This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAC.2018.2886170, IEEE Transactions on Automatic Control



# Fast ADMM for sum-of-squares programs using partial orthogonality

Yang Zheng, Giovanni Fantuzzi, and Antonis Papachristodoulou, *Senior Member, IEEE*

***Abstract*—When sum-of-squares (SOS) programs are recast as semidefinite programs (SDPs) using the standard monomial basis, the constraint matrices in the SDP possess a structural property that we call *partial orthogonality*. In this paper, we leverage partial orthogonality to develop a fast first-order method, based on the alternating direction method of multipliers (ADMM), for the solution of the homogeneous self-dual embedding of SDPs describing SOS programs. Precisely, we show how a "diagonal plus low rank" structure implied by partial orthogonality can be exploited to project efficiently the iterates of a recent ADMM algorithm for generic conic programs onto the set defined by the affine constraints of the SDP. The resulting algorithm, implemented as a new package in the solver CDCS, is tested on a range of large-scale SOS programs arising from constrained polynomial optimization problems and from Lyapunov stability analysis of polynomial dynamical systems. These numerical experiments demonstrate the effectiveness of our approach compared to common state-of-the-art solvers.**

***Index Terms*—Sum-of-squares (SOS), ADMM, large-scale optimization.**

## I. Introduction

Optimizing the coefficients of a polynomial in $n$ variables, subject to a nonnegativity constraint on the entire space $\mathbb{R}^n$ or on a semialgebraic set $\mathcal{S} \subseteq \mathbb{R}^n$ (*i.e.*, a set defined by a finite number of polynomial equations and inequalities), is a fundamental problem in many fields. For instance, linear, quadratic and mixed-integer optimization problems can be recast as polynomial optimization problems (POPs) of the form [1]

$$\min_{x\in\mathcal{S}} p(x), \tag{1}$$

where $p(x)$ is a multivariate polynomial and $\mathcal{S} \subseteq \mathbb{R}^n$ is a semialgebraic set. Problem (1) is clearly equivalent to

$$\begin{aligned} \max \quad & \gamma \\ \text{s. t.} \quad & p(x) - \gamma \geq 0 \quad \forall x \in \mathcal{S}, \end{aligned} \tag{2}$$

so POPs of the form (1) can be solved globally if a linear cost function can be optimized subject to polynomial nonnegativity constraints on a semialgebraic set.

Another important example is the construction of a Lyapunov function $V(x)$ to certify that an equilibrium point $x^*$ of a dynamical system $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t))$ is locally stable. Taking $x^* = 0$ without loss of generality, given a neighbourhood $\mathcal{D}$ of the origin, local stability follows if $V(0) = 0$ and

$$V(x) > 0, \quad \forall x \in \mathcal{D} \setminus \{0\}, \tag{3a}$$

$$-f(x)^{\mathsf{T}} \nabla V(x) \geq 0, \quad \forall x \in \mathcal{D}. \tag{3b}$$

Often, the vector field $f(x)$ is polynomial [2] and, if one restricts the search to polynomial Lyapunov functions $V(x)$, conditions (3a)-(3b) amount to a feasibility problem over nonnegative polynomials.

Testing for nonnegativity, however, is NP-hard for polynomials of degree as low as four [3]. This difficulty is often resolved by requiring that the polynomials under consideration are a sum of squares (SOS) of polynomials of lower degree. In fact, checking for the existence (or lack) of an SOS representation amounts to solving a semidefinite program (SDP) [3]. In particular, consider a polynomial of degree $2d$ in $n$ variables,

$$p(x) = \sum_{\alpha\in\mathbb{N}^n, |\alpha|\leq 2d} p_\alpha x_1^{\alpha_1} \dots x_n^{\alpha_n}.$$

The key observation in [3] is that an SOS representation of $p(x)$ exists if and only if there exists a positive semidefinite matrix $X$ such that

$$p(x) = v_d(x)^{\mathsf{T}} X v_d(x), \tag{4}$$

where

$$v_d(x) = [1, x_1, x_2, \dots, x_n, x_1^2, x_1x_2, \dots, x_n^d]^{\mathsf{T}} \tag{5}$$

is the vector of monomials of degree no larger than $d$. Upon equating coefficients on both sides of (4), testing if $p(x)$ is an SOS reduces to a feasibility SDP of the form

$$\begin{aligned} \text{find} \quad & X \\ \text{s. t.} \quad & \langle B_\alpha, X \rangle = p_\alpha, \quad \alpha \in \mathbb{N}^n_{2d}, \\ & X \succeq 0, \end{aligned} \tag{6}$$

where $\mathbb{N}^n_{2d}$ is the set of $n$-dimensional multi-indices with length at most $2d$, $B_\alpha$ are known symmetric matrices indexed by such multi-indices (see Section II for more details), and $\langle A, B \rangle = \mathrm{trace}(AB)$ is the standard Frobenius inner product of two symmetric matrices $A$ and $B$.

Despite the tremendous impact of SOS techniques in the fields of polynomial optimization [4] and systems analysis [5], the current poor scalability of second-order interior-point algorithms for semidefinite programming prevents the use of SOS methods to solve POPs with many variables, or to analyse dynamical systems with many states. The main issue is that, when the full monomial basis (5) is used, the linear dimension of the matrix $X$ and the number of constraints in (6) are

Y. Zheng, and A. Papachristodoulou are with Department of Engineering Science at the University of Oxford. (E-mail: {yang.zheng, antonis}@eng.ox.ac.uk). G. Fantuzzi is with Department of Aeronautics at Imperial College London (E-mail: gf910@ic.ac.uk.). Y. Zheng is supported by the Clarendon Scholarship and the Jason Hu Scholarship, G. Fantuzzi is supported by an EPSRC Doctoral Prize Fellowship, and A. Papachristodoulou is supported by EPSRC Grant EP/M002454/1.

$N = \binom{n+d}{d}$ and $m = \binom{n+2d}{2d}$, respectively, both of which grow quickly as a function of $n$ and $d$.

One strategy to mitigate the computational cost of optimization problems with SOS constraints (hereafter called *SOS programs*) is to replace the SDP obtained from the basic formulation outlined above with one that is less expensive to solve using second-order interior-point algorithms. Facial reduction techniques [6], including the Newton polytope [7] and diagonal inconsistency [8], and symmetry reduction strategies [9] can be utilised to eliminate unnecessary monomials in the basis $v_d(x)$, thereby reducing the size of the positive semidefinite (PSD) matrix variable $X$. Correlative sparsity [10] can also be exploited to construct sparse SOS representations, wherein a polynomial $p(x)$ is written as a sum of SOS polynomials, each of which depends only on a subset of the entries of $x$. This enables one to replace the large PSD matrix variable $X$ with a set of smaller PSD matrices, which can be handled more efficiently. Further computational gains are available if one replaces any PSD constraints—either the original condition $X \succeq 0$ in (6) or the PSD constraints obtained after applying the aforemention techniques—with the stronger constraints the PSD matrices are diagonally or scaled-diagonally dominant [11]. These conditions can be imposed with linear and second-order cone programming, respectively, and are therefore less computationally expensive. However, while the conservativeness introduced by the requirement of diagonal dominance can be reduced with a basis pursuit algorithm [12], it cannot generally be removed.

Another strategy to enable the solution of large SOS programs is to replace the computationally demanding interior-point algorithms with first-order methods, at the expense of reducing the accuracy of the solution. The design of efficient first-order algorithms for large-scale SDPs has recently received increasing attention: Wen *et al.* proposed an alternating-direction augmented-Lagrangian method for large-scale dual SDPs [13]; O'Donoghue *et al.* developed an operator-splitting method to solve the homogeneous self-dual embedding of conic programs [14], which has recently been extended by the authors to exploit aggregate sparsity via chordal decomposition [15]–[17]. Algorithms that specialize in SDPs from SOS programming exist [18], [19], but can be applied only to unconstrained POPs—not to constrained POPs of the form (2), nor to the Lyapunov conditions (3a)-(3b). First-order regularization methods have also been applied to large-scale constrained POPs, but without taking into account any problem structure [20]. Finally, the sparsity of the matrices $B_\alpha$ in (6) was exploited in [21] to design an operator-splitting algorithm that can solve general large-scale SOS programs, but fails to detect infeasibility (however, recent developments [22], [23] may offer a solution for this issue).

One major shortcoming of all but the last of these recent approaches is that they can only be applied to particular classes of SOS programs. For this reason, in this paper we develop a fast first-order algorithm, based on the alternating direction method of multipliers, for the solution of generic large-scale SOS programs. Our algorithm exploits a particular structural property of SOS programs and can also detect infeasibility. Specifically, our contributions are:

1) We highlight a structural property of SDPs derived from SOS programs using the standard monomial basis: the equality constraints are *partially orthogonal*. Notably, the SDPs formulated by common SOS modeling toolboxes [24]–[26] possess this property.
2) We show how partial orthogonality leads to a "diagonal plus low rank" matrix structure in the ADMM algorithm of [14], so the matrix inversion lemma can be applied to reduce its computational cost. Precisely, a system of $m \times m$ linear equations to be solved at each iteration can be replaced with a $t \times t$ system, often with $t \ll m$.
3) We demonstrate the efficiency of our method—available as a new package in the MATLAB solver CDCS [27]—compared to many common interior-point solvers (SeDuMi [28], SDPT3 [29], SDPA [30], CSDP [31], Mosek [32]) and to the first-order solver SCS [33]. Our results on large-scale SOS programs from constrained POPs and Lyapunov stability analysis of nonlinear polynomial systems suggest that the proposed algorithm will enlarge the scale of practical problems that can be handled via SOS techniques.

The rest of this work is organized as follows. To make this paper self-contained, Section II briefly reviews SOS programs and their reduction to SDPs. Section III discusses partial orthogonality in the equality constraints of SDPs arising from SOS programs, while Section IV shows how to exploit it to facilitate the solution of large-scale SDPs using ADMM. Section V extends our results to matrix-valued SOS programs. Numerical experiments are presented in Section VI, and Section VII concludes the paper.

## II. Preliminaries

### A. Notation

The sets of nonnegative integers and real numbers are, respectively, $\mathbb{N}$ and $\mathbb{R}$. For $x \in \mathbb{R}^n$ and $\alpha \in \mathbb{N}^n$, the monomial $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ has degree $|\alpha| := \sum_{i=1}^n \alpha_i$. Given $d \in \mathbb{N}$, we let $\mathbb{N}^n_d = \{\alpha \in \mathbb{N}^n : |\alpha| \leq d\}$ and $\mathbb{R}[x]_{n,2d}$ be the set of polynomials in $n$ variables with real coefficients of degree $2d$ or less. A polynomial $p(x) \in \mathbb{R}[x]_{n,2d}$ is a sum-of-squares (SOS) if $p(x) = \sum_{i=1}^q [f_i(x)]^2$, for some polynomials $f_i \in \mathbb{R}[x]_{n,d}$, $i = 1, \ldots, q$. We denote by $\Sigma[x]_{n,2d}$ the set of SOS polynomials in $\mathbb{R}[x]_{n,2d}$. Finally, $\mathbb{S}^n_+$ is the cone of $n \times n$ PSD matrices and $I_{r\times r}$ is the $r \times r$ identity matrix.

### B. General SOS programs

Consider a vector of optimization variables $u \in \mathbb{R}^t$, a cost vector $w \in \mathbb{R}^t$, and note that any polynomial $p_j(x) \in \mathbb{R}[x]_{n,2d_j}$ whose coefficients depend affinely on $u$ can be written as $p_j(x) = g_0^j(x) - \sum_{i=1}^t u_i g_i^j(x)$ for a suitable choice of polynomials or monomials $g_0^j, \ldots, g_t^j \in \mathbb{R}[x]_{n,2d_j}$. We consider SOS programs written in the standard form

$$\begin{aligned} \min_{u,\, s_1,\ldots,s_k} \quad & w^{\mathsf{T}} u \\ \text{s. t.} \quad & s_j(x) = g_0^j(x) - \sum_{i=1}^t u_i g_i^j(x) \;\forall j = 1, \ldots, k, \\ & s_j \in \Sigma[x]_{n,2d_j}, \quad j = 1, \ldots, k. \end{aligned} \tag{7}$$

Note that any linear optimization problem with polynomial nonnegativity constraints on fixed semialgebraic sets can be relaxed into an SOS program of the form (7). For instance, when $\mathcal{S} \equiv \mathbb{R}^n$ problem (2) can be relaxed as [3]

$$\begin{aligned} \min_{\gamma,s} \quad & -\gamma \\ \text{s. t.} \quad & s(x) = p(x) - \gamma, \\ & s \in \Sigma[x]_{n,2d}. \end{aligned} \tag{8}$$

Similarly, the global stability of the origin for a polynomial dynamical system such that $f(0) = 0$ may be established by looking for a polynomial Lyapunov function of the form $V(x) = -\sum_{i=1}^{t} u_i g_i(x)$, where $g_1(0) = \cdots = g_t(0) = 0$. With $\mathcal{D} \equiv \mathbb{R}^n$, and after subtracting $x^\mathsf{T} x$ from the left-hand side of (3a) to ensure strict positivity away from the origin [5], suitable values $u_i$ can be found via the SOS feasibility program

$$\begin{aligned} &\text{find } u,\, s_1,\, s_2 \\ &\text{s. t.}\, s_1(x) = -x^\mathsf{T} x - \sum_{i=1}^{t} u_i g_i(x), \\ &\qquad s_2(x) = \sum_{i=1}^{t} u_i f(x)^\mathsf{T} \nabla g_i(x), \\ &\qquad s_1,\, s_2 \in \Sigma[x]_{n,2d}. \end{aligned} \tag{9}$$

It can be checked that SOS programs arising from polynomial nonnegativity constraints over fixed semialgebraic sets, such as Lasserre's relaxations of constrained POPs [4] and SOS relaxations of local Lyapunov inequalities [2], [34], can also be recast as in (7) by adding extra polynomials to represent the SOS multipliers introduced after applying the *Positivstellensatz* [2].[1]

To simplify the exposition in the rest of this work, instead of (7) we will consider the basic problem

$$\begin{aligned} \min_{u,\,s} \quad & w^\mathsf{T} u \\ \text{s. t.} \quad & s(x) = g_0(x) - \sum_{i=1}^{t} u_i g_i(x), \\ & s \in \Sigma[x]_{n,2d}. \end{aligned} \tag{10}$$

All of our results extend to (7) when $k > 1$, as well as to SOS programs with additional linear constraints on $u$, because each of $s_1, \ldots, s_k$ enters one and only one equality constraint.

### C. SDP formulation

The SOS program (10) can be converted into an SDP upon fixing a basis to represent the SOS polynomial variables. The simplest and most common choice to represent a degree-$2d$ SOS polynomial is the basis $v_d(x)$ of monomials of degree no greater than $d$, defined in (5). As discussed in [3] and [35], the polynomial $s(x)$ in (10) is SOS if and only if

$$s(x) = v_d(x)^\mathsf{T} X v_d(x) = \left\langle X, v_d(x) v_d(x)^\mathsf{T} \right\rangle, \; X \succeq 0. \tag{11}$$

Let $B_\alpha$ be the $0/1$ indicator matrix for the monomial $x^\alpha$ in the outer product matrix $v_d(x)v_d(x)^\mathsf{T}$, *i.e.*,

$$(B_\alpha)_{\beta,\gamma} = \begin{cases} 1 & \text{if } \beta + \gamma = \alpha \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

where the natural ordering of multi-indices $\beta, \gamma \in \mathbb{N}^n_d$ is used to index the entries of $B_\alpha$. Then,

$$v_d(x) v_d(x)^\mathsf{T} = \sum_{\alpha \in \mathbb{N}^n_{2d}} B_\alpha x^\alpha. \tag{13}$$

Upon writing $g_i(x) = \sum_{\alpha \in \mathbb{N}^n_{2d}} g_{i,\alpha} x^\alpha$ for each $i = 0, 1, \ldots, t$, and representing $s(x)$ as in (11), the equality constraint in (10) becomes

$$\begin{aligned} \sum_{\alpha \in \mathbb{N}^n_{2d}} \left( g_{0,\alpha} - \sum_{i=1}^{t} u_i g_{i,\alpha} \right) x^\alpha &= \left\langle X, v_d(x) v_d(x)^\mathsf{T} \right\rangle \\ &= \sum_{\alpha \in \mathbb{N}^n_{2d}} \left\langle B_\alpha, X \right\rangle x^\alpha. \end{aligned} \tag{14}$$

Matching the coefficients on both sides yields

$$g_{0,\alpha} - \sum_{i=1}^{t} u_i g_{i,\alpha} = \langle B_\alpha, X \rangle, \quad \forall \alpha \in \mathbb{N}^n_{2d}. \tag{15}$$

We refer to (15) as the *coefficient matching conditions* [21]. The SOS program (10) is then equivalent to the SDP

$$\begin{aligned} \min_{u} \;\; & w^\mathsf{T} u \\ \text{s. t.} \;\; & \langle B_\alpha, X \rangle + \sum_{i=1}^{t} u_i g_{i,\alpha} = g_{0,\alpha} \;\; \forall \alpha \in \mathbb{N}^n_{2d}, \\ & X \succeq 0. \end{aligned} \tag{16}$$

As already mentioned in Section I, when the full monomial basis $v_d(x)$ is used to formulate the SDP (16), the size of $X$ and the number of constraints are, respectively, $N = \binom{n+d}{d}$ and $m = \binom{n+2d}{2d}$. The size of SDP (16) may be reduced (often significantly) by eliminating redundant monomials in $v_d(x)$ based on the structure of the polynomials $g_0(x), \ldots, g_t(x)$; the interested reader is referred to Refs. [6]–[9].

## III. Partial orthogonality in SOS programs

For simplicity, we re-index the coefficient matching conditions (15) using integers $i = 1, \ldots, m$ instead of the multi-indices $\alpha$. Let $\mathrm{vec} : \mathbb{S}^N \to \mathbb{R}^{N^2}$ map a matrix to the stack of its columns and define $A_1 \in \mathbb{R}^{m \times t}$ and $A_2 \in \mathbb{R}^{m \times N^2}$ as

$$A_1 := \begin{bmatrix} g_{1,1} & \cdots & g_{t,1} \\ \vdots & \ddots & \vdots \\ g_{1,m} & \cdots & g_{t,m} \end{bmatrix}, \quad A_2 := \begin{bmatrix} \mathrm{vec}(B_1)^\mathsf{T} \\ \vdots \\ \mathrm{vec}(B_m)^\mathsf{T} \end{bmatrix}. \tag{17}$$

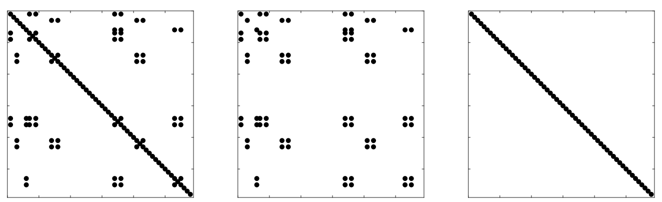
In other words, $A_1$ collects the coefficients of polynomials $g_i(x)$ column-wise, and $A_2$ lists the vectorized matrices $B_\alpha$ (after re-indexing) in a row-wise fashion. Finally, let $\mathcal{S}_+$ be the vectorized positive semidefinite cone, such that $\mathrm{vec}(X) \in \mathcal{S}_+$ if and only if $X \succeq 0$, and define

$$\begin{aligned} A &:= [A_1,\; A_2] \in \mathbb{R}^{m \times (t+N^2)}, \\ b &:= [g_{0,1}, \ldots, g_{0,m}]^\mathsf{T} \in \mathbb{R}^m, \\ c &:= \left[w^\mathsf{T}, 0, \ldots, 0\right]^\mathsf{T} \in \mathbb{R}^{t+N^2}, \\ \xi &:= \left[u^\mathsf{T}, \mathrm{vec}(X)^\mathsf{T}\right]^\mathsf{T} \in \mathbb{R}^{t+N^2}, \\ \mathcal{K} &:= \mathbb{R}^t \times \mathcal{S}_+ . \end{aligned}$$

[1] Adding extra polynomials is clearly undesirable; an extended manuscript (https://arxiv.org/abs/1708.04174) discusses how they can be avoided.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAC.2018.2886170, IEEE Transactions on Automatic Control





(a) (b) (c)

Fig. 1: Sparsity patterns for (a) $AA^\mathsf{T}$, (b) $A_1A_1^\mathsf{T}$, and (c) $A_2A_2^\mathsf{T}$ for problem `sosdemo2` in SOSTOOLS [24].

Then, noticing from the definition of the trace inner product of matrices that $\langle B_m, X\rangle = \mathrm{vec}(B_m)^\mathsf{T}\mathrm{vec}(X)$, we can rewrite (16) as the primal-form conic program

$$\begin{aligned} \min_{\xi} \quad & c^\mathsf{T}\xi \\ \text{s. t.} \quad & A\xi = b, \\ & \xi \in \mathcal{K}. \end{aligned} \tag{18}$$

The key observation at this stage is that the rows of the constraint matrix $A$ are *partially orthogonal*. We show this next, assuming without loss of generality that $t < m$; in fact, very often $t \ll m$ in practice (cf. Tables I and III in Section VI).

*Proposition 1:* Let $A = [A_1,\ A_2]$ be the constraint matrix in the conic formulation (17) of a SOS program modeled using the monomial basis. The $m\times m$ matrix $AA^\mathsf{T}$ is of the "diagonal plus low rank" form. Precisely, $D := A_2A_2^\mathsf{T}$ is diagonal and $AA^\mathsf{T} = D + A_1A_1^\mathsf{T}$.

*Proof:* The definition of $A$ implies $AA^\mathsf{T} = A_1A_1^\mathsf{T} + A_2A_2^\mathsf{T}$, so we need to show that $A_2A_2^\mathsf{T}$ is diagonal. This follows from the definition (12) of the matrices $B_\alpha$: if an entry of $B_\alpha$ is nonzero, the same entry in $B_\beta$, $\alpha \neq \beta$, must be zero. Upon re-indexing the matrices using integers $i = 1, \ldots, m$ as explained above and letting $n_i$ be the number of nonzero entries in $B_i$, it is clear that $\mathrm{vec}(B_i)^\mathsf{T}\mathrm{vec}(B_j) = n_i$ if $i = j$, and zero otherwise. Thus, $A_2A_2^\mathsf{T} = \mathrm{diag}(n_1, \ldots, n_m)$. ■

In essence, Proposition 1 states that the constraint submatrices corresponding to the matrix $X$ in the SOS decomposition (11) are orthogonal. This fact is a basic structural property for *any* SOS program formulated using the usual monomial basis. It is not difficult to check that Proposition 1 also holds when the full monomial basis $v_d(x)$ is reduced using any of the techniques implemented in any of the modeling toolboxes [24]–[26].

*Remark 1:* In general, the product $A_1A_1^\mathsf{T}$ has no particular structure, and $AA^\mathsf{T}$ is not diagonal except for very special problem classes. For example, Figure 1 illustrates the sparsity pattern of $AA^\mathsf{T}$, $A_1A_1^\mathsf{T}$, and $A_2A_2^\mathsf{T}$ for `sosdemo2` in SOSTOOLS [24], an SOS formulation of a Lyapunov function search: $A_2A_2^\mathsf{T}$ is diagonal, but $A_1A_1^\mathsf{T}$ and $AA^\mathsf{T}$ are not. This makes the algorithms proposed in [18], [19] inapplicable, as they require that $AA^\mathsf{T}$ is diagonal.

*Remark 2:* Using the monomial basis to formulate the coefficient matching conditions (15) makes the matrix $A$ sparse, because only a small subset of entries of the matrix $v_d(x)v_d(x)^\mathsf{T}$ are equal to a given monomial $x^\alpha$. In particular, the density of the nonzero entries of $A_2$ is $\mathcal{O}(n^{-2d})$ [21]. However, the aggregate sparsity pattern of SDP (18) is dense, so methods that exploit aggregate sparsity in SDPs [15]–[17], [36] are not useful for general SOS programs.

## IV. A FAST ADMM-BASED ALGORITHM

Partial orthogonality of the constraint matrix $A$ in conic programs of the form (18) allows for the extension of a first-order, ADMM-based method proposed in [14]. To make this paper self-contained, we summarize this algorithm first.

### A. The ADMM algorithm

The algorithm in [14] solves the homogeneous self-dual embedding [37] of the conic program (18) and its dual,

$$\begin{aligned} \max_{y,z} \quad & b^\mathsf{T}y \\ \text{s. t.} \quad & A^\mathsf{T}y + z = c. \\ & z \in \mathcal{K}^*, \end{aligned} \tag{19}$$

where the cone $\mathcal{K}^*$ is the dual of $\mathcal{K}$. When strong duality holds, optimal solutions for (18) and (19) or a certificate of primal or dual infeasibility can be recovered from a nonzero solution of the homogeneous linear system

$$\begin{bmatrix} z \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 & -A^\mathsf{T} & c \\ A & 0 & -b \\ -c^\mathsf{T} & b^\mathsf{T} & 0 \end{bmatrix} \begin{bmatrix} \xi \\ y \\ \tau \end{bmatrix}, \tag{20}$$

provided that it also satisfies $(\xi, y, \tau) \in \mathcal{K} \times \mathbb{R}^m \times \mathbb{R}_+$ and $(z, s, \kappa) \in \mathcal{K}^* \times \{0\}^m \times \mathbb{R}_+$. The interested reader is referred to [14] and references therein for more details. Consequently, upon defining

$$u := \begin{bmatrix} \xi \\ y \\ \tau \end{bmatrix}, \quad v := \begin{bmatrix} z \\ s \\ \kappa \end{bmatrix}, \quad Q := \begin{bmatrix} 0 & -A^\mathsf{T} & c \\ A & 0 & -b \\ -c^\mathsf{T} & b^\mathsf{T} & 0 \end{bmatrix}, \tag{21}$$

and introducing the cones $\mathcal{C} := \mathcal{K} \times \mathbb{R}^m \times \mathbb{R}_+$ and $\mathcal{C}^* := \mathcal{K}^* \times \{0\}^m \times \mathbb{R}_+$ to ease notation, a primal-dual optimal point for problems (18) and (19) or a certificate of infeasibility can be computed from a nonzero solution of the homogeneous self-dual feasibility problem

$$\begin{aligned} \text{find} \quad & (u, v) \\ \text{s. t.} \quad & v = Qu, \\ & (u, v) \in \mathcal{C} \times \mathcal{C}^*. \end{aligned} \tag{22}$$

It was shown in [14] that (22) can be solved using a simplified version of the classical ADMM algorithm (see *e.g.*, [38]), whose $k$-th iteration consists of the following three steps ($\mathbb{P}_\mathcal{C}$ denotes projection onto the cone $\mathcal{C}$, and the superscript $(k)$ indicates the value of a variable after the $k$-th iteration):

$$\hat{u}^{(k)} = (I + Q)^{-1}\left(u^{(k-1)} + v^{(k-1)}\right), \tag{23a}$$

$$u^{(k)} = \mathbb{P}_\mathcal{C}\left(\hat{u}^{(k)} - v^{(k-1)}\right), \tag{23b}$$

$$v^{(k)} = v^{(k-1)} - \hat{u}^{(k)} + u^{(k)}. \tag{23c}$$

Practical implementations of the algorithm rely on being able to carry out these steps at moderate computational cost. We next show that partial orthogonality allows for an efficient implementation of (23a) when (22) represents an SOS program.

### *B. Application to SOS programming*

Each iteration of the ADMM algorithm requires: a projection onto a linear subspace in (23a) through the solution of a linear system with coefficient matrix $I+Q$; a projection onto the cone $\mathcal{C}$ in (23b); and the inexpensive step (23c). The conic projection (23b) can be computed efficiently when the cone size is not too large. On the other hand, $Q \in \mathbb{S}^{t+N^2+m+1}$ and $m = \mathcal{O}(n^{2d})$ is extremely large in SDPs arising from SOS programs. For instance, an SOS program with polynomials of degree $2d = 6$ in $n = 16$ variables has a PSD variable of size $N = 969$ and $m = 74\,613$ equality constraints. This makes step (23a) computationally expensive not only if $I+Q$ is factorized directly, but also when applying the strategies proposed in [14]. Fortunately, $Q$ is highly structured and, in the context of SOS programming, the block-entry $A$ has partially orthogonal rows (cf. Propositions 1 and 2). As we will now show, these properties can be taken advantage of to achieve substantial computational savings.

To show how partial orthogonality can be exploited, we begin by noticing that (23a) requires the solution of a linear system of equations of the form

$$\begin{bmatrix} I & -A^{\mathsf{T}} & c \\ A & I & -b \\ -c^{\mathsf{T}} & b^{\mathsf{T}} & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}. \tag{24}$$

After letting

$$M := \begin{bmatrix} I & -A^{\mathsf{T}} \\ A & I \end{bmatrix}, \quad \zeta := \begin{bmatrix} c \\ -b \end{bmatrix},$$

and eliminating $\hat{u}_3$ from the first and second block-equations in (24) we obtain

$$\left(M + \zeta\zeta^{\mathsf{T}}\right) \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} - \omega_3 \zeta. \tag{25a}$$

$$\hat{u}_3 = \omega_3 + c^{\mathsf{T}}\hat{u}_1 - b^{\mathsf{T}}\hat{u}_2. \tag{25b}$$

Applying the matrix inversion lemma [39] to (25a) yields

$$\begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \end{bmatrix} = \left[ I - \frac{(M^{-1}\zeta)\zeta^{\mathsf{T}}}{1+\zeta^{\mathsf{T}}(M^{-1}\zeta)} \right] M^{-1} \begin{bmatrix} \omega_1 - c\omega_3 \\ \omega_2 + b\omega_3 \end{bmatrix}. \tag{26}$$

Note that the first matrix on the right-hand side of (26) only depends on problem data, and can be computed before iterating the ADMM algorithm. Consequently, all that is left to do at each iteration is to solve a linear system of equations of the form

$$\begin{bmatrix} I & -A^{\mathsf{T}} \\ A & I \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = \begin{bmatrix} \hat{\omega}_1 \\ \hat{\omega}_2 \end{bmatrix}. \tag{27}$$

Eliminating $\sigma_1$ from the second block-equation in (27) gives

$$\sigma_1 = \hat{\omega}_1 + A^{\mathsf{T}}\sigma_2, \tag{28a}$$

$$(I + AA^{\mathsf{T}})\sigma_2 = -A\hat{\omega}_1 + \hat{\omega}_2. \tag{28b}$$

It is at this stage that partial orthogonality comes into play: by Proposition 1, there exists a diagonal matrix $P$ such that $I + AA^{\mathsf{T}} = I + A_1A_1^{\mathsf{T}} + A_2A_2^{\mathsf{T}} = P + A_1A_1^{\mathsf{T}}$. Recalling from Section III that $A_1 \in \mathbb{R}^{m\times t}$ with $t \ll m$ for typical SOS programs (*e.g.*, $t = 3$ and $m = 58$ for problem `sosdemo2` in SOSTOOLS), it is therefore convenient to apply the matrix inversion lemma to (28b) and write

$$\begin{aligned}(I + AA^{\mathsf{T}})^{-1} &= (P + A_1A_1^{\mathsf{T}})^{-1} \\ &= P^{-1} - P^{-1}A_1(I + A_1^{\mathsf{T}}P^{-1}A_1)^{-1}A_1^{\mathsf{T}}P^{-1}.\end{aligned}$$

Since $P$ is diagonal, its inverse is immediately computed. Then, $\sigma_1$ and $\sigma_2$ in (28) are found upon solving a $t\times t$ linear system with coefficient matrix

$$I + A_1^{\mathsf{T}}P^{-1}A_1 \in \mathbb{S}^t, \tag{29}$$

plus relatively inexpensive matrix-vector, vector-vector, and scalar-vector operations. Moreover, since the matrix $I + A_1^{\mathsf{T}}P^{-1}A_1$ depends only on the problem data and does not change at each iteration, its preferred factorization can be cached before iterating steps (23a)-(23c). Once $\sigma_1$ and $\sigma_2$ have been computed, the solution of (24) can be recovered using vector-vector and scalar-vector operations.

*Remark 3:* In [14], system (27) is solved either through a "direct" method based on a cached $LDL^{\mathsf{T}}$ factorization, or by applying the "indirect" conjugate-gradient (CG) method to (28b). Both these approaches are reasonably efficient, but exploiting partial orthogonality is advantageous because only a smaller linear system with size $t\times t$ need be solved, with $t \le m$ and typically $t \ll m$. Indeed, to solve (27), when sparsity is ignored, each iteration of our method requires $\mathcal{O}(t^2+mN^2+mt)$ floating-point operations (flops), compared to $\mathcal{O}((t+N^2+m)^2)$ flops for the "direct" method of [14] and $\mathcal{O}(n_{\rm cg}m^2+mN^2+mt)$ flops for the "indirect" method with $n_{\rm cg}$ CG iterations.[2] Of course, practical implementations of the methods of [14] exploit sparsity and have a much lower complexity than stated, but the results in Section VI confirm that the strategy outlined in this work remains more efficient.

## V. Matrix-valued SOS programs

Up to this point we have discussed partial orthogonality for scalar-valued SOS programs, but our results and the algorithm proposed in Section IV extend also to the matrix-valued case.

Given symmetric matrices $C_\alpha \in \mathbb{S}^r$, we say that the symmetric matrix-valued polynomial

$$P(x) := \sum_{\alpha\in\mathbb{N}^n_{2d}} C_\alpha x^\alpha$$

is an SOS matrix if there exits a $q\times r$ polynomial matrix $H(x)$ such that $P(x) = H(x)^{\mathsf{T}}H(x)$. Clearly, an SOS matrix is positive semidefinite for all $x\in\mathbb{R}^n$. It is known [40] that $P(x)$ is an SOS matrix if and only if there exists a PSD matrix $Y\in\mathbb{S}^l_+$ with $l = r\times\binom{n+d}{d}$ such that

$$P(x) = \left(I_r\otimes v_d(x)\right)^{\mathsf{T}} Y \left(I_r\otimes v_d(x)\right). \tag{30}$$

Similar to (10), we consider the matrix-valued SOS program

$$\begin{aligned} \min_u \quad & w^{\mathsf{T}}u \\ \text{s. t.} \quad & P(x) = P_0(x) - \sum_{h=1}^{t} u_h P_h(x), \\ & P(x) \text{ is SOS}, \end{aligned} \tag{31}$$

[2]More details can be found in an extended version of this work, available at https://arxiv.org/abs/1708.04174.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAC.2018.2886170, IEEE Transactions on Automatic Control



where $P_0(x)$, …, $P_t(x)$ are given symmetric polynomial matrices. Using (30), matching coefficients, and vectorizing, the matrix-valued SOS program (31) can be recast as a conic program of standard primal-form (18), for which the following proposition holds.

*Proposition 2:* The constraint matrix $A$ in the conic program formulation of the matrix-valued SOS problem (31) has partially orthogonal rows, *i.e.*, it can be partitioned into $A = \begin{bmatrix} A_1 \; A_2 \end{bmatrix}$ such that $A_2 A_2^{\mathsf{T}}$ is diagonal.

*Proof:* First, introduce matrices $C_\alpha(u)$, affinely dependent on $u$, such that

$$P_0(x) - \sum_{h=1}^{t} u_h P_h(x) = \sum_{\alpha \in \mathbb{N}^n_{2d}} C_\alpha(u)\, x^\alpha.$$

By virtue of (13), the SOS representation (30) of $P(x)$ can be written as

$$P(x) = \sum_{\alpha \in \mathbb{N}^n_{2d}} \begin{bmatrix} \langle Y_{11}, B_\alpha\rangle & \dots & \langle Y_{1r}, B_\alpha\rangle \\ \vdots & \ddots & \vdots \\ \langle Y_{r1}, B_\alpha\rangle & \dots & \langle Y_{rr}, B_\alpha\rangle \end{bmatrix} x^\alpha,$$

where $Y_{ij} \in \mathbb{S}^N$, $i, j = 1, \dots, r$ is the $(i, j)$-th block of matrix $Y \in \mathbb{S}^l_+$. Then, the equality constraints in (31) require

$$C_\alpha(u) = \begin{bmatrix} \langle Y_{11}, B_\alpha\rangle & \dots & \langle Y_{r1}, B_\alpha\rangle \\ \vdots & \ddots & \vdots \\ \langle Y_{r1}, B_\alpha\rangle & \dots & \langle Y_{rr}, B_\alpha\rangle \end{bmatrix}, \quad \forall \alpha \in \mathbb{N}^n_{2d}. \tag{32}$$

Upon vectorization, this set of affine equalities can be written compactly as

$$\begin{bmatrix} A_1 & A_2 \end{bmatrix} \begin{bmatrix} u \\ \mathrm{vec}(Y) \end{bmatrix} = b \tag{33}$$

for suitably defined matrices $A_1$, $A_2$ and a vector $b$.

The matrix $A_1$ depends on the matrices $C_\alpha(u)$, and generally has no particular structure. Instead, $A_2$ has orthogonal rows, hence $A_2 A_2^{\mathsf{T}}$ is diagonal. To see this, let $e_i \in \mathbb{R}^r$ be the standard unit vector in the $i$-th direction and define

$$E_i := e_i \otimes I_N \in \mathbb{R}^{l \times N},$$

so $E_i^{\mathsf{T}} Y E_j = Y_{ij}$ selects the $(i, j)$-th $N \times N$ block of $Y$. Moreover, let $(C_\alpha)_{ij}$ denote the $(i, j)$-th element of the matrix $C_\alpha$. The linear equalities (32) require that, for all $i, j = 1, \dots, r$ and all $\alpha \in \mathbb{N}^n_{2d}$,

$$\langle E_i^{\mathsf{T}} Y E_j, B_\alpha\rangle = (C_\alpha)_{ij}. \tag{34}$$

Vectorization of the left-hand side yields

$$\mathrm{vec}(B_\alpha)^{\mathsf{T}} (E_j^{\mathsf{T}} \otimes E_i^{\mathsf{T}}) \mathrm{vec}(Y) = (C_\alpha)_{ij}.$$

It is then not difficult to see that the rows of the matrix $A_2$ in (33) are the vectors $\mathrm{vec}(B_\alpha)^{\mathsf{T}} \cdot (E_j^{\mathsf{T}} \otimes E_i^{\mathsf{T}})$ for all triples $(\alpha, i, j)$ (the precise order of the rows is not important). To show that $A_2 A_2^{\mathsf{T}}$ is diagonal, therefore, it suffices to show that, for any two different triples $(\alpha_1, i_1, j_1)$ and $(\alpha_2, i_2, j_2)$,

$$\begin{aligned} 0 &= \mathrm{vec}(B_{\alpha_1})^{\mathsf{T}} (E_{j_1}^{\mathsf{T}} \otimes E_{i_1}^{\mathsf{T}})(E_{j_2} \otimes E_{i_2}) \mathrm{vec}(B_{\alpha_2}) \\ &= \mathrm{vec}(B_{\alpha_1})^{\mathsf{T}} (E_{j_1}^{\mathsf{T}} E_{j_2} \otimes E_{i_1}^{\mathsf{T}} E_{i_2}) \mathrm{vec}(B_{\alpha_2}), \end{aligned} \tag{35}$$

where the second equality follows from the properties of the Kronecker product. To show (35), we invoke the properties of the Kronecker product once again to write

$$E_i^{\mathsf{T}} E_j = (e_i^{\mathsf{T}} e_j) \otimes I_N = \begin{cases} I_N, & \text{if } i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{36a}$$

$$\mathrm{vec}(B_\alpha)^{\mathsf{T}} \mathrm{vec}(B_\beta) = \begin{cases} n_\alpha, & \text{if } \alpha = \beta, \\ 0, & \text{otherwise}, \end{cases} \tag{36b}$$

where $n_\alpha$ is the number of nonzeros in $B_\alpha$. It is then clear that (35) holds if, and in fact only if, $(\alpha_1, i_1, j_1) \neq (\alpha_2, i_2, j_2)$. Consequently, $A_2 A_2^{\mathsf{T}}$ is diagonal. ■

Proposition 2 reveals an inherent structural property of SDPs derived from matrix-valued SOS programs using the monomial basis, and the algorithm of Section IV applies *verbatim* because the conic program representation of scalar- and matrix-valued SOS programs has the same general form.

## VI. Numerical Experiments

We implemented the algorithm of [14], extended to take into account partial orthogonality in SOS programs, as a new package in the open-source MATLAB solver CDCS [27]. Our implementation, which we refer to as CDCS-sos, solves step (23a) using a sparse permuted Cholesky factorization of the matrix in (29). The source code can be downloaded from https://github.com/oxfordcontrol/CDCS.

We tested CDCS-sos on a series of SOS programs and our scripts are available from https://github.com/zhengy09/sosproblems. CPU times were compared to the direct and indirect implementations of the algorithm of [14] provided by the solver SCS [33], referred to as SCS-direct and SCS-indirect, respectively. In our experiments, the termination tolerance for CDCS-sos and SCS was set to $10^{-3}$, and the maximum number of iterations was 2 000. Since first-order methods only aim at computing a solution of moderate accuracy, we assessed the suboptimality of the solution returned by CDCS-sos by comparing it to an accurate solution computed with the interior-point solver SeDuMi [28]. Besides, to demonstrate the low memory requirements of first-order algorithms, we also tested the interior-point solvers SDPT3 [29], SDPA [30], CSDP [31] and Mosek [32] for comparison. All interior-point solvers were called with their default parameters and their optimal values (when available) agree to within $10^{-8}$. All computations were carried out on a PC with a 2.8 GHz Intel® Core™ i7 CPU and 8GB of RAM; memory overflow is marked by ** in the tables below.

### A. Constrained polynomial optimization

As our first numerical experiment, we considered the constrained quartic polynomial minimization problem

$$\begin{aligned} \min_x \quad & \sum_{1 \le i < j \le n} (x_i x_j + x_i^2 x_j - x_j^3 - x_i^2 x_j^2) \\ \text{s. t.} \quad & \sum_{i=1}^{n} x_i^2 \le 1. \end{aligned} \tag{37}$$

We used the Lasserre relaxation of order $2d = 4$ and the parser GloptiPoly [25] to recast (37) into an SDP.

TABLE I: CPU time (in seconds) to solve the SDP relaxations of (37). $N$ is the size of the largest PSD cone, $m$ is the number of constraints, $t$ is the size of the matrix factorized by CDCS-sos.

| | Dimensions | | | CPU time (s) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $N$ | $m$ | $t$ | SeDuMi | SDPT3 | SDPA | CSDP | Mosek | SCS-direct | SCS-indirect | CDCS-sos |
| 10 | 66 | 1 000 | 66 | 2.6 | 2.1 | 1.6 | 2.5 | 0.8 | 0.4 | 0.4 | 0.4 |
| 12 | 91 | 1 819 | 91 | 12.3 | 7.0 | 5.7 | 4.0 | 2.4 | 0.7 | 0.8 | 0.7 |
| 14 | 120 | 3 059 | 120 | 68.4 | 24.2 | 18.1 | 13.5 | 6.5 | 1.7 | 1.7 | 1.4 |
| 17 | 171 | 5 984 | 171 | 516.9 | 129.6 | 97.9 | 75.8 | 38.1 | 4.6 | 4.4 | 3.5 |
| 20 | 231 | 10 625 | 231 | 2 547.4 | 494.1 | 452.7 | 374.2 | 178.9 | 10.6 | 10.6 | 8.5 |
| 24 | 325 | 20 474 | 325 | ** | ** | 2 792.8 | 2 519.3 | 1 398.3 | 32.0 | 31.2 | 22.8 |
| 29 | 465 | 40 919 | 465 | ** | ** | ** | ** | ** | 125.9 | 126.3 | 67.1 |
| 35 | 666 | 82 250 | 666 | ** | ** | ** | ** | ** | 425.3 | 431.3 | 216.9 |
| 42 | 946 | 163 184 | 946 | ** | ** | ** | ** | ** | 1 415.8 | 1 436.9 | 686.6 |

TABLE II: Terminal objective value from interior-point solvers, SCS-direct, SCS-indirect and CDCS-sos for the SDP relaxation of (37).

| n | †Interior-point solvers | SCS-direct | SCS-indirect | CDCS-sos |
|---|---|---|---|---|
| 10 | −9.11 | −9.12 | −9.13 | −9.10 |
| 12 | −11.12 | −11.10 | −11.10 | −11.11 |
| 14 | −13.12 | −13.09 | −13.09 | −13.12 |
| 17 | −16.12 | −16.09 | −16.09 | −16.06 |
| 20 | −19.12 | −19.17 | −19.17 | −19.08 |
| 24 | −23.12 | −23.04 | −23.04 | −23.15 |
| 29 | ** | −28.17 | −28.18 | −28.17 |
| 35 | ** | −34.05 | −34.05 | −34.08 |
| 42 | ** | −41.21 | −41.21 | −41.05 |

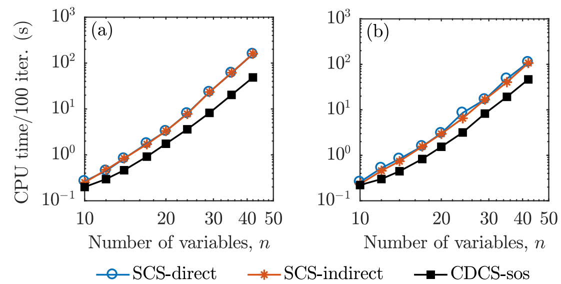



Fig. 2: Average CPU time per 100 iterations for the SDP relaxations of: (a) the POP (37); (b) the Lyapunov function search problem.

Table I reports the CPU time (in seconds) required by each of the solvers we tested to solve the SDP relaxations as the number of variables $n$ was increased. CDCS-sos is the fastest method in all cases. For large-scale POPs ($n \geq 29$), the number of constraints in the resulting SDP is over $40,000$, and all interior-point solvers (SeDuMi, SDPT3, SDPA, CSDP and Mosek) ran out of memory on our machine. The first-order solvers do not suffer from this limitation, and for POPs with $n \geq 29$ variables our MATLAB solver was approximately twice as fast as SCS. This is remarkable considering the SCS is written in C, and is due to the fact that $t \ll m$, cf. Table I, so the cost of the affine projection step (23a) in CDCS-sos is greatly reduced compared to the methods implemented in SCS. Figure 2(a) illustrates that, for all test problems, CDCS-sos was faster than both SCS-direct and SCS-indirect also in terms of average CPU time per 100 iterations (this metric is unaffected by differences in the termination criteria used by different solvers). Finally, Table II shows that although first-order methods only aim to provide solutions of moderate accuracy, the objective value returned by CDCS-sos and SCS was always within 0.5% of the high-accuracy optimal value computed using interior-point solvers. Such a small difference may be considered negligible in many applications.

### B. Finding Lyapunov functions

In our next numerical experiment, we considered the problem of constructing Lyapunov functions to verify local stability of polynomial systems, *i.e.*, we solved the SOS relaxation of (3a)-(3b) for different system instances. We used SOSTOOLS [24] to generate the corresponding SDPs.

In the experiment, we randomly generated polynomial dynamical systems $\dot{x} = f(x)$ of degree three with a linearly stable equilibrium at the origin. We then checked for local nonlinear stability in the ball $\mathcal{D} = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i^2 \leq 0.1\}$ using a quadratic Lyapunov function of the form $V(x) = x^{\mathsf{T}}Qx$ and Positivstellensatz to derive SOS conditions from (3a) and (3b) (see *e.g.*, [2] for more details). The total CPU time required by the solvers we tested are reported in Table III, while Figure 2(b) shows the average CPU times per 100 iterations for SCS and CDCS-sos. As in our previous experiment, the results clearly show that the iterations in CDCS-sos are faster than in SCS for all our random problem instances, and that both first-order solvers have low memory requirements and are able to solve large-scale problems ($n \geq 29$) beyond the reach of interior-point solvers.

### C. A practical example: Nuclear receptor signalling

As our last example, we considered a 37-state model of nuclear receptor signalling with a cubic vector field and an equilibrium point at the origin [41, Chapter 6]. We verified its local stability within a ball of radius $0.1$ by constructing a quadratic Lyapunov function. SOSTOOLS [24] was used to recast the SOS relaxation of (3a)-(3b) as an SDP with constraint matrix of size $102\,752 \times 553\,451$ and a large PSD cone of linear dimension $741$. Such a large-scale problem is currently beyond the reach of interior-point methods on a regular desktop computer, and all of the interior point solvers we tested (SeDuMi, SDPT3, SDPA, CSDP and Mosek) ran out of memory on our machine. On the other hand, the first-order solvers CDCS-sos and SCS managed to construct a valid Lyapunov function, with our partial-orthogonality-exploiting algorithm being more than twice as fast as SCS ($148\,$s vs. $\approx 400\,$s for both SCS-direct and SCS-indirect).

## VII. Conclusion

In this paper, we proved that SDPs arising from SOS programs formulated using the standard monomial basis possess a structural property that we call partial orthogonality. We then demonstrated that this property can be leveraged to substantially reduce the computational cost of an ADMM algorithm for conic programs proposed in [14]. Specifically, we showed that the iterates of this algorithm can be projected

TABLE III: CPU time (in seconds) to solve the SDP relaxations of (3a)-(3b). $N$ is the size of the largest PSD cone, $m$ is the number of constraints, $t$ is the size of the matrix factorized by CDCS-sos.

| | Dimensions | | | CPU time (s) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $N$ | $m$ | $t$ | SeDuMi | SDPT3 | SDPA | CSDP | Mosek | SCS-direct | SCS-indirect | CDCS-sos |
| 10 | 65 | 1 100 | 110 | 2.8 | 1.8 | 2.0 | 2.6 | 0.7 | 0.2 | 0.2 | 0.3 |
| 12 | 90 | 1 963 | 156 | 6.3 | 4.9 | 3.5 | 1.0 | 2.1 | 0.3 | 0.3 | 0.4 |
| 14 | 119 | 3 255 | 210 | 36.2 | 16.3 | 44.8 | 2.6 | 5.5 | 0.8 | 0.7 | 0.6 |
| 17 | 170 | 6 273 | 306 | 265.1 | 78.0 | 204.7 | 9.5 | 26.9 | 1.3 | 1.3 | 1.1 |
| 20 | 230 | 11 025 | 420 | 1 346.0 | 361.3 | 940.5 | 40.4 | 112.5 | 3.1 | 3.0 | 2.4 |
| 24 | 324 | 21 050 | 600 | ** | ** | 8 775.5 | 238.4 | 632.2 | 15.1 | 6.6 | 5.1 |
| 29 | 464 | 41 760 | 870 | ** | ** | ** | ** | ** | 17.1 | 16.9 | 14.3 |
| 35 | 665 | 83 475 | 1260 | ** | ** | ** | ** | ** | 67.6 | 57.1 | 37.4 |
| 42 | 945 | 164 948 | 1806 | ** | ** | ** | ** | ** | 133.7 | 129.2 | 92.8 |

efficiently onto a set defined by the affine constraints of the SDP. The key idea is to exploit a "diagonal plus low rank" structure of a large matrix that needs to be inverted/factorized, which is a direct consequence of partial orthogonality. Numerical experiments on large-scale SOS programs demonstrate that the method proposed in this paper yield considerable savings compared to many state-of-the-art solvers. For this reason we expect that our method will facilitate the use of SOS programming for the analysis and design of large-scale systems.

## References


[1] J. B. Lasserre, *Moments, positive polynomials and their applications*. World Scientific, 2009.

[2] J. Anderson and A. Papachristodoulou, "Advances in computational Lyapunov analysis using sum-of-squares programming," *Discrete Contin. Dynam. Syst. B*, vol. 20, no. 8, 2015.

[3] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Math. Program.*, vol. 96, no. 2, pp. 293–320, 2003.

[4] J. B. Lasserre, "Global optimization with polynomials and the problem of moments," *SIAM J. Optim.*, vol. 11, no. 3, pp. 796–817, 2001.

[5] A. Papachristodoulou and S. Prajna, "A tutorial on sum of squares techniques for systems analysis," in *Am. Control Conf. (ACC)*. IEEE, 2005, pp. 2686–2700.

[6] F. Permenter and P. A. Parrilo, "Basis selection for SOS programs via facial reduction and polyhedral approximations," in *53rd IEEE Conf. Decis. Control (CDC)*, 2014, pp. 6615–6620.

[7] B. Reznick *et al.*, "Extremal PSD forms with few terms," *Duke Math. J.*, vol. 45, no. 2, pp. 363–374, 1978.

[8] J. Löfberg, "Pre-and post-processing sum-of-squares programs in practice," *IEEE Trans. Autom. Control*, vol. 54, no. 5, pp. 1007–1011, 2009.

[9] K. Gatermann and P. A. Parrilo, "Symmetry groups, semidefinite programs, and sums of squares," *J. Pure Appl. Algebra*, vol. 192, no. 1, pp. 95–128, 2004.

[10] H. Waki, S. Kim, M. Kojima, and M. Muramatsu, "Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity," *SIAM J. Control Optim.*, vol. 17, no. 1, pp. 218–242, 2006.

[11] A. A. Ahmadi and A. Majumdar, "DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization," arXiv:1706.02586, 2017.

[12] A. A. Ahmadi and G. Hall, "Sum of squares basis pursuit with linear and second order cone programming," arXiv:1510.01597, 2015.

[13] Z. Wen, D. Goldfarb, and W. Yin, "Alternating direction augmented Lagrangian methods for semidefinite programming," *Math. Program. Comput.*, vol. 2, no. 3-4, pp. 203–230, 2010.

[14] B. O'Donoghue, E. Chu, and S. Parikh, Nealand Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *J. Optim. Theory Appl.*, vol. 169, no. 3, pp. 1042–1068, 2016.

[15] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn, "Fast ADMM for Semidefinite Programs with Chordal Sparsity," in *Proc. 2017 Am. Control Conf.* IEEE, 2017, pp. 3335–3340.

[16] ——, "Fast ADMM for homogeneous self-dual embeddings of sparse SDPs," *IFAC-PapersOnline*, vol. 50, no. 1, pp. 8411–8416, 2017.

[17] ——, "Chordal decomposition in operator-splitting methods for sparse semidefinite programs," arXiv:1707.05058, 2017.

[18] D. Henrion and J. Malick, "Projection methods in conic optimization," in *Handbook on Semidefinite, Conic and Polynomial Optimization*. Springer, 2012, pp. 565–600.

[19] D. Bertsimas, R. M. Freund, and X. A. Sun, "An accelerated first-order method for solving SOS relaxations of unconstrained polynomial optimization problems," *Optim. Methods Softw.*, vol. 28, no. 3, pp. 424–441, 2013.

[20] J. Nie and L. Wang, "Regularization methods for SDP relaxations in large-scale polynomial optimization," *SIAM J. Optim.*, vol. 22, no. 2, pp. 408–428, 2012.

[21] Y. Zheng, G. Fantuzzi, and A. Papachristodoulou, "Exploiting aparsity in the coefficient matching conditions in sum-of-squares programming using ADMM," *IEEE Control Syst. Lett.*, vol. 1, no. 1, pp. 80–85, 2017.

[22] G. Banjac, P. Goulart, B. Stellato, and S. Boyd, "Infeasibility detection in the alternating direction method of multipliers for convex optimization," optimization-online.org, Jun 2017.

[23] Y. Liu, E. K. Ryu, and W. Yin, "A new use of Douglas-Rachford splitting and ADMM for identifying infeasible, unbounded, and pathological conic programs," arXiv:1706.02374, 2017.

[24] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. Parrilo, "SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB," arXiv:1310.4716, 2013.

[25] D. Henrion and J.-B. Lasserre, "GloptiPoly: Global optimization over polynomials with MATLAB and SeDuMi," *ACM Trans. Math. Softw.*, vol. 29, no. 2, pp. 165–194, 2003.

[26] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in matlab," in *Proc. IEEE Int. Symp. Computer Aided Control Syst. Design*. IEEE, 2004, pp. 284–289.

[27] CDCS: Cone Decomposition Conic Solver. https://github.com/oxfordcontrol/CDCS, Sep. 2016.

[28] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimiz. Meth. Softw.*, vol. 11, no. 1-4, pp. 625–653, 1999.

[29] K.-C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3—a MATLAB software package for semidefinite programming, version 1.3," *Optimiz. Meth. Softw.*, vol. 11, no. 1-4, pp. 545–581, 1999.

[30] M. Yamashita, K. Fujisawa, and M. Kojima, "Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0)," *Optimiz. Meth. Softw.*, vol. 18, no. 4, pp. 491–505, 2003.

[31] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimiz. Meth. Softw.*, vol. 11, no. 1-4, pp. 613–623, 1999.

[32] The MOSEK optimization software. http://www.mosek.com.

[33] SCS: Splitting Conic Solver. https://github.com/cvxgrp/scs, Apr. 2016.

[34] A. Papachristodoulou and S. Prajna, "On the construction of Lyapunov functions using the sum of squares decomposition," in *41st IEEE Conf. Decis. Control (CDC)*, vol. 3. IEEE, 2002, pp. 3482–3487.

[35] V. Powers and T. Wörmann, "An algorithm for sums of squares of real polynomials," *Journal of pure and applied algebra*, vol. 127, no. 1, pp. 99–104, 1998.

[36] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, "Exploiting sparsity in semidefinite programming via matrix completion I: General framework," *SIAM J. Control Optim.*, vol. 11, no. 3, pp. 647–674, 2001.

[37] Y. Ye, M. J. Todd, and S. Mizuno, "An $\mathcal{O}(\sqrt{n}l)$-iteration homogeneous and self-dual linear programming algorithm," *Math. Oper. Res.*, vol. 19, no. 1, pp. 53–67, 1994.

[38] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, pp. 1–122, 2011.

[39] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[40] C. W. Scherer and C. W. Hol, "Matrix sum-of-squares relaxations for robust semi-definite programs," *Math. Program.*, vol. 107, no. 1, pp. 189–211, 2006.

[41] S. H. A. Khoshnaw, "Model reductions in biochemical reaction networks," Ph.D. dissertation, Department of Mathematics, 2015.