SOSOPT: A Toolbox for Polynomial Optimization Version 2.00

Pete Seiler

seiler@aem.umn.edu

December 12, 2010

Abstract

SOSOPT is a Matlab toolbox for formulating and solving Sum-of-Squares (SOS) polynomial optimizations. This document briefly describes the use and functionality of this toolbox. Section 1 introduces the problem formulations for SOS tests, SOS feasibility problems, SOS optimizations, and generalized SOS problems. Section 2 reviews the SOSOPT toolbox for solving these optimizations. This section includes information on toolbox installation, formulating constraints, solving SOS optimizations, and setting optimization options. Finally, Section 3 briefly reviews the connections between SOS optimizations and semidefinite programs (SDPs). It is the connection to SDPs that enables SOS optimizations to be solved in an efficient manner.

1 Sum of Squares Optimizations

This section describes several optimizations that can be formulated with sum-of-squares (SOS) polynomials [14, 11, 15]. A multivariable polynomial is a SOS if it can be expressed as a sum of squares of other polynomials. In other words, a polynomial p is SOS if there exists polynomials $\{f_i\}_{i=1}^{m}$ such that $p = \sum_{i=1}^{m} f_i^2$. An SOS polynomial is globally nonnegative because each squared term is nonnegative. This fact enables sufficient conditions for many analysis problems to be posed as optimizations with polynomial SOS constraints. This includes many nonlinear analysis problems such as computing regions of attraction, reachability sets, input-output gains, and robustness with respect to uncertainty for nonlinear polynomial systems [14, 25, 7, 9, 8, 15, 17, 12, 6, 4, 22, 10, 13, 21, 23, 26, 30, 29, 27, 24, 28, 1]. The remainder of this section defines SOS tests, SOS feasibility problems, SOS optimizations, and generalized SOS optimizations.

Given a polynomial $p(x)$, a **sum-of-squares test** is an analysis problem of the form:

$$\text{Is } p \text{ a SOS?} \tag{1}$$

A **sum-of-squares feasibility problem** is to construct decision variables to ensure that certain polynomials are SOS. More specifically, an SOS feasibility problem is an optimization with constraints on polynomials that are affine functions of the decision variables:

$$
\begin{aligned}
&\text{Find } d \in \mathbb{R}^r \text{ such that} \\
&a_k(x,d) \in \text{ SOS}, \quad k = 1, \dots N_s \\
&b_j(x,d) = 0, \quad j = 1, \dots N_e
\end{aligned}
\tag{2}
$$

$d \in \mathbb{R}^r$ are decision variables. The polynomials $\{a_k\}$ and $\{b_j\}$ are given as part of the problem data and are affine in d, i.e. they are of the form:

$$
\begin{aligned}
a_k(x,d) &:= a_{k,0}(x) + a_{k,1}(x)d_1 + \cdots + a_{k,n}(x)d_n \\
b_j(x,d) &:= b_{j,0}(x) + b_{j,1}(x)d_1 + \cdots + b_{j,n}(x)d_n
\end{aligned}
$$

A **sum-of-squares optimization** is a problem with a linear cost and constraints on polynomials that are affine

functions of the decision variables:

$$\min_{d \in \mathbb{R}^r} c^T d \tag{3}$$

subject to:

$$a_k(x,d) \in \text{SOS}, \quad k = 1, \ldots N_s$$
$$b_j(x,d) = 0, \quad j = 1, \ldots N_e$$

Again, $d \in \mathbb{R}^r$ denotes the decision variables and the polynomials $\{a_k\}$ and $\{b_j\}$ are given polynomials that are affine in d. SOS tests, feasibility problems, and optimizations are all convex optimization problems. These problems are solved by exploiting the connections between SOS polynomials and positive semidefinite matrices. This is briefly reviewed in the Section 3.

Finally, a **generalized sum-of-squares optimization** is a problem of the form:

$$\min_{d \in \mathbb{R}^r, t \in \mathbb{R}} t \tag{4}$$

subject to:

$$tb_k(x,d) - a_k(x,d) \in \text{SOS}, \quad k = 1, \ldots N_g$$
$$b_k(x,d) \in \text{SOS}, \quad k = 1, \ldots N_g$$
$$c_j(x,d) = 0, \quad j = 1, \ldots N_e$$

$t \in \mathbb{R}$ and $d \in \mathbb{R}^r$ are decision variables. The polynomials $\{a_k\}$, $\{b_k\}$, and $\{c_k\}$ are given data and are affine in d. The optimization cost is t which is linear in the decision variables. The optimization involves standard SOS and polynomial equality constraints. However, this is not an SOS optimization because the constraints, $tb_k(x,d) - a_k(x,d)$ is SOS, are bilinear in the decision variables t and u. However, the generalized SOS program is quasiconvex [18] and it can also be solved efficiently as described in the next subsection.

2 Using SOSOPT

This section describes the `sosopt` toolbox for solving SOS optimizations.

2.1 Installation

The toolbox was tested with MATLAB versions R2009a and R2009b. To install the toolbox:

- Download the zip file and extract the contents to the directory where you want to install the toolbox.

- Add the `sosopt` directory to the Matlab path, e.g. using Matlab's `addpath` command.

The `sosopt` toolbox requires the `multipoly` toolbox to construct the polynomial constraints. `multipoly` can be obtained from `http://www.aem.umn.edu/~AerospaceControl/`. `sosopt` also requires one of the following optimization codes for solving semidefinite programs (SDPs): SeDuMi, SDPT3, CSDP, DSDP, SDPAM, or SDPLR. `sosopt` has been most extensively tested on SeDuMi version 1.3 [20, 19]. The latest version of SeDuMi can be obtained from `http://sedumi.ie.lehigh.edu/`.

2.2 Formulating Constraints

Polynomial SOS and equality constraints are formulated using `multipoly` toolbox objects. The relational operators `<=` and `>=` are overloaded to create SOS constraints. If p and q are polynomials then `p>=q` and `p<=q` denote the constraints $p - q \in \text{SOS}$ and $q - p \in \text{SOS}$, respectively. The relational operator `==` is overloaded to create a polynomial equality constraint. If p and q are polynomials then `p==q` denotes the constraint $p - q = 0$. These overloaded relational operators create a `polyconstr` constraint object. For example, the following code constructs the constraints $6 + d_1 x_1^2 - 5x_2^2 \in \text{SOS}$ and $d_1 x_1^2 + d_2 - 6x_1^2 + 4 = 0$.

```
>> pvar x1 x2 d1 d2
>> p = 6+d1*x1^2;
>> q = 5*x2^2;
```

2

```
>> p>=q
ans =
  d1*x1^2 - 5*x2^2 + 6
  >= 0

>> class(ans)
ans =
polyconstr

>> p=d1*x1^2+d2;
>> q=6*x1^2+4
>> p==q
ans =
  d1*x1^2 - 6*x1^2 + d2 - 4
  == 0
```

The polynomial constraints are displayed in a standard form with all terms moved to one side of the constraint. The polynomials on the left and right sides of the constraint are stored and can be accessed with `.LeftSide` and `.RightSide`. The one-sided constraint that is displayed can be accessed with `.OneSide`. In addition, multiple polynomial constraints can be stacked into a vector list of constraints using the standard Matlab vertical concatenation with brackets and rows separated by a semicolon. Finally, it is also possible to reference and assign into a list of polynomial constraints using standard Matlab commands. These features are shown below.

```
>> pvar x1 x2 d1 d2
>> constraint1 =  6+d1*x1^2 >= 5*x2^2;
>> constraint1.LeftSide
ans =
  d1*x1^2 + 6

>> constraint1.RightSide
ans =
  5*x2^2

>> constraint1.OneSide
ans =
  d1*x1^2 - 5*x2^2 + 6

>> constraint2 =  d1*x1^2+d2 == 6*x1^2+4;
>> constraints = [constraint1; constraint2]
constraints =
  polyconstr object with 2 constraints.

>> constraints(1)
ans =
  d1*x1^2 - 5*x2^2 + 6
  >= 0

>> constraints(1).OneSide
ans =
  d1*x1^2 - 5*x2^2 + 6

>> constraints(2)
ans =
  d1*x1^2 - 6*x1^2 + d2 - 4
  == 0

>> constraints(2) = (d2==8);
```

```
>> constraints(2)
ans =
  d2 - 8
  == 0

>> constraints.RelOp
ans =
    '>='
    '=='
```

2.3 Solving SOS Optimizations

The four SOS problems introduced in Section 1 can be solved using the `sosopt` functions described below. Documentation for each function can be obtained at the Matlab prompt using the `help` Command.

1. **SOS test**: The function `issos` tests if a polynomial p is SOS. The syntax is:

   ```
   [feas,z,Q,f] = issos(p,opts)
   ```

   `p` is a `multipoly` polynomial object. `feas` is equal to 1 if the polynomial is SOS and 0 otherwise. If `feas=1` then f is a vector of polynomials that provide the SOS decomposition of p, i.e. $p = \sum_i f_i^2$. z is a vector of monomials and and Q is a positive semidefinite matrix such that $p = z^T Q z$. z and Q are a Gram matrix decomposition for p. This is described in more detail in Section 3. The `opts` input is an `sosoptions` object. Refer to Section 2.6 for more details on these options.

2. **SOS feasibility**: The function `sosopt` solves SOS feasibility problems. The syntax is:

   ```
   [info,dopt,sossol] = sosopt(pconstr,x,opts);
   ```

   `pconstr` is an $N_p \times 1$ vector of polynomial SOS and equality constraints constructed as described in Section 2.2. `x` is a vector list of polynomial variables. The variables listed in `x` are the independent polynomial variables in the constraints. All other variables that exist in the polynomial constraints are assumed to be decision variables. The polynomial constraints must be affine functions of these decision variables. The `opts` input is an `sosoptions` object (See Section 2.6).

   The `info` output is a structure that contains a variety of information about the construction of the SOS optimization problem. The main data in this structure is the `feas` field. This field is equal to 1 if the problem is feasible and 0 otherwise.

   The `dopt` output is a polynomial array of the optimal decision variables. The first column of `dopt` contains the decision variables and the second column contains the optimal values. The polynomial `subs` command can be used to replace the decision variables in any polynomial with their optimal values, e.g `subs(pconstr(1).LeftSide, dopt)` substitutes the optimal decision variables into the left side of the first constraint. `dopt` is returned as empty if the optimization is infeasible.

   `sossol` is an $N_p \times 1$ structure array with fields `p`, `z`, and `Q`. `sossol(i).p` is `pconstr(i)` evaluated at the optimal decision variables. If `pconstr(i)` is an SOS constraint then `sossol(i).z` and `sossol(i).Q` are the vector of monomials and positive semidefinite matrix for the Gram matrix decomposition of `sossol(i).p`, i.e. $p = z^T Q z$. This Gram matrix decomposition is described in more detail in Section 3. If `pconstr(i)` is a polynomial equality constraint then these two fields are returned as empty. `sossol` is empty if the optimization is infeasible.

3. **SOS optimization**: The function `sosopt` also solves SOS optimization problems. The syntax is:

   ```
   [info,dopt,sossol] = sosopt(pconstr,x,obj,opts);
   ```

`obj` is a polynomial that specifies the objective function. This must be be an affine function of the decision variables and it cannot depend on the polynomial variables. In other words, `obj` must have the form $c_0 + \sum_i c_i d_i$ where c_i are real numbers and d_i are decision variables. The remaining inputs and outputs are the same as described for SOS feasibility problems. The `info` output has one additional field `obj` that specifies the minimal value of the objective function. This field is the same as `subs(obj,dopt)`. `obj` is set to `+inf` if the problem is infeasible.

4. **Generalized SOS optimization**: The function `gsosopt` solves generalized SOS optimization problems. The syntax is:

```
[info,dopt,sossol] = gsosopt(pconstr,x,t,opts)
```

`pconstr` is again an $N_p \times 1$ vector of polynomial SOS and equality constraints constructed as described in Section 2.2. `x` is a vector list of polynomial variables. The variables listed in `x` are the independent polynomial variables in the constraints. All other variables that exist in the polynomial constraints are assumed to be decision variables. The objective function is specified by the third argument `t`. This objective must be a single polynomial variable and it must be one of the decision variables. The constraints must have the special structure specified in the Generalized SOS problem formulation. Let (`d`,`t`) denote the complete list of decision variables. The constraints are allowed to have bilinear terms involving products of `t` and `d`. However, they must be linear in `d` for fixed `t` and linear in `t` for fixed `d`. The `opts` input is an `gsosoptions` object (See Section 2.6).

The outputs are the same as described for SOS feasibility and optimization problems. The only difference is that the `info` output does not have an `obj` field. `gsosopt` uses a bisection to solve the generalized SOS problem. It computes lower and upper bounds on the optimal cost such that the bounds are within a specified stopping tolerance. These bounds are returned in the `tbnds` field. This is a 1×2 vector $[t_{lb}, t_{ub}]$ giving the lower bound t_{lb} and upper bound t_{ub} on the minimum value of `t`. `tbnds` is empty if the optimization is infeasible.

2.4   Constructing Polynomial Decision Variables

The `sosopt` and `multipoly` toolboxes contain several functions to quickly and easily construct polynomials whose coefficients are decision variables. The `mpvar` and `monomials` functions in the `multipoly` toolbox can be used to construct a matrix of polynomial variables and a vector list of monomials, respectively. Examples are shown below:

```
>> P = mpvar('p',[4 2])
P =
  [ p_1_1, p_1_2]
  [ p_2_1, p_2_2]
  [ p_3_1, p_3_2]
  [ p_4_1, p_4_2]

>> pvar x1 x2
>> w = monomials([x1;x2],0:2)
w =
  [      1]
  [     x1]
  [     x2]
  [   x1^2]
  [ x1*x2]
  [   x2^2]
```

The first argument of `mpvar` specifies the prefix for the variable names in the matrix and the the second argument specifies the matrix size. The first argument of `monomials` specifies the variables used to construct the monomials vector. The second argument specifies the degrees of monomials to include in the monomials vector. In the example above, the vector `w` returned by `monomials` contains all monomials in variables `x1` and `x2` of degrees 0,1, and 2.

These two functions can be used to quickly construct a polynomial p that is a linear combination of monomials in x with coefficients specified by decision variables d.

```
>> pvar x1 x2
>> w = monomials([x1;x2],0:2);
>> d = mpvar('d',[length(w),1]);
>> [w, d]
ans =
   [      1, d_1]
   [     x1, d_2]
   [     x2, d_3]
   [   x1^2, d_4]
   [ x1*x2, d_5]
   [   x2^2, d_6]

>> p = d'*w
p =
   d_4*x1^2 + d_5*x1*x2 + d_6*x2^2 + d_2*x1 + d_3*x2 + d_1
```

This example constructs a quadratic function in variables (x_1, x_2) with coefficients given by the entries of d. p could alternatively be interpreted as a cubic polynomial in variables (x, d).

The `polydecvar` function can be used to construct polynomials of this form in one command:

```
>> p = polydecvar('d',w)
p =
   d_4*x1^2 + d_5*x1*x2 + d_6*x2^2 + d_2*x1 + d_3*x2 + d_1
```

The first argument of `polydecvar` specifies the prefix for the coefficient names and the second argument specifies the monomials to use in constructing the polynomial. The output of `polydecvar` is a polynomial in the form: `p=d'*w` where `d` is a coefficient vector generated by `mpvar`. This is called the <u>vector</u> form because the coefficients are specified in the vector `d`.

The Gram matrix provides an alternative formulation for specifying polynomial decision variables. In particular, one can specify a polynomial as $p(x, D) = z(x)^T D z(x)$ where $z(x)$ is a vector of monomials and D is a symmetric matrix of decision variables. A quadratic function in variables (x_1, x_2) with coefficient matrix D is constructed as follows:

```
>> pvar x1 x2
>> z = monomials([x1;x2],0:1);
>> D = mpvar('d',[length(z) length(z)],'s')
D =
   [ d_1_1, d_1_2, d_1_3]
   [ d_1_2, d_2_2, d_2_3]
   [ d_1_3, d_2_3, d_3_3]

>> s = z'*D*z
s =
   d_2_2*x1^2 + 2*d_2_3*x1*x2 + d_3_3*x2^2 + 2*d_1_2*x1 + 2*d_1_3*x2 + d_1_1
```

The `'s'` option specifies that `mpvar` should return a symmetric matrix. This construction can be equivalently performed using the `sosdecvar` command:

```
>> [s,D] = sosdecvar('d',z)
s =
   d_2_2*x1^2 + 2*d_2_3*x1*x2 + d_3_3*x2^2 + 2*d_1_2*x1 + 2*d_1_3*x2 + d_1_1
D =
   [ d_1_1, d_1_2, d_1_3]
   [ d_1_2, d_2_2, d_2_3]
   [ d_1_3, d_2_3, d_3_3]
```

This is called the <u>matrix</u> form because the coefficients are specified in the symmetric matrix `D`.

In the examples above, the vector and matrix forms both use six independent coefficients to specify a quadratic polynomial in (x_1, x_2). In general, the matrix form uses many more variables than the vector form to represent the coefficients of a polynomial. Thus the vector form will typically lead to more efficient problem formulations. The only case in which `sosdecvar` leads to more efficient implementations is when the resulting polynomial is directly constrained to be SOS. Specifically, the `sosdecvar` command should be used to construct polynomials that will be directly added to the list of SOS constraints, as in the example below:

```
>> [s,D] = sosdecvar('d',z);
>> pconstr(i) = s>=0;
```

NOTE: Creating a polynomial variable `s` using the `sosdecvar` command will not cause `sosopt` or `gsosopt` to constrain the polynomial to be SOS. The constraint `s>=0` must be added to the list of constraints to enforce `s` to be SOS.

2.5 Demos

`sosopt` includes several demo files that illustrate the use of the toolbox. These demo files can be found in the `Demos` subfolder. A brief description of the existing demo files is given below.

1. **SOS test**: `issosdemo1` demonstrates the use of the `issos` function for testing if a polynomial p is a sum of squares. This example uses `issos` to construct an SOS decomposition for a degree four polynomial in two variables. The example polynomial is taken from Section 3.1 of the SOSTOOLs documentation [17]. `sosoptdemo1` solves the same SOS test using the `sosopt` function.

2. **SOS feasibility**: There are three demo files that solve SOS feasibility problems: `sosoptdemo2`, `sosoptdemo4`, and `sosoptdemo5`. These examples are taken from Sections 3.2, 3.4, and 3.5 of the SOSTOOLs documentation [17], respectively. Demo 2 solves for a global Lyapunov function of a rational, nonlinear system. Demo 4 verifies the copositivity of a matrix. Demo5 computes an upper bound for a structured singular value problem.

3. **SOS optimization**: There are three demo files that solve SOS optimization problems: `sosoptdemo3`, `sosoptdemoLP`, and `sosoptdemoEQ`. Demo 3 is taken from Section 3.3 of the SOSTOOLs documentation [17]. This demo uses SOSOPT to compute a lower bound on the global minimum of the Goldstein-Price function. The EQ demo provides a simple example with polynomial equality constraints in addition to SOS constraints. Finally, the LP demo shows that linear programming constraints can be formulated using `sosopt`.

4. **Generalized SOS optimization**: There are two demo files that solve generalized SOS optimization problems: `gsosoptdemo1` and `pcontaindemo1`. `gsosoptdemo1` `gsosopt` to compute an estimate of the region of attraction for the van der Pol oscillator using the Lyapunov function obtained via linearization. `pcontaindemo1` solves for the radius of the largest circle that lies within the contour of a 6th degree polynomial. This is computed using the specialized function `pcontain` for verifying set containments. The set containment problem is a specific type of generalized SOS optimization.

2.6 Options

The `sosoptions` command will create a default options structure for the `issos` and `sosopt` functions. The `sosoptions` command will return an object with the fields:

- `solver`: Optimization solver to be used. The choices are: 'sedumi', 'sdpam', 'dsdp', 'sdpt3', 'csdp', or 'sdplr'. The default solver is 'sedumi'.

- `form`: Formulation for the optimization. The choices are 'image' or 'kernel'. These forms are described in Section 3. The default is 'image'.

- `simplify`: SOS simplification procedure to remove monomials that are not needed in the Gram matrix form. This reduces the size of the related semidefinite programming problem and hence also reduces the computational time. The choices are 'on' or 'off' and the default is 'on'.

- `scaling`: Scaling of SOS constraints. This scales each constraint by the Euclidean norm (2-norm) of the one-sided polynomial coefficient vector. The choices are 'on' or 'off' and the default is 'off'.

- `checkfeas`: Check feasibility of solution. The choices are 'off', 'fast', 'full', and 'both'. The default is 'fast'. 'fast' checks feasibility information returned by the solver. 'full' checks the validity of the Gram matrix decomposition in the output sossol. The 'full' check is more computationally costly. 'both' does both feasibility checks.

- `feastol`: Feasibility tolerance used in the 'full' feasibility check. This should be a positive, scalar, double. The default is `1e-6`.

- `solveropts`: Structure with options passed directly to the optimization solver. The default is empty. The solver display is turned off with this default.

The `gsosoptions` command will create a default options structure for the `gsosopt` function. The `gsosoptions` command will return an object with all fields contained in an `sosoptions` structure. In addition it will contain the fields:

- `minobj`: Minimum value of objective for bisection. This should be a scalar double. The default is `-1e3`.

- `maxobj`: Maximum value of objective for bisection. This should be a scalar double. Moreover, `maxobj` should be $\geq$ `minobj`. The default is `1e3`.

- `absbistol`: Absolute bisection stopping tolerance This should be a positive, scalar, double. The default is `1e-3`. The bisection terminates if $t_{ub} - t_{lb} \leq$ `absbistol`.

- `relbistol`: Relative bisection stopping tolerance This should be a positive, scalar, double. The default is `1e-3`. The bisection terminates if $t_{ub} - t_{lb} \leq$ `relbistol` $\times t_{lb}$.

- `display`: Display bisection iteration information. The choices are 'on' or 'off' and the default is 'off'. If `display = 'on'` then `gsosoptions` displays, for each iteration, the attempted value of t, feasibility result and the current upper and lower bounds on the optimal value of t. The display information generated by the optimization solver is not affected by this option.

3 Connections to SDPs

Given a polynomial p, an SOS test is to determine if p is a SOS. To solve this problem, the polynomial if expressed in the form $p = z^T Q z$ where z is a vector of monomials and Q is a symmetric "Gram" matrix [1]. The Gram matrix is not unique and a known result is that p is a SOS if and only if there exists $Q = Q^T \succeq 0$ such that $p = z^T Q z$ [5, 16]. Equating the coefficients of p and $z^T Q z$ leads to linear equality constraints on the entries of Q. There exists a matrix A and vector b such that these equality constraints can be represented as $Aq = b$ where $q := vec(Q)$ denotes the vector obtained by vertically stacking the columns of Q. Thus the SOS test can be converted to a problem of the form:

$$\text{Given a matrix } A \text{ and vector } b, \text{ find } Q \succeq 0 \text{ such that } Aq = b. \tag{5}$$

This is a semidefinite programming (SDP) problem [2, 31]. In general, there are fewer equality constraints than independent entries of Q, i.e. A has fewer rows than columns. One can compute a particular solution Q_0 such that $p = z^T Q_0 z$ and a basis of homogeneous solutions $\{N_i\}$ such that $z^T N_i z = 0$ for each i where 0 is the zero polynomial. The matrix A has special structure that can be exploited to efficiently compute these matrices. Thus every matrix Q satisfying $p = z^T Q z$ can be expressed in the form $Q_0 + \sum_i \lambda_i N_i \succeq 0$ where $\lambda_i \in \mathbb{R}$. This enables the SOS test to be converted into the alternative formulation:

$$\text{Given matrices } Q_0 \text{ and } \{N_i\} \text{ find a vector } \lambda \text{ such that } Q_0 + \sum_i \lambda_i N_i \succeq 0. \tag{6}$$

This problem has a single linear matrix inequality (LMI) and is also a semidefinite programming problem. The SDPs in Equation 5 and Equation 6 are dual optimization problems [31]. There exist many freely available codes to solve these types of problem, e.g. SeDuMi [20, 19]. In the SeDuMi formulation, Equation 5 is called the primal or image problem and Equation 6 is the dual or kernel problem.

---

[1] A monomial is a term of the form $x^\alpha \doteq x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ where the α_i are non-negative integers.

The constraints in SOS feasibility and optimization problems are similarly converted to semidefinite matrix constraints. For example, $a_k(x, d)$ is SOS if and only if there exists $Q \succeq 0$ such that

$$a_{k,0}(x) + a_{k,1}(x)d_1 + \cdots + a_{k,n}(x)d_n = z(x)^T Q z(x) \tag{7}$$

Equating the coefficients leads to linear equality constraints on the decision variables d and the entries of Q. There exist matrices A_d, A_q and a vector b such that these equality constraints can be represented as $A_d d + A_q q = b$ where $q := vec(Q)$. Thus $a_k(x, d)$ is SOS if and only if there exists $Q \succeq 0$ such that $A_d d + A_q q = b$. Each SOS constraint can be replaced in this way by a positive semidefinite matrix subject to equality constraints on its entries and on the decision variables. The polynomial equality constraints are equivalently represented by equality constraints on the decision variables. Performing this replacement for each constraint in an SOS feasibility or optimization problem leads to an optimization with equality and semidefinite matrix constraints. This is an SDP in SeDuMi primal/image form. An SDP in SeDuMi dual/kernel is obtained by replacing the positive semidefinite matrix variables Q that that arise from each SOS constraint with linear combinations of a particular solution Q_0 and homogeneous solutions $\{N_i\}$. This is similar to the steps described above for the SOS test and full details can be found in [1].

Finally, the generalized SOS optimization has SOS constraints that are bilinear in decision variables t and d. A consequence of this bilinearity is that the SOS constraints cannot be replaced with linear equality constraints on the decision variables. However, the generalized SOS program is quasiconvex [18] and it can be efficiently solved. In particular, for fixed values of t the constraints are linear in the remaining decision variables d. An SOS feasibility problem can be solved to determine if the constraints are feasible for fixed t. Bisection can be used to find the minimum value of t, to within a specified tolerance, for which the constraints are feasible. In principle this problem can also be converted to a generalized eigenvalue problem [3] (subject to some additional technical assumptions) but the theory and available software for generalized eigenvalue problems are not as well-developed as for SDPs.

`sosopt` converts the SOS optimizations into SDPs in either primal/image or dual/kernel form. The form can be specified with the `form` option in the `sosoptions` object. Interested users can see the lower level functions `gramconstraint` and `gramsol` for implementation details on this conversion. `sosopt` then solves the SDP using one of the freely available solvers that have been interfaced to the toolbox. The `solver` option is used to specify the solver. Finally, `sosopt` converts the SDP solution back to polynomial form. Specifically, the optimal SOS decision variables and the Gram matrix decompositions are constructed from the SDP solution. `sosopt` also checks the feasibility of the returned solution. The `checkfeas` option specifies the feasibility check performed by `sosopt`. The `fast` option simply checks the feasibility information returned by the SDP solver. The `full` option verifies the Gram matrix decomposition for each SOS constraint. In particular, it checks that the Gram matrix is positive semidefinite and it checks that $p = z^T Q z$ within some tolerance. The `full` feasibility check also verifies that each SOS equality constraint is satisfied within a specified tolerance.

4   Acknowledgments

References

[1] G.J. Balas, A. Packard, P. Seiler, and U. Topcu. Robustness analysis of nonlinear systems. http://www.aem.umn.edu/~AerospaceControl/, 2009.

[2] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, 1994.

[3] S. Boyd and L. El Ghaoui. Method of centers for minimizing generalized eigenvalues. *Linear Algebra and Its Applications*, 188:63–111, 1993.

[4] G. Chesi. On the estimation of the domain of attraction for uncertain polynomial systems via LMIs. In *Proc. of the IEEE Conference on Decision and Control*, pages 881–886, 2004.

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

[6] K. Gatermann and P. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. *Journal of Pure and Applied Algebra*, 192:95–128, 2004.

[7] O. Hachicho and B. Tibken. Estimating domains of attraction of a class of nonlinear dynamical systems with LMI methods based on the theory of moments. In *Proc. of the IEEE Conference on Decision and Control*, pages 3150–3155, 2002.

[8] Z. Jarvis-Wloszek. *Lyapunov Based Analysis and Controller Synthesis for Polynomial Systems using Sum-of-Squares Optimization*. PhD thesis, University of California, Berkeley, 2003.

[9] Z. Jarvis-Wloszek, R. Feeley, W. Tan, K. Sun, and A. Packard. Some controls applications of sum of squares programming. In *Proc. of the 42nd IEEE Conference on Decision and Control*, volume 5, pages 4676–4681, 2003.

[10] Z. Jarvis-Wloszek, R Feeley, W. Tan, K. Sun, and A. Packard. *Positive Polynomials in Control*, volume 312 of *Lecture Notes in Control and Information Sciences*, chapter Controls Applications of Sum of Squares Programming, pages 3–22. Springer-Verlag, 2005.

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

[12] J. Lofberg. Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proc. of the CACSD Conference*, Taipei, Taiwan, 2004.

[13] A. Papachristodoulou. *Scalable analysis of nonlinear systems using convex optimization*. PhD thesis, California Institute of Technology, 2005.

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

[15] P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming Ser. B*, 96(2):293–320, 2003.

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

[17] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2004.

[18] P. Seiler and G.J. Balas. Quasiconvex sum-of-squares programming. In *Proc. of the IEEE Conference on Decision and Control*, 2010.

[19] J. Sturm. SeDuMi version 1.05. http://fewcal.kub.nl/sturm/software/sedumi.html, 2001.

[20] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, pages 625–653, 1999.

[21] W. Tan. *Nonlinear Control Analysis and Synthesis using Sum-of-Squares Programming*. PhD thesis, University of California, Berkeley, 2006.

[22] W. Tan and A. Packard. Searching for control Lyapunov functions using sums of squares programming. In *42nd Annual Allerton Conference on Communications, Control and Computing*, pages 210–219, 2004.

[23] W. Tan, A. Packard, and T. Wheeler. Local gain analysis of nonlinear systems. In *Proc. of the American Control Conference*, pages 92–96, 2006.

[24] W. Tan, U. Topcu, P. Seiler, G. Balas, and A. Packard. Simulation-aided reachability and local gain analysis for nonlinear dynamical systems. In *Proc. of the IEEE Conference on Decision and Control*, pages 4097–4102, 2008.

[25] B. Tibken. Estimation of the domain of attraction for polynomial systems via LMIs. In *Proc. of the IEEE Conference on Decision and Control*, pages 3860–3864, 2000.

[26] B. Tibken and Y. Fan. Computing the domain of attraction for polynomial systems via BMI optimization methods. In *Proc. of the American Control Conference*, pages 117–122, 2006.

[27] U. Topcu. *Quantitative Local Analysis of Nonlinear Systems*. PhD thesis, University of California, Berkeley, 2008.

[28] U. Topcu and A. Packard. Linearized analysis versus optimization-based nonlinear analysis for nonlinear systems. In *submitted to the 2009 American Control Conference*, 2009.

[29] U. Topcu, A. Packard, and P. Seiler. Local stability analysis using simulations and sum-of-squares programming. *Automatica*, 44(10):2669–2675, 2008.

[30] U. Topcu, A. Packard, P. Seiler, and T. Wheeler. Stability region analysis using simulations and sum-of-squares programming. In *Proc. of the American Control Conference*, pages 6009–6014, 2007.

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