(x)$. For example, picking the monomial basis[3] $[x]_d$, the equation can be rewritten using vectors as:

$$p(x) = \begin{bmatrix} q_0(x) \\ \vdots \\ q_m(x) \end{bmatrix}^\intercal \begin{bmatrix} q_0(x) \\ \vdots \\ q_m(x) \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ x_n^d \end{bmatrix}^\intercal \mathrm{Q} \begin{bmatrix} 1 \\ \vdots \\ x_n^d \end{bmatrix}$$

[2] μ is a measure *on* $\mathcal{X}$, if the support of μ is a subset of $\mathcal{X}$

[3] Monomial basis is defined by a vectors of monomials $[x]_d = [1, \ldots, x_n^d]$ in the graded-lexicographical order, where n is the number of variables, and d is the degree bound.

where, since p is nonnegative, Q must be positive semidefinite[4]. Additionally, if $p(x)$ has a Sum-of-Squares decomposition, it involves at most $\binom{n+d}{n}$ polynomials, and their degrees are no more than d (This will not be the case for constrained optimization). Ensuring the equality of coefficients on both sides requires a finite number of linear constraints, and with the Q's only positive-semidefiniteness constraint, this is exactly a semidefinite program.

Nonnegative polynomials on compact semialgebraic sets For example, in a polynomial game, we are not interested in optimizing over the whole $\mathbb{R}^n$, but rather some compact subset of it. In our case, the strategy-spaces are basic-semialgebraic sets[5]. We are no longer interested in conditions for a single polynomial, but for systems of polynomial equations and inequalities. Furthermore, rather than trying to satisfy the equations and inequalities themselves, what we are interested in is the object they define. Sacrificing a bit of generality, which we did not need in the first place, and constraining our sets to be compact, we can use Putinar's simplification of the Positivstellensatz.

Theorem 2 (Putinar's Positivstellensatz (Putinar, 1993, p. 972))**.** *Given a compact semialgebraic set*[6] $\mathcal{X}$*, if a polynomial* $p(x)$ *is positive*[7] *on* $\mathcal{X}$*, then*

$$p(x) = s_0(x) + \sum s_i(x) g_i(x)$$

where $s_0(x)$ *and* $s_i(x)$ *are* $\mathcal{SOS}$*.*

Putinar's Positivestellensatz does not form a semidefinite program on its own, as it does not guarantee any bounds on the degrees of $s_i(x)$'s. It turns out, however, that simply iteratively bounding their degrees approximates, and eventually converges to the answer of the original problem (3.1); more on that in the section 3.2 on *Lasserre's Hierarchies*.

Because Putinar's Positivstellensatz applies only to strictly positive polynomials, we can only generate approximate representations in general. Alternative approximations exist, as well as special problem classes which lead to exact representations. The existence of an exact semidefinite representation is currently an open question.

Univariate polynomials on an interval Univariate polynomials nonnegative on an interval are an example of a special problem class with an exact characterization.

Theorem 3 (Theorem of Lukács (Szegő, 1939))**.** *Let* $p(x)$ *be an* n*th degree polynomial nonnegative on* $[-1, 1]$*. Then* $p(x)$ *can be written as:*

$$p(x) = \begin{cases} s(x) + (1+x)(1-x)t(x), & \text{if } n \text{ is even} \\ (1+x)s(x) + (1-x)t(x), & \text{if } n \text{ is odd} \end{cases}$$

[4] We always assume PSD matrices are symmetric

[5] Basic(-closed)-semialgebraic sets are subsets of $\mathbb{R}^n$ defined by polynomials inequalities as: $\{x \in \mathbb{R}^n \mid g_0(x) \geq 0,\, g_1(x) \geq 0, \dots\}$.

[6] Compact semialgebraic sets are compact sets formed by finite sequences of unions, intersections and complements of basic-semialgebraic sets.

[7] By positive we mean strictly positive, otherwise we use the term nonnegative.

Where $s(x)$ and $t(x)$ are sums-of-squares of polynomials, such that the degrees of the single terms on the right-hand side do not exceed n.

We can easily rewrite the Theorem of Lukács as an SDP. Since the conditions are both sufficient and necessary, the SDP will be exact.

Semidefinite Program 1. *A polynomial p with an even*[8] *degree $2n$ is non-negative on the interval $[-1, 1]$, if the following SDP constraints are feasible:*

$$\begin{aligned} p(x) &= a(x) + (1 - x^2)b(x) \\ b(x) &= [x]_{n-1} B [x]_{n-1}^\intercal \\ a(x) &= [x]_n A [x]_n^\intercal \\ A, B &\succeq 0 \end{aligned}$$

where polynomial equality is understood as $f = g \iff \vec{f} = \vec{g}$; and the variables $A \in S^{n+1}$, $B \in S^n$.[9] *The equivalence to the multivariate case is apparent when the interval $[-1, 1]$ is written as the basic semialgebraic set $\mathcal{X} = \{x \in \mathbb{R} \mid 1 - x^2 \geq 0\}$.*

Conditions for Valid Moment Sequences

We previously mentioned that Sum-of-Squares optimization is useful for representing not only nonnegative polynomials but also measures. The problems are, in fact, duals. Instead of relaxing the original dual formulation using SOS, we relax the original primal formulation by projecting it into a higher-dimensional space. The primal formulation tries to minimize the expectation of a polynomial $p(x)$ under a variable measure μ:

$$\mathbb{E}_\mu[p] = \int p \, d\mu$$

Factoring out the coefficients[10] of $p(x)$ leaves the expectation of monomials, which are exactly the moments of μ. The expectation of p under μ is then equal to $\vec{p} \cdot \vec{\mu}$, where $\vec{p}$ are the coefficients of $p(x)$, and $\vec{\mu}$ is the moment vector of μ. This automatically relaxes the problem by replacing non-linear monomials by moments acting as lifting variables. This procedure introduces what is known as the *Problem of Moments*. The moment vector $\vec{\mu}$ must actually have a representing measure. For multivariate moments, only necessary conditions exist; for univariate moments, the following conditions on moment matrices are sufficient as well as necessary.

Moment matrices (Lasserre, 2004) In the most general sense, a moment matrix $M(\mu)$ is an infinite generalized Hankel matrix, where each entry is a moment of a measure μ. Vectors of monomials provide a simple way to describe

[8] Odd degree polynomials can be interpreted as even degree polynomials with the leading coefficient equal to zero.

[9] S^n denotes the set of symmetric matrices in $\mathbb{R}^{n \times n}$

[10] A polynomial $p(x) = \vec{p} \cdot [x]_d$, where $\vec{p}$ are coefficients of monomials $[x]_d$.

their structure. For example, given an $[x]_2 = [1, a, b, a^2, ab, b^2]$:

$$[x]_2^\intercal [x]_2 = \begin{bmatrix} 1 & a & b & a^2 & ab & b^2 \\ a & a^2 & ab & a^3 & a^2b & ab^2 \\ b & ab & b^2 & a^2b & ab^2 & b^3 \\ a^2 & a^3 & a^2b & a^4 & a^3b & a^2b^2 \\ ab & a^2b & ab^2 & a^3b & a^2b^2 & ab^3 \\ b^2 & ab^2 & b^3 & a^2b^2 & ab^3 & b^4 \end{bmatrix}$$

Similarly — replacing every entry $a^i b^j$ by $\int a^i b^j \, d\mu = \mu_{i,j}$ — a moment matrix $M_m(\mu)$ truncated to order $m = 2$ has the structure[11]:

$$M_2(\mu) = \begin{bmatrix} \mu_{0,0} & \mu_{1,0} & \mu_{0,1} & \mu_{2,0} & \mu_{1,1} & \mu_{0,2} \\ \mu_{1,0} & \mu_{2,0} & \mu_{1,1} & \mu_{3,0} & \mu_{2,1} & \mu_{1,2} \\ \mu_{0,1} & \mu_{1,1} & \mu_{0,2} & \mu_{2,1} & \mu_{1,2} & \mu_{0,3} \\ \mu_{2,0} & \mu_{3,0} & \mu_{2,1} & \mu_{4,0} & \mu_{3,1} & \mu_{2,2} \\ \mu_{1,1} & \mu_{2,1} & \mu_{1,2} & \mu_{3,1} & \mu_{2,2} & \mu_{1,3} \\ \mu_{0,2} & \mu_{1,2} & \mu_{0,3} & \mu_{2,2} & \mu_{1,3} & \mu_{0,4} \end{bmatrix}$$

The matrix $[x]_m^\intercal [x]_m$ is always positive-semidefinite, and in order for the moment vector $\vec{\mu}$ to have a representing measure, $M_m(\mu)$ must be as well. The reason is that given a multivariate polynomial $p(x) = \vec{p} \cdot [x]_n$, the expected value of $p^2(x)$ can be written as:

$$\mathbb{E}_\mu[p^2] = \vec{p}^\intercal \, M_n(\mu) \, \vec{p}$$

and given that μ is nonnegative, the result should be too; therefore $M_n(\mu)$ must be positive-semidefinite. As in the dual problem, we need conditions for measures supported on a compact semialgebraic set. Given a polynomial $g(x)$, start analogously to the Moment Matrix construction with $g(x)[x]^\intercal[x]$, and define the associated Localizing Matrix, as its linearization. For example, given $g(x) = 1 - x^2$ and a univariate μ:

$$g(x)[x]_1^\intercal [x]_1 = \begin{bmatrix} x^0 - x^2 & x^1 - x^3 \\ x^1 - x^3 & x^2 - x^4 \end{bmatrix} \qquad \text{and} \qquad M_1(g\mu) = \begin{bmatrix} \mu_0 - \mu_2 & \mu_1 - \mu_3 \\ \mu_1 - \mu_3 & \mu_2 - \mu_4 \end{bmatrix}$$

where $M_1(g\mu)$ is a Localizing matrix of order 1. As before, for μ to be a representing measure of $\vec{\mu}$ on $\mathcal{X} = \{x \in \mathbb{R}^n \mid g \geq 0\}$, $M(g\mu)$ must be positive-semidefinite, because for every polynomial $p(x)$, the result of $\mathbb{E}\mu[g\, p^2] = \vec{p}^\intercal \, M_n(g\mu) \, \vec{p}$ (or $\int g(x) p^2(x) d\mu$) must be nonnegative when $g(x) \geq 0$. A simple combination of these two definitions gives a relaxation for probability measures on a compact semialgebraic set:

[11] The similarity to a Hankel matrix is visible from the degrees of the monomials.

$$\begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 \\ 1 & 2 & 2 & 3 & 3 & 3 \\ 1 & 2 & 2 & 3 & 3 & 3 \\ 2 & 3 & 3 & 4 & 4 & 4 \\ 2 & 3 & 3 & 4 & 4 & 4 \\ 2 & 3 & 3 & 4 & 4 & 4 \end{bmatrix}$$

For univariate measures, moment matrices and Hankel matrices are equivalent.

Lemma 1 (Moment matrix of a measure on $\mathcal{X}$ (Lasserre, 2004, p. 805))**.** *Let μ be a probability measure supported on a compact semialgebraic set defined by polynomials $g_i(x)$, then for all even $m \geq max(deg(g_0), \dots)$:*

$$\begin{aligned} M_{m-deg(g_i)}(g_i\mu) &\succeq 0 \qquad \forall i \\ M_m(\mu) &\succeq 0 \\ \mu_0 &= 1 \end{aligned}$$

where $\mu_0 = 1$, constrains the mass of μ to be 1, as it is supposed to be a probability measure.

Conditions for univariate measures The inverse condition, when every solution is also a valid measure, will be useful for solving polynomial games exactly. In that case, the measure is only univariate.

Semidefinite Program 2. *A sequence of moments $\vec{\mu} = [\mu_0, \dots, \mu_n]$ has a univariate representing measure on $[-1, 1]$ if and only if the following SDP constraints are feasible:*

$$\begin{aligned} \mu_0 &= 1 \\ M_{n-1}((1 - x^2)\mu) &\succeq 0 \end{aligned}$$

This result follows from the characterization of univariate polynomials (3.1) by the duality between nonnegative polynomials and moment spaces, and the fact that $[-1, 1]$ is compact.

3.2 Polynomial Optimization using Lasserre Hierarchies

Neither of the previously discussed relaxations (3.1, 3.1) directly forms a semidefinite program, as it is not obvious how to bound the order of moment matrices, nor the degrees of the constituent polynomial squares. The Moment–SOS hierarchy is a numerical approach, which fixes the degree, starting with the smallest possible degree, and then iteratively increases it. All solutions of the hierarchy provide a monotone sequence of bounds on the solution of the original problem (3.1), and eventually, they are guaranteed to converge.

Substituting the original intractable problems with the semidefinite constraints from Lemma 1 and Putinar's Positivstellensatz, and using m as an upper bound on their degrees, results in the following relaxations ordered by m.

Semidefinite Program 3 (Moment SDP)**.** *Optimization of a polynomial $p(x)$[12] over a compact semialgebraic set $\mathcal{X}$ with n inequalities — in other words, finding $\min_{x \in \mathcal{X}} p(x)$ — can be formulated as a moment semidefinite program in the following way:*[13]

$$\begin{aligned} \underset{\vec{\mu} \in \mathbb{R}^m}{\text{minimize}} \quad & \vec{p} \cdot \vec{\mu} \\ \text{such that} \quad & M_{m-deg(g_i)}(g_i\mu) \succeq 0 \qquad \forall i \\ & M_m(\mu) \succeq 0 \\ & \mu_0 = 1 \end{aligned} \qquad \text{(Mom. opt.)}$$

[12] A polynomial of a lower degree can be written as a polynomial of a higher degree with the leading coefficients equal to zero.

[13] $\vec{p}$ is the coefficient vector of $p(x) = \vec{p} \cdot [x]_m$, and $A \succeq 0$ denotes a positive-semidefiniteness constraint.

where $m \geq max(deg(p), deg(g_1), \ldots, deg(g_n))$, and $\vec{\mu}$ is the moment vector of a (hypothetical) measure μ.

Semidefinite Program 4 (Sum-of-Squares SDP)**.** *For an even parameter $m \geq max(deg(p), deg(g_1), \ldots, deg(g_n))$, the corresponding dual problem is formulated as follows:*

$$\begin{aligned} \underset{\gamma \in \mathbb{R}}{\text{maximize}} \quad & \gamma \\ \text{such that} \quad & s_0(x) + \sum_{i=1}^{n} s_i(x) g_i(x) = p(x) - \gamma \\ & deg(s_0),\ deg(s_i g_i) \leq m \qquad \forall i \\ & s_0,\ s_i \in \mathcal{SOS} \qquad \forall i \end{aligned} \tag{SOS opt.}$$

where by $f \in SOS$ we mean the Gram-matrix method (3.1).

The Lasserre hierarchy converges for *some* value of $m \geq \max(deg(p),\ \ldots)$. Unfortunately, when $dim(\mathcal{X}) \geq 3$, there exist $p(x)$s for which the optimal m is infinite. For univariate polynomials the lowest relaxation is exact already.

3.3 Pablo A. Parrilo's Algorithm for Polynomial Games

Parrilo's algorithm (Parrilo, 2007) uses semidefinite representations of non-negative univariate polynomials and measures to solve two-player zero-sum polynomial games supported on $[-1, 1] \times [-1, 1]$. While the resulting formulation looks unlike its finite-strategy counterpart, the underlying ideas are similar.

Both players are trying to find the best mixed-strategy. Parrilo defines their goals probabilistically — as an optimization of their expected payoffs:[14]

$$\max_{\mu} \min_{\nu} E_{\mu \times \nu}[p] \qquad \Big| \qquad \min_{\nu} \max_{\mu} E_{\mu \times \nu}[p]$$

where μ and ν are probability measures defining the strategies of players x and y; and p is the payoff polynomial. We can rewrite the goals as bilinear forms and, using an idea similar to Lasserre's moment relaxations, optimize over truncated sequences of moments instead of their representing measures.

$$\max_{\mu} \min_{\nu} E_{\mu \times \nu}[P(x, y)] \qquad \longrightarrow \qquad \max_{\vec{\mu}} \min_{\vec{\nu}} \sum_i \sum_j p_{ij} \mu_i \nu_j$$

Where μ_i and ν_j are the ith and jth moments of μ and ν. At this point, the Minimax Theorem (1) confirms that strategies found using this approach correspond to the equilibria of the new "relaxed" game. In our case $\mathcal{X}$ and $\mathcal{Y}$ are moment spaces, which are convex and compact (Karlin & Shapley, 1972); and since f is a bilinear form, it is continuous and trivially satisfies the concave-convexity.

The last idea exploits the separability of the payoff function.

$$\sum_i \sum_j p_{ij} \mu_i \nu_j^\star \quad \leq \quad \sum_i \sum_j p_{ij} \mu_i^\star \nu_j^\star \quad \leq \quad \sum_i \sum_j p_{ij} \mu_i^\star \nu_j$$

[14] $E_{\mu \times \nu}[\ldots]$ denotes the expectation under the product measure.

where $\mu^\star$ and $\nu^\star$ are any strategies resulting in a saddle-point. Just like in finite-games, it does not matter if a player plays first or responds to a strategy of the other player. The opponent can also do no better than to respond with the (constrained) global minimum. Thus simplifying the requirement from the optimal mixed-strategy to a set of strategies with the highest payoffs. For polynomial payoffs, this is a simple lower-bound constraint. This means that instead of a problem with a min, a max, and two variables, we can solve a constrained univariate polynomial optimization problem. Furthermore, since the problem is only univariate, optimizing over truncated sequences of moments instead of their representing measures does not actually "relax" the payoff function in any way (3.1).

Reformulation into a semidefinite program

Given a payoff function $p(x, y) = \sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij}x^i y^j$, player 2 can find a strategy with the smallest payoff on $\mathcal{Y} = [-1, 1]$ by solving the following optimization problem over measures:

$$\begin{aligned} \underset{\alpha,\,\mu}{\text{minimize}} \quad & \alpha \\ \text{such that} \quad & \mathbb{E}_\mu[p] \leq \alpha \quad \forall x \in [-1, 1] \\ & \mu(\mathcal{Y}) = 1 \end{aligned}$$

where $\alpha \in \mathbb{R}$, and $\mu \in \mathscr{M}_+(\mathcal{Y})$. Using the ideas above, Parrilo, 2007 then reformulates this as an optimization over moment vectors:

$$\begin{aligned} \underset{\alpha,\,\vec{\mu}}{\text{minimize}} \quad & \alpha \\ \text{such that} \quad & \alpha - \textstyle\sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij}x^i\mu_j \geq 0 \qquad \forall x \in [-1, 1] \\ & \mu_0 = 1 \\ & \vec{\mu} \text{ has a representing measure } \mu \end{aligned}$$

Finally, applying the SDP conditions for univariate representing measures (3.1) and nonnegative univariate polynomials (3.1) results in a semidefinite program for polynomial games.

Semidefinite Program 5 (Parrilo, 2007's polynomial-game solver (Parrilo, 2007))**.** *Given a polynomial game on $\mathcal{X} = \{x \in \mathbb{R} \mid g_1(x) \geq 0\}$, and $\mathcal{Y} = \{y \in \mathbb{R} \mid g_2(x) \geq 0\}$, where $g_1(x) = 1 - x^2$ and $g_2(y) = 1 - y^2$, with a payoff function $p(x, y) = \sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij}x^i y^j$, player 2 can find a truncated moment sequence of a representing optimal mixed-strategy using the following semidefinite program:*

$$\begin{aligned} \underset{\alpha,\,\vec{\nu}}{\text{minimize}} \quad & \alpha \\ \text{such that} \quad & \alpha - \textstyle\sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij}x^i\nu_j - s(x)g_1(x) = [x]_z Z [x]_z^\intercal \\ & s(x) = [x]_w W [x]_w^\intercal \\ & Z \succeq 0 \\ & W \succeq 0 \\ & M_{m-deg(g_2)/2}(g_2\nu) \succeq 0 \\ & M_m(\nu) \succeq 0 \\ & \nu_0 = 1 \end{aligned} \tag{3.2}$$

where polynomial equality is understood as $f = g \iff \vec{f} = \vec{g}$*; and the variables* $Z \in S^z$, $W \in S^w$,[15] $z = \lceil \frac{n-1}{2} \rceil$, *and* $w = \lceil \frac{n-1}{2} \rceil - \lceil \frac{deg(g_1)}{2} \rceil$.

3.4 Extension to Semialgebraic Games

At the end of his paper, Parrilo, 2007 mentions that his algorithm 3.3 could be extended to solve *Semialgebraic games*, i.e. games with a polynomial payoff, and strategy spaces defined by semialgebraic sets. While the problem setup is only slightly more complex than its univariate counterpart, it loses the necessary guarantees for exactness. It is no longer the case that all globally nonnegative polynomial are sums of squares, nor can the dual sets completely classify the moment problem. The problem suddenly becomes NP-hard.

That is not to say that the approach is no longer usable, nor do we have to significantly modify it. Applying the very same framework to the extended problem automatically results in an approximation algorithm.

The reasoning is anologous to the univariate problem. The two players x and y are given two algebraically-compact basic semialgebraic sets $\mathcal{X}$ and $\mathcal{Y}$, over which they are trying to find the strategies $\mu^\star$ and $\nu^\star$ with the best payoff determined by a polynomial p. This results in the following min-max problem:

$$\max_{\nu \in \mathcal{Y}} \min_{\mu \in \mathcal{X}} \int \int p \, d\mu \, d\nu \quad \Big| \quad \min_{\mu \in \mathcal{X}} \max_{\nu \in \mathcal{Y}} \int \int p \, d\mu \, d\nu$$

Due to the compactness of $\mathcal{X}$ and $\mathcal{Y}$, we can again use the Minimax theorem to conclude that at optimality, the two problems are equal. In other words, there exists an equilibrium for some optimal strategies $\mu^\star$ and $\nu^\star$, such that

$$\text{value of the game } = \int \int p \, d\mu^\star \, d\nu^\star$$

At this point the solutions start to differ.

Reformulation into a hierarchy of semidefinite programs

Whereas in the original problem we had concrete bounds on the size of the SOS decomposition of p and the orders of moment matrices, we now have two unbounded and growing hierarchies in the same problem. Unlike in the simple cases of polynomial optimization using either the moment or SOS hierarchy, the convergence of this complex problem is no longer monotone.

Given a multidimensional payoff function $p(x, y) = \sum_i \sum_j p_{ij} x^i y^j$, where i and j are now multiindexes, player 2 can find a strategy with the smallest payoff on $\mathcal{Y} = [-1, 1]$ by solving the following optimization problem:

$$\begin{aligned} \underset{\alpha,\, \mu}{\text{minimize}} \quad & \alpha \\ \text{such that} \quad & \mathbb{E}_\mu[p] \leq \alpha \quad \forall x \in \mathcal{X} \\ & \mu(\mathcal{Y}) = 1 \end{aligned}$$

[15] S^n denotes the set of symmetric matrices in $\mathbb{R}^{n \times n}$

where $\alpha \in \mathbb{R}$, and $\mu \in \mathscr{M}_+(\mathcal{Y})$. Using the same idea as in the univariate case, we can reformulate this as an optimization over moment vectors:

$$\begin{aligned} \underset{\alpha,\,\vec{\mu}}{\text{minimize}} \quad & \alpha \\ \text{such that} \quad & \alpha - \textstyle\sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij}x^i\mu_j \geq 0 \qquad \forall x \in \mathcal{X} \\ & \mu_0 = 1 \\ & \vec{\mu} \text{ has a representing measure } \mu \text{ on } \mathcal{Y} \end{aligned}$$

Finally, applying the full Lasserre hierarchy results in a hierarchy of semidefinite programs for semialgebraic games.

Semidefinite Program 6. *Given a polynomial game with a payoff function $p(x,y) = \sum_i \sum_j p_{ij}x^i y^j$, where $i \in \mathbb{N}^a$ and $j \in \mathbb{N}^c$ are multiindexes, and compact basic-semialgebraic strategy sets $\mathcal{X}$ and $\mathcal{Y}$, where*

$$\mathcal{X} = \{x \in \mathbb{R}^a \mid g_i(x) \geq 0,\ i = 1,\dots,b\}$$

$$\mathcal{Y} = \{x \in \mathbb{R}^c \mid h_j(x) \geq 0,\ j = 1,\dots,d\}$$

player 2 can find an optimal strategy $\nu^\star$ by solving the following hierarchy of semidefinite programs (where $g_0 = 1$)

$$\begin{aligned} \underset{\alpha,\,\vec{\nu}}{\text{minimize}} \quad & \alpha \\ \text{such that} \quad & \alpha - \textstyle\sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij}x^i\nu_j = \sum_{i=0}^{b} s_i(x)g_i(x) \\ & [x]_{d_i} Q_i [x]_{d_i}^\intercal = s_i \quad \forall i = 0,\dots,b \\ & M_{t-deg(h_j)/2}(h_j\nu) \succeq 0 \quad \forall j = 1,\dots,d \\ & M_t(\nu) \succeq 0 \\ & \nu_0 = 1 \end{aligned} \tag{3.3}$$

where t is the order of the hierarchy, and the degrees $2d_i$ of s_i are such that the degrees of $s_i(x)g_i(x)$, do not exceed $2t$, and the size of Q_i is $d_i - deg(g_i)/2$.

We also denote as t_0 the *minimal order* required to formulate the SOS and moment constraints, that is t_0 equals the highest exponent of p, any g_i's, or any h_j's, depending on whichever is highest.

Finally, let $\nu^\star$ be an optimal solution of 6 at some order t. We can then conclude optimality if

$$rank(M_{t-maxdeg(\mathcal{Y})/2}(\nu^\star)) = rank(M_t(\nu^\star)).$$

This condition is sufficient, but not necessary. Global optimum may have been attained, yet the rank condition may still be unsatisfied. If the condition holds, however, it also proves that $\nu^\star$ is a $rank(M_t(\nu^\star))$-atomic measure supported on $\mathcal{Y}$, and we have a technique to extract its atoms.

3.5 Recovering Players' Mixed Strategies

Neither of the algorithms (3.4, 3.3) returns an optimal strategy, only a moment vector that describes one. There exist several techniques[16] to extract the

[16] On the other hand, no techniques exist for recovering the global minimizers straight from the dual SOS decomposition.

(atomic) representing measure of a moment vector, but we will only describe one numerical approach (Henrion & Lasserre, 2005) which seems to work well in practice.

Since the optimal strategy μ has finitely many atoms, and each entry of its moment matrix $M(\mu)$ is a moment[17] $\int x^n \, d\mu = \sum_{i=1}^{r} w_i \overset{\star i}{x}{}^n$, we can exploit the structure of any truncated moment matrix $M_k(\mu)$[18],

$$M_k(\mu) = \sum_{i=1}^{r} w_i^2 [\overset{\star i}{x}]_k^\intercal \cdot [\overset{\star i}{x}]_k = \overset{\star}{V} W \overset{\star}{V}{}^\intercal$$

where $\overset{\star}{V} = \left[[\overset{\star 1}{x}]_k, [\overset{\star 2}{x}]_k, \ldots, [\overset{\star r}{x}]_k \right]$, and W is a diagonal matrix of weights w_i^2.

Given a $M_k(\mu)$ obtained from *some* level of the hierarchy (3.4), we can use Cholesky-like decomposition to get $M_k(\mu) = VV^\intercal$ instead. The columns of $V^\intercal$ are indexed by monomials of $[x]_k$, and after reducing $V^\intercal$ to row-echelon form, the pivots correspond to the basis of μ's atoms.

To extract the atoms, first create *multiplication matrices* for each variable x_i. Given a $V^\intercal$ reduced to row echelon form, for example

$$\begin{bmatrix} 1 & 0 & 0 & -2 & -4 & -6 \\ 0 & 1 & 0 & 3 & 2 & 0 \\ 0 & 0 & 1 & 0 & 2 & 5 \end{bmatrix}, \\ \begin{matrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \end{matrix}$$

its basis is $[1, x_1, x_2]$, and the *multiplication matrix* of x_1 consists of all the columns of $V^\intercal$ indexed by $x_1 \times [1, x_1, x_2]$ (i.e. the columns x_1, x_1^2, and x_1x_2). That is

$$N_{x_1}^\intercal = \begin{bmatrix} 0 & -2 & -4 \\ 1 & 3 & 2 \\ 0 & 0 & 2 \end{bmatrix} \quad \text{whereas} \quad N_{x_2}^\intercal = \begin{bmatrix} 0 & -4 & -6 \\ 0 & 2 & 0 \\ 1 & 2 & 5 \end{bmatrix}.$$

The solutions are now embedded as common eigenvalues of the matrices N_{x_i}. To recover them without duplicates, pick a vector $\lambda \in \Delta^n$ from random simplex[19], and create a linear combination of all the multiplication matrices

$$N = \sum_{i=1}^{n} \lambda_i N_{x_i} \, .$$

Finally, compute the ordered Schur decomposition $N = QUQ^\intercal$, and use the columns of $Q = [q_1, \ldots, q_r]$ to recover the r solutions:

$$\overset{\star i}{x}_j = q_i^\intercal N_{x_j} q_i$$

for each variable x_j, $j = 1, \ldots, n$ and each atom $\overset{\star i}{x}$, $i = 1, \ldots, r$.

[17] We use the notation $\overset{\star i}{x}$ to denote the ith global minimizer (in this case, an atom).

[18] Similarly, the notation $[\overset{\star i}{x}]_d$ denotes the basis vector $[x]_d$, except $x = \overset{\star i}{x}$.

[19] The standard simplex in $\mathbb{R}^n$ is $\Delta^n = \{x \in \mathbb{R}^n \mid \sum_{i=1}^{n} x_i = 1, \; x_i \geq 0 \;\; \forall i = 1 \ldots n\}$

Extracting atoms of univariate measures from the primal-dual solutions

In the original paper, Parrilo (2007) suggests we extract the atomic measures from the primal-dual solution instead. Since many solvers solve the primal and dual problems simultaneously anyway, no additional overhead is created. The following approach to recover a univariate atomic measure from its moments is based on a classical procedure. Only the first step differs, when we extract the support from the dual problem.

The first step uses the fact that the entire reason for the polynomial (3.3) $\alpha - E[p]$ nonnegative on $\mathcal{X}$, was to be a proxy for the opponent's best response. The zeros of the polynomial must then give the support of the optimal strategy, that is, an atomic measure. Then let μ_j for $j = [0:n]$ be the moments of μ, and z_i for $i = [0:n]$ zeros of p. The corresponding weights can then be obtained by solving a Vandermonde system given by

$$\sum_{i=0}^{n} w_i z_i^j = \mu_j,$$

or in a matrix format by

$$\begin{bmatrix} z_0^0 & \cdots & z_0^n \\ \vdots & \ddots & \vdots \\ z_n^0 & \cdots & z_n^n \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mu_0 \\ \vdots \\ \mu_n \end{bmatrix}.$$

However, this procedure will work only if the supports and the moments are sufficiently accurate; otherwise, it runs into numerical problems. The general technique, which extracts measures from their moment matrices, can handle inaccuracies better. In that case, we may have to solve a larger problem first.

Chapter 4

Approximation

We follow up on Parrilo's algorithm idea 3.3 and apply on top of it a relaxation sequence based on diagonally-dominant matrices recently introduced by A. Ahmadi and Majumdar, 2017. Their approach promises to improve scalability at the cost of suboptimal solutions. The basic idea is simple:

A semidefinite program optimizes over the intersections of a cone of PSD matrices and an affine subspace. If we could inner-approximate the cone of PSD matrices with linear constraints, we could transform any SDP problem into a linear program and get only feasible solutions. The idea of A. Ahmadi and Majumdar was to use Diagonally-Dominant matrices for this purpose.

4.1 DSOS Optimization

Every symmetric diagonally-dominant real matrix with nonnegative diagonal entries is positive-semidefinite. We assume that the *symmetric* and *real* constraints are satisfied. On top of that, diagonal-dominance with nonnegative diagonal entries requires only linear constraints.

Definition 1. *A matrix M is diagonally-dominant, or $M \in dd$, if:*

$$|M_{ii}| \geq \sum_{i \neq j} |M_{ij}|, \quad \forall i$$

Removing the absolute value function from the left-hand-side of the inequality also constrains the diagonal entries of M to be nonnegative (for matrices with side-length greater than 1).

This idea can immediately relax any SDP into an LP (also called DDP):

$$\begin{array}{rl} \min & \langle C, X \rangle \\ \text{subject to} & \langle A_i, X \rangle = b_i \quad \forall i \\ & X \succeq 0 \end{array} \qquad \longmapsto \qquad \begin{array}{rl} \min & \langle C, X \rangle \\ \text{subject to} & \langle A_i, X \rangle = b_i \quad \forall i \\ & X \in \text{dd} \end{array}$$

In case the trade-off in optimality is unacceptable, the relaxation can be iteratively improved using knowledge from a previous iteration. Instead of X being diagonally-dominant directly, X could be dd in a different basis. For this A. Ahmadi and Majumdar define a family of cones parametrized U:

$$DD(U) = \{X \in S_n \mid X = U^\intercal Q U, \ Q \in \text{dd}.\}$$

where S_n are symmetric matrices of size n.

The nth relaxation in the $DSOS$ sequence is then defined as:

$$DSOS_k = \begin{array}{rl} \min & \langle C, X_k \rangle \\ \text{subject to} & \langle A_i, X_k \rangle = b_i \quad \forall i \\ & X_k \in DD(U_k) \end{array} \tag{4.1}$$

And the sequence of U's (starting at 0) as: $[\mathcal{I}, chol(X_0), chol(X_1), \dots]$, where $chol(X)$ is the Cholesky-like[1] factor V, as in: $X = VV^\intercal$.

Since $U_{k+1} = V_k$ is a factor of the solution from the previous iteration, then $X_k = U_{k+1}^\intercal \mathcal{I} U_{k+1}$; and since $\mathcal{I}$ is dd, then $X_k \in DD(U_{k+1})$. This means that X_k is feasible solution to $DSOS_{k+1}$, and so the optimal solution can not be worse than it. Therefore the sequence of solutions is non-increasing.

SDP Relaxation using Scaled-Diagonally-Dominant Matrices

The same idea as with diagonally-dominant matrices applies to scaled-diagonally-dominant matrices (A. Ahmadi & Majumdar, 2017), except second-order-cone constraints are required.

Definition 2. *A symmetric scaled-diagonally-dominant matrix is a matrix of the form DMD, where M is symmetric and diagonally-dominant, and D is an arbitrary nonsingular diagonal matrix.*

Checking if a matrix is *sdd* is possible using linear programming, but a constraint for *sdd* requires second-order-cone constraints.

Lemma 2. *A symmetric matrix Q is sdd iff $Q = \sum_{i \le j} M^{ij}$, where each M^{ij} only has four nonzero entries $(M^{ij})_{ii}$, $(M^{ij})_{ij}$, $(M^{ij})_{ji}$, $(M^{ij})_{jj}$, such that:*

$$\begin{bmatrix} (M^{ij})_{ii} & (M^{ij})_{ij} \\ (M^{ij})_{ji} & (M^{ij})_{jj} \end{bmatrix} \succeq 0$$

The family of $SDD(U)$ cones is then defined analogously to the diagonally-dominant case:

$$SDD(U) = \{X \in S_n \mid X = U^\intercal Q U,\ Q \in \text{sdd.}\}$$

and the SDSOS sequence is analogous as well. The nth iteration in the $SDSOS$ sequence is then defined as:

$$SDSOS_k = \begin{array}{rl} \min & \langle C, X_k \rangle \\ \text{subject to} & \langle A_i, X_k \rangle = b_i \quad \forall i \\ & X_k \in SDD(U_k) \end{array} \tag{4.2}$$

with the same sequence of U's $= [\mathcal{I}, chol(X_0), chol(X_1), \dots]$.

[1] A. Ahmadi and Majumdar (2017) define $chol(\mathcal{X})$ as a Cholesky factor. However, since diagonally-dominant matrices are only positive-semidefinite, not positive-definite, Cholesky decomposition is not applicable. In another related paper (A. A. Ahmadi & Hall, 2015) the authors suggest using the spectral decomposition, or the LDL decomposition. Either of those approaches would work; however, the proof of iterative improvement would be different. In a related survey (Majumdar et al., 2019), the authors also suggest using "the square root operation", by which we assume they mean the factorization $X = F^* F$, where F^* is a conjugate transpose of F. For positive-semidefinite matrices, such a factor is non-complex.

4.2 Application of DSOS to Semialgebraic Games

We can use several techniques to attempt to simplify the problem of finding equilibria in semialgebraic games:

The full iterative change-of-basis approach In this method, we first transform the abstract formulation into the standard SDP form, then use the literal interpretation of transformation 4.1, or 4.2. Apart from the requirement of a specific SDP form, the apparent flaw in this approach is that the resulting number of constraints will be excessively large, as the method does not exploit the structure of the problem. This method will almost certainly never lead to faster solve-times compared to SOS optimization. The change-of-basis method is likely only a simplified academic example of a converging SDSOS hierarchy, and was not intended to be practical.

By inner-approximating each SDP matrix separately This is both simpler to implement and does not lose the structure of the problem.

Using (S)DSOS polynomials A. Ahmadi and Majumdar also define DSOS and SDSOS polynomials analogously to SOS polynomials, except their Gram-matrices are diagonally-dominant instead of positive semidefinite. This method is the simplest to implement, reduces the number of PSD constraints, but does not lead to LP and SOCP programs.

A disadvantage of DSOS and SDSOS polynomial constraints is that it is not obvious how to extract the minimizers from the dual.

4.3 Double Oracle Algorithm for Polynomial Games

The Double Oracle Algorithm (McMahan et al., 2003) is an efficient and converging iterative algorithm originally created to solve zero-sum matrix games for which it was impractical to account for all possible strategies. As the main idea of the algorithm requires only that we be able to compute the best response to an opponent's strategy, we see no reason why it could not be applied to continuous games. Nonetheless, to the best of our knowledge, such use does not appear in the literature. We formulate the natural extension of the Double Oracle algorithm to polynomial games, implement a proof-of-concept, and test its convergence in practice.

Conceptually, the algorithm has four main parts:

1. Guess random initial supports of the optimal strategies
2. Evaluate the utility polynomial at each point in the Cartesian product of the supports of the current strategies, thereby reducing the problem to a matrix game.
3. Solve the matrix game, and using the solutions compute the loss functions for the possible strategies of each player.
4. Minimize the loss function, and append the best responses to the current supports.

Steps 2 through 4 repeat until the strategies meet *some* optimality criteria. The original Double Oracle algorithm is guaranteed to converge to a minimax equilibrium eventually. Unfortunately, there exists no such guarantee for the continuous version. We are also not aware of any bounds on the speed of convergence of either this or the original algorithm.

An obvious difficulty in our continuous formulation are the "oracles" which should provide the best responses against concrete strategies (Step 4). The choice depends on the dimensionality of the utility polynomial, the complexity of strategy spaces, and whether an exact best response is required. Options range from Monte Carlo Sampling, through Lasserre's Moment Hierarchy, all the way to local optima search. The selection will ultimately dictate the practicality of the algorithm, however, as we do not know the implications of any of them, we can only guess. Going forward, we will assume that for polynomial games on a unit square, Simulated Annealing is a sufficient oracle.

Pseudocode

Algorithm 1 shows a template for our simple proof-of-concept program.[2] While it is obviously inefficient, it should be simple to understand. We take the notational liberty and assume that functions are transparently vectorized and broadcast, that is, that a function $f(x)$ evaluated on a sequence $f([a, b, c])$ is equivalent to it being evaluated on each element $[f(a), f(b), f(c)]$. Line 6 solves a matrix game (in our case, using linear programming 2.1) and returns the optimal strategies as probabilities corresponding to the supports given. Lines 9 and 10 invoke the "best-response oracles" *min*, which minimize the loss functions. Convergence in the original algorithm (McMahan et al., 2003) was concluded when best responses were already parts of the current strategies. For continuous games, which can have infinitely many equilibria, either a deterministic oracle is necessary, or we may, for example, infer it from the change in the value of the game.

Algorithm 1 Double Oracle for Polynomial Games

1: **procedure** DOUBLE_ORACLE($p(x, y), \mathcal{X}, \mathcal{Y}$)
2: $support_x \leftarrow$ random points on $\mathcal{X}$ ▷ Initial guess
3: $support_y \leftarrow$ random points on $\mathcal{Y}$
4: **repeat**
5: $matrix \leftarrow p(support_x \times support_y)$
6: $prob_x,\ prob_y \leftarrow solve\_matrix\_game(matrix)$
7: $loss_x \leftarrow -prob\_y \cdot p(x, support_y)$
8: $loss_y \leftarrow \ \ prob\_x \cdot p(support_x, y)$
9: $best\_response_x \leftarrow min(loss_x)$
10: $best\_response_y \leftarrow min(loss_y)$
11: $support_x \leftarrow [support_x;\ best\_response_x]$
12: $support_y \leftarrow [support_y;\ best\_response_y]$
13: **until** convergence
14: **end procedure**

[2] The corresponding Julia implementation is in the appendix of this thesis.

Chapter 5

Conclusions and results

The following tests were performed based on techniques from sections 3.3, 3.4, 3.5, 4.1, and 4.1, using implementations described in appendix A. We use t_0 to denote the minimum order (3.5) of the hierarchies required for a problem, and call each additional order an *iteration* (order t_0 is the first iteration). As these tests were mainly intended as a proof-of-concept, run-time should not be viewed as an estimator of real performance. All tests were performed on an *Intel Core i5-7200U Processor*, and 8 GB of RAM.

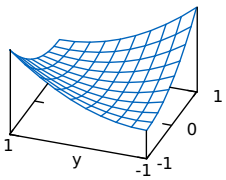
5.1 Numerical Experiments with SOS Optimization and the comparison to DSOS and SDSOS Optimization

In this section, we test the algorithm for polynomial games by Parrilo (3.3) along with its extension to semialgebraic games (3.4). We then take each known converging solution (at the appropriate order in the hierarchy), and attempt to solve it using DSOS and SDSOS. Below each example, we list every such successful application. We consider a failure every relaxation that fails to complete within 5 minutes or fails due to numerical issues.

The following examples are not original[1], except for the polynomials generated using the equation (5.1) of Gale and Gross (1958).

Examples on intervals

The following examples and solutions were taken from Parrilo (2007), and those, in turn, have been "extracted from the literature". In these examples, the sets are: $\mathcal{X} = [-1, 1]$ and $\mathcal{Y} = [-1, 1]$, where $[-1, 1]$ has the defining polynomial $g = 1 - x^2$.



1. This example is a guessing game defined by

$$p(x, y) = (x - y)^2.$$

We can solve this game using Parrilo's algorithm (3.3) and recover the optimal strategies from the primal-dual solutions (3.5). Choosing to

[1] Some of the following results differ in their sign compared to the original solutions. As the solutions are identical otherwise, we conclude this is likely caused by the accidental inversion of players.

extract the atoms using the general method (3.5) instead, would require the solution to the second iteration (order $t_0 + 1$) of the hierarchies (3.4).

The Lasserre hierarchy converges during the second iteration, while the value of the game is exact at the first iteration (order t_0) already. Both the Parrilo's algorithm and it's semialgebraic extension return the correct result:

$$\overset{\star}{\mu} = \begin{array}{l} -1 \text{ with weight } 0.5 \\ 1 \text{ with weight } 0.5 \end{array}, \qquad \overset{\star}{\nu} = 0 \text{ with weight } 1$$

resulting in a value of the game $\alpha = 1$.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials result in the same value of the game and the same strategies.

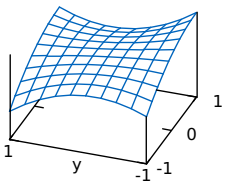
Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD and SDD constraints results in the same answers.

Iterative change-of-basis The series based on DD matrices returns the same answers after one iteration.

---

2. Consider the function

$$p(x, y) = 2xy^2 - x^2 - y,$$



which is convex in x, and therefore has only pure strategy solutions. Because of this, the first iteration is sufficient for convergence. Both algorithms return the correct result:

$$\overset{\star}{\mu} = 0.397 \text{ with weight } 1\,, \qquad \overset{\star}{\nu} = 0.63 \text{ with weight } 1$$

resulting in a value of the game $\alpha = -0.4724$.

DSOS and SDSOS polynomials Relaxation using SDSOS polynomials results in the same value of the game and the same strategies. Relaxation using DSOS polynomials approximates the value of at -0.56, and returns the strategies

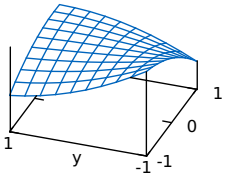
$$\mu' = 0.25 \text{ with weight } 1\,, \qquad \nu' = 0.5 \text{ with weight } 1\,.$$

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD constraints approximates the value of the game at 0 and fails to extract any strategies. SDD constraints fail similarly, with an approximation of the value of the game at 0 and strategies incorrectly centered at 1.

Iterative change-of-basis The series based on DD matrices results in the sequence of approximations $[-0.7, -0.5, -0.479, -0.478]$, but does not improve further. The best approximations of the strategies is:

$$\overset{\star}{\mu} = \ 0.353 \ \text{ with weight } \ 1\ , \qquad \overset{\star}{\nu} = \ 0.84 \ \text{ with weight } \ 1$$

Using SDD matrices instead returns the correct result after one iteration.



3. The payoff function is given by

$$p(x, y) = 5xy - 2x^2 - 2xy^2 - y,$$

which is neither convex nor concave. The hierarchy converges during the second iteration, while the value is exact already on the first iteration. Both algorithms return the correct result:

$$\overset{\star}{\mu} = \ 0.2 \ \text{ with weight } \ 1\ , \qquad \overset{\star}{\nu} = \begin{matrix} 1 & \text{with weight} & 0.78 \\ -1 & \text{with weight} & 0.22 \end{matrix}$$

resulting in a value of the game $\alpha = -0.48$.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials result in the same value of the game and the same strategies.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD constraints approximates the value of the game at 0 and returns strategies incorrectly centered at $(0, 0)$. DD constraints approximate the value at 0.5 and return the correct strategy for player 1, but not player 2.

Iterative change-of-basis The series based on SDD matrices results in the sequence of approximations $[-0.28, -0.42, -0.453]$, but does not improve further. The best approximations of the strategies is:

$$\overset{\star}{\mu} = \ 0.2 \ \text{ with weight } \ 1\ , \qquad \overset{\star}{\nu} = \begin{matrix} 1 & \text{with weight} & 0.73 \\ -1 & \text{with weight} & 0.26 \end{matrix}$$

The run-time for all of the examples above is trivial, regardless of the solver and formulation used.

Examples on simplices

The following examples and their expected solutions were taken from Nie et al. (2018). In these examples, the sets are $\mathcal{X} = \Delta^n$ and $\mathcal{Y} = \Delta^m$, where $\Delta^n = \{x \in \mathbb{R}^n \mid e^\intercal x = 1,\ x \geq 0\}$ is a simplex with the defining polynomial tuple $g = (e^\intercal x - 1, x_1, \ldots, x_n)$. As these examples are multi-dimensional and defined using Semialgebraic sets, they can not be solved using Parrilo's algorithm.

1. Let $n = m = 3$, and

$$p(x, y) = x_1x_2 + x_2x_3 + x_3y_1 + x_1y_3 + y_1y_2 + y_2y_3.$$

We confirm one saddle point of Nie et al., 2018, and find two more solutions. The hierarchy converges during iteration 3, while the value of the game is exact from the second iteration. The resulting strategies are:

$$\overset{\star}{\mu} = \ 0,\ 1,\ 0 \ \text{ with weight } \ 1\ , \quad \overset{\star}{\nu} = \begin{array}{l} 0.02,\ 0.5,\ 0.48 \ \text{ with weight } \sim 0.3 \\ 0.25,\ 0.5,\ 0.25 \ \text{ with weight } \sim 0.3 \\ 0.48,\ 0.5,\ 0.02 \ \text{ with weight } \sim 0.3 \end{array}$$

which result in a value of the game of $\alpha = 0.25$. The run-time is around 3 seconds, and the results are inaccurate due to the large order.

All attempts to relax the problem using DSOS and SDSOS optimization result in infeasibility or fail due to numerical issues.

---

2. Let $n = m = 3$, and

$$\begin{aligned} p(x, y) = {} & x_3y_1y_2(y_1 + y_2) + x_2y_1y_3(y_1 + y_3) + x_1y_2y_3(y_2 + y_3) \\ & + x_1^3 + x_2^3 - x_3^3 - y_1^3 - y_2^3 - y_3^3. \end{aligned}$$

Our results agree with the original paper. The answer is already exact at first iteration:

$$\overset{\star}{\mu} = \ 0,\ 1,\ 0 \ \text{ with weight } \ 1\ , \qquad \overset{\star}{\nu} = \ 0,\ 0,\ 1 \ \text{ with weight } \ 1$$

which result in a value of the game of $\alpha = 0$. The run-time is around a tenth of a second.

DSOS and SDSOS polynomials Relaxations using DSOS and SDOS polynomials result in the same value of the game, but both fail during the extraction of the optimal strategies.

Other attempts to use SDSOS optimization result in infeasible problems.

---

3. Let $n = m = 4$, and

$$p(x, y) = \sum_{i,j=1}^{4} x_i^2 y_j^2 - \sum_{i \neq j} (x_i x_j + y_i y_j)$$

Our results agree completely. The strategies and the value of the game converge at the second iteration:

$$\overset{\star}{\mu} = \ 0.25,\ 0.25,\ 0.25,\ 0.25 \ \text{ with weight } \ 1,$$

and

$$\overset{\star}{\nu} = \begin{array}{l} 1,\ 0,\ 0,\ 0 \ \text{ with weight } \ 0.25 \\ 0,\ 1,\ 0,\ 0 \ \text{ with weight } \ 0.25 \\ 0,\ 0,\ 1,\ 0 \ \text{ with weight } \ 0.25 \\ 0,\ 0,\ 0,\ 1 \ \text{ with weight } \ 0.25 \end{array},$$

which result in a value of the game of $\alpha = 0.688$. The run-time is around a tenth of a second.

DSOS and SDSOS polynomials Relaxation using DSOS and DSOS polynomials result in the same value of the game, but the extraction of the optimal strategies fails.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by SDD constraints approximates the value of the game at 0.79, but fails to return strategies.

The change-of-basis method failed to return a result.

---

4. Let $n = m = 3$, and

$$p(x, y) = x_1x_2y_1y_2 + x_2x_3y_2y_3 + x_3x_1y_3y_1 - x_1^2y_3^2 - x_2^2y_1^2 - x_3^2y_3^2.$$

In this case, the original paper found no saddle points, optimal mixed-strategies exist regardless. Our strategies converge at second iteration, and the value is exact on first iteration:

$$\overset{\star}{\mu} = \begin{array}{lll} 1, 0, 0 & \text{with weight} & 1/3 \\ 0, 1, 0 & \text{with weight} & 1/3 \\ 0, 0, 1 & \text{with weight} & 1/3 \end{array}, \quad \overset{\star}{\nu} = 1/3, 1/3, 1/3 \text{ with weight } 1$$

which result in a value of the game of $\alpha = 0.11$. The run-time is around a tenth of a second.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials result in the same value of the game, but the extraction of the optimal strategies fails.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by SDD constraints returns the correct value of the game and strategies for player 1, but not for player 2.

The change-of-basis method failed to return a result.

Examples on boxes

In these examples, the sets are $\mathcal{X} = [0, 1]^n$ and $\mathcal{Y} = [0, 1]^m$, where $[0, 1]^n$ is a box with the defining polynomial tuple $g = (x_1, \ldots, x_n, 1 - x_1, \ldots, 1 - x_n)$.

1. Let $n = m = 2$, and

$$p(x, y) = (x_1 + x_2 + y_1 + y_2 + 1)^2 - 4(x_1x_2 + x_2y_1 + y_1y_2 + y_2 + x_1).$$

Our strategies disagree with the saddle points, but the value of the game is the same. The strategies and the value converge at second iteration:

$$\overset{\star}{\mu} = \begin{array}{lll} 0.639, 0.639 & \text{with weight} & 0.605 \\ 0.144, 0.144 & \text{with weight} & 0.368 \end{array}, \quad \overset{\star}{\nu} = 1, 0 \text{ with weight } 1$$

which result in a value of the game of $\alpha = -4$. The run-time is less than a tenth of a second.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials results in the same value of the game, but the extraction of the optimal strategies fails.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by SDD constraints returns the correct value of the game, but fails to extract strategies.

The change-of-basis method failed to return a result.

---

2. Let $n = m = 3$, and

$$p(x, y) = \sum_{i=1}^{n}(x_i + y_i) + \sum_{i<j}(x_i^2 y_j^2 - y_i^2 x_j^2).$$

In this case, the original paper found no saddle points. An equilibrium is guaranteed, however, we had trouble finding it. Our strategies converge at third iteration:

$$\overset{\star}{\mu} = \ 0, 0, 0 \ \text{ with weight } \ 1 \ , \qquad \overset{\star}{\nu} = \ 1, 1, 1 \ \text{ with weight } \ 1$$

which result in a value of the game of $\alpha = 4$. The run-time was a couple of seconds. In this example most solvers failed to return the correct strategy: the *SCS* solver exited with *Solved/Inaccurate*; the *CSDP* solver only found player 2's strategy, and gave up for a *lack of progress*; and while the *SDPA* solver did eventually find the correct solution, it resulted in a *segmentation fault* the first couple of times. We rationalize this by the fact that the *Julia* language is fairly new, and its modeling framework *JuMP* is an open source effort, currently undergoing rapid development.

DSOS and SDSOS All attempt to approximate the problem run into numerical issues.

Examples on hypercube sets

In these examples, the sets are $\mathcal{X} = [-1, 1]^3$ and $\mathcal{Y} = [-1, 1]^3$, where $[-1, 1]^n$ is a hypercube with the defining polynomial tuple $g = (1 - x_1^2, \ldots, 1 - x_n^2)$.

1. Consider the function

$$p(x, y) = \sum_{i=1}^{3}(x_i + y_i) - \prod_{i=1}^{3}(x_i - y_i).$$

We confirm the results of Nie et al., 2018. The hierarchy converges at iteration 2 returning the strategies:

$$\overset{\star}{\mu} = \begin{array}{llll} & -1, & 1, & 1 \text{ with weight } 1/3 \\ & 1, & -1, & 1 \text{ with weight } 1/3 \\ & 1, & 1, & -1 \text{ with weight } 1/3 \end{array}, \quad \overset{\star}{\nu} = -1, -1, -1 \text{ with weight } 1,$$

which result in a value of the game of $\alpha = -2$. The run-time was less than a tenth of a second.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials results in the same value of the game and the same strategies.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD and SDD constraints approximates the value of the game at -4, but fails to extract strategies.

The Iterative change-of-basis method failed to return a result.

---

2. Consider the function

$$p(x, y) = y^\intercal y - x^\intercal x + \sum_{1 \le i < j \le 3} (x_i y_j - x_j y_i).$$

Our results find the original saddle point, but our full solutions are larger. The hierarchies converge during the second iteration, returning the strategies:

$$\overset{\star}{\mu} = \begin{array}{llll} & 1, & 1, & 1 \text{ with weight } 1/8 \\ & 1, & 1, & -1 \text{ with weight } 1/8 \\ & 1, & -1, & 1 \text{ with weight } 1/8 \\ & 1, & -1, & -1 \text{ with weight } 1/8 \\ & -1, & 1, & 1 \text{ with weight } 1/8 \\ & -1, & 1, & -1 \text{ with weight } 1/8 \\ & -1, & -1, & 1 \text{ with weight } 1/8 \\ & -1, & -1, & -1 \text{ with weight } 1/8 \end{array}, \quad \overset{\star}{\nu} = \begin{array}{llll} & 1, & 1, & 1 \text{ with weight } 1/8 \\ & 1, & 1, & -1 \text{ with weight } 1/8 \\ & 1, & -1, & 1 \text{ with weight } 1/8 \\ & 1, & -1, & -1 \text{ with weight } 1/8 \\ & -1, & 1, & 1 \text{ with weight } 1/8 \\ & -1, & 1, & -1 \text{ with weight } 1/8 \\ & -1, & -1, & 1 \text{ with weight } 1/8 \\ & -1, & -1, & -1 \text{ with weight } 1/8 \end{array}$$

which result in a value of the game of $\alpha = 0$. The run-time was less than a tenth of a second.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials result in the same value of the game and the same strategies.

Other methods of approximations failed to return a result.

Example on a sphere set

The sets are $\mathcal{X} = \mathbb{O}^2$ and $\mathcal{Y} = \mathbb{O}^2$, where $\mathbb{O}^{n-1} = \{x \in \mathbb{R}^n \mid \|x\| = 1\}$ is a hypersphere with the defining polynomial $g = 1 - x^\intercal x$.

1. Consider the function

$$\begin{aligned} p(x, y) = {} & 2(x_1 x_2 y_1 y_2 + x_1 x_3 y_1 y_3 + x_2 x_3 y_2 y_3) \\ & + x_1^3 + x_2^3 + x_3^3 + y_1^3 + y_2^3 + y_3^3. \end{aligned}$$

We confirm the results of Nie et al., 2018. The hierarchy converges on the first iteration, and returns the strategies:

$$\overset{\star}{\mu} = \begin{array}{lll} -1,\,0,\,0 & \text{with weight} & 1/3 \\ 0,\,-1,\,0 & \text{with weight} & 1/3\,, \\ 0,\,0,\,-1 & \text{with weight} & 1/3 \end{array} \qquad \overset{\star}{\nu} = \begin{array}{lll} 1,\,0,\,0 & \text{with weight} & 1/3 \\ 0,\,1,\,0 & \text{with weight} & 1/3\,, \\ 0,\,0,\,1 & \text{with weight} & 1/3 \end{array}$$

which results in a value of the game of $\alpha = 0$. As an aside, we note that, purely thanks to the immediate convergence, our run-time was less than a tenth of a second (compared to over a minute) on comparable hardware.

DSOS and SDSOS polynomials Relaxations using DSOS and SDSOS polynomials result in the same value of the game and the same strategies.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD constraints approximates the value of the game at -1. The corresponding strategies are

$$\mu = \begin{array}{lll} -1,\,0,\,0 & \text{with weight} & 1/2 \\ 1,\,0,\,0 & \text{with weight} & 1/2\,, \end{array} \qquad \nu = \begin{array}{lll} 0,\,0,\,-1 & \text{with weight} & 1/2 \\ 0,\,0,\,\;\;1 & \text{with weight} & 1/2\,. \end{array}$$

Approximation using SDD constraints performs similarly.

The iterative change-of-basis failed to return a result.

Example on a ball set

The sets are $\mathcal{X} = \mathbb{B}^3 = \mathcal{Y}$, where $\mathbb{B}^n = \{x \in \mathbb{R}^n \mid \|x\| \leq 1\}$ is a hypersphere with the defining polynomial $g = 1 - x^\intercal x$.

1. Let

$$p(x, y) = x_1^2 * y_1 + 2 * x_2^2 * y_2 + 3 * x_3^2 * y_3 - x_1 - x_2 - x_3.$$

Our results agree with those of Nie et al., 2018. The hierarchy converges after a single iteration. The returned strategies are:

$$\overset{\star}{\mu} = \; 0.688,\, 0.546,\, 0.477 \;\text{ w. }\; 1\,; \qquad \overset{\star}{\nu} = \; 0.726,\, 0.458,\, 0.349 \;\text{ w. }\; 1$$

which result in a value of the game of $\alpha = -0.77$. The run-time was, again, less than a tenth of a second.

DSOS and SDSOS polynomials Relaxation using DSOS polynomials results in the same value of the game and the same strategies for player 1, but the extraction of strategies fails for player 2. SDSOS polynomials result in the correct answer in both cases.

Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD constraints approximates the value of the game at -1.5, but fails to extract strategies.

Approximation using SDD constraints also fails to return strategies, but approximates the value of the game at -1.1.

The iterative change-of-basis failed to return a result.

The remaining examples from Nie et al. (2018) are not applicable as their sets are not compact.

A generated game with a prescribed solution

The sets are intervals $\mathcal{X} = [-1, 1]$ and $\mathcal{Y} = [-1, 1]$.

1. Consider a game generated using the Gale and Gross equation 5.1, with prescribes strategies $\mu = 0.2$ with weight 1, and $\nu = -0.3$ with weight 1. The resulting polynomial has degree 12.

   After two iterations the resulting solutions are optimal

   $$\overset{\star}{\mu} = \ 0.1997 \ \text{ with weight } \ 1 \ ; \qquad \overset{\star}{\nu} = \ -0.2987 \ \text{ with weight } \ 1$$

   which result in a value of the game of $\alpha = 0$. The run-time was, again, less than a tenth of a second.

   DSOS and SDSOS polynomials Relaxation using DSOS polynomials results in the approximate value of 0.06 and the strategies

   $$\mu' = \begin{matrix} 0.2 \ \text{ with weight } \ 0.57 \\ -1 \ \text{ with weight } \ 0.42 \end{matrix} ; \qquad \nu' = \begin{matrix} -0.3 \ \text{ with weight } \ 0.33 \\ 1 \ \text{ with weight } \ 0.67 \end{matrix}$$

   SDSOS polynomials improve the approximation of the value of the game to 0.01 and the strategies to:

   $$\mu' = \begin{matrix} 0.2 \ \text{ with weight } \ 0.85 \\ -1 \ \text{ with weight } \ 0.15 \end{matrix} ; \qquad \nu' = \begin{matrix} -0.3 \ \text{ with weight } \ 0.7 \\ 1 \ \text{ with weight } \ 0.3 \end{matrix}$$

   Approximating each PSD constraint Inner-approximation of PSD constrained matrices by DD and SDD constraints returns the approximate value of the game 0.05. The strategies of player 2 are incorrectly centered around 0 in both cases, while the extraction of the strategy of player 1 fails altogether.

   Iterative change-of-basis fails due to numerical issues.

5.2 The Double Oracle Algorithm

This section demonstrates the speed of convergence of the continuous Double Oracle Algorithm 1 on polynomial games. Our implementation uses the *Julia*

language, its optimization modeling framework *JuMP*, and the extension for polynomial optimization *PolyJuMP*. The problem setup is equivalent to that of Parrilo's paper (Parrilo, 2007), and we borrow all of his examples.

We apply no post-processing to the output of our algorithm, except in the case of visualization, where we apply a convolution with a bell curve, purely to aid readability.

We have tested multiple "oracles", including stochastic search algorithms (simulated annealing), simplicial homology global optimization, sum-of-squares optimization, and local non-linear optimization via the interior point method. In the following examples, we mainly show the performance of the Sum-of-Squares oracle. Only note that NLP is much faster and has equivalent accuracy unless it gets stuck in a local optimum.

- The payoff function is a polynomial $p(x, y) = 2xy^2 - x^2 - y$ convex in x. Due to its convexity, it has only pure strategy solutions. player 1 is expected to play 0.3969, while player 2 should play 0.6299. The resulting expected value of the game is -0.4725.

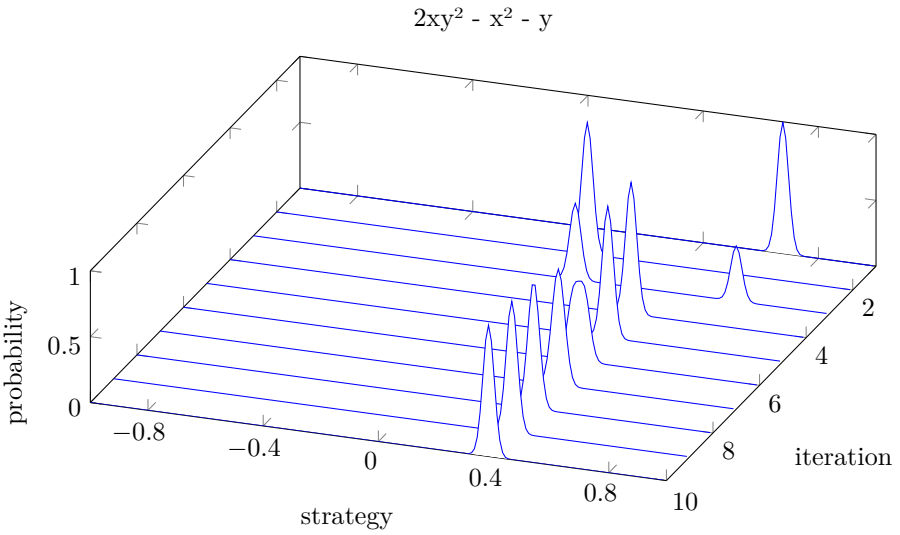


Figure 5.1: Example 5.2. Graph showing the convergence of player 1's best-response strategies on $2xy^2 - x^2 - y$, using the Double Oracle algorithm over the course of 10 iterations. The SOS optimization algorithm (SOS opt.) was used as an oracle.

Using an SOS algorithm (SOS opt.) as an oracle, the algorithm starts converging during the 4th iteration. After 10 iterations, the actual strategies are: player 1 plays 0.3955 with probability 51.6 %, and 0.3985 with probability 48.4 %; while player 2 plays 0.6321 with 57.22%, and 0.6274 with 42.78% probability. The actual value of the game is -0.4725.

- The payoff function is a nonconvex polynomial $p(x, y) = 5xy - 2x^2 - 2xy^2 - y$. player 1 is expected to play 0.2, while player 2 should play -1

with probability 22 % and 1 with probability 78 %. The resulting expected value of the game is −0.48.

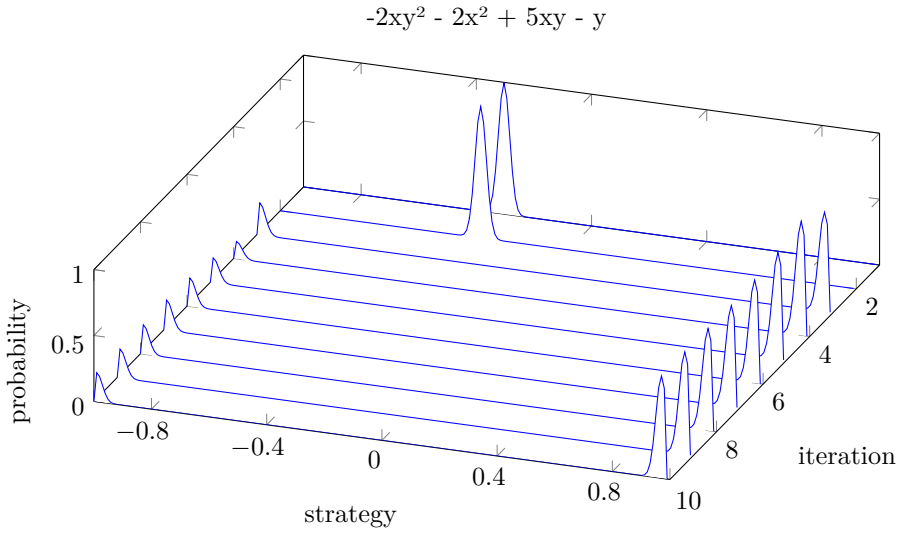


Figure 5.2: Example 5.2. Graph showing the convergence of player 2's best-response strategies on $5xy - 2x^2 - 2xy^2 - y$, using the Double Oracle algorithm over the course of 10 iterations. The SOS optimization algorithm (SOS opt.) was used as an oracle.

Using an SOS algorithm (SOS opt.) as an oracle, the algorithm starts converging during the 3rd iteration. After 10 iterations, the actual strategies are: player 1 plays 0.1925 with probability 14.29 %, and 0.2013 with probability 85.71 %; while player 2 plays −1 with 22.12%, and 1 with 77.88%. The actual value of the game is −0.48.

- The payoff function is a simple "guessing game" defined by the polynomial $p(x, y) = (x - y)^2$. player 1 should play −1 and 1 with equal probability, while player 2 is expected to play 0. The resulting expected value of the game is 1.

  This example appears to be more difficult for the Double Oracle algorithm than others. The choice of player 1 oscillates around the optimum as player 2 tries to guess his strategy. The algorithm can also run into numerical problems when both players begin with the same initial support. By using random initialization, this eventuality is unlikely.

  After 10 iterations, the actual strategies are: player 1 plays −1 with probability 49.84 %, and 1 with probability 50.16 %; while player 2 plays 0.0125 with 60 %, and −0.0188 with 40% probability. The actual value of the game is 1.

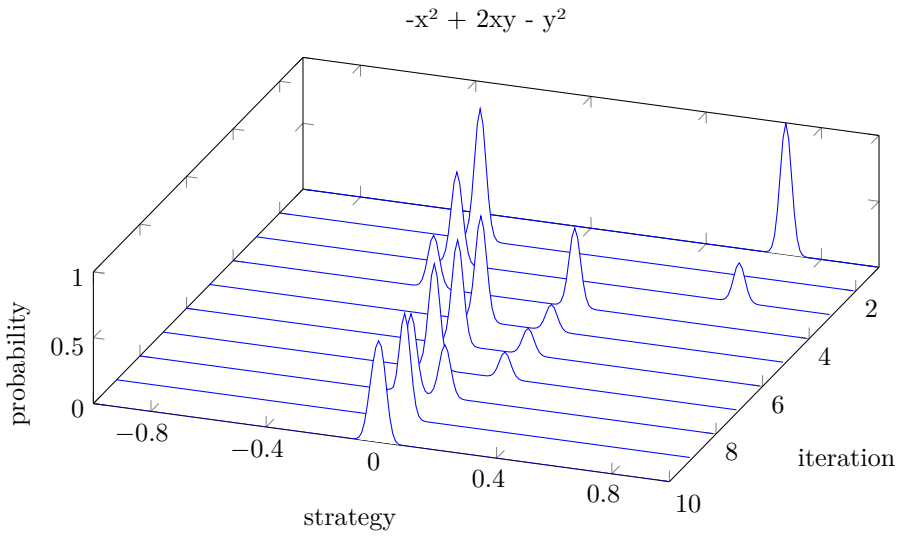


Figure 5.3: Example 5.2. Graph showing the convergence of player 2's best-response strategies on $(x-y)^2$ ("guessing game"), using the Double Oracle algorithm over the course of 10 iterations. The SOS optimization algorithm (SOS opt.) was used as an oracle.

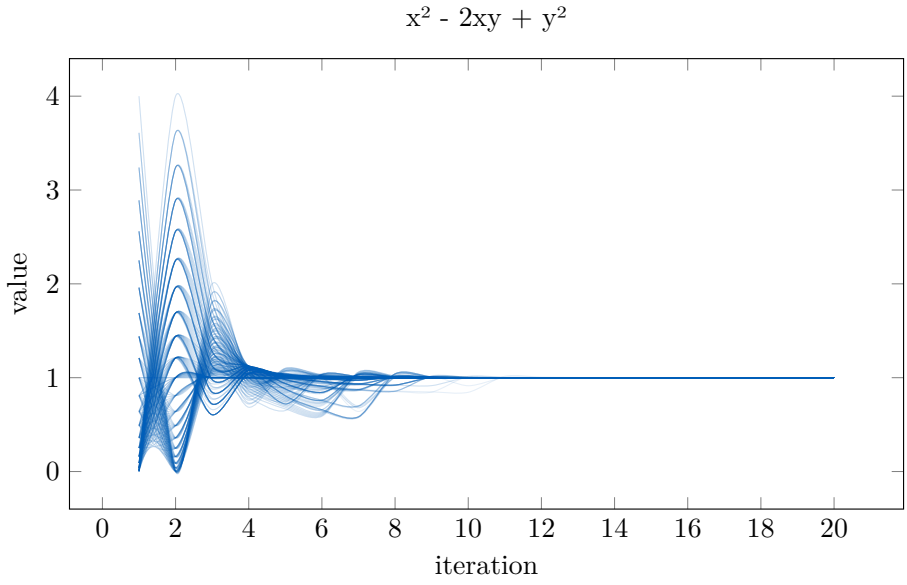


Figure 5.4: Example 5.2. Graph showing the convergence to the optimal value of the ("guessing game"). The lines follow the executions of the Double Oracle algorithm for each of 411 initial guesses on $[-1,1]\times[-1,1]$, from a grid with a 0.1 step. The SOS optimization algorithm (SOS opt.) was used as an oracle.

Two semialgebraic examples

If we use the SOS hierarchy as an oracle, we can solve semialgebraic games as well. Take the third example on simplices. The sets are $\mathcal{X} = \Delta^n$ and $\mathcal{Y} = \Delta^m$.

- Let $n = m = 4$, and

$$p(x, y) = \sum_{i,j=1}^{4} x_i^2 y_j^2 - \sum_{i \neq j} (x_i x_j + y_i y_j)$$

After 20 iterations, we get the results

$$\mu = \begin{array}{l} 0.23,\ 0.26,\ 0.26,\ 0.25 \ \text{ with weight } \ 0.0, \\ 0.25,\ 0.25,\ 0.26,\ 0.24 \ \text{ with weight } \ 0.1, \\ 0.25,\ 0.26,\ 0.25,\ 0.25 \ \text{ with weight } \ 0.1, \\ 0.25,\ 0.25,\ 0.25,\ 0.25 \ \text{ with weight } \ 0.6, \end{array}$$

and

$$\nu = \begin{array}{l} 1,\ 0,\ 0,\ 0 \ \text{ with weight } \ 0.27 \\ 0,\ 1,\ 0,\ 0 \ \text{ with weight } \ 0.23 \\ 0,\ 0,\ 1,\ 0 \ \text{ with weight } \ 0.23 \\ 0,\ 0,\ 0,\ 1 \ \text{ with weight } \ 0.25 \end{array},$$

which result in a value of the game of $\alpha = 0.688$.

- Let $n = m = 3$, and

$$p(x, y) = x_1 x_2 y_1 y_2 + x_2 x_3 y_2 y_3 + x_3 x_1 y_3 y_1 - x_1^2 y_3^2 - x_2^2 y_1^2 - x_3^2 y_3^2.$$

In this case, the original paper found no saddle points, optimal mixed-strategies exist regardless. Our strategies converge at second iteration, and the value is exact on first iteration:

$$\mu = \begin{array}{l} 1,\ 0,\ 0 \ \text{ with weight } \ 1/3 \\ 0,\ 1,\ 0 \ \text{ with weight } \ 1/3 \\ 0,\ 0,\ 1 \ \text{ with weight } \ 1/3 \end{array}, \quad \nu = \begin{array}{l} 0.34,\ 0.33,\ 0.33 \ \text{ with weight } \ 0.49 \\ 0.33,\ 0.34,\ 0.33 \ \text{ with weight } \ 0.22 \\ 0.33,\ 0.33,\ 0.33 \ \text{ with weight } \ 0.29 \end{array}$$

which result in a value of the game of $\alpha = -0.11$.

A generated game with a prescribed solution

The sets are intervals $\mathcal{X} = [-1, 1]$ and $\mathcal{Y} = [-1, 1]$.

1. Consider a game generated using the Gale and Gross equation 5.1, with prescribes strategies $\mu = 0.2$ with weight 1, and $\nu = -0.3$ with weight 1. The resulting polynomial has degree 12.

   Using the local optimizer *Ipopt*, we obtain the optimal answer during the second iteration.

$$\overset{\star}{\mu} = \ 0.2 \ \text{ w. } \ 1\ ; \qquad \overset{\star}{\nu} = \ -0.3 \ \text{ w. } \ 1$$

   which result in a value of the game of $\alpha = 0$. The run-time was around a tenth of a second.

Polynomial games with prescribed unique solutions

We use a simplified algorithm by Gale and Gross (1958) to generate games with known unique equilibria.

Theorem 4. *If $\mathcal{X}$ and $\mathcal{Y}$ are algebraically-compact subsets of $\mathbb{R}^n$, then for any finitely-atomic measures μ and ν there exists a polynomial p, such that the associated game has μ and ν as its unique equilibrium.*

Let $\mathcal{X}$ and $\mathcal{Y}$ be compact basic semialgebraic sets (5) defining the strategy spaces of each player. Let μ on $\mathcal{X}$ and ν on $\mathcal{Y}$ be the optimal strategies, with atoms n and m atoms respectively. Then define the following functions:

$$f(x,\mu) = \prod_{a\in a(\mu)} |\, x-a\,|^2,$$
$$f_i(x,\mu) = \prod_{\substack{a\in a(\mu)\\ a\neq a(\mu)_i}} \frac{|\, x-a\,|^2}{|\, a(\mu)_i - a\,|^2},$$

where $a(\mu)$ are the atoms of μ and $m = length(a(\mu))$; Finally, let $r(\mathcal{X})$ and $r(\mathcal{Y})$ be $m+1$ and $n+1$ distinct points (which do not overlap with the atoms) in each set, and define:

$$\phi(x,\mathcal{X}) = \prod_{a\in r(\mathcal{X})} |\, x-a\,|^2,$$
$$\phi_i(x,\mathcal{X}) = \prod_{\substack{a\in r(\mathcal{X})\\ a\neq r(\mathcal{X})_{i+1}}} |\, x-a\,|^2,$$

The polynomial p solving Theorem 4 is then defined as

$$\begin{aligned} p = {} & f(x,\mu)\phi(x,\mathcal{X})\Big(f(y,\nu)\phi_{m+1}(x,\mathcal{X}) + \textstyle\sum_{j=1}^{n}\big(f_j(y,\nu) - w(\nu)_j\big)\phi_j(x,\mathcal{X})\Big) \\ & - f(y,\nu)\phi(y,\mathcal{Y})\Big(f(x,\mu)\phi_{n+1}(y,\mathcal{Y}) + \textstyle\sum_{j=1}^{m}\big(f_j(x,\mu) - w(\mu)_j\big)\phi_j(y,\mathcal{Y})\Big) \\ & - \big(f(x,\mu)\phi(x,\mathcal{X})\big)^2 + \big(f(y,\nu)\phi(y,\mathcal{Y})\big)^2 \end{aligned} \tag{5.1}$$

where $w(\mu)$ are the weights of atoms $a(\mu)$ of μ.

Disadvantages of this approach The games generated by this equation are larger than necessary. For example, given a game on $[-1,\,1]\times[-1,\,1]$ with solutions $\mu = [(100\%, 0.0)]$, and $\nu = [(50\%, -1.0), (50\%, 1.0)]$, the following polynomial would suffice

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

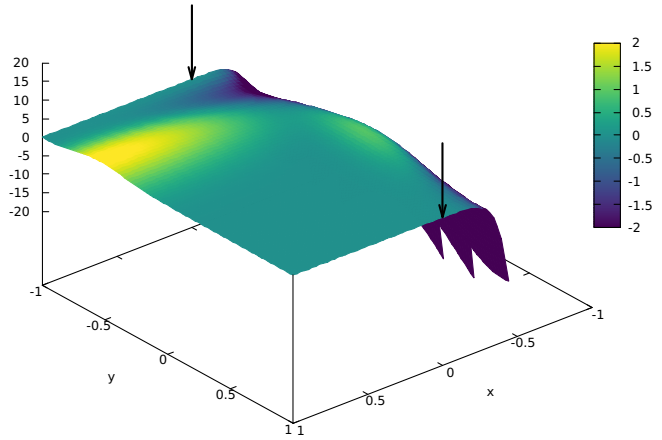


Figure 5.5: A 16th degree game generated by equation 5.1 with an equilibrium at $x = 0$ and $y = -1$ with probability 50 % and $y = 1$ with probability 50 % indicated by an arrow.

as polynomials of second degree can have two roots. However, the algorithm by Gale and Gross generates a polynomial of 16th degree:

$$
\begin{aligned}
p(x,y) = {} & -x^{16} + x^{12}y^4 + y^{16} + 13.084x^{15} - 10.437x^{11}y^4 - 6.187y^{15} - 79.592x^{14} \\
& - 1.5x^{12}y^2 + 49.569x^{10}y^4 + 12.674y^{14} + 298.609x^{13} + 16.306x^{11}y^2 - 141.526x^9y^4 \\
& - 0.817y^{13} - 772.258x^{12} + 0.002x^{11}y - 80.297x^{10}y^2 + 270.26x^8y^4 - x^2y^{10} \\
& - 36.307y^{12} + 1455.555x^{11} - 0.02x^{10}y + 236.739x^9y^2 - 363.19x^7y^4 + 4.37x^2y^9 \\
& + 2.0xy^{10} + 50.314y^{11} - 2056.646x^{10} + 0.072x^9y - 465.094x^8y^2 + 351.653x^6y^4 \\
& - 5.91x^2y^8 - 8.741xy^9 + 3.094y^{10} + 2199.351x^9 - 0.153x^8y + 640.801x^7y^2 \\
& - 246.716x^5y^4 - 1.151x^2y^7 + 11.819xy^8 - 66.18y^9 - 1764.811x^8 + 0.214x^7y \\
& - 634.091x^6y^2 + 124.202x^4y^4 + 10.745x^2y^6 + 2.301xy^7 + 52.415y^8 + 1025.39x^7 \\
& - 0.204x^6y + 453.314x^5y^2 - 43.635x^3y^4 - 9.649x^2y^5 - 21.489xy^6 + 10.799y^7 \\
& - 388.975x^6 + 0.135x^5y - 231.898x^4y^2 + 10.224x^2y^4 + 19.299xy^5 - 36.949y^6 \\
& + 58.331x^5 - 0.062x^4y + 82.576x^3y^2 + 5.269x^2y^3 - 1.591xy^4 + 16.846y^5 \\
& + 29.33x^4 + 0.019x^3y - 23.17x^2y^2 - 10.538xy^3 + 2.827y^4 - 22.165x^3 + 1.157x^2y \\
& + 10.278xy^2 - 4.73y^3 + 6.662x^2 - 2.32xy + 1.19y^2 - 0.8x - 0.046y + 0.047
\end{aligned}
$$

For this reason, we do not show the concrete polynomials generated for our test examples. The high, redundant degree, combined with the relative “flatness” of the generated polynomials, causes numerical problems in some solvers.

5.3 Conclusion

While DSOS and SDSOS optimization can speed up polynomial optimization, the application to semialgebraic games appears limited. Diagonally-dominant and Scaled-diagonally-dominant constraints on PSD matrices often resulted in poor approximations or even infeasibility. Attempts to improve the approximations using the change-of-basis method required the use of an excessively large number of constraints. They failed to produce either faster-to-solve problems or reasonably accurate solutions to most examples. This, combined with the requirement on a specific SDP formulation, makes the method impractical.

On the other hand, the "Double oracle" algorithm is trivial to implement, and was able to converge to the equilibrium in only a few iterations. The algorithm's ultimate practicality depends on the choice of the "oracles"; however, in our experience, even stochastic methods result in good approximations.

Appendix A

Code

The complete source code of our implementation is available online (the Julia solver for semialgebraic games and the remaining algorithms and this thesis). This section explains some of the more important parts of our code.

A.1 Julia

Our solver for semialgebraic games is implemented in Julia[1] using the JuMP modeling framework, and SumOfSquares.jl, the *JuMP* extension for Sum-of-Squares programming.

Problems in *JuMP* are modeled roughly as follows:

```
model = Model(Clp.optimizer)    # wrap an optimizer in an adapter model

@variable model x[1:2]          # create a variable of size 2
@objective model Min sum.(x)    # objective is to minimize the sum of x
@constraint model x .>= 0       # constrain x to be nonnegative

optimize!(model)                # solve the problem
```

The `Model` is actually a complex hierarchy of caches and adapters intended to transform the internal DSL model into a formulation accepted by the optimizer.

Instead of the more straight-forward approach of reformulating the problems into various "standard" forms (conic or otherwise), since the adoption of MathOptInterface, *JuMP* takes a different approach. The internal model is now a completely generic list of $X \in$ AbstractSet constraints. Solvers only specify types of constraints they accept, and a network of *JuMP bridges* attempts to reformulate the problem into that form. This means, incidentally, that the internal problem representation is now an implementation detail, and is inaccessible by the user. This is unfortunate for the iterative DSOS and SDSOS implementations, as it means that we have to implement a complete model.

Formulating a semialgebraic game The implementation of a solver for semialgebraic games is straight-forward. *SumOfSquares.jl* can already formulate

[1] For anyone unfamiliar with the *Julia* language, we only have three remarks. Firstly, arrays are 1-indexed by default; Secondly, functions are broadcast using the "dot-notation": $f.([a\ b\ c])$ means $[f(a)\ f(b)\ f(c)]$. Thirdly, *Julia* has implicit *return*.

constraints necessary for a nonnegative polynomial on a semialgebraic set. It remains only to create the constraints for the dual (*moment*) problem, and to combine the constraints.

```
function solve_game(p::AbstractPolynomial, Sx::AbstractSemialgebraicSet,
        Sy::AbstractSemialgebraicSet, optimizer; iteration::Integer = 0)

        order = min_order(p, Sx, Sy) + iteration * 2        # hierarchy order
        monoms = reverse(monomials(variables(Sy), 0:order))

        m = SOSModel(optimizer)

        @variable  m a                    # upper bound
        @variable  m u[1:length(monoms)] # moments of measure
        @objective m Min a                # objective: minimize upper bound a

        us = measure(u, monoms)

        @constraint(m, c, expect(us, p) <= a, domain=Sx, maxdegree=order)
        @constraint(m, us in MomentSequence(), domain=Sy)     # valid moments
        @constraint(m, u[1] == 1)

        optimize!(m)        # optimize

        sos_atoms = extractatoms(moment_matrix(c), 1e-4)      # extract atoms
        mom_atoms = extractatoms(moment_matrix(us), 1e-4)
        value(a), sos_atoms, mom_atoms
end
```

We formulate the problem as in 3.4. *SumOfSquares.jl* transparently reinterprets the constraint `expect(us, p) <= a` as the appropriate SOS conditions for nonnegativity on the semialgebraic set `Sx`. `min_order` is the smallest order required to build the moment matrices and to compute the expectation of the payoff function p. After optimization, we dualize the *SOS* constraint, create its moment matrix, and attempt to extract the global optimizers. Similarly, we attempt to extract the optimizers from the moment matrix of the measure `us`.

Reformulation into iterative DSOS/SDSOS To solve a semialgebraic game, we need both SOS constraints and moment constraints. For this reason we can not directly apply the DSOS relaxations to each polynomial, and have to use the general method (4.1) instead. This immediately means that the problem requires two passes through *JuMP*: first, to reformulate the abstract problem into the standard SDP trace form

$$\begin{aligned} \min \quad & \langle C, X \rangle \\ \text{subject to} \quad & \langle A_i, X \rangle = b_i \quad \forall i \\ & X \succeq 0 \end{aligned}$$

and then after the modification, a second pass to reformulate it into the standard LP or SOCP form.

We also can not leverage the internal *JuMP* framework of bridges, as the mapping $X \succeq 0 \mapsto X \in DD(U)$ is not a mapping into an equivalent constraint. Forcing the transformation regardless would lead to different results when applied to problems that are not in the standard SDP form.

Instead, we transform the problem into the standard form and then apply the transformation as

```
X = @variable(optimizer, [1:m, 1:m], Symmetric)
@objective optimizer Min tr(C*X)
@constraint(optimizer, [i in 1:n], tr(As[i]*X) == b[i]) # forall A[i]
@constraint(optimizer, X in DD(U))
```

which is exactly the DDP form

$$\begin{aligned} \min \quad & \langle C, X \rangle \\ \text{subject to} \quad & \langle A_i, X \rangle = b_i \quad \forall i \\ & X \in DD(U) \end{aligned}$$

This is only a simplified example. The variable X may contain a large number of nonnegativity constraints encoded as a diagonal block. Since any solver can handle nonnegativity constraints, we can extract them and create a much smaller problem.

Semialgebraic Double Oracle Listing A.1 shows a trivial Julia implementation of the continuous double oracle algorithm for semialgebraic games. To reiterate the idea of the algorithm: we make an initial guess of the supports; from our current guesses, create a matrix game; solve the game; compute the best response against the current strategies and append them to our guesses. The implementation is trivial for the purpose of being easy to follow. We will walk through it nonetheless.

```
function double_oracle(p::Polynomial, domain_x::BasicSemialgebraicSet,
        domain_y::BasicSemialgebraicSet)

        x, y = variables(domain_x), variables(domain_y)

        support_x, prob_x, vx = [], nothing, nothing        # allocation
        support_y, prob_y, vy = [], nothing, nothing

        response_x = oracle(polynomial(x), domain_x)        # initial guess
        response_y = oracle(polynomial(y), domain_y)

        for i = 1:10       # no convergence criteria
                append!(support_x, response_x)
                append!(support_y, response_y)

                matrix = [p(x=>a, y=>b) for a in support_x, b in support_y]
                vx, prob_x = matrix_game( matrix )
                vy, prob_y = matrix_game(-matrix')          # solve game

                loss_x = -prob_y' * [subs(p, y => sy) for sy in support_y]
                loss_y =  prob_x' * [subs(p, x => sx) for sx in support_x]
                response_x = oracle(loss_x, domain_x)
                response_y = oracle(loss_y, domain_y)       # minimize loss
        end

        vx, support_x, prob_x, support_y, prob_y
end
```

Lines 9 and 10 invoke the "oracles" with the polynomials $\sum x_i$. This is simply to obtain a feasible initial solution.

Line 16 is a matrix comprehension which evaluates the payoff polynomial on the cross product of the support guesses and thereby creates a matrix game. The following two lines solve the matrix game.

loss_x and *loss_y* are the loss functions, which are the weighted sums of the polynomial "cross sections". Or in different words, they define the loss weighted by the probability of the opponent playing the corresponding strategy. Finally, the oracles minimize this loss function.

The SOS Oracle The next listing (A.1) shows an implementation of such an oracle, implemented using Lasserre hierarchies. As many frameworks implement Sum-Of-Squares optimization already, we only have to formulate the problem. *SumOfSquares.jl* even transparently interprets the constraint $f \geq lb$ as an SOS constraint for nonnegativity with a bounded degree, on a specified domain.

To extract the optimizers, we dualize to SOS constraint, create its moment matrix, and extract its atoms (for example using procedure 3.5).

```
1  function oracle(f::Polynomial, set::BasicSemialgebraicSet)
2          deg = order_guess(p, set)
3
4          model = SOSModel(CSDP.Optimizer)
5          @variable model lb
6          @objective model Max lb
7          @constraint(model, con, f >= lb, domain = set, maxdegree = deg)
8
9          set_silent(model)
10         optimize!(model)
11
12         measure = extractatoms(moment_matrix(con), 1e-3)         # get atoms
13         [a.center for a in measure.atoms]
14 end
```

Solving a matrix game We use the "standard" formulation (2.1) to solve matrix games using LP. Only two things to note: Firstly, line 6 is equivalent to "Let $p \in \mathbb{R}^m_+$"; Secondly, the operator . >= denotes an elementwise $\geq$ relation.

```
1  function matrix_game(payoff::AbstractArray)
2          m, n = size(payoff)
3
4          model = Model(Clp.Optimizer)               # standard LP formulation
5          @variable model v
6          @variable model p[1:m] >= 0
7          @objective model Max v
8          @constraint(model, payoff' * p .>= v * ones(n))
9          @constraint(model, sum(p) == 1)
10
11         set_silent(model)
12         optimize!(model)
13
14         JuMP.objective_value(model), value.(p)          # value and strategy
15 end
```

Different oracles Of course, using an SOS oracle might not make much sense in practice. Replacing the requirement of semialgebraic strategy spaces with, for example, an interval, allows us to use most local and stochastic methods.

Listing A.1 shows an oracle implemented using the *Ipopt* local optimizer. To use it we need to provide the first and second derivatives of our function. Also, since modeling in *JuMP* is done using macros, we have to register our user-defined function before using it in the objective.

```
function oracle(loss::Polynomial, x::PolyVar, interval::Tuple)
        lp  = differentiate(loss, x)
        lpp = differentiate(lp, x)        # Ipopt requires second derivative

        f(x)   = loss(x)                # Polynomial isn't a Function (°_°>)
        fp(x)  = lp(x)
        fpp(x) = lpp(x)

        model = Model(Ipopt.Optimizer)
        @variable model interval[1] <= v <= interval[2] # constrain variable
        JuMP.register(model, :loss, 1, f, fp, fpp)   # register user defined
        @NLobjective model Min loss(v)

        optimize!(model)

        value(v)
end
```

In exactly the same manner we could have used *simulated annealing*, or *random sampling*, and since we have roughly halved the dimensionality of the problem, perhaps even discretization.

A.2 Matlab

We have also implemented Parrilo's algorithm (3.3) in *YALMIP* (Löfberg, 2004) with the use of the SeDuMi[2] SDP solver. YALMIP already implements SOS constraints on polynomials; however, it does not implement the SOS conditions for nonnegative polynomials on semialgebraic sets. YALMIP also has a built-in module for polynomial optimization using moment relaxations and a procedure for the extraction of global optimizers. These modules can, unfortunately, not be readily combined. We can still abuse the provided features to formulate our problem as follows:

Firstly, given an input polynomial $p(x, y)$, we create a new univariate polynomial $p_x(x)$ such that $deg(p_x) \leq deg(p)$, and create the nonlinear constraint

$$\|\text{coefficients with respect to } x \text{ of } (p - px)\| = 0$$

Then we apply the Putinar's Positivstellensatz (2) on p_x to constrain it to be nonnegative on a semialgebraic set $\mathcal{X} = \{z \in \mathbb{R} \mid g_x(z) \geq 0\}$ given by the polynomial g_x. For this we only need one additional polynomial $s(x)$, such that the degree of $s(x)g_x(x)$ is no more than the degree of p_x.

Similarly, we use the method for constructing a moment matrix 3.1 from vectors of monomials. We create the matrix $[y]^\intercal[y]$ and constrain it to be

[2]SeDuMi: Optimization over symmeric cones, http://sedumi.ie.lehigh.edu/

positive semidefinite. This constraint is not only nonlinear, it is also redundant as any matrix $[y]^\intercal[y]$ is both rank-1 and positive semidefinite by construction.

The final step is to use *YALMIP*'s setting "relax = 3", to treat nonlinear variables as independent variables, that is, y^2 and y^3 are interpreted as the independent variables y_2 and y_3.

We can now call *YALMIP*'s module for solving SOS problems, using the coefficients $\vec{s}$ of $s(x)$ as decision variables (but not the coefficients of p_x, otherwise *YALMIP* can not tell which variables are decision variables, and which are polynomial variables).

```
function [val, mom, pol] = moment_solve(p, x, y, gx, gy)
        % automatic linearization
        settings = sdpsettings('relax', 3);

        % Even degree simplifies positivity on interval
        deg_px = round_up_even(degree(p, x));
        deg_py = round_up_even(degree(p, y));
        deg_gx = degree(gx, x);
        deg_gy = degree(gy, y);
        % hierarchy order
        deg_max_x = max(deg_px, deg_gx);
        deg_max_y = max(deg_py, deg_gy);
        % value of game
        val = sdpvar;

        % mu for moment relaxations; mg reduces degree of interval constraint
        mu = monolist(y, deg_max_y);
        mg = mu(1:end-deg_gy);

        % px is expectation of p; s is sos multiplier.
        [px, cx] = polynomial(x, deg_px);
        [s , cs] = polynomial(x, deg_max_x - deg_gx);

        % moment matrices - half-hankel
        st_mom = [hankh(mu) >= 0, hankh(mg*gy) >= 0];
        st_sos = [coefficients(p - px, x) == 0, sos(val - px - s*gx), sos(s)];

        res = solvesos([st_sos, st_mom], val, settings, cs);

        e1  = eye(deg_px + 1, 1);
        pol = double(val * e1 - cx);
        mom = relaxdouble(mu);
        val = double(val);
end
```

A.3 Python

A Python implementation corresponding to the LP formulation 2.1 for solving a matrix game. The implementation uses *CVXPY* (Diamond and Boyd, 2016, Agrawal et al., 2018). The input is an m-by-n NumPy array, and the outputs are: the value of the game, and the optimal strategy for player x.

```
def strategy(payoff, optimizer):
    m, n = payoff.shape

    e = np.ones(m)
    v = cp.Variable()
    x = cp.Variable(n)

    objective = cp.Maximize(v)
    constraints = [v * e - payoff @ x <= 0, cp.sum(x) == 1, x >= 0]
    cp.Problem(objective, constraints).solve(solver=optimizer)

    return v.value, x.value
```

Note that *CVXPY* interprets the constraints `<=` and `=>` as element-wise.

Bibliography

Agrawal, A., Verschueren, R., Diamond, S., & Boyd, S. (2018). A rewriting system for convex optimization problems. *Journal of Control and Decision*, *5*(1), 42–60.

Ahmadi, A., & Majumdar, A. (2017). Dsos and sdsos optimization: More tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebra and Geometry*, *3*. https://doi.org/10.1137/18M118935X

Ahmadi, A. A., & Hall, G. (2015). Sum of squares basis pursuit with linear and second order cone programming.

Choi, M.-D., Lam, T., & Reznick, B. (1994). Sums of squares of real polynomials. *Proceedings of Symposia in Pure Mathematics*, *58*. https://doi.org/10.1090/pspum/058.2/1327293

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

Gale, D., & Gross, O. (1958). A note on polynomial and separable games. *Pacific J. Math.*, *8*(4), 735–741. https://projecteuclid.org:443/euclid.pjm/1103039699

Henrion, D., & Lasserre, J.-B. (2005). Detecting global optimality and extracting solutions in gloptipoly. https://doi.org/10.1007/10997703_15

Karlin, S., & Shapley, L. (1972). *Geometry of moment spaces*. American Mathematical Society. https://books.google.cz/books?id=LrLwtAEACAAJ

Kjeldsen, T. (2001). John von neumann's conception of the minimax theorem: A journey through different mathematical contexts. *Archive for History of Exact Sciences*, *56*, 39–68. https://doi.org/10.1007/s004070100041

Lasserre, J.-B. (2004). Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, *11*. https://doi.org/10.1137/S1052623400366802

Löfberg, J. (2004). Yalmip : A toolbox for modeling and optimization in matlab. *In Proceedings of the CACSD Conference*.

Majumdar, A., Hall, G., & Ahmadi, A. A. (2019). A survey of recent scalability improvements for semidefinite programming with applications in machine learning, control, and robotics.

McMahan, H., Gordon, G., & Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary., 536–543.

Nie, J., Yang, Z., & Zhou, G. (2018). The saddle point problem of polynomials.

Parrilo, P. (2007). Polynomial games and sum of squares optimization, 2855–2860. https://doi.org/10.1109/CDC.2006.377261

Putinar, M. (1993). Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, *42*(3), 969–984. http://www.jstor.org/stable/24897130

Szegő, G. (1939). *Orthogonal polynomials*. American Mathematical Society.

Attachments

Attached to this thesis are the following items:

1. A *Julia* solver for semialgebraic games implemented using Algorithm 3.4 and extended by generic implementations of Algorithms 4.1 and 4.2 based on iterative change-of-basis.
2. Two trivial implementations of the Double Oracle Algorithm 1 in *Julia*.
3. A *YALMIP* solver for polynomial games on intervals implemented using Algorithm 3.3.
4. The source code of this thesis,

The items above are also accessible on Github and Gitlab (The Julia solver for semialgebraic games[3] and the remaining algorithms and this thesis[4]).

[3] https://github.com/votroto/SemialgebraicGamesCVUT.jl
[4] https://gitlab.fel.cvut.cz/votroto1/sdsos