Proceedings of IEEE CDC 2002

Introducing SOSTOOLS: A General Purpose Sum of Squares Programming Solver

Stephen Prajna[*,1], Antonis Papachristodoulou[*,1], and Pablo A. Parrilo[†]

[*] Control and Dynamical Systems, California Inst. of Tech., Pasadena, CA 91125 - USA

[†] Institut für Automatik, ETH Zürich, CH-8092 Zürich - Switzerland

Abstract

SOSTOOLS is a MATLAB toolbox for constructing and solving sum of squares programs. It can be used in combination with semidefinite programming software, such as SeDuMi, to solve many continuous and combinatorial optimization problems, as well as various control-related problems. This paper provides an overview on sum of squares programming, describes the primary features of SOSTOOLS, and shows how SOSTOOLS is used to solve sum of squares programs. Some applications from different areas are presented to show the wide applicability of sum of squares programming in general and SOSTOOLS in particular.

1 Introduction

SOSTOOLS is a free, third-party MATLAB[2] toolbox for solving sum of squares programs. The techniques behind it are based on the sum of squares decomposition for multivariate polynomials [1], which can be efficiently computed using semidefinite programming [16]. SOSTOOLS is developed as a consequence of the recent interest in sum of squares polynomials [14, 1, 13, 9, 10, 7, 6], partly due to the fact that these techniques provide convex relaxations for many hard problems such as global, constrained, and boolean optimization.

In addition to the optimization problems mentioned above, sum of squares polynomials (and hence SOSTOOLS) find applications in several control theory problems. For instance: construction of Lyapunov functions to prove stability of a dynamical system, and computation of tight upper bounds for the structured singular value μ. Examples of these problems, as well as several other optimization-related examples, are provided and solved in the demo files that are distributed together with SOSTOOLS.

This paper is based on the SOSTOOLS User's Guide [12], and is organized as follows. In Section 2 a brief review on sum of squares polynomials is given and the notion of sum of squares programs is introduced. Section 3 describes the main features of SOSTOOLS, including the system requirements. To illustrate how SOSTOOLS is used, a step-by-step example in finding a Lyapunov function for a system with a rational vector field is given in Section 4, and finally some additional application examples are presented in Section 5.

2 Sum of Squares Polynomials and Sum of Squares Programs

A multivariate polynomial $p(x_1, ..., x_n) \triangleq p(x)$ is a sum of squares (SOS, for brevity), if there exist polynomials $f_1(x), ..., f_m(x)$ such that

$$p(x) = \sum_{i=1}^{m} f_i^2(x). \tag{1}$$

It is clear from the definition that the set of sums of squares polynomials on n variables is a convex cone, and it is also true (but not obvious) that it is closed [13]. Condition (1) can be shown to be equivalent to the existence of a positive semidefinite matrix Q, such that

$$p(x) = Z^T(x) Q Z(x), \tag{2}$$

where $Z(x)$ is some properly chosen vector of monomials. Expressing an SOS polynomial using a quadratic form as in (2) has also been referred to as the Gram matrix method [1, 11].

As mentioned in the introduction, sums of squares techniques can be used to provide tractable relaxations for many hard optimization problems. A very general and powerful relaxation methodology, introduced in [9, 10],

[1]Work of the first and second authors was supported by AFOSR MURI "Mathematical Infrastructure for Robust Virtual Engineering" and "Unified Theory for Complex Biological and Engineering Networks", NIH/NIGMS AfCS (Alliance for Cellular Signalling), DARPA "Enlightened multiscale simulation of biochemical networks", the Kitano ERATO Systems Biology Project, and URI "Protecting Infrastructures from Themselves.".

[2]A registered trademark of The MathWorks, Inc.

is based on the *Positivstellensatz*, a central result in real algebraic geometry. The examples in this paper, mostly taken from [9], illustrate the practical application of these relaxation methods.

In this type of relaxations, we are interested in finding polynomials $p_i(x)$, $i = 1, 2, ..., \hat{N}$ and sums of squares $p_i(x)$ for $i = (\hat{N}+1), ..., N$ such that

$$a_{0,j}(x) + \sum_{i=1}^{N} p_i(x) a_{i,j}(x) = 0, \quad \text{for } j = 1, 2, ..., J,$$

where the $a_{i,j}(x)$'s are some given constant coefficient polynomials. Problems of this type will be termed "sum of squares programs." Solutions to SOS programs like the above provide certificates, or *Positivstellensatz refutations*, which can be used to prove the nonexistence of real solutions of systems of polynomial equalities and inequalities (see [10] for details).

To this end, the feasibility problem in SOS programming will be formulated as follows[3]:

SOS Program 1 (Feasibility)

Find

polynomials $p_i(x)$, for $i = 1, 2, ..., \hat{N}$
sums of squares $p_i(x)$, for $i = (\hat{N}+1), ..., N$

such that

$$a_{0,j}(x) + \sum_{i=1}^{N} p_i(x) a_{i,j}(x) = 0 \quad \textit{for } j = 1, 2, ..., \hat{J}, \quad (3)$$

$$a_{0,j}(x) + \sum_{i=1}^{N} p_i(x) a_{i,j}(x) \ \textit{are SOS } (\geq 0)^4, \quad \textit{for } j = (\hat{J}+1), ..., J. \quad (4)$$

In this formulation, $a_{i,j}(x)$'s are some scalar constant coefficient polynomials. The $p_i(x)$'s will be termed *SOS program variables*, and the constraints (3)–(4) are termed *SOS program constraints*. It is obvious that the same program can be formulated in terms of constraints (3) only, by introducing some extra sums of squares as slack program variables. But avoiding this will in most cases make the problem statement clearer.

As $f(x)$ being an SOS naturally implies $f(x) \geq 0$, and since many problems are more naturally formulated using inequalities, we will call the constraints (4) "inequality constraints", and denote them by ≥ 0. We remind the reader of the equivalence between "nonnegativity" and "sum of squares" in the cases of univariate, multivariate quadratic, and binary quartic polynomials (see [13] and the references therein). Nevertheless, it should be noted that in the general multivariate case, $f(x) \geq 0$ in the usual sense does not necessarily imply that $f(x)$ is an SOS. The condition that $f(x)$ is an SOS is stricter, yet more computationally tractable, than $f(x) \geq 0$ [9, 13].

Besides feasibility, there exists another class of problems in sum of squares programming, which involves optimization of an objective function that is linear in the coefficients of $p_i(x)$'s. The general form of these optimization problems is as follows:

SOS Program 2 (Optimization)

Minimize the linear objective function

$$w^T c,$$

where c is a vector formed from the (unknown) coefficients of

polynomials $p_i(x)$, for $i = 1, 2, ..., \hat{N}$
sum of squares $p_i(x)$, for $i = (\hat{N}+1), ..., N$

such that

$$a_{0,j}(x) + \sum_{i=1}^{N} p_i(x) a_{i,j}(x) = 0 \quad \textit{for } j = 1, 2, ..., \hat{J}, \quad (5)$$

$$a_{0,j}(x) + \sum_{i=1}^{N} p_i(x) a_{i,j}(x) \ \textit{are SOS } (\geq 0)^4, \quad \textit{for } j = (\hat{J}+1), ..., J. \quad (6)$$

where in this formulation w is the vector of weighting coefficients in the linear objective function.

Both the feasibility and optimization problems as formulated above are quite general, and in specific cases reduce to well-known problems. In particular, notice that if all the unknown polynomials p_i are restricted to be constants, and the $a_{i,j}(x)$'s are quadratic forms, then we exactly recover the standard LMI problem formulation. Nevertheless, these extra degrees of freedom are a bit illusory, as every SOS program can be exactly converted to an equivalent semidefinite program [1]. For several reasons, the problem specification outlined above has definite practical and methodological advantages, and establishes a useful framework within which many specific problems can be solved, as we will see later in Sections 4 and 5.

3 SOSTOOLS in a Nutshell

At the present time, sum of squares programs are handled by reformulating them as semidefinite programs

[3]Another way of formulating sum of squares programs can be found in the SOSTOOLS manual.

[4]Whenever constraint $f(x) \geq 0$ is encountered in an SOS program, it should always be interpreted as "$f(x)$ is an SOS".

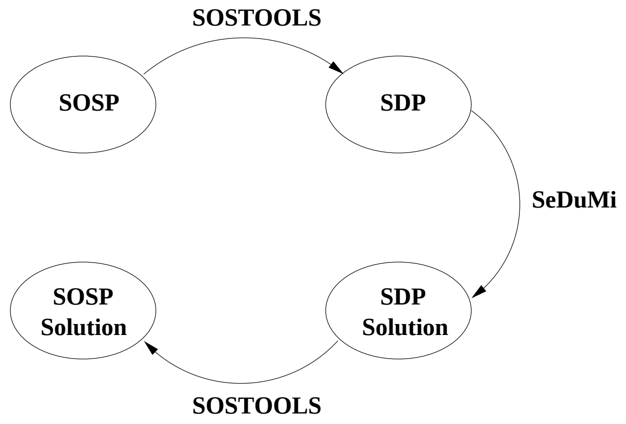

Figure 1: Diagram depicting relations between SOS program (SOSP), semidefinite program (SDP), SOSTOOLS, and SeDuMi.

(SDPs), which in turn are solved efficiently e.g. using interior point methods. Several commercial as well as non-commercial software packages are available for solving SDPs. While the conversion from SOS programs to SDPs can be manually performed for small size instances or tailored for specific problem classes, such a conversion can be quite cumbersome to perform in general. It is therefore desirable to have a computational aid that automatically performs this conversion for general SOS programs. This is exactly where SOSTOOLS comes to play[5]. It automates the conversion from SOS program to SDP, calls the SDP solver, and converts the SDP solution back to the solution of the original SOS program. At present, it uses another free MATLAB add-on called SeDuMi [15] as the SDP solver. This whole process is depicted in Figure 1.

All polynomials in SOSTOOLS are implemented as symbolic objects, making full use of the capabilities of the MATLAB Symbolic Math Toolbox. This gives to the user the benefit of being able to do all polynomial manipulations using the usual arithmetic operators: `+`, `-`, `*`, `/`, `^`; as well as differentiation, integration, point evaluation, etc. In addition, this provides the possibility of interfacing with the Maple[6] symbolic engine and library, which is very advantageous.

The user interface has been designed to be as simple, easy to use, and transparent as possible. The user creates an SOS program by declaring SOS program variables (the $p_i(x)$'s in Section 2), adding SOS program constraints, setting the objective function, and so on. After the program is created, the user calls one function to run the solver. Finally, the user retrieves solutions to the SOS program using another function. These steps will be presented in greater details in Section 4.

[5]A related recent software is GloptiPoly [4], which solves global optimization problems over polynomials, based on the method in [6].

[6]A registered trademark of Waterloo Maple Inc.

SOSTOOLS is available for free under the GNU General Public License. The software and its user's manual can be downloaded from `http://www.cds.caltech.edu/sostools` or `http://www.aut.ee.ethz.ch/~parrilo/sostools`. It requires MATLAB version 6.0 or later, SeDuMi [15] version 1.05, and the Symbolic Math Toolbox version 2.1.2. SOSTOOLS can be easily run on a UNIX workstation or on a Windows PC desktop, or even a laptop. It utilizes the MATLAB sparse matrix representation for good performance and to reduce the amount of memory needed. To give an illustrative figure of the computational load, all examples in Sections 4 and 5, except for the μ upper bound example, are solved in less than 10 seconds by SOSTOOLS running on a PC with Intel Celeron 700 MHz processor and 96 MBytes of RAM. Even the μ upper bound example is solved in less than 25 seconds using the same system.

4 Solving Sum of Squares Programs with SOSTOOLS

SOSTOOLS can solve two kinds of SOS programs: the feasibility and optimization problems, as formulated in Section 2. To define and solve an SOS program using SOSTOOLS, the user simply needs to follow these steps:

1. Initialize a SOS program, declare the SOS program variables.
2. Define SOS program constraints
3. Set objective function (for optimization problems)
4. Call solver
5. Get solutions.

In this section, we will give a step-by-step example on how to create and solve an SOS program. For this purpose, we will consider the search for a Lyapunov function of a nonlinear dynamical system. The presentation given here is not meant to be comprehensive; readers are referred to the SOSTOOLS User's Manual for a detailed explanation on each function used.

The Lyapunov stability theorem (see e.g. [5]) has been a cornerstone of nonlinear system analysis for several decades. In principle, the theorem states that an equilibrium of the system $\dot{x} = f(x)$ (with the equilibrium assumed to be at the origin) is stable if there exists a positive definite function $V(x)$ such that its derivative along the system trajectories is non-positive. For our example, consider the system

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -x_1^3 - x_1 x_3^2 \\ -x_2 - x_1^2 x_2 \\ -x_3 - \frac{3x_3}{x_3^2+1} + 3x_1^2 x_3 \end{bmatrix} \triangleq f(x), \qquad (7)$$

which has its only equilibrium at the origin. Notice that the linearization of (7) has an eigenvalue equal to zero, and therefore cannot be used to analyze the local stability of the equilibrium. Assume that we are interested to see whether a quadratic Lyapunov function $V(x)$ with no cross-term monomials can be used for proving stability of the system. For $V(x)$ to be a Lyapunov function, it must satisfy

$$\begin{aligned} V(x) - \epsilon(x_1^2 + x_2^2 + x_3^2) &\geq 0, \\ -\frac{\partial V}{\partial x_1}\dot{x}_1 - \frac{\partial V}{\partial x_2}\dot{x}_2 - \frac{\partial V}{\partial x_3}\dot{x}_3 &\geq 0. \end{aligned} \tag{8}$$

The first inequality, with ϵ being any constant greater than zero, is needed to guarantee positive definiteness of $V(x)$, whereas the second inequality will guarantee that the derivative of $V(x)$ along the system trajectories is non-positive. However, notice that $\dot{x}_3$ is a rational function, and therefore (8) is not a valid SOS program constraint. But since $x_3^2 + 1 > 0$ for any x_3, we can equivalently reformulate (8) as

$$-(x_3^2+1)\left(\frac{\partial V}{\partial x_1}\dot{x}_1 - \frac{\partial V}{\partial x_2}\dot{x}_2 - \frac{\partial V}{\partial x_3}\dot{x}_3\right) \geq 0.$$

Thus, we have the following (we choose $\epsilon = 1$):

SOS Program 3 *Find a polynomial*

$$V(x) = a_1x_1^2 + a_2x_2^2 + a_3x_3^2, \tag{9}$$

(where the a_i's are the unknown decision variables), such that

$$V(x) - (x_1^2 + x_2^2 + x_3^2) \geq 0, \tag{10}$$

$$-(x_3^2+1)\left(\frac{\partial V}{\partial x_1}\dot{x}_1 - \frac{\partial V}{\partial x_2}\dot{x}_2 - \frac{\partial V}{\partial x_3}\dot{x}_3\right) \geq 0. \tag{11}$$

We will now show how to search for Lyapunov function using SOSTOOLS. We start by defining the independent variables x_1, x_2, x_3 as symbolic objects and collecting them in a vector.

```
>> syms x1 x2 x3;
>> vars = [x1; x2; x3];
```

Next, we define the decision variables, i.e., the a_i in (9):

```
>> syms a1 a2 a3;
>> decvars = [a1; a2; a3];
```

Then the vector field (7) can be defined symbolically as follows.

```
>> f = [ -x1^3-x1*x3^2;
      -x2-x1^2*x2;
      -x3-3*x3/(x3^2+1)+3*x1^2*x3];
```

At this point, our SOS program is ready to be constructed. An empty SOS program is initialized by the function `sosprogram.m`, with the list of independent and decision variables as its arguments.

```
>> Program1 = sosprogram(vars,decvars);
```

The next step is to define the SOS program constraints (10)–(11). Constraint (10) is defined using:

```
>> V = a1*x1^2 + a2*x2^2 + a3*x3^2 ;
>> Program1 = sosineq(Program1, V-(x1^2+x2^2+x3^2));
```

The first command simply defines V, and the second command adds the inequality constraint to the already defined SOS program.

In the same manner, Constraint (11) can be added to our SOS program. This is performed by typing

```
>> Vdot = diff(V,x1)*f(1) + diff(V,x2)*f(2) + ...
>>        diff(V,x3)*f(3);
>> Program1 = sosineq(Program1, -Vdot*(x3^2+1));
```

where the first command line explicitly constructs the time derivative of V, and the second one adds the inequality constraint (11).

The SOS feasibility program has now been completely created, and can be solved using the following command:

```
>> Program1 = sossolve(Program1);
```

The solution $V(x)$ can then be retrieved by typing

```
>> SOLV = sosgetsol(Program1,V)

SOLV =

7.1525*x1^2+5.7870*x2^2+2.1434*x3^2
```

As we see above, in this example SOSTOOLS gives the following $V(x)$ as a Lyapunov function for our system:

$$V(x) = 7.1525x_1^2 + 5.7870x_2^2 + 2.1434x_3^2. \tag{12}$$

Notice that the whole problem is coded and solved with only a handful of MATLAB commands.

5 More Applications

SOSTOOLS comes with several demo files containing applications from different domains. We describe some of these applications here; many other ones are described in the SOSTOOLS User's Manual.

5.1 Bound on Global Extremum

Consider the problem of finding a lower bound for the global minimum of a function $f(x), x \in \mathbb{R}^n$. This problem is addressed in [14], where an SOS-based approach was first used. A relaxation method can be formulated as follows. Suppose that there exists a strictly positive $r(x)$ such that $r(x)(f(x)-\gamma)$ is an SOS, then we know that $f(x) \geq \gamma$, for every $x \in \mathbb{R}^n$.

In this example we will use the Goldstein-Price test function, which is given by

$$\begin{aligned} f(x) = & [1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2 ... \\ & +6x_1x_2+3x_2^2)][30+(2x_1-3x_2)^2(18-32x_1 ... \\ & +12x_1^2+48x_2-36x_1x_2+27x_2^2)] \end{aligned}$$

We start with $r(x)=1$. If needed, we can use higher order $r(x)$ later. The SOS program for this problem is

SOS Program 4 *Minimize $-\gamma$, such that $f(x)-\gamma$ is a sum of squares (≥ 0).*

The demo file `demo3.m` contains the MATLAB code for this problem. The optimal value of γ, as given by SOSTOOLS, is $\gamma_{\text{opt}} = 3$. This is in fact the global minimum of $f(x)$, which is achieved at $x = (0, -1)$.

5.2 Upper Bound of Structured Singular Value

Now we will show how SOSTOOLS can be used to compute an upper bound for the structured singular value μ, a crucial object in robust control theory (see e.g. [2, 8]). The following conditions can be derived from Proposition 8.25 of [2] and Theorem 6.1 of [9]. Given a matrix $M \in \mathbb{C}^{n \times n}$ and structured scalar uncertainties $\Delta = \text{diag}(\delta_1, \delta_2, ..., \delta_n)$, $\delta_i \in \mathbb{C}$, the structured singular value $\mu(M, \Delta)$ is less than γ, if there exist $Q_i \geq 0 \in \mathbb{R}^{2n \times 2n}$ and $r_{ij} \geq 0$ satisfying

$$-\sum_{i=1}^{n} Q_i(x)A_i(x) - \sum_{1 \leq i<j \leq n} r_{ij}A_i(x)A_j(x) + I(x) \geq 0, \tag{13}$$

where $x \in \mathbb{R}^{2n}$,

$$Q_i(x) = x^TQ_ix, \tag{14}$$

$$I(x) = -\left(\sum_{i=1}^{2n} x_i^2\right)^2, \tag{15}$$

$$A_i(x) = x^TA_ix, \tag{16}$$

$$A_i = \begin{bmatrix} \text{Re}(H_i) & -\text{Im}(H_i) \\ \text{Im}(H_i) & \text{Re}(H_i) \end{bmatrix}, \tag{17}$$

$$H_i = M^*e_i^*e_iM - \gamma^2e_i^*e_i, \tag{18}$$

and e_i is the i-th unit vector in $\mathbb{C}^n$.

Thus, the SOS program for this problem can be formulated as follows:

SOS Program 5 *Choose a fixed value of γ. For $I(x)$ and $A_i(x)$ as described in (15)–(18), find sums of squares*

$$Q_i(x) = x^TQ_ix, \quad i = 1, ..., 2n, \tag{19}$$

$$r_{ij} \geq 0, \quad 1 \leq i < j \leq 2n, \tag{20}$$

such that (13) is satisfied.

The optimal value of γ can be found for example by bisection. In `demo5.m`, we consider the following M (from [8]):

$$M = UV^*, \qquad U = \begin{bmatrix} a & 0 \\ b & b \\ c & jc \\ d & f \end{bmatrix}, \qquad V = \begin{bmatrix} 0 & a \\ b & -b \\ c & -jc \\ -jf & -d \end{bmatrix},$$

with $a = \sqrt{2/\alpha}$, $b = c = 1/\sqrt{\alpha}$, $d = -\sqrt{\beta/\alpha}$, $f = (1+j)\sqrt{1/(\alpha\beta)}$, $\alpha = 3+\sqrt{3}$, $\beta = \sqrt{3}-1$. It is known that $\mu(M, \Delta) \approx 0.8723$. Using `demo5.m`, we can prove that $\mu(M, \Delta) < 0.8724$.

5.3 MAX CUT

We will next consider the MAX CUT problem. MAX CUT is the problem of partitioning nodes in a graph into two disjoint sets V_1 and V_2, such that the weighted number of nodes that have an endpoint in V_1 and the other in V_2 is maximized. This can be formulated as a boolean optimization problem

$$\max_{x_i \in \{-1,1\}} \frac{1}{2} \sum_{i,j} w_{ij}(1 - x_ix_j),$$

or equivalently as a constrained optimization

$$\max_{x_i^2=1} f(x) \triangleq \max_{x_i^2=1} \frac{1}{2} \sum_{i,j} w_{ij}(1 - x_ix_j).$$

Here w_{ij} is the weight of the edge connecting nodes i and j. For example we can take $w_{ij} = 0$ if nodes i and j are not connected, and $w_{ij} = 1$ if they are connected. If node i belongs to V_1, then $x_i = 1$, and conversely $x_i = -1$ if node i is in V_2.

A sufficient condition for $\max_{x_i^2=1} f(x) \leq \gamma$ is as follows. Assume that our graph contains n nodes. Given $f(x)$ and γ, then $\max_{x_i^2=1} f(x) \leq \gamma$ if there exist a positive definite sum of squares $p_1(x)$ and polynomials $p_2(x), ..., p_{n+1}(x)$ such that

$$p_1(x)(\gamma - f(x)) + \sum_{i=1}^{n} \left(p_{i+1}(x)(x_i^2 - 1)\right) \geq 0. \tag{21}$$

This can be proved by contradiction. Suppose there exists $x \in \{-1, 1\}^n$ such that $f(x) > \gamma$. Then the first term in (21) will be negative and the terms under summation will be zero, thus we have a contradiction. In relation to this, it can be noted that the SDP relaxation of Goemans and Williamson [3] is a special case of (21), obtained by taking the $p_i(x)$'s to be constants (i.e., degree zero).

For the demo file, we consider the 5-cycle, i.e., a graph with 5 nodes and 5 edges forming a closed chain. The number of cuts for 5-cycle is given by

$$f(x) = 2.5 - .5x_1x_2 - .5x_2x_3 - .5x_3x_4 - .5x_4x_5 - .5x_5x_1. \tag{22}$$

Our SOS program is as follows.

SOS Program 6 *Choose a fixed value for γ. For $f(x)$ given in (22), find*

sum of squares $p_1(x) = \begin{bmatrix}1\\x\end{bmatrix}^T Q \begin{bmatrix}1\\x\end{bmatrix}$

polynomials $p_{i+1}(x)$ *of degree 2,* $i = 1, ..., n$

such that (21) is satisfied.

Using `demo6.m`, we can show that $f(x) \leq 4$. Four is indeed the maximum number of cuts for 5-cycle.

6 Concluding Remarks

We have presented a brief overview on sum of squares polynomials and introduced the notion of sum of squares programs. The main features of SOSTOOLS, a general purpose sum of squares programming solver, have been subsequently described.

We have further shown a detailed example on how SOSTOOLS can be used to solve control-related sum of squares programs. From different domains we have presented several applications, such as computing bound for global optimization, computing upper bounds for μ, and MAX CUT combinatorial optimization, to illustrate the wide applicability of sum of squares programming and SOSTOOLS.

SOSTOOLS is under active development. Upcoming versions of SOSTOOLS will, among other improvements, provide the user with the dual solutions of the semidefinite program to check if the optimal solution to an optimization problem is achieved. Additionally, special structure in the input polynomials, such as sparsity, symmetries, and reduction over polynomial ideals will be fully exploited in order to speed up the computation and allow the possibility of solving larger scale problems.

References

[1] M. D. Choi, T. Y. Lam, and B. Reznick. Sum of squares of real polynomials. *Proceedings of Symposia in Pure Mathematics*, 58(2):103–126, 1995.

[2] G. E. Dullerud and F. Paganini. *A Course in Robust Control Theory: A Convex Approach*. Springer-Verlag NY, 2000.

[3] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[4] D. Henrion and J. B. Lasserre. GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi. In *Proceedings of the 41st IEEE Conf. on Decision and Control*, 2002. Available at `http://www.laas.fr/~henrion/software/gloptipoly`.

[5] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, Inc., second edition, 1996.

[6] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, 11(3):796–817, 2001.

[7] Y. Nesterov. Squared functional systems and optimization problems. In J. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 405–440. Kluwer Academic Publishers, 2000.

[8] A. Packard and J. C. Doyle. The complex structured singular value. *Automatica*, 29(1):71–109, 1993.

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

[10] P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. To appear in *Mathematical Programming*, 2001.

[11] V. Powers and T. Wörmann. An algorithm for sums of squares of real polynomials. *Journal of Pure and Applied Linear Algebra*, 127:99–104, 1998.

[12] S. Prajna, A. Papachristodoulou, and P. A. Parrilo. SOSTOOLS – Sum of Squares Optimization Toolbox, User's Guide. Available at `http://www.cds.caltech.edu/sostools` and `http://www.aut.ee.ethz.ch/~parrilo/sostools`, 2002.

[13] B. Reznick. Some concrete aspects of Hilbert's 17th problem. In *Contemporary Mathematics*, volume 253, pages 251–272. American Mathematical Society, 2000.

[14] N. Z. Shor. Class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987.

[15] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Available at `http://fewcal.kub.nl/sturm/software/sedumi.html`.

[16] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.