Resolvendo LMIs no MATLAB com os pacotes Yalmip e Sedumi

Autor: **Flávio A. Faria**
E-mail: flaviof15@yahoo.com.br

Sumário

Pré-requisitos 3
Definição dos pacotes 3
Objetivo do curso 4
Instalação 4
Introdução ao yalmip 9
Projeto de controladores usando realimentação de estados 10
Projeto de um controlador ótimo $\mathbf{H}_2$ usando LMI 12
Projeto de controladores fuzzy usando realimentação de estados 13
Condições menos conservadoras para o projeto de controladores fuzzy usando realimentação de estados 16
Análise de estabilidade considerando uma função de Lyapunov dependente de parâmetros 19
Notas finais 20
Referências 20

Pré-requisitos

- Noções básicas de informática.
    - Estrutura de arquivos e pastas.
    - Compactação de arquivos.
- Noções básicas do MATLAB

Definição dos pacotes

Yalmip: Esse pacote (toolbox) permite que problemas envolvendo programação matemática sejam representados de maneira mais natural no MATLAB. Ele pode ser usado em diversas situações. Por exemplo: em problemas de programação linear, programação inteira e mista, problemas de programação semi-definida e em desigualdades matriciais bilineares. Uma das grandes vantagens desse pacote é que ele apresenta suporte para vários solvers (resolvedores). Por exemplo, esse pacote pode resolver LMIs com o LMILab (solver do LMI control toolbox) ou com o Sedumi.
Uma lista completa com todos os solvers que esse pacote suporta pode ser encontrada no arquivo 'yalmip.htm' que está dentro da pasta yalmip.

Sedumi: Esse pacote é um solver para problemas de programação semi-definida. Nesse trabalho ele será usado exclusivamente para a solução de LMIs. Atualmente os dois solvers mais usados na literatura especializada são o LMILab e o Sedumi. E o Yalmip funciona mais rápido com o Sedumi. Daí a motivação para usar esse pacote.
O LMILab possui um algoritmo baseado no método de pontos interiores, e a sua complexidade é dada por: K^3L. Já o Sedumi usa uma técnica de otimização sobre cones homogêneos duais, e a sua complexidade é dada por: $K^2L^{2.5} + L^{3.5}$, sendo K o número de variáveis escalares e L o número de linhas usadas nas LMIs. Assim, dependendo do problema um solver será mais rápido que o outro. Atualmente é comum encontrar na literatura especializada, trabalhos em que o autor compara as soluções obtidas por esses dois solvers.

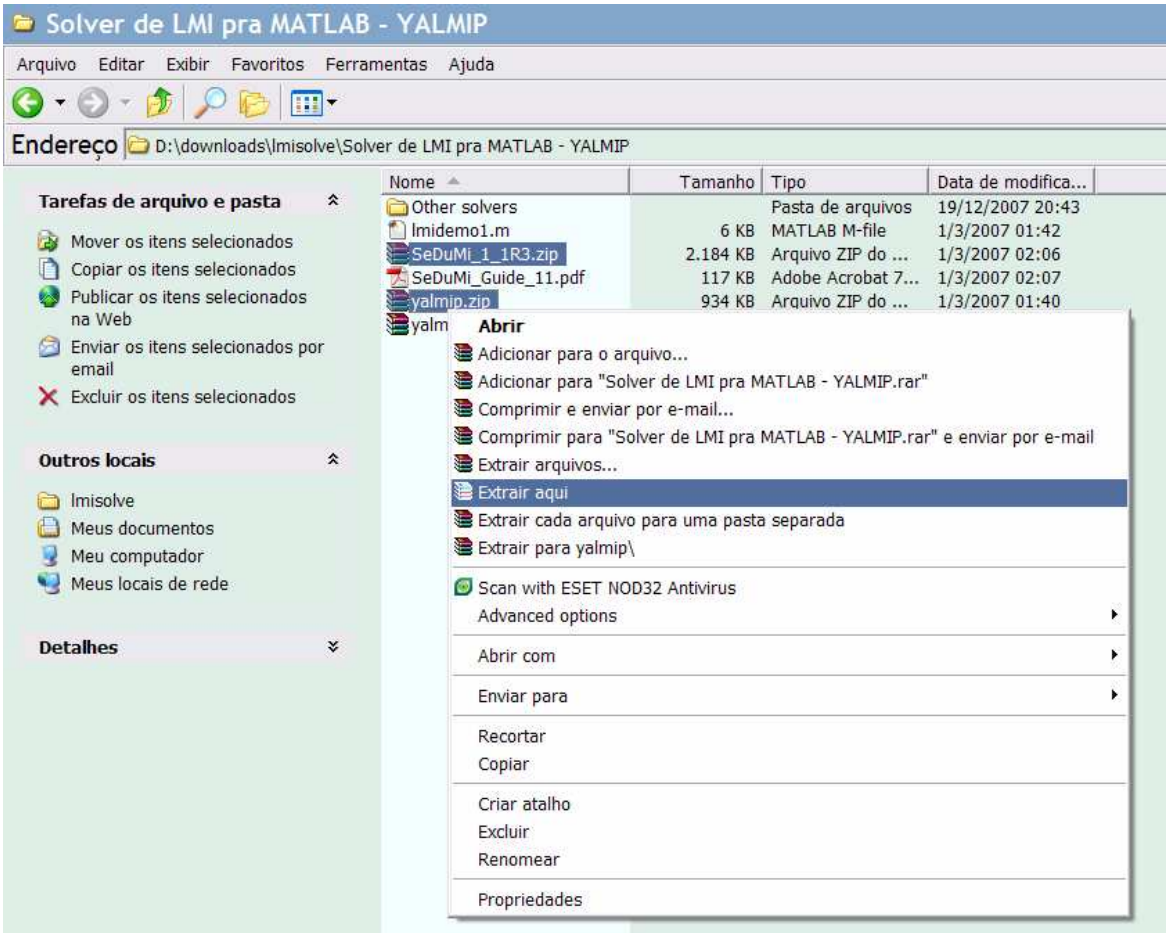
Objetivo do curso

Esse texto apresenta os comandos básicos do pacote Yalmip, necessários para a solução de LMIs no MATLAB.

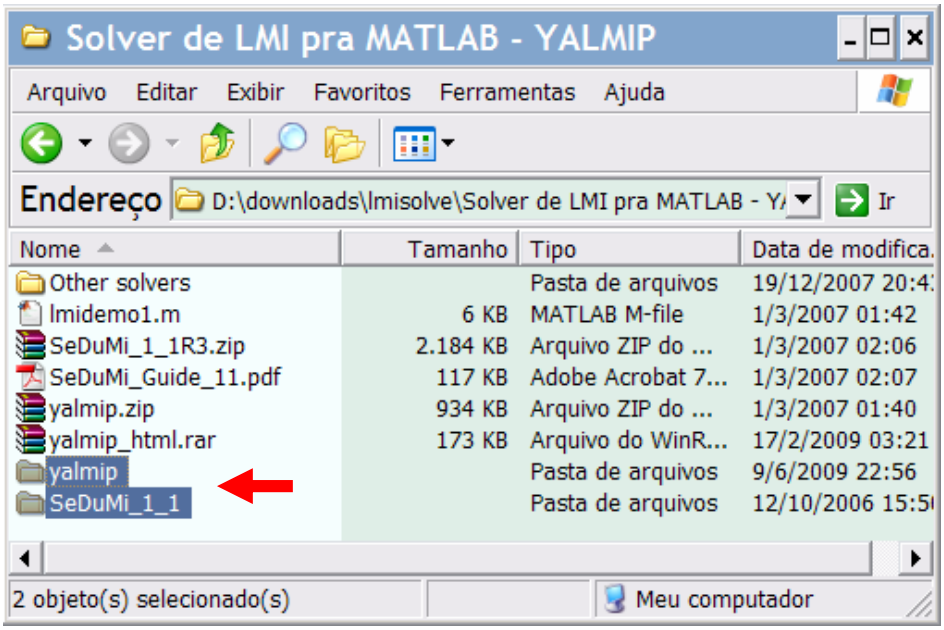
Instalação

O CD de instalação do MATLAB não vem com esses pacotes. Dessa forma eles precisam ser instalados manualmente. Os pacotes podem ser baixados pela internet nas seguintes URLs:
Sedumi 1.1R3
http://sedumi.ie.lehigh.edu/
Yalmip
http://control.ee.ethz.ch/~joloef/wiki/pmwiki.php

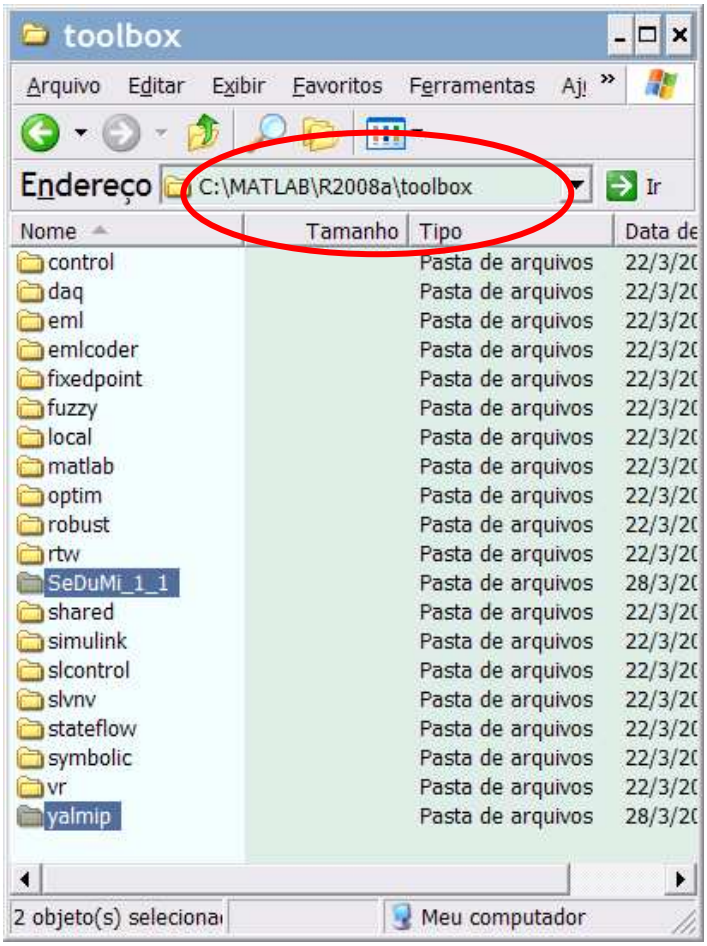
Depois do download o usuário terá baixado dois arquivos compactados (ZIP). Use um descompactador e extraia o conteúdo dos arquivos em uma pasta!



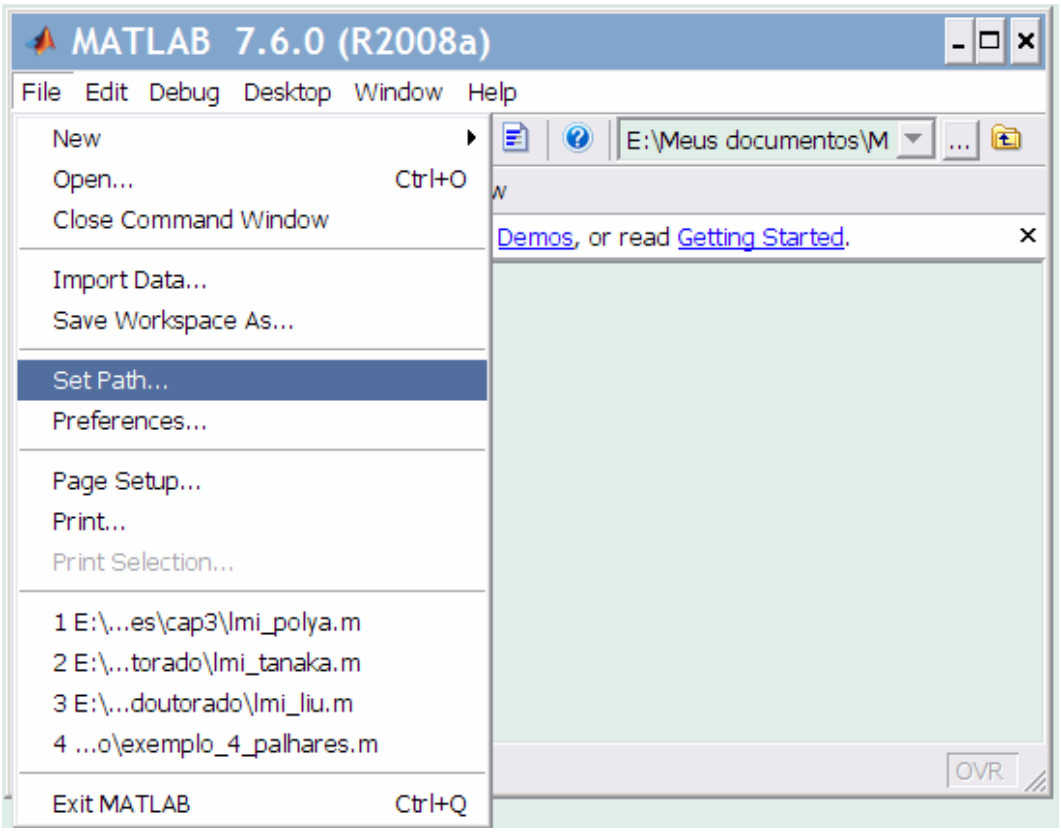
A extração do arquivo vai criar as pastas ‘yalmip’ e ‘SeDuMi_1_1’.



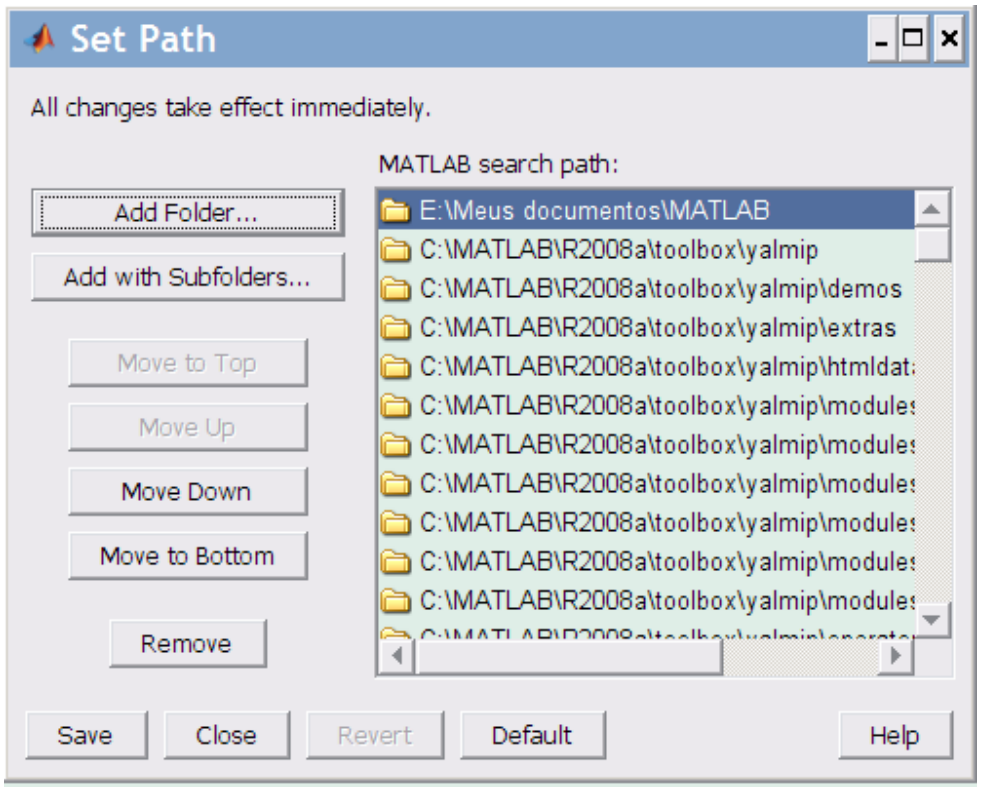
Os arquivos necessários para a instalação dos pacotes já estão prontos, agora só falta mapear essas pastas no MATLAB.
SUGESTÃO: Para evitar ficar com arquivos espalhados pelo computador, é interessante mover as pastas ‘yalmip’ e ‘SeDuMi_1_1’ para dentro da pasta toolbox do MATLAB.



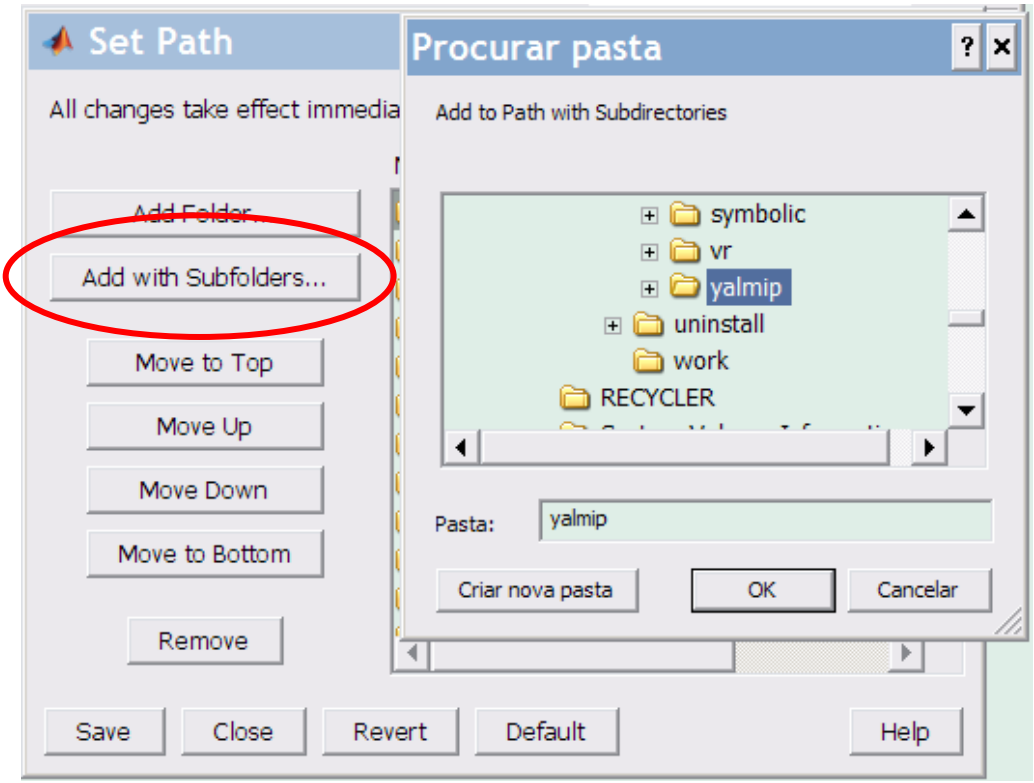
Para fazer o mapeamento (ou instalação) dos pacotes no MATLAB deve-se ir em:
Menu Files → Set Path...



Ao clicar nessa opção abre-se a seguinte janela

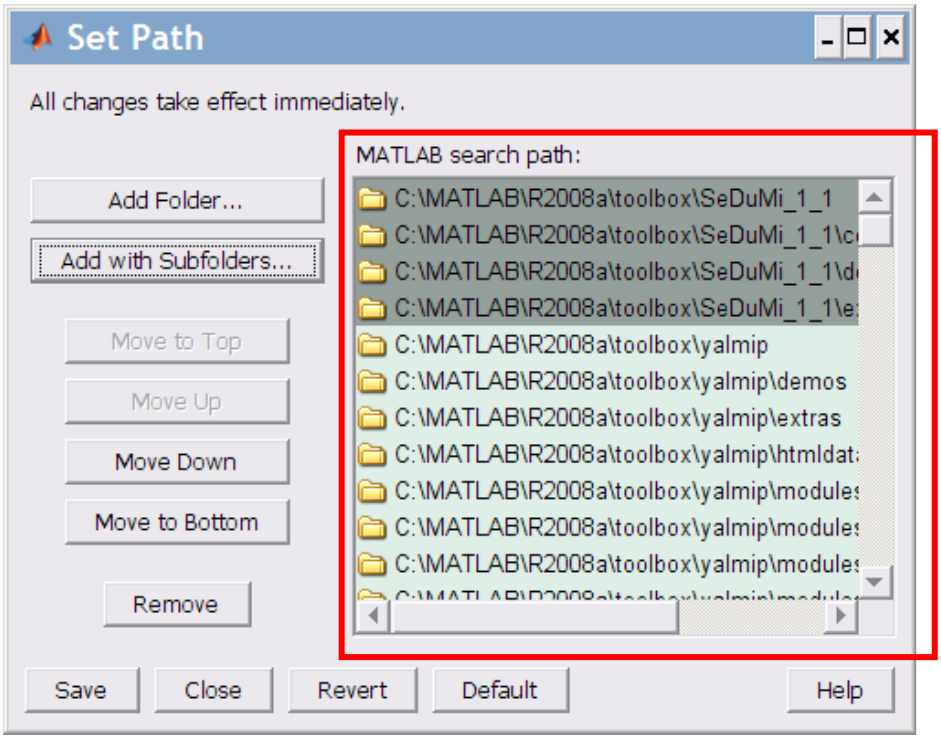


Aperte no botão ‘Add with Subfolders’ e navegue até o local onde está a pasta ‘yalmip’, selecione-a e aperte o botão OK.



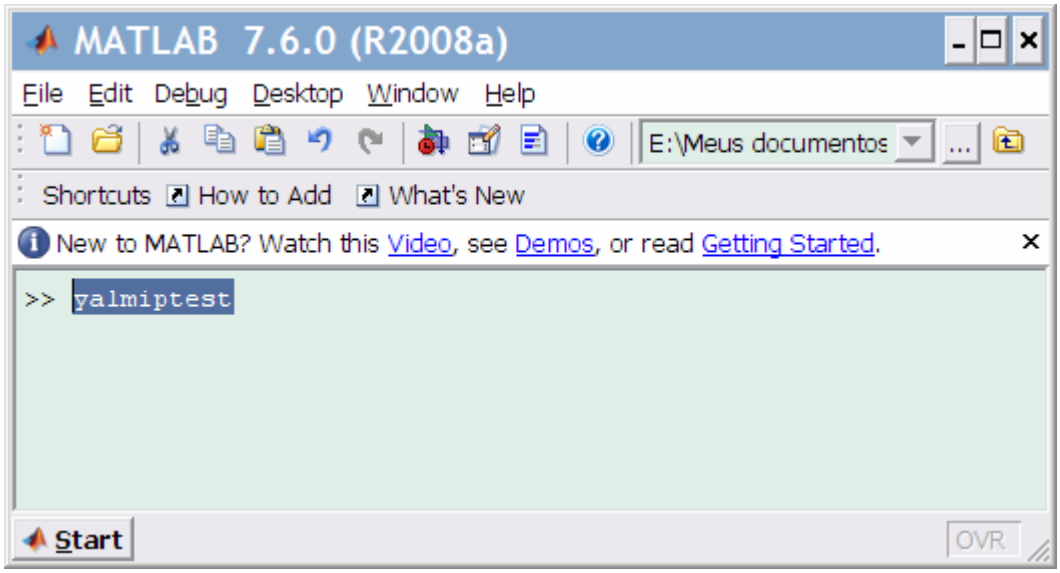
OBSERVAÇÃO: Se o usuário seguiu a sugestão da pág. 4, então a pasta 'yalmip' está dentro da pasta toolbox do MATLAB.

Repita esse procedimento com a pasta 'SeDuMi_1_1'. No final do procedimento o MATLAB terá adicionado o conteúdo dessas pastas no *MATLAB search path*

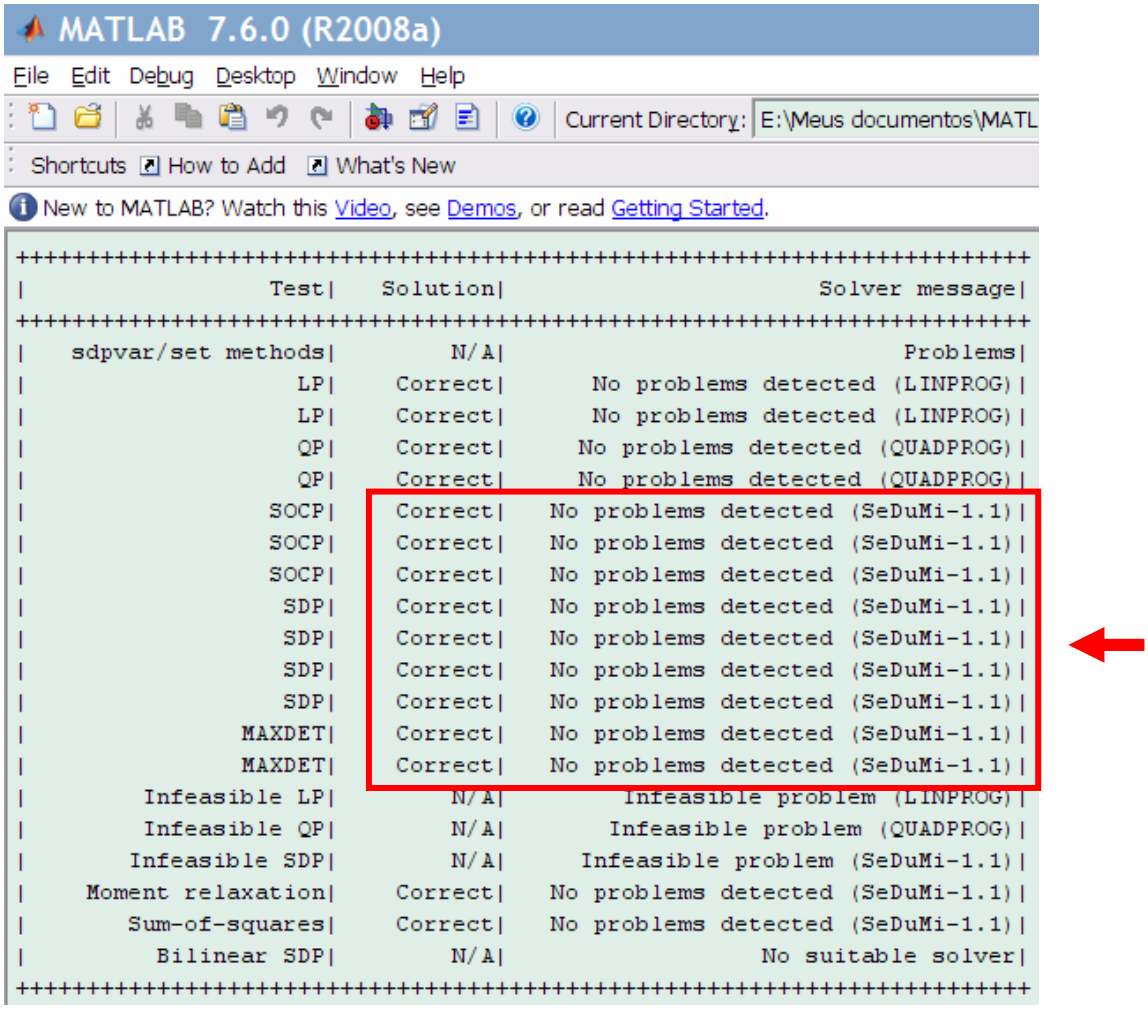


Agora aperte no botão *SAVE* e depois *CLOSE*.

Para testar se os pacotes foram instalados corretamente basta o usuário digitar ‘yalmiptest’ na linha de comando do MATLAB



Se os pacotes foram instalados corretamente o ‘yalmip’ fará um teste completo para verificar quais solvers estão disponíveis no MATLAB. ATENÇÃO, se o sedumi foi instalado corretamente, então o resultado do yalmip será igual ao da figura abaixo:



Uma vez que os pacotes foram instalados, então agora vamos começar a ilustrar o uso do ‘yalmip’ para a solução de LMIs no MATLAB.

Introdução ao yalmip

Agora nós vamos apresentar alguns comandos do pacote yalmip, necessários para a solução de LMIs.

Primeiramente vamos começar com o critério de estabilidade segundo Lyapunov.

Para tanto considere um sistema linear autônomo dado por:

$$\dot{x} = Ax. \tag{1}$$

Da teoria de estabilidade de Lyapunov, sabe-se que o sistema (1) é assintoticamente estável se e somente se, existir uma matriz simétrica P satisfazendo as seguintes LMIs:

$$\left.\begin{aligned} P > 0, \\ A'P + PA < 0. \end{aligned}\right\} \tag{2}$$

Por exemplo: considere o seguinte sistema dinâmico

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ -3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3}$$

A verificação de estabilidade do sistema (3) é realizada no MATLAB da seguinte maneira:

Exemplo 1: Estabilidade Lyapunov

```
1)A = [-1 2;-3 -4];
2)P = sdpvar(2,2);
3)Restr = set(P > 0) + set(A'*P+P*A < 0);
4)solvesdp(Restr,[]);
5)[r,d] = checkset(Restr);
6) if  sum(r < 0) == 0
7)    disp('Sistema estável')
8)    P_feasible = double(P);
9) else
10)   disp('Sistema instável')
11)end
```

Nesse caso a única variável do problema é a matriz P, que deverá ser encontrada pelo solver. Para definir uma variável no yalmip usa-se o comando sdpvar (linha 2). Esse comando tem os seguintes argumentos:

sdpvar(n,m,'field','type')

n = n^o. de linhas da matriz

m = n^o. de colunas da matriz

field = {'real','complex'}, diz se a matriz variável é real ou se assume números complexos

type = {'symmetric','full','hermitian','toeplitz','hankel','skew'}, diz se a matriz variável é simétrica, completa,....

Observe na linha 2) do exemplo que os argumentos field e type foram omitidos. Na verdade esses argumentos são muito pouco usados. Por exemplo, se a matriz variável é quadrada, então o yalmip entende que **`sdpvar(2,2)`** é equivalente à: **`sdpvar(2,2,'real','symmetric')`**

A linha 3) mostra como devemos armazenar as restrições do problema. Todas as restrições (LMIs) do problema devem ser escritas dentro do comando **`set()`** e armazenadas dentro de uma variável, nesse caso **Restr**. Essa variável é usada na linha 4) para resolver as LMIs.

A função **`solvesdp`** da linha 4) é o comando que diz para o yalmip resolver as LMIs. O **`solvesdp`** é usado para resolver qualquer problema de otimização. Os argumentos dessa função são:

sol = solvesdp(Restr,obj,opts)

sol = estrutura contendo as principais informações sobre o processo de resolução das LMIs. Por exemplo, a estrutura mostra se o solver teve algum problema numérico durante os cálculos, se o solver encontrou uma solução factível, tempo gasto pelo yalmip para montar as LMIs e tempo gasto pelo solver para resolvê-las.

Restr = Essa variável contém todas as LMIs do problema (veja linha 3).

obj = Essa variável representa a função objetivo do problema. Por default o yalmip entende que a variável obj é uma função de minimização, quando existir a necessidade maximização, o usuário deverá multiplicar a variável obj pelo sinal negativo (-obj). Esse argumento também pode ser vazio ([ ]), veja por exemplo na linha 4) do exemplo 1, como nós queríamos apenas uma solução factível, então a variável obj é vazia (obj=[ ]).

opts = essa variável contém uma estrutura do tipo sdpsettings. Com ela podemos alterar o solver usado pelo solvesdp, para resolver o problema. Esse parâmetro é discutido com mais detalhes nos Exemplos 4.1 e 4.2.

Ao longo do texto a função **`solvesdp`** é discutida com mais detalhes!

A função **`checkset`** (linha 5) obtém os resíduos das restrições do problema de otimização 'Primal' e 'Dual'. Esses detalhes podem ser ignorados pelo usuário agora no início, sendo necessário apenas saber como verificar se a solução obtida pelo **`solvesdp`** realmente satisfaz as restrições LMI.

Isso é feito na linha 6) com o laço **`if`**. A função **`double`** da linha 8) converte a estrutura P (matriz variável do yalmip) em uma matriz numérica.

Resolvendo o exemplo 1 no MATLAB se obtém os seguintes valores:

$$P = \begin{bmatrix} 0.5592 & 0.0449 \\ 0.0449 & 0.2481 \end{bmatrix}, \qquad \text{Autovalores de P} = 0.2417 \text{ e } 0.5655. \tag{4}$$

Observe que os autovalores matriz P são positivos, logo a matriz é definida positiva, além disso, os autovalores da matriz A^TP+PA são negativos (*-1.8528* e *-1.3399*). Então, existe uma matriz P satisfazendo as condições de Lyapunov (2) e, portanto o sistema (3) é assintoticamente estável.

Um detalhe importante a ser mencionado, é que nem sempre a resposta obtida pelo **`solvesdp`** é satisfatória e por isso a função **`checkset`** tem sempre que ser usada para verificar as restrições LMI.

Deste ponto em diante usaremos o yalmip para resolver diversos problemas de teoria de controle. Sempre que algum recurso novo for usado, ele será devidamente comentado.

Projeto de controladores usando realimentação de estados

Considere o sistema dinâmico

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{5}$$

sendo $x(t)\in\mathbb{R}^n$ os estados do sistema, $A\in\mathbb{R}^{nxn}$ e $B\in\mathbb{R}^{nxm}$ matrizes conhecidas e $u(t)\in\mathbb{R}^m$ a entrada de controle. O projeto de controladores com realimentação de estados consiste em encontrar uma matriz $L\in\mathbb{R}^{mxn}$ tal que ao realimentar o sistema (5) com a entrada `u(t)=-Lx(t)`, o sistema em malha fechada

$$\dot{x}(t) = (A - BL)x(t), \tag{6}$$

é assintoticamente estável.

O controlador `L` que garante a estabilidade assintótica de (6) pode ser encontrado com as LMIs

$$\left.\begin{array}{c} X > 0, \\ AX + XA' - BM - M'B' < 0. \end{array}\right\} \tag{7}$$

Sendo que o controlador `L` desejado é dado por $L=MX^{-1}$.

Para resolver esse problema considere o sistema dinâmico

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u. \tag{8}$$

Os autovalores desse sistema são: 1 e 4, logo o sistema é instável. O nosso objetivo é projetar um controlador L que seja capaz de estabilizar o sistema (8).

Exemplo 2: Projeto da realimentação de estados

```
12) A = [1 2;0 4];
13) B = [0;1];
14) X = sdpvar(2,2);
15) M = sdpvar(1,2);
16) Restr = set(X > 0) + set(A*X+X*A'-B*M-M'*B' < 0);
17) solvesdp(Restr,[]);
18) [r,d] = checkset(Restr);
19)  if  sum(r < 0) == 0
20)     disp('O sistema pode ser controlado')
21)     X = double(X);
22)     M = double(M);
23)     L = M*inv(X)
24)     Autovalor = eig(A-B*L)
25)  else
26)    disp('O sistema NÃO pode ser controlado')
27) end
```

Observe as linhas 14) e 15), como já foi comentado antes, quando a matriz é quadrada o yalmip automaticamente entende que a matriz é simétrica, na linha 14) o comando é equivalente à:
X = sdpvar(2,2,'real ','symmetric');

Já na linha 15) a matriz possui o número de linhas diferente do número de colunas, dessa forma o comando da linha 15) é equivalente à: **M = sdpvar(1,2,'real ','full');**

Um ponto importante a ser mencionado é que o parâmetro **`field`** ainda não foi usado, e provavelmente nunca será. Quando o parâmetro **`field`** é omitido o **`spdvar`** entende que a variável é real. Essa definição já é o suficiente para resolver a maioria dos problemas envolvendo o projeto de LMIs.

Resolvendo o exemplo 2 no MATLAB se obtém os seguintes valores:

$$\left.\begin{aligned} X &= \begin{bmatrix} 0.8397 & -0.6 \\ -0.6 & 1.4397 \end{bmatrix}, \\ M &= \begin{bmatrix} -0.1206 & 6.2589 \end{bmatrix}, \\ L &= MX^{-1} = \begin{bmatrix} 4.219 & 6.1056 \end{bmatrix}. \end{aligned}\right\} \tag{9}$$

Autovalores da matriz $A - BL = -0.5528 \pm 2.455i$.

Projeto de um controlador ótimo $\mathbf{H}_2$ usando LMI

Até o momento nenhum problema envolvendo otimização foi usado. Contudo é muito comum problemas de otimização em LMIs. Por exemplo, considere o sistema dinâmico dado por:

$$\left.\begin{aligned} \dot{x} &= Ax + B_1 w + B_2 u \\ y &= Cx \end{aligned}\right\} \tag{10}$$

Sendo w uma entrada exógena (ruído) no sistema. O nosso objetivo é projetar um controlador $L \in \mathbb{R}^{mxn}$ tal que o ruído w tenha a menor influência possível no comportamento dinâmico do sistema (10) realimentado com a entrada `u=-Lx`. Esse projeto pode ser feito minimizando a norma $\mathbf{H}_2$ entre a entrada w e a saída y. Que é equivalente ao seguinte problema de otimização [1]:

$$\left.\begin{aligned} &\min \ Tr(Z) \\ &s.a \quad \begin{bmatrix} X & B_1 \\ B_1^T & Z \end{bmatrix} > 0, \\ &\quad \begin{bmatrix} AX + XA^T + B_2 Y + Y^T B_2^T & XC^T \\ CX & -\mathrm{I} \end{bmatrix} < 0, \\ &\quad X > 0, \\ &\quad Z > 0. \end{aligned}\right\} \tag{11}$$

Dadas as matrizes

$$A_1 = \begin{bmatrix} -2 & 1 & 1 & 1 \\ 3 & 0 & 0 & 2 \\ -1 & 0 & -2 & -3 \\ -2 & -1 & 2 & -1 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Esse problema pode ser resolvido no MATLAB com o seguinte código:

Exemplo 3: Minimizando a norma $\mathbf{H}_2$

```
28)A = [-2 1 1 1;3 0 0 2;-1 0 -2 -3;-2 -1 2 -1];
29)B1 = [1 0 0;0 0 0;1 0 0;0 0 0];
30)B2 = [0 0;1 0;0 0;0 1];
31)C = [1 0 -1 0;0 0 0 0;0 0 0 0]
32)n= size(A,1);
33)m1=size(B1,2);
34)m2=size(B2,2);
```

```
35) mC=size(C,1);
36) IdC = eye(mC)
37) X = sdpvar(n,n)
38) Z = sdpvar(mC,mC)
39) Y = sdpvar(m2,n)
40) Restr = set(X > 0) + set(Z > 0) + ...
    set([X B1;B1' Z]>0) + ...
    set([A*X+X*A'-B2*Y-Y'*B2' X*C';C*X -IdC]<0);
41) sol = solvesdp(Restr,trace(Z))
42) [p,d]=checkset(Restr);
43) if sum(p < 0) == 0
44) disp('O sistema pode ser controlado')
45) X = double(X)
46) Y = double(Y)
47) L = Y*inv(X)
48) Autovalor_AN = eig(A-B2*L)
49) else
50) disp('O sistema NÃO pode ser controlado')
51) end
```

Nesse exemplo começamos a usar comandos para automatizar o processo da criação das LMIs. Os comandos das linhas 32) à 35) são usados para armazenar o número de linhas (ou colunas) das matrizes A, B1, B2 e C. Esse tipo de operação é interessante porque permite que as LMIs sejam facilmente reaproveitadas para a solução de outros exemplos numéricos. A linha 36) cria uma matriz identidade de dimensão 3x3 (número de linhas da matriz C) que será usada na LMI da linha 40.3).

A grande novidade nesse exemplo é que ele representa um problema de otimização. Para resolver um problema desse tipo, basta carregar a função objetivo no segundo argumento do comando **`solvesdp`** (linha 41)). O traço de uma matriz pode ser calculado com o comando **`trace`**. Como foi mencionado anteriormente, o **`solvesdp`** entende que é para minimizar o traço da matriz Z, se fosse um problema de maximização, então teríamos que multiplicar a função objetivo pelo sinal negativo. Dessa forma, o comando da linha 41) seria reescrito como:

```
max Tr(Z) = solvesdp(Restr,-trace(Z))
```

O yalmip NÃO consegue calcular a dimensão de matrizes escalares em LMIs, por exemplo, se na linha 40.3) colocássemos **-1** ao invés de **–IdC = –eye(3)**, o programa iria gerar um erro de dimensionamento. Esse detalhe é muito importante, já que o pacote LMI control toolbox faz isso automaticamente. Ao implementar LMIs no yalmip deve-se ter muita atenção no dimensionamento das matrizes escalares (identidades ou matrizes nulas).

A partir desse ponto não exibiremos as soluções encontradas pelo MATLAB. O leitor poderá ver os resultados executando os respectivos arquivos do MATLAB que acompanham o material.

Projeto de controladores fuzzy usando realimentação de estados

Considere um modelo fuzzy Takagi-Sugeno dado por:

$$\dot{x} = \sum_{i=1}^{r} \alpha_i(z(t)) \left(A_i x + B_i u \right) \tag{12}$$

$$\alpha_i(z(t)) \geq 0, \quad i = 1, 2, \cdots, r, \quad \sum_{i=1}^{r} \alpha_i(z(t)) = 1 \tag{13}$$

Pode-se projetar um controlador fuzzy para o sistema (12), usando a Compensação Paralela Distribuída [2]. Esse procedimento consiste em projetar controladores para cada um dos modelos locais e o controlador global, que é não-linear em geral, é obtido através da combinação fuzzy dos controladores locais. Dessa forma, o projeto da realimentação de estados pode ser redefinido como:

Encontrar matrizes constantes L_1, L_2,…, $L_r \in \mathbb{R}^{mxn}$ tal que ao realimentar o sistema (12) com a entrada $u = -\sum_{i=1}^{r} \alpha_i(z(t)) L_i x = -L(\alpha)x$, o sistema em malha fechada seja globalmente assintoticamente estável. As próximas LMIs fornecem uma condição suficiente para a existência da matriz $L(\alpha)$, que garante a estabilidade do sistema (12).

$$\left.\begin{aligned} X > 0, \\ A_i X + X A_i^T - B_i Y_i - Y_i^T B_i^T < 0, \\ (A_i + A_j) X + X (A_i + A_j)^T - B_i Y_j - Y_j^T B_i^T - B_j Y_i - Y_i^T B_j^T < 0. \end{aligned}\right\} \tag{14}$$

sendo $L_i = M_i X^{-1}, \quad i = 1, 2, \cdots, r.$

Abaixo segue a solução de um exemplo numérico.

Exemplo 4.1: PDC_LMILAB

```
52) A1 = [0.5 0 0;0 -1 0;0 0 2]
53) A2 = [1.5 0 0;0 -1 0;0 0 6]
54) B1 = [1;1;1]
55) B2 = [1;1;1]
56) n=size(A1,2); m=size(B1,2);
57) X = sdpvar(n,n)
58) M1 = sdpvar(m,n)
59) M2 = sdpvar(m,n)
60) Restr = set(X > 0) +
           set(A1*X+X*A1'-B1*M1-M1'*B1' < 0) +...
           set(A2*X+X*A2'-B2*M2-M2'*B2' < 0) +...
           set((A1+A2)*X+X*(A1+A2)'-...
                B1*M2-B2*M1-M2'*B1'-M1'*B2' < 0);
61) opts=sdpsettings;
62) opts.solver='lmilab';
63) opts.verbose=0;
64) sol = solvesdp(Restr,[],opts)
65) p,d]=checkset(Restr);
66) if sum(p < 0) == 0
67)     disp('O sistema pode ser controlado')
68)     X = double(X)
69)     M1 = double(M1)
70)     M2 = double(M2)
71)     L1 = M1*inv(X)
72)     L2 = M2*inv(X)
73)     Autovalor_AN = [eig(A1-B1*L1) eig(A2-B2*L2)]
```

```
74) else
75)     disp('O sistema NÃO pode ser controlado')
76) end
```

Esse exemplo foi colocado por dois motivos. Primeiramente, ilustramos o fato que o yalmip suporta vários solvers, dessa maneira pode-se escolher qual solver será usado para resolver as LMIs. Ao longo do texto foi discutida a instalação do SEDUMI. Quando o SEDUMI está instalado, o yalmip automaticamente o transforma no solver padrão. Contudo, o yalmip também permite que as LMIs sejam resolvidas com o solver do MATLAB (LMILAB). Para fazer isso basta usar os comandos das linhas 61) e 62). Na linha 61) é criada uma variável com a estrutura do tipo **`sdpsettings`**. Essa estrutura permite escolher o solver, e alterar alguns dos seus parâmetros. A configuração da linha 63) oculta os resultados gerados pelo solver. Esses resultados se referem ao algoritmo usado pelo solver e não são necessários. Para fazer com que o comando **`solvesdp`** use as opções é necessário passar a variável **opts** como argumento (veja linha 64).

Um outro motivo para a abordagem de sistemas fuzzy com o yalmip, é que o pacote possui uma estrutura que facilita a programação de LMIs. O código do Exemplo 4: PDC_LMILAB

é específico para o exemplo numérico apresentado, se precisássemos usar o código em um outro exemplo, provavelmente o código teria que ser reescrito. O yalmip permite que usemos algumas estruturas do MATLAB para efetuar uma programação mais eficiente.

Por exemplo, pode-se resolver as LMIs (14) de uma maneira mais eficiente com o seguinte código:

Exemplo 4.2: PDC_Automatico

```
77) ri = 2;
78) A = cell(ri,1);
79) B = cell(ri,1);
80) A{1,1} = [0.5 0 0;0 -1 0;0 0 2];
81) A{2,1} = [1.5 0 0;0 -1 0;0 0 6];
82) celldisp(A)
83) B{1,1} = [1;1;1];
84) B{2,1} = [1;1;1];
85) celldisp(B)
86) n=size(A{1,1},2);
87) m=size(B{1,1},2);
88) X = sdpvar(n,n)
89) M = cell(ri,1);
90) for a1=1:ri,
91)    M{a1,1} = sdpvar(m,n);
92) end
93) celldisp(M)
94) Restr = set(X > 0);
95) for a1=1:ri,
96)   Restr = Restr + set(A{a1,1}*X+X*A{a1,1}'-...
                B{a1,1}*M{a1,1}-M{a1,1}'*B{a1,1}' < 0);
97)     for a2=a1+1:ri,
98)       Restr = Restr + set((A{a1,1}+A{a2,1})*X+...
          X*(A{a1,1}+A{a2,1})'-B{a1,1}*M{a2,1}-...
          B{a2,1}*M{a1,1}-M{a2,1}'*B{a1,1}'-...
          M{a1,1}'*B{a2,1}' < 0);
```

```
99)    end
100) end
101) opts=sdpsettings;
102) opts.solver='lmilab';
103) sol = solvesdp(Restr,[],opts)
104)  [p,d]=checkset(Restr);
105) if sum(p < 0) == 0
106)     disp('O sistema pode ser controlado')
107)     X = double(X)
108)     L = cell(ri,1);
109)     Autovalor_AN = [];
110)     for a1=1:ri,
111)         M{a1,1} = double(M{a1,1});
112)         L{a1,1} = M{a1,1}*inv(X);
113)     end
114)     celldisp(L)
115)     disp(Autovalor_AN)
116) else
117)     disp('O sistema NÃO pode ser controlado')
118) end
```

Estruturas do tipo **`cell`** permitem a criação de vetores ou matrizes de elementos complexos. Na linha 78) criamos um vetor 2x1 de matrizes. Esse vetor é usado nas linhas 80) e 81) para armazenar as matrizes A_1 e A_2 dos modelos locais. O mesmo é realizado com a matriz B. Esse tipo de programação permite usarmos o laço **`for`** para gerar as LMIs. Com isso o código MATLAB se torna geral, e pode ser usado para resolver diversos exemplos sem a necessidade de alteração do código das LMIs. Nesse caso é necessário mudar apenas as variáveis **`ri`** (linha 77), $\mathbf{A_i}$ (linhas 80 e 81) e $\mathbf{B_i}$ (linhas 83 e 84).

Esse tipo de programação é interessante quando se pretende disponibilizar os arquivos do MATLAB. No próximo tópico um outro projeto de controladores fuzzy é abordado com esse tipo de programação.

Condições menos conservadoras para o projeto de controladores fuzzy usando realimentação de estados

A partir da Compensação Paralela Distribuída pode-se obter condições menos conservadoras para o projeto da realimentação de estados em sistema fuzzy, por exemplo em [3] os autores obtém condições menos conservadoras com as seguintes LMIs

$$\left.\begin{array}{c} X>0, \\ A_i X + X A_i^T - B_i Y_i - Y_i^T B_i^T < Z_{ii} + Z_{ii}^T, \\ (A_i + A_j)X + X(A_i + A_j)^T - B_i Y_j - Y_j^T B_i^T - B_j Y_i - Y_i^T B_j^T < Z_{ij} + Z_{ji}, \quad i<j \\ \begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1r} \\ Z_{21} & Z_{22} & \cdots & Z_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{r1} & Z_{r2} & \cdots & Z_{rr} \end{bmatrix} < 0. \end{array}\right\} \quad (15)$$

sendo $L_i = M_i X^{-1}$, $Z_{ij} = Z_{ji}^T$, $i = 1, 2, \cdots, r.$

As LMIs (15) podem ser resolvidas no MATLAB com o seguinte código.

Exemplo 5: Automatico

```
ri = 2;
A = cell(ri,1);
B = cell(ri,1);
A{1,1} = [0.5 0 0;0 -1 0;0 0 2];
A{2,1} = [1.5 0 0;0 -1 0;0 0 6];
B{1,1} = [1;1;1];
B{2,1} = [1;1;1];
n=size(A{1,1},2);
m=size(B{1,1},2);
X = sdpvar(n,n)
M = cell(ri,1);
Z = cell(ri,ri);
for a1=1:ri,
   M{a1,1} = sdpvar(m,n);
   Z{a1,a1} = sdpvar(n,n,'full');
   for a2=a1+1:ri,
       Z{a1,a2} = sdpvar(n,n,'full');
       Z{a2,a1} = Z{a1,a2}';
   end
end
celldisp(M)
celldisp(Z)
Restr = set(X > 0);
for a1=1:ri,
  Restr = Restr + set(A{a1,1}*X+...
          X*A{a1,1}'-B{a1,1}*M{a1,1}-...
          M{a1,1}'*B{a1,1}' < Z{a1,a1}+Z{a1,a1}');
  for a2=a1+1:ri,
     Restr = Restr + set((A{a1,1}+...
             A{a2,1})*X+ X*(A{a1,1}+A{a2,1})'-...
             B{a1,1}*M{a2,1}-B{a2,1}*M{a1,1}-...
```

```
                M{a2,1}'*B{a1,1}'-M{a1,1}'*B{a2,1}'<...
                Z{a1,a2}+Z{a1,a2}');
146)   end
147) end
148) str='[';
149) for t1 = 1:ri,
150)     for t2 = 1:ri,
151)         str=strcat(str,sprintf(' Z{%d,%d}',
                 t1,t2));
152)     end
153)     if t1< ri
154)         str=strcat(str,';');
155)     else
156)         str=strcat(str,' ]<0');
157)     end
158) end
159) disp(str)
160) Restr = Restr + set(eval(str));
161) sol = solvesdp(Restr,[])
162)  [p,d]=checkset(Restr);
163) if sum(p < 0) == 0
164)     disp('O sistema pode ser controlado')
165)     X = double(X)
166)     L = cell(ri,1);
167)     Autovalor_AN = [];
168)     for a1=1:ri,
169)         M{a1,1} = double(M{a1,1});
170)         L{a1,1} = M{a1,1}*inv(X);
171)     end
172) else
173)     disp('O sistema NÃO pode ser controlado')
174) end
```

Observe nas linhas 133) e 135) que o parâmetro **‘full’** foi usado. Como a variável Z_{ij} é quadrada, então a opção tem que ser mencionada, senão o yalmip cria uma variável simétrica (veja linha 128). O problema desse código está na escrita da última LMI em (15). A dimensão dessa LMI depende do número de vértices nos modelos locais. Uma maneira de escrever essa LMI no MATLAB é usando manipulação de strings. Na linha 148) iniciamos uma string com o símbolo de colchetes e nas linhas 149) à 158) usamos o comando **for** para escrever a estrutura da LMI. Na linha 160) o comando **eval** é usado para transformar a string **str** em uma expressão matemática para ser usada no comando **set**.

O yalmip permite armazenar expressões com variáveis para serem usadas ao longo do programa. No próximo tópico ilustramos um problemas envolvendo um somatório de variáveis.

Análise de estabilidade considerando uma função de Lyapunov dependente de parâmetros

Uma outra linha de pesquisa em modelos Takagi-Sugeno consiste em obter condições menos conservadoras para o sistema não forçado $\dot{x} = \sum_{i=1}^{r} \alpha_i(z(t))A_i$, usando uma função de Lyapunov dependente de parâmetros. Isto é, encontrar uma matriz $\sum_{i=1}^{r} \alpha_i(z(t))P_i = P(\alpha)$, satisfazendo as LMIs:

$$\left.\begin{array}{c} P(\alpha) > 0, \\ \dot{P}(\alpha) + P(\alpha)A(\alpha) + A(\alpha)^T P(\alpha) < 0. \end{array}\right\} \tag{16}$$

Resolver as LMIs (16) para todo α do simplex unitário é um problema de dimensão infinita, o que dificulta o seu uso na prática. Considerando $\left|\dot{\alpha}_i\right| \le \delta, i = 1, 2, \cdots, r,$ sendo $\delta > 0$ uma constante conhecida. Em [4] os autores mostram que uma condição suficiente de dimensão finita para a solução de (16) pode ser obtida com:

$$\left.\begin{array}{c} P_i > 0, \\ \delta\sum_{i=1}^{r} P_k + \frac{1}{2}\left(P_j A_i + A_i^T P_j + P_i A_j + A_j^T P_i\right) < 0, \qquad i \le j \end{array}\right\} \quad i, j = 1, 2, \cdots, r. \tag{17}$$

O código usado para resolver essas LMIs é:

Exemplo 6: Automático

```
175) ri = 2;
176) A = cell(ri,1);
177) A{1,1} = [-5 -4;-1 -2];
178) A{2,1} = [-2 -4;20 -2];
179) n=size(A{1,1},2);
180) P = cell(ri,1);
181) Pfi = 0;
182) Restr = [];
183) for a1=1:ri,
184)    P{a1,1} = sdpvar(n,n);
185)    Pfi = Pfi + 0.85*P{a1,1}; % delta=0.85
186)    Restr = Restr + set(P{a1,1}>0);
187) end
188) for a1=1:ri,
189)   for a2=a1:ri,
190)      Restr = Restr + set(Pfi + ...
         0.5*(P{a1,1}*A{a2,1}+A{a2,1}'*P{a1,1}+...
         P{a2,1}*A{a1,1}+A{a1,1}'*P{a2,1})< 0);
191)     end
192) end
193) sol = solvesdp(Restr,[])
194)  [p,d]=checkset(Restr);
195) if sum(p < 0) == 0
196)     disp('O sistema é estável')
197)     for a1=1:ri,
```

```
198)         P{a1,1} = double(P{a1,1});
199)     end
200)     celldisp(P)
201) else
202)     disp('O sistema é instável')
203) end
```

É complicado criar um código geral para as LMIs (17) com o pacote padrão do MATLAB. O problema está em gerar o somatório $\delta\sum_{i=1}^{r} P_k$ para todo i≤j. Já o yalmip permite que esse somatório seja armazenado em uma variável (linha 185) para ser usado posteriormente nas restrições do problema (linha 190).

Notas finais

Nesse texto foram abordados os principais comandos do yalmip para a solução de LMIs. Também foi explorado alguns recursos de programação que podem ser usados com esse pacote para melhorar a escrita de LMIs no MATLAB. Ao longo do texto foram exibidos seis exemplos de aplicação em engenharia de controle. Acreditamos que essas aplicações já sejam o suficiente para o leitor ter uma idéia da potencialidade do yalmip, e também servem como base para a solução de outros problemas com LMIs.
Em caso de dúvidas, críticas, ou sugestões, envie um email para:
flaviof15@yahoo.com.br

Referências

[1] E. Assunção, M. C. M. Teixeira. “Projeto de sistemas de controle via LMIS usando o MATLAB,” in J. Balthazar, V. Oliveira, G. Silva and J. Rosário (eds), Aplicações em Dinâmica e Controle, São Carlos, pp. 350–368, 2001.

[2] K. TANAKA, T. IKEDA, H. O. WANG. “Fuzzy regulators and fuzzy observers: Relaxed stability conditions and LMI-based designs,” IEEE Transactions on Fuzzy Systems, v. 6, n. 2, p. 250–265, 1998.

[3] X. Liu and Q. Zhang, “New approaches to $\mathbf{H}_\infty$ controller designs based on fuzzy observers for T–S fuzzy systems via LMI,” Automatica, vol. 39, no. 5, pp. 1571–1582, 2003.

[4] K. Tanaka, T. Hori, H.O. Wang, “A multiple Lyapunov function approach to stabilization of fuzzy control systems”, IEEE Trans. on Fuzzy Systems, Vol. 11 (4), pp 582-589, 2003.