LMITOOL: a Package for LMI Optimization in Scilab

User's Guide

R. Nikoukhah* F. Delebecque† L. El Ghaoui‡

Abstract

This report describes a user-friendly Scilab package, and in particular its two main functions `lmisolver` and `lmitool` for solving Linear Matrix Inequalities problems. This package uses Scilab function `semidef`, an interface to the program Semidefinite Programming **SP** (Copyright ©1994 by Lieven Vandenberghe and Stephen Boyd) distributed with Scilab.

Contents

*`Ramine.Nikoukhah@inria.fr`

†`Francois.Delebecque@inria.fr`

‡ENSTA, 32, Bvd. Victor, 75739 Paris, France. Internet: `elghaoui@ensta.fr`. Research supported in part by DRET under contract 92017-BC14.

1 Purpose

Many problems in systems and control can be formulated as follows (see [2]):

$$\Sigma: \quad \left\{ \begin{array}{ll} \text{minimize} & f(X_1,\ldots,X_M) \\ \text{subject to} & \left\{ \begin{array}{l} G_i(X_1,\ldots,X_M) = 0, \quad i = 1,2,...,p, \\ H_j(X_1,\ldots,X_M) \geq 0, \quad j = 1,2,..,q. \end{array} \right. \end{array} \right.$$

where

- $X_1,\ldots,X_M$ are unknown real matrices, referred to as the *unknown matrices,*
- f is a real linear scalar function of the entries of the unknown matrices $X_1,\ldots,X_M$; it is referred to as the *objective function*,
- G_i's are real matrices with entries which are affine functions of the entries of the unknown matrices, $X_1,\ldots,X_M$; they are referred to as "Linear Matrix Equality" (LME) functions,
- H_j's are real symmetric matrices with entries which are affine functions of the entries of the unknown matrices $X_1,\ldots,X_M$; they are referred to as "Linear Matrix Inequality" (LMI) functions. (In this report, the $V \geq 0$ stands for V positive semi-definite unless stated otherwise).

The purpose of `LMITOOL` is to solve problem Σ in a user-friendly manner in Scilab, using the code SP [1]. This code is intended for small and medium-sized problems (say, up to a few hundred variables).

2 Function `lmisolver`

`LMITOOL` is built around the Scilab function `lmisolver`. This function computes the solution $X_1,\ldots,X_M$ of problem Σ, given functions f, G_i and H_j. To solve Σ, user must provide an evaluation function which "evaluates" f, G_i and H_j as a function the unknown matrices, as well as an initial guess on the values of the unknown matrices. User can either invoke `lmisolver` directly, by providing the necessary information in a special format or he can use the interactive function `lmitool` described in Section 3.

2.1 Syntax

```
[XLISTF[,OPT]] = lmisolver(XLIST0,EVALFUNC[,options])
```

where

- `XLIST0:` a list structure including matrices and/or list of matrices. It contains initial guess on the values of the unknown matrices. In general, the ith element of `XLIST0` is the initial guess on the value of the unknown matrix X_i. In some cases however it is more convenient to define one or more elements of `XLIST0` to be lists (of unknown matrices) themselves. This is a useful feature when the number of unknown matrices is not fixed a priori (see Example of Section 2.2.2).

  The values of the matrices in `XLIST0`, if compatible with the LME functions, are used as intial condition for the optimization algorithm; they are ignored otherwise. The size and structure of `XLIST0` are used to set up the problem and determine the size and structure of the output `XLISTF`.

- `EVALFUNC:` a Scilab function called *evaluation function* (supplied by the user) which evaluates the LME, LMI and objective functions, given the values of the unknown matrices. The syntax is:

  ```
  [LME,LMI,OBJ]=EVALFUNC(XLIST)
  ```

  where

  - `XLIST:` a list, identical in size and structure to `XLIST0`.

- `LME:` a list of matrices containing values of the LME functions G_i's for X values in `XLIST`. `LME` can be a matrix in case there is only one LME function to be evaluated (instead of a list containing this matrix as unique element). It can also be a list of a mixture of matrices and lists which in turn contain values of LME's, and so on.
- `LMI:` a list of matrices containing the values of the LMI functions H_j's for X values in `XLIST`. `LMI` can also be a matrix (in case there is only one LMI function to be evaluated). It can also be a list of a mixture of matrices and lists which in turn contain values of of LMI's, and so on.
- `OBJ:` a scalar equal to the value of the objective function f for X values in `XLIST`.

If the Σ problem has no equality constraints then `LME` should be `[]`. Similarly for `LMI` and `OBJ`.

- `options:` a 5×1 vector containing optimization parameters `Mbound`, `abstol`, `nu`, `maxiters`, and `reltol`, see manual page for `semidef` for details (`Mbound` is a multiplicative coefficient for `M`). This argument is optional, if omitted, default parameters are used.
- `XLISTF:` a list, identical in size and structure to `XLIST0` containing the solution of the problem (optimal values of the unknown matrices).
- `OPT:` a scalar corresponding to the optimal value of the minimization problem Σ.

2.2 Examples

2.2.1 State-feedback with control saturation constraint

Consider the linear system

$$\dot{x} = Ax + Bu$$

where A is an $n \times n$ and B, an $n \times n_u$ matrix. There exists a stabilizing state feedback K such that for every initial condition $x(0)$ with $\|x(0)\| \leq 1$, the resulting control satisfies $\|u(t)\|$ for all $t \geq 0$, if and only if there exist an $n \times n$ matrix Q and an $n_u \times n$ matrix Y satisfying the equality constraint

$$Q - Q^T = 0$$

and the inequality constraints

$$\begin{aligned} Q &\geq 0 \\ -AQ - QA^T - BY - Y^T B^T &> 0 \\ \begin{pmatrix} u_{max}^2 I & Y \\ Y^T & Q \end{pmatrix} &\geq 0 \end{aligned}$$

in which case one such K can be constructed as $K = YQ^{-1}$.

To solve this problem using `lmisolver`, we first need to construct the evaluation function.

```
function [LME,LMI,OBJ]=sf_sat_eval(XLIST)
 [Q,Y]=XLIST(:)
 LME=Q-Q'
 LMI=list(-A*Q-Q*A'-B*Y-Y'*B',[umax^2*eye(Y*Y'),Y;Y',Q],Q-eye())
 OBJ=[]
```

Note that `OBJ=[]` indicates that the problem considered is a feasibility problem, i.e., we are only interested in finding a set of X's that satisfy LME and LMI functions.

Assuming `A`, `B` and `umax` already exist in the environment, we can call `lmisolver`, and reconstruct the solution in Scilab, as follows:

```
--> Q_init=zeros(A);
--> Y_init=zeros(B');
--> XLIST0=list(Q_init,Y_init);
--> XLIST=lmisolver(XLIST0,sf_sat_eval);
--> [Q,Y]=XLIST(:)
```

These Scilab commands can of course be encapsulated in a Scilab function, say `sf_sat`. Then, To solve this problem, all we need to do is type:

```
--> [Q,Y]=sf_sat(A,B,umax)
```

We call `sf_sat` the *solver function* for this problem.

2.2.2 Control of jump linear systems

We are given a linear system

$$\dot{x} = A(r(t))x + B(r(t))u,$$

where A is $n \times n$ and B is $n \times n_u$. The scalar parameter $r(t)$ is a continuous-time Markov process taking values in a finite set $\{1, \ldots, N\}$.

The transition probabilities of the process r are defined by a "transition matrix" $\Pi = (\pi_{ij})$, where π_{ij}'s are the transition probability rates from the i-th mode to the j-th. Such systems, referred to as "jump linear systems", can be used to model linear systems subject to failures.

We seek a state-feedback control law such that the resulting closed-loop system is mean-square stable. That is, for every initial condition $x(0)$, the resulting trajectory of the closed-loop system satisfies $\lim_{t\to\infty} \mathbf{E}\|x(t)\|^2 = 0$.

The control law we look for is a mode-dependent linear state-feedback, *i.e.* it has the form $u(t) = K(r(t))x(t)$; $K(i)$'s are $n_u \times n$ matrices (the unknowns of our control problem).

It can be shown that this problem has a solution if and only if there exist $n \times n$ matrices $Q(1), \ldots, Q(N)$, and $n_u \times n$ matrices $Y(1), \ldots, Y(N)$, such that

$$\begin{aligned} Q(i) - Q(i)^T &= 0, \\ \mathbf{Tr}Q(1) + \ldots + \mathbf{Tr}Q(N) - 1 &= 0. \end{aligned}$$

and

$$\begin{aligned} \begin{bmatrix} Q(i) & Y(i)^T \\ Y(i) & I \end{bmatrix} &> 0, \\ -\left[A(i)Q(i) + Q(i)A(i)^T + B(i)Y(i) + Y(i)^T B(i)^T + \sum_{j=1}^{N} \pi_{ji} Q(j) \right] &> 0, \quad i = 1, \ldots, N, \end{aligned}$$

If such matrices exist, a stabilizing state-feedback is given by $K(i) = Y(i)Q(i)^{-1}$, $i = 1, \ldots, N$.

In the above problem, the data matrices are $A(1), \ldots, A(N)$, $B(1), \ldots, B(N)$ and the transition matrix Π. The unknown matrices are $Q(i)$'s (which are symmetric $n \times n$ matrices) and $Y(i)$'s (which are $n_u \times n$ matrices). In this case, both the number of the data matrices and that of the unknown matrices are a-priori unknown.

The above problem is obviously a Σ problem. In this case, we can let `XLIST` be a list of two lists: one representing the Q's and the other, the Y's.

The evaluation function required for invoking `lmisolver` can be constructed as follows:

```
function [LME,LMI,OBJ]=jump_sf_eval(XLIST)
[Q,Y]=XLIST(:)
N=size(A); [n,nu]=size(B(1))
LME=list(); LMI1=list(); LMI2=list()
tr=0
for i=1:N
   tr=tr+trace(Q(i))
   LME(i)=Q(i)-Q(i)'
   LMI1(i)=[Q(i),Y(i)';Y(i),eye(nu,nu)]
   SUM=zeros(n,n)
   for j=1:N
     SUM=SUM+PI(j,i)*Q(j)
```

```
    end
    LMI2(i)= A(i)*Q(i)+Q(i)*A(i)'+B(i)*Y(i)+Y(i)'*B(i)'+SUM
 end
 LMI=list(LMI1,LMI2)
 LME(N+1)=tr-1
 OBJ=[]
```

Note that `LMI` is also a list of lists containing the values of the LMI matrices. This is just a matter of convenience.

Now, we can solve the problem in Scilab as follows (assuming lists `A` and `B`, and matrix `PI` have already been defined).

First we should initialize `Q` and `Y`.

```
--> N=size(A);  [n,nu]=size(B(1)); Q_init=list(); Y_init=list();
--> for i=1:N, Q_init(i)=zeros(n,n);Y_init(i)=zeros(nu,n);end
```

Then, we can use `lmisolver` as follows:

```
--> XLIST0=list(Q_init,Y_init)
--> XLISTF=lmisolver(XLIST0,jump_sf_eval)
--> [Q,Y]=XLISTF(:);
```

The above commands can be encapsulated in a solver function, say `jump_sf`, in which case we simply need to type:

```
--> [Q,Y]=jump_sf(A,B,PI)
```

to obtain the solution.

2.2.3 Descriptor Lyapunov inequalities

In the study of descriptor systems, it is sometimes necessary to find (or find out that it does not exist) an $n \times n$ matrix X satisfying

$$\begin{aligned} E^T X = X^T E &\geq 0 \\ A^T X + X^T A + I &\leq 0 \end{aligned}$$

where E and A are $n \times n$ matrices such that E, A is a regular pencil. In this problem, which clearly is a Σ problem, the LME functions play important role. The evaluation function can be written as follows

```
function [LME,LMI,OBJ]=dscr_lyap_eval(XLIST)
X=XLIST(:)
LME=E'*X-X'*E
LMI=list(-A'*X-X'*A-eye(),E'*X)
OBJ=[]
```

and the problem can be solved by (assuming E and A are already defined)

```
--> XLIST0=list(zeros(A))
--> XLISTF=lmisolver(XLIST0,dscr_lyap_eval)
--> X=XLISTF(:)
```

2.2.4 Mixed H_2/H_∞ Control

Consider the linear system

$$\begin{aligned} \dot{x} &= Ax + B_1 w + B_2 u \\ z_1 &= C_1 x + D_{11} w + D_{12} u \\ z_2 &= C_2 x + D_{22} u \end{aligned}$$

The mixed H_2/H_∞ control problem consists in finding a stabilizing feedback which yields $\|T_{z_1w}\|_\infty < \gamma$ and minimizes $\|T_{z_2w}\|_2$ where $\|T_{z_1w}\|_\infty$ and $\|T_{z_2w}\|_2$ denote respectively the closed-loop transfer functions from w to z_1 and z_2. In [3], it is shown that the solution to this problem can be expressed as $K = LX^{-1}$ where X and L are obtained from the problem of minimizing Trace(Y) subject to:

$$X - X^T = 0, \;\; Y - Y^T = 0,$$

and

$$-\begin{pmatrix} AX + B_2L + (AX + B_2L)^T + B_1B_1^T & XC_1^T + L^TD_{12}^T + B_1D_{11}^T \\ C_1X + D_{12}L + D_{11}B_1^T & -\gamma^2 I + D_{11}D_{11}^T \end{pmatrix} > 0$$

$$\begin{pmatrix} Y & C_2X + D_{22}L \\ (C_2X + D_{22}L)^T & X \end{pmatrix} > 0$$

To solve this problem with `lmisolver`, we define the evaluation function:

```
function [LME,LMI,OBJ]=h2hinf_eval(XLIST)
[X,Y,L]=XLIST(:)
LME=list(X-X',Y-Y');
LMI=list(-[A*X+B2*L+(A*X+B2*L)'+B1*B1',X*C1'+L'*D12'+B1*D11';...
              (X*C1'+L'*D12'+B1*D11')',-gamma^2*eye()+D11*D11'],...
                                  [Y,C2*X+D22*L;(C2*X+D22*L)',X])
OBJ=trace(Y);
```

and use it as follows:

```
--> X_init=zeros(A); Y_init=zeros(C2*C2'); L_init=zeros(B2')
--> XLIST0=list(X_init,Y_init,L_init);
--> XLISTF=lmisolver(XLIST0,h2hinf_eval);
--> [X,Y,L]=XLISTF(:)
```

2.2.5 Descriptor Riccati equations

In Kalman filtering for descriptor system

$$\begin{aligned} Ex(k+1) &= Ax(k) + u(k) \\ y(k+1) &= Cx(k+1) + r(k) \end{aligned}$$

where u and r are zero-mean, white Gaussian noise sequences with covariance Q and R respectively, one needs to obtain the positive solution to the descriptor Riccati equation (see [4])

$$P = -\begin{pmatrix} 0 & 0 & I \end{pmatrix} \begin{pmatrix} APA^T + Q & 0 & E \\ 0 & R & C \\ E^T & C^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ I \end{pmatrix}.$$

It can be shown that this problem can be formulated as a Σ problem as follows: maximize Trace(P) under constraints

$$P - P^T = 0$$

and

$$\begin{pmatrix} APA^T + Q & 0 & EP \\ 0 & R & CP \\ P^TE^T & P^TC^T & P \end{pmatrix} \geq 0.$$

The evaluation function is:

```
function [LME,LMI,OBJ]=ric_dscr_eval(XLIST)
LME=P-P'
LMI=[A*P*A'+Q,zeros(A*C'),E*P;zeros(C*A'),R,C*P;P*E',P*C',P]
OBJ=-trace(P)
```

which can be used as follows (asuming E, A, C, Q and R are defined and have compatible sizes–note that E and A need not be square).

```
--> P_init=zeros(A'*A)
--> P=lmisolver(XLIST0,ric_dscr_eval)
```

2.2.6 Linear programming with equality constraints

Consider the following classical optimization problem

$$\begin{array}{ll} \text{minimize} & e^T x \\ \text{subject to} & Ax + b \geq 0, \\ & Cx + d = 0. \end{array}$$

where A and C are matrices and e, b and d are vectors with appropriate dimensions. Here the sign $\geq$ is to be understood elementwise.

This problem can be formulated in LMITOOL as follows:

```
function [LME,LMI,OBJ]=linprog_eval(XLIST)
 [x]=XLIST(:)
 [m,n]=size(A)
 LME=C*x+d
 LMI=list()
 tmp=A*x+b
 for i=1:m
    LMI(i)=tmp(i)
 end
 OBJ=e'*x
```

and solved in Scilab by (assuming A, C, e, b and d and an initial guess x0 exist in the environment):

```
--> x=lmisolver(x0,linprog_eval)
```

2.2.7 Sylvester Equation

The problem of finding matrix X satisfying

$$AX + XB = C$$

or

$$AXB = C$$

where A and B are square matrices (of possibly different sizes) is a well-known problem. We refer to the first equation as the continuous Sylvester equation and the second, the discrete Sylvester equation.

These two problems can easily be formulated as Σ problems as follows:

```
function [LME,LMI,OBJ]=sylvester_eval(XLIST)
[X]=XLIST(:)
if flag=='c' then
  LME=A*X+X*B-C
else
  LME=A*X*B-C
end
LMI=[]
OBJ=[]
```

with a solver function such as:

```
function [X]=sylvester(A,B,C,flag)
[na,ma]=size(A);[nb,mb]=size(B);[nc,mc]=size(C);
if ma<>na|mb<>nb|nc<>na|mc<>nb then error("invalid dimensions");end
XLISTF=lmisolver(zeros(nc,mc),sylvester_eval)
X=XLISTF(:)
```

Then, to solve the problem, all we need to do is to (assuming A, B and C are defined)

```
--> X=sylvester(A,B,C,'c')
```

for the continuous problem and

```
--> X=sylvester(A,B,C,'d')
```

for the discrete problem.

3 Function LMITOOL

The purpose of LMITOOL is to automate most of the steps required before invoking lmisolver. In particular, it generates a *.sci file including the solver function and the evaluation function or at least their skeleton. The solver function is used to define the initial guess and to modify optimization parameters (if needed).

lmitool can be invoked with zero, one or three arguments.

3.1 Non-interactive mode

lmitool can be invoked with three input arguments as follows:

3.1.1 Syntax

```
txt=lmitool(probname,varlist,datalist)
```

where

- probname: a string containing the name of the problem,
- xlist: a string containing the names of the unknown matrices (separated by commas if there are more than one).
- dlist: a string containing the names of data matrices (separated by commas if there are more than one).
- txt: a string providing information on what the user should do next.

In this mode, lmitool generates a file in the current directory. The name of this file is obtained by adding ".sci" to the end of probname. This file is the skeleton of a solver function and the corresponding evaluation function.

3.1.2 Example

Suppose we want to use lmitool to solve the problem presented in Section 2.2.1. Invoking

```
-->txt=lmitool('sf_sat','Q,Y','A,B,umax')
```

yields the output

```
--> txt  =

!    To solve your problem, you need to                          !
!                                                                !
!1- edit file /usr/home/DrScilab/sf_sat.sci                      !
!                                                                !
!2- load (and compile) your functions:                           !
!                                                                !
!   getf('/usr/home/DrScilab/sf_sat.sci','c')                    !
!                                                                !
!3- Define A,B,umax and call sf_sat function:                    !
!                                                                !
!  [Q,Y]=sf_sat(A,B,umax)                                        !
!                                                                !
!To check the result, use [LME,LMI,OBJ]=sf_sat_eval(list(Q,Y))  !
```

and results in the creation of the file '/usr/home/curdir/sf_sat.sci' with the following content:

```
 function [Q,Y]=sf_sat(A,B,umax)
// Generated by lmitool on Tue Feb 07 10:30:35 MET 1995

  Mbound = 1e3;
  abstol = 1e-10;
  nu = 10;
  maxiters = 100;
  reltol = 1e-10;
  options=[Mbound,abstol,nu,maxiters,reltol];

//////////DEFINE INITIAL GUESS BELOW
Q_init=...
Y_init=...
//////////

XLIST0=list(Q_init,Y_init)
XLIST=lmisolver(XLIST0,sf_sat_eval,options)
[Q,Y]=XLIST(:)



////////////////EVALUATION FUNCTION//////////////////////////

function [LME,LMI,OBJ]=sf_sat_eval(XLIST)
[Q,Y]=XLIST(:)

////////////////DEFINE LME, LMI and OBJ BELOW
LME=...
LMI=...
OBJ=...
```

It is easy to see how a small amount of editing can do the rest!

3.2 Interactive mode

`lmitool` can be invoked with zero or one input argument as follows:

3.2.1 Syntax

```
txt=lmitool()
txt=lmitool(file)
```

where

- `file`: is a string giving the name of an existing ".sci" file generated by `lmitool`.

In this mode, `lmitool` is fully interactive. Using a succession of dialogue boxes, user can completely define his problem. This mode is very easy to use and its operation completely self explanatory. Invoking `lmitool` with one argument allows the user to start off with an existing file. This mode is useful for modifying existing files or when the new problem is not too much different from a problem already treated by `lmitool`.

3.2.2 Example

Consider the following estimation problem

$$y = Hx + Vw$$

where x is unknown to be estimated, y is known, w is a unit-variance zero-mean Gaussian vector, and

$$H \in \mathbf{Co}\{H(1), ..., H(N)\}, \quad V \in \mathbf{Co}\{V(1), ..., V(N)\}$$

where **Co** denotes the convex hull and $H(i)$ and $V(i)$, $i = 1, ..., N$, are given matrices.

The objective is to find L such that the estimate

$$\hat{x} = Ly$$

is unbiased and the worst case estimation error variance $\mathrm{E}(\|x - \hat{x}\|^2)$ is minimized.

It can be shown that this problem can be formulated as a Σ problem as follows: minimize γ subject to

$$\begin{aligned} I - LH(i) &= 0, \quad i = 1, ..., N, \\ X(i) - X(i)^T &= 0, \quad i = 1, ..., N, \end{aligned}$$

and

$$\begin{aligned} \begin{pmatrix} I & (L(i)V(i))^T \\ L(i)V(i) & X(i) \end{pmatrix} \geq 0, \quad i = 1, ..., N, \\ \gamma - \mathrm{Trace}(X(i)) \geq 0, \quad i = 1, ..., N. \end{aligned}$$

To use `lmitool` for this problem, we invoke it as follows:

```
--> lmitool()
```

This results is an interactive session which is partly illustrated in following figures.

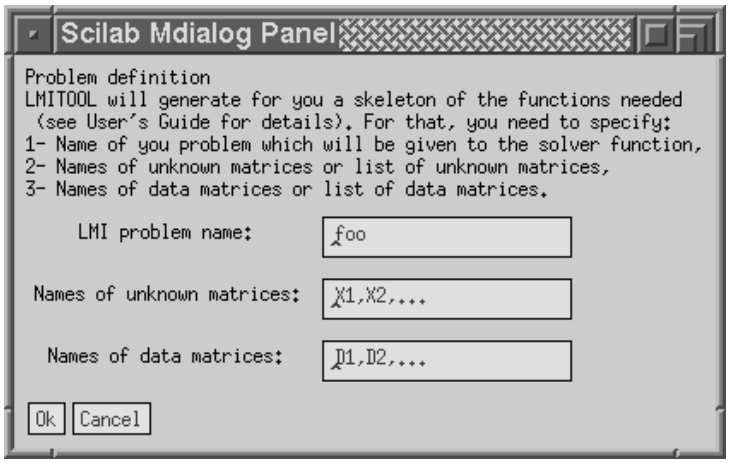

Figure 1: This window must be edited to define problem name and the name of variables used.

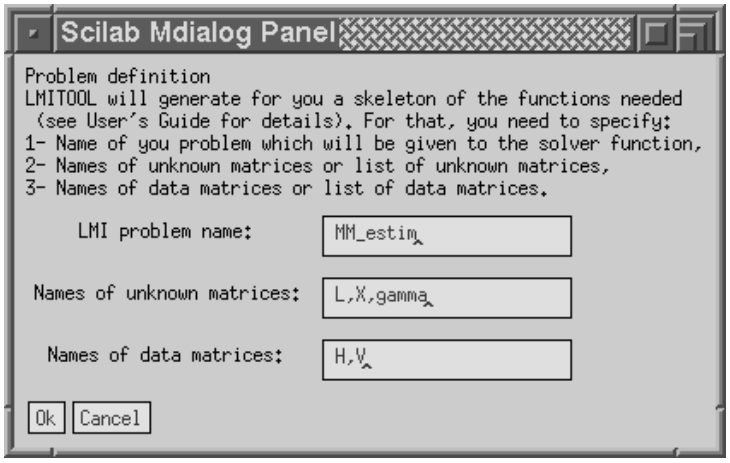

Figure 2: For the example at hand the result of the editing should look something like this.

A How `lmisolver` works

The function `lmisolver` works essentially in four steps:

1. *Initial set-up.* The sizes and structure of the initial guess are used to set up the problem, and in particular the size of the unknown vector.

2. *Elimination of equality constraints.* Making repeated calls to the evaluation function, `lmisolver` generates a canonical representation of the form

$$\begin{array}{ll} \text{minimize} & \tilde{c}^T z \\ \text{subject to} & \tilde{F}_0 + z_1\tilde{F}_1 + \cdots + z_{\tilde{m}}\tilde{F}_{\tilde{m}} \geq 0, \quad Az + b = 0, \end{array}$$

   where z contains the coefficients of all matrix variables. This step uses extensively sparse matrices to speed up the computation and reduce memory requirement.

3. *Elimination of variables.* Then, `lmisolver` eliminates the redundant variables. The equality constraints are eliminated by computing the null space N of A and a solution z_0 (if any) of $Ax + b = 0$. At this stage, all solutions of the equality constraints are parametrized by

$$z = Nx + z_0,$$

   where x is a vector containing the independent variables. The computation of N, z_0 is done using sparse LU functions of Scilab.

   Once the equality constraints are eliminated, the problem is reformulated as

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F_0 + x_1F_1 + \cdots + x_mF_m \geq 0, \end{array}$$

   where c is a vector, and $F_0, \ldots, F_m$ are symmetric matrices, and x contains the *independent* elements in the matrix variables $X_1, \ldots, X_M$. (If the F_i's are dependent, a column compression is performed.)

4. *Optimization.* Finally, `lmisolver` makes a call to the function `semidef` (an interface to **SP** [1]). This phase is itself divided into a feasibility phase and a minimization phase (only if the linear objective function is not empty). The feasibility phase is avoided if the initial guess is found to be feasible.

   The function `semidef` is called with the optimization parameters `abstol`, `nu`, `maxiters`, `reltol`. The parameter `M` is set above the value

```
Mbnd*max(sum(abs([F0 ... Fm])))
```

   For details about the optimization phase, and the meaning of the above optimization parameters see manual page for `semidef`.

B Other versions

`LMITOOL` is also available on Matlab. The Matlab version can be obtained by anonymous ftp from `ftp.ensta.fr` under `/pub/elghaoui/lmitool`.

References

[1] Vandenberghe, L., and S. Boyd, "Semidefinite Programming," Internal Report, Stanford University, 1994 (submitted to SIAM Review).

[2] Boyd, S., L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in Systems and Control Theory*, SIAM books, 1994.

[3] Khargonekar, P. P., and M. A. Rotea, "Mixed H_2/H_∞ Control: a Convex Optimization Approach," *IEEE Trans Aut. Contr.*, 39 (1991), pp. 824-837.

[4] Nikoukhah, R., Willsky, A. S., and B. C. Levy, "Kalman Filtering and Riccati Equations for Descriptor Systems," *IEEE Trans Aut. Contr.*, 37 (1992), pp. 1325-1342.

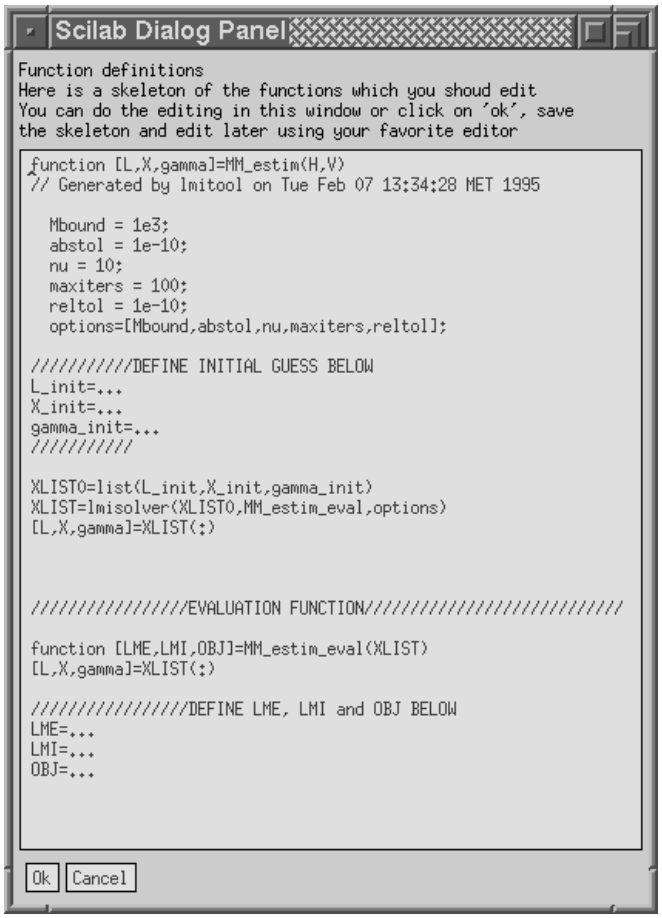

Figure 3: This is the skeleton of the solver function and the evaluation function generated by `LMITOOL` using the names defined previously.

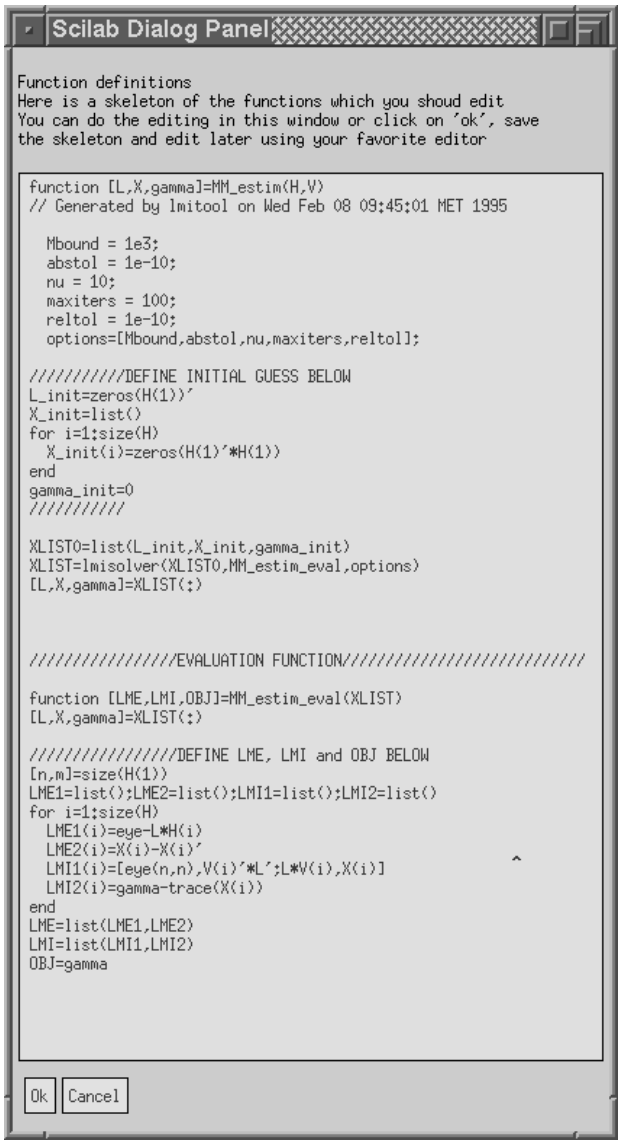

Figure 4: After editing, we obtain.

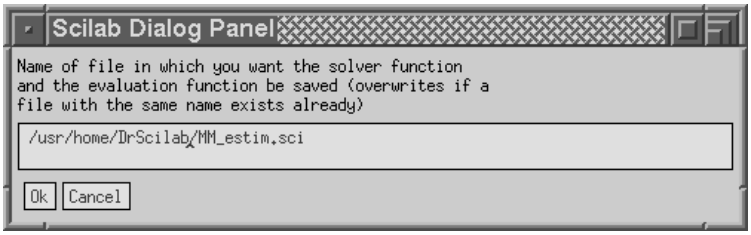

Figure 5: A file is proposed in which the solver and evaluation functions are to be saved. You can modify it if you want.