Pre- and Post-processing Sum-of-squares Programs in Practice

Johan Löfberg

***Abstract*—Checking non-negativity of polynomials using sum-of-squares has recently been popularized and found many applications in control. Although the method is based on convex programming, the optimization problems rapidly grow and result in huge semidefinite programs. Additionally, they often become increasingly ill-conditioned. To alleviate these problems, it is important to exploit properties of the analyzed polynomial, and post-process the obtained solution. This paper describes how the sum-of-squares module in the MATLAB toolbox YALMIP handles these issues.**

***Index Terms*—Optimization methods, Polynomials, Software packages.**

I. Introduction

The purpose of this paper is to introduce the reader to some straightforward, practical, and easily implemented ways to simplify and improve sum-of-squares programs by exploiting sparsity and symmetry in a pre-processing phase and apply suitable post-processing. More precisely, the sum-of-squares module of the MATLAB toolbox YALMIP [5] is presented.

The underlying problem addressed is checking non-negativity of a polynomial $f(x) : \mathbb{R}^n \to \mathbb{R}$, defined by its coefficients[1] $c \in \mathbb{R}^N$ and exponents $p_i \in \mathbb{N}_0^n$.

$$f(x) = f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{N} c_i \prod_{j=1}^{n} x_j^{p_{ji}} = \sum_{i=1}^{N} c_i x^{p_i}.$$

The idea in a sum-of-squares approach is to replace non-negativity with the, obviously sufficient, condition that the polynomial is a sum of squared terms.

$$f(x) = \sum_{i=1}^{M} h_i^2(x) = \sum_{i=1}^{M} (q_i^T v(x))^2 = v^T(x) Q v(x),\ Q = Q^T \succeq 0.$$

Hence, if we can find a vector of monomials $v(x)$ (from now on denoted the candidate monomials) and a positive semidefinite matrix Q, called the Gramian, non-negativity is guaranteed. For a given $v(x)$, equivalence of the two polynomials implies a set of linear equalities on the elements of the matrix Q. This together with the semidefiniteness constraint turns the sum-of-squares program into a semidefinite program (SDP). This paper will not elaborate on semidefinite programming or sum-of-squares programs in general, the reader is referred to [15], [12], [8] for introductions to these topics.

The sum-of-squares program boils down to an SDP whose solution time is polynomially related to the dimension of the matrix Q and thus the number of candidate monomials. The first problem we address in this paper is therefore how to select the candidate monomials $v(x) = \left[x^{s_1}, \ldots, x^{s_M}\right]^T,\ s_i \in \mathbb{N}_0^n$.

A second problem is to exploit properties of $f(x)$, once $v(x)$ has been derived, to parameterize the matrix Q in a more efficient way. Polynomials with sign-symmetries (a generalization of univariate even polynomials) can be dealt with very efficiently and we present straightforward methods to detect these symmetries and use them to block diagonalize the matrix Q. It should be mentioned that more advanced symmetry reduction based on group theory is discussed in [2] in a more general setting, and the concept of exploiting symmetries in sum-of-squares was discussed already in [12]. Recently, there has also been work on symmetry reductions of general SDPs, using linear algebra tools [7]. The purpose of this paper is however to address the problem from an implementation oriented point of view, and the goal is to derive a symmetry reduction technique that can be applied completely automatically, without any user guidance, using a minimal amount of computations and theoretical background, but still be competitive in terms of reduction strength for the problems at hand.

Finally, we address the problem of post-processing a solution to improve accuracy. It is a common mistake to believe that once the SDP has been solved successfully, a certificate of non-negativity is found. Due to the inherently sensitive formulation of many sum-of-squares programs (described more later) and finite precision in the solver, this is far from true.

A. Notation

The matrix P denotes exponents in $f(x)$ (the support), $P = \text{supp}(f) = \left[p_1, \ldots, p_N\right]$. In the same sense, S denotes the exponents in the candidate monomials $\left[s_1, \ldots, s_M\right]$. The scalar s_{ji} denotes the degree, with respect to the variable x_j, of the ith candidate monomial. Similar notation applies to p_{ji}.

II. Sum-of-squares in YALMIP

YALMIP features a general framework to develop optimization based algorithms. The bottom layer of YALMIP consists of numerical interfaces to most of the state-of-the-art numerical solvers. Around this numerical layer, a scripting language is available. Compared to alternative specialized software packages for polynomial optimization based on convex optimization [10], [4], [16], the sum-of-squares module in YALMIP is entirely integrated in the general modeling framework, and can thus be used in much more advanced and general scenarios. For example, YALMIP can be used to automatically derive and solve robust counter-parts of uncertain sum-of-squares programs, solve sum-of-squares programs with integer decision variables, or problems involving nonconvex sum-of-squares. However, the purpose of the paper is not to show-case the generality of YALMIP, but to describe the functionality that simplifies and post-processes the sum-of-squares programs that arise.

III. Pre-processing sum-of-squares programs

This section will introduce the main[2] algorithms used in YALMIP to reduce and simplify the sum-of-squares program.

A. Newton polytope

A useful and elegant tool in the analysis of multivariate polynomials is the Newton polytope. In our framework, the Newton polytope result guarantees that all necessary candidate monomials in a sum-of-squares decomposition of $f(x)$ are contained in the convex hull (**Co**) of the exponents in $f(x)$ [11, Theorem 1].

Theorem 1: If $f(x) = \sum_{i=1}^{N} c_i x^{p_i} = \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_{ij} x^{s_i} x^{s_j}$ then $2s_i \in \mathbf{Co}\left(p_1, p_2, \ldots, p_N\right) \triangleq \mathcal{P}$.

The author is with the Division of Automatic Control at the Department of Electrical Engineering, Linköpings universitet, SE-581 83 Linköping, Sweden (e-mail: johanl@isy.liu.se)

[1]The coefficients may depend on design variables, but we omit this to simplify notation.

[2]Minor pre-processing strategies used for homogeneous polynomials, simple degree-reducing variable changes and checks for trivially infeasible problems are omitted.

All candidate monomials s_i violating this condition can be omitted. A straightforward application of this theorem leads to a strategy to generate a sufficient initial set of candidate monomials.

Corollary 1: A sufficient initial set of candidates can be taken as all lattice points in the bounding box of the polytope $\frac{1}{2}\mathcal{P}$, hence $\min_i \frac{1}{2}p_{ji} \leq s_j \leq \frac{1}{2}\max_i p_{ji}$.

Example 1: Consider the polynomial $f(x) = 1 + x_1^4x_2^2 + x_1^2x_2^4$. Generating the initial candidate monomials in the bounding box of the Newton polytope yields

$$P = \begin{bmatrix} 0 & 4 & 2 \\ 0 & 2 & 4 \end{bmatrix},\ S = \begin{bmatrix} 0 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 & 1 & 0 & 2 & 1 & 2 \end{bmatrix}.$$

After applying the Newton polytope reduction the set of candidates reduces to

$$S = \begin{bmatrix} 0 & 2 & 1 & 1 \\ 0 & 1 & 2 & 1 \end{bmatrix}.$$

The Newton polytope based reduction can be implemented in different ways, and two approaches are discussed below.

1) Implementation using $\mathcal{H}$-representation of $\mathcal{P}$: The first and most direct approach is to calculate an $\mathcal{H}$-representation (hyperplane representation) of the convex hull, $\mathcal{P} = \{z \ : \ Az \leq b\}$, and then check the set of linear inequalities for all s_i. There are however some problems with this method. The first issue is when all points p_k are located on an affine subspace (homogeneous polynomials). Practical experience indicates that convex hull codes often have robustness problems in this case, and an initial projection of the data is needed. A second and more profound problem is the computational complexity. A set of N vertices in $\mathbb{R}^n$ can, in worst-case, define a polytope with $O(N^{\lfloor \frac{n}{2} \rfloor})$ facets [17]. Hence, calculating A and b and finally checking the candidates might become prohibitive. In fairness, it should be said that this does not seem to be an issue for the convex hulls typically occurring in sparse sum-of-squares decompositions.

2) Implementation based on separating hyperplanes: An alternative approach is to work with the set $\mathcal{P}$ implicitly. A candidate monomial s_i is not in $\frac{1}{2}\mathcal{P}$ if we can find a separating hyperplane between s_i and all points $\frac{1}{2}p_k$. Hence, instead of finding an $\mathcal{H}$-representation of $\mathcal{P}$ and checking each candidate against the sets of inequalities defining $\mathcal{P}$, we find a separating hyperplane for each candidate monomial.

Finding a separating hyperplane between s_i and $\frac{1}{2}\mathcal{P}$ can be formulated as the following linear program (LP).

$$\begin{aligned} \max_{a,b} \quad & a^T s_i - b \\ & \tfrac{1}{2}a^T p_k - b \ \leq \ 0 \ \forall k = 1, \ldots, N \\ & a^T s_i - b \ \geq \ 0 \end{aligned}$$

There are three possible solutions to this problem. If there exist no (strictly) separating hyperplane, the problem is infeasible and s_i must be kept. If the optimal solution is infinite, there exist a strictly separating hyperplane and s_i can be omitted. A third case is when s_i is on the boundary of $\frac{1}{2}\mathcal{P}$ (and thus should be kept) leading to an objective value of zero.

This indirect approach might seem inefficient at first, since it requires the solution of a large number of LPs. The LPs are however very small ($n+1$ variables). Moreover, a separating hyperplane for the candidate s_i can be used to check redundancy also for other candidates, thus possibly ruling out some candidates almost for free. This can be seen in the examples later where the number of LPs solved is far less than the number of candidate monomials. Another way to reduce the number of LPs is to check a-priori if s_i is equal to some point $\frac{1}{2}p_k$. If this is the case, the candidate monomial is trivially not outside the Newton polytope and must be kept.

The motivation for using this approach in YALMIP is five-fold. It is easily implemented, does not require a dedicated convex hull solver but only an LP solver, guaranteed polynomial-time worst case performance (assuming a polynomial-time LP solver is used). Additionally, the numerical problem with homogeneous polynomials does not seem to be an issue with this approach, hence no initial projection is needed and practice has shown that it is numerically robust. Finally, and this is the main reason, since YALMIP has a built-in framework for finding and interfacing a large number optimization packages (including more than 10 LP solvers), the algorithm will automatically have access to almost any LP solver installed on the system.

B. Diagonal inconsistency

The Newton polytope is very efficient at reducing the number of candidate monomials. It is however easy to construct polynomials where the Newton polytope based reduction is insufficient and leaves obviously redundant candidate monomials. In lack of a better term, we will denote the particular issue discussed here diagonal inconsistency. An example illustrates the idea.

Example 2: Consider once again $f(x) = 1 + x_1^4x_2^2 + x_1^2x_2^4$. Generating the candidate monomials and applying the Newton polytope leads to the decomposition $f(x) = v^T(x)Qv(x)$ with $v(x) = \left[1, x_1^2x_2, x_1x_2^2, x_1x_2\right]$. The last candidate monomial x_1x_2 will generate a term $Q_{44}x_1^2x_2^2$. Since this is the only way to obtain an $x_1^2x_2^2$ term, it follows that $Q_{44} = 0$. For Q to be positive semidefinite, this implies that the last row and column of Q must be zero. Correspondingly, x_1x_2 can be omitted.

The example hopefully illustrates what we are doing. If a squared candidate monomial is not a term in $f(x)$, and the squared monomial only is generated by squaring said monomial, it cannot be (a non-zero) part of the decomposition. The following result follows.

Theorem 2: A candidate monomial x^{s_i} is redundant in a sum-of-squares decomposition of a polynomial $f(x)$ with support in x^{p_j} if $\nexists (k, j \neq i, l \neq i) : \{2s_i = p_k, 2s_i = s_l + s_j\}$.

The theorem is a reformulation of Proposition 3.7 in [1]. Note that the reduction can be applied iteratively. If a candidate monomial is removed, it might happen that another candidate monomial, which previously not could be removed, turns out to satisfy the conditions for diagonal inconsistency.

In most cases, using the diagonal consistency check after applying the Newton polytope test does not reduce the number of candidate monomials, except for very particular polynomials. However, there are cases where applying the diagonal inconsistency check before the Newton polytope reduction can reduce the overall computational effort, and vice-versa. By default, the feature is turned off in YALMIP.

C. Block diagonalization

The strategies introduced in the previous sections focused on reducing the number of candidate monomials in the decomposition, using standard results on Newton polytopes and diagonal inconsistency. The material in this section aims at simplifying the decomposition, given a set of candidate monomials, using a simplified approach to detect and exploit symmetries automatically.

1) Univariate case: Let us for simplicity begin with an even univariate polynomial $f(x) = f(-x)$ (or sign-symmetric as we will denote this property later).

$$f(x) = v^T(x)Qv(x) = \begin{bmatrix} v_o(x) \\ v_e(x) \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix} \begin{bmatrix} v_o(x) \\ v_e(x) \end{bmatrix}. \quad (1)$$

In the decomposition above, even monomials $v_e(x)$ and odd monomials $v_o(x)$ are grouped separately. The key observation is that odd

terms only are generated by $v_e^T(x)Q_{12}v_o(x)$, while even terms only are generated by $v_e^T(x)Q_{11}v_e(x)$ and $v_o^T(x)Q_{22}v_o(x)$. Hence, it has to hold $v_e^T(x)Q_{12}v_o(x) = 0$. If there exist a decomposition with $Q \succeq 0$, changing Q_{12} to zero still yields $Q \succeq 0$ and all constraints on Q ensuring equivalence between the two polynomials are still satisfied (constraints related to even terms only involve Q_{11} and Q_{22}). It thus follows that there is no loss in generality to use a block diagonal decomposition.

$$f(x) = v^T(x)Qv(x) = \begin{bmatrix} v_o(x) \\ v_e(x) \end{bmatrix}^T \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \begin{bmatrix} v_o(x) \\ v_e(x) \end{bmatrix}. \quad (2)$$

The computational complexity of solving an SDP depends on several factors, such as the solution algorithm, sparsity, number of equality constraints etc., but as a rule of thumb, it is at least cubic in the size of the SDP constraints. Hence, by having two half as big SDP constraints $Q_{11} \succeq 0$ and $Q_{22} \succeq 0$, one can expect a considerable gain.

To prepare for a notation that will be used in the general case, we say that we have partitioned the candidate monomials into two sets, described by the indices $\mathcal{I}_1$ and $\mathcal{I}_2$. For the univariate case, the partition corresponds to the even monomials $\mathcal{I}_1 = \{i : s_i = 0 \pmod 2\}$ and odd $\mathcal{I}_2 = \{i : s_i = 1 \pmod 2\}$.

2) Multivariate case: As we saw above, block diagonalizing the sum-of-squares program for an even univariate polynomial is straightforward, and results in an SDP with two blocks. The multivariate case is slightly more involved, but the idea remains the same. The main difference in the multivariate case is that the polynomial can be sign-symmetric in multiple ways, with respect to several variables.

Example 3: Consider $f(x) = 1+x_1^4+x_1x_2+x_2^4+x_3^2$. Obviously, it is even in the variable x_3, but not in x_1 and x_2. However, it is sign-symmetric with respect to (x_1, x_2) jointly, i.e., $f(x_1, x_2, x_3) = f(-x_1, -x_2, x_3)$.

From the example it should be clear, that in the general multivariate case, there are 2^n possible sign-symmetries $f(x_1, \ldots, x_n) = f(\pm x_1, \ldots, \pm x_n)$.

To exploit the sign-symmetries to simplify the sum-of-squares program, we first have to find and represent the sign-symmetries. A sign-symmetry will be described using a binary vector $r \in \{0, 1\}^n$. If a variable x_j is involved in the sign-symmetry r, the jth element in r will be 1. Testing if a polynomial is symmetric w.r.t to the symmetry r can now be done by checking whether $r^T p_j$ is even for all exponents p_j. This leads us to the following definition.

Definition 1: The sign-symmetries of a polynomial $f(x)$ with support in the monomials x^{p_i} are defined by all vectors r_i such that $r_i \in \{0, 1\}^n$, $r_i^T P = 0 \pmod 2$

A brute force method to find all sign-symmetries is to generate all 2^n possible vectors r_i and check if $r_i^T P = 0 \pmod 2$. This is easily done for moderate dimensions, but for higher dimensions (roughly $n > 15$), more advanced schemes are needed. More efficient schemes can be implemented by exploiting properties of the symmetry. As an example, if the polynomial is symmetric w.r.t r_i and r_j, it immediately follows that the polynomial is symmetric w.r.t $r_i + r_j$. However, since sum-of-squares still is limited to relatively low dimensions, we will not consider these extensions.

Once a set of symmetries r_i have been found, the next step is to use these symmetries to block diagonalize the problem.

The block diagonalization algorithm is most intuitively explained in a recursive fashion, although the final result below is given in a more direct format. Partition the monomials x^{s_j} into two sets of candidates, those which satisfy the symmetry $r_1^T s_j = 0 \pmod 2$ and those which do not. Using the same argument as in the univariate case, it follows that the decomposition can be block diagonalized based on these two sets. The strategy is now applied separately to the two sets for the symmetry r_2, and the procedure is repeated for all symmetries.

Concretely, we are generating a partition of the candidates into q disjoint sets, where members of each set share a unique combination of sign-symmetries. This partitioning can thus be defined according to the following strategy.

Theorem 3: Given M candidate monomials x^{s_j} and m_R detected sign-symmetries defined by the binary matrix $R = \begin{bmatrix} r_1, \ldots, r_{m_R} \end{bmatrix}$. Let $W = R^T S = \begin{bmatrix} w_1, \ldots, w_M \end{bmatrix}$. The candidate monomials can be block partitioned into q blocks $\mathcal{I}_1, \ldots, \mathcal{I}_q$ where

$$\cup_{k=1}^{q} \mathcal{I}_k = \{1, \ldots, M\}, w_i = w_j \pmod 2 \Leftrightarrow i, j \in \mathcal{I}_k. \quad (3)$$

Hence, given the symmetries R, generated according to Definition (1), calculate the matrix $R^T S$ and find equal (mod 2) columns. An actual implementation thus requires nothing more than a matrix multiplication and a simple search.

Example 4: Let $f(x) = 1 + x_1^4 + x_1x_2 + x_2^4 + x_3^2$. Generation of a sufficient set of candidate monomials yields $1, x_1, x_2, x_1^2, x_1x_2, x_2^2$ and x_3. The polynomial is sign-symmetric in x_3 and jointly in x_1 and x_2. We thus have

$$R = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \; S = \begin{bmatrix} 0 & 1 & 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (4)$$

$$W = \begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (5)$$

Identifying the equal (mod 2) columns in W yields $\mathcal{I}_1 = \{1, 4, 5, 6\}$, $\mathcal{I}_2 = \{2, 3\}$ and $\mathcal{I}_3 = \{7\}$. Hence, the SDP will have three diagonal blocks with sizes 4, 2 and 1. The sum-of-squares decomposition can be solved using the following decomposition.

$$f(x) = \begin{bmatrix} 1 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}^T Q_1 \begin{bmatrix} 1 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T Q_2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + x_3 Q_3 x_3. \quad (6)$$

D. Matrix-valued sum-of-squares programs

The sum-of-squares concept can easily be extended to handle matrix-valued polynomials, and so can the block diagonalization strategy. The details are omitted and the interested reader is referred to [6].

IV. Post-processing sum-of-squares programs

The desired property of sum-of-squares programs is that it gives a certificate that a polynomial is non-negative. However, in practice, the certificate is typically not valid, due to termination criteria in the SDP solvers and the fundamental limitation posed by floating-point implementations. This is a fact that many users of sum-of-squares techniques seem to fail to appreciate. Even though the SDP solver reports success, the solution does not have to be a valid sum-of-squares certificate. To see why, recall the primal SDP problem which typically is solved when solving a sum-of-squares program.

$$\begin{aligned} \min_Q \quad & C \bullet Q \\ \text{subject to} \quad & A(Q) = b \\ & Q \succeq 0 \end{aligned}$$

The matrix Q is our Gramian, whereas the equality constraints $A(Q) = b$ describe equality between the analyzed polynomial and the sum-of-squares decomposition. The crucial fact is that the equality constraints only are satisfied in the limit in the solution process, and the optimization algorithm will terminate before they are exactly satisfied. The positive semidefiniteness constraint on the matrix Q

is however[3] guaranteed. This means that the polynomial constructed in the sum-of-squares decomposition indeed is non-negative, but it is not identical to the analyzed polynomial. In other words, we have shown non-negativity of a slightly perturbed polynomial[4].

In many cases, this is not an issue. The user is only interested in a strong numerical indication of non-negativity, not a strict certificate. However, in some cases a true certificate is required. Luckily, we can conclude non-negativity also from approximate decompositions in some cases, as the following simple theorem shows.

Theorem 4: Let $Q \in \mathbb{R}^{M\times M}$, A and b be solution and data for an SDP solving the sum-of-squares decomposition $f(x) = v(x)^T Q v(x)$. If $\lambda_{min}(Q) \geq M||A(Q)-b||_\infty$, the polynomial is non-negative.

Proof: Since the decomposition is inexact, we have $f(x) = v(x)^T Q v(x) + r(x)$, where $r(x)$ is the difference between the original polynomial and the decomposition. Since the equality constraints $A(Q)-b$ represent equality between the polynomial and the decomposition, the coefficients of $r(x)$ are bounded by $\epsilon = ||A(Q)-b||_\infty$. Assuming that a correct basis has been used in the decomposition, we can write $r(x)$ as a quadratic form $r(x) = v(x)^T R v(x)$. The matrix R can be chosen to have (fractions of) coefficients of $r(x)$ as some of its elements. Hence it follows that $||R||_\infty \leq M\epsilon$. Since the spectral norm of a matrix is bounded from above by any p-norm, it follows that the smallest eigenvalue of R is larger than $-M\epsilon$. Returning to the decomposition $f(x) = v(x)^T (Q+R) v(x)$, we conclude that $Q+R$ is guaranteed to be positive semidefinite if $\lambda_{min}(Q) \geq M||A(Q)-b||_\infty$. ∎

This means that if the Gramian Q is sufficiently positive definite, the polynomial is *certifiably* non-negative, even though the SDP solver failed to compute an exact decomposition. Unfortunately, the Gramian lives on the singularity border in many cases.

A. A posteriori block diagonalization

The pre-processing algorithms are not included only to make the SDP problems smaller, but also to improve the numerical conditioning of the problem. One of the most important contributions of the pre-processing step is to remove candidate monomials that *cannot* be part of the decomposition. If these terms are kept, the corresponding rows and columns of the Gramian have to be zero, thus causing all feasible Gramians to be singular. Clearly, in light of Theorem 4, this will cause severe issues if a true certificate is required, and just as important, cause numerical problems when solving the SDP.

The classical Newton reduction scheme and the proposed block diagonalization do work wonderfully on many examples, but there are even more cases where there are remaining candidate monomials that should be removed, and more block structure to exploit. Some of this structure could possibly be addressed by employing a more much advanced pre-processing step by, e.g., exploiting more general symmetries in the polynomial [2]. However, this requires a far more advanced machinery and can also be insufficient in some cases. Instead, the idea in YALMIP is to perform a post-processing step and analyze a given solution, make an educated guess from the numerical data to find an improved block diagonalization, reformulate the sum-of-squares program, and solve a new smaller problem where the complicating singular structure has been eliminated or at least reduced.

The idea is very simple; numerically small elements in the approximate Gramian Q are cleared to zero. Based on this new matrix, a sparsity pattern matrix is defined, having a 1 if the cleared Gramian has a non-zero element in the corresponding position, and a 0 otherwise. This sparsity pattern matrix can now be used for two purposes. To begin with, if a zero is found in the diagonal, it indicates that the corresponding monomial possibly can be removed from the decomposition. Secondly, after the zero diagonal rows and columns have been removed from the sparsity pattern matrix, we can search for a block diagonal structure. Typically, the matrix is not block diagonal immediately, but by performing a simple symmetric row and column permutation, blocks are easily detected. This is implemented in YALMIP by using MATLABs built-in block detecting Dulmage-Mendelsohn permutation. By applying the same permutation to the candidate monomials $v(x)$, a natural partition arise.

Example 5: Consider the following simple numerical example, obtained after solving a sum-of-squares program,

$$\begin{bmatrix} 1 \\ x_1 \\ x_1^2 \\ x_1x_2 \\ x_1x_2^2 \end{bmatrix}^T \begin{bmatrix} 0.82 & -0.00 & -0.00 & 0.00 & -0.07 \\ -0.00 & 2.29 & 0.57 & 0.00 & 0.00 \\ -0.00 & 0.57 & 1.29 & 0.00 & -0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & -0.00 \\ -0.07 & 0.00 & -0.00 & -0.00 & 0.57 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \\ x_1x_2 \\ x_1x_2^2 \end{bmatrix}.$$

By manual inspection, we can suspect that there is structure in this problem which the pre-solve algorithms missed. The fourth row (and column) is close to zero, indicating that monomial x_1x_2 should be removed. Furthermore, after removing the small terms, a block diagonal structure is easily found by a row and column permutation. This indicates that we should resolve the problem with two blocks, using the monomials $(1, x_1x_2^2)$ and (x_1, x_1^2).

V. Numerical examples

To illustrate the impact of the described algorithms, we study three polynomials previously addressed in the literature. The first example is based on the co-positivity problem in [8].

$$\begin{aligned} h(x) &= \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \\ x_5^2 \end{bmatrix}^T \begin{bmatrix} 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \\ x_5^2 \end{bmatrix} \\ f(x) &= h(x)(x_1^2+x_2^2+x_3^2+x_4^2+x_5^2). \end{aligned}$$

After defining this polynomial (code omitted[5]), the sum-of-squares solver in YALMIP reveals that 31 sign-symmetries have been found. The polynomial is even w.r.t to all variables, and YALMIP reports all 2^5-1 non-trivial symmetries. The block diagonalization transforms the original 35×35 semidefinite constraint into 10 element-wise constraints and 5 semidefinite constraints of dimension 5×5. For this polynomial, the Newton polytope computations turn out to be trivial, the pre-processing phase in the Newton polytope algorithm immediately detects that all candidate monomials s_j correspond to a point $\frac{1}{2}p_k$, hence all candidate monomials are trivially guaranteed to be inside $\frac{1}{2}\mathcal{P}$.

The second example is a substantially larger polynomial of degree 20 in 4 variables with 123 terms. See [9, equation (8)] for an exact definition. An initial generation leads to 330 candidate monomials.

[3]Assuming a standard primal-dual SDP solver is used.

[4]Alternatively one may parameterize the affine space describing the feasible Gramians and solve a problem involving a semidefinite constraint of the type $E-F(y)\succeq 0$, i.e., a problem described in a standard dual SDP setting. This will lead to a solution where the polynomial and the decomposition coincide, but the Gramian will in most solvers not be guaranteed to be feasible, since the positive semidefinite dual space normally is approached from the exterior. Hence, if the feasible set lacks a strict interior, the solution will almost surely be infeasible. As a side note, both SDP formulations are easily available in the sum-of-squares module in YALMIP.

[5]Computations were performed on a 2.4GHz PC running MATLAB 7.4, using GLPK [3] and SeDuMi [14]. Implementations can be found at `control.ee.ethz.ch/~joloef/wiki/pmwiki.php`.

The Newton polytope does a good job at reducing the number of candidate monomials by solving 26 LPs, but the result is still a fairly large SDP with a 137×137 constraint. Block diagonalization reduces the problem to a set of 16 coupled semidefinite constraints with sizes ranging from 6×6 to 17×17. This block diagonalized SDP is solved in less than 1 second, whereas the original 137×137 problem requires roughly 1 minute. The total cost for performing the initial problem reduction using the Newton polytope and block diagonalization based on sign-symmetries, sums up to one tenth of a second, which is negligible compared to the time it would have taken to solve the problem without this pre-processing step. Applying the a posteriori block diagonalization with a zero threshold at 10^{-6} leads to an SDP with total size 103×103, thus revealing some additional redundant monomials.

It may be interesting to compare the results to those obtained in [9]. The algorithm used there decomposes the original 137×137 semidefinite constraint to 14 constraints in sizes ranging from 2×2 up to 11×11. Considering the considerably more involved algebra needed in [9] to derive the decomposition, the results here have to be considered surprisingly successful.

The final example is a practical problem from control theory, and is a reformulation of the rolling disc example in [13]. To compute an upper bound on the $\mathcal{L}_2$ gain γ from u to y of a dynamical system

$$\dot{x}(t) = f(x(t)) + g(x(t))u(t) \quad (7)$$

$$y(x(t)) = h(x(t)) \quad (8)$$

we need to show that there exist a non-negative function $V(x)$ such that the following matrix inequality holds

$$H(x) = \begin{pmatrix} 2\frac{dV}{dx} f(x) + h'(x)h(x) & \frac{dV}{dx} g(x) \\ \left(\frac{dV}{dx} g(x)\right)^T & -\gamma^2 \end{pmatrix} \preceq 0 \quad \forall x. \quad (9)$$

A sufficient condition for this is obtained by introducing a new variable w and require the polynomial $s(x, w) = -w^T H(x) w$ to be a sum-of-squares[6]. For $V(x)$, we use a parameterization of a polynomial, and require it to be a sum-of-squares.

We have $f(x) = \left[x_2, -0.5x_1 - 0.5x_1^3 - 0.5x_2\right]^T$, $g(x) = \left[0, \frac{1}{2}\right]^T$ and $h(x) = x_1$, use $V(x) = c^T z(x)$ where $z(x)$ is a vector of monomials with degree less than or equal to four, and solve a feasibility problem for $\gamma = 1.52$[7].

No symmetries are found (to be expected since we use a completely dense parameterization of $V(x)$), but the Newton reduction is effective on the sum-of-squares constraint related to $-w^T H(x) w \geq 0$, reducing the number of monomials from 35 to 8 using 5 LPs. Solving the problem reveals no problems and the SDP solver returns success. However, for both sum-of-squares constraints, the largest residual is on the order of 10^{-8}, whereas the smallest eigenvalues are around 10^{-10}. Hence, Theorem 4 fails to give us a certificate. The reason is that the decomposition includes monomials that cannot have non-zero coefficients. As a remedy, we let YALMIP apply the proposed a posteriori block diagonalization. After two passes, the SDP constraint related to the sum-of-squares model of $V(x) \geq 0$ is reduced to two 2×2 blocks from an original 6×6 constraint, and the second SDP constraint has been decomposed to one 3×3 and one 1×1 block. More importantly though, the singular parts have been detected and eliminated. The residuals are now 10^{-12}, while the smallest eigenvalues range from 10^{-3} to 10^{-1}. With this numerical data, Theorem 4 gives a certificate that the two polynomials are positive.

[6]A more direct way would be to apply matrix sum-of-squares on $H(x)$ [6]. However, the post-processing capabilities in YALMIP, which we want to illustrate in this example, are currently limited to the scalar case.

[7]If we minimize γ, the solution will always be on the feasibility border, hence leading to a singular Gramian and making it impossible to create a certificate from Theorem 4.

VI. Conclusions

Straightforward practical implementations of algorithms to detect structure and reduce problem size have been presented. Numerical experiments and success reports from users indicate that a dramatic reduction of problem size can be obtained completely automatically without any significant computational cost. Additionally, a post-processing step in the sum-of-squares solver has turned out to be very useful in some scenarios. Although most users use sum-of-squares programs as an approximate indicator, and not as an exact certificate, it is sometimes important to obtain a certifiable sum-of-squares decomposition, despite the lack of infinite precision in the SDP solver. By combining the pre-solve algorithms and repeated post-processing iterations, the possibility for this has been improved considerably.

References

[1] M. D. Choi, T. Y. Lam, and B. Reznick. Sums of squares of real polynomials. In *Proceedings of Symposia in Pure Mathematics*, volume 58, pages 103–126, 1995.

[2] K. Gatermann and P. A. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. *Journal of Pure and Applied Algebra*, 192(1-3):95–128, 2004.

[3] Nicolo Giorgetti. GLPKMEX : A Matlab MEX Interface for the GLPK library. Available from `http://glpkmex.sourceforge.net/`.

[4] D. Henrion and J. B. Lasserre. Gloptipoly: Global optimization over polynomials with Matlab and SeDuMi. *ACM Transactions on Mathematical Software*, 29(2):165–194, 2003.

[5] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. Available from `control.ee.ethz.ch/~joloef/wiki/pmwiki.php`.

[6] J. Löfberg. Block diagonalization of matrix-valued sum-of-squares programs. Technical report LiTH-ISY-R-2845, Department of Electrical Engineering, Linköping University, Sweden, March 2008.

[7] K. Murota, Y. Kanno, M. Kojima, and S. Kojima. A numerical algorithm for block-diagonal decomposition of matrix *-algebras. Technical report METR-2007-52, Department of Mathematical Informatics, The University of Tokyo, September 2007.

[8] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, Pasadena, CA, 2000.

[9] P. A. Parrilo and R. Peretz. An inequality for circle packings proved by semidefinite programming. *Discrete and Computational Geometry*, 31(3):357 – 367, February 2004.

[10] S. Prajna, A. Papachristodoulou, and P. A. Parrilo. SOSTOOLS: a general purpose sum of squares programming solver. In *Proceedings of the* 41^{th} *Conference on Decision & Control*, Las Vegas, USA, 2002.

[11] B. Reznick. Extremal PSD forms with few terms. *Duke Mathematical Journal*, 45(2):363–374, 1978.

[12] B. Reznick. *Sums of Even Powers of Real Linear Forms*, volume 96 of *Memoirs of the American Mathematical Society*. American Mathematical Society, Providence, Rhode Islands, 1992.

[13] J. Sjöberg. Computation of upper bound on the l2-gain for polynomial differential-algebraic systems, using sum of squares decomposition. Technical report LiTH-ISY-R-2806, Department of Electrical Engineering, Linköping University, Sweden, June 2007.

[14] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12(1-4):625–653, 1999. Available from `http://sedumi.mcmaster.ca`.

[15] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, March 1996.

[16] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto. Sparsepop : a sparse semidefinite programming relaxation of polynomial optimization problems. Technical report B-414, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, March 2005.

[17] G. M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics. Springer-Verlag, New York, 1994.