Exploiting Sparsity of Coefficient Matching Conditions in Sum-of-Squares Programs using ADMM

Yang Zheng[1], Giovanni Fantuzzi[2], Antonis Papachristodoulou[1]

***Abstract*— This paper introduces an efficient first-order method based on the alternating direction method of multipliers (ADMM) to solve the semidefinite programs (SDPs) arising from sum-of-squares (SOS) programming. We exploit the sparsity of the *coefficient matching conditions* when SOS programs are formulated in the usual monomial basis to reduce the computational costs of the ADMM algorithm. Each iteration of our algorithm consists of one projection onto the positive semidefinite cone and the solution of multiple quadratic programs with closed-form solutions free of any matrix inversion. Our techniques are implemented in an open-source MATLAB solver SOSADMM. Numerical experiments on SOS problems arising from unconstrained polynomial minimization and from Lyapunov stability analysis for polynomial systems show speed-ups compared to an interior-point solver SeDuMi, and a first-order solver CDCS.**

I. Introduction

Checking whether a given polynomial is positive semidefinite has applications in many areas. For example, the unconstrained polynomial optimization problem $\min_{x\in\mathbb{R}^n} p(x)$ is equivalent to the maximization problem

$$\begin{aligned} \max \quad & \gamma \\ \text{subject to} \quad & p(x)-\gamma \geq 0. \end{aligned} \tag{1}$$

Moreover, the stability of an equilibrium x^* of a polynomial dynamical system $\dot{x}=f(x)$, $x\in\mathbb{R}^n$, in a neighbourhood $\mathcal{D}$ of x^*—a fundamental problem in control theory—can be established by constructing a polynomial Lyapunov function $V(x)$ satisfying the polynomial inequalities

$$\begin{cases} V(x)>0, & \forall x\in\mathcal{D}\backslash\{0\}, \\ -\dot{V}(x)=-\langle\nabla V(x), f(x)\rangle\geq 0, & \forall x\in\mathcal{D}. \end{cases} \tag{2}$$

Throughout this work, $\langle\cdot,\cdot\rangle$ denotes the inner product in the appropriate Hilbert space.

A powerful way to verify polynomial inequalities is to employ a sum-of-squares (SOS) relaxation (we refer the reader to [1], [2] for details on SOS relaxations in polynomial optimization, and to [3] for a tutorial on SOS techniques for system analysis). In fact, while testing the non-negativity of a polynomial is NP-hard in general, the existence of a SOS decomposition can be checked in polynomial time by solving a semidefinite program (SDP) [1]. Unfortunately, however, the size of the SDP for the SOS relaxation of a degree-d polynomial in n variables is $\binom{n+d}{d}$. Consequently, SOS relaxations are limited to small problem instances; with the current technology, for example, Lyapunov-based analysis is impractical for systems with ten or more states.

In order to mitigate scalability issues, one can act at the *modeling* level, *i.e.* one can try to replace the SDP obtained from a SOS relaxation with an optimization problem that is cheaper to solve using second-order interior-point methods (IPMs), implemented in efficient solvers such as SeDuMi [4] and SDPT3 [5]. One approach is to exploit structural properties of the polynomial whose positivity is being tested [6]–[10]. For example, computing the Newton polytope [6] or checking for diagonal inconsistency [7] can restrict the monomials basis required in the SOS decomposition by eliminating redundant monomials. Further improvements are made possible by group-theoretic symmetry reduction techniques [8] and graph-theoretic correlative sparsity [9]. Facial reduction has also been applied to select a reduced monomial basis for SOS programs in [10]. A second approach is to approximate the positive semidefinite (PSD) cone using diagonally dominant or scaled diagonally dominant matrices [11], [12]. These relaxations can be solved with linear programs (LPs) or second-order-cone programs (SOCPs), rather than SDPs, and the conservativeness introduced by approximating the PSD cone can be reduced with a recently proposed basis pursuit algorithm [13].

Further improvements are available on the *computational* level if IPMs are replaced by more scalable first-order methods (FOMs) at the cost of reduced accuracy. The design of efficient first-order algorithms for large-scale SDPs has received particular attention in recent years. For instance, Wen *et al.* proposed an alternating direction augmented Lagrangian method for large-scale dual SDPs [14]. O'Donoghue *et al.* developed an operator-splitting method to solve the homogeneous self-dual embedding of conic programs [15], which has recently been extended by the authors to exploit aggregate sparsity via chordal decomposition [16], [17]. In the context of SOS programming, Bertsimas *et al.* proposed an accelerated FOM for unconstrained polynomial optimization [18], while Henrion & Malick introduced a projection-based method for general SOS relaxations [19].

In this paper, we propose a first-order algorithm based on the alternating direction method of multipliers (ADMM) to solve the SDPs arising in SOS optimization. The main idea is that while the aggregate sparsity pattern of such SDPs is dense, so the methods of [16], [17] are of less advantages, when a SOS program is formulated in the usual monomial basis each equality constraint in the SDP only involves a small subset of decision variables. This sparsity

Y. Zheng is supported by the Clarendon Scholarship and the Jason Hu Scholarship.

[1]Department of Engineering Science, University of Oxford, Parks Road, Oxford, OX1 3PJ, U.K. (e-mail: {yang.zheng, antonis}@eng.ox.ac.uk)

[2]Department of Aeronautics, Imperial College London, South Kensington Campus, London, SW7 2AZ, United Kingdom (e-mail: gf910@ic.ac.uk)

can be exploited to formulate an efficient ADMM algorithm, the iterations of which consist of conic projections and optimization problems with closed-form solutions that—crucially—are free of any matrix inversion. We implement our techniques in SOSADMM, an open-source MATLAB solver. The efficiency of our methods compared to the IPM solver SeDuMi [4] and a first-order solver CDCS [20] are demonstrated on SOS problems arising from unconstrained polynomial optimization and from Lyapunov stability analysis of polynomial systems.

The rest of this paper is organized as follows. Section II briefly reviews SOS polynomials and the ADMM algorithm. Constraint sparsity for SDPs arising in SOS programs is discussed in Section III, and we show how to to exploit it to build an efficient ADMM algorithm in Section III. Numerical experiments are reported in Section V. Section VI concludes the paper.

II. Preliminaries

A. SOS polynomials and SDPs

Let $x \in \mathbb{R}^n$, $\alpha \in \mathbb{N}^n$, and let $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ denote a monomial in x of degree $|\alpha| = \sum_{i=1}^n \alpha_i$. Given an integer $d \in \mathbb{N}$, we denote $\mathbb{N}_d^n = \{\alpha \in \mathbb{N}^n : |\alpha| \leq d\}$, and the vector of all monomials of degree no greater than d is denoted by

$$\begin{aligned} v_d(x) &= \{x^\alpha \mid \alpha \in \mathbb{N}_d^n\} \\ &= [1, x_1, x_2, \ldots, x_n, x_1^2, x_1 x_2, \ldots, x_n^d]^T. \end{aligned} \tag{3}$$

The length of $v_d(x)$ is $|\mathbb{N}_d^n| = \binom{n+d}{d}$. A real polynomial $p(x)$ is a finite, real linear combination of monomials of x

$$p(x) = \sum_{\alpha \in \mathbb{N}^n} p_\alpha x^\alpha, \qquad p_\alpha \in \mathbb{R}.$$

The degree of $p(x)$ is the maximum of the degrees of all monomials with nonzero coefficients. We denote the set of real polynomials in x by $\mathbb{R}[x]$.

Definition 1: A polynomial $p(x) \in \mathbb{R}[x]$ of degree $2d$ is a sum-of-squares (SOS) if there exist polynomials $f_i(x) \in \mathbb{R}[x]$, $i = 1, \ldots, m$ of degree no greater than d such that

$$p(x) = \sum_{i=1}^m [f_i(x)]^2 .$$

Clearly, the existence of an SOS representation guarantees that $p(x) \geq 0$. The following theorem gives an equivalent characterization of SOS polynomials .

Proposition 1 ([1]): A polynomial $p(x) \in \mathbb{R}[x]$ of degree $2d$ is an SOS polynomial if and only if there exists a $\binom{n+d}{d} \times \binom{n+d}{d}$ symmetric PSD matrix $X \succeq 0$ such that

$$p(x) = v_d(x)^T X v_d(x) \qquad \forall x. \tag{4}$$

Condition (4) gives a set of affine equalities on the elements of X to match the coefficients of $p(x)$. Together with $X \succeq 0$, this makes the problem of finding a SOS representation for $p(x)$ an SDP. The formulation of such SDPs can be assisted by software packages, such as SOSTOOLS [21] and GloptiPoly [22].

Remark 1: The size of the PSD matrix X in (4) is $\binom{n+d}{d} \times \binom{n+d}{d}$ because we have used the full set of monomials of degree no greater than d in our representation. This number can usually be reduced by inspecting the structural properties of $p(x)$ to identify and eliminate redundant monomials in $v_d(x)$; well-known techniques include Newton polytope [6], diagonal inconsistency [7], symmetry property [8], and facial reduction [10].

B. ADMM algorithm

The ADMM algorithm solves the optimization problem

$$\begin{aligned} \min_{y,z} \quad & f(y) + g(z) \\ \text{subject to} \quad & Ay + Bz = c, \end{aligned} \tag{5}$$

where $y \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$ are the decision variables, $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^m \to \mathbb{R}$ are convex functions, and $A \in \mathbb{R}^{l \times n}$, $B \in \mathbb{R}^{l \times m}$ and $c \in \mathbb{R}^l$ are the constraint data. Given a penalty parameter $\rho > 0$ and a multiplier $\lambda \in \mathbb{R}^l$ (known as the *dual variable*), the ADMM algorithm solves (5) by finding a saddle point of the augmented Lagrangian

$$L_\rho(y, z, \lambda) = f(y) + g(z) + \frac{\rho}{2} \left\| Ay + Bz - c + \frac{1}{\rho}\lambda \right\|^2$$

with the following steps:

$$y^{k+1} = \arg\min_y L_\rho(y, z^k, \lambda^k), \tag{6a}$$

$$z^{k+1} = \arg\min_z L_\rho(y^{k+1}, z, \lambda^k), \tag{6b}$$

$$\lambda^{k+1} = \lambda^k + \rho(Ay^{k+1} + Bz^{k+1} - c). \tag{6c}$$

In these equations, the superscript k denotes the value of a variable at the k-th iteration of the algorithm. The ADMM is particularly suitable when the minimization with respect to each of the variables y and z in (6a) and (6b) can be carried out efficiently through closed-form expressions. More details can be found in [23].

III. Row Sparsity in SDPs from SOS programs

A. SDP formulations of SOS relaxations

Let A_α be the indicator matrix for the monomials x^α in the rank-one matrix $v_d(x)v_d(x)^T$; in other words, the entry of A_α with row index β and column index γ (where the natural ordering for multi-indices $\beta, \gamma \in \mathbb{N}_d^n$ is used) satisfies

$$(A_\alpha)_{\beta,\gamma} = \begin{cases} 1 & \text{if } \beta + \gamma = \alpha \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The SOS constraint (4) can then be reformulated as

$$p(x) = \langle v_d(x)v_d(x)^T, X \rangle = \sum_{\alpha \in \mathbb{N}_d^n} \langle A_\alpha, X \rangle x^\alpha. \tag{8}$$

Matching the coefficients of the left- and right-hand sides gives the equality constraints

$$\langle A_\alpha, X \rangle = p_\alpha \quad \forall\, \alpha \in \mathbb{N}_{2d}^n. \tag{9}$$

TABLE I

DENSITY OF NONZERO ELEMENTS IN THE EQUALITY CONSTRAINTS OF SDP (10)

n	4	6	8	10	12	14	16
$2d=4$	1.42×10^{-2}	4.76×10^{-3}	2.02×10^{-3}	9.99×10^{-4}	5.49×10^{-4}	3.27×10^{-4}	2.06×10^{-4}
$2d=6$	4.76×10^{-3}	1.08×10^{-3}	3.33×10^{-4}	1.25×10^{-4}	5.39×10^{-5}	2.58×10^{-5}	1.34×10^{-5}
$2d=8$	2.02×10^{-3}	3.33×10^{-4}	7.77×10^{-5}	2.29×10^{-5}	7.94×10^{-6}	3.13×10^{-6}	1.36×10^{-6}

We refer to these equalities as *coefficient matching conditions*. The existence of a SOS decomposition for $p(x)$ (or lack thereof) can then be checked with the feasibility SDP

$$\begin{aligned} \text{find} \quad & X \\ \text{subject to} \quad & \langle A_\alpha, X\rangle = p_\alpha, \quad \alpha \in \mathbb{N}^n_{2d}, \\ & X \succeq 0. \end{aligned} \tag{10}$$

When the full monomial basis is used, as in this case, the dimension of X and the number of constraints in (10) are, respectively,

$$N = |\mathbb{N}^n_d| = \binom{n+d}{d}, \quad m = |\mathbb{N}^n_{2d}| = \binom{n+2d}{2d}. \tag{11}$$

B. Properties of the coefficient matching conditions

In this section, for simplicity, we re-index the constraint matching conditions (9) using integer indices $i = 1, \dots, m$ instead of the multi-indices α.

The conditions (9) inherit two important properties from the data matrices A_i, $i = 1, \dots, m$. The first one follows from the fact that the matrices A_i are orthogonal. In particular, if n_i denotes the number of nonzero entries (with value 1) in A_i we have

$$\langle A_i, A_j\rangle = \begin{cases} n_i & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

After letting $\text{vec} : \mathbb{S}^N \to \mathbb{R}^{N^2}$ be the usual operator mapping a matrix to the stack of its columns, and defining

$$A = \begin{bmatrix} \text{vec}(A_1) & \cdots & \text{vec}(A_m) \end{bmatrix}^T, \tag{13}$$

the equality constraints in (10) can be rewritten as

$$A \cdot \text{vec}(X) = b,$$

where $b \in \mathbb{R}^m$ is a vector collecting the coefficient p_i, $i = 1, \dots, m$. Property (12) directly implies the following lemma, which formed the basis of the FOMs of [18], [19].

Lemma 1 (Orthogonality of constraints): The matrix and AA^T is an $m \times m$ diagonal matrix with $(AA^T)_{ii} = n_i$.

The second property of the coefficient matching conditions is that they are sparse, in the sense that each equality constraint in (10) only involves a small subset of entries of X. In fact, only a small subset of entries of the product $v_d(x)v_d(x)^T$ are equal to a given monomial x^α. This implies that the vectorized matrix A is *row sparse*, meaning that each row is a sparse vector. In particular, we have

Lemma 2 (Sparsity of constraints): Let A be the vectorized matrix for (10), and let N and m be as in (11). The number of nonzero elements in A is N^2, and the density of nonzero elements in A is equal to $m^{-1} = \mathcal{O}(n^{-2d})$.

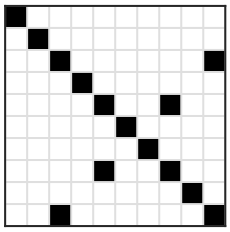

Fig. 1. Sparsity pattern of AA^T for the example (14)

Proof: Since the matrix $v_d(x)v_d(x)^T$ contains all monomials x^α, $\alpha \in \mathbb{N}^n_{2d}$, all entries of the PSD matrix X enter at least one of the equality constraints in (10). Moreover, (12) implies that each entry of X enters at most one constraint. Therefore, A must contain N^2 nonzero elements. Its density is then given by

$$\frac{N^2}{N^2 \times m} = \frac{1}{m} = \left[\binom{n+2d}{2d}\right]^{-1} = \mathcal{O}(n^{-2d}).$$

■

Remark 2: While the constraint matrix A is sparse (see typical values in Table I), the aggregate sparsity pattern of the SDP (10) is dense because all entries of the matrix variable X appear in the equality constraints. This implies that X is generally a dense variable, so the first-order algorithms of [16], [17] are not particularly suitable.

Remark 3: The property of orthogonality in Lemma 1 holds for standard SOS feasibility problems. However, this property fails for the following example:

$$\begin{aligned} \text{find} \quad & a,\, b \\ \text{subject to} \quad & ax^4 + bx^2 + x + 1 \text{ is SOS}, \\ & bx^4 + ax^2 + x + 1 \text{ is SOS}. \end{aligned} \tag{14}$$

Fig. 1 shows the sparsity pattern of AA^T for (14) obtained using SOSTOOLS, demonstrating that the constraints are not orthogonal. The reason is that (14) involves the free parameters a, b as well as the PSD matrices for the SOS representation, and this destroys the orthogonal of the equality constraints in the SDP. This issue is common in control applications; see, *e.g.*, the condition (2) when finding Lyapunov functions. Consequently, the first-order algorithms in [18], [19] cannot be applied to many problems with SOS constraints because they rely on constraint orthogonality.

IV. EXPLOITING ROW SPARSITY IN SDPS

As we have seen, the algorithms in [16]–[19] are not useful for generic SOS optimization problems because their aggregate sparsity pattern is dense, and the orthogonality

property only holds for simple SOS feasibility problems. However, the data matrix A is always row-sparse due to the coefficient matching conditions and this property can be exploited to construct an efficient ADMM algorithm that is particularly suited to SOS optimization. In the following, we consider a generic SDP in the vectorized form

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \in \mathcal{K}, \end{aligned} \tag{15}$$

where x is the optimization variable, A, b and c are the problem data, and $\mathcal{K}$ is a product of cones, at least one of which is the PSD cone.

A. Reformulation considering individual row sparsity

Let us represent $A = [a_1, a_2, \ldots, a_m]^T$, so each vector a_i is a row of A, and let $H_i, i = 1, \ldots, m$ be "entry-selector" matrices of 1's and 0's selecting the nonzero elements of a_i. Note that the rows of H_i are orthonormal, since each selects a different entry of a_i. Then,

$$Ax = b \Leftrightarrow \begin{cases} (H_i a_i)^T z_i = b_i, & i = 1, \ldots, m, \\ z_i = H_i x, & i = 1, \ldots, m. \end{cases} \tag{16}$$

In (16), z_i is a copy of the elements of x which enter the i-th affine constraint, its dimension being equal to the number of nonzero elements of a_i. It is also convenient to introduce an additional slack variable $z = x$, so the affine constraints in (16) and conic constraint in (15) are decoupled when applying the ADMM algorithm. We can then reformulate (15) as

$$\begin{aligned} \min_{z_i, z, x} \quad & c^T x \\ \text{subject to} \quad & (H_i a_i)^T z_i = b_i \quad i = 1, \ldots, m, \\ & z_i = H_i x, \qquad\quad i = 1, \ldots, m, \\ & z = x, \\ & z \in \mathcal{K}. \end{aligned} \tag{17}$$

B. ADMM steps

To apply ADMM, we move the affine constraints $(H_i a_i)^T z_i = b_i$ and the conic constraint $z \in \mathcal{K}$ in (17) to the objective using the indicator functions $\delta_0(\cdot)$ and $\delta_{\mathcal{K}}(\cdot)$, respectively:

$$\begin{aligned} \min_{z_i, z, x} \quad & c^T x + \delta_{\mathcal{K}}(z) + \sum_{i=1}^{m} \delta_0\left((H_i a_i)^T z_i - b_i\right) \\ \text{subject to} \quad & z_i = H_i x, \quad i = 1, \ldots, m, \\ & z = x. \end{aligned} \tag{18}$$

The augmented Lagrangian of (18) is

$$\begin{aligned} L = {} & c^T x + \delta_{\mathcal{K}}(z) + \sum_{i=1}^{m} \delta_0\left((H_i a_i)^T z_i - b_i\right) \\ & + \frac{\rho}{2} \sum_{k=1}^{m} \left\| z_i - H_i x + \frac{\mu_i}{\rho} \right\|^2 + \frac{\rho}{2} \left\| z - x + \frac{\xi}{\rho} \right\|^2, \end{aligned} \tag{19}$$

and we group the variables as

$$\mathcal{Y} = \{x\}, \quad \mathcal{Z} = \{z, z_1, \ldots, z_m\}, \quad \mathcal{D} = \{\mu_1, \ldots, \mu_m, \xi\}.$$

According to (6a)–(6c), the ADMM iterations for (18) consist of the following subproblems.

1) Minimization over $\mathcal{Y}$: The minimization of (19) over the variables in $\mathcal{Y}$ is an unconstrained quadratic program,

$$\min_{x} c^T x + \frac{\rho}{2} \sum_{i=1}^{m} \left\| z_i^k - H_i x + \frac{\mu_i^k}{\rho} \right\|^2 + \frac{\rho}{2} \left\| z^k - x + \frac{\xi^k}{\rho} \right\|^2. \tag{20}$$

The updated variable x^{k+1} at iteration $k+1$ is then simply given by

$$x = D^{-1} \left[\sum_{i=1}^{m} H_i^T \left(z_i^k + \frac{\mu_i^k}{\rho} \right) + \left(z^k + \frac{\xi^k}{\rho} \right) - \frac{1}{\rho} c \right], \tag{21}$$

where the matrix $D = I + \sum_{i=1}^{m} H_i^T H_i$ is diagonal because the rows of each matrix H_i are orthonormal. This means that (21) is cheap to calculate. Note that (21) can be viewed as an averaging/consensus step that takes all of the equality constraints into account.

2) Minimization over $\mathcal{Z}$: Minimizing (19) over the variables in $\mathcal{Z}$ amounts to a conic projection,

$$\begin{aligned} \min_{z} \quad & \left\| z - x^{k+1} + \frac{\xi^k}{\rho} \right\|^2 \\ \text{subject to} \quad & z \in \mathcal{K}, \end{aligned} \tag{22}$$

plus m independent quadratic programs

$$\begin{aligned} \min_{z_i} \quad & \left\| z_i - H_i x^{k+1} + \frac{\mu_i^k}{\rho} \right\|^2 \\ \text{subject to} \quad & (H_i a_i)^T z_i = b_i. \end{aligned} \tag{23}$$

The projection (22) is easy to compute when $\mathcal{K}$ is a product of $\mathbb{R}^n$, the non-negative orthant, second-order cones, and PSD cones; for example, a projection onto the PSD cone only requires the eigen-decomposition. As for problem (23), its KKT conditions are

$$z_i - H_i x^{k+1} + \frac{\mu_i^k}{\rho} + (H_i a_i)\omega_i = 0, \tag{24a}$$

$$(H_i a_i)^T z_i = b_i, \tag{24b}$$

where ω_i is the Lagrangian multiplier for the equality constraint in (23). Simple algebra shows that

$$\omega_i = \frac{1}{\|H_i a_i\|^2} \left(-b_i + (H_i a_i)^T H_i x^{k+1} - (H_i a_i)^T \frac{\mu_i^k}{\rho} \right),$$

so the solution z_i^{k+1} to (23) can be calculated easily with (24a). Note that the minimization over the block $\mathcal{Z}$ is free of any matrix inversion, and all subproblems can be solved in parallel.

3) Update multipliers $\mathcal{D}$: The final step in the $(k+1)$-th ADMM iteration is to update the multipliers in $\mathcal{D}$ with the usual gradient ascent rule:

$$\begin{aligned} & \mu_i^{k+1} = \mu_i^k + \rho(z_i^{k+1} - H_i x^{k+1}), i = 1, \ldots, m; \\ & \xi^{k+1} = \xi^k + \rho(z^{k+1} - x^{k+1}). \end{aligned} \tag{25}$$

This step is inexpensive and can be computed in parallel.

C. Summary of the computations in the ADMM algorithm

In the proposed ADMM algorithm, the subproblems (6a) and (6b) have explicit closed-form solutions. Each iteration requires solving

1) one unconstrained quadratic program, given in (20);
2) one conic projection, given in (22);
3) m independent quadratic programs, given in (23).

Note that only the nonzero elements of a_i appear in (23), and since we have assumed that a_i is sparse only vector-vector operations between vectors of small size are required. Besides, our algorithm is free of matrix inversion (with the exception of the diagonal matrix D), which results from introducing the local variables z_i so each affine constraint can be considered individually. In contrast, the FOMs in [16]–[19] require the solution of an $m \times m$ linear system of equations with sparsity pattern equal to that of the product AA^T. In SDPs arising from generic SOS relaxations, the number m of affine constraints is usually large ($m = 18564$ if $n = 12$ and $2d = 6$ in (10)), making this step computationally demanding if AA^T is not diagonal (and as we have already noticed, this is often the case).

V. Numerical Experiments

We implemented our techniques in SOSADMM, an open-source first-order MATLAB solver for conic programs with row sparsity. Currently, SOSADMM supports cartesian products of the following cones: $\mathbb{R}^n$, non-negative orthant, second-order cone, and the positive definite cone. SOSADMM is available from https:CodesURLsosadmm .

We tested SOSADMM on random unconstrained polynomial optimization problems and Lyapunov stability analysis of polynomial systems. To assess the suboptimality of the solution returned by SOSADMM, we compared it to the accurate one computed with the interior-point solver SeDuMi [4]. CPU times were compared to the first-order solver CDCS [20], which exploits aggregate sparsity in SDPs; in particular, the primal method in CDCS was used [16]. In all our experiments, the termination tolerance for SOSADMM and CDCS was set to 10^{-4}, and the maximum number of iterations was set to 2000. All tests were run on a PC with a 2.8 GHz Intel® Core™ i7 CPU and 8GB of RAM.

A. Unconstrained polynomial optimization

Consider the global polynomial minimization problem

$$\min_{x \in \mathbb{R}^n} \; p(x), \tag{26}$$

where $p(x)$ is a given polynomial. This problem is equivalent to (1), and we can obtain an SDP relaxation by replacing the non-negativity constraint with a SOS condition on $p(x) - \gamma$. Motivated by [19], we generated $p(x)$ according to

$$p(x) = p_0(x) + \sum_{i=1}^{n} x_i^{2d},$$

where $p_0(x)$ is a random polynomial of degree strictly less than $2d$. We used GloptiPoly [22] to generate the examples.

TABLE II

CPU time (in seconds) to solve the SDP relaxations of (26). N is the size of the PSD cone, m is the number of constraints.

Dimensions			CPU time (s)		
n	N	m	SeDuMi	CDCS (primal)	SOS-ADMM
2	6	14	0.23	0.08	0.05
4	15	69	0.13	0.11	0.06
6	28	209	0.24	0.16	0.14
8	45	494	1.16	0.18	0.18
10	66	1000	3.17	0.25	0.39
12	91	1819	13.89	0.46	0.55
14	120	3059	54.63	0.79	0.84
16	153	4844	187.0	0.92	0.82
18	190	7314	610.2	2.91	1.92
20	231	10625	1739	4.93	2.32

TABLE III

Lyapunov functions for the system (27)

Solver	Time (s)	Lyapunov function $V(x)$
SeDuMi	0.054	$6.659x_1^2 + 4.628x_2^2 + 2.073x_3^2$
CDCS-primal	0.21	$7.008x_1^2 + 1.477x_2^2 + 2.172x_3^2$
SOSADMM	0.58	$6.699x_1^2 + 1.803x_2^2 + 2.172x_3^2$

Table II compares the CPU time (in seconds) required to solve the SOS relaxation as the number n of variables was increased with $d = 2$. Both SOSADMM and CDCS-primal are faster than SeDuMi on these examples (note that SeDuMi's runtime reduces if a weaker termination tolerance is set, but not significantly). Also, the optimal value returned by SOSADMM was within 0.05% of the high-accuracy value returned by SeDuMi. For all examples in Table II, the cone size N is moderate (less than 300), while the number of constraints m is large. SeDuMi's algorithm assembles and solves an $m \times m$ linear system at each iteration, which is computationally expensive even though for the SDPs arising in (26) the constraints are orthogonal, *i.e.*, AA^T is diagonal. CDCS, instead, is almost as fast as SOSADMM because the linear system is only assembled and factorized once.

B. Finding Lyapunov functions

Next, we consider the problem of constructing Lyapunov functions to check local stability of polynomial/rational systems when (2) is replaced by SOS conditions. We used SOSTOOLS [21] to generate the corresponding SDPs.

The first system we study is

$$\begin{aligned} \dot{x}_1 &= -x_1^3 - x_1 x_3^2, \\ \dot{x}_2 &= -x_2 - x_1^2 x_2, \\ \dot{x}_3 &= -x_3 - \frac{3x_3}{x_3^2 + 1} + 3x_1^2 x_3. \end{aligned} \tag{27}$$

see demo 2 in SOSTOOLS. The system has an equilibrium at the origin, and we search for a homogeneous quadratic polynomial Lyapunov function $V(x) = ax_1^2 + bx_2^2 + cx_3^2$ to prove its global stability. The results given by SeDuMi, CDCS-primal and SOSADMM are listed in Table III. For such a small system, SeDuMi was slightly faster than CDCS-primal and SOSADMM, which is expected since IPMs

TABLE IV

CPU TIME (S) TO CONSTRUCT A QUADRATIC LYAPUNOV FUNCTION FOR RANDOMLY GENERATED POLYNOMIAL SYSTEMS.

	Statistics		CPU time (s)		
n	Size of A	nonzero density	SeDuMi	CDCS (primal)	SOS-ADMM
10	1100×2365	1.50×10^{-3}	3.3	7.5	5.3
12	1963×4407	8.76×10^{-4}	11.0	11.8	7.7
14	3255×7560	5.25×10^{-4}	49.9	21.0	11.2
16	5100×12172	3.13×10^{-4}	181.9	31.7	16.2
18	7638×18639	2.13×10^{-4}	574.8	55.0	24.6
20	11025×27405	1.48×10^{-4}	1617.2	100.3	37.7
22	15433×38962	1.11×10^{-4}	7442.7	265.9	65.6
25	24375×62725	6.87×10^{-5}	*	729.1	104.7
30	47275×124620	3.64×10^{-5}	*	3509.2	259.0

* SeDuMi fails due to memory requirements.

are well-suited for small-scale SDPs. Note that since the problem of constructing a Lyapunov functions is a feasibility problem, the solutions returned by SeDuMi, CDCS-primal and SOSADMM need not be the same.

We then consider randomly generated polynomial dynamical systems $\dot{x} = f(x)$ of degree three with a linearly stable equilibrium at the origin, and checked for local nonlinear stability in the ball $\mathcal{D} = \{x \in \mathbb{R}^n | 0.1 - \|x\|^2 \geq 0\}$. using a complete quadratic polynomial as the candidate Lyapunov function. Table IV summarizes the average CPU times required to search for such a Lyapunov function, when successful (note that we cannot detect infeasible problems because we only solve the primal form (15)). The results clearly show that SOSADMM is faster than both SeDuMi and CDCS-primal for the largest problem instances ($n \geq 18$). Also, FOMs have much lower memory requirements, and SOSADMM can solve problems that are not accessible with IMPs: SeDuMi failed due to memory issues when $n > 22$. Finally, note that for the problem of finding Lyapunov functions the $m \times m$ linear system solved in SeDuMi and CDCS is not diagonal, and solving it is expensive: when $n = 30$ it took over 150 s for CDCS just to factorize AA^T, which is over 50% of the total time taken by SOSADMM to return a solution.

VI. CONCLUSION

In this paper, we proposed an efficient ADMM algorithm to exploit the row-sparsity of SDPs that arise in SOS programming, are implemented it in the solver SOSADMM. The subproblems of our algorithm consist of one conic projection and multiple quadratic programs with closed-form solutions, which can be computed efficiently and—most importantly—do not require any matrix inversion. Our numerical experiments on random unconstrained polynomial optimization and on Lyapunov stability analysis of polynomial/rational systems demonstrate that our methods can provide speedups compared to the interior-point solver SeDuMi and the first-order solver CDCS. One major drawback of our methods is the inability to detect infeasibility; future work will try to exploit the sparsity of SDPs from SOS relaxations in a self-dual embedding formulation similar to that of [15], [17].

REFERENCES

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

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

[3] A. Papachristodoulou and S. Prajna, "A tutorial on sum of squares techniques for systems analysis," in *American Control Conference, 2005. Proceedings of the 2005*. IEEE, 2005, pp. 2686–2700.

[4] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization methods and software*, vol. 11, no. 1-4, pp. 625–653, 1999.

[5] K.-C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3a MATLAB software package for semidefinite programming, version 1.3," *Optimization methods and software*, vol. 11, no. 1-4, pp. 545–581, 1999.

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

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

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

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

[10] F. Permenter and P. A. Parrilo, "Basis selection for SOS programs via facial reduction and polyhedral approximations," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 6615–6620.

[11] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control and verification of high-dimensional systems with DSOS and SDSOS programming," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 394–401.

[12] A. A. Ahmadi and A. Majumdar, "DSOS and SDSOS optimization: LP and SOCP-based alternatives to sum of squares optimization," in *Information Sciences and Systems (CISS), 2014 48th Annual Conference on*. IEEE, 2014, pp. 1–5.

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

[14] Z. Wen, D. Goldfarb, and W. Yin, "Alternating direction augmented lagrangian methods for semidefinite programming," *Mathematical Programming Computation*, vol. 2, no. 3-4, pp. 203–230, 2010.

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

[16] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn, "Fast ADMM for semidefinite programs with chordal sparsity," *arXiv preprint arXiv:1609.06068*, 2016.

[17] ——, "Fast ADMM for homogeneous self-dual embeddings of sparse SDPs," *arXiv preprint arXiv:1611.01828*, 2016.

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

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

[20] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn, "CDCS: Cone decomposition conic solver, version 1.0," https://github.com/OxfordControl/CDCS, Sep. 2016.

[21] 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 preprint arXiv:1310.4716*, 2013.

[22] D. Henrion and J.-B. Lasserre, "GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 2, pp. 165–194, 2003.

[23] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.