SOSTOOLS and its Control Applications

Stephen Prajna[1], Antonis Papachristodoulou[1], Peter Seiler[2], and Pablo A. Parrilo[3]

[1] Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA. E-mail: {prajna,antonis}@cds.caltech.edu
[2] Mechanical and Industrial Engineering Dept., University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. E-mail: pseiler@uiuc.edu
[3] Automatic Control Laboratory, Swiss Federal Institute of Technology, CH-8092 Zürich, Switzerland. E-mail: parrilo@control.ee.ethz.ch

Summary. In this chapter we present SOSTOOLS, a third-party MATLAB toolbox for formulating and solving sum of squares optimization problems. Sum of squares optimization forms a basis for formulating convex relaxations to computationally hard problems such as some that appear in systems and control. Currently, sum of squares programs are solved by casting them as semidefinite programs, which can in turn be solved using interior-point based numerical methods. SOSTOOLS helps this translation in such a way that the underlying computations are abstracted from the user. Here we give a brief description of the toolbox, its features and capabilities (with emphasis on the recently added ones), as well as show how it can be applied to solving problems of interest in systems and control.

1 Introduction

There has been a great interest recently in sum of squares polynomials and sum of squares optimization [34, 1, 31, 17, 18, 13, 11], partly due to the fact that these techniques provide convex polynomial time relaxations for many hard problems such as global, constrained, and Boolean optimization, as well as various problem in systems and control. The observation that the sum of squares decomposition can be computed efficiently using semidefinite programming [17] has initiated the development of software tools that facilitate the formulation of the semidefinite programs from their sum of squares equivalents. One such a software is SOSTOOLS [24, 25, 26], a free third-party MATLAB[4] toolbox for solving sum of squares programs.

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

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

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

It follows from the definition that the set of sums of squares polynomials in n variables is a convex cone. The existence of an SOS decomposition (1) can be shown to be equivalent to the existence of a positive semidefinite matrix Q such that

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

where $Z(x)$ is the vector of monomials of degree less than or equal to $\deg(p)/2$, i.e., its entries are of the form $x^\alpha = x_1^{\alpha_1} \dots x_n^{\alpha_n}$, where the α's are nonnegative integers and $\alpha_1 + \ldots + \alpha_n \leq \deg(p)/2$. Expressing an SOS polynomial as a quadratic form in (2) has also been referred to as the Gram matrix method [1, 20]. The decomposition (1) can be easily converted into (2) and vice versa. This equivalence makes an SOS decomposition computable using semidefinite programming, since finding a symmetric matrix $Q \succeq 0$ subject to the affine constraint (2) is nothing but a semidefinite programming problem.

It is clear that a sum of squares polynomial is *globally nonnegative*. This is a property of SOS polynomials that is crucial in many control applications, where we replace various polynomial inequalities with SOS conditions. However, it should be noted that not all nonnegative polynomials are necessarily sums of squares. The equivalence between nonnegativity and sum of squares is only guaranteed in three cases, those of univariate polynomials of any even degree, quadratic polynomials in any number of indeterminates, and quartic polynomials in three variables [31]. Indeed nonnegativity is NP-hard to test [12], whereas the SOS conditions are polynomial time verifiable through solving appropriate semidefinite programs. Despite this, in many cases we are able to obtain solutions to computational problems that are otherwise at the moment unsolvable, simply by replacing the nonnegativity conditions with SOS conditions.

A sum of squares program is a convex optimization problem of the following form:

$$\text{Minimize } \sum_{j=1}^{J} w_j c_j \tag{3}$$

subject to

$$a_{i,0}(x) + \sum_{j=1}^{J} a_{i,j}(x) c_j \text{ is SOS, for } i = 1, ..., I, \tag{4}$$

where the c_j's are the scalar real decision variables, the w_j's are given real numbers, and the $a_{i,j}(x)$ are given polynomials (with fixed coefficients). See also another equivalent canonical form of SOS programs in [24, 25]. While the conversion from SOS programs to semidefinite programs (SDPs) can be manually performed for small size instances or tailored for specific problem

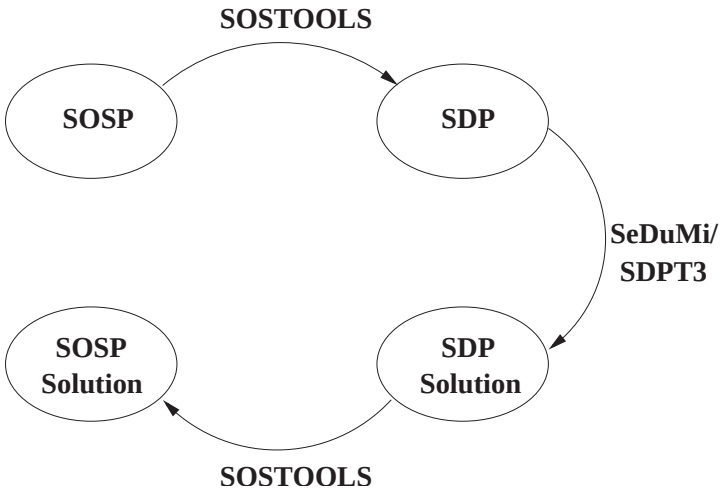

Fig. 1. Diagram depicting relations between sum of squares program (SOSP), semidefinite program (SDP), SOSTOOLS, and SeDuMi or SDPT3.

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 what SOSTOOLS is useful for. 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 (see Figure 1). The current version of SOSTOOLS uses either SeDuMi [35] or SDPT3 [37], both of which are free MATLAB add-ons, as the SDP solver. The user interface of SOSTOOLS has been designed to be as simple, as easy to use, and as transparent as possible, while keeping a large degree of flexibility.

In addition to the optimization problems mentioned above (a related recent software in this regard is GloptiPoly [8], which solves global optimization problems over polynomials, based on the method in [11]), sum of squares polynomials and SOSTOOLS find applications in several control theory problems. These problems include stability analysis of nonlinear, hybrid, and time-delay systems [17, 15, 23, 14], robustness analysis [17, 15, 23], estimation of domain of attraction [17, 33], LPV analysis and synthesis [39], nonlinear synthesis [9, 28, 27], safety verification [22], and model validation [21]. Other areas in which SOSTOOLS is applicable are, for instance, geometric theorem proving [19] and quantum information theory [2].

In Section 2, we present the main features of SOSTOOLS and point out improvements that have been made in the user interface, custom-made functions, and modularity with respect to the choice of semidefinite programming solver. Some control oriented application examples will then be provided in Section 3. In particular, we will consider nonlinear stability analysis, parametric robustness analysis, analysis of time-delay systems, safety verification, and

nonlinear controller synthesis. We show how sum of squares programs corresponding to these problems can be formulated, which in turn can be solved using SOSTOOLS.

2 SOSTOOLS Features

It has been mentioned in the introduction that the main purpose of SOSTOOLS is to efficiently transform SOS programs of the form (3)–(4) into semidefinite programs (SDPs), which can then be solved using an SDP solver. The solution is then retrieved from the solver and translated into the original polynomial variables. In this way, the details of the reformulation are abstracted from the user, who can work at the polynomial object level.

Polynomial variables in SOSTOOLS can be defined in two different ways: as a symbolic object through the use of the MATLAB Symbolic Math Toolbox, or as a custom-built polynomial object using the integrated Multivariate Polynomial Toolbox. The former option provides to the user the benefit of making use of all the features in the Symbolic Math Toolbox, which range from simple arithmetic operations to differentiation, integration and polynomial manipulation. On the other hand, the Multivariate Polynomial Toolbox allows users that do not have access to the Symbolic Math Toolbox to use SOSTOOLS. Some basic polynomial manipulation functions are also provided in this toolbox.

To define and solve an SOS program using SOSTOOLS, the user simply needs to follow these steps:

1. Initialize the SOS program.
2. Declare the SOS program variables.
3. Define the SOS program constraints, namely Eq. (4).
4. Set the objective function, namely Eq. (3).
5. Call solver.
6. Get solutions.

We will give a short illustration of these steps in Section 2.1. However, we will not entail in a discussion of how each of these steps is performed nor the SOSTOOLS commands relevant to this. A detailed description can be found in the SOSTOOLS user's guide [25].

In many cases, the SOS program we wish to solve have certain structural properties, such as sparsity, symmetry, and so on. The formulation of the SDP in this case should take into account these properties. This will not only reduce significantly the computational burden of solving it, as the size of the SDP will reduce considerably, but also it removes numerical ill-conditioning. With regard to this, provision has been taken in SOSTOOLS for exploitation of polynomial sparsity when formulating the SDP. The details will be described in Section 2.2.

The frequent use of certain sum of squares programs, such as those corresponding to

- finding the sum of squares decomposition of a polynomial,
- finding lower bounds on polynomial minima, and
- constructing Lyapunov functions for systems with polynomial vector fields

are reflected in the inclusion of customized functions in SOSTOOLS. Some of these customized functions will be discussed at the end of the section.

2.1 Formulating Sum of Squares Programs

In the original release of SOSTOOLS, polynomials were implemented solely 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 operations such as differentiation, integration, point evaluation, etc. In addition, it provides the possibility of interfacing with the Maple[5] symbolic engine and the Maple library (which is very advantageous). On the other hand, this prohibited those without access to the Symbolic Toolbox (such as those using the student edition of MATLAB) from using SOSTOOLS. In the current SOSTOOLS release, the user has the option of using an alternative custom-built polynomial object, along with some basic polynomial manipulation methods to represent and manipulate polynomials.

Using the Symbolic Toolbox, a polynomial is created by declaring its independent variables as symbolic variables in the symbolic toolbox and constructing it in a similar way. For example, to create the polynomial $p(x, y) = 2x^2 + 3xy + 4y^4$, one declares the variables x and y by typing

```
>> syms x y
```

and constructs $p(x, y)$ as follows:

```
>> p = 2*x^2 + 3*x*y + 4*y^4
```

In a similar manner, one can define this polynomial using the Multivariate Polynomial Toolbox, a freely available toolbox that has been integrated in SOSTOOLS for constructing and manipulating multivariate polynomials. Polynomial variables are created with the `pvar` command. Here the same polynomial can be constructed by declaring first the variables:

```
>> pvar x y
```

Note that `pvar` is used to replace the command `syms`. New polynomial objects can now be created from these variables, and manipulated using standard addition, multiplication, and integer exponentiation functions:

```
>> p = 2*x^2 + 3*x*y + 4*y^4
```

[5] A registered trademark of Waterloo Maple Inc.

Matrices of polynomials can also be created from polynomials using horizontal/vertical concatenation and block diagonal augmentation. A few additional operations exist in this initial version of the toolbox such as trace, transpose, determinant, differentiation, logical equal, and logical not equal.

The input to the SOSTOOLS commands can be specified using either the symbolic objects or the new polynomial objects. There are some minor variations in performance depending on the degree/number of variables of the polynomials, due the fact that the new implementation always keeps an expanded internal representation, but for most reasonable-sized problems the difference is minimal.

For an illustration, let us now consider the problem of finding a lower bound for the global minimum of the Goldstein-Price test function [5]

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

The SOS program for this problem is

$$\begin{aligned} &\text{Minimize } -\gamma, \\ &\text{such that} \\ &\qquad (f(x) - \gamma) \text{ is SOS.} \end{aligned}$$

It is clear that any value of γ for which $f(x) - \gamma$ is an SOS will serve as a lower bound for the polynomial, since in that case $f(x) - \gamma$ is nonnegative. By maximizing γ (or equivalently, minimizing $-\gamma$), a tighter lower bound can be computed.

In this example, the SOS program is initialized and the decision variable γ is declared using the following commands (assuming that we use the polynomial objects)

```
>> pvar x1 x2 gam;
>> prog = sosprogram([x1; x2],gam);
```

The function $f(x)$ is constructed as follows

```
>> f1 = x1+x2+1;
>> f2 = 19-14*x1+3*x1^2-14*x2+6*x1*x2+3*x2^2;
>> f3 = 2*x1-3*x2;
>> f4 = 18-32*x1+12*x1^2+48*x2-36*x1*x2+27*x2^2;
>> f = (1+f1^2*f2)*(30+f3^2*f4);
```

Then the SOS program constraint "$f(x)-\gamma$ is SOS" and the objective function are set, and the SOS program is solved using the following commands

```
>> prog = sosineq(prog,f-gam);
>> prog = sossetobj(prog,-gam);
>> prog = sossolve(prog);
```

The optimal lower bound is then retrieved by

```
>> gam = sosgetsol(prog,gam);
```

The result given by SOSTOOLS is $\gamma = 3$. This is in fact the global minimum of $f(x)$, achieved at $x_1 = 0$, $x_2 = -1$. The same example can also be solved using the customized function `findbound` as follows

```
>> [gam,vars,xopt] = findbound(f);
```

2.2 Exploiting Sparsity

The complexity of computing a sum of squares decomposition for a polynomial $p(x)$ depends on two factors: the dimension of the independent variable x and the degree of the polynomial. As mentioned previously, when $p(x)$ has special structural properties, the computation effort can be notably simplified through the reduction of the size of the semidefinite program, removal of degeneracies, and better numerical conditioning.

The first type of simplification can be performed when $p(x)$ is sparse. The notion of sparseness for multivariate polynomials is stronger than the one commonly used for matrices. While in the matrix case this word usually means that many coefficients are zero, in the polynomial case the specific vanishing pattern is also taken into account. This idea is formalized by the concept of *Newton polytope* [36], defined as the convex hull of the set of exponents, considered as vectors in $\mathbb{R}^n$. It was shown by Reznick in [30] that $Z(x)$ need only contain monomials whose squared degrees are contained in the convex hull of the degrees of monomials in $p(x)$. Consequently, for sparse $p(x)$ the size of the vector $Z(x)$ and matrix Q appearing in the sum of squares decomposition can be reduced which results in a decrease of the size of the semidefinite program.

Since the initial version of SOSTOOLS, Newton polytopes techniques have been available via the optional argument `'sparse'` to the function `sosineq`, and in the new release, the support for sparse polynomials has been improved. SOSTOOLS takes the sparse structure into account, and chooses an appropriate set of monomials for the sum of squares decompositions with the convex hull computation performed either by the native MATLAB command `convhulln` (which is based on the software QHULL), or the specialized external package CDD [3]. Special care is taken with the case when the set of exponents has lower affine dimension than the number of variables (this case occurs for instance for homogeneous polynomials, where the sum of the degrees is equal to a constant), in which case a projection to a lower dimensional space is performed prior to the convex hull computation.

A special sparsity structure that appears frequently in robust control theory when considering, for instance, Lyapunov function analysis for linear systems with parametric uncertainty (see Section 3.2), is called the multipartite structure (see [26] for a definition). Such a structure also appears when considering infinitely constrained linear matrix inequalities (LMIs) of the form:

$$\text{Minimize } \sum_{j=1}^{J} w_j c_j$$

subject to

$$A_0(x) + \sum_{j=1}^{J} A_j(x) c_j \succeq 0 \quad \forall x \in \mathbb{R}^n,$$

where the c_j's and w_j's are again the decision variables and given real numbers, respectively, and the $A_j(x)$'s are given symmetric polynomial matrices. By introducing a new set of indeterminates y, defining $p_j(x,y) = y^T A_j(x) y$, and replacing the positive semidefiniteness condition $A_0(x) + \sum A_j(x) c_j \succeq 0$ by sum of squares condition

$$p(x,y) = p_0(x,y) + \sum_{j=1}^{J} p_j(x,y) c_j \quad \text{is SOS}, \tag{5}$$

obviously the original LMIs can be computationally relaxed to an SOS program (a positive semidefinite matrix $A(x)$ for which $y^T A(x) y$ is an SOS is called an SOS matrix in [4]). The resulting polynomial (5) has the multipartite structure (in this case it is actually bipartite): its independent variable can be naturally partitioned into two sets x and y, where the degree of the polynomial in x can be arbitrary, and the degree in y is always equal to two. What distinguishes this case from a general sparsity, is that the Newton polytope of $p(x,y)$ is the *Cartesian product* of the individual Newton polytopes corresponding to the blocks of variables. Because of this structure, only monomials of the form $x^\alpha y_i$ will appear in the monomial vector $Z(x,y)$. The current version of SOSTOOLS provides a support for the multipartite structure via the argument `'sparsemultipartite'` to the function `sosineq`, by computing a reduced set of monomials in an efficient manner.

To illustrate the benefit of using the sparse multipartite option, consider the problem of checking whether a polynomial matrix inequality

$$F(x) = F^T(x) \succeq 0 \quad \forall x \in \mathbb{R}^n$$

holds, where $F \in \mathbb{R}[x]^{m \times m}$. A sufficient test for positive semidefiniteness of $F(x)$ is obtained by showing that the bipartite polynomial $\mathbf{y}^T F(x) \mathbf{y}$ is a sum of squares (equivalently, showing that $F(x)$ is an SOS matrix). We denote the degree of F by d. For various values of (m,n,d), the sizes of the resulting semidefinite programs are depicted in Table 1.

2.3 Customized Functions

The SOSTOOLS package includes several "ready-made" customized functions that solve specific problems directly, by internally reformulating them as SOS

(m, n, d)	Without multipartite option	With multipartite option
$(3, 2, 2)$	15×15, 90 constraints	9×9, 36 constraints
$(4, 2, 2)$	21×21, 161 constraints	12×12, 60 constraints
$(3, 3, 2)$	21×21, 161 constraints	12×12, 60 constraints
$(4, 3, 2)$	28×28, 266 constraints	16×16, 100 constraints
$(3, 2, 4)$	35×35, 279 constraints	18×18, 90 constraints
$(4, 2, 4)$	53×53, 573 constraints	24×24, 150 constraints
$(3, 3, 4)$	59×59, 647 constraints	30×30, 210 constraints
$(4, 3, 4)$	84×84, 1210 constraints	40×40, 350 constraints

Table 1. Sizes of the semidefinite programs for proving $F(x) \succeq 0$, where $F \in \mathbb{R}[x]^{m \times m}$ has degree d and $x \in \mathbb{R}^n$, with and without the sparse multipartite option.

programs. One of these functions is `findbound`, a function for finding a lower bound of a polynomial, whose usage we have seen at the end of Section 2.1. In the new version, these customized functions have been updated and several new capabilities have been added. For instance, the customized function `findbound`, which previously could only handle unconstrained global polynomial optimization problems, can now be used to solve constrained polynomial optimization problems of the form:

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g_i(x) \geq 0, \quad i = 1, ..., M \\ &\qquad\qquad h_j(x) = 0, \quad j = 1, ..., N. \end{aligned}$$

A lower bound for $f(x)$ can be computed using Positivstellensatz-based relaxations. Assume that there exists a set of sums of squares $\sigma_j(x)$'s, and a set of polynomials $\lambda_i(x)$'s, such that

$$\begin{aligned} f(x) - \gamma = \sigma_0(x) + \sum_j \lambda_j(x) h_j(x) + \sum_i \sigma_i(x) g_i(x) + \\ + \sum_{i_1, i_2} \sigma_{i_1, i_2}(x) g_{i_1}(x) g_{i_2}(x) + \cdots \end{aligned} \tag{6}$$

then it follows that γ is a lower bound for the constrained optimization problem stated above. This specific kind of representation corresponds to Schmüdgen's theorem [32]. By maximizing γ, we can obtain a lower bound that becomes increasingly tighter as the degree of the expression (6) is increased.

Another new feature can be found in the customized function `findsos`, which is used for computing an SOS decomposition. For certain applications, it is particularly important to ensure that the SOS decomposition found numerically by SDP methods actually corresponds to a true solution, and is not the result of roundoff errors. This is specially true in the case of ill-conditioned problems, since SDP solvers can sometimes produce in this case unreliable results. There are several ways of doing this, for instance using backwards error

analysis, or by computing rational solutions, that we can fully verify symbolically. Towards this end, we have incorporated an experimental option to round to rational numbers a candidate floating point SDP solution, in such a way to produce an exact SOS representation of the input polynomial (which should have integer or rational coefficients). The procedure will succeed if the computed solution is "well-centered," far away from the boundary of the feasible set; the details of the rounding procedure will be explained elsewhere. Currently, this facility is available only through the customized function `findsos`, by giving an additional input argument `'rational'`. On future releases, we may extend this to more general SOS program formulations.

3 Control Applications

We will now see how sum of squares programs can be formulated to solve several problems arising in systems and control, such as nonlinear stability analysis, parametric robustness analysis, stability analysis of time-delay systems, safety verification, and nonlinear controller synthesis.

3.1 Nonlinear Stability Analysis

The Lyapunov stability theorem (see e.g. [10]) has been a cornerstone of nonlinear system analysis for several decades. In principle, the theorem states that a system $\dot{x} = f(x)$ with equilibrium at the origin is stable if there exists a positive definite function $V(x)$ such that the derivative of V along the system trajectories is non-positive.

We will now show how the search for a Lyapunov function can be formulated as a sum of squares program. Readers are referred to [17, 15, 23] for more detailed discussions and extensions. For our example, consider the system

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

which has an equilibrium at the origin. Notice that the linearization of (7) has zero eigenvalue, and therefore cannot be used to analyze local stability of the equilibrium. Now assume that we are interested in a quadratic Lyapunov function $V(x)$ for proving stability of the system. Then $V(x)$ must satisfy

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

The first inequality, with ϵ being any constant greater than zero (in what follows we will choose $\epsilon = 1$), is needed to guarantee positive definiteness of

$V(x)$. We will formulate an SOS program that computes a Lyapunov function for this system, by replacing the above nonnegativity conditions with SOS conditions. However, notice that $\dot{x}_3$ is a rational function, and therefore (8) is not a polynomial expression. But since $x_3^2 + 1 > 0$ for any x_3, we can just reformulate (8) as

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

Next, we parameterize the candidate quadratic Lyapunov function $V(x)$ by some unknown real coefficients c_1, ..., c_6:

$$V(x) = c_1x_1^2 + c_2x_1x_2 + c_3x_2^2 + ... + c_6x_3^2,$$

and the following SOS program (with no objective function) can be formulated

Find a polynomial $V(x)$, (equivalently, find $c_1, ..., c_6$)
such that
$V(x) - (x_1^2 + x_2^2 + x_3^2)$ is SOS,
$(x_3^2 + 1)\left(-\frac{\partial V}{\partial x_1}\dot{x}_1 - \frac{\partial V}{\partial x_2}\dot{x}_2 - \frac{\partial V}{\partial x_3}\dot{x}_3\right)$ is SOS.

In this example, SOSTOOLS returns $V(x) = 5.5489x_1^2 + 4.1068x_2^2 + 1.7945x_3^2$ as a Lyapunov function that proves the stability of the system.

3.2 Parametric Robustness Analysis

When the vector field of the system is uncertain, e.g., dependent on some unknown but bounded parameters p, robust stability analysis can be performed by finding a parameter dependent Lyapunov function, which serves as a Lyapunov function for the system for all possible parameter values. Details on computation of such Lyapunov functions can be found in [15, 39].

We will illustrate such robustness analysis by considering the system:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -p_1 & 1 & -1 \\ 2 - 2p_2 & 2 & -1 \\ 3 & 1 & -p_1p_2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where p_1 and p_2 are parameters. The region in the parameter space (p_1, p_2) for which stability is retained is shown in Figure 2. Operating conditions for this system are $p_1 \in [\underline{p_1}, \overline{p_1}]$ and $p_2 \in [\underline{p_2}, \overline{p_2}]$, where $\underline{p_i}, \overline{p_i}$ are real numbers and $\underline{p_i} \leq \overline{p_i}$. We capture this parameter set by constructing two inequalities:

$$a_1(p) \triangleq (p_1 - \underline{p_1})(p_1 - \overline{p_1}) \leq 0$$
$$a_2(p) \triangleq (p_2 - \underline{p_2})(p_1 - \overline{p_2}) \leq 0.$$

We assume that the nominal parameter value is $(2.7, 7)$, which is the center of the rectangular regions shown in Figure 2. The robust stability of this system will be verified by constructing a Lyapunov function. We look for a parameter dependent Lyapunov function of the form $V(x;p)$ that is bipartite: quadratic in the state x and any order in p. To ensure that the two Lyapunov inequalities are satisfied in the region of interest, we will adjoint the parameter constraint $a_i(p) \leq 0$ multiplied by sum of squares multipliers $q_{i,j}(x;p)$ to the two Lyapunov inequalities, using a technique that can be considered as an extension of the S-procedure [40]. In this way, the search for a parameter dependent Lyapunov function can be formulated as the following SOS program:

$$
\begin{aligned}
&\text{Find } V(x;p), \text{ and } q_{i,j}(x,p),\\
&\text{such that}\\
&V(x;p) - \|x\|^2 + \sum_{j=1}^{2} q_{1,j}(x;p)a_i(p) \text{ is SOS},\\
&-\dot{V}(x;p) - \|x\|^2 + \sum_{j=1}^{2} q_{2,j}(x;p)a_i(p) \text{ is SOS},\\
&q_{i,j}(x;p) \text{ is SOS, for } i,j = 1,2.
\end{aligned}
$$

In this case, the Lyapunov function candidate $V(x;p)$ and the sum of squares multiplier $q_{i,j}(x,p)$'s are linearly parameterized by some unknown coefficients, which are the decision variables of our SOS program. We choose the $q_{i,j}(x,p)$'s to be bipartite sums of squares, quadratic in x and of appropriate order in p.

When the Lyapunov function $V(x;p)$ is of degree zero in p, we can prove stability for $p_1 \in [2.19, 3.21]$ and $p_2 \in [6.47, 7.53]$. When $V(x;p)$ is affine in p, then we can prove stability for $p_1 \in [1.7, 3.7]$ and $p_2 \in [5.16, 8.84]$. When it is quadratic in p, we can prove stability for the maximum rectangular region centered at the nominal parameter value, i.e., $p_1 \in [1.7, 3.7]$ and $p_2 \in [5, 9]$. See Figure 2.

This example also illustrates the benefit of exploiting the bipartite structure. In the case of quadratic parameter dependence, if the bipartite structure of the conditions is not taken into account then the dimension of the vector $Z(x;p)$ corresponding to a non-structured $V(x;p)$ is 279; taking into account the bipartite structure this number is reduced to 90.

3.3 Stability Analysis of Time-Delay systems

The stability analysis of time-delay systems, i.e., systems described by functional differential equations (FDEs), can be done by constructing appropriate Lyapunov certificates, which are in the form of *functionals* instead of the well known Lyapunov functions that are used in the case of systems described by ordinary differential equations (ODEs). This difference is due to the fact

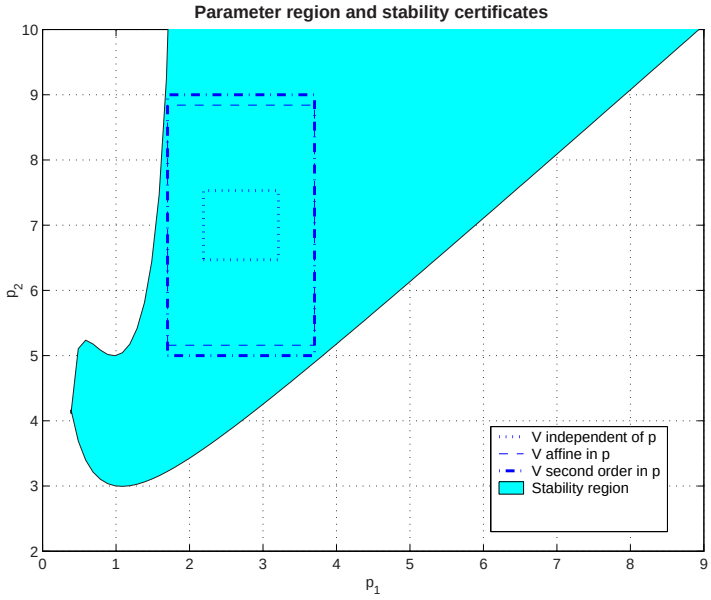

Fig. 2. The full stability region (shaded) and the regions for which stability can be proven by constructing bipartite Lyapunov functions using SOSTOOLS. Bigger regions require higher order certificates, which nonetheless can be easily computed because of their structure.

that the state-space in the case of FDEs is the space of functions and not an Euclidean space [7].

Consider a time delay system of the form:

$$\dot{x} = f(x_t), \tag{9}$$

where $x_t = x(t+\theta)$, $\theta \in [-\tau, 0]$ is the *state*. In order to obtain stability conditions for this system, we use the Lyapunov-Krasovskii functional:

$$V(x_t) = a_0(x(t)) + \int_{-\tau}^{0}\int_{-\tau}^{0} a_1(\theta,\xi,x(t),x(t+\theta),x(t+\xi))d\theta d\xi + \\ + \int_{-\tau}^{0}\int_{t+\theta}^{t} a_2(x(\zeta))d\zeta d\theta + \int_{-\tau}^{0}\int_{t+\xi}^{t} a_2(x(\zeta))d\zeta d\xi \tag{10}$$

where by $a_1(\theta,\xi,x(t),x(t+\theta),x(t+\xi))$ we mean a polynomial in $(\theta,\xi,x(t),x(t+\theta),x(t+\xi))$. In the case in which the time delay system is linear, of the form

$$\dot{x}(t) = A_0 x(t) + A_1 x(t-\tau) = f(x_t), \tag{11}$$

the above functional (10) resembles closely the complete Lyapunov functional presented in [6] and we can further restrict ourselves to specially structured kernels, i.e., the polynomial a_1 need only be bipartite - quadratic in

$(x(t), x(t+\theta), x(t+\xi))$ but any order in (θ, ξ). The polynomials a_0 and a_2 are also quadratic in their arguments. There is also a symmetric structure that should be taken into account:

$$a_1(\theta, \xi, x(t), x(t+\theta), x(t+\xi)) = a_1(\xi, \theta, x(t), x(t+\xi), x(t+\theta)).$$

Here we present the Lyapunov-Krasovskii conditions for stability for concreteness:

Theorem 1 ([6]). *The system described by Eq. (11) is asymptotically stable if there exists a bounded quadratic Lyapunov functional $V(x_t)$ such that for some $\epsilon > 0$, it satisfies*

$$V(x_t) \geq \epsilon \|x(t)\|^2 \tag{12}$$

and its derivative along the system trajectory satisfies

$$\dot{V}(x_t) \leq -\epsilon \|x(t)\|^2. \tag{13}$$

The Lyapunov-Krasovskii conditions for stability can be satisfied by imposing sums of squares conditions on the *kernels* of the corresponding conditions. There are also extra constraints that have to be added, in the sense that the kernels need to be non-negative only in the integration interval:

$$g_1(\theta) = \theta(\theta + \tau) \leq 0$$
$$g_2(\xi) = \xi(\xi + \tau) \leq 0.$$

Such constraints can be adjoined to the Lyapunov conditions as in the previous example. This yields the following SOS program:

Find polynomials $a_0(x(t)), a_1(\theta, \xi, x(t), x(t+\theta), x(t+\xi)), a_2(x(\zeta)), \epsilon > 0$
and sums of squares $q_{i,j}(\theta, \xi, x(t), x(t+\theta), x(t+\xi))$ for $i, j = 1, 2$
such that

$a_0(x(t)) - \epsilon\|x\|^2$ is SOS,

$$a_1(\theta, \xi, x(t), x(t+\theta), x(t+\xi)) + \sum_{j=1}^{2} q_{1,j} g_j \text{ is SOS},$$

$a_2(x(\zeta))$ is SOS,

$$-\left\{\begin{array}{c} \frac{da_0}{dx(t)} f(x_t) + \tau^2 \frac{\partial a_1}{\partial x(t)} f(x_t) - \tau^2 \frac{\partial a_1}{\partial \theta} - \tau^2 \frac{\partial a_1}{\partial \xi} + \\ +\tau a_1(0, \xi, x(t), x(t), x(t+\xi)) - \tau a_1(-\tau, \xi, x(t), x(t-\tau), x(t+\xi)) + \\ +\tau a_1(\theta, 0, x(t), x(t+\theta), x(t)) - \tau a_1(\theta, -\tau, x(t), x(t+\theta), x(t-\tau)) + \\ +2\tau a_2(x(t)) - \tau a_2(x(t+\theta)) - \tau a_2(x(t+\xi)) \end{array}\right\} + \ldots$$

$$- \epsilon\|x\|^2 + \sum_{j=1}^{2} q_{2,j} g_j \text{ is SOS}.$$

The first three conditions guarantee positive definiteness of the functional (10) and the last condition guarantees negative definiteness of its time derivative.

Order of polynomial a in θ and ξ	0	1	2	3	4	5	6
τ	4.472	4.973	5.421	5.682	5.837	5.993	6.028

Table 2. The maximum delay τ_{max} for different degree polynomials a_1 in θ and ξ corresponding to the example in Section 3.3.

In order to keep the symmetric and sparse structure in the corresponding sum of squares conditions we have to make a judicious choice for the multipliers $q_{i,j}$.

As an example, consider the following time delay system:

$$\dot{x}_1(t) = -2x_1(t) - x_1(t-\tau) \triangleq f_1$$
$$\dot{x}_2(t) = -0.9x_2(t) - x_1(t-\tau) - x_2(t-\tau) \triangleq f_2.$$

The system is asymptotically stable for $\tau \in [0, 6.17]$. The best bound on τ that can be obtained with a simple LMI condition is $\tau \in [0, 4.3588]$ in [16]. More complicated LMI conditions that yield better bounds and which are based on a discretization procedure can be found in [6]. Using the Lyapunov functional (10) we get the bounds given in Table 2, where we see that as the order of a with respect to θ and ξ is increased, better bounds are obtained that approach the analytical one.

The symmetric structure and sparsity of the kernels should be taken into account in the construction of the functionals, as this not only reduces the size of the corresponding semidefinite programs but also removes numerical errors. This can be done using the `'sparsemultipartite'` feature in SOSTOOLS. The construction of Lyapunov functionals can also be extended to uncertain nonlinear systems where delay-dependent and delay-independent conditions can be obtained in a similar manner [14].

3.4 Safety Verification

Complex behaviors that can be exhibited by modern engineering systems make the safety verification of such systems both critical and challenging. It is often not enough to design a system to be stable, but a certain bad region in the state space must be completely avoided. Safety verification or reachability analysis aims to show that starting at some initial conditions, a system cannot evolve to an unsafe region in the state space. Here we will show how safety verification can be performed by solving an SOS program, based on what is termed *barrier certificates*. See [22] for detailed discussions and extensions.

For example, let us consider the system (from [22]),

$$\dot{x}_1 = x_2,$$
$$\dot{x}_2 = -x_1 + \frac{1}{3}x_1^3 - x_2,$$

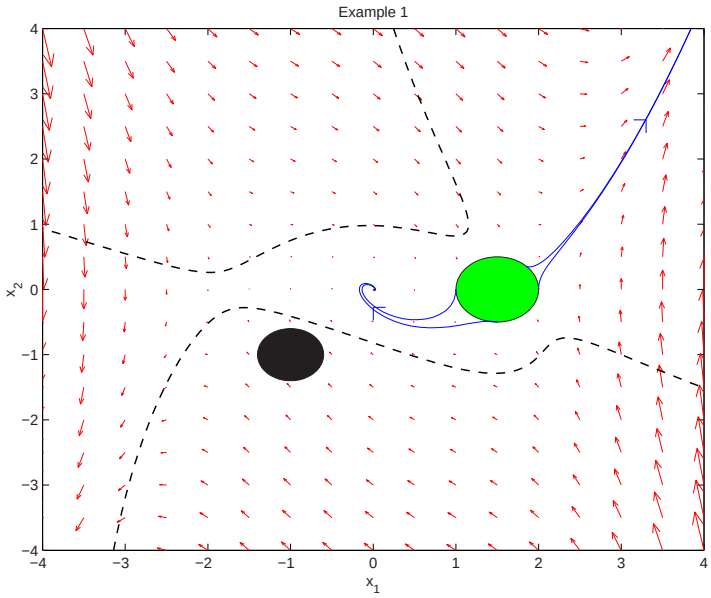

Fig. 3. Phase portrait of the system in Section 3.4. Solid patches are (from the left) $\mathcal{X}_u$ and $\mathcal{X}_0$, respectively. Dashed curves are the zero level set of $B(x)$, whereas solid curves are some trajectories of the system.

whose safety we want to verify, with initial set $\mathcal{X}_0 = \{x : g_{\mathcal{X}_0}(x) = (x_1-1.5)^2+x_2^2-0.25 \leq 0\}$ and unsafe set $\mathcal{X}_u = \{x : g_{\mathcal{X}_u}(x) = (x_1+1)^2+(x_2+1)^2-0.16 \leq 0\}$. Here we will find a barrier certificate $B(x)$ which satisfy the following three conditions: $B(x) < 0 \quad \forall x \in X_0$, $B(x) > 0 \quad \forall x \in X_u$, and $\frac{\partial B}{\partial x_1}\dot{x}_1 + \frac{\partial B}{\partial x_2}\dot{x}_2 \leq 0$. It is clear that the existence of such a function guarantees the safety of the system, and the zero level set of $B(x)$ will separate an unsafe region from all system trajectories starting from a given set of initial conditions. By using the higher degree S-procedure and replacing nonnegativity by SOS conditions, we can formulate the following SOS program:

$$
\begin{aligned}
&\text{Find } B(x), \text{ and } \sigma_i(x),\\
&\text{such that } -B(x) - 0.1 + \sigma_1(x) g_{\mathcal{X}_0}(x) \text{ is SOS},\\
&\qquad B(x) - 0.1 + \sigma_2(x) g_{\mathcal{X}_u}(x) \text{ is SOS},\\
&\qquad -\frac{\partial B}{\partial x_1}\dot{x}_1 + \frac{\partial B}{\partial x_2}\dot{x}_2 \text{ is SOS},\\
&\qquad \sigma_i(x) \text{ is SOS, for } i = 1, 2.
\end{aligned}
$$

In this example, we are able to find a quartic barrier certificate $B(x)$ proving the safety of the system, whose zero level set is shown in Figure 3.

3.5 Nonlinear Controller Synthesis

For a system $\dot{x} = f(x) + g(x)u$, where $f(x)$ and $g(x)$ are polynomials, application of the SOS technique to the state feedback synthesis problem amounts to finding a polynomial state feedback law $u = k(x)$ and a polynomial Lyapunov function $V(x)$ such that $V(x) - \phi(x)$ and $-\frac{\partial V}{\partial x}(f(x) + g(x)k(x))$ are sums of squares, for some positive definite $\phi(x)$. Yet the set of $V(x)$ and $k(x)$ satisfying these conditions is not jointly convex, and hence a simultaneous search for such $V(x)$ and $k(x)$ is hard — it is equivalent to solving some bilinear matrix inequalities (BMIs). Because of this, a dual approach to the state feedback synthesis based on density functions [29] has also been proposed, which has a better convexity property. The idea in this case is to find a density function $\rho(x)$ and a controller $k(x)$ such that $\rho(x)f(x)/|x|$ is integrable on $\{x \in \mathbb{R}^n : |x| \geq 1\}$ and

$$[\nabla \cdot (\rho(f + gk)](x) > 0 \quad \text{for almost all } x. \tag{14}$$

If such $\rho(x)$ and $k(x)$ can be found, then for almost all initial states $x(0)$ the trajectory $x(t)$ of the closed-loop system exists for $t \in [0, \infty)$ and tends to zero as $t \to \infty$. See [28] for details. It is interesting to note that even if the system is not asymptotically stabilizable, it is sometimes possible to design a controller which makes the zero equilibrium almost globally attractive.

Consider for example the system (taken from [28]).

$$\begin{aligned} \dot{x}_1 &= -6x_1x_2^2 - x_1^2x_2 + 2x_2^3, \\ \dot{x}_2 &= x_2u, \end{aligned}$$

whose zero equilibrium is not asymptotically stabilizable, since any state with $x_2 = 0$ is necessarily an equilibrium. Using the following parameterization

$$\rho(x) = \frac{a(x)}{(x_1^2 + x_2^2)^\alpha}; \quad \rho(x)k(x) = \frac{c(x)}{(x_1^2 + x_2^2)^\alpha},$$

the positivity of $\rho(x)$ and the divergence condition (14) can be formulated as the following SOS program:

$$\begin{aligned} &\text{Find } a(x) \text{ and } c(x), \\ &\text{such that} \\ &\quad a(x) - 1 \text{ is SOS}, \\ &\quad [b\nabla \cdot (fa + gc) - \alpha \nabla b \cdot (fa + gc)](x) \text{ is SOS}, \end{aligned}$$

where $b(x) = x_1^2 + x_2^2$. For $\alpha = 3$, we find that the SOS conditions are fulfilled for $a(x) = 1$ and $c(x) = 2.229x_1^2 - 4.8553x_2^2$. Since the integrability condition is also satisfied, we conclude that the controller $u(x) = \frac{c(x)}{a(x)} = 2.229x_1^2 - 4.8553x_2^2$ renders the origin almost globally attractive. The phase portrait of the closed loop system is shown in Figure 4.

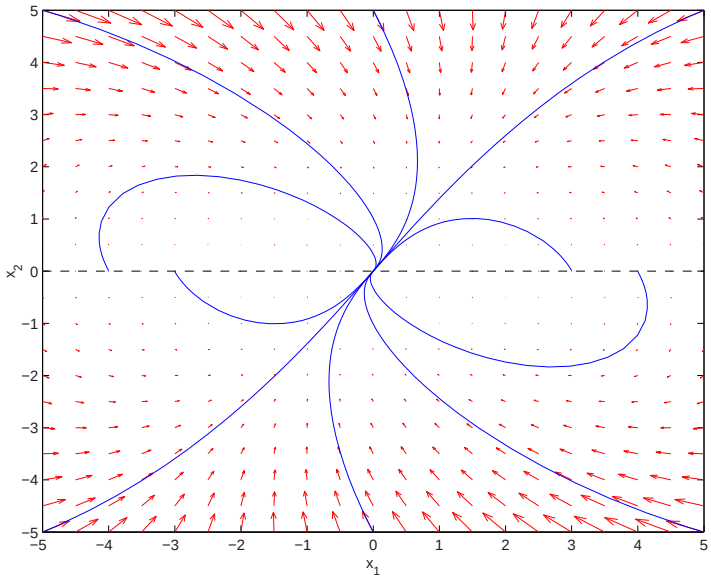

Fig. 4. Phase portrait of the closed-loop system in Section 3.5. Solid curves are trajectories; dashed line is the set of equilibria.

4 Conclusions

In this chapter we have presented some of the features of SOSTOOLS, a free MATLAB toolbox for formulating and solving SOS programs. We have shown how it can be used to solve some control problems, such as nonlinear stability analysis, parametric robustness analysis, stability analysis of time-delay systems, safety verification, and nonlinear controller synthesis. Future improvements to SOSTOOLS will incorporate symmetry reduction and SOS over quotients, e.g., to handle the case where an SOS decomposition is sought for a polynomial $p(x)$ that is invariant under the action of a finite group.

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. A. C. Doherty, P. A. Parrilo, and F. M. Spedalieri. Distinguishing separable and entangled states. *Physical Review Letters*, 88(18), 2002.
3. K. Fukuda. *CDD/CDD+ reference manual*, 2003. Available at `http://www.ifor.math.ethz.ch/staff/fukuda`.
4. K. Gaterman and P. A. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. To appear in *Journal of Pure and Appl. Algebra*, 2004.
5. A. A. Goldstein and J. F. Price. On descent from local minima. *Mathematics of Computation*, 25:569–574, 1971.

6. K. Gu, V. L. Kharitonov, and J. Chen. *Stability of Time-Delay systems*, Birkhäuser, 2003.
7. J. K. Hale and S. M. Verduyn Lunel. *Introduction to Functional Differential Equations*, Applied Mathematical Sciences (99), Springer-Velag, 1993.
8. D. Henrion and J. B. Lasserre. GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi. In *ACM Transactions on Mathematical Software*, (29(2):165–194, 2003. Available at `http://www.laas.fr/~henrion/software/gloptipoly`.
9. Z. Jarvis-Wloszek, R. Feeley, W. Tan, K. Sun, and A. Packard. Some controls applications of sum of squares programming. In *Proceedings of IEEE Conference on Decision and Control*, 2003.
10. H. K. Khalil. *Nonlinear Systems*. Prentice Hall, Inc., second edition, 1996.
11. J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.
12. K. G. Murty and S. N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.
13. 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.
14. A. Papachristodoulou. Analysis of nonlinear time delay systems using the sum of squares decomposition. In *Proceedings of the American Control Conference*, 2004.
15. A. Papachristodoulou and S. Prajna. On the construction of Lyapunov functions using the sum of squares decomposition. In *Proceedings of IEEE Conference on Decision and Control*, 2002.
16. P. G. Park. A delay-dependent stability criterion for systems with uncertain time-invariant delays. *IEEE Transactions on Automatic Control*, 44(2):876–877, 1999.
17. P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, Pasadena, CA, 2000.
18. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming Series B*, 96(2):293–320, 2003.
19. P. A. Parrilo and R. Peretz. An inequality for circle packings proved by semidefinite programming. *Discrete and Computational Geometry*, 31(3):357–367, 2004.
20. 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.
21. S. Prajna. Barrier certificates for nonlinear model validation. In *Proceedings of IEEE Conference on Decision and Control*, 2003.
22. S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 477 – 492. Springer-Verlag, 2004.
23. S. Prajna and A. Papachristodoulou. Analysis of switched and hybrid systems – Beyond piecewise quadratic methods. In *Proceedings of the American Control Conference*, 2003.
24. S. Prajna, A. Papachristodoulou, and P. A. Parrilo. Introducing SOSTOOLS: A general purpose sum of squares programming solver. In *Proceedings of IEEE Conference on Decision and Control*, 2002.

25. S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. SOSTOOLS – Sum of Squares Optimization Toolbox, User's Guide, Version 2.00. Available at `http://www.cds.caltech.edu/sostools` and `http://control.ee.ethz.ch/~parrilo/sostools`, 2004.
26. S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. New developments in sum of squares optimization and SOSTOOLS. In *Proceedings of the American Control Conference*, 2004.
27. S. Prajna, A. Papachristodoulou, and F. Wu. Nonlinear control synthesis by sum of squares optimization: A Lyapunov-based approach. In *Proceedings of Asian Control Conference*, 2004.
28. S. Prajna, P. A. Parrilo, and A. Rantzer. Nonlinear control synthesis by convex optimization. *IEEE Transactions on Automatic Control*, 49(2):310–314, 2004.
29. A. Rantzer. A dual to Lyapunov's stability theorem. *Systems & Control Letters*, 42(3):161–168, 2001.
30. B. Reznick. Extremal PSD forms with few terms. *Duke Mathematical Journal*, 45(2):363–374, 1978.
31. B. Reznick. Some concrete aspects of Hilbert's 17th problem. In *Contemporary Mathematics*, volume 253, pages 251–272. American Mathematical Society, 2000.
32. K. Schmüdgen. The k-moment problem for compact semialgebraic sets. *Mathematische Annalen*, 289:203–206, 1991.
33. P. Seiler. Stability region estimates for SDRE controlled systems using sum of squares optimization. In *Proceedings of the American Control Conference*, 2003.
34. N. Z. Shor. Class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987.
35. 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`.
36. B. Sturmfels. Polynomial equations and convex polytopes. *American Mathematical Monthly*, 105(10):907–922, 1998.
37. K. C. Toh, R. H. Tütüncü, and M. J. Todd. *SDPT3 - a MATLAB software package for semidefinite-quadratic-linear programming.* Available at `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`.
38. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.
39. F. Wu and S. Prajna. A new solution approach to polynomial LPV system analysis and synthesis. In *Proceedings of the American Control Conference*, 2004.
40. V. A. Yakubovich. S-procedure in nonlinear control theory. *Vestnik Leningrad University*, 4(1):73–93, 1977. English translation.