

# Regularization Methods for Sum of Squares Relaxations in Large Scale Polynomial Optimization

Jiawang Nie\*

September 18, 2009

## Abstract

We study how to solve sum of squares (SOS) and Lasserre's relaxations for large scale polynomial optimization. When interior-point type methods are used, typically only small or moderately large problems could be solved. This paper proposes the regularization type methods which would solve significantly larger problems. We first describe these methods for general conic semidefinite optimization, and then apply them to solve large scale polynomial optimization. Their efficiency is demonstrated by extensive numerical computations. In particular, a general dense quartic polynomial optimization with 100 variables would be solved on a regular computer, which is almost impossible by applying prior existing SOS solvers.

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

**AMS subject classification** 65K05, 90C22

## 1 Introduction

Consider the polynomial optimization

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

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 various sum of squares (SOS) or Lasserre's relaxations. The basic idea is approximating nonnegative polynomials by SOS polynomials. One of their attracting properties is they are equivalent to certain semidefinite programming (SDP) problems. Thus they would be solved by SDP packages (like SDPT3 [36], SeDuMi [33], SDPA [7]). Numerical experiments show they give good approximations for (1.1), which is NP-hard. For instance, in minimizing a general class of polynomials, SOS relaxations work very well in finding their global minimum [28]. Typically SOS and Lasserre's relaxations are very successful in solving

---

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

(1.1). We refer to [9, 13, 17, 23, 28, 27, 34] for the work in this area. However, their applications are very limited in solving big problems. For instance, to minimize a quartic polynomial, it is very difficult to solve its SOS relaxation on a regular computer when it has more than 20 variables. So far, SOS relaxations can only be solved for small or moderately large problems, which severely affects their practical applications. The motivation of this paper is proposing cheaper numerical methods for solving big polynomial optimization.

A standard semidefinite programming problem is

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

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. Denote by  $\mathcal{S}_+^N$  the cone of positive semidefinite matrices in  $\mathcal{S}^N$ . The  $C \in \mathcal{S}^N$  and  $b \in \mathbb{R}^m$  are constant, and  $\mathcal{A} : \mathcal{S}^N \rightarrow \mathbb{R}^m$  is a linear operator. The dual of (1.2) is

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

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

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

There is much work on solving SDP by interior point type methods. Most of them generate a sequence  $\{(X_k, y_k, Z_k)\}$  converging to an optimal solution. At each step, a search direction  $(\Delta X, \Delta y, \Delta Z)$  need to be computed. To compute  $\Delta y$ , typically an  $m \times m$  linear system need to be solved. To compute  $\Delta X$  and  $\Delta Z$ , two linear matrix equations need to be solved. Typically the cost for computing  $\Delta y$  is  $\mathcal{O}(m^3)$ . When  $m = \mathcal{O}(N)$ , the cost for computing  $\Delta y$  is  $\mathcal{O}(N^3)$ . In this case, solving SDP is not very expensive if  $N$  is not big (like less than 1,000). However, when  $m = \mathcal{O}(N^2)$ , the cost for computing  $\Delta y$  would be  $\mathcal{O}(N^6)$ , which is very expensive even for moderately large  $N$  (like 500). In this case, the cost for computing  $\Delta y$  is very big. It requires the storage of a matrix of dimension  $m \times m$  and  $\mathcal{O}(m^3)$  arithmetic operations. We refer to [35, 37, 39] for numerical methods solving SDP.

Unfortunately, SDP problems arising from polynomial optimization belong to the latter case that  $m = \mathcal{O}(N^2)$ , which is why the SDP packages based on interior point methods have difficulty in solving big SOS relaxations (like degree 4 with 100 variables). Now let us illustrate this. Let  $p(x)$  be a polynomial of degree  $2d$ . Then  $p(x)$  is SOS if and only if there exists a matrix  $X \in \mathcal{S}_+^N$  [27] such that  $p(x) = [x]_d^T X [x]_d$ , where  $[x]_d$  denotes the column vector of all monomials up to degree  $d$  in graded alphabetical ordering. Note the length of  $[x]_d$  is  $N = \binom{n+d}{d}$ . If we write  $p(x)$  as

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

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.

then  $p(x)$  being SOS is equivalent to the existence of  $X$  satisfying

$$\begin{aligned} A_\alpha \bullet X &= p_\alpha \quad \forall \alpha \in \mathbb{N}^n : |\alpha| \leq 2d, \\ X &\succeq 0. \end{aligned} \tag{1.5}$$

Here  $A_\alpha$  are certain constant symmetric matrices. The number of equalities is  $m = \binom{n+2d}{2d}$ . For any fixed  $d$ ,  $m = \mathcal{O}(N^2) = \mathcal{O}(n^{2d})$ . 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 can see, when interior point methods are applied to solve (1.5), at each step we need to solve a linear system of  $m$  variables and two matrix equations. To compute  $\Delta y$ , we need to store an  $m \times m$  matrix and do  $\mathcal{O}(n^{6d})$  arithmetic operations. This is very expensive for even moderately large  $n$  and  $d$ , and hence severely limits the applicability of SOS relaxations. On a regular computer, to solve a quartic polynomial optimization, it is quite difficult to apply interior point methods when there are more than 20 variables.

Recently there has been much work on designing efficient numerical methods on solving large scale SDPs. The so-called *regularization methods* are specially designed for solving SDPs whose number of equality constraints  $m$  is significantly bigger than the matrix length  $N$ . We refer to [19, 30, 40] for the work in this area. Their numerical experiments show that these methods are practical and efficient in solving large scale SDPs. Currently, these methods are only designed for standard SDPs of form (1.2) or its dual (1.3). However, the SDPs arising from polynomial optimization are usually in the form of conic semidefinite optimization, that is, they have block structures. Therefore, we first propose the regularization methods for solving general conic semidefinite optimization, and then apply them to solve polynomial optimization. Our numerical experiments show that these new methods could solve significantly bigger polynomial optimization problems efficiently.

This paper is organized as follows. Section 2 describes the regularization methods for solving general conic semidefinite optimization. Section 3 discusses how to solve unconstrained polynomial optimization. Section 4 discusses how to solve homogeneous polynomial optimization. Section 5 discusses how to solve constrained polynomial optimization. Section 6 concludes this paper with some discussions.

**Notations.** The symbol  $\mathbb{N}$  (resp.,  $\mathbb{R}$ ) denotes the set of nonnegative integers (resp., real numbers). For any  $t \in \mathbb{R}$ ,  $\lceil t \rceil$  (resp.,  $\lfloor t \rfloor$ ) denotes the smallest integer not smaller (resp., the largest integer not bigger) than  $t$ . For  $x \in \mathbb{R}^n$ ,  $x_i$  denotes the  $i$ -th component of  $x$ ,

that is,  $x = (x_1, \dots, x_n)$ . The  $\mathbb{S}^{n-1}$  denotes the  $n - 1$  dimensional unit sphere  $\{x \in \mathbb{R}^n : x_1^2 + \dots + x_n^2 = 1\}$ . For  $\alpha \in \mathbb{N}^n$ , denote  $|\alpha| = \alpha_1 + \dots + \alpha_n$ . The symbol  $\mathbb{N}_{\leq k}$  denotes the set  $\{\alpha \in \mathbb{N}^n : |\alpha| \leq k\}$ , and  $\mathbb{N}_k$  denotes  $\{\alpha \in \mathbb{N}^n : |\alpha| = k\}$ . For each  $i$ ,  $e_i$  denotes the  $i$ -th standard unit vector. The  $\mathbf{1}$  denotes the vector of all ones. For  $x \in \mathbb{R}^n$  and  $\alpha \in \mathbb{N}^n$ ,  $x^\alpha$  denotes  $x_1^{\alpha_1} \dots x_n^{\alpha_n}$ . The  $[x]_d$  denotes the vector of all monomials having degrees at most  $d$  in graded alphabetical ordering, that is,

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

and  $[x^d]$  denotes the homogeneous part of  $[x]_d$  having degree  $d$ , that is,

$$[x^d]^T = [x_1^d \quad x_1^{d-1} x_2 \quad \dots \dots \quad x_n^d].$$

For a polynomial  $p(x)$ ,  $\text{supp}(p)$  denotes the support of  $p(x)$ , i.e., the set of  $\alpha \in \mathbb{N}^n$  such that the monomial  $x^\alpha$  appears in  $p(x)$ . For a finite set  $S$ ,  $|S|$  denotes its cardinality. For a matrix  $A$ ,  $A^T$  denotes its transpose. The  $I_N$  denotes the  $N \times N$  identity matrix. For any vector  $u \in \mathbb{R}^N$ ,  $\|u\|_2 = \sqrt{u^T u}$  denotes the standard Euclidean norm.

## 2 Regularization methods for conic semidefinite optimization

This section describes the regularization methods for solving conic semidefinite optimization problems having block diagonal structures. They are natural generalizations of the regularization methods introduced in [19, 30, 40] for solving standard SDP problems.

### 2.1 Conic semidefinite optimization

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

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

It belongs to the space  $\mathcal{M} = \mathcal{S}^{N_1} \times \dots \times \mathcal{S}^{N_\ell}$ . Each  $X \in \mathcal{M}$  is a tuple  $X = (X_1, \dots, X_\ell)$  with every  $X_i \in \mathcal{S}^{N_i}$ . So  $X$  could be thought of as a symmetric block diagonal matrix. Thus  $X \in \mathcal{K}$  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  $X = (X_1, \dots, X_\ell) \in \mathcal{M}$  and  $Y = (Y_1, \dots, Y_\ell) \in \mathcal{M}$ , define their inner product as  $X \bullet Y = X_1 \bullet Y_1 + \dots + X_\ell \bullet Y_\ell$ . Denote by  $\|\cdot\|$  the norm in  $\mathcal{M}$  induced by this inner product. Note  $\mathcal{K}$  is a self-dual cone, that is,

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

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

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

The projection function  $(\cdot)_{\mathcal{K}}$  is Lipschitz continuous, because for any  $X, Y \in \mathcal{K}$

$$\|X_{\mathcal{K}} - Y_{\mathcal{K}}\|^2 = \|X - Y\|^2 + 2(X_{\mathcal{K}} \bullet Y_{-\mathcal{K}} + Y_{\mathcal{K}} \bullet X_{-\mathcal{K}}) - \|X_{-\mathcal{K}} - Y_{-\mathcal{K}}\|^2 \leq \|X - Y\|^2.$$

The projection norm function  $\|(\cdot)_{\mathcal{K}}\|$  is convex. To see this, note

$$\|(X)_{\mathcal{K}}\| = \min_{F \in \mathcal{K}} \|X + F\|.$$

Then, for any two  $X, Y \in \mathcal{M}$ , it holds

$$\begin{aligned} \left\| \left( \frac{X+Y}{2} \right)_{\mathcal{K}} \right\| &\leq \left\| \frac{X+Y}{2} - \frac{(X)_{-\mathcal{K}} + (Y)_{-\mathcal{K}}}{2} \right\| = \left\| \frac{X - (X)_{-\mathcal{K}}}{2} + \frac{Y - (Y)_{-\mathcal{K}}}{2} \right\| \\ &\leq \left\| \frac{X - (X)_{-\mathcal{K}}}{2} \right\| + \left\| \frac{Y - (Y)_{-\mathcal{K}}}{2} \right\| = \frac{1}{2} \|(X)_{\mathcal{K}}\| + \frac{1}{2} \|(Y)_{\mathcal{K}}\|. \end{aligned}$$

A general conic semidefinite optimization problem is

$$\begin{aligned} \min \quad & C \bullet X \\ \text{s.t.} \quad & \mathcal{A}(X) = b, X \in \mathcal{K}. \end{aligned} \tag{2.1}$$

Here  $C \in \mathcal{M}$ ,  $b \in \mathbb{R}^m$ , and  $\mathcal{A} : \mathcal{M} \rightarrow \mathbb{R}^m$  is a linear operator. The dual of (2.1) is

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & \mathcal{A}^*(y) + Z = C, Z \in \mathcal{K}. \end{aligned} \tag{2.2}$$

If  $\mathcal{A}(X)$  is a summation like

$$\mathcal{A}(X) = \mathcal{A}_1(X_1) + \dots + \mathcal{A}_\ell(X_\ell),$$

where every  $\mathcal{A}_i : \mathcal{S}^{N_i} \rightarrow \mathbb{R}^m$  is a linear operator, then its adjoint  $\mathcal{A}^*(y)$  has the form

$$\mathcal{A}^*(y) = (\mathcal{A}_1^*(y), \dots, \mathcal{A}_\ell^*(y)).$$

## 2.2 Regularization methods

There are two typical regularizations for a standard SDP: *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 (2.1) and its dual (2.2). The Moreau-Yosida regularization for (2.1) is

$$\begin{aligned} \min_{X, Y \in \mathcal{M}} \quad & C \bullet X + \frac{1}{2\sigma} \|X - Y\|^2 \\ \text{s.t.} \quad & \mathcal{A}(X) = b, X \in \mathcal{K}. \end{aligned} \tag{2.3}$$

Obviously (2.3) is equivalent to (2.1), because for each fixed feasible  $X \in \mathcal{M}$  the optimal  $Y \in \mathcal{M}$  in (2.3) is equal to  $X$ . The Augmented Lagrangian regularization for (2.2) is

$$\begin{aligned} \max_{y \in \mathbb{R}^m, Z \in \mathcal{M}} \quad & b^T y - (Z + \mathcal{A}^*(y) - C) \bullet Y - \frac{\sigma}{2} \|Z + \mathcal{A}^*(y) - C\|^2 \\ \text{s.t.} \quad & Z \in \mathcal{K}. \end{aligned} \tag{2.4}$$

When  $\mathcal{K} = \mathcal{S}_+^N$  is a single semidefinite cone, it was shown (see Section 2 of [19]) that for every fixed  $Y$ , (2.4) is the dual of the following optimization

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

By fixing  $y \in \mathbb{R}^m$  and optimizing over  $Z \succeq 0$ , Malick, Povh, Rendl, and Wiegale [19] further showed that (2.4) can be reduced to

$$\max_{y \in \mathbb{R}^m} b^T y - \frac{\sigma}{2} \|(\mathcal{A}^*(y) - C + Y/\sigma)_{\mathcal{K}}\|^2 + \frac{1}{2\sigma} \|Y\|^2. \quad (2.5)$$

**Proposition 2.1.** *Assume there exists  $\bar{X} \succ_{\mathcal{K}} 0$  such that  $\mathcal{A}(\bar{X}) = b$ . The following statements are equivalent:*

(i)  $(X, y, Z)$  is optimal for (2.3)-(2.4).

(ii)  $(X, y, Z)$  satisfies

$$\begin{cases} \mathcal{A}(X) = b, X = \sigma(Z + \mathcal{A}^*(y) - C) + Y, \\ X \in \mathcal{K}, Z \in \mathcal{K}, X \bullet Z = 0. \end{cases}$$

(iii)  $(X, y, Z)$  satisfies

$$\begin{cases} X = \sigma(Y/\sigma + \mathcal{A}^*(y) - C)_{\mathcal{K}}, \\ Z = -(Y/\sigma + \mathcal{A}^*(y) - C)_{-\mathcal{K}}, \\ \mathcal{A}\mathcal{A}^*y + \mathcal{A}(Z - C) = (b - \mathcal{A}(Y))/\sigma. \end{cases} \quad (2.6)$$

The proof of Proposition 2.6 in [19] would be naturally generalized from a single semidefinite cone  $\mathcal{S}_+^N$  to the cone  $\mathcal{K}$  of block diagonal positive semidefinite matrices. So we omit the proof of Proposition 2.1.

In (2.6),  $\mathcal{A}\mathcal{A}^*$  is a symmetric matrix defined in the way that

$$(\mathcal{A}\mathcal{A}^*y)^T z = \mathcal{A}^*(y) \bullet \mathcal{A}^*(z) \quad \forall y, z \in \mathbb{R}^m.$$

The optimality condition (2.6) leads to the following algorithm for solving (2.1):

**Algorithm 2.2.** Choose parameters  $\sigma > 0$ ,  $\epsilon > 0$ ,  $Y \in \mathcal{S}^N$ , set  $Z = 0$ . Do the loop:

*While* ( $\|b - \mathcal{A}(X)\| \geq \epsilon$  or  $\|C - Z - \mathcal{A}^*(y)\| \geq \epsilon$ )

Solve  $\mathcal{A}\mathcal{A}^*y = \mathcal{A}(C - Z) + (b - \mathcal{A}(Y))/\sigma$  for  $y$ .

Compute the projections

$$X = \sigma(Y/\sigma + \mathcal{A}^*(y) - C)_{\mathcal{K}}, \quad Z = -(Y/\sigma + \mathcal{A}^*(y) - C)_{-\mathcal{K}}.$$

Set  $Y := X$  and update  $\sigma$ .

When  $\mathcal{K} = \mathcal{S}_+^N$  is a single semidefinite cone, Algorithm 2.2 reduces to the famous *Boundary Point Method (BPM)*, which was originally proposed by Povh, Rendl, and Wiegale [30] (also see [19]) for solving big SDPs. This method was proven surprisingly efficient in some applications, as illustrated in [19, 30]. Algorithm 2.2 can be thought of as a natural generalization of BPM to general conic semidefinite optimization (2.1). It is a specification of the following general regularization algorithm:

**Algorithm 2.3.** Choose  $Y \in \mathcal{S}^N$ ,  $y \in \mathbb{R}^m$ , and  $\epsilon^{in}, \epsilon^{out}$ , set  $Z = 0$  and do the loop  
Repeat until  $\|Z + \mathcal{A}^*(y) - C\| \leq \epsilon^{out}$  (*outer loop*):  
Repeat until  $\|b - \mathcal{A}(X)\| \leq \epsilon^{in}$  (*inner loop*):  
Compute the projections  
 $X = \sigma(Y/\sigma + \mathcal{A}^*(y) - C)_{\mathcal{K}}$ ,  $Z = -(Y/\sigma + \mathcal{A}^*(y) - C)_{-\mathcal{K}}$ .  
Set  $g = b - \mathcal{A}(X)$ .  
Update  $y \leftarrow y + \tau Wg$  with appropriate  $\tau$  and  $W$ .  
end  
Set  $Y := X$  and update  $\sigma$ .  
end

Algorithm 2.3 is also a natural generalization of the regularization framework proposed by Malick, Povh, Rendl, and Wiegale [19, Algorithm 4.3] for solving large scale SDP when  $\mathcal{K} = \mathcal{S}_+^N$  is a single semidefinite cone. In that case, if  $W$  is chosen to be  $(\mathcal{A}\mathcal{A}^*)^{-1}$  and  $\tau = 1/\sigma$ , Algorithm 2.3 reduces to Algorithm 2.2, as shown in [19, Prop. 4.4]. Its convergence was discussed in [19, Theorem 4.4].

For solving large scale SOS relaxations, Algorithm 2.2 might converge fast at the beginning, but generally has slow convergence when we get close to optimal solutions. This is because it is basically a gradient type method. To get faster convergence, we need better solvers for the inner loop of Algorithm 2.3. Typically, Newton type methods have good properties like local superlinear or quadratic convergence. This leads to another kind of regularization methods called *Newton-CG Augmented Lagrangian*, which was proposed by Zhao, Sun and Toh [40]. It gives surprisingly good numerical performance in solving big standard SDP problems. In the rest of this section, we generalize this important method to solve general conic semidefinite optimization (2.1).

Denote by  $\varphi_\sigma(Y, y)$  the objective in (2.5). When  $\mathcal{K} = \mathcal{S}_+^N$ ,  $\varphi_\sigma(Y, y)$  is differentiable [19, Proposition 3.2] and

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

The above is also true when  $\mathcal{K}$  is a cross product of semidefinite cones. The inner loop of Algorithm 2.3 is essentially solving the maximization problem

$$\max_{y \in \mathbb{R}^m} \varphi_\sigma(Y_k, y). \quad (2.7)$$

Since  $\varphi_\sigma$  is concave in  $y$ , a point  $\hat{y}$  is a maximizer of (2.7) 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 by Zhao, Sun and Toh [40, Section 3.2]. So we need to compute the generalized Hessian of  $\varphi_\sigma$ . If

$$\mathcal{A}^*(y) = (\mathcal{A}_1^*(y), \dots, \mathcal{A}_\ell^*(y)), \quad C = (C_1, \dots, C_\ell),$$

then for  $Y = (Y_1, \dots, Y_\ell) \in \mathcal{M}$

$$\mathcal{A}^*(y) - C + Y/\sigma = \left(\mathcal{A}_1^*(y) - C_1 + Y_1/\sigma, \dots, \mathcal{A}_\ell^*(y) - C_\ell + Y_\ell/\sigma\right).$$

So we get  $\varphi_\sigma(Y, y) = b^T y + \frac{1}{2\sigma} \|Y\|^2 - \sum_{k=1}^{\ell} \varphi_\sigma^{(k)}(Y, y)$ , where

$$\varphi_\sigma^{(k)}(Y, y) = \frac{\sigma}{2} \|(\mathcal{A}_k^*(y) - C_k + Y_k/\sigma)_+\|^2. \quad (2.8)$$

This implies the formula

$$\nabla_y^2 \varphi_\sigma(Y, y) = - \sum_{k=1}^{\ell} \nabla_y^2 \varphi_\sigma^{(k)}(Y, y).$$

A numerical method for evaluating  $\nabla_y^2 \varphi_\sigma^{(k)}(Y, y)$  was given in [40, Section 3.2]. This can be done by computing eigenvalue decompositions. Let

$$\mathcal{A}_k^*(y) - C_k + Y_k/\sigma = Q_k \Lambda^{(k)} Q_k^T,$$

where  $Q_k \in \mathbb{R}^{N_k \times N_k}$  is orthogonal and  $\Lambda^{(k)} = \text{diag}(\lambda_1^{(k)}, \dots, \lambda_{N_k}^{(k)})$  is diagonal with  $\lambda_1^{(k)} \geq \lambda_2^{(k)} \geq \dots \geq \lambda_{N_k}^{(k)}$ . Denote index sets

$$\Gamma_+^{(k)} = \{1 \leq i \leq N_k : \lambda_i^{(k)} \geq 0\}, \quad \Gamma_-^{(k)} = \{1 \leq i \leq N_k : \lambda_i^{(k)} < 0\}.$$

Then define a symmetric matrix  $\Omega^{(k)} \in \mathbb{R}^{N_k \times N_k}$  as follows:

$$\Omega^{(k)}(i, j) = \begin{cases} 1 & \text{if } i, j \in \Gamma_+^{(k)}, \\ \frac{\lambda_i}{\lambda_i - \lambda_j} & \text{if } i \in \Gamma_+^{(k)}, j \in \Gamma_-^{(k)}, \\ \frac{\lambda_j}{\lambda_j - \lambda_i} & \text{if } i \in \Gamma_-^{(k)}, j \in \Gamma_+^{(k)}, \\ 0 & \text{otherwise.} \end{cases}$$

It was shown [40, Section 3.2] that

$$\nabla_y^2 \varphi_\sigma^{(k)}(Y, y) \cdot z = \sigma \mathcal{A}_k \left( Q_k (\Omega^{(k)} \circ (Q_k^T \mathcal{A}_k^*(z) Q_k)) Q_k^T \right),$$

where  $\circ$  denotes Hadamard product. Now we define a matrix  $\mathcal{L}_{Y,y}^\sigma \in \mathbb{R}^{m \times m}$  satisfying

$$\mathcal{L}_{Y,y}^\sigma \cdot z = \sigma \sum_{k=1}^{\ell} \mathcal{A}_k \left( Q_k (\Omega^{(k)} \circ (Q_k^T \mathcal{A}_k^*(z) Q_k)) Q_k^T \right).$$

It was shown [40, Prop. 3.2] that we always have  $\mathcal{L}_{Y,y}^\sigma \succeq 0$ . Furthermore,  $\mathcal{L}_{Y,y}^\sigma \succ 0$  if some nondegeneracy condition holds. In either case, an approximate semismooth Newton direction  $d_{new}$  for (2.7) can be determined by the linear system

$$\left( \mathcal{L}_{Y,y}^\sigma + \epsilon \cdot I_N \right) d_{new} = \nabla_y \varphi_\sigma. \quad (2.9)$$

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 (2.9) by direct methods like Cholesky factorization. To avoid this difficulty, Zhao, Sun and Toh [40] proposed applying conjugate gradient (CG) iterations to solve (2.9) approximately. To fasten the convergence of CG, a good preconditioner for (2.9) is  $(\mathcal{A}\mathcal{A}^*)^{-1}$ , as mentioned in [40]. In SOS relaxations for

unconstrained and homogeneous polynomial optimization, there is a nice property that the resulting  $(\mathcal{AA}^*)^{-1}$  has explicit neat formulae, as will be shown in Sections 3 and 4. So we use  $(\mathcal{AA}^*)^{-1}$  as a preconditioner in this situation. However, in Lasserre's relaxation for constrained polynomial optimization, typically there is no explicit expression for  $(\mathcal{AA}^*)^{-1}$ , and usually computing  $(\mathcal{AA}^*)^{-1}$  is quite difficult. In this case, we just use  $(\text{diag}(\mathcal{AA}^*))^{-1}$  as a preconditioner, as suggested in [40]. In either case, we set a cap  $K$  on the maximum number of CG iterations to insure solving (2.9) do not take too much time. A standard preconditioned CG algorithm (like Algorithm 6.12 in [4]) can be applied to solve (2.9). The numerical experiments in [40] show that this method is surprisingly powerful in solving some big standard SDPs when  $\mathcal{K} = \mathcal{S}_+^N$  is a single semidefinite cone.

Now we present the Newton-CG Augmented Lagrangian regularization method for solving general conic semidefinite optimization (2.1)-(2.2).

**Algorithm 2.4.** Choose  $\epsilon^{in}, \epsilon^{out} \in (0, 1)$ ,  $\epsilon > 0$ ,  $\delta \in (0, 1)$ ,  $\rho > 1$ ,  $\sigma_{\max}, K \in \mathbb{N}$ .

Initialization: Choose  $X_0, Z_0 \in \mathcal{S}^N$ ,  $y_0 \in \mathbb{R}^m$ ,  $\sigma_0$  and set  $k = 0$ .

While  $(\|Z_k + \mathcal{A}^*(y)_k - C\| \leq \epsilon^{out})$  (*outer loop*):

Set  $Y_k := X_k$ .

Set  $j = 0$  and  $y_{k,j} = y_k$ .

While  $(\|\nabla_y \varphi_{\sigma_k}(Y_k, y_{k,j})\| \leq \epsilon^{in})$  (*inner loop*):

Set  $g_j = \nabla_y \varphi_{\sigma_k}(Y_k, y_{k,j})$ .

Compute  $d_{new}$  from (2.9) by applying preconditioned CG at most  $K$  steps.

Find the smallest integer  $\alpha > 0$  such that

$$\varphi_{\sigma_k}(Y_k, y_{k,j} + \delta^\alpha \cdot d_{new}) \geq \varphi_{\sigma_k}(Y_k, y_{k,j}) + \delta^\alpha \cdot g_j^T d_{new}. \quad (2.10)$$

Set  $y_{k,j+1} := y_{k,j} + \delta^\alpha \cdot d_{new}$ .

Compute the projections into the cone  $\mathcal{K}$ :

$$X_k = \sigma(Y_k/\sigma + \mathcal{A}^*(y_{k,j+1}) - C)_{\mathcal{K}}, Z_k = -(Y_k/\sigma + \mathcal{A}^*(y_{k,j+1}) - C)_{-\mathcal{K}}.$$

Set  $j := j + 1$ .

end (*inner loop*)

Set  $y_{k+1} = y_{k,j}$

If  $\sigma_k \leq \sigma_{\max}$ , set  $\sigma_{k+1} = \rho\sigma_k$ .

Set  $k := k + 1$ .

end (*outer loop*)

When  $\mathcal{K} = \mathcal{S}_+^N$  is a single semidefinite cone, the convergence of Algorithm 2.4 has been discussed in [40, Theorems 3.5, 4.1,4.2]. These results could be generalized naturally to the cone  $\mathcal{K}$  of block diagonal positive semidefinite matrices. The specifics about the convergence are beyond the scope of this paper. We refer to [19, 31, 32, 40].

Algorithm 2.2 and 2.4 are basically specifications of Algorithm 2.3. Generally Algorithm 2.2 has slower convergence than Algorithm 2.4 does, because it is a gradient type method while the latter is a Newton type method. They have similar performance when computing eigenvalue decompositions is not expensive. However, when eigenvalue decompositions are expensive to compute, Algorithm 2.2 usually takes more time because it requires computing more eigenvalue decompositions. In large scale polynomial optimization, the matrices often have big dimension, and hence Algorithm 2.4 is more suitable. So in later sections, the numerical experiments are mainly based on using Algorithm 2.4.

### 3 Unconstrained polynomial optimization

Consider unconstrained optimization problem

$$f_{min}^{uc} := \min_{x \in \mathbb{R}^n} f(x) \quad (3.1)$$

where  $f(x)$  is a multivariate polynomial of degree  $2d$ . Here  $f_{min}^{uc}$  denotes the global minimum of  $f(x)$  over  $\mathbb{R}^n$ . Problem (3.1) is NP-hard when  $\deg(f) \geq 4$ , as shown in [21]. There has much recent work on solving (3.1) via SOS relaxations, like [9, 13, 15, 23, 27, 28]. The standard SOS relaxation for (3.1) is

$$f_{sos}^{uc} := \max \quad \gamma \\ \text{s.t.} \quad f(x) - \gamma \text{ is SOS.} \quad (3.2)$$

Obviously the above optimal value  $f_{sos}^{uc}$  satisfies the relation  $f_{sos}^{uc} \leq f_{min}^{uc}$ . Though it is possible that  $f_{sos}^{uc} < f_{min}^{uc}$ , it was observed in [28, 27] that (3.1) works very well in practice. Typically (3.2) gives a reasonably well approximation for (3.1), which is good enough in many applications. SOS relaxations are usually very expensive to solve when interior-point methods are used. In this section, we show how to apply regularization methods described in Section 2 to solve (3.2). We begin with the SDP formulation of (3.2).

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

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

Here  $\binom{n+d}{d}$  is the total number of monomials in  $x$  of degrees up to  $d$ . Define 0/1 constant symmetric matrices  $C$  and  $A_\alpha$  in the way that

$$[x]_d [x]_d^T = C + \sum_{\alpha \in \mathbb{N}^n: 0 < |\alpha| \leq 2d} A_\alpha x^\alpha. \quad (3.3)$$

Denote  $\mathbb{U}_{2d}^n = \{\alpha \in \mathbb{N}^n : 0 < |\alpha| \leq 2d\}$ . If  $f(x) = f_0 + \sum_{\alpha \in \mathbb{U}_{2d}^n} f_\alpha x^\alpha$ , then  $\gamma$  is feasible for (3.2) if and only if  $X$  satisfies

$$\begin{aligned} C \bullet X + \gamma &= f_0, \\ A_\alpha \bullet X &= f_\alpha \quad \forall \alpha \in \mathbb{U}_{2d}^n, \\ X &\succeq 0. \end{aligned}$$

Let  $b = (f_\alpha)_{\alpha \in \mathbb{U}_{2d}^n}$  be a vector of dimension  $m = \binom{n+2d}{2d} - 1$ . Define  $\mathcal{A}$  and  $\mathcal{K}$  as

$$\mathcal{A}(X) = (A_\alpha \bullet X)_{\alpha \in \mathbb{U}_{2d}^n}, \quad \mathcal{K} = \mathcal{S}_+^{\binom{n+d}{d}}.$$

Then SOS relaxation (3.2) is equivalent to

$$f_{sdp}^{uc} := \min \quad C \bullet X \\ \text{s.t.} \quad \mathcal{A}(X) = b, X \in \mathcal{K}. \quad (3.4)$$

The dual of the above is

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & \mathcal{A}^*(y) + Z = C, Z \in \mathcal{K}. \end{aligned} \quad (3.5)$$

Here the adjoint  $\mathcal{A}^*(y) = \sum_{\alpha \in \mathbb{U}_{2d}^n} y_\alpha A_\alpha$ .

**Theorem 3.1.** For notations defined above, it holds  $f_{sos}^{uc} = -f_{sdp}^{uc} + f_0 \leq f_{min}^{uc}$ . If SOS relaxation (3.2) is exact, then  $f_{min}^{uc} = -f_{sdp}^{uc} + f_0$ .

*Proof.* As we have seen in the above, optimization problems (3.2) and (3.4) are equivalent. Their objectives satisfy the relation

$$C \bullet X = -\gamma + f_0.$$

Maximizing  $\gamma$  in (3.2) is equivalent to minimizing  $C \bullet X$  in (3.4). The feasible sets of (3.2) and (3.4) are the same. Hence we have  $f_{sos}^{uc} = -f_{sdp}^{uc} + f_0$ . Furthermore,  $f_{sos}^{uc} \leq f_{min}^{uc}$  is obvious because (3.2) is a relaxation of (3.1).

If SOS relaxation (3.2) is exact, i.e.,  $f_{sos}^{uc} = f_{min}^{uc}$ , then obviously  $f_{min}^{uc} = -f_{sdp}^{uc} + f_0$ .  $\square$

Now we turn to finding  $(\mathcal{AA}^*)^{-1}$ . Since  $A_\alpha$  defined in (3.3) are orthogonal to each other, the matrix  $\mathcal{AA}^*$  is diagonal. So we get the formula

$$(\mathcal{AA}^*)^{-1} = \text{diag}(\mathbf{a}), \quad \text{where} \quad \mathbf{a} = ((\mathbf{1}^T A_\alpha \mathbf{1})^{-1} : \alpha \in \mathbb{U}_{2d}^n). \quad (3.6)$$

Here  $\mathbf{1}$  is the vector of all ones.

Algorithm 2.4 can be applied to solve the SDP (3.4)-(3.5). We use  $(\mathcal{AA}^*)^{-1}$  via the formula (3.6) as a preconditioner. Suppose  $(X^*, y^*, Z^*)$  is optimal for (3.4)-(3.5). Then  $-f_{sdp}^{uc} + f_0$  is a lower bound for the minimum  $f_{min}^{uc}$ . Furthermore, if  $Z^*$  has rank one, then  $f_{min}^{uc} = f_{sos}^{uc}$  and a global minimizer for (3.1) can be obtained easily. This can be illustrated as follows. When  $\text{rank}(Z^*) = 1$ , the constraint in (3.5) implies  $Z^* = [x^*]_d [x^*]_d^T$  for some  $x^* \in \mathbb{R}^n$ , and hence  $y^* = -[x^*]_{2d}$ . Then, for any  $x \in \mathbb{R}^n$ ,

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

The inequality above follows the optimality of  $y^*$  and the fact that  $Z = [x]_d [x]_d^T$  is always feasible for (3.5). So  $x^*$  is a global minimizer.

When  $\text{rank}(Z^*) > 1$ , several global minimizers for (3.1) could be obtained if a so called *flat extension condition (FEC)* holds. We refer to Curto and Fialkow [3] for FEC, and Henrion and Lasserre [8] for how to extract global minimizers from  $Z^*$ . Typically, FEC fails if there are a lot of global minimizers or the SOS relaxation is not exact.

### 3.1 Numerical experiments

Now we show some numerical examples of applying Algorithm 2.4 to solve the SDP (3.4)-(3.5) which is equivalent to SOS relaxation (3.2). The computation is implemented on a Linux Desktop of 3.8GB memory and 2.8GHz CPU frequency. Algorithm 2.4 is implemented in Matlab, and its parameters are:  $\epsilon^{in} = \epsilon^{out} = 10^{-6}$ ,  $\rho = 5$ ,  $\sigma_0 = 1$ ,  $\sigma_{max} = 10^6$ ,  $\delta = \frac{1}{2}$ ,  $K = 500$ . The inverse  $(\mathcal{AA}^*)^{-1}$  via the formula (3.6) is used as a preconditioner. Its inner loop is terminated if it does not converge within 25 steps, and the outer loop is terminated if it does not converge within 20 steps. If the optimal  $Z^*$  satisfies FEC, we extract one or several global minimizers  $x^*$  by using the method in [8]; otherwise, we just set  $x^* = Z^*(2 : n + 1, 1)$ . In either case, the relative error of  $x^*$  is measured as

$$err = \frac{|f(x^*) - f_{sos}^{uc}|}{\max\{1, |f(x^*)|\}}.$$

Since  $f_{sos}^{uc} \leq f_{min}^{uc} \leq f(x^*)$ ,  $err = 0$  if and only if  $x^*$  is a global minimizer. So the smaller  $err$  is, the better the obtained solution is.

**Example 3.2.** Minimize the quartic polynomial  $(x^2)^T Ax^2 - x^T Bx$ . Here  $x^2$  denotes the component-wise square of  $x$ , and  $A, B$  are symmetric matrices given as

$$A = \begin{bmatrix} n & n-1 & n-2 & \cdots & 2 & 1 \\ n-1 & n & 0 & \ddots & 0 & 2 \\ n-2 & 0 & n & \ddots & 0 & 3 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 2 & 0 & \cdots & 0 & n & n-1 \\ 1 & 2 & \cdots & n-2 & n-1 & n \end{bmatrix}, B = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 0 & \ddots & 0 & 1 \\ 1 & 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 & 1 \end{bmatrix}.$$

When  $n = 50$ , the size of the resulting SDP (3.4)-(3.5) is  $(N, m) = (1326, 316250)$ . Solving (3.4)-(3.5) takes about 10 hours. The computed lower bound  $f_{sos}^{uc} = -1.2919$  (only four decimal digits are shown). The optimal  $Z^*$  has rank two and satisfies FEC. So we get two solutions from  $Z^*$  (only four decimal digits are shown):

$$\pm(0.0000, 0.1504, 0.1480, 0.1456, 0.1432, \dots, 0.0584, 0.0572, 0.0560, 0.0548, 0.2188).$$

The relative error is about  $10^{-8}$ . So they are global minimizers up to a tiny numerical error.

**Example 3.3.** Solve the quartic polynomial optimization

$$\min_{x \in \mathbb{R}^n} (x_1^2 + \cdots + x_n^2)^2 + \sum_{1 \leq i < j, j+1 < k \leq n} \frac{1}{n} x_i x_j x_k.$$

When  $n = 50$ , the resulting SDP (3.4)-(3.5) has size  $(N, m) = (1326, 316250)$ . Solving it takes about 1.5 hours. We get  $f_{sos}^{uc} = -0.1246$  (only four decimal digits are shown) and  $\text{rank}(Z^*) = 1$ . The solution extracted (only four decimal digits are shown) is

$$-(0.1127, 0.1127, 0.1126, 0.1125, 0.1124, \dots, 0.1086, 0.1085, 0.1085, 0.1082, 0.1128).$$

Its relative error is about  $2 \cdot 10^{-7}$ . So it is a global minimizer up to a small numerical error.

**Example 3.4.** Minimize the following least squares polynomial

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

where  $x_0 = x_{n+1} = 0$ . For  $n = 16$ , the resulting SDP (3.4)-(3.5) has size  $(N, m) = (969, 74612)$ . Solving (3.4)-(3.5) takes about 12 hours. The computed lower bound  $f_{sos}^{uc} = 7.5586$  (only four decimal digits are shown). The optimal  $Z^*$  has rank two and FEC holds. Two extracted solutions (only four decimal digits are shown) are

$$(0.0039, 0.6285, 0.5370, 0.0259, -0.4324, -0.4266, 0.1540, -0.5108, 0.2172, -0.4029, \\ 0.4400, -0.4307, 0.0230, 0.5378, 0.6285, 0.0039), \\ (0.0039, 0.6285, 0.5378, 0.0230, -0.4307, 0.4400, -0.4029, 0.2172 - 0.5108, 0.1540 \\ -0.4266, -0.4324, 0.0259, 0.5370, 0.6285, 0.0039).$$

The relative error is around  $10^{-8}$ . So they are global minimizers up to a tiny numerical error.

**Example 3.5.** Solve the least square problem

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^n \left( 1 - \sum_{j=1}^i (x_j + x_j^2) \right)^2 + \left( 1 - \sum_{j=i}^n (x_j + x_j^3) \right)^2.$$

For  $n = 15$ , the resulting SDP (3.4)-(3.5) has size  $(N, m) = (816, 54263)$ . Solving (3.4)-(3.5) takes about 4 hours. The computed lower bound  $f_{sos}^{uc} = 2.1011$  (only four decimal digits are shown). Since  $\text{rank}(Z^*) = 1$ , we get a solution (only four decimal digits are shown)

$$(0.5233, 0.2429, 0.0802, 0.0011, -0.0003, 0.0001, -0.0000, 0.0000, -0.0000, 0.0001, -0.0005, 0.0020, -0.0088, 0.0361, 0.6011).$$

Its relative error is around  $10^{-9}$ . So it is a global minimizer up to a tiny numerical error.

**Example 3.6** (Random polynomials). We test the performance of Algorithm 2.4 in solving SOS relaxations for minimizing randomly generated polynomials. To insure the polynomials have finite global minimum, we generate  $f(x)$  randomly as follows

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

where  $f \in \mathbb{R}^{\binom{n+2d-1}{2d-1}}$  is a random vector of Gaussian distribution and  $F \in \mathbb{R}^{\binom{n+d-1}{d} \times \binom{n+d-1}{d}}$  is a random matrix of Gaussian distribution. Here  $[x^d]$  denotes the vector of monomials of degree equaling  $d$ . The generated  $f(x)$  almost always has a finite global minimum. The computational results are listed in Tables 2-5. The first column is the number of variables; the second column is the size of the resulting SDP (3.4)-(3.5); the third column is the number of instances generated; the fourth column is the maximum, median, and minimum time consumed by the computer; and the last column is the maximum, median, and minimum relative errors of obtained solutions.

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
20	(231, 10625)	20	0:00:45	0:01:07	0:01:59	(5e-9, 7e-8, 2e-8)
30	(496, 46375)	20	0:10:45	0:11:44	0:12:43	(6e-9, 5e-8, 1e-8)
40	(861, 135750)	10	0:28:42	1:02:36	1:22:24	(3e-9, 3e-8, 9e-8)
50	(1326, 316250)	5	2:04:43	3:33:31	4:15:38	(6e-8, 4e-7, 9e-7)
60	(1891, 635375)	5	5:17:42	8:03:04	15:23:23	(7e-8, 1e-7, 5e-7)
70	(2556, 1150625)	5	10:25:07	13:06:38	20:44:31	(2e-9, 7e-8, 3e-8)
80	(3321, 1929500)	3	30:09:34	32:17:43	33:30:12	(6e-8, 8e-8, 1e-7)
90	(4186, 3049500)	1	63:55:41	63:55:41	63:55:41	(4e-7, 4e-7, 4e-7)
100	(5151, 4598125)	1	199:49:33	199:49:33	199:49:33	(6e-7, 6e-7, 6e-7)

Table 2: Computational results for random (3.1) of degree 4

When  $f(x)$  has degree 4 ( $d = 2$ ), the computational results are in Table 2. As we can see, SOS relaxation (3.2) is solved quite well. For  $n = 20 \sim 30$ , solving (3.4)-(3.5) takes a couple of minutes; for  $n = 40 \sim 50$ , it takes a couple of hours; for  $n = 60 \sim 80$ , it takes about five to thirty hours; for  $n = 90$ , it takes about sixty hours; for  $n = 100$ , it takes about one week.

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
10	(286, 8007)	20	0:01:45	0:00:54	0:03:16	(9e-10, 4e-8, 6e-7)
15	(816, 54263)	10	0:39:05	1:07:42	1:33:51	(3e-8, 6e-7, 2e-6)
20	(1771, 230229)	3	11:47:23	21:35:17	25:36:08	(9e-8, 7e-8, 1e-6)
25	(3276, 736280)	1	104:41:36	104:41:36	104:41:36	(6e-7, 6e-7, 6e-7)

Table 3: Computational results for random (3.1) of degree 6

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
8	(495, 12869)	20	0:03:44	0:07:37	0:19:42	(1e-8, 3e-7, 8e-6)
10	(1001, 43757)	10	1:14:48	1:40:58	2:20:22	(9e-9, 3e-8, 4e-6)
12	(1820, 125969)	3	13:04:06	13:11:35	13:34:40	(8e-8, 1e-7, 6e-7)
15	(3876, 490313)	1	217:46:12	217:46:12	217:46:12	(9e-8, 9e-8, 9e-8)

Table 4: Computational results for random (3.1) of degree 8

When  $f(x)$  has degree 6 ( $d = 3$ ), the computational results are in Table 3. For  $n = 15$ , solving (3.4)-(3.5) takes about forty to ninety minutes; for  $n = 20$ , it takes about ten to twenty five hours; for  $n = 25$ , it takes about one hundred hours.

When  $f(x)$  has degree 8 ( $d = 4$ ), the computational results are in Table 4. For  $n = 10$ , solving (3.4)-(3.5) takes about one or two hours; for  $n = 12$ , it takes about thirteen hours; for  $n = 25$ , it takes about two hundred hours.

When  $f(x)$  has degree 10 ( $d = 5$ ), the computational results are in Table 5. For  $n = 6$ , solving (3.4)-(3.5) takes a couple of minutes; for  $n = 8$ , it takes about a couple of hours; for  $n = 9$ , it takes about ten to twenty hours; for  $n = 10$ , it takes about seventy hours.

For polynomials generated in this example, SOS relaxation (3.2) is very successful in solving (3.1) globally. For all the generated instances, the optimal  $Z^*$ s have rank one, and highly accurate global optimal solutions with tiny errors are found. This is consistent with the computational results shown in [28]. The computational results demonstrate that Algorithm 2.4 is very efficient in solving SOS relaxations for large scale polynomial optimization. Significantly bigger problems would be solved very well. A quartic polynomial optimization with 100 variables would be solved on a regular computer. This is almost impossible by applying prior existing SOS solvers using interior point type methods.

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
6	(462, 8007)	20	0:03:18	0:05:19	0:06:22	(2e-8, 1e-7, 2e-7)
8	(1287, 43757)	10	1:51:38	3:47:52	6:53:22	(5e-8, 3e-7, 4e-6)
9	(2002, 92377)	3	9:18:17	12:09:07	20:52:36	(2e-8, 2e-7, 3e-6)
10	(3003, 184755)	1	72:40:36	72:40:36	72:40:36	(9e-6, 9e-6, 9e-6)

Table 5: Computational results for random (3.1) of degree 10

## 4 Homogeneous polynomial optimization

A frequently encountered polynomial optimization is

$$f_{min}^{hmg} := \min_{x \in \mathbb{R}^n} f(x) \quad (4.1)$$

$$s.t. \quad \|x\|_2 = 1$$

where  $f(x)$  is a form (homogeneous polynomial). Assume its degree  $\deg(f) = 2d$  is even, i.e.,  $f(x)$  is an even form. The case that  $\deg(f)$  is odd can be transformed to minimizing an even form over the unit sphere  $\mathbb{S}^{n-1} = \{x \in \mathbb{R}^n : \|x\|_2 = 1\}$ , as will be shown later. In (4.1),  $f_{min}^{hmg}$  denotes the minimum objective value. Problem (4.1) is NP-hard when  $d > 1$ .

The standard SOS relaxation for (4.1) is

$$f_{sos}^{hmg} := \max \gamma \quad (4.2)$$

$$s.t. \quad f(x) - \gamma \cdot (x^T x)^d \quad \text{is SOS.}$$

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

$$[x^d]^T = [x_1^d \quad x_1^{d-1}x_2 \quad x_1^{d-2}x_2^2 \quad \cdots \quad x_1 x_2^{d-1} \quad x_2^d].$$

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  $D = \text{diag}(D_\alpha)$  be a diagonal matrix. It holds the relation

$$(x^T x)^d = \sum_{\alpha \in \mathbb{N}_d} D_\alpha x_1^{2\alpha_1} \cdots x_n^{2\alpha_n} = [x^d]^T D [x^d] = ([x^d][x^d]^T) \bullet D. \quad (4.3)$$

Define 0/1 matrices  $H_\alpha$  such that

$$[x^d][x^d]^T = \sum_{\alpha \in \mathbb{N}_{2d}} H_\alpha x_1^{\alpha_1} \cdots x_n^{\alpha_n}. \quad (4.4)$$

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

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

Letting  $\alpha = 2de_1$  in the above, we get

$$\gamma = f_{2de_1} - H_{2de_1} \bullet X.$$

Denote  $\mathbb{H}_{2d}^n = \mathbb{N}_{2d} \setminus \{2de_1\}$ , and set  $r = (r_\alpha)_{\alpha \in \mathbb{H}_{2d}^n}$  as

$$r_\alpha = \begin{cases} D_{\alpha/2} & \text{if } \alpha \in 2\mathbb{N}_d \setminus \{2de_1\}, \\ 0 & \text{if } \alpha \in \mathbb{N}_{2d} \setminus \{2\mathbb{N}_d\}. \end{cases} \quad (4.5)$$

Define matrices  $C, A_\alpha$  and scalars  $b_\alpha$  as

$$C = H_{2de_1}, \quad A_\alpha = H_\alpha - r_\alpha H_{2de_1}, \quad b_\alpha = f_\alpha - r_\alpha f_{2de_1} \quad \forall \alpha \in \mathbb{H}_{2d}^n. \quad (4.6)$$

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

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

Then SOS relaxation (4.2) is equivalent to

$$\begin{aligned} f_{sdp}^{hmg} &:= \min C \bullet X \\ \text{s.t. } &\mathcal{A}(X) = b, X \in \mathcal{K}. \end{aligned} \quad (4.7)$$

The dual of (4.7) is

$$\begin{aligned} \max &b^T y \\ \text{s.t. } &\mathcal{A}^*(y) + Z = C, Z \in \mathcal{K}. \end{aligned} \quad (4.8)$$

Here  $\mathcal{A}^*(y) = \sum_{\alpha \in \mathbb{H}_{2d}^n} y_\alpha A_\alpha$  is the adjoint of  $\mathcal{A}$ .

**Theorem 4.1.** *For notations defined above, it holds  $f_{sos}^{hmg} = -f_{sdp}^{hmg} + f_{2de_1} \leq f_{min}^{hmg}$ . If SOS relaxation (3.2) is exact, then  $f_{min}^{hmg} = -f_{sdp}^{hmg} + f_{2de_1}$ .*

Its proof is the same as for Theorem 3.1, and hence is omitted. A general error bound for (4.2) approximating (4.1) is given in [24].

Now we turn to finding  $(\mathcal{A}\mathcal{A}^*)^{-1}$ . Note the matrices  $H_\alpha$  defined in (4.4) are orthogonal to each other. Thus  $\mathcal{H}\mathcal{H}^*$  is diagonal, and

$$(\mathcal{H}\mathcal{H}^*)^{-1} = \text{diag}((\mathbf{1}^T H_\alpha \mathbf{1})^{-1} : \alpha \in \mathbb{H}_{2d}^n).$$

Here  $\mathbf{1}$  is the vector of all ones. Thus, we have for all  $\alpha, \beta \in \mathbb{H}_{2d}^n$

$$\begin{aligned} A_\alpha \bullet A_\beta &= (H_\alpha - r_\alpha H_{2de_1}) \bullet (H_\beta - r_\beta H_{2de_1}) \\ &= H_\alpha \bullet H_\beta - H_{2de_1} \bullet (r_\alpha H_\beta + r_\beta H_\alpha) + r_\alpha r_\beta H_{2de_1} \bullet H_{2de_1} \\ &= H_\alpha \bullet H_\beta + r_\alpha r_\beta, \end{aligned}$$

that is,  $\mathcal{A}\mathcal{A}^* = \mathcal{H}\mathcal{H}^* + rr^T$ . By Sherman-Morrison formula, we have

$$(\mathcal{A}\mathcal{A}^*)^{-1} = (\mathcal{H}\mathcal{H}^*)^{-1} - \frac{(\mathcal{H}\mathcal{H}^*)^{-1} r r^T (\mathcal{H}\mathcal{H}^*)^{-1}}{1 + r^T (\mathcal{H}\mathcal{H}^*)^{-1} r}.$$

Setting vectors

$$\mathbf{h} = ((\mathbf{1}^T H_\alpha \mathbf{1})^{-1} : \alpha \in \mathbb{H}_{2d}^n), \quad \mathbf{p} = (1 + r^T \text{diag}(\mathbf{h})r)^{-1/2} \text{diag}(\mathbf{h})r,$$

we get a neat formula

$$(\mathcal{A}\mathcal{A}^*)^{-1} = \text{diag}(\mathbf{h}) - \mathbf{p}\mathbf{p}^T. \quad (4.9)$$

Algorithm 2.4 can be applied to solve (4.7)-(4.8). We use  $(\mathcal{A}\mathcal{A}^*)^{-1}$  via the formula (4.9) as a preconditioner. Let  $(X^*, y^*, Z^*)$  be optimal for (4.7)-(4.8). Then  $-f_{sdp}^{hmg} + f_{2de_1}$  is a lower bound for the minimum  $f_{min}^{hmg}$ . We would also get global minimizers from  $Z^*$  when FEC holds. Note that (4.3) and (4.6) imply

$$A_\alpha \bullet D = 0 \quad \forall \alpha \in \mathbb{H}_{2d}^n \quad \text{and} \quad Z^* \bullet D = H_{2de_1} \bullet D = 1. \quad (4.10)$$

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 (4.10). 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 (see [3]). If  $\widehat{Z}^*$  satisfies FEC, using the method in [8], we can get  $v^{(1)}, \dots, v^{(r)} \in \mathbb{R}^{n-1}$  such that

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

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

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

for some scalars  $\nu_i > 0$ . Then the relations (4.3) and (4.10) imply  $\nu_1 + \dots + \nu_r = 1$ . Since every  $Z^{(i)} = [(x^{(i)})^d] [(x^{(i)})^d]^T$  is feasible for (4.8), the optimality of  $Z^*$  implies every  $x^{(i)}$  is a global minimizer for (4.1).

In particular, if  $\text{rank}(Z^*) = 1$ , then  $\widehat{Z}^*$  satisfies FEC automatically. When FEC fails, it is not clear how to extract a minimizer from  $Z^*$ . This happens usually because there are a lot of global minimizers or SOS relaxation (4.2) is not exact ( $f_{\text{sos}}^{\text{hmg}} < f_{\text{min}}^{\text{hmg}}$ ).

#### 4.1 Numerical experiments

Now we present some numerical examples of applying Algorithm 2.4 to solve the SDP (4.7)-(4.8) which is equivalent to SOS relaxation (4.2). As in Section 3.1, the computation is implemented in Matlab on the same machine. The parameters and stopping criteria in Algorithm 2.4 are also the same as there. The inverse  $(\mathcal{A}\mathcal{A}^*)^{-1}$  via the formula (4.9) is used as a preconditioner. If the optimal  $Z^*$  satisfies FEC, we apply the preceding procedure to extract one or several global minimizers  $x^*$ ; otherwise, we just choose  $x^*$  to be the normalization of  $Z^*(1 : n, 1)$ . The relative error of  $x^*$  is measured as

$$\text{err} = \frac{|f(x^*) - f_{\text{sos}}^{\text{hmg}}|}{\max\{1, |f(x^*)|\}}.$$

Since  $f_{\text{sos}}^{\text{hmg}} \leq f_{\text{min}}^{\text{hmg}} \leq f(x^*)$ ,  $\text{err} = 0$  if and only if  $x^*$  is a global minimizer of (4.1).

**Example 4.2.** Minimize the following square free quartic form over  $\mathbb{S}^{n-1}$

$$\sum_{1 \leq i < j < k < \ell \leq n} (-i - j + k + \ell) x_i x_j x_k x_\ell.$$

For  $n = 50$ , the resulting SDP (4.7)-(4.8) has size  $(N, m) = (1275, 292824)$ . Solving (4.7)-(4.8) takes about 3.5 hours. The computed lower bound  $f_{\text{sos}}^{\text{hmg}} = -140.4051$  (only four decimal digits are shown). The optimal  $Z^*$  has rank two and FEC holds. There are 2 solutions (only four decimal digits are shown) extracted from  $Z^*$ :

$$(0.6393, 0.1217, 0.0827, 0.0575, 0.0401, \dots, -0.1653, -0.1619, -0.1581, -0.1541, -0.1499), \\ (0.1499, 0.1541, 0.1581, 0.1619, 0.1653, \dots, -0.0401, -0.0575, -0.0827, -0.1217, -0.6393).$$

The error is around  $3 \cdot 10^{-9}$ . So they are global minimizers up to a tiny numerical error.

**Example 4.3.** Minimize the following tridiagonal sextic form over  $\mathbb{S}^{n-1}$

$$\sum_{1 \leq i \leq n} x_i^6 + \sum_{1 \leq i \leq n-1} 2x_i^3 x_{i+1}^3 = \begin{bmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ \vdots \\ x_{n-1}^3 \\ x_n^3 \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \ddots & 0 \\ 0 & 1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ \vdots \\ x_{n-1}^3 \\ x_n^3 \end{bmatrix}.$$

For  $n = 20$ , the size of the resulting SDP (4.7)-(4.8) is  $(N, m) = (1540, 177099)$ . Solving (4.7)-(4.8) takes about 10 hours, and the computed lower bound  $f_{sos}^{hmg} = 3.446 \cdot 10^{-5}$  (only four leading digits are shown). So this form is positive definite, which is contrast to the fact that the associated tridiagonal matrix is not positive semidefinite. The optimal  $Z^*$  has rank one, and the extracted solution (only four decimal digits are shown) is

$$(0.1474, -0.1811, 0.2024, -0.2176, 0.2290, -0.2375, 0.2439, -0.2484, 0.2514, -0.2528, \\ 0.2528, -0.2514, 0.2484, -0.2439, 0.2375, -0.2290, 0.2176, -0.2024, 0.1811, -0.1474).$$

Its relative error is around  $8 \cdot 10^{-11}$ . So it is a global minimizer up to a tiny numerical error.

**Example 4.4.** Minimize the following sextic form over  $\mathbb{S}^{n-1}$

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

For  $n = 20$ , the resulting SDP (4.7)-(4.8) has size  $(N, m) = (1540, 177099)$ . Solving (4.7)-(4.8) takes about 11 hours. The computed lower bound  $f_{sos}^{hmg} = -0.3827$  (only four decimal digits are shown). The optimal  $Z^*$  has rank one, and the extracted solution (only four decimal digits are shown) is

$$-(-0.6551, -0.6286, 0.1109, 0.1089, 0.1071, 0.1054, 0.1038, 0.1023, 0.1009, 0.0996, \\ 0.0983, 0.0971, 0.0960, 0.0949, 0.0939, 0.0929, 0.0920, 0.0911, 0.0902, 0.0894).$$

Its relative error is around  $6 \cdot 10^{-11}$ . So it is a global minimizer up to a tiny numerical error.

An important application of homogenous polynomial optimization (4.1) is estimating stability numbers of graphs, as shown below.

**Example 4.5** (Stability number 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 [20] (also see De Klerk and Pasechnik [6]) that

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

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

$$\alpha(G)^{-1} = \min_{\|x\|_2=1} \sum_{i=1}^n x_i^4 + 2 \sum_{(i,j) \in E} x_i^2 x_j^2. \quad (4.11)$$

$n$	(N,m)	# inst	time (min, med, max)		
20	(210, 8854)	20	0:00:22	0:03:04	0:06:25
30	(465, 40919)	10	0:04:12	0:39:58	1:33:47
40	(820, 123409)	10	0:20:52	1:21:01	5:24:12
50	(1275, 292824)	3	23:50:02	26:27:12	27:15:46
60	(1830, 595664)	1	75:46:58	75:46:58	75:46:58

Table 6: Computational results for stability number of random graphs

This is a quartic homogeneous polynomial optimization. When a lower bound  $f_{sos}^{hmg}$  for (4.11) via its SOS relaxation (4.2) is computed, we round  $\left(f_{sos}^{hmg}\right)^{-1}$  to the nearest integer which will be used as an estimate for  $\alpha(G)$ .

We generate random graphs  $G$ , and solve the SDP (4.7)-(4.8) which is equivalent to its SOS relaxation by applying Algorithm 2.4. The generation of random graphs is in the similar way as in Bomze and De Klerk [2, 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 6. The first column is the number of vertices, the second column is the size of the resulting SDP (4.7)-(4.8), the third column is the number of generated instances, and the last column is the minimum, median and maximum time consumed by the computer.

As we can see from Table 6, for  $n = 20$ , solving (4.7)-(4.8) takes a couple of minutes; for  $n = 30$ , it takes about four to ninety minutes; for  $n = 40$ , it takes about twenty minutes to five hours; for  $n = 50$ , it takes about twenty five hours; for  $n = 60$ , it takes about seventy five hours. In all the instances, we get correct stability numbers via solving the SOS relaxation of (4.11).

**Example 4.6** (Random forms). We test the performance of Algorithm 2.4 in solving SOS relaxation (4.2) for random forms. Generate a form  $f(x)$  randomly as

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

where the coefficients  $f_{\alpha}$  are Gaussian random variables. We apply Algorithm 2.4 to solve the SDP (4.7)-(4.8) which is equivalent to the corresponding SOS relaxation (4.2). The random forms of degrees 4, 6, 8, 10 are tested. The computational results are in Tables 7-10. The meaning of their columns are the same as in Tables 2-5.

When  $f(x)$  has degree 4 ( $d = 2$ ), the computational results are in Table 7. For  $n = 30$ , solving (4.7)-(4.8) takes a couple of minutes; for  $n = 40$ , it takes about thirty to forty minutes; for  $n = 50$ , it takes a few hours; for  $n = 60$ , it takes about six to nineteen hours; for  $n = 70$ , it takes about fifteen to thirty four hours; for  $n = 80$ , it takes about one hundred hours. In all the instances, the optimal  $Z^*$  has rank one and FEC holds. The obtained solutions have tiny relative errors.

When  $f(x)$  has degree 6 ( $d = 3$ ), the computational results are in Table 8. For  $n = 15$ , solving (4.7)-(4.8) takes around seventeen to twenty six minutes; for  $n = 20$ , it takes a couple of hours; for  $n = 25$ , it takes about fifty hours. The optimal  $Z^*$  now does not always satisfy FEC. When  $n = 10$  or 15, there are some instances that their errors are relatively bigger than

others. When  $n = 20$ , there are a few instances that their relative errors are significantly much bigger. This is probably due to the inexactness of their SOS relaxations. For  $n = 25$ , we have generated only one instance since its computation takes more than two days, and luckily a solution of high accuracy was found. But we should not expect this generally.

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
20	(210, 8854)	20	0:00:26	0:00:31	0:00:39	(2e-9, 9e-9, 3e-8)
30	(465, 40919)	20	0:06:04	0:06:38	0:08:41	(4e-9, 8e-9, 2e-8)
40	(820, 123409)	10	0:35:54	0:36:03	0:41:01	(4e-10, 7e-9, 3e-8)
50	(1275, 292824)	10	2:29:23	2:35:09	3:58:45	(5e-8, 7e-7, 3e-7)
60	(1830, 595664)	3	6:31:22	9:02:36	19:07:32	(9e-9, 5e-8, 1e-8)
70	(2485, 1088429)	3	15:48:39	24:57:13	34:03:29	(1e-10, 6e-9, 1e-9)
80	(3240, 1837619)	1	99:26:20	99:26:20	99:26:20	(7e-8, 2e-8, 9e-7)

Table 7: Computational results for random (4.1) of degree 4

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
10	(220, 5004)	20	0:00:21	0:00:25	0:00:30	(1e-9, 4e-7, 5e-5)
15	(680, 38759)	20	0:17:02	0:18:24	0:26:27	(6e-9, 9e-7, 3e-4)
20	(1540, 177099)	5	5:49:08	5:51:44	6:12:59	(5e-11, 5e-9, 3.63)
25	(2925, 593774)	1	51:05:51	51:05:51	51:05:51	(4e-9, 4e-9, 4e-9)

Table 8: Computational results for random (4.1) of degree 6

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
8	(330, 6434)	20	0:01:56	0:02:28	0:03:11	(4e-9, 0.85, 2.08)
10	(715, 24309)	20	0:29:18	0:33:55	0:40:24	(5e-11, 0.02, 2.84)
13	(1820, 125969)	3	10:36:40	16:35:53	18:14:41	(1.64, 1.91, 3.26)
15	(3060, 319769)	1	104:41:59	104:41:59	104:41:59	(1.24, 1.24, 1.24)

Table 9: Computational results for random (4.1) of degree 8

When  $f(x)$  has degree 8 ( $d = 4$ ), the computational results are in Table 9. For  $n = 8$ , solving (4.7)-(4.8) takes a few minutes; for  $n = 10$ , it takes about thirty to forty minutes; for  $n = 13$ , it takes about ten to eighteen hours; for  $n = 15$ , it takes about one hundred hours. The optimal  $Z^*$  usually does not satisfy FEC, and the solutions obtained have very low accuracy. For a few instances, very accurate solutions are found. However, for most instances, the obtained solutions have big errors. This phenomena is probably due to the inexactness of their SOS relaxations.

When  $f(x)$  has degree 10 ( $d = 5$ ), the computational results are in Table 10. For  $n = 8$ , solving (4.7)-(4.8) takes about thirty to ninety minutes; for  $n = 10$ , it takes about twenty five to thirty hours; for  $n = 11$ , it takes about one hundred hours. Similar to the case of degree 8, the optimal  $Z^*$  usually does not satisfy FEC. For most instances, the solutions obtained have big errors. This is probably because of the inexactness of their SOS relaxations.

From the above, for randomly generated forms, we know SOS relaxation (4.2) is very successful when  $\deg(f) = 4$  and also works well when  $\deg(f) = 6$ . However, when  $\deg(f) = 8$

$n$	(N,m)	# inst	time (min, med, max)			err (min, med, max)
6	(252, 3002)	20	0:00:43	0:00:57	0:01:16	(3e-9, 0.98, 2.24)
8	(792, 19447)	10	0:36:26	0:59:51	1:28:46	(0.14, 1.67, 2.15)
10	(2002, 92377)	3	25:58:15	29:30:35	30:37:08	(3e-3, 0.87, 2.51)
11	(3003, 184755)	1	110:48:52	110:48:52	110:48:52	(1.27, 1.27, 1.27)

Table 10: Computational results for random (4.1) of degree 10

or 10, SOS relaxation (4.2) usually fails to find the true global minimum. This observation is consistent with the discovery of Blerkherman [1]. On a regular computer, Algorithm 2.4 would solve big homogeneous polynomial optimization problems which are not solvable by prior existing SOS solvers. This is a big advantage of regularization type methods.

## 4.2 Minimizing odd forms over unit spheres

In some applications, we might need to optimize an odd form  $f(x)$  ( $\deg(f)$  is odd) over  $\mathbb{S}^{n-1}$ . This problem is usually also very difficult. Nesterov [22] proved that it is already NP-hard when  $\deg(f) = 3$ . For odd forms, SOS relaxation (4.2) can not be applied directly. However, after a reformulation, this problem is equivalent to minimizing certain even forms.

Let  $f(x)$  be an odd form of degree  $2d - 1$ , and  $f_{min}^{hmg}$  be its minimum on  $\mathbb{S}^{n-1}$ . Let  $\hat{f}(x, t) = f(x)t$  be a new even form in  $(x, t)$ . Consider homogeneous optimization

$$\widehat{f_{min}^{hmg}} := \min_{\|x\|_2^2 + t^2 = 1} f(x)t. \quad (4.12)$$

It was shown in [24] that

$$f_{min}^{hmg} = \sqrt{2d-1} \left(1 - \frac{1}{2d}\right)^{-d} \widehat{f_{min}^{hmg}}.$$

Since the form  $\hat{f}(x, t) = f(x)t$  is even, SOS relaxation (4.2) can be applied to get a lower bound  $\widehat{f_{sos}^{hmg}}$  for  $\widehat{f_{min}^{hmg}}$ . Then

$$f_{sos}^{hmg} := \sqrt{2d-1} \left(1 - \frac{1}{2d}\right)^{-d} \widehat{f_{sos}^{hmg}}$$

is a lower bound for  $f_{min}^{hmg}$ . When a minimizer  $(\hat{x}, \hat{t})$  is obtained for (4.12), the normalization of  $\hat{x}/\hat{t}$  is then a minimizer of  $f(x)$  over  $\mathbb{S}^{n-1}$ .

**Example 4.7.** Minimize the following cubic form over  $\mathbb{S}^{n-1}$

$$f(x) = \sum_{1 \leq i < j < k \leq n} (x_i x_j x_k + x_i^2 x_j - x_i^2 x_k + x_j x_k^2).$$

We first transform it to (4.12), and then solve its SOS relaxation by Algorithm 2.4. For  $n = 49$ , the computation takes about four hours, and the lower bound found is  $-124.9645$  (only four decimal digits are shown). The optimal  $Z^*$  has rank one, and the extracted solution (only four decimal digits are shown) is

$$-(0.0320, 0.0344, 0.0368, 0.0394, 0.0420, \dots, 0.2304, 0.2434, 0.2600, 0.2836, 0.3238).$$

Its relative error is about  $10^{-9}$ . So it is a global minimizer up to a tiny numerical error.

## 5 Constrained polynomial optimization

Consider general constrained polynomial optimization

$$\begin{aligned} f_{min}^{con} &:= \min_{x \in \mathbb{R}^n} f(x) \\ \text{s.t. } &g_1(x) \geq 0, \dots, g_\ell(x) \geq 0, \end{aligned} \quad (5.1)$$

where  $f(x)$  and  $g_1(x), \dots, g_\ell(x)$  are all multivariate polynomials and have degrees no greater than  $2d$ . Problem (5.1) is NP-hard, even when  $f(x)$  is quadratic and the feasible set is a simplex. Lasserre's relaxation is a typical approach for solving (5.1) approximately. We refer to [5, 13, 16, 17, 34] for the work in this area.

The  $d$ -th Lasserre's relaxation ( $d$  is also called the relaxation order) for (5.1) is

$$\begin{aligned} f_{sos}^{con} &:= \max \quad \gamma \\ \text{s.t. } &f(x) - \gamma = \sigma_0(x) + g_1(x)\sigma_1(x) + \dots + g_\ell(x)\sigma_\ell(x), \\ &\sigma_0(x), \sigma_1(x), \dots, \sigma_\ell(x) \text{ are SOS,} \\ &\deg(\sigma_0), \deg(\sigma_1 g_1), \dots, \deg(\sigma_\ell g_\ell) \leq 2d. \end{aligned} \quad (5.2)$$

Let  $N(k) = \binom{n+k}{k}$ ,  $d_i = \lceil \deg(g_i)/2 \rceil$  and  $g_0(x) = 1$ . Then  $\gamma$  is feasible for (5.2) if and only if there exists  $X^{(i)} \in \mathcal{S}^{N(d-d_i)}$  ( $i = 0, 1, \dots, \ell$ ) such that

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

Define constant symmetric matrices  $A_\alpha^{(0)}, A_\alpha^{(1)}, \dots, A_\alpha^{(\ell)}$  in the way that

$$g_i(x) [x]_{d-d_i} [x]_{d-d_i}^T = \sum_{\alpha \in \mathbb{N}^n: |\alpha| \leq 2d} A_\alpha^{(i)} x^\alpha, \quad i = 0, 1, \dots, \ell. \quad (5.3)$$

Denote  $A_\alpha = (A_\alpha^{(0)}, A_\alpha^{(1)}, \dots, A_\alpha^{(\ell)})$ ,  $X = (X^{(0)}, X^{(1)}, \dots, X^{(\ell)})$ , and define

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

Recall that  $\mathbb{U}_{2d}^n = \{\alpha \in \mathbb{N}^n : 0 < |\alpha| \leq 2d\}$ . If  $f(x) = f_0 + \sum_{\alpha \in \mathbb{U}_{2d}^n} f_\alpha x^\alpha$ , then  $\gamma$  is feasible for (5.2) if and only if there exists  $X$  satisfying

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

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

$$\mathcal{A}(X) = (A_\alpha \bullet X)_{\alpha \in \mathbb{U}_{2d}^n}, \quad b = (f_\alpha)_{\alpha \in \mathbb{U}_{2d}^n}, \quad C = A_0. \quad (5.4)$$

The vector  $b$  has dimension  $m = N(2d) - 1$ . Then SOS relaxation (5.2) is equivalent to

$$\begin{aligned} f_{sdp}^{con} &:= \min \quad C \bullet X \\ \text{s.t. } &\mathcal{A}(X) = b, X \in \mathcal{K}. \end{aligned} \quad (5.5)$$

The dual of (5.5) is

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & \mathcal{A}^*(y) + Z = C, Z \in \mathcal{K}. \end{aligned} \tag{5.6}$$

Here the adjoint  $\mathcal{A}^*(y)$  has the form

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

**Theorem 5.1.** *For notations defined above, it holds  $f_{sos}^{con} = -f_{sdp}^{con} + f_0 \leq f_{min}^{con}$ . If SOS relaxation (5.2) is exact, then  $f_{min}^{con} = -f_{sdp}^{con} + f_0$ .*

The proof is the same as for Theorem 3.1, and hence is omitted. A general approximation bound analysis for Lasserre's relaxation is given in [25].

Algorithm 2.4 can be applied to solve the conic SDP (5.5)-(5.6). Unlike in Sections 3 and 4, there is usually no explicit formula for  $(\mathcal{A}\mathcal{A}^*)^{-1}$  in Lasserre's relaxation, and it is typically quite difficult to compute  $(\mathcal{A}\mathcal{A}^*)^{-1}$ . So we just simply use  $(\text{diag}(\mathcal{A}\mathcal{A}^*))^{-1}$  as a preconditioner. Suppose  $(X^*, y^*, Z^*)$  is optimal for (5.5)-(5.6). Then  $-f_{sdp}^{con} + f_0$  is a lower bound for the minimum  $f_{min}^{con}$ . The information for minimizers could be obtained from  $Z^*$ . Note  $Z^* = (Z_0^*, Z_1^*, \dots, Z_\ell^*)$ . Since  $Z^* \in \mathcal{K}$ , every  $Z_i^* \succeq 0$ . When  $Z_0^*$  has rank one,  $f_{min}^{con} = f_{sos}^{con}$  and a global minimizer for (5.1) can be extracted. Actually, if  $\text{rank}(Z_0^*) = 1$ , (5.3) implies  $Z_0^* = [x^*]_d [x^*]_d^T$  for some  $x^* \in \mathbb{R}^n$ , and hence  $y_\alpha^* = -(x^*)^\alpha$  for every  $\alpha \in \mathbb{U}_{2d}^n$ . Furthermore, the structures of  $Z_i^*$  implies that  $Z_i^* = g_i(x^*) [x^*]_{d-d_i} [x^*]_{d-d_i}^T$ . Hence, every  $g_i(x^*) \geq 0$  and  $x^*$  is feasible for (5.1). Then, for every  $x \in \mathbb{R}^n$  feasible in (5.1), we have

$$-f(x^*) = -f_0 + b^T y^* \geq -f_0 + \sum_{\alpha \in \mathbb{U}_{2d}^n} -b_\alpha x^\alpha = -f(x),$$

because  $y^*$  is optimal and for every  $x$  feasible in (5.1)

$$Z = ([x]_d [x]_d^T, g_1(x) [x]_{d-d_1} [x]_{d-d_1}^T, \dots, [x]_{d-d_\ell} [x]_{d-d_\ell}^T)$$

is feasible for (5.6). So  $x^*$  is a global minimizer of (5.1).

When  $\text{rank}(Z_0^*) > 1$ , if FEC holds, several global minimizers could be extracted, e.g., by the method in [8]. Typically, FEC fails if there are a lot of global minimizers or Lasserre's relaxation (5.2) is not exact. If this happens, it is not clear how to get minimizers.

## 5.1 Numerical experiments

Now we present some numerical examples of applying Algorithm 2.4 to solve the conic SDP (5.5)-(5.6) which is equivalent to Lasserre's relaxation (5.2). As in Section 3.1, the computations are implemented in Matlab on the same machine. The parameters and stopping criteria of Algorithm 2.4 are also the same as there. The  $(\text{diag}(\mathcal{A}\mathcal{A}^*))^{-1}$  is used as a preconditioner. If FEC holds, we use the method in [8] to get one or several global minimizers  $x^*$ ; otherwise, just set  $x^* = Z^*(2 : n + 1, 1)$ . In either case, the relative error of  $x^*$  is measured as

$$err = \frac{|f(x^*) - f_{sos}^{con}|}{\max\{1, |f(x^*)|\}}.$$

Note we always have  $f_{sos}^{con} \leq f_{min}^{con}$ . If the obtained  $x^*$  is feasible for (5.1), then  $err = 0$  if and only if  $x^*$  is a global minimizer.

**Example 5.2.** Minimize the polynomial

$$\sum_{i=1}^n x_i + \sum_{1 \leq i < j \leq n} x_i x_j + \sum_{1 \leq i < j < k \leq n} x_i x_j x_k + \sum_{1 \leq i < j < k < l \leq n} x_i x_j x_k x_l$$

over the unit ball  $B(0,1) = \{x \in \mathbb{R}^n : \|x\|_2 \leq 1\}$ . It has degree 4 and is a summation of square free monomials. We apply the 2<sup>nd</sup> Lasserre's relaxation (5.2) to solve this problem. The resulting cone  $\mathcal{K}$  has 2 blocks. When  $n = 50$ , solving (5.5)-(5.6) takes about 5 hours. The computed lower bound  $f_{sos}^{con} = -1.5240$  (only four decimal digits are shown). The optimal  $Z^*$  has rank 50 and FEC holds. There are 50 solutions extracted. One of them (only four decimal digits are shown) is

$$(0.8852, \quad -0.0665, \quad -0.0665, \quad \dots, \quad -0.0665, \quad -0.0665).$$

It has unit norm, and hence is feasible. Its relative error is  $7 \cdot 10^{-7}$ . So it is a global minimizer up to a tiny numerical error. Actually all its permutations are also solutions of the same accuracy, since the objective polynomial is symmetric in  $x$ .

**Example 5.3.** Minimize the polynomial

$$\sum_{1 \leq i < j \leq n} x_i^2 x_j^2 + i x_i^3 x_j + j x_i x_j^3$$

over the hypercube  $[-1,1]^n = \{x \in \mathbb{R}^n : x_i^2 \leq 1\}$ . It has degree 4. So we apply the 2<sup>nd</sup> Lasserre's relaxation (5.2) to solve this problem. The resulting cone  $\mathcal{K}$  has  $n + 1$  blocks. When  $n = 40$ , solving (5.5)-(5.6) takes about 22 hours, and the computed lower bound  $f_{sos}^{con} = -2751.5$  (only the first 5 digits are shown). The optimal  $Z^*$  has rank two and FEC holds. The two extracted solutions (only 4 decimal digits are shown) are:

$$\begin{aligned} &\pm(1, 1, 1, 1, 1, 1, 1, 0.9434, 0.8920, 0.8483, 0.8105, 0.7775, 0.7482, 0.7220, 0.6985, 0.6771, \\ &\quad 0.6576, 0.6397, 0.6232, 0.6080, 0.5938, 0.5806, 0.5682, 0.5566, 0.5457, 0.5354, 0.5257, \\ &\quad 0.5165, 0.5077, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1). \end{aligned}$$

They are obviously feasible solutions. The relative error is around  $6 \cdot 10^{-7}$ . So they are global minimizers up to a small numerical error.

**Example 5.4.** Minimize the polynomial

$$\sum_{1 \leq i < j \leq n} x_i x_j + x_i^3 + x_j^3 - x_i^2 x_j^2$$

over the hypercube  $[-1,1]^n = \{x \in \mathbb{R}^n : x_i^2 \leq 1\}$ . The degree is 4. So we apply the 2<sup>nd</sup> Lasserre's relaxation (5.2). The resulting cone  $\mathcal{K}$  has  $n + 1$  blocks. For  $n = 50$ , solving (5.5)-(5.6) takes about 18 hours. The computed lower bound  $f_{sos}^{con} = -3600$  (rounded to integers). The optimal  $Z^*$  has rank one, and the solution extracted is (rounded to integers)

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

It is clearly a feasible solution. Its relative error is zero, so it is a global minimizer.

**Example 5.5.** Minimize the polynomial

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

over the unit ball  $B(0,1)$ . It has degree 6. So we apply the 3<sup>rd</sup> Lasserre's relaxation (5.2). The resulting cone  $\mathcal{K}$  has 2 blocks. When  $n = 20$ , solving (5.5)-(5.6) takes about 57 hours. The computed lower bound  $f_{sos}^{con} = -20$  (rounded to integers). The optimal  $Z^*$  has rank one, and the solution extracted (rounded to integers) is

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

It is feasible and has zero relative error. So it is a global minimizer.

**Example 5.6.** Consider the polynomial optimization

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{1 \leq i < j < k \leq n} (i+j)x_ix_jx_k + (j+k)x_i^2x_j^2x_k^2 \\ \text{s.t.} \quad & x_1^4 + \dots + x_n^4 \leq 1. \end{aligned}$$

The constraint is the unit ball defined by the standard 4-norm. The degree is 6. We apply the 3<sup>rd</sup> Lasserre's relaxation (5.2) to solve this problem. The resulting cone  $\mathcal{K}$  has 2 blocks. When  $n = 15$ , solving (5.5)-(5.6) takes about 1.5 hours. The computed lower bound  $f_{sos}^{con} = -575.5928$  (only four decimal digits are shown). The optimal  $Z^*$  has rank one, and the solution extracted (only four decimal digits are shown) is

$$\begin{aligned} & -(0.4034, 0.4274, 0.4486, 0.4674, 0.4839, 0.4983, 0.5107, 0.5211, 0.5296, 0.5363, \\ & \quad 0.5410, 0.5437, 0.5444, 0.5430, 0.5393). \end{aligned}$$

It has unit 4-norm, and hence is feasible. The relative error is around  $4 \cdot 10^{-7}$ . So it is a global minimizer up to a small numerical error.

**Example 5.7.** Consider the polynomial optimization

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{1 \leq i < j < k \leq n} x_ix_jx_k(1+x_i+x_j+x_k) + ix_i^6 + jx_j^6 + kx_k^6 \\ \text{s.t.} \quad & x_1^4 + \dots + x_{\frac{n}{2}}^4 \leq 1, \quad x_{\frac{n}{2}+1}^4 + \dots + x_n^4 \leq 1 \end{aligned}$$

where  $n$  is an even integer. The degree is 6, and the constraint is a bi-unit ball under the 4-norm. We apply the 3<sup>rd</sup> Lasserre's relaxation (5.2) to solve this problem. The resulting cone  $\mathcal{K}$  has 3 blocks. For  $n = 16$ , solving (5.5)-(5.6) takes about 19 hours, and the computed lower bound  $f_{sos}^{con} = -1.1078$  (only four decimal digits are shown). The optimal  $Z^*$  has rank one, and the extracted solution (only four decimal digits are shown) is

$$\begin{aligned} & -(0.2418, 0.2208, 0.2085, 0.2000, 0.1934, 0.1882, 0.1838, 0.1800, 0.1767, 0.1738, \\ & \quad 0.1712, 0.1688, 0.1667, 0.1647, 0.1629, 0.1612). \end{aligned}$$

It is a feasible solution, and the relative error is around  $4 \cdot 10^{-9}$ . So it is a global minimizer up to a tiny numerical error.

**Example 5.8.** Consider the polynomial optimization

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{1 \leq i < j < k \leq n/2} ix_i x_j x_k + jx_{\frac{n}{2}+i} x_{\frac{n}{2}+j} x_{\frac{n}{2}+k} + kx_i x_j x_k x_{\frac{n}{2}+i} x_{\frac{n}{2}+j} x_{\frac{n}{2}+k} \\ \text{s.t.} \quad & x_1^4 + \cdots + x_{\frac{n}{2}}^4 \leq 1, \quad x_{\frac{n}{2}+1}^4 + \cdots + x_n^4 \leq 1 \end{aligned}$$

where  $n$  is even. Since the degree is 6, we apply the  $3^{rd}$  Lasserre's relaxation (5.2). The cone  $\mathcal{K}$  has 3 blocks. For  $n = 20$ , solving (5.5)-(5.6) takes about 32 hours. The computed lower bound  $f_{sos}^{con} = -153.6180$  (only four decimal digits are shown). The optimal  $Z^*$  has rank one, and the solution extracted (only four decimal digits are shown) is

$$\begin{aligned} & -(-0.3642, 0.3955, 0.5042, 0.5589, 0.5892, 0.6049, 0.6109, 0.6104, 0.6057, 0.5991, \\ & \quad 0.5828, 0.5173, 0.5193, 0.5306, 0.5459, 0.5619, 0.5763, 0.5869, 0.5919, 0.5896). \end{aligned}$$

The first and second subvector both have unit 4-norm, so it is a feasible solution. Its relative error is around  $5 \cdot 10^{-8}$ . So it is a global minimizer up to a tiny numerical error.

## 5.2 Sparse SOS relaxation for constrained optimization

In many applications, polynomials in optimization are often sparse. It is important to exploit sparsity for computational efficiency. We refer to [10, 12, 11, 14, 26, 29, 38].

Suppose each polynomial  $g_j$  only involves a subvector  $x_{I_j} = \{x_i : i \in I_j\}$  for an index set  $I_j \subseteq \{1, 2, \dots, n\}$ . Consider the sparse polynomial optimization

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_1(x_{I_1}) \geq 0, \dots, g_\ell(x_{I_\ell}) \geq 0. \end{aligned} \tag{5.7}$$

The sparse generalized Lagrangian function (see Waki et al. [38]) for the above is

$$L(x, \phi) = f(x) - \sum_{j=1}^{\ell} \phi_j(x_{I_j}) g_j(x_{I_j}),$$

where  $\phi_j(x_{I_j})$  are SOS polynomials in  $x_{I_j}$ . Let  $\{C_1, \dots, C_r\}$  be the set of maximal cliques of a chordal extension of correlative sparsity pattern (csp) matrix of  $L(x, \phi)$  (see Waki et al. [38]). A sparse Lasserre's relaxation of order  $d$  proposed in [38] for (5.7) is

$$\begin{aligned} \max \quad & \gamma \\ \text{s.t.} \quad & f(x) - \gamma = \sum_{i=1}^r \sigma_i(x_{C_i}) + \sum_{j=1}^{\ell} \phi_j(x_{I_j}) g_j(x_{I_j}), \\ & \deg(\sigma_i), \deg(\phi_j g_j) \leq 2d, \sigma_i, \phi_j \text{ are SOS,} \\ & i = 1, \dots, r, j = 1, \dots, \ell. \end{aligned} \tag{5.8}$$

Like the general dense case, (5.8) can also be formulated in the form of conic semidefinite optimization (2.1). For convenience, we still use the same notations as before. Note  $\gamma$  is feasible for (5.8) if and only if there exists  $X^{(i)}$  ( $i = 1, \dots, r + \ell$ ) satisfying

$$\begin{aligned} f(x) - \gamma &= \sum_{i=1}^r [x_{C_i}]_d^T X^{(i)} [x_{C_i}]_d + \sum_{j=1}^{\ell} g_j(x) [x_{I_j}]_{d-d_j}^T X^{(r+j)} [x_{I_j}]_{d-d_j}, \\ X^{(1)} &\succeq 0, \dots, X^{(r)} \succeq 0, X^{(r+1)} \succeq 0, \dots, X^{(r+\ell)} \succeq 0. \end{aligned}$$

Let  $\mathbb{W}$  be the set of exponents of nonconstant monomials appearing in (5.8). For  $\alpha \in \mathbb{W}$ , define constant symmetric matrices  $A_\alpha^{(1)}, \dots, A_\alpha^{(r+\ell)}$  in the way that

$$\begin{aligned} [x_{C_i}]_d [x_{C_i}]_d^T &= \sum_{\alpha \in \{0\} \cup \mathbb{W}} A_\alpha^{(i)} x^\alpha, i = 1, \dots, r, \\ g_j(x) [x]_{d-d_j} [x]_{d-d_j}^T &= \sum_{\alpha \in \{0\} \cup \mathbb{W}} A_\alpha^{(r+j)} x^\alpha, j = 1, \dots, \ell. \end{aligned} \quad (5.9)$$

It is obvious that

$$A_\alpha^{(i)} \in \mathcal{S}^{\binom{|C_i|+d}{d}}, i = 1, \dots, r, \quad A_\alpha^{(r+j)} \in \mathcal{S}^{\binom{|I_j|+d-d_j}{d-d_j}}, j = 1, \dots, \ell.$$

Denote  $A_\alpha = (A_\alpha^{(1)}, \dots, A_\alpha^{(r+\ell)})$ ,  $X = (X^{(1)}, \dots, X^{(r+\ell)})$ , and define

$$\mathcal{K} = \mathcal{S}_+^{\binom{|C_1|+d}{d}} \times \dots \times \mathcal{S}_+^{\binom{|C_r|+d}{d}} \times \mathcal{S}_+^{\binom{|I_1|+d-d_1}{d-d_1}} \times \dots \times \mathcal{S}_+^{\binom{|I_\ell|+d-d_\ell}{d-d_\ell}}.$$

If  $f(x) = f_0 + \sum_{\alpha \in \mathbb{W}} f_\alpha x^\alpha$ , the constraint of (5.8) is equivalent to

$$\begin{aligned} A_0 \bullet X + \gamma &= f_0, \\ A_\alpha \bullet X &= f_\alpha \quad \forall \alpha \in \mathbb{W}, \\ X &\in \mathcal{K}. \end{aligned}$$

Similar to (5.4), define  $\mathcal{A}, b, C$  as

$$\mathcal{A}(X) = (A_\alpha \bullet X)_{\alpha \in \mathbb{W}}, \quad b = (f_\alpha)_{\alpha \in \mathbb{W}}, \quad C = A_0. \quad (5.10)$$

The dimension of  $b$  is  $m = |\mathbb{W}|$ , the cardinality of  $\mathbb{W}$ . So (5.8) is equivalent to

$$\begin{aligned} \min \quad & C \bullet X \\ \text{s.t.} \quad & \mathcal{A}(X) = b, X \in \mathcal{K}. \end{aligned} \quad (5.11)$$

The dual of the above is

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & \mathcal{A}^*(y) + Z = C, Z \in \mathcal{K}. \end{aligned} \quad (5.12)$$

The adjoint  $\mathcal{A}^*(y)$  now takes the form

$$\mathcal{A}^*(y) = \left( \sum_{\alpha \in \mathbb{W}} y_\alpha A_\alpha^{(1)}, \sum_{\alpha \in \mathbb{W}} y_\alpha A_\alpha^{(2)}, \dots, \sum_{\alpha \in \mathbb{W}} y_\alpha A_\alpha^{(r+\ell)} \right).$$

The information about minimizers would be obtained from  $Z^*$ . We refer to Laurent and Mourrain [18].

**Example 5.9.** Consider the following sparse polynomial optimization

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{k \in \{1, \dots, 34\}} \left( \sum_{i \in J_k} (x_i + x_i^6) - \sum_{i, j \in J_k: i < j} x_i^2 x_j^2 \right) \\ \text{s.t.} \quad & 1 - \|x_{J_k}\|_4^4 \geq 0, k = 1, \dots, 34. \end{aligned}$$

Here the index set  $J_k = \{j : \max(1, 3k - 3) \leq j \leq \min(n, 3k + 2)\}$ , and  $n = 100$  is the total number of variables. The degree is 6. We apply the sparse Lasserre's relaxation (5.8) of order 3. Since the csp matrix here is banded, its csp graph is chordal. So the set of maximal cliques is  $\{J_1, J_2, \dots, J_{33}\}$  (note  $J_{34} \subset J_{33}$ ). We apply Algorithm 2.4 to solve the resulting conic SDP (5.11)-(5.12). It takes about 15 minutes, and the computed lower bound is  $-194.8056$  (only four decimal digits are shown). To get a minimizer, we just simply set  $x_i^* = y_{e_i}^*$  for  $i = 1, \dots, n$ , and get the solution (only four decimal digits are shown)

$$-(0.7066 * 2, (0.6394 * 3, 0.6385 * 3) * 16, 0.7076 * 2).$$

In the above, the product  $a * t$  means  $a$  is repeated for  $t$  times in a row. The above solution has relative error around  $8 \cdot 10^{-9}$ , so it is a global minimizer up to a tiny numerical error.

## 6 Conclusions and some discussions

This paper proposes regularization type methods for solving SOS and Lasserre's relaxations in large scale polynomial optimization. After a description of these methods for general conic semidefinite optimization, we show how to apply them in solving unconstrained, homogeneous and constrained polynomial optimization. As demonstrated by extensive numerical experiments, significantly larger problems would be solved efficiently by them. For a quartic polynomial optimization problem with 100 variables, its SOS relaxation would be solved on a regular computer, which is almost impossible by applying prior existing SOS solvers.

The proposed regularization methods for solving large scale polynomial optimization are still in their very early stages. Much work need to be done for improving their practical performance. From our experience, the following issues are important.

- The performance of Algorithm 2.4 depends very much on CG iterations for solving linear system (2.9). In the inner loop, if (2.9) is solved efficiently by CG, then Algorithm 2.4 usually converges fast; otherwise, a lot of time would be spent on solving (2.9), which finally makes Algorithm 2.4 run very slow. So the conditioning of (2.9) seriously affects the performance. Generally, the accuracy parameters in Algorithm 2.4 can not be set too small.
- For unconstrained and homogeneous polynomial optimization, there are explicit formulae for the inverse  $(\mathcal{AA}^*)^{-1}$  which is used as a preconditioner for solving (2.9) by CG. For constrained polynomial optimization, there is no explicit formula for  $(\mathcal{AA}^*)^{-1}$ , and we just use  $(\text{diag}(\mathcal{AA}^*))^{-1}$  as a preconditioner. Generally, is there a better preconditioner in polynomial optimization? This is not clear for us.
- It is critical to update  $\sigma_k$  in Algorithm 2.4. From our numerical experience, it is better to choose relatively bigger  $\rho$  if (2.9) is solved accurately by CG; otherwise, we should choose relatively small  $\rho$ . Generally, what is the best strategy for updating  $\sigma_k$ ?
- Algorithms 2.2 and 2.4 are basically special cases of Algorithm 2.3. Is there a more efficient specification of Algorithm 2.3 for solving large scale SOS or Lasserre's relaxations? Or does polynomial optimization have any special features in applying Algorithm 2.3 or its specifications efficiently? This is not clear for us.

The regularization methods in Section 3 are designed for solving general big conic semidefinite optimization (2.1). A big advantage of them is the low memory requirement for storing matrices and vectors during the computation. Generally, the computer memory could be used to a big extent for storing problem data and the computation does not require much more extra memory. Therefore, if a big SDP fits the computer memory, we can always run the algorithm while possibly waiting for a long time.

**Acknowledgement** The author would like very much to thank Bill Helton and Igor Klep for fruitful discussions on this work.

## References

- [1] G. Blekherman. There are significantly more nonnegative polynomials than sums of squares. *Israel J. Math.* 153 (2006), 355-380.
- [2] I.M. Bomze and E. de Klerk. Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. *Journal of Global Optimization*, 24(2), 163-185.
- [3] R. Curto and L. Fialkow. The truncated complex K-moment problem. *Trans. Am. Math. Soc.* 352, 2825-2855 (2000).
- [4] J. Demmel. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.
- [5] J. Demmel, J. Nie and V. Powers. Representations of positive polynomials on non-compact semialgebraic sets via KKT ideals. *Journal of Pure and Applied Algebra*, Vol. 209, No. 1, pp. 189-200, 2007.
- [6] E. de Klerk, and D.V. Pasechnik, D.V. Approximating of the stability number of a graph via copositive programming. *SIAM Journal on Optimization*, 12(4), 875-892.
- [7] 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>
- [8] D. Henrion and J. Lasserre. Detecting global optimality and extracting solutions in GloptiPoly. *Positive polynomials in control*, D. Henrion, A. Garulli Eds., Lecture Notes on Control and Information Sciences, Vol. 312, Springer, Berlin, 2005, pp. 293-310.
- [9] D. Jibeteau and M. Laurent. Semidefinite approximations for global unconstrained polynomial optimization. *SIAM J. Optim.* 16 (2005), no. 2, 490-514
- [10] S. Kim, M. Kojima and H. Waki. Generalized lagrangian duals and sums of squares relaxations of sparse polynomial optimization problems. *SIAM Journal on Optimization*, Vol. 15 (3) 697-719 (2005).
- [11] Kim, M. Kojima and H. Waki. Exploiting sparsity in SDP relaxation for sensor network localization. *SIAM Journal on Optimization*, Vol. 20 (1) 192-215, 2009.

- [12] M. Kojima, S. Kim and H. Waki. Sparsity in Sums of Squares of Polynomials. *Mathematical Programming*, Vol.103 (1) 45-62 (2005).
- [13] J. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. on Optim.*, Vol. 11, No. 3, 796-817, 2001.
- [14] J. Lasserre. Convergent SDP-relaxations in polynomial optimization with sparsity. *SIAM Journal on Optimization*, Vol. 17, No. 3, 822-843, 2006.
- [15] J. Lasserre. A sum of squares approximation of nonnegative polynomials. *SIAM Journal on Optimization*, Vol. 16, No. 3, pp. 751-765.
- [16] J. Lasserre. A semidefinite programming approach to the generalized problem of moments. *Math. Program.* 112 (2008), no. 1, Ser. B, 65–92.
- [17] M. Laurent. Sums of squares, moment matrices and optimization over polynomials. *Emerging Applications of Algebraic Geometry, IMA Volumes in Mathematics and its Applications*, Vol. 149, M. Putinar and S. Sullivant (eds.), Springer, pp. 157–270, 2009.
- [18] M. Laurent and B. Mourrain. A generalized flat extension theorem for moment matrices. *Archiv der Mathematik*, 93(1), 2009.
- [19] 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.
- [20] T.S. Motzkin and E.G. Straus. Maxima for graphs and a new proof of a theorem of Turán. *Canadian J. Math.*, 17 (1965), pp. 533-540.
- [21] Y. Nesterov. Squared functional systems and optimization problems. *High Performance Optimization* H.Frenk et al. (eds.), Kluwer Academic Publishers, 2000, pp. 405-440.
- [22] Y. Nesterov. Random walk in a simplex and quadratic optimization over convex polytopes. *CORE Discussion Paper*, UCL, Louvain-la-Neuve, Belgium, 2003.
- [23] J. Nie, J. Demmel and B. Sturmfels. Minimizing polynomials via sum of squares over the gradient ideal. *Math. Program.* 106 (2006), no. 3, Ser. A, 587–606.
- [24] J. Nie. Sum of squares methods for minimizing polynomial forms over spheres and hypersurfaces. *Preprint*, 2009. <http://www.math.ucsd.edu/~njw>
- [25] J. Nie. An approximation bound analysis for Lasserre’s relaxation in multivariate polynomial optimization. *Preprint*, 2009. <http://www.math.ucsd.edu/~njw>
- [26] 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).
- [27] P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, Ser. B 96 (2003), No. 2, 293-320.

- [28] 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), American Mathematical Society, 2003, pp. 83-100.
- [29] P.A. Parrilo. Exploiting structure in sum of squares programs. *Proceedings for the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii, 2003.
- [30] J. Povh, F. Rendl, and A. Wiegele. A boundary point method to solve semidefinite programs. *Computing* 78, 277-286 (2006).
- [31] R.T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control and Optim.*, 14 (1976), No.5, pp. 877-898.
- [32] R.T. Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Math. Oper. Res.* , 1 (1976), no. 2, 97-116.
- [33] J.F. Sturm. SeDuMi 1.02 a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11 & 12 (1999), pp. 625-653.
- [34] M. Schweighofer. Optimization of polynomials on compact semialgebraic sets. *SIAM J. Optim.* 15 (2005), no. 3, 805-825.
- [35] M. Todd. Semidefinite Optimization. *Acta Numerica* 10 (2001), pp. 515-560.
- [36] K.C. Toh, M.J. Todd, and R.H. Tutuncu. SDPT3 a Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11 (1999), pp. 545-581.
- [37] L. Vandenberghe and S. Boyd. Semidefinite Programming. *SIAM Review* 38 (1996), pp. 49-95.
- [38] 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 (1) 218-242 (2006).
- [39] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of semidefinite programming*, Kluwers Publisher, 2000.
- [40] X.Y. Zhao, D.F. Sun, and K.C. Toh. A Newton-CG Augmented Lagrangian method for semidefinite programming. *Preprint*, National University of Singapore, March 2008.