Estudo preliminar de LMI's

Ramon C. Lopes
Disciplina: Detecção e Diagnose de Falhas em Sistemas Dinâmicos
professores: Reinaldo Martinez Palhares
Walmir Matos Caminhas

***Resumo*— Este texto apresenta uma descrição sucinta de Desigualdades Matriciais Lineares (LMI - Linear Matrix Inequalities) com o objetivo de aplicá-las como restrições em problemas de otimização convexa orientados para a síntese de controladores e observadores dinâmicos robustos no contexto de diagnose de falhas em sistemas incertos com entradas desconhecidas. Após uma fundamentação inicial, é apresentada a formulação do problema e um exemplo prático é implementado.**

I. Introdução

Um estudo sobre o projeto de um observador ótimo baseado em um modelo dinâmico livre de incertezas nos parâmetros é projetado e um segundo observador robusto é obtido de forma a obter um resíduo que busca maximizar a sensibilidade às faltas minimizando os alarmes falsos devido a variação nos parâmetros. Este estudo foi baseado no artigo *An LMI approach to design robust fault detection filter for uncertain LTI systems*[1].

II. Otimização Convexa

1) Método da Barreira: O método da barreira propõe inicialmente a solução para um problema do tipo:

$$\begin{array}{ll} \texttt{min} & f(x) \\ \texttt{s.a:} & g_i(x) \geq 0, \ \texttt{i=1,...,m.} \end{array} \tag{1}$$

considerando as funções continuamente diferenciáveis e o método estritamente factível, trabalhando com pontos no interior da região viável [2]. O método consiste de se criar uma barreira através de uma função logarítimica:

$$\phi(x) = -\sum_{i=1}^{m} log(g_i(x)) \tag{2}$$

ou através de uma função inversa:

$$\phi(x) = \sum_{i=1}^{m} \frac{1}{g_i(x)} \tag{3}$$

cria-se a função da barreira como:

$$\beta(x,\mu) = f(x) - \mu \sum_{i=1}^{m} log(g_i(x)) \tag{4}$$

a partir da função logarítmica ou:

$$\beta(x,\mu) = f(x) + \mu \sum_{i=1}^{m} \frac{1}{g_i(x)} \tag{5}$$

a partir da função inversa.

Estas duas funções têm o valor do custo muito aumentado quando o ponto viável se aproxima da fronteira das restrições.

2) Barreira auto-concordante: Para problemas quadráticos o método de Newton converge em uma iteração. O grau de não linearidade das funções envolvidas na função de custo define o comportamento do método de Newton. Para a garantia de um bom desempenho do método em funções mais complexas exige-se o conceito de auto-concordância aplicado ao subproblema com uma barreira simples.

Definição II-.2.1 (Barreira auto-concordante): Uma função de barreira convexa é chamada auto-concordante se:

$$|\nabla^3 F(x)[h,h,h]| \leq 2(h^T \nabla^2 F(x) h)^{3/2} \tag{6}$$

cuja sequência $\{x_i\} \subset intS$ e todo $h \in \mathbb{R}^n$.

□

Considerando uma função de barreira logarítmica auto-concordante do tipo:

$$F(x) = -ln\ Det(x) \tag{7}$$

pode-se encontrar a primeira e a segunda derivadas [3] como:

$$DF(x)[h] = -Tr\{x^{-1}h\} \tag{8}$$

e

$$D^2F(x)[h,h] = Tr\{x^{-1}hx^{-1}h\} \tag{9}$$

que podem ser implementadas como [4]:

$$g_i(x) = -TrF(x)^{-1}F_i \tag{10}$$

$$H_{ij}(x) = TrF(x)^{-1}F_iF(x)^{-1}F_j \tag{11}$$

para $i,j = 1,...,m$.

3) Centro Analítico: Baseado na manutenção de uma desigualdade matricial linear $F(x) > 0$ estritamente factível funcionando como restrição para um problema de otimização convexa, o método busca o ponto ótimo a partir de uma função estritamente convexa [4] do tipo:

$$x^* \triangleq \underset{x}{\texttt{argmin}} \quad \phi(x). \tag{12}$$

O método consiste em definir a direção da função de custo pelo método de Newton e definir o tamanho ideal do passo pelo método da bisseção [5].

Algoritmo II-.3.1 - Método dos Centros Analíticos

passo 1: Computar a direção de Newton a partir da direção $[F(x_k)]^{-1}\nabla f(x_k)^T$ dada pelas equações (10) e (11);

passo 2: Encontrar o passo ótimo da equação $x_{k+1} = x_k - \texttt{p}[F(x_k)]^{-1}\nabla f(x_k)^T$ pelo método da bisseção;

passo 3: Atualizar a variável de otimização através da equação $x_{k+1} = x_k - \texttt{p}[F(x_k)]^{-1}\nabla f(x_k)^T$.

A. Formulação de LMI's

1) Norma Euclideana: Considere o exemplo de aplicação da programação semidefinida em funções que admitem uma representação definida positiva sugerida em [3] para $\| y \|_2$, $y \in \mathbb{R}^2$ com $\mu = \{n+1\}$,

$$\{(t,y) |\ t \geq \| y \|_2\} = \beta^{-1}(S_\mu^+), \quad \beta(t,y) = \begin{pmatrix} t & y^T \\ y & tI_n \end{pmatrix} \tag{13}$$

cuja solução para o cone de segunda ordem encontra-se na figura 1:

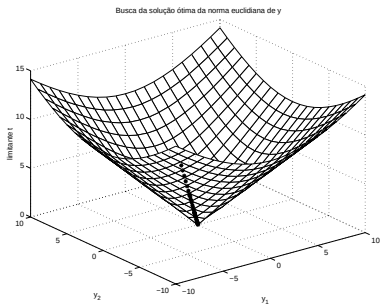

Fig. 1. Cone de segunda ordem de matriz definida positiva simétrica

Esta solução foi obtida a partir da utilização do método da seção II-.3 inicializado no ponto $y_0 = [-8, -8]$. A formulação torna-se:

$$\begin{array}{ll} \texttt{min} & t \\ \texttt{s.a:} & \begin{bmatrix} t & y^T \\ y & tI_n \end{bmatrix} \geq 0 \end{array} \tag{14}$$

Este método com alguns refinamentos é disponibilizado através de rotinas computacionais e denominado método projetivo (Ex. Matlab) a partir da formulação feita em [6] ou primal-dual (Ex. Scilab) com o tratado feito em [7] para a solução das Inequações Matriciais Lineares, como apresentado no código a seguir em Scilab:

Codigo 1 Inicialização da LMI {arquivo: testanormay.sci}

```
function [t]=testanormay(y)
getf('diretorio/ normay_eval.sci');
LIST0=list(t_init);
LISTF=lmisolver(LIST0,normay_eval);
[t]=LISTF(:)
```

Codigo 2 Formulação da LMI {arquivo: normay_eval.sci}

```
function [LME,LMI,OBJ]=normay_eval(XLIST)
[t]=XLIST(:)
LME=[ ];
LMI1=[t,y;...
  y',t*eye(size(y',1),size(y,2))];
LMI=list(LMI1);
OBJ=[t];
```

Para a execução da rotina, deve ser carregado o arquivo testanormay.sci a partir do comando getf() no diretório corrente e digitado o comando [t]=testanormay([1,2,3]), por exemplo para encontrar a norma do vetor [1,2,3]. O mesmo código em Matlab pode ser visto no Código (3):

Codigo 3 Formulação da LMI {arquivo: normay.m}

```
function [t]=normay(y)
setlmis([]);
T=lmivar(1,[1 1]);
lmi1=newlmi;
lmiterm([-lmi1 1 1 T],1,1);
lmiterm([-lmi1 1 2 0],y);
lmiterm([-lmi1 2 2 T],eye(size(y',1),size(y,2)),1);
a=getlmis;
options=[1e-15,0,0,0,0];
c=mat2dec(a,1);
[topt]=mincx(a,c,options)
t=dec2mat(a,topt,T)
```

digitando-se y=normay([1,2,3]) na linha de comando do Matlab.

III. Geração do Modelo Residual de Referência

Dado o sistema:

$$\dot{x} = (A + \Delta A)x + (B + \Delta B)u + B_f f + B_d d, \tag{15}$$

$$y = Cx + Du + D_f f + D_d d \tag{16}$$

O Modelo Residual de Referência (MRR) sugerido no artigo [1] pode ser obtido com a solução do problema:

$$\min J = \frac{\|V(D_d+(sI-A+HC)^{-1}(B_d-HD_d)\|_\infty}{inf_{\omega\in\Omega}\sigma_i(V(D_f+(j\omega I-A+HC)^{-1}(B_f-HD_f)))}, \ \Omega \subseteq [0, \infty) \tag{17}$$

Teorema 1 Dado um sistema como nas Equações 15 e 16 com $\Delta A = 0$ e $\Delta B = 0$, considerando (A1),(A2) e (A3) válidas, então:

$$H^* = (B_d D_d^T + Y C^T)Q^{-1}, \tag{18}$$

$$V^* = Q^{-1/2} \tag{19}$$

resolve o problema de otimização 17, sendo $Q = D_d D_d^T$, $Y \geq 0$ é uma solução da equação algébrica de Riccati:

$$Y(A - B_d D_d^T Q^{-1} C)^T + (A - B_d D_d^T Q^{-1} C)Y - Y C^T Q^{-1} C Y + B_d(I - D_d^T Q^{-1} D_d)^2 B_d^T = 0 \tag{20}$$

cuja solução é encontrada através do Código 4:

Codigo 4 Resolução da Equação Algébrica de Riccati {arquivo: riccati_artigo.sci}

```
function [He,Ve]=riccati_artigo()
A=[0,1,0,0;6.52,24.77,33.82,35.56;0,0,0,1;-12.56,-49.71,-
56.25,-68.50];
B=[0;.2471;0;-.4758]; Bd=[0,0,0;0,0,-1;0,0,0;0,0,.5];
Bf=[0;.5;0;-.5]; C=[1,0,0,0;0,1,0,0];
Dd=[1,0,0;0,1,0]; D=[0;0]; Df=0;
E=[0;.1;0;.1]; E1=E; E2=E; F1=[.2,.1,.1,.1]; F2=.1;
Q=Dd*Dd';
T1=A-Bd*Dd'*inv(Q)*C;
T2=C'*inv(Q)*C;
T3=Bd*(eye(size(Dd',1),size(Dd,2))-
Dd'*inv(Q)*Dd)^2*Bd';
Y=riccati(T1,T2,T3,'c');
He=(Bd*Dd'+Y*C')*inv(Q);
Ve=Q^(-.5);
```

O sistema considerando as incertezas nos parâmetros passa a ser representado conforme o conjunto de equações 21, gerando um problema de otimização da equação 22 conforme [1]:

$$\begin{bmatrix} \dot{e}^T \\ \dot{x}_f^T \\ \dot{x}^T \end{bmatrix} = \begin{bmatrix} A-HC & 0 & 0 \\ 0 & A_0 & 0 \\ 0 & 0 & A \end{bmatrix} . \begin{bmatrix} e^T \\ x_f^T \\ x^T \end{bmatrix} + \begin{bmatrix} 0 & B_d - HD_d & B_f - HD_f \\ 0 & B_{0d} & B_{0f} \\ B & B_d & B_f \end{bmatrix} . \begin{bmatrix} u^T \\ f^T \\ d^T \end{bmatrix}$$

$$[r_e] = \begin{bmatrix} VC & -C_0 & 0 \end{bmatrix} . \begin{bmatrix} e^T \\ x_f^T \\ x^T \end{bmatrix} + \begin{bmatrix} 0 & VD_f - D_{of} & VD_d - D_{od} \end{bmatrix} . \begin{bmatrix} u^T \\ f^T \\ d^T \end{bmatrix} \tag{21}$$

$$\min(\gamma)$$

$$s.a: \begin{bmatrix} P_1A + A^TP_1 - Y_1C - C^TY_1 & 0 & 0 & 0 & P_1B_f - Y_1D_f & P_1B_d - Y_1D_d & C^TV^T & P_1E_1 & P_1E_2 \\ 0 & P_2A_0 + A_0^TP_2 & 0 & 0 & P_2B_{0f} & P_2B_{0d} & -C_0^T & 0 & 0 \\ 0 & 0 & P_3A + A^TP_3 + \varepsilon_1F_1^TF_1 & P_3B & P_3B_f & P_3B_d & 0 & P_3E_1 & P_3E_2 \\ 0 & 0 & * & -\gamma^2I + \varepsilon_2F_2^TF_2 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & 0 & -\gamma^2I & 0 & D_f^TV^T - D_{0f}^T & 0 & 0 \\ * & * & * & 0 & 0 & -\gamma^2I & D_d^TV^T - D_{od}^T & 0 & 0 \\ * & * & 0 & 0 & * & * & -I & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 & 0 & -\varepsilon_1I & 0 \\ * & 0 & * & 0 & 0 & 0 & 0 & 0 & -\varepsilon_2I \end{bmatrix} < 0 \tag{22}$$

O código que soluciona 22 é apresentado a seguir:

Codigo 5 Inicialização da LMI {arquivo: testaartigo.sci}

```
function [P1,P2,P3,Y1,V,epsilon1,epsilon2,gama2]=
        testaartigo(gama)
A=[0,1,0,0;6.52,24.77,33.82,35.56;0,0,0,1;-12.56,-49.71,-
56.25,-68.50];
B=[0;.2471;0;-.4758]; Bd=[0,0,0;0,0,-1;0,0,0;0,0,.5];
Bf=[0;.5;0;-.5]; B0f=Bf; C=[1,0,0,0;0,1,0,0]; C0=C;
Dd=[1,0,0;0,1,0]; D0d=Dd; D=[0;0];
Df=[0;0]; D0f=Df; E=[0;.1;0;.1]; E1=E; E2=E;
F1=[.2,.1,.1,.1];
F2=.1;
deltaA=E1*(eye(size(E1,2),size(F1,1)))*F1;
deltaB=E2*(eye(size(E2,2),size(F2,1)))*F2;
A0=A; B0=B; B0d=Bd; A=A+deltaA; B=B+deltaB;
[lC,cC]=size(C); [lE1t,cE1t]=size(E1');
[lE2t,cE2t]=size(E2');
[lBt,cBt]=size(B'); P1_init=zeros(A); P2_init=zeros(A);
P3_init=zeros(A);
Y1_init=zeros(size(A,2),size(C,1));
V_init=zeros(size(D0d,1),size(Dd,1)); epsilon1_init=[0];
epsilon2_init=[0]; gama2_init=[0];
LIST0=list(P1_init,P2_init,P3_init,Y1_init,V_init,
        epsilon1_init,epsilon2_init,gama2_init);
LISTF=lmisolver(LIST0,artigo_eval);
[P1,P2,P3,Y1,V,epsilon1,epsilon2,gama2]=LISTF(:)
```

Codigo 6 Formulação da LMI {arquivo: artigo_eval.sci}

```
function [LME,LMI,OBJ]=artigo_eval(XLIST)
[P1,P2,P3,Y1,V,epsilon1,epsilon2,gama2]=XLIST(:)
LME1=[P1-P1'];LME2=[P2-P2'];LME3=[P3-
P3'];LME4=[V-V'];
LME=list(LME1,LME2,LME3,LME4);
LMI1=[P1*A+A'*P1-Y1*C-
C'*Y1',zeros(size(P1,1),size(P2,2)),
zeros(size(P1,1),size(P3,1)),zeros(size(P1,1),size(B,2)),
P1*Bf-Y1*Df,P1*Bd-Y1*Dd,C'*V',P1*E1,
P1*E2; zeros(size(P2,1),size(P1,1)),P2*A0+A0'*P2,
zeros(size(P2,1),size(P3,1)),zeros(size(P2,1),size(B,2)),
P2*B0f,P2*B0d,-C0',
zeros(size(P2,1),size(E1,2)),zeros(size(P2,1),size(E2,2));
zeros(size(P3,1),size(P1,1)),zeros(size(P3,1),size(P2,2)),
P3*A+A'*P3+epsilon1*F1'*F1,P3*B,P3*Bf,P3*Bd,
zeros(size(P3,1),size(D0d',2)),P3*E1,P3*E2;
zeros(size(B,2),size(P1,1)),zeros(size(B,2),size(P2,2)),(P3*B)',
-gama2*eye(size(B,2),size(B,2))+epsilon2*F2'*F2,
zeros(size(F2',1),size(Df',1)),zeros(size(F2',1),size(Dd',1)),
zeros(size(F2',1),size(D0d',2)),zeros(size(F2',1),size(E1,2)),
zeros(size(F2',1),size(E2,2));(P1*Bf-Y1*Df)',(P2*B0f)',
(P3*Bf)',zeros(size(Df',1),size(B,2)),
-gama2*eye(size(Df',1),size(Df',1)),
zeros(size(Df',1),size(Dd',1)),Df'*V'-D0f',
zeros(size(Df',1),size(E1,2)),zeros(size(Df',1),size(E2,2));
(P1*Bd-Y1*Dd)',(P2*B0d)',(P3*Bd)',
zeros(size(Dd',1),size(B,2)),zeros(size(Dd',1),size(Df',1)),
-gama2*eye(size(Dd',1),size(Dd',1)),Dd'*V'-D0d',
zeros(size(Dd',1),size(E1,2)),zeros(size(Dd',1),size(E2,2));
(C'*V')',(-C0')',zeros(size(D0d',2),size(P3,1)),
zeros(size(D0d',2),size(B,2)),(Df'*V'-D0f')',
(Dd'*V'-D0d')',-eye(size(D0d',2),size(D0d',2)),
zeros(size(D0d',2),size(E1,2)),zeros(size(D0d',2),size(E2,2));
(P1*E1)',zeros(size(E1,2),size(P2,2)),(P3*E1)',
zeros(size(E1,2),size(B,2)),zeros(size(E1,2),size(Df',1)),
zeros(size(E1,2),size(Dd',1)),zeros(size(E1,2),size(D0d',2)),
-epsilon1*eye(size(E1,2),size(E1,2)),
zeros(size(E1,2),size(E2,2));P1*E2)',
zeros(size(E2,2),size(P2,2)),(P3*E2)',
zeros(size(E2,2),size(B,2)),zeros(size(E2,2),size(Df',1)),
zeros(size(E2,2),size(Dd',1)),zeros(size(E2,2),size(D0d',2)),
zeros(size(E2,2),size(E1,2)),
-epsilon2*eye(size(E2,2),size(E2,2))];
LMI=list(-LMI1,epsilon1,epsilon2);
//LMI=list(-LMI1,epsilon1,epsilon2,gama2-.75^2);//{Caso
queira limitar γ como no artigo}
OBJ=[gama2];
```

REFERNCIAS

[1] Zhong, M. , Ding, S.X. ,Lam , J. and Wang, H. An lmi approach to design robust fault detection filter for uncertain lti systems. *Automatica*, 39:543–550, november 2003.

[2] Nash, S. G., Sofer, A. *Linear and Nonlinear Programming*. McGraw-Hill, 1996.

[3] Nesterov,Y. and Nemirovskii,A. *Interior-Point Polynomial Algorithms in Convex Programming*. Siam, Philadelphia, 1994.

[4] Vandenberghe,L. and Boyd,S. *Positive-Definite Programming*. Mathematical Programming, Michigan, 1994.

[5] Vandenberghe,L. and Boyd,S. *Semidefinite Programming, 38(1):49-95*. SIAM Review, Maro, 1996.

[6] Nemirovskii, A. e Gahinet, P. The projective method for solving linear matrix inequalities. *Proceedings of the American Control Conference*, pages 840–844, Junho 1994.

[7] Vandenberghe,L. and Boyd,S. *A primal-dual potencial reduction method for problems involving matrix inequalities - Mathematical Programming Series B.* Stanford, Stanford, 1994.