REGULARIZATION METHODS FOR SDP RELAXATIONS IN LARGE SCALE POLYNOMIAL OPTIMIZATION

JIAWANG NIE* AND LI WANG*

Abstract. We study how to solve semidefinite programming (SDP) relaxations for large scale polynomial optimization. When interior-point methods are used, typically only small or moderately large problems could be solved. This paper studies regularization methods for solving polynomial optimization problems. We describe these methods for semidefinite optimization with block structures, and then apply them to solve large scale polynomial optimization problems. The performance is tested on various numerical examples. By regularization methods, significantly bigger problems could be solved on a regular computer, which is almost impossible by interior point methods.

Key words. polynomial optimization, Lasserre's relaxation, regularization methods, semidefinite programming, sum of squares

AMS subject classifications. 65K05, 90C22

1. Introduction.

Consider the polynomial optimization problem

$$\min_{x\in\mathbb{R}^n} \quad f(x) \quad s.t. \quad x \in S \tag{1.1}$$

where $f(x)$ is a multivariate polynomial and $S \subseteq \mathbb{R}^n$ is a semialgebraic set (defined by a boolean combination of polynomial equalities or inequalities). Recently, there has been much work on solving (1.1) by *semidefinite programming (SDP) relaxation* (also called *Lasserre's relaxation* in the literature). The basic idea is approximating nonnegative polynomials by sum of squares (SOS) type polynomials, which is equivalent to solving some SDP problems. Thus, the SDP packages (like SDPT3 [26], SeDuMi [25], SDPA[8]) would be applied to solve polynomial optimization problems. Typically, SDP relaxation is very successful in solving (1.1), as demonstrated by the pioneer work of Lasserre [14], Parrilo and Sturmfels [20] and many others. However, their applications are very limited in solving big problems. For instance, to minimize a general quartic polynomial, it is almost impossible to solve its SDP relaxation on a regular computer when it has more than 30 variables. So far, SDP relaxations for polynomial optimization can only be solved for small or moderately large problems, which severely limits their practical applications. Bigger problems would be solved if sparsity is exploited, like in the work [18, 27]. The motivation of this paper is proposing new methods for solving large scale SDP relaxations arising from general polynomial optimization.

A standard SDP problem is

$$\min_{X\in\mathcal{S}^N} \quad C \bullet X \quad s.t. \quad \mathcal{A}(X) = b,\ X \succeq 0. \tag{1.2}$$

Here $\mathcal{S}^N$ denotes the space of $N \times N$ real symmetric matrices, $X \succeq 0$ (resp. $X \succ 0$) means X is positive semidefinite (resp. definite), and $\bullet$ denotes the standard Frobenius inner product. The $C \in \mathcal{S}^N$ and $b \in \mathbb{R}^m$ are constant, and $\mathcal{A} : \mathcal{S}^N \to \mathbb{R}^m$ is a linear operator. The dual problem of (1.2) is

$$\max \quad b^T y \quad s.t. \quad \mathcal{A}^*(y) + Z = C,\ Z \succeq 0. \tag{1.3}$$

*Department of Mathematics, University of California, 9500 Gilman Drive, La Jolla, CA 92093. Emails: njw@math.ucsd.edu, liw022@math.ucsd.edu. The research was partially supported by NSF grants DMS-0757212 and DMS-0844775.

Here $\mathcal{A}^*$ is the adjoint of $\mathcal{A}$. An X is optimal for (1.2) and (y, Z) is optimal for (1.3) if the triple (X, y, Z) satisfies the optimality condition

$$\left.\begin{array}{rl} \mathcal{A}(X) & = b \\ \mathcal{A}^*(y) + Z & = C \\ X, Z \succeq 0,\ XZ & = 0 \end{array}\right\}. \tag{1.4}$$

There is much work on solving SDP by interior point methods. We refer to [28] for theory and algorithms for SDP. Most of them generate a sequence $\{(X_k, y_k, Z_k)\}$ converging to an optimal triple. At each step, a search direction $(\Delta X, \Delta y, \Delta Z)$ needs to be computed. To compute Δy, typically an $m \times m$ linear system needs to be solved. To compute ΔX and ΔZ, two linear matrix equations need to be solved. The cost for computing Δy is $\mathcal{O}(m^3)$. When $m = \mathcal{O}(N)$, the cost for computing Δy is $\mathcal{O}(N^3)$. In this case, solving SDP is not very expensive if N is not too big (like less than $1,000$). However, when $m = \mathcal{O}(N^2)$, the cost for computing Δy would be $\mathcal{O}(N^6)$, which is very expensive even for moderately large N (like 500). In this case, computing Δy is very expensive. It requires storing a matrix of dimension $m \times m$ in computer and $\mathcal{O}(m^3)$ arithmetic operations.

Unfortunately, SDP relaxations arising from polynomial optimization belong to the bad case that $m = \mathcal{O}(N^2)$, which is why the SDP solvers based on interior point methods have difficulty in solving big polynomial optimization problems (like degree 4 with 100 variables). We explain why this is the case. Let $p(x)$ be a polynomial of degree $2d$. Then, $p(x)$ is SOS if and only if there exists $X \succeq 0$ such that $p(x) = [x]_d^T X [x]_d$ (cf. [21]), where

$$[x]_d^T := [\,1 \quad x_1 \quad \cdots \quad x_n \quad x_1^2 \quad x_1x_2 \quad \cdots\cdots \quad x_1^d \quad x_1^{d-1}x_2 \quad \cdots\cdots \quad x_n^d\,].$$

Note the length of $[x]_d$ is $N = \binom{n+d}{d}$. If we write

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

then $p(x)$ being SOS is equivalent to the existence of a symmetric $N \times N$ matrix X satisfying

$$\left.\begin{array}{rcll} A_\alpha \bullet X & = & p_\alpha & \forall\, \alpha \in \mathbb{N}^n : |\alpha| \le 2d, \\ X & \succeq & 0. & \end{array}\right\} \tag{1.5}$$

Here A_α are certain constant symmetric matrices. The number of equalities is $m = \binom{n+2d}{2d}$. For any fixed d, $m = \mathcal{O}(n^{2d}) = \mathcal{O}(N^2)$. The size of SDP (1.5) is huge for moderately large n and d. Table 1 lists the size of SDP (1.5) for some typical values of $(n, 2d)$. As we have seen earlier, when interior point methods are applied to solve (1.2)-(1.3), at each step we need to solve a linear system and two matrix equations. To compute Δy, we need to store an $m \times m$ matrix and implement $\mathcal{O}(n^{6d})$ arithmetic operations. This is very expensive for even moderately large n and d, and hence severely limits the solvability of SDP relaxations in polynomial optimization. For instance, on a regular computer, to solve a general quartic polynomial optimization problem, it is almost impossible to apply interior point methods when there are more than 30 variables.

Recently, there has been much work on designing efficient numerical methods on solving big SDP problems. *Regularization methods* are such a kind of algorithms

n=	10	20	30	40	50
2d = 4	(66, 1001)	(231, 10626)	(496, 46376)	(861, 135751)	(1326, 316251)
n=	60	70	80	90	100
2d=4	(1891, 635376)	(2556, 1150626)	(3321, 1929501)	(4186, 3049501)	(5151, 4598126)
n=	10	15	20	25	30
2d = 6	(286, 8008)	(816, 54264)	(1771, 230230)	(3276, 736281)	(5456, 1947792)
n=	5	10	15	20	25
2d = 8	(126, 1287)	(1001, 43758)	(3876, 490314)	(10626,3108105)	(23751,13884156)
n=	5	8	9	10	15
2d = 10	(252, 3003)	(1287,43758)	(2002, 92378)	(3003, 184756)	(15504,3268760)

TABLE 1

A list of sizes of SDP (1.5). In each pair (N,m), N is the length of matrix and m is the number of equality constraints.

that are designed to solve SDP problems whose number of equality constraints m is significantly bigger than the matrix length N. We refer to [15, 22, 29] for the work in this area. Their numerical experiments show that these methods are practical and efficient in solving large scale SDP problems. In this paper, we study how to apply regularization methods to solve large scale polynomial optimization problems.

This paper is organized as follows. Section 2 reviews SDP relaxations in polynomial optimization. Section 3 shows how the regularization methods work for solving SDP problems whose matrices have block structures. Section 4 gives numerical experiments in solving large scale polynomial optimization problems, and Section 5 makes some discussions about numerical issues.

Notations. The symbol $\mathbb{N}$ (resp., $\mathbb{R}$) denotes the set of nonnegative integers (resp., real numbers). For any $t \in \mathbb{R}$, $\lceil t \rceil$ denotes the smallest integer not smaller than t. For $x \in \mathbb{R}^n$, x_i denotes the i-th component of x, that is, $x = (x_1, \ldots, x_n)$. The $\mathbb{S}^{n-1}$ denotes the $n-1$ dimensional unit sphere $\{x \in \mathbb{R}^n : x_1^2 + \cdots + x_n^2 = 1\}$. For $\alpha \in \mathbb{N}^n$, denote $|\alpha| = \alpha_1 + \cdots + \alpha_n$. The symbol $\mathbb{N}^n_{\leq k}$ denotes the set $\{\alpha \in \mathbb{N}^n : |\alpha| \leq k\}$, and $\mathbb{N}^n_k$ denotes $\{\alpha \in \mathbb{N}^n : |\alpha| = k\}$. For each i, e_i denotes the i-th standard unit vector. The $\mathbf{1}$ denotes a vector of all ones. For $x \in \mathbb{R}^n$ and $\alpha \in \mathbb{N}^n$, x^α denotes $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. For a finite set T, $|T|$ denotes its cardinality. For a matrix A, A^T denotes its transpose. The I_N denotes the $N \times N$ identity matrix, and $\mathcal{S}^N_+$ denotes the cone of symmetric positive semidefinite $N \times N$ matrices. For any vector $u \in \mathbb{R}^N$, $\|u\|_2 = \sqrt{u^T u}$ denotes the standard Euclidean norm.

2. SDP relaxations for polynomial optimization.

This section reviews constructions of SDP relaxations for polynomial optimization problems of three different types: unconstrained polynomial optimization, homogeneous polynomial optimization, and constrained polynomial optimization.

2.1. Unconstrained polynomial optimization.

Consider the optimization problem

$$f^{uc}_{min} := \min_{x \in \mathbb{R}^n} \quad f(x) \tag{2.1}$$

where $f(x)$ is a polynomial of degree $2d$, and f^{uc}_{min} denotes the global minimum of $f(x)$ over $\mathbb{R}^n$. A standard SOS relaxation for (2.1) (cf. [20, 21]) is

$$f^{uc}_{sos} := \max \quad \gamma \quad s.t. \quad f(x) - \gamma \text{ is SOS.} \tag{2.2}$$

Obviously the above optimal value f^{uc}_{sos} satisfies the relation $f^{uc}_{sos} \leq f^{uc}_{min}$. Though it is possible that $f^{uc}_{sos} < f^{uc}_{min}$, it was observed in [20, 21] that (2.2) works very well in practice. In the following, we show how to transform (2.2) into a standard SDP.

Denote $\mathbb{U}^n_{2d} = \{\alpha \in \mathbb{N}^n : 0 < |\alpha| \le 2d\}$ and write

$$f(x) = f_0 + \sum_{\alpha \in \mathbb{U}^n_{2d}} f_\alpha x^\alpha,$$

then $f(x) - \gamma$ is SOS if and only if there exists $X \in \mathcal{S}^{\binom{n+d}{d}}$ satisfying

$$f(x) - \gamma = [x]_d^T X [x]_d = X \bullet ([x]_d [x]_d^T), \quad X \succeq 0. \tag{2.3}$$

Note that $\binom{n+d}{d}$ is the length of $[x]_d$. Let $b = (f_\alpha)_{\alpha \in \mathbb{U}^n_{2d}}$, whose dimension is $m = \binom{n+2d}{2d} - 1$. Define 0/1 constant symmetric matrices C and A_α such that

$$[x]_d [x]_d^T = C + \sum_{\alpha \in \mathbb{N}^n : 0 < |\alpha| \le 2d} A_\alpha x^\alpha. \tag{2.4}$$

Then, (2.3) can be expressed as follows:

$$f(x) - \gamma = C \bullet X + \sum_{\alpha \in \mathbb{N}^n : 0 < |\alpha| \le 2d} (A_\alpha \bullet X) x^\alpha, \quad X \succeq 0. \tag{2.5}$$

So, γ is feasible for (2.2) if and only if there is a symmetric matrix X satisfying

$$\left.\begin{array}{rcll} C \bullet X + \gamma & = & f_0, & \\ A_\alpha \bullet X & = & f_\alpha & \forall\, \alpha \in \mathbb{U}^n_{2d}, \\ X & \succeq & 0. & \end{array}\right\}$$

Define a linear operator $\mathcal{A}(X) = (A_\alpha \bullet X)_{\alpha \in \mathbb{U}^n_{2d}}$. Then, up to a constant, SOS relaxation (2.2) is equivalent to the SDP problem

$$f^{uc}_{sdp} := \min \quad C \bullet X \quad s.t. \quad \mathcal{A}(X) = b,\, X \succeq 0. \tag{2.6}$$

The dual optimization of the above is

$$\max \quad b^T y \quad s.t. \quad \mathcal{A}^*(y) + Z = C,\, Z \succeq 0. \tag{2.7}$$

Here $\mathcal{A}^*(y) = \sum_{\alpha \in \mathbb{U}^n_{2d}} y_\alpha A_\alpha$. Clearly, it holds that $f^{uc}_{sos} = -f^{uc}_{sdp} + f_0$.

Suppose (X^*, y^*, Z^*) is an optimal triple for (2.6)-(2.7). Then $-f^{uc}_{sdp} + f_0$ is a lower bound of the minimum f^{uc}_{min}. As is well known, if Z^* has rank one, then $f^{uc}_{min} = f^{uc}_{sos}$ and a global minimizer for (2.1) can be obtained easily. This can be illustrated as follows. When $\mathrm{rank}(Z^*) = 1$, the constraint in (2.7) implies $Z^* = [x^*]_d [x^*]_d^T$ for some $x^* \in \mathbb{R}^n$, and hence $y^* = -[x^*]_{2d}$. Then, for any $x \in \mathbb{R}^n$,

$$-f(x^*) = -f_0 + b^T y^* \ge -f_0 + \sum_{\alpha \in \mathbb{U}^n_{2d}} -b_\alpha x^\alpha = -f(x).$$

In the above, we have used the optimality of y^* and the fact that $Z = [x]_d [x]_d^T$ is always feasible for (2.7). So x^* is a global minimizer.

When $\mathrm{rank}(Z^*) > 1$, several global minimizers for (2.1) could be obtained if the *flat extension condition (FEC)* holds. We refer to Curto and Fialkow [6] for FEC, and Henrion and Lasserre [9] for a numerical method on how to get global minimizers when FEC holds. Typically, FEC fails if the SDP relaxation is not exact.

2.2. Homogeneous polynomial optimization. Consider the homogeneous polynomial optimization problem

$$f_{min}^{hmg} := \min_{x\in\mathbb{R}^n} \quad f(x) \quad s.t. \quad \|x\|_2 = 1, \tag{2.8}$$

where $f(x)$ is a form (homogeneous polynomial). Assume its degree $\deg(f) = 2d$ is even. An interesting application of (2.8) is computing stability number of graphs [7]. This will also be shown in Section 4.2.

A standard SOS relaxation for (2.8) is

$$f_{sos}^{hmg} := \max \quad \gamma \quad s.t. \quad f(x) - \gamma\cdot(x^Tx)^d \quad \text{is SOS.} \tag{2.9}$$

Let $[x^d]$ be the vector of monomials of degree d ordered lexicographically, i.e.,

$$[x^d]^T = \begin{bmatrix} x_1^d & x_1^{d-1}x_2 & x_1^{d-1}x_3 & \cdots & x_{n-1}x_n^{d-1} & x_n^d \end{bmatrix}.$$

Denote $\mathbb{N}_d = \{\alpha\in\mathbb{N}^n : |\alpha| = d\}$. For each $\alpha\in\mathbb{N}_d$, define $D_\alpha = \frac{(|\alpha|)!}{\alpha_1!\cdots\alpha_n!}$. Let $D = \mathrm{diag}(D_\alpha)$ be a diagonal matrix. Then, it holds the relation

$$(x^Tx)^d = \sum_{\alpha\in\mathbb{N}_d} D_\alpha x_1^{2\alpha_1}\cdots x_n^{2\alpha_n} = [x^d]^T D[x^d] = ([x^d][x^d]^T)\bullet D. \tag{2.10}$$

Define 0/1 matrices H_α such that

$$[x^d][x^d]^T = \sum_{\alpha\in\mathbb{N}_{2d}} H_\alpha x_1^{\alpha_1}\cdots x_n^{\alpha_n}. \tag{2.11}$$

Then, $f(x) - \gamma\cdot(x^Tx)^d$ being SOS is equivalent to the existence of X satisfying

$$\begin{array}{rl} f_\alpha - \gamma D_{\alpha/2} &= H_\alpha\bullet X \quad \forall\,\alpha\in 2\mathbb{N}_d,\\ f_\alpha &= H_\alpha\bullet X \quad \forall\,\alpha\in \mathbb{N}_{2d}\backslash 2\mathbb{N}_d,\\ X &\succeq 0, \end{array}$$

where f_α is the coefficient of x^α in $f(x)$. Letting $\alpha = 2de_1$ in the above, we get $\gamma = f_{2de_1} - H_{2de_1}\bullet X$. Denote $\mathbb{H}_{2d}^n = \mathbb{N}_{2d}\backslash\{2de_1\}$, and set $r = (r_\alpha)_{\alpha\in\mathbb{H}_{2d}^n}$ as

$$r_\alpha = \begin{cases} D_{\alpha/2} & \text{if } \alpha\in 2\mathbb{N}_d\backslash\{2de_1\},\\ 0 & \text{if } \alpha\in\mathbb{N}_{2d}\backslash\{2\mathbb{N}_d\}. \end{cases} \tag{2.12}$$

Define matrices C, A_α and scalars b_α as

$$C = H_{2de_1}, \quad A_\alpha = H_\alpha - r_\alpha H_{2de_1}, \quad b_\alpha = f_\alpha - r_\alpha f_{2de_1}, \quad \alpha\in\mathbb{H}_{2d}^n. \tag{2.13}$$

Let $b = (b_\alpha)_{\alpha\in\mathbb{H}_{2d}^n}$. Define linear operators $\mathcal{H}, \mathcal{A} : \mathcal{S}^{\mathbb{N}_d} \to \mathbb{R}^{\mathbb{H}_{2d}^n}$ as

$$\mathcal{H}(X) = (H_\alpha\bullet X)_{\alpha\in\mathbb{H}_{2d}^n}, \quad \mathcal{A}(X) = (A_\alpha\bullet X)_{\alpha\in\mathbb{H}_{2d}^n}.$$

Then, SOS relaxation (2.9) is equivalent to the SDP problem

$$f_{sdp}^{hmg} := \min \quad C\bullet X \quad s.t. \quad \mathcal{A}(X) = b,\ X\succeq 0. \tag{2.14}$$

The dual problem of (2.14) is

$$\max \quad b^Ty \quad s.t. \quad \mathcal{A}^*(y) + Z = C,\ Z\succeq 0. \tag{2.15}$$

In the above, $\mathcal{A}^*(y) = \sum_{\alpha\in\mathbb{H}^n_{2d}} y_\alpha A_\alpha$. Clearly, $f^{hmg}_{sos} = -f^{hmg}_{sdp} + f_{2de_1}$.

Let (X^*, y^*, Z^*) be an optimal triple for (2.14)-(2.15). Then $-f^{hmg}_{sdp} + f_{2de_1}$ is a lower bound of the minimum f^{hmg}_{min}. We could also get global minimizers from Z^* when FEC holds. Note that (2.10) and (2.13) imply

$$A_\alpha \bullet D = 0 \quad \forall \alpha \in \mathbb{H}^n_{2d} \quad \text{and} \quad Z^* \bullet D = H_{2de_1} \bullet D = 1. \tag{2.16}$$

So $Z^*(de_i, de_i)$ (Z is indexed by integer vectors in $\mathbb{N}^n$) can not vanish for every i, because otherwise we would get $Z^* = 0$ contradicting (2.16). Up to a permutation of x, we can assume $Z^*(de_1, de_1) \neq 0$, and normalize Z^* as $\widehat{Z^*} = Z^*/Z^*(de_1, de_1)$. Then $\widehat{Z^*}$ would be thought of as a moment matrix of order d in $n-1$ variables (cf. [6]). If $\widehat{Z^*}$ satisfies FEC, using the method in [9], we can get $v^{(1)}, \ldots, v^{(r)} \in \mathbb{R}^{n-1}$ such that

$$\widehat{Z^*} = \lambda_1 [v^{(1)}]_d [v^{(1)}]_d^T + \cdots + \lambda_r [v^{(r)}]_d [v^{(r)}]_d^T$$

for some scalars $\lambda_i > 0$. Setting $x^{(i)} = \left(1 + \|v^{(i)}\|_2^2\right)^{-1/2} \begin{bmatrix} 1 \\ v^{(i)} \end{bmatrix} \in \mathbb{S}^{n-1}$, we get

$$Z^* = \nu_1 [(x^{(1)})^d][(x^{(1)})^d]^T + \cdots + \nu_r [(x^{(r)})^d][(x^{(r)})^d]^T$$

for some scalars $\nu_i > 0$. Then, the relations (2.10) and (2.16) imply $\nu_1 + \cdots + \nu_r = 1$. Since every $Z^{(i)} = [(x^{(i)})^d][(x^{(i)})^d]^T$ is feasible for (2.15), the optimality of Z^* implies every $x^{(i)}$ is a global minimizer of (2.8).

2.3. Constrained polynomial optimization. Consider the general polynomial optimization problem

$$\begin{cases} f^{con}_{min} := \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \qquad\quad s.t. & g_1(x) \geq 0, \ldots, g_\ell(x) \geq 0 \end{cases} \tag{2.17}$$

where $f(x)$ and $g_1(x), \ldots, g_\ell(x)$ are all polynomials in x and their degrees are no greater than $2d$. Problem (2.17) is NP-hard, even when $f(x)$ is quadratic and the feasible set is a simplex. Lasserre's relaxation [14] is a typical approach for solving (2.17). The d-th Lasserre's relaxation (d is also called the relaxation order) for (2.17) is

$$\begin{cases} f^{con}_{sos} := \max & \gamma \\ \qquad s.t. & f(x) - \gamma = \sigma_0 + g_1\sigma_1 + \cdots + g_\ell\sigma_\ell, \\ & \sigma_0, \sigma_1, \ldots, \sigma_\ell \text{ are SOS}, \\ & \deg(\sigma_0), \deg(\sigma_1 g_1), \ldots, \deg(\sigma_\ell g_\ell) \leq 2d. \end{cases} \tag{2.18}$$

Let $N(k) = \binom{n+k}{k}$, $d_i = \lceil \deg(g_i)/2 \rceil$ and $g_0(x) = 1$. Then, γ is feasible for (2.18) if and only if there exists $X^{(i)} \in \mathcal{S}^{N(d-d_i)}$ ($i = 0, 1, \ldots, \ell$) such that

$$\begin{aligned} f(x) - \gamma = \sum_{i=0}^{\ell} g_i(x) [x]_{d-d_i}^T X^{(i)} [x]_{d-d_i} = \sum_{i=0}^{\ell} X^{(i)} \bullet (g_i(x) [x]_{d-d_i} [x]_{d-d_i}^T), \\ X^{(0)} \succeq 0, X^{(1)} \succeq 0, \ldots, X^{(\ell)} \succeq 0. \end{aligned}$$

Define constant symmetric matrices $A^{(0)}_\alpha, A^{(1)}_\alpha, \ldots, A^{(\ell)}_\alpha$ such that

$$g_i(x) [x]_{d-d_i} [x]_{d-d_i}^T = \sum_{\alpha\in\mathbb{N}^n : |\alpha| \leq 2d} A^{(i)}_\alpha x^\alpha, \quad i = 0, 1, \ldots, \ell. \tag{2.19}$$

Denote $A_\alpha = (A_\alpha^{(0)}, A_\alpha^{(1)}, \ldots, A_\alpha^{(\ell)})$, $X = (X^{(0)}, X^{(1)}, \ldots, X^{(\ell)})$, and define a cone of products

$$\mathcal{K} := \mathcal{S}_+^{N(d-d_0)} \times \mathcal{S}_+^{N(d-d_1)} \times \cdots \times \mathcal{S}_+^{N(d-d_\ell)}.$$

Recall that $\mathbb{U}_{2d}^n = \{\alpha \in \mathbb{N}^n : 0 < |\alpha| \le 2d\}$. If $f(x) = f_0 + \sum_{\alpha \in \mathbb{U}_{2d}^n} f_\alpha x^\alpha$, then γ is feasible for (2.18) if and only if there exists X satisfying

$$\begin{array}{rcll} A_0 \bullet X + \gamma & = & f_0, & \\ A_\alpha \bullet X & = & f_\alpha & \forall\, \alpha \in \mathbb{U}_{2d}^n, \\ X & \in & \mathcal{K}. & \end{array}$$

Now define $\mathcal{A}, b, C$ as

$$\mathcal{A}(X) = (A_\alpha \bullet X)_{\alpha \in \mathbb{U}_{2d}^n}, \quad b = (f_\alpha)_{\alpha \in \mathbb{U}_{2d}^n}, \quad C = A_0. \tag{2.20}$$

The vector b has dimension $m = N(2d)-1$. Then, up to a constant, (2.18) is equivalent to the SDP problem

$$f_{sdp}^{con} := \min \quad C \bullet X \quad s.t. \quad \mathcal{A}(X) = b,\ X \in \mathcal{K}. \tag{2.21}$$

Its dual optimization is

$$\max \quad b^T y \quad s.t. \quad \mathcal{A}^*(y) + Z = C,\ Z \in \mathcal{K}. \tag{2.22}$$

The adjoint $\mathcal{A}^*(y)$ is defined as

$$\mathcal{A}^*(y) = \sum_{\alpha \in \mathbb{U}_{2d}^n} y_\alpha \ \mathrm{diag}\left(A_\alpha^{(0)}, A_\alpha^{(1)}, \ldots, A_\alpha^{(\ell)}\right).$$

Note the relation $f_{sos}^{con} = -f_{sdp}^{con} + f_0 \le f_{min}^{con}$.

Suppose (X^*, y^*, Z^*) is an optimal triple for (2.21)-(2.22). Then $-f_{sdp}^{con} + f_0$ is a lower bound of the minimum f_{min}^{con}. The information for minimizers could be obtained from Z^*. Note $Z^* = (Z_0^*, Z_1^*, \ldots, Z_\ell^*)$. Since $Z^* \in \mathcal{K}$, every $Z_i^* \succeq 0$. If Z_0^* satisfies FEC, one or several global minimizers can be obtained (cf. [9]).

3. Regularization methods. This section describes regularization methods for solving semidefinite optimization problems having block diagonal structures. They are natural generalizations of regularization methods introduced in [15, 22, 29] for solving standard SDP problems of a single block structure.

Let $\mathcal{K}$ be a cross product of several semidefinite cones

$$\mathcal{K} = \mathcal{S}_+^{N_1} \times \cdots \times \mathcal{S}_+^{N_\ell}.$$

It belongs to the space $\mathcal{M} = \mathcal{S}^{N_1} \times \cdots \times \mathcal{S}^{N_\ell}$. Each $X \in \mathcal{M}$ is a tuple $X = (X_1, \ldots, X_\ell)$ with every $X_i \in \mathcal{S}^{N_i}$. So, X could also be thought of as a symmetric block diagonal matrix, and $X \in \mathcal{K}$ if and only if its every block $X_i \succeq 0$. The notation $X \succeq_{\mathcal{K}} 0$ (resp. $X \succ_{\mathcal{K}} 0$) means every block of X is positive semidefinite (resp. definite). For $X = (X_1, \ldots, X_\ell) \in \mathcal{M}$ and $Y = (Y_1, \ldots, Y_\ell) \in \mathcal{M}$, define their inner product as $X \bullet Y = X_1 \bullet Y_1 + \cdots + X_\ell \bullet Y_\ell$. Denote by $\|\cdot\|$ the norm in $\mathcal{M}$ induced by this inner product. Note $\mathcal{K}$ is a self-dual cone, that is,

$$\mathcal{K}^* := \{Y \in \mathcal{M} : Y \bullet X \ge 0 \quad \forall X \in \mathcal{K}\} = \mathcal{K}.$$

For a symmetric matrix W, denote by $(W)_+$ (resp. $(W)_-$) the projection of W into the positive (resp. negative) semidefinite cone, that is, if W has spectral decomposition

$$W = \sum_{\lambda_i>0} \lambda_i u_i u_i^T + \sum_{\lambda_i<0} \lambda_i u_i u_i^T,$$

then $(W)_+$ and $(W)_-$ are defined as

$$(W)_+ = \sum_{\lambda_i>0} \lambda_i u_i u_i^T, \quad (W)_- = \sum_{\lambda_i<0} \lambda_i u_i u_i^T.$$

For $X = (X_1, \ldots, X_\ell) \in \mathcal{M}$, its projections into $\mathcal{K}$ and $-\mathcal{K}$ are given by

$$(X)_{\mathcal{K}} = ((X_1)_+, \ldots, (X_\ell)_+), \quad (X)_{-\mathcal{K}} = ((X_1)_-, \ldots, (X_\ell)_-).$$

A general conic semidefinite optimization problem is

$$\min \quad C \bullet X \quad s.t. \quad \mathcal{A}(X) = b,\ X \in \mathcal{K}. \tag{3.1}$$

Here $C \in \mathcal{M}$, $b \in \mathbb{R}^m$, and $\mathcal{A} : \mathcal{M} \to \mathbb{R}^m$ is a linear operator. The dual of (3.1) is

$$\max \quad b^T y \quad s.t. \quad \mathcal{A}^*(y) + Z = C,\ Z \in \mathcal{K}. \tag{3.2}$$

SDP relaxations for constrained polynomial optimization often have block diagonal structure, e.g., (2.21).

There are two typical regularizations for standard SDP problems: *Moreau-Yosida regularization* for the primal (1.2) and *Augmented Lagrangian regularization* for the dual (1.3). They would be naturally generalized to conic semidefinite optimization problem (3.1) and its dual (3.2). The Moreau-Yosida regularization for (3.1) is

$$\begin{cases} \min\limits_{X,Y\in\mathcal{M}} & C \bullet X + \frac{1}{2\sigma}\|X - Y\|^2 \\ s.t. & \mathcal{A}(X) = b,\ X \in \mathcal{K}. \end{cases} \tag{3.3}$$

Obviously (3.3) is equivalent to (3.1), because for each fixed feasible $X \in \mathcal{M}$ the optimal $Y \in \mathcal{M}$ in (3.3) is equal to X. The Augmented Lagrangian regularization for (3.2) is

$$\begin{cases} \max\limits_{y\in\mathbb{R}^m, Z\in\mathcal{M}} & b^T y - (Z + \mathcal{A}^*(y) - C) \bullet Y - \frac{\sigma}{2}\|Z + \mathcal{A}^*(y) - C\|^2 \\ s.t. & Z \in \mathcal{K}. \end{cases} \tag{3.4}$$

When $\mathcal{K} = \mathcal{S}_+^N$ is a single product, it can be shown (cf. [15, Section 2]) that for every fixed Y, (3.4) is the dual optimization problem of

$$\min_{X\in\mathcal{M}} \quad C \bullet X + \frac{1}{2\sigma}\|X - Y\|^2 - y^T(\mathcal{A}(X) - b) - Z \bullet X.$$

By fixing $y \in \mathbb{R}^m$ and optimizing over $Z \succeq 0$, Malick, Povh, Rendl, and Wiegele [15] further showed that (3.4) can be reduced to

$$\max_{y\in\mathbb{R}^m} \quad b^T y - \tfrac{\sigma}{2}\|(\mathcal{A}^*(y) - C + Y/\sigma)_{\mathcal{K}}\|^2 + \tfrac{1}{2\sigma}\|Y\|^2. \tag{3.5}$$

When $\mathcal{K} = \mathcal{S}_+^N$ is a single product, Malick, Povh, Rendl, and Wiegele [15] proposed a general framework (cf. [15, Algorithm 4.3]) of regularization methods for solving large scale SDP problems. It can be readily generalized to the case that $\mathcal{K}$ is a product of several semidefinite cones. We describe it as follows:

ALGORITHM 3.1. *Choose* $\epsilon^{in}, \epsilon^{out} \in (0,1)$.
Initialization: Choose $Y_0 \in \mathcal{S}^N$, $Z_0 = 0$, $y_0 \in \mathbb{R}^m$, σ_0 *and set* $k = 0$.
While ($\|Z_k + \mathcal{A}^*(y_k) - C\| \geq \epsilon^{out}$) *(outer loop):*
  Set $j := 0$, $y_{k,j} := y_k$ *and* $X_{k,j} := Y_k$.
  While ($\|b - \mathcal{A}(X_{k,j})\| \geq \epsilon^{in}$) *(inner loop):*
    Compute the projections
      $X_{k,j} := \sigma_k(Y_k/\sigma_k + \mathcal{A}^*(y_{k,j}) - C)_{\mathcal{K}}$,
      $Z_{k,j} := -(Y_k/\sigma_k + \mathcal{A}^*(y_{k,j}) - C)_{-\mathcal{K}}$.
    Set $g_j := b - \mathcal{A}(X_{k,j})$.
    Set $y_{k,j+1} := y_{k,j} + \tau W g_j$ *with appropriate* τ *and* W.
    Set $j := j + 1$.
  end (inner loop)
  Set $Y_{k+1} := X_{k,j}$, $y_{k+1} := y_{k,j}$ *and update* σ_k.
  Set $k := k + 1$.
end (outer loop)

In Algorithm 3.1, if W is chosen to be $(\mathcal{A}\mathcal{A}^*)^{-1}$ and $\tau = 1/\sigma$, Algorithm 3.1 becomes the *Boundary Point Method (BPM)*, which was originally proposed by Povh, Rendl, and Wiegele [22] (also see [15]) for solving big SDP problems. This method was proven efficient in some applications, as illustrated in [15, 22]. The description of this method is:

ALGORITHM 3.2. *Choose* $\epsilon \in (0,1)$.
Initialization: Choose $Y_0 \in \mathcal{S}^N$, $Z_0 = 0$, $y_0 \in \mathbb{R}^m$, σ_0 *and set* $k := 0$.
While ($\|b - \mathcal{A}(X_k)\| \geq \epsilon$ *or* $\|C - Z_k - \mathcal{A}^*(y_k)\| \geq \epsilon$)
  Solve $\mathcal{A}\mathcal{A}^* y_{k+1} = \mathcal{A}(C - Z_k) + (b - \mathcal{A}(Y_k))/\sigma_k$ *for* y_{k+1}.
  Compute the projections
    $X_{k+1} := \sigma_k(Y_k/\sigma_k + \mathcal{A}^*(y_{k+1}) - C)_{\mathcal{K}}$,
    $Z_{k+1} := -(Y_k/\sigma_k + \mathcal{A}^*(y_{k+1}) - C)_{-\mathcal{K}}$.
  Set $Y_{k+1} := X_{k+1}$ *and update* σ_k.
  Set $k := k + 1$.
end

For solving large scale SDP relaxations in polynomial optimization, Algorithm 3.2 might converge fast at the beginning, but generally has relatively slow convergence when it gets close to optimal solutions. This is because it is basically a gradient type method. When eigenvalue decompositions are not expensive, Algorithm 3.2 usually works very well, as demonstrated in [15, 22]. When it is expensive to compute eigenvalue decompositions, if Algorithm 3.2 takes a large number of iterations, then it might consume a lot of time. On the other hand, Algorithm 3.2 has simple iterations and is easily implementable. In each iteration, we only need to solve a linear system (its coefficient matrix $\mathcal{A}\mathcal{A}^*$ is fixed) and compute an eigenvalue decomposition. This is a big advantage. It would be applied to get a good approximate solution.

To get more accurate solutions, we need more efficient methods for the inner loop of Algorithm 3.1. Typically, Newton type methods have good properties like local superlinear or quadratic convergence. This leads to the *Newton-CG Augmented Lagrangian* method, which was proposed by Zhao, Sun and Toh [29]. It also has good numerical performance in solving big SDP problems. In the following, we describe this

important method for solving (3.1)-(3.2) when $\mathcal{K}$ is a product of several semidefinite cones.

Denote by $\varphi_\sigma(Y,y)$ the objective in (3.5). When $\mathcal{K}=\mathcal{S}^N_+$ is a single product, $\varphi_\sigma(Y,y)$ is differentiable [15, Proposition 3.2] and

$$\nabla_y\varphi_\sigma(Y,y) = b - \sigma\mathcal{A}\Big((\mathcal{A}^*(y)-C+Y/\sigma)_{\mathcal{K}}\Big).$$

The above is also true when $\mathcal{K}$ is a product of several semidefinite cones. The inner loop of Algorithm 3.1 is solving the maximization problem

$$\max_{y\in\mathbb{R}^m} \quad \varphi_\sigma(Y_k,y). \tag{3.6}$$

Since φ_σ is concave in y, a point $\hat{y}$ is a maximizer of (3.6) if and only if

$$\nabla_y\varphi_\sigma(Y_k,\hat{y}) = 0.$$

The function $\varphi_\sigma(Y,y)$ is not twice differentiable, so the standard Newton's method is not applicable. However, the function $\varphi_\sigma(Y,y)$ is semismooth, and semismooth Newton's method could be applied to get local superlinear or quadratic convergence, as pointed out in [29, Section 3.2]. For this purpose, we need the generalized Hessian of φ_σ in computation. We refer to [29, Section 3.2] for a numerical method of evaluating $\nabla^2_y\varphi_\sigma(Y,y)$. It is important to point out that the Hessian $\nabla^2_y\varphi_\sigma(Y,y)$ does not need to be explicitly formulated. It is implicitly available such that the matrix vector product $\nabla^2_y\varphi_\sigma(Y,y)\cdot z$ can be evaluated efficiently. Generally, we always have $\nabla^2_y\varphi_\sigma(Y,y)\succeq 0$, and $\nabla^2_y\varphi_\sigma(Y,y)\succ 0$ if some nondegeneracy condition holds (cf. [29, Prop. 3.2]). In either case, an approximate semismooth Newton direction d_{new} for (3.6) can be determined from the linear system

$$\Big(\nabla^2_y\varphi_\sigma(Y,y)+\epsilon\cdot I_N\Big)d_{new} = \nabla_y\varphi_\sigma. \tag{3.7}$$

Here $\epsilon>0$ is a tiny number ensuring the positive definiteness of the above linear system. When m is huge, it is usually not practical to solve (3.7) by direct methods like Cholesky factorization. To avoid this difficulty, conjugate gradient (CG) iterations are suitable, as proposed in [29].

Now we describe the Newton-CG Augmented Lagrangian regularization method.

ALGORITHM 3.3. *Choose $\epsilon^{in},\epsilon^{out}\in(0,1)$, $\epsilon>0$, $\delta\in(0,1)$, $\rho>1$, $\sigma_{\max},K\in\mathbb{N}$.*
Initialization: Choose $X_0,Z_0\in\mathcal{S}^N$, $y_0\in\mathbb{R}^m$, σ_0 and set $k:=0$.
While ($\|Z_k+\mathcal{A}^*(y_k)-C\|\geq\epsilon^{out}$) (outer loop):
  Set $Y_k:=X_k$.
  Set $j:=0$ and $y_{k,j}:=y_k$.
  While ($\|\nabla_y\varphi_{\sigma_k}(Y_k,y_{k,j})\|\geq\epsilon^{in}$) (inner loop):
    Set $g_j:=\nabla_y\varphi_{\sigma_k}(Y_k,y_{k,j})$.
    Compute d_{new} from (3.7) by applying preconditioned CG at most K steps.
    Find the smallest integer $\alpha>0$ such that

$$\varphi_{\sigma_k}(Y_k,y_{k,j}+\delta^\alpha\cdot d_{new}) \geq \varphi_{\sigma_k}(Y_k,y_{k,j})+\delta^\alpha\cdot g_j^T d_{new}. \tag{3.8}$$

    Set $y_{k,j+1}:=y_{k,j}+\delta^\alpha\cdot d_{new}$.
    Compute the projections:
      $X_k:=\sigma_k(Y_k/\sigma_k+\mathcal{A}^*(y_{k,j+1})-C)_{\mathcal{K}}$,

$Z_k := -(Y_k/\sigma_k + \mathcal{A}^*(y_{k,j+1}) - C)_{-\mathcal{K}}$.
Set $j := j + 1$.
end (inner loop)
Set $y_{k+1} := y_{k,j}$.
If $\sigma_k \le \sigma_{\max}$, *set* $\sigma_{k+1} := \rho\sigma_k$.
Set $k := k + 1$.
end (outer loop)

When $\mathcal{K} = \mathcal{S}_+^N$ is a single product, the convergence of Algorithm 3.3 has been discussed in [29, Theorems 3.5, 4.1, 4.2]. These results could be readily generalized to $\mathcal{K}$ being a product of several semidefinite cones. The specifics about the convergence are beyond the scope of this paper. We refer to [15, 23, 24, 29].

4. Computational experiments. This section presents numerical experiments of applying Algorithm 3.3 in solving polynomial optimization problems. An excellent implementation of Algorithm 3.3 is software `SDPNAL` [30]. We use it to solve the SDP relaxations (its earlier version posted in early 2010 was used). The computation is implemented in Matlab 7.10 on a Dell Linux Desktop running CentOS (5.6) with 8GB memory and Intel Core CPU 2.8GHz. We use the following parameters of `SDPNAL`: $\sigma_0 = 10,\ K = 500,\ \texttt{Tol} = 10^{-6}$. Set

$$R_P := \frac{\|\mathcal{A}(X) - b\|_2}{1 + \|b\|_2}, \quad R_D := \frac{\|\mathcal{A}^*(y) + Z - C\|_2}{1 + \|C\|_2},$$

which measure the feasibilities of the computed solutions for the primal and dual SDP problems respectively. We terminate computation when $\max\{R_P, R_D\} \le \texttt{Tol}$. Other parameters are set to be the default ones of `SDPNAL`.

If the computed dual optimal solution Z^* of (2.7) or (2.22) satisfies FEC, we could extract a global minimizer x^*; otherwise, we just set $Z^*(2 : n + 1, 1)$ as a starting point and get a local optimal solution x^* by using nonlinear programming solvers in Matlab Optimization Toolbox. In either case, the error of computed x^* is measured as

$$\text{errsol} \quad = \quad \frac{|f(x^*) - \underline{f}|}{\max\{1, |f(x^*)|\}}, \tag{4.1}$$

where $\underline{f}$ is a lower bound returned by solving the SDP relaxation. The error of a computed optimal triple (X, y, Z) for the SDP relaxation itself is measured as

$$\text{errsdp} \quad = \quad \max\left\{\frac{|b^\top y - \langle C, X\rangle|}{1 + |b^\top y| + |\langle C, X\rangle|}, R_P, R_D\right\}. \tag{4.2}$$

The consumed computer time is in the format `hr:mn:sc` with hr (resp. `mn, sc`) standing for the consumed hours (resp. minutes, seconds). In the tables of this paper, `min, med` and `max` respectively stands for the minimum, median, and maximum of quantities like time, errsol, errsdp, etc.

We would like to point out that extracting global minimizers is a difficult problem. When FEC is satisfied, one or several global minimizers of (1.1) can be found by solving eigenvalue problems (cf. [9]). When FEC fails, it's an open question how to extract global minimizers. In such situations, we just use nonlinear programming methods to get a local minimizer with $Z^*(2 : n + 1, 1)$ being a starting point. The experiment results in [27] show that this approach usually works very well. In our experiments, we also use this technique to get a local minimizer when FEC fails.

The testing problems in our experiments are in three categories: (a) unconstrained polynomial optimization and it's application in sensor network localization; (b) homogeneous polynomial optimization and it's application in computing stability numbers; (c) constrained polynomial optimization.

4.1. Unconstrained polynomial optimization.

Example 4.1. Minimize the following least squares polynomial

$$\sum_{k=1}^{3}\left(\sum_{i=1}^{n} x_i^k - 1\right)^2 + \sum_{i=1}^{n}\left(x_{i-1}^2 + x_i^2 + x_{i+1}^2 - x_i^3 - 1\right)^2$$

where $x_0 = x_{n+1} = 0$. For $n = 16$, the resulting SDP relaxation (2.6)-(2.7) has size $(N, m) = (969, 74612)$. Solving it by `SDPNAL` takes about 34 minutes. The computed solution of the SDP relaxation has error around $6 \cdot 10^{-7}$. The computed lower bound $f_{sos}^{uc} \approx 7.5586$. The optimal Z^* has rank two and FEC holds. We get two optimal solutions. Their errors are about $2 \cdot 10^{-6}$.

n	(N,m)	#Inst	time (min, med, max)			errsol (min, med, max)	errsdp (min, med, max)
20	(231, 10625)	20	0:00:02	0:00:04	0:00:09	(4.1e-7, 1.0e-5, 1.6e-4)	(2.5e-7, 7.3e-7, 1.3e-6)
30	(496, 46375)	20	0:00:12	0:00:21	0:00:31	(1.3e-7, 5.6e-5, 1.5e-4)	(3.2e-7, 6.8e-7, 1.0e-6)
40	(861, 135750)	10	0:00:57	0:01:10	0:01:24	(7.8e-7, 1.2e-4, 3.1e-4)	(4.2e-7, 4.7e-7, 9.6e-7)
50	(1326, 316250)	5	0:02:44	0:03:17	0:04:08	(1.3e-5, 3.2e-5, 2.3e-4)	(5.6e-7, 6.4e-7, 8.3e-7)
60	(1891, 635375)	5	0:07:55	0:08:49	0:09:48	(4.6e-5, 1.8e-4, 5.1e-4)	(4.8e-7, 9.1e-7, 9.5e-7)
70	(2556, 1150625)	5	0:17:38	0:19:34	0:22:33	(8.0e-5, 2.8e-4, 3.3e-4)	(4.1e-7, 5.7e-7, 9.2e-7)
80	(3321, 1929500)	3	0:38:45	0:38:48	0:42:46	(9.3e-5, 2.7e-4, 9.6e-4)	(3.7e-7, 7.0e-7, 9.9e-7)
90	(4186, 3049500)	3	1:37:04	1:46:57	2:02:01	(1.1e-4, 2.7e-4, 6.4e-4)	(4.3e-7, 5.2e-7, 9.5e-7)
100	(5151, 4598125)	3	2:48:03	2:55:34	3:35:27	(2.1e-4, 2.6e-4, 4.5e-4)	(7.1e-7, 7.9e-7, 8.7e-7)

TABLE 2
Computational results for random unconstrained polynomial optimization of degree 4

n	(N,m)	#Inst	time (min, med, max)			errsol (min, med, max)	errsdp (min, med, max)
10	(286, 8007)	20	0:00:07	0:00:17	0:00:36	(2.7e-7, 5.2e-6, 6.6e-5)	(2.4e-8, 4.2e-7, 1.1e-6)
15	(816, 54263)	10	0:01:12	0:01:51	3:07:37	(5.1e-6, 3.6e-5, 7.0e-5)	(2.0e-7, 7.4e-7, 9.6e-7)
20	(1771, 230229)	3	2:54:42	2:57:57	15:10:08	(1.4e-4, 2.5e-4, 4.0e-4)	(3.1e-7, 4.9e-7, 6.0e-7)
25	(3276, 736280)	3	2:02:59	5:25:06	7:34:03	(1.6e-3, 8.0e-3, 4.7e-2)	(2.6e-6, 8.6e-6, 5.7e-5)

TABLE 3
Computational results for random unconstrained polynomial optimization of degree 6

n	(N,m)	#Inst	time (min, med, max)			errsol (min, med, max)	errsdp (min, med, max)
8	(495, 12869)	20	0:00:18	0:00:42	0:01:11	(1.6e-7, 2.6e-5, 5.6e-4)	(1.0e-7, 5.8e-7, 4.1e-6)
10	(1001, 43757)	10	0:04:46	0:06:42	0:08:05	(3.9e-5, 8.4e-5, 5.3e-4)	(2.4e-7, 5.9e-7, 3.0e-6)
12	(1820, 125969)	3	0:26:32	0:37:43	1:02:37	(1.3e-5, 4.4e-5, 5.7e-3)	(1.1e-7, 7.2e-7, 5.3e-6)
15	(3876, 490313)	3	6:31:11	9:08:29	10:21:21	(6.8e-4, 8.1e-4, 4.5e-3)	(9.9e-7, 1.1e-6, 5.6e-6)

TABLE 4
Computational results for random unconstrained polynomial optimization of degree 8

Example 4.2 (Random polynomials). We test the performance of Algorithm 3.3 (via `SDPNAL`) in solving SDP relaxations for randomly generated polynomial optimization problems. To ensure the existence of a global minimizer, generate $f(x)$ randomly as

$$f(x) = f^T[x]_{2d-1} + [x^d]^T F^T F[x^d],$$

n	(N,m)	#Inst	time (min, med, max)			errsol (min, med, max)	errsdp (min, med, max)
6	(462, 8007)	20	0:00:10	0:00:18	0:00:32	(3.6e-7, 1.4e-5, 1.4e-4)	(3.2e-8, 5.2e-7, 3.1e-6)
8	(1287, 43757)	10	0:04:13	0:05:33	0:10:23	(5.6e-6, 5.2e-5, 3.1e-4)	(2.2e-7, 3.9e-7, 1.8e-6)
9	(2002, 92377)	3	0:13:13	0:18:31	0:43:28	(2.2e-4, 7.3e-4, 8.4e-4)	(1.1e-6, 2.5e-6, 2.9e-6)
10	(3003, 184755)	3	3:53:13	3:58:15	4:02:11	(2.3e-3, 2.4e-3, 4.1e-3)	(4.7e-7, 1.2e-6, 4.2e-6)

TABLE 5
Computational results for random unconstrained polynomial optimization of degree 10

where f/F is a Gaussian random vector/matrix of a proper dimension. Here $[x^d]$ denotes the vector of monomials of degree equal to d. The computational results are shown in Tables 2-5. There #Inst denotes the number of randomly generated instances, and (N, m) denotes the size of the corresponding SDP relaxation (2.6)-(2.7).

When $f(x)$ has degree 4 ($d = 2$), SDP relaxation (2.6)-(2.7) is solved quite well. For $n = 20 \sim 30$, the computation takes up to half a minute; for $n = 40 \sim 60$, it takes a couple of minutes; for $n = 70 \sim 80$, it takes less than one hour; for $n = 90 \sim 100$, it takes a few hours. When $f(x)$ has degree 6 ($d = 3$), for $n = 15$, solving (2.6)-(2.7) takes up to a few hours; for $n = 20 \sim 25$, it takes a couple of hours. When $f(x)$ has degree 8 ($d = 4$), for $n = 10$, solving (2.6)-(2.7) takes a couple of minutes; for $n = 12 \sim 15$, it takes about one to ten hours. When $f(x)$ has degree 10 ($d = 5$), for $n = 8$, solving (2.6)-(2.7) takes a couple of minutes; for $n = 9$, it takes less than one hour; for $n = 10$, it takes a few hours. From Tables 2 to 5, we can see that the SDP relaxations are solved successfully. The obtained solutions for polynomial optimization are also reasonably very well. They are slightly less accurate than the computed solutions of the SDP relaxation itself. This is probably because the SDP relaxation (2.6)-(2.7) is not exact in minimizing the generated polynomials.

The computations here show that Algorithm 3.3 could solve large scale polynomial optimization problems. A quartic polynomial optimization with 100 variables could be solved within a couple of hours on a regular computer. This is almost impossible by using SDP solvers based on interior point methods.

Example 4.3 (Sensor Network Localization). Given a graph $G = (V, E)$ and a distance for each edge, the sensor network localization problem is to find locations of vertices so that their distances are equal to the desired ones. This problem can be formulated as follows: find a sequence of unknown vectors (*sensors*) $u_1, u_2, \ldots, u_s \in \mathbb{R}^k$ (typically $k = 1, 2, 3$, we focus on $k = 2$ in this example) such that the distances between these sensors and some other fixed vectors (*anchors*) $a_1, \ldots, a_\ell$ are equal to given distances. Recently, there is much work on solving sensor network localization by SDP techniques, like [2, 13, 19]. Given edge subsets

$$\mathcal{E}_S \subset \{(i,j) : 1 \le i < j \le s\}, \quad \mathcal{E}_A = \{(i,j) : 1 \le i \le s, 1 \le j \le \ell\},$$

for every $(i,j) \in \mathcal{E}_S$, let d_{ij} be the distance between u_i and u_j, and for every $(i,j) \in \mathcal{E}_A$, let e_{ij} be the distance between u_i and a_j. Denote $u_i = (x_{ki-k+1}, \ldots, x_{ki})$ for $i = 1, \ldots, s$. The sensor network localization problem is equivalent to finding coordinates $x_{k1}, \ldots, x_{ks}$ satisfying the equations

$$\|u_i - u_j\|_2^2 = d_{ij}^2 \quad \forall\, (i,j) \in \mathcal{E}_S, \qquad \|u_i - a_j\|_2^2 = e_{ij}^2 \quad \forall\, (i,j) \in \mathcal{E}_A.$$

It is also equivalent to the quartic polynomial optimization problem

$$\min_{u_1,\ldots,u_s} \quad \sum_{(i,j)\in\mathcal{E}_S} \left(\|u_i - u_j\|_2^2 - d_{ij}^2\right)^2 + \sum_{(i,j)\in\mathcal{E}_A} \left(\|u_i - a_j\|_2^2 - e_{ij}^2\right)^2. \tag{4.3}$$

Typically, it is large scale. We use `SDPNAL` to solve its SDP relaxation (2.6)-(2.7). To test its performance, we randomly generate sensors $u_1, \ldots, u_s$ from the square

#sensor	#Inst	time (min, med, max)			RMSD (min, med, max)	errsdp (min, med, max)
15	15	0:00:24	0:00:52	0:02:02	(8.1e-6, 2.4e-5, 1.4e-4)	(1.1e-7, 4.2e-7, 1.6e-6)
20	15	0:02:04	0:03:19	0:09:12	(1.5e-5, 5.5e-5, 1.5e-4)	(2.9e-7, 4.4e-7, 2.0e-6)
25	10	0:14:18	0:35:02	1:12:21	(4.3e-5, 8.7e-5, 2.2e-4)	(2.4e-7, 6.3e-7, 1.6e-6)
30	10	1:22:18	2:44:05	5:51:36	(2.3e-5, 2.3e-4, 2.7e-3)	(9.2e-8, 1.8e-6, 5.3e-4)
35	3	09:59:35	19:13:58	27:08:37	(1.3e-3, 1.6e-3, 2.2e-3)	(6.5e-6, 5.1e-5, 6.5e-4)
40	3	48:33:59	50:54:34	61:19:58	(1.2e-3, 1.6e-3, 2.7e-3)	(2.2e-3, 3.2e-3, 4.0e-3)

TABLE 6
Computational results for sensor network localization problems.

$[-0.5, 0.5] \times [-0.5, 0.5]$. Fix four anchors as $(\pm 0.45, \pm 0.45)$. For each pair (i, j), select it to $\mathcal{E}_{\mathcal{S}}$ with probability 0.6 and to $\mathcal{E}_{\mathcal{A}}$ with probability 0.3. Then compute each distance d_{ij} and e_{ij}. After the SDP relaxation is solved, we use $Z^*(2:n+1, 1)$ as a starting point and apply function `lsqnonlin` in Matlab Optimization Toolbox to get a local minimizer $(\hat{u}_1, \ldots, \hat{u}_s)$ of (4.3) (we use the technique that was proposed in [13]). The errors of computed locations are measured by the Root Mean Square Distance $\text{RMSD} = (\frac{1}{s}\Sigma_{i=1}^{s}\|\hat{u}_i - u_i^*\|^2)^{1/2}$, as used in [2].

The computational results are shown in Table 6. We can see that the SDP relaxation of (4.3) is solved reasonably well. In many instances, FEC is not satisfied, so we can only get a local minimizer of (4.3) by using the technique from [13]. The true locations of sensors are found with small errors. Possible reasons for FEC fails might be: the SDP relaxation was not solved accurately enough, or it is not exact for (4.3).

We would like to remark that the SDP relaxation (2.6)-(2.7) for (4.3) does not exploit the sparsity pattern. There exists work of using sparse SDP or SOS type relaxations for solving sensor network localization problems (e.g., Kim et al. [13] and Nie [19]). Generally, solving (2.6)-(2.7) for (4.3) is much more difficult than solving its sparse versions like in [13, 19]. The numerical experiments in [13, 19] show that exploiting sparsity will allow us to solve much bigger problems. However, in the view of quality of approximations, sparse SDP relaxations are typically weaker than the general dense one. Thus, it is still meaningful if we can solve (2.6)-(2.7) for large scale sensor network localization problems.

4.2. Homogeneous polynomial optimization.

Example 4.4. Minimize the following square free quartic form over $\mathbb{S}^{n-1}$

$$\sum_{1\le i<j<k<\ell\le n} (-i-j+k+\ell)x_ix_jx_kx_\ell.$$

For $n = 50$, the resulting SDP (2.14)-(2.15) has size $(N, m) = (1275, 292824)$. Solving (2.14)-(2.15) takes about 38 minutes. The error of the computed solution for the SDP relaxation is around $8 \cdot 10^{-8}$. The computed lower bound $f_{sos}^{hmg} \approx -140.4051$. The optimal Z^* has rank two and FEC holds, so we get two optimizers. Their errors are around $2 \cdot 10^{-7}$.

Example 4.5. Minimize the following sextic form over $\mathbb{S}^{n-1}$

$$\sum_{1\le i\le n} x_i^6 + \sum_{1\le i\le n-1} x_i^3x_{i+1}^3.$$

For $n = 20$, the size of the resulting SDP (2.14)-(2.15) is $(N, m) = (1540, 177099)$. In this problem, we set parameter $\texttt{Tol} = 10^{-10}$ in running `SDPNAL` (For the default choice

n	(N,m)	# Inst	time (min, med, max)			errsdp (min, med, max)
20	(210, 8854)	20	0:00:06	0:00:11	0:00:26	(1.2e-7, 6.7e-7, 1.0e-6)
30	(465, 40919)	20	0:00:45	0:01:21	0:01:52	(2.0e-7, 5.2e-7, 1.1e-6)
40	(820, 123409)	10	0:02:31	0:05:19	0:08:58	(4.1e-7, 7.4e-7, 1.6e-6)
50	(1275, 292824)	10	0:13:30	0:19:10	0:29:29	(4.5e-7, 5.9e-7, 9.6e-7)
60	(1830, 595664)	5	0:44:19	1:05:32	1:47:51	(2.4e-7, 5.6e-7, 3.8e-6)
70	(2485, 1088429)	5	2:33:24	4:20:13	5:07:37	(4.2e-7, 6.2e-7, 7.9e-7)
80	(3240, 1837619)	3	7:31:21	9:43:40	10:52:27	(3.6e-7, 4.4e-7, 7.8e-7)
90	(4095, 2919734)	3	17:10:41	17:44:02	18:45:28	(2.1e-7, 3.7e-7, 3.7e-7)

TABLE 7
Computational results for stability number of random graphs

`Tol` $= 10^{-6}$, `SDPNAL` does not converge very well for this example), i.e., we terminate the computation when $\max\{R_p, R_D\} < 10^{-10}$. Solving (2.14)-(2.15) takes about 2.5 hours. The computed solution of the SDP relaxation has error around $1 \cdot 10^{-5}$. The computed lower bound $f_{sos}^{hmg} \approx 1.1451 \times 10^{-4}$. The computed optimal Z^* has rank one and we get one global minimizer from it. Its error is around $1.3 \cdot 10^{-5}$.

Example 4.6. Minimize the following sextic form over $\mathbb{S}^{n-1}$

$$\sum_{1 \le i < j < k \le n} x_i^2 x_j^2 x_k^2 + x_i^3 x_j^2 x_k + x_i^2 x_j^3 x_k + x_i x_j^3 x_k^2.$$

For $n = 20$, the resulting SDP (2.14)-(2.15) has size $(N, m) = (1540, 177099)$. Solving (2.14)-(2.15) takes about 1.8 hours. The error of the computed solution of the SDP relaxation is around $1.7 \cdot 10^{-6}$. The computed lower bound $f_{sos}^{hmg} \approx -0.3827$. The computed optimal Z^* has rank one, so we get one global minimizer. Its error is around $7.4 \cdot 10^{-7}$.

An important application of homogenous polynomial optimization (2.8) is computing stability numbers of graphs.

Example 4.7 (Stability numbers of graphs). Let $G = (V, E)$ be a graph with $|V| = n$. The stability number $\alpha(G)$ is the cardinality of the biggest stable subset(s) (their vertices are not connected by any edges) of V. It was shown in Motzkin and Straus [17] (also see De Klerk and Pasechnik [7]) that

$$\alpha(G)^{-1} = \min_{x \in \Delta_n} \quad x^T (A + I_n) x,$$

where Δ_n is the standard simplex in $\mathbb{R}^n$ and A is the adjacency matrix associated with G. If replacing every $x_i \ge 0$ by x_i^2, we get

$$\alpha(G)^{-1} = \min_{\|x\|_2 = 1} \quad \sum_{i=1}^n x_i^4 + 2 \sum_{(i,j) \in E} x_i^2 x_j^2. \tag{4.4}$$

This is a quartic homogeneous polynomial optimization. When a lower bound f_{sos}^{hmg} of (4.4) is computed from its SDP relaxation, we round $\left(f_{sos}^{hmg}\right)^{-1}$ to the nearest integer which will be used to estimate $\alpha(G)$.

We generate random graphs G, and solve the SDP relaxation of (4.4). The generation of random graphs is in a similar way as in Bomze and De Klerk [3, Section 6]. For $n = 20, 30, 40, 50, 60$, we generate random graphs $G = (V, E)$ with $|V| = n$. Select a random subset $M \subset V$ with $|M| = n/2$. The edges $e_{ij} (\{i, j\} \not\subset M)$ are generated with probability $\frac{1}{2}$. The computational results are in Table 7. As one can see, for $n = 20, 30, 40, 50$, solving (2.14)-(2.15) takes less than half an hour; for $n = 60, 70,$

it takes a few hours; for $n = 80, 90$, it takes 7 to 19 hours. In all the instances, we get correct stability numbers. All the SDP relaxations themselves are also solved successively.

4.3. Constrained polynomial optimization.

Example 4.8. Minimize the sextic polynomial

$$\sum_{i=1}^{n} ix_i^3 + \sum_{1\le i<j\le n} (i+j)x_i^3x_j^3$$

over the unit ball $B(0,1)$. We apply the 3^{rd} Lasserre's relaxation (2.18). The resulting cone $\mathcal{K}$ has 2 blocks. When $n = 20$, solving (2.21)-(2.22) takes about 34 minutes. The computed lower bound $f_{sos}^{con} = -20$. The optimal Z^* has rank one, and we get the solution:

$$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1).$$

It is feasible and a global minimizer.

Example 4.9. Consider the polynomial optimization

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \sum_{1\le i<j<k\le n/2} \big(x_ix_jx_k + x_{\frac{n}{2}+i}x_{\frac{n}{2}+j}x_{\frac{n}{2}+k} + x_ix_jx_kx_{\frac{n}{2}+i}x_{\frac{n}{2}+j}x_{\frac{n}{2}+k}\big) \\ s.t. \quad & x_1^4 + \cdots + x_{\frac{n}{2}}^4 \le 1, \quad x_{\frac{n}{2}+1}^4 + \cdots + x_n^4 \le 1, \end{aligned}$$

where n is even. Since the degree is 6, we apply the 3^{rd} Lasserre's relaxation (2.18). The cone $\mathcal{K}$ has 3 blocks. For $n = 20$, solving (2.21)-(2.22) takes about 3.2 hours. The computed solution of the SDP relaxation has error around $1 \cdot 10^{-7}$. The computed lower bound $f_{sos}^{con} \approx -38.8840$. The computed optimal Z^* has rank one, and we get a global optimal solution. Its error is around $1 \cdot 10^{-7}$.

Example 4.10. Minimize the quartic polynomial

$$\sum_{1\le i<j\le n} \big(x_ix_j + x_i^2x_j - x_j^3 - x_i^2x_j^2\big)$$

over the hypercube $[-1,1]^n = \{x \in \mathbb{R}^n : x_i^2 \le 1\}$. We apply the 2^{nd} Lasserre's relaxation (2.18). The resulting cone $\mathcal{K}$ has $n+1$ blocks. For $n = 50$, solving (2.21)-(2.22) takes about 2.3 hours. The error of the computed solution for the SDP relaxation is around 10^{-6}. The computed lower bound $f_{sos}^{con} \approx -1250$. The computed optimal Z^* does not satisfy FEC. So, we use $Z^*(2:n+1,1)$ as a starting point, and get the following local minimizer x^* (by function `fmincon` in Matlab Optimization Toolbox):

$$\begin{aligned} &(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,\\ &-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,-1,1,1,1). \end{aligned}$$

Its objective value $f(x^*) = -1248$ is slightly bigger than the lower bound f_{sos}^{con}.

Example 4.11 (Random polynomials). We test the performance of Algorithm 3.3 (implemented by `SDPNAL`) in minimizing polynomials over the unit ball. Generate $f(x)$ randomly as

$$f(x) = \sum_{\alpha\in\mathbb{N}^n:|\alpha|\le 2d} f_\alpha x^\alpha,$$

$(n,2d)$	#Inst	time (min, med, max)			errsol(min, med, max)	errsdp(min, med, max)
(30,4)	10	0:00:28	0:00:52	0:02:47	(5.6e-8, 1.3e-6, 6.9e-6)	(1.3e-7, 8.1e-7, 2.9e-6)
(40,4)	10	0:03:35	0:06:38	0:10:32	(8.8e-8, 1.8e-6, 9.5e-6)	(2.2e-7, 1.0e-6, 4.5e-6)
(50,4)	3	0:20:34	0:22:18	0:24:59	(5.7e-6, 5.6e-6, 7.0e-6)	(2.7e-6, 2.8e-6, 3.4e-6)
(60,4)	3	0:35:02	1:20:15	1:20:38	(1.5e-7, 3.5e-6, 2.5e-5)	(1.7e-7, 1.7e-6, 1.2e-5)
(20,6)	3	0:36:31	0:49:17	0:56:35	(8.5e-7, 2.7e-6, 4.4e-6)	(5.8e-7, 1.3e-6, 2.7e-6)
(12,8)	3	0:27:11	0:44:06	0:59:30	(5.5e-7, 2.8e-6, 9.0e-6)	(9.0e-7, 1.3e-6, 4.2e-6)
(9,10)	3	0:16:31	0:36:05	0:40:53	(2.6e-7, 3.3e-6, 1.4e-5)	(2.7e-7, 1.6e-6, 6.3e-6)
(80,4)	3	10:52:30	15:12:40	15:57:30	(5.3e-6, 5.5e-6, 2.2e-1)	(2.6e-6, 2.6e-6, 2.7e-3)
(25,6)	3	10:38:04	11:00:48	12:57:59	(5.9e-3, 6.6e-3, 1.4e-2)	(3.6e-3, 5.8e-3, 6.1e-3)

TABLE 8
Computational results for random constrained polynomial optimization

where the coefficients f_α are Gaussian random variables. We solve the SDP relaxation (2.21)-(2.22) by `SDPNAL`. The cases of degrees $4, 6, 8, 10$ are tested. The computational results are in Table 8. When $(n, 2d) = (80, 4)$ or $(25, 6)$, the SDP relaxations are not solved very well sometimes. This is probably because of the incurred ill-conditioning. In all the other cases, the SDP relaxations are solved quite well, and accurate global minimizers are found.

5. Some discussions. In this section, we discuss some numerical issues about the performance of regularization methods in solving SDP relaxations for large scale polynomial optimization problems.

5.1. Scaling polynomial optimization. SDP relaxations arising from polynomial optimization are harder to solve than general SDP problems. A reason for this is that the polynomials are not scaled very well sometimes. For instance, if the optimal Z^* has rank 1, then $Z^* = [x^*]_d[x^*]_d^T$ (x^* is a minimizer) has entries of the form

$$1, x_1^*, \ldots, (x_1^*)^2, \ldots, \ldots, (x_1^*)^{2d}, \ldots, (x_n^*)^{2d}.$$

Clearly, if some coordinate x_i^* is small or big, then Z^* is badly scaled and its entries Z_{ij}^* easily get underflow/overflow during the computation. This might cause severe ill-conditioning in computations and make the computed solutions less accurate. Scaling is a useful approach to overcome this issue. In [10, 20], it was pointed out that scaling is important in solving polynomial optimization problems efficiently. Generally, there is no simple rule to select the best scaling factor. In the following, we propose a practical scaling procedure.

Let $s = (s_1, \ldots, s_n) > 0$ and scale x to $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)$ as

$$x \quad = \quad (s_1\hat{x}_1, \ldots, s_n\hat{x}_n).$$

Then $f(x)$ is scaled to be the polynomial $f(s_1\hat{x}_1, \ldots, s_n\hat{x}_n)$ in $\hat{x}$. The best scaling factor s should be such that the global minimizers of the scaled polynomial have coordinates close to one or negative one. This is difficult because optimizers are usually unknown. However, as the algorithm runs, one often gets close to minimizers and would estimate them from the computations. Typically, we only need to scale the problem when the algorithm fails to converge. Sometimes, we might need to do scaling several times. From our experiences, a practical scaling procedure is:

Step 1 If Algorithm 3.3 converges well, we do no scaling and let it run; otherwise, select a scaling vector $s = (s_1, \ldots, s_n) > 0$ as:

$$s_i = \begin{cases} \tau & \text{if } |y_{e_i}| \le \tau, \\ |y_{e_i}| & \text{otherwise.} \end{cases} \tag{5.1}$$

Here $\tau > 0$ is fixed and y is the most recent update for an optimal y^* of (2.7).

Step 2 Scale $f(x)$ as $f(s_1\hat{x}_1, \ldots, s_n\hat{x}_n)$. Go back to Step 1 and solve the scaled polynomial optimization again.

In the above, $\tau > 0$ is usually (but not too) small, because the coefficients of the scaled polynomial $f(s_1\hat{x}_1, \ldots, s_n\hat{x}_n)$ should not be very tiny. We use $\tau = 10^{-3}$ in the examples below.

Example 5.1. Consider the polynomial optimization

$$\min_{x\in\mathbb{R}^n} \quad x_1^4 + \ldots + x_n^4 + \sum_{1\le i<j<k\le n} x_i x_j x_k. \tag{5.2}$$

For this kind of polynomials, its global minimizers usually have large negative values and lead to ill-conditioning of the SDP relaxation (cf. [20, §5.1]). Here we show the importance of scaling for the case $n = 20$.

Iter	time	low. bdd.	sdp err.
1	0:01:15	-1.0806e+7	0.7555
2	0:01:15	-1.9444e+7	0.0460
3	0:00:37	-2.1883e+7	0.0082
4	0:01:16	-2.2266e+7	2.4e-6

TABLE 9
Results of scaling process for Example 5.1.

We use the scaling procedure described above. The computational results are shown in Table 9. The "low. bdd." there stands for the computed optimal value of SDP relaxation (2.6)-(2.7), which is always a lower bound of the global minimum, and "sdp err." stands for the error of the computed solution of (2.6)-(2.7), which is defined in (4.2). It takes four times of scaling to solve the SDP relaxation reasonably well.

Example 5.2. Consider the least square problem (Watson function [16]):

$$\min_{x\in\mathbb{R}^n} \quad \sum_{i=1}^{m} f_i^2(x). \tag{5.3}$$

Here $n = 30$ and the polynomials f_i are defined as follows:

$$f_i(x) = \sum_{j=2}^{n} (j-1)x_j t_i^{j-2} - \left(\sum_{j=1}^{n} x_j t_i^{j-1}\right)^2 - 1,\ t_i = \tfrac{i}{29},\ \ 1 \le i \le 29, \tag{5.4}$$

and $f_{30} = x_1$, $\ f_{31} = x_2 - x_1^2 - 1$. Its SDP relaxation (2.6)-(2.7) has size $(N, m) = (496, 46375)$. We solve it by the scaling procedure mentioned earlier. The results are in Table 10. It takes six times of scaling to solve the SDP relaxation reasonably well.

Remark 5.3. In each step of the scaling process, we need to solve a new SDP problem of the same size as the earlier one. As shown in Section 4, sometimes the SDP relaxations in polynomial optimization could be solved very well without scaling. But this is not always the case (e.g., Examples 5.1 and 5.2). Typically, we need to do scaling only when Algorithm 3.3 has troubles to solve a problem. The performance of Algorithm 3.3 is bad when the SDP problem is ill-conditioned or has degeneracy. Our experiments show that scaling can help solve the problem more efficiently.

Iter	time	low. bdd.	sdp err.
1	0:28:09	-9.1556	0.9955
2	0:45:34	0.0134	8.1e-3
3	0:30:40	0.1468	9.1e-4
4	0:26:33	0.1298	4.7e-4
5	0:25:18	0.0969	3.1e-4
6	0:18:10	0.0648	8.3e-5

TABLE 10
Results of scaling process for Example 5.2.

5.2. Regularization methods are suitable for large scale polynomial optimization. As we have seen in Introduction, a major issue of interior point methods is that in each step one needs to solve an $m \times m$ linear system and two $N \times N$ matrix equations. This would be a big restriction in applications if m is huge, because it requires storing an $m \times m$ matrix in computer and $\mathcal{O}(m^3)$ arithmetic operations. Unfortunately, SDP relaxations from polynomial optimization have an unfavorable property that $m = \mathcal{O}(N^2)$. As shown in Table 1, in minimizing a general quartic polynomial of 100 variables, the SDP relaxation has m greater than 4 million. To solve such an SDP relaxation by interior point methods, one needs to store a square matrix of length bigger than 4 million in memory. On a regular computer, this is almost impossible. However, regularization methods requires much less memory storage. In Algorithm 3.3, in each inner loop, we still need to solve the linear system (3.7) which is also $m \times m$. But, the Hessian $\nabla_y^2 \varphi_\sigma(Y, y)$ does not need to be explicitly formulated. Actually, the authors of [29] showed that the matrix vector product $\nabla_y^2 \varphi_\sigma(Y, y) \cdot z$ would be evaluated in $\mathcal{O}(m)$ arithmetic operations and the memory requirement for (3.7) has linear order in m. Because of this special feature, CG type methods are very suitable for solving (3.7). This property has been successfully used by software `SDPNAL`.

For unconstrained polynomial optimization, its SDP relaxation has an attractive feature. From the construction of A_α in subsection 2.1, we can easily see that distinct A_α's have no common nonzero entries. Thus, the matrices A_α in (2.4) are orthogonal to each other, and the matrix $\mathcal{A}\mathcal{A}^*$ is diagonal. In this case, Algorithm 3.2 is easily implementable, because its every step only involves solving a diagonal linear system and computing an eigenvalue decomposition. In Algorithm 3.3, the diagonal $(\mathcal{A}\mathcal{A}^*)^{-1}$ would be used as a preconditioner for (3.7) in CG iterations.

5.3. Other numerical methods. To the authors' best knowledge, there are few efficient numerical methods for solving large scale polynomial optimization problems. One method that might be useful in applications is the low rank method proposed by Burer and Monteiro [4] (implemented in software `SDPLR` [5]). In some cases, the dual optimal Z^* of SDP relaxations might have low rank. Thus, in such situations, `SDPLR` would be applied to solve the dual SDP relaxation like (2.7) or (2.22) (not the primal SDP relaxation (2.6) or (2.21), since X^* typically has high rank). We tested `SDPLR` on some examples in this paper. Its performance is similar to `SDPNAL`. However, `SDPLR` is less attractive theoretically and suitable only when Z^* has low rank. This is because the basic idea of `SDPLR` is to change SDP into a nonlinear programming problem via matrix factorization, and typically one would only get a local optimizer. However, by `SDPLR`, it is not guaranteed to get an optimizer of the SDP relaxation. Moreover, even if an optimizer of SDP is obtained, its optimality can not be certified. A reason for

this is that SDPLR is not a primal-dual type method, and typically a primal-dual pair is required to check optimality. On the other hand, the computational performance of SDPLR is promising. It is an interesting future work to investigate properties of the low rank method in solving polynomial optimization.

There are interesting recent work on solving large scale polynomial optimization problems by other methods. Bertsimas, Freund and Sun [1] proposed an accelerated first order method to solve unconstrained polynomial optimization problems. A nice theoretical property of first order type methods is that there are bounds on the complexity of computations, as proved in [1]. Henrion and Malick [11, 12] proposed a projection method for solving conic optimization and SOS relaxations. These methods can solve bigger problems than the interior point methods do, but they might take a big number of iterations to get an accurate optimal solution and generally its convergence is slow. In practical computations of solving big polynomial optimization problems, Algorithm 3.3 typically has faster convergence, because it uses second order information (e.g., approximate Newton directions).

5.4. Convergence and nondegeneracy. The performance of Algorithm 3.3 is not always very good for solving SDP relaxations in polynomial optimization. As we have seen earlier, a typical reason is the ill-conditioning. Another reason might be the degeneracy of the SDP relaxations. In [29], it was shown that if the SDP problem is nondegenerate, then Algorithm 3.3 has good convergence; otherwise, it might converge very badly or even does not converge. Generally, it is difficult to check in advance whether an SDP relaxation is degenerate or not. For SDP relaxation (2.6)-(2.7) in unconstrained polynomial optimization, or (2.21)-(2.22) in constrained optimization, a typical case for it to be degenerate is that a polynomial optimization problem has several distinct global minimizers. To see this for the unconstrained polynomial optimization (2.1), suppose it has two distinct global minimizers u^*, v^* and the SOS relaxation (2.2) is exact. Then, the optimal values of (2.6) and (2.7) are equal, and (2.7) has two distinct optimal Z^* (being $[u^*]_d[u^*]_d^T$ and $[v^*]_d[v^*]_d^T$). This implies the primal SDP relaxation (2.6) is degenerate. The situation is similar for constrained polynomial optimization. From this observation, Algorithm 3.3 might not be very efficient if the SDP relaxation is exact and there are more than one distinct optimizers. Of course, Algorithm 3.3 might still work if an SDP problem is degenerate, like in Example 4.1. But this is occasional and typically not the case in practice.

Acknowledgement The authors would like to thank Xinyuan Zhao, Defeng Sun and Kim-Chuan Toh for sharing their software SDPNAL, and Gabor Pataki for comments on the degeneracy of SDP. They also thank Bill Helton and Igor Klep for fruitful discussions on this work.

REFERENCES

[1] D. Bertsimas, R. Freund, and X. Sun. An accelerated first-order method for solving unconstrained SOS polynomial optimization problems. *Optimization Methods and Software*, to appear.

[2] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. *Proc. 3rd IPSN*, pp. 46–54, 2004.

[3] I.M. Bomze and E. de Klerk. Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. *Journal of Global Optimization*, Vol. 24, No. 2, pp. 163–185, 2002.

[4] S. Burer and R. Monteiro. A nonlinear programming algorithm for solving semidefinite pro-

grams via low-rank factorization. *Mathematical Programming*, Ser. B, Vol. 95, No. 2, pp. 329–357, 2003.

[5] S. Burer. SDPLR: a C package for solving large-scale semidefinite programming problems. `http://dollar.biz.uiowa.edu/~sburer`

[6] R. Curto and L. Fialkow. Truncated K-moment problems in several variables. *Journal of Operator Theory*, 54, pp. 189–226, 2005.

[7] E. de Klerk and D.V. Pasechnik. Approximating of the stability number of a graph via copositive programming. *SIAM Journal on Optimization*, Vol. 12, No. 4, pp. 875–892, 2002.

[8] K. Fujisawa, Y. Futakata, M. Kojima, S. Matsuyama, S. Nakamura, K. Nakata, and M. Yamashita. SDPA-M (SemiDefinite Programming Algorithm in MATLAB), `http://homepage.mac.com/klabtitech/sdpa-homepage/download.html`

[9] D. Henrion and J. Lasserre. Detecting global optimality and extracting solutions in GloptiPoly. *Positive polynomials in control (Eds. D. Henrion, A. Garulli), Lecture Notes on Control and Information Sciences*, Vol. 312, pp. 293–310, Springer, Berlin, 2005.

[10] D. Henrion and J. Lasserre. Gloptipoly: Global optimization over polynomials with matlab and sedumi. *ACM Transactions on Mathematical Software*, Vol. 29, No. 2, pp. 165–194, 2003.

[11] D. Henrion and J. Malick. Projection methods for conic feasibility problems: applications to polynomial sum-of-squares decompositions. *Optimization Methods and Software*, Vol. 26, No. 1, pp. 23–46, 2011.

[12] D. Henrion and J. Malick. Projection methods for conic optimization. LAAS-CNRS Research Report No.10730, 2010. *Handbook of Semidefinite, Cone and Polynomial Optimization* (Eds. M. Anjos and J. B. Lasserre), Springer, 2011.

[13] S. Kim, M. Kojima and H. Waki. Exploiting sparsity in SDP relaxation for sensor network localization. *SIAM Journal on Optimization*, Vol. 20, No. 1, pp. 192–215, 2009.

[14] J. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, Vol. 11, No. 3, pp. 796–817, 2001.

[15] J. Malick, J. Povh, F. Rendl and A. Wiegele. Regularization methods for semidefinite programming. *SIAM Journal on Optimization*, Vol. 20, No. 1, pp. 336–356, 2009.

[16] J. Moré, B. Garbow and K. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Soft.*, Vol. 7, pp. 17–41, 1981.

[17] T.S. Motzkin and E.G. Straus. Maxima for graphs and a new proof of a theorem of Turán. *Canadian J. Math.*, 17, pp. 533–540, 1965.

[18] J. Nie and J. Demmel. Sparse SOS relaxations for minimizing functions that are summations of small polynomials. *SIAM Journal On Optimization*, Vol. 19, No. 4, pp. 1534–1558, 2008.

[19] J. Nie. Sum of squares method for sensor network localization. *Computational Optimization and Applications*, Vol. 43, No. 2, pp. 151–179, 2009.

[20] P. Parrilo and B. Sturmfels. Minimizing polynomial functions. *Proceedings of the DIMACS Workshop on Algorithmic and Quantitative Aspects of Real Algebraic Geometry in Mathematics and Computer Science (March 2001)* (eds. S. Basu and L. Gonzalez-Vega), pp. 83–100, American Mathematical Society, 2003.

[21] P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, Ser. B, Vol. 96, No. 2, pp. 293–320, 2003.

[22] J. Povh, F. Rendl, and A. Wiegele. A boundary point method to solve semidefinite programs. *Computing*, Vol. 78, pp. 277–286, 2006.

[23] R.T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control and Optim.*, Vol. 14, No. 5, pp. 877–898, 1976.

[24] R.T. Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Math. Oper. Res.* , Vol. 1, No. 2, pp. 97–116, 1976.

[25] J.F. Sturm. SeDuMi 1.02:a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11 & 12, No. 1-4, pp. 625–653, 1999.

[26] K.C. Toh, M.J. Todd, and R.H. Tutuncu. SDPT3: a Matlab software package for semidefinite programming. *Optimization Methods and Software*, Vol. 11, pp. 545–581, 1999.

[27] H. Waki, S. Kim, M. Kojima and M. Muramatsu. Sums of squares and semidefinite programming relaxations for polynomial optimization problems with Structured Sparsity. *SIAM Journal on Optimization*, Vol.17, No. 1, pp. 218–242, 2006.

[28] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of semidefinite programming*, Kluwer, Publisher, 2000.

[29] X.Y. Zhao, D.F. Sun, and K.C. Toh. A Newton-CG Augmented Lagrangian method for semidefinite programming. *SIAM Journal on Optimization*, Vol. 20, No. 4, pp. 1737–1765, 2010.

[30] X.Y. Zhao, D.F. Sun, and K.C. Toh. *SDPNAL* version 0.1 – a MATLAB software for semidefinite programming based on a semi-smooth Newton-CG augmented Lagrangian method. `http://www.math.nus.edu.sg/~mattohkc/SDPNAL.html`