SOS Lecture 1: Introduction / Boaz Barak

Admin

Reading Sections 1 and 2 of my survey with Steurer ”Sum of Squares proofs and the quest toward optimal algorithms”

Additional reading Introduction of Ryan O’Donnell and Yuan Zhou’s paper ”Approximability and Proof Complexity”

Lecture notes of Monique Laurent (`https://sites.google.com/site/mastermathsdp/lectures`) and Pablo Parrilo (`http://stellar.mit.edu/S/course/6/sp14/6.256/materials.html`)

Disclaimer I haven’t tried to solve most of the exercises myself, and it could be that you’ll run into various inaccuracies or issues while trying to solve them, let me know if you do. Also, all historical discussions and references are from memory or second-hand sources, and not based on the original texts, and so may be inaccurate.

Prelude

Consider the following questions:

1. Do we need a different algorithm to solve every computational problem, or can a single algorithm give the best performance for a large class of problems?

2. In statistical physics and other areas, many people believe in the existence of a *computational threshold* effect, where a small change in the parameters of a computational problems seems to lead to a huge change in its computational complexity. Can we give rigorous evidence for this intuition?

3. In machine learning there often seem to be tradeoffs between sample complexity, error probability, and computation time. Is there a way to map the curve of this tradeoff?

4. Suppose you are given a 3SAT formula φ with a unique (but unknown) satisfying assignment x. Is there a way to make sense of statements such as ”The probability that $x_{17} = 1$ is 0.6” or ”The entropy of x is 1000”? (even though of course information theoretically x is completely determined by φ, and hence that probability is either 0 or 1 and x has zero entropy).

5. Is Khot’s Unique Games Conjecture true?

If you learn the answers to these questions by the end of this seminar series, then I hope you’ll explain them to me, since I definitely don’t know them. However we will see that, despite these questions a-priori having nothing to do with Sums of Squares, that the SOS algorithm can yield a powerful lens to shed light on some of those questions, and perhaps be a step towards providing some of their answers.

1 Introduction

Theoretical computer science studies many computational models for different goals. There are some models, such as bounded-depth (i.e. AC_0) circuits, that we can prove unconditional lower bounds on, but do not aim to capture all relevant algorithmic techniques for a given problem. (For example, we don't view the results of Furst-Sax-Sipser and Håstad as evidence that computing the parity of n bits is a hard problem.) Other models, such as bilinear circuits for matrix multiplication, are believed to be strong enough to capture all known algorithmic techniques for some problems, but then we often can't prove lower bounds on them.

The *Sum of Squares (SOS)* algorithm (discovered independently by researchers from different communities including Shor, Parrilo, Nesterov and Lasserre) can be thought of as another example of a concrete computational model. On one hand, it is sufficiently weak for us to know at least some unconditional lower bounds for it. In fact, there is a sense that it is weaker than AC_0, since for a given problem and input length, SOS is a *single algorithm* (as opposed to an exponential-sized family of circuits). Despite this fact, proving lower bounds for SOS is by no means trivial, even for a single distribution or instances (such as a random graph) or even a single instance. On the other hand, while this deserves more investigation, it does seem that for many interesting problems, SOS does encapsulate all the algorithmic techniques we are aware of, and that there is some hope that SOS is an *optimal algorithm* for some interesting family of problems, in the sense that no other algorithm with similar efficiency can beat SOS's performance on these problems.

The possibility of the existence of such an *optimal algorithm* is very exciting. Even if at the moment we can't hope to prove its optimality unconditionally, this means that we can (modulo some conjectures) reduce analyzing the difficulty of a problem to analyzing a single algorithm, and this has several important implications. For starters, it reduces the need for creativity in designing the algorithm, making it required only for the algorithm's *analysis*. In some sense, much of the progress in science can be described as attempting to automate and make routine and even boring what was once challenging. Just as we today view as commonplace calculations that past geniuses such as Euler and Gauss spent much of their time on, it is possible that in the future much of algorithm design, which now requires an amazing amount of creativity, would be systematized and made routine. Another application of optimality is automating *hardness results*— if we prove the optimal algorithm *can't* solve a problem X then that means that X can't be solved by any efficient algorithm.

But beyond just systematizing what we already can do, optimal algorithms could yield qualitative new insights on algorithms and complexity. For example, in many problems arising in statistical physics and machine learning, researchers believe that there exist *computational phase transitions*— where a small change in the parameter of a problems causes a huge jump in its computational complexity. Understanding this phase transitions is of great interest for both researchers in these areas and theoretical computer scientists. The problem is that these problems involve *random inputs* (i.e., *average case complexity*) and so, based on current state of art, we have no way of proving the existence of such phase transitions based on assumptions such as $\mathbf{P} \neq \mathbf{NP}$. In some cases, such as the planted clique problem, the problem has been so well studied that the existence of a computational phase transition had been proposed as a conjecture in its own right, but we don't know of good ways to reduce such reductions to one another, and we clearly don't want to have as many conjectures as there are problems. If we assume that an algorithm is optimal for a class of problems, then we can prove a computational phase transition by analyzing the running time of this algorithm as a function of the parameters. While by no means trivial, this is a tractable approach to understanding this question and getting very precise estimates as to the location of the threshold where the phase transition occurs. (Note that in some sense the existence of a computational phase transition implies the existence of an optimal algorithm, since in particular it means that there is a single algorithm A such that beating A's performance by a little bit requires an algorithm taking much more resources.)

Beyond all that, an optimal algorithm gives us a new understanding of just what is it about a problem that makes it easy or hard, and a new way to look at efficient computation. I don't find explanations such as "Problem A is easy because it has an algorithm" or "Problem B is hard because it has a reduction from SAT" very satisfying. I'd rather get an explanation such as "Problem A is easy because it has property P" and "Problem B is hard because it doesn't have P" where P is some meaningful property (e.g., being convex, supporting some polymorphisms, etc..) such that every problem (in some domain) with P is easy and every problem without it is hard. For that, we would want an algorithm that will solve all problems in P and a proof (or other evidence) that it is optimal. Such understanding of computation could bear other fruits as well. For example, as we will see in this seminar series, if the SOS algorithm is optimal in a certain domain, then we can use this to build a theory of "computational Bayesian reasoning" that can capture the "computational beliefs" of a bounded-time agent about a certain quantity, just as traditional Bayesian reasoning captures the beliefs of an unbounded-time agent about quantities on which it is given partial information.

I should note that while much of this course is very specific to the SOS algorithms, not all of it is, and it is possible that even if the SOS algorithm is superseded by another one, some of the ideas and tools we develop will still be useful. Also, note that I have deliberately ignored the question of *what* family of problems would the SOS be optimal for. This is clearly a crucial issue— every computational model (even AC_0) is optimal for *some* problems, and every model falling short of general polynomial-time Turing machines would not be optimal for *all* problems. It definitely seems that some algebraic problems, such as integer factoring, have very special structure that makes it hard to conjecture that any generic algorithm (and definitely not the SOS algorithm) would be optimal for them. (See also my blog post `http://wp.me/p2bJCi-Gp` on this topic.) The reason I don't discuss this issue is that we still don't have a good answer for it, and one of the research goals in this area is to understand what should be the right *conjecture* about optimality of SOS. However, we do have some partial evidence and intuition, including those arising from the SOS algorithm's complex (and not yet fully determined) relation to Khot's Unique Games Conjecture, that leads us to believe that SOS could be an optimal algorithm for a non-trivial and interesting class of problems.

In this course we will see:

1. A description of the SOS algorithm from different viewpoints— the traditional semidefinite programming/convex optimization view, as well as the proof system view, and the "pseudo-distribution" view.

2. Discussion of positive results (aka "upper bounds") using the SOS algorithms to solve graph problems such as sparsest cut, and problems in machine learning.

3. Discussion of known negative results (aka "lower bounds" / "integrality gaps") for this algorithm.

4. Discussion of the interesting (and not yet fully understood) relation of the SOS algorithm to Khot's *Unique Games Conjecture* (UGC). On one hand, implies that the SOS algorithm is *optimal* for a large class of problems. On the other hand, the SOS algorithm is currently the main candidate to refute the UGC.

2 Polynomial optimization

The SOS algorithm is an algorithm for solving a computational problem. Let us now define what this problem is:

Definition 1. A *polynomial equation* is an equation of the form $\{P(x) \geq 0\}$ (in which case it is called an *inequality*) or an equation of the form $\{P(x) = 0\}$ (in which case it is called an *equality*) where P is a multivariate polynomial mapping $x \in \mathbb{R}^n$ to $\mathbb{R}$. The equation $\{P(x) \geq 0\}$ (resp. $\{P(x) = 0\}$) is *satisfied* by $x \in \mathbb{R}^n$ if $P(x) \geq 0$ (resp. $P(x) = 0$).

A set $\mathcal{E}$ of polynomial equations is *satisfiable* if there exists an x that satisfies all equations in $\mathcal{E}$.

The *polynomial optimization* problem is to output, given a set $\mathcal{E}$ of polynomial equations as input, either an x satisfying all equations in $\mathcal{E}$ or a proof that $\mathcal{E}$ is unsatisfiable.

(Note: throughout this seminar we will ignore all issues of numerical accuracy— assume the polynomials always have rational coefficients with bounded numerator and denominator, and all equalities/inequalities can be satisfied up to some small error $\epsilon > 0$.)

Here are some examples for polynomial optimization problems:

Linear programming If all the polynomials are *linear* then this is of course linear programming that can be done in polynomial time.

Least squares If the equations consist of a single quadratic then this is the least squares algorithm. Similarly, one can capture computing eigenvalues by two quadratics.

3SAT Can encode 3SAT formula as degree 3 polynomial equations: the equation $x_i^2 = x_i$ is equivalent to $x_i \in \{0,1\}$. The equation $x_i x_j(1 - x_k) = 0$ is equivalent to $\overline{x_i \wedge x_j \wedge \overline{x_k}} = \overline{x_i} \vee \overline{x_j} \vee x_k$.

Clique Given a graph $G = (V, E)$ the following equations encode that x is a $0/1$ indicator vector of a k-clique: $x_i^2 = x_i$, $\sum x_i = k$, $x_i x_j = 0$ for all $(i,j) \notin E$.

Other examples Learning, etc...

The SOS algorithm is designed to solve the polynomial optimization problem. As we can see from these examples, the full polynomial optimization problem is NP hard, and hence we can't expect SOS (or any other algorithm) to efficiently solve it on every instance. (**Exercise 1:** prove that this is the case even if all polynomials are quadratic, i.e. of degree at most 2.) Understanding how close the SOS algorithm gets in particular cases is the main technical challenge we will be dealing with.

These examples also show that polynomial optimziation is an extremely versatile formalism, and many other computational problems (including SAT and CLIQUE) can be directly and easily phrased as instances of it. Henceforth we will ignore the question of *how* to formalize a problem as a polynomial optimization, and either assume the problem is already given in this form, or use the simplest most straightforward translation if it isn't. While there are examples where choosing between different natural formulations could make a difference in complexity, this is not the case (to my knowledge) in the questions we will look at.

Note: We can always assume without loss of generality that all our equations are *equalities*, since we can always replace the equation $P(x) \geq 0$ by $P(x) - y^2 = 0$ where y is some new auxiliary variable. Also, we sometimes will ask the question of minimizing (or maximizing) a polynomial $P(x)$ subject to x satisfying equations $\mathcal{E}$, which can be captured by looking for the largest μ such that $\mathcal{E} \cup \{P \geq \mu\}$ is satisfiable.

3 The SOS algorithm

The Sum of Squares algorithm is an algorithm to solve the polynomial optimization problem. Given that it is NP hard, the SOS algorithm cannot run in polynomial time on all instances. The main focus of this course is trying to understand in which cases the SOS algorithm takes a small (say polynomial or quasipolynomial) amount of time, in which cases it takes a large (say exponential) amount. An equivalent form of this question (which is the one we'll mostly use) is that, for some small ℓ (e.g. a constant or logarithmic) we want to understand in which cases the "n^{ℓ}-capped" version of SOS succeeds to solve the problem and in which cases it doesn't, where the "$T(n)$-capped" version of the SOS algorithm halts in time $T(n)$ regardless of whether or not it solved the problem.

In fact, we will see that for every value of d, the SOS of squares always returns some type of a meaningful output. The main technical challenge is to understand whether that output can be transformed to an exact or approximate solution for the polynomial optimization problem.

Definition 2 (Sum of Squares - informal definition)**.** The SOS algorithm gets a parameter ℓ and a set of equations $\mathcal{E}$, runs in time $n^{O(\ell)}$ and outputs either:

- An object we will call a "degree-ℓ pseudo solution" (or more accurately a degree-ℓ *pseudo-distribution* over solutions).

  or

- A proof that a solution doesn't exist.

We will later make this more precise: what is exactly a degree-ℓ pseudo solution, what is exactly the form of the proof, and how does the algorithm work.

History. The SOS algorithm has its roots in questions raised in the late 19*th* century by Minkowski and Hilbert of whether any non-negative polynomial can be represented as a sum of squares of other polynomials. Hilbert realized that except for some special cases (most notably univariate polynomials and quadratic polynomials), the answer is negative and that there is an example (which he constructed by non constructive means) of non-negative polynomial that cannot be represented in this way. It was only in the 1960's that Motzkin gave a very concrete example of such a polynomial

$$1 + x^4y^2 + x^2y^4 3x^2x^2 \tag{1}$$

In his famous 2000 address, Hilbert asked as his 17th problem whether any polynomial can be represented as a sum of squares of *rational* functions. (For example, Motzkin's polynomial (1) can be shown to be the sum of squares of (I think) four rational functions of denominator and numerator degree at most 6). This was answer positively by Artin in 1927. His approach can be summarized as, given a hypothetical polynomial P that cannot be represented in this form, to use the fact that the rational functions are a field to extend the reals into a "pseudo-real" field $\tilde{\mathbb{R}}$ on which there would actually be an element $\tilde{x} \in \tilde{R}$ such that $P(\tilde{x}) < 0$, and then use a "transfer principle" to show that there is an actual real $x \in \mathbb{R}$ such that $P(x) < 0$. (This description is not meant to be understandable but to make you curious enough to look it up..) Later in the 60's and 70's Krivine and Stengle extended this result to show that any unsatisfiable system of polynomial equations can be certified to be unsatisfiable via a Sum of Squares proof, a result known as the Positivstallensatz.

Figure 1: SOS was used to analyze the “falling leaf” mode of the U.S. Navy F/A-18 ”Hornet”, see A. Chakraborty, P. Seiler, and G. J. Balas, Journal of guidance, control, and dynamics, 34(1):7385, 2011

In the late 90’s / early 2000’s, there were two separate efforts on getting quantitative or algorithmic versions of this result. On one hand Grigoriev and Vorobjov asked the question of *how large* the degree of an SOS proof needs to be, and in particular Grigoriev proved several lower bounds on this degree for some interesting polynomials. On the other hand Parrilo and Lasserre (independently) came up with hierarchies of algorithms for polynomial optimization based on the Positivstallensatz using semidefinite programming. (Something along those lines was also described by Naum Shor in a 1987 Russian paper, and mentioned by Nesterov as well.)

It took some time for people to realize the connection between all these works, and in particular the relation between Grigoriev-Vorbjov’s work and the works from the optimization literature took some time to be discovered, and even 10 years after, it was still the case that some results of Grigoriev were rediscovered and reproven in the Lasserre language.

Applications of SOS SOS has applications to: equilibrium analysis of dynamics and control (robotics, flight controls, ...), robust and stochastic optimization, statistics and machine learning, continuous games, software verification, filter design, quantum computation and information, automated theorem proving, packing problems, etc... (For two very different examples, see Figures 1, 2.)

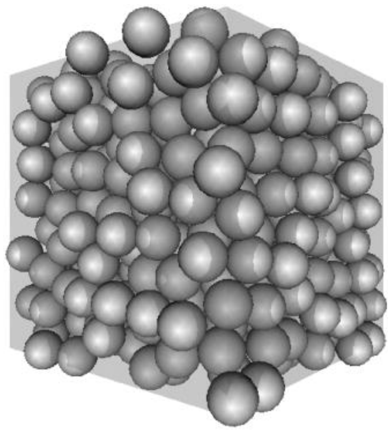

Figure 2: SOS was used to get the best known bounds on the classical "sphere packing" problem. See D. de Laat, F.M. de Oliveira Filho, F. Vallentin, Forum of Mathematics, Sigma, 2 (2014), e23

The TCS vs Mathematical Programming view of SOS

While the SOS algorithm is intensively studied in several communities, there are some differences in emphasizes between the different aspects. While I am not an expert on all SOS works, my impression that the main characteristics of the TCS viewpoint, as opposed to others are:

1. In the TCS world, we typically think of the number of variables n as large and tending to infinity (as it corresponds to our input size), and the degree d of the SOS algorithm as being relatively small— a constant or logarithmic. In contrast, in the optimization and control world, the number of variables can often be very small (e.g. around ten or so, maybe even smaller) and hance d may be large compared to it.

   Note that since both time and space complexity of the general SOS algorithm scale roughly like n^d, even $d = 6$ and $n = 100$ would take something like a petabyte of memory (in practice, though we didn't try to optimize too much, David Steurer and I had a hard time executing a program with $n = 16$ and $d = 4$ on a Cornell cluster). This may justify the optimization/control view of keeping n small, although if we show that SOS yields a polynomial-time algorithm for a particular problem, then we can hope that we would be able to then optimize further and obtain an algorithm that doesn't require a full-fledged SOS solver.

2. Typically in TCS our inputs are discrete and the polynomials are simple, with integer coefficients etc. Often we have constraints such as $x_i^2 = x_i$ that restrict attention to the Boolean cube, and so we are less concerned with issues of numerical accuracy, boundedness, etc..

3. Traditionally people have been concerned with *exact convergence* of the SOS algorithm—- when does it yield an exact solution to the optimization problem. This often precludes d from being much smaller than n. In contrast as TCS'ers we would often want to understand *approximate convergence*— when does the algorithm yield an "approximate" solution (in some problem-dependent sense).

   Since the output of the algorithm in this case is not actually in the form of a solution to the equations, this raises the question of a obtaining *rounding* algorithms, which are procedures to translate the output of the algorithm to an approximate solution.

4 Several views of the SOS algorithm

We now describe the SOS algorithm more formally. For simplicity, we consider the case that the set $\mathcal{E}$ only consists of equalities (which is without loss of generality as we mentioned before). When convenient we will assume all equalities are homogenous polynomials of degree d. (This can be always be arranged by multiplying the constraints.) You can restrict attention to $d = 4$— this will capture all of the main issues of the general case.

4.1 SOS Algorithm: convex optimization view

We start by presenting one view of the SOS algorithm, which technically might be the simplest, though perhaps at first not conceptually insightful.

Definition 3. Let $\mathbb{R}^n_d$ denote the set of n-variate polynomials of degree at most d. Note that this is a linear subspace of dimension roughly n^d.

We will sometimes also write this as $\mathbb{R}[x]_d$ where we want to emphasize that these polynomials take the formal input $x = x_1 \dots x_n$.

Definition 4. Let $\mathcal{E} = \{p_1 = \cdots p_m = 0\}$ be a set of polynomial equations where $p_i \in \mathbb{R}^n_d$ for all i. Let $\ell \in \mathbb{N}$ be some integer multiple of $2d$. The *degree-ℓ SOS algorithm* either outputs `'fail'` or a bilinear operator $M : \mathbb{R}^n_{\ell/2} \times \mathbb{R}^n_{\ell/2} \to \mathbb{R}$ satisfying:

- Normalization: $M(1, 1) = 1$ (where 1 is simply the polynomial $p(x) = 1$).
- Symmetry: If $p, q, r, s \in \mathbb{R}^n_{\ell/2}$ satisfy $pq = rs$ then $M(p, q) = M(r, s)$.
- Non-nonnegativity (positive semi definiteness): For every p, $M(p, p) \geq 0$.
- Feasibility: For every $i \in [m]$, $p \in \mathbb{R}^n_{\ell/2-d}$, $q \in \mathbb{R}^n_{\ell/2}$, $M(p_i p, q) = 0$.

Exercise 2: Show that if the symmetry and feasibility constraints hold for monomials they hold for all polynomials as well.

Exercise 3: Show that the set of M's satisfying the conditions above is convex and has an efficient separation oracle.

Indeed, such an M can be represented as an $n^{\ell/2} \times n^{\ell/2}$ PSD matrix satisfying some linear constraints. (Can you see why?) Thus by semidefinite programming finding such an M if it exists can be done in $n^{O(\ell)}$ time (throughout this seminar we ignore issues of precision etc..). The question is why does this have anything to do with solving our equations, and one answer is given by the following lemma:

Lemma 5. *Suppose that $\mathcal{E}$ is satisfiable. Then there exists an operator M satisfying the conditions above.*

Proof. Let x^0 be a solution for the equations and let $M(p, q) = p(x^0)q(x^0)$. Note that M clearly satisfies all the conditions. □

Since the set of such operators M is convex, for every distribution μ over solutions of $\mathcal{E}$, the operator $M(p, q) = \mathbb{E}_{x\sim\mu} p(x)q(x)$ also satisfies the conditions. As ℓ grows, eventually the only operators that satisfy the condition will be of this form.

For this reason we will call M a *degree-ℓ pseudo-expectation operator*. For a polynomial p of degree at most ℓ, we define $M(p)$ as follows: we write $p = \sum \alpha_i p_i$ where each p_i is a *monomial* of degree at most ℓ, and then decompose $p_i = p_i' p_i''$ where the degree of p_i' and p_i'' is at most $\ell/2$ and then define $M(p) = \sum \alpha_i M(p_i', p_i'')$. We will often use the suggestive notation $\tilde{\mathbb{E}}p$ for $M(p)$.

Exercise 4: Show that $M(p)$ is well defined and does not depend on the decomposition.

4.2 Intuition— the Boolean cube

To get some intuition, we now focus attention about the special case that our goal is to maximize some polynomial $p_0(x)$ over over Boolean cube $\{\pm 1\}^n$ (i.e., the set of x's satisfying $x_i^2 = 1$.) This case is not so special in the sense that (a) it captures much of what we want to do in TCS and (b) the intuition it yields largely applies to more general settings.

Recall that we said that for every distribution μ over x's satisfying the constraints, we can get an operator M as above by looking at $\mathbb{E}_{x\sim\mu} p(x)q(x)$. We now show that in some sense *every* operator has this form, if, in a manner related to and very reminiscent of quantum information theory, we allow the probabilities to go negative.

Definition 6. A function $\mu : \{\pm 1\}^n \to \mathbb{R}$ is a *degree-ℓ pseudo-distribution* if it satisfies:

- Normalization: $\sum_{x\in\{\pm 1\}^n} \mu(x) = 1$.
- Restricted non-negativity: For every polynomial p of degree at most $\ell/2$, $\tilde{\mathbb{E}}_{x\sim\mu} p(x)^2 \geq 0$, where we define $\tilde{\mathbb{E}}_{x\sim\mu} f(x)$ as $\sum_{x\in\{\pm 1\}^n} \mu(x) f(x)$.

Note that if μ was actually pointwise non-negative then it would be an actual distribution on the cube. Thus an actual distribution over the cube is always a pseudo distribution.

Exercise 5: Show that a degree $2n$ pseudo-distribution is an actual distribution.

Exercise 6: Show that if μ is a degree ℓ pseudo-distribution, then there exists a degree-ℓ pseudo-distribution μ' such that $\tilde{\mathbb{E}}_{x\sim\mu} p(x) = \tilde{\mathbb{E}}_{x\sim\mu'} p(x)$ for every polynomial p and that $\mu'(x)$ is a degree ℓ polynomial in the variables of x. (Hence for our purposes we can always represent such pseudo-distributions with $n^{O(\ell)}$ numbers.)

Exercise 7: Show that for every polynomial p_0 of degree at most $\ell/2$, there exists a degree ℓ pseudo-distribution μ on the cube satisfying $\tilde{\mathbb{E}}_{x\sim\mu} p_0(x) \geq \lambda$ if and only if there exists a degree ℓ pseudo-expectation operator M as above satisfying $\{x_i^2 = 1 : i = 1..n\}$ such that $M(p_0) \geq \lambda$.

Therefore, we can say that the degree-ℓ SOS algorithm outputs either a degree-ℓ pseudo-distribution over the solutions to $\mathcal{E}$ or `'fail'` and only outputs the latter if the former doesn't exist. In particular if it outputs `'fail'` then there isn't any *actual* distribution over the solutions, and so the fact that the algorithm outputs `'fail'` is a *proof* that the original equations are unsatisfiable. We will see that by convex duality, the algorithm actually outputs an explicit proof of this fact that has a natural interpretation.

Exercise 8: (optional— for people who have heard about the Sherali-Adams linear programming hierarchy) Show that the variant of pseudo-distributions where we replace the condition that expectation is non-negative on all squares of degree $\ell/2$ polynomials with the condition that it should be non-negative on all non-negative functions that depend on at most ℓ variables can be optimized over using linear programming and is equivalent to ℓ rounds of the Sherali-Adams LP.

Are all pseudo-distributions distributions? For starters, we can always find a distribution matching all the quadratic moments.

Lemma 7 (Gaussian Sampling Lemma)**.** *Let M be a degree-ℓ pseudo-expectation operator for $\ell \geq 2$. Then there exists a distribution $(y_1, \ldots, y_n)$ over $\mathbb{R}^n$ such that for every polynomial p of degree at most 2, $M(p) = \mathbb{E}p(y)$. Moreover, y is a (correlated) Gaussian distribution.*

Note that even if M comes from a pseudo-distribution μ over the cube, the output of y will be real numbers that although satisfying $\mathbb{E}y_i^2 = 1$, will be in $\{\pm 1\}$.

Unfortunately, we don't have an analogous result for higher moments:

Exercise 9: Prove that if there was an analog of the Gaussian Sampling Lemma for every polynomial p of degree at most 6 then P=NP. (Hint: show that you could solve 3SAT, can you improve the degree to 4? maybe 3?)

Unfortunately, this will not be our way to get fame and fortune:

Exercise 10: Prove that there exists a degree 4 pseudo-distribution μ over the cube such that there does not exist any actual distribution ν that matches its expectation on all polynomials of degree at most 4. (Can you improve this to 3?)

5 Sum of Square Proofs

As we said, when the SOS algorithm outputs `'fail'` this can be interpreted as a proof that the system of equations is unsatisfiable. However, it turns out this proof actually has a special form that is known as an SOS proof or *positivstenelsatz*. An SOS proof uses the following rules of inference

$$\begin{aligned} p \geq 0, q \geq 0 &\models p + q \geq 0 \\ p \geq 0, q \geq 0 &\models pq \geq 0 \\ &\models p^2 \geq 0 \end{aligned}$$

They should be interpreted as follows. If you know that a set of conditions $\mathcal{E} = \{p_1 \geq 0, \ldots, p_m \geq 0\}$ is satisfied on some set S, then any conditions derived by the rules above would on that set as well. (Note that we only mentioned inequalities above, but of course $\{p = 0\}$ is equivalent to the conditions $\{p \geq 0, -p \geq 0\}$.)

Definition 8. Let $\mathcal{E}$ be a set of equations. We say that $\mathcal{E}$ implies $p \geq 0$ via a degree-ℓ SOS proof, denoted $\mathcal{E} \models_\ell p \geq 0$, if $p \geq 0$ can be inferred from the constraints in $\mathcal{E}$ via a sequence of applications of the rules above where all intermediate polynomials are of syntactic degree $\leq \ell$.

The *syntactic degree* of the polynomials in $\mathcal{E}$ is their degree, while the syntactic degree of $p + q$ (resp. pq) is equal to the maximum (resp. the sum) of the syntactic degrees of p, q. That is, the syntactic degree tracks the degrees of the intermediate polynomials without accounting for cancellations.

(Note: If we kept track of the actual degree instead of the syntactic degree we get a much stronger proof system for which we don't have a static equivalent form, and can prove some things that the static system cannot. See the paper of Grigoriev, Hirsch and Pasechnik `http://eccc.hpi-web.de/report/2001/103/` for discussion of this other system.**)**

Definition 9. Let $\mathcal{E}$ be a set $\{p_1 = \cdots = p_m = 0\}$ of polynomial equalities. We say that $\mathcal{E}$ has a *degree-ℓ SOS refutation* if $\mathcal{E} \models_\ell 0 \geq 1$.

It turns out that a degree-ℓ refutation can always be put in a particular compact *static* form.

Exercise 11: For every $d < \ell$, prove that $\mathcal{E} = \{p_1 = \cdots = p_m = 0\}$ (where all p_i's are of degree d) has a *degree-ℓ SOS refutation* if and only if there exists $q_1, \ldots, q_m$ of degree at most $\ell' = O(\ell)$ and $r_1, \ldots, r_{m'}$ of degree at most $\ell'/2$ such that

$$\sum q_i p_i = 1 + s \tag{2}$$

where $s = \sum_{i=1}^{m'} r_i^2$, i.e. it is a *sum of squares*. (It's OK if you lose a bit in each direction, i.e., in the if direction it could be that $\ell' = 2\ell$ while in the only if direction it could be that $\ell' = \ell/2$.)

Exercise 12: Show that we can take m' to be at most $n^{2\ell}$.

Exercise 13: Show that the set $(p_1, \ldots, p_m, s)$ satisfying (2) is a convex set with an efficient separation oracle.

Positivstellensatz (Stengle 64, Krivine 74) For every unsatisfiable system $\mathcal{E}$ of equalities there exists a finite ℓ s.t. $\mathcal{E}$ has a degree ℓ proof of unsatisfiability. **Exercise 14:** Prove P-satz for systems that include the constraint $x_i^2 = x_i$ for all i. In this case, show that ℓ needs to be at most $2n$ (where n is the number of variables). As a corollary, we get that the SOS algorithm does not need more than $n^{O(n)}$ time to solve polynomial equations on n Boolean variables. (Not very impressive bound, but good to know. In all TCS applications I am aware of, it's easy to show that the SOS algorithm will solve the problem in exponential time.)

Exercise 15: Show that if there exists a degree-ℓ SOS proof that $\mathcal{E}$ is unsatisfiable then there is no degree-ℓ pseudo-distribution consistent with $\mathcal{E}$.

SOS Theorem (Shor, Nesterov, Parrilo, Lasserre) Under some mild conditions (see Theorem 2.7 in survey), there is an $n^{O(\ell)}$ time algorithm that given a set $\mathcal{E}$ of polynomial equalities either outputs:

- A degree-ℓ pseudo-distribution μ consistent with $\mathcal{E}$

  or

- A degree-ℓ SOS proof that $\mathcal{E}$ is unsatisfiable.

6 Discussion

The different views of pseudo distributions The notion of pseudo-distribution is somewhat counter-intuitive and takes a bit of time to get used to. It can be viewed from the following perspectives:

- Pseudo-distributions is simply a fancy name for a PSD matrix satisfying some linear constraints, which is the dual object to SOS proofs.

- SOS proofs of unbounded degree is a sound and complete proof system in the sense that they can prove any true fact (phrased as polynomial equations) about actual distributions over $\mathbb{R}^n$.

  SOS proofs of degree d is a sound and not complete proof system for actual distributions, but it is a (sound and) complete system for degree d pseudo-distributions, in the sense that any true fact that holds not merely for actual distributions but also for degree d pseudo-distributions has a degree d SOS proof.

- In statistical learning problems (and economics) we often capture our knowledge (or lack thereof) by a distribution. If an unknown quantity X is selected and we are given the observations y about it, we often describe our knowledge of by a the distribution $X|y$. In computational problems, often the observations y completely determine the value X, but pseudo-distribution can still capture our "computational knowledge".

- The proof system view can also be considered as a way to capture our limited computational abilities. In the example above, a computationally unbounded observer can deduce from the observations y all the true facts it implies and hence completely determine X. One way to capture the limits of a computationally bounded observer is that it can only deduce facts using a more limited, sound but not complete, proof system.

Lessons from History It took about 80 years from the time Hilbert showed that polynomials that are not SOS exist non-constructively until Motzkin came up with an explicit example, and even that example has a low degree SOS proof of positivity. One lesson from that is the following:
"Theorem": If a polynomial P is non-negative and "natural" (i.e., constructed by methods known to Hilbert— not including probabilistic method), then there should be a low degree SOS proof for this fact.
Corollary (Marley, 1980): If you analyze the performance of an SOS based algorithm pretending pseudo-distributions are actual distributions, then unless you used Chernoff+union bound type arguments, then every little thing gonna be alright.

We will use Marley's corollary extensively in analyzing SOS algorithms. There is a recurring theme in mathematics of "power from weakness". For example, we can often derandomize certain algorithms by observing that they fall in some restricted complexity classes and hence can be fooled by certain pseudorandom generator. Another example, perhaps closer to ours, is that even though the original way people defined calculus with "infitesimal" amounts were based on false permises, still much of the results they deduced were correct. One way to explain this is that they used a weak proof system that cannot prove all true facts about the real numbers, and in particular cannot detect if the real numbers are replaced with an object that does have such an "infitesimal" quantity added to it. In a similar way, if you analyze an algorithm using a weak proof system (e.g. one that is captured by a small degree SOS proof), then the analysis will still hold even if we replaced actual distributions with a pseudo-distribution of sufficiently large degree.