User's Guide for SEDUMI INTERFACE 1.04

Dimitri Peaucelle - Didier Henrion - Yann Labit - Krysten Taitz
LAAS - CNRS
7, Avenue du Colonel Roche - 31077 Toulouse cedex 4 - France
Tel. 05 61 33 64 17 fax: 05 61 33 69 69
email: peaucelle@laas.fr - henrion@laas.fr - ylabit@laas.fr

13th September 2002

Abstract

This report describes a user-friendly MATLAB package for defining Linear Matrix Constraints (LMCs). 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 LMC problem.
  Five MATLAB functions allow to define completely an LMC problem which can be characterised by scalar and matrix variables, linear matrix equality (LME) constraints, linear matrix inequality (LMI) constraints and a linear objective:
  - Initialise the LMC problem: `sdmpb`.
  - Declare the matrix variables: `sdmvar`.
  - Declare the block partitioned equality constraints: `sdmlme` and `sdmequ`.
  - Declare the block partitioned inequality constraints: `sdmlmi` and `sdminequ`.
  - Declare the linear objective: `sdmobj`.
- Solve an LMC problem.
  A unique function, `sdmsol`, calls the SEDUMI solver. Options allow to tune the solver parameters.
- Modify an LMC problem.
  At any moment it is possible to append an LMC 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 constraints are satisfied.

This document is an update with respect to SEDUMI INTERFACE 1.01 [23], [22], SEDUMI INTERFACE 1.02 [17] and SEDUMI INTERFACE 1.03 . The last major modifications concern the **blocks partitioning** of LMCs, the **maximisation of** $Trace(BX)$ where B is a data matrix and the **increase of LMC problem construction speed**.

Contents

1 Purpose **2**
- 1.1 Notations . . . 2
- 1.2 LMC problems . . . 3
- 1.3 Existing solvers . . . 4
- 1.4 Existing Interfaces . . . 5
- 1.5 Functionalities . . . 5
- 1.6 Installing SeDuMi Interface . . . 6

2 Using SeDuMi Interface - Getting started **6**
- 2.1 LMC problem: an `sdmpb` object . . . 6
- 2.2 Defining matrix variables: `sdmvar` . . . 8
- 2.3 Defining an inequality constraint: `sdmlmi` . . . 9
- 2.4 Add a term to an inequality constraint: `sdmineq` . . . 10
- 2.5 Add a linear term to the objective: `sdmobj` . . . 11
- 2.6 Solve an LMC problem: `sdmsol` . . . 12
- 2.7 Extracting the computed solution . . . 14
- 2.8 Analysis of the computed solution . . . 14

3 Advanced use of SeDuMi Interface **15**
- 3.1 Structured variables: `sdmvar` . . . 15
- 3.2 Advanced declaration of some inequality terms: `sdmineq` . . . 19
  - 3.2.1 Block partitioning . . . 19
  - 3.2.2 Left and right sides of an inequality . . . 20
  - 3.2.3 Transpose of some matrix variable ($L\mathbf{X}^T R$) . . . 21
  - 3.2.4 Matrix terms depending on a scalar variable . . . 21
  - 3.2.5 Scalar L and R multipliers . . . 22
  - 3.2.6 Hermitian terms ($L\mathbf{X}L^*$ and LL^*) . . . 22
  - 3.2.7 Terms with no multiplying data . . . 22
  - 3.2.8 Terms with Kronecker products ($L(K\otimes\mathbf{X})R$ and $L(\mathbf{X}\otimes K)R$) . . . 23
- 3.3 Equality constraints . . . 24
  - 3.3.1 Defining an equality constraint: `sdmlme` . . . 24
  - 3.3.2 Add a term to an equality constraint: `sdmeq` . . . 25
- 3.4 The trace as an objective (max trace($\mathbf{BX}$)) . . . 28
- 3.5 Set a value to some variable: `sdmset` . . . 28
- 3.6 Tuning the solver parameters . . . 29
  - 3.6.1 SeDuMi parameters . . . 30
  - 3.6.2 Definite and semi-definite inequalities . . . 31
  - 3.6.3 Feasibility radius . . . 32

4 Warnings for MATLAB LMI Toolbox users **33**
- 4.1 "Left" and "right" sides of an inequality . . . 33
- 4.2 Matrix and scalar multipliers for inequality terms . . . 34

5 Conclusions **34**

References **35**

1 Purpose

The tool described in this report is designed to associate both efficient Semi-Definite Programming (SDP) algorithms and the nice Linear Matrix Constraint (LMIs and LMEs) formalism used for control applications. This work was inspired by the observation that on the one hand Linear Matrix Constraints (LMCs) have a major position in current academic research [6, 10, 27], and on the other hand there are new promising tools for solving relatively large-scale SDP problems [20]. But there are few tools that associate both an efficient SDP solver and a pleasant interface for declaring LMC problems within the most commonly used software environment: MATLAB.

SEDUMI INTERFACE is designed as an add-on for MATLAB and allows to declare LMC problems to be solved with the SEDUMI solver proposed by Jos Sturm [25]. 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 [13].

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

i is the imaginary unit equal to the square root of -1.
$\mathbb{R}^{m\times n}$ ($\mathbb{C}^{m\times n}$) is the set of m-by-n real (complex) matrices.
$\mathbb{1}$ and $\mathbb{0}$ are respectively the identity and the zero matrices of appropriate dimensions.
All matrices are written using capital letters (A) while scalars and vectors are in lowercase (a).

$\bar{A}$ is the conjugate of the complex matrix A, A^T is its transpose and A^* is its conjugate transpose. We remind that in the MATLAB environment the conjugate transpose writes as `A'` while the transpose is obtained by `A.'`. If $A = A^T$, the matrix is symmetric and Hermitian if $A = A^*$. For real valued matrices both notions are equivalent.
'He' is the matrix operator such that: $\mathrm{He}\{A\} = A + A^*$.

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

In matrix equalities and 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).

Note that in the MATLAB environment real integers are nicely displayed as follows, while complex integers are displayed as any other complex number:

```
>> 2
ans =
     2
>> 2+2i
ans =
   2.0000 + 2.0000i
```

Therefore, the `num2str` MATLAB function will sometimes appear for nice display. The result is a string that cannot be used for computation.

```
>> num2str(2+2i)
ans =
2+2i
```

1.2 LMC problems

The academic results on LMIs use the following optimisation formalism [6, 10]:

$$p^{opt} = \min \quad c^T \mathbf{x} \qquad s.t. \qquad F_0 + \sum_{j=1}^{m} \mathbf{x_j} F_j \geq \mathbb{0} \tag{1}$$

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 $\mathbf{x_j}$. 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 [25]:

$$p^{opt} = \min \quad c^T \mathbf{x} \qquad s.t. \qquad b - A\mathbf{x} \text{ is positive semi-definite}$$

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:

$$A^T \mathbf{P} + \mathbf{P} A < \mathbb{0}$$

Assume A is a 2-by-2 matrix. $\mathbf{P}$ is a symmetric matrix considered as the variable in the LMI problem. Expressed in terms of the scalar data and the scalar variables (bold face) the LMI writes 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:

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

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

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. It ranges from full block matrices to any structured matrix variable including the symmetric, anti-symmetric, Hermitian and anti-Hermitian variables.
- Second, both linear matrix inequality (LMI) and equality (LME) constraints can be declared. In the sequel, an LMC problem is an optimisation problem with a linear objective and possibly both LMI and LME constraints.

- Third, various types of linear matrix terms can be declared including Kronecker products between data matrices and matrix variables.
- Fourth, the linear objective is a sum of linear terms and can contain the trace operator.
- Fifth, data matrices and variables can be real or complex valued. LMIs are then interpreted as Hermitian positive definite constraints.
- Sixth, the LMCs can be declared taking into account a block partitioning.

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 [28] and the LMI-lab which evolved into the MATLAB LMI Control Toolbox [13]. Since then, a wide variety of solvers have been developed among which: SEDUMI [25], SDPA [12], SDPHA [7], SDPpack [1], SDPT3 [26], CSDP [5, 4], CUTSDP [16], 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 [20].

Having ourselves tested some of the solvers and in view of the report [20], we came to the choice of SEDUMI. The advantages of this solver that guided our choice are:

- Asymptotic computational complexity. Let n be the number of decision variables and m the number of rows of the LMIs. The computational complexity of SEDUMI (including main and inner iterations) is in $O(n^2m^{2.5}+m^{3.5})$ while the algorithm in [13] has a complexity $O(n^3m)$. The former algorithm is more efficient for problems with a large number of variables. This is of major interest when solving large scale problems or when implementing LMI-based iterative algorithms as in [11, 15, 21, 14, 18].
- 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.
- Complex valued problems. Both the data and the variables may be given or constrained with real or complex values. At the difference of [8], this is done without increasing the size of the problem.
- Equality constraints. SEDUMI allows to declare explicitely linear equality constraints without computing any kernel or artificially defining an equality as two opposite-side inequalities.
- 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. These potentialities will be integrated into SEDUMI INTERFACE in the future, depending on possible 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 [13]. Then comes the sdpsol software [29], which is associated to the SP solver [28], and the LMITOOL package [9], which calls three different solvers: SP [28], SDPHA [7] and SDPPack [1]. All three interfaces work in MATLAB environment.

Except the plurality of the solvers, the difference between these three interfaces is the way LMCs are declared. They all adopt different formalisms. LMITOOL is purely a graphical user interface (GUI) tool and is nice and convivial. As a by-product, the transformation of data into the various solver formalisms is quite slow. The LMC 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 [13], 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 LMC problems as in the LMI Control Toolbox [13]. The result is less convivial than a GUI but allows more flexibility. Nevertheless, efforts were made on nice display and some complications we noticed in the LMI Control Toolbox do not occur in SEDUMI INTERFACE.

Note that more recently, almost at the same time as SEDUMI INTERFACE 1.01 was achieved, have appeared two other tools [24, 19]. They adopt a quite close approach but focus on offering the interface with multiple solvers. LMIlab Translator [24], proposes a translator that allows to call various solvers, among which SEDUMI, using the interface of the LMI Control Toolbox. YALMIP [19], is a completely new interface more alike LMITOOL [9] but without the GUI. Having focused on translation these two tools are highly valuable to test the efficiency of solvers on different hard problems. The by-product is that they do not exploit all the potentialities of each solver.

1.5 Functionalities

The functionalities of SEDUMI INTERFACE are the following:

- Declare an LMC problem.
  Five MATLAB functions allow to define completely an LMC problem characterised by matrix variables, linear matrix equalities (LMEs), linear matrix inequalities (LMIs) and a linear objective:
  - Initialise the LMC problem: `sdmpb`.
  - Declare the matrix variables: `sdmvar`.
  - Declare the block partitioned equality constraints: `sdmlme` and `sdmequ`.
  - Declare the block partitioned inequality constraints: `sdmlmi` and `sdminequ`.
  - Declare the linear objective: `sdmobj`.
- Solve an LMC problem.
  A unique function, `sdmsol`, calls the SEDUMI solver. Options allow to tune the solver parameters.
- Modify an LMC problem.
  At any moment it is possible to append an LMC 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 constraints are satisfied.

1.6 Installing SEDUMI INTERFACE

SEDUMI INTERFACE is composed of simple MATLAB files (`sdm***.m`) gathered in a directory, `SeDuMiInt104`, 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`, `sdmrank`, `sdmclear` : five files called by some of the SEDUMI INTERFACE operators.

The subdirectory, `@sdmpb`, contains the 14 essential operators for LMC problem declaration.

The interface works with MATLAB version 5.3 or more and with SEDUMI version 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/software/sedumi.html`. 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/SeDuMiInt104');
```

Then test if SEDUMI INTERFACE and SEDUMI are properly installed by typing:

```
>> sdmdemo
```

Demonstration examples of SEDUMI INTERFACE are then run. One follows the example in this user's guide. The other 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 LMC problem: an `sdmpb` object

An LMC problem within SEDUMI INTERFACE is described by a single MATLAB variable. It contains all the information on the matrix variables, the inequality and equality constraints, the linear objective and the optimal solution. The various fields containing this information are assigned as the LMC 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 LMC 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.

For 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:

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

The optimal synthesis problem is an LMC problem composed of three variables, a linear objective and two inequality constraints. The second matrix inequality is necessarily symmetric and composed of four blocks. It can write as:

variables:	$\mathbf{Q} = \mathbf{Q}^T \in \mathbb{R}^{n\times n},\ \mathbf{Y} \in \mathbb{R}^{m\times n},\ \boldsymbol{\gamma}^2 \in \mathbb{R}$
inequalities:	$n \;\updownarrow \quad \mathrm{He}\{-\frac{1}{2}\mathbf{Q}\} < \mathbb{0}$ $\begin{array}{l} n \;\updownarrow \\ p \;\updownarrow \end{array} \quad \mathrm{He}\left\{ \left[\begin{array}{c\|c} A\mathbf{Q} + B_u\mathbf{Y} + \frac{1}{2}B_w B_w^T & \mathbb{0} \\ \hline C_z\mathbf{Q} + D_{zu}\mathbf{Y} & -\frac{1}{2}\boldsymbol{\gamma}^2\mathbb{1} \end{array} \right] \right\} < \mathbb{0}$
objective:	$\max \; -\boldsymbol{\gamma}^2$

First, let us initialise the `sdmpb` object within MATLAB and give it a name:

```
>> quiz=sdmpb('Optimal Hinfty State-Feedback Synthesis')
LMC problem: Optimal Hinfty State-Feedback Synthesis
no matrix variable
no equality constraint
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 LMC problem. This label is used only for nice display. At this step we have declared an empty `sdmpb` object. The class of the variable `quiz` in MATLAB is `sdmpb`. Each declared LMC problem is an `sdmpb object`:

```
>> whos quiz
  Name      Size         Bytes  Class
  quiz       1x1          6406  sdmpb object
```

In the following subsections we assume that the data `A`, `Bu`, `Bw`, `Cz`, `Dzu`, `n`, `m`, `q` and `p`, 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 ];
```

2.2 Defining matrix variables: `sdmvar`

All variables in SEDUMI INTERFACE are real or complex matrix variables. Extensions to complex variables were introduced in 1.02 version of SEDUMI INTERFACE. By default, the variables are real full rectangular matrices declared as follows:

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

IN The first input argument of the function is the `sdmpb` object for 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.

OUT The output arguments are first the appended `sdmpb` object, and second an index, `Yindex`, that makes reference to the declared variable. The index is an integer used later on for various purposes. It can be retrieved at any time by the user as follows:

```
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
no equality constraint
no inequality constraint
no linear objective
unsolved
```

In this example **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:

```
>> [quiz, gindex] = sdmvar(quiz, 1, 1, 'gammaˆ2');
```

In LMC problems, the matrix variables may have some structural properties. In our example **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:

```
>> [quiz, Qindex] = sdmvar(quiz, 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 LMC problem of the example is characterised by:

```
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
no equality constraint
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(quiz, 'varnb')
ans =
     3
```

the name of a variable referenced by its index:

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

and also the structure of a variable:

```
>> get(quiz, '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 its 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 section 1.2 and the advanced usage of `sdmvar` in section 3.1.

2.3 Defining an inequality constraint: `sdmlmi`

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

```
>> [quiz, lmi1index] = sdmlmi(quiz,   [n], 'Q>0');
>> [quiz, lmi2index] = sdmlmi(quiz, [n p], 'Hinfty state-feedback');
```

IN The first input argument is the `sdmpb` object to append. The second input argument is a vector containing the size of the diagonal blocks in the LMI constraint. The last input argument is an optional label used for nice display.

OUT The appended `sdmpb` object is returned along with an index, `lmi1index`, that makes reference to the declared constraint.

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

```
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gammaˆ2
                         3               Q
no equality constraint
inequality constraints:  index   meig    name
                         1       -Inf    Q>0
                         2       -Inf    Hinfty state-feedback
no linear objective
unsolved
```

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

```
>> get(quiz, 'ineqnb')
ans =
     2
>> get(quiz, 'ineqname', lmi2index)
ans =
Hinfty state-feedback
>> get(quiz, 'ineqdim', lmi2index)
ans =
    4 2
```

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 at the left side of the inequality sign $<$. 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 = Cz;
>> R = 1;
>> quiz = sdmineq(quiz, [lmi2index 2 1], Qindex, L, R);
```

IN The first input argument is the `sdmpb` object to append. The second input argument is a vector containing the index of the inequality constraint and the block index in row and column (`blrow`, `blcol`) 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.

OUT The appended `sdmpb` object is the unique retrieved output argument.

Note that since matrix inequalities are Hermitian (symmetric in the case of real valued matrices), the conjugate transpose term is automatically added. This is an important feature of SEDUMI INTERFACE. By default, `sdmineq` always adds the conjugate transposed term $R^*\mathbf{X}^*L^*$ to the symmetric block given by (`blcol`, `blrow`) along with the declared term $L\mathbf{X}R$. The last command line adds the term $C_z\mathbf{Q}$ in the $(2,1)$ block, the symmetric term $\mathbf{Q}C_z'$ is automatically added to the same $(1,2)$ block.

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 the user is

assumed to define both left (fourth input argument) and right (fifth input argument) matrices. Moreover, the conjugate transpose term is automatically added: R^*L^* is added in the (`blcol`, `blrow`) block.

In our example, the constant term is such as $B_w B_w^T$ in the (1,1) block. It can be declared as follows:

```
>> L = Bw;
>> R = 0.5*L';
>> quiz = sdmineq(quiz, [lmi2index 1 1], 0, L, R);
```

Note that the factor $0.5 = \frac{1}{2}$ is necessary. For blocks on the diagonal (`blrow` = `blcol`), the term R^*L^* is added to LR in the same block. Beware not to declare it twice! This is why one of the multiplying data matrices is divided by two in this example.

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

```
>> quiz = sdmineq(quiz, [lmi1index 1 1], Qindex, -0.5, 1);

>> quiz = sdmineq(quiz, [lmi2index 1 1], Qindex,    A, 1);
>> quiz = sdmineq(quiz, [lmi2index 1 1], Yindex,   Bu, 1);
>> quiz = sdmineq(quiz, [lmi2index 2 1], Yindex,  Dzu, 1);
>> quiz = sdmineq(quiz, [lmi2index 2 2], gindex, -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 section 3.2.

2.5 Add a linear term to the objective: `sdmobj`

By default the objective is empty. The LMC 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 LMCs, the objective defined by `sdmobj` will be maximised. To minimise some objective, one has then to maximise its opposite. Remark that if the objective is complex, SEDUMI will maximise its real part.

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:

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

IN 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.

OUT The unique output argument contains the appended LMC problem `quiz`.

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 an objective term. The advanced usage of `sdmobj` is described in section 3.4.

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

```
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
no equality constraint
inequality constraints:  index   meig    name
                         1       -Inf    Q>0
                         2       -Inf    Hinfty state-feedback
maximise objective:-gamma^2
unsolved
```

Table 1: Completely declared LMC problem

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

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

2.6 Solve an LMC problem: `sdmsol`

Having defined an LMC 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 LMC 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 section 3.6.1.

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

An LMC 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 `-27.3`. When no objective is specified, feasibility means that the LMCs admit at least one solution found by SEDUMI.
- **infeasible**
  This means that the LMCs have no solution at all.
- **marginal feasibility**
  This occurs when SEDUMI has some numerical problems and cannot determine exactly a feasible solution. To avoid such complications see comments in section 3.6.

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

```
>> get(quiz, '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 marginally feasible and a string '`unsolved`' if SEDUMI was not executed on this LMC problem.

```
>> quiz = sdmsol(quiz);

SeDuMi 1.05 by Jos F. Sturm, 1998, 2001.
Alg = 2: xz-corrector, Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 16, order n = 13, dim = 69, blocks = 4
nnz(A) = 74 + 0, nnz(ADA) = 256, nnz(L) = 136
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg
  0 :            1.73E+07 0.000
  1 :  -7.50E-01 1.02E+06 0.000 0.0588 0.0000 0.9000   1.14  1  1
  2 :  -1.81E+00 2.66E+05 0.000 0.2614 0.9000 0.9000   0.71  1  1
  3 :  -2.91E+00 9.37E+04 0.000 0.3520 0.9000 0.9000   0.28  1  1
  4 :  -8.32E+00 2.08E+04 0.000 0.2225 0.9000 0.9121  -0.21  1  1
  5 :  -4.47E+01 5.37E+02 0.000 0.0257 0.9000 0.9173  -0.41  1  1
  6 :  -2.99E+01 4.72E+01 0.054 0.0879 0.9900 0.9900   1.41  1  1
  7 :  -2.82E+01 4.05E+00 0.000 0.0859 0.9000 0.9174   1.00  1  1
  8 :  -2.78E+01 1.34E+00 0.000 0.3316 0.9000 0.9000   0.78  1  1
  9 :  -2.75E+01 1.43E-01 0.000 0.1067 0.9000 0.9194   0.78  1  1
 10 :  -2.74E+01 2.66E-02 0.121 0.1852 0.0000 0.9000   0.70  1  1
 11 :  -2.74E+01 7.74E-06 0.000 0.0003 0.9000 0.8498   0.67  1  1
 12 :  -2.73E+01 1.37E-06 0.000 0.1777 0.8558 0.9000   0.24  2  2
 13 :  -2.73E+01 9.81E-08 0.000 0.0714 0.9900 0.9900   0.84  2  2
 14 :  -2.73E+01 3.03E-09 0.003 0.0308 0.9900 0.9900   0.98  2  2
 15 :  -2.73E+01 1.89E-10 0.000 0.0625 0.9900 0.9900   1.00  2  2
 16 :  -2.73E+01 3.82E-11 0.000 0.2019 0.9000 0.9000   1.00  3  2
 17 :  -2.73E+01 1.71E-12 0.000 0.0447 0.9900 0.9900   1.00  3  3
iter seconds digits       c*x               b*y
 17      1.1   Inf -2.7287769476e+01 -2.7287769465e+01
|Ax-b| =   1.1e-09, [Ay-c]_+ =   0.0E+00, |x|=  4.1e+02, |y|=  3.0e+02
Max-norms: ||b||=1, ||c|| = 1,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 193074.
feasible
```

Table 2: Solving an LMC problem with SeDuMi 1.05

```
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gammaˆ2
                         3               Q
no equality constraint
inequality constraints:  index   meig    name
                         1       eps     Q>0
                         2       eps     Hinfty state-feedback
maximise objective:-gammaˆ2 = -27.3
feasible
```

Table 3: Solved LMC 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(quiz, 'varvalue', gindex)
g2_opt =
   27.2878
```

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

```
>> Y_opt = quiz(Yindex)
Y_opt =
  -41.2455  -53.0257   -0.8462  -25.0895
```

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(quiz,'objopt')
ans =
  -27.2878
```

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

$$\gamma^{opt} = \sqrt{27.2878} = 5.2238$$

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 LMC problem (table 3) and can also be obtained with the `sdmpb/get` function as:

```
>> get(quiz, 'ineqmeig', lmi1index)
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 LMC problem has not been solved.
- `0` : occurs when the minimal eigenvalue 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 LMI 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 section 3.6. 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 LMC problem. Moreover, the operator allows to specify the structure of 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 with respect to some vector containing all independent decision variables (see section 1.2 for the definition of this vector of decision variables).

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

```
>> get(quiz, 'vardec', Yindex)
ans =
     1     2     3     4
>> quiz{gindex}
ans =
     5
>> quiz{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

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 all depend 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; γ 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).

In the sequel assume that the new variables are independent of the previous ones.

- **real full rectangular**
  The first input argument to sdmvar is the variable describing the LMC problem (`quiz`); 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`).

```
>> [quiz, VARindex] = sdmvar(quiz, 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 LMC 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`.

- **complex full rectangular**
  To declare a complex valued matrix variable, the second input argument is slightly modified to be an imaginary integer. It writes as (`m*i`) where (`m`) is the number of rows and (`i`) is the imaginary number ($i = \sqrt{-1}$).

```
>> [quiz, VARindex] = sdmvar(quiz, m*i, n, name);
```

  This rule holds for all following structured matrix variables.

- `d` **diagonal real or complex**
  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 $\mathbb{C}^3$ diagonal variable is:

```
>> [quiz, Dindex] = sdmvar(quiz, 3i, 'd', 'D : diagonal');
>> num2str(quiz{Dindex})
ans =
16+16i    0+0i     0+0i
 0+0i    17+17i    0+0i
 0+0i     0+0i    18+18i
```

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

- `as` **anti-symmetric real or complex**
  Square anti-symmetric matrix variable.
  Contains $m(m-1)/2$ independent decision variables when m 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:

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

- `h` **Hermitian**
  Square Hermitian matrix variable.
  Only defined if the matrix is complex valued, i.e. the second input argument is imaginary: `m*i`.
  Contains $m(m+1)/2$ independent decision variables when m is the number of rows of the matrix.
  The third input argument of `sdmvar` is set to the string value `'h'`.
  An example of $\mathbb{C}^2$ Hermitian variable is:

```
>> [quiz, Hindex] = sdmvar(quiz, 2i, 'h', 'H');
>> num2str(quiz{Hindex})
ans =
22+0i    23-23i
23+23i   24+0i
```

- `ah` **anti-Hermitian**
  Square anti-Hermitian matrix variable.
  Only defined if the matrix is complex valued, i.e. the second input argument is imaginary: `m*i`.
  Contains $m(m-1)/2$ independent decision variables when m is the number of rows of the matrix.
  The third argument of `sdmvar` is set to the string value `'ah'`.
  An example of $\mathbb{C}^3$ anti-Hermitian variable is:

```
>> [quiz, Jindex] = sdmvar(quiz, 3i, 'ah', 'J');
>> num2str(quiz{Jindex})
ans =
 0+0i   -25+25i  -26+26i
25+25i    0+0i   -27+27i
26+26i   27+27i    0+0i
```

At this step, note that the matrix describing the structure of the variables can be composed of zeros and integers that are possibly multiplied by (-1), $(1+i)$, $(1-i)$, $(-1+i)$ and $(-1-i)$. The rule is the following. Let $\mathbf{X}$ be a matrix variable and `Xdec=quiz{Xindex}` be the matrix that describes its structure. Moreover, let $\mathbf{x}$ be the vector of scalar (real or complex) decision variables (see section 1.2 for its definition). $\mathbf{X}_{(j,k)}$ is the element of $\mathbf{X}$ in the j-th row and k-th column. $\mathbf{x_n}$ is the n-th decision variable.

`Xdec(j,k)= 0`	if $\mathbf{X}_{(j,k)} = 0$	constrained to be equal to zero.
`Xdec(j,k)=+n`	if $\mathbf{X}_{(j,k)} = \mathbf{x_n}$	the n-th decision variable is real, $\mathbf{x_n} \in \mathbb{R}$.
`Xdec(j,k)=-n`	if $\mathbf{X}_{(j,k)} = -\mathbf{x_n}$	opposite of $\mathbf{x_n} \in \mathbb{R}$.
`Xdec(j,k)=+n+n*i`	if $\mathbf{X}_{(j,k)} = \mathbf{x_n}$	the n-th decision variable is complex, $\mathbf{x_n} \in \mathbb{C}$.
`Xdec(j,k)=-n-n*i`	if $\mathbf{X}_{(j,k)} = -\mathbf{x_n}$	opposite of $\mathbf{x_n} \in \mathbb{C}$.
`Xdec(j,k)=+n-n*i`	if $\mathbf{X}_{(j,k)} = \overline{\mathbf{x}}_\mathbf{n}$	conjugate of $\mathbf{x_n} \in \mathbb{C}$.
`Xdec(j,k)=-n+n*i`	if $\mathbf{X}_{(j,k)} = -\overline{\mathbf{x}}_\mathbf{n}$	opposite conjugate of $\mathbf{x_n} \in \mathbb{C}$.

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 real or complex "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 **F** should be rectangular and composed of the conjugate of a diagonal block and an anti-symmetric block as follows:

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

where **D** and **G** are already defined matrix variables. To declare **F** the commands are:

```
>> Ddec = quiz{Dindex};
>> Ddecbar = conj(Ddec);
>> Gdec = quiz{Gindex};
>> Fdec = [ Ddecbar , Gdec ];
>> [quiz, Findex] = sdmvar(quiz, Fdec, 'st', '[D, G]');
>> num2str(quiz{Findex})
ans =
16-16i    0+0i     0+0i     0+0i   -19+0i   -20+0i
 0+0i    17-17i    0+0i    19+0i     0+0i   -21+0i
 0+0i     0+0i    18-18i   20+0i    21+0i     0+0i
```

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

$$\mathbf{K} = \begin{bmatrix} k_1 & 0 & -k_1 \\ k_3 & k_2 & \bar{k}_3 \end{bmatrix}$$

where $k_1 \in \mathbb{C}$, $k_2 \in \mathbb{R}$ and $k_3 \in \mathbb{C}$. 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(quiz, 'vardecnb')
ans =
    27
>> Kdec = [28+28i 0 -28-28i;30+30i 29 30-30i];
>> [quiz, Kindex] = sdmvar(quiz, Kdec, 'st', 'K');
>> num2str(quiz{Kindex})
ans =
28+28i    0+0i   -28-28i
30+30i   29+0i    30-30i
```

3.2 Advanced declaration of some inequality terms: `sdmineq`

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

```
>> quiz = sdmineq(quiz, [LMIindex blrow blcol], Xindex, L, R);
```

The modified LMI constraint is declared by its index, `LMIindex`, and the matrix variable is referred to by the index, `Xindex`. The term $L\mathbf{X}R$ is added to the (`blrow`,`blcol`) block and the Hermitian term $R^*\mathbf{X}^*L^*$ is added in the symmetric block (`blcol`,`blrow`). If `Xindex=0` then the constant terms LR and R^*L^* are respectively added in the symmetric blocks.

Some options to this function are now detailed.

3.2.1 Block partitioning

For some LMI constraints it may not be necessary to define a block partitioning. For example take the first LMI of the previous section. It writes as:

$$\mathbf{Q} > \mathbb{0} \quad \Leftrightarrow \quad \mathrm{He}\{-\frac{1}{2}\mathbf{Q}\} < \mathbb{0}$$

It can be declared as previously as if composed of a single block:

```
>> [quiz, lmi1index] = sdmlmi(quiz, [n], 'Q>0');
>> quiz = sdmlmi(quiz, [lmi1index 1 1], Qindex, -0.5, 1);
```

An alternative is to remove all references to the blocks:

```
>> [quiz, lmi1index] = sdmlmi(quiz, n, 'Q>0');
>> quiz = sdmlmi(quiz, lmi1index, Qindex, -0.5, 1);
```

Of course, the two notations are equivalent. But the usage of `sdmineq` without specifying blocks may also be used for block partitioned LMIs (more than one block). The rule is the following:

- **specifying blocks**
  When the second input argument of `sdmineq` is composed of three entries (`[LMIindex blrow blcol]`) then the term $L\mathbf{X}R$ is added to the (`blrow,blcol`) block and the Hermitian term $R^*\mathbf{X}^*L^*$ is added to the symmetric (`blcol,blrow`) block (same comment for constant terms LR).
- **not specifying block**
  If the second input argument of `sdmineq` is a scalar (`LMIindex`) then the term $L\mathbf{X}R$ is added to the entire LMI so as the Hermitian term $R^*\mathbf{X}^*L^*$ (same comment for constant terms LR). Note that in this case the number of rows of L and the number of columns of R should be equal to the global LMI dimension (sum of the blocks dimensions).

An example of this usage is now described. First note that the second LMI of the previous example writes equivalently as:

$$\mathrm{He}\left\{\left[\begin{array}{c|c} A\mathbf{Q}+B_u\mathbf{Y}+\frac{1}{2}B_wB_w^T & \mathbb{0} \\ \hline C_z\mathbf{Q}+D_{zu}\mathbf{Y} & -\frac{1}{2}\boldsymbol{\gamma}^2\mathbb{1} \end{array}\right]\right\} < \mathbb{0}$$

$$\Updownarrow$$

$$\mathrm{He}\left\{\left[\begin{array}{c|c} A\mathbf{Q}+\frac{1}{2}B_wB_w^T & \mathbb{0} \\ \hline C_z\mathbf{Q} & -\frac{1}{2}\boldsymbol{\gamma}^2\mathbb{1} \end{array}\right]+\left[\begin{array}{c} B_u \\ D_{zu} \end{array}\right]\mathbf{Y}\left[\begin{array}{cc} \mathbb{1} & \mathbb{0} \end{array}\right]\right\} < \mathbb{0}$$

Without modifying any other declaration (block decomposition of the LMI, block declaration of other terms...) the **Y** dependent terms can be equivalently declared either by:

```
>> quiz = sdmineq(quiz, [lmi2index 1 1], Yindex,   Bu, 1);
>> quiz = sdmineq(quiz, [lmi2index 2 1], Yindex,  Dzu, 1);
```

or with the unique command line:

```
>> quiz = sdmineq(quiz, lmi2index, Yindex, [Bu; Dzu], [eye(n), zeros(n,p)]);
```

The usage of `sdmineq` without the block partitioning may be useful for LMI constraints where only part of the terms are described with a block partitioning and the others do not explicitely respect this partitioning.

Any LMI constraint can be seen either with a block partitioning of without. In the sequel, this block partitioning is sometimes avoided not to have complicated notations.

3.2.2 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 (without describing the possible block partitioning):

$$L(\mathbf{x}) < \mathbb{0}$$

The command

```
>> quiz = sdmineq(quiz, [LMIindex blrow blcol], Xindex, L, R);
```

modifies the constraint into:

$$L(\mathbf{x}) + \left[\begin{array}{c|c|c} \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & L\mathbf{X}R + R^*\mathbf{X}^*L^* & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} \end{array}\right] < \mathbb{0} \qquad \text{or} \qquad L(\mathbf{x}) + \left[\begin{array}{c|c|c|c|c} \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} & L\mathbf{X}R & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & R^*\mathbf{X}^*L^* & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \end{array}\right] < \mathbb{0}$$

(Diagonal blocks) (Other blocks)

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

```
quiz = sdmineq(quiz,[-LMIindex blrow blcol], Xindex, L, R);
```

modifies the constraint into:

$$L(\mathbf{x}) < \left[\begin{array}{c|c|c} \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & L\mathbf{X}R + R^*\mathbf{X}^*L^* & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} \end{array}\right] \qquad \text{or} \qquad L(\mathbf{x}) < \left[\begin{array}{c|c|c|c|c} \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} & L\mathbf{X}R & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & R^*\mathbf{X}^*L^* & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \hline \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \end{array}\right]$$

(Diagonal blocks) (Other blocks)

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

- **left**
  If the first entry of the second input argument of `sdmineq` is positive (`+LMIindex`) the term is added in the `LMIindex` constraint, to the left-hand side of the inequality sign $<$.

- **right**
  If the first entry of the second input argument of `sdmineq` is negative (`-LMIindex`) the term is added in the `LMIindex` constraint, to the right-hand side of the inequality sign $<$.

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

```
>> quiz = sdmineq(quiz, +LMIindex, 0, -L, R);

>> quiz = sdmineq(quiz, -LMIindex, 0, +L, R);
```

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

$$L(\mathbf{x}) - LR - R^*L^* \leq R(\mathbf{x}) \qquad\qquad L(\mathbf{x}) \leq R(\mathbf{x}) + LR + R^*L^*$$

The two resulting constraints are identical.

3.2.3 Transpose of some matrix variable ($L\mathbf{X}^T R$)

By default, a new declared term depends linearly on the matrix variable such as in $L\mathbf{X}R$. The conjugate transpose term $R^*\mathbf{X}^*L^*$ being automatically added, there is no point in declaring terms that depend on the conjugate transpose of the matrix variable. In the case when $\mathbf{X}$ is real there is therefore no need have a functionality to add a term that depends on the transpose of $\mathbf{X}$ ($\mathbf{X}^T = \mathbf{X}^*$). Nevertheless, when $\mathbf{X}$ is complex valued it may be useful to declare terms that depend on the transpose (not conjugate) of some variable. SeDuMi Interface allows the user to do so. In order to declare a term depending on the transpose of a matrix variable, such as $L\mathbf{X}^T R$, 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: $L\mathbf{X}R$ (and $R^*\mathbf{X}^*L^*$ is added to the symmetric block).

- **transposed**
  If the third input argument of `sdmineq` is negative (`-Xindex`) the term depends on the transpose of the variable ($\mathbf{X}^T$) such as in: $L\mathbf{X}^T R$ (and $R^*\mathbf{X}L^*$ is added to the symmetric block).

Following the rule, the two next command lines are equivalent if the variable is real and are not if it is complex valued:

```
>> quiz = sdmineq(quiz, [LMIindex blrow blcol], +Xindex, L, R);

>> quiz = sdmineq(quiz, [LMIindex blcol blrow], -Xindex, R', L');
```

3.2.4 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. 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.

3.2.5 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.

3.2.6 Hermitian terms ($L\mathbf{X}L^*$ and LL^*)

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

$$L\mathbf{X}L^* \qquad\qquad LL^*$$

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

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

- **Hermitian**
  If 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). This can only be done if the block is on the diagonal and the variable $\mathbf{X}$ is Hermitian.

To show that this syntax simplifies some notations, take the constant term in the LMC problem of the previous section ($B_wB_w^T = \mathrm{He}\{\frac{1}{2}B_wB_w^T\}$). This term can be equivalently declared with the two following syntaxes:

```
>> quiz = sdmineq(quiz, [lmi2index 1 1], 0, Bw, 0.5*Bw');

>> quiz = sdmineq(quiz, [lmi2index 1 1], 0, Bw);
```

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

3.2.7 Terms with no multiplying data

Quite often, some terms of the constraints depend directly on a matrix variable without any dependency on the data of the problem. A famous example is the Lyapunov matrix. In all control problems the Lyapunov matrix is constrained to be definite positive ($\mathbf{Q} > \mathbb{0}$). In the example of the previous section, this specification is translated into $\mathrm{He}\{-\frac{1}{2}\mathbf{Q}\} < \mathbb{0}$. 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 $\mathbf{Q} > \mathbb{0}$ has the two equivalent syntaxes:

```
>> quiz = sdmineq(quiz, +lmi1index, Qindex, -0.5, 1);

>> quiz = sdmineq(quiz, -lmi1index, Qindex);
```

3.2.8 Terms with Kronecker products ($L(K\otimes\mathbf{X})R$ and $L(\mathbf{X}\otimes K)R$)

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 frequently 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:

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

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 term $L(K\otimes\mathbf{X})R$ is added to the `(blrow,blcol)` block and the Hermitian term $(L(K\otimes\mathbf{X})R)^*$ is added to the symmetric `(blcol,blrow)` block.
- **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 term $L(\mathbf{X}\otimes K)R$ is added to the `(blrow,blcol)` block and the Hermitian term $(L(\mathbf{X}\otimes K)R)^*$ is added to the symmetric `(blcol,blrow)` block.

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

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

These two inequality constraints share the common expression:

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

with $K=\left[\begin{array}{cc}0&1\\1&0\end{array}\right]$ and $K=\left[\begin{array}{cc}-1&0\\0&1\end{array}\right]$ 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:

```
>> quiz = sdmineq(quiz, LMIindex, Pindex, [eye(n),A'], [], K);
```

3.3 Equality constraints

Equality constraints can be defined quite in the same way as the inequality constraints. The main difference is that equality constraints may not be square and may not be Hermitian. For this user's guide the following constraint is used. It is a purely fictitious constraints with no relation the the H_∞ problem in automatic control.

$$\begin{array}{c} \\ 2\updownarrow \\ 1\updownarrow \end{array} \begin{array}{c} \begin{array}{cc} 3 & 1 \\ \leftrightarrow & \leftrightarrow \end{array} \\ \left[\begin{array}{cc} \mathbf{K} & \mathbb{0} \\ \mathbb{0} & \mathbf{Q}_{(\mathbf{1},\mathbf{1})} \end{array}\right] \end{array} = \left[\begin{array}{cc} \mathbf{H}\left[\begin{array}{ccc} i & i & 0 \\ 0 & 0 & 1 \end{array}\right] + \left[\begin{array}{ccc} 2i & i & 0 \\ 0 & 0 & 0 \end{array}\right] & \mathbb{0} \\ \mathbb{0} & \bar{\mathbf{D}}_{(\mathbf{1},\mathbf{1})} - 1 + i \end{array}\right]$$

3.3.1 Defining an equality constraint: `sdmlme`

To initialise an equality constraint the usage is:

```
>> [quiz, lmeindex] = sdmlme(quiz, [2 1], [3 1], '[K 0;0 Q] = [H 0;0 conj(D11)-1+i]');
```

The first input argument is the `sdmpb` object to append. The second and third input arguments are vectors containing respectively row block dimension and column block dimension in the equality constraint. The last input argument is an optional label used for nice display. The appended `sdmpb` object is returned along with an index, `lmeindex`, that makes reference to the declared constraint.

At this step the LMC problem is described in table 5.

```
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
                         4               D : diagonal
                         5               G
                         6               H
                         7               J
                         8               [D, G]
                         9               K
equality constraints:    index   norm    name
                         1       -Inf    [K 0;0 Q11] = [H 0;0 conj(D11)-1+i]
inequality constraints:  index   meig    name
                         1       -Inf    Q>0
                         2       -Inf    Hinfty state-feedback
maximise objective:-gamma^2
unsolved
```

Table 5: Completely declared LMC problem

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

```
>> get(quiz, 'eqnb')
ans =
     1
get(quiz, 'eqdimrow', lmeindex)
ans =
     2     1
get(quiz, 'eqdimcol', lmeindex)
ans =
     3     1
>> get(quiz, 'eqname', lme1index)
ans =
[K 0;0 Q11] = [H 0;0 conj(D11)-1+i]
```

3.3.2 Add a term to an equality constraint: `sdmeq`

All terms in an LME constraint are declared with `sdmeq` as follows:

```
quiz = sdmeq(quiz, [LMEindex blrow blcol], Xindex, L, R);
```

This command adds to the `(blrow,blcol)` block of the equality constraint referenced by the index `LMEindex` a term $L\mathbf{X}R$ where $\mathbf{X}$ is the matrix variable referenced by the index `Xindex`. If the index is equal to 0 a constant term LR is added.

Note that for the equalities there is no necessity for the constraints to be Hermitian. Therefore, nothing similar to the Hermitian terms added in LMIs is performed when adding a term to an LME.

On the other hand, as for the inequality constraints, the generic rule for defining LME terms has advanced usages. These are now described shortly.

Block partitioning

- **specifying blocks**
  When the second input argument of `sdmeq` is composed of three entries (`[LMEindex blrow blcol]`) then the term $L\mathbf{X}R$ is added to the `(blrow,blcol)` block (same comment for constant terms LR).

- **not specifying block**
  If the second input argument of `sdmeq` is a scalar (`LMEindex`) then the term $L\mathbf{X}R$ is added to the entire LME (same comment for constant terms LR). Note that in this case the number of rows of L must be equal to the number of rows of the whole LME (sum of row dimensions of the blocks) and the number of columns of R should be equal to the number of columns of the whole LME (sum of column dimensions of the blocks).

Left ad right sides of the equality

- **left**
  If the first entry of the second input argument of `sdmeq` is positive (`+LMEindex`) the term is added in the `LMEindex` constraint, to the left-hand side of the inequality sign $=$.

- **right**
  If the first entry of the second input argument of `sdmeq` is negative (`-LMEindex`) the term is added in the `LMEindex` constraint, to the right-hand side of the inequality sign $=$.

Transpose and conjugate of some matrix variable

- **not transposed, not conjugate**
  If the third input argument of `sdmineq` is positive and real (`+Xindex`) the term depends on the variable ($\mathbf{X}$) such as in: $L\mathbf{X}R$.
- **transposed, not conjugate**
  If the third input argument of `sdmineq` is negative and real (`-Xindex`) the term depends on the transpose of the variable ($\mathbf{X}^T$) such as in: $L\mathbf{X}^T R$.
- **not transposed, conjugate**
  If the third input argument of `sdmineq` is positive and purely imaginary (`+i*Xindex`) the term depends on the conjugate of the variable ($\overline{\mathbf{X}}$) such as in: $L\overline{\mathbf{X}}R$.
- **transposed, conjugate**
  If the third input argument of `sdmineq` is negative and purely imaginary (`-i*Xindex`) the term depends on the conjugate transpose of the variable ($\mathbf{X}^*$) such as in: $L\mathbf{X}^*R$.

Hermitian terms ($L\mathbf{X}L^*$ and LL^*)

- **not Hermitian**
  If both the fourth (L) and the fifth (R) input arguments are declared and are non empty, then the added term is $L\mathbf{X}R$ or LR (variable dependent term or constant term, respectively).
- **Hermitian**
  If 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). For LME constraints this advanced usage does not require the term to be added in a diagonal block.
- **Elementary**
  If both the fourth and the fifth input arguments (L and R) are omitted, then the added term is $\mathbf{X}$.

Terms with Kronecker products ($L(K\otimes\mathbf{X})R$ and $L(\mathbf{X}\otimes K)R$)

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

All these rules can be applied together. They are applied for the two examples of equalities assumed at the start of section 3.3.

A first declaration writes as:

```
quiz = sdmeq(quiz, [lmeindex 1 1],   Kindex,          1,                1);
quiz = sdmeq(quiz, [lmeindex 1 1],   Hindex,         -1, [ i i 0 ; 0 0 1]);
quiz = sdmeq(quiz, [lmeindex 1 1],        0,         -1, [2i i 0 ; 0 0 0]);

quiz = sdmeq(quiz, [lmeindex 2 2],   Qindex,  [1 0 0 0],  [1 ; 0 ; 0 ; 0]);
quiz = sdmeq(quiz, [lmeindex 2 2],        0,         -1,             1-i);
quiz = sdmeq(quiz, [lmeindex 2 2], i*Dindex,   -[1 0 0],      [1 ; 0 ; 0]);
```

An equivalent version, but more involved, writes as:

```
quiz = sdmeq(quiz, [ lme1index 1 1],   Kindex);
quiz = sdmeq(quiz, [-lme1index 1 1],   Hindex,         1, [ i i 0 ; 0 0 1]);
quiz = sdmeq(quiz, [-lme1index 1 1],        0,         1, [2i i 0 ; 0 0 0]);

quiz = sdmeq(quiz, [ lme2index 2 2],   Qindex, [1 0 0 0]);
quiz = sdmeq(quiz, [ lme2index 2 2],        0,         1,              1-i);
quiz = sdmeq(quiz, [-lme2index 2 2], i*Dindex,   [1 0 0]);
```

Analysis of the computed solution

To analyse the obtained point, the `norm` data allow to check that every equality constraint is satisfied. These data are displayed with the LMC problem (table 6) and can also be obtained with the `get` function as:

```
>> get(quiz, 'eqnorm', lme1index)
ans =
eps
```

```
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2
                         3               Q
                         4               D : diagonal
                         5               G
                         6               H
                         7               J
                         8               [D, G]
                         9               K
equality constraints:    index   norm    name
                         1       eps     [K 0;0 Q11] = [H 0;0 conj(D11)-1+i]
inequality constraints:  index   meig    name
                         1       eps     Q>0
                         2       eps     Hinfty state-feedback
maximise objective:-gamma^2 = -27.3
LMIs are feasible
```

Table 6: Solved LMC problem

The `norm` data correspond to the norm of each equality constraint evaluated on the point obtained by SEDUMI. The constraints are satisfied if their minimal norm is zero. The `norm` data can have different values:

- `-Inf` : occurs when the LMC problem has not been solved (as in table 5).
- `0` : occurs when the constraint is strictly equal to zero.
- `eps` : occurs when the constraint is "close" to zero. The SEDUMI algorithm has an accuracy set by default to 10^{-9}. All constraints with a norm less than this accuracy level are assumed to be "equal" to zero. The `norm` data is set to `eps` in SEDUMI INTERFACE.
- positive scalar : occurs when the equality constraint is not satisfied.

3.4 The trace as an objective (max **trace**(**BX**))

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:

$$\max \ \text{trace}(\mathbf{BX})$$

SEDUMI INTERFACE can declare directly the trace of a matrix variable. The rule is:

trace objective
If the third and fourth input arguments of `sdmobj` are respectively the string `'tr'` and a matrix (or a scalar) (B), the term added to the maximisation objective is : $\text{trace}(\mathbf{BX})$. With the help of this syntax, the objective can be declared in one line as follows:

```
>> quiz = sdmobj(quiz, Xindex, 'tr', B, 'trace(BX)');
>> get(quiz, 'objname')
ans =
 trace(BX)
```

Remark that if the dimensions fit, the following relation holds:

$$\text{trace}(AC) = \text{trace}(CA)$$

Therefore, to declare a term such as $\text{trace}(C\mathbf{X}C^T)$ one can declare $\text{trace}(B\mathbf{X})$ with $B = C^T C$.

3.5 Set a value to some variable: `sdmset`

In some cases the user may want to solve an LMC 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:

```
>> quiz = sdmset(quiz, 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. Having set a variable to some value, the indexes of the other variables are not modified. The equalities, 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 constraints.

To illustrate the use of `sdmset` let us consider the H_∞ state-feedback problem of the previous section. The LMC problem was defined in order to minimise the H_∞ cost and is displayed by SEDUMI INTERFACE as shown in table 6. One can expect that since the optimal value is $\gamma^{opt} = \sqrt{27.2878}$, the LMC problem also has a solution if γ is set to a higher value, for example $\gamma = \sqrt{30}$. To check this, set the value of $\boldsymbol{\gamma}$ and solve the new LMC problem as shown in table 7.

Note that the LMC 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 **F** defined as the concatenation of two previously defined variables:

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

```
>> quiz = sdmset(quiz, gindex, 30);
>> quiz = sdmsol(quiz);
...
>> quiz
LMC problem: Optimal Hinfty State-Feedback Synthesis
matrix variables:        index           name
                         1               Y
                         2               gamma^2 SET TO A CONSTANT
                         3               Q
                         4               D : diagonal
                         5               G
                         6               H
                         7               J
                         8               [D, G]
                         9               K
equality constraints:    index   norm    name
                         1       eps     [K 0;0 Q11] = [H 0;0 conj(D11)-1+i]
inequality constraints:  index   meig    name
                         1       0.001   Q>0
                         2       0.001   Hinfty state-feedback
maximise objective:-gamma^2 = 0
feasible
>> quiz(gindex)
warning : index refers to a variable that was set to a constant
ans =
    30
```

Table 7: Same LMC problem with a frozen value of γ

The question is: *What happens to the variable* **F** *if the value of* **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 **G** is frozen, its structure matrix becomes empty, while the variable **F** still depends on the same decision variables (see table 8).

```
>> quiz = sdmset(quiz, Gindex, [0 -1 1 ; 1 0 2 ; -1 -2 0]);
>> quiz{Gindex}
Warning: index refers to a variable that was set to a constant
ans =
     []
>> num2str(quiz{Findex})
ans =
15-15i    0+0i     0+0i     0+0i   -18+0i   -19+0i
 0+0i    16-16i    0+0i    18+0i     0+0i   -20+0i
 0+0i     0+0i    17-17i   19+0i    20+0i     0+0i
```

Table 8: Structure of **G** and **F** when **G** is set to a constant

3.6 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 9.

```
>> quiz = sdmsol(quiz)
...
LMC 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 9: Optimally solved LMC 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.

3.6.1 SeDuMi parameters

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

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

The optional second input argument of `sdmsol` must have the structure adopted by SEDUMI [25]. 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 10 shows the computation result.

```
>> pars.eps=1e-12;
>> quiz=sdmsol(quiz,pars)
...
LMC problem: H2/Hinfty state-feedback synthesis
matrix variables:        index   name
                         1       X
                         2       T
                         3       S
no equality constraint
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 10: Same LMC problem solved with a smaller precision parameter

As can be seen, SEDUMI failed to find a feasible solution satisfying the new precision. In fact, 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 LMC 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(quiz,'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 [25]. The last field `numerr: 1` is non-zero in this example. This explicitly shows that some numerical error has occurred.

3.6.2 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:

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

This operation transforms 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 LMC 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 11.

The result is satisfying: all eigenvalues of the constraints in the LMC problem `quiz2` are positive or

```
>> quiz2 = quiz - 1e-6;

>> quiz2 = sdmsol(quiz2)
...
LMC problem: H2/Hinfty state-feedback synthesis
matrix variables:        index   name
                         1       X
                         2       T
                         3       S
no equality constraint
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 11: Same LMC problem solved with α-translated constraints

at least greater than the accuracy level -10^{-9}. This means that the solution is acceptable for the LMC problem `quiz` with all eigenvalues superior or equal to $\alpha - \varepsilon = 10^{-6} - 10^{-9} > 0$.

3.6.3 Feasibility radius

Another and last way to tune the solver convergence without modifying strongly the LMC problem, is to impose a feasibility radius. It constrains the norm of the vector of decision variables. The convexity of the LMC 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 12.

```
>> quiz=sdmsol(quiz,[],1e9)
...
LMC problem: H2/Hinfty state-feedback synthesis
matrix variables:        index   name
                         1       X
                         2       T
                         3       S
no equality constraint
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 12: Same LMC problem solved with a feasibility radius

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 radius

is appropriately chosen. To guide the user in choosing this radius we recommend to solve the LMC 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(quiz,'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 "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:

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

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, if `A` is a real valued matrix,

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

is a command that adds a term such as:

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

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

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

that declares a term such that:

$$L(\mathbf{x}) + (-A)\mathbf{X}(-A^T) \;\leq\; R(\mathbf{x}) \qquad \text{that is} \qquad L(\mathbf{x}) + A\mathbf{X}A^T \;\leq\; R(\mathbf{x})$$

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:

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

They respectively declare the following terms (if `B` is real):

$$L(\mathbf{x}) \;\leq\; R(\mathbf{x}) + BB^T \qquad \text{and} \qquad L(\mathbf{x}) + (-B)(-B)^T \;\leq\; R(\mathbf{x})$$

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

4.2 Matrix and scalar multipliers for inequality terms

A second warning concerns the use of scalar multipliers in `sdmineq`. As described previously, SEDUMI INTERFACE allows to declare terms such as:

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

These commands add terms such that (assume all matrices are real):

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

Therefore the three following notations are equivalent (**R** is assumed to be symmetric):

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

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

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

5 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 possibly pass on some examples to illustrate this report. These remarks and contributions may influence the future developments and make the tool more complete. For example, since SEDUMI INTERFACE 1.02, both complex LMIs and complex variables have been included to the tool, since SEDUMI INTERFACE 1.03, linear matrix equalities can be added to the LMI constraints and since SEDUMI INTERFACE 1.04, block partitioning, maximisation of $Trace(BX)$ are available.

Among future evolutions are:

- Concatenation.
  The concatenation of several LMC problems can be viewed as the optimisation problem constructed with all the linear matrix constraints of each LMC problem and where some of the matrix variables are common. Such an operator can help for optimisation problems defined over the intersection of feasible domains.

- Translator.
  Following the example of [19, 24, 9], the tool can include some translator functionalities in order to call other solvers when the translation is admissible.

Acknowledgements

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: `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", Tech. report n. ANL/MCS-P851-1000, November 2000, Argonne National Laboratory.

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

[4] B. Borchers, "CSDP, 2.3 User's Guide", *Optimization Methods and Software*, vol. 11, n. 1, 1999, pages 597-611.

[5] B. Borchers, "CSDP, A C Library for Semidefinite Programming", *Optimization Methods and Software*, vol. 11, n. 1, 1999, pages 613-623.

[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: `www.cs.uiowa.edu /~brixius/SDPHA`.

[8] M. Chilali, "Méthodes LMI pour l'Analyse et la Synthèse Multi-Critère", PhD thesis, Paris IX, Dauphine, February 1996.

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

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

[11] 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*, vol. 42, n. 8, 1997, pages 1171-1176.

[12] 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`.

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

[14] 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.

[15] T. IWASAKI, "LPV System Analysis with Quadratic Separator for Uncertain Implicit Systems", *LMI Methods in optimisation, identification and control*, Seminar, Compiègne, France, May 1999.

[16] 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: www.imm.dtu.dk /~sk/cutsdp.

[17] Y. LABIT, D. PEAUCELLE and D. HENRION, "SeDuMi interface 1.02: a tool for solving LMI problems with SeDuMi", proceedings of *CACSD*, Glasgow, Scotland, September 2002, à paraître.

[18] F. LEIBFRITZ, "An LMI-based algorithm for designing suboptimal static H_2/H_∞ output feedback controllers.", *SIAM Journal on Control and Optimization*, vol. 39, n. 6, 2001, pages 1711 - 1735.

[19] J. LÖFBERG, "YALMIP: Yet Another LMI Parser", August 2001, URL: www.control.isy.liu.se /~johanl/yalmip.html.

[20] H.D. MITTELMANN, "An Independent Benchmarking of SDP and SOCP Solvers", Tech. report, July 2001, Arizona State University, URL: www.optimization-online.org /DB_HTML/2001/07/358.html.

[21] D. PEAUCELLE and D. ARZELIER, "An Efficient Numerical Solution for H_2 Static Output Feedback Synthesis", proceedings of *European Control Conference*, Porto, Portugal, September 2001, pages 3800-3805.

[22] D. PEAUCELLE, D. HENRION and Y. LABIT, "User's Guide for SEDUMI INTERFACE 1.01", Optimization Online, November 2001, URL: www.optimization -online.org /DB_HTML/2001 /11/398.html.

[23] D. PEAUCELLE, D. HENRION and Y. LABIT, "User's Guide for SEDUMI INTERFACE 1.01: Solving LMI problems with SEDUMI", Tech. report n. 01445, October 2001, LAAS-CNRS, Toulouse, France, URL: www.laas.fr /~peaucell/SeDuMiInt.html.

[24] PETE SEILER, "LMIlab Translator", Tech. report, July 2001, University of Berkeley, California, URL: vehicle.me.berkeley.edu /~guiness/lmitrans.html.

[25] J.F. STURM, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones", *Optimization Methods and Software*, vol. 11-12, 1999, pages 625-653, URL: fewcal.kub.nl /sturm/software/sedumi.html.

[26] 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*, vol. 11, 1999, pages 545-581, URL: www.math.nus.edu.sg /~mattohkc/index.html.

[27] J.G. VANANTWERP and R.D. BRAATZ, "A Tutorial on Linear and Bilinear Matrix Inequalities", *J. Process Control*, vol. 10, 2000, pages 363-385.

[28] L. VANDENBERGHE and S. BOYD, "SP : Softvare for semidefinite programming. User's guide, beta version", Tech. report, 1994, K.U. Leuven and Stanford University, URL: www.stanford.edu /~boyd/SP.html.

[29] S.-P. WU and S. BOYD, "Design and implementation of a Parser/Solver for SDPs with Matrix Structure", proceedings of *IEEE Conference on Computer Aided Control System Design*, New York, 1996.