User's Guide for SeDuMi Interface 1.01: Solving LMI problems with SeDuMi

Dimitri Peaucelle - Didier Henrion - Yann Labit
LAAS - CNRS
7, Avenue du Colonel Roche - 31077 Toulouse cedex - France
Tel. 05 61 33 64 17 fax: 05 61 33 69 69
email: peaucelle@laas.fr - henrion@laas.fr - ylabit@laas.fr

2nd November 2001

Abstract

This report describes a user-friendly Matlab package for defining Linear Matrix Inequality (LMI) problems. It acts as an interface for the Self-Dual-Minimisation package (SeDuMi) developed by Jos F. Sturm.

The functionalities of SeDuMi Interface are the following:

- Declare an LMI problem.
  Five Matlab functions allow to define completely an LMI problem which can be characterised by variables, inequality constraints and a linear objective:
  - Initialise the LMI problem: `sdmpb`.
  - Declare the matrix variables: `sdmvar`.
  - Declare the inequality constraints: `sdmlmi` and `sdminequ`.
  - Declare the linear objective: `sdmobj`.
- Solve an LMI problem.
  A unique function, `sdmsol`, calls the SeDuMi solver. Options allow to tune the solver parameters.
- Modify an LMI problem.
  At any moment it is possible to append an LMI problem by adding variables, inequalities or linear terms to the objective. Moreover, the `sdmset` function allows to freeze matrix variables to specified values.
- Analyse the solution issued from the solver.
  For all (feasible or not) problems, the solver outputs the last computed iterate (`sdmget`). SeDuMi Interface allows to analyse this result in a convivial display. The solution is displayed directly in matrix format and indicators show which inequality constraints are satisfied.

Contents

1 Purpose

The tool described in this report is designed to associate both efficient Semi-Definite Programming (SDP) algorithms and the nice Linear Matrix Inequality (LMI) formalism used for control applications. This work was inspired by the observation that on the one hand LMIs have a major position in current academic research [6, 9, 23], and on the other hand there are new promising tools for solving relatively large-scale SDP problems [17]. But there are few tools that associate both an efficient SDP solver and a pleasant interface for declaring LMIs within the most commonly used software environment: MATLAB.

SEDUMI INTERFACE is designed as an add-on for MATLAB and allows to declare LMI problems to be solved with the SEDUMI solver proposed by Jos Sturm [21]. The major part of this report is a description of SEDUMI INTERFACE functions. Before that, we expose the choices that lead us to choose the SEDUMI solver and an interface much alike the LMI Control Toolbox for MATLAB [12].

The user mostly interested in using the interface can skip the remaining part of this section and go directly to section 2.

1.1 Notations

$\mathbb{R}^{m\times n}$ is the set of m-by-n real matrices. All matrices are written using capital letters (A) while scalars and vectors are in lowercase (a).

A' is the transpose of the matrix A.

'sym' is the matrix operator such that: $\mathrm{sym}\{A\} = A + A'$.

$\mathbb{1}$ and $\mathbb{0}$ are respectively the identity and the zero matrices of appropriate dimensions.

For symmetric matrices, $>$ ($\geq$) is the Löwner partial order, i.e., $A > (\geq) B$ if and only if $A - B$ is positive (semi) definite.

In matrix inequalities as well as in optimisation problems, the decision variables and unknowns are in bold face ($\mathbf{x}$) while the data is written using the usual mathematic fonts (x).

1.2 LMI problems

The academic results on LMI use the following optimisation formalism [6, 9]:

$$p^* = \min \quad c'\mathbf{x} \qquad s.t. \qquad F_0 + \sum_{i=1}^{m} \mathbf{x_i} F_i \geq \mathbb{0} \tag{1}$$

where the data are the $m+1$ symmetric matrices F_0, ..., F_m and the column vector c, while the optimisation variables are gathered in the vector $\mathbf{x}$ with components $\mathbf{x_i}$. The formalism underlines that an LMI problem is an optimisation problem with a linear objective and positive semi-definite constraints involving symmetric matrices that are affine in the decision variables.

At this point note that there is another general formalism for writing LMIs [21]:

$$p^* = \min \quad c'\mathbf{x} \qquad s.t. \qquad b - A\mathbf{x} \text{ is positive semi-definite}$$

Here the data are two vectors b, c and a matrix A. The expression is of course abusive. A vector $z = b - A\mathbf{x}$ is said to be positive semi-definite if the symmetric matrix Z, build out of z with some stack operator, is positive semi-definite.

We do not get into more details. The point is that the generic formalisms of LMIs on which are based SDP solvers are much too compact to be adapted easily to application problems. In particular, a major difficulty is that control problems are formulated with matrix variables while the generic formulations exposed above depend on vectors of decision variables. Going from one formalism to another may be quite tedious.

Take for example the Lyapunov inequality, it writes as an LMI constraint:

$$A'\mathbf{P} + \mathbf{P}A < \mathbb{0}$$

Assume A is a 2-by-2 matrix. $\mathbf{P}$ is a symmetric matrix considered as the variable in the LMI problem. Making appear the scalar data and variables, the LMI writes also as:

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} \mathbf{p_1} & \mathbf{p_2} \\ \mathbf{p_2} & \mathbf{p_3} \end{bmatrix} + \begin{bmatrix} \mathbf{p_1} & \mathbf{p_2} \\ \mathbf{p_2} & \mathbf{p_3} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} < \mathbb{0}$$

and this corresponds in the formalism (1) to:

$$F_0 = \mathbb{0} \qquad \mathbf{x}' = (\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3})$$

$$F_1 = \begin{bmatrix} -2a_{11} & -a_{12} \\ -a_{12} & 0 \end{bmatrix} \quad F_2 = \begin{bmatrix} -2a_{21} & -a_{11} - a_{22} \\ -a_{11} - a_{22} & -2a_{12} \end{bmatrix} \quad F_3 = \begin{bmatrix} 0 & -a_{21} \\ -a_{21} & -2a_{22} \end{bmatrix}$$

The manipulations are trivial but tedious even for small size problems. SeDuMi Interface is designed to tackle these manipulations. In particular, the syntax adopted in SeDuMi Interface is adapted to usual control LMI formulations. A large spectrum of options allows to simplify some recurrent declarations:

- First, a large variety of matrix variables are admissible in SeDuMi Interface. This goes from full block matrices to any structured matrix variable including the symmetric and anti-symmetric variables.
- Second, various types of LMI terms can be declared going up to Kronecker products between data matrices and matrix variables.
- Third, the linear objective is a sum of linear terms and can contain the trace operator.

1.3 Existing solvers

Several research groups have produced software packages for SDP problems and many improvements are currently made. Among the first solvers were the SP solver [24] and the LMI-lab which evolved into the Matlab LMI Control Toolbox [12]. Since then, a wide variety of solvers have been developed among which: SeDuMi [21], SDPA [11], SDPHA [7], SDPpack [1], SDPT3 [22], CSDP [5, 4], CUTSDP [15], DSDP [3, 2]. This list is not exhaustive. All solvers have particularities. They have their own SDP formalism, their options, their potentialities, their convergence speed, their robustness... For a recent comparison of these solvers see [17].

Having ourself tested some of the solvers and in view of the report [17], we came to the choice of SeDuMi. The advantages of this solver that guided our choice are:

- Speed. The convergence speed of SeDuMi on the problems we tested is attractive compared to other solvers. This is of major interest when solving large scale problems or when implementing LMI-based iterative algorithms as in [10, 14, 18, 13, 16].

- Sparse format. SeDuMi takes the SDP problem data in sparse format. Therefore, the disk space memory needed for defining problems is reduced in the case of structured data. The results are satisfying for automatic control problems for which most of the data are sparse.

- Large scale problems. This remark is closely related to the two previous ones. It appears that SeDuMi is quite competitive for medium-size problems and can solve relatively large-scale problems.

- Matlab. Most of the researchers in the control community are used to work with Matlab. It is therefore attractive to have a tool that can be used within the Matlab environment.

- Free software. SeDuMi is developed with an open source free software policy as well as SeDuMi Interface. We hope this will encourage the scientific community to support and follow up this initiative.

- Potentialities. In SeDuMi Interface we mainly took advantage of SDP programming. But SeDuMi has other potentialities. It can deal simultaneously with linear programming and quadratic cones. Moreover, complex valued data and variables can be defined. All these potentialities will be integrated to SeDuMi Interface in the future, depending on eventual feedback remarks.

1.4 Existing Interfaces

Quite few LMI interfaces for SDP solvers are available. The most famous is the software in the LMI Control Toolbox [12]. Then comes the sdpsol software [25], which is associated to the SP solver [24], and the LMITOOL package [8], which calls three different solvers: SP [24], SDPHA [7] and SDPPack [1]. All three interfaces work with Matlab.

Except the plurality of the solvers, the difference between these three interfaces is in the way LMIs are declared. They all adopt different formalisms. LMITOOL is purely a graphical user interface (GUI) tool and is quite nice and convivial. As a by-product, the transformation of data into the various solver formalisms is quite slow. The LMI problems are often faster solved than converted to the convenient format. For the sdpsol interface, the speed results are less patent. The LMIs are declared in a text file in a natural way. They are then interpreted by the interface. The speed of the interpretation is more efficient than for LMITOOL but still is not convenient for large scale problems. At last, a GUI is also available in the LMI Control Toolbox [12], but it is less convivial than the two former tools and similar low speed complications are noticed.

We therefore chose not to develop such a GUI tool. First, we believe such tools are necessarily quite slow. The second reason is that we do not have the required programming skills. The chosen framework is an in-line declaration of the LMI problems as in the LMI Control Toolbox [12]. The result is less convivial than a GUI but allows more flexibility. Nevertheless, efforts where made on nice display and some complications we noticed in the LMI Control Toolbox do not occur in SeDuMi Interface.

At the time we finished this report we were informed about a closely related work [20]. It proposes a translator that allows to call various solvers, among which SeDuMi, using the interface of the LMI Control Toolbox. Comparisons with SeDuMi Interface will be done in the future.

1.5 Functionalities

The functionalities of SeDuMi Interface are the following:

- Declare an LMI problem.
  Five Matlab functions allow to define completely an LMI problem characterised by variables, inequality constraints and a linear objective:
  - Initialise the LMI problem: `sdmpb`.
  - Declare the matrix variables: `sdmvar`.
  - Declare the inequality constraints: `sdmlmi` and `sdminequ`.
  - Declare the linear objective: `sdmobj`.
- Solve an LMI problem.
  A unique function, `sdmsol`, calls the SeDuMi solver. Options allow to tune the solver parameters.
- Modify an LMI problem.
  At any moment it is possible to append an LMI problem by adding variables, inequalities or linear terms to the objective. Moreover, the `sdmset` function allows to freeze matrix variables to specified values.
- Analyse the solution issued from the solver.
  For all (feasible or not) problems, the solver outputs the last computed iterate (`sdmget`). SeDuMi Interface allows to analyse this result in a convivial display. The solution is displayed directly in matrix format and indicators show which inequality constraints are satisfied.

1.6 Installing SeDuMi Interface

SeDuMi Interface is composed of simple Matlab files (`sdm***.m`) gathered in a directory, `SeDuMiInt101`, and a subdirectory, `@sdmpb`. The first directory contains:

- `sdmguide.ps` : this report.
- `COPYING` : the GNU general public licence.
- `Contents.m` : the help file for SeDuMi Interface.
- `sdmdemo.m` : the demonstration Matlab file.
- `sdmupq.m, sdmvec.m, sdmmat.m` : three files called by some of the SeDuMi Interface operators.

The subdirectory, `@sdmpb`, contains the 12 essential operators for LMI problem declaration.

The interface works with Matlab version 5 or more and with SeDuMi versions 1.04 and 1.05. We assume that one of these two versions of the software is installed on your computer. If it is not the case, refer to the instructions at `http://fewcal.kub.nl/sturm/`. To install the SeDuMi Interface, you only have to link it to your Matlab path. In the Matlab environment this can be done with the following command:

```
>> path(path, 'TheDirectoryWhereIputIt/SeDuMiInt101');
```

Then test if SeDuMi Interface and SeDuMi are properly by typing:

```
>> sdmdemo
```

A demonstration example of SeDuMi Interface is then run. It describes the declaration of a H_2/H_∞ state-feedback problem from control theory. The example is run for a random problem of large size. It demonstrates the nice convergence speed of SeDuMi.

2 Using SeDuMi Interface - Getting started

2.1 LMI problem: an sdmpb object

An LMI problem within SeDuMi Interface is described by a single Matlab variable. It contains all the information on the matrix variables, the inequality constraints, the linear objective and the optimal solution. The various fields containing this information are assigned as the LMI problem is declared and then solved. We do not enter here in the detail of the structure of this object. The user cannot access directly this class of variables but can operate on it in order to get some data or append the object. It is defined within Matlab as a new class of objects called `sdmpb`. To guide the user of SeDuMi Interface we propose a helpful display of `sdmpb` objects that allows to get at every step important information about the LMI problem. In addition the function `sdmpb/get` allows to retrieve any data of the problem. The various possibilities of `sdmpb/get` are described in the sequel as we expose step by step the usage of SeDuMi Interface functions.

To conduce this tutorial we choose a standard control problem of static state feedback synthesis for Linear Time Invariant (LTI) systems with an H_∞ objective, [6]. Let Σ be an LTI system and K a state-feedback gain such that:

$$\Sigma \ : \ \left\{ \begin{array}{l} \dot{x}(t) = Ax(t) + B_u u(t) + B_w w(t) \\ z(t) = C_z x(t) + D_{zu} u(t) \end{array} \right. \qquad\qquad u(t) = Kx(t)$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $w \in \mathbb{R}^q$ and $z \in \mathbb{R}^p$. The system Σ is stabilisable via a state-feedback gain $K = YQ^{-1}$ and the closed loop transfer from w to z has an H_∞ norm less than γ if and only if:

$$\left\{ \begin{array}{l} \mathbf{Q} = \mathbf{Q}' > \mathbb{1} \\ \left[\begin{array}{cc} A\mathbf{Q} + \mathbf{Q}A' + B_u\mathbf{Y} + \mathbf{Y}'B_u' + B_wB_w' & \mathbf{Q}C_z' + \mathbf{Y}'D_{zu}' \\ C_z\mathbf{Q} + D_{zu}\mathbf{Y} & -\boldsymbol{\gamma}^2\,\mathbb{1} \end{array} \right] < \mathbb{0} \end{array} \right.$$

The optimal synthesis problem is an LMI problem composed of three variables, a linear objective and two inequality constraints. It can write as:

variables:	$\mathbf{Q} = \mathbf{Q}' \in \mathbb{R}^{n\times n}, \quad \mathbf{Y} \in \mathbb{R}^{m\times n}, \quad \boldsymbol{\gamma}^2 \in \mathbb{R}$
inequalities:	$\mathbb{1} < \mathbf{Q}$ $\mathrm{sym}\left\{ \left[\begin{array}{c} A \\ C_z \end{array} \right] \mathbf{Q}E_1' + \left[\begin{array}{c} B_u \\ D_{zu} \end{array} \right] \mathbf{Y}E_1' \right\} + E_1B_wB_w'E_1' - \boldsymbol{\gamma}^2\,E_2E_2' < \mathbb{0}$
objective:	$\max \ -\boldsymbol{\gamma}^2$

where the matrices E_1 and E_2 are defined as:

$$E_1 = \begin{bmatrix} \mathbb{1}_n \\ \mathbb{0}_{p\times n} \end{bmatrix} \qquad E_2 = \begin{bmatrix} \mathbb{0}_{n\times p} \\ \mathbb{1}_p \end{bmatrix}$$

This way of writing the LMI problem differs slightly from the usual notations. It corresponds to the logic used in SeDuMi Interface to declare the LMI problem. In the sequel this point is carefully described.

But first let us initialise the `sdmpb` object within Matlab and give it a name:

```
>> lmi=sdmpb('Optimal Hinfty State Feedback Synthesis')
LMI problem: Optimal Hinfty State Feedback Synthesis
no matrix variable
no inequality constraint
no linear objective
unsolved
```

The operator `sdmpb` creates an empty object. An optional input argument allows to give a label to the LMI problem. This label is used only for nice display.

At this step we have declared an empty `sdmpb` object. The class of the variable `lmi` in Matlab is `sdmpb`. Each declared LMI problem is an `sdmpb object`:

```
>> whos lmi
  Name      Size          Bytes  Class
  lmi       1x1            3358  sdmpb object
```

In the following subsections we assume that the data `A`, `Bu`, `Bw`, `Cz`, `Dzw`, `n`, `m`, `q`, `p`, `E1`, `E2`, describing the LTI system are defined in the Matlab environment. For example, we chose for this tutorial:

```
>> n=4; m=1; q=1; p=2;
>> A = [ 1 0 0 -1 ; 0 -1 1 0 ; -1 0 1 0 ; 0 -1 0 1 ];
>> Bu= [ 1 ; 0 ; 0 ; 0 ];
>> Bw= [ 0 ; 1 ; 0 ; 0 ];
>> Cz= [ 0 0 0 1 ; 1 0 0 1 ];
>> Dzu=[ 1 ; 0 ];
>> E1 = [ eye(n); zeros(p,n)];
>> E2 = [ zeros(n,p); eye(p)];
```

2.2 Defining variables: sdmvar

All variables in SeDuMi Interface are real matrix variables. Extensions to complex variables will be possible in future versions of SeDuMi Interface. By default, the variables are full rectangular matrices declared as follows:

```
>> [lmi, Yindex] = sdmvar(lmi, m, n, 'Y');
```

The first input argument of the function is the `sdmpb` object on which the user wants to declare a new matrix variable. The second and third input arguments are respectively the number of rows and the number of columns in the matrix variable. The last input argument is optional. It is a label used mainly for nice explicit display. The output arguments of the `sdmvar` function are first the appended `sdmpb` object, and second an index, `Yindex`, that makes reference to the declared variable. The index is used later on for various purposes. It can be retrieved at any time by the user as follows:

```
>> lmi
LMI problem: Optimal Hinfty State Feedback Synthesis
matrix variables:        index           name
                         1               Y
no inequality constraint
no linear objective
unsolved
```

In this example $\mathbf{Y}$ is the first variable with an index equal to `1`.

For scalar variables the usage of `sdmvar` is identical. Scalar variables are 1-by-1 matrices:

```
>> [lmi, gindex] = sdmvar(lmi, 1, 1, 'gamma^2');
```

In LMI problems, the matrix variables may have some structural properties. In our example $\mathbf{Q}$ is symmetric. The SeDuMi Interface tool makes possible to declare a large variety of structured variables. For this purpose, see the `sdmvar` complete description in the following section of this guide. Here we expose only how to declare symmetric variables:

```
>> [lmi, Qindex] = sdmvar(lmi, n, 's', 'Q');
```

To declare symmetric (therefore square) variables, the user sets the third argument of `sdmvar` to the string `'s'`.

At this stage the LMI problem of the example is characterised by:

```
>> lmi
LMI problem: Optimal Hinfty State Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
no inequality constraint
no linear objective
unsolved
```

The function `sdmpb/get` also allows to get information on the declared variables such as the number of variables:

```
>> get(lmi, 'varnb')
ans =
     3
```

the name of a variable referenced by its index:

```
>> get(lmi, 'varname', gindex)
ans =
gamma^2
```

and also the structure of a variable:

```
>> get(lmi, 'vardec', Qindex)
ans =
     6     7     8     9
     7    10    11    12
     8    11    13    14
     9    12    14    15
```

In the considered example ($n = 4$) the variable $\mathbf{Q}$ is a 4-by-4 symmetric matrix. The array of integers generated here by the `sdmpb/get` function describes this structure. The integers relate the dependence of the matrix variable $\mathbf{Q}$ to the independent scalar decision variables of the matrix inequality canonical form. For more information on this point see the introductive section and the advanced usage of `sdmvar`.

2.3 Defining an inequality constraint: sdmlmi

All matrix inequality constraints in SeDuMi Interface are square symmetric matrices characterised by their number of rows and an optional label. To initialise an inequality constraint the usage is:

```
>> [lmi, c1index] = sdmlmi(lmi, n, 'Q>1');
>> [lmi, c2index] = sdmlmi(lmi, n+p, 'Hinfty state feedback');
```

The first input argument is the `sdmpb` object to append. The second input argument is the size of the matrix inequality constraint. The last input argument is an optional label used for nice display. The appended `sdmpb` object is retured along with an index, `c1index`, that makes reference to the declared constraint.

At this step the LMI problem of the example is composed of 3 variables and 2 constraints:

```
>> lmi
LMI problem: Optimal Hinfty State Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
inequality constraints:  index   meig    name
                         1       -Inf    Q>1
                         2       -Inf    Hinfty state feedback
no linear objective
unsolved
```

To get data on these inequality constraints, such as the number of constraints and their names, use the following syntaxes:

```
>> get(lmi, 'ineqnb')
ans =
     2
>> get(lmi, 'ineqname', c2index)
ans =
Hinfty state feedback
```

2.4 Add a term to an inequality constraint: sdmineq

By default the inequality constraints are empty. When a term is declared, it is added to the existing terms. Iteratively all the terms are therefore declared with the unique function `sdmineq`. The constraints are linear in the matrix variables, hence all terms can write as $L\mathbf{X}R$ where L and R are data matrices and where $\mathbf{X}$ is a matrix variable. To add a term to an inequality the usage is:

```
>> L = [ A ; Cz ];
>> R = E1';
>> lmi = sdmineq(lmi, c2index, Qindex, L, R);
```

The first input argument is the `sdmpb` object to append (which is retrieved as the unique output argument). The second input argument corresponds to the index of the inequality constraint to which the term is added. The third input argument is the index of the variable. The last two input arguments correspond to the left and right multiplying data matrices.

Note that since matrix inequalities are symmetric, the transposed term is automatically added. This is an important feature of SeDuMi Interface. By default, `sdmineq` allways adds the symmetric term $R'\mathbf{X}'L'$ along with the declared term $L\mathbf{X}R$.

To declare a constant term (such as LR for example), the usage of the function is the same at the difference that the variable index (third input argument) is set to zero. Pay attention to the fact that by default the user has to define both left (fourth input argument) and right (fifth input argument) matrices. Moreover, the symmetric term is automatically added: $R'L'$ is added to the term LR.

In our example, the constant term is such as MM' with $M = E_1 B_w$. Beware not to declare it twice! Therefore, divide one of the multiplying data matrices by two:

```
>> L = E1*Bw;
>> R = 0.5*L';
>> lmi = sdmineq(lmi, c2index, 0, L, R);
```

Following these rules the inequality constraints on the example are completely defined by adding the three remaining terms:

```
>> lmi = sdmineq(lmi, c2index, Yindex, [ Bu ; Dzu ], E1');
>> lmi = sdmineq(lmi, c2index, gindex, -0.5*E2, E2');
>> lmi = sdmineq(lmi, c1index, Qindex, -0.5, 1);
>> lmi = sdmineq(lmi, c1index, 0, 0.5, 1);
```

The usage of `sdmineq` described in this section is the most generic and allows to define all possible terms. In SeDuMi Interface, other usages of `sdmineq` are also defined. A complete description of these advanced functionalities can be found in the next section.

2.5 Add a linear term to the objective: sdmobj

By default the objective is empty. The LMI problem is then a feasibility problem. In order to define a maximisation or minimisation problem, a unique function is used: `sdmobj`. Since SeDuMi is an algorithm that maximises the objective under LMI type constraints, the objective defined by `sdmobj` will be maximised. To minimise some objective one has then to maximise its opposite.

As for inequality constraints, the objective is recursively declared by adding linear objective terms. All terms write in a generic manner as $e_l \mathbf{X} e_r$ where e_l and e_r are respectively a row and a column vector so that the product $e_l \mathbf{X} e_r$ defines a scalar linearly depending on the matrix variable $\mathbf{X}$. For the H_∞ state feedback example of this section the objective is declared as:

```
>> lmi = sdmobj(lmi, gindex, -1, 1, '-gamma^2');
```

The second input argument is the index of the variable on which depends the objective term.

The third and fourth input arguments are respectively the left and right multiplying vectors e_l and e_r. The last input argument is an optional label describing the objective.

As for the `sdmvar` and `sdmineq` operators, an advanced usage of `sdmobj` is available. In particular it allows to declare the trace of some matrix variable as a objective term. The advanced usage of `sdmobj` is described in the next section.

At this stage the LMI problem is entirely declared in the MATLAB environment. The `sdmpb` object contains all the information (see table 1).

```
>> lmi
LMI problem: Optimal Hinfty State Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
inequality constraints:  index   meig    name
                         1       -Inf    Q>1
                         2       -Inf    Hinfty state feedback
maximise objective: -gamma^2
unsolved
```

Table 1: Completely declared LMI problem

The `sdmpb/get` operator makes it possible to get the name of the linear objective:

```
>> get(lmi,'objname')
ans =
 -gamma^2
```

2.6 Solve an LMI problem: sdmsol

Having defined an LMI problem, SEDUMI INTERFACE makes the interface with the SEDUMI solver through the function `sdmsol`. The use is shown in table 2.

The variable containing the LMI problem is then appended and contains the last iterate of the SEDUMI algorithm. SEDUMI runs with the default parameters. To modify these parameters see the advanced description of `sdmsol` in the next section.

When a problem is solved with SEDUMI, the display informs the user on the feasibility of the LMI problem. For the example chosen in this tutorial, the solved LMI problem is displayed in table 3.

An LMI problem solved with SEDUMI may have three status:

- **feasible**
  This means that the maximisation problem was solved and converged towards the optimal point. Here the optimal criteria is `-412`. When no objective is specified, feasibility means that the LMI constraints admit at least one solution found by SEDUMI.

- **infeasible**
  This means that the LMI constraints have no solution at all.

- **marginal feasibility**
  This occurs when SEDUMI has some numerical problems and cannot determine exactly

```
>> lmi = sdmsol(lmi);

SeDuMi 1.05 Development by Jos F. Sturm, 1998, 2001.
Alg = 1: v-corrector, theta = 0.250, beta = 0.500
eqs m = 16, order n = 13, dim = 69, blocks = 4
nnz(A) = 71 + 0, nnz(ADA) = 256, nnz(L) = 136
 it :     b*y       gap   delta  rate  t/maxt   feas cg cg
  0 :             1.30E+01 0.000
  1 :  -1.32E+00 4.61E+00 0.000 0.3547 0.9000  -1.90  1  1
  2 :  -5.42E+00 1.48E+00 0.000 0.3210 0.9000  -0.99  1  1
  3 :  -2.49E+01 3.69E-01 0.000 0.2496 0.9000  -0.86  1  1
  4 :  -6.41E+01 1.54E-01 0.000 0.4165 0.9000  -0.88  1  1
  5 :  -2.50E+02 2.93E-02 0.125 0.1904 0.9000  -0.80  1  1
  6 :  -3.84E+02 9.11E-03 0.000 0.3108 0.9000  -0.01  1  1
  7 :  -4.17E+02 2.38E-03 0.000 0.2615 0.9000   0.61  1  1
  8 :  -4.27E+02 5.38E-04 0.000 0.2257 0.9000   0.85  1  1
  9 :  -4.22E+02 1.64E-04 0.000 0.3049 0.9000   0.85  1  1
 10 :  -4.18E+02 5.60E-05 0.000 0.3415 0.9000   0.72  1  1
 11 :  -4.15E+02 1.76E-05 0.000 0.3143 0.9000   0.71  1  1
 12 :  -4.13E+02 5.40E-06 0.000 0.3071 0.9000   0.74  1  1
 13 :  -4.13E+02 1.93E-06 0.000 0.3570 0.9000   0.66  2  2
 14 :  -4.12E+02 7.43E-07 0.000 0.3853 0.9000   0.75  2  2
 15 :  -4.12E+02 5.66E-08 0.000 0.0761 0.9900   0.96  1  2
 16 :  -4.12E+02 1.04E-08 0.000 0.1841 0.9000   0.98  2  2
 17 :  -4.12E+02 2.73E-10 0.000 0.0262 0.9900   1.00  2  2
 18 :  -4.12E+02 2.48E-13 0.000 0.0009 0.9999   1.00  3  3
iter seconds digits       c*x               b*y
 18      1.4   Inf -4.1191818869e+02 -4.1191818867e+02
|Ax-b| =   1.2e-10, [Ay-c]_+ =   0.0E+00, |x|=  4.2e+02, |y|=  4.1e+03
Max-norms: ||b||=1, ||c|| = 1,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 5345.71.
feasible
```

Table 2: Solving an LMI problem with SeDuMi 1.05

```
>> lmi
LMI problem: Optimal Hinfty State Feedback Synthesis

matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
inequality constraints:  index   meig    name
                         1       eps     Q>1
                         2       eps     Hinf state feedback
maximise objective: -gamma^2 = -412
feasible
```

Table 3: Solved LMI problem

may help. Otherwise, the SeDuMi parameters `pars` can also be adjusted.

This information on the feasibility of the LMI problem is also available via the `sdmpb/get` operator:

```
>> get(lmi, 'feas')
ans =
     1
```

The `sdmpb/get` output is `1` if the problem is feasible, `-1` if the problem is infeasible, `0` if the problem is marginal feasible and a string '`unsolved`' if SeDuMi was not executed on this LMI problem.

2.7 Extracting the computed solution

In all cases SeDuMi returns the last iterate. The `sdmpb/get` function in SeDuMi Interface allows the user to get this solution directly in a matrix format. The user specifies `'varvalue'` as a second input argument and provides a third input argument containing the index of the desired matrix:

```
>> g2_opt = get(lmi, 'varvalue', gindex)
g2_opt =
  411.9182
```

A faster way to obtain the values of the variables is the sub-reference with brackets:

```
>> Y_opt = lmi(Yindex)
Y_opt =
 -623.9321 -813.1962  -12.5567 -383.1822
```

This syntax allows to get directly the value of the variable referenced by its index in a matrix format.

The function `sdmpb/get` allows also to obtain the objective value at the optimum:

```
>> get(lmi,'objopt')
ans =
 -411.9182
```

For the specified H_∞ state feedback problem, SeDuMi has found the minimal H_∞ norm achievable by state feedback:

$$\gamma^{opt} = \sqrt{411.9182} = 20.2958$$

2.8 Analysis of the computed solution

To analyse the obtained point, the `meig` data allow to check that every inequality constraint is satisfied. These data are displayed with the LMI problem and can also be obtained with the `get` function as:

```
>> get(lmi, 'ineqmeig', c1index)
ans =
eps
```

The `meig` data correspond to the minimal eigenvalue of each inequality constraint evaluated on the point obtained by SeDuMi. The constraints are satisfied if their minimal eigenvalue is positive. The `meig` data can have different values:

- `-Inf` : occurs when the LMI problem has not been solved.
- `0` : occurs when the constraint is strictly equal to zero (this may happen because in SeDuMi the inequalities are in fact semi-definite).
- `eps` or `-eps` : occurs when the minimal eigenvalue is positive or negative and "close" to zero. The SeDuMi algorithm has an accuracy set by default to 10^{-9}. All eigenvalues with absolute value less than this accuracy level are assumed to be "equal" to zero and set to `eps` or `-eps` in SeDuMi Interface. Even if they are negative (`-eps`), the related inequality constraint is considered to be satisfied.
- positive scalar : occurs when the constraint is strictly satisfied (positive definite).
- negative scalar : the constraint is not satisfied.

We believe that the discussion on semi-definite and definite inequality constraints initiated in this section may confuse some readers. Therefore, for more details we recommend reading the next section, in particular the part on the solver parameters. For less curious readers, we may say that except for badly conditioned problems the indication on feasibility is relevant. "Playing" with the solver parameters almost always confirms this information.

3 Advanced use of SeDuMi Interface

3.1 Structured variables: sdmvar

As exposed briefly in the previous section, the operator `sdmvar` adds a new matrix variable to the LMI problem. Moreover, the operator allows to specify the structure to this variable. Four classical structures are directly available and a fifth usage of `sdmvar` enables to declare any other structure. Before describing the arguments of `sdmvar` for each case, note that SeDuMi Interface represents the structure of a variable as a matrix of the same size with integer elements. This representation illustrates the dependency of the matrix variable elements to the vector of independent decision variables.

For example, take the variables declared in the LMI problem of the previous section. The `sdmpb/get` operator can give their dependency to the decision variables as shown in table 4.

Note that two different notations were used here to get the structure matrices. The notations are equivalent. The second one, composed of curly braces, is a faster manner to get directly the structure of the variable referenced by its index.

When comparing the "structure" matrices one can see that they have the same structure as expected for the variable. The three matrix variables are independent because they depend all on different decision variables: $\mathbf{Y}$ is a 1-by-4 full block matrix that depends on the decision variables indexed from 1 to 4; $\boldsymbol{\gamma}$ is a scalar variable that depends on the 5-th decision variable; $\mathbf{Q}$ is a 4-by-4 full block symmetric matrix that depends on the decision variables indexed from 6 to 15 (10 independent elements in a 4-by-4 symmetric matrix).

The following four ways to declare matrix variables assume that the new variable is independent of the others.

```
>> get(lmi, 'vardec', Yindex)
ans =
     1     2     3     4
>> lmi{gindex}
ans =
     5
>> lmi{Qindex}
ans =
     6     7     8     9
     7    10    11    12
     8    11    13    14
     9    12    14    15
```

Table 4: Get the structure of the variables

- **full rectangular**
  The first input argument to sdmvar is the variable describing the LMI problem (`lmi`); the second and third input arguments are respectively the number of rows (`m`) and columns (`n`) of the rectangular matrix variable; at last, as an option, the user can give a label to the variable (`name`).

```
>> [lmi, VARindex] = sdmvar(lmi, m, n, name);
```

The `sdmvar` operator outputs the appended `sdmpb` object and the index of the created variable. Beware not to confuse this index with the decision variable structure. The index is an integer that makes reference to a matrix variable of the LMI problem, while the integer elements of the structure matrix make reference to scalar decision variables. The variable index `VARindex` is used in the sequel for specifying a matrix variable in other operators of SeDuMi Interface, while the decision variable indexes are only used as arguments for `sdmvar`.

- `d`, **diagonal**
  Square matrix variable with independent entries on the diagonal and zero off-diagonal entries. The usage of `sdmvar` is the same as for full rectangular matrices except for the third input argument that is replaced with the string `'d'`. An example of $\mathbb{R}^3$ diagonal variable is:

```
>> [lmi, Dindex] = sdmvar(lmi, 3, 'd', 'D : diagonal');
>> lmi{Dindex}
ans =
    16     0     0
     0    17     0
     0     0    18
```

- `s`, **symmetric**
  Square symmetric matrix variable. Contains $n(n+1)/2$ independent decision variables when n is the number of rows of the matrix. The third input argument of `sdmvar` is set to the string value `'s'`. An example of symmetric variable is given in the previous section (variable $\mathbf{Q}$).

- `as`, **anti-symmetric**
  Square anti-symmetric matrix variable. Contains $n(n-1)/2$ independent decision variables when n is the number of rows of the matrix. The third argument of `sdmvar` is set to the string value `'as'`. An example of $\mathbb{R}^3$ anti-symmetric variable is:

```
>> [lmi, Gindex] = sdmvar(lmi, 3, 'as', 'G');
>> lmi{Gindex}
ans =
     0   -19   -20
    19     0   -21
    20    21     0
```

At this step, note that the matrix describing the structure of the variables can be composed of positive integers, negative integers and zeros. The rule is the following. Let $\mathbf{X}$ be a matrix variable and `Xdec=lmi{Xindex}` be the matrix that describes its structure:

`Xdec(i,j)= 0` if the $\mathbf{X}_{(i,j)}$ element of $\mathbf{X}$ is equal to zero.

`Xdec(i,j)=+n` if the $\mathbf{X}_{(i,j)}$ element of $\mathbf{X}$ is equal to the n-th decision variable.

`Xdec(i,j)=-n` if the $\mathbf{X}_{(i,j)}$ element of $\mathbf{X}$ is the opposite of the n-th decision variable.

Following this rule any structured variable can be declared:

- `st`, **structured rectangular**
  Rectangular structured matrix variable depending on other decision variables. The second input argument of `sdmvar` must be a matrix of integers describing the structure as exposed above. The third input argument must be set to the string value `'st'`. Some examples follow.

The first example consists in building a new matrix variable that depends only on existing decision variables. Assume that the new variable $\mathbf{F}$ should be rectangular and composed of a diagonal block and an anti-symmetric block as follows:

$$\mathbf{F} = \left[\begin{array}{cc} \mathbf{D} & \mathbf{G} \end{array} \right]$$

where $\mathbf{D}$ and $\mathbf{G}$ are already defined matrix variables. To declare $\mathbf{F}$ the commands are:

```
>> Ddec = lmi{Dindex};
>> Gdec = lmi{Gindex};
>> Fdec = [ Ddec , Gdec ];
>> [lmi, Findex] = sdmvar(lmi, Fdec, 'st', '[D, G]');
>> lmi{Findex}
ans =
    16     0     0     0   -19   -20
     0    17     0    19     0   -21
     0     0    18    20    21     0
```

The second example consists in building a new matrix variable depending on new (not yet declared) decision variables. Assume that the new variable $\mathbf{H}$ should depend on three new independent scalar variables with the following structure:

$$\mathbf{H} = \left[\begin{array}{ccc} h_1 & 0 & h_1 \\ h_3 & h_2 & h_3 \end{array} \right]$$

The procedure to declare this variable is first to get the number of existing decision variables, then to build the structure matrix making reference to the new decision variables and finally to declare the variable using `sdmvar`:

```
>>  get(lmi, 'vardecnb')
ans =
    21
>> Hdec = [22 0 22;24 23 24];
>> [lmi, Hindex] = sdmvar(lmi, Hdec, 'st', 'H');
>> lmi{Hindex}
ans =
    22     0    22
    24    23    24
```

3.2 Advanced declaration of some inequality terms

All terms in an LMI constraint can be declared with `sdmineq` as follows:

```
>> lmi = sdmineq(lmi, Cindex, Xindex, L, R);
```

This command adds to the constraint referenced by the index `Cindex` a term $L\mathbf{X}R + R'\mathbf{X}'L'$ where $\mathbf{X}$ is the matrix variable referenced by the index `Xindex`. If this index is equal to 0 a constant term $LR + R'L'$ is added.

3.2.1 Left and right sides of an inequality

By default, a new declared term is added to the left of the inequality sign $\leq$. For example assume that up to this point some terms have been declared and the inequality constraint writes in a schematic form as:

$$\mathcal{L}(\mathbf{x}) \leq \mathbb{0}$$

The command

```
>> lmi = sdmineq(lmi, Cindex, Xindex, L, R);
```

modifies the constraint into:

$$\mathcal{L}(\mathbf{x}) + L\mathbf{X}R + R'\mathbf{X}'L' \leq \mathbb{0}$$

But SeDuMi Interface allows also to declare terms on the right-hand side. This can be done by multiplying by -1 the constraints index. The command

```
lmi = sdmineq(lmi,-Cindex, Xindex, L, R);
```

modifies the constraint into:

$$\mathcal{L}(\mathbf{x}) \leq L\mathbf{X}R + R'\mathbf{X}'L'$$

The rule for left and right sides of inequality constraints are the following:

- **left**
  If the second input argument of `sdmineq` is positive (`+Cindex`) the term is added in the `Cindex` constraint, to the left-hand side of the inequality sign $\leq$.
- **right**
  If the second input argument of `sdmineq` is negative (`-Cindex`) the term is added in the `Cindex` constraint, to the right-hand side of the inequality sign $\leq$.

Following this rule, the next two command lines are equivalent:

```
>> lmi = sdmineq(lmi, +Cindex, 0, -L, R);

>> lmi = sdmineq(lmi, -Cindex, 0, +L, R);
```

Assume that the constraint before this command was schematically described as $\mathcal{L}(\mathbf{x}) \leq \mathcal{R}(\mathbf{x})$. The two commands append respectively the constraint such that:

$$\mathcal{L}(\mathbf{x}) - LR - R'L' \leq \mathcal{R}(\mathbf{x}) \qquad\qquad \mathcal{L}(\mathbf{x}) \leq \mathcal{R}(\mathbf{x}) + LR + R'L'$$

The two resulting constraints are identical.

3.2.2 Transpose of some matrix variable

By default, a new declared term depends linearly on the matrix variable such as in $L\mathbf{X}R$. The symmetric term $R'\mathbf{X}'L'$ being automatically added, there is no point in declaring terms that depend on the transpose of the matrix variable. Nevertheless SeDuMi Interface allows the user to do so. In order to declare a term depending on the transpose of a matrix variable, such as $L\mathbf{X}'R$, the syntax is to multiply by -1 the third input argument of `sdmineq`.

The rule for matrix variable transpose is the following:

- **not transposed**
  If the third input argument of `sdmineq` is positive (`+Xindex`) the term depends on the variable ($\mathbf{X}$) such as in: $L\mathbf{X}R + R'\mathbf{X}'L'$.
- **transposed**
  If the third input argument of `sdmineq` is negative (`-Xindex`) the term depends on the transpose of the variable ($\mathbf{X}'$) such as in: $L\mathbf{X}'R + R'\mathbf{X}L'$.

Note that not only the transposed version is unnecessary, but moreover we do not recommend its use because it implies extra computation. Following the rule, the two next command lines are equivalent:

```
>> lmi = sdmineq(lmi, Cindex, +Xindex, L, R);

>> lmi = sdmineq(lmi, Cindex, -Xindex, R', L');
```

3.2.3 Matrix terms depending on a scalar variable

A 1-by-1 matrix variable can be confounded with a scalar, but from a mathematical point of view multiplying a 1-by-1 matrix with an other matrix assumes that the dimensions fit together. For example, in the LMI problem proposed in the previous section, $\boldsymbol{\gamma}$ is a scalar and E_2 is a matrix that can have any dimension. The operation $E_2\,\boldsymbol{\gamma}\,E_2'$ cannot be performed if $\boldsymbol{\gamma}$ is seen as a 1-by-1 matrix.

Problems could therefore occur when dealing with scalar variables declared as 1-by-1 matrices. But it is not the case with SeDuMi Interface. The user may proceed without worrying.

3.2.4 Scalar L and R multipliers

For the same reason as exposed in the last paragraph, there could be some trouble when giving scalar values to the matrices L and R (fourth and fifth input arguments of `sdmineq`). But

SeDuMi Interface handles also these cases in the natural way.

3.2.5 Symmetric terms

Noticing that for symmetric terms it is quite tedious to be aware of automatic duplication, another syntax is accepted by `sdmineq`. This syntax is based on the fact that most symmetric terms can be reformulated as:

$$L\mathbf{X}L' \qquad\qquad LL'$$

for variable dependent and constant terms, respectively. In order to declare such symmetric terms the following rule is adopted:

- **not symmetric**
  If both the fourth (L) and the fifth (R) input arguments are declared and are non empty, then the added term is $L\mathbf{X}R + R'\mathbf{X}'L'$ or $LR + R'L'$ (variable dependent term or constant term, respectively).

- **symmetric**
  If both the fourth input argument (L) is specified and the fifth input argument (R) is omitted or empty (`R=[ ]`), then the added term is $L\mathbf{X}L'$ or LL' (variable dependent term or constant term, respectively).

To show that this syntax simplifies a lot some notations, take the constant term in the LMI problem of the previous section ($E_1 B_w B_w' E_1'$). This term can be equivalently declared with the two following syntaxes:

```
>> lmi = sdmineq(lmi, Cindex, 0, E1*Bw, 0.5*Bw'*E1');

>> lmi = sdmineq(lmi, Cindex, 0, E1*Bw);
```

When declaring symmetric terms such as $L\mathbf{X}L'$, the matrix $\mathbf{X}$ is assumed to be symmetric. Otherwise, there might be some complications. In fact, if $\mathbf{X}$ is square non-symmetric, SeDuMi Interface outputs a warning message and implements by default the symmetrised term: $0.5L(\mathbf{X} + \mathbf{X}')L'$.

3.2.6 Terms with no multiplying data

Quite often, some inequality constraints concern directly matrix variables without any dependency on the data of the problem. A famous example is the Lyapunov matrix. In all control problems the Lyapunov matrix $\mathbf{Q}$ is constrained to be definite positive. In the example of the previous section, this specification is translated into $\mathbf{Q} > \mathbb{1}$. To simplify the declaration of such simple terms the following rule is adopted:

- **elementary terms**
  If the fourth and the fifth input arguments of `sdmineq` are omitted, then the added term is equal to the specified matrix variable.

Following this rule and the left-right rule, the declaration of the inequality $\mathbf{Q} > \mathbb{1}$ has the two equivalent syntaxes:

```
>> lmi = sdmineq(lmi, +Cindex, Qindex, -0.5, 1);
>> lmi = sdmineq(lmi, +Cindex, 0, 0.5, 1);

>> lmi = sdmineq(lmi, -Cindex, Qindex);
>> lmi = sdmineq(lmi, +Cindex, 0);
```

3.2.7 Terms with Kronecker products

Sometimes, it may happen that the inequality constraint has one or more terms with a Kronecker product between a matrix variable and a data variable. Such a product is linear in the variables but is quite tedious to implement. It is usually used to simplify the inequality notations but may complicate the programming. Fortunately SeDuMi Interface is designed to tackle the Kronecker products and does it quite fast. Two configurations are considered:

$$L(K \otimes \mathbf{X})R + (L(K \otimes \mathbf{X})R)' \qquad \text{or} \qquad L(\mathbf{X} \otimes K)R + (L(\mathbf{X} \otimes K)R)'$$

To deal with such terms the `sdmineq` operator is called with 6 or 7 input arguments (one or two more input arguments than for the generic usage). The rule is as follows:

- **first Kronecker product** : $K \otimes \mathbf{X}$
  When the `sdmineq` operator is called with a sixth input argument (K), the added term is of the form $L(K \otimes \mathbf{X})R + (L(K \otimes \mathbf{X})R)'$.
- **second Kronecker product** : $\mathbf{X} \otimes K$
  When the `sdmineq` operator is called with a sixth input argument (K) and a seventh input argument set to -1, the added term is of the form $L(\mathbf{X} \otimes K)R + (L(\mathbf{X} \otimes K)R)'$.

To illustrate the use of this rule consider the Lyapunov inequalities assessing Hurwitz or Schur stability of a matrix A, respectively:

$$A'\mathbf{P} + \mathbf{P}A < \mathbb{0} \qquad \text{and} \qquad A'\mathbf{P}A - \mathbf{P} < \mathbb{0}$$

These two inequality constraints share the common expression:

$$\left[\begin{array}{cc} \mathbb{1} & A' \end{array}\right] R \otimes \mathbf{P} \left[\begin{array}{c} \mathbb{1} \\ A \end{array}\right] < \mathbb{0}$$

with $R = \left[\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right]$ and $R = \left[\begin{array}{cc} -1 & 0 \\ 0 & 1 \end{array}\right]$ for Hurwitz and Schur stability, respectively. Applying both the Kronecker product rule and the symmetric matrix rule the inequality constraint is entirely defined for both stability conditions with the unique command:

```
>> lmi = sdmineq(lmi, Cindex, Pindex, [eye(n),A'], [], R);
```

3.3 The trace as an objective

The operator `sdmobj` is designed to declare linear objectives by adding recursively scalar terms of the type $e_l \mathbf{X} e_r$ where e_l and e_r are respectively a row and a column vector. This usage was already defined in the previous section. Here we demonstrate how it can be used to define a linear trace objective. Assume the objective is:

$$\max \;\; 3 \; \mathrm{trace}(\mathbf{X})$$

With the e_l and e_r vectors it is impossible to define this term in one command. Assume $\mathbf{X}$ is a 4-by-4 matrix, the MATLAB script could be:

```
>> for ii=1:4
el=sparse(1,ii,1,1,4);
er=el';
name=sprintf('+3*X(\%i,\%i)',ii,ii);
lmi = sdmobj(lmi, Xindex, 3*el, er, name);
end;
>> get(lmi, 'objname')
ans =
 +3*X(1,1) +3*X(2,2) +3*X(3,3) +3*X(4,4)
```

To simplify such objective declaration SEDUMI INTERFACE can declare directly the trace of a matrix variable. The rule is:

- **generic scalar term**
  If the third and fourth input arguments of `sdmobj` are respectively a row (e_l) and a column (e_r) vector, the term $e_l \mathbf{X} e_r$ is added to the maximisation objective.
- **trace objective**
  If the third and fourth input arguments of `sdmobj` are respectively the string `'tr'` and a scalar (r), the term added to the maximisation objective is : r trace($\mathbf{X}$).

With the help of this syntax, the objective defined recursively above can be declared in one line:

```
>> lmi = sdmobj(lmi, Xindex, 'tr', 3, '3*trace(X)');
>> get(lmi, 'objname')
ans =
 3*trace(X)
```

3.4 Set a value to some variable: sdmset

In some cases the user may want to solve an LMI problem with one (or more) variable frozen to a specified value. This allows to test if some value belongs to the admissible set constrained by the inequalities. The syntax is the following:

```
>> lmi = sdmset(lmi, Xindex, Xvalue);
```

The first input argument is the `sdmpb` object to append; the second input argument is the index of the variable; the third input argument is the value of the variable the user has chosen. Note that `sdmset` does not check whether the value fits the structure of the variable. The function checks only the dimensions of `Xvalue` but allows to give any full block non-symmetric values to structured variables. Having set a variable to some value, the indexes of the other variables are not modified. The inequalities and the objective are rearranged to take into account that the variable becomes a constant. The label of the removed variable is appended as well as the labels of removed inconsistent inequalities.

To illustrate the use of `sdmset` let us consider the H_∞ state feedback problem of the previous section. The LMI problem was defined in order to minimise the H_∞ cost and is displayed by SEDUMI INTERFACE as shown in table 5.

One can expect that since the optimal value is $\gamma^{opt} = \sqrt{412}$, the LMI problem also has a solution if γ is set to a higher value, for example $\gamma = \sqrt{500}$. To check this, set the value of $\boldsymbol{\gamma}$ and solve

```
> lmi
LMI problem: Optimal Hinfty State Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
inequality constraints:  index   meig    name
                         1       eps     Q>1
                         2       eps     Hinfty state feedback
maximise objective: -gamma^2 = -412
feasible
```

Table 5: The original LMI problem

the new LMI problem as shown in table 6.

```
>> lmi2 = sdmset(lmi, gindex, 500);
>> lmi2 = sdmsol(lmi2);
...
>> lmi2
LMI problem: Optimal Hinfty State Feedback Synthesis

matrix variables:        index           name
                         1               Y
                         2               gamma^2 SET TO A CONSTANT
                         3               Q
inequality constraints:  index   meig    name
                         1       0.08    Q>1
                         2       0.05    Hinf state feedback
maximise objective: -gamma^2 = 0
feasible
>> lmi2(gindex)
ans =
    500
```

Table 6: Same LMI problem with a frozen value of $\boldsymbol{\gamma}$

Note that the LMI problem has been transformed from a optimisation problem to a feasibility problem (the objective is then equal to zero). Moreover, note that the variable $\boldsymbol{\gamma}$ was removed from the inequalities but it is still possible to get its fixed value.

For **recursively defined variables** some complications may occur when using `sdmset`. Take as an example the variable $\mathbf{F}$ defined as the concatenation of two previously defined variables:

$$\mathbf{F} = \begin{bmatrix} \mathbf{D} & \mathbf{G} \end{bmatrix}$$

The question is: *What happens to the variable* $\mathbf{F}$ *if the value of* $\mathbf{G}$ *is set to a constant?*

The answer is: *Nothing.*

SeDuMi Interface only modifies the inequalities and the objective that depend explicitly on the variable whose value is set. For recursively defined matrices, there is no modification on the "copies" of the frozen variable. This is illustrated in the above example by the fact that if $\mathbf{G}$ is frozen, its structure matrix becomes empty, while the variable $\mathbf{F}$ still depends on the same decision variables (see table 7).

```
>> lmi2 = sdmset(lmi, Gindex, Gvalue);
>> lmi2{Gindex}
ans =
     []
>> lmi2{Findex}
ans =
    20     0     0     0   -23   -24
     0    21     0    23     0   -25
     0     0    22    24    25     0
```

Table 7: Structure of $\mathbf{G}$ and $\mathbf{F}$ when $\mathbf{G}$ is set to a constant

3.5 Tuning the solver parameters

To illustrate the influence of some parameters on the solution issued from the SeDuMi solver we chose an example for which some complications are noticed. The example is a mixed H_2/H_∞ state feedback synthesis problem. The exact data of this example are not given for conciseness reasons (system of order $n = 10$). When solved without any modification of the default solver parameters the result is shown in table 8.

```
>> lmi = sdmsol(lmi)
...
LMI problem: H2/Hinfty state-feedback synthesis
matrix variables:        index   name
                         1       X
                         2       T
                         3       S
inequality constraints:  index   meig    name
                         1       0.005   X>0
                         2       3e-09   Hinf
                         3       3e-08   gram
                         4       -eps    H2
maximise objective: -trace(T) -g = -3.86
feasible
```

Table 8: Optimally solved LMI problem having accuracy complications

The display claims that the problem is feasible but the `meig` value of the fourth inequality is negative. As exposed in the previous section, this status is not surprising. It indicates that SeDuMi stopped while the computed point was as close to the optimum as the accuracy level.

3.5.1 SeDuMi parameters

To improve the accuracy the user has to specify new parameters to SeDuMi. This is done with the syntax:

```
>> lmi=sdmsol(lmi,pars);
```

The optional second input argument of `sdmsol` must have the structure adopted by SeDuMi [21]. One of the fields of the structure `pars` allows to choose a precision level (default is 10^{-9}). Let us modify in this way the accuracy of SeDuMi and hopefully the obtained iterate will have all eigenvalues strictly positive. Table 9 shows the computation result.

As can be seen, SeDuMi failed to find a feasible solution satisfying the new precision. In fact,

```
>> pars.eps=1e-12;
>> lmi=sdmsol(lmi,pars)
...
LMI problem: H2/Hinfty state-feedback synthesis
matrix variables:        index   name
                         1       X
                         2       T
                         3       S
inequality constraints:  index   meig    name
                         1       0.005   X>0
                         2       3e-09   Hinf
                         3       3e-08   gram
                         4       -7e-10  H2
maximise objective: -trace(T) -g = -3.86
marginal feasible
```

Table 9: Same LMI problem solved with a smaller precision parameter

the last iterate is identical to the previous one. SeDuMi stopped due to numerical problems. The display explicitly warns that the feasibility is not satisfied with the required precision (an eigenvalue is negative and less than -10^{-12}) but the algorithm does not prove that the LMI problem is infeasible. To have details on the status of the obtained iterate, the user can get the output information from SeDuMi with the help of the `sdmpb/get` operator:

```
>> get(lmi,'solver')
ans =
       cpusec: 2.5600
         iter: 17
    feasratio: 0.9403
         pinf: 0
         dinf: 0
       numerr: 1
```

For details on the fields of this output see [21]. The last field `numerr: 1` is non-zero in this example. This explicitly shows that some numerical error has occurred.

3.5.2 Definite and semi-definite inequalities

Up to this point we have discussed how to modify the algorithm parameters using the second argument of `sdmsol` and how to get the status of the SeDuMi solver issued after computation. But we have not yet solved the H_2/H_∞ state feedback synthesis problem. There is still one non-strictly satisfied inequality in the result (the `meig` data of the fourth inequality is negative `-7e-10`).

To explain the difficulty encountered here, note that for SeDuMi the constraints are semi-definite inequalities. Therefore, SeDuMi assumes that if a minimal eigenvalue `meig` of some constraint is equal to zero (or close to zero at the precision level), the constraint is satisfied. In the H_2/H_∞ synthesis problem that is used as an example here, we are therefore disappointed not to have strictly positive eigenvalues while the computed iterate is satisfying for SeDuMi.

To avoid such misunderstandings a way out is then to "transform" all semi-definite inequalities into definite inequalities. To do so, the user has to introduce a constant positive definite term $\alpha\mathbb{1}$ in all inequalities such that:

$$\mathcal{L}(\mathbf{x}) \leq \mathcal{R}(\mathbf{x}) - \alpha\mathbb{1} \qquad \Longrightarrow \qquad \mathcal{L}(\mathbf{x}) < \mathcal{R}(\mathbf{x})$$

This amounts to transform the constraint "*all eigenvalues must be strictly positive*" into "*all values must be superior or equal to* $\alpha > 0$". The modified constraint is conservative but one can expect that the modification is not disturbing as long as α is small.

We now test this procedure on the H_2/H_∞ state feedback synthesis problem. First, one has to add the constant term $\alpha\mathbb{1}$ to all inequalities. This may be tedious, therefore a simplified syntax is adopted in SeDuMi Interface:

- **add a scalar to a LMI problem**
  If a scalar α is added (or subtracted) to a `sdmpb` object, the result of this operation is an identical `sdmpb` object where a constant matrix $\alpha\mathbb{1}$ of appropriate dimension has been added (or subtracted) to the right side of all inequality constraints.

The subtraction operator is applied to the example with $\alpha = 10^{-6}$. The result is given in table 10.

```
>> lmi2 = lmi - 1e-6;

>> lmi2 = sdmsol(lmi2)
...
LMI problem: H2/Hinfty state-feedback synthesis
matrix variables:         index   name
                          1       X
                          2       T
                          3       S
inequality constraints:   index   meig    name
                          1       0.005   X>0
                          2       eps     Hinf
                          3       7e-09   gram
                          4       -eps    H2
maximise objective: -trace(T) -g = -3.86
feasible
```

Table 10: Same LMI problem solved with α-translated constraints

The result is satisfying: all eigenvalues of the constraints in the LMI problem `lmi2` are positive or at least greater than the accuracy level -10^{-9}. This means that the solution is acceptable for the LMI problem `lmi` with all eigenvalues superior or equal to $\alpha - \epsilon = 10^{-6} - 10^{-9} > 0$.

3.5.3 Feasibility radius

Another and last way to tune the solver convergence without modifying strongly the LMI problem, is to impose a feasibility radius. It amounts to constrain the norm of the vector of decision variables. The convexity of the LMI problem is not modified but the additive constraint may add some extra conservatism. The "trick" is to impose a sufficiently large feasibility radius and in the same time reduce at its maximum the domain in which the solver has to seek the optimal solution.

The syntax for constraining the feasibility radius is to give a third input argument to `sdmsol`. Constrained with a radius of 10^9, the mixed H_2/H_∞ state feedback synthesis problem converges to the solution of table 11.

Note that the solution found this time has all its eigenvalues strictly positive. It is a by-product of the feasibility radius. The SeDuMi solver has quite often less numerical problems if a feasibility

```
>> lmi=sdmsol(lmi,[],1e9)
...
LMI problem: H2/Hinfty state-feedback synthesis
matrix variables:        index   name
                         1       X
                         2       T
                         3       S
inequality constraints:  index   meig    name
                         1       0.01    X>0
                         2       6e-06   Hinf
                         3       2e-05   gram
                         4       8e-07   H2
maximise objective: -trace(T) -g = -3.86
feasible
```

Table 11: Same LMI problem solved with a feasibility radius

radius is appropriately chosen. To guide the user in choosing this radius we recommend to solve the LMI problem once without any radius. If the solution is not convenient, get the value of the norm on the decision variable vector using the `sdmpb/get` operator:

```
>> get(lmis,'ynorm')
ans =
   2.7383e+06
```

At last solve again the same problem with a feasibility radius greater than the previously obtained norm. This empirical procedure often gives successful results.

4 Warnings for MATLAB LMI Toolbox users

This section is specially written for users that are familiar with the LMI Control Toolbox of MATLAB. Other readers can skip this section and therefore avoid potentially confusing remarks.

4.1 Block partitioning of an inequality

In contrast with the LMI Toolbox of MATLAB, there is no partitioning of inequality constraints in matrix blocks. This may be confusing for LMI Toolbox adepts and maybe will be implemented in future versions of SEDUMI INTERFACE. Nevertheless, note that this partitioning in blocks is not always useful since it can complicate the definition of inequality constraints. For example, consider a classical inequality constraint for $\mathcal{H}_\infty$ analysis of continuous-time LTI systems:

$$\left[\begin{array}{cc} A'\mathbf{P}+\mathbf{P}A+C'C & \mathbf{P}B+C'D \\ B'\mathbf{P}+D'C & -\gamma\mathbb{1}+D'D \end{array}\right] < \mathbb{0}$$

Using block partitioning this constraint in the LMI Control Toolbox is declared in 6 lines:

```
>> lmiterm([c1, 1, 1, Pindex], 1, A, 's');
>> lmiterm([c1, 1, 1, 0], C'*C);
>> lmiterm([c1, 1, 2, Pindex], 1, B);
>> lmiterm([c1, 1, 2, 0], C'*D);
>> lmiterm([-c1, 2, 2, gindex], 1, 1);
>> lmiterm([c1, 2, 2, 0], D'*D);
```

But the constraint can also be written without the notion of blocks as:

$$sym\left\{\left[\begin{array}{c}\mathbb{1}\\ \mathbb{0}\end{array}\right]\mathbf{P}\left[\begin{array}{cc}A & B\end{array}\right]\right\}+\left[\begin{array}{c}C'\\ D'\end{array}\right]\left[\begin{array}{c}C'\\ D'\end{array}\right]'<\gamma\left[\begin{array}{c}\mathbb{0}\\ \mathbb{1}\end{array}\right]\left[\begin{array}{c}\mathbb{0}\\ \mathbb{1}\end{array}\right]'$$

which is declared in 3 command lines in the SEDUMI INTERFACE interface:

```
>> lmi = sdmineq(lmi, c1, Pindex, [eye(n);zeros(q,n)] , [A,B] );
>> lmi = sdmineq(lmi, c1, 0, [C';D'] );
>> lmi = sdmineq(lmi,-c1, gindex, [zeros(n,q);eye(q)] );
```

4.2 "Left" and "right" sides of an inequality

The notions of "left" and "right" sides of a matrix inequality are quite different in SEDUMI INTERFACE and in the LMI Control Toolbox. In both interfaces the left and right sides correspond to the formulation:

$$\mathcal{L}(\mathbf{x}) \;\leq\; \mathcal{R}(\mathbf{x})$$

But while in the LMI Control Toolbox it is possible on a feasible point x^* to evaluate separately both sides, in SEDUMI INTERFACE the notion of sides is only used to define the constraints.

The right side in SEDUMI INTERFACE is used to declare terms that appear with the minus sign in a negative definite constraint. For example

```
>> lmi = sdmineq(lmi, -c, Xindex, +A)
```

is a command that adds a term such as:

$$\mathcal{L}(\mathbf{x}) \;\leq\; \mathcal{R}(\mathbf{x}) + A\mathbf{X}A' \qquad \text{or} \qquad \mathcal{L}(\mathbf{x}) - A\mathbf{X}A' \;\leq\; \mathcal{R}(\mathbf{x})$$

In contrast with the LMI Control Toolbox it is **NOT** equivalent to:

```
>> lmi = sdmineq(lmi, +c, Xindex, -A)
```

that declares a term such that:

$$\mathcal{L}(\mathbf{x}) + (-A)\mathbf{X}(-A') \ \leq \ \mathcal{R}(\mathbf{x}) \qquad \text{that is} \qquad \mathcal{L}(\mathbf{x}) + A\mathbf{X}A' \ \leq \ \mathcal{R}(\mathbf{x})$$

This remark also holds for constant terms that we chose to declare with the same structure as variable terms. Therefore, beware that the two following commands are totally different:

```
>> lmi = sdmineq(lmi, -c, 0, +B)
>> lmi = sdmineq(lmi, +c, 0, -B)
```

They respectively declare the following terms:

$$\mathcal{L}(\mathbf{x}) \ \leq \ \mathcal{R}(\mathbf{x}) + BB' \qquad \text{and} \qquad \mathcal{L}(\mathbf{x}) + (-B)(-B)' \ \leq \ \mathcal{R}(\mathbf{x})$$

which is **NOT** the same at all.

4.3 Matrix and scalar multipliers for inequality terms

A third warning concerns the use of scalar multipliers in `sdmineq`. As described previously, SeDuMi Interface allows to declare terms such as:

```
>> lmi = sdmineq(lmi, c, Xindex, 2, 3);
>> lmi = sdmineq(lmi, c, Pindex, 4, A);
>> lmi = sdmineq(lmi, c, Qindex, 5);
>> lmi = sdmineq(lmi, c, Rindex);
```

These commands add terms such that:

$$\begin{aligned}
(2\mathbb{1})\mathbf{X}(3\mathbb{1}) + (3\mathbb{1})'\mathbf{X}'(2\mathbb{1})' &= 6(\mathbf{X} + \mathbf{X}') \\
(4\mathbb{1})\mathbf{P}A + A'\mathbf{P}'(4\mathbb{1})' &= 4(\mathbf{P}A + A'\mathbf{P}') \\
(5\mathbb{1})\mathbf{Q}(5\mathbb{1})' &= 25\ \mathbf{Q} \\
(\mathbb{1})\mathbf{R}(\mathbb{1})' &= \mathbf{R}
\end{aligned}$$

Therefore the three following notations are equivalent ($\mathbf{R}$ is assumed to be symmetric):

```
>> lmi = sdmineq(lmi, c, Rindex);
>> lmi = sdmineq(lmi, c, Rindex, 1);
>> lmi = sdmineq(lmi, c, Rindex, 1, 0.5);
```

but they are not equivalent as in the LMI Control Toolbox to the notation:

```
>> lmi = sdmineq(lmi, c, Rindex, 1, 1);
```

3 Practice your SeDuMi Interface skills

As for lecture notes, let us make some exercises to practice our SeDuMi Interface skills. First, we will practice the declaration of structured variables and afterwards we will test our art on inequality definition. The solution to each exercise is given in the tables going from 12 to 17.

Exercise 1 : Declare in the same `sdmpb` object the following matrix variables:

- $\mathbf{R} \in \mathbb{R}^{3\times 4}$
- $\mathbf{S} \in \mathbb{R}^{3\times 3}$ symmetric
- $\mathbf{T} \in \mathbb{R}^{4\times 4}$ diagonal
- $\mathbf{U} \in \mathbb{R}^{3\times 3}$ anti-symmetric
- $\mathbf{V} = \begin{bmatrix} \mathbf{S} & \mathbf{R} \\ \mathbf{R}' & \mathbf{T} \end{bmatrix}$
- $\mathbf{W} = \begin{bmatrix} \mathbf{w_1} & \mathbf{u_1} & \mathbf{u_2} \\ -\mathbf{u_1} & \mathbf{w_1} & \mathbf{u_3} \\ -\mathbf{u_2} & -\mathbf{u_3} & \mathbf{w_2} \end{bmatrix}$ where $\mathbf{U} = \begin{bmatrix} 0 & -\mathbf{u_1} & -\mathbf{u_2} \\ \mathbf{u_1} & 0 & -\mathbf{u_3} \\ \mathbf{u_2} & \mathbf{u_3} & 0 \end{bmatrix}$

Exercise 2 : Declare the `sdmpb` object containing the following LMI problem:

variables:	$\mathbf{P} = \mathbf{P}' \in \mathbb{R}^{4\times 4}, \quad \mathbf{Q} = \mathbf{Q}' \in \mathbb{R}^{3\times 3}, \quad \mathbf{r} \in \mathbb{R}$
inequalities:	$\mathbb{1} \leq \mathbf{P}$
	$A_1\mathbf{P} + \mathbf{P}A_1' + B_1\mathbf{Q}B_1' \leq \mathbf{r}\mathbb{1}$
	$A_2\mathbf{P} + \mathbf{P}A_2' + B_2\mathbf{Q}B_2' \geq \mathbf{r}\mathbb{1}$
objective:	$\max \quad \mathbf{r} - \mathrm{trace}(\mathbf{P})$

Exercise 3 : Assume that the unknowns (in bold face) have already been defined in the LMI problem, declare the following matrix inequality:

$$\begin{bmatrix} A\mathbf{X} + \mathbf{X}A' + B\mathbf{R} + \mathbf{R}'B' & \mathbf{P}' + A + B\mathbf{S}C & D + B\mathbf{S}E \\ \mathbf{P} + A' + C'\mathbf{S}'B' & A'\mathbf{Y} + \mathbf{Y}A + \mathbf{Q}C + C'\mathbf{Q}' & \mathbf{Y}D + \mathbf{Q}E \\ D' + E'\mathbf{S}'B' & D'\mathbf{Y}' + E'\mathbf{Q}' & -FF' \end{bmatrix} < \mathbb{0}$$

where the first block has l rows, the second m rows and the third n rows.

Exercise 4 : Assume that the existing `sdmpb` object is such that:

```
>> ex4
LMI problem: for exercise 4
matrix variables:        index           name
                         1               R
inequality constraints:  index   meig    name
                         1       -Inf    R>0
maximise objective: -trace(T)
unsolved
```

Declare a new inequality with n rows such that:

$$a\mathbf{R} - B + CC' \leq D\mathbf{S} + \mathbf{S}'D' + \mathbf{T} \otimes E$$

where a, B, C, D and E are data matrices, $\mathbf{S}$ is a m-by-n matrix variable and $\mathbf{T}$ is a 2-by-2 symmetric matrix variable.

```
>> ex1 = sdmpb('sdmpb object for exercise 1');
>> [ex1, Rindex] = sdmvar(ex1,    3,    4, 'R');
>> [ex1, Sindex] = sdmvar(ex1,    3,  's', 'S');
>> [ex1, Tindex] = sdmvar(ex1,    4,  'd', 'T');
>> [ex1, Uindex] = sdmvar(ex1,    3, 'as', 'U');
>> Vdec = [ ex1{Sindex} , ex1{Rindex} ; (ex1{Rindex})' , ex1{Tindex} ];
>> [ex1, Vindex] = sdmvar(ex1, Vdec, 'st', 'V');
>> decnb = get(ex1, 'vardecnb');
>> Wdec = -ex1{Uindex};
>> Wdec(1,1) = decnb+1;
>> Wdec(2,2) = decnb+1;
>> Wdec(3,3) = decnb+2;
>> [ex1, Windex] = sdmvar(ex1, Wdec, 'st', 'W');
```

Table 12: Solution to exercise 1

```
>> ex2 = sdmpb('sdmpb object for exercise 2');
>> [ex2, Pindex] = sdmvar(ex2,    4, 's', 'P');
>> [ex2, Qindex] = sdmvar(ex2,    3, 's', 'Q');
>> [ex2, rindex] = sdmvar(ex2,    1,   1, 'r');
>> [ex2, c1] = sdmlmi(ex2, 4, '1<P');
>> ex2 = sdmineq(ex2, c1,      0,  1,      0.5);
>> ex2 = sdmineq(ex2, c1, Pindex,  1,     -0.5);
>> [ex2, c2] = sdmlmi(ex2, 4, 'A1P+PA1...<r');
>> ex2 = sdmineq(ex2, c2, Pindex, A1,        1);
>> ex2 = sdmineq(ex2, c2, Qindex, B1,  0.5*B1');
>> ex2 = sdmineq(ex2, c2, rindex,  1,     -0.5);
>> [ex2, c3] = sdmlmi(ex2, 4, 'A2P+PA2...>r');
>> ex2 = sdmineq(ex2, c3, Pindex, A2,       -1);
>> ex2 = sdmineq(ex2, c3, Qindex, B2, -0.5*B2');
>> ex2 = sdmineq(ex2, c3, rindex,  1,      0.5);
>> ex2 = sdmobj(ex2, rindex,    1,  1,        'r');
>> ex2 = sdmobj(ex2, Pindex, 'tr', -1, 'trace(P)');
```

Table 13: Solution to exercise 2 using only basic usage of sdmineq

```
>> [ex2, c1] = sdmlmi(ex2, 4, '1<P');
>> ex2 = sdmineq(ex2, c1,      0);
>> ex2 = sdmineq(ex2,-c1, Pindex);
>> [ex2, c2] = sdmlmi(ex2, 4, 'A1P+PA1...<r');
>> ex2 = sdmineq(ex2, c2, Pindex, A1, 1);
>> ex2 = sdmineq(ex2, c2, Qindex, B1);
>> ex2 = sdmineq(ex2,-c2, rindex);
>> [ex2, c3] = sdmlmi(ex2, 4, 'A2P+PA2...>r');
>> ex2 = sdmineq(ex2,-c3, Pindex, A2, 1);
>> ex2 = sdmineq(ex2,-c3, Qindex, B2);
>> ex2 = sdmineq(ex2, c3, rindex);
```

Table 14: Solution to exercise 2 with advanced usage of sdmineq

```
>> [ex3, c] = sdmlmi(ex3, l+m+n, name);
>> El = [eye(l);zeros(m,l);zeros(n,l)];
>> Em = [zeros(l,m);eye(m);zeros(n,m)];
>> En = [zeros(l,n);zeros(m,n);eye(n)];
>> ex3 = sdmineq(ex3, c, Xindex, El*A, El');
>> ex3 = sdmineq(ex3, c, Rindex, El*B, El');
>> ex3 = sdmineq(ex3, c, Pindex, Em, El');
>> ex3 = sdmineq(ex3, c, 0, El*A, Em');
>> ex3 = sdmineq(ex3, c, Sindex, El*B, C*Em');
>> ex3 = sdmineq(ex3, c, 0, El*D, En');
>> ex3 = sdmineq(ex3, c, Sindex, El*B, E*En');
>> ex3 = sdmineq(ex3, c, Yindex, Em, A*Em');
>> ex3 = sdmineq(ex3, c, Qindex, Em, C*Em');
>> ex3 = sdmineq(ex3, c, Yindex, Em, D*En');
>> ex3 = sdmineq(ex3, c, Qindex, Em, E*En');
>> ex3 = sdmineq(ex3,-c, 0, En*F);
```

Table 15: Solution to exercise 3

```
>> [ex4, c] = sdmlmi(ex4, n, name);
>> ex4 = sdmineq(ex4, c, 1, a, 0.5);
>> ex4 = sdmineq(ex4, c, 0, B, -0.5);
>> ex4 = sdmineq(ex4, c, 0, C, 0.5*C');
>> [ex4, Sindex] = sdmvar(ex4, m, n, 'S');
>> ex4 = sdmineq(ex4, c, Sindex, D, -1);
>> [ex4, Tindex] = sdmvar(ex4, 2, 's', 'T');
>> ex4 = sdmineq(ex4, c, Tindex, 1, -0.5, E, -1);
```

Table 16: Solution to exercise 4 using only basic usage of sdmineq

```
>> [ex4, c] = sdmlmi(ex4, n, name);
>> ex4 = sdmineq(ex4, c, 1, a, 0.5);
>> ex4 = sdmineq(ex4,-c, 0, B, 0.5);
>> ex4 = sdmineq(ex4, c, 0, C);
>> [ex4, Sindex] = sdmvar(ex4, m, n, 'S');
>> ex4 = sdmineq(ex4,-c, Sindex, D, 1);
>> [ex4, Tindex] = sdmvar(ex4, 2, 's', 'T');
>> ex4 = sdmineq(ex4,-c, Tindex, [], [], E, -1);
```

Table 17: Solution to exercise 4 with advanced usage of sdmineq

6 Examples

6.1 Call for examples

In this version of the SeDuMi User's Guide only one example is proposed. In future versions other examples will follow. To have as much examples as possible covering a large variety of different theoretical fields, we make here a **call for examples**.

Please contact us if you have some interesting examples to include in this User's Guide. It will improve this document and make some publicity for your results. The best way to contribute is to send us a MATLAB script of the LMI problem declaration and a LaTeX file that describes the theoretical problem in a few words. Thanks a lot in advance for your contributions.

6.2 Example of sdmdemo

A demonstration of SeDuMi Interface is provided with the package. To run the demonstration type:

```
>> sdmdemo
```

The example is a mixed H_2/H_∞ state-feedback problem. A random, large-size, continuous-time LTI system is generated such that:

$$\left\{\begin{array}{rrlll} & & w_1 \in \mathbb{R}^2 & w_2 \in \mathbb{R}^2 & u \in \mathbb{R}^3 \\ \dot{x} = & Ax & +B_1 w_1 & +B_2 w_2 & +Bu \qquad\qquad x \in \mathbb{R}^{10} \\ z_1 = & C_1 x & +D_1 w_1 & & +E_1 u \qquad\qquad z_1 \in \mathbb{R}^2 \\ z_2 = & C_2 x & & & +E_2 u \qquad\qquad z_2 \in \mathbb{R}^4 \end{array}\right.$$

The objective is to design a state feedback gain $u(t) = \mathbf{K}x(t)$ such that the H_∞ norm of the transfer from w_1 to z_1 is less than 10 and the H_2 norm of the transfer from w_2 to z_2 is minimised. Applying a "Lyapunov Shaping Paradigm" [19], and the standard invertible change of variable $\mathbf{K} = \mathbf{S}\mathbf{X}^{-1}$ the synthesis has an LMI formulation:

$$\left\{\begin{array}{l} ||T(s)_{w_1 \to z_1}||_\infty \leq 10 \\ ||T(s)_{w_2 \to z_2}||_2 \leq \sqrt{\mathrm{trace}(\mathbf{T})} \end{array}\right.$$

$$\text{if} \quad \left\{\begin{array}{l} \mathbf{X} > \mathbb{0} \\ \left[\begin{array}{cc} A\mathbf{X} + \mathbf{X}A' + B\mathbf{S} + \mathbf{S}'B' + B_1B_1' & \mathbf{X}C_1' + \mathbf{S}'E_1' + B_1D_1' \\ C_1\mathbf{X} + E_1\mathbf{S} + D_1B_1' & -\gamma\mathbb{1} + D_1D_1' \end{array}\right] < \mathbb{0} \\ A\mathbf{X} + \mathbf{X}A' + B\mathbf{S} + \mathbf{S}'B' + B_2B_2' < \mathbb{0} \\ \left[\begin{array}{cc} -\mathbf{T} & C_2\mathbf{X} + E_2\mathbf{S} \\ \mathbf{X}C_2' + \mathbf{S}'E_2' & -\mathbf{X} \end{array}\right] < \mathbb{0} \\ \gamma \leq 10^2 \end{array}\right.$$

The variables of the LMI problem are $\mathbf{X} \in \mathbb{R}^{10\times 10}$ symmetric, $\mathbf{S} \in \mathbb{R}^{3\times 10}$, $\mathbf{T} \in \mathbb{R}^{4\times 4}$ symmetric and γ scalar. These variables are defined in SeDuMi Interface as follows:

```
>> lmi = sdmpb('H2/Hinfty state-feedback synthesis');

>> [lmi,Xindex] = sdmvar(lmi, 10, 's', 'X')
>> [lmi,Tindex] = sdmvar(lmi, 4, 's', 'T')
>> [lmi,Sindex] = sdmvar(lmi, 3, 10, 'S')
>> [lmi,gindex] = sdmvar(lmi, 1, 1, 'g')
```

The first inequality constraint can be reformulated with the previous notations as:

$$\mathbb{0} < \mathbf{X}$$

In SeDuMi Interface it is defined as:

```
>> [lmi, c1] = sdmlmi(lmi, 10, 'X>0')
>> lmi = sdmineq(lmi, -c1, Xindex)
```

The second constraint writes as:

$$\mathrm{sym}\left\{\begin{bmatrix} A \\ C_1 \end{bmatrix} \mathbf{X} \begin{bmatrix} \mathbb{1} & \mathbb{0} \end{bmatrix} + \begin{bmatrix} B \\ E_1 \end{bmatrix} \mathbf{S} \begin{bmatrix} \mathbb{1} & \mathbb{0} \end{bmatrix}\right\} + \begin{bmatrix} B_1 \\ D_1 \end{bmatrix} \begin{bmatrix} B_1 \\ D_1 \end{bmatrix}' < \gamma \begin{bmatrix} \mathbb{0} \\ \mathbb{1} \end{bmatrix} \begin{bmatrix} \mathbb{0} \\ \mathbb{1} \end{bmatrix}'$$

In SeDuMi Interface it is defined as:

```
>> [lmi, c2] = sdmlmi(lmi, 12, 'Hinf');
>> lmi = sdmineq(lmi, c2, Xindex, [A;C1] , [eye(10),zeros(10,2)] );
>> lmi = sdmineq(lmi, c2, Sindex, [B;E1] , [eye(10),zeros(10,2)] );
>> lmi = sdmineq(lmi, c2, 0, [B1;D1] );
>> lmi = sdmineq(lmi,-c2, gindex, [zeros(10,2);eye(2)] );
```

At last, the three remaining inequalities are defined with SeDuMi Interface operators as shown in table 18.

```
>> [lmi, c3] = sdmlmi(lmi, 10, 'gram');
>> lmi = sdmineq(lmi, c3, Xindex, A, 1);
>> lmi = sdmineq(lmi, c3, Sindex, B, 1);
>> lmi = sdmineq(lmi, c3, 0, B2);

>> [lmi, c4] = sdmlmi(lmi, 14, 'H2');
>> lmi = sdmineq(lmi, c4, Xindex, [C2;zeros(10,10)] , [zeros(10,4),eye(10)] );
>> lmi = sdmineq(lmi, c4, Sindex, [E2;zeros(10,3)] , [zeros(10,4),eye(10)] );
>> lmi = sdmineq(lmi,-c4, Tindex, [eye(4);zeros(10,4)] );
>> lmi = sdmineq(lmi,-c4, Xindex, [zeros(4,10);eye(10)] );

>> [lmi, c5] = sdmlmi(lmi, 1, 'g<G1^2');
>> lmi = sdmineq(lmi, c5, gindex);
>> lmi = sdmineq(lmi,-c5, 0     , 10);
```

Table 18: Three last inequality constraints

In order to minimise the H_2 norm, the linear objective is the maximisation of $-\mathrm{trace}(\mathbf{T})$. It writes as:

```
>> lmi = sdmobj(lmi, Tindex, 'tr', -1, '-trace(T)');
```

The LMI problem is entirely defined. Table 19 shows the convergence of the SeDuMi solver.

```
>> lmi = sdmsol(lmi);

SeDuMi 1.05 Development by Jos F. Sturm, 1998, 2001.
Alg = 1: v-corrector, theta = 0.250, beta = 0.500
eqs m = 96, order n = 49, dim = 637, blocks = 6
nnz(A) = 2296 + 0, nnz(ADA) = 9216, nnz(L) = 4656
 it :     b*y       gap   delta  rate  t/maxt   feas cg cg
  0 :            4.90E+01 0.000
  1 :  -6.01E+00 1.52E+01 0.000 0.3102 0.9000  -0.98  1  1
  2 :  -2.96E+01 5.35E+00 0.000 0.3518 0.9000  -0.91  1  1
  3 :  -5.19E+01 1.69E+00 0.000 0.3153 0.9000  -0.66  1  1
  4 :  -1.85E+01 6.38E-01 0.000 0.3787 0.9000   0.44  1  1
  5 :  -6.21E+00 2.74E-01 0.000 0.4296 0.9000   1.07  1  1
  6 :  -3.26E+00 6.59E-02 0.000 0.2402 0.9000   1.14  1  1
  7 :  -2.14E+00 2.41E-02 0.000 0.3657 0.9000   1.18  1  1
  8 :  -1.76E+00 7.45E-03 0.000 0.3092 0.9000   1.13  1  1
  9 :  -1.64E+00 1.71E-03 0.000 0.2299 0.9000   1.11  1  1
 10 :  -1.61E+00 6.89E-04 0.000 0.4026 0.9000   1.11  1  1
 11 :  -1.60E+00 4.63E-05 0.000 0.0672 0.9900   1.09  1  1
 12 :  -1.60E+00 1.16E-05 0.000 0.2505 0.9000   1.10  1  1
 13 :  -1.60E+00 3.88E-06 0.000 0.3340 0.9000   1.10  2  2
 14 :  -1.60E+00 1.33E-06 0.000 0.3439 0.9000   1.08  2  2
 15 :  -1.60E+00 5.57E-07 0.057 0.4179 0.9000   1.02  2  2
 16 :  -1.60E+00 6.37E-08 0.026 0.1146 0.9450   1.02  3  2
 17 :  -1.60E+00 2.49E-08 0.057 0.3914 0.9000   1.07  5  6
 18 :  -1.60E+00 9.79E-09 0.492 0.3925 0.9000   1.02  6  6
 19 :  -1.60E+00 3.82E-10 0.000 0.0391 0.9900   1.03  6  7
 20 :  -1.60E+00 1.04E-10 0.000 0.2727 0.9000   1.03 14 13
 21 :  -1.60E+00 2.45E-11 0.000 0.2351 0.9000   1.00 11 12
iter seconds digits       c*x               b*y
 21      5.8   9.5 -1.6005447530e+00 -1.6005447535e+00
|Ax-b| =   6.2e-11, [Ay-c]_+ =   3.6E-13, |x|=  6.6e+00, |y|=  1.1e+01
Max-norms: ||b||=1, ||c|| = 9.993072e+01,
Cholesky |add|=0, |skip| = 2, ||L.L|| = 5621.26.
feasible
```

Table 19: Solve the $\mathcal{H}_2/\mathcal{H}_\infty$ state-feedback synthesis

A feasible solution is found. The output of SeDuMi shows that the objective at the optimum is `b*y = -1.60E+00` and the norm (radius) on the decision variables of the LMI problem is `|y| = 1.1e+01`. The state feedback solution may be obtained by:

```
>> X = lmi(1);
>> S = lmi(3);
>> K = S*inv(X);
```

Note that this example is randomly generated. The size of the LMI problem is quite large. There are `m = 96` decision variables and `dim = 637` lines in the inequality constraints (when all dimensions are added). Nevertheless the solution is obtained quite fast (`seconds = 5.8`, on an SUN Ultra 5 computer). The computation time is given in seconds of CPU time.

7 Conclusions

SeDuMi Interface makes the interface between a standard LMI formalism and the solver SeDuMi. For control applications it is well suited and we are already working on future evolutions. We expect potential users to contact us with remarks and eventually pass on some examples to illustrate this report. These remarks and contributions may influence the future developments and make the tool more complete.

Among evolutions that are considered for the near or far future are:

- Complex variables.
  SeDuMi Interface at this point only deals with real valued data and variables. Extensions to complex valued problems will eventually be done in a near future, catching up with the SeDuMi solver that already manages complex valued information.

- Linear Matrix Equalities.
  In SeDuMi, linear matrix equalities cannot be taken into account explicitly simultaneously with LMI constraints. A trick is then to declare two identical inequalities of opposite sign which may slow down the solver considerably and generate numerical problems. Therefore more accurate programming may be needed.

- Block partitioning.
  The block partitioning of matrix inequalities is quite natural in control problems. We believe that not only LMI Control Toolbox users may require that SeDuMi Interface exploits explicitly such structural specifications. This will probably be done in future evolutions.

Acknowledgments

Thanks a lot to Jos Sturm of the Faculty of Economics, Department of Econometrics, Tilburg University, The Netherlands, for his work on SeDuMi and his encouragements. Thanks also to Michal Kvasnica of the Slovak University of Technology in Bratislava, Slovakia, for his useful comments.

References

[1] F. Alizadeh, J.-P. Haeberly, M.V. Nayakkankuppam, M.L. Overton, and S. Schmieta. *SDPpack Version 0.9 Beta for Matlab 5.0 Semidefinite Quadratic Linearly Constrained Programs.* New York University, 1997. URL: `http://www.cs.nyu.edu/faculty/overton/sdppack/sdppack.html`.

[2] S.J. Benson and Y. Ye. DSDP3: Dual scaling algorithm for general positive semidefinite programs. Technical Report ANL/MCS-P851-1000, Argonne National Laboratory, November 2000.

[3] S.J. Benson, Y. Ye, and X. Zhang. Solving large scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal of Optimization*, 10:443–461, 2000.

[4] B. Borchers. CSDP, 2.3 user's guide. *Optimization Methods and Software*, 11(1):597–611, 1999.

[5] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

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

[7] N. Brixius, R. Sheng, and F.A. Potra. *SDPHA User Guide*. Department of Computer Science, University of Iowa, July 1998. URL: `http://www.cs.uiowa.edu/~brixius/SDPHA`.

[8] L. El Ghaoui, F. Delebecque, and R. Nikoukhah. `LMITOOL` *: a User-Friendly Interface for LMI Optimisation*, 1995. User's Guide, Beta version, URL: `http://robotics.eecs.berkeley.edu/~elghaoui/lmitool/lmitool.html`.

[9] L. El Ghaoui and S.-I. Niculescu, editors. *Advances in Linear Matrix Inequality Methods in Control*. Advances in Design and Control. SIAM, Philadelphia, 2000.

[10] L. El Ghaoui, F. Oustry, and M. AitRami. A cone complementarity linearization algorithm for static ouput-feedback and related problems. *IEEE Trans. on Automat. Control*, 42(8):1171–1176, August 1997.

[11] K Fujisawa, M. Kojima, and K. Nakata. *SDPA (SemiDefinite Programming Algorithm) User's Manual - Version 5.00*. Tokyo Institute of Technology, August 1999. URL: `ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA`.

[12] P. Gahinet, A. Nemirovski, A.J. Laub, and M. Chilali. *LMI Control Toolbox User's Guide*. The Mathworks Partner Series, 1995.

[13] K.M. Grigoriadis and E.B. Beran. *Advances in Linear Matrix Inequality Methods in Control*, chapter 13 : Alternating Projection Algorithms for Linear Matrix Inequalities Problems with Rank Constraints, pages 251–267. SIAM, Philadelphia, 2000. Edited by L. El Ghaoui and S.-I. Niculescu.

[14] T. Iwasaki. LPV system analysis with quadratic separator for uncertain implicit systems. *LMI Methods in optimisation, identification and control*, Seminar, Compiègne, France, May 1999.

[15] S.E. Karisch. *CUTSDP - A Toolbox for a Cutting-Plane Approach Based on Semidefinite Programming*. Department of Mathematical Modelling, Technical University of Denmark, June 1998. Technical Report IMM-REP-1998-10, URL: `http://www.imm.dtu.dk/~sk/cutsdp/`.

[16] F. Leibfritz. An LMI-based algorithm for designing suboptimal static H_2/H_∞ output feedback controllers. *SIAM Journal on Control and Optimization*, 39(6):1711 – 1735, 2001.

[17] H.D. Mittelmann. An independent benchmarking of SDP and SOCP solvers. URL: `http://www.optimization-online.org/DB_HTML/2001/07/358.html`, July 2001. Arizona State University.

[18] D. Peaucelle and D. Arzelier. An efficient numerical solution for H_2 static output feedback synthesis. In *European Control Conference*, pages 3800–3805, Porto, Portugal, September 2001.

[19] C. Scherer, P. Gahinet, and M. Chilali. Multiobjective output-feedback control via LMI optimisation. *IEEE Trans. on Automat. Control*, 42(7):896–911, July 1997.

[20] Pete Seiler. Lmilab translator. URL: `http://vehicle.me.berkeley.edu/~guiness/lmitrans.html`, July 2001. University of Berkeley, California.

[21] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12:625–653, 1999. Special issue on Interior Point Methods.

[22] T.C. Toh, M.J. Todd, and R.H. Tutuncu. SDPT3 — a MATLAB software package for semidefinite programming, version 2.1. *Optimization Methods and Software*, 11:545–581, 1999. URL: `http://www.math.nus.edu.sg/~mattohkc/index.html`.

[23] J.G. VanAntwerp and R.D. Braatz. A tutorial on linear and bilinear matrix inequalities. *J. Process Control*, 10:363–385, 2000.

[24] L. Vandenberghe and S. Boyd. `SP` : Softvare for semidefinite programming. User's guide, beta version. Technical report, K.U. Leuven and Stanford University, 1994. URL: `http://www.stanford.edu/~boyd/SP.html`.

[25] S.-P. Wu and S. Boyd. Design and implementation of a parser/solver for SDPs with matrix structure. In *IEEE Conference on Computer Aided Control System Design*, New York, 1996.