

The Robust LMI Parser: A Toolbox to Construct LMI Conditions for Uncertain Systems

Version 3.0

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

April 19, 2016

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 on the unit simplex (or multi-simplex). 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 based 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, a LMI solver is applied to find a solution (if any), for example SeDuMi [18], SDPT3 [19] 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, 20], a toolbox for the manipulation and resolution of conditions related to robust multi-objective control.

Consider, for instance, the problem of analyzing the stability of a discrete-time linear system given by

$$x(k+1) = Ax(k), \quad (1)$$

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

$$\begin{bmatrix} P & A'P \\ PA & P \end{bmatrix} > 0 \quad (2)$$

*Federal University of Technology of Parana – UTFPR, Campus Cornélio Procopio, 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

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 condition (2) as

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

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

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

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

$$\Delta_N = \left\{ \alpha \in \mathbb{R}^N : \sum_{i=1}^N \alpha_i = 1, \quad \alpha_i \geq 0, \quad i = 1, \dots, N \right\}. \quad (5)$$

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 ,

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

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.,

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

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

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

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

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

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. 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 Parser¹, 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.

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

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, so Matlab routines that already use ROLMIP will keep working.

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 is still part of ROLMIP, but it is 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 Preliminaries

The Robust LMI Parser 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 `>>`.

3 Defining the polynomial structure

All polynomial variable, regardless if dependent on (multi) simplex parameters or dependent on parameters with known bounds, are defined as variables of type `rolmipvar`. The variable definition may be performed using several different syntaxes² depending on how the polynomial is described by the user, as detailed in the following.

If the polynomial is composed by known matrices (as opposed to `sdpvar`³ 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 $g = 1$. There are three ways of inputting the coefficients A_1 and A_2 through the 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, on 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)
```

Generically, the exponent of the monomial $\alpha_1^{n_1} \alpha_2^{n_2} \dots \alpha_N^{n_N}$ is encoded as $[n_1 \ n_2 \ \dots \ n_N]$. 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 L^AT_EX code of a polynomial, as presented in the next section. 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.

NOTE: 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.

²The syntaxes are somewhat similar to the `poly_struct` command available on prior versions.

³Variables used on YALMIP toolbox.

Example 1

A polynomial matrix $A(\alpha)$ with degree $g = 1$ that represents a 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 Matlab commands.

```
>> A = [A1 A2 A3];  
>> poly_A = rolmipvar(A,'A',3,1);  
To retrieve the monomial  $A_2$ , 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 Matlab 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 monomial A_4 , for example, it suffices to type >> A4 = poly_A([2 0],[0 1 0]);

NOTE: In the Example 2, two simplexes were used: α and β . Note that, when defining the structure of $A\{\cdot\}$, 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 could occur. For ease of notation, in this manual the simplexes are sorted, by default, in the order $\{\alpha, \beta, \gamma, \delta, \varepsilon\}$.

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, on Example 2, if the monomial with the term $\alpha_1^2 \beta_2$ was to be set to the identity I , one might enter the command >> poly_A([2 0],[0 1 0]) = I;

If the coefficients of the monomials are **sdpvar** variables, one may use the following syntax

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

The matrix coefficients **M** with dimensions **rows** \times **cols** will be internally declared, and returned as a polynomial of **sdpvar** variables to the output **poly**. 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'). The matrices will be declared as symmetric if **param** is not informed. Note that such argument is similar to the **sdpvar** instruction used to define the variables in the YALMIP parser.

If the variable is a scalar 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 variable, the following syntax may be used

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

which returns a scalar variable **M**.

Example 4

A symmetric polynomial variable $P(\alpha) \in \mathbb{R}^{3 \times 3}$ of degree 2, used in an uncertain system with $N = 3$ vertices, 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

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

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 on the parameters that accompany them, defined in a similar way as (11). The input **label** informs the label of the variable, and **bounds** is an $m \times 2$ vector that contains the bounds of the m parameters considered.

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 Matlab 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]);
```

If the coefficients are **sdpvar** variables, one may use the syntax

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

The matrix coefficients **M** with dimensions **rows** \times **cols** will be internally declared, and returned as a polynomial of **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**'). The matrices will be declared as symmetric if **parametr** is not informed. 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 known-bounded parameters 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

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

and then the polynomial is homogenized.

4 Operating on polynomials

Once defined the `rolmipvar` variables, the most common operations may be performed between polynomials or between standard 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;
```

results on 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;
```

results on a polynomial given by

$$A(\alpha)B(\alpha) = \alpha_1^2(A_1B_1) + \alpha_1\alpha_2(A_1B_2 + A_2B_1) + \alpha_2^2(A_2 + B_2).$$

Finally, operations between polynomials and non-polynomial matrices are performed as expected. Let C be a matrix defined on Matlab. The operation

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

results on

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

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 every monomial;

- Command **blkdiag**: used if one needs to construct a block-diagonal matrix, whose diagonals are composed by **rolmipvar** variables;
- 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 accompany each monomial, 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]);
```

results on 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)

will 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 3], [3 4]);
```

results on the polynomial `polyout` that represents

$$A(\gamma, \beta, \delta) = \gamma_1 \beta_1 \delta_1 A_1 + \gamma_2 \beta_1 \delta_2 A_2 + \gamma_1 \beta_2 \delta_3 A_3 + \gamma_2 \beta_2 \delta_1 A_4.$$

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 systems (LPV), where the parameters may be time-varying, the analysis and synthesis LMI conditions depend on the time-derivative of some parameter-dependent matrix variables. For example, the continuous-time LPV system

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

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

$$\dot{P}(\alpha(t)) = \hat{P}(\beta) = \sum_{i=1}^M \beta_i \hat{P}_i, \quad \beta \in \Delta_M.$$

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 will be given by `labelout`⁴, and `dotbounds` contains the bounds of the variation rates of the parameters. Note that, without knowing such bounds, it is not possible to generate a polytope where the vector $\alpha(t)$ 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

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

The derivative of the polynomial is obtained by the command

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

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

⁴If the input `labelout` is not informed, then the label of the output polynomial will be the string `dot_` concatenated with the label of the input polynomial.

```
>> 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

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

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 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 needed. 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 L^AT_EX code

$A'P + PA$.

Finally, the command

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

returns the L^AT_EX 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}).
```

5 Composing matrices and LMIs

The construction of matrices of polynomials and LMIs using ROLMIP is done in an intuitively way, just as in YALMIP⁵. 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 variables Ai and Pi , respectively. The LMI is defined using

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

Any necessary degree homogenization is automatically performed by ROLMIP. The built-in MATLAB command `blkdiag` is also implemented on ROLMIP.

In the following subsection, a full example is presented for a better understanding of ROLMIP.

6 Guaranteed \mathcal{H}_{∞} Cost Computation

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

$$\begin{cases} \dot{x}(t) = A(\alpha)x(t) + B(\alpha)w(t) \\ y(t) = C(\alpha)x(t) + D(\alpha)w(t) \end{cases} \quad (12)$$

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

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

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}_{∞} 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

$$\begin{bmatrix} A(\alpha)'P(\alpha) + P(\alpha)A(\alpha) + C(\alpha)'C(\alpha) & \star \\ B(\alpha)'P(\alpha) + D(\alpha)'C(\alpha) & D(\alpha)'D(\alpha) - \gamma^2 \mathbf{I} \end{bmatrix} < 0, \quad \forall \alpha \in \Delta_N \quad (13)$$

⁵On 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.

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

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

with

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

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

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

and the matrices can be rewritten as

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

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

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

```
degP = 1;
A{1} = {[0 0 0],[0 0 1 0; 0 0 0 1; 0 0 0 0; 0 0 0 0]};
A{2} = {[1 0 0],[0 0 0 0; 0 0 0 0; -2 1 0 0; 0 0 0 0]};
A{3} = {[1 0 1],[0 0 0 0; 0 0 0 0; 0 0 -1 0; 0 0 0 0]};
A{4} = {[0 1 0],[0 0 0 0; 0 0 0 0; 0 0 0 0; 1 -1 0 0]};
A{5} = {[0 1 1],[0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 -1]};
Ai = rolmipvar(A,'A',[2/3 2; 0.8 4/3; 1 3]);
Bi = [1 0 0],[0;0;1;0];
Bi = rolmipvar(B,'B',[2/3 2]);
Ci = rolmipvar([0 1 0 0],'C',0,0);
Di = rolmipvar(0,'D',0,0);
Pi = rolmipvar(4,4,'P','sym',[2 2 2],[degP degP degP]);
mu = sdpvar(1,1);
T11 = Ai'*Pi + Pi*Ai + Ci'*Ci;
T21 = Bi'*Pi + Di'*Ci;
T22 = Di'*Di - mu*eye(1);
T = [T11 T21'; T21 T22];
LMIs = [LMIs, T < 0];
solvesdp(LMIs,mu);
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}_∞ guaranteed cost. Table 1 shows the values of $\gamma^* = \min \gamma$ subject to (13) obtained when the degree of the variable $P(\alpha)$ increases. With $g = 2$, γ^* reaches the worst case value of the \mathcal{H}_∞ 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

7 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, i.e., for parameter-dependent LMI conditions whose entries are algebraic manipulations of homogeneous polynomials of generic degree with parameters in the unit simplex. 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}_∞ 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] 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/>.
- [19] 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.
- [20] 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.