Manual of "The Robust LMI Parser" – Version 3.0

Cristiano M. Agulhari*, Alexandre Felipe, Ricardo C. L. F. Oliveira, and Pedro L. D. Peres †

January 30, 2018

Abstract

The ROLMIP (Robust LMI Parser), built to work under MATLAB jointly with YALMIP, is a toolbox aimed to ease the programming of parameter-dependent Linear Matrix Inequality (LMI) conditions with parameters lying inside known intervals (or on unit simplexes). Through simple commands, the user is able to define matrix polynomials, as well as to described the desired LMIs in a easy way, considerably reducing the programming time. The main commands and definitions are thoroughly explained in this manual.

1 Introduction

In the last decades, problems formulated in terms of Linear Matrix Inequality (LMI) conditions and solved by Semidefinite Programming (SPD) techniques became more and more common in several fields related to engineering and applied mathematics. Specifically in control theory, the growing usage of such important tools has led to important results on the analysis of systems stability, synthesis of stabilizing robust controllers for uncertain systems and synthesis of optimal control models, just to name a few problems [3].

Accompanying the growth of the usage of LMI optimization, a large number of solvers based on interior point methods were developed, as well as interfaces for parsing the LMIs, most of them free and easily accessible. Thanks to such remarkable advance in the computational tools to define, manipulate and solve LMIs, in many cases one can say that if a problem can be cast as a set of LMIs, then it can be considered as solved [3]. Unfortunately, this is not completely true for large scale systems, since LMI solvers are limited to a few thousands of variables and LMI rows, but progresses are being made.

Usually, LMIs are solved in two steps: first, an interface for parsing the conditions is used, for example the YALMIP [9] or the LMI Control Toolbox from MATLAB [6]; then, an LMI solver is applied to find a solution (if any), for example SeDuMi [20], SDPT3 [21] or MOSEK [1]. Some auxiliary toolboxes may also be used in addition to the parser and solver, for example the SOSTOOLS [15], which is used to transform a sum of squares problem into an SDP formulation, GloptiPoly [8], used to handle optimization problems over polynomials, and the R-RoMuLOC [14, 4, 13, 22], a toolbox for the manipulation and resolution of conditions related to robust multi-objective control.

To motivate the user of ROLMIP consider, for instance, the problem of analyzing the stability of a discrete-time linear system given by

$$x(k+1) = Ax(k), \tag{1}$$

with $x(k) \in \mathbb{R}^n$ being the state vector of the system and $A \in \mathbb{R}^{n\times n}$ being the dynamic matrix. Using the Lyapunov stability theory, such system is stable if and only if there exists a symmetric matrix $P \in \mathbb{R}^{n\times n}$ such that the LMI

$$\begin{bmatrix} P & A'P \\ PA & P \end{bmatrix} > 0 \tag{2}$$

holds. Such LMI can be easily programmed and solved using, respectively, any LMI parser and solver available.

Consider now that system (1) is affected by uncertainties, i.e., the system matrix is parameter-dependent and is given by $A(\alpha)$. The robust stability analysis of such system can be performed by rewriting the LMI

*Federal University of Technology of Parana – UTFPR, Campus Cornélio Procópio, PR. E-mail: agulhari@utfpr.edu.br

†School of Electrical and Computer Engineering, University of Campinas – UNICAMP, 13083-852, Campinas, SP, Brazil. E-mails: o.alexandre.felipe@gmail.com, {ricfow,peres}@dt.fee.unicamp.br

condition (2) as

$$\begin{bmatrix} P(\alpha) & A(\alpha)'P(\alpha) \\ P(\alpha)A(\alpha) & P(\alpha) \end{bmatrix} > 0 \tag{3}$$

but, in this case, (3) must hold for all admissible α. In order to transform the parameter-dependent LMI into a finite set of standard LMIs, some information about $A(\alpha)$ must be added, as well as some structure must be imposed to the unknown variable $P(\alpha)$. For instance, consider that $A(\alpha)$ and $P(\alpha)$ have a polytopic structure

$$A(\alpha) = \sum_{i=1}^{N} \alpha_i A_i \ , \ P(\alpha) = \sum_{i=1}^{N} \alpha_i P_i \ , \quad \alpha \in \Delta_N, \tag{4}$$

being A_i and P_i the vertices of the respective polytopes, N the number of vertices and Δ_N the set known as unit simplex, given by

$$\Delta_N = \Big\{\alpha \in \mathbb{R}^N : \sum_{i=1}^{N} \alpha_i = 1 \ , \quad \alpha_i \geq 0 \ , \quad i = 1, \ldots, N\Big\}. \tag{5}$$

Applying the definition of $A(\alpha)$ and the chosen structure for $P(\alpha)$, given by (4), to the robust stability condition expressed in the parameter-dependent LMI (3), one gets the following homogeneous polynomial matrix inequality, of degree 2 on α,

$$\begin{bmatrix} P(\alpha) & A(\alpha)'P(\alpha) \\ P(\alpha)A(\alpha) & P(\alpha) \end{bmatrix} = \sum_{i=1}^{N} \alpha_i^2 \begin{bmatrix} P_i & A_i'P_i \\ P_iA_i & P_i \end{bmatrix} + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \alpha_i\alpha_j \begin{bmatrix} P_i + P_j & A_i'P_j + A_j'P_i \\ P_iA_j + P_jA_i & P_i + P_j \end{bmatrix} > 0. \tag{6}$$

A sufficient (but not necessary) way to guarantee that (3) holds is to impose that all the matrix coefficients of the monomials are positive definite. This has been done, for instance, in [16] (discrete-time systems) and [17] (continuous-time systems). A less conservative set of conditions may be obtained by modeling the variable $P(\alpha)$ in (6) as a homogeneous polynomial with generic degree $g > 1$ and then imposing the positivity of all matrix coefficients [10, 2]. Programming these LMIs requires an *a priori* knowledge on the formation law of the monomials, which depends on the number N of uncertain parameters and on the degree of the polynomial variable $P(\alpha)$. In [11], a systematic way to deal with such cases has been developed, but in the context of robust LMIs presenting at most products between two parameter-dependent matrices. When the LMIs to be solved are more complex and have products involving three or more parameter-dependent matrices, a systematic procedure to generate the coefficients of the monomials can be very hard (at least tedious) to achieve.

Consider now that matrix $A(\cdot)$ depends on two parameters, α and β, being each parameter contained in an independent unit simplex, *i.e.*,

$$A(\alpha, \beta) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \alpha_i\beta_j A_{ij}, \quad \alpha \in \Delta_{N_1}, \beta \in \Delta_{N_2}, \tag{7}$$

That is, the parameters α and β lie in the multi-simplex $\Omega = \Delta_{N_1} \times \Delta_{N_2}$. Defining

$$P(\alpha, \beta) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \alpha_i\beta_j P_{ij}, \tag{8}$$

then the parameter-dependent LMI to verify the robust stability of the system becomes

$$\begin{bmatrix} P(\alpha, \beta) & A(\alpha, \beta)'P(\alpha, \beta) \\ P(\alpha, \beta)A(\alpha, \beta) & P(\alpha, \beta) \end{bmatrix} > 0, \tag{9}$$

which is a completely different case than the one in (6), with $A(\alpha)$ described in terms of one simplex as in (4), therefore with a different formation law for the monomials. This illustrates that each new case requires the manipulation of different polynomials and such task, as well as programming the resulting LMIs, can be tedious, time-demanding and also a source of programming errors.

The toolbox described in this manual, called Robust LMI Parser[1], intends to overcome these problems by automatically computing all the coefficients of the monomials of a given parameter-dependent LMI. The toolbox is developed for MATLAB and works jointly with YALMIP, returning the entire set of LMIs through a few simple commands that describe the structure of the known matrices and variables involved in the parameter-dependent LMIs to be investigated.

[1]Available at `http://www.dt.fee.unicamp.br/~agulhari/softwares/robust_lmi_parser.zip`

2 Installation notes

ROLMIP was built on the top of YALMIP [9], a freely distributed general purpose optimization parser MATLAB toolbox. Therefore the first step before using ROLMIP is to download[2] and install YALMIP.

The ROLMIP installation is simple: just unzip the installation file (`robust_lmi_parser.zip`) and then add the decompressed folder on MATLAB's path. To check if ROLMIP is working properly, run the script `rolmiptest.m`.

2.1 Notes on Version 3.0

The current version is considerably different from the previous ones, mainly due to the introduction of a new variable type, named `rolmipvar`. With this modification, the commands and definitions are more straightforward and easy to use and understand, and even the computational time necessary to parse the parameter-dependent LMIs has been optimized. We strongly suggest the use of `rolmipvar` instead of the old commands `poly_struct` and `parser_poly`. However, such old commands are still implemented and even somewhat improved on the current version, and the MATLAB routines that already use ROLMIP continue to work.

Concerning specifically the automatic creation of executable MATLAB files: the latter version provided a set of procedures (using the old commands `poly_struct` and `parser_poly`) whose objective was to generate automatically MATLAB .m files, saving computational time if a set of conditions are executed recurrently. Such features are still part of ROLMIP, but not yet adapted to work using the `rolmipvar` variables included in the present version. Please refer to the manual of the latter version for further details.

[2] `https://yalmip.github.io/download/`

3 Preliminaries

ROLMIP currently considers the following assumptions:

- The polynomial matrices are dependent on parameters that either are defined over a unit (or multi) simplex domain, or have bounds known by the user;
- The number of vertices of a given simplex is always the same, although different simplex domains may present different numbers of vertices.

Several examples illustrating the usage of the commands are presented throughout the manual. Such commands are displayed using the `Typewriter` font and start with `>>`.

4 Defining the polynomial structure

Every polynomial variable, regardless if depending on (multi) simplex or bounded parameters, is defined of type `rolmipvar`. The variable definition may be performed using several different syntaxes[3] depending on how the polynomial is described by the user, as detailed as follows.

If the polynomial is composed by known matrices (as opposed to `sdpvar`[4] variables), the polynomial variable is obtained by

`poly = rolmipvar(M,label,vertices,degree).`

The output `poly` is a `rolmipvar` variable that fully describes the polynomial. The input `M` corresponds to the coefficients of the monomials of the homogeneous polynomial to be defined. For example, consider that matrix $A(\alpha)$ has the following structure

$$A(\alpha) = \alpha_1 A_1 + \alpha_2 A_2, \quad \alpha_1 + \alpha_2 = 1, \quad \alpha_1 \geq 0, \alpha_2 \geq 0$$

i.e., $A(\alpha)$ is characterized by a polytope of $N = 2$ vertices and is modeled as a homogeneous polynomial of degree $g = 1$ with parameters in the unit simplex. There are three ways of inputting the coefficients A_1 and A_2 through variable `M`:

1. Concatenating the matrices A_1 and A_2;

```
>> M = [A1 A2];
```

2. Using `M` as a cell array;

```
>> M{1} = A1; M{2} = A2;                (10)
```

3. Informing, in the cell array `M`, the exponent of the monomial to which each matrix is related.

```
>> M{1} = {[1 0],A1}; M{2} = {[0 1],A2};                (11)
```

Using this syntax, the exponent of the monomial $\alpha_1^{n_1}\alpha_2^{n_2}\ldots\alpha_N^{n_N}$ is encoded as $[n_1\ n_2\ \ldots\ n_N]$. If more simplexes are used, then each cell has extra elements; refer to Example 2.

Please note that, if the first two ways are used, then every monomial must be defined, and the order of elements is important. On the other hand, if the third way is used, then there is no specific order, and the undefined monomials are set to zero by default.

The input `label` is a string and consists of a name used to refer the variable, mainly to generate the LaTeX code of a polynomial using the command `texify`, detailed in Section 5. The inputs `vertices` and `degree` inform, respectively, the vertices and the degrees of each simplex associated to the polynomial. If a parameter-independent matrix is to be defined, then the parameters `vertices` and `degree` may be omitted or set to zero.

A constant matrix may be defined by setting `vertices` with the number of vertices of the considered system and `degree = 0`. However, the structure will be more complex and may result on more expensive computations.

[3]The syntaxes are somewhat similar to the `poly_struct` command available on prior versions.
[4]Variables used in YALMIP toolbox.

Example 1

A polynomial matrix $A(\alpha)$ with degree $g = 1$ that represents a polytopic linear system with $N = 3$ vertices is given by

$$A(\alpha) = \alpha_1 A_1 + \alpha_2 A_2 + \alpha_3 A_3, \quad \alpha \in \Delta_3$$

and can be defined using the following sequence of commands.

```
>> A = [A1 A2 A3];
>> poly_A = rolmipvar(A,'A',3,1);
```

where `A1`, `A2` and `A3` are previously defined matrices. To retrieve the matrix-valued coefficient A_2, for example, it suffices to type `>> A2 = poly_A([0 1 0]);`

Example 2

A polynomial matrix $A(\alpha, \beta)$, given by

$$A(\alpha) = \alpha_1^2\beta_1 A_1 + \alpha_1\alpha_2\beta_1 A_2 + \alpha_2^2\beta_1 A_3 + \alpha_1^2\beta_2 A_4 + \alpha_1\alpha_2\beta_2 A_5 + \alpha_2^2\beta_2 A_6 + \alpha_1^2\beta_3 A_7 + \alpha_1\alpha_2\beta_3 A_8 + \alpha_2^2\beta_3 A_9,$$

i.e., with 2 vertices and degree 2 on the first simplex, and 3 vertices and degree 1 on the second simplex, can be defined using the following sequence of commands.

```
>> A{1} = {[2 0],[1 0 0],A1};
>> A{2} = {[1 1],[1 0 0],A2};
>> A{3} = {[0 2],[1 0 0],A3};
>> A{4} = {[2 0],[0 1 0],A4};
>> A{5} = {[1 1],[0 1 0],A5};
>> A{6} = {[0 2],[0 1 0],A6};
>> A{7} = {[2 0],[0 0 1],A7};
>> A{8} = {[1 1],[0 0 1],A8};
>> A{9} = {[0 2],[0 0 1],A9};
>> poly_A = rolmipvar(A,'A',[2 3],[2 1]);
```

To retrieve the matrix-valued coefficient A_4, for example, it suffices to type `>> A4 = poly_A([2 0],[0 1 0]);`

In Example 2, two simplexes were used: α and β. Note that, when defining the structure of `A{·}`, the first element of the cell array corresponds to α and the second corresponds to β. When using ROLMIP, it is important to keep the track on the assortment of the defined simplexes, otherwise errors can occur. For ease of notation, in this manual the simplex names are sorted, by default, in the order $\{\alpha, \beta, \gamma, \delta, \ldots\}$.

Example 3

A 3×3 identity matrix can be defined using

```
>> poly_I = rolmipvar(eye(3),'I');
```

A monomial can be individually set after being defined. For instance, in Example 2, to set the monomial with the term $\alpha_1^2\beta_2$ equal to identity I, one might enter the command `>> poly_A([2 0],[0 1 0]) = eye(3);`

To declare a polynomial matrix as an optimization variable, one may use the following syntax

```
poly = rolmipvar(rows,cols,label,param,vertices,degree).
```

The coefficients of the monomials are matrices with dimensions `rows` $\times$ `cols` whose entries are optimization variables and, since ROLMIP is built on the top of YALMIP, they are internally stored using the `sdpvar` type provided by YALMIP. The argument `param` is a string that indicates if the variable is symmetric (`'symmetric'`), rectangular (`'full'`), symmetric Toeplitz (`'toeplitz'`), symmetric Hankel (`'hankel'`) or

Skew-symmetric (`'skew'`). If `param` is not informed, square matrices are declared as symmetric and non-square matrices are defined as full. Note that such argument is similar to the `sdpvar` instruction used to define the variables in the YALMIP parser.

To define a known scalar in terms of a ROLMIP structure, one may use the syntax

```
poly = rolmipvar(M,label,'scalar'),
```

which returns the structure related to the scalar `M` with label given by `label`. If the scalar is a `sdpvar` variable, the following syntax may be used

```
poly = rolmipvar(label,'scalar').
```

It is important to highlight that it is not necessary to define parameter-independent matrices and scalars as `rolmipvar` variables, unless the user intends to generate the LaTeXcode of the polynomials, as discussed in details in Section 5.

Example 4

A symmetric polynomial variable $P(\alpha) \in \mathbb{R}^{3\times 3}$ of degree 2, with α in a simplex of dimension $N = 3$, can be defined by

```
>> poly_P = rolmipvar(3,3,'P','symmetric',3,2);
```

Example 5

A full polynomial variable $P(\alpha, \beta) \in \mathbb{R}^{3\times 3}$ of degree 2 and $N = 3$ vertices on the first simplex, and degree 1 and $N = 4$ vertices on the second simplex, can be defined by

```
>> poly_P = rolmipvar(3,3,'P','symmetric',[3 4],[2 1]);
```

Suppose now that one needs to define a polynomial matrix $A(\theta_1, \theta_2)$, given by

$$A(\theta_1, \theta_2) = A_0 + \theta_1 A_1 + \theta_2 A_2 + \theta_1^2 \theta_2 A_3,$$

with

$$\underline{a}_1 \leq \theta_1 \leq \overline{a}_1, \quad \underline{a}_2 \leq \theta_2 \leq \overline{a}_2.$$

ROLMIP allows the definition of such polynomial, provided that the bounds of the parameters are also informed. The syntax for such is

```
poly = rolmipvar(M,label,bounds).
```

The input `M` is a cell array that contains the informations on the matrix coefficients and the associated monomials, defined in a similar way as in (11). The input `label` informs the label of the variable, and `bounds` is an $m \times 2$ matrix that contains the bounds of the m parameters considered (first column with the lower limits and the second column with the upper limits).

Example 6

The polynomial matrix $A(\theta_1, \theta_2)$, given by

$$A(\theta_1, \theta_2) = A_0 + \theta_1 A_1 + \theta_2 A_2 + \theta_1^2 \theta_2 A_3,$$

with

$$-2 \leq \theta_1 \leq 3, \quad 4 \leq \theta_2 \leq 8,$$

can be defined using the following sequence of commands.

```
>> A{1} = {[0 0],A0};
>> A{2} = {[1 0],A1};
>> A{3} = {[0 1],A2};
>> A{4} = {[2 1],A3};
>> poly_A = rolmipvar(A,'A',[-2 3; -4 8]);
```

Alternatively, if the polynomial matrix is an optimization variable, one may use the syntax

`poly = rolmipvar(rows,cols,label,parametr,polmask,bounds).`

The matrix coefficients with dimensions `rows` $\times$ `cols` are internally declared, and returned as `sdpvar` variables to the output `poly`. The argument `parametr` is a string that indicates if the variable is symmetric (`'symmetric'`), rectangular (`'full'`), symmetric Toeplitz (`'toeplitz'`), symmetric Hankel (`'hankel'`) or Skew-symmetric (`'skew'`). If `parametr` is not informed, square matrices are declared as symmetric and non-square matrices are defined as full. Note that such command is similar to the `sdpvar` instruction used to define the variables in the YALMIP parser. The input `polmask` is a cell array containing the exponents of the parameters desired for the output polynomial, and `bounds` contains the bounds of the parameters.

Example 7

To define the 3×3 symmetric polynomial variable $P(\theta_1, \theta_2)$, whose desired structure is given by

$$P(\theta_1, \theta_2) = P_0 + \theta_1 P_1 + \theta_2^3 P_2 + \theta_1^2 \theta_2^4 P_3,$$

with

$$-2 \leq \theta_1 \leq 3, \quad 4 \leq \theta_2 \leq 8,$$

one may use the syntax

```
>> poly_P = rolmipvar(3,3,'P','sym',{[0 0],[1 0],[0 3],[2 4]},[-2 3; -4 8]);
```

Internally, the polynomial with parameters lying in known intervals is converted to a multi-simplex representation, each parameter generating a different simplex with 2 vertices. In general terms, the parameter θ_i is rewritten as

$$\theta_i = \alpha_1 \underline{a}_i + \alpha_2 \overline{a}_i,$$

and then the polynomial is homogenized.

5 Operating on polynomials

Once the `rolmipvar` variables are defined, the most common operations may be performed between polynomials or between constant matrices and polynomials. Every necessary homogenization is automatically performed by ROLMIP, as illustrated in the following example.

Example 8

Suppose that the `rolmipvar` variables `poly_A` and poly_B have already been defined and are respectively given by

$$A(\alpha) = \alpha_1 A_1 + \alpha_2 A_2, \quad B(\alpha) = \alpha_1 B_1 + \alpha_2 B_2.$$

The sum

```
>> resul = poly_A + poly_B;
```

yields a polynomial given by

$$A(\alpha) + B(\alpha) = \alpha_1(A_1 + B_1) + \alpha_2(A_2 + B_2)$$

The product

```
>> resul = poly_A*poly_B;
```

yields a polynomial given by

$$A(\alpha)B(\alpha) = \alpha_1^2(A_1 B_1) + \alpha_1 \alpha_2(A_1 B_2 + A_2 B_1) + \alpha_2^2(A_2 + B_2).$$

Finally, operations between polynomials and non-polynomial matrices are performed as expected. Let C be a matrix (with compatible dimensions) defined on MATLAB. The operation

```
>> resul = poly_A + C;
```

produces

$$A(\alpha) + C = \alpha_1(A_1 + C) + \alpha_2(A_2 + C),$$

and the operation

```
>> resul = poly_A*C;
```

results on

$$A(\alpha)C = \alpha_1(A_1C) + \alpha_2(A_2C).$$

In general, the MATLAB operations that are adapted to work over `rolmipvar` variables are:

- Transpose operation: the transpose is applied over the matrix-valued coefficients of all monomials;
- Command `blkdiag`: used if one needs to construct a `rolmipvar` block-diagonal matrix;
- Command `trace`: returns a `rolmipvar`, with each coefficient corresponding to the trace of the respective matrix monomial, since

$$\text{trace}\left(\sum_{i=1}^{N} \alpha_i A_i\right) = \sum_{i=1}^{N} \alpha_i \text{trace}(A_i).$$

Other implemented commands are described in the following.

Command FORK

This command replaces the parameters from one (multi) simplex domain to another.

The syntax

```
polyout = fork(polyin,newlabel)
```

changes the entire multi-simplex domain of the polynomial `polyin`, generating a new variable with label `newlabel`.

Example 9

Suppose that `polyin` represents the polynomial

$$A(\alpha) = \alpha_1 A_1 + \alpha_2 A_2.$$

The application of the command `fork` results on the polynomial `polyout` that represents

$$A(\beta) = \beta_1 A_1 + \beta_2 A_2.$$

Example 10

Suppose that `polyin` represents the polynomial

$$A(\alpha, \beta) = \alpha_1\beta_1 A_1 + \alpha_2\beta_1 A_2 + \alpha_1\beta_2 A_3 + \alpha_2\beta_2 A_4.$$

The application of the command `fork` results on the polynomial `polyout` that represents

$$A(\gamma, \delta) = \gamma_1\delta_1 A_1 + \gamma_2\delta_1 A_2 + \gamma_1\delta_2 A_3 + \gamma_2\delta_2 A_4.$$

Note that the command `fork` only changes the parameters that define the monomials, but not the values of the coefficients of the monomials.

The syntax

```
polyout = fork(polyin,newlabel,targetin)
```

only change the simplexes given in input vector `targetin`.

Example 11

Suppose that polyin represents the polynomial

$$A(\alpha, \beta, \gamma) = \alpha_1\beta_1\gamma_1 A_1 + \alpha_2\beta_1\gamma_2 A_2 + \alpha_1\beta_2\gamma_3 A_3 + \alpha_2\beta_2\gamma_1 A_4.$$

The command

```
>> polyout = fork(polyin,'B',[2]);
```

provides the polynomial polyout that represents

$$B(\alpha, \delta, \gamma) = \alpha_1\delta_1\gamma_1 A_1 + \alpha_2\delta_1\gamma_2 A_2 + \alpha_1\delta_2\gamma_3 A_3 + \alpha_2\delta_2\gamma_1 A_4.$$

Finally, the syntax

```
polyout = fork(polyin,newlabel,targetin,targetout)
```

change the simplexes on input vector targetin, respectively, to the simplexes informed on targetout.

Example 12

Suppose that polyin represents the polynomial

$$A(\alpha, \beta, \gamma) = \alpha_1\beta_1\gamma_1 A_1 + \alpha_2\beta_1\gamma_2 A_2 + \alpha_1\beta_2\gamma_3 A_3 + \alpha_2\beta_2\gamma_1 A_4.$$

The command

```
>> polyout = fork(polyin,'B',[1 2],[2 4]);
```

results on the polynomial polyout that represents

$$A(\beta, \delta, \gamma) = \beta_1\delta_1\gamma_1 A_1 + \beta_2\delta_1\gamma_2 A_2 + \beta_1\delta_2\gamma_3 A_3 + \beta_2\delta_2\gamma_1 A_4.$$

Command ADDSIMPLEX

The command addsimplex, whose syntax is

```
>> polyout = addsimplex(polyin,vertices);
```

adds a new simplex to the informed polynomial polyin, and the inserted simplex has the number of vertices indicated in input vertices.

Example 13

Suppose that polyin represents the polynomial

$$A(\alpha, \beta, \gamma).$$

The command

```
>> polyout = addsimplex(polyin,3);
```

results on the polynomial polyout that represents

$$A(\alpha, \beta, \gamma, \delta),$$

being δ a simplex with 3 vertices.

Command EVALPAR

The command `evalpar` is used to compute the value of the matrix polynomial given the values of the parameters. The syntax is

```
>> var = evalpar(poly,paramval)
```

The input `poly` contains the `rolmipvar` variable, and `paramval` contains the values of the parameters. If the polynomial is defined directly on the (multi) simplex, then `paramval` is a cell array (even if the polynomial is defined over only one simplex), and `paramval` must be a vector if `poly` is a matrix polynomially dependent on bounded parameters.

Example 14

Suppose that `poly` represents the polynomial

$$A(\alpha) = \alpha_1^2 \begin{bmatrix} 1 & 0 \\ 2 & -1 \end{bmatrix} + \alpha_1\alpha_2 \begin{bmatrix} -3 & 4 \\ 2 & 3 \end{bmatrix} + \alpha_2^2 \begin{bmatrix} 0 & 1 \\ -2 & 4 \end{bmatrix}.$$

The command

```
>> var = evalpar(poly,{[0.3 0.7]})
```

performs the calculation

$$(0.3)^2 \begin{bmatrix} 1 & 0 \\ 2 & -1 \end{bmatrix} + (0.3)(0.7) \begin{bmatrix} -3 & 4 \\ 2 & 3 \end{bmatrix} + (0.7)^2 \begin{bmatrix} 0 & 1 \\ -2 & 4 \end{bmatrix} = \begin{bmatrix} -0.54 & 1.33 \\ -0.38 & 2.5 \end{bmatrix}$$

Example 15

Suppose that `poly` represents the polynomial

$$A(\alpha, \beta) = \alpha_1\beta_1^2 \begin{bmatrix} 1 & 0 \\ 2 & -1 \end{bmatrix} + \alpha_1\beta_1\beta_2 \begin{bmatrix} -3 & 4 \\ 2 & 3 \end{bmatrix} + \alpha_2\beta_1\beta_2 \begin{bmatrix} 0 & 1 \\ -2 & 4 \end{bmatrix} + \alpha_2\beta_2^2 \begin{bmatrix} 4 & 7 \\ 1 & -2 \end{bmatrix}.$$

The command

```
>> var = evalpar(poly,{[0.3 0.7],[0.1 0.9]})
```

performs the calculation

$$(0.3)(0.1)^2 \begin{bmatrix} 1 & 0 \\ 2 & -1 \end{bmatrix} + (0.3)(0.1)(0.9) \begin{bmatrix} -3 & 4 \\ 2 & 3 \end{bmatrix} + (0.7)(0.1)(0.9) \begin{bmatrix} 0 & 1 \\ -2 & 4 \end{bmatrix}$$
$$+ (0.7)(0.9)^2 \begin{bmatrix} 4 & 7 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} 2.190 & 4.140 \\ 0.501 & -0.804 \end{bmatrix}$$

Example 16

Suppose that `poly` represents the polynomial

$$A(\theta) = \begin{bmatrix} 0 & 1 \\ 2 & 4 \end{bmatrix} + \theta_1 \begin{bmatrix} 2 & 3 \\ -2 & 5 \end{bmatrix} + \theta_1^2\theta_2 \begin{bmatrix} 5 & -1 \\ 2 & 0 \end{bmatrix} + \theta_2^2 \begin{bmatrix} 3 & 3 \\ 0 & 8 \end{bmatrix}.$$

The command

```
>> var = evalpar(poly,[2 3])
```

performs the calculation

$$A(\theta) = \begin{bmatrix} 0 & 1 \\ 2 & 4 \end{bmatrix} + 2 \begin{bmatrix} 2 & 3 \\ -2 & 5 \end{bmatrix} + (2^2)3 \begin{bmatrix} 5 & -1 \\ 2 & 0 \end{bmatrix} + 3^2 \begin{bmatrix} 3 & 3 \\ 0 & 8 \end{bmatrix} = \begin{bmatrix} 91 & 22 \\ 22 & 86 \end{bmatrix}.$$

Command DOUBLE

On YALMIP parser [9], the command `double` is applied on `sdpvar` variables, returning the double precision values of the variables (if such values are already assigned). If the command `double` is applied on a `rolmipvar` variable, it returns the respective polynomial with monomials given on the double precision values.

Command DIFF

Usually, when dealing with linear parameter-varying (LPV) systems, where the parameters may be time-varying, the LMI conditions depend on the time-derivative of some parameter-dependent matrix variables. For example, the continuous-time LPV system

$$\dot{x}(t) = A(\alpha(t))x(t), \quad \alpha(t) \in \Delta_N,$$

is asymptotically stable if there is a symmetric definite positive matrix $P(\alpha(t))$ satisfying

$$A(\alpha(t))'P(\alpha(t)) + P(\alpha(t))A(\alpha(t)) + \frac{d}{dt}P(\alpha(t)) < 0.$$

For ease of notation, consider that

$$\frac{d}{dt}P(\alpha(t)) = \dot{P}(\alpha(t)).$$

Suppose that

$$P(\alpha(t)) = \sum_{i=1}^{N} \alpha_i(t)P_i.$$

Then one has

$$\dot{P}(\alpha(t)) = \sum_{i=1}^{N} \dot{\alpha}_i(t)P_i.$$

If the variation rates of the parameters $\alpha(t)$ are bounded, and such bounds are previously known, then $\dot{P}(\alpha(t))$ can be represented in a polytopic way over a different simplex, such as

$$\dot{P}(\alpha(t)) = \sum_{i=1}^{N}\sum_{j=1}^{M} \beta_j H_j^i P_i, \qquad \beta \in \Delta_M,$$

being H_j^i the element (i, j) of the matrix H, used to describe the new polytope. For more details on such transformation, please refer, for instance, to [7, 5].

The presented transformation can be automatically performed using the command `diff` of ROLMIP, whose standard syntax is

```
>> dotpoly = diff(poly,labelout,dotbounds)
```

The polynomial to be derived is given by `poly`, the label of the derived polynomial is given by `labelout`[5], and `dotbounds` contains the bounds of the variation rates of the parameters. Note that, withouth knowing such bounds, it is not possible to generate a polytope where the vector $\dot{\alpha}(t)$ lie. If the input polynomial depends on only one simplex, then `dotbounds` can be a vector; otherwise, it must be a cell array, each cell containing the information about one simplex.

Example 17

Suppose that one needs to calculate the derivative of the polynomial `poly`, defined over one simplex of four vertices, being the variation rates of each parameter bounded by

$$-1 \le \dot{\alpha}_1(t) \le 2, \quad -3 \le \dot{\alpha}_2(t) \le 4, \quad -8 \le \dot{\alpha}_3(t) \le 6, \quad -5 \le \dot{\alpha}_4(t) \le 9.$$

The derivative of the polynomial is obtained by the command

```
>> dotpoly = diff(poly,labelout,[-1 2; -3 4; -8 6; -5 9]);
```

[5] If the input `labelout` is not informed, then the label of the output polynomial is composed by the string `dot_` concatenated with the label of the input polynomial.

Example 18

Suppose that one needs to calculate the derivative of the polynomial `poly`, defined over two simplexes: the first (α) of three vertices, and the second (β) of two vertices. The variation rates of each parameter are bounded by

$$-1 \leq \dot{\alpha}_1(t) \leq 2, \quad -3 \leq \dot{\alpha}_2(t) \leq 4, \quad -8 \leq \dot{\alpha}_3(t) \leq 6, \quad -4 \leq \dot{\beta}_1(t) \leq 1, \quad -6 \leq \dot{\beta}_2(t) \leq 9.$$

The derivative of the polynomial is obtained by the commandss

```
>> dotbounds{1} = [-1 2; -3 4; -8 6];
>> dotbounds{2} = [-4 1; -6 9];
>> dotpoly = diff(poly,labelout,dotbounds);
```

If the polynomial is defined from a matrix polynomially dependent on bounded parameters, then the derivative can be obtained using the syntax

```
>> dotpoly = diff(poly,labelout,bounds,dotbounds)
```

The input `bounds` informs the bounds of each parameter, and the bounds of their variation rates are given by the vector `dotbounds`.

Example 19

Suppose that one needs to calculate the derivative of the polynomial `poly`, depending on parameters θ_1, θ_2 and θ_3. The bounds and variation rates are given by

$$-2 \leq \theta_1(t) \leq 4, \quad 3 \leq \theta_2(t) \leq 6, \quad 1 \leq \theta_3(t) \leq 8,$$
$$-1 \leq \dot{\theta}_1(t) \leq 5, \quad -3 \leq \dot{\theta}_2(t) \leq 2, \quad -7 \leq \dot{\theta}_3(t) \leq 3.$$

The derivative of such polynomial can be calculated using the command

```
>> dotpoly = diff(poly,labelout,[-2 4; 3 6; 1 8],[-1 5; -3 2; -7 3]);
```

Command PARTIAL

The command `partial` computes the partial derivatives of a polynomial variable with respect to the simplex parameters. The syntax of `partial` is

```
>> dpoly = partial(poly,labelout,simpnum)
```

where `poly` is the input polynomial, `labelout` is the label of the resultant polynomial, `simpnum` is the index of the simplex domain in which respect the polynomial is differentiated, and `dpoly` is a cell structure containing the computed gradient vector.

For instance, suppose that $P(\alpha, \beta)$ is a polynomial with α being a simplex with three vertices, and β representing a polytope with two vertices. The command

```
>> dpoly = partial(poly,'dPda',1);
```

computes the gradient vector

$$\frac{\partial P(\alpha,\beta)}{\partial \alpha} = \begin{bmatrix} \frac{\partial P(\alpha,\beta)}{\partial \alpha_1} & \frac{\partial P(\alpha,\beta)}{\partial \alpha_2} & \frac{\partial P(\alpha,\beta)}{\partial \alpha_3} \end{bmatrix}.$$

Similarly, the command

```
>> dpoly = partial(poly,'dPdb',2);
```

yields

$$\frac{\partial P(\alpha,\beta)}{\partial \beta} = \begin{bmatrix} \frac{\partial P(\alpha,\beta)}{\partial \beta_1} & \frac{\partial P(\alpha,\beta)}{\partial \beta_2} \end{bmatrix}.$$

Command TEXIFY

The command `texify`, with syntax given by

```
>> out = texify(poly,option,var1,...,varN,simplexnames)
```

returns the LaTeX code of the input polynomial `poly`. The argument `option` is a string that informs if the desired output format presents an explicit dependence of the polynomial on the simplexes (`option = 'explicit'`), only the polynomial variables without showing their dependence on the simplexes (`option = 'implicit'`), or the polynomial structure showing each monomial (`option = 'polynomial'`). The arguments `var1, ..., varN` are the variables that are used in the expression to be output; it is necessary to inform separately each variable in such a way that the procedure can assess the information needed. Finally, the input `simplexnames` is a cell array of strings containing the desired names for each simplex, already in LaTeX format. If only one string is informed instead of a cell array of strings, then it is supposed that the user intends to represent all the simplexes using one single (multi-simplex) variable. If `simplexnames` is not informed, then standard names are used. The name of the polynomial variables is equal to the `label` informed in their definition.

Example 20

Consider the polynomial $A(\alpha,\beta)$, being both simplexes of degree 1 and 2 vertices and defined by the variable `Ai`, and the polynomial $P(\alpha)$, of degree 1 and defined by the variable `Pi`. The command

```
>> out = texify(Ai'*Pi + Pi*Ai,'explicit',Ai,Pi,{'\alpha','\beta'})
```

returns the LaTeX code
`A(\alpha,\beta)'P(\alpha)+P(\alpha)A(\alpha,\beta)`.

On the other hand, the command

```
>> out = texify(Ai'*Pi + Pi*Ai,'explicit',Ai,Pi,'\alpha')
```

returns the LaTeX code
`A(\alpha)'P(\alpha)+P(\alpha)A(\alpha)`.

The command

```
>> out = texify(Ai'*Pi + Pi*Ai,'implicit',Ai,Pi)
```

returns the LaTeX code
`A'P+PA`.

Finally, the command

```
>> out = texify(Ai'*Pi + Pi*Ai,'polynomial',Ai,Pi,{'\alpha','\beta'})
```

returns the LaTeX code

```
\alpha_{1}^{2}\beta_{1}(A_{1}'P_{1}+P_{1}A_{1})
 + \alpha_{1}\alpha_{2}\beta_{1}(A_{1}'P_{2}+A_{2}'P_{1}+P_{1}A_{2}+P_{2}A_{1})
 + \alpha_{2}^{2}\beta_{1}(A_{2}'P_{2}+P_{2}A_{2})
 + \alpha_{1}^{2}\beta_{2}(A_{3}'P_{1}+P_{1}A_{3})
 + \alpha_{1}\alpha_{2}\beta_{2}(A_{3}'P_{2}+A_{4}'P_{1}+P_{1}A_{4}+P_{2}A_{3})
 + \alpha_{2}^{2}\beta_{2}(A_{4}'P_{2}+P_{2}A_{4}).
```

6 Composing matrices and LMIs

The construction of matrices of polynomials and LMIs using ROLMIP is done in an intuitive way, just as in YALMIP[6]. For example, suppose that the LMI given in (3) is to be implemented, and suppose that the matrices $A(\alpha)$ and $P(\alpha)$ are defined by the `rolmipvar` variables `Ai` and `Pi`, respectively. The resulting set of LMIs is defined using

```
>> LMIs = [[Pi Ai'*Pi; Pi*Ai Pi] > 0];
```

Any necessary degree homogenization is automatically performed by ROLMIP.

6.1 Pólya's relaxation

An interesting relaxation usually performed when dealing with LMI conditions over homogeneous polynomials is based on the Pólya's theorem for the case of positive polynomials with matrix-valued coefficients [18, 19, 12]. For a better understanding of Pólya's relaxation, suppose for example condition (3), and suppose that there exists a matrix $P(\alpha) = P(\alpha)^T > 0$ of degree g satisfying the condition. A simple way to verify that $P(\alpha)$ satisfies the condition is to check if each monomial of the homogeneous polynomial is positive definite. This is a sufficient way, but not necessary; in some cases, the polynomial is positive definite but not all the coefficients of the monomials are so. In this case, according to Pólya's theorem, there exists an integer $d > 0$ such that the monomials of

$$\left(\sum_{j=1}^{M}\sum_{i=1}^{N}\alpha_i^{(j)}\right)^d \begin{bmatrix} P(\alpha) & A(\alpha)'P(\alpha) \\ P(\alpha)A(\alpha) & P(\alpha) \end{bmatrix} \tag{12}$$

are definite positive, being $\alpha^{(j)}$ the j-th simplex on a multi-simplex domain.

The Pólya's relaxation technique is currently implemented on ROLMIP within the command `polya`, with syntax

```
>> polyout = polya(polyin,d)
```

being `d` the degree d of the relaxation.

In order to implement the Pólya's relaxation depicted in (12) for a given value of d, for instance $d = 3$, use the sequence of commands as follows.

```
>> cond = [Pi Ai'*Pi; Pi*Ai Pi]; >> d = 3;
>> condpolya = polya(cond,d);
>> LMIs = [condpolya > 0];
```

In the following section, more involved examples based on uncertain linear systems problems, are presented for a better understanding of ROLMIP.

7 Solving Uncertain Linear Systems Problems with ROLMIP

7.1 Robust Stability Analysis

Consider the problem of robust stability analysis of the discrete-time linear system given in (1) with the following uncertain dynamic matrix

$$A(\alpha) = \alpha_1 \begin{bmatrix} 0.1 & 0.9 \\ 0 & 0.1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0.5 & 0 \\ 1 & 0.5 \end{bmatrix}$$

The first step is the the declaration of $A(\alpha)$ as a `rolmipvar`, which can be performed as (there are other possibilities)

[6] On previous versions of ROLMIP, such construction is performed using the command `construct_lmi`; this command is still implemented for backwards compatibility, but it will be discontinued.

```
>> A1 = [0.1 0.9;0 0.1];
>> A2 = [0.5 0;1 0.5];
>> A = rolmipvar(A,'A(\alpha)',2,1)
```

and the last command prints in screen:

```
Label: A(\alpha)
Vertices: [2]
Degrees:  [1]

a1*[0.1         0.9]  + a2*[0.5           0]
[0          0.1]          [1          0.5]
```

The next step is to choose the structure for the Lyapunov matrix. For instance, consider an affine (degree one) polynomial dependence on α, as in (4). In this case the declaration of $P(\alpha)$ is done through the command

```
>> P = rolmipvar(2,2,'P(\alpha)',2,1)
```

echoing in screen

```
Linear matrix variable 2x2 (symmetric, real, 3 variables)
Label: P(\alpha)
Vertices: [2]
Degrees:  [1]
```

As shown in last section, the programming of the robust stability condition is very easy (the echo in screen, produced by YALMIP, is also shown):

```
>> LMIs = [[P A'*P; P*A P] >= 0]

+++++++++++++++++++++++++++++++++++++++++++++++
|   ID|      Constraint|                   Type|
+++++++++++++++++++++++++++++++++++++++++++++++
|   #1|   Numeric value|   Matrix inequality 4x4|
|   #2|   Numeric value|   Matrix inequality 4x4|
|   #3|   Numeric value|   Matrix inequality 4x4|
+++++++++++++++++++++++++++++++++++++++++++++++
```

Note that the number of LMIs (three) agrees with the number of monomials of the expression given in (6) when $N = 2$. Actually, each monomial gives rise to one LMI. After solving the set of LMIs (using SeDuMi) with the command:

```
>> optimize(LMIs,[],sdpsettings('verbose',0,'solver','sedumi'));
```

and checking the status of the constraints after the optimization, one has

```
>> checkset(LMIs)

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|   ID|      Constraint|               Type|   Primal residual|   Dual residual|
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|   #1|   Numeric value|   Matrix inequality|           0.17957|      1.9163e-14|
|   #2|   Numeric value|   Matrix inequality|           0.12168|       3.058e-14|
|   #3|   Numeric value|   Matrix inequality|          0.056108|       2.283e-14|
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

As the primal residuals are positive, the LMIs are strictly feasible, assuring the robust stability of $A(\alpha)$. Moreover, the Lyapunov matrix that certifies the robust stability can be inspected using the command:

```
>> double(P)
Label: P(\alpha)
Vertices: [2]
Degrees:  [1]

a1*[0.56664   0.0039574]    + a2*[1.0987    0.062489]
[0.0039574     0.87634]        [0.062489     0.45827]
```

7.2 Guaranteed $\mathcal{H}_\infty$ Cost Computation

Consider the continuous-time uncertain time-invariant system given by

$$\begin{cases} \dot{x}(t) = & A(\alpha)x(t) + B(\alpha)w(t) \\ y(t) = & C(\alpha)x(t) + D(\alpha)w(t) \end{cases} \tag{13}$$

with $A(\alpha) \in \mathbb{R}^{n\times n}$, $B(\alpha) \in \mathbb{R}^{n\times r}$, $C(\alpha) \in \mathbb{R}^{q\times n}$ and $D(\alpha) \in \mathbb{R}^{q\times r}$. The transfer function from the disturbance input w to the output y, for a fixed α, is given by

$$H(s,\alpha) = C(\alpha)\big(s\mathrm{I} - A(\alpha)\big)^{-1}B(\alpha) + D(\alpha)$$

The bounded real lemma assures the Hurwitz stability of $A(\alpha)$ (*i.e.*, all eigenvalues have negative real part) for all $\alpha \in \Delta_N$ and a bound γ to the $\mathcal{H}_\infty$ norm of the transfer function from w to y. It can be formulated as follows [3].

Lemma 1
Matrix $A(\alpha)$ is Hurwitz and $\|H(s,\alpha)\|_\infty < \gamma$ for all $\alpha \in \Delta_N$ if there exists a positive definite symmetric matrix $P(\alpha) = P(\alpha)' > 0$ such that

$$\begin{bmatrix} A(\alpha)'P(\alpha) + P(\alpha)A(\alpha) + C(\alpha)'C(\alpha) & \star \\ B(\alpha)'P(\alpha) + D(\alpha)'C(\alpha) & D(\alpha)'D(\alpha) - \gamma^2\mathbf{I} \end{bmatrix} < 0, \quad \forall \alpha \in \Delta_N \tag{14}$$

In this example, consider a fourth-order mass-spring system, also investigated in [12], whose matrices are given by

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{2}{m_1} & \frac{1}{m_1} & -\frac{c_0}{m_1} & 0 \\ \frac{1}{m_2} & -\frac{1}{m_2} & 0 & -\frac{c_0}{m_2} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_1} \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}, \quad D = 0,$$

with

$$0.5 \le m_1 \le 1.5, \quad 0.75 \le m_2 \le 1.25, \quad 1 \le c_0 \le 2.$$

Setting $\theta_1 = 1/m_1$, $\theta_2 = 1/m_2$ and $\theta_3 = c_0$, one has

$$2/3 \le \theta_1 \le 2, \quad 0.8 \le \theta_2 \le 4/3, \quad 1 \le \theta_3 \le 3,$$

and the matrices can be rewritten as

$$A(\theta) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \theta_1 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \theta_1\theta_3 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \theta_2 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} + \theta_2\theta_3 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix},$$

$$B(\theta) = \theta_1 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad C(\theta) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}, \quad D(\theta) = 0.$$

The implementation of LMI relaxations that search for a feasible solution while minimizing $\mu = \gamma^2$ is given in the sequence[7].

[7] This code is available as `Example_5_1.m` in the installation file.

Algorithm 1

```
1     degP = 1;
2     A{1} = {[0 0 0],[0 0 1 0; 0 0 0 1; 0 0 0 0; 0 0 0 0]};
3     A{2} = {[1 0 0],[0 0 0 0; 0 0 0 0; -2 1 0 0; 0 0 0 0]};
4     A{3} = {[1 0 1],[0 0 0 0; 0 0 0 0; 0 0 -1 0; 0 0 0 0]};
5     A{4} = {[0 1 0],[0 0 0 0; 0 0 0 0; 0 0 0 0; 1 -1 0 0]};
6     A{5} = {[0 1 1],[0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 -1]};
7     Ai = rolmipvar(A,'A',[2/3 2; 0.8 4/3; 1 3]);
8     B1 = {[1 0 0],[0;0;1;0]};
9     Bi = rolmipvar(B,'B',[2/3 2]);
10    Ci = rolmipvar([0 1 0 0],'C',0,0)
11    Di = rolmipvar(0,'D',0,0);
12    Pi = rolmipvar(4,4,'P','sym',[2 2 2],[degP degP degP]);
13    mu = sdpvar(1,1);
14    T11 = Ai'*Pi + Pi*Ai + Ci'*Ci;
15    T21 = Bi'*Pi + Di'*Ci;
16    T22 = Di'*Di - mu*eye(1);
17    T = [T11 T21'; T21 T22];
18    LMIs = [LMIs, T < 0];
19    optimize(LMIs,mu);
20    gama = sqrt(double(mu));
```

As illustrated, the implementation of LMI conditions using the Robust LMI Parser is simple and straightforward. Moreover, the different sets of LMIs for larger degrees of the homogeneous polynomial variable $P(\alpha)$ can be readily obtained by simply changing `degP`. Those LMIs, that would demand complex and tedious manipulations without using the parser, provide clear improvements in the computation of the $\mathcal{H}_\infty$ guaranteed cost. Table 1 shows the values of $\gamma^* = \min \gamma$ subject to (14) obtained when the degree of the variable $P(\alpha)$ increases. With $g = 2$, γ^* reaches the worst case value of the $\mathcal{H}_\infty$ norm (computed through brute force).

Table 1: Values of $\gamma^* = \min \gamma$ obtained when varying the degree of the polynomial variable $P(\alpha)$.

degP	γ^*
0	2.8429
1	1.0540
2	1.0108
3	1.0108
4	1.0108
5	1.0108

In order to analyze the results obtained when applying the Pólya's relaxation, as described in Section 6.1, on Algorithm 1 for $d = 3$, it suffices to insert the following block of commands between lines 17 and 18:

```
d = 3;
t = polya(T,d);
```

The results obtained for the application of the latter addition on the code on the minimization of γ, subject to (14) with Pólya's relaxation, are shown on Table 2.

8 Conclusion

A computational package, named Robust LMI Parser, is presented in this manual. The main objective of the toolbox is to facilitate the task of programming LMIs that are sufficient conditions for robust LMIs,

Table 2: Values of $\gamma^* = \min \gamma$ obtained when varying the degree of the polynomial variable $P(\alpha)$ for $d = 3$.

degP	γ^*
0	2.8429
1	1.0308
2	1.0108
3	1.0108
4	1.0108
5	1.0108

i.e., for parameter-dependent LMI conditions whose entries are algebraic manipulations of homogeneous polynomials of generic degree with parameters in the unit simplex or in known intervals. The toolbox is under constant evolution and some new features are to be implemented. Suggestions of improvements and bug reports are welcome and can be sent to the email `agulhari@utfpr.edu.br`.

References

[1] E. D. Andersen and K. D. Andersen. The MOSEK interior point optimizer for linear programming: An implementation of the homogeneous algorithm. In H. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, volume 33 of *Applied Optimization*, pages 197–232. Springer US, 2000. `http://www.mosek.com`.

[2] P.-A. Bliman, R. C. L. F. Oliveira, V. F. Montagner, and P. L. D. Peres. Existence of homogeneous polynomial solutions for parameter-dependent linear matrix inequalities with parameters in the simplex. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1486–1491, San Diego, CA, USA, December 2006.

[3] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM Studies in Applied Mathematics, Philadelphia, PA, 1994.

[4] M. Chamanbaz, F. Dabbene, D. Peaucelle, and R. Tempo. R-RoMulOC: a unified tool for randomized and robust multiobjective control. In *IFAC Symposium on Robust Control Design*, Bratislava, July 2015.

[5] G. Chesi, A. Garulli, A. Tesi, and A. Vicino. Robust stability of time-varying polytopic systems via parameter-dependent homogeneous Lyapunov functions. *Automatica*, 43(2):309–316, February 2007.

[6] P. Gahinet, A. Nemirovskii, A. J. Laub, and M. Chilali. *LMI Control Toolbox User's Guide*. The Math Works, Natick, MA, 1995.

[7] J. C. Geromel and P. Colaneri. Robust stability of time varying polytopic systems. *Systems & Control Letters*, 55(1):81–85, January 2006.

[8] D. Henrion and J. B. Lasserre. GloptiPoly: global optimization over polynomials with Matlab and SeDuMi. *ACM Transactions on Mathematical Software*, 29(2):165–194, June 2003.

[9] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the 2004 IEEE International Symposium on Computer Aided Control Systems Design*, pages 284–289, Taipei, Taiwan, September 2004. `http://control.ee.ethz.ch/~joloef/yalmip.php`.

[10] R. C. L. F. Oliveira and P. L. D. Peres. LMI conditions for robust stability analysis based on polynomially parameter-dependent Lyapunov functions. *Systems & Control Letters*, 55(1):52–61, January 2006.

[11] R. C. L. F. Oliveira and P. L. D. Peres. Parameter-dependent LMIs in robust analysis: Characterization of homogeneous polynomially parameter-dependent solutions via LMI relaxations. *IEEE Transactions on Automatic Control*, 52(7):1334–1340, July 2007.

[12] R. C. L. F. Oliveira and P. L. D. Peres. A convex optimization procedure to compute $\mathcal{H}_2$ and $\mathcal{H}_\infty$ norms for uncertain linear systems in polytopic domains. *Optimal Control Applications and Methods*, 29(4):295–312, July/August 2008.

[13] D. Peaucelle and D. Arzelier. Robust multi-objective control toolbox. In *Proceedings of the CACSD Conference*, Munich, Germany, 2006.

[14] D. Peaucelle, A. Tremba, D. Arzelier, A. Bortott, G.C. Calafiore, G. Chevarria, F. Dabbene, E. Gryazina, B.T. Polyak, P.S. Shcherbakov, M. Sevin, Ph. Spiesser, and R. Tempo. R-romuloc : Randomized and robust multi-objective control toolbox, January 2014.

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

[16] D. C. W. Ramos and P. L. D. Peres. A less conservative LMI condition for the robust stability of discrete-time uncertain systems. *Systems & Control Letters*, 43(5):371–378, August 2001.

[17] D. C. W. Ramos and P. L. D. Peres. An LMI condition for the robust stability of uncertain continuous-time linear systems. *IEEE Transactions on Automatic Control*, 47(4):675–678, April 2002.

[18] C. W. Scherer. Higher-order relaxations for robust LMI problems with verifications for exactness. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 4652–4657, Maui, HI, USA, December 2003.

[19] C. W. Scherer. Relaxations for robust linear matrix inequality problems with verifications for exactness. *SIAM Journal on Matrix Analysis and Applications*, 27(2):365–395, June 2005.

[20] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1–4):625–653, 1999. `http://sedumi.mcmaster.ca/`.

[21] K. C. Toh, M. J. Todd, and R. Tütüncü. SDPT3 — A Matlab software package for semidefinite programming, Version 1.3. *Optimization Methods and Software*, 11(1):545–581, 1999.

[22] A. Tremba, G.C. Calafiore, F. Dabbene, E. Gryazina, B.T. Polyak, P.S. Shcherbakov, and R. Tempo. RACT: Randomized Algorithms Control Toolbox for MATLAB. In *Proceedings 17th IFAC World Congress, Seoul*, 2008.