Noname manuscript No.
(will be inserted by the editor)

POS3POLY—a MATLAB Preprocessor for Optimization with Positive Polynomials

Bogdan C. Şicleru · Bogdan Dumitrescu

Received: date / Accepted: date

Abstract Positive polynomials, relaxed to sum-of-squares in the multivariate case, are a very powerful instrument having applications in signal processing, control and other engineering fields. Hence, appeared the need of a library which can work with positive polynomials as variables in a convex optimization problem. We present here the POS3POLY library, which transforms polynomial positivity into positive semidefinite constraints, thus enabling the user to solve such problems without the need of knowing the parameterization for each type of polynomial. POS3POLY is able to handle three types of polynomials: trigonometric, real and hybrid. The positivity of the polynomials can be global or only on a semialgebraic domain. POS3POLY allows also to define Bounded Real Lemma constraints. The library is written in MATLAB and uses SeDuMi for solving the convex optimization problems. POS3POLY can also work inside CVX. To show the usage of our library we give several examples of 2-D FIR filter design.

Keywords library · convex optimization · positive polynomials · trigonometric/real/hybrid polynomials · filter design

Bogdan C. Şicleru
Dept. of Automatic Control and Computers, Politehnica University of Bucharest, 313 Spl. Independenţei, 060042, Bucharest, Romania
Tel.: +40-21-4029167
E-mail: bogdan.sicleru@schur.pub.ro

Bogdan Dumitrescu
Dept. of Automatic Control and Computers, Politehnica University of Bucharest, 313 Spl. Independenţei, 060042, Bucharest, Romania
E-mail: bogdan.dumitrescu@schur.pub.ro

Bogdan Dumitrescu
Tampere Int. Center for Signal Processing, Tampere University of Technology, P.O. Box 553, SF-33101, Tampere, Finland
E-mail: bogdan.dumitrescu@tut.fi

1 Introduction

A significant contribution of the last decade in convex optimization is the parameterization of sum-of-squares polynomials with positive semidefinite matrices. This approach lead to the appearance of a myriad of new algorithms in signal processing and in control. Filter design [15] has reached a new level using sum-of-squares optimization [3]; also filter bank design [20], stability analysis [5] and many more. These are based on trigonometric polynomials. Real sum-of-squares polynomials have found application in e.g. controller design [12], stability analysis [24]. Recently, hybrid sum-of-squares polynomials have also been used in applications [9]. Therefore, the necessity of a tool that would incorporate the definitions of all types of sum-of-squares polynomials is obvious.

POS3POLY is a library for solving convex [1,17] optimization problems whose variables include positive or, in the multivariate case, sum-of-squares polynomials. It assures the transparency of the parameterization of positive polynomials, thus making the user able to solve problems with positive polynomials without knowing what is needed for their characterization. POS3POLY is written in MATLAB and uses SeDuMi [23] toolbox for optimization over symmetric cones. For the high-level user POS3POLY can be used within CVX [10]. One can also work with POS3POLY in conjunction with SDPT3 [25].

A general POS3POLY problem written in equality form is

$$\begin{array}{l} \min\ \mathbf{c}^T\mathbf{x} \\ \text{s.t.}\ \ \mathbf{A}\mathbf{x} = \mathbf{b}, \qquad \mathbf{x} \in \mathbb{K} \times \mathbb{P} \end{array} \tag{1}$$

where $\mathbb{K}$ is a symmetric cone (which is a cartesian product of nonnegative orthants, second order cones and cones of semidefinite matrices) and $\mathbb{P}$ denotes generically a cartesian product of cones of (diverse) positive polynomials, which are defined by their *coefficients*. $\mathbb{K}$ is the cone used by SeDuMi and other convex optimization libraries in semidefinite-quadratic-linear programming (SQLP).

Using sum-of-squares relaxations [6,14,9], the POS3POLY library transforms the problem (1) into the SQLP problem

$$\begin{array}{l} \min\ \tilde{\mathbf{c}}^T\tilde{\mathbf{x}} \\ \text{s.t.}\ \ \tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, \qquad \mathbf{x} \in \mathbb{K} \end{array} \tag{2}$$

which can be solved by SeDuMi.

Having built $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$ from (1), the `pos3poly` function can be used to solve the problem (1). A typical call for this function is

```
[ x, y ] = pos3poly( AsP, bsP, csP, KsP );
```

The syntax is similar to that used by SeDuMi namely `AsP`, `bsP`, `csP` are $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$ from (1), respectively; `KsP` is a structure that describes the cone $\mathbb{K} \cup \mathbb{P}$ from (1). The output parameters `x` and `y` are the solutions of the primal problem (1) and, respectively, the dual [1] of the equivalent problem (2).

POS3POLY allows full liberty in describing optimization problems that involve trigonometric, real and hybrid positive polynomials. (Note that the '3' from POS3POLY comes from the three types of polynomials that can be used.) The coefficients of the polynomials can be scalar or matrix, real or complex. The positivity of the polynomials can be global or only on a semialgebraic domain. A distinctive feature is the possibility of working with the Bounded Real Lemma (BRL) type of constraint.

Some of the features offered by POS3POLY cannot be found in other libraries. GloptiPoly [11] and SOSTOOLS [21] deal only with real polynomials and are oriented specifically towards solving sum-of-squares problems. YALMIP [16] and CVX have a richer syntax and hence the possibility of solving larger classes of problems. YALMIP has a module for real sum-of-squares, while CVX offers limited support for univariate scalar positive polynomials.

Outline

The remainder of this article is structured as follows. Section 2 presents the variables which characterize each type of polynomial. In Section 3 we treat sum-of-squares polynomials positive on domains. Section 4 is dedicated to describing the BRL type constraints. Working with POS3POLY inside CVX is presented in Section 5. We conclude in Section 6.

Notation

We denote by $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$ the sets of integer, real and complex numbers respectively. $\mathbb{T}$ denotes the unit circle. Bold characters denote multivariate entities (vectors, matrices). For a polynomial $H(\mathbf{q})$, we denote by $\mathbf{h}$ its vector of coefficients. a^* is the conjugate of a. The superscripts ^{T} and ^{H} denote transposition and Hermitian transposition, respectively. $\mathbf{Tr}\,\mathbf{M}$ is the trace of the matrix $\mathbf{M}$. $\mathbf{M} \succeq \mathbf{0}$ means that $\mathbf{M}$ is a positive semidefinite matrix. $\|H(\mathbf{q})\|_\infty$ and $\|\mathbf{H}(\mathbf{q})\|_\infty$ are $\max(|H(\mathbf{q})|)$ and $\sigma_{\max}(\mathbf{H}(\mathbf{q}))$, respectively, where $\sigma_{\max}(\mathbf{M})$ is the maximum singular value of the matrix $\mathbf{M}$ and $\|\cdot\|_\infty$ is the H_∞ norm.

2 Polynomial description

This section shows how to manipulate positive polynomial variables in the POS3POLY library. Following the SeDuMi style, a polynomial is characterized by a structure containing its degree, type and other relevant information, as shown in Figure 1 and detailed throughout the paper, and a vector of coefficients, which are variables of the optimization problem.

To describe the polynomials, we introduce in the SeDuMi structure, named here `KsP`, two new fields: `p` and `ptype`.

If the optimization problem (1) has N positive polynomials among its variables, then `KsP.p` and `KsP.ptype` are N-by-1 cell arrays, each cell describing a polynomial.

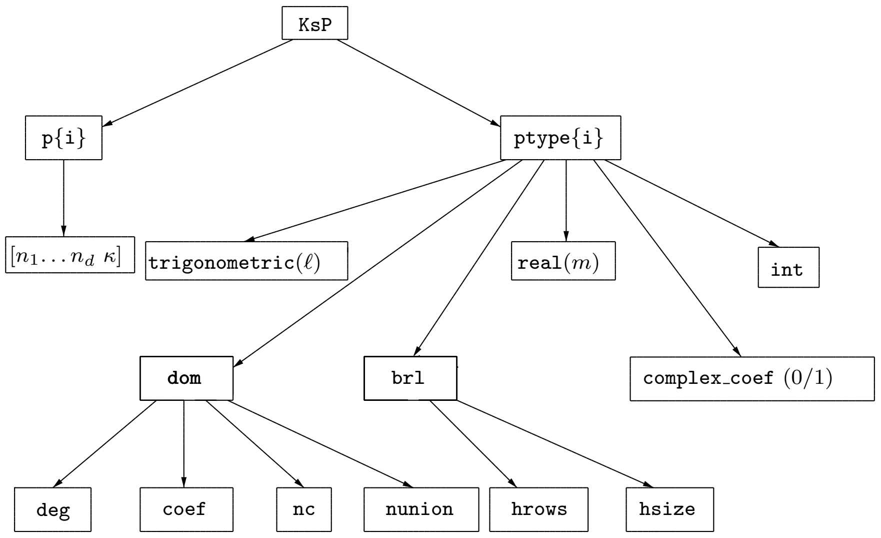

Fig. 1 `KsP`—the structure of POS3POLY.

The field p holds the degree of a d-variate polynomial, $\mathbf{n} = (n_1, n_2, \dots, n_d) \in \mathbb{Z}^d$, and the size of the $\kappa \times \kappa$ matrix polynomial coefficients.

To specify the type of a polynomial (trigonometric, real or hybrid), the user must give the *number* of trigonometric and/or real variables using the fields `trigonometric` and/or `real` of `ptype`. By default, the polynomials are trigonometric.

We present next the coefficients that represent the positive polynomials. The polynomial variables are positioned at the end of $\mathbf{x}$ (see (1)), after the free, linear, quadratic or semidefinite variables accepted by SeDuMi. Implicitly, the coefficient vectors are real; to set them complex, the field `complex_coef` is set to 1.

2.1 Scalar polynomials

2.1.1 Trigonometric polynomial

A (Hermitian) trigonometric polynomial of degree $\mathbf{n}$ with complex coefficients is

$$R(\mathbf{z}) = \sum_{\mathbf{k}=-\mathbf{n}}^{\mathbf{n}} r_{\mathbf{k}} \mathbf{z}^{-\mathbf{k}}, \qquad r_{-\mathbf{k}} = r_{\mathbf{k}}^*, \tag{3}$$

$\mathbf{k}, \mathbf{n} \in \mathbb{Z}^d$, $\mathbf{z} \in \mathbb{T}^d$, $r_{\mathbf{k}} \in \mathbb{C}$; the sum is taken over all $\mathbf{k}$ with $-\mathbf{n} \le \mathbf{k} \le \mathbf{n}$. The coefficients defining the polynomial belong to a halfspace $\mathcal{H}_d$ and form the vector

$$\begin{aligned} &\big[r_{(0,\dots,0)}, r_{(1,0,\dots,0)}, \dots, r_{(n_1,0,\dots,0)}, \\ &r_{(-n_1,1,0,\dots,0)}, \dots, r_{(-n_1,n_2,\dots,n_d)}, \dots, r_{(n_1,n_2,\dots,n_d)}\big]. \end{aligned} \tag{4}$$

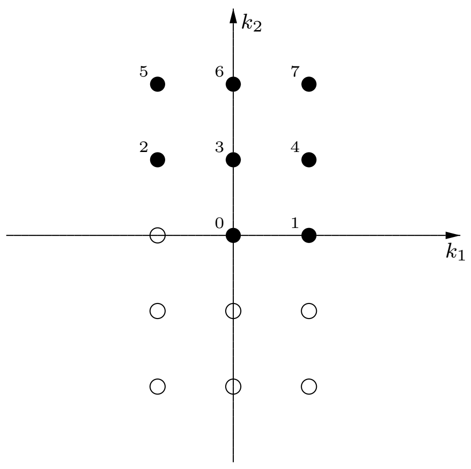

Fig. 2 Coefficients from a halfspace in 2-D.

The total number of coefficients in (4) is

$$M = \frac{1 + \prod_{i=1}^{d}(2n_i + 1)}{2}. \tag{5}$$

Example 1 Consider the case of a bivariate trigonometric polynomial of degree $(1, 2)$. The coefficients are those of the monomials with the following degrees:

$$\begin{array}{l} \{ \qquad\quad (0,0),\ (1,0), \\ (-1,1),\ (0,1),\ (1,1), \\ (-1,2),\ (0,2),\ (1,2)\}. \end{array}$$

Figure 2 shows the order in which the coefficients appear; filled circles belong to the considered halfspace, empty circles do not—they are in the complementary halfspace. □

Example 2 Consider the (symmetric) trigonometric polynomial

$$R(z) = 2z^2 - 3z + 6 - 3z^{-1} + 2z^{-2},$$

$z \in \mathbb{T}$, for which we want to find the minimum on the unit circle. The minimum can be found by solving the optimization problem

$$\begin{aligned} \mu^\star = {} & \max \mu \\ & \text{s.t. } R(z) - \mu \geq 0, \qquad \forall z \in \mathbb{T} \end{aligned} \tag{6}$$

Denoting

$$S(z) = R(z) - \mu, \tag{7}$$

the problem (6) is equivalent with

$$\begin{aligned} \mu^\star = {} & \max_{\mu, \mathbf{s}} \mu \\ & \text{s.t. } (7) \\ & \qquad S(z) \geq 0, \qquad \forall z \in \mathbb{T} \end{aligned} \tag{8}$$

Table 1 POS3POLY program for solving the problem (8).

```
 1  AsP = [ 1 1 0 0 ;      % AsP is a 3x4 matrix
 2          0 0 1 0 ;      % 3 constraints; 4 variables: mu, s0, s1, s2
 3          0 0 0 1 ; ];   % S = s2z^(-2) + s1z^(-1) + s0 + s1z + s2z^2
 4  bsP = [ 6 -3 2 ]';     % bsP = [ r0 r1 r2 ]'; the free term
 5  csP = [ -1 0 0 0 ]';   % the objective function
 6  KsP.f = 1;             % mu is a free variable
 7  KsP.p{ 1 } = [ 2 1 ]; % degree of polynomial and size of coefficients (1x1)
 8  KsP.ptype{ 1 }.trigonometric = 1;   % univariate trigonometric polynomial
 9  x = pos3poly( AsP, bsP, csP, KsP ); % call POS3POLY library
10  mu = x( 1 );           % the minimum of the polynomial
```

where $\mathbf{s} = [\ s_0\ s_1\ s_2\ ]^T$ is the vector of the coefficients of the positive polynomial $S(z)$, enumerated as in (4). Because in the univariate case the sets of nonnegative and sum-of-squares polynomials are identical, the constraint from the above problem is the same as requiring $S(z)$ to be a sum-of-squares polynomial.

Using the parameterization from Theorem 1, for the sum-of-squares constraint, POS3POLY can transform (8) into an equivalent SQLP problem. We have to express the constraints of (8) into the form (1). In order to do this, we put the constraint from (7) in a matrix equation form $\mathbf{Ax} = \mathbf{b}$, considering the real μ and the coefficients of $S(z)$ as the variables from $\mathbf{x}$ in (1). The equality (7) is equivalent to

$$\begin{bmatrix} 1\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix} \begin{bmatrix} \mu \\ s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \\ 2 \end{bmatrix}, \tag{9}$$

We now have identified $\mathbf{A}$ and $\mathbf{b}$ from (1). The vector $\mathbf{c}$ contains four elements, the first one being -1 and the rest zero.

The problem posed here can be solved with the code shown in Table 1. With it, we obtain $\mu^\star = 0.8750$.

Comments. The matrix from the left-hand side (LHS) of (9) is set in lines 1–3 of the code. We initialize the free term—the right-hand side (RHS) of (9)—in line 4. The objective function is set in line 5. The unrestricted variable μ is described in line 6 in the `KsP` structure. The degree of the polynomial and the size of the coefficients are set in line 7. The polynomial is univariate, as stated in line 8. We call POS3POLY in line 9. The minimum is extracted from the solution in line 10. □

2.1.2 Real polynomial

Let us take a real polynomial

$$P(\mathbf{t}) = \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{n}} p_{\mathbf{k}} \mathbf{t}^{\mathbf{k}},$$

$p_{\mathbf{k}} \in \mathbb{R}$, $\mathbf{t} \in \mathbb{R}^d$. There is no symmetry here, so we need all the coefficients of the polynomial. The order of the coefficients is illustrated in the vector describing the polynomial:

$$\begin{aligned}&\big[p_{(0,\dots,0)}, \dots, p_{(n_1,0,\dots,0)}, p_{(0,1,0,\dots,0)}, \dots, \\ &p_{(n_1,1,0,\dots,0)}, \dots, p_{(0,n_2,\dots,n_d)}, \dots, p_{(n_1,n_2,\dots,n_d)}\big] .\end{aligned} \tag{10}$$

Example 3 Considering a degree $\mathbf{n} = (2,1)$, the coefficients describing a real polynomial correspond to the following degrees:

$$\begin{aligned}\{&(0,0),\ (1,0),\ (2,0),\\ &(0,1),\ (1,1),\ (2,1)\}.\end{aligned}$$

□

2.1.3 Hybrid polynomial

Consider a hybrid polynomial [9]

$$\begin{aligned}H(z_1,\dots,z_\ell,t_1,\dots,t_m) = \textstyle\sum_{i_1=-n_1}^{n_1} \cdots \sum_{i_\ell=-n_\ell}^{n_\ell} \sum_{i_{\ell+1}=0}^{n_{\ell+1}} \cdots \\ \cdots \textstyle\sum_{i_d=0}^{n_d} h_{(i_1,\dots,i_\ell,i_{\ell+1},\dots,i_d)} z_1^{i_1} \cdots z_\ell^{i_\ell} t_1^{i_{\ell+1}} \cdots t_m^{i_d},\end{aligned} \tag{11}$$

with $z_i \in \mathbb{T}$, $t_j \in \mathbb{R}$ and $h_{(i_1,\dots,i_\ell,i_{\ell+1},\dots,i_d)} \in \mathbb{C}$, $\forall i = 1 : \ell$, $j = 1 : m$ (with $\ell + m = d$). The relation

$$h_{(i_1,\dots,i_\ell,i_{\ell+1},\dots,i_d)} = h^*_{(-i_1,\dots,-i_\ell,i_{\ell+1},\dots,i_d)}, \tag{12}$$

$i_i = -n_i : n_i$, $\forall i = 1 : \ell$, $i_{\ell+j} = 0 : n_j$, $\forall j = 1 : m$, implies that the polynomial (11) takes real values on $\mathbb{T}^\ell \times \mathbb{R}^m$.

The coefficients describing the polynomial are those corresponding to an ℓ-tuple from $\mathcal{H}_\ell$, namely

$$\{h_{(i_1,\dots,i_\ell,i_{\ell+1},\dots,i_d)}\}, \qquad (i_1, i_2, \dots, i_\ell) \in \mathcal{H}_\ell. \tag{13}$$

The elements of the set from (13) are ordered as follows: for each $(i_{\ell+1}, \dots, i_d)$ m-tuple, covered like in (10), we enumerate all the $(i_1, \dots, i_\ell)$ possible ℓ-tuples, the order being just like in (4).

Example 4 Adjustable FIR filters [9] can be described by the transfer function

$$H(z,t) = \sum_{k=0}^{K} (t - t_0)^k H_k(z), \tag{14}$$

where $H_k(z)$, $k = 0 : K$, are FIR filters and $t_0 \in \mathbb{R}$ is a constant. □

2.2 Matrix polynomials

For polynomials with matrix coefficients the order of the coefficients is the same as in the scalar case. The difference is that now we have matrices, instead of scalars. So, we vectorize the matrices by stacking their columns; for symmetric matrices we vectorize only their lower triangular part.

Example 5 The problem of finding the minimum value of the smallest eigenvalue of a univariate matrix trigonometric polynomial can be written as

$$\begin{aligned} \mu^\star = \max\ & \mu \\ \text{s.t.}\ & \mathbf{R}(z) - \mu \mathbf{I}_\kappa \succeq \mathbf{0}, \qquad \forall z \in \mathbb{T} \end{aligned} \tag{15}$$

Denoting

$$\mathbf{S}(z) = \mathbf{R}(z) - \mu \mathbf{I}_\kappa, \tag{16}$$

the problem (15) is equivalent to

$$\begin{aligned} \mu^\star = \max\ & \mu \\ \text{s.t.}\ & (16) \\ & \mathbf{S}(t) \succeq \mathbf{0}, \qquad \forall z \in \mathbb{T} \end{aligned} \tag{17}$$

The matrix equality characterizing the problem is

$$\begin{bmatrix} \text{vecs}(\mathbf{I}_\kappa) & \\ & \mathbf{I}_{\frac{\kappa(\kappa+1)}{2}+n\kappa^2} \\ \mathbf{0} & \end{bmatrix} \begin{bmatrix} \mu \\ \text{vecs}(\mathbf{S}_0) \\ \text{vec}(\mathbf{S}_1) \\ \vdots \\ \text{vec}(\mathbf{S}_n) \end{bmatrix} = \begin{bmatrix} \text{vecs}(\mathbf{R}_0) \\ \text{vec}(\mathbf{R}_1) \\ \vdots \\ \text{vec}(\mathbf{R}_n) \end{bmatrix}, \tag{18}$$

where $\mathbf{S}_i$, $i = 0 : n$, are the matrix coefficients of the polynomial $\mathbf{S}(t)$, $\text{vec}(\cdot)$ is the function that transforms a matrix into a vector by stacking its columns and $\text{vecs}(\cdot)$ is the function that transforms the lower triangular part of a Hermitian matrix into a vector, by stacking the relevant part of its columns. The function that solves the problem (17) is shown in Table 2 and commented below.

Comments. We stack all the vectorized coefficients in the variable `Rvec`. We have $\kappa(\kappa+1)/2$ scalar constraints for the lower part of $\mathbf{R}_0$ and κ^2 scalar constraints for each $\mathbf{R}_k$, $k = 1 : n$; the total number of constrains is computed in line 3. The number of variables is computed in line 4. In line 6, we put in the `AsP` matrix the multipliers for the variable μ, namely $\text{vecs}(\mathbf{I}_\kappa)$, and in line 8 we set the multipliers for the coefficient variables, the matrix $\mathbf{I}_{\frac{\kappa(\kappa+1)}{2}+n\kappa^2}$, as in (18). The only variable that appears in the objective function is μ (see (17)), which is set in line 9. The POS3POLY structure has one unrestricted variable, μ, and one positive univariate trigonometric polynomial with matrix coefficients, as stated in lines 10–12. The `pos3poly` function is used in line 13

Table 2 POS3POLY program for solving the problem (17).

```
 1   function mu = min_eig_trig( Rvec, n, K )
 2   N = K * ( K + 1 ) / 2;    % number of elements in "half" a matrix
 3   nConstr = N + n * K ^ 2;  % nr. of scalar constraints
 4   nVar = 1 + nConstr;       % total number of POS3POLY variables
 5   AsP = sparse( nConstr, nVar );        % initialize AsP
 6   AsP( 1 : N, 1 ) = vecs( speye( K ) ); % coefficients of mu
 7   % multipliers for the scalar elements of the matrix coefficients
 8   AsP( :, 2 : end ) = speye( nConstr );
 9   csP = -speye( nVar, 1 );  % the objective function
10   KsP.f = 1;                % the unrestricted variable mu
11   KsP.p{ 1 } = [ n K ];     % degree & coefficient size
12   KsP.ptype{ 1 }.trigonometric = 1;    % trig. univariate polynomial
13   x = pos3poly( AsP, Rvec, csP, KsP ); % use the POS3POLY library
14   mu = x( 1 );              % the minimum eigenvalue
```

to solve the optimization problem. Finally, the minimum eigenvalue is returned in line 14. □

Let us take now the univariate real matrix polynomial

$$\mathbf{R}(z) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} z^{-1} + \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix} z$$

and apply on it the program from Table 2.

The RHS of (18) is

```
Rvec = [ 1 0 1 0 1 2 0 ]';
```

Now we use the function that computes $\mu^\star$:

```
mu = min_eig_trig( Rvec, 1, 2 );
```

We find that $\mu^\star = -2$. □

2.3 Causal trigonometric polynomial

We consider the (matrix) polynomial

$$\mathbf{H}(\mathbf{z}) = \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{n}} \mathbf{H}_{\mathbf{k}} \mathbf{z}^{-\mathbf{k}}, \tag{19}$$

$\mathbf{z} \in \mathbb{T}^d$, $\mathbf{H}_{\mathbf{k}} \in \mathbb{C}^{\kappa_1 \times \kappa_2}$. The polynomial (19) is called a *causal* trigonometric polynomial (known as a filter in the signal processing literature) and is described by all the elements of all the matrix coefficients, considered as in the vector:

$$[\mathrm{vec}(\mathbf{H}_{(0,\ldots,0)}); \mathrm{vec}(\mathbf{H}_{(1,0,\ldots,0)}); \ldots; \mathrm{vec}(\mathbf{H}_{(n_1,0,\ldots,0)}); \ldots; \mathrm{vec}(\mathbf{H}_{(n_1,\ldots,n_d)})]. \tag{20}$$

(Note that the order is the same as for the coefficients of a real polynomial.)

Example 6 For a 2-D polynomial of degree $\mathbf{n} = (2, 1)$, the matrix coefficients are

$$\{\mathbf{H}_{(0,0)}, \mathbf{H}_{(1,0)}, \mathbf{H}_{(2,0)}, \mathbf{H}_{(0,1)}, \mathbf{H}_{(1,1)}, \mathbf{H}_{(2,1)}\}.$$

3 Positivity on domains

So far we have discussed only about sum-of-squares polynomials which are globally positive. For describing polynomials that are positive on (semialgebraic) domains, one should use the `int` or `dom` subfields of `ptype`.

3.1 Univariate polynomials

Univariate polynomials can be positive on an interval or on a union of intervals. To impose positivity on a union of intervals, one must concatenate the intervals into the field `int`.

Example 7 The commands necessary to create two univariate trigonometric scalar polynomial variables of degree n, one positive on the interval $[0,\ \pi/4]$ and the other positive on $[\pi/3,\ \pi]$, are

```
KsP.p{ 1 } = [ n 1 ];
KsP.p{ 2 } = [ n 1 ];
KsP.ptype{ 1 }.trigonometric = 1;
KsP.ptype{ 2 }.trigonometric = 1;
KsP.ptype{ 1 }.int = [ 0   pi/4 ];
KsP.ptype{ 2 }.int = [ pi/3  pi ];
```

□

Example 8 To create a trigonometric polynomial (of degree n), positive on $[0,\ 0.3\pi] \cup [0.5\pi,\ 0.7\pi]$, one can use the following commands:

```
KsP.p{ 1 } = [ n 1 ];
KsP.ptype{ 1 }.trigonometric = 1;
KsP.ptype{ 1 }.int = [ 0 0.3*pi 0.5*pi 0.7*pi ];
```

□

3.2 Multivariate polynomials

In the case of *multivariate* polynomials, positivity can be characterized on domains of the form

$$\mathcal{D} = \{\mathbf{q} \mid \mathcal{D}_\ell(\mathbf{q}) \geq 0,\ \ell = 1 : L\}, \tag{21}$$

where $\mathbf{q}$ is a multivariate variable which can contain trigonometric and/or real variables (taking values on the unit circle or the real axis, respectively) and $\mathcal{D}_\ell(\mathbf{q})$ are given polynomials.

The set (21) must satisfy the relatively mild conditions (in particular boundedness) from [22,13] (for trigonometric polynomials, no special condition is needed), under which a polynomial

$$\mathbf{R}(\mathbf{q}) > 0, \quad \forall \mathbf{q} \in \mathcal{D}, \tag{22}$$

is relaxed as

$$\mathbf{R}(\mathbf{q}) = \mathbf{S}_0(\mathbf{q}) + \sum_{\ell=1}^{L} \mathcal{D}_\ell(\mathbf{q})\mathbf{S}_\ell(\mathbf{q}), \tag{23}$$

where $\mathbf{S}_\ell(\mathbf{q})$, $\ell = 0 : L$, are sum-of-squares. POS3POLY implements (23) with sum-of-squares of minimal degree, which is enough for most practical purposes in engineering applications; if needed, the degree of $\mathbf{R}(\mathbf{q})$ can be increased artificially, thus increasing the degree of the relaxation. The sum-of-squares ar parameterized via Theorem 1.

To describe domains like (21), the field `dom` contains three subfields: `deg`, `coef` and `nc`. There are two possibilities of describing the polynomials $\mathcal{D}_\ell(\mathbf{q})$. In the first, the field `deg` is a matrix with L rows, row ℓ being the degree of the polynomial $\mathcal{D}_\ell(\mathbf{q})$; the field `coef` is a vector that contains the concatenated coefficients of the polynomials, enumerated as shown in Section 2. In this case, the field `nc` must *not* be used. The usage of the field `nc` describes the second case, in which the polynomials are sparse and so it is more efficient to give their nonzero coefficients. The field `nc` is a vector of length L, whose ℓ-th element is the number of nonzero coefficients of $\mathcal{D}_\ell(\mathbf{q})$; `deg` is a matrix whose rows contain the degrees of the monomials with nonzero coefficients and `coef` is a vector containing these coefficients (in the same order, obviously). If a polynomial has some symmetry, due to its nature, only the relevant coefficients are needed; for instance, a *symmetric* real trigonometric polynomial is described only by its "half", see again Section 2.

POS3POLY can also handle a union of domains like those in (21):

$$\mathcal{D}' = \bigcup_{\ell'=1}^{L'} \mathcal{D}_{\ell'}. \tag{24}$$

The description of the domain $\mathcal{D}'$ is made via the field `nunion` of `dom`. The field `nunion` is a vector with L' elements—the ℓ'-th element being the number of polynomials used to describe $\mathcal{D}_{\ell'}$. The fields `nc`, `deg`, `coef` have the same meaning as described above, but concatenating the data for all the polynomials and all the domains $\mathcal{D}_{\ell'}$.

As an example of optimization with polynomials that are positive on domains, we consider the minimax optimization of 2-D linear-phase FIR filters with diamond shape passband. The problem can be stated as [7, Section 5.2.2]

$$\begin{array}{ll} \min\limits_{\gamma_s,\mathbf{h}} & \gamma_s \\ \text{s.t.} & 1 + \gamma_p - H(\omega) \geq 0, \quad \forall \omega \\ & -1 + \gamma_p + H(\omega) \geq 0, \quad \forall \omega \in \mathcal{D}_p \\ & \gamma_s - H(\omega) \geq 0, \quad \forall \omega \in \mathcal{D}_s \\ & \gamma_s + H(\omega) \geq 0, \quad \forall \omega \in \mathcal{D}_s \end{array} \tag{25}$$

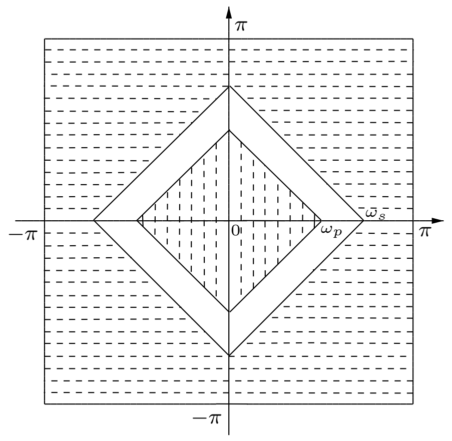

Fig. 3 Diamond shape passband for domain $\mathcal{D}_p = \{\omega_{1,2} \mid |\omega_1| + |\omega_2| \le \omega_p\}$; $a = \cos\omega_p$, $b = \cos\omega_s$.

where $H(\mathbf{z})$ is a bivariate filter and $H(\omega)$ is the frequency response for $\mathbf{z} = \mathrm{e}^{j\omega}$, $d = 2$ in (3), γ_p and γ_s are passband and stopband error bound, respectively, and $\mathcal{D}_p$, $\mathcal{D}_s$ are the passband and stopband domain, respectively. A diamond shape (see Figure 3, passband with vertical lines and stopband with horizontal lines) is obtained by adopting the following definitions:

$$\begin{array}{l} \mathcal{D}_p = \{\omega_{1,2} \mid \cos(\omega_1+\omega_2) - a \ge 0, \cos(\omega_1-\omega_2) - a \ge 0, \cos\omega_1 + \cos\omega_2 \ge 0\} \\ \mathcal{D}_s = \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3} \end{array} \tag{26}$$

where

$$\begin{aligned} \mathcal{D}_{s_1} &= \{\omega_{1,2} \mid -\cos(\omega_1+\omega_2) + b \ge 0\} \\ \mathcal{D}_{s_2} &= \{\omega_{1,2} \mid -\cos(\omega_1-\omega_2) + b \ge 0\} \\ \mathcal{D}_{s_3} &= \{\omega_{1,2} \mid -\cos\omega_1 - \cos\omega_2 \ge 0\}. \end{aligned} \tag{27}$$

The relations (26) and (27) can be written equivalently as

$$\mathcal{D}_p = \{z_{1,2} \mid D_{p_1}(z_1,z_2) \ge 0,\ D_{p_2}(z_1,z_2) \ge 0,\ D_{p_3}(z_1,z_2) \ge 0\} \tag{28}$$

and

$$\begin{aligned} \mathcal{D}_{s_1} &= \{z_{1,2} \mid D_{s_1}(z_1,z_2) \ge 0\} \\ \mathcal{D}_{s_2} &= \{z_{1,2} \mid D_{s_2}(z_1,z_2) \ge 0\} \\ \mathcal{D}_{s_3} &= \{z_{1,2} \mid D_{s_3}(z_1,z_2) \ge 0\}, \end{aligned}$$

where

$$\begin{aligned} D_{p_1}(z_1,z_2) &= (z_1z_2 + z_1^{-1}z_2^{-1})/2 - a \\ D_{p_2}(z_1,z_2) &= (z_1^{-1}z_2 + z_1z_2^{-1})/2 - a \\ D_{p_3}(z_1,z_2) &= (z_1 + z_1^{-1})/2 + (z_2 + z_2^{-1})/2 \\ D_{s_1}(z_1,z_2) &= -(z_1z_2 + z_1^{-1}z_2^{-1})/2 + b \\ D_{s_2}(z_1,z_2) &= -(z_1^{-1}z_2 + z_1z_2^{-1})/2 + b \\ D_{s_3}(z_1,z_2) &= -(z_1 + z_1^{-1})/2 - (z_2 + z_2^{-1})/2. \end{aligned}$$

The problem (25) is relaxed [7, Section 3.5] to

$$\begin{array}{ll}\displaystyle\min_{\gamma_s,\mathbf{h},\mathbf{s}_i,i=0:3} & \gamma_s \\ & S_0(\mathbf{z}) \text{ is sum-of-squares} \\ & S_1(\omega) \geq 0,\ \forall \omega \in \mathcal{D}_p \\ & S_2(\omega) \geq 0,\ \forall \omega \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3} \\ & S_3(\omega) \geq 0,\ \forall \omega \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3}\end{array} \tag{29}$$

where

$$\begin{aligned}S_0(\mathbf{z}) &= 1 + \gamma_p - H(\mathbf{z}) \\ S_1(\mathbf{z}) &= -1 + \gamma_p + H(\mathbf{z}) \\ S_2(\mathbf{z}) &= \gamma_s - H(\mathbf{z}) \\ S_3(\mathbf{z}) &= \gamma_s + H(\mathbf{z}).\end{aligned}$$

The system that characterizes the problem (29) is

$$\left[\begin{array}{cccccc} \mathbf{0} & \mathbf{I}_M & \mathbf{I}_M & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & -\mathbf{I}_M & \mathbf{0} & \mathbf{I}_M & \mathbf{0} & \mathbf{0} \\ \hline \overset{-1}{\mathbf{0}} & \mathbf{I}_M & \mathbf{0} & \mathbf{0} & \mathbf{I}_M & \mathbf{0} \\ \hline \overset{-1}{\mathbf{0}} & -\mathbf{I}_M & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_M \end{array}\right] \begin{bmatrix} \gamma_s \\ \mathbf{h} \\ \mathbf{s}_0 \\ \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix} = \left[\begin{array}{c} 1 + \gamma_p \\ \mathbf{0} \\ \hline -1 + \gamma_p \\ \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array}\right], \tag{30}$$

with M as in (5).

Let us detail now how to describe positivity on a domain. We consider two approaches: the first in which the polynomials which define the positivity domain are given by all their coefficients and the second in which the polynomials are described only by their nonzero coefficients.

3.2.1 Full polynomial description

We consider here the description of the positivity for the polynomial $S_1(\mathbf{z})$ using all the coefficients of the polynomials.

The polynomial $S_1(\mathbf{z})$ is positive on $\mathcal{D}_p$—which is defined by the positivity of three polynomials, as (28) shows. We use the fields `deg` and `coef` to set the positivity domain.

We set the degrees for the polynomials of the positivity domain with

```
KsP.ptype{ 2 }.dom.deg  = [ 1 1; 1 1; 1 1 ];
```

All the coefficients of the polynomials are set using

```
KsP.ptype{ 2 }.dom.coef = [ -a 0 0 0 0.5 ...
                            -a 0 0.5 0 0 ...
                             0 0.5 0 0.5 0 ];
```

3.2.2 Sparse polynomial description

We take now the case where we describe the polynomials used for positivity as sparse polynomials. We exemplify on the polynomial $S_1(\mathbf{z})$. (See also the program from Table 3, which considers the sparse approach.) To describe the positivity domains we use the fields `nc`, `deg` and `coef`.

Each of the three polynomials which describe the positivity for the polynomial $S_1(\mathbf{z})$ has two nonzero coefficients; this is imposed in the `KsP` structure by

```
KsP.ptype{ 2 }.dom.nc = [ 2 2 2 ];
```

The degrees of the monomials corresponding to the nonzero coefficients are described by the POS3POLY code

```
KsP.ptype{ 2 }.dom.deg  = [ 0 0; 1 1; 0 0; -1 1; 1 0; 0 1 ];
```

Finally, the associated coefficients are

```
KsP.ptype{ 2 }.dom.coef = [ -a 0.5 -a 0.5 0.5 0.5 ];
```

The program that solves the problem (29) is in Table 3 and is discussed in the next paragraph.

Comments. We initialize the variables `AsP`, `bsP`, `csP` in lines 6–8. The lines 9–18 set the matrix from the LHS of (30). In lines 19–20 we set the nonzero elements of the RHS of (30). We set the objective function in line 21. γ_s and the filter H are unrestricted variables, as set in line 22. The degrees of the polynomials are set in lines 23–24. The number of variables for the four polynomials are set in lines 25–26. The positivity domain for the second polynomial is set in lines 27–29. The positivity domains for the last two polynomials are set in lines 30–37. Remark the usage of the field `nunion`: each domain used in the union is defined by one polynomial. We call the POS3POLY library in line 38. Finally, in line 39, we obtain the causal part of the filter. □

Example 9 For example, the commands

```
h = lp_fir2d( [ 7 7 ], 0.1, 0.1, -0.7 );
freqz2d( half2all2d( h, [ 7 7 ] ) );
```

design a 2-D FIR filter using the program from Table 3 and plot its frequency response, shown in Figure 4. (The functions `freqz2d` and `half2all2d` are part of the POS3POLY library.) The optimal stopband error is $\gamma_s = 0.0102$. □

4 Bounded Real Lemma

POS3POLY also offers support to specify a Bounded Real Lemma (BRL) [7, Section 4.3], [8], of the form

$$\|\mathbf{H}(\mathbf{q})\|_\infty < |A(\mathbf{q})|, \quad \forall \mathbf{q} \in \mathcal{D}, \tag{31}$$

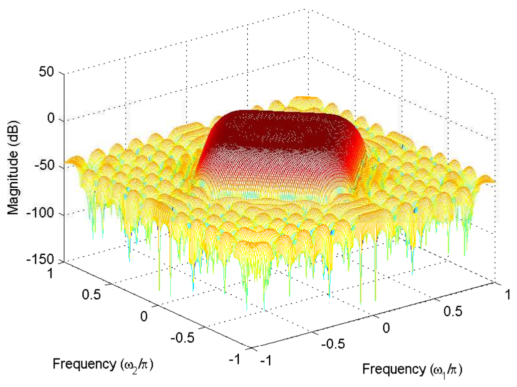

Fig. 4 Frequency response of linear-phase 2-D FIR filter.

where $\mathbf{H}(\mathbf{q})$ and $A(\mathbf{q})$ are *causal* (positive orthant) polynomials of the same degree and $\mathcal{D}$ is like in (24); $A(\mathbf{q})$ has scalar coefficients, but $\mathbf{H}(\mathbf{q})$ may have matrix coefficients of size $\kappa_1 \times \kappa_2$. Relation (31) can be written as

$$\mathbf{H}(\mathbf{q})\mathbf{H}(\mathbf{q}^*)^H < R(\mathbf{q}) \cdot \mathbf{I}_{\kappa_1}, \quad \forall \mathbf{q} \in \mathcal{D}, \tag{32}$$

with $R(\mathbf{q}) = A(\mathbf{q})A(\mathbf{q}^*)^H$. The scalar case of (31) is

$$\|H(\mathbf{q})\|_\infty < |A(\mathbf{q})|, \quad \forall \mathbf{q} \in \mathcal{D}, \tag{33}$$

which is equivalent to $H(\mathbf{q})H(\mathbf{q}^*)^* < R(\mathbf{q})$, $\forall \mathbf{q} \in \mathcal{D}$. The particular case of (33), where $A(\mathbf{q})$ is a constant, has been extensively discussed in control and signal processing, see e.g. [26] and [4], respectively.

Relation (32) is equivalent to (22) and a Linear Matrix Inequality (LMI) condition on $\mathbf{H}(\mathbf{q})$, which is given in Theorem 2 for the case $\mathcal{D} = \mathbb{T}^d$ and scalar coefficients, but is similar in the general case.

In POS3POLY, a BRL is defined by $\mathbf{H}(\mathbf{q})$ and $R(\mathbf{q})$. The polynomial $R(\mathbf{q})$ is described together with the domain $\mathcal{D}$, as a variable that is positive on $\mathcal{D}$. Also, one has to introduce the dependence of the coefficients of $\mathbf{H}(\mathbf{q})$ on the variables $\mathbf{x}$ of the problem (1). To specify a BRL in a POS3POLY problem, the field `brl` in the `KsP` structure has two subfields: `hrows` and `hsize`. The field `hrows` is a vector containing the rows of the matrix `AsP` that describe the polynomial $\mathbf{H}(\mathbf{q})$ via

$$\mathtt{Hvec} = \mathtt{AsP}(\mathtt{hrows}, :) \cdot \mathbf{x} - \mathtt{bsP}(\mathtt{hrows}),$$

where `Hvec` denotes the vector of the coefficients of $\mathbf{H}(\mathbf{q})$, in the order described in Section 2.3; obviously, the length of the vector `hrows` must be equal to the number of elements of the coefficients of $\mathbf{H}(\mathbf{q})$. The field `hsize` contains the vector $[\kappa_1\ \kappa_2]$; if $\kappa_1 = \kappa_2$, then `hsize` may be a scalar, equal to κ_1.

Table 3 POS3POLY program for solving the problem (29).

```
function h = lp_fir2d( n, gp, a, b )
N = nch( n );            % number of n-tuples
nConstr = 4 * N;         % number of constraints
nVar = nConstr + N + 1; % number of variables
I = speye( N );
AsP = sparse( nConstr, nVar ); % initialize POS3POLY
bsP = sparse( nConstr, 1    ); % variables
csP = sparse( 1, nVar );
AsP( 1: N, 2 : N + 1 ) = I;           % 1 + gp = H + S0
AsP( 1: N, N + 2 : 2 * N + 1 ) = I;
AsP( N + 1 : 2 * N, 2 : N + 1 ) = -I; % -1 + gp = -H + S1, Dp
AsP( N + 1 : 2 * N, 2 * N + 2 : 3 * N + 1 ) = I;
AsP( 2 * N + 1 , 1 ) = -1;            % gs = H + S2, Ds1, Ds2, Ds3
AsP( 2 * N + 1 : 3 * N, 2 : N + 1 ) = I;
AsP( 2 * N + 1 : 3 * N, 3 * N + 2 : 4 * N + 1 ) = I;
AsP( 3 * N + 1 , 1 ) = -1;            % gs = -H + S3, Ds1, Ds2, Ds3
AsP( 3 * N + 1 : 4 * N, 2 : N + 1 ) = -I;
AsP( 3 * N + 1 : 4 * N, 4 * N + 2 : 5 * N + 1 ) = I;
bsP( 1     ) =  1 + gp; % free terms for the
bsP( 1 + N ) = -1 + gp; % polynomial constraints
csP( 1 ) = 1;  % 1 for eps in the objective function
KsP.f = 1 + N; % h and gs are free variables
KsP.p{ 1 } = [ n 1 ]; KsP.p{ 2 } = [ n 1 ];
KsP.p{ 3 } = [ n 1 ]; KsP.p{ 4 } = [ n 1 ];
KsP.ptype{ 1 }.trigonometric = 2; KsP.ptype{ 2 }.trigonometric = 2;
KsP.ptype{ 3 }.trigonometric = 2; KsP.ptype{ 4 }.trigonometric = 2;
KsP.ptype{ 2 }.dom.nc = [ 2 2 2 ];
KsP.ptype{ 2 }.dom.deg = [ 0 0; 1 1; 0 0; -1 1; 1 0; 0 1; ];
KsP.ptype{ 2 }.dom.coef = [ -a 0.5 -a 0.5 0.5 0.5 ];
KsP.ptype{ 3 }.dom.nunion = [ 1 1 1 ];
KsP.ptype{ 3 }.dom.nc = [ 2 2 2 ];
KsP.ptype{ 3 }.dom.deg = [ 0 0; 1 1; 0 0; -1 1; 1 0; 0 1 ];
KsP.ptype{ 3 }.dom.coef = [ b -0.5 b -0.5 -0.5 -0.5 ];
KsP.ptype{ 4 }.dom.nunion = [ 1 1 1 ];
KsP.ptype{ 4 }.dom.nc = [ 2 2 2 ];
KsP.ptype{ 4 }.dom.deg = [ 0 0; 1 1; 0 0; -1 1; 1 0; 0 1 ];
KsP.ptype{ 4 }.dom.coef = [ b -0.5 b -0.5 -0.5 -0.5 ];
x = pos3poly( AsP, bsP, csP, KsP ); % use POS3POLY
h = x( 2 : N + 1 ); % "half" of the coefficients of the filter
```

To exemplify the usage of the `brl` field we present the minimax optimization of 2-D *approximately* linear-phase FIR filters. The problem is discussed in [7, Section 5.2.3] and can be cast as

$$\begin{array}{ll} \min\limits_{\gamma_s,\mathbf{h}} & \gamma_s \\ \text{s.t.} & |H(\omega) - G(\omega)| \le \gamma_p, \quad \forall \omega \in \mathcal{D}_p \\ & |H(\omega)| \le \gamma_s, \quad \forall \omega \in \mathcal{D}_s \end{array} \tag{34}$$

where $H(\mathbf{z})$ is a bivariate causal filter and $G(\mathbf{z}) = \mathbf{z}^{-\tau}$ is a delay. The design data are: the delay $\tau \in \mathbb{N}^2$, the passband error bound γ_p and the diamond-shaped passband and stopband $\mathcal{D}_p$, $\mathcal{D}_s$, respectively, given by (26). The stopband error γ_s is minimized.

Introducing positive polynomials for the description of the two BRL constraints, the problem (34) becomes the POS3POLY problem (in which the condition (22) is written explicitly)

$$\begin{array}{ll}
\min\limits_{\gamma_s,\mathbf{h},\mathbf{s}_0,\mathbf{s}_1} & \gamma_s^2 \\
\text{s.t.} & S_0(\mathbf{z}) = \gamma_p^2 \\
 & S_1(\mathbf{z}) = \gamma_s^2 \\
 & S_0(\omega) \geq 0, \quad \forall \omega \in \mathcal{D}_p \\
 & S_1(\omega) \geq 0, \quad \forall \omega \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3} \\
 & |H(\omega) - G(\omega)|^2 \leq S_0(\omega), \quad \forall \omega \in \mathcal{D}_p \\
 & |H(\omega)|^2 \leq S_1(\omega), \quad \forall \omega \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3}
\end{array} \tag{35}$$

where $\mathcal{D}_{s_i}$, $i = 1 : 3$, are as in (27).

The characteristic system for the problem (35) is

$$\left[\begin{array}{cccc}
\mathbf{0} & \mathbf{0} & \mathbf{I}_M & \mathbf{0} \\ \hline
\overset{-1}{\mathbf{0}} & \mathbf{0} & \mathbf{0} & \mathbf{I}_M \\ \hline
\mathbf{0} & \mathbf{I}_N & \mathbf{0} & \mathbf{0} \\ \hline
\mathbf{0} & \mathbf{I}_N & \mathbf{0} & \mathbf{0}
\end{array}\right]
\left[\begin{array}{c} \gamma_s^2 \\ \mathbf{h} \\ \mathbf{s}_0 \\ \mathbf{s}_1 \end{array}\right]
=
\left[\begin{array}{c} \gamma_p^2 \\ \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{g} \\ \hline \mathbf{0} \end{array}\right] \tag{36}$$

where M is computed as in (5) and $N = (n_1 + 1)(n_2 + 1)$. (The vector $\mathbf{g}$ has one on position $\tau_2(n_1 + 1) + \tau_1 + 1$ and zeros elsewhere.) The program which solves the problem (35) is listed in Table 4 and commented next.

Comments. We introduce the degree of the polynomial $H(\omega)$, γ_p and the delay τ as input parameters of the function. The number of coefficients for $S_0(\mathbf{z})$ and $S_1(\mathbf{z})$ is computed in line 2. In line 3 we compute the number of coefficients of the filter $H(\mathbf{z})$. In lines 5 and 6 we compute the total number of constraints and variables, respectively. In lines 11–17 we construct the `AsP` and `bsP` matrices—the four block constraints from (36). We set $H(\mathbf{z})$ and γ_s as free variables in line 18. In lines 19–20 we set the degrees of the positive polynomials and we declare them as bivariate in lines 21–22. The positivity domains for the polynomials are set in lines 23–29. In lines 30–33 we set the BRL constraints. We call the POS3POLY library in line 34 and return the coefficients of the filter in line 35. □

5 POS3POLY and CVX

We discuss here the support of POS3POLY for CVX. Taking advantage of the possibility to define convex sets in CVX, POS3POLY allows with a single function the creation of all types of positive (sum-of-squares) polynomials or BRL variables.

The command to create a positive (sum-of-squares) polynomial is

```
R == sos_pol( p, ptype );
```

Table 4 POS3POLY program for solving the problem (35).

```
function h = app_lp_fir2d( n, gp, tau )
N = nch( n );                   % number of n-tuples
M = prod( n + 1 );              % number of filter coefs.
N2 = 2 * N;                     % nr. coefs. for the SOS pols.
nConstr = N2 + 2 * M;           % number of eq. constraints
nVar = N2 + M + 1;              % number of variables
In = speye( N ); Im = speye( M );
AsP = sparse( nConstr, nVar ); % initialize POS3POLY
bsP = sparse( nConstr, 1 );    % variables
csP = speye( 1, nVar );
AsP( 1: N, M + 2 : M + N + 1 ) = In; % S0
bsP( 1 ) = gp ^ 2; % for the free term
AsP( N + 1 , 1 ) = -1; % S1
AsP( N + 1 : 2 * N, M + N + 2 : M + 2 * N + 1 ) = In;
AsP( N2 + 1 : N2 + M, 2 : M + 1 ) = Im;         % for the 1st BRL filter
bsP( N2 + tau( 2 ) * ( n( 1 ) + 1 ) + tau( 1 ) + 1 ) = 1;
AsP( N2 + M + 1 : N2 + M * 2, 2 : M + 1 ) = Im; % for the 2nd BRL filter
KsP.f = 1 + M;        % h and gs are free variables
KsP.p{ 1 } = [ n 1 ]; % the degrees of the polynomials are 'n'
KsP.p{ 2 } = [ n 1 ]; % and its coefficients are scalars
KsP.ptype{ 1 }.trigonometric = 2; % the polynomial variables
KsP.ptype{ 2 }.trigonometric = 2;
KsP.ptype{ 1 }.dom.nc = [ 2 2 2 ];     % Dp
KsP.ptype{ 1 }.dom.deg = [ 0 0; 1 1; 0 0; -1 1; 1 0; 0 1; ];
KsP.ptype{ 1 }.dom.coef = [ -a 0.5 -a 0.5 0.5 0.5 ];
KsP.ptype{ 2 }.dom.nunion = [ 1 1 1 ]; % Ds1 U Ds2 U Ds3
KsP.ptype{ 2 }.dom.nc = [ 2 2 2 ];
KsP.ptype{ 2 }.dom.deg = [ 0 0; 1 1; 0 0; 1 -1; 1 0; 0 1 ];
KsP.ptype{ 2 }.dom.coef = [ -b -0.5 -b -0.5 -0.5 -0.5 ];
KsP.ptype{ 1 }.brl.hsize = 1; % BRL 1
KsP.ptype{ 1 }.brl.hrows = N2 + 1 : N2 + M;
KsP.ptype{ 2 }.brl.hsize = 1; % BRL 2
KsP.ptype{ 2 }.brl.hrows = N2 + M + 1 : N2 + 2 * M;
x = pos3poly( AsP, bsP, csP, KsP ); % call POS3POLY
h = x( 2 : M + 1 ); % the coefficients of the filter
```

where `p` and `ptype` are structures for *one* polynomial, as presented in Section 2. `R` is a vector variable, containing the coefficients of the polynomial described by `p` and `ptype`, ordered as described also in Section 2.

When one wants to create a BRL, then the command is

```
HR == sos_pol( p, ptype );
```

where `p` and `ptype` are, as above, the structures that describe the polynomial. The variable `HR` is formed by `[ H; R ]` where `H` and `R` denote $\mathbf{H}$ and R from (32), respectively. So, the function describes a single vector variable, obtained by the concatenation of the vectors `H` and `R`. Note that now the field `hrows` of `brl` is no longer needed due to the fact that the matrix `AsP` is not involved.

Table 5 POS3POLY–CVX program for solving the problem (38).

```
1   function [ mu ] = min_pol_cvx( n, r )
2   nC = ( 1 + prod( 2 * n + 1 ) ) / 2; % number of coefficients
3   cvx_begin
4       variable mu;
5       variable S( nC );
6       maximize mu;
7       S == sos_pol( [ n 1 ] );        % SOS polynomial
8       mu * eye( nC, 1 ) + S == r;     % equality constraint
9   cvx_end
```

5.1 Minimum of a polynomial

We present here the computation of the minimum value of a multivariate trigonometric polynomial. For the polynomial $R(\mathbf{z})$ from (3) the problem can be cast as a POS3POLY problem in the form

$$\begin{aligned} \mu^\star = \max_{\mu,\mathbf{s}} \ & \mu \\ \text{s.t.} \ & S(\mathbf{z}) = R(\mathbf{z}) - \mu \\ & S(\mathbf{z}) \geq 0, \quad \forall \mathbf{z} \in \mathbb{T}^d \end{aligned} \tag{37}$$

The problem (37) is relaxed to

$$\begin{aligned} \mu_1 = \max_{\mu,\mathbf{s}} \ & \mu \\ \text{s.t.} \ & S(\mathbf{z}) = R(\mathbf{z}) - \mu \\ & S(\mathbf{z}) \text{ is sum-of-squres} \end{aligned} \tag{38}$$

The equality constraint from (38) can be written as

$$\begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} \mu + \mathbf{s} = \mathbf{r}, \tag{39}$$

where $\mathbf{s}$, $\mathbf{r}$ are variables denoting the coefficients of $S(\mathbf{z})$ and $R(\mathbf{z})$, respectively. The function that solves the problem (38) is listed in Table 5.

Comments. The input parameters of the function describe the polynomial by its degree `n` and the vector `r` of its coefficients. The number of coefficients which describes the polynomial is computed in line 2, using (5). The variable μ is denoted by `mu` in line 4 and the polynomial $S(\mathbf{z})$ by `S` in line 5; `S` is a vector denoting the coefficients of the polynomial. We maximize `mu`. In line 7 we create the sum-of-squares polynomial variable. The equality constraint (39) is enforced in line 8. The command `cvx_end` runs CVX and solves the problem. □

Example 10 We consider the bivariate polynomial

$$R(\mathbf{z}) = \text{sym}^{-1} + 38 + 18z_1 + 4z_1^2 + z_1^{-2}z_2 + 2z_1^{-1}z_2 + z_2 - 8z_1z_2 - 5z_1^2z_2$$

where sym^{-1} is the symmetric part of the polynomial. The MATLAB code to find the minimum is

```
mu = min_pol_cvx( [ 2 1 ], [ 38 18 4 1 2 1 -8 -5 ] );
```

The minimum is $\mu_1 = 1.8214$. (Note that the order of the coefficients in `r` corresponds to the order given in (4).) □

5.2 2-D H_∞ deconvolution

We tackle the 2-D H_∞ deconvolution problem [7, Section 5.3]

$$\begin{aligned} &\min_{\mathbf{X}} \gamma \\ &\text{s.t. } \|\mathbf{A}(\mathbf{z})\mathbf{X}(\mathbf{z}) - \mathbf{B}(\mathbf{z})\|_\infty \le \gamma \end{aligned} \tag{40}$$

where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{X}$ are 2-D causal trigonometric polynomials with matrix coefficients of size $\kappa_1 \times \kappa_2$. The problem (40) can be reformulated as a POS3POLY problem

$$\begin{aligned} &\min_{\gamma,\mathbf{x},\mathbf{s}} \gamma^2 \\ &\text{s.t. } S(\mathbf{z}) = \gamma^2 \\ &\quad\;\; \mathbf{A}(\mathbf{z})\mathbf{X}(\mathbf{z}) - \mathbf{B}(\mathbf{z}) = \mathbf{Y}(\mathbf{z}) \\ &\quad\;\; S(\mathbf{z}) \ge 0 \\ &\quad\;\; \|\mathbf{Y}(\mathbf{z})\|_\infty^2 \le S(\mathbf{z})\mathbf{I}_{\kappa_1} \end{aligned} \tag{41}$$

where $S(\mathbf{z})$ is a trigonometric polynomial whose degree is the maximum between the degrees of $\mathbf{A}(\mathbf{z})\mathbf{X}(\mathbf{z})$ and $\mathbf{B}(\mathbf{z})$. The two equality constraints from (41) are

$$\mathbf{s} = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} \gamma^2 \tag{42}$$

and

$$\mathbf{y} = \mathbf{C}\mathbf{x} - \mathbf{b} \tag{43}$$

where $\mathbf{b}$, $\mathbf{x}$, $\mathbf{y}$ are the vectors of $\mathbf{B}(\mathbf{z})$, $\mathbf{X}(\mathbf{z})$, $\mathbf{Y}(\mathbf{z})$, enumerated as in (20), and $\mathbf{C}$ is a convolution matrix made of the coefficients of $\mathbf{A}(\mathbf{z})$. The MATLAB program which solves the problem (41) is listed in Table 6.

Comments. We set as input parameters the degrees and coefficients of the polynomials $\mathbf{A}(\mathbf{z})$ and $\mathbf{B}(\mathbf{z})$, the degree of $\mathbf{X}(\mathbf{z})$ and the size of the matrix coefficients of the causal polynomials. In lines 3 and 4 we create the coefficient vectors for the polynomials $\mathbf{A}(\mathbf{z})$ and $\mathbf{B}(\mathbf{z})$, respectively, as in (20). The number of scalar coefficients for a matrix coefficient of a causal polynomial is computed in line 5. In lines 6 and 7 we compute the number of scalar coefficients for the polynomials $\mathbf{B}(\mathbf{z})$ and $\mathbf{X}(\mathbf{z})$, respectively. The number of coefficients for the scalar trigonometric polynomial $S(\mathbf{z})$ is computed in line 8; this polynomial is bivariate as stated in line 9. The size of the matrix coefficients from the BRL is set in line 10. We create a BRL variable in line 15. The constraint in line 17 denotes the block constraint from (42) and the one in line 18, the one in (43). (The functions `spcausal2fvec` and `polconv2d` are part of the POS3POLY library.) □

Table 6 POS3POLY–CVX program for solving the problem (41).

```
function [ X, g ] = deconv2d( dA, cA, dB, cB, nX, K )
nA = max( dA ); nB = nA + nX;        % degrees for pol. A and B
vA = spcausal2fvec( nA, dA, cA, K ); % vector for the coef. of A
vB = spcausal2fvec( nB, dB, cB, K ); % vector for the coef. of B
KK = prod( K );            % number of scalar elements in a matrix
pB = prod( nB + 1 ) * KK; % number of scalar elements for pol. B
pX = prod( nX + 1 ) * KK; % number of scalar elements for pol. X
N  = nch( nB );            % number of elements for the trig. pol.
ptype.trigonometric = 2;  % bivariate polynomial
ptype.brl.hsize     = K;  % size of the matrix coefficients
cvx_begin
    variable X( pX ); % vector for the pol. X
    variable g;       % gamma^2
    variable HR( N + pB );
    HR == sos_pol( [ nB 1 ], ptype ); % BRL variable
    minimize g;
    HR( pB + 1 : end ) == g * speye( N, 1 );
    HR( 1 : pB ) == polconv2d( vA, nA, X, nX, [ 2 2 ], [ 2 2 ], 0 ) - vB;
cvx_end
g = sqrt( g );
```

Example 11 We take the polynomials

$$\mathbf{A}(\mathbf{z}) = \begin{bmatrix} 7 & 2 \\ 2 & 3 \end{bmatrix} + \begin{bmatrix} 10 & 0 \\ 0 & -6 \end{bmatrix} z_1^{-1} + \begin{bmatrix} 2 & 3 \\ -1 & -2 \end{bmatrix} z_2^{-1} + \begin{bmatrix} 5 & 8 \\ 2 & 4 \end{bmatrix} z_1^{-1} z_2^{-1}$$

and $\mathbf{B}(\mathbf{z}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} z_1^{-2} z_2^{-2}$. We consider the degree of the polynomial $\mathbf{X}(\mathbf{z})$ to be $\mathbf{n} = (1, 1)$. The MATLAB commands to compute the polynomial $\mathbf{X}(\mathbf{z})$ are:

```
cA{ 1 } = [ 7 2; 2 3]; cA{ 2 } = [ 10 0; 0 -6];
cA{ 3 } = [ 2 3; -1 -2]; cA{ 4 } = [ 5 8; 2 4];
dA = [ 0 0; 1 0; 0 1; 1 1 ];
cB{ 1 } = eye( 2 ); dB = [ 2 2 ];
K = [ 2 2 ];
[ X, g ] = deconv2d( dA, cA, dB, cB, [ 1 1 ], K );
```

We obtain $\gamma = 0.95$; for $\mathbf{n} = (2, 2)$ and $\mathbf{n} = (3, 3)$ we obtain $\gamma = 0.77$ and $\gamma = 0.38$, respectively. As expected, by increasing the degree of $\mathbf{X}(\mathbf{z})$, the error is improved. □

6 Conclusions

We have introduced a MATLAB library able to solve convex optimization problems which involve positive polynomials among their variables. The library offers the possibility to work with three types of polynomials: trigonometric, real and hybrid. The coefficients can be scalar or matrix, real or complex. POS3POLY also offers the support for declaring a BRL type constraint. The POS3POLY library can be downloaded from `www.schur.pub.ro/pos3poly`.

7 Acknowledgements

This work was supported by CNCSIS-UEFISCSU, project PNII – IDEI 309/2007 and the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/6/1.5/S/16.

A Parameterizations

A.1 Sum-of-squares

To illustrate the parameterization behind a sum-of-squares polynomial we give the next result for scalar trigonometric polynomials.

Theorem 1 *The polynomial from (3) is sum-of-squares if and only if there exists a positive semidefinite matrix* $\mathbf{Q} \in \mathbb{C}^{N\times N}$ *such that*

$$r_{\mathbf{k}} = \mathbf{Tr}[\boldsymbol{\Theta}_{k_d} \otimes \cdots \otimes \boldsymbol{\Theta}_{k_1} \cdot \mathbf{Q}], \quad \mathbf{k} \in \mathcal{H}_d, \tag{44}$$

where $\boldsymbol{\Theta}_k$ *is the elementary Toeplitz matrix with ones on the* k*-th diagonal and zeros elsewhere and* $N = \prod_{i=1}^{d}(n_i+1)$*. The matrix* $\mathbf{Q}$ *is called a* Gram *[19, 2, 18] matrix associated with the polynomial (3).* □

Hence, in order to create a scalar trigonometric sum-of-squares polynomial, POS3POLY implements the equalities from (44).

A.2 Bounded Real Lemma

A BRL type constraint for scalar trigonometric polynomials is implemented in POS3POLY using the following result.

Theorem 2 *Let* $H(\mathbf{z})$ *and* $A(\mathbf{z})$ *be causal polynomials. Denoting* $R(\mathbf{z}) = A(\mathbf{z})A^*(\mathbf{z}^{-1})$*, the inequality*

$$|H(\omega)| \le |A(\omega)|$$

is satisfied, if and only if there exists a matrix $\mathbf{Q} \succeq \mathbf{0}$*, such that the relations (44) and* $\begin{bmatrix} \mathbf{Q} & \mathbf{h} \\ \mathbf{h}^H & 1 \end{bmatrix} \succeq \mathbf{0}$ *hold.* □

References

1. S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.
2. M.D. Choi, T.Y. Lam, and B. Reznick. Sums of squares of real polynomials. *Proc. Symp. Pure Math.*, 58(2):103–126, 1995.
3. T.N. Davidson, Z.Q. Luo, and J.F. Sturm. Linear matrix inequality formulation of spectral mask constraints with applications to fir filter design. *IEEE Trans. Signal Proc.*, 50(11):2702–2715, 2002.
4. C. Du, L. Xie, and Y.C. Soh. H_∞ filtering of 2-D discrete systems. *IEEE Trans. Signal Proc.*, 48(6):1760–1768, 2000.
5. B. Dumitrescu. Multidimensional stability test using sum-of-squares decomposition. *IEEE Trans. Autom. Circ. Syst. I*, 53(4):928–936, 2006.
6. B. Dumitrescu. Trigonometric polynomials positive on frequency domains and applications to 2-D FIR filter design. *IEEE Trans. Signal Proc.*, 54(11):4282–4292, 2006.

7. B. Dumitrescu. *Positive Trigonometric Polynomials and Signal Processing Applications.* Springer, 2007.
8. B. Dumitrescu. Bouded real lemma for multivariate trigonometric matrix polynomials and FIR filter design applications. In *Proc. European Sign. Proc. Conf. (EUSIPCO)*, pages 676–680, Glasgow, Scotland, 2009.
9. B. Dumitrescu, Bogdan C. Şicleru, and R. Ştefan. Positive hybrid real-trigonometric polynomials and applications to adjustable filter design and absolute stability analysis. *Circ. Syst. and Sign. Proc.*, 29(5):881–899, 2010.
10. M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming, version 1.21. http://cvxr.com/cvx, May 2010.
11. D. Henrion and J.B. Lasserre. GloptiPoly: Global optimization over polynomials with MATLAB and SeDuMi. *ACM Trans. Math. Soft.*, 29(2):165–194, June 2003.
12. H. Ichihara. Optimal control for polynomial systems using matrix sum of squares relaxations. *IEEE Trans. Autom. Contr.*, 54(5):1048–1053, 2009.
13. T. Jacobi. A representation theorem for certain partially ordered commutative rings. *Math. Z.*, 237:259–273, 2001.
14. J.B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, 11(3):796–814, 2001.
15. S.J. Li, K.L. Teo, X.Q. Yang, and S.Y. Wu. Robust envelope-constrained filter with orthonormal bases and semi-definite and semi-infinite programming. *Optim. Eng.*, 8(3):299–319, 2007.
16. J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
17. A. Magnani and S.P. Boyd. Convex piecewise-linear fitting. *Optim. Eng.*, 10(1):1–17, 2009.
18. J.W. McLean and H.J. Woerdeman. Spectral factorizations and sum of squares representations via semidefinite programming. *SIAM J. Matr. Anal. Appl.*, 23(3):646–655, 2001.
19. P.A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program., Ser.B*, 96:293–320, 2003.
20. C. Popeea and B. Dumitrescu. Optimal compaction gain by eigenvalue minimization. *Sign. Proc.*, 81(5):1113–1116, 2001.
21. S. Prajna and A. Papachristodoulou. SOSTOOLS: Sum of squares optimization toolbox for Matlab, 2004.
22. M. Putinar. Positive polynomials on compact semialgebraic sets. *Indiana. Univ. Math. J.*, 42(3):969–984, 1993.
23. J.F. Sturm. Using SeDuMi, a MATLAB toolbox for optimization over symmetric cones. *Opt. Meth. Soft.*, 11–12:625–653, 1999.
24. W. Tan and A. Packard. Stability region analysis using polynomial and composite polynomial lyapunov functions and sum-of-squares programming. *IEEE Trans. Autom. Contr.*, 53(2):565–571, 2008.
25. K.C. Toh, M.J. Todd, and R.H. Tütüncü. SDPT3—a MATLAB software package for semidefinite programming. *Opt. Meth. Soft.*, 11(1):545–581, 1999.
26. L. Xie and C.E. de Souza. Robust H_∞ control for linear time-invariant systems with norm-bounded uncertainty in the input matrix. *Syst. Contr. Lett.*, 14(5):389–396, 1990.