High accuracy solution of large scale semidefinite programs

Thomas Davi, Florian Jarre
Institut für Mathematik, Universität Düsseldorf
Universitätsstraße 1, D–40225 Düsseldorf, Germany

Abstract

We present a first order approach for solving semidefinite programs. Goal of this approach is to compute a solution of the SDP up to high accuracy in spite of using only partial second order information. We propose a hybrid approach that uses an accelerated projection method to generate an approximate solution and then switches to the QMR algorithm applied to a symmetrized version of the AHO-system to improve this approximation. Some numerical experiments based on a number of random test examples illustrate the potential of this approach.

Keywords: Semidefinite program, AHO direction, QMR algorithm.

1 Introduction

We present a first order approach for solving large scale semidefinite programs. Efficient approaches for the numerical solution of such programs have recently been developed, see e.g. [4, 10]. Goal of the approach in this paper is to compute a solution of the SDP up to high accuracy in spite
of using only partial second order information. It is assumed that eigen-
value decompositions of the matrix variable are computable, i.e. that the
dimension of the matrix blocks is not too large, and that the linear equality
constraints defining the semidefinite program are sparse.

We propose a hybrid approach that starts with an accelerated projection
method proposed in [3] to generate an approximation to the optimal solution.
This method (called APD method in [3]) only uses first order search steps.
In a second stage the hybrid approach switches to the QMR method [2]
applied to a symmetrized version of the AHO system [1]. For the AHO-
QMR-approach, neither positive definiteness of the iterates nor a homotopy
parameter \( \mu > 0 \) are required; in fact, some form of partial complementarity
of the iterates is exploited in order to simplify the AHO-system. The switch
from the APD-method to AHO-QMR is controlled by monitoring the rate
of convergence of the APD iterates; whenever this rate drops below a preset
threshold, the switch is carried out. A safeguard controls the convergence
of the AHO-QMR search steps. We close with some numerical experiments
based on a number of random test examples.

1.1 The Problem and Notation

By \( S^n \) we denote the space of real symmetric \( n \times n \) matrices. The scalar
product of two real matrices \( P, Q \) (of same dimensions) is given by \( P \cdot Q = \text{trace}(P^T Q) \) and is associated with the Frobenius norm \( \|Q\|_F^2 = Q \cdot Q \). Given \( m \) matrices \( A^{(i)} \), \( 1 \leq i \leq m \) in \( S^n \), a linear operator \( A : S^n \to \mathbb{R}^m \)
is defined via

\[
A(X) = \begin{pmatrix}
A^{(1)} \cdot X \\
\vdots \\
A^{(m)} \cdot X
\end{pmatrix}.
\]

The adjoint operator \( A^* \) is then defined via \( A^*(y) = \sum_{i=1}^m y_i A^{(i)} \). We con-
sider a semidefinite program,

\[
(P) \quad \text{minimize } C \cdot X \mid X \succeq 0, \ A(X) = b
\]
with dual program

\[(D) \quad \text{maximize } b^T y \mid S \succeq 0, \quad A^*(y) + S = C.\]

Throughout this paper we assume that \((P)\) and \((D)\) satisfy Slaters condition and that \((P)\) and \((D)\) have a unique and strictly complementary solution \((X^{opt}, S^{opt})\). In particular, we assume that the matrices \(A^{(i)}\) are linearly independent for \(1 \leq i \leq m\).

By \(\Pi_{S^n_+}\) we denote the orthogonal projection of a matrix in \(S^n\) onto the positive semidefinite cone \(S^n_+\). For a given vector \(x \in \mathbb{R}^n\) and a matrix \(X \in S^n\) we denote the diagonal matrix with entries \(x_i\) (\(1 \leq i \leq n\)) by \(\text{Diag}(x)\), and the vector with entries \(X_{i,i}\) by \(\text{diag}(X)\). By \(P \circ Q\) we denote the Hadamard product (componentwise product) of two matrices \(P\) and \(Q\) of same dimensions. Furthermore, we set

\[
\text{svec} : S^n \to \mathbb{R}^{n(n+1)/2}, \quad \text{svec}(X) = ((\delta_{ij}(1 - \sqrt{2}) + \sqrt{2})X_{ij})_{i=1 \ldots n, j=i \ldots n},
\]

\[
\text{vec} : S^n \to \mathbb{R}^{n^2}, \quad \text{vec}(X) = (X_{ij})_{i=1 \ldots n, j=1 \ldots n},
\]

Here, \(\delta_{ij}\) represents the Kronecker delta.

These linear transformations allow to represent the operator \(A\) by

\[
A = \begin{bmatrix}
\text{svec}(A^{(1)})^T \\
\vdots \\
\text{svec}(A^{(m)})^T
\end{bmatrix} \quad \text{or} \quad \bar{A} = \begin{bmatrix}
\text{vec}(A^{(1)})^T \\
\vdots \\
\text{vec}(A^{(m)})^T
\end{bmatrix},
\]

where \(A(X) = A \cdot \text{svec}(X) = \bar{A} \cdot \text{vec}(X)\).

2 AHO-QMR-Method

2.1 Motivation

In [3], a first order method (augmented primal dual method, or short, APD method) is proposed for solving a suitable reformulation of the primal-dual pair \((P)\) and \((D)\). The APD method applies the L-BFGS algorithm to
minimize an augmented primal-dual function \( \Psi \) (which is the sum of some “squared distance function” \( \phi \) and a regularization term \( \alpha f \)). The numerical results reported in [3] are promising, but for certain numerical test problems, the convergence of the APD algorithm is rather slow when the iterates are close to the optimal solution of \((P)\) and \((D)\). To accelerate convergence we implemented several forms of preconditioning to the Hessian of \( \Psi \). We did not observe any substantial improvement of the rate of convergence. To understand the reason for the observed slow rate of convergence, we applied a conjugate gradient method to the (generalized) Newton step for \( \Psi \), and solved each Newton step to machine precision, thus imitating the “best possible form of preconditioning”. Nevertheless, with increasing values of the dimension \( n \), the observed rate of convergence for the outer iterations became slower. The theoretical result in [3] stating that the generalized Hessian of \( \Psi \) is positive definite at the optimal solution (under standard assumptions) refers to a fixed value of \( n \). It does not address the question how the condition number will develop for problems of a given structure when \( n \) is increasing. For larger values of \( n \), the above observation suggests to switch to another characterization of the optimal solution as soon as the convergence of the APD method slows down.

In the next subsection we aim at exploiting the fact that a “reasonable” approximation to the primal-dual optimal solution can be generated by the APD method. For many examples such an approximation is obtained after a small number of iterations.

### 2.2 The AHO System

Solving \((P)\) and \((D)\) is equivalent to solving the system

\[
\begin{align*}
\mathcal{A}(X) &= b, \\
\mathcal{A}^*(y) + S &= C, \\
XS + SX &= 0, \\
X \succeq 0, \quad S \succeq 0.
\end{align*}
\]
Here, we applied the AHO-symmetrization (Alizadeh, Haeberly, and Overton [1]) leading to the linearized system

\[
\begin{align*}
A(\Delta X) &= b - A(X), \\
A^*(\Delta y) + \Delta S &= C - A^*(y) - S, \\
\Delta XS + S\Delta X &= -XS - SX.
\end{align*}
\]

The AHO-direction is of particular interest. While the local superlinear convergence of most other interior-point search directions is restricted to iterates near a central path, see e.g. [7], Newtons method using the AHO direction is quadratically convergent (under the assumptions in Section 1.1) for any starting point that is sufficiently close to the primal-dual solution, see [1]. In particular, positive definiteness of the initial point is not required.

When \( m \) is large, a direct solution of system (1) is not possible. For such situations we propose a reformulation of the above AHO-linearization that is suitable for the solution by iterative methods.

### 2.3 Initial Point

Let \( k \in \{1, \ldots, n - 1\} \) be the number of positive eigenvalues of \( X^{\text{opt}} \) and \( m \) be the number of linear equality constraints (in \( A \)). Then, it is easy to see that uniqueness of \((X^{\text{opt}}, S^{\text{opt}})\) implies \( m \in \left\{ \frac{k(k+1)}{2}, \ldots, \frac{k(k+1)}{2} + k(n-k) \right\} \).

Conversely, if \( m \) is given, then it follows that

\[
k \in \{\left\lfloor \frac{1}{2}(2n + 1 - \sqrt{(2n+1)^2 - 8m}) \right\rfloor, \ldots, \left\lceil \frac{1}{2} \sqrt{1 + 8m} - 1 \right\rceil \}. \tag{2}
\]

We assume that some initial approximation \((X^{\text{APD}}, S^{\text{APD}})\) of \((X^{\text{opt}}, S^{\text{opt}})\) is given.

First, we compute a corrected pair \((X^C, S^C) \in S^n_+ \times S^n_+\) which is semidefinite and has a common eigenbasis. In the following procedure we exploit the fact that eigenspaces to distinct eigenvalues are locally Lipschitz continuous. If \(X^{\text{APD}}\) and \(S^{\text{APD}}\) are (nearly) complementary, this allows to “separate” the eigenvectors of \(X^{\text{APD}}\) and of \(S^{\text{APD}}\) by forming the difference \(Z = X^{\text{APD}} - S^{\text{APD}}\).
1. Compute \((X_P, S_P) := (\Pi_{S^+_T}(X^{APD}), \Pi_{S^+_T}(S^{APD}))\)

2. Define \(Z := \frac{X_P}{\|X_P\|_F} - \frac{S_P}{\|S_P\|_F}\) and calculate the eigenvalue-decomposition \(Z = U D U^T\).

3. Form \(X^e := U^T X^{APD} U, S^e := U^T S^{APD} U, D_X := \text{Diag}(\text{diag}(X^e)), D_S := \text{Diag}(\text{diag}(S^e))\).

4. Let \(\delta := \left(\|X^e - D_X\|^2_F + \|S^e - D_S\|^2_F\right)^{1/2}\)
   Increase the diagonal values of \(D_X\) and \(D_S\), if necessary, such that for each \(i\) at least one of the numbers \((D_X)_{i,i}\) or \((D_S)_{i,i}\) is at least \(\delta\), and such that the number of diagonal entries in \(D_X\) (and in \(D_S\)) with value at least \(\delta\) conforms to the bound (2).

5. Set \(X^C := U D_X U^T\) and \(S^C := U D_S U^T\).

Note: Unless \((X_P, S_P) = (X^{APD}, S^{APD})\), the projected point \((X_P, S_P)\) is closer to the unknown point \((X^{opt}, S^{opt})\) than \((X^{APD}, S^{APD})\). The measure \(\delta\) is an upper bound for the distance of \((X_P, S_P)\) to a matrix pair with common eigenbasis. Due to the modifications of the diagonal entries, the distance of \((X^C, S^C)\) to \((X_P, S_P)\) may be somewhat larger than \(\delta\).

Having a common eigenbasis, a sufficient number of “large” elements (of value at least \(\delta\)) in \(X^C\) and \(S^C\), and \(X^C + S^C \succeq \delta I\), is crucial to guarantee that the reformulations in the next section do not rely on inverses of dense or of poorly conditioned matrices.

When \(X^{APD}, S^{APD}\) are close to a strictly complementary solution \(X^{opt}, S^{opt}\), the corrected point \(X^C, S^C\) is also close to \(X^{opt}, S^{opt}\). This closeness was also observed in numerical tests as reported in Section 3.

### 2.4 Symmetrizing the AHO System

Next, we illustrate a reformulation of the AHO linearization. We assume that a given pair \((X, S) = (X^C, S^C)\) is the output of the procedure above and recall the AHO linearization (1). Let the unitary matrix \(U\) satisfy
\[ X = U \Lambda U^T \text{ and } S = U \Sigma U^T \] with diagonal matrices \( \Lambda \), \( \Sigma \). The mapping \( U : S^n \to S^n \) defined by \( U(Y) := U Y U^T \) is a bijective linear transformation. Let \( \Delta \hat{X} := U^T \Delta X U \), \( \Delta \hat{S} := U^T \Delta S U \), \( \hat{C} := U^T C U \) and the linear operator \( \hat{A} \) be defined by

\[
\hat{A}(\hat{X}) := \begin{pmatrix}
\hat{A}(1) \cdot \hat{X} \\
\vdots \\
\hat{A}(m) \cdot \hat{X}
\end{pmatrix} := \begin{pmatrix}
U^T A(1) U \cdot \hat{X} \\
\vdots \\
U^T A(m) U \cdot \hat{X}
\end{pmatrix}.
\]

The transformation \( U \) yields the following reformulation of system (1):

\[
\begin{align*}
\hat{A}(\Delta \hat{X}) &= b - A(X), \\
\hat{A}^*(\Delta y) + \Delta \hat{S} &= \hat{C} - \hat{A}^*(y) - \Sigma, \\
\Sigma \Delta \hat{X} + \Delta \hat{X} \Sigma + \Lambda \Delta \hat{S} + \Delta \hat{S} \Lambda &= -2 \Lambda \Sigma.
\end{align*}
\]

By definition, \( A(\Delta X) = \hat{A}(\Delta \hat{X}) \) holds.

For \( 1 \leq i, j \leq n \) let the entries of the symmetric \( n \times n \)-matrix \( \tilde{\Lambda} \) be given by \( \tilde{\Lambda}_{i,j} := \Lambda_i + \Lambda_j \) and likewise, \( \tilde{\Sigma} := (\Sigma_i + \Sigma_j)(i,j) \). Then, the last equation of (T1) can be reformulated as

\[
\tilde{\Sigma} \circ \Delta \hat{X} + \tilde{\Lambda} \circ \Delta \hat{S} = -2 \Lambda \Sigma.
\]

Note that the above formulation (redundantly) lists equations associated with the off-diagonal elements twice. This redundancy is eliminated in the formulation (T2) below. For the reformulation (T2) it will be convenient to use \texttt{svec} and the following linear transformation:

\[
\text{vec} : S^n \to \mathbb{R}^{\frac{n(n+1)}{2}}, \quad \text{vec}(X) = (X_{ij})_{i=1...n, \ j=i...n}.
\]

Set \( \hat{\Lambda} := \text{vec}(\Lambda) \), \( \hat{\Sigma} = \text{vec}(\tilde{\Sigma}) \), \( \Delta \hat{x} = \text{svec}(\Delta \hat{X}) \), \( \Delta \hat{s} = \text{svec}(\Delta \hat{S}) \) and

\[
\hat{\Lambda}^D = \text{Diag}(\hat{\Lambda}), \quad \hat{\Sigma}^D = \text{Diag}(\hat{\Sigma}), \quad \hat{A} = \begin{bmatrix} \text{svec}(\hat{A}(1))^T \\ \vdots \\ \text{svec}(\hat{A}(m))^T \end{bmatrix}.
\]

\[\]
so that $\hat{A}(\Delta \hat{X}) = \hat{A}\Delta \hat{x}$.

Then, we can reformulate $(T_1)$ as

$$(T_2) \quad \begin{bmatrix} \hat{A} & I \\ \hat{A}^T & \hat{D} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta \hat{x} \end{bmatrix} = \begin{bmatrix} b - \hat{A}(X) \\ \text{svec}(\hat{C} - \hat{A}^*(y) - \Sigma) \end{bmatrix} =: \begin{bmatrix} \text{rhs}_1 \\ \text{rhs}_2 \end{bmatrix}.$$ 

Next, we eliminate part of the variable $\Delta \hat{s}$. To this end, we choose the $m$ largest values of $\{\hat{\Lambda}_d \hat{\Sigma}_d \mid 1 \leq d \leq \frac{n(n+1)}{2}\}$, with the assignment $\hat{\Lambda}_d \hat{\Sigma}_d = \infty$, and collect the corresponding indices $d$ in the set $I$. Note, that there can be no index $d$, such that $\hat{\Lambda}_d \hat{\Sigma}_d = 0$. Otherwise, there exists a pair $(i,j)$, such that $\tilde{\Lambda}_{ij} = \Lambda_{ii} + \Lambda_{jj} = 0, \tilde{\Sigma}_{ij} = \Sigma_{ii} + \Sigma_{jj} = 0$. This contradicts $X + S \succ 0$.

Due to Step 4 in Section 2.3 there exist

$$\hat{k} \in \{\lfloor \frac{1}{2}(2n + 1 - \sqrt{(2n + 1)^2 - 8m}) \rfloor, \ldots, \lfloor \frac{1}{2}(\sqrt{1 + 8m - 1}) \rfloor\}$$

and $q_1, \ldots, q_n \in \{1, \ldots, n\}$, $q_\mu \neq q_\nu$ for $\mu \neq \nu$, such that $\Lambda_{q_1q_1}, \ldots, \Lambda_{q_kq_k} \geq \delta$ and $\Sigma_{q_{k+1}q_{k+1}}, \ldots, \Sigma_{q_nq_n} \geq \delta$ holds. Hence, there are at least $m$ elements $d \in \{1, \ldots, \frac{n(n+1)}{2}\}$ satisfying $\hat{\Lambda}_d > 0$. Therefore, $\hat{\Lambda}_d > 0$ for all $d \in I$.

Furthermore, there are at least $\frac{n(n+1)}{2} - m$ elements $d \in \{1, \ldots, \frac{n(n+1)}{2}\}$ satisfying $\hat{\Sigma}_d > 0$. Therefore, by definition of $I$, every index $d$ with $\hat{\Sigma}_d = 0$ is in $I$. We conclude:

The assignment

$$\Delta \hat{s}_d = \frac{\text{rhs}_3 - \hat{\Sigma}_d \Delta \hat{x}_d}{\Lambda_d}$$

is well-defined for $d \in I$. Moreover, when $\Pi := \{1, \ldots, \frac{n(n+1)}{2}\} \setminus I$ we have

$$\hat{\Sigma}_d > 0 \ \forall \ d \in \Pi.$$
We partition \((T_2)\) as

\[
\begin{bmatrix}
\hat{A}_I & \hat{A}_{II} \\
\hat{A}^T_I & I \\
\hat{A}^T_{II} & I \\
\hat{\Sigma}_I & \hat{\Lambda}_I \\
\hat{\Sigma}^D_{II} & \hat{\Lambda}^D_{II}
\end{bmatrix}
\begin{bmatrix}
\Delta y \\
\Delta \hat{x}_I \\
\Delta \hat{x}_{II} \\
\Delta \hat{\Sigma}_I \\
\Delta \hat{\Lambda}_{II}
\end{bmatrix} =
\begin{bmatrix}
\text{rhs}_1 \\
\text{rhs}_2^I \\
\text{rhs}_3^I \\
\text{rhs}_4^I \\
\text{rhs}_5^I
\end{bmatrix}.
\]

Using (3) we eliminate \(\Delta \hat{s}_I\) from \((T'_2)\) and get

\[
\begin{bmatrix}
\hat{A}_I & \hat{A}_{II} \\
\hat{A}^T_I & Q_I \\
\hat{A}^T_{II} & I \\
\hat{\Sigma}^D_{II} & \hat{\Lambda}^D_{II}
\end{bmatrix}
\begin{bmatrix}
\Delta y \\
\Delta \hat{x}_I \\
\Delta \hat{x}_{II} \\
\Delta \hat{s}_{II}
\end{bmatrix} =
\begin{bmatrix}
\text{rhs}_1 \\
\text{rhs}_2^I \\
\text{rhs}_3^I \\
\text{rhs}_4^I
\end{bmatrix}.
\]

Here, \(Q_I := -\left(\hat{\Lambda}^P_I\right)^{-1}\hat{\Sigma}_I^D\) and \(\text{rhs}_3^I := \left(\text{rhs}_4^I / \hat{\Sigma}_d\right)_{d \in I}\).

The final step is the symmetrization of \((T_3)\):

It follows from (4), that \(Q_{II} := \left(\hat{\Sigma}^D_{II}\right)^{-1}\hat{\Lambda}^D_{II}\) and \(\text{rhs}_3^{II} := \left(\text{rhs}_4^{II} / \hat{\Sigma}_d\right)_{d \in II}\) are well-defined, and the reformulation

\[
\begin{bmatrix}
\hat{A}_I & \hat{A}_{II} \\
\hat{A}^T_I & Q_I \\
\hat{A}^T_{II} & I \\
\hat{A}^T_{II} & Q_{II}
\end{bmatrix}
\begin{bmatrix}
\Delta y \\
\Delta \hat{x}_I \\
\Delta \hat{x}_{II} \\
\Delta \hat{s}_{II}
\end{bmatrix} =
\begin{bmatrix}
\text{rhs}_1 \\
\text{rhs}_2^I \\
\text{rhs}_3^I \\
\text{rhs}_4^I
\end{bmatrix}
\]

is equivalent to \((T_3)\). The resulting system \((T_4)\) is symmetric and of dimension \(n(n + 1)\).

Note that the (diagonal) matrices \(Q_I\) and \(Q_{II}\) only involve the inverses of the “large” components of \(\hat{\Lambda}\) and \(\hat{\Sigma}\). A further elimination of \(\Delta \hat{x}_{II}\) or \(\Delta \hat{s}_{II}\) using the third or fourth block row is easily possible, but a stable symmetrization of the resulting system turns out to be rather difficult.

We stress that the matrix \(\hat{A}\) in \((T_4)\) is typically dense, and that this matrix is never formed explicitly. Only multiplications with the system matrix in \((T_4)\) are being used. A multiplication with \(\hat{A}\), for example, is carried out by rescaling the matrix variable with \(U\) and \(U^T\) and applying \(A\) to the result.
2.5 Preconditioning the Symmetrized System

The APD-approach in [3] is based on computing a Cholesky factor of $AA^T$. When $A$ is sparse, this Cholesky factor is typically several orders of magnitude cheaper to compute than the factorizations used in interior-point methods. In the numerical examples below, the Cholesky factor is reused to precondition the first block row and block column of $(T_A)$ leading to a unitary first block row and block column. As before, we stress that this unitary block is not computed explicitly.

The resulting preconditioned system was solved with the symmetric form of the QMR algorithm, see [2, 5]. The accuracy of the QMR method was determined adaptively depending on the norm of the right hand side; for smaller norms a higher relative accuracy of the AHO-QMR-correction was required.

In our numerical examples, the AHO-QMR-method converges very rapidly when started near the optimal solution. On the other hand, it may not converge (or converge to a point violating the semidefiniteness constraints) when applied to initial points far away from the optimal solution. Therefore, we propose a hybrid approach for solving large-scale semidefinite programs:

1. Use the APD-approach described in Section 2.1 and [3] until its convergence slows down.

2. Switch to AHO-QMR, i.e. solve the system $(T_A)$ with a preconditioned QMR algorithm. Repeat AHO-QMR as long as it is more effective than APD. Revert to APD if necessary, i.e. GOTO Step 1 if the AHO-QMR step does not satisfy certain convergence criteria.

For simplicity, we will denote this approach as HYBRID.

2.5.1 Ill-Conditioning

Forming the normal equations in interior-point methods generally leads to a systematic form of ill-conditioning. The direct factorizations of the normal
equations nevertheless often yield satisfactory numerical solutions. Iterative methods, in contrast, are considerably more sensitive to ill-conditioning. The definition of $X^C, S^C$ implies that the above reformulations leading to $(T_4)$ do not introduce any systematic ill-conditioning. By avoiding a nonsymmetric (non-normal) system, the number of QMR iterations can be reduced, and by exploiting the symmetry, the computational effort for each iteration is also reduced by 50%.

Since first order methods are sensitive to scaling, the input data is rescaled to an equivalent problem as follows before starting APD or HYBRID: Let $\mathcal{L}$ denote the null space of $\mathcal{A}$. First, a symmetric matrix $B$ is generated with $\mathcal{A}(B) = b$. Then $B$ is projected onto $\mathcal{L}^\perp$, and $C$ is projected onto $\mathcal{L}$. Finally, $B$ and $C$ are replaced with $B/\|B\|_F$ and $C/\|C\|_F$, and $b$ is replaced with $\mathcal{A}(B)$. (Note that given the Cholesky factor of $\mathcal{A}\mathcal{A}^T$, this rescaling is computationally inexpensive.)

The rescaling implies that $\|X\|_F \geq 1$, and $\|S\|_F \geq 1$ for any feasible solution $X, S$. The iterates generated by the APD method are always projected onto the set satisfying the primal-dual equations as well as $C \cdot X = b^Ty$. Thus, the accuracy of the approximate solution obtained by the APD algorithm or by the HYBRID algorithm can be measured by $|\lambda_{\min}(X)| + |\lambda_{\min}(S)|$, where $X$ and $S$ are the iterates associated with the rescaled data. This is the error measure used in the second part of the numerical results below, where APD and HYBRID are compared to each other.

3 Numerical Examples

To illustrate the potential of the HYBRID-approach, we present some numerical results:
### 3.1 Comparison with SeDuMi

First, HYBRID is compared with the software package SeDuMi, see [9, 6], for some sparse semidefinite programs of dimension up to 300. It was not possible to run SeDuMi for such programs of dimension 400 or higher on our PC. The examples were generated with a generator for random sparse semidefinite programs by the research group of Franz Rendl, [8]. The input for this generator consists of the dimension of the matrix variable \( n \), the number of equality constraints \( m \), a sparsity parameter \( p \), and a random number \texttt{rand.seed}.

We used \( p=3 \) for every example. The input for the remaining parameters is listed in Table 1 for the smaller problems and in Table 3 for the larger ones, which are beyond the computability of SeDuMi on our machine. In our implementation, we represent \( \mathcal{A} \) by \( \tilde{\mathcal{A}} \) as defined in Section 1.1. In Table 1 we state the sparsity of \( \tilde{\mathcal{A}} \) and of the Cholesky factor \( L \) used in the HYBRID-approach, namely the absolute number (and the relative value in \%) of nonzero elements in both matrices. The variable \( X \) is stored as a \( n^2 \)-dimensional vector, since \( \tilde{\mathcal{A}} \) contains \( n^2 \) columns. For the problems in Table 1, the density of the Cholesky factor used in SeDuMi is about 50\% for every example.

<table>
<thead>
<tr>
<th>( n^2 )</th>
<th>( m )</th>
<th>\texttt{rand_seed}</th>
<th>( \text{nnz}(\tilde{\mathcal{A}}) ) (%)</th>
<th>( \text{nnz}(L) ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1225</td>
<td>315</td>
<td>353153</td>
<td>1880 (0.487)</td>
<td>3673 (3.702)</td>
</tr>
<tr>
<td>2500</td>
<td>638</td>
<td>506383</td>
<td>3796 (0.238)</td>
<td>12432 (3.054)</td>
</tr>
<tr>
<td>4900</td>
<td>1243</td>
<td>7012433</td>
<td>7422 (0.122)</td>
<td>40046 (2.592)</td>
</tr>
<tr>
<td>10000</td>
<td>2525</td>
<td>10025253</td>
<td>15084 (0.058)</td>
<td>154908 (2.430)</td>
</tr>
<tr>
<td>19600</td>
<td>4935</td>
<td>14049353</td>
<td>29454 (0.030)</td>
<td>544214 (2.235)</td>
</tr>
<tr>
<td>40000</td>
<td>10050</td>
<td>200100503</td>
<td>60002 (0.015)</td>
<td>2327485 (2.304)</td>
</tr>
<tr>
<td>90000</td>
<td>22575</td>
<td>300225753</td>
<td>134816 (0.007)</td>
<td>11501478 (2.257)</td>
</tr>
</tbody>
</table>

Table 1: Small random semidefinite programs - input data
We tested the problems in Table 1 with SeDuMi and with the HYBRID-approach. The focus of this paper lies on the final accuracy that can be obtained for some random – and evidently reasonably well-conditioned – problems by a first order approach. For a given triple \((X, y, S)\) we define the primal-dual error as

\[
\text{error}_{PD} := \left( \frac{\|X - \Pi_S S^n(X)\|_F^2}{1 + \|X\|_F^2} + \frac{\|S - \Pi_S S^n(S)\|_F^2}{1 + \|S\|_F^2} + \frac{\|XS\|_F^2}{1 + \|X\|_F^2 \|S\|_F^2} \right)^\frac{1}{2} + \frac{\|A(X) - b\|_2^2}{1 + \|b\|_2^2} + \frac{\|A^*(y) + S - C\|_2^2}{1 + \|C\|_2^2} \right)^\frac{1}{2}.
\]

In Table 2 we list the dimensions of the programs described by Table 1, the accuracy achieved by SeDuMi (set to compute the most accurate approximation possible, i.e. setting \(\text{pars.eps}:=0\)) and the computation time in seconds \((\text{error}_{PD}^{SeD}, \text{CPU-SeD})\), and the final accuracy and the total time for the HYBRID-approach \((\text{error}_{PD}^{HYB}, \text{CPU-HYB})\). When using the default options of SeDuMi, the error measure \(\text{error}_{PD}^{SeD}\) is about \(4 \cdot 10^{-6}\) for these examples, and the running time is about half as long as for \(\text{pars.eps}=0\).

For the examples described in Table 1, the HYBRID-approach consistently returns solutions of higher accuracy than SeDuMi. While SeDuMi is much faster for the smaller dimensions, HYBRID is considerably more efficient for the higher ones. Figure 1 gives an impression of the relation between

<table>
<thead>
<tr>
<th>(n^2)</th>
<th>(m)</th>
<th>(\text{error}_{PD}^{SeD})</th>
<th>CPU-SeD</th>
<th>(\text{error}_{PD}^{HYB})</th>
<th>CPU-HYB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1225</td>
<td>315</td>
<td>1.6040e-07</td>
<td>2.0</td>
<td>1.2854e-15</td>
<td>29.6</td>
</tr>
<tr>
<td>2500</td>
<td>638</td>
<td>1.3767e-07</td>
<td>5.5</td>
<td>1.1763e-15</td>
<td>65.6</td>
</tr>
<tr>
<td>4900</td>
<td>1243</td>
<td>1.6557e-07</td>
<td>14.2</td>
<td>1.7410e-15</td>
<td>117.6</td>
</tr>
<tr>
<td>10000</td>
<td>2525</td>
<td>1.0454e-07</td>
<td>91.5</td>
<td>1.4504e-15</td>
<td>375.5</td>
</tr>
<tr>
<td>19600</td>
<td>4935</td>
<td>5.1526e-08</td>
<td>610.0</td>
<td>1.2759e-14</td>
<td>995.1</td>
</tr>
<tr>
<td>40000</td>
<td>10050</td>
<td>3.1180e-08</td>
<td>5583.9</td>
<td>1.6705e-15</td>
<td>2844.3</td>
</tr>
<tr>
<td>90000</td>
<td>22575</td>
<td>7.3666e-08</td>
<td>52698.5</td>
<td>2.9179e-15</td>
<td>7496.8</td>
</tr>
</tbody>
</table>

Table 2: Small random semidefinite programs - results
dimension and computation time for both methods.

![Figure 1: SeDuMi vs HYBRID](image)

**Figure 1: SeDuMi vs HYBRID**

### 3.2 Comparison with the plain APD approach

In Table 4, we compare the HYBRID-approach with the APD approach. APD was proposed to minimize sparse large-scale semidefinite problems. It was not intended for computing highly accurate solutions – this is the aim of the HYBRID-approach. We used the random generator [8] to generate sparse semidefinite programs of dimension 400 and up with 30000 and more equality constraints. The problems in Table 3 were tested with the APD
method and with the HYBRID-approach. We set the stopping criterion for the APD method to 4000 iterations resulting in rather high running times – almost two days for the largest example. The HYBRID-approach was run until some internal stopping criterion (depending on the current accuracy and the condition of the AHO-system) was satisfied. Since all iterates of APD and HYBRID satisfy the linear equations up to rounding errors, and since the iterates are normalized, we use the error measure proposed at the end of Section 2.5.1 for this comparison.

In Table 4 we list the dimensions of the programs described by Table 3 and the computation time and accuracy achieved by the APD approach (error$^{APD}$, CPU-APD) and by the HYBRID-approach (error$^{HYB}$, CPU-HYB).

All examples were tested with a MATLAB(R) implementation on a PC with eight Intel(R) Xeon(R) CPUs X3470 with 2.93GHz and with 16 GB RAM. (The Matlab version used operates only with one processor.)

### 3.3 A large example of the Lovasz $\vartheta$-number

We also applied the HYBRID-approach to the problem of computing the Lovasz-$\vartheta$-number of a randomly generated graph with $n=1000$ vertices and

<table>
<thead>
<tr>
<th>$n^2$</th>
<th>$m$</th>
<th>rand_seed</th>
<th>nnz($A$) (%)</th>
<th>nnz($L$) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>160000</td>
<td>30000</td>
<td>4003030</td>
<td>179094 (0.004)</td>
<td>5602524 (0.620)</td>
</tr>
<tr>
<td>250000</td>
<td>30000</td>
<td>5003030</td>
<td>179062 (0.002)</td>
<td>179065 (0.020)</td>
</tr>
<tr>
<td>360000</td>
<td>40000</td>
<td>6004030</td>
<td>238830 (0.002)</td>
<td>148600 (0.009)</td>
</tr>
<tr>
<td>490000</td>
<td>50000</td>
<td>7005030</td>
<td>298404 (0.001)</td>
<td>112005 (0.004)</td>
</tr>
<tr>
<td>640000</td>
<td>70000</td>
<td>8007030</td>
<td>417856 (0.001)</td>
<td>245311 (0.005)</td>
</tr>
<tr>
<td>810000</td>
<td>100000</td>
<td>90010030</td>
<td>597130 (0.001)</td>
<td>1932642 (0.019)</td>
</tr>
<tr>
<td>1000000</td>
<td>100000</td>
<td>100010030</td>
<td>596964 (0.001)</td>
<td>222603 (0.002)</td>
</tr>
</tbody>
</table>

Table 3: Random semidefinite programs - input data
Table 4: Random semidefinite programs - results

<table>
<thead>
<tr>
<th>$n^2$</th>
<th>$m$</th>
<th>$\text{error}^{APD}$</th>
<th>CPU-APD</th>
<th>$\text{error}^{HYB}$</th>
<th>CPU-HYB</th>
</tr>
</thead>
<tbody>
<tr>
<td>160000</td>
<td>30000</td>
<td>4.8026e-09</td>
<td>16688.9</td>
<td>4.6000e-14</td>
<td>7487.4</td>
</tr>
<tr>
<td>250000</td>
<td>30000</td>
<td>9.5350e-08</td>
<td>21923.3</td>
<td>6.2800e-13</td>
<td>8371.9</td>
</tr>
<tr>
<td>360000</td>
<td>40000</td>
<td>6.2031e-08</td>
<td>36430.7</td>
<td>1.0000e-15</td>
<td>16229.4</td>
</tr>
<tr>
<td>490000</td>
<td>50000</td>
<td>1.1035e-07</td>
<td>50053.0</td>
<td>2.4000e-14</td>
<td>29313.0</td>
</tr>
<tr>
<td>640000</td>
<td>70000</td>
<td>9.5654e-008</td>
<td>78572.9</td>
<td>0</td>
<td>32933.8</td>
</tr>
<tr>
<td>810000</td>
<td>100000</td>
<td>6.4942e-008</td>
<td>104463.2</td>
<td>1.2000e-14</td>
<td>68287.9</td>
</tr>
<tr>
<td>1000000</td>
<td>100000</td>
<td>1.0493e-007</td>
<td>142167.5</td>
<td>0</td>
<td>89141.6</td>
</tr>
</tbody>
</table>

$m=249670$ edges. In this case, the matrix $\tilde{A}$ has $n^2 = 10^6$ columns and $m=249670$ rows with $\text{nnz}(\tilde{A}) = 500338$. The Cholesky factor $L$ is a diagonal matrix with 249670 rows and columns. After a total computation time of 87829.8 sec the primal-dual error was reduced to 0 (machine precision).

4 Conclusion

A first order method is presented for computing a solution of a semidefinite program up to high accuracy while using only partial second order information. Key points are the generation of a suitable starting point that allows a partial elimination and symmetrization of the AHO system and an approach for preconditioning the linear equations.

The theoretical motivation of the approach in this paper was carried out under strong nondegeneracy assumptions. The algorithm, however, is well-defined also when these assumptions are violated; it converges – possibly at a slower rate – as long as a primal-dual solution exists. On the other hand, there certainly exist problems that do satisfy the strong assumptions but that are ill-conditioned, and for which the approach in this paper will be very slow. The examples reported in this paper happen to be less ill-conditioned and lead to very encouraging numerical results.
References


