# Appendix B: Using the MATLAB® LMI Toolbox

LMIs have been shown to provide a powerful control design tool. For solution of LMIs, there are basically two types of effective methods, namely, the ellipsoid methods and the interior point methods.

Based on the interior-point method, Gahinet, Nemirovskii, and some other researchers produced a software package called LMI Toolbox, which is for use with MATLAB®. This LMI toolbox is designed to assist control engineers and researches with a user-friendly interactive environment. Through this environment, one can specify and solve many often encountered engineering problems involving LMIs.

In this appendix, we have given a very brief introduction to the MATLAB LMI toolbox to help the readers to get a quick start. For a much more detailed description, one needs to refer to the User's Guide for LMI Control Toolbox by Gahinet et al. (1995). One can also check with the help document for LMI Lab from the MATLAB window following the procedure:

$$\text{Help} \rightarrow \text{Product Help} \rightarrow \text{Robust Control Toolbox} \rightarrow \text{LMI Lab.}$$

This is convenient when you are really working on some LMI problems with MATLAB since it provides you with a sort of online help environment.

As is all know, for usage of a particular MATLAB command in the LMI toolbox, one can simply check with the help message in the command file simply by typing in the MATLAB working window "help M-file name."

## B.1 How to Describe LMIs?

The LMI toolbox can handle any system of LMIs of the form

$$N^{\mathrm{T}}L(X_1,\ldots,X_K)N < M^{\mathrm{T}}R(X_1,\ldots,X_K)M,$$

where $X_1,\ldots,X_K$ are the decision variables, $L(X_1,\ldots,X_K)$ and $R(X_1,\ldots,X_K)$ are linear functions and $M$, $N$ are matrices of appropriate dimensions.

The specification of an LMI system involves two steps:

1. Declare the dimensions and structure of each matrix variable.
2. Describe the term content of each LMI.

There are two ways of generating the internal description of a given LMI system:

- **Command-based description** by a sequence of `lmivar/lmiterm` commands that build it incrementally
- **LMI editor description** via the `lmiedit` where LMIs can be specified directly as symbolic matrix expressions

### B.1.1 Command-Based Description

The description of LMI problems in the MATLAB LMI toolbox involves several MATLAB commands.

#### `setlmis`

The description of an LMI system should begin with `setlmis` and end with `getlmis`. The function `setlmis` initializes the LMI system description. It can be used in the following two ways:

- `setlmis([])`: Resets the internal variables used for creating LMIs so that one can create a new system of LMIs.
- `setlmis(lmisys0)`: Uses an existing LMI system `lmisys0` as the starting point for creating a new system of LMIs. Subsequent commands will add to the LMI system `lmisys0` to construct a new set of LMIs.

#### `getlmis`

When the LMI system is completely specified, type

```
lmisys = getlmis
```

This returns the internal representation `lmisys` of this LMI system. The command `getlmis` must be used only once and after declaring all the matrix variables and all the LMI terms.

#### `lmivar`

After initializing the description with the command `setlmis`, the matrix variables are declared one at a time with `lmivar` and are characterized by their structure. The function `lmivar` can be used as follows:

- `X = lmivar(type, struct)` adds a new matrix variable `X` to the LMI system currently specified. You can use the identifier `X` for subsequent references to the variable `X` in calls to `lmiterm`.
- `[X,ndec,xdec] = lmivar(type, struct)` also returns the total number of decision variables associated with `X`, `ndec`, and the entry-wise dependence of `X` on these decision variables, `xdec`.

In command `lmivar`, the first input specifies the structure type and the second input contains additional information about the structure of the variable.

*Inputs:*

`type`: Structure of `X`, which takes values of 1, 2, or 3, as defined next:
When `type = 1`, `X` is symmetric block diagonal.
When `type = 2`, `X` is full rectangular.
When `type = 3`, `X` takes some other form.

`struct`: Additional data on the structure of `X`, which varies with the variable `type` as follows:

i. When `type = 1`, this second input `struct` is a matrix with two columns and as many rows as diagonal blocks. The first column lists the sizes of the diagonal blocks and the second column specifies their nature with the following convention:
`1` stands for full symmetric block.
`0` stands for scalar block.
`-1` stands for zero block.

ii. When `type = 2`, `struct = [m,n]` if `X` is an `m`×`n` matrix.

iii. When `type = 3`, `struct` is a matrix of the same dimension as `X`, where each entry is
`0` if `X(i,j) = 0`.
`+n` if `X(i,j) =` *n*th decision variable.
`-n` if `X(i,j)` = (−1) × *n*th decision variable.

*Outputs:*

`X`: Identifier for the new matrix variable. Its value is $k$ if $k-1$ matrix variables have already been declared. This identifier is not affected by subsequent modifications of the LMI system.

`nedc`: Total number of decision variables.

`xdec`: Entry-wise dependence of `X` on the decision variables.

## `lmiterm`

`lmiterm(termid,A,B,flag)` adds one term to some LMI in the LMI system currently specified.

*Inputs:*

`termid`: four-entry vector specifying the term location and nature.

i. The first entry specifies which LMI.
   `+n`: Left-hand side of the $n$th LMI.
   `-n`: right-hand side of the $n$th LMI.

ii. The next two entries state which block. For outer factors, set `termid (2:3) = [0 0]`; otherwise, set `termid (2:3) = [i j]` if the term belongs to the `(i,j)` block of the LMI.

iii. The last one is what type of term.
   `0`: Constant term.
   `X`: Variable term AXB.
   `-X`: variable term $AX^TB$.
   In the earlier points, `X` is a matrix variable by `lmivar`.

`A`: Value of the outer factor, constant term, or left coefficient in variable terms AXB or $AX^TB$.

`B`: Right coefficient in variable terms AXB or $AX^TB$.

`flag`: Quick way of specifying the expression $AXB+(AXB)^T$ in a diagonal block. Set `flag = 's'` to specify it with only one `lmiterm` command.

Two examples are given as follows:

**Example B.1**

Given

$$A_1 = \begin{bmatrix} -1 & 2 \\ 1 & -3 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -0.8 & 1.5 \\ 1.3 & -2.7 \end{bmatrix}, \quad A_3 = \begin{bmatrix} -1.4 & 0.9 \\ 0.7 & -2.0 \end{bmatrix},$$

describe the following LMIs in MATLAB:

$$\begin{cases} A_1^T P + P A_1 < 0 \\ A_2^T P + P A_2 < 0 \\ A_3^T P + P A_3 < 0 \\ I < P. \end{cases}$$

The MATLAB file for solving the aforementioned problem is given as follows:

```
% specify parameter matrices
A1 = [-1 2;1 -3]; A2 = [-0.8 1.5;1.3 -2.7];
A3 = [-1.4 0.9;0.7 -2.0];
% define the LMI system
```

```
setlmis ([]) % initialing
P = lmivar (1, [2 1]); % declare the LMI variable
lmiterm ( [1 1 1 P],1, A1, 's'); % #1 LMI, left-hand side
lmiterm ( [2 1 1 P],1, A2, 's'); % #2 LMI, left-hand side
lmiterm ( [3 1 1 P],1, A3, 's'); % #3 LMI, left-hand side
lmiterm ( [-4 1 1 P],1,1); % #4 LMI, right-hand side
lmiterm ( [4 1 1 0],1); % #4 LMI, left-hand side
lmis1 = getlmis; %finishing description
```

**Example B.2**

Given

$$A = \begin{bmatrix} -1 & -2 & 1 \\ 3 & 2 & 1 \\ 1 & -2 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

describe the following LMIs in MATLAB:

$$\begin{cases} \begin{bmatrix} A^T X + XA + Q & XB \\ B^T X & -I \end{bmatrix} < 0 \\ 0 < X \\ 0 < Q. \end{cases}$$

The MATLAB file for solving the aforementioned problem is given as follows:

```
% specify parameter matrices
A = [-1 -2 1;3 2 1;1 -2 -1]; B = [1; 0; 1];
% define the LMI system
setlmis ([]) %initialing
X=lmivar (1, [3 1]); % declare the LMI variable
Q=lmivar (1, [3 1]); % declare the LMI variable
lmiterm ( [1 1 1 X],1, A, 's'); % #1 LMI, the (1,1) block
lmiterm ( [1 1 1 Q],1,1); % #1 LMI, (1,1) block
lmiterm ( [1 2 2 0],-1); % #1 LMI, the (2,2) block
lmiterm ( [1 2 1 X],B',1); % #1 LMI, the (2,1) block
lmiterm ( [-2 1 1 X],1,1); % #2 LMI, right-hand side
lmiterm ( [-3 1 1 Q],1,1); % #3 LMI, right-hand side
lmis2 = getlmis; % finishing description
```

### B.1.2 LMI Editor Description

The LMI editor `lmiedit` is a graphical user interface (GUI) to specify LMI systems in a straightforward symbolic manner. Typing

```
lmiedit
```

calls up a window with several editable text areas and various push-buttons. To specify your LMI system,

1. Declare each matrix variable (name and structure) in the upper half of the worksheet. The structure is characterized by its type and by an additional "structure" matrix. This matrix contains specific information about the structure and corresponds to the second argument of `lmivar`.
2. Specify the LMIs as MATLAB expressions in the lower half of the worksheet.

Though somewhat less flexible and powerful than the command-based description, the LMI editor is more straightforward to use, hence particularly well-suited for beginners. Thanks to its coding and decoding capabilities, it also constitutes a good tutorial introduction to `lmivar` and `lmiterm`.

## B.2 How to Modify LMIs?

Once specified, a system of LMIs can be modified in several ways with the functions `dellmi`, `delmvar`, and `setmvar`.

#### `dellmi`

The first possibility is to remove an entire LMI from the system with `dellmi`. The format is as follows:

```
newsys = dellmi(lmisys,lmid)
```

It removes the LMI with identifier `lmid` from the system of LMIs described in `lmisys`. `lmid` is the ranking of the LMI in the LMI system initially created with `setlmis/getlmis`. To easily keep track of LMIs after deletions, set `lmid` to the identifier returned by `newsys` when this LMI is created.

#### `delmvar`

Another way of modifying an LMI system is to delete a matrix variable, that is, to remove all variable terms involving this matrix variable. This operation is

performed by `delmvar`. Note that `delmvar` automatically removes all LMIs that depended only on the deleted matrix variable. The format is as follows:

```
newsys = delmvar(lmisys,xid)
```

It deletes the matrix variable with identifier `xid` from the LMI system `lmisys`. The updated LMI system, `newsys`, is returned. For safety and to easily keep track of modifications, set `xid` to the value returned by `lmivar` when the matrix variable was created.

### `setmvar`

The function `setmvar` is used to set a matrix variable to some given values. As a result, this variable is removed from the problem and all terms involving it become constant terms. The format is as follows:

```
newsys = setmvar(lmisys,xid,xval)
```

It sets the matrix variable with identifier `xid` to the value `xval`. All LMI terms involving this matrix variable are evaluated and added to the constant terms. Since `setmvar` does not alter the variable identifiers, the remaining matrix variables can still be referred to by their original identifier.

*Inputs:*

`lmisys`: Array describing the system of LMIs.

`xid`: Identifier of the variable matrix to be set.

`xval`: Matrix value assigned to the matrix variable `xid` (a scalar $t$ is interpreted as $t * I$).

*Output:*

`newsys`: Updated description of the LMI system.

## B.3 How to Retrieve Information?

Recalling that the full description of an LMI system is stored as a single vector called the internal representation, the user should not attempt to read or retrieve information directly from this vector. Three functions called `lmiinfo`, `lminbr`, and `matnbr` are provided to extract and display all relevant information in a user-readable format.

### `lmiinfo`

`Lmiinfo` is an interactive facility to retrieve qualitative information about LMI systems. This includes the number of LMIs, the number of matrix variables and

their structure, the term content of each LMI block, etc. To invoke `lmiinfo`, enter

```
lmiinfo(lmisys)
```

where `lmisys` is the internal representation of the LMI system produced by `getlmis`.

### `lminbr` and `matnbr`

These two functions return the number of LMIs and the number of matrix variables in the system. To get the number of LMIs in LMI system `lmisys`, for instance, enter

```
nlmis = lminbr(lmisys)
```

To get the number of matrix variables in LMI system `lmisys`, type

```
[nmvars,varid] = matnbr(lmisys)
```

or

```
nmvars = matnbr(lmisys)
```

where `nmvars` is the number of matrix variables, and `varid` is the vector of matrix variable identifiers in the LMI system `lmisys`.

## B.4 How to Convert between Decision and Matrix Variables?

While LMIs are specified in terms of their matrix variables $X_1, \ldots, X_k$, the LMI solvers optimize the vector $x$ of free scalar entries of these matrices, called the decision variable. The two functions `mat2dec` and `dec2mat` perform the conversion between these two descriptions of the problem variables.

### `mat2dec`

Command `mat2dec` constructs decision variables vector from values of the matrix variables in the LMI systems. The format of this function is

```
xdec = mat2dec(lmisys,x1,x2,...,xn)
```

*Inputs:*

`lmisys`: The LMI system.

`x1,x2,...,xn`: The particular values of the matrix variables $X_1, X_2, \ldots, X_3$ of system `lmisys`. `mat2dec` accepts up to 20 matrix variables. An error is issued if some matrix variable remains unassigned.

*Outputs:*

`xdec`: The decision variables from matrix variable values.

### `dec2mat`

Command `dec2mat` extracts matrix variable value from vector of decision variables. The format of this function is

```
X = dec2mat(lmisys,xdec,xid)
```

*Inputs:*

`lmisys`: The LMI system.

`xdec`: The vector of decision variables.

`xid`: The identifier of matrix variable of LMI system `lmisys`.

*Outputs:*

`X`: The corresponding values of matrix variables with the identifier `xid`.

## B.5 How to Solve LMIs?

With the MATLAB LMI toolbox, one can solve three types of basic LMI problems with commands `feasp`, `mincx`, and `gevp`, respectively.

### B.5.1 `feasp` Command

The function `feasp` solves the feasibility problem, that is, Problem 3.7, which can be stated as follows.

*Find a solution $x \in R^n$ satisfying the following LMI problem*

$$A(x) < B(x).$$

The corresponding function in MATLAB is

```
[tmin,xfeas] = feasp(lmisys,options,target)
```

which solves the feasibility problem defined by the system `lmisys` of LMI constraints. When the problem is feasible, the output `xfeas` is a feasible value of the vector of (scalar) decision variables. The format is as follows:

*Inputs:*

`lmisys`: Array describing the system of LMI constraints.

`options`: (optional) Five-entry vector of control parameters. Default values are selected by setting options ($i$)=0:

`options(1)`: Not used.

`options(2)`: Maximum number of iterations (default = 100).

`options(3)`: Feasibility radius $R$. $R > 0$ constrains $x$ to $x^T x < R^2$ (default = $10^9$). $R < 0$ means "no bound."

`options(4)`: When set to an integer value $L > 1$, forces termination when $t$ has not decreased by more than 1% over the last $L$ iterations (default = 10).

`options(5)`: When nonzero, the trace of execution is turned off.

`target`: (optional) target for `tmin`. The code terminates as soon as $t <$ `target` (default = 0).

*Outputs:*

`tmin`: Value of $t$ upon termination. The LMI system is feasible if and only if `tmin` $\leq 0$.

`xfeas`: corresponding minimizer. If `tmin` $\leq 0$, `xfeas` is a feasible vector for the set of LMI constraints.

Given a feasibility problem of the form $A(x) < B(x)$, `feasp` solves the auxiliary convex programming:

$$\begin{cases} \min & t \\ \text{s.t.} & A(x) < B(x) + tI. \end{cases}$$

The system of LMIs is feasible if and only if the global minimum `tmin` is negative. The current best value of $t$ is displayed by `feasp` at each iteration.

The following example illustrates the use of the `feasp` solver.

**Example B.3**

Find a positive definite symmetric matrix $P$ such that the LMIs in Example B.1 hold.

The MATLAB commands for the LMI description (step 1) are given in Example B.1. Here, we only write the commands for the feasibility problem (step 2).

```
% Step 2. Solve this LMI problem and use dec2mat to
% get the corresponding matrix P
% solve the feasibility problem
[tmin,xfeas] = feasp(lmis1);
% extract the matrix P from the decision variable xfeas
P = dec2mat(lmis1,xfeas,P);
```

### B.5.2 `mincx` Command

The function `mincx` solves the convex minimization problem, that is, Problem 3.8, which has the following form:

$$\begin{cases} \min\limits_{x} & c^T x \\ \text{s.t.} & A(x) < B(x), \end{cases}$$

where

$x \in \mathbb{R}^n$ is the vector of decision variables

$c \in \mathbb{R}^n$ is a given vector

The corresponding function in MATLAB is

```
[copt,xopt] = mincx( lmisys,c,options,xinit,target)
```

*Inputs:*

`lmisys:` Description of the system of LMI constraints.

`c:` Vector of the same size as $x$. Use `defcx` to specify the objective $c^T x$ directly in terms of matrix variables.

`options:` (optional) Is similar to that in `feasp`.

`xinit:` (optional) Initial guess for $x$ ([] if none, ignored when unfeasible).

`target:` (optional) Target for the objective value. The code terminates as soon as a feasible $x$ is found such that $c^T x <$ `target`. (default = $-10^{20}$).

*Outputs:*

`copt:` Global minimum of the objective $c^T x$.

`xopt:` Minimizing value of the vector $x$ of decision variables.

The following example illustrates the use of the `mincx` solver.

**Example B.4**

Given the following linear system

$$\begin{cases} \dot{x} = Ax + B_2u + B_1w \\ z = Cx + Du, \end{cases}$$

where

$$A = \begin{bmatrix} -3 & -2 & 1 \\ 1 & 2 & 1 \\ 1 & -1 & -1 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad D = I_2,$$

design a state feedback control law $u = Kx$ and find the minimal positive scalar $\gamma_{\min}$ such that the closed-loop transfer function matrix satisfies $\|G_{zw}(s)\|_2 < \gamma_{\min}$.

According to (8.16), the $H_2$ control problem with minimum disturbance attenuation level $\gamma_{\min}$ can be turned into the following convex minimization problem:

$$\begin{cases} \min\limits_{X, Z, W, \rho} & \rho \\ \text{s.t.} & (AX + B_2W)^{\mathrm{T}} + AX + B_2W + B_1B_1^{\mathrm{T}} < 0 \\ & \begin{bmatrix} -Z & CX + DW \\ (CX + DW)^{\mathrm{T}} & -X \end{bmatrix} < 0 \\ & \text{trace}\,(Z) < \rho, \end{cases}$$

where $X$, $W$, and $Z$ are matrix variables to be solved, and the minimal value of $\sqrt{\rho}$ equals to $\gamma_{\min}$.

The MATLAB file for solving the aforementioned problem is given as follows:

```
% Step 1. Describe this LMIs in MATLAB
% specify parameter matrices
A = [-3 -2 1;1 2 1;1 -1 -1]; B1 = [1; 0; 1];
B2 = [2 0;0 2;0 1]; C = [1 0 1;0 1 0]; D = eye(2);
% define the LMI system
setlmis ([]) % initialing
% declare the LMI variable
X = lmivar(1, [3 1]); W = lmivar(2,[2,3]);
Z = lmivar(1, [2 1]); rou = lmivar(1, [1 1]);
% #1 LMI, left-hand side
lmiterm ( [1 1 1 X],A, 1, 's');
lmiterm ( [1 1 1 W],B2, 1, 's');
```

```
lmiterm ( [1 1 1 0],B1*B1');
% #2 LMI, left-hand side
lmiterm ( [2 1 1 Z],1,-1); %  the (1,1) block
lmiterm ( [2 1 2 W],D,1); %  the (1,2) block
lmiterm ( [2 1 2 X],C,1); %  the (1,2) block
lmiterm ( [2 2 2 X],1,-1); %  the (2,2) block
% #3 LMI, left-hand side
lmiterm ( [3 1 1 Z],[1 0],[1 0]');
lmiterm ( [3 1 1 Z],[0 1],[0 1]');
% #3 LMI, right-hand side
lmiterm ( [3 1 1 rou],1,-1);
lmis3=getlmis; % finishing description
% Step 2. Solve this LMI problem and use dec2mat to get the
% corresponding matrix X
c = mat2dec(lmis3,0,0,ones(2)); % obtain vector c
[copt,xopt] = mincx(lmis3,c); % solve the
  optimization problem
X = dec2mat(lmis3,xopt,1); % extract matrix X
W = dec2mat(lmis3,xopt,2); % extract matrix W
Z = dec2mat(lmis3,xopt,3); % extract matrix Z
K = W*inv(X);  % get the feedback gain
```

By the LMI Toolbox in MATLAB, we obtain the controller gain matrix

$$K = \begin{bmatrix} -1.0000 & -0.0000 & -1.0000 \\ -0.0000 & -1.0000 & 0.0000 \end{bmatrix}.$$

### B.5.3 gevp Command

The function gevp solves the generalized eigenvalue problem, that is, Problem 3.9, which has the following form:

$$\begin{cases} \min\limits_{x,\lambda} & \lambda \\ \text{s.t.} & A(x) < \lambda B(x) \\ & 0 < B(x) \\ & C(x) < D(x). \end{cases}$$

The corresponding format in MATLAB is

```
[tmin,xopt] = gevp(lmisys,nlfc,options,t0,x0,target)
```

*Inputs:*

`lmisys:` Description of the system of LMI constraints.

`nlfc:` Number of linear fractional constraints (LMIs involving $t$).

`options:` (optional) Is similar to that in `feasp`.

`t0, x0:` (optional) Initial guesses for $t$, $x$ (ignored when unfeasible).

`target:` (optional) Target for `tmin`. The code terminates as soon as $t$ falls below this value (default = $-10^5$).

*Outputs:*

`tmin:` Minimal value of $t$.

`xopt:` Minimizing value of the vector $x$ of decision variables.

**Example B.5**

Consider the following optimization problem

$$\begin{cases} \min\limits_{x} & \alpha \\ \text{s.t.} & I < P \\ & A_1^T P + P A_1 < \alpha P \\ & A_2^T P + P A_2 < \alpha P \\ & A_3^T P + P A_3 < \alpha P, \end{cases}$$

where $A_1$, $A_2$, and $A_3$ are given in Example B.1.

The MATLAB file for solving the aforementioned problem is given as follows:

```
%Step 1. Describe this LMIs in MATLAB
% specify parameter matrices
A1 = [-1 2;1 -3]; A2 = [-0.8 1.5;1.3 -2.7];
A3 = [-1.4 0.9;0.7 -2.0];
% define the LMI system
setlmis ([]) % initialing
P = lmivar (1, [2 1]); % declare the LMI variable
lmiterm ([1 1 1 0],1); % #1 LMI, left-hand
lmiterm ([-1 1 1 P],1,1); % #1 LMI, right-hand
lmiterm ([2 1 1 P],1, A1, 's'); % #2 LMI, left-hand
lmiterm ([-2 1 1 P],1,1); % #2 LMI, right-hand
lmiterm ([3 1 1 P],1, A2, 's'); % #3 LMI, left-hand
lmiterm ([-3 1 1 P],1,1); % #3 LMI, right-hand
lmiterm ([4 1 1 P],1, A3, 's'); % #4 LMI, left-hand
lmiterm ([-4 1 1 P],1,1); % #4 LMI, right-hand
lmis4 = getlmis % finishing description
```

```
%Step 2. Solve this LMI problem
[aa,xopt] = gevp(lmis4,3);
```

## B.6 How to Validate Results?

The LMI toolbox offers two functions to analyze and validate the results of an LMI optimization.

### evallmi

The function `evallmi` evaluates all variable terms in an LMI system for a given value of the vector of decision variables, for instance, the feasible or optimal vector returned by the LMI solvers. The format of this function is

```
evalsys = evallmi(lmisys,xdec)
```

where

`lmisys` is a created LMI system
`xdec` is the vector of decision variables

Recall that decision variables fully determine the values of the matrix variables. The "evaluation" consists of replacing all terms involving the matrix variables by their matrix value. The output `evalsys` is an LMI system containing only constant terms. The matrix values of the left and right sides of each LMI are then returned by `showlmi`.

### showlmi

Once the function `evallmi` is performed, the left- and right-hand sides of a particular LMI are returned by `showlmi`.

```
[lhs,rhs] = showlmi(evalsys,n)
```

*Inputs:*

`evalsys`: Array describing the set of evaluated LMIs (output of `evallmi`).

`n`: Label of the selected LMI as returned by `newlmi`.

*Outputs:*

`lhs`: The left-hand-side value of the *n*th LMI in the LMI system `lmisys`.

`rhs`: The right-hand-side value of the *n*th LMI in the LMI system `lmisys`.