# User's Guide for POS3POLY—a MATLAB Preprocessor for Positive Polynomials

*Bogdan C. Şicleru*[1] *and Bogdan Dumitrescu*[1,2]

[1]Dept. of Automatic Control and Computers
"Politehnica" University of Bucharest
313 Spl. Independenţei, 060042
Bucharest, Romania

[2]Tampere Int. Center for Signal Processing
Tampere University of Technology
P.O. Box 553, SF-33101
Tampere, Finland

e-mail: bogdan.sicleru@schur.pub.ro, bogdan.dumitrescu@tut.fi

December 20, 2011

Version 3.1

---

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.

# Contents

# Chapter 1

# Introduction

POS3POLY is a library for solving convex optimization problems whose constraints involve positive polynomials (by relaxation to sum-of-squares). It assures the transparency of the parameterization of positive polynomials, basically making the user able to solve problems with positive polynomials without knowing what is needed for their characterization. POS3POLY is written in MATLAB[1] and uses SeDuMi [10] toolbox for optimization over symmetric cones.

Let us see now, at a first glance, how does POS3POLY work. For this, consider a semidefinite-quadratic-linear problem written in the equality form

$$\begin{array}{ll} \min & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}, \quad \boldsymbol{x}\in\mathbb{K}\times\mathbb{P} \end{array} \tag{1.1}$$

Here $\mathbb{K}$ is a symmetric cone which is a cartesian product of a nonnegative orthant, 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). This is the general optimization problem solved by SeDuMi, which also allows the use of free (unrestricted) variables.

With POS3POLY, one can add positive polynomials as variables in problem (1.2), hence solving

$$\begin{array}{ll} \min & \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{x}} \\ \text{s.t.} & \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}}=\tilde{\boldsymbol{b}}, \quad \tilde{\boldsymbol{x}}\in\mathbb{K} \end{array} \tag{1.2}$$

which can be solved by SeDuMi.

---

[1] MATLAB is a registered trademark of The MathWorks, Inc.

In dual or inequality form, a POS3POLY problem is

$$\begin{array}{ll} \max & \boldsymbol{b}^T \boldsymbol{y} \\ \text{s.t.} & \boldsymbol{w} = \boldsymbol{c} - \boldsymbol{A}^T \boldsymbol{y} \in \mathbb{K} \times \mathbb{P} \\ & \boldsymbol{y} \in \mathbb{R}^{\nu} \end{array} \tag{1.3}$$

Similarly with the primal form, POS3POLY transforms (1.3) into an SQLP problem.

Having built $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$ from (1.1) or (1.3), the `pos3poly` function can be used to solve the problem (1.1). A typical call for this function is

```
x_or_y = pos3poly( AsP, bsP, csP, KsP, pars );
```

The syntax is similar to that used by SeDuMi, namely `AsP`, `bsP`, `csP` are $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$ from (1.1), or $\boldsymbol{A}^T$, $\boldsymbol{b}$, $\boldsymbol{c}$ from (1.3), respectively; `KsP` is a structure that describes the cone $\mathbb{K} \times \mathbb{P}$ from (1.1) and `pars` (which the user is not obliged to use) is a structure that can change the parameter settings for SeDuMi; for more information on the latter two parameters, consult SeDuMi's user guide and MATLAB help. Essentially, what `pos3poly` does is to transform the problem (1.1) or (1.3) into the form (1.2), call SeDuMi to solve it, then retrieve the solution. The output parameter is the solution to (1.1) or (1.3), the primal or dual form being recognized automatically.

POS3POLY can also be used within CVX [6] for describing positive polynomial variables. The reader interested only by this mode can jump at chapter 5 after the next chapter.

### Features

The current POS3POLY version supports the following:

- sum-of-squares multivariate polynomials
- real and/or trigonometric variables
- scalar/matrix coefficients
- real/complex coefficients
- positivity intervals/domains
- Bounded Real Lemma (BRL) for polynomials

Some of these features cannot be found in other libraries. GloptiPoly [7] and SOSTOOLS [9] deal only with real polynomials and are oriented specifically towards solving sum-of-squares problems. YALMIP [8] 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 positive polynomials. As a distinctive feature, POS3POLY allows full liberty in describing optimization problems that involve trigonometric and hybrid positive polynomials. (Note that the '3' from POS3POLY comes from the three types of polynomials that can be used: real, trigonometric and hybrid.) Also new is the possibility of working with the BRL type of constraint.

### Setup

We offer POS3POLY in a `.zip` archive[2]. To set up the MATLAB POS3POLY library you can follow these steps:

1. Extract POS3POLY to a desired directory.
2. Enter in the main directory of the library, which is `POS3POLY`.
3. Run the `p3p_setup` script to setup the POS3POLY library.

### Feedback

For any bugs, requests or suggestions send us an e-mail.

---

[2] You can download POS3POLY from http://www.schur.pub.ro/pos3poly.

# Chapter 2

# Polynomial description

This chapter shows how to manipulate positive polynomial variables in POS3POLY. Following the SeDuMi style, a polynomial is characterized by a structure containing its type, degree and other relevant information, and a vector of coefficients, which are variables of the optimization problem. The description of the coefficients is presented in Section 2.1 and the structure in Section 2.2. All positive polynomials are implemented via sum-of-squares relaxations that are described in the Appendix. Here we discuss mostly the programming aspects.

## 2.1 Coefficient description

We present next the coefficients chosen to represent the positive polynomials, coefficients which become variables in a POS3POLY problem. More precisely, we define the variables that are present in the $\boldsymbol{x}$ term of (1.1) or $\boldsymbol{w}$ term of (1.3) for each type of polynomial. The polynomial variables are positioned at the end of $\boldsymbol{x}$ or $\boldsymbol{w}$, after the free, linear, quadratic or semidefinite variables accepted by SeDuMi.

### 2.1.1 Scalar coefficients

- *Trigonometric polynomial.* Let us take a (Hermitian) trigonometric polynomial of degree $\boldsymbol{n}$ with complex coefficients

$$R(\boldsymbol{z}) = \sum_{\boldsymbol{k}=-\boldsymbol{n}}^{\boldsymbol{n}} r_{\boldsymbol{k}} \boldsymbol{z}^{-\boldsymbol{k}}, \quad r_{-\boldsymbol{k}} = r_{\boldsymbol{k}}^*, \tag{2.1}$$

$\boldsymbol{k}, \boldsymbol{n} \in \mathbb{Z}^d$, $\boldsymbol{z} \in \mathbb{T}^d$, $r_{\boldsymbol{k}} \in \mathbb{C}$, where $\mathbb{T}$ is the unit circle. (The sum is considered for all the $d$-tuples such that $-\boldsymbol{n} \leq \boldsymbol{k} \leq \boldsymbol{n}$; we denote $R(\boldsymbol{\omega})$ the frequency response for the filter $R(\boldsymbol{z})$, obtained by putting $\boldsymbol{z} = \mathrm{e}^{j\boldsymbol{\omega}}$.) We store all the coefficients that belong to the halfspace $\mathcal{H}_d \in \mathbb{Z}^d$ defined by: $\boldsymbol{k} \in \mathcal{H}_d$ if $(k_d > 0)$ or $(k_d = 0$ and $(k_1, \ldots, k_{d-1}) \in \mathcal{H}_{d-1})$. Therefore, POS3POLY needs the parameters

$$\{r_{(0,\ldots,0)}, r_{(1,0,\ldots,0)}, \ldots, r_{(n_1,0,\ldots,0)}, r_{(-n_1,1,0,\ldots,0)}, \ldots, r_{(-n_1,n_2,\ldots,n_d)}, \ldots, r_{(n_1,n_2,\ldots,n_d)}\}. \tag{2.2}$$

The total number of coefficients in (2.2) is

$$M = \frac{1 + \prod_{i=1}^{d}(2n_i + 1)}{2}. \tag{2.3}$$

For an easier understanding of the order in which the coefficients appear in (2.2), we consider below some examples.

Example 2.1.1. For the univariate case, $d = 1$, for a polynomial of degree $n$, the coefficients are

$$\{r_0, r_1, \ldots, r_n\}. \tag{2.4}$$

■

Example 2.1.2. We take now the case of a bivariate trigonometric polynomial of degree $(1, 2)$. The coefficients are those of the monomials with the following degrees:

$$\begin{array}{lll} \{ & (0,0), & (1,0), \\ (-1,1), & (0,1), & (1,1), \\ (-1,2), & (0,2), & (1,2)\}. \end{array} \tag{2.5}$$

Figure 2.1 shows the order in which the coefficients appear: we go above from below and from the left to the right. (Filled circles belong to the considered halfspace, empty circles do not—they are in the complementary halfspace.) ■

Example 2.1.3. Let us extend the previous example and consider a 3-D polynomial, like in (2.1), with degree $\boldsymbol{n} = (1, 2, 1)$. To get the coefficients, we take the ones from (2.5) and put 0 for the third dimension, and then, for the other coefficients we take all the positive values for $k_3$ for which we consider all possible pairs $(k_1, k_2)$. Having

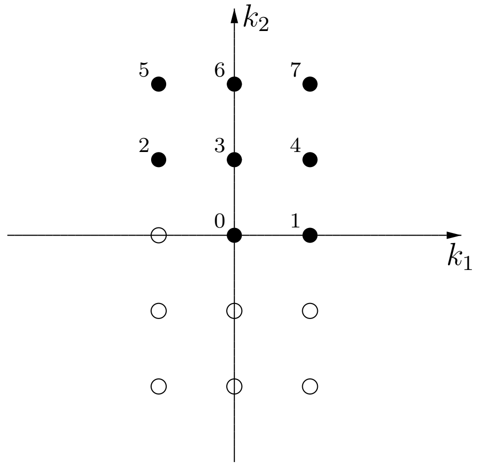


Figure 2.1: Coefficients from a halfspace in 2-D.

this said, the corresponding triples of monomial degrees are

$$\begin{array}{lccc} \{ & & (0,0,0), & (1,0,0), \\ & (-1,1,0), & (0,1,0), & (1,1,0), \\ & (-1,2,0), & (0,2,0), & (1,2,0), \\ & (-1,-2,1), & (0,-2,1), & (1,-2,1), \\ & (-1,-1,1), & (0,-1,1), & (1,-1,1), \\ & (-1,0,1), & (0,0,1), & (1,0,1), \\ & (-1,1,1), & (0,1,1), & (1,1,1), \\ & (-1,2,1), & (0,2,1), & (1,2,1)\}. \end{array} \tag{2.6}$$

■

- *Real Polynomial.* Let $P \in \mathbb{R}_{\boldsymbol{n}}[\boldsymbol{t}]$,

$$P(\boldsymbol{t}) = \sum_{\boldsymbol{k}=\boldsymbol{0}}^{\boldsymbol{n}} p_{\boldsymbol{k}} \boldsymbol{t}^{\boldsymbol{k}}, \tag{2.7}$$

be a real polynomial. There is no symmetry here, so we need all the coefficients of the polynomial. The order of coefficients is

$$\{p_{(0,\ldots,0)}, \ldots, p_{(n_1,0,\ldots,0)}, p_{(0,1,0,\ldots,0)}, \ldots, p_{(n_1,1,0,\ldots,0)}, \ldots, p_{(0,n_2,\ldots,n_d)}, \ldots, p_{(n_1,n_2,\ldots,n_d)}\}. \tag{2.8}$$

EXAMPLE 2.1.4. For a bivariate real polynomial of degree $(2,2)$ the coefficients are

$$\begin{array}{lll} \{p_{(0,0)}, & p_{(1,0)}, & p_{(2,0)}, \\ p_{(0,1)}, & p_{(1,1)}, & p_{(2,1)}, \\ p_{(0,2)}, & p_{(1,2)}, & p_{(2,2)}\}. \end{array} \tag{2.9}$$

■

- *Hybrid polynomial.* Consider a hybrid polynomial

$$\begin{array}{rcl} H(z_1,\ldots,z_\ell,t_1,\ldots,t_m) & = & \sum_{i_1=-n_1}^{n_1}\cdots\sum_{i_\ell=-n_\ell}^{n_\ell}\sum_{i_{\ell+1}=0}^{n_{\ell+1}}\cdots \\ & & \cdots\sum_{i_d=0}^{n_d} h_{(i_1,\ldots,i_\ell,i_{\ell+1},\ldots,i_d)} z_1^{i_1}\cdots z_\ell^{i_\ell} t_1^{i_{\ell+1}}\cdots t_m^{i_d}, \end{array} \tag{2.10}$$

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

$$h_{(i_1,\ldots,i_\ell,i_{\ell+1},\ldots,i_d)} = h^*_{(-i_1,\ldots,-i_\ell,i_{\ell+1},\ldots,i_d)}, \tag{2.11}$$

$i_i = -n_i : n_i$, $\forall i = 1 : \ell$, $i_{\ell+j} = 0 : n_j$, $\forall j = 1 : m$, implies that the polynomial (2.10) takes real values on $\mathbb{T}^\ell \times \mathbb{R}^m$.

The coefficients we need are the ones that correspond to an $\ell$-tuple from $\mathcal{H}_\ell$, namely

$$\{h_{(i_1,\ldots,i_\ell,i_{\ell+1},\ldots,i_d)}\}, \quad (i_1,i_2,\ldots,i_\ell) \in \mathcal{H}_\ell, \quad i_{\ell+j} = 0 : n_j, \forall j = 1 : m. \tag{2.12}$$

The elements of the set from (2.12) are ordered as follows: for every $(i_{\ell+1},\ldots,i_d)$ $(d-\ell)$-tuple, covered like in (2.8), we choose all the $(i_1,\ldots,i_\ell)$ possible $\ell$-tuples, the order being just like in (2.2).

EXAMPLE 2.1.5. We take a hybrid polynomial of degree $(1,2)$ with one trigonometric variables. The considered coefficients are

$$\begin{array}{ll} \{h_{(0,0)}, & h_{(1,0)}, \\ h_{(0,1)}, & h_{(1,1)}, \\ h_{(0,2)}, & h_{(1,2)}\}. \end{array} \tag{2.13}$$

■

EXAMPLE 2.1.6. We consider now a hybrid polynomial of degree (1,1,1) with two trigonometric variables. The coefficients which characterize the polynomial are

$$\begin{array}{lll} \{h_{(0,0,0)}, & h_{(1,0,0)}, & h_{(-1,1,0)}, \\ h_{(0,1,0)}, & h_{(1,1,0)}, & h_{(0,0,1)}, \\ h_{(1,0,1)}, & h_{(-1,1,1)}, & h_{(0,1,1)}, \\ h_{(1,1,1)}\}. & & \end{array} \tag{2.14}$$

■

### 2.1.2 Matrix coefficients

Next, we extend the description to polynomials with matrix coefficients. The order in which we consider 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.

- *Trigonometric polynomial.* We consider a trigonometric polynomial with matrix coefficients having the form

  $$\boldsymbol{R}(\boldsymbol{z}) = \sum_{\boldsymbol{k}=-\boldsymbol{n}}^{\boldsymbol{n}} \boldsymbol{R}_{\boldsymbol{k}} \boldsymbol{z}^{-\boldsymbol{k}}, \quad \boldsymbol{R}_{-\boldsymbol{k}} = \boldsymbol{R}_{\boldsymbol{k}}^H, \tag{2.15}$$

  $\boldsymbol{R}_{\boldsymbol{k}} \in \mathbb{C}^{\kappa \times \kappa}$. Let vec() be the function that transforms a matrix into a vector by stacking its columns. The symmetry relation from (2.15) tells that $\boldsymbol{R}_{\boldsymbol{0}}$ is Hermitian, hence only half of its elements are needed. Let vecs() be the function that transforms the lower part of a Hermitian matrix into a vector, by stacking the relevant part of its columns. Taking (2.2) into account, the coefficients of (2.15) are described by the vector

  $$\begin{array}{l} \{\mathrm{vecs}(\boldsymbol{R}_{(0,\ldots,0)}), \mathrm{vec}(\boldsymbol{R}_{(1,0,\ldots,0)}), \ldots, \mathrm{vec}(\boldsymbol{R}_{(n_1,0,\ldots,0)}), \\ \mathrm{vec}(\boldsymbol{R}_{(-n_1,1,0,\ldots,0)}), \ldots, \mathrm{vec}(\boldsymbol{R}_{(n_1,n_2,\ldots,n_d)})\}. \end{array} \tag{2.16}$$

  EXAMPLE 2.1.7. For a polynomial with degree 2 and $\kappa = 2$ the scalar coefficients are

  $$\begin{array}{llll} \{\boldsymbol{R}_0(1,1), & \boldsymbol{R}_0(2,1), & \boldsymbol{R}_0(2,2), & \\ \boldsymbol{R}_1(1,1), & \boldsymbol{R}_1(2,1), & \boldsymbol{R}_1(1,2), & \boldsymbol{R}_1(2,2), \\ \boldsymbol{R}_2(1,1), & \boldsymbol{R}_2(2,1), & \boldsymbol{R}_2(1,2), & \boldsymbol{R}_2(2,2)\}. \end{array} \tag{2.17}$$

  ■

- *Real polynomial.* For a real polynomial with matrix coefficients,

  $$\boldsymbol{P}(\boldsymbol{t}) = \sum_{\boldsymbol{k}=\boldsymbol{0}}^{\boldsymbol{n}} \boldsymbol{P}_{\boldsymbol{k}} \boldsymbol{t}^{\boldsymbol{k}}, \quad \boldsymbol{P}_{\boldsymbol{k}} = \boldsymbol{P}_{\boldsymbol{k}}^H, \tag{2.18}$$

  $\boldsymbol{P}_{\boldsymbol{k}} \in \mathbb{C}^{\kappa \times \kappa}$, the coefficients are described by

  $$\{\mathrm{vecs}(\boldsymbol{P}_{(0,\ldots,0)}), \mathrm{vecs}(\boldsymbol{P}_{(1,0,\ldots,0)}), \ldots, \mathrm{vecs}(\boldsymbol{P}_{(n_1,0,\ldots,0)}), \ldots, \mathrm{vecs}(\boldsymbol{P}_{(n_1,\ldots,n_d)})\}. \tag{2.19}$$

  (See (2.8).)

Example 2.1.8. For a polynomial of degree 2 the scalar coefficients describing the polynomial are

$$\begin{array}{lll}\{\boldsymbol{P}_0(1,1), & \boldsymbol{P}_0(2,1), & \boldsymbol{P}_0(2,2),\\ \boldsymbol{P}_1(1,1), & \boldsymbol{P}_1(2,1), & \boldsymbol{P}_1(2,2),\\ \boldsymbol{P}_2(1,1), & \boldsymbol{P}_2(2,1), & \boldsymbol{P}_2(2,2)\}.\end{array} \tag{2.20}$$

■

- *Hybrid polynomial.* Let

$$\begin{aligned}\boldsymbol{H}(z_1,\ldots,z_\ell,t_1,\ldots,t_m) \;=\; & \textstyle\sum_{i_1=-n_1}^{n_1}\cdots\sum_{i_\ell=-n_\ell}^{n_\ell}\sum_{i_{\ell+1}=0}^{n_{\ell+1}}\cdots\\ & \cdots\textstyle\sum_{i_d=0}^{n_d}\boldsymbol{H}_{(i_1,\ldots,i_\ell,i_{\ell+1},\ldots,i_d)}z_1^{i_1}\cdots z_\ell^{i_\ell}t_1^{i_{\ell+1}}\cdots t_m^{i_d},\end{aligned} \tag{2.21}$$

with $z_i \in \mathbb{T}$ and $t_j \in \mathbb{R}$, $i = 1:\ell$, $j = 1:m$, be a multivariate hybrid polynomial with complex matrix coefficients. The symmetry relation corresponding to (2.11) is

$$\boldsymbol{H}_{(i_1,\ldots,i_\ell,i_{\ell+1},\ldots,i_d)} = \boldsymbol{H}^H_{(-i_1,\ldots,-i_\ell,i_{\ell+1},\ldots,i_d)}, \tag{2.22}$$

$i_i = -n_i : n_i$, $\forall i = 1:\ell$, $i_{\ell+j} = 0 : n_j$, $\forall j = 1:m$. The values that represent the polynomial are the matrix coefficients vectorized by applying the vec() function to all of them but $\boldsymbol{H}_{(i_1,\ldots,i_\ell,i_{\ell+1},\ldots,i_d)}$ with $i_1 = \ldots = i_\ell = 0$, i.e. where the degree of all the trigonometric variables is $\mathbf{0}$, to which we apply vecs(). The order in which we take the coefficients is the same as in (2.12).

### 2.1.3 Causal polynomials

We consider the (matrix) polynomial

$$\boldsymbol{H}(\boldsymbol{q}) = \sum_{\boldsymbol{k}=\boldsymbol{0}}^{\boldsymbol{n}} \boldsymbol{H}_{\boldsymbol{k}}\boldsymbol{q}^{\boldsymbol{k}}, \tag{2.23}$$

$\boldsymbol{q} \in \mathbb{T}^\ell \times \mathbb{R}^m$, $\ell + m = d$, $\boldsymbol{H}_{\boldsymbol{k}} \in \mathbb{C}^{\kappa_1\times\kappa_2}$. The polynomial (2.23) is called a *causal* polynomial and is described by *all* the elements of all the matrix coefficients, considered in the following order:

$$\{\text{vec}(\boldsymbol{H}_{(0,\ldots,0)}), \text{vec}(\boldsymbol{H}_{(1,0,\ldots,0)}), \ldots, \text{vec}(\boldsymbol{H}_{(n_1,0,\ldots,0)}), \ldots, \text{vec}(\boldsymbol{H}_{(n_1,\ldots,n_d)})\}. \tag{2.24}$$

(Note that the order is the same as for the coefficients of a real polynomial. However, the coefficient vectors are identical only in the scalar case.)

Example 2.1.9. For a 2-D polynomial of degree $\boldsymbol{n} = (2,1)$ the matrix coefficients are enumerated in the order

$$\{\boldsymbol{H}_{(0,0)}, \boldsymbol{H}_{(1,0)}, \boldsymbol{H}_{(2,0)}, \boldsymbol{H}_{(0,1)}, \boldsymbol{H}_{(1,1)}, \boldsymbol{H}_{(2,1)}\}. \tag{2.25}$$

■

## 2.2 Structure

Figure 2.2 gives an overview of the fields *added* to the SeDuMi structure to describe the polynomials in POS3POLY. Precisely, to characterize variables belonging to the general cone of positive polynomials $\mathbb{P}$ from (1.1), we introduce two new fields in the structure `KsP`, namely `ptype` and `p`.

If the optimization problem (1.1) has $N$ positive polynomials among its variables, then `KsP.ptype` and `KsP.p` are $N$-by-1 cell arrays, each cell describing a polynomial.

The field `p` holds the degree of a $d$-variate polynomial, $\boldsymbol{n} = (n_1, n_2, \ldots, n_d) \in \mathbb{Z}^d$, and the size of the $\kappa \times \kappa$ matrix polynomial coefficients. (Scalar coefficients are of size $1 \times 1$). Hence, `KsP.p` has the following structure:

$$[\ n_1\ n_2\ \ldots n_d\ \kappa\ ]. \tag{2.26}$$

If `KsP.p` has only one element, $n$, the polynomial is univariate with scalar coefficients and degree $n$.

The field `ptype` gives additional information on the polynomial variables, through its subfields. There are three types of polynomials: trigonometric, real and hybrid. In order to specify the type of polynomial, the user must explicitly give the *number* of trigonometric and/or real variables; for this, each cell of `KsP.ptype` has two fields: `trigonometric` and `real`. If the polynomial does not have real variables the field `real` can be omitted. By default, if none of the fields `trigonometric` and `real` appears, it is considered that the polynomial is trigonometric; its number of variables is given by `KsP.p`, see (2.26).

The coefficients of the polynomials can be real or complex. By default, real coefficients are used. To specify complex coefficients, the subfield `complex_coef` of `ptype` must be set to 1. In this case, remember also to set the field `KsP.ycomplex` for describing complex equality constraints, as requested by SeDuMi.

By setting only the above fields, one describes sum-of-squares polynomials (which are obviously globally positive). For describing polynomials that are positive on (semialgebraic) domains, one should use the `int` or `dom` subfields of `ptype`. *Univariate* polynomials can be positive on an interval or on a union of intervals. Table 2.1 gives the values of

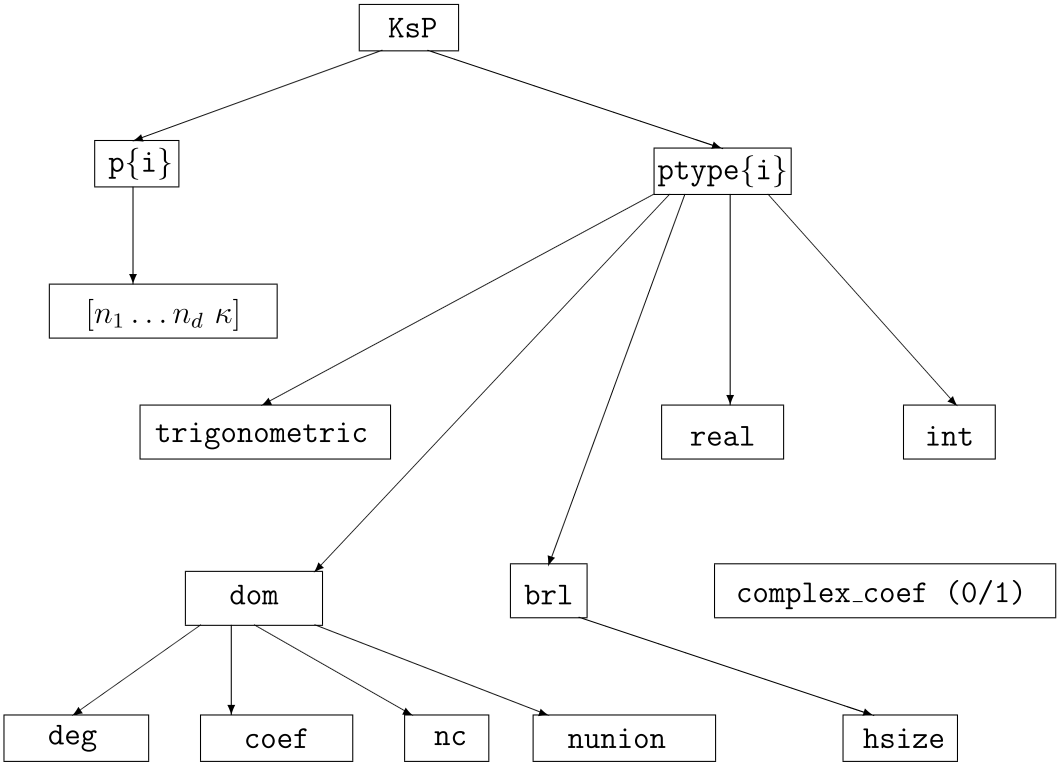


Figure 2.2: `KsP`—the structure of POS3POLY.

the field `int` for all possible cases for a single interval; to impose positivity on a union of intervals, one must concatenate the intervals (into the vector `int`).

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

$$\mathcal{D} = \{\boldsymbol{q} \mid \mathcal{D}_\ell(\boldsymbol{q}) \geq 0,\ \ell = 1 : L\}, \tag{2.27}$$

where $\boldsymbol{q}$ is a multivariate variable which can contain real and/or trigonometric variables (taking values on the real axis or the unit circle, respectively) and $\mathcal{D}_\ell(\boldsymbol{q})$ are given polynomials. To describe domains like (2.27), the field `dom` contains three subfields: `deg`, `coef` and `nc`. There are two possibilities of describing the polynomials $\mathcal{D}_\ell(\boldsymbol{q})$. In the first, the field `deg` is a matrix with $L$ rows, row $\ell$ being the degree of the polynomial $\mathcal{D}_\ell(\boldsymbol{q})$; the field `coef` is a vector that contains the concatenated coefficients of the polynomials, enumerated as shown in the previous section. The second possibility assumes that the polynomials are sparse and so it is more efficient to give their nonzero coefficients. The field `nc` is vector of length $L$, whose $\ell$-th element is the number of nonzero coefficients of $\mathcal{D}_\ell(\boldsymbol{q})$; `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 the previous section.

We consider now the generalized version of a domain, which is a union of domains like those in (2.27):

$$\mathcal{D}' = \bigcup_{\ell'=1}^{L'} \mathcal{D}_{\ell'}. \tag{2.28}$$

The domain $\mathcal{D}'$ can be described by the field `nunion` of `dom`. The field `nunion` is a vector with $L'$ elements—the number of polynomials used to describe each domain $\mathcal{D}_{\ell'}$. The fields `nc`, `deg`, `coef` have the same meaning as described above, but contain information on all the polynomials for all the domains $\mathcal{D}_{\ell'}$.

POS3POLY also offers support to specify a Bounded Real Lemma (BRL) [1, Section 4.3], [2]. Let us consider a general form for the BRL,

$$\|\boldsymbol{H}(\boldsymbol{q})\| < |A(\boldsymbol{q})|, \quad \forall \boldsymbol{q} \in \mathcal{D}', \tag{2.29}$$

where $\boldsymbol{H}(\boldsymbol{q})$ and $A(\boldsymbol{q})$ are *causal* (positive orthant) polynomials of the same degree; $A(\boldsymbol{q})$ has scalar coefficients, but $\boldsymbol{H}(\boldsymbol{q})$ may have matrix coefficients of size $\kappa_1 \times \kappa_2$. By $\|\cdot\|$ we denote the $H_\infty$ norm. Relation (2.29) can be written as

$$\boldsymbol{H}(\boldsymbol{q})\boldsymbol{H}(\boldsymbol{q}^*)^H \prec R(\boldsymbol{q}) \cdot \boldsymbol{I}_{\kappa_1}, \quad \forall \boldsymbol{q} \in \mathcal{D}', \tag{2.30}$$

Table 2.1: Positivity intervals types.

| Case | Domain | MATLAB input | Type of polynomial |
|---|---|---|---|
| 1 | $[a, b] \in (-\pi, \pi)$ | [a b] | trig. pol. complex coef. |
| 2 | $[a, \pi] \in (-\pi, \pi)$ | [a pi] | trig. pol. complex coef. |
| 3 | $[-\pi, b] \in (-\pi, \pi)$ | [-pi b] | trig. pol. complex coef. |
| 4 | $[a, b] \in [0, \pi]$ | [a b] | trig. pol. real coef. |
| 5 | $[a, b] \in (-\infty, \infty)$ | [a b] | real pol. |
| 6 | $[a, \infty)$ | [a Inf] | real pol. |
| 7 | $(-\infty, b]$ | [-Inf b] | real pol. |

with $R(\boldsymbol{q}) = A(\boldsymbol{q})A(\boldsymbol{q}^*)^H$ and $^*$ denoting conjugation.

In POS3POLY, a BRL is defined by the polynomials $\boldsymbol{H}(\boldsymbol{q})$ and $R(\boldsymbol{q})$, sharing the same `KsP` description; note that they share the same degree. The polynomial $R(\boldsymbol{q})$ is described together with the domain $\mathcal{D}$, as a scalar polynomial that is positive on $\mathcal{D}$; the order of its coefficients is as in Section 2.1.1, depending on the type of the polynomial. The coefficients of the causal polynomial $\boldsymbol{H}(\boldsymbol{q})$ are ordered as shown in Section 2.1.3; the size of its matrix coefficients is given in the field `brl.hsize`, which contains the vector $[\kappa_1\ \kappa_2]$; if $\kappa_1 = \kappa_2$, then `hsize` may be a scalar, equal to $\kappa_1$. The presence of the field `brl` indicates that we deal with a BRL and the vectors corresponding to $\boldsymbol{H}(\boldsymbol{q})$ and $R(\boldsymbol{q})$ are connected through the BRL relation.

# Chapter 3

# Demos

We provide some demos for the users to get started with POS3POLY. The directory of the demos application is `demos`.

After setting up POS3POLY, one can use the command

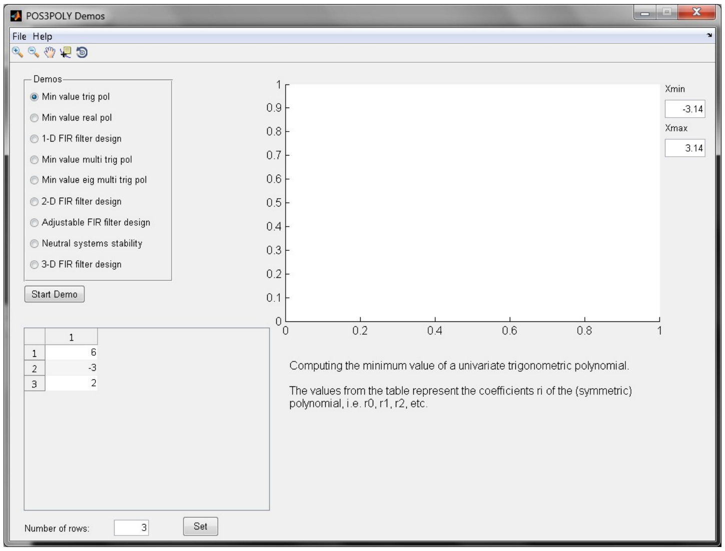
```
>> p3p_demos
```

to start the demos. This command will start the demos application, shown in Figure 3.1.

The POS3POLY Demos is a collection of examples, some of them being discussed in the next chapter. The user can tweak the input parameters of the problems; to start a demo press the `Start Demo` button.

We list now the names of the demos included in the demos application:

- Minimum value of a univariate trigonometric polynomial. See Section 4.1.2.
- Minimum value of a univariate real polynomial. See [1, Section 2.7].
- 1-D FIR filter design. See Section 4.4.1 .
- Minimum value of a multivariate trigonometric polynomial. See [1, Section 3.5.2].
- Minimum eigenvalue of a multivariate trigonometric polynomial. See Section 4.2.2.
- 2-D FIR filter design. See Section 4.4.2.
- Adjustable FIR filter design. See Section 5.2.2.



Xmin: -3.14

Xmax: 3.14

|   | 1 |
|---|---|
| 1 | 6 |
| 2 | -3 |
| 3 | 2 |

Number of rows: 3

Computing the minimum value of a univariate trigonometric polynomial.

The values from the table represent the coefficients ri of the (symmetric) polynomial, i.e. r0, r1, r2, etc.

Figure 3.1: POS3POLY Demos.

- Neutral systems stability. See [4] .
- 3-D FIR filter design.

# Chapter 4

# Examples

We present here some examples of working with POS3POLY. All the examples solved here can be found in the `examples` directory of the POS3POLY library; the function `test_p3p_example` can be used for testing the examples.

In describing the examples, we concentrate on building the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ from (1.1) or the vector $\boldsymbol{w} = \boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}$ from (1.3) and assume that the user knows how to use SeDuMi.

## 4.1 Minimum value of a polynomial

### 4.1.1 A particular polynomial

Consider the (symmetric) trigonometric polynomial

$$R(z) = 2z^2 - 3z + 6 - 3z^{-1} + 2z^{-2}, \tag{4.1}$$

$z \in \mathbb{T}$, for which we want to find the minimum on the unit circle, problem solved in [1, Example 2.12]. The minimum of a polynomial $R(z)$ can be found by solving the optimization problem

$$\begin{array}{rcll} \mu^* & = & \max & \mu \\ & & \text{s.t.} & R(z) - \mu \geq 0, \quad \forall z \in \mathbb{T} \end{array} \tag{4.2}$$

The problem (4.2) is already in the dual form, involving the positivity of a single

Table 4.1: POS3POLY program for solving the problem (4.3).

```
 1  % AsP is a 3x1 matrix multiplying mu
 2  AsP = [ 1 0 0 ]';
 3  % the coefficients of R(z)
 4  csP = [ 6 -3 2 ]';
 5  % the objective function is mu
 6  bsP = 1;
 7  % degree of polynomial and size of coefficients (1x1)
 8  KsP.p{ 1 } = [ 2 1 ];
 9  % univariate trigonometric polynomial
10  KsP.ptype{ 1 }.trigonometric = 1;
11  % call POS3POLY
12  mu = pos3poly( AsP, bsP, csP, KsP );
```

polynomial, which depends on a single variable ($\mu$). The problem (4.2) can be written as

$$\begin{aligned} \mu^\star &= \max \; \mu \\ &\text{s.t.} \quad \begin{bmatrix} 6 \\ -3 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \mu \in \mathbb{P} \end{aligned} \tag{4.3}$$

where now $\mathbb{P}$ is specifically the cone of positive trigonometric polynomials of degree two. Note that the order of the coefficients corresponds to (2.2). Identifying $\boldsymbol{A}^T$, $\boldsymbol{b}$ and $\boldsymbol{c}$ from (1.3), the problem can be solved with the code shown in Table 4.1.

***Comments.*** The matrix $\boldsymbol{A}^T$ is introduced in the variable `AsP` in line 2. The column vector $\boldsymbol{c}$ is introduced in the variable `csP` in line 4. The variable $\boldsymbol{b}$ is the scalar 1 and is retained in the variable `bsP` in line 6. The degree of the polynomial is 2 and its coefficients are scalars, as set in line 8. In line 10 set the polynomial to be trigonometric and univariate. We call POS3POLY to find the minimum in line 12. ■

### 4.1.2 An arbitrary polynomial

Let us consider the general form of the univariate trigonometric polynomial

$$R(z) = \sum_{k=-n}^{n} r_k z^{-k}, \qquad r_{-k} = r_k^* \tag{4.4}$$

with $z$ belonging to the unit circle.

The dual form for the minimum of an arbitrary trigonometric polynomial of degree $n$ is

$$\begin{array}{ll} \max & \mu \\ \text{s.t.} & \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_n \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mu \in \mathbb{P} \end{array} \tag{4.5}$$

where $\mathbb{P}$ is the cone of trigonometric polynomials of degree $n$.

The function that implements the dual form for the minimum of a trigonometric polynomial is `min_poly_value_general_trig_dual`.

### 4.1.3 A multivariate polynomial

We consider a multivariate hybrid polynomial $H(\boldsymbol{z}, \boldsymbol{t})$ as in (2.10), for which we want to find the minimum value.

Using convex optimization, the problem of finding the minimum value for the hybrid polynomial can be cast as

$$\begin{array}{rcll} \mu^* & = & \max & \mu \\ & & \text{s.t.} & H(\boldsymbol{z}, \boldsymbol{t}) - \mu \geq 0, \quad \forall (\boldsymbol{z}, \boldsymbol{t}) \in \mathbb{T}^\ell \times \mathbb{R}^m \end{array} \tag{4.6}$$

The problem (4.6) is hard, so we relax it to

$$\begin{array}{rcll} \mu_1 & = & \max & \mu \\ & & \text{s.t.} & H(\boldsymbol{z}, \boldsymbol{t}) - \mu \text{ is sum-of-squares} \end{array} \tag{4.7}$$

By solving it, we obtain $\mu_1 \leq \mu^*$, but in many practical cases $\mu_1$ is equal to or a very good approximation of $\mu^*$.

The problem (4.7) is in the dual form and one can write it as

$$\begin{array}{rcll} \mu_1 & = & \max & \mu \\ & & \text{s.t.} & \boldsymbol{h} - \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mu \in \mathbb{P} \end{array} \tag{4.8}$$

where $\mathbb{P}$ is the cone of sum-of-squares polynomials with $\ell$ trigonometric variables and $m$ real variables. The length of the vector $\boldsymbol{h}$ is

$$N = \frac{(1 + \prod_{i=1}^{\ell}(2n_i + 1))}{2} \prod_{i=\ell+1}^{\ell+m} (n_i + 1). \tag{4.9}$$

The function that implements the problem (4.8) is listed in Table 4.2 and commented below.

***Comments.*** The variables $\boldsymbol{n}$, $\boldsymbol{h}$ and $\ell$ are given as the input parameters in `n`, `h` and `l`. The number of real variables $m$ is returned in line 3. The column vector $\boldsymbol{A}^T$ is constructed in line 5 using the function `unitpol` which creates the unit polynomial of given degree, size of coefficients and number of trigonometric and real variables. $\boldsymbol{b}$ is the scalar 1 and $\boldsymbol{c}$ is actually the vector $\boldsymbol{h}$. The type of the polynomial is set in lines 7–10. We find the minimum of the polynomial in line 12. ■

The problem (4.8) can be tested using the command

```
test_p3p_example( 53 );
```

Example 4.1.1. Let us consider a simple example and take the hybrid polynomial of degree $\boldsymbol{n} = (1, 2)$,

$$H(z, t) = 3t^2 + z + z^{-1}. \tag{4.10}$$

The degrees of the nonzero coefficients are

```
deg = [ 1 0; 0 2 ];
```

and the corresponding coefficients are

```
coef = { 1, 3 };
```

We construct the vector $\boldsymbol{h}$ with

```
h = sppol2fvec( [ 1 2 ], deg, coef, 1, 1 );
```

Now the minimum can be retrieved with

```
mu = min_poly_value_multi_general_hybrid( [1 2], h, 1 );
```

Table 4.2: POS3POLY program for solving the problem (4.8).

```
function mu = min_poly_value_multi_general_hybrid_dual( n, h, l )

m = length( n ) - 1;

AsP = unitpol( n, 1, l, m );

KsP.ptype{ 1 }.trigonometric = 1;
KsP.ptype{ 1 }.real = m;

KsP.p{ 1 } = [ n 1 ];

mu = pos3poly( AsP, 1, h, KsP );
```

The computed value is $\mu_1 = -2$. (`sppol2fvec` is the function that creates the vector that describes a polynomial using the nonzero coefficients of the polynomial.)

Alternatively the vector $\boldsymbol{h}$ can be constructed by inserting the nonzero coefficients in the zero polynomial. We store the degree of the polynomial with

```
n = [ 1 2 ];
```

We create a zero polynomial

```
h = zeropol( n, 1, 1, 1 );
```

and insert the nonzero coefficients

```
h( coefpos( n, [ 1 0 ], 1, 1 ) ) = 1;
h( coefpos( n, [ 0 2 ], 1, 1 ) ) = 3;
```

where `zeropol` is the function that creates the zero polynomial and `coefpos` is the function that returns the position of a coefficient in the vector that describes a polynomial. ■

## 4.2 Minimal value of the smallest eigenvalue

### 4.2.1 A univariate polynomial

Let us consider a univariate real matrix polynomial

$$\boldsymbol{P}(t) = \sum_{k=0}^{n} \boldsymbol{P}_k t^k, \quad \boldsymbol{P}_k = \boldsymbol{P}_k^T, \tag{4.11}$$

which is a particular case ($d = 1$) of (2.18).

The problem of finding the minimum value of the smallest eigenvalue can be written as

$$\begin{array}{rcl} \mu^* & = & \max \ \mu \\ & & \text{s.t.} \quad \boldsymbol{P}(t) - \mu \boldsymbol{I}_\kappa \succeq \boldsymbol{0}, \quad \forall t \in \mathbb{R} \end{array} \tag{4.12}$$

Explicitly the problem (4.12) can be written as

$$\begin{array}{rcl} \mu^* & = & \max \ \mu \\ & & \text{s.t.} \quad \begin{bmatrix} \text{vecs}(\boldsymbol{P}_0) \\ \text{vecs}(\boldsymbol{P}_1) \\ \vdots \\ \text{vecs}(\boldsymbol{P}_n) \end{bmatrix} - \begin{bmatrix} \text{vecs}(\boldsymbol{I}_\kappa) \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix} \mu \in \mathbb{P} \end{array} \tag{4.13}$$

where $\mathbb{P}$ is the cone of univariate real matrix polynomials. The function that solves the problem (4.13) is `min_matrix_poly_eigen_general_real_dual`, which is shown in Table 4.3 and commented below.

***Comments.*** We put the matrix coefficients in a 3-D array. The length of the third dimension of the array is the number $(n + 1)$ of matrix coefficients, obtained in line 3, which gives the degree $n$ of the polynomial, in line 4. Next, in line 6, we retain in the variable `K` the dimension of the matrix coefficients. The only variable in the problem is $\mu$, hence the matrix $\boldsymbol{A}^T$, denoted as `AsP`, is a vector. The matrix $\boldsymbol{A}^T$ is constructed in line 8 using the `unitpol` function. The POS3POLY structure has one positive univariate real polynomial with matrix coefficients, as stated in lines 10–11. All being set, in line 14, the `pos3poly` function is used to solve the optimization problem and return the minimum eigenvalue. Note the usage of the vecg() function in line 14, for obtaining the $\boldsymbol{c}$ term in (1.3); check the MATLAB help for details on this function. ■

EXAMPLE 4.2.1. Consider the real polynomial with matrix coefficients

$$\boldsymbol{P}(t) = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} t + \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} t^2. \tag{4.14}$$

Table 4.3: POS3POLY program for solving problem (4.13).

```
function mu = min_matrix_poly_eigen_general_real_dual( P )

n1 = size( P, 3 );% n1 = n + 1 (degree + 1)
n = n1 - 1;       % the degree of the polynomial

K = size( P, 1 ); % matrix dimension

AsP = unitpol( n, K, 0, 1 ); % initialize AsP

KsP.ptype{ 1 }.real = 1; % one real variable
KsP.p{ 1 } = [ n K ];    % degree & matrix dimension

% use POS3POLY
mu = pos3poly( AsP, 1, vecg( P, 0 ), KsP );
```

We put the three matrices in a 3-D array, in MATLAB, as follows:

```
P( :, :, 1 ) = [ 5 0; 0 5 ];
P( :, :, 2 ) = [ 0 1; 1 0 ];
P( :, :, 3 ) = [ 2 0; 0 2 ];
```

Now we use the function that finds $\mu^*$:

```
mu = min_matrix_poly_eigen_general_real_dual( P );
```

We find that $\mu^* = 4.8750$. ■

### 4.2.2 A multivariate polynomial

We discuss here the problem of computing the minimum value of the smallest eigenvalue of a multivariate trigonometric polynomial (2.15). The problem can be expressed using positive polynomials by

$$\begin{array}{rcl} \mu^* & = & \max \quad \mu \\ & & \text{s.t.} \quad \boldsymbol{R}(\boldsymbol{z}) - \mu \boldsymbol{I}_\kappa \geq 0, \quad \forall \boldsymbol{z} \in \mathbb{T}^d \end{array} \tag{4.15}$$

The problem (4.15) is relaxed via sum-of-squares to

$$\begin{array}{rcl} \mu_1 & = & \max \quad \mu \\ & & \text{s.t.} \quad \boldsymbol{R}(\boldsymbol{z}) - \mu \boldsymbol{I}_\kappa \text{ is sum-of-squares} \end{array} \tag{4.16}$$

The problem (4.16) is in dual form, hence can be written as

$$\begin{array}{rcl} \mu^\star & = & \max \quad \mu \\ & & \text{s.t.} \quad \begin{bmatrix} \operatorname{vecs}(\boldsymbol{R}_{0,\ldots,0}) \\ \operatorname{vec}(\boldsymbol{R}_{1,0,\ldots,0}) \\ \vdots \\ \operatorname{vec}(\boldsymbol{R}_{n_1,\ldots,n_d}) \end{bmatrix} - \begin{bmatrix} \operatorname{vecs}(\boldsymbol{I}_\kappa) \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix} \mu \in \mathbb{P} \end{array} \tag{4.17}$$

The lengths of the vectors from the problem (4.17) are equal to $(M-1)\kappa^2 + \frac{\kappa(\kappa+1)}{2}$, where where $M$ is as in (2.3).

We present in Table 4.4 an example of solving the problem (4.17) for the polynomial

$$\boldsymbol{R}(\boldsymbol{z}) = \operatorname{sym}^{-1} + \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} z_1 + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} z_2 + \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} z_1 z_2. \tag{4.18}$$

***Comments.*** The matrix polynomial has degree `n = [1 1]`, as set in line 1. In lines 3–6 we set the degrees for the nonzero coefficients of the polynomial. The nonzero matrix coefficients are set in lines 8–11. We set in line 13 the size of the matrix coefficients $\kappa$. Using the `sppol2fvec` function, in line 14 we construct the vector that characterizes the polynomial $\boldsymbol{R}$, which is takes the place of the vector $\boldsymbol{c}$. The `sppol2fvec` function inserts the nonzero coefficients into their positions. Finally, in line 16 we solve the problem (4.16) for the polynomial (4.18) and we obtain $\mu_1 = 1$. ■

Table 4.4: Example of solving the problem (4.16).

```
n = [ 1 1 ];

deg( 1, :  )  = [ 0 0 ];
deg( 2, :  )  = [ 1 0 ];
deg( 3, :  )  = [ 0 1 ];
deg( 4, :  )  = [ 1 1 ];

coef{ 1 } = [ 4 0; 0 4 ];
coef{ 2 } = [ 1 0; 0 0 ];
coef{ 3 } = [ 0 0; 0 1 ];
coef{ 4 } = [ 0 0; -1 0 ];

K = length( coef{ 1 } );
r = sppol2fvec( n, deg, coef, K, 2 );

mu = min_matrix_poly_eigen_multi_general_trig_dual( n, r, K );
```

## 4.3 Nearest autocorrelation

Suppose we are given the Hermitian sequence $\hat{r}_k$, $k = -n : n$, with $\hat{r}_{-k} = \hat{r}_k^*$. We want to find the nonnegative sequence $r_k$ that is nearest from $\hat{r}_k$. The problem is discussed in [1, Sections 1.2 and 2.2].

Our problem can be expressed as

$$\begin{array}{ll} \min\limits_{\boldsymbol{r}} & (\boldsymbol{r} - \hat{\boldsymbol{r}})^H \boldsymbol{\Gamma} (\boldsymbol{r} - \hat{\boldsymbol{r}}) \\ \text{s.t.} & R(\omega) \geq 0, \quad \forall \omega \in [-\pi, \pi] \end{array} \tag{4.19}$$

where $\boldsymbol{r} = [r_0 \ r_1 \ldots r_n]^T \in \mathbb{C}^{n+1}$ and $\boldsymbol{\Gamma} = \operatorname{diag}(1, 2, \ldots, 2)$ (this value stands for the Euclidian norm; other positive definite matrices $\boldsymbol{\Gamma}$ could be used for other quadratic norms).

Taking into account that

$$(\boldsymbol{r}-\hat{\boldsymbol{r}})^H\boldsymbol{\Gamma}(\boldsymbol{r}-\hat{\boldsymbol{r}}) = ||\boldsymbol{\Gamma}^{\frac{1}{2}}(\boldsymbol{r}-\hat{\boldsymbol{r}})||^2 \tag{4.20}$$

the problem (4.19) is equivalent to

$$\begin{array}{ll} \min\limits_{\alpha,\boldsymbol{y},\boldsymbol{r}} & \alpha \\ \text{s.t.} & ||\boldsymbol{y}|| \leq \alpha \\ & R(z) \geq 0, \quad \forall z \in \mathbb{T} \end{array} \tag{4.21}$$

where $\boldsymbol{y} = \boldsymbol{\Gamma}^{\frac{1}{2}}(\boldsymbol{r}-\hat{\boldsymbol{r}})$. The first constraint of (4.21) is that $\alpha$ and $\boldsymbol{y}$ belong to a second-order cone (SOC).

Knowing that $-\boldsymbol{\Gamma}^{-\frac{1}{2}}\boldsymbol{y}+\boldsymbol{r} = \hat{\boldsymbol{r}}$ the matrix equality that describes the equality constraints of the problem, i.e. the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ from (1.1), is

$$\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{\Gamma}^{-\frac{1}{2}} & \boldsymbol{I}_{n+1} \end{bmatrix} \begin{bmatrix} \alpha \\ \boldsymbol{y} \\ \boldsymbol{r} \end{bmatrix} = \hat{\boldsymbol{r}}. \tag{4.22}$$

Hence, the problem (4.19) can be written in the primal form as

$$\begin{array}{ll} \min\limits_{\alpha,\boldsymbol{y},\boldsymbol{r}} & \alpha \\ \text{s.t.} & (4.22) \\ & ||\boldsymbol{y}|| \leq \alpha \\ & R(z) \geq 0, \quad \forall z \in \mathbb{T} \end{array} \tag{4.23}$$

The function that solves the problem (4.23) is `nearest_autocorrelation`.

## 4.4 Optimization of linear-phase FIR filters

### 4.4.1 One-dimensional filters

We tackle now the problem of designing linear-phase FIR filters of even order $\tilde{n} = 2n$, using optimization, problem which can be found in [1, Section 5.1.1]. We optimize only the magnitude, therefore we can work with zero-phase filters

$$H(z) = \sum_{k=-n}^{n} h_k z^{-k}, \quad h_{-k} = h_k. \tag{4.24}$$

A peak constrained least-squares (PCLS) problem can be formulated as

$$\begin{array}{ll}\min\limits_{H\in\mathbb{R}_n[z]} & E_s \\ \text{s.t.} & 1+\gamma_p - H(\omega) \geq 0, \forall\omega\in[0,\omega_p] \\ & H(\omega)-1+\gamma_p \geq 0, \forall\omega\in[0,\omega_p] \\ & \gamma_s - H(\omega) \geq 0, \forall\omega\in[\omega_s,\pi] \\ & H(\omega)+\gamma_s \geq 0, \forall\omega\in[\omega_s,\pi]\end{array} \tag{4.25}$$

where $E_s$ is the stopband energy of the FIR filter and $\gamma_p$, $\gamma_s$ are the passband and stopband given error bounds, respectively. The filter is lowpass, $\omega_p$ and $\omega_s$ being the edges of the passband and stopband, respectively. We have now a problem with polynomials that are nonnegative on given intervals.

Let us see the expression of the stopband energy. Considering $\boldsymbol{h} = [h_0\ h_1 \ldots h_n]^T$ we have

$$E_s = \boldsymbol{h}^T\tilde{\boldsymbol{C}}\boldsymbol{h},\ \text{with}\ \tilde{\boldsymbol{C}} = \boldsymbol{P}^T\boldsymbol{C}\boldsymbol{P} \succeq \boldsymbol{0} \tag{4.26}$$

where

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{J}_n \\ 1 & 0 \\ \boldsymbol{0} & \boldsymbol{I}_n \end{bmatrix} \tag{4.27}$$

and $\boldsymbol{C} = \text{Toeplitz}(c_0, c_1, \ldots, c_{\tilde{n}}) \succeq \boldsymbol{0}$ with

$$c_k = \begin{cases} 1-\omega_s/\pi, & \text{if } k=0; \\ -\dfrac{\sin k\omega_s}{k\pi}, & \text{if } k>0. \end{cases} \tag{4.28}$$

($\boldsymbol{J}_n$ is the counteridentity matrix of size $n\times n$.)

The problem (4.25) is equivalent to

$$\begin{array}{ll}\min\limits_{\varepsilon,\boldsymbol{h}} & \varepsilon \\ \text{s.t.} & ||\tilde{\boldsymbol{C}}^{\frac{1}{2}}\boldsymbol{h}|| \leq \varepsilon \\ & 1+\gamma_p - H(\omega) \geq 0, \forall\omega\in[0,\omega_p] \\ & H(\omega)-1+\gamma_p \geq 0, \forall\omega\in[0,\omega_p] \\ & \gamma_s - H(\omega) \geq 0, \forall\omega\in[\omega_s,\pi] \\ & H(\omega)+\gamma_s \geq 0, \forall\omega\in[\omega_s,\pi]\end{array} \tag{4.29}$$

The following expression characterizes the inequality constraints of the problem (4.29):

$$\left[\begin{array}{c} 0 \\ \hline \mathbf{0} \\ \hline 1+\gamma_p \\ \mathbf{0} \\ \hline -1+\gamma_p \\ \mathbf{0} \\ \hline \gamma_s \\ \mathbf{0} \\ \hline \gamma_s \\ \mathbf{0} \end{array}\right] - \left[\begin{array}{cc} -1 & \mathbf{0} \\ \hline \mathbf{0} & -\tilde{\boldsymbol{C}}^{\frac{1}{2}} \\ \hline \mathbf{0} & \boldsymbol{I}_{n+1} \\ \hline \mathbf{0} & -\boldsymbol{I}_{n+1} \\ \hline \mathbf{0} & \boldsymbol{I}_{n+1} \\ \hline \mathbf{0} & -\boldsymbol{I}_{n+1} \end{array}\right] \left[\begin{array}{c} \varepsilon \\ \boldsymbol{h} \end{array}\right]. \tag{4.30}$$

The function `lin_phase_fir_dual` can be used to design linear-phase FIR filters (see the code for the usage of the field `int` for describing polynomials that are positive on an interval). For instance, choosing $\tilde{n} = 50$, $\omega_p = 0.2\pi$, $\omega_s = 0.25\pi$, $\gamma_p = 0.1$, $\gamma_s = 0.0158$, one should enter the following command in MATLAB:

```
th = lin_phase_fir_dual( 25, 0.2 * pi, 0.25 * pi, 0.1, 0.0158 );
```

(The value given above to $\gamma_s$ corresponds to a stopband attenuation of 36 dB.)

To plot the frequency response of the filter (shown in Figure 4.1), one can use the command

```
fvtool( [ flipud( th( 2 :  end ) ); th ] );
```

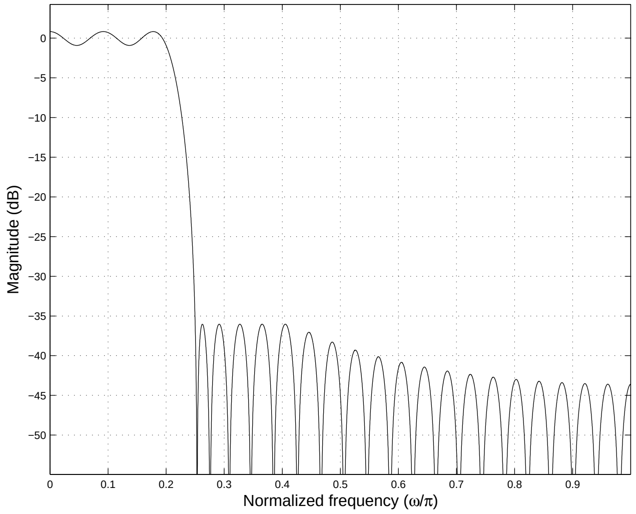


Figure 4.1: Frequency response of linear-phase FIR filter.

### 4.4.2 Two-dimensional filters—first variant

We consider here the minimax optimization of 2-D linear-phase FIR filters with diamond shape passband. The problem can be stated as [1, Section 5.2.2]

$$\begin{array}{ll} \min\limits_{\gamma_s, H} & \gamma_s \\ \text{s.t.} & 1 + \gamma_p - H(\boldsymbol{\omega}) \geq 0, \quad \forall \boldsymbol{\omega} \\ & -1 + \gamma_p + H(\boldsymbol{\omega}) \geq 0, \quad \forall \boldsymbol{\omega} \in \mathcal{D}_p \\ & \gamma_s - H(\boldsymbol{\omega}) \geq 0, \quad \forall \boldsymbol{\omega} \in \mathcal{D}_s \\ & \gamma_s + H(\boldsymbol{\omega}) \geq 0, \quad \forall \boldsymbol{\omega} \in \mathcal{D}_s \end{array} \tag{4.31}$$

where $H(\boldsymbol{\omega})$ is a bivariate filter, $d = 2$ in (2.1), $\gamma_p$ and $\gamma_s$ are passband and stopband error bound, respectively, and $\mathcal{D}_p$, $\mathcal{D}_s$ are the passband and stopband domain, respectively. A diamond shape is obtained by adopting the following definitions (only the constants 0.1 and 0.7 have to be changed below in order to change the size of the passband and stopband):

$$\begin{array}{rcl} \mathcal{D}_p & = & \{\omega_{1,2} \mid \cos(\omega_1 + \omega_2) - 0.1 \geq 0, \cos(\omega_1 - \omega_2) - 0.1 \geq 0, \cos\omega_1 + \cos\omega_2 \geq 0\} \\ \mathcal{D}_s & = & \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3} \end{array} \tag{4.32}$$

where

$$\begin{array}{rcl} \mathcal{D}_{s_1} & = & \{\omega_{1,2} \mid -\cos(\omega_1+\omega_2) - 0.7 \geq 0\} \\ \mathcal{D}_{s_2} & = & \{\omega_{1,2} \mid -\cos(\omega_1-\omega_2) - 0.7 \geq 0\} \\ \mathcal{D}_{s_3} & = & \{\omega_{1,2} \mid -\cos\omega_1 - \cos\omega_2 \geq 0\}. \end{array} \tag{4.33}$$

The relations (4.32) and (4.33) can be written equivalently as

$$\begin{array}{rcl} \mathcal{D}_p & = & \{z_{1,2} \mid D_{p_1}(z_1,z_2) \geq 0,\ D_{p_2}(z_1,z_2) \geq 0,\ D_{p_3}(z_1,z_2) \geq 0\} \\ \mathcal{D}_{s_1} & = & \{z_{1,2} \mid D_{s_1}(z_1,z_2) \geq 0\} \\ \mathcal{D}_{s_2} & = & \{z_{1,2} \mid D_{s_2}(z_1,z_2) \geq 0\} \\ \mathcal{D}_{s_3} & = & \{z_{1,2} \mid D_{s_3}(z_1,z_2) \geq 0\} \end{array} \tag{4.34}$$

where

$$\begin{array}{rcl} D_{p_1}(z_1,z_2) & = & (z_1z_2 + z_1^{-1}z_2^{-1})/2 - 0.1 \\ D_{p_2}(z_1,z_2) & = & (z_1^{-1}z_2 + z_1z_2^{-1})/2 - 0.1 \\ D_{p_3}(z_1,z_2) & = & (z_1 + z_1^{-1})/2 + (z_2 + z_2^{-1})/2 \\ D_{s_1}(z_1,z_2) & = & -(z_1z_2 + z_1^{-1}z_2^{-1})/2 - 0.7 \\ D_{s_2}(z_1,z_2) & = & -(z_1^{-1}z_2 + z_1z_2^{-1})/2 - 0.7 \\ D_{s_3}(z_1,z_2) & = & -(z_1 + z_1^{-1})/2 - (z_2 + z_2^{-1})/2. \end{array} \tag{4.35}$$

The problem (4.31) can be written as

$$\begin{array}{ll} \min\limits_{\gamma_s, H} & \gamma_s \\ & 1 + \gamma_p - H(\boldsymbol{z}) \text{ is sum-of-squares} \\ & -1 + \gamma_p + H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_p \\ & \gamma_s - H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_1} \\ & \gamma_s - H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_2} \\ & \gamma_s - H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_3} \\ & \gamma_s + H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_1} \\ & \gamma_s + H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_2} \\ & \gamma_s + H(\boldsymbol{z}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_3} \end{array} \tag{4.36}$$

The problem (4.36) is difficult to solve, reason for which we relax it via Theorem A.3.1

(see the Appendix), using sum-of-squares. Hence, denoting

$$\begin{array}{rcl} S_0(\boldsymbol{z}) & = & 1+\gamma_p - H(\boldsymbol{z}) \\ S_1(\boldsymbol{z}) & = & -1+\gamma_p + H(\boldsymbol{z}) \\ S_2(\boldsymbol{z}) & = & \gamma_s - H(\boldsymbol{z}) \\ S_3(\boldsymbol{z}) & = & \gamma_s - H(\boldsymbol{z}) \\ S_4(\boldsymbol{z}) & = & \gamma_s - H(\boldsymbol{z}) \\ S_5(\boldsymbol{z}) & = & \gamma_s + H(\boldsymbol{z}) \\ S_6(\boldsymbol{z}) & = & \gamma_s + H(\boldsymbol{z}) \\ S_7(\boldsymbol{z}) & = & \gamma_s + H(\boldsymbol{z}) \end{array} \tag{4.37}$$

the problem (4.36) is relaxed to

$$\begin{array}{ll} \min\limits_{\gamma_s, H, \boldsymbol{s}_i, i=0:7} & \gamma_s \\ & S_0(\boldsymbol{z}) = \mathcal{S}_0(\boldsymbol{z}) \\ & S_1(\boldsymbol{z}) = \mathcal{S}_1(\boldsymbol{z}) + D_{p_1}(\boldsymbol{z})\mathcal{S}_2(\boldsymbol{z}) + D_{p_2}(\boldsymbol{z})\mathcal{S}_3(\boldsymbol{z}) + D_{p_3}(\boldsymbol{z})\mathcal{S}_4(\boldsymbol{z}) \\ & S_2(\boldsymbol{z}) = \mathcal{S}_5 + D_{s_1}(\boldsymbol{z})\mathcal{S}_6(\boldsymbol{z}) \\ & S_3(\boldsymbol{z}) = \mathcal{S}_7 + D_{s_2}(\boldsymbol{z})\mathcal{S}_8(\boldsymbol{z}) \\ & S_4(\boldsymbol{z}) = \mathcal{S}_9 + D_{s_3}(\boldsymbol{z})\mathcal{S}_{10}(\boldsymbol{z}) \\ & S_5(\boldsymbol{z}) = \mathcal{S}_{11} + D_{s_1}(\boldsymbol{z})\mathcal{S}_{12}(\boldsymbol{z}) \\ & S_6(\boldsymbol{z}) = \mathcal{S}_{13} + D_{s_2}(\boldsymbol{z})\mathcal{S}_{14}(\boldsymbol{z}) \\ & S_7(\boldsymbol{z}) = \mathcal{S}_{15} + D_{s_3}(\boldsymbol{z})\mathcal{S}_{16}(\boldsymbol{z}) \end{array} \tag{4.38}$$

where $\mathcal{S}_i(\boldsymbol{z})$, $i = 0:16$, are sum-of-squares. However, the user does not need to implement problem (4.38) explicitly, but only describe the following problem

$$\begin{array}{ll} \min\limits_{\gamma_s, H, \boldsymbol{s}_i, i=0:7} & \gamma_s \\ & S_0(\boldsymbol{z}) \text{ is sum-of-squares} \\ & S_1(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_p \\ & S_2(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_1} \\ & S_3(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_2} \\ & S_4(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_3} \\ & S_5(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_1} \\ & S_6(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_2} \\ & S_7(\boldsymbol{\omega}) \geq 0,\ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_3} \end{array} \tag{4.39}$$

POS3POLY takes care of the sum-of-squares relaxation.

The expression $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}$ from (1.3) is

$$\left[\begin{array}{c} 1+\gamma_p \\ \mathbf{0} \\ \hline -1+\gamma_p \\ \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array}\right] - \left[\begin{array}{cc} \mathbf{0} & \boldsymbol{I}_M \\ \hline \mathbf{0} & -\boldsymbol{I}_M \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & \boldsymbol{I}_M \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & \boldsymbol{I}_M \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & \boldsymbol{I}_M \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & -\boldsymbol{I}_M \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & -\boldsymbol{I}_M \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & -\boldsymbol{I}_M \end{array}\right] \begin{bmatrix} \gamma_s \\ \boldsymbol{h} \end{bmatrix} \tag{4.40}$$

where $M$ is defined as in (2.3).

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

### Full polynomial description

We consider here the description of the positivity for the polynomial $S_1(\boldsymbol{z})$ using all the coefficients of the polynomials. (See also the function `lin_phase_fir2d_v2_dual.m`, which uses the full description of the polynomials which describe the positivity.)

The polynomial $S_1(\boldsymbol{z})$ is positive on $\mathcal{D}_p$—which is defined by the positivity of three polynomials, as (4.34) 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 = [ -0.1 0 0 0 0.5 -0.1 0 0.5 0 0 0 0.5 0 0.5 0 ];
```

### Sparse polynomial description

We take now the case where we describe the polynomials used for positivity as sparse polynomials. We exemplify on the polynomial $S_1(\boldsymbol{z})$. (See also the function `lin_phase_fir2d_dual.m`, 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 $S_1(\boldsymbol{z})$ 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 = [ -0.1 0.5 -0.1 0.5 0.5 0.5 ];
```

A filter $H(\boldsymbol{z})$ like the one in (4.38) can be designed using the `test_p3p_example.m`:

```
test_p3p_example( 56 );
```

Figure 4.2 shows a 2-D FIR filter designed using the problem (4.38).

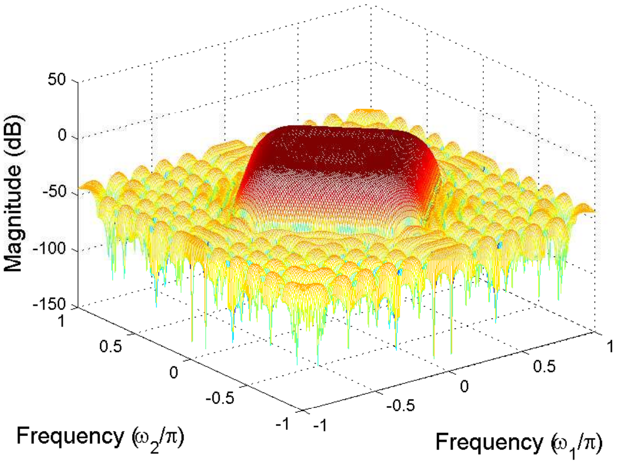


Figure 4.2: Frequency response of linear-phase 2-D FIR filter for $\boldsymbol{n} = (7, 7)$, $\gamma_p = 0.1$.

### 4.4.3 Two-dimensional filters—second variant

A shorter program is obtained by substituting the positivity of several equal polynomials on multiple domains with the positivity of a single polynomial on a union of domains. To do so, one must use the fields `nunion`, `nc`, `deg` and `coef`.

Instead of (4.36) we consider the equivalent problem

$$\begin{array}{ll} \displaystyle\min_{\gamma_s, H, \boldsymbol{s}_i, i=0:3} & \gamma_s \\ & S_0(\boldsymbol{z}) \text{ is sum-of-squares} \\ & S_1(\boldsymbol{\omega}) \geq 0, \ \forall \boldsymbol{\omega} \in \mathcal{D}_p \\ & S_2(\boldsymbol{\omega}) \geq 0, \ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3} \\ & S_3(\boldsymbol{\omega}) \geq 0, \ \forall \boldsymbol{\omega} \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \cup \mathcal{D}_{s_3} \end{array} \tag{4.41}$$

where

$$\begin{array}{rcl} S_0(\boldsymbol{z}) & = & 1 + \gamma_p - H(\boldsymbol{z}) \\ S_1(\boldsymbol{z}) & = & -1 + \gamma_p + H(\boldsymbol{z}) \\ S_2(\boldsymbol{z}) & = & \gamma_s - H(\boldsymbol{z}) \\ S_3(\boldsymbol{z}) & = & \gamma_s + H(\boldsymbol{z}). \end{array} \tag{4.42}$$

The expression $\boldsymbol{c} - \boldsymbol{A}^T \boldsymbol{y}$ that characterizes the problem (4.41) is

$$\left[ \begin{array}{c} 1+\gamma_p \\ \mathbf{0} \\ \hline -1+\gamma_p \\ \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array} \right] - \left[ \begin{array}{cc} \mathbf{0} & \boldsymbol{I}_M \\ \hline \mathbf{0} & -\boldsymbol{I}_M \\ \hline \begin{smallmatrix} -1 \\ \mathbf{0} \end{smallmatrix} & \boldsymbol{I}_M \\ \hline \begin{smallmatrix} -1 \\ \mathbf{0} \end{smallmatrix} & -\boldsymbol{I}_M \end{array} \right] \left[ \begin{array}{c} \gamma_s \\ \boldsymbol{h} \end{array} \right] \tag{4.43}$$

with $M$ as in (2.3).

The program that solves the problem (4.41), in dual form, is in Table 4.5 and is discussed in the next paragraph.

***Comments.*** We initialize the variables `AsP`, `bsP`, `csP` in lines 5–7. (The function `lenpol` returns the number of coefficients used to characterize a polynomial.) We set the objective function in line 8. In lines 9–10 we set the nonzero elements of the $\boldsymbol{c}$ vector from (1.3). The lines 11–16 set the matrix $\boldsymbol{A}^T$. The numbers of variables for the four polynomials are set in lines 17–18. The positivity domain for the second polynomial is set in lines 19–21. The positivity domains for the last two polynomials are set in lines 22–29. Remark the usage of the field `nunion`: each domain used in the union is defined by one polynomial. The degrees of the polynomials are set in lines 30–31. We call the POS3POLY library in line 32. Finally, in line 33 we obtain the filter. ■

To design a 2-D filter using the program from Table 4.5 one can use the following commands:

```
n = [ 7 7 ];

h = lin_phase_fir2d_v3_dual( n, 0.1 );

freqz2d( half2all2d( h, n ) );
```

## 4.5 The stabilizability radius

We present now a method for computing the stabilizability radius, which is discussed in [3]. Considering a pair of matrices $(\boldsymbol{A}, \boldsymbol{B})$, the distance to unstabilizability is approximated

Table 4.5: POS3POLY program for solving problem (4.41).

```
function [ h ] = lin_phase_fir2d_v3_dual( n, gp )
N = lenpol( n ); % number of coefficients
nConstr = 4 * N; nVar = N + 1;
I = speye( N );
AsP = sparse( nConstr, nVar ); % initialize POS3POLY
bsP = sparse( 1, nVar ); % variables
csP = sparse( nConstr, 1 );
bsP( 1 ) = -1; % maximize -gs
csP( 1 ) = 1 + gp; % free terms for the
csP( 1 + N ) = -1 + gp; % polynomial constraints
AsP( 1:  N, 2 :  N + 1 ) = I; % 1 + gp - H(w) >= 0
AsP( N + 1 :  2 * N, 2 :  N + 1 ) = -I; % -1 + gp + H(w) >= 0, Dp
AsP( 2 * N + 1 , 1 ) = -1; % gs - H(w) >= 0, Ds1, Ds2, Ds3
AsP( 2 * N + 1 :  3 * N, 2 :  N + 1 ) = I;
AsP( 3 * N + 1 , 1 ) = -1; % gs + H(w) >= 0, Ds1, Ds2, Ds3
AsP( 3 * N + 1 :  4 * N, 2 :  N + 1 ) = -I;
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 = [ -0.1 0.5 -0.1 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 = [ -0.7 -0.5 -0.7 -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 = [ -0.7 -0.5 -0.7 -0.5 -0.5 -0.5 ];
KsP.p{ 1 } = [ n 1 ]; KsP.p{ 2 } = [ n 1 ];
KsP.p{ 3 } = [ n 1 ]; KsP.p{ 4 } = [ n 1 ];
[ y ] = pos3poly( AsP, bsP, csP, KsP ); % use POS3POLY
h = y( 2 :  N + 1 ); % "half" of the coefficients of the filter
```

by solving the problem

$$\begin{array}{ll} \max & \tau \\ \text{s.t.} & \boldsymbol{P}(t_1,t_2) - \tau\boldsymbol{I} = \mathcal{S}_0(t_1,t_2) + t_1\mathcal{S}_1(t_1,t_2) + (r^2 - t_1^2 - t_2^2)\mathcal{S}_2 \end{array} \tag{4.44}$$

where $r$ is an upper bound depending on $\boldsymbol{A}$ (and is constant), $\mathcal{S}_i$, $i = 0 : 2$, are sum-of-squares and

$$\boldsymbol{P}(t_1,t_2) = t_1^2\boldsymbol{I} + t_2^2\boldsymbol{I} - t_1(\boldsymbol{A} + \boldsymbol{A}^H) - jt_2(\boldsymbol{A}^H - \boldsymbol{A}) + \boldsymbol{A}\boldsymbol{A}^H + \boldsymbol{B}\boldsymbol{B}^H. \tag{4.45}$$

The stabilizability radius is $\sqrt{\tau}$.

We denote $\boldsymbol{S}(t_1,t_2) = \boldsymbol{P}(t_1,t_2) - \tau\boldsymbol{I}$, of degree $(n_1,n_2)$, and remark that the constraint of (4.44) is the sum-of-squares relaxation of the positivity of $\boldsymbol{S}(t_1,t_2)$ on a domain defined by the positivity of the polynomials

$$\begin{array}{l} D_1(t_1,t_2) = t_1, \\ D_2(t_1,t_2) = r^2 - t_1^2 - t_2^2. \end{array} \tag{4.46}$$

The expression $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}$ for the dual form in POS3POLY is

$$\begin{array}{c} (0,0) \\ (1,0) \\ (2,0) \\ (0,1) \\ (1,1) \\ (2,1) \\ (0,2) \\ (1,2) \\ (2,2) \end{array} \begin{bmatrix} \text{vecs}(\boldsymbol{A}\boldsymbol{A}^H + \boldsymbol{B}\boldsymbol{B}^H) \\ \text{vecs}(-(\boldsymbol{A} + \boldsymbol{A}^H)) \\ \text{vecs}(\boldsymbol{I}) \\ \text{vecs}(-j(\boldsymbol{A}^H - \boldsymbol{A})) \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \text{vecs}(\boldsymbol{I}) \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \text{vecs}(\boldsymbol{I}) \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} \tau, \tag{4.47}$$

where the length of the vectors from (4.47) is $N = (n_1 + 1)(n_2 + 1)\kappa(\kappa + 1)/2$, $\boldsymbol{S}_{i,j}$, $i = 0 : n_1$, $j = 0 : n_2$, are the coefficients of the polynomial $\boldsymbol{S}(t_1,t_2)$ and the matrix $\boldsymbol{A}$ is of size $\kappa \times \kappa$. In the left most part of (4.47) we show the degrees corresponding to the matrix coefficients.

We present in Table 4.6 the program `dist2unstab_r_dual.m` which solves the problem (4.44) in the dual form. Next, we make some remarks on how we have implemented problem program from Table 4.6.

***Comments.*** Let `n` (a vector of two elements) be the degree $\boldsymbol{n} = (n_1, n_2)$ of the polynomial $\boldsymbol{P}(t_1,t_2)$. We put the degrees of the nonzero coefficients in the variable `deg`—line

9; we set the variable `coef` with those coefficients in lines 11–12. The variable $\tau$ is maximized, hence `csP` is set the scalar 1 in line 13. We initialize `AsP` in line 15. Next, in line 16 we use the `sppol2fvec` function to form the vector $\boldsymbol{c}$. We begin defining the structure of the POS3POLY problem in line 17 where we define one bivariate real polynomial. The degree of the polynomial is `n` and the size of the coefficients is `K`, facts declared in line 18. Three lines, 21, 23, 25, describe the positivity domain of the polynomial. In the line 23 appear the degrees of the polynomials (4.46) defining the positivity domain, while in line 25, their coefficients. The first polynomial has one coefficient and the second has three coefficients, as set in line 21. In line 26 we declare that the coefficients are complex and we treat the constraints as equalities between complex numbers in line 27. We call the POS3POLY library in line 28. The stabilizability radius is obtained in line 29. ■

For the pair of matrices

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 1 \\ 0.1 & 3 & 5 \\ 0 & -1 & -1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1 \\ 0.1 \\ 0 \end{bmatrix}, \tag{4.48}$$

the `dist2unstab_r.m` function returns the value 0.0392404 for the stabilizability radius.

Table 4.6: POS3POLY program for solving problem (4.44).

```
function sigma = dist2unstab_r_dual( A, B, n )
K  = size( A, 1 ); % size of the matrix A
hK = sumn( K );    % nr.  elem.  half matrix
% ident.  mat.; nr.  of coefs.; j
I = speye( K ); N = prod( n + 1 );
r = sqrt( max( abs( eig( ( A + A' ) / 2 ) ) ) ^ 2 + ...
          max( abs( eig( ( A - A' ) / ( 2 * 1i ) ) ) ) ^ 2 );
% degrees for nonzero coefs.  of P(t1,t2)
deg = [ 2 0; 0 2; 1 0; 0 1; 0 0 ];
% nonzero coefficiets of the polynomial P(t1,t2)
coef{ 1 } = I; coef{ 2 } = I; coef{ 3 } = -( A + A' );
coef{ 4 } = -1i * ( A' - A ); coef{ 5 } = A * A' + B * B';
bsP = 1; % build 'bsP'
nConstr = N * hK;      % nr.  of POS3POLY constraints
AsP = unitpol( n, K, 0, 2 );
csP = sppol2fvec( 1, 5, n, deg, coef, K, 0, 1 );
KsP.ptype{ 1 }.real = 2; % 2-D polynomial P(t1,t2)
KsP.p{ 1 } = [ n K ];  % degree and size of the coefficients
% positivity domain (pos.  dom.);
% number of coefficients
KsP.ptype{ 1 }.dom.nc = [ 1 3 ];
% degrees of the coefficients
KsP.ptype{ 1 }.dom.deg = [ 1 0; 0 0; 2 0; 0 2 ];
% coefficients from the pos.  dom.
KsP.ptype{ 1 }.dom.coef = [ 1, r ^ 2, -1, -1 ];
KsP.ptype{ 1 }.complex_coef = 1; % complex coefficients
KsP.ycomplex = 1 :  nConstr;     % complex constraints
[ y ] = pos3poly( AsP, bsP, csP, KsP ); % call POS3POLY
sigma = sqrt( y ); % distance to unstabilizability
```

## 4.6 Optimization of approximately linear-phase FIR filters

We show in this section the implementation of two BRL type problems using POS3POLY.

### 4.6.1 One-dimensional filters

One-dimensional approximately linear-phase lowpass filters can be designed solving the optimization problem [1, Section 5.1.3]

$$\begin{array}{ll} \min\limits_{H\in\mathbb{R}_{n+}[z]} & E_s \\ \text{s.t.} & |H(\omega)-\mathrm{e}^{-j\tau\omega}|\le\gamma_p, \quad \omega\in[0,\omega_p] \\ & |H(\omega)|\le\gamma_s, \quad \omega\in[\omega_s,\pi] \end{array} \tag{4.49}$$

where $H(z)$ is a causal filter of degree $n$, $E_s$ is the stopband energy and the integer $\tau$ is the desired group delay. Each of the constraints of (4.49) is a BRL. To describe a BRL POS3POLY uses a variable formed by a vector describing the filter and a positive polynomial to describe the polynomial $R(\boldsymbol{q})$ from the right hand side of (2.30).

Although the implementation in POS3POLY would be much simpler in the dual form, we discuss the primal form. The problem (4.49) is equivalent to the POS3POLY problem

$$\begin{array}{ll} \min\limits_{\boldsymbol{h},\varepsilon,\boldsymbol{y},\boldsymbol{h}_0,\boldsymbol{s}_0,\boldsymbol{h}_1,\boldsymbol{s}_1} & \varepsilon \\ \text{s.t.} & \boldsymbol{y}=\boldsymbol{C}^{1/2}\boldsymbol{h} \\ & |H_0(\omega)|^2\le S_0(\omega), \quad \forall\omega\in[0,\omega_p] \\ & H_0(z)=H(z)-z^{-\tau} \\ & S_0(z)=\gamma_p^2 \\ & |H(\omega)|^2\le S_1(\omega), \quad \forall\omega\in[\omega_s,\pi] \\ & H_1(z)=H(z) \\ & S_1(z)=\gamma_s^2 \\ & \|\boldsymbol{y}\|\le\varepsilon \\ & S_0(z)\ge 0, \quad \forall\omega\in[0,\omega_p] \\ & S_1(z)\ge 0, \quad \forall\omega\in[\omega_s,\pi] \end{array} \tag{4.50}$$

where $H_0(z)$ and $H_1(z)$ are univariate causal polynomials and $\boldsymbol{h}_0$, $\boldsymbol{h}_1$ are the vectors of their coefficients, respectively and $S_0(z)$ and $S_1(z)$ are univariate positive polynomials and $\boldsymbol{s}_0$, $\boldsymbol{s}_1$ are the vectors of their coefficients, respectively.

The characteristic system for the primal form of the problem (4.50) is

$$\left[\begin{array}{ccccccc} -\boldsymbol{C}^{1/2} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{array}\right] \left[\begin{array}{c} \boldsymbol{h} \\ \varepsilon \\ \boldsymbol{y} \\ \boldsymbol{h}_0 \\ \boldsymbol{s}_0 \\ \boldsymbol{h}_1 \\ \boldsymbol{s}_1 \end{array}\right] = \left[\begin{array}{c} \boldsymbol{0} \\ \hline \boldsymbol{0} \\ 1 \\ \boldsymbol{0} \\ \hline \boldsymbol{0} \\ \hline \gamma_p^2 \\ \boldsymbol{0} \\ \hline \gamma_s^2 \\ \boldsymbol{0} \end{array}\right] \begin{array}{c} \\ \\ (n+\tau+2) \\ \\ \\ \\ \\ \\ \\ \end{array} . \tag{4.51}$$

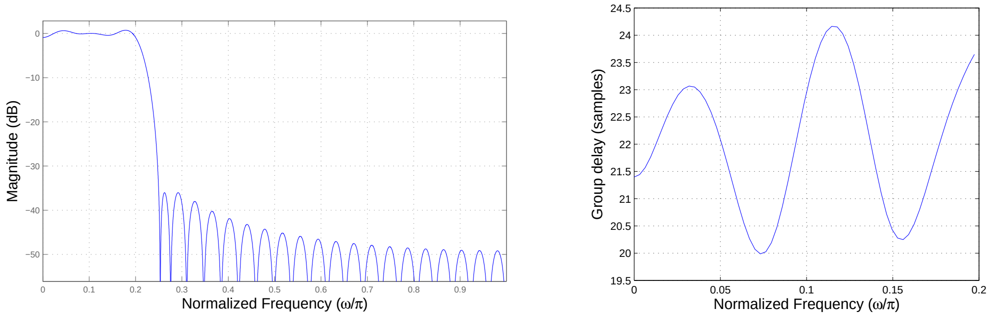
The first block line describes the connection between $\boldsymbol{y}$ and $\boldsymbol{h}$. The second block line describes the connection between the filters $H_0(z)$ and $H(z)$. The third line describes the dependence of the filter $H_1(z)$ on the filter $H(z)$. The last two block lines set the constraints for the polynomials $S_0(z)$ and $S_1(z)$. In the right most part of (4.51) we have emphasized the place for the nonzero coefficient of the "polynomial" $z^{-\tau}$ in the free term.

The program that solves the problem (4.50) is listed in the Table 4.7 and discussed in the next paragraph.

***Comments.*** We consider in line 1, as input arguments, the MATLAB variables `n`, `wp`, `ws`, `gp`, `gs`, `tau` for $n$, $\omega_p$, $\omega_s$, $\gamma_p$, $\gamma_s$, $\tau$, respectively. We initialize the matrix $\boldsymbol{A}$, free term and the objective function in lines 4–5. The line 11–12 set the constraint between $\boldsymbol{y}$ and $\boldsymbol{h}$. Lines 13-15 set the constraints between $H_0(z)$ and $H(z)$, while lines 16–17 set the constraints between $H_1(z)$ and $H(z)$. Lines 18–18 set the constraints for the polynomial $S_0(z)$, while lines 20–21 set the constraints for the polynomial $S_1(z)$. The filter $H(z)$ is set as a free variable in line 22. $\boldsymbol{y}$ and $\varepsilon$ belong to a SOC, as stated in line 23. The lines 24–25 declare the positive polynomials $S_0(z)$ and $S_1(z)$ and the lines 26–27 their positivity intervals, respectively. The degree of these polynomials is `n` and they have scalar coefficients, as set in line 28–29. The lines 30 and 31 describe the size of the matrix coefficients of $H(\omega)$. The POS3POLY library is called in line 32 and the filter is extracted from the solution in line 33. ■

Table 4.7: POS3POLY program for solving the problem (4.50).

```
function [ h ] = approx_lin_phase_fir( n, wp, ws, gp, gs, tau )
% nr of coefs.; nr of constr.; nr of variables; identity matrix
n1 = n + 1; nConstr = 5 * n1; nVar = 1 + n1 * 6; I = speye( n1 );
AsP = sparse( nConstr, nVar ); bsP = sparse( nConstr, 1 );
csP = sparse( 1, nVar ); csP( n1 + 1 ) = 1;
c = ( 1 - ws / pi ) * eye( n1, 1 ); % the vector c
for i = 2 :  n1
    c( i ) = -sin( ( i - 1 ) * ws ) / ( ( i - 1 ) * pi );
end
C = toeplitz( c ); C12 = real( sqrtm( C ) );
AsP( 2 * n1 + 1 :  3 * n1, n1 + 2 :  2 * n1 + 1 ) = I; % y=C12*h
AsP( 2 * n1 + 1 :  3 * n1, 1 :  n1 ) = -C12;
AsP( n1 + 1 :  2 * n1, 1 :  n1 ) = I; % h, h0
AsP( n1 + 1 :  2 * n1, 2 * n1 + 2 :  3 * n1 + 1 ) = -I;
bsP( n1 + tau + 1 ) = 1;
AsP( 2 * n1 + 1 :  3 * n1, 1 :  n1 ) = I; % h, h1
AsP( 2 * n1 + 1 :  3 * n1, 4 * n1 + 2 :  5 * n1 + 1 ) = -I;
AsP( 3 * n1 + 1 :  4 * n1, 3 * n1 + 2 :  4 * n1 + 1 ) = I; % s0
bsP( 3 * n1 + 1 ) = gp ^ 2;
AsP( 4 * n1 + 1 :  5 * n1, 5 * n1 + 2 :  6 * n1 + 1 ) = I; % s1
bsP( 4 * n1 + 1 ) = gs ^ 2;
KsP.f = n1; % the coefs.  of the filter are unrestricted
KsP.q = n1 + 1; % SOC: ||y|| <= eps
KsP.ptype{ 1 }.trigonometric = 1; % the polynomial variables
KsP.ptype{ 2 }.trigonometric = 1;
KsP.ptype{ 1 }.int = [ 0 wp ]; % positivity intervals
KsP.ptype{ 2 }.int = [ ws pi ];
KsP.p{ 1 } = [ n 1 ]; % degree for the polynomials
KsP.p{ 2 } = [ n 1 ]; % and scalar coefficients
KsP.ptype{ 1 }.brl.hsize = 1;
KsP.ptype{ 2 }.brl.hsize = 1;
[ x ] = pos3poly( AsP, bsP, csP, KsP ); % use POS3POLY
h = x( 1 :  n1 ); % coefficients of the filter
```



(a) Frequency response.

(b) Group delay.

Figure 4.3: Frequency response and group delay of approximately linear-phase filter for $n = 50$, $\omega_p = 0.2\pi$, $\omega_s = 0.25\pi$, $\gamma_p = 0.1$, $\gamma_s = 0.0158$, $\tau = 22$.

Figure 4.3a shows a filter designed using problem (4.50) and Figure 4.3b shows its group delay. (To design a filter using problem (4.50) one can use the function `approx_lin_phase_fir.m`.)

### 4.6.2 Two-dimensional filters

In this section we present the minimax optimization of 2-D approximately linear-phase filters. The problem is discussed in [1, Section 5.2.3] and can be cast as

$$
\begin{array}{ll}
\min\limits_{\gamma_s, H} & \gamma_s \\
\text{s.t.} & |H(\boldsymbol{\omega}) - \boldsymbol{G}(\boldsymbol{\omega})| \le \gamma_p, \quad \forall \boldsymbol{\omega} \in \mathcal{D}_p \\
 & |H(\boldsymbol{\omega})| \le \gamma_s, \quad \forall \boldsymbol{\omega} \in \mathcal{D}_s
\end{array}
\tag{4.52}
$$

where $H(\boldsymbol{\omega})$ is a bivariate causal filter and $\boldsymbol{G}(\boldsymbol{\omega}) = \mathrm{e}^{-j\boldsymbol{\tau}\boldsymbol{\omega}}$. The design data are: the delay $\boldsymbol{\tau} \in \mathbb{N}^2$, the passband error bound $\gamma_p$ and the diamond-shaped passband and stopband $\mathcal{D}_p$, $\mathcal{D}_s$ given by (4.32). The stopband error $\gamma_s$ is minimized.

The problem (4.52) is in dual form and the expression $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}$ is

$$
\left[\begin{array}{c}
\boldsymbol{g} \\ \hline
\gamma_p^2 \\
\mathbf{0} \\ \hline
\mathbf{0} \\ \hline
\mathbf{0}
\end{array}\right]
-
\left[\begin{array}{cc}
\mathbf{0} & \boldsymbol{I}_N \\ \hline
\mathbf{0} & \mathbf{0}_{M\times N} \\ \hline
\mathbf{0} & -\boldsymbol{I}_N \\ \hline
\overset{-1}{\mathbf{0}} & \mathbf{0}_{M\times N}
\end{array}\right]
\left[\begin{array}{c}
\gamma_s^2 \\
\boldsymbol{h}
\end{array}\right],
\tag{4.53}
$$

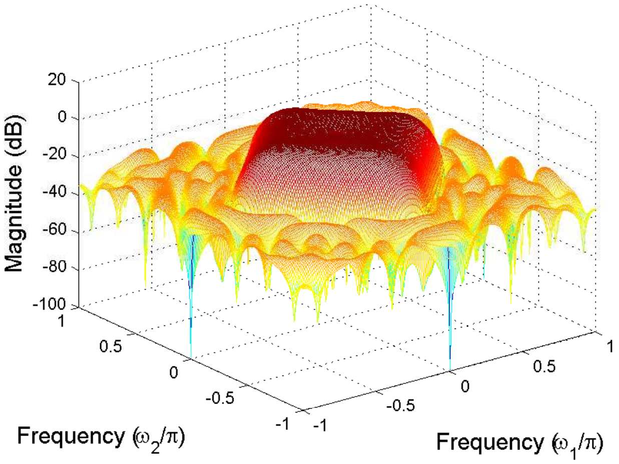


Figure 4.4: Frequency response of approximately linear-phase 2-D FIR filter for $\boldsymbol{n} = (10, 10)$, $\gamma_p = 0.1$, $\boldsymbol{\tau} = (4, 4)$.

where $M$ is computed as in (2.3) and $N = (n_1 + 1)(n_2 + 1)$. The scalar 1 from the right hand side of (4.53) corresponds to $z_1^{-\tau_1} z_2^{-\tau_2}$. The first two block lines describe the first BRL constraint and the last two describe the second BRL constraint: the first and the third block lines describe the filters, while the second and fourth describe the positive polynomial.

To design a filter using the description from (4.53) one can use the command

```
[ h ] = app_lp_fir2d( [10 10], 0.1, [ 4 4 ] )
```

Then, the frequency response of the filter `h` can be plotted using the command

```
freqz2d( mat( h ) );
```

Figure 4.4 illustrates an example for the filter $H(\boldsymbol{z})$.

## 4.7 Matrix filter design

This section presents the minimax optimization of matrix FIR filters [2]. Matrix filters process blocks of data $\boldsymbol{x} \in \mathbb{C}$ using the linear transformation

$$\boldsymbol{y} \quad = \quad \boldsymbol{A}\boldsymbol{x} \tag{4.54}$$

where $\boldsymbol{A}$ can be real or complex; we choose $\boldsymbol{A}$ to be complex and of size $N \times N$.

Denoting $\boldsymbol{\psi}(z) = [1 \; z \; \ldots \; z^{N-1}]^T$, the matrix filter has the form

$$\boldsymbol{H}(z) = \boldsymbol{A}\boldsymbol{\psi}(z^{-1}) = \sum_{k=0}^{N-1} \boldsymbol{a}_k z^{-k} \tag{4.55}$$

where $\boldsymbol{a}_k$, $k = 0 : N-1$, are the columns of the matrix $\boldsymbol{A}$.

To design a bandpass matrix filter with passband $[0, \omega_p]$, stopbands $[\omega_{s1}, \pi]$ and $[-\pi, \omega_{s2}]$ and equal maximum errors in the passband and stopband, we solve the problem

$$\begin{array}{ll} \min & \gamma_s \\ \text{s.t.} & \gamma_p = \gamma_s \\ & \|\boldsymbol{H}(\omega) - \boldsymbol{\psi}(\mathrm{e}^{-j\omega})\| \leq \gamma_p, \quad \forall \omega \in [0, \omega_p] \\ & \|\boldsymbol{H}(\omega)\| \leq \gamma_s, \quad \forall \omega \in [\omega_{s_1}, \pi] \cup [-\pi, \omega_{s_2}] \end{array} \tag{4.56}$$

The POS3POLY expression that characterizes the dual form for the problem (4.56) is

$$\left[\begin{array}{c} \operatorname{vec}(\boldsymbol{I}) \\ \hline \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array}\right] - \left[\begin{array}{cc} \mathbf{0} & \boldsymbol{I} \\ \hline \begin{smallmatrix}-1 \\ \mathbf{0}\end{smallmatrix} & \mathbf{0} \\ \hline \mathbf{0} & \boldsymbol{I} \\ \hline \begin{smallmatrix}-1 \\ \mathbf{0}\end{smallmatrix} & \mathbf{0} \end{array}\right] \left[\begin{array}{c} \gamma_s \\ \boldsymbol{h} \end{array}\right]. \tag{4.57}$$

***Comments.*** The Table 4.8 present the MATLAB program for the problem (4.57). The program has input parameters $N$, $\omega_p$, $\omega_{s_1}$, $\omega_{s_2}$. The first two block lines of `AsP` are set in lines 7–10. Next, we set the constraints for the filter in the BRL in lines 11–13. The problem has $\gamma_s$ and $H$ as free variables, which is set in line 14. We have two univariate trigonometric variables—lines 15–16. For these polynomials we set positivity intervals in lines 17–18. The degree for both of the polynomials is $N-1$ and the coefficients are scalars as set in line 19. The BRL settings are done in the lines 20–23. (Note the field `hsize` for each polynomial.) The polynomials have complex coefficients (line 24) , the constraints are complex and the filter is also complex (line 25). We call the POS3POLY library in line 26 and return the filter in line 27. ■

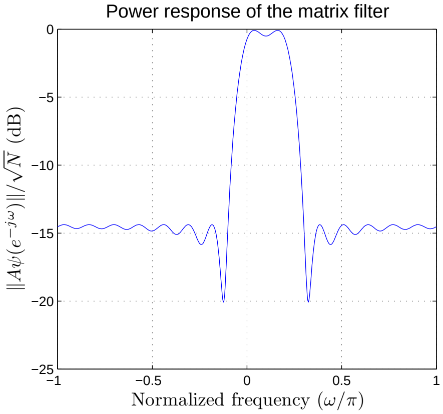


Figure 4.5: Power response of matrix FIR filter for $\omega_p = 0.2\pi$, $\omega_{s_1} = 0.3\pi$, $\omega_{s_2} = -0.1\pi$.

One can design the matrix filter and plot the power response using the command

```
test_p3p_example(27);
```

The power response is shown in Figure 4.5.

Table 4.8: POS3POLY program for solving the problem (4.57).

```
function H = mat_filt( N, wp, ws1, ws2 )
N2 = N ^ 2; nConstr = 2 * N + 2 * N2; nVar = 1 + N2;
AsP = sparse( nConstr, nVar );
bsP = -speye( 1, nVar );
csP = sparse( nConstr, 1 );
I = speye( N ); I2 = speye( N2 );
AsP( 1 :  N2, 2 :  N2 + 1 ) = I2; % H0
csP( 1 :  N2 ) = vec( I );
AsP( N2 + 1, 1 ) = -1; % S0
AsP( N2 + N + 1 :  2 * N2 + N, 2 :  N2 + 1 ) = I2; % H1
AsP( 2 * N2 + N + 1, 1 ) = -1; % S1
KsP.ptype{ 1 }.trigonometric = 1; % the polynomial variables
KsP.ptype{ 2 }.trigonometric = 1;
KsP.ptype{ 1 }.int = [ 0 wp ];    % positivity domains
KsP.ptype{ 2 }.int = [ ws1 pi -pi ws2 ];
KsP.p{ 1 } = [N-1 1]; KsP.p{ 2 } = [N-1 1];
KsP.ptype{ 1 }.brl.hsize = [ 1 N ];
KsP.ptype{ 2 }.brl.hsize = [ 1 N ];
KsP.ptype{ 1 }.complex_coef = 1; KsP.ptype 2 .complex_coef = 1;
KsP.ycomplex = 1 :  nConstr; KsP.xcomplex = 2 :  1 + N2;
[ y ] = pos3poly( AsP, bsP, csP, KsP ); % use POS3POLY
H = y( 2 :  N2 + 1 ); % coefficients of the filter
```

# Chapter 5

# POS3POLY and CVX

## 5.1 Introduction

We discuss here the support of POS3POLY for CVX [6]. 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) polynomial or BRL variables.

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

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

where `p` and `ptype` are structures for *one* polynomial, as presented in section 2.2. `R` is a vector variable, containing the coefficients of the polynomial described by `p` and `ptype`, ordered as described in Section 2.1.

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 $\boldsymbol{H}$ and $R$ from (2.30), respectively. So, the function describes a single vector variable, obtained by the concatenation of the vectors `H` and `R`.

## 5.2 Examples

This section presents examples of problems solved using POS3POLY and CVX. The main tool is the function `sos_pol` for creating sum-of-squares polynomials. All the examples using CVX are in the `examples|cvx` directory.

### 5.2.1 Minimum value of a multivariate trigonometric polynomial

We present here the computation of the minimum value of a multivariate trigonometric polynomial. For the polynomial $R(\boldsymbol{z})$ from (2.1) the problem can be cast as a POS3POLY problem in the form

$$\begin{array}{rcl} \mu^* & = & \max\limits_{\mu} \quad \mu \\ & & \text{s.t.} \quad R(\boldsymbol{z}) - \mu \geq 0, \quad \forall \boldsymbol{z} \in \mathbb{T}^d \end{array} \tag{5.1}$$

The problem is relaxed to

$$\begin{array}{rcl} \mu_1 & = & \max\limits_{\mu} \quad \mu \\ & & \text{s.t.} \quad R(\boldsymbol{z}) - \mu \text{ is sum-of-squares} \end{array} \tag{5.2}$$

The function that solves the problem (5.2) is listed in Table 5.1.

***Comments.*** The input parameters of the function describe the polynomial by the vector `r` of its coefficients and its degree `n`. The variable $\mu$ is denoted by `m` in line 4. We maximize `m` in line 5. The polynomial $R(\boldsymbol{z}) - \mu$ is enforced to be sum-of-squares in line 8. The command `cvx_end` runs CVX and solves the problem. ■

Table 5.1: POS3POLY–CVX program for solving the problem (5.2).

```
function [ m ] = min_poly_value_multi_general_trig_cvx( n, r )

cvx_begin
    variable m;
    maximize m;
    subject to
    % SOS polynomial
    r - m * unitpol( n ) == sos_pol( [ n 1 ] );
cvx_end
```

We consider the bivariate polynomial

$$R(\boldsymbol{z}) = \text{sym}^{-1} + 38 + 18z_1 + 4z_1^2 + z_1^{-2}z_2 + 2z_1^{-1}z_2 + z_2 - 8z_1z_2 - 5z_1^2z_2 \tag{5.3}$$

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

```
r = [ 38 18 4 1 2 1 -8 -5 ]’;
n = [ 2 1 ];
m = min_poly_value_multi_general_trig_cvx( n, r );
```

The minimum is $\mu_1 = 1.8214$. (Note that the order of the coefficients in `r` corresponds to the order chosen in (2.2).)

### 5.2.2 Adjustable linear-phase FIR filters

We discuss now the design of adjustable linear-phase FIR filters [5] with transfer function

$$H(z, p) = \sum_{k=0}^{K} (p - p_0)^K H_k(z), \tag{5.4}$$

where $H_k(z)$, $k = 0 : K$, are FIR filters, $p_0 \in \mathbb{R}$ is a constant and $p \in \mathbb{R}$ is a variable. The implementation uses the Farrow structure shown in Figure 5.1.

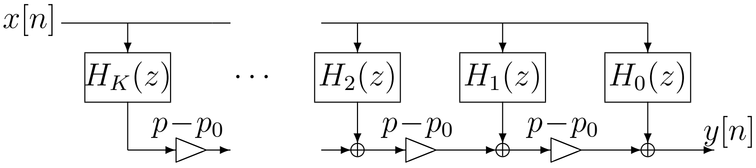


Figure 5.1: Farrow structure for the implementation of adjustable filters.

The filters $H_k(z)$ are zero-phase

$$H_k(z) = \sum_{i=-N}^{N} h_{k,i} z^{-i}, \quad h_{k,i} = h_{k,-i} \tag{5.5}$$

and thus the transfer function (5.4) is a hybrid filter. We denote $t = p - p_0$ and $H(t, z)$ the polynomial (5.4) transformed with this substitution.

The parameter $p$ lies in an interval $[p_\ell, p_u]$ and $p_0$ is a constant that can have any value; we fix $p_0$ to $(p_\ell + p_u)/2$. Given a passband error bound $\gamma_p$, we aim to design an adjustable filter by solving the problem

$$\begin{array}{ll} \min & \gamma_s \\ \text{s.t.} & 1 - \gamma_p \leq H(\mathrm{e}^{j\omega}, p) \leq 1 + \gamma_p, \quad \forall \cos\omega \in [p + \Delta, 1] \\ & -\gamma_s \leq H(\mathrm{e}^{j\omega}, p) \leq \gamma_s, \quad \forall \cos\omega \in [-1, p - \Delta] \end{array} \tag{5.6}$$

where the parameter $\Delta$ determines the width of the transition band.

We transform the problem (5.6) into

$$\begin{array}{ll} \min & \gamma_s \\ \text{s.t.} & H(z,t) + \gamma_p - 1 \geq 0, \quad \forall (t,z) \in \mathcal{D}_p \\ & \gamma_p + 1 - H(z,t) \geq 0, \quad \forall (t,z) \in \mathcal{D}_p \\ & H(z,t) + \gamma_s \geq 0, \quad \forall (t,z) \in \mathcal{D}_s \\ & \gamma_s - H(z,t) \geq 0, \quad \forall (t,z) \in \mathcal{D}_s \end{array} \tag{5.7}$$

with domains

$$\begin{array}{l} \mathcal{D}_p = \{(z,t) \in \mathbb{R} \times \mathbb{T} \mid D_{p\ell}(z,t) \geq 0, \ell = 1:2\} \\ \mathcal{D}_s = \{(z,t) \in \mathbb{R} \times \mathbb{T} \mid D_{s\ell}(z,t) \geq 0, \ell = 1:2\}. \end{array} \tag{5.8}$$

Considering that $\cos\omega = (z + z^{-1})/2$, the polynomials from (5.8) are defined by

$$\begin{array}{l} D_{p1}(z,t) = \frac{1}{2}(z + z^{-1}) - t - p_0 - \Delta \\ D_{s1}(z,t) = -\frac{1}{2}(z + z^{-1}) + t + p_0 - \Delta \\ D_{p2}(z,t) = D_{s2}(z,t) = (t + p_0 - p_\ell)(p_u - t - p_0) \end{array} \tag{5.9}$$

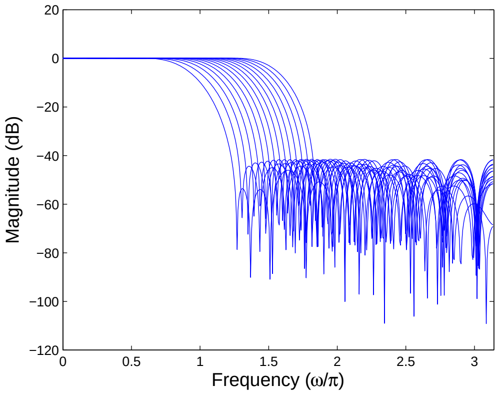


Figure 5.2: Adjustable filters for $N = 13$, $K = 6$, $\gamma_p = 0.01$, $\Delta = 0.25$, $p_\ell = 0$, $p_u = 0.56$.

The expression $\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}$ from (1.3) which describes the POS3POLY problem (5.7) is

$$\left[\begin{array}{c} 1-\gamma_p \\ \mathbf{0} \\ \hline 1+\gamma_p \\ \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array}\right] - \left[\begin{array}{cc} \mathbf{0} & -\boldsymbol{I} \\ \hline \mathbf{0} & \boldsymbol{I} \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & -\boldsymbol{I} \\ \hline \begin{smallmatrix}-1\\ \mathbf{0}\end{smallmatrix} & \boldsymbol{I} \end{array}\right] \left[\begin{array}{c} \gamma_s \\ \boldsymbol{h} \end{array}\right]. \tag{5.10}$$

The program solving the problem (5.7) is in Table 5.2.

***Comments.*** Each polynomial has one trigonometric variable and one real variable, as set using the `trigonometric` and `real` fields. The settings for the polynomials are done in lines 9–40. The CVX mode starts in line 42. We declare `gs`, the filter `H` and the four polynomial variables in line 44; note that each domain (5.8) is defined by two polynomials, each having three nonzero coefficients (symmetry taken into account). The lines 46–49 are the four constraints of the problem, the block lines from (5.10). ■

The problem listed in Table 5.2 can be executed using the command

```
>> test_p3p_example(35);
```

Figure 5.2 shows the adjustable filters obtained.

Table 5.2: POS3POLY–CVX program for solving the problem (5.7).

```
function H = adj_fir_cvx( n, gp, D, pl, pu, p0 )

% nr. of coefs. in halfspace for
% a hybrid pol. with one trig. var.
N = lenpol( n, 1, 1, 1 );

Ipol = eye( N, 1 );

% 1st SOS polynomial S0
ptype{ 1 }.trigonometric = 1;
ptype{ 1 }.real = 1;
p{ 1 } = [ n 1 ];
ptype{ 1 }.dom.nc = [ 3 3 ];
ptype{ 1 }.dom.deg = [ 0 0; 1 0; 0 1; 0 0; 0 1; 0 2 ];
ptype{ 1 }.dom.coef = [ -D+p0 -0.5 1 (pu-p0)*(p0-pl) ...
                        (pu+pl-2*p0) -1 ];
% 2nd SOS polynomial S1
ptype{ 2 }.trigonometric = 1;
ptype{ 2 }.real = 1;
p{ 2 } = [ n 1 ];
ptype{ 2 }.dom.nc = [ 3 3 ];
ptype{ 2 }.dom.deg = [ 0 0; 1 0; 0 1; 0 0; 0 1; 0 2 ];
ptype{ 2 }.dom.coef = [ -D+p0 -0.5 1 (pu-p0)*(p0-pl) ...
                        (pu+pl-2*p0) -1 ];
% 3rd SOS polynomial S2
ptype{ 3 }.trigonometric = 1;
ptype{ 3 }.real = 1;
p{ 3 } = [ n 1 ];
ptype{ 3 }.dom.nc = [ 3 3 ];
ptype{ 3 }.dom.deg = [ 0 0; 1 0; 0 1; 0 0; 0 1; 0 2 ];
ptype{ 3 }.dom.coef = [ -D-p0 0.5 -1 (pu-p0)*(p0-pl) ...
                        (pu+pl-2*p0) -1 ];
% 4th SOS polynomial S3
ptype{ 4 }.trigonometric = 1;
```

Continued on next page...

Table 5.2 – Continued

```
ptype{ 4 }.real = 1;
p{ 4 } = [ n 1 ];
ptype{ 4 }.dom.nc = [ 3 3 ];
ptype{ 4 }.dom.deg = [ 0 0; 1 0; 0 1; 0 0; 0 1; 0 2 ];
ptype{ 4 }.dom.coef = [ -D-p0 0.5 -1 (pu-p0)*(p0-pl) ...
                        (pu+pl-2*p0) -1 ];

cvx_begin
    cvx_solver sedumi;
    variables gs H( N );
    minimize gs;
    H + gs * Ipol == sos_pol( p{ 1 }, ptype{ 1 } );
    gs * Ipol - H == sos_pol( p{ 2 }, ptype{ 2 } );
    H - ( 1 - gp ) * Ipol == sos_pol( p{ 3 }, ptype{ 3 } );
    ( 1 + gp ) * Ipol - H == sos_pol( p{ 4 }, ptype{ 4 } );
cvx_end
```

### 5.2.3 Design of 2-D MIMO filters

We discuss here the design of 2-D MIMO FIR filters [2]. Given a desired response $\boldsymbol{D}(\boldsymbol{z})$, a passband error bound $\gamma_p$, a passband edge $\omega_p$ and a stopband edge $\omega_s$ (we presume that the passband and the stopband are delimited by squares), find the optimal filter

$$\begin{array}{ll} \min & \gamma_s \\ \text{s.t.} & \sigma_{\max}(\boldsymbol{H}(\mathrm{e}^{j\boldsymbol{\omega}}) - \boldsymbol{D}(\mathrm{e}^{j\boldsymbol{\omega}})) \le \gamma_p, \quad \forall |\omega_i| \le \omega_p, i = 1:2 \\ & \sigma_{\max}(\boldsymbol{H}(\mathrm{e}^{j\boldsymbol{\omega}})) \le \gamma_s, \quad \exists i \in 1:2, |\omega_i| \ge \omega_s \end{array} \tag{5.11}$$

where $\sigma_{\max}(\cdot)$ is the maximum singular value function and $\boldsymbol{H}(\boldsymbol{z})$ is a $2 \times 2$ causal polynomial matrix of degree $\boldsymbol{n} = (n_1, n_2)$.

The problem is equivalent to

$$\begin{array}{ll} \min & \gamma_s \\ \text{s.t.} & \sigma_{\max}(\boldsymbol{H}(\mathrm{e}^{j\boldsymbol{\omega}}) - \boldsymbol{D}(\mathrm{e}^{j\boldsymbol{\omega}})) \le \gamma_p, \quad \forall \boldsymbol{z} \in \mathcal{D}_p \\ & \sigma_{\max}(\boldsymbol{H}(\mathrm{e}^{j\boldsymbol{\omega}})) \le \gamma_s, \quad \forall \boldsymbol{z} \in \mathcal{D}_{s_1} \cup \mathcal{D}_{s_2} \end{array} \tag{5.12}$$

where

$$\begin{array}{rcl} \mathcal{D}_p(\boldsymbol{z}) & = & \{\boldsymbol{z} \in \mathbb{T}^2 \mid D_\ell(\boldsymbol{z}) = z_\ell + z_\ell^{-1} - 2\cos\omega_p \geq 0, \ell = 1:2\} \\ \mathcal{D}_{s_1}(\boldsymbol{z}) & = & \{\boldsymbol{z} \in \mathbb{T}^2 \mid 2\cos\omega_s - z_1 - z_1^{-1} \geq 0\} \\ \mathcal{D}_{s_2}(\boldsymbol{z}) & = & \{\boldsymbol{z} \in \mathbb{T}^2 \mid 2\cos\omega_s - z_2 - z_2^{-1} \geq 0\} \end{array} \tag{5.13}$$

The problem (5.12) has two BRL constraints.

The characteristic expression for the dual form of the problem (5.12) is

$$\left[\begin{array}{c} \vdots \\ \mathrm{vec}(\boldsymbol{D}_{i,j}) \\ \vdots \\ \hline \gamma_p^2 \\ \mathbf{0} \\ \hline \mathbf{0} \\ \hline \mathbf{0} \end{array}\right] - \left[\begin{array}{cc} \mathbf{0} & \boldsymbol{I}_N \\ \hline \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & -\boldsymbol{I}_N \\ \hline \begin{smallmatrix} -1 \\ \mathbf{0} \end{smallmatrix} & -\boldsymbol{I}_N \end{array}\right] \left[\begin{array}{c} \gamma_s \\ \boldsymbol{h} \end{array}\right] \tag{5.14}$$

where $i = 0 : n_1$, $j = 0 : n_2$ and $N = 4(n_1 + 1)(n_2 + 1)$.

The MATLAB program which solves (5.12) is listed in Table 5.3 and commented below.

***Comments.*** We denote by `n` the degree of the polynomial $\boldsymbol{H}(\boldsymbol{z})$, by `gp` the passband error $\gamma_p$ and by `K` the number of inputs and outputs of the system $\boldsymbol{H}(\boldsymbol{z})$ which in this case is `[2 2]`. `degD` holds in its lines the nonzero degrees of the polynomial $\boldsymbol{D}(\boldsymbol{z})$ and in `coefD` the nonzero coefficients. `wp` and `ws` are the passband and stopband edges, respectively. The number of *scalar* variables for a filter is set in line 6. We initialize the matrix `D` for the polynomial $\boldsymbol{D}(\boldsymbol{z})$ in line 8. Next, we set two bivariate polynomial variables in lines 11–12. In lines 13–22 we set the positivity domains. For each polynomial we set the degrees for the positivity polynomials, their coefficients and the number of coefficients for each polynomials from the positivity domain. For the second polynomial we also specify the number of polynomials for the domains from the union. In lines 23–26 we activate the BRL for each polynomial by setting the field `hsize`. The degree of the polynomials and the size of the coefficients are set in line 29. The vector `D` which denotes the polynomial $\boldsymbol{D}(\boldsymbol{z})$ is constructed in lines 30–34. We start writing for CVX in line 36. We declare the stopband error bound `gs` as variable, the polynomial `H` and the BRL variables `HR1`, `HR2`,

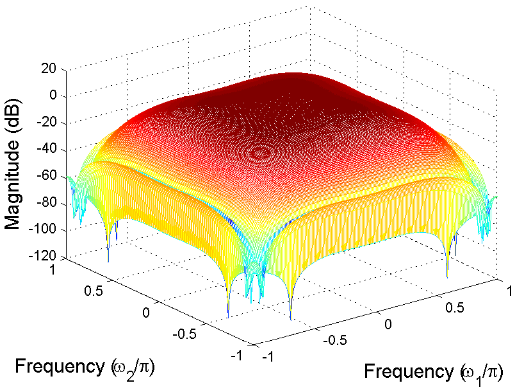


Figure 5.3: Frequency response of $H_{1,1}(\boldsymbol{z})$ for $\boldsymbol{n} = (4,4)$, $\gamma_p = 0.1$, $\omega_p = 0.4\pi$, $\omega_s = 0.9\pi$, $\boldsymbol{D}(\boldsymbol{z}) = z_1^{-2} z_2^{-2} \boldsymbol{I}_2$.

`HR3`. We use the `sos_pol` function to create the variables for the polynomials in lines 41–42. The problem minimizes `gs`. The lines 44–45 represent the second and the fourth block constraints from (5.14). The lines 46–47 represent the constraints applied on the filter $\boldsymbol{H}(z)$ from the BRL. The variables returned for the BRLs by the `sos_pol` function must be equal to bounded systems. The lines represent the first and the third block lines from (5.14). The command `cvx_end` solves the problem (5.12). ■

Figure 5.3 represents the transfer function from the first input to the first output. To plot the FIR filter after we have solved the problem, we have used the command

```
freqz2d( mat( H( 1 :  4 :  prod( n + 1 ) * prod( K ) ) ) );
```

Table 5.3: POS3POLY–CVX program for solving the problem (5.12).

```
function H = mimo_fir2d_cvx( n, gp, K, degD, coefD, wp, ws )

% number of n-tuples;
M = lenpol( n );
% number of scalar filter coefs.
N = lencpol( n );

D = sparse( N, 1 );

% two polynomial variables
ptype{ 1 }.trigonometric = 2;
ptype{ 2 }.trigonometric = 2;
% positivity domains
% Dp
ptype{ 1 }.dom.nc = [ 2 2 ];
ptype{ 1 }.dom.deg = [ 0 0; 1 0; 0 0; 0 1 ];
ptype{ 1 }.dom.coef = [ -2 * cos( wp ) 1 -2 * cos( wp ) 1 ];
% Ds1 U Ds2
ptype{ 2 }.dom.nunion = [ 1 1 ];
ptype{ 2 }.dom.nc = [ 2 2 ];
ptype{ 2 }.dom.deg = [ 0 0; 1 0; 0 0; 0 1 ];
ptype{ 2 }.dom.coef = [ 2 * cos( ws ) -1 2 * cos( ws ) -1 ];
% 1st BRL
ptype{ 1 }.brl.hsize = [ K1 K2 ];
% 2nd BRL
ptype{ 2 }.brl.hsize = [ K1 K2 ];

% degrees of polynomials 'n', and coefficients are scalars
p{ 1 } = [ n 1 ]; p{ 2 } = [ n 1 ];
% introduce D(z)
for i = 1 :  size( degD, 1 )
    nrdeg = degD( i, 2 ) * ( n( 1 ) + 1 ) + degD( i, 1 ) + 1;
    D( (nrdeg-1)*prod(K)+1 :  nrdeg*prod(K) ) = vec(coefD{i});
end
```

Continued on next page…

Table 5.3 – Continued

```

cvx_begin
    variable gs;
    variable H( N );
    variable HR1( M + N );
    variable HR2( M + N );
    HR1 == sos_pol( p{ 1 }, ptype{ 1 } );
    HR2 == sos_pol( p{ 2 }, ptype{ 2 } );
    minimize gs;
    gp ^ 2 * unitpol( n ) == HR1( N + 1 :  end );
    gs     * unitpol( n ) == HR2( N + 1 :  end );
    HR1( 1 :  MK1K2 ) == H - D;
    HR2( 1 :  MK1K2 ) == H;
cvx_end
```

# Chapter 6

# POS3POLY and SDPT3

POS3POLY can also work with SDPT3 [11]. In order to use SDPT3 one must use the POS3POLY command `sedumi_data` which takes the same input parameters as the `pos3poly` command:

```
[ As, bs, cs, Ks, D, IB, iE, b0, iB, Bl, Blc, f, f0, nU ] = sedumi_data( AsP,
bsP, csP, KsP );
```

Next, we use three SDPT3 calls. First, we convert the SeDuMi data to SDPT3 data with

```
[ blk, AA, CC, bb, perm ] = read_sedumi( As, bs, cs ,Ks );
```

Then, we solve the optimization problem using the SDPT3 library

```
[ obj, X, y, Z ] = sdpt3( blk, AA, CC, bb );
```

One must convert the SDPT3 solution to the SeDuMi solution using

```
[ xx, yy, zz ] = SDPT3soln_SEDUMIsoln( blk, X, y, Z, perm );
```

Finally, the POS3POLY solution is extracted using

```
[ xy ] = sdm2p3p( xx, D, IB, iE, b0, iB, Bl, Blc, f, f0, nU );
```

# Appendix A

# The parameterizations

We present in this chapter the parameterizations used for the sum-of-squares polynomials.

## A.1 Univariate polynomials

### A.1.1 Trigonometric polynomials

#### Scalar polynomials

Considering $d = 1$ in (2.1) we obtain

$$R(z) = \sum_{k=-n}^{n} r_k z^{-k}, \quad r_{-k} = r_k^*, \tag{A.1}$$

where $k, n \in \mathbb{Z}$, $z \in \mathbb{T}$ and $r_k \in \mathbb{C}$.

Theorem A.1.1. *The polynomial $R(z)$ is positive if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that*

$$r_k = \mathbf{Tr}[\boldsymbol{\Theta}_k \cdot \boldsymbol{Q}], \quad k = 0 : n, \tag{A.2}$$

*where $\boldsymbol{\Theta}_k$ is the elementary Toeplitz matrix with ones on the $k$-th diagonal and zeros elsewhere and $N = n + 1$. The matrix $\boldsymbol{Q}$ is called a* Gram *matrix associated with the polynomial (A.1). (**Tr** $\boldsymbol{M}$ stands for the trace operator applied on the matrix $\boldsymbol{M}$.)* ■

### Matrix polynomials

We take $d = 1$ in (2.15) and obtain

$$\boldsymbol{R}(z) = \sum_{k=-n}^{n} \boldsymbol{R}_k z^{-k}, \quad \boldsymbol{R}_{-k} = \boldsymbol{R}_k^H, \tag{A.3}$$

where $k, n \in \mathbb{Z}$, $z \in \mathbb{T}$ and $\boldsymbol{R}_k \in \mathbb{C}^{\kappa \times \kappa}$.

THEOREM A.1.2. *The polynomial $\boldsymbol{R}(z)$ is positive if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that*

$$\boldsymbol{R}_k(i,j) = \mathbf{Tr}[\boldsymbol{\Theta}_k \otimes \boldsymbol{E}_{j,i} \cdot \boldsymbol{Q}], \quad k = 0 : n, \ i, j = 1 : \kappa, \tag{A.4}$$

*where $\boldsymbol{E}_{j,i}$ is the matrix with one in the $(j, i)$ position and zeros elsewhere and $N = (n + 1)\kappa$.* ■

## A.1.2 Real polynomials

### Scalar polynomials

Taking $d = 1$ in (2.7) leads to

$$P(t) = \sum_{k=0}^{n} p_k t^k, \tag{A.5}$$

where $k \in \mathbb{N}$, $t \in \mathbb{R}$, $p_k \in \mathbb{R}$.

THEOREM A.1.3. *The polynomial $P(t)$ is positive if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that*

$$p_k = \mathbf{Tr}[\boldsymbol{\Upsilon}_k \cdot \boldsymbol{Q}], \quad k = 0 : n, \tag{A.6}$$

*where $\boldsymbol{\Upsilon}_k$ is the elementary Hankel matrix with ones on the $k$-th antidiagonal and zeros elsewhere and $N = n/2 + 1$.* ■

### Matrix polynomials

Let $d = 1$ in (2.18) which leads to

$$\boldsymbol{P}(t) = \sum_{k=0}^{n} \boldsymbol{P}_k t^k, \quad \boldsymbol{P}_k = \boldsymbol{P}_k^H, \tag{A.7}$$

where $k \in \mathbb{N}$, $t \in \mathbb{R}$, $\boldsymbol{P}_k \in \mathbb{C}^{\kappa \times \kappa}$.

THEOREM A.1.4. *The polynomial $\boldsymbol{P}(t)$ is positive if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that*

$$\boldsymbol{P}_k(i,j) = \mathbf{Tr}[\boldsymbol{\Upsilon}_k \otimes \boldsymbol{E}_{j,i} \cdot \boldsymbol{Q}], \quad k = 0 : n,\ i,j = 1 : \kappa, \tag{A.8}$$

*where $\boldsymbol{E}_{j,i}$ is the matrix with one in the $(j,i)$ position and zeros elsewhere and $N = (n/2+1)\kappa$.* ■

# A.2 Multivariate polynomials

## A.2.1 Trigonometric polynomials

### Scalar polynomials

The polynomial from (2.1) is sum-of-squares if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that

$$r_{\boldsymbol{k}} = \mathbf{Tr}[\boldsymbol{\Theta}_{k_d} \otimes \cdots \otimes \boldsymbol{\Theta}_{k_1} \cdot \boldsymbol{Q}], \quad \boldsymbol{k} \in \mathcal{H}_d, \tag{A.9}$$

where $N = \prod_{i=1}^{d}(n_i+1)$. ■

### Matrix polynomials

The polynomial from (2.15) is sum-of-squares if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that

$$\boldsymbol{R}_{\boldsymbol{k}}(i,j) = \mathbf{Tr}[\boldsymbol{\Theta}_{k_d} \otimes \cdots \otimes \boldsymbol{\Theta}_{k_1} \otimes \boldsymbol{E}_{j,i} \cdot \boldsymbol{Q}], \quad \boldsymbol{k} \in \mathcal{H}_d,\ i,j = 1 : \kappa, \tag{A.10}$$

where $N = (\prod_{i=1}^{d}(n_i+1))\kappa$. ■

## A.2.2 Real polynomials

### Scalar polynomials

The polynomial from (2.7) is sum-of-squares if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ such that

$$p_{\boldsymbol{k}} = \mathbf{Tr}[\boldsymbol{\Upsilon}_{k_d} \otimes \cdots \otimes \boldsymbol{\Upsilon}_{k_1} \cdot \boldsymbol{Q}], \quad \boldsymbol{k} = \boldsymbol{0} : \boldsymbol{n} \tag{A.11}$$

where $N = \prod_{i=1}^{d}(n_i/2+1)$. ■

### Matrix polynomials

The polynomial from (2.18) is sum-of-squares if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N\times N}$ such that

$$\boldsymbol{P}_{\boldsymbol{k}}(i,j) = \mathbf{Tr}[\boldsymbol{\Upsilon}_{k_d} \otimes \cdots \otimes \boldsymbol{\Upsilon}_{k_1} \otimes \boldsymbol{E}_{j,i} \cdot \boldsymbol{Q}], \quad \boldsymbol{k} = \mathbf{0} : \boldsymbol{n},\ i,j = 1 : \kappa, \tag{A.12}$$

where $N = (\prod_{i=1}^{d}(n_i/2+1))\kappa$. ■

## A.2.3 Hybrid polynomials

### Scalar polynomials

The polynomial from (2.10) is sum-of-squares if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N\times N}$ such that

$$h_{\boldsymbol{k}} = \mathbf{Tr}[\boldsymbol{\Upsilon}_{k_{\ell+m}} \otimes \cdots \otimes \boldsymbol{\Upsilon}_{k_{\ell+1}} \otimes \boldsymbol{\Theta}_{k_\ell} \otimes \cdot s \otimes \boldsymbol{\Theta}_{k_1} \cdot \boldsymbol{Q}], \tag{A.13}$$

where $(k_1,\ldots,k_\ell) \in \mathcal{H}_\ell$, $(h_{\ell+1},\ldots,h_{\ell+m}) = \mathbf{0} : (n_{\ell+1},\ldots,n_{\ell+m})$, $N = 0.5(1+\prod_{i=1}^{\ell}(2n_i+1))\prod_{i=\ell+1}^{\ell+m}(n_i+1)$. ■

### Matrix polynomials

The polynomial from (2.21) is sum-of-squares if and only if there exists a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{C}^{N\times N}$ such that

$$\boldsymbol{H}_{\boldsymbol{k}}(i,j) = \mathbf{Tr}[\boldsymbol{\Upsilon}_{k_{\ell+m}} \otimes \cdots \otimes \boldsymbol{\Upsilon}_{k_{\ell+1}} \otimes \boldsymbol{\Theta}_{k_\ell} \otimes \cdots \otimes \boldsymbol{\Theta}_{k_1} \otimes \boldsymbol{E}_{i,j} \cdot \boldsymbol{Q}], \quad i,j = 1 : \kappa \tag{A.14}$$

where $(k_1,\ldots,k_\ell) \in \mathcal{H}_\ell$, $(h_{\ell+1},\ldots,h_{\ell+m}) = \mathbf{0} : (n_{\ell+1},\ldots,n_{\ell+m})$, $N = 0.5(1+\prod_{i=1}^{\ell}(2n_i+1))\prod_{i=\ell+1}^{\ell+m}(n_i+1)\kappa$. ■

# A.3 Positivity on domains

We consider here the case of positivity on domains for trigonometric polynomials. For real and hybrid polynomials the formulation is similar.

Let us consider a frequency domain

$$\mathcal{D} = \{\boldsymbol{\omega} \in [-\pi,\pi]^d \mid \mathcal{D}_\ell(\boldsymbol{\omega}) \geq 0,\ \ell = 1 : L\}. \tag{A.15}$$

Theorem A.3.1. *The polynomial $R \in \mathbb{C}_n[\boldsymbol{z}]$ is sum-of-squares on $\mathcal{D}$ if and only if there exists sum-of-squares polynomials $S_\ell$, $\ell = 0 : L$, such that*

$$R(\boldsymbol{z}) = S_0(\boldsymbol{z}) + \sum_{\ell=1}^{L} \mathcal{D}_\ell(\boldsymbol{z}) S_\ell(\boldsymbol{z}). \tag{A.16}$$

*The relation (A.16) is equivalent to*

$$r_{\boldsymbol{k}} = \mathbf{Tr}[\boldsymbol{\Theta}_{k_d} \otimes \ldots \otimes \boldsymbol{\Theta}_{k_1} \cdot \boldsymbol{Q}_0 + \sum_{\ell=1}^{L} \boldsymbol{\Psi}_{\ell k} \cdot \boldsymbol{Q}_\ell] \tag{A.17}$$

*where*

$$\boldsymbol{\Psi}_{\ell \boldsymbol{k}} = \sum_{\boldsymbol{i}+\boldsymbol{l}=\boldsymbol{k}} (d_\ell)_{\boldsymbol{i}} \boldsymbol{\Theta}_{\boldsymbol{l}}, \tag{A.18}$$

*with $\boldsymbol{\Theta}_{\boldsymbol{l}} = \boldsymbol{\Theta}_{l_d} \otimes \ldots \otimes \boldsymbol{\Theta}_{l_1}$. We denote $(d_\ell)_{\boldsymbol{i}}$ the coefficients of the polynomial $\mathcal{D}_\ell(\boldsymbol{z})$.* ■

## A.4 Bounded Real Lemma

We consider in this sections the parameterizations for the BRL in the case of trigonometric polynomials. The cases of real and hybrid polynomials are similar.

### A.4.1 Scalar polynomial

Theorem A.4.1. *Let $H(\boldsymbol{z})$ and $A(\boldsymbol{z})$ be two positive orthant polynomials and $\mathcal{D}$ a frequency domain defined as in (A.15). Denote*

$$R(\boldsymbol{z}) = A(\boldsymbol{z}) A^*(\boldsymbol{z}^{-1}). \tag{A.19}$$

*The inequality*

$$|H(\boldsymbol{\omega})| \le |A(\boldsymbol{\omega})|, \quad \forall \boldsymbol{\omega} \in \mathcal{D} \tag{A.20}$$

*is satisfied, if and only if there exists matrices $\boldsymbol{Q}_\ell \succeq \boldsymbol{0}$, $\ell = 0 : L$, such that the relations (A.17) and*

$$\begin{bmatrix} \boldsymbol{Q}_0 & \boldsymbol{h} \\ \boldsymbol{h}^H & 1 \end{bmatrix} \succeq \boldsymbol{0} \tag{A.21}$$

*hold, where $\boldsymbol{h}$ is the vector of coefficients of the filter $H(\boldsymbol{z})$.* ■

## A.4.2 Matrix polynomial

THEOREM A.4.2. *Let* $\boldsymbol{H}(\boldsymbol{z})$ *and* $A(\boldsymbol{z})$ *be two matrix positive orthant polynomials,* $\boldsymbol{H}(\boldsymbol{z})$ *with matrix coefficients of size* $\kappa_1 \times \kappa_2$ *and* $A(\boldsymbol{z})$ *with scalar coefficients. We define* $R(\boldsymbol{z})$ *as in (A.19). The inequality*

$$\|\boldsymbol{H}(\boldsymbol{\omega})\| \leq |A(\boldsymbol{\omega})|, \quad \boldsymbol{\omega} \in \mathcal{D}, \tag{A.22}$$

*holds true if and only if the matrices* $\boldsymbol{Q}_\ell$, $\ell = 0 : L$ *exist such that*

$$r_k \delta_{i-j} = \mathbf{Tr}[\boldsymbol{\Theta}_{k_d} \otimes \ldots \otimes \boldsymbol{\Theta}_{k_1} \otimes \boldsymbol{E}_{j,i} \cdot \boldsymbol{Q}_0 + \sum_{\ell=1}^{L} \boldsymbol{\Psi}_{\ell k} \otimes \boldsymbol{E}_{j,i} \cdot \boldsymbol{Q}_\ell], \quad i, j = 1 : \kappa_1, \tag{A.23}$$

*and*

$$\begin{bmatrix} \boldsymbol{Q}_0 & \overline{\boldsymbol{H}} \\ \overline{\boldsymbol{H}}^H & \boldsymbol{I}_{\kappa_2} \end{bmatrix} \succeq \mathbf{0}, \tag{A.24}$$

*where* $\overline{\boldsymbol{H}}$ *is a matrix of size* $N\kappa_1 \times \kappa_2$, *with* $N = \prod_{i=1}^{d}(n_i + 1)$, *obtained by stacking the coefficients of the polynomial* $\boldsymbol{H}(\boldsymbol{z})$ *and* $\delta_\ell$ *is the Dirac function.* ■

EXAMPLE A.4.1. For a 2-D polynomial with $\boldsymbol{n} = (2, 1)$ we have

$$\overline{\boldsymbol{H}} = [\ \boldsymbol{H}_{0,0}^T\ \boldsymbol{H}_{1,0}^T\ \boldsymbol{H}_{2,0}^T\ \boldsymbol{H}_{0,1}^T\ \boldsymbol{H}_{1,1}^T\ \boldsymbol{H}_{2,1}^T\ ]^T. \tag{A.25}$$

■

# Appendix B

# List of functions

We list here some of the functions of the POS3POLY library.

`cf.m` conditional function

`coefpos.m` get position of coefficient

`degmon.m` compute degrees of the monomials of a polynomial

`eqm.m` test matrix equality

`findvm.m` find vector in matrix

`geqm.m` test matrix inequality

`iseven.m` test for even number

`isinhalf.m` check if degree is in the right halfspace

`iszero.m` test if matrix is zero

`lenpol.m` compute number of scalar coefficients for polynomials

`lencpol.m` compute number of scalar coefficients for causal polynomials

`leqm.m` test matrix inequality

`mat.m` matrix from vector

`mextend.m` matrix extend

`polconv2d.m` 2-D causal polynomial convolution

`pos3poly.m` call the POS3POLY library

`prod_poly.m` product of two polynomials

`sedumi_data.m` retrieve the data passed to SeDuMi

`sdm2p3p.m` convert SDPT3 solution to POS3POLY solution

`sos_pol.m` create a sum-of-squares polynomial for CVX

`spcausal2fvec.m` create a coefficient vector from a sparse description of a causal polynomial

`sppol2fvec.m` create a coefficient vector from a sparse description of a polynomial

`sppols2fvec.m` create a coefficient vector from several sparse descriptions of polynomials

`sum_poly.m` sum of two polynomials

`test_p3p_example.m` run examples of the POS3POLY library

`unitpol.m` unit polynomial

`unitcpol.m` unit causal polynomial

`vec.m` vectorize a matrix

`vecs.m` vectorize a Hermitian matrix

`vecs2.m` vectorize a Hermitian matrix, for CVX variables

`zeropol.m` zero polynomial

`zerocpol.m` zero causal polynomial

# Bibliography

[1] B. Dumitrescu. *Positive Trigonometric Polynomials and Signal Processing Applications*. Springer-Verlag, 2007.

[2] 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, August 2009.

[3] B. Dumitrescu, B.C. Şicleru, and R. Ştefan. Computing the controllability radius: a semi-definite programming approach. *IET Control Theory & Applications*, 3(6):654–660, 2009.

[4] B. Dumitrescu, B.C. Şicleru, and R. Ştefan. Delay-dependent stability analysis of neutral systems using positive polynomials optimization. In *Proc. Control Applications of Optimization (CAO)*, Jyväskylä, Finland, May 2009.

[5] 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.

[6] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. http://cvxr.com/cvx, May 2010.

[7] 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.

[8] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[9] S. Prajna and A. Papachristodoulou. SOSTOOLS: Sum of squares optimization toolbox for Matlab, 2004.

[10] J.F. Sturm. Using SeDuMi, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.

[11] 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.