## Command overview

<table>
<thead>
<tr>
<th>Category</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| **Workspace**        | `yalmip('clear')`  
                      | `yalmip('info')`   
                      | `yalmip('nvars')`   |
| **Symbolic variable**| `X = sdpvar(nx, ny, type)`  
                      | `y = double(X)`    
                      | `see(X)`          |
| **LMI definition**   | `F = lmi('expression')`  
                      | `F = addlmi(F, 'expression')` |
| **Solution of SDP**  | `sol = solvesdp(F, G, h, options, x0)`  
                      | `options = sdpsettings(parameterlist)` |
| **Other & test**     | `yalmipdemo`  
                      | `dbtest`           
                      | `savesdpafile(F, h, 'file')` |
# Contents

1 Introduction ........................................... 7
   1.1 Requirements ........................................ 7
   1.2 Installation ......................................... 7

2 Semidefinite programming and linear matrix inequalities .................. 9

3 Using YALMIP .............................................. 11
   3.1 Examples ............................................. 11

4 Supported solvers ....................................... 15

5 Command Reference ...................................... 19
   5.1 Overloaded `sdpvar` functions ...................... 20
   5.2 `addlmi` .............................................. 21
   5.3 `lmi` .................................................. 23
   5.4 `savedpfile` ......................................... 24
   5.5 `see` ................................................ 25
   5.6 `sdpvar` .............................................. 26
   5.7 `sdp2var` ............................................. 27
   5.8 `sdpvarfun` .......................................... 28
   5.9 `sdpsettings` ....................................... 29
   5.10 `solvesdp` .......................................... 31
## Contents

5.11 updateimi ........................................ 33
5.12 yalmip ............................................... 34

Bibliography ............................................. 35

Index ..................................................... 35
1 Introduction

This manual gives an overview of the LMI parser YALMIP (Yet Another LMI Parser). The YALMIP package is developed with the aim to have a parser that is completely integrated and developed in the MATLAB environment. The advantages with this approach is the platform independence and the ease to extend the package with new features. The drawback is of course the performance loss. The package is therefore intended for small problems with 10-100 variables and constraints.

1.1 Requirements

The package uses the object oriented capabilities of MATLAB and therefore MATLAB 5 or higher is needed. The package is currently used on UNIX MATLAB 6.0 and Windows MATLAB 5.3 without problems.

Since YALMIP works as an interface to the free solvers SP, SOCP, MAXDET, CSDP, SeDuMi, SDPT3 and SDPA, at least one of these packages has to be installed on the system.

1.2 Installation

To install YALMIP, unzip the file yalmip.zip in the directory where you want to have your YALMIP directory. After the installation, you should have obtained the following directory structure

    ..\yalmip
    ..\yalmip\@lmi
    ..\yalmip\@sdpvar
    ..\yalmip\extras
    ..\yalmip\doc

Put the ..\yalmip and ..\yalmip\extras directories in your MATLAB path.

The installation of the solvers should be performed as recommended in the
respective manuals. Some additional important information concerning the installation of the solvers is given in Chapter 4.
2 Semidefinite programming and linear matrix inequalities

In the field of optimization, the crucial property is not linearity but convexity. Recently, there has been much development for problems where the constraints can be written as the requirement of a matrix to be positive semidefinite. This is a convex constraint and motivates the definition of a linear matrix inequality, LMI.

**Definition 1 (LMI)** An LMI (linear matrix inequality) is an inequality, in the free scalar variables $x_i$, that for some fixed symmetric matrices $F_i$ can be written as

$$F(x) = F_0 + x_1 F_1 + x_2 F_2 + \ldots + x_n F_n \succeq 0$$

or in a more compact notation

$$F(x) \succeq 0$$

In the definition above, we introduced the notion of a semidefinite matrix $F \succeq 0$ which means $F = F^T$ and $z^T F z \geq 0 \forall z$.

As an example of an LMI, the nonlinear constraint $x_1 x_2 \geq 1, x_1 \geq 0, x_2 \geq 0$ can be written

$$\begin{bmatrix} x_1 & 1 \\ 1 & x_2 \end{bmatrix} \succeq 0$$

The matrix can be decomposed into a set of basis matrices and we obtain

$$F(x) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \succeq 0$$

An excellent introduction to LMIs, with special attention to problems in control theory, can be found in [8].

By using LMIs, many convex optimization problems, such as linear programming and quadratic programming, can be unified by the introduction of semidefinite programming [12].
**Semidefinite programming and linear matrix inequalities**

**Definition 2 (SDP)** An SDP (semidefinite program), is an optimization problem that can be written as

$$\min_x \ c^T x$$

subject to \( F(x) \geq 0 \)

SDPs can today be solved with high efficiency, i.e., with polynomial complexity, due to the recent development of solvers using interior-point methods [11]. One particular solver is [13] which is freely available.

A special class of SDP is MAXDET problems.

**Definition 3 (MAXDET)** A MAXDET problem (determinant maximization) is an optimization problem that can be written as

$$\min_x \ c^T x - \log \det(G(x))$$

subject to \( F(x) \geq 0 \)

\( G(x) \geq 0 \)

This is an optimization problem that frequently occurs in problems where the analysis is based on ellipsoids. A MAXDET problem can be converted to a standard SDP [11], and thus solved with general SDP solvers. However, there exist special purpose solvers for MAXDET problems [14].

Another special SDP is SOCP [10].

**Definition 4 (SOCP)** A SOCP (second order cone program) is an optimization problem with the structure

$$\min_x \ c^T x$$

subject to \( ||A_i x + b_i|| \leq c_i^T x + d_i \)

This problem can easily be rewritten into an SDP. Due to the special structure however, a more efficient method to solve the problem is to use special purpose solvers such as [9].
3 Using YALMIP

YALMIP is intended for problems that can be written as

\[ \min_x \ h(x) - \log \det(G(x)) \]

subject to

\[ F(x) \succeq 0 \]
\[ G(x) \succeq 0 \]
\[ Ax = b \]

The function \( h(x) \) is a scalar linear function. In principle, this is the structure that we saw in the definition of a MAXDET problem. The difference is the more loosely described goal function, and the linear equality constraints.

3.1 Examples

We introduce the basic concepts in YALMIP with two worked examples.

3.1.1 Lyapunov stability

We have a linear system

\[ \dot{x}(t) = Ax(t) \]

and our goal is to find positive definite matrix \( P \) satisfying

\[ A^T P + P^T A \prec 0 \]

Before defining a new problem, it is advised that you clear the internal structure. This is done with the command

\[ >> \ yalmip(’clear’); \]

It is not necessary to perform this command, but it cleans up some memory structures and old solution structures.
Defining a symbolic variable is done with the command `sdpvar`. In our case, we need a symmetric 2x2 matrix

```matlab
>> P = sdpvar(2, 2, 'symmetric');
```

We are now ready to define the LMIs. To begin with, we create an LMI structure containing the constraint that \( P \) is positive definite

```matlab
>> F = lmi('P>0');
```

The Lyapunov equation is added to the LMI structure

```matlab
>> F = addlmi(F, 'A''*P+P''*A<0');
```

Notice the double transpose operator that has to be used inside a string (standard MATLAB notation).

At this point, we are ready to solve our problem. We only search for a feasible solution, so one argument is sufficient

```matlab
>> solution = solvesdp(F);
```

The variable `solution` will contain information related to the solution. The solution will also be automatically stored and allow us to use the overloaded `double` function to obtain the numerical solution

```matlab
>> P_feasible = double(P);
```

The code for the example above is contained in the file `manualex1.m`.

### 3.1.2 Model predictive control

As a second example, we formulate a model predictive control problem with YALMIP.

Once again we have a linear system

\[
\begin{align*}
x(k+1) &= Ax(k) + Bu(k) \\
y(k) &= Cx(k)
\end{align*}
\]
The goal is to find the control sequence that minimizes a quadratic finite horizon performance criteria

$$\min_{u(:,k)} \sum_{j=0}^{N-1} ||y(k+j+1|k)||^2 + ||u(k+j|k)||^2$$

The minimization should be performed under the control constraint

$$|u(k+j|k)| \leq 1, \quad j = 0, 1, \ldots, N - 1$$

and the terminal constraint

$$|y(k+N|k)| \leq 1$$

To solve this, we first write the problem in a vector notation. Define the predicted outputs and the control sequence

$$Y = \begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ y(k+N|k) \end{bmatrix}, \quad U = \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}$$

By using the linear system model, we can write

$$Y = Hx(k) + SU$$

The optimization problem can be written as

$$\min_{t, U} t$$

subject to

$$Y^T Y + U^T U \leq t$$
$$|U| \leq 1$$
$$|y(N)| \leq 1$$

The constraint can be rewritten using a Schur complement and we obtain

$$\min_{t, U} t$$

subject to

$$\begin{bmatrix} t & Y^T & U^T \\ Y & I & 0 \\ U & 0 & I \end{bmatrix} \succeq 0$$
$$|U| \leq 1$$
$$|y(N)| \leq 1$$

To solve this with YALMIP, we first clear the internal structure

`>> yalmip('clear');`

We need two variables
Using YALMIP

>> t = sdpvar(1,1);
>> U = sdpvar(N,1);

The output prediction is defined (assuming $x$, $H$ and $S$ are available)

>> Y = H*x+S*U;

We create the control constraint by defining a new lmi object

>> F = lmi('U<1')
>> F = addlmi(F,'U>-1')

The terminal constraint is added by exploiting the overloaded indexing functionality (for simplicity we assume $y$ to be scalar)

>> F = addlmi(F,'Y(N)>-1');
>> F = addlmi(F,'Y(N)<1');

Finally we add the LMI (now using a faster notation without strings)

>> F = addlmi(F,[t Y' U';Y eye(N) zeros(N);U zeros(N) eye(N)])

The solution is obtained with

>> F = solvesdp(F,[],t);

Check $U$ and $Y(N)$ to see that the constraint indeed is satisfied

>> double(U)
>> double(Y(N))

The source code for the model predictive control example can be found in the file manualex2.m.
4 Supported solvers

YALMIP supports a number of different semidefinite programming solvers. However, the level of support differs for the solvers. In this chapter, we give a short description on the interface to the solvers.

SP

Support: YALMIP was originally developed for SP [13, 7]. Since SP has been the main solver since the beginning, the interface to SP is fully supported.

Status: The SP interface has been extensively tested on both Solaris and Windows.

Installation: Make sure that the SP library is in your MATLAB path.

MAXDET

Support: The MAXDET [14, 2] solver has also been supported by YALMIP from the beginning and is fully supported.

Status: The MAXDET interface has been tested extensively on both Solaris and PC.

Installation: Make sure that the MAXDET directory is in your MATLAB path. For Windows versions, it is important that the additional dlls MKL, BLAS and LAPACK (explained in the installation information for MAXDET) are in the Windows path. A simple way to guarantee that everything works is to place these dlls in the same directory as the solvers.

SOCP

Support: The interface to the SOCP [9, 6] solver has also been a part of YALMIP since the beginning. YALMIP supports all features of SOCP.
Supported solvers

**Status**: The SOCP interface has been tested on Solaris.

**Installation**: Make sure that the SOCP directory is in your MATLAB path.

---

**SeDuMi**

**Support**: The SeDuMi [5] solver is partially supported from version 1.1 of YALMIP. Currently, YALMIP does not exploit the possibility to solve second order cone programs with the specialized code in SeDuMi.

**Status**: The SeDuMi interface has been tested for a number of problems on both Solaris and PC.

**Installation**: Make sure that the SeDuMi directory is in your MATLAB path.

---

**CSDP**

**Support**: The CSDP solver [1] is partially supported from version 1.1 of YALMIP. No options structure is available for the CSDP solver. Note that the communication with CSDP is done by writing the problem data to a file. Therefore, for large problems, much time will be spent in reading and writing the problem definition.

**Status**: The CSDP interface has been tested for a small number of problems on Solaris.

**Installation**: Make sure that the CSDP directory is in your MATLAB path. Furthermore, a file named `csdp.m` should be placed in the same directory the CSDP executable.

---

**SDPT3**

**Support**: The SDPT3 [4] solver is fully supported from version 1.1 of YALMIP. The communication with SDPT3 is done by writing the problem data to a file, hence performance will suffer for large problems.

**Status**: The SDPT3 interface has been tested for a small number of problems on Windows.

**Installation**: Make sure that the SDPT3 directory is in your MATLAB path.
SDPA

Support: The SDPA [3] solver is partially supported from version 1.1 of YALMIP. The options structure is currently not used, hence all problems will be solved with the default settings (found in the file parameters.m in the SDPA directory). The communication also for SDPA is done by writing the problem data to a file.

Status: The SDPA interface has been tested for a small number of problems on Solaris.

Installation: Make sure that the SDPA directory is in your MATLAB path. Furthermore, a file named sdpa.m should be placed in the same directory as the SDPA executable.
Supported solvers
5 Command Reference
5.1 Overloaded \texttt{sdpvar} functions

**Description**  
Most operators and functions necessary to describe symbolic variables (\texttt{sdpvar} objects) are overloaded to the \texttt{sdpvar} class.

**Examples**  
Addition, subtraction and multiplication are the most basic overloaded functions. Assuming \texttt{X} and \texttt{Y} to be \texttt{sdpvar} objects and \texttt{A} to be a double, the following code is valid (assuming sizes are compatible)

```matlab
>> X = A*Y+X
```

Standard linear operators such as \texttt{diag}, \texttt{trace}, \texttt{sum} and many more are of course also overloaded

```matlab
>> X = A*diag(Y)+sum(trace(Y))
```

Concatenation and indexing can also be used

```matlab
>> X = Y(1:2,1:2)+[X(end,end) Y(1,1);X(3,1:2)]
```

Finally, note that multiplication of \texttt{sdpvar} is possible as long as all nonlinear terms disappear. As an example (\texttt{x} and \texttt{y} scalar \texttt{sdpvar} objects)

```matlab
>> [x 1]*[1;y]
```

is valid, but not

```matlab
>> [x 1]*[y;1]
```

The complete list of overloaded functions are \texttt{plus}, \texttt{minus}, \texttt{times}, \texttt{mtimes}, \texttt{mdivide}, \texttt{transpose}, \texttt{ctranspose}, \texttt{trace}, \texttt{diag}, \texttt{blkdiag}, \texttt{repmat}, \texttt{triu}, \texttt{tril}, \texttt{rot90}, \texttt{flipud}, \texttt{fliplr}, \texttt{sum}, \texttt{diff}, \texttt{kron}, \texttt{size}, \texttt{length}, \texttt{horzcat}, \texttt{vertcat}, \texttt{subasgn}, \texttt{subsref}, \texttt{double}, \texttt{display}

**See Also**  
\texttt{sdpvar}
addlmi

5.2 addlmi

Purpose  addlmi adds a constraint to an existing LMI object.

Synopsis  \[ F = \text{addlmi}(G,X); \]

Description  addlmi is used to add constraints on a matrix variable. Both equalities and inequalities are supported, as well as matrix constraints and element-wise constraints. Furthermore, a special constraint to identify second order constraints is supported.

\[ G \text{ LMI object.} \]

\[ X \text{ New constraint, given either as an } \text{sdpvar} \text{ object required to be positive semidefinite (element-wise non-negative if unsymmetric), or as a string symbolically describing the constraint.} \]

Examples  Adding the constraint

\[
\begin{bmatrix}
1 & x^T \\
x & I
\end{bmatrix} \preceq 0 \quad (x \text{ sdpmvar object in } \mathbb{R}^n)
\]

to an existing LMI structure \( F \) is done with

\[
>> F = \text{addlmi}(F,'[1 x';x eye(n)]<0')
\]

Another variant is to skip the string notation and use an \text{sdpvar} object instead.

\[
>> F = \text{addlmi}(F,-[1 x';x eye(n)])
\]

Notice the sign that has to be used since default is positive semidefiniteness.

Constraints are interpreted as element-wise if an . is used in front of the quantifier. As an example, requiring all elements to be positive is done with

\[
>> F = \text{addlmi}(F,'x.>0')
\]

Second order cone constraints can be added with the special purpose \( \| \cdot \| \) operator.
Command Reference

\[ \text{\texttt{\textbackslash{}gg F = addlmi(F,\textquotesingle{|A\star x+b|<c\star x+d\textquotesingle})}} \]

For the expression to make sense, the expression \( A\star x+b \) must be a vector, and \( c\star x+d \) must evaluate to a scalar.

Equality constraints can also be added. A constraint that the sum of the vector components is equal to 1 is obtained with

\[ \text{\texttt{\textbackslash{}gg F = addlmi(F,\textquotesingle{\text{sum(x)}=1\textquotesingle})}} \]

See Also

lmi, sdpvar
5.3 lmi

Purpose

lmi creates a new LMI object.

Synopsis

F = lmi(X);

Description

lmi creates an LMI object containing the constraint described by X. An empty lmi object is generated if lmi is called without any in-data.

X Constraint, given either as an sdpvar object required to be positive semidefinite (element-wise non-negative if unsymmetric), or as a string symbolically describing the constraint.

Examples

See examples in addlmi.

See Also

addlmi, sdpvar

YALMIP 23
5.4 savesdpfile

Purpose
sdp2mat writes the problem to a file in the SDPA format.

Synopsis
savesdpfile(F,h,file);

Description
Many solvers are capable of reading problem definitions in the SDPA format, so this function allows the user to use any preferred solver.

F lmi object. The LMIs in the problem.
h sdpvar object. Describes the objective function.
file A string with a filename. If omitted, a dialog box will appear.

See Also
lmi, sdpvar
Purpose

sdpvar shows the internal base matrices used to define an sdpvar object.

Synopsis

see(X);

Description

Every symbolic variable is represented with a set of basis matrices (see definition of LMI). These matrices can be displayed using the see command.

X sdpvar object.

See Also

sdpvar
5.6  sdpvar

Purpose

sdpvar creates a new sdpvar objects

Synopsis

X = sdpvar(n,m,type);

Description

sdpvar is used to create new symbolic variables, so called sdpvar objects.

n  Number of rows.

m  Number of columns.

type Kind of matrix (’symmetric’ or ’full’)

Examples

To generate a symmetric 2x2 variable the full notation is

>> X = sdpvar(2,2,’symmetric’)

Since square matrices are assumed symmetric by default, the matrix can also be defined with

X = sdpvar(2,2)

To create a square unsymmetric (fully parametrized) matrix we have to give three arguments

>> X = sdpvar(2,2,’full’)

See further examples on how to work with sdpvar objects in overloaded functions.

See Also

overloaded functions
**sdp2var**

### 5.7 sdp2var

#### Purpose

`sdp2mat` converts the internal solution formats to `double`.

#### Synopsis

```matlab
Y = sdp2mat(X,sol);
```

#### Description

`sdp2mat` is used to obtain the numerical value from a specific solution. Most often, the command `double` is used.

- **X** `sdpvar` object.
- **sol** Solution structure.

#### See Also

overloaded functions
5.8  sdpvarfun

Purpose  

sdp2mat applies linear unary function on sdpvar object.

Synopsis  

Y = sdpvarfun(X,fun);

Description  

sdp2mat is used to apply any linear unary (one input, one output) matrix operator on a sdpvar object.

X sdpvar object.
fun string with function name.

Examples  

To obtain the trace of a sdpvar object (of course, trace is already available as an overloaded operator, this code is only to examplify the command)

>> Y = sdpvarfun(X,'trace')

See Also  

overloaded functions
5.9 sdpsettings

Purpose

sdpsettings creates an options structure to be used in solvesdp

Synopsis

ops = sdpsettings(parameter,value,parameter,value,...)

Description

sdpsettings creates a structure containing the parameters needed in the solvers. By calling the functions without any argument, the default values are obtained.

variable String with parameter name.
value Numerical value of parameter.

Examples

To create a structure with default settings

>> ops = sdpsettings
>> ops =
>>
>> Solver: ''
>> Silent: 0
>> SaveSolverOutput: 0
>> sp: [1x1 struct]
>> maxdet: [1x1 struct]
>> socp: [1x1 struct]
>> sdpt3: [1x1 struct]
>> sdpa: [1x1 struct]
>> sedumi: [1x1 struct]

The three first parameters are general settings for YALMIP. Solver allows the user to explicitly force a specific solver to be used

>> ops = sdpsettings('Solver','SP');

By default, the solvers operate in silent mode (no printing). This is chosen with Silent. By setting SaveSolverOutput to 1, all output from the solvers will be saved in the solution structure (see solvesdp).

The options structure also contains a number of structures with parameters used in each supported solver. As an example, the following parameters can be tuned for the SP solver
>> ops.sp
>> ans =

>> AbsTol: 1.0000e-05
>> RelTol: 1.0000e-05
>> nu: 30
>> tv: -Inf
>> Mfactor: [50000 1.1000]
>> NTiters: 50

See Also

solvesdp
solvesdp

5.10 solvesdp

Purpose solvesdp solves the semidefinite program

Synopsis solution = solvesdp(F,G,h,ops,x0);

Description solvesdp is the interface to the solvers SP, MAXDET and SOCP.

F LMI object.
G LMI object.
h Linear objective function. See the example for description.
ops Structure holding parameters used in the solvers.
x0 A feasible initial solution. See the example for description.

Examples Let us create an extremely simple problem

\[ \min x(1) + x(2), \quad x(1) \geq 0, \quad x(2) \geq 0 \]

We clear the internal structures and set up the variables and the constraints

\[ \text{>> yalmip('clear');} \]
\[ \text{>> x = sdpvar(2,1);} \]
\[ \text{>> F = lmi('x>0');} \]

Let us start by just finding a feasible solution.

\[ \text{>> sol = solvesdp(F);} \]

If we want to find an optimal solution we must use at least three arguments. Since there is no \( G(x) \) term, we let \( G \) be an empty matrix

\[ \text{>> sol = solvesdp(F,[],x(1)+x(2));} \]

Notice how easy we defined the objective. Another way is to use a notation that is more similar to that used in the solvers

\[ \text{>> sol = solvesdp(F,[],[1 1]);} \]
This way to define the objective is not recommended since it requires knowledge of how the `sdpvar` object `x` relates to the internal free variables in YALMIP. However, for the advanced user it might be of interest since it removes the need for parsing, hence speeding things up a bit.

Finally, we solve the problem again, but this time we give an initial feasible solution. This can be done in two ways. The first method appends a list of variables and numerical values to the input argument

```matlab
>> sol = solvesdp(F,[],sum(x),[],x(1),1,x(2),1);
```

The important thing is that initial values are given to all free variables. The following command does the same thing

```matlab
>> sol = solvesdp(F,[],sum(x),[],x,[1;1]);
```

Once again, with knowledge of the internal structure, the initial feasible solution can be given as one vector

```matlab
>> sol = solvesdp(F,[],sum(x),[],[1;1]);
```

In all the three examples above, the options structure was omitted and an empty matrix was passed. When this is done, the default parameter values are used.

Now, let us solve a MAXDET problem

$$\min \ t - \log \det(t), \quad t \geq 0$$

This is solved with

```matlab
>> t = sdpvar(1,1);
>> F = lmi('t>0');
>> G = lmi('t>0');
>> sol = solvesdp(F,G,t);
```

The optimal solution is saved directly internally, and the numerical values can be found with the overloaded `double` function

```matlab
>> topt = double(t);
```

**See Also**

overloaded functions, `lmi`, `addlmi`, `sdpvar`, `sdpsettings`
**Purpose**

`updatelmi` updates a symbolic LMI with current numerical values.

**Synopsis**

```matlab
F = updatelmi(F,n);
```

**Description**

`updatelmi` is used to change symbolic `lmi` objects. When a symbolic LMI is defined, the LMI is immediately converted to an internal numerical representation. By using `updatelmi`, the symbolic expression is re-evaluated using the current numerical values in the workspace.

- `F` LMI object.
- `n` LMI which should be updated.

**Examples**

Suppose we have a scalar variable `t` and the constraint

\[
\begin{bmatrix}
t & x \\
\hline
x & 1
\end{bmatrix} \succeq 0
\]

and we want to minimize `t` for the two cases `x = 1` and `x = 2`. This is done with

```matlab
>> t = sdpvar(1,1);
>> x = 1;
>> F = lmi(F,’[t x;x 1] > 0’); # 1 :[t x;x 1] > 0 Matrix inequality
>> solvesdp(F,[],t);t1 = double(t);
>> x = 2;
>> F = updatelmi(F,1)
>> solvesdp(F,[],t);t2 = double(t);
```

**See Also**

`lmi`, `addlmi`
5.12 yalmip

Purpose

`yalmip` performs a number of administrative tasks.

Synopsis

`yalmip('clear')`
`yalmip('info')`
`n = yalmip('nvars')`

Description

`sdpsettings` serves as an interface to some internal structures used by the YALMIP package.

Examples

Before a new problem is defined, it is advised that the internal structures are cleared

```matlab
>> yalmip('clear')
```

To find out how many free variables there currently are in all `sdpvar` objects

```matlab
>> yalmip('nvars')
```

To get some more information about the current status

```matlab
>> yalmip('info')
```

See Also

`solvesdp`, `sdpvar`
Bibliography


