An accelerated first-order method for solving SOS relaxations of unconstrained polynomial optimization problems

Dimitris Bertsimas\textsuperscript{a}, Robert M. Freund\textsuperscript{a} & Xu Andy Sun\textsuperscript{b}

\textsuperscript{a} Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA
\textsuperscript{b} IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 10598, USA

Published online: 12 Mar 2012.

To cite this article: Dimitris Bertsimas , Robert M. Freund & Xu Andy Sun (2013) An accelerated first-order method for solving SOS relaxations of unconstrained polynomial optimization problems, Optimization Methods and Software, 28:3, 424-441, DOI: 10.1080/10556788.2012.656114

To link to this article: http://dx.doi.org/10.1080/10556788.2012.656114

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the "Content") contained in the publications on our platform. However, Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Any opinions and views expressed in this publication are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor and Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden. Terms &
An accelerated first-order method for solving SOS relaxations of unconstrained polynomial optimization problems

Dimitris Bertsimas\textsuperscript{a}, Robert M. Freund\textsuperscript{a} and Xu Andy Sun\textsuperscript{b}\textsuperscript{*}

\textsuperscript{a}Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA; \textsuperscript{b}IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA

\textit{(Received 6 March 2011; final version received 1 January 2012)}

Our interest lies in solving sum of squares (SOS) relaxations of large-scale unconstrained polynomial optimization problems. Because interior-point methods for solving these problems are severely limited by the large-scale, we are motivated to explore efficient implementations of an accelerated first-order method to solve this class of problems. By exploiting special structural properties of this problem class, we greatly reduce the computational cost of the first-order method at each iteration. We report promising computational results as well as a curious observation about the behaviour of the first-order method for the SOS relaxations of the unconstrained polynomial optimization problem.

\textbf{Keywords:} polynomial optimization, semidefinite programming relaxation, accelerated first order methods, global optimization

1. Introduction

We are interested in solving the unconstrained polynomial optimization problem:

\[(P) \quad \gamma^* = \min_{x \in \mathbb{R}^n} p(x),\]

where \(p(x) : \mathbb{R}^n \to \mathbb{R}\) is a real-valued polynomial in \(n\) variables and of even degree \(d = 2m\), where \(m\) is an integer. Polynomial optimization has connections to many fields such as algebraic geometry [7], the problem of moments [6,13] and semidefinite programming (SDP) [27]. It has also been a useful tool in control theory [22], optimized moment inequalities in probability theory [5], combinatorial optimization [17] and stochastic modelling [4]. However, despite interesting theoretical properties and rich modelling capabilities, polynomial optimization techniques have not been extensively used in real applications. One reason for this is the limitation of current computational schemes for solving large instances of polynomial optimization problems. Using the most successful solution methodology – the hierarchy of SDP relaxations [14,23], the resulting SDP problem involves a (primal) matrix variable of order \(\left(\begin{array}{c}n+m \\ n \end{array}\right) \times \left(\begin{array}{c}n+m \\ n \end{array}\right) = O(n^m \times n^m)\) and a (dual) vector variable of dimension \(\left(\begin{array}{c}n+2m \\ n \end{array}\right) = O(n^2m)\), which grows exponentially in the degree of the polynomial. Currently, the most widely used software (such as Gloptipoly [9] and SOSTOOLS

\textsuperscript{*}Corresponding author. Email: sunx@mit.edu

© 2013 Taylor & Francis
solve the SDP relaxation by interior-point methods (IPMs), where the exponential growth in the SDP dimension severely limits the size of solvable instances.

In this paper, we explore the possibility of improving the practical solvability of polynomial optimization problems. Instead of using powerful but expensive IPMs, we take advantage of the recent progress on solving conic optimization problems with first-order algorithms. In particular, we apply a variant of Nesterov’s accelerated first-order method (AFOM) [1,19] to the SDP relaxation of an unconstrained polynomial optimization problem, using a reformulation proposed in Lu [15]. The key to the success of such an approach depends on the ability to exploit the special structure of the reformulation in the context of the projection operators that are used in the AFOM. To accomplish this, we identify a special property of the SDP reformulation, and we show that this property enables very fast computation of projections onto a certain affine manifold that arises in the AFOM. The reduced computational cost of the first-order algorithm is then demonstrated via numerical experiments, which show that the AFOM can solve larger instances than the existing IPM-based software. For both 4-degree and 6-degree polynomials, the first-order method solves instances of dimensions roughly twice as large (in terms of the number of polynomial variables n) as existing IPM software (namely, the SOS package in SeDuMi) can handle. The first-order method is also faster than the existing software for medium size problems when solved to comparable accuracy.

Over the past several years, research on AFOMs has grown substantially, especially on the computational frontier of applying such methods to solving large-scale problems (see, e.g. compressed sensing [3,15], covariance selection [8,16], and image processing [2]). However, to the best of our knowledge, such methods have not been applied to general unconstrained sum of squares (SOS) polynomial optimization. Herein, we study the behaviour of AFOMs on solving this important class of problems. We also discuss a curious observation on the convergence behaviour of this method that is not a derivative of the usual convergence analysis.

The paper is organized as follows: In Section 2, we first briefly review the method of SDP relaxation of unconstrained polynomial optimization problems and reformulations that are suitable for applying the intended AFOM. We then demonstrate the special structure of the reformulation that allows for fast computation of the affine manifold projection problem at each iteration. In Section 3, we report computational results and provide a comparison with existing interior-point-based methods. We also provide some interesting observations on the convergence behaviour of the AFOM.

1.1 Notation and definitions

Let \( S^n \) denote the space of \( n \times n \) real symmetric matrices, and \( S^n_+ \) denote the cone of \( n \times n \) positive semidefinite matrices. Let \( \succeq \) denote the standard ordering on \( S^n \) induced by the cone \( S^n_+ \). For \( X, Y \in S^n \), let \( X \bullet Y \) denote the Frobenius inner product, and \( \|X\|_F \) denote the induced matrix norm, namely \( \|X\|_F = \sqrt{X \bullet X} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^2} \). For \( x, y \in \mathbb{R}^n \), let \( x^T y \) denote the standard Euclidean inner product, and \( \|x\|_2 \) denote the Euclidean norm. More generally, for a finite-dimensional inner product space \( U \), let \( \| \cdot \| \) denote the induced norm. For a compact convex set \( C \subset U \), define a distance function \( d_C(u) := \min\{\|u - v\| : v \in C\} \) for all \( u \in U \). Define \( \Pi_C(u) := \text{arg min}\{\|u - v\| : v \in C\} \). The ‘prox’ or squared distance function \( d^2_C(u) \) is a Lipschitz differentiable convex function with Lipschitz constant \( L = 2 \), whose gradient operator is given by \( \nabla_u d^2_C(u) = 2(u - \Pi_C(u)) \), see [10].

Let \( \mathbb{N} \) denote the set of nonnegative integers, and \( \mathbb{N}^d_0 := \{\alpha \in \mathbb{N}^n \mid \alpha := \sum_{i=1}^{n} \alpha_i \leq d\} \) for \( n, d \in \mathbb{N} \) and \( n > 0 \). For \( \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n \), and \( x = (x_1, \ldots, x_n) \in \mathbb{R}^n \), let \( x^\alpha \) denote the monomial \( x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} \) of degree \( \alpha \). For \( d = 2m \), let

\[
v_{2m}(x) := \{1, x_1, x_2, \ldots, x_n, x_1^2, x_1 x_2, \ldots, x_1 x_n, x_2^2, x_2 x_3, \ldots, x_n^{2m}\}
\]
be a basis for the $n$ variable and degree $d = 2m$ real-valued polynomial, so that we can write
\[ p(x) = \sum_{\alpha \in \mathbb{N}^n_{2m}} p_\alpha x^\alpha = p^T v_{2m}(x). \] (2)

Let $M$ denote the number of components of $v_{2m}(x)$, namely $M = \binom{n+2m}{n}$. Let $N := \binom{n+m}{n}$. For each $\alpha \in \mathbb{N}^n_{2m}$, let $A_\alpha \in \mathbb{S}^N$ denote the matrix corresponding to the following operator:

\[ A_\alpha(X) := A_\alpha \cdot X = \sum_{\delta} \sum_{\nu = \alpha - \delta} X_{\delta \nu} \quad \text{for all } X \in \mathbb{S}^N, \] (3)

where $X$ is indexed by $\delta, \nu \in \mathbb{N}^n_m$. The matrices $A_\alpha$ are called the basic moment matrices, which are used to define the moment matrix $M_{2m}(y)$ of the finite moment sequence $y \in \mathbb{R}^M$ as

\[ M_{2m}(y) := \sum_{\alpha \in \mathbb{N}^n_{2m}} y_\alpha A_\alpha. \]

A polynomial $p(x)$ is called a SOS polynomial if $p(x)$ can be written as

\[ p(x) = \sum_{i=1}^s q_i^2(x) \]

for some polynomials $q_i(x)$ of degree at most $m$, and $X \in \mathbb{S}^N_+$. Matching monomial coefficients on both sides above, it follows that $X$ must satisfy:

\[ X_{00} = -\gamma, \]

\[ \sum_{\delta} \sum_{\nu = \alpha - \delta} X_{\delta \nu} = p_\alpha \quad \text{for } \alpha \in \mathbb{N}^n_{2m}, |\alpha| > 0, \] (5)

which yields the following SDP formulation of (4)

\[ \gamma^*_p = \min_{X \in \mathbb{S}^N} A_0 \cdot X \]

s.t. $A_\alpha \cdot X = p_\alpha \quad \text{for } \alpha \in \mathbb{N}^n_{2m}, |\alpha| > 0,$

\[ X \succeq 0. \] (6)
The dual of (6) is
\[
\gamma_d^* = \max_{y \in \mathbb{R}^M, A \in \mathbb{N}^N} \sum_{\alpha \in \mathbb{N}_{2m}^n, |\alpha| > 0} p_\alpha y_\alpha,
\]
s.t. \[
\sum_{\alpha \in \mathbb{N}_{2m}^n, |\alpha| > 0} y_\alpha A_\alpha + S = A_0,
\]
\[
S \succeq 0.
\]
We have \( \gamma_p^* = -\gamma_{sos}^* \). If (6) has a feasible solution, then there is no duality gap between (6) and (7), i.e. \( \gamma_p^* = \gamma_d^* \), see [14, Proposition 3.1].

### 2.2 Reformulations of the SDP problem

In order to apply an AFOM to solve (6) and (7), one needs to reformulate these problems so that the resulting reformulated feasible region is a simple convex set for which computation of projections can be done very efficiently. Several such reformulations have been proposed in the recent literature, see, for example [11,12,15]. In particular, Lu [15] has derived a detailed worst-case complexity analysis and comparison of the cited proposals, and successfully applied one recommended scheme to solve a class of linear programs, MaxCut SDP relaxation, and the Lovász capacity problem. For completeness, we present these reformulations and choose a scheme (different from that in [15]) here that is most suited for computing solutions of (6)–(7).

Herein, we are concerned with those instances of \((P)\) for which \(p(x) - \gamma^*\) is SOS. Hence we presume that (6) is feasible, in which case strong duality holds in (6) and (7). We then write the optimality conditions for the primal–dual pair as follows:

\[
A_\alpha \bullet X = p_\alpha, \quad \text{for all } \alpha \in \mathbb{N}_{2m}^n, \quad |\alpha| > 0,
\]
\[
\sum_{\alpha \in \mathbb{N}_{2m}^n, |\alpha| > 0} y_\alpha A_\alpha + S = A_0,
\]
\[
A_0 \bullet X = p^T y,
\]
\[
X, S \in \mathbb{S}^N_+, \quad y \in \mathbb{R}^{M-1}.
\]

Note that any solution of (8)–(11) is an optimal primal–dual solution of (6) and (7), and vice versa. Define the finite-dimensional space \( U := \mathbb{S}^N \times \mathbb{S}^N \times \mathbb{R}^{M-1} \). For \( u = (X, S, y), \hat{u} = (\hat{X}, \hat{S}, \hat{y}) \in U \), define the inner product \( u \bullet \hat{u} := X \bullet \hat{X} + S \bullet \hat{S} + y^T \hat{y} \). The solution set of equations (8)–(10) define an affine subspace \( \mathcal{L} \subset U \), and the inclusion (11) defines a convex cone \( \mathcal{K} := \mathbb{S}^N_+ \times \mathbb{S}^N_+ \times \mathbb{R}^{M-1} \subset U \). We represent \( \mathcal{L} \) in a more compact form by introducing a linear operator \( \mathcal{A} : \mathbb{S}^N \to \mathbb{R}^{M-1} \) and its adjoint \( \mathcal{A}^* : \mathbb{R}^{M-1} \to \mathbb{S}^N \) defined as follows:

\[
z = \mathcal{A}(X) \quad \text{where } z_\alpha := A_\alpha \bullet X \quad \text{for all } \alpha \in \mathbb{N}_{2m}^n, |\alpha| > 0,
\]
\[
v = \mathcal{A}^*(y) \quad \text{where } v := \sum_{\alpha \in \mathbb{N}_{2m}^n, |\alpha| > 0} y_\alpha A_\alpha.
\]

Here the matrices \( A_\alpha \) are the basic moment matrices defined in Section 1.1. Similarly, consider another linear operator \( \mathcal{A}_0 : \mathbb{S}^N \to \mathbb{R} \) defined as \( \mathcal{A}_0(X) := A_0 \bullet X \) for all \( X \in \mathbb{S}^N \). We also denote \( \mathcal{I}_1 : \mathbb{S}^N \to \mathbb{S}^N \) as the identity operator. Using all of this notation, the subspace \( \mathcal{L} \) defined above as the solution of the equations (8)–(10) can be conveniently represented as \( \mathcal{L} := \{ u \in U : \mathcal{E} u = q \} \)
where

\[ E = \begin{bmatrix} A_0 & 0 & 0 \\ 0 & I & A^* \\ A_0 & 0 & -p^T \end{bmatrix}, \quad u = \begin{bmatrix} X \\ S \\ y \end{bmatrix}, \quad q = \begin{bmatrix} p \\ A_0 \end{bmatrix}. \quad (14) \]

Note of course that solving (8)–(11) (and hence solving (6) and (7)) is equivalent to finding a point \( u \in \mathcal{L} \cap \mathcal{K} \), for which any of the following reformulations are possible candidates for computing a solution via an AFOM:

\[
\begin{align*}
&\min \{ f_1(u) := \|E u - q\|^2_2 : u \in \mathcal{K} \}, \\
&\min \{ f_2(u) := d_\mathcal{L}^2(u) + d_\mathcal{K}^2(u) : u \in \mathcal{U} \}, \\
&\min \{ f_3(u) := d_\mathcal{L}^2(u) : u \in \mathcal{K} \}, \\
&\text{and/or} \quad \min \{ f_4(u) := d_\mathcal{K}^2(u) : u \in \mathcal{L} \}.
\end{align*}
\]

Here \( d_\mathcal{L} \) and \( d_\mathcal{K} \) are distance functions as defined in Section 1.1. Formulation (15) was first proposed in [12], (16) was studied in [11], and (17) and (18) were proposed in [15]. Scheme (18) is the most efficient from the vantage point of worst-case complexity analysis, especially when \( \|E\| \) is large, see [15]. However, when applied to the class of polynomial optimization problems under consideration herein, we have observed that (17) converges faster than (18), as we look at a converging sequence different from the one in the theoretical analysis (see Section 3.4 for more discussion). We therefore have chosen scheme (17) for implementation and extensive computational experiments, which we report in Section 3 with further discussion.

### 2.3 An accelerated first-order method

We wish to solve problem (17), namely:

\[
\begin{align*}
&\min \ f(u) := d_\mathcal{L}^2(u) \\
&\text{s.t.} \quad u \in \mathcal{K},
\end{align*}
\]

which, under the presumption that \( p(x) - y^* \) is SOS, is an equivalent reformulation of (8)–(11) and hence of (6)–(7). Here the objective function is the squared distance function of a point to the affine subspace \( \mathcal{L} \). As introduced in Section 1.1, \( d_\mathcal{L}^2(u) \) has Lipschitz continuous derivative with Lipschitz constant \( L = 2 \). We use the following AFOM, which was first presented and analysed in [1] in a more general setting, but is stated here as presented for the specific setting of solving (19) treated in [15]:

<table>
<thead>
<tr>
<th>Accelerated first-order method (AFOM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0) Choose ( \tilde{u}_0 = \bar{u}_0 \in \mathcal{K}, k \leftarrow 0 ).</td>
</tr>
<tr>
<td>(1) Compute ( u_k := \frac{2}{k+2} \tilde{u}_k + \frac{k}{k+2} \bar{u}_k ).</td>
</tr>
<tr>
<td>(2) Compute ( (\tilde{u}<em>{k+1}, \bar{u}</em>{k+1}) \in \mathcal{K} \times \mathcal{K} ) as follows:</td>
</tr>
<tr>
<td>( \tilde{u}<em>{k+1} := \Pi</em>\mathcal{K} (\bar{u}<em>k - \frac{k+2}{2} (u_k - \Pi</em>\mathcal{L}(u_k))) ),</td>
</tr>
<tr>
<td>( \bar{u}<em>{k+1} := \frac{2}{k+2} \tilde{u}</em>{k+1} + \frac{k}{k+2} \bar{u}_k ).</td>
</tr>
<tr>
<td>(3) ( k \leftarrow k + 1 ), and go to (1).</td>
</tr>
</tbody>
</table>

Note that each iteration of AFOM requires the computation of two projections: \( \Pi_\mathcal{L}(u) \) which projects \( u = (X, S, y) \) onto the affine subspace \( \mathcal{L} \), and \( \Pi_\mathcal{K}(u) \) which projects \( u \) onto the cone \( \mathcal{K} := S_+^N \times S_+^N \times \mathbb{R}^{M-1} \). Computing \( \Pi_\mathcal{K}(u) \) requires computing an eigenvalue decomposition of
the two symmetric matrices \(X\) and \(S\); fortunately existing numerical packages are quite efficient for this task, see, for example, the \(eig\) function in Matlab, which works very effectively for matrices with several thousand rows/columns. Computing the projection \(\Pi_L(u)\) is typically quite costly, but as we show herein in Section 2.4, the special structure of the unconstrained polynomial optimization problem can be exploited to substantially speed up the computation of this projection.

Algorithm AFOM has the following convergence guarantee, originally proved in [1] but stated below from [15] in the context of problem (19):

**Theorem 1** [1,15] Let \(\{\tilde{u}_k\}\) be the sequence generated by algorithm AFOM applied to (19) and let \(u^*\) be any optimal solution of (19). For any given \(\varepsilon > 0\), \(d_L(\tilde{u}_k) \leq \varepsilon\) can be obtained in at most \(\lceil 2\|u^* - \tilde{u}_0\|/\varepsilon \rceil\) iterations.

### 2.4 Special structure of \((P)\) and improved efficiency of projection onto \(L\)

In this subsection, we identify and then exploit special structure of the SDP formulation of the unconstrained polynomial optimization problem, namely the orthogonality property of the basic moment matrices. This special structure enables a significant speed-up of the projection \(\Pi_L\) used in the AFOM. We demonstrate easily computable formulas for \(\Pi_L\) and we demonstrate that this computation uses \(O(N^2)\) operations, which represents a speed-up of \(O(M^2)\) operations over the traditional cost (namely \(O(M^2N^2)\)) for computing the projection.

We start with a remark on the structure of the basic moment matrices, from which the orthogonality property follows.

**Remark 1** Recall the definition of the basic moment matrix \(A_{\alpha}:\) for all \(\delta, \nu \in \mathbb{N}_m^n\), the \((\delta, \nu)\)-entry of \(A_{\alpha}\) is

\[
A_{\alpha}(\delta, \nu) = \begin{cases} 
1 & \text{if } \delta + \nu = \alpha, \\
0 & \text{otherwise.}
\end{cases}
\]

Let \(S_{\alpha}\) denote the support of \(A_{\alpha}\), namely \(S_{\alpha} := \{(\delta, \nu) \in \mathbb{N}_m^n \times \mathbb{N}_m^n | A_{\alpha}(\delta, \nu) \neq 0\}\). Then it follows that \(S_{\alpha} \cap S_{\beta} = \emptyset\) for \(\alpha \neq \beta\).

**Lemma 1** For all \(\alpha \in \mathbb{N}_m^2\), the basic moment matrices \(A_{\alpha}\) are orthogonal to one another. The linear operator \(D := A\mathcal{A}^*: \mathbb{R}^{M-1} \rightarrow \mathbb{R}^{M-1}\) is diagonal, whose \(\alpha\)th diagonal component is \(D_{\alpha} = A_{\alpha} \bullet A_{\alpha}\) and satisfies \(1 \leq D_{\alpha} \leq N\) for all \(\alpha \in \mathbb{N}_m^2\) satisfying \(|\alpha| > 0\). Therefore

1. \(A(A_0) = 0\), equivalently \(A\mathcal{A}_0 = 0\),
2. \((A\mathcal{A})^{-1} = D^{-1}\)
3. \((\mathcal{I}_1 + A^*A)^{-1} = \mathcal{I}_1 - A^*(\mathcal{I}_2 + D)^{-1}A\),
4. \(A(\mathcal{I}_1 + A^*A)^{-1} = (\mathcal{I}_2 + D)^{-1}A\), and \(A(\mathcal{I}_1 + A^*A)^{-1}(A_0) = 0\),
5. \((\mathcal{I}_1 + A^*A)^{-1}A^* = A^*(\mathcal{I}_2 + D)^{-1}\),
6. \(A(\mathcal{I}_1 + A^*A)^{-1}A^* = D(\mathcal{I}_2 + D)^{-1}\) and
7. \(\mathcal{I}_2 - A(\mathcal{I}_1 + A^*A)^{-1}A^* = (\mathcal{I}_2 + D)^{-1}\),

where the identity operators are defined as \(\mathcal{I}_1 : \mathbb{S}^N \rightarrow \mathbb{S}^N\) and \(\mathcal{I}_2 : \mathbb{R}^{M-1} \rightarrow \mathbb{R}^{M-1}\).
Proof. It follows from Remark 1 that the basic moment matrices are orthogonal to each other, i.e., for all \( \alpha, \beta \in \mathbb{N}^{n}_{2m} \),

\[
A_\alpha \bullet A_\beta = \begin{cases} 
0, & \alpha \neq \beta, \\
n_\alpha, & \alpha = \beta,
\end{cases}
\tag{20}
\]

where \( n_\alpha \geq 1 \) is the number of 1's in the matrix \( A_\alpha \). Let \( z := A \cdot A^* (y) \), then the \( \alpha \)-entry of \( z \) is given by:

\[
z_\alpha = \sum_\nu y_\nu A_\nu \bullet A_\alpha = n_\alpha y_\alpha.
\tag{21}
\]

Therefore \( D := A \cdot A^* \) is a diagonal operator, in which the \( \alpha \)-entry on the diagonal is \( D_\alpha = n_\alpha \).

Since \( n_\alpha = A_\alpha \bullet A_\alpha \), it follows that \( n_\alpha \) is equal to the number of different ways that \( \alpha \) can be decomposed as \( \alpha = \delta + \nu \) for \( \delta, \nu \in \mathbb{N}^{n} m \). Since there are in total \( N \) different \( \delta \)'s in \( \mathbb{N}^{n} m \), it also follows that \( 1 \leq n_\alpha \leq N \).

Item (1) follows from the orthogonality of the basic moment matrices, and (2) follows since \( D_\alpha = n_\alpha \geq 1 \) for all \( \alpha \in \mathbb{N}^{n}_{2m} \). Item (3) follows from the Sherman–Morrison–Woodbury identity:

\[
(I_1 + A^* A)^{-1} = I_1 - A^*(I_2 + AA^*)^{-1} A
\tag{22}
\]

and then substituting in (2). To prove the first part of (4) we premultiply (3) by \( A \) and derive:

\[
A(I_1 + A^* A)^{-1} = A(I_1 - A^*(I_2 + D)^{-1} A) = A - D(I_2 + D)^{-1} A = (I_2 + D)^{-1} A,
\]

and the second part of (4) follows the first part using \( A A_0^* = 0 \) from (1). Item (5) is simply the adjoint of the first identity of (4). Item (6) follows from (5) by premultiplying (5) by \( A \), and (7) follows from (6) by rearranging and simplifying terms.

Lemma 1 allows improved computation of the projection \( \Pi_\mathcal{L} \), as we now show.

**Lemma 2.** Let \( u = (X, S, y) \in \mathbb{U} \) be given, and let \( (\bar{X}, \bar{S}, \bar{y}) := \Pi_\mathcal{L}(u) \). Then the following is a valid method for computing \( (\bar{X}, \bar{S}, \bar{y}) \). First compute the scalars:

\[
\bar{\xi} \leftarrow 1 + p^T (I_2 + D)^{-1} p,
\tag{23}
\]

\[
\bar{\eta} \leftarrow A_0 \bullet X - p^T (I_2 + D)^{-1} y + p^T (I_2 + D)^{-1} A(S),
\tag{24}
\]

\[
\bar{r} \leftarrow \frac{\bar{\eta}}{\bar{\xi}}.
\tag{25}
\]

Then compute \( \Pi_\mathcal{L}(u) \) as follows:

\[
\bar{X} = X - A^* D^{-1} (A(X) - p) - \bar{r} A_0,
\tag{26}
\]

\[
\bar{S} = A^* (I_2 + D)^{-1} (A(S) - y - \bar{r} p) + A_0,
\tag{27}
\]

\[
\bar{y} = (I_2 + D)^{-1} (-A(S) + y + \bar{r} p).
\tag{28}
\]
Recalling the definition of $\mathcal{E}$ in (14), we note that $\tilde{u} = (\tilde{X}, \tilde{S}, \tilde{y})$ along with Lagrange multipliers $\lambda = (\tilde{w}, \tilde{V}, \tilde{r}) \in \mathbb{R}^{m-1} \times \mathbb{S}^n \times \mathbb{R}$ are a solution of the following normal equations:

$$\mathcal{E}(\tilde{u}) = q \quad \text{and} \quad u - \tilde{u} = \mathcal{E}^*\lambda.$$  

(29)

Indeed, it is sufficient to determine $\tilde{\lambda}$ that solves:

$$\mathcal{E}\mathcal{E}^*\tilde{\lambda} = \mathcal{E}u - q,$$

(30)

and then to assign

$$\tilde{u} = u - \mathcal{E}^*\tilde{\lambda}.$$  

(31)

Using (14) and the results in Lemma 1, we obtain:

$$\mathcal{E}\mathcal{E}^* = \begin{bmatrix} AA^* & 0 & AA_0^* \\ 0 & I_1 + A^*A & -A^*(p) \\ A_0A^* & -p^TA & A_0A_0^* + p^TPp \end{bmatrix} = \begin{bmatrix} AA^* & 0 & 0 \\ 0 & I_1 + A^*A & -A^*(p) \\ 0 & -p^TA & 1 + p^TPp \end{bmatrix},$$

(32)

therefore the system (30) can be written as

$$\begin{bmatrix} AA^* & 0 & 0 \\ 0 & I_1 + A^*A & -A^*(p) \\ 0 & -p^TA & 1 + p^TPp \end{bmatrix} \begin{bmatrix} \tilde{w} \\ \tilde{V} \\ \tilde{r} \end{bmatrix} = \begin{bmatrix} A(X) - p \\ S + A^*(y) - A_0 \\ A_0 \bullet X - p^T y \end{bmatrix}. $$

(33)

It is straightforward to verify that the following computations involving auxiliary scalars $\tilde{\xi}$ and $\tilde{\eta}$ yield a solution to (33):

$$\tilde{w} \leftarrow (AA^*)^{-1}(A(X) - p),$$

(34)

$$\tilde{\xi} \leftarrow 1 + p^Tp - p^TA(I_1 + A^*A)^{-1}A^*(p),$$

(35)

$$\tilde{\eta} \leftarrow A_0 \bullet X - p^Ty + p^TA(I_1 + A^*A)^{-1}(S + A^*(y) - A_0),$$

(36)

$$\tilde{r} \leftarrow \frac{\tilde{\eta}}{\tilde{\xi}},$$

(37)

$$\tilde{V} \leftarrow (I_1 + A^*A)^{-1}(S + A^*y - A_0 + A^*(p)\tilde{r}).$$

(38)

(Indeed, (34) immediately implies the first equation of (33), and (38) immediately implies the second equation of (33). To verify the third equation of (33), observe that $-p^TA\tilde{V} + (1 + p^Tp)\tilde{r} = -p^TA(I_1 + A^*A)^{-1}(S + A^*y - A_0 + A^*(p)\tilde{r}) + (1 + p^TPp)\tilde{r} = -p^TA(I_1 + A^*A)^{-1}(S + A^*y - A_0) + (1 + p^TPp - p^TA(I_1 + A^*A)^{-1}A^*(p))\tilde{r} = -p^TA(I_1 + A^*A)^{-1}(S + A^*y - A_0) + \tilde{r}\tilde{\xi} = -p^TA(I_1 + A^*A)^{-1}(S + A^*y - A_0) + \tilde{\eta} = A_0 \bullet X - p^Ty.$) We then use (31) to compute $\tilde{u} = (\tilde{X}, \tilde{S}, \tilde{y})$ as follows:

$$\tilde{X} \leftarrow X - A^*\tilde{w} - \tilde{r}A_0 = X - A^*(AA^*)^{-1}(A(X) - p) - \tilde{r}A_0,$$

(39)

$$\tilde{S} \leftarrow S - \tilde{V} = S - (I_1 + A^*A)^{-1}(S - A_0 + A^*(y + \tilde{r}p)),$$

(40)

$$\tilde{y} \leftarrow y - A\tilde{V} + \tilde{r}p = y + \tilde{r}p - A(I_1 + A^*A)^{-1}(S + A^*(y + \tilde{r}p) - A_0).$$

(41)

Summarizing, we have shown that (34)–(41) solve the normal equation (29) and hence (39)–(41) are valid formulas for the projection $\Pi_C(u)$. We now show that these formulas can alternatively
be computed using (23–(28). To derive (23), we have from (35) and (7) of Lemma 1:

\[ \bar{\xi} = 1 + p^T [I_2 - A(I_1 + A^*A)^{-1}A^*] p = 1 + p^T (I_2 + \mathcal{D})^{-1} p. \]

To derive (24), we have from (36) and (7) and (4) of Lemma 1:

\[ \bar{\eta} = A_0 \cdot X - p^T [I_2 - A(I_1 + A^*A)^{-1}A^*] y + p^T A(I_1 + A^*A)^{-1}(S - A_0) \]
\[ = A_0 \cdot X - p^T (I_2 + \mathcal{D})^{-1} y + p^T (I_2 + \mathcal{D})^{-1} A(S). \]

Equations (25) and (37) are identical, and to derive (26), we have from (39) and (2) of Lemma 1:

\[ \bar{X} = X - A^*\mathcal{D}^{-1}(A(X) - p) - \bar{\eta} A_0. \]

To derive (27), we have from (40) and (5), (3), and (1) of Lemma 1:

\[ \bar{S} = S - (I_1 + A^*A)^{-1}(S - A_0 + A^*(y + \bar{\eta}p)) \]
\[ = [I_1 - (I_1 + A^*A)^{-1}](S) - (I_1 + A^*A)^{-1}A^*(y + \bar{\eta}p) + (I_1 + A^*A)^{-1}(A_0) \]
\[ = A^*(I_2 + \mathcal{D})^{-1} A(S) - A^*(I_2 + \mathcal{D})^{-1}(y + \bar{\eta}p) + A_0. \]

To derive (28), we have from (41) and (7) and (4) of Lemma 1:

\[ \bar{y} = [I_2 - A(I_1 + A^*A)^{-1}A^*](y + \bar{\eta}p) - A(I_1 + A^*A)^{-1}(S) + A(I_1 + A^*A)^{-1}(A_0) \]
\[ = (I_2 + \mathcal{D})^{-1}(y + \bar{\eta}p) - (I_2 + \mathcal{D})^{-1} A(S). \]

Remark 2 The above computations are numerically well-behaved. To see this, note that there are no explicit matrix solves, so the issue of solving simultaneous equations is avoided. Second, note that divisors involve either \( \bar{\xi} \) or the (diagonal) components \( \mathcal{D}_a \) of \( \mathcal{D} \). From (23) we see that \( \bar{\xi} \) is bounded from above and below as

\[ 1 \leq \bar{\xi} \leq 1 + \frac{1}{2} \|p\|^2, \tag{42} \]

whereby division by \( \bar{\xi} \) is well-behaved relative to the data \( p \). It also follows from Lemma 1 that

\[ 1 \leq \mathcal{D}_a \leq N, \tag{43} \]

whereby working with \( \mathcal{D}^{-1} \) and/or \( (I_2 + \mathcal{D})^{-1} \) are also well-behaved relative to the dimension \( N \) of the SDP.

The computational cost of computing \( \Pi_\mathcal{L}(u) \) using Lemma 2 is bounded from above using the following lemma.

**Lemma 3** The computation of \( \Pi_\mathcal{L}(u) \) requires \( O(N^2) \) arithmetic operations.

**Proof** From Lemma 2, the computation of \( \Pi_\mathcal{L}(u) \) involves computations with \( A(\cdot), A^*(\cdot), \mathcal{D}^{-1}(\cdot), (I_2 + \mathcal{D})^{-1}(\cdot) \), matrix additions, and vector inner products. These last two involve \( O(N^2) \) and \( O(M) \) arithmetic operations, respectively. From Remark 1, we observe that \( A(X) \) adds the entries of \( X \) and accesses each entry exactly once, so is \( O(N^2) \) additions. Similarly, \( A^*(y) \) forms an \( N \times N \) matrix by assigning the entries of \( y \) to specific locations, and involves \( O(N^2) \) operations. Since \( \mathcal{D} \) is diagonal and has \( M \) positive diagonal elements, \( \mathcal{D}^{-1}(\cdot) \) and \( (I_2 + \mathcal{D})^{-1}(\cdot) \) involve \( M \) multiplications. Finally, since \( N^2 \geq M \) (this can be seen from the fact that \( A^*(y) \) maps exclusive
components of \( y \) into entries of an \( N \times N \) matrix), the computation of \( \Pi_L(u) \) requires \( O(N^2) \) arithmetic operations.

Let us now contrast the above bound with the more general case. Consider a general SDP problem with primal-and-dual decision variables in \( \mathbb{S}^N \times \mathbb{S}^N \times \mathbb{R}^{M-1} \). From (30), computing \( \Pi_L \) involves (at a minimum) solving systems of equations with the operator \( \mathcal{A} \mathcal{A}^* \), which requires \( O(M^2N^2) \) operations to form and \( O(M^3) \) operations to solve, so the total computational effort is \( O(M^3 + M^2N^2) = O(M^2N^2) \) operations. Here we see that by identifying and exploiting the structure of the SDP for unconstrained polynomial optimization, we reduce the computational cost of computing \( \Pi_L(u) \) by a factor of \( O(M^2) \). This saving is substantial especially for high degree polynomials, since \( M = \binom{n+d}{n} = O(n^d) \) grows exponentially in the degree \( d \) of the polynomial.

3. Computational results and discussion

In this section, we present computational results for the AFOM and compare these results to those of the SeDuMi sovsolve IPM on solving SOS relaxations of unconstrained polynomial optimization problems of degree \( d = 4 \) and 6. The AFOM is implemented in Matlab R2008, and the SeDuMi package is version 1.1. The computational platform is a laptop with an Intel Core(TM) 2Duo 2.50 GHz CPU and 3GB memory.

3.1 Generating instances of polynomials

We generated instances of polynomials \( p(x) \) in \( n \) variables and of degree \( d = 2m \) for \( m = 2 \) and 3. Each instance was generated randomly as follows. We first chose a random point \( x^* \) uniformly distributed in \([-1, 1]^n\), and then generated the coefficients of \( k \) polynomials \( q_i(x) \) in \( n \) variables and degree \( m \) for \( i = 1, 2, \ldots, k \), using a uniform distribution on \([0, 3]\). We then set \( y^* := -\sum_{i=1}^k q_i(x^*)^2 \) and \( p(x) := \sum_{i=1}^k (q_i(x) - q_i(x^*))^2 + y^* \). The resulting polynomial \( p(x) \) satisfies \( y^* = \min_{x \in \mathbb{R}^n} p(x) \) and \( p(x) - y^* \) is an SOS polynomial, and furthermore \( p(x) \) has zero constant term since \( p(0) = 0 \). We set \( k = 4 \) for all instances. The resulting instances have fully dense coefficient vectors \( p \in \mathbb{R}^{M-1} \). When generating the 6-degree polynomials, we wanted the polynomials to have a density of coefficients of 10% on average. This was accomplished by suitably controlling the density of the coefficients of the polynomials \( q_i(x) \), for \( i = 1, 2, \ldots, k \).

3.2 Algorithm stopping rules

Both AFOM and the SeDuMi IPM seek to solve the primal/dual SDP instance pair (6)–(7). We use SeDuMi’s stopping rule, see [26], which we briefly review herein. Using the notation of conic convex optimization, we re-write the primal and dual (6)–(7) in the standard linear conic optimization format:

\[
\begin{align*}
\text{(P)} & \quad \min_{x} \ c^T x \quad \text{s.t.} \ A x = b, \quad x \in C, \\
\text{(D)} & \quad \max_{y,z} \ b^T y \quad \text{s.t.} \ A^* y + z = c, \quad z \in C^*.
\end{align*}
\]

with correspondence \( b \equiv p, A x \equiv A(X), c^T x \equiv A_0 \bullet X \), and \( C, C^* \equiv \mathbb{S}_+^N \). SeDuMi solves a homogeneous self-dual (HSD) embedding model until the primal infeasibility, dual infeasibility and the duality gap of the current trial iterate \((\hat{x}, \hat{y}, \hat{z})\) (which automatically satisfies the cone inclusions
\( \hat{x} \in C, \hat{z} \in C^* \) are small. More specifically, define:

\[ r_p := b - A\hat{x} \quad \text{(primal infeasibility gap)}, \]
\[ r_d := A^T\hat{y} + \hat{z} - c \quad \text{(dual infeasibility gap)}, \]
\[ r_k := c^T\hat{x} - b^T\hat{y} \quad \text{(duality gap)}. \]

SeDuMi’s stopping criterion is:

\[
2 \frac{\|r_p\|_{\infty}}{1 + \|b\|_{\infty}} + 2 \frac{\|r_d\|_{\infty}}{1 + \|c\|_{\infty}} + \frac{(r_k)^+}{\max(|c^T\hat{x}|, |b^T\hat{y}|)} \leq \varepsilon.
\]

Here \( \varepsilon \) is the specified tolerance. The denominator of the third term above depends on the value \( \hat{\tau} \) of a scaling variable \( \tau \) in the HSD model, see [26]. Since the AFOM does not have any such scaling variable, we modify the above stopping rule when applied to AFOM to be:

\[
2 \frac{\|r_p\|_{\infty}}{1 + \|b\|_{\infty}} + 2 \frac{\|r_d\|_{\infty}}{1 + \|c\|_{\infty}} + \frac{(r_k)^+}{\max(|c^T\hat{x}|, |b^T\hat{y}|)} \leq \varepsilon.
\]

Note that the stopping criterion for AFOM is (marginally) more conservative than that of IPM due to the absence of the component involving \( \hat{\tau} \) in the denominator of the third term above. The stopping rule tolerance \( \varepsilon \) is set to \( 10^{-4} \) for all of our computational experiments. Returning to the notation of the SDP primal and dual problems (6)–(7) and letting \( u = (X, S, y) \in K \) be a current iterate of either AFOM or IPM, then our output of interest is objective gap \( := |p^T\hat{y} + \gamma^*| \), which measures the objective function value gap between the current dual (near-feasible) solution, namely \( \langle -p^Ty \rangle \), and the true global optimal value of the polynomial \( \gamma^* \). For AFOM, according to Theorem 1, the iterate sequence \( \bar{u}_k \) satisfies the convergence bound, which suggests we should compute Objective Gap using \( |p^T\bar{y}_k + \gamma^*| \). However, we have observed that \( d_C(\bar{u}_k) \) also consistently obeys the theoretical convergence rate of Theorem 1, and that both \( d_C(\bar{u}_k) \) and \( |p^T\bar{y}_k + \gamma^*| \) converge to zero much faster than does \( d_C(\bar{u}_k) \) and \( |p^T\bar{y}_k + \gamma^*| \). For this reason we compute Objective Gap using \( |p^T\bar{y}_k + \gamma^*| \) and report our results using the \( \bar{u}_k = (\bar{X}_k, \bar{S}_k, \bar{y}_k) \) sequence.

### 3.3 Computational results

Our computational results for 4-degree and 6-degree polynomials are presented in Tables 1 and 2, respectively. In each of these tables, the first column displays the dimension \( n \) of the optimization variable \( x \), followed by the size dimensions \( N \) and \( M \) for the SDP instances (6)–(7). The fourth column shows the number of sample instances that were generated for each value of \( n \), where out of necessity we limited the number of sample instances when the dimension \( n \) of the problems (and hence the dimensions \( N \) and \( M \) of the SDP problems (6)–(7)) were larger.

For small size problems, IPM runs faster than AFOM. However, AFOM computation time dominates IPM computation time for \( n \geq 12 \) and \( n \geq 8 \) for 4-degree polynomials and 6-degree polynomials, respectively. More importantly, since AFOM has much lower memory requirements, AFOM is able to solve problems in much larger dimension \( n \) than IPM (about twice as large for both 4-degree and 6-degree polynomials). This extends the practical solvability of the SDP representation of the polynomial optimization problems. For 4-degree polynomials, IPM produces slightly better objective gap values, but no such difference is discernible for 6-degree polynomials. Figures 1 and 2 illustrate the computation time results in a graphical form for 4-degree and 6-degree polynomials, respectively.
Table 1. Computational results for degree-4 polynomials.

<table>
<thead>
<tr>
<th>n</th>
<th>N</th>
<th>M</th>
<th>s</th>
<th>SDP Size</th>
<th>Time (seconds)</th>
<th>Objective Gap</th>
<th>Iterations</th>
<th>Time (seconds)</th>
<th>Objective Gap</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Median</td>
<td>Average</td>
<td>Median</td>
<td>Median</td>
<td>Average</td>
<td>Median</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.96 × 10⁻¹</td>
<td>3.9 × 10⁻¹</td>
<td>1.77 × 10⁻³</td>
<td>1.62 × 10⁻³</td>
<td>820.0</td>
<td>1068.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.04</td>
<td>1.66</td>
<td>2.57 × 10⁻³</td>
<td>2.17 × 10⁻³</td>
<td>2113.5</td>
<td>3276.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.35</td>
<td>7.29</td>
<td>4.93 × 10⁻³</td>
<td>5.15 × 10⁻³</td>
<td>8416.0</td>
<td>9599.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>52.69</td>
<td>36.92</td>
<td>9.10 × 10⁻³</td>
<td>1.36 × 10⁻³</td>
<td>19955.5</td>
<td>13661.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.09</td>
<td>30.22</td>
<td>1.36 × 10⁻²</td>
<td>8.10 × 10⁻³</td>
<td>2432.5</td>
<td>6662.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.31</td>
<td>28.06</td>
<td>8.12 × 10⁻³</td>
<td>1.18 × 10⁻²</td>
<td>1624.0</td>
<td>3974.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12.56</td>
<td>23.42</td>
<td>3.09 × 10⁻³</td>
<td>8.19 × 10⁻³</td>
<td>1223.5</td>
<td>2311.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14.56</td>
<td>23.22</td>
<td>1.41 × 10⁻³</td>
<td>4.13 × 10⁻³</td>
<td>952.0</td>
<td>1151.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21.42</td>
<td>27.61</td>
<td>1.42 × 10⁻³</td>
<td>2.93 × 10⁻³</td>
<td>970.0</td>
<td>1282.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33.24</td>
<td>40.55</td>
<td>1.74 × 10⁻³</td>
<td>3.05 × 10⁻³</td>
<td>1139.5</td>
<td>1401.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>55.17</td>
<td>60.18</td>
<td>1.76 × 10⁻³</td>
<td>2.76 × 10⁻³</td>
<td>1240.0</td>
<td>1389.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82.70</td>
<td>91.12</td>
<td>1.49 × 10⁻³</td>
<td>2.37 × 10⁻³</td>
<td>1398.0</td>
<td>1530.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128.08</td>
<td>141.19</td>
<td>3.90 × 10⁻³</td>
<td>2.83 × 10⁻³</td>
<td>1609.0</td>
<td>1781.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>168.99</td>
<td>184.77</td>
<td>3.66 × 10⁻³</td>
<td>4.63 × 10⁻³</td>
<td>1693.5</td>
<td>2225.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>263.76</td>
<td>312.94</td>
<td>3.09 × 10⁻³</td>
<td>6.09 × 10⁻³</td>
<td>1963.0</td>
<td>2325.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>385.48</td>
<td>414.21</td>
<td>3.03 × 10⁻³</td>
<td>5.70 × 10⁻³</td>
<td>2170.5</td>
<td>2357.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>586.72</td>
<td>664.94</td>
<td>4.35 × 10⁻³</td>
<td>6.48 × 10⁻³</td>
<td>2521.0</td>
<td>2912.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>833.10</td>
<td>923.29</td>
<td>5.45 × 10⁻³</td>
<td>9.45 × 10⁻³</td>
<td>2741.5</td>
<td>3002.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1083.90</td>
<td>1125.41</td>
<td>4.60 × 10⁻³</td>
<td>9.93 × 10⁻³</td>
<td>3011.0</td>
<td>3097.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4682.34</td>
<td>5271.75</td>
<td>8.32 × 10⁻³</td>
<td>2.13 × 10⁻²</td>
<td>4595.0</td>
<td>5116.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21323.53</td>
<td>21837.43</td>
<td>3.96 × 10⁻⁵</td>
<td>5.28 × 10⁻⁵</td>
<td>6466.7</td>
<td>6351.5</td>
</tr>
</tbody>
</table>

Notes: $n$ is the dimension of the variable $x$, and $s$ is the number of sample instances. The stopping rule tolerance is $\varepsilon = 10^{-4}$ for both AFOM and IPM. For $n > 15$, IPM fails due to memory requirements.
**Table 2. Computational results for degree-6 polynomials.** \( n \) is the dimension of the variable \( x \), and \( s \) is the number of sample instances.

| \( n \) | \( N \) | \( M \) | \( s \) | AFOM Performance | | | | IPM Performance | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Median | Average | Median | Average | Median | Average |
| | | | | Time (seconds) | Objective Gap | Iterations | Time (seconds) | Objective Gap | Iterations |
| 2 | 10 | 28 | 100 | 0.47 | 2.00 | 3.94 \( \times 10^{-5} \) | 1.14 \( \times 10^{-4} \) | 753.0 | 3398.9 |
| 3 | 20 | 84 | 100 | 5.09 | 8.63 | 8.50 \( \times 10^{-5} \) | 1.72 \( \times 10^{-4} \) | 2384.5 | 4053.3 |
| 4 | 35 | 210 | 100 | 29.23 | 37.74 | 5.41 \( \times 10^{-4} \) | 6.24 \( \times 10^{-4} \) | 5227.5 | 6719.8 |
| 5 | 56 | 462 | 100 | 78.15 | 101.38 | 1.38 \( \times 10^{-4} \) | 1.35 \( \times 10^{-3} \) | 5924.5 | 7691.5 |
| 6 | 84 | 924 | 100 | 48.76 | 79.28 | 1.45 \( \times 10^{-3} \) | 1.36 \( \times 10^{-3} \) | 1529.0 | 2420.5 |
| 7 | 120 | 1716 | 100 | 48.23 | 67.14 | 9.0 \( \times 10^{-5} \) | 3.81 \( \times 10^{-4} \) | 429.5 | 506.7 |
| 8 | 165 | 3003 | 100 | 67.01 | 78.35 | 3.13 \( \times 10^{-4} \) | 10.19 | 101.38 | 10 |
| 9 | 220 | 5005 | 100 | 123.34 | 137.42 | 1.03 \( \times 10^{-4} \) | 3.725 | 427.6 |
| 10 | 286 | 8008 | 60 | 229.26 | 250.46 | 1.50 \( \times 10^{-4} \) | 3.96 \( \times 10^{-4} \) | 404.6 | 373.0 |
| 11 | 364 | 12376 | 50 | 471.12 | 478.16 | 1.38 \( \times 10^{-4} \) | 3.73 \( \times 10^{-4} \) | 439.5 | 448.3 |
| 12 | 455 | 18564 | 50 | 778.63 | 835.25 | 1.83 \( \times 10^{-4} \) | 6.49 \( \times 10^{-4} \) | 478.0 | 516.0 |
| 13 | 560 | 27132 | 10 | 1380.52 | 1404.66 | 9.83 \( \times 10^{-5} \) | 2.16 \( \times 10^{-4} \) | 521.5 | 538.7 |
| 14 | 680 | 38760 | 10 | 2944.96 | 3002.51 | 8.31 \( \times 10^{-5} \) | 1.01 \( \times 10^{-3} \) | 707.5 | 725.5 |
| 15 | 816 | 54264 | 10 | 5393.01 | 5754.98 | 5.64 \( \times 10^{-4} \) | 7.91 \( \times 10^{-4} \) | 799.5 | 859.2 |
| 16 | 969 | 74613 | 10 | 10041.98 | 10705.34 | 6.91 \( \times 10^{-4} \) | 1.40 \( \times 10^{-3} \) | 962.0 | 1017.3 |

Notes: The stopping rule tolerance is \( \varepsilon = 10^{-4} \) for both AFOM and IPM. For \( n > 9 \), IPM fails due to memory requirements.
Figure 1. Graph of the median and average running times (in seconds) of AFOM and IPM for degree-4 polynomials in \( n \) variables. The values for IPM end in the middle of the graph where IPM fails due to excessive memory requirements.

Figure 2. Graph of the median and average running times (in seconds) of AFOM and IPM for degree-6 polynomials in \( n \) variables. The values for IPM end in the middle of the graph where IPM fails due to excessive memory requirements.

One of the generic advantages of IPMs is the very low variability in the number of iterations, and hence in computational burden. By contrast, AFOM has higher variability in terms of both computation time and objective gap values. Such variability is illustrated in the histogram plots in Figure 3. The histograms in Figure 3(a) and (b) show AFOM computation time and objective gap values.
values for the \( s = 50 \) sample instances of 4-degree polynomials in dimension \( n = 19 \), respectively. The histograms in Figure 3(c) and (d) show AFOM computation time and objective gap values for the \( s = 50 \) sample instances of 6-degree polynomials in dimension \( n = 12 \), respectively. Observe that tails and outliers contribute to the higher variation in the performance AFOM in these two metrics.

### 3.4 A curious observation

In its simplest form, a traditional first-order (gradient) algorithm produces one sequence of points, where the next point is determined from the current point by taking a step in the direction of the negative of the gradient of the current point, according to some specified step-size rule. A crucial difference between the modern class of AFOMs and the traditional algorithm is that the AFOMs involve more than one sequence in the gradient descent scheme. Nesterov [20, p. 80] devised a very simple AFOM with two sequences. The method used in this paper is a variant of Nesterov’s original algorithm, and constructs three sequences – namely \( \{ u_k \} \), \( \{ \bar{u}_k \} \) and \( \{ \tilde{u}_k \} \). The convergence analysis establishes the \( O(1/\varepsilon) \) convergence rate of the objective function in terms of the sequence values of \( \{ \tilde{u}_k \} \). From our computational experiments, we observe that both \( \nabla p^T \bar{y}_k \) and \( d_L(\bar{u}_k) \) closely follow the theoretical rate \( O(1/\varepsilon) \).

For the unconstrained polynomial optimization problem, we are ultimately interested in the convergence of \(-p^T\tilde{y}_k\) to the global optimal value \( \gamma^* \) of the polynomial \( p(x) \), and of \( d_L(\tilde{u}_k) \) to zero. However, we have consistently observed in our experiments that both \(-p^T\tilde{y}_k\) and \( d_L(\tilde{u}_k) \) exhibit much faster convergence to their respective limits than does \(-p^T\bar{y}_k\) and \( d_L(\bar{u}_k) \), with the difference being more apparent for larger-scale problems (i.e. when \( n \) is larger). We illustrate this...
Figure 4. Objective gap values (left) and $d_L(\cdot)$ values (right) for a 4-degree polynomial instance for $n = 20$. In the left figure, the solid curve plots $|p^Ty_k^+ + y^*|$ and the dashed curve plots $|p^Ty_k + y^*|$. In the right figure, the solid curve plots $d_L(\bar{u}_k)$ and the dashed curve plots $d_L(\tilde{u}_k)$. As shown, $|p^Ty_k^+ + y^*|$ and $d_L(\bar{u}_k)$ converge to zero much faster than does $|p^Ty_k + y^*|$ and $d_L(\tilde{u}_k)$. For example, at iteration $k = 2200$, $|p^Ty_k^+ + y^*| = 6.03 \times 10^{-10}$ and $d_L(\bar{u}_k) = 3.63 \times 10^{-9}$, while $|p^Ty_k + y^*| = 42.93$ and $d_L(\tilde{u}_k) = 0.049$.

observation in Figure 4, which shows typical convergence behaviour of the two sequences for one of the problem instances of 4-degree polynomials for $n = 20$. We have also observed that when $p^Ty_k$ converges, it converges to a very high accuracy, which is also illustrated in Figure 4. We do not yet have either a satisfactory theoretical or practical explanation of this phenomenon.

4. Discussion

We have applied an AFOM to solve the SOS relaxations of unconstrained polynomial optimization problems. By exploiting special structural properties of the problem, namely the orthogonality of the moment matrices, the computational cost of the AFOM is greatly reduced. Numerical results on degree-4 and degree-6 polynomials demonstrate that the AFOM is capable of solving polynomials with at least twice as many variables as existing IPM software (SeDuMi) can handle.

We have observed some curious convergence behaviour of the iterate sequences of the AFOM that does not follow from the usual convergence analysis of such methods. This observation bears further study.

In the last several years, several researchers have worked on algorithms for solving large-scale SDPs, see, for example, [18,28]. In the recent work of Nie [21], these algorithms were applied to solve large-scale unconstrained polynomial optimization problems. The main algorithmic framework in these recent algorithms is the augmented Lagrangian method, where the linear system at each iteration is solved by a combination of a semismooth Newton method and a conjugate gradient algorithm. The computational experiments recently reported for applying these methods to polynomial optimization in [21] are very promising, for instance, they solve 4-degree polynomial optimization problems in up to $n = 100$ variables with high precision. These methods utilize second-order Hessian information but have low memory requirements inherited from the conjugate gradient algorithm. However, the augmented Lagrangian method in general does not have a global convergence rate guarantee; and indeed, the results in [25,28] show global convergence and local convergence rates only under regularity conditions. In comparison, the AFOM has virtually no regularity requirement and admits an explicit global convergence rate. The parameters in the
convergence rate can be derived or bounded a priori in some cases (for instance, if we know that the optimal solution is contained in a given ball). As a future research direction, it would be very interesting to merge these two types of algorithms, i.e. to use an AFOM in the initial phase to quickly compute solutions that are in a neighborhood of the optimal solution, and then switch over to a low-memory second-order method to compute highly accurate solution iterates thereafter.

Acknowledgements

The authors are grateful for the comments of an anonymous referee. This research has been supported through AFOSR Grant No. FA9550-08-1-0350, AFOSR Grant No. FA9550-11-1-0141 and the Singapore-MIT Alliance.

References


