

SOSTOOLS

Sum of Squares Optimization Toolbox for MATLAB

User's guide

Version 1.00

April 11, 2002

Stephen Prajna¹

Antonis Papachristodoulou¹

Pablo A. Parrilo²

¹Control and Dynamical Systems
California Institute of Technology
Pasadena, CA 91125 – USA

²Automatic Control Laboratory
Swiss Federal Institute of Technology Zürich
CH-8092 Zürich – Switzerland

Copyright (C) 2002 S. Prajna, A. Papachristodoulou, P. A. Parrilo

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Contents

1	Getting Started with SOSTOOLS	7
1.1	Sum of Squares Polynomials and Sum of Squares Programs	7
1.2	What SOSTOOLS Does	9
1.3	System Requirements and Installation Instruction	10
1.4	Other Things You Need to Know	11
2	Solving Sum of Squares Programs	13
2.1	Polynomial Representation and Manipulations	13
2.2	Initializing a Sum of Squares Program	14
2.3	Variable Declaration	14
2.3.1	Scalar Decision Variables	14
2.3.2	Polynomial Variables	15
2.3.3	An Aside: Constructing Vectors of Monomials	16
2.3.4	Sum of Squares Variables	17
2.4	Adding Constraints	17
2.4.1	Equality Constraints	18
2.4.2	Inequality Constraints	18
2.5	Setting Objective Function	19
2.6	Calling Solver	19
2.7	Getting Solutions	19
3	Applications of Sum of Squares Programming	23
3.1	Sum of Squares Test	23
3.2	Lyapunov Function Search	25
3.3	Bound on Global Extremum	26
3.4	Matrix Copositivity	29
3.5	Upper Bound of Structured Singular Value	29
3.6	MAX CUT	31
3.7	Chebyshev Polynomials	34
3.8	Bounds in Probability	36
4	List of Functions	39

Chapter 1

Getting Started with SOSTOOLS

SOSTOOLS is a free, third-party MATLAB¹ toolbox for solving sum of squares programs. The techniques behind it are based on the sum of squares decomposition for multivariate polynomials [2], which can be efficiently computed using semidefinite programming [17]. SOSTOOLS is developed as a consequence of the recent interest in sum of squares polynomials [9, 10] (see also [15, 2, 14, 7, 6]), partly due to the fact that these techniques provide convex relaxations for many hard problems such as global, constrained, and boolean optimization.

Besides the optimization problems mentioned above, sum of squares polynomials (and hence SOSTOOLS) find applications in many other applications areas. This includes control theory problems, such as: search for Lyapunov functions to prove stability of a dynamical system, computation of tight upper bounds for the structured singular value μ [9], and stabilization of nonlinear systems [12]. Some examples related to these problems, as well as several other optimization-related examples, are provided and solved in the demo files that are distributed together with SOSTOOLS.

In the next two sections, we will provide a quick overview on sum of squares polynomials and programs, and show the role of SOSTOOLS in sum of squares programming.

1.1 Sum of Squares Polynomials and Sum of Squares Programs

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

$$p(x) = \sum_{i=1}^m f_i^2(x). \quad (1.1)$$

It is clear that $f(x)$ being an SOS naturally implies $f(x) \geq 0$ for all $x \in \mathbb{R}^n$. For a (partial) converse statement, we remind you of the equivalence, proven by Hilbert, between “nonnegativity” and “sum of squares” in the following cases:

- Univariate polynomials, any (even) degree.
- Quadratic polynomials, in any number of variables.
- Quartic polynomials in two variables.

(see [14] and the references therein). In the general multivariate case, however, $f(x) \geq 0$ in the usual sense does not necessarily imply that $f(x)$ is SOS. Notwithstanding this fact, the crucial

¹A registered trademark of The MathWorks, Inc.

thing to keep in mind is that, while being stricter, the condition that $f(x)$ is SOS is much more *computationally tractable* than nonnegativity [9]. At the same time, practical experience indicates that replacing nonnegativity with the SOS property in many cases leads nevertheless to the exact solution.

The SOS condition (1.1) is equivalent to the existence of a positive semidefinite matrix Q , such that

$$p(x) = Z^T(x)QZ(x), \quad (1.2)$$

where $Z(x)$ is some properly chosen vector of monomials. Expressing an SOS polynomial using a quadratic form as in (1.2) has also been referred to as the Gram matrix method [2, 11].

As hinted above, 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], is based on the *Positivstellensatz*, a central result in real algebraic geometry. Most examples in this manual can be interpreted as special cases of the practical application of this general relaxation method. In this type of relaxations, we are interested in finding polynomials $p_i(x)$, $i = 1, 2, \dots, \hat{N}$ and sums of squares $p_i(x)$ for $i = (\hat{N} + 1), \dots, N$ such that

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

where the $a_{i,j}(x)$'s are some given constant coefficient polynomials. Problems of this type will be termed “sum of squares programs” (SOSP). Solutions to SOSPs 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).

The basic feasibility problem in SOS programming will be formulated as follows:

FEASIBILITY:

Find

$$\begin{aligned} &\text{polynomials } p_i(x), && \text{for } i = 1, 2, \dots, \hat{N} \\ &\text{sums of squares } p_i(x), && \text{for } i = (\hat{N} + 1), \dots, N \end{aligned}$$

such that

$$a_{0,j}(x) + \sum_{i=1}^N p_i(x)a_{i,j}(x) = 0, \quad \text{for } j = 1, 2, \dots, \hat{J}, \quad (1.3)$$

$$\begin{aligned} &a_{0,j}(x) + \sum_{i=1}^N p_i(x)a_{i,j}(x) \quad \text{are sums of squares } (\geq 0)^2, \\ &\text{for } j = (\hat{J} + 1), (\hat{J} + 2), \dots, J. \end{aligned} \quad (1.4)$$

In this formulation, the $a_{i,j}(x)$ are given scalar constant coefficient polynomials. The $p_i(x)$'s will be termed *SOSP variables*, and the constraints (1.3)–(1.4) are termed *SOSP constraints*. The feasible set of this problem is convex, and as a consequence SOS programming can in principle be solved using the powerful tools of *convex optimization*.

It is obvious that the same program can be formulated in terms of constraints (1.3) only, by introducing some extra sums of squares as slack program variables. However, we will keep this

²Whenever constraint $f(x) \geq 0$ is encountered in an SOSP, it should always be interpreted as “ $f(x)$ is an SOS”.

more explicit notation for its added flexibility, since in most cases it will help make the problem statement clearer.

Since many problems are more naturally formulated using inequalities, we will call the constraints (1.4) “inequality constraints”, and denote them by ≥ 0 . It is important, however, to keep in mind the (possible) gap between nonnegativity and SOS.

Besides pure feasibility, the other natural class of problems in convex SOS programming involves optimization of an objective function that is linear in the coefficients of $p_i(x)$ ’s. The general form of such optimization problem is as follows:

OPTIMIZATION:

Minimize the linear objective function

$$w^T c,$$

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

$$\begin{array}{ll} \text{polynomials } p_i(x), & \text{for } i = 1, 2, \dots, \hat{N} \\ \text{sums of squares } p_i(x), & \text{for } i = (\hat{N} + 1), \dots, N \end{array}$$

such that

$$a_{0,j}(x) + \sum_{i=1}^N p_i(x) a_{i,j}(x) = 0, \quad \text{for } j = 1, 2, \dots, \hat{J}, \quad (1.5)$$

$$\begin{aligned} a_{0,j}(x) + \sum_{i=1}^N p_i(x) a_{i,j}(x) & \text{ are sums of squares } (\geq 0), \\ & \text{for } j = (\hat{J} + 1), (\hat{J} + 2), \dots, J, \end{aligned} \quad (1.6)$$

In this formulation, w is the vector of weighting coefficients for 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 $a_{i,j}, b_{i,j}$ are quadratic forms, then we exactly recover the standard linear matrix inequality (LMI) problem formulation. The extra degrees of freedom in SOS programming are actually a bit illusory, as every SOSP can be exactly converted to an equivalent semidefinite program [9]. Nevertheless, 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 Chapter 3.

1.2 What SOSTOOLS Does

Currently, sum of squares programs are solved by reformulating them as semidefinite programs (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 SOSPs 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 SOSPs. This is exactly where SOSTOOLS comes to play. It automates the conversion from SOSP to SDP, calls the SDP solver, and converts the SDP solution back to the solution of the original

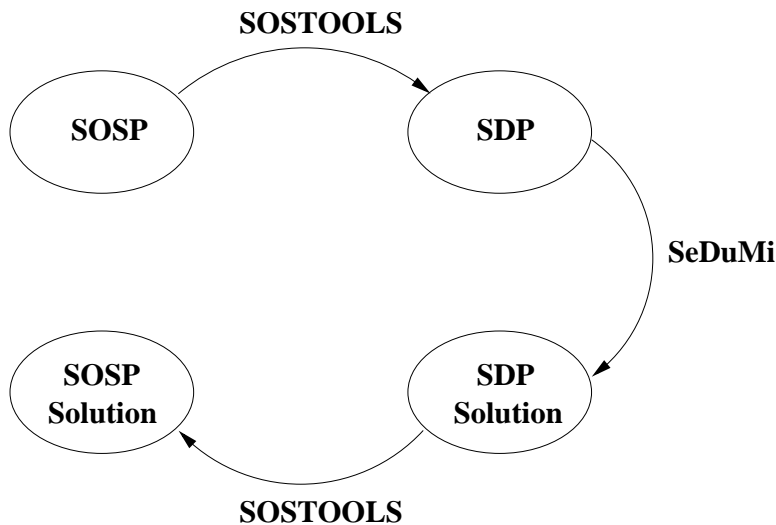


Figure 1.1: Diagram depicting relations between sum of squares program (SOSP), semidefinite program (SDP), SOSTOOLS, and SeDuMi.

SOSP. At present, it uses another free MATLAB add-on called SeDuMi [16] as the SDP solver. This whole process is depicted in Figure 1.1.

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: $+$, $-$, $*$, $/$, \wedge ; as well as operations such as differentiation, integration, point evaluation, etc. In addition, this provides the possibility of interfacing with the Maple³ symbolic engine and the Maple library (which is very advantageous).

The user interface has been designed to be as simple, as easy to use, and as transparent as possible, while keeping a large degree of flexibility. You create an SOSP by declaring SOSP variables (e.g., the $p_i(x)$'s in Section 1.1), adding SOSP constraints, setting the objective function, and so forth. After the program is created, you call one function to run the solver and finally retrieve solutions to the SOSP using another function. These steps will be presented in more details in Chapter 2.

Alternatively, “customized” functions for special problem classes (such as Lyapunov function computation, etc.) can be directly used, with no user programming whatsoever required. These are presented in the first three sections of Chapter 3.

1.3 System Requirements and Installation Instruction

To install and run SOSTOOLS, you need:

- MATLAB version 6.0 or later. Presumably, SOSTOOLS also works with MATLAB version 5, although this has not been tested yet.
- Symbolic Math Toolbox version 2.1.2.

³A registered trademark of Waterloo Maple Inc.

- SeDuMi version 1.05. This software and its documentation can be downloaded from Jos F. Sturm's homepage at <http://fewcal.kub.nl/sturm>. For information on how to install SeDuMi, you are referred to the installation instruction of the software.

SOSTOOLS can be easily run on a UNIX workstation or on a Windows PC desktop, or even a laptop. It utilizes 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 Chapter 3, except 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.

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>. Once you download the zip file, you should extract its contents to the directory where you want to install SOSTOOLS. In UNIX, you may use

```
unzip -U SOSTOOLS.nnn.zip -d your_dir
```

where **nnn** is the version number, and **your_dir** should be replaced by the directory of your choice. In Windows operating systems, you may use programs like Winzip to extract the files.

After this has been done, you must add the SOSTOOLS directory and its subdirectories to the MATLAB path. This is done in MATLAB by choosing the menus File --> Set Path --> Add with Subfolders ..., and then typing the name of SOSTOOLS main directory. This completes the SOSTOOLS installation.

1.4 Other Things You Need to Know

The directory in which you install SOSTOOLS contains several subdirectories. Two of them are:

- **sostools/docs** : containing this user's manual and license file
- **sostools/demos** : containing several demo files.

The demo files in the second subdirectory above implement the SOSPs corresponding to examples in Chapter 3.

Throughout this user's manual, we use the **typewriter** typeface to denote MATLAB variables and functions, MATLAB commands that you should type, and results given by MATLAB. MATLAB commands that you should type will also be denoted by the symbol **>>** before the commands. For example,

```
>> x = sin(1)
```

```
x =
```

```
0.8415
```

In this case, **x = sin(1)** is the command that you type, and **x = 0.8415** is the result given by MATLAB.

Finally, you can send bug reports, comments, and suggestions to sostools@cds.caltech.edu. Any feedback is greatly appreciated.

Chapter 2

Solving Sum of Squares Programs

SOSTOOLS can solve two kinds of sum of squares programs: the feasibility and optimization problems, as formulated in Chapter 1. To define and solve an SOSP using SOSTOOLS, you simply need to follow these steps:

1. Initialize the SOSP.
2. Declare the SOSP variables.
3. Define the SOSP constraints.
4. Set objective function (for optimization problems).
5. Call solver.
6. Get solutions.

In the next sections, we will describe each of these steps in detail. But first, we will look at how polynomials are represented and manipulated in SOSTOOLS.

2.1 Polynomial Representation and Manipulations

Polynomials in SOSTOOLS are represented as symbolic objects, using the MATLAB Symbolic Toolbox. Typically, a polynomial is created by first declaring its independent variables and then constructing it using the usual algebraic manipulations. For example, to create a polynomial $p(x, y) = 2x^2 + 3xy + 4y^4$, you declare the independent variables x and y by typing

```
>> syms x y;
```

and then construct $p(x, y)$ as follows:

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

```
p =
```

```
2*x^2+3*x*y+4*y^4
```

Polynomials such as the one created above can then be manipulated using the usual operators: $+$, $-$, $*$, $/$, $^$. Another operation which is particularly useful for control-related problems such

as Lyapunov function search is differentiation, which can be done using the function `diff`. For instance, to find the partial derivative $\frac{\partial p}{\partial x}$, you should type

```
>> dpdx = diff(p,x)

dpdx =

4*x+3*y
```

For other types of symbolic manipulations, we refer you to the manual and help comments of the Symbolic Math Toolbox.

2.2 Initializing a Sum of Squares Program

A sum of squares program is initialized using the command `sosprogram`. A vector containing independent variables in the program has to be given as an argument to this function. Thus, if the polynomials in our program have x and y as the independent variables, then we initialize the SOSP using

```
>> Program1 = sosprogram([x;y]);
```

The command above will initialize an empty SOSP called `Program1`.

2.3 Variable Declaration

After the program is initialized, the SOSP decision variables have to be declared. There are three functions used for this purpose, corresponding to variables of these types:

- Scalar decision variables.
- Polynomial variables.
- Sum of squares variables.

Each of them will be described in the following subsections.

2.3.1 Scalar Decision Variables

Scalar decision variables in an SOSP are meant to be unknown scalar constants. The variable γ in `sosdemo3.m` (see Section 3.3) is an example of such a variable. These variables can be declared either by specifying them when an SOSP is initialized with `sosprogram`, or by declaring them later using the function `sosdecvar`.

To declare decision variables, you must first create symbolic objects corresponding to your decision variables. This is performed using the functions `syms` or `sym` from the Symbolic Math Toolbox, in a way similar to the one you use to define independent variables in Section 2.1. As explained earlier, you can declare the decision variables when you initialize an SOSP, by giving them as a second argument to `sosprogram`. Thus, to declare variables named `a` and `b`, use the following command:

```
>> syms x y a b;
>> Program2 = sosprogram([x;y],[a;b]);
```

Alternatively, you may declare these variables after the SOSP is initialized, or add some other decision variables to the program, using the function `sosdecvar`. For example, the sequence of commands above is equivalent to

```
>> syms x y a b;
>> Program3 = sosprogram([x;y]);
>> Program3 = sosdecvar(Program3,[a;b]);
```

and also equivalent to

```
>> syms x y a b;
>> Program4 = sosprogram([x;y],a);
>> Program4 = sosdecvar(Program4,b);
```

2.3.2 Polynomial Variables

Polynomial variables in a sum of squares program are simply polynomials with unknown coefficients (e.g. $p_1(x)$ in the feasibility problem formulated in Chapter 1). Polynomial variables can obviously be created by declaring its unknown coefficients as decision variables, and then constructing the polynomial itself via some algebraic manipulations. For example, to create a polynomial variable $v(x, y) = ax^2 + bxy + cy^2$, where a , b , and c are the unknowns, you can use the following commands:

```
>> Program5 = sosdecvar(Program5,[a;b;c]);
>> v = a*x^2 + b*x*y + c*y^2;
```

However, such an approach would be inefficient for polynomials with many coefficients. In such a case, you should use the function `sospolyvar` to declare a polynomial variable:

```
>> [Program6,v] = sospolyvar(Program6,[x^2; x*y; y^2]);
```

In this case v will be

```
>> v

v =

coeff_1*x^2+coeff_2*x*y+coeff_3*y^2
```

We see that `sospolyvar` automatically creates decision variables corresponding to monomials in the vector which is given as the second input argument to it, and then constructs a polynomial variable from these coefficients and monomials. This polynomial variable is returned as the second output argument of `sospolyvar`.

NOTE:

1. `sospolyvar` and `soassosvar` (see Section 2.3.4) name the unknown coefficients in a polynomial/SOS variable by `coeff_nnn`, where `nnn` is a number. Names that begin with `coeff_` are reserved for this purpose, and therefore must not be used elsewhere.
2. By default, the decision variables `coeff_nnn` created by `sospolyvar` or `soassosvar` will only be available in the function workspace, and therefore cannot be manipulated in the MATLAB workspace. Sometimes it is desirable to have these decision variables available in the MATLAB workspace, such as when we want to set an objective function of an SOSP

that involves one or more of these variables. In this case, a third argument `'wscoeff'` has to be given to `sospolyvar` or `soisosvar`. For example, using

```
>> [Program7,v] = sospolyvar(Program7,[x^2; x*y; y^2],'wscoeff');
>> v
```

```
v =
```

```
coeff_1*x^2+coeff_2*x*y+coeff_3*y^2
```

you will be able to directly use `coeff_1` and `coeff_2` in the MATLAB workspace, as shown below.

```
>> w = coeff_1+coeff_2
```

```
w =
```

```
coeff_1+coeff_2
```

3. SOSTOOLS requires monomials that are given as the second input argument to `sospolyvar` and `soisosvar` to be unique, meaning that there are no repeated monomials.

2.3.3 An Aside: Constructing Vectors of Monomials

We have seen in the previous subsection that for declaring SOSP variables using `sospolyvar` we need to construct a vector whose entries are monomials. While this can be done by creating the individual monomials and arranging them as a vector, SOSTOOLS also provides a function, named `monomials`, that can be used to construct a column vector of monomials with some pre-specified degrees. This will be particularly useful when the vector contains a lot of monomials. The function takes two arguments: the first argument is a vector containing all independent variables in the monomials, and the second argument is a vector whose entries are the degrees of monomials that you want to create. As an example, to construct a vector containing all monomials in x and y of degree 1, 2, and 3, type the following command:

```
>> VEC = monomials([x; y],[1 2 3])
```

```
VEC =
```

```
[    x]
[    y]
[  x^2]
[  x*y]
[  y^2]
[  x^3]
[ x^2*y]
[ x*y^2]
[  y^3]
```

We clearly see that `VEC` contains all monomials in x and y of degree 1, 2, and 3.

2.3.4 Sum of Squares Variables

Sum of squares variables are also polynomials with unknown coefficients, similar to polynomial variables described in Section 2.3.2. The difference is, as its name suggests, that an SOS variable is constrained to be an SOS. This is imposed by internally representing an SOS variable in the Gram matrix form (cf. Section 1.1),

$$p(x) = Z^T(x)QZ(x) \quad (2.1)$$

and requiring the coefficient matrix Q to be positive semidefinite.

To declare an SOS variable, you must use the function `soossosvar`. The monomial vector $Z(x)$ in (2.1) has to be given as the second input argument to the function. Like `sospolyvar`, this function will automatically declare all decision variables corresponding to the matrix Q . For example, to declare an SOS variable

$$p(x, y) = \begin{bmatrix} x \\ y \end{bmatrix}^T Q \begin{bmatrix} x \\ y \end{bmatrix}, \quad (2.2)$$

type

```
>> [Program8,p] = soossosvar(Program8,[x; y]);
```

where the second output argument is the name of the variable. In this example, the coefficient matrix

$$Q = \begin{bmatrix} \text{coeff_1} & \text{coeff_3} \\ \text{coeff_2} & \text{coeff_4} \end{bmatrix} \quad (2.3)$$

will be created by the function. When this matrix is substituted into the expression for $p(x, y)$, we obtain

$$p(x, y) = \text{coeff_1}x^2 + (\text{coeff_2} + \text{coeff_3})xy + \text{coeff_4}y^2, \quad (2.4)$$

which is exactly what `soossosvar` returns:

```
>> p
```

```
p =
```

```
coeff_4*y^2+(coeff_2+coeff_3)*x*y+coeff_1*x^2
```

We would like to note that at first the coefficient matrix does not appear to be symmetric, especially because the number of decision variables (which seem to be independent) is the same as the number of entries in the coefficient matrix. However, some constraints are internally imposed by the semidefinite programming solver SeDuMi (which is used by SOSTOOLS) on some of these decision variables, such that the solution matrix obtained by the solver will be symmetric. The primal formulation of a semidefinite program in SeDuMi uses n^2 decision variables to represent an $n \times n$ positive semidefinite matrix, which is the reason why SOSTOOLS also uses n^2 decision variables for its $n \times n$ coefficient matrices.

2.4 Adding Constraints

Sum of squares program constraints such as (1.3)–(1.4) are added to a sum of squares program using the functions `soseq` and `sosineq`.

2.4.1 Equality Constraints

For adding an equality constraint to a sum of squares program, you must use the function `soseq`. As an example, assume that `p` is an SOSP variable, then

```
>> Program9 = soseq(Program9,diff(p,x)-x^2);
```

will add the equality constraint

$$\frac{\partial p}{\partial x} - x^2 = 0 \quad (2.5)$$

to `Program9`.

2.4.2 Inequality Constraints

Inequality constraints are declared using the function `soseq`, whose basic syntax is similar to `soseq`. For example, type

```
>> Program1 = sosineq(Program1,diff(p,x)-x^2);
```

to add the inequality constraint

$$\frac{\partial p}{\partial x} - x^2 \geq 0^1. \quad (2.6)$$

However, several differences do exist. In particular, a third argument can be given to `sosineq` to handle the following cases:

- When there is only one independent variable in the SOSP (i.e., if the polynomials are univariate), a third argument can be given to specify the range of independent variable for which the inequality constraint has to be satisfied. For instance, assume that `p` and `Program2` are respectively univariate polynomial and univariate SOSP, then

```
>> Program2 = sosineq(Program2,diff(p,x)-x^2,[-1 2]);
```

will add the constraint

$$\frac{\partial p}{\partial x} - x^2 \geq 0, \quad \text{for } -1 \leq x \leq 2 \quad (2.7)$$

to the SOSP. See Sections 3.7 and 3.8 for application examples where this option is used.

- When the left side of the inequality is a high degree sparse polynomial (i.e., containing a few nonzero terms), it is computationally more efficient to impose the SOS condition using a reduced set of monomials (see [13]) in the Gram matrix form. This will result in a smaller size semidefinite program, which is easier to solve. By default, SOSTOOLS does not try to obtain this optimal reduced set of monomials, since this itself takes an additional amount of computational effort (however, SOSTOOLS always does some reasonably efficient and computationally cheap heuristics to reduce the set of monomials). The optimal reduced set of monomials will be computed and used only if a third argument `'sparse'` is given to `sosineq`, as illustrated by the following command,

```
>> Program3 = sosineq(Program3,x^16+2*x^8*y^2+y^4,'sparse');
```

¹We remind you that $\frac{\partial p}{\partial x} - x^2 \geq 0$ has to be interpreted as $\frac{\partial p}{\partial x} - x^2$ being a sum of squares. See the discussion in Section 1.1.

which tests whether or not $x^{16} + 2x^8y^2 + y^4$ is a sum of squares.

2.5 Setting Objective Function

The function `sossetobj` is used to set the objective function in an optimization problem. The objective function has to be a linear function of the decision variables, and will be minimized by the solver. For instance, if `a` and `b` are symbolic decision variables in an SOSP named `Program4`, then

```
>> Program4 = sossetobj(Program4,a-b);
```

will set

$$\text{minimize } (a - b) \quad (2.8)$$

as the objective of `Program4`.

Sometimes you may want to minimize an objective function that contains one or more reserved variables `coeff_nnn`, which are created by `sospolyvar` or `soosvar`. These variables are not individually available in the MATLAB workspace by default. You must give the argument `'wscoeff'` to the corresponding `sospolyvar` or `soosvar` call in order to have these variables available in the MATLAB workspace. This has been described in Section 2.3.2.

2.6 Calling Solver

A sum of squares program that has been completely defined can be solved using `soosolve.m`. For example, to solve `Program5`, the command is

```
>> Program5 = soosolve(Program5);
```

This function converts the SOSP into an equivalent SDP, calls SeDuMi, and converts the result given by SeDuMi back into solutions to the original SOSP.

Typical output that you will get on your screen is shown in Figure 2.1. Several things deserve some explanation:

- **Size** indicates the size of the resulting SDP.
- **Residual norm** is the norm of numerical error in the solution.
- **pinf=1** or **dinf=1** indicate primal or dual infeasibility.
- **numerr=1** gives a warning of numerical inaccuracy. This is usually accompanied by large **Residual norm**. On the other hand, **numerr=2** is a sign of complete failure because of numerical problem.

2.7 Getting Solutions

After your sum of squares program has been solved, you can get the solutions to the program using `sogetsol.m`. The function takes two arguments, where the first argument is the SOSP, and the second is a symbolic expression, which typically will be an SOSP variable. All decision

Size: 10 5

SeDuMi 1.05 by Jos F. Sturm, 1998, 2001.

Alg = 2: xz-corrector, Step-Differentiation, theta = 0.250, beta = 0.500

eqs m = 5, order n = 9, dim = 13, blocks = 3

nnz(A) = 13 + 0, nnz(ADA) = 11, nnz(L) = 8

it	b*y	gap	delta	rate	t/tP*	t/tD*	feas	cg	cg
0		7.00E+000	0.000						
1	-3.03E+000	1.21E+000	0.000	0.1734	0.9026	0.9000	0.64	1	1
2	-4.00E+000	6.36E-003	0.000	0.0052	0.9990	0.9990	0.94	1	1
3	-4.00E+000	2.19E-004	0.000	0.0344	0.9900	0.9786	1.00	1	1
4	-4.00E+000	1.99E-005	0.234	0.0908	0.9459	0.9450	1.00	1	1
5	-4.00E+000	2.37E-006	0.000	0.1194	0.9198	0.9000	0.91	1	2
6	-4.00E+000	3.85E-007	0.000	0.1620	0.9095	0.9000	1.00	3	3
7	-4.00E+000	6.43E-008	0.000	0.1673	0.9000	0.9034	1.00	4	4
8	-4.00E+000	2.96E-009	0.103	0.0460	0.9900	0.9900	1.00	3	4
9	-4.00E+000	5.16E-010	0.000	0.1743	0.9025	0.9000	1.00	5	5
10	-4.00E+000	1.88E-011	0.327	0.0365	0.9900	0.9905	1.00	5	5

iter seconds digits c*x b*y

10 0.4 Inf -4.0000000000e+000 -4.0000000000e+000

|Ax-b| = 9.2e-011, [Ay-c]_+ = 1.1E-011, |x| = 9.2e+000, |y| = 6.8e+000

Max-norms: ||b||=2, ||c|| = 5,

Cholesky |add|=0, |skip| = 1, ||L.L|| = 2.00001.

Residual norm: 9.2143e-011

cpusec: 0.3900
 iter: 10
 feasratio: 1.0000
 pinf: 0
 dinf: 0
 numerr: 0

Figure 2.1: Output of SOSTOOLS (some is generated by SeDuMi).

variables in this expression will be substituted by the numerical values obtained as the solution to the corresponding SDP. Typing

```
>> SOLp1 = sosgetsol(Program6,p1);
```

where `p1` is an polynomial variable, for example, will return in `SOLp1` a polynomial with some numerical coefficients, which is obtained by substituting all decision variables in `p1` by the numerical solution to the SOSP `Problem6`, provided this SOSP has been solved beforehand.

By default, all the numerical values returned by `sosgetsol` will have a five-digit presentation. If needed, this can be changed by giving the desired number of digits as the third argument to `sosgetsol`, such as

```
>> SOLp1 = sosgetsol(Program7,p1,12);
```

which will return the numerical solution with twelve digits. Note however, that this does not change the accuracy of the SDP solution, but only its presentation.

Chapter 3

Applications of Sum of Squares Programming

In this chapter we present some problems that can be solved using SOSTOOLS. The majority of the examples here are from [9], except when noted otherwise. Many more application examples and customized files will be included in the near future.

Note: For some of the problems here (in particular, copositivity and equality-constrained ones such as MAXCUT) the SDP formulations obtained by SOSTOOLS are not the most efficient ones, as the special structure of the resulting polynomials is not fully exploited in the current distribution. This will be incorporated in the next release of SOSTOOLS, whose development is already in progress.

3.1 Sum of Squares Test

As mentioned in Chapter 1, testing if a polynomial $p(x)$ is nonnegative for all $x \in \mathbb{R}^n$ is a hard problem, but can be relaxed to the problem of checking if $p(x)$ is an SOS. This can be solved using SOSTOOLS, by casting it as a feasibility problem.

SOSDEMO1:

Given a polynomial $p(x)$, determine if

$$p(x) \geq 0 \tag{3.1}$$

is feasible.

Notice that even though there are no explicit decision variables in this SOSP, we still need to solve a semidefinite programming problem to decide if the program is feasible or not.

The MATLAB code for solving this SOSP can be found in `sosdemo1.m`, shown in Figure 3.1 where we consider $p(x) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$. Since the program is feasible, it follows that $p(x) \geq 0$.

In addition, SOSTOOLS provides a function named `findsos` to find an SOS decomposition of a polynomial $p(x)$. This function returns the coefficient matrix Q and the monomial vector $Z(x)$ which are used in the Gram matrix form. For the same polynomial as above, we may as well type

```
>> [Q,Z] = findsos(p);
```

to find Q and $Z(x)$ such that $p(x) = Z^T(x)QZ(x)$. If $p(x)$ is not a sum of squares, the function will return empty Q and Z .

```

% SOSDEMO1 --- Sum of Squares Test
% Section 3.1 of SOSTOOLS User's Manual
%

clear; maple clear; echo on;
syms x1 x2;
varitable = [x1, x2];
% =====
% First, initialize the sum of squares program
prog = sosprogram(varitable); % No decision variables.

% =====
% Next, define the inequality

% p(x1,x2) >= 0
p = 2*x1^4 + 2*x1^3*x2 - x1^2*x2^2 + 5*x2^4;
prog = sosineq(prog,p);

% =====
% And call solver
prog = sossolve(prog);

% =====
% Program is feasible, thus p(x1,x2) is an SOS.
echo off;

```

Figure 3.1: Sum of squares test – `sosdemo1.m`

3.2 Lyapunov Function Search

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 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 to search for Lyapunov function using SOSTOOLS. Consider the system

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -x_1^3 - x_1x_3^2 \\ -x_2 - x_1^2x_2 \\ -x_3 - \frac{3x_3}{x_3^2+1} + 3x_1^2x_3 \end{bmatrix}, \quad (3.2)$$

with an equilibrium at the origin. Notice that the linearization of (3.2) 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

$$\begin{aligned} V - \epsilon(x_1^2 + x_2^2 + x_3^2) &\geq 0, \\ -\frac{\partial V}{\partial x_1}\dot{x}_1 - \frac{\partial V}{\partial x_2}\dot{x}_2 - \frac{\partial V}{\partial x_3}\dot{x}_3 &\geq 0. \end{aligned} \quad (3.3)$$

The first inequality, with ϵ being any constant greater than zero, is needed to guarantee positive definiteness of $V(x)$. However, notice that \dot{x}_3 is a rational function, and therefore (3.3) is not a valid SOSP constraint. But since $x_3^2 + 1 > 0$ for any x_3 , we can just reformulate (3.3) as

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

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

SOSDEMO2:

Find a polynomial $V(x)$, such that

$$V - (x_1^2 + x_2^2 + x_3^2) \geq 0, \quad (3.4)$$

$$-\frac{\partial V}{\partial x_1}(x_3^2 + 1)\dot{x}_1 - \frac{\partial V}{\partial x_2}(x_3^2 + 1)\dot{x}_2 - \frac{\partial V}{\partial x_3}(x_3^2 + 1)\dot{x}_3 \geq 0. \quad (3.5)$$

The MATLAB code is available in `sosdemo2.m`, and is also shown in Figure 3.2. The result given by SOSTOOLS is

$$V(x) = 5.5489x_1^2 + 4.1068x_2^2 + 1.7945x_3^2.$$

The function `findlyap` is provided by SOSTOOLS and can be used to compute a polynomial Lyapunov function for a dynamical system with polynomial vector field. This function take three arguments, where the first argument is the vector field of the system, the second argument is the ordering of the independent variables, and the third argument is the degree of the Lyapunov function. Thus, for example, to compute a quadratic Lyapunov function $V(x)$ for the system

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

type

```
>> syms x1 x2;
>> V = findlyap([-x1^3+x2; -x1-x2],[x1; x2],2)
```

If no such Lyapunov function exists, the function will return an empty V.

3.3 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 [15], where an SOS-based approach was first used. A relaxation method can be formulated as follows. Suppose that there exists a scalar γ such that

$$f(x) - \gamma \geq 0 \text{ (is an SOS),}$$

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 [4], which is given by

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

The SOSP for this problem is

SOSDEMO3:

Minimize $-\gamma$, such that

$$(f(x) - \gamma) \geq 0. \tag{3.6}$$

Figure 3.3 depicts 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 $x_1 = 0$, $x_2 = -1$.

The function `findbound` is provided by SOSTOOLS and can be used to find a global lower bound for a polynomial. This function takes just one argument, the polynomial to be minimized. The function will return a lower bound (which may be possibly be $-\infty$), a vector with the variables of the polynomial, and, if an additional condition is satisfied (the dual solution has rank one), also a point where the bound is achieved. Thus, for example, to compute a global minimum for the polynomial:

$$F = (a^4 + 1)(b^4 + 1)(c^4 + 1)(d^4 + 1) + 2a + 3b + 4c + 5d,$$

you would type:

```
>> syms a b c d;
>> F = (a^4+1)*(b^4+1)*(c^4+1)*(d^4+1) + 2*a + 3*b + 4*c + 5*d;
>> [bnd,vars,xopt] = findbound(F)
```

For this problem (a polynomial of total degree 16 in four variables), SOSTOOLS returns a certified lower bound (`bnd=-7.759027`) and also the corresponding optimal point in less than thirty seconds.

```

% SOSDEMO2 --- Lyapunov Function Search
% Section 3.2 of SOSTOOLS User's Manual
%

clear; maple clear; echo on;
syms x1 x2 x3;
vars = [x1; x2; x3];

% Constructing the vector field dx/dt = f
f = [-x1^3-x1*x3^2;
     -x2-x1^2*x2;
     -x3+3*x1^2*x3-3*x3/(x3^2+1)];

% =====
% First, initialize the sum of squares program
prog = sosprogram(vars);

% =====
% The Lyapunov function V(x):
[prog,V] = sospolyvar(prog,[x1^2; x2^2; x3^2],'wscoeff');

% =====
% Next, define SOS constraints

% Constraint 1 : V(x) - (x1^2 + x2^2 + x3^2) >= 0
prog = sosineq(prog,V-(x1^2+x2^2+x3^2));

% Constraint 2: -dV/dx*(x3^2+1)*f >= 0
expr = -(diff(V,x1)*f(1)+diff(V,x2)*f(2)+diff(V,x3)*f(3))*(x3^2+1);
prog = sosineq(prog,expr);

% =====
% And call solver
prog = sossolve(prog);

% =====
% Finally, get solution
SOLV = sosgetsol(prog,V)

echo off;

```

Figure 3.2: Lyapunov function search – sosdemo2.m

```

% SOSDEMO3 --- Bound on Global Extremum
% Section 3.3 of SOSTOOLS User's Manual
%

clear; maple clear; echo on;
syms x1 x2 gam;
varitable = [x1, x2];

% =====
% First, initialize the sum of squares program
prog = sosprogram(varitable);

% =====
% Declare decision variable gam
prog = sosdecvar(prog,gam);

% =====
% Next, define SOSP constraints

% Constraint :  $r(x) \cdot (f(x) - \text{gam}) \geq 0$ 
%  $f(x)$  is the Goldstein-Price function
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);

prog = sosineq(prog,(f-gam));

% =====
% Set objective : maximize gam
prog = sossetobj(prog,-gam);

% =====
% And call solver
prog = sossolve(prog);

% =====
% Finally, get solution
SOLgamma = sosgetsol(prog,gam)
echo off

```

Figure 3.3: Bound on global extremum – sosdemo3.m

3.4 Matrix Copositivity

The matrix copositivity problem can be stated as follows:

Given a matrix $J \in \mathbb{R}^{n \times n}$, check if it is copositive, i.e. if $y^T J y \geq 0$ for all $y \in \mathbb{R}^n$, $y_i \geq 0$.

It is known that checking copositivity of a matrix is a co-NP complete problem. However, there exist computationally tractable relaxations for copositivity checking. One relaxation [9] is performed by writing $y_i = x_i^2$, and checking if

$$\left(\sum_{i=1}^n x_i^2 \right)^m \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \end{bmatrix}^T J \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \end{bmatrix} \triangleq R(x) \quad (3.7)$$

is an SOS.

Now consider the matrix

$$J = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \end{bmatrix}.$$

It is known that the matrix above is copositive. This will be proven using SOSTOOLS. For this purpose, we have the following SOSP.

SOSDEMO4:

Determine if

$$R(x) \geq 0, \quad (3.8)$$

is feasible, where $R(x)$ is as in (3.7).

Choosing $m = 0$ does not prove that J is copositive. However, DEMO4 is feasible for $m = 1$, and therefore it proves that J is copositive. The MATLAB code that implements this is given in `sosdemo4.m` and shown in Figure 3.4.

3.5 Upper Bound of Structured Singular Value

Now we will show how SOSTOOLS can be used for computing upper bound of structured singular value μ , a crucial object in robust control theory (see e.g. [3, 8]). The following conditions can be derived from Proposition 8.25 of [3] and Theorem 6.1 of [9]. Given a matrix $M \in \mathbb{C}^{n \times n}$ and structured scalar uncertainties

$$\Delta = \text{diag}(\delta_1, \delta_2, \dots, \delta_n), \quad \delta_i \in \mathbb{C},$$

the structured singular value $\mu(M, \Delta)$ is less than γ , if there exists solutions $Q_i \geq 0 \in \mathbb{R}^{2n \times 2n}$, $T_i \in \mathbb{R}^{2n \times 2n}$ and $r_{ij} \geq 0$ such that

$$-\sum_{i=1}^n Q_i(x) A_i(x) - \sum_{1 \leq i < j \leq n} r_{ij} A_i(x) A_j(x) + I(x) \geq 0, \quad (3.9)$$

```

% SOSDEMO4 --- Matrix Copositivity
% Section 3.4 of SOSTOOLS User's Manual
%

clear; maple clear; echo on;
syms x1 x2 x3 x4 x5;
varitable = [x1; x2; x3; x4; x5];

% The matrix under consideration
J = [1 -1 1 1 -1;
     -1 1 -1 1 1;
     1 -1 1 -1 1;
     1 1 -1 1 -1;
     -1 1 1 -1 1];

% =====
% First, initialize the sum of squares program
prog = sosprogram(varitable); % No decision variables.

% =====
% Next, define SOS constraints

% Constraint :  $r(x) \cdot J(x) - p(x) = 0$ 
J = [x1^2 x2^2 x3^2 x4^2 x5^2]*J*[x1^2; x2^2; x3^2; x4^2; x5^2];
r = x1^2 + x2^2 + x3^2 + x4^2 + x5^2;

prog = sosineq(prog,r*J);

% =====
% And call solver
prog = sossolve(prog,[],1e-7);

% =====
% Program is feasible. The matrix J is copositive.

echo off

```

Figure 3.4: Matrix copositivity – sosdemo4.m

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

$$Q_i(x) = x^T Q_i x, \quad (3.10)$$

$$I(x) = -\sum_{i=1}^{2n} x_i^2, \quad (3.11)$$

$$A_i(x) = x^T A_i x, \quad (3.12)$$

$$A_i = \begin{bmatrix} \operatorname{Re}(H_i) & -\operatorname{Im}(H_i) \\ \operatorname{Im}(H_i) & \operatorname{Re}(H_i) \end{bmatrix}, \quad (3.13)$$

$$H_i = M^* e_i^* e_i M - \gamma^2 e_i^* e_i, \quad (3.14)$$

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

Thus, the SOSP for this problem can be formulated as follows.

SOSDEMO5:

Choose a fixed value of γ . For $I(x)$ and $A_i(x)$ as described in (3.11) – (3.14), find sums of squares

$$\begin{aligned} Q_i(x) &= x^T Q_i x, & \text{for } i = 1, \dots, 2n, \\ r_{ij} &\geq 0 & \text{(zero order SOS), for } 1 \leq i < j \leq 2n, \end{aligned}$$

such that (3.9) is satisfied.

The optimal value of γ can be found for example by bisection. In `sosdemo5.m` (Figures 3.5–3.6), we consider the following M (from [8]):

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

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 `sosdemo5.m`, we can prove that $\mu(M, \Delta) < 0.8724$.

3.6 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

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

or equivalently as a constrained optimization

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

```

% SOSDEMO5 --- Upper bound for the structured singular value mu
% Section 3.5 of SOSTOOLS User's Manual
%

clear; maple clear; echo on;
syms x1 x2 x3 x4 x5 x6 x7 x8;
varitable = [x1; x2; x3; x4; x5; x6; x7; x8];

% The matrix under consideration
alpha = 3 + sqrt(3);
beta = sqrt(3) - 1;
a = sqrt(2/alpha);
b = 1/sqrt(alpha);
c = b;
d = -sqrt(beta/alpha);
f = (1 + i)*sqrt(1/(alpha*beta));
U = [a 0; b b; c i*c; d f];
V = [0 a; b -b; c -i*c; -i*f -d];
M = U*V';

% Constructing A(x)'s
gam = 0.8724;
Z = monomials(varitable,1);
for i = 1:4
    H = M(i,:)'*M(i,:) - (gam^2)*sparse(i,i,1,4,4,1);
    H = [real(H) -imag(H); imag(H) real(H)];
    A{i} = (Z.')*H*Z;
end;

% =====
% Initialize the sum of squares program
prog = sosprogram(varitable);

% =====
% Define SOSP variables

% -- Q(x)'s -- : sums of squares
% Monomial vector: [x1; ... x8]
for i = 1:4
    [prog,Q{i}] = sossosvar(prog,Z);
end;

```

Figure 3.5: Upper bound of structured singular value – sosdemo5.m, part 1 of 2.


```

% -- r's -- : constant sum of squares
Z = monomials(vartable,0);
r = cell(4,4);
for i = 1:4
    for j = (i+1):4
        [prog,r{i,j}] = sossosvar(prog,Z,'wscoeff');
    end;
end;

% =====
% Next, define SOSP constraints

% Constraint : -sum(Qi(x)*Ai(x)) - sum(rij*Ai(x)*Aj(x)) + I(x) >= 0
expr = 0;
% Adding term
for i = 1:4
    expr = expr - A{i}*Q{i};
end;
for i = 1:4
    for j = (i+1):4
        expr = expr - A{i}*A{j}*r{i,j};
    end;
end;
% Constant term: I(x) = -(x1^4 + ... + x8^4)
I = -sum(vartable.^4);
expr = expr + I;

prog = sosineq(prog,expr);

% =====
% And call solver
prog = sossolve(prog);

% =====
% Program is feasible, thus 0.8724 is an upper bound for mu.

echo off

```

Figure 3.6: Upper bound of structured singular value – sosdemo5.m, part 2 of 2.

Here w_{ij} is the weight of 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 exists sum of squares $p_1(x)$ and polynomials $p_2(x), \dots, p_{n+1}(x)$ such that

$$p_1(x)(\gamma - f(x)) + \sum_{i=1}^n (p_{i+1}(x)(x_i^2 - 1)) - (\gamma - f(x))^2 \geq 0. \quad (3.15)$$

This can be proved by a contradiction. Suppose there exists $x \in \{-1, 1\}^n$ such that $f(x) > \gamma$. Then the first term in (3.15) will be negative, the terms under summation will be zero, and the last term will be negative. Thus we have a contradiction.

For `sosdemo6.m` (see Figure 3.7), we consider the 5-cycle, i.e., a graph with 5 nodes and 5 edges forming a closed chain. The number of cut is given by

$$f(x) = 2.5 - 0.5x_1x_2 - 0.5x_2x_3 - 0.5x_3x_4 - 0.5x_4x_5 - 0.5x_5x_1. \quad (3.16)$$

Our SOSp is as follows.

SOSDEMO6:

Choose a fixed value of γ . For $f(x)$ given in (3.16), find

$$\begin{aligned} \text{sum of squares } p_1(x) &= \begin{bmatrix} 1 \\ x \end{bmatrix}^T Q \begin{bmatrix} 1 \\ x \end{bmatrix} \\ \text{polynomials } p_{i+1}(x) &\text{ of degree 2, for } i = 1, \dots, n \end{aligned}$$

such that (3.15) is satisfied.

Using `sosdemo6.m`, we can show that $f(x) \leq 4$. Four is indeed the maximum cut for 5-cycle.

3.7 Chebyshev Polynomials

This example illustrates the `sosineq` range-specification option for univariate polynomials (see Section 2.4.2), and is based on a well-known extremal property of the Chebyshev polynomials. Consider the optimization problem:

SOSDEMO7:

Let $p_n(x)$ be a univariate polynomial of degree n , with γ being the coefficient of x^n .

Maximize γ , subject to:

$$|p_n(x)| \leq 1, \quad \forall x \in [-1, 1].$$

The absolute value constraint can be easily rewritten using two inequalities, namely:

$$\begin{aligned} 1 + p_n(x) &\geq 0 \\ 1 - p_n(x) &\geq 0 \end{aligned}, \quad \forall x \in [-1, 1].$$

```

% SOSDEMO6 --- MAX CUT
% Section 3.6 of SOSTOOLS User's Manual

clear; maple clear; echo on
syms x1 x2 x3 x4 x5;
varitable = [x1; x2; x3; x4; x5];

% Number of cuts
f = 2.5 - 0.5*x1*x2 - 0.5*x2*x3 - 0.5*x3*x4 - 0.5*x4*x5 - 0.5*x5*x1;

% Boolean constraints
bc{1} = x1^2 - 1 ;
bc{2} = x2^2 - 1 ;
bc{3} = x3^2 - 1 ;
bc{4} = x4^2 - 1 ;
bc{5} = x5^2 - 1 ;

% =====
% First, initialize the sum of squares program
prog = sosprogram(varitable);

% Then define SOS variables

% -- p1(x) -- : sum of squares
% Monomial vector: 5 independent variables, degree <= 1
Z = monomials(varitable,[0 1]);
[prog,p{1}] = sossosvar(prog,Z);

% -- p2(x) ... p6(x) : polynomials
% Monomial vector: 5 independent variables, degree <= 2
Z = monomials(varitable,0:2);
for i = 1:5
    [prog,p{1+i}] = sospolyvar(prog,Z);
end;

% Next, define SOS constraints

% Constraint : p1(x)*(gamma - f(x)) + p2(x)*bc1(x)
%              + ... + p6(x)*bc5(x) - (gamma-f(x))^2 >= 0
gamma = 4;
expr = p{1}*(gamma-f);
for i = 2:6
    expr = expr + p{i}*bc{i-1};
end;
expr = expr - (gamma-f)^2;

prog = sosineq(prog,expr);

% And call solver
prog = sossolve(prog);

% Program is feasible, thus 4 is an upper bound for the cut.
echo off

```

Figure 3.7: MAX CUT – sosdemo6.m.

The optimal solution is $\gamma^* = 2^{n-1}$, with $p_n^*(x) = \arccos(\cos nx)$ being the n -th Chebyshev polynomial of the first kind.

Using `sosdemo7.m` (shown in Figure 3.8), the problem can be easily solved for small values of n (say $n \leq 13$), with SeDuMi aborting with numerical errors for larger values of n . This is due to the ill-conditioning of the problem (at least, when using the standard monomial basis).

3.8 Bounds in Probability

In this example we illustrate how the sums of squares programming machinery can be used to obtain bounds on the worst-case probability of an event, given some moment information on the distribution. We refer the reader to the work of Bertsimas and Popescu [1] for a detailed discussion of the general case, as well as references to earlier related work.

Consider an unknown arbitrary probability distribution $q(x)$, with support in $x \in [0, 5]$. We know that its mean μ is equal to 1, and its standard deviation σ is equal to $1/2$. The question is: what is the worst-case probability, over all feasible distributions, of a sample having $x \geq 4$?

Using the tools in [1], it can be shown that a bound on (or in this case, the optimal) worst case value can be found by solving the optimization problem:

SOSDEMO8:

Minimize $am_0 + bm_1 + cm_2$, subject to

$$\begin{cases} a + bx + cx^2 \geq 0, & \forall x \in [0, 5] \\ a + bx + cx^2 \geq 1, & \forall x \in [4, 5], \end{cases}$$

where $m_0 = 1$, $m_1 = \mu$, and $m_2 = \mu^2 + \sigma^2$.

The optimization problem above is clearly an SOSp, and is implemented in `sosdemo8.m` (shown in Figure 3.9).

The optimal bound, computed from the optimization problem, is equal to $1/37$, with the optimal polynomial being $a + bx + cx^2 = \left(\frac{12x-11}{37}\right)^2$. The worst case probability distribution is atomic:

$$q^*(x) = \frac{36}{37} \delta\left(x - \frac{11}{12}\right) + \frac{1}{37} \delta(x - 4).$$

All these values (actually, their floating point approximations) can be obtained from the numerical solution obtained using SOSTOOLS.

```

% SOSDEMO7 --- Chebyshev polynomials
% Section 3.7 of SOSTOOLS User's Manual

clear; maple clear; echo on;

ndeg = 8;    % Degree of Chebyshev polynomial

syms x gam;

% =====
% First, initialize the sum of squares program
prog = sosprogram([x],[gam]);

% =====
% Create the polynomial P
Z = monomials(x,[0:ndeg-1]);
[prog,P1] = sospolyvar(prog,Z);
P = P1 + gam * x^ndeg;          % The leading coeff of P is gam

% =====
% Next, impose the inequalities
prog = sosineq(prog, 1 - P, [-1, 1]);
prog = sosineq(prog, 1 + P, [-1, 1]);

% =====
% And set the objective
prog = sossetobj(prog, -gam);

% =====
% Then solve the program
prog = sossolve(prog);

% =====
% Finally, get solution
SOLV = sosgetsol(prog, P)
GAM = sosgetsol(prog, gam)

echo off

```

Figure 3.8: Chebyshev polynomials – `sosdemo7.m`.

```

% SOSDEMO8 --- Bounds in Probability
% Section 3.8 of SOSTOOLS User's Manual

clear; maple clear; echo on;
syms x a b c;

% The probability adds up to one.
m0 = 1 ;

% Mean
m1 = 1 ;

% Variance
sig = 1/2 ;

% E(x^2)
m2 = sig^2+m1^2;

% Support of the random variable
R = [0,5];

% Event whose probability we want to bound
E = [4,5];

% =====
% Constructing and solving the SOS program
prog = sosprogram([x],[a,b,c]);

P = a + b*x + c*x^2 ;

% Nonnegative on the support
prog = sosineq(prog, P ,R);

% Greater than one on the event
prog = sosineq(prog,P-1,E);

% The bound
bnd = a * m0 + b * m1 + c * m2 ;

% Objective: minimize the bound
prog = sossetobj(prog, bnd) ;

prog = sossolve(prog);

% =====
% Get solution
BND = sosgetsol(prog,bnd,16)
PP = sosgetsol(prog,P);
echo off;

```

Figure 3.9: Bounds in probability – sosdemo8.m.

Chapter 4

List of Functions

For the moment, refer to the help comments of each Matlab function. They can be displayed using `help (function name)`. We plan to add more information to this chapter in the future.

```
% SOSTOOLS --- Sum of Squares Toolbox
% Version 1.00, 11 April 2002.
%
% Monomial vectors construction:
%   MONOMIALS    --- Construct a vector of monomials with
%                   prespecified degrees.
%
% General purpose sum of squares program (SOSP) solver:
%   SOSPROGRAM   --- Initialize a new SOSP.
%   SOSDECVAR    --- Declare new decision variables in an SOSP.
%   SOSPOLYVAR   --- Declare a new polynomial variable in an SOSP.
%   SOSSOSVAR    --- Declare a new sum of squares variable in an SOSP.
%   SOSEQ        --- Add a new equality constraint to an SOSP.
%   SOSINEQ      --- Add a new inequality constraint to an SOSP.
%   SOSSETOBJ    --- Set the objective function of an SOSP.
%   SOSSOLVE     --- Solve an SOSP.
%   SOSGETSOL    --- Get the solution from a solved SOSP.
%
% Customized functions:
%   FINDSOS      --- Find a sum of squares decomposition of a given polynomial.
%   FINDLYAP     --- Find a Lyapunov function for a dynamical system.
%   FINDBOUND    --- Find a global lower bound (Shor) for a polynomial.
%
% Demos:
%   SOSDEM01     --- Sum of squares test.
%   SOSDEM02     --- Lyapunov function search.
%   SOSDEM03     --- Bound on global extremum.
%   SOSDEM04     --- Matrix copositivity.
%   SOSDEM05     --- Upper bound for the structured singular value  $\mu$ .
%   SOSDEM06     --- MAX CUT.
%   SOSDEM07     --- Chebyshev polynomials.
```

```
% SOSDEM08 --- Bound in probability.
```


Bibliography

- [1] D. Bertsimas and I. Popescu. Optimal inequalities in probability: A convex optimization approach. INSEAD working paper, available at <http://www.insead.edu/facultyresearch/tm/popescu/>, 1999-2001.
- [2] 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.
- [3] G. E. Dullerud and F. Paganini. *A Course in Robust Control Theory: A Convex Approach*. Springer-Verlag NY, 2000.
- [4] A. A. Goldstein and J. F. Price. On descent from local minima. *Mathematics of Computation*, 25:569–574, 1971.
- [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. Available at <http://www.control.ethz.ch/~parrilo/pubs/index.html>.
- [10] P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. Preprint submitted to *Mathematical Programming*, available at <http://www.control.ethz.ch/~parrilo/pubs/index.html>, 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] A. Rantzer and P. A. Parrilo. On convexity in stabilization of nonlinear systems. In *Proceedings of the 39th IEEE Conf. on Decision and Control*, volume 3, pages 2942–2945, 2000.
- [13] B. Reznick. Extremal PSD forms with few terms. *Duke Mathematical Journal*, 45(2):363–374, 1978.

- [14] B. Reznick. Some concrete aspects of Hilbert's 17th problem. In *Contemporary Mathematics*, volume 253, pages 251–272. American Mathematical Society, 2000.
- [15] N. Z. Shor. Class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987.
- [16] 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>.
- [17] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.