Example 1 (from Duan & Yu : LMIs in control systems)

problem

4.5 Time-Delay Systems

The problem of stability analysis for a time-delay system can be stated as follows.

Problem 4.3 Given matrices $A, A_d \in \mathbb{R}^{n\times n}$, check the stability of the following linear time-delay system

$$\begin{cases} \dot{x}(t) = Ax(t) + A_d x(t-d) \\ x(t) = \phi(t),\ t \in [-d, 0],\ 0 < d \le \bar{d}, \end{cases} \tag{4.47}$$

where
$\phi(t)$ is the initial condition
d represents the time-delay
$\bar{d}$ is a known upper bound of d

4.5.1 *Delay-Independent Condition*

The following theorem gives a sufficient condition for the stability problem in terms of an LMI.

Theorem 4.8 The system (4.47) is asymptotically stable if there exist two symmetric matrices $P, S \in \mathbb{S}^n$, such that

$$\begin{cases} P > 0 \\ \begin{bmatrix} A^{\mathrm{T}}P + PA + S & PA_d \\ A_d^{\mathrm{T}}P & -S \end{bmatrix} < 0. \end{cases} \tag{4.48}$$

Example 4.15

The following time-delay linear system has been considered in Fu et al. (2004):

$$\dot{x}(t) = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t-d(t)) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t) + \begin{bmatrix} 1 \\ 2 \end{bmatrix} w(t).$$

Proving the stability using LMI-optimization

<table>
<tr><th>Matlab code</th><th>Code output</th></tr>
<tr><td><pre>
%% BEFORE RUNNING THE CODE ADD "YALMIP" AND
"SDPT3" libraries
%% yalmip param robust ness anlaysis
clear all,close all,clc;yalmip('clear');

A=diag([-1,-2]);
Ad=eye(2);

nx=2;

eps1=1e-5;
% P=sdpvar(nx,nx,'symmetric');
% S=sdpvar(nx,nx,'symmetric');
P=sdpvar(nx,nx,'diagonal');
S=sdpvar(nx,nx,'diagonal');

F=[];
F=[F;P>=eps1*eye(nx)];
F=[F;[[P*A]+A'*P+S,[P*Ad];Ad'*P,-
S]<=0*eye(2*nx)];
F=[F;0<=vec(P)<=10000];
F=[F;0<=vec(S)<=10000];

ops = sdpsettings('solver','sdpt3');
sol = optimize(F,[],ops);
sol.info

P0=value(P)
eig(P0)
S0=value(S)
eig(S0)
max(eig([[P0*A]+[P0*A]'+S0,[P0*Ad];[P0*Ad]',-
S0]))
</pre></td><td><pre>
>> P0

P0 =

   1.0e+03 *

    2.2427   0
         0   4.6712

>> S0

S0 =

   1.0e+03 *

    2.2427   0
         0   4.8618
</pre></td></tr>
</table>

Simulating the system using dde23 Function Phase Plane

Matlab code

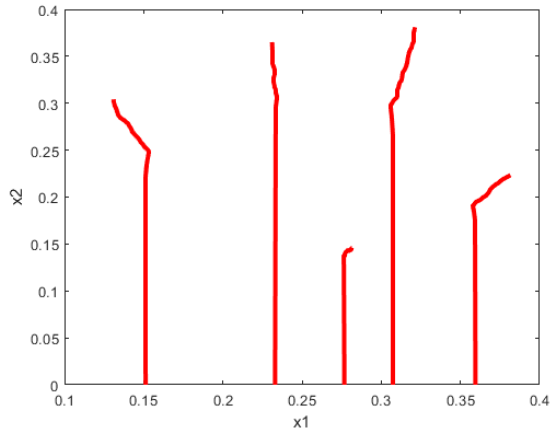
```
%% matlab delayed-diff system simulation
clear all,close all,clc;yalmip('clear');

lags=[0.2];
t_vec=[0:1e-4:20]';
fig1=figure(1); fig1.Color=[1,1,1];
for ii=1:1:5
    sol1=dde23(@dde_func,lags,@x_history,t_vec);
    x_vec = deval(sol1,t_vec);
    x1_trajectory=x_vec(1,:);
    x2_trajectory=x_vec(2,:);
    plot(x1_trajectory,x2_trajectory,...
        'LineStyle','-',...
        'LineWidth',[3],...
        'Color','r'); hold on;
xlabel('x1'); ylabel('x2');
%     plot(t_vec,x1_trajectory,'LineStyle','-
','LineWidth',[3],'Color','r'); hold on;
%     plot(t_vec,x2_trajectory,'LineStyle','-
','LineWidth',[3],'Color','b');
%     legend('x1','x2');
end

function xdot=dde_func(t,x,x_delayed)
x1=x(1);
x2=x(2);
xdot=zeros(2,1);
x1_delayed=x_delayed(1);
x2_delayed=x_delayed(2);
    A=diag([-1,-2]);
    Ad=eye(2);
    xdot=A*[x1;x2]+Ad*[x1_delayed;x2_delayed];
end

function x=x_history(t)
%     x=ones(2,1);
    x=0.1*ones(2,1)+rand(2,1)*0.3;
end
```

Code output



Simulating the system using dde23 Function State Signals

<table>
<tr><th>Matlab code</th><th>Code output</th></tr>
<tr><td>

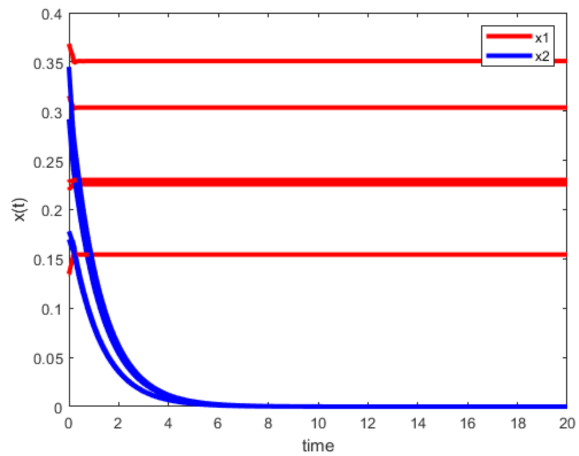
```
%% matlab delayed-diff system simulation
clear all,close all,clc;yalmip('clear');

lags=[0.2];
t_vec=[0:1e-4:20]';
fig1=figure(1); fig1.Color=[1,1,1];
for ii=1:1:5
    sol1=dde23(@dde_func,lags,@x_history,t_vec);
    x_vec = deval(sol1,t_vec);
    x1_trajectory=x_vec(1,:);
    x2_trajectory=x_vec(2,:);
%     plot(x1_trajectory,x2_trajectory,...
%         'LineStyle','-',...
%         'LineWidth',[3],...
%         'Color','r'); hold on; xlabel('x1');
ylabel('x2');
    plot(t_vec,x1_trajectory,'LineStyle','-
','LineWidth',[3],'Color','r'); hold on;
    plot(t_vec,x2_trajectory,'LineStyle','-
','LineWidth',[3],'Color','b');
    legend('x1','x2'); xlabel('time');
ylabel('x(t)');
end

function xdot=dde_func(t,x,x_delayed)
x1=x(1);
x2=x(2);
xdot=zeros(2,1);
x1_delayed=x_delayed(1);
x2_delayed=x_delayed(2);
    A=diag([-1,-2]);
    Ad=eye(2);
    xdot=A*[x1;x2]+Ad*[x1_delayed;x2_delayed];
end

function x=x_history(t)
%     x=ones(2,1);
    x=0.1*ones(2,1)+rand(2,1)*0.3;
end
```

</td><td>



</td></tr>
</table>

Example 2 (from Duan & Yu : LMIs in control systems)

problem

4.5 Time-Delay Systems

The problem of stability analysis for a time-delay system can be stated as follows.

Problem 4.3 Given matrices $A, A_d \in \mathbb{R}^{n\times n}$, check the stability of the following linear time-delay system

$$\begin{cases} \dot{x}(t) = Ax(t) + A_d x(t-d) \\ x(t) = \phi(t),\ t \in [-d, 0],\ 0 < d \leq \bar{d}, \end{cases} \tag{4.47}$$

where
  $\phi(t)$ is the initial condition
  d represents the time-delay
  $\bar{d}$ is a known upper bound of d

$$\Phi(X) = X(A + A_d)^{\mathrm{T}} + (A + A_d)X + \bar{d}A_d A_d^{\mathrm{T}}.$$

Theorem 4.9 The time-delay system (4.47) is uniformly asymptotically stable if there exist a symmetric positive definite matrix X and a scalar $0 < \beta < 1$, such that

$$\begin{bmatrix} \Phi(X) & \bar{d}XA^{\mathrm{T}} & \bar{d}XA_d^{\mathrm{T}} \\ \bar{d}AX & -\bar{d}\beta I & 0 \\ \bar{d}A_d X & 0 & -\bar{d}(1-\beta)I \end{bmatrix} < 0, \tag{4.52}$$

Example 4.16

Consider the following time-delay linear system:

$$\dot{x}(t) = \begin{bmatrix} -2 & 0 & 1 \\ 0 & -3 & 0 \\ 1 & 0 & -2 \end{bmatrix} x(t) + \begin{bmatrix} -1 & 1 & 1 \\ 2 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} x(t-d). \tag{4.50}$$

Proving the stability using LMI-optimization

<table>
<tr><th>Matlab code</th><th>Code output</th></tr>
<tr><td>
<pre>
%% BEFORE RUNNING THE CODE ADD "YALMIP" AND
"SDPT3" libraries
%% yalmip analysis
%
set(findall(gcf,'type','line'),'linewidth',[1]);
clear all,close all,clc;yalmip('clear');
A=[-2,0,1;0,-3,0;1,0,-2];
Ad=[-1,1,1;2,-1,1;0,0,-1];

nx=3;   % state-dimension
dmax=0.3;   % max delay [max value 0.3
seconds]
eps1=1e-3;   % epsilon for numerical accuracy
% define the decision variables
X=sdpvar(nx,nx,'symmetric');
beta=sdpvar(1,1,'symmetric');
% enter the constraints
F=[];
F=[F;X>=eps1*eye(nx)];
M_11=[(A+Ad)*X]+[(A+Ad)*X]'+dmax*Ad*Ad';
M_12=dmax*X*A';
M_13=dmax*X*Ad';
M_21=dmax*A*X;
M_22=-dmax*beta*eye(nx);
M_23=zeros(nx);
M_31=dmax*Ad*X;
M_32=zeros(nx);
M_33=-dmax*(1-beta)*eye(nx);
M=[M_11,M_12,M_13;M_21,M_22,M_23;M_31,M_32,M_33];
F=[F;M<=-eps1*eye(3*nx)];
F=[F;-1e2<=vec(X)<=1e2];
F=[F;eps1<=beta<=1-eps1];
% solve the opt-problem
ops = sdpsettings('solver','sdpt3');
sol = optimize(F,[],ops);
sol.info
% get the decision variables
X=value(X);P=inv(X);beta=value(beta)
</pre>
</td><td>
<pre>
ans =

    'Successfully solved (SDPT3-4)'

X =

    0.6726   0.0096   -0.2245
    0.0096   0.8211   -0.0202
   -0.2245   -0.0202   0.5847

P =

    1.7051   -0.0039   0.6544
   -0.0039   1.2189   0.0407
    0.6544   0.0407   1.9627

beta =

    0.5931
</pre>
</td></tr>
</table>

Additional check

Matlab code	Code output

`%% check if there is any error`
`X=value(X)`
`P=inv(X)`
`beta=value(beta)`
 	
`eig(P)`
`M_11=[(A+Ad)*X]+[(A+Ad)*X]'+dmax*Ad*Ad';`
`M_12=dmax*X*A';`
`M_13=dmax*X*Ad';`
 	
`M_21=dmax*A*X;`
`M_22=-dmax*beta*eye(nx);`
`M_23=zeros(nx);`
 	
`M_31=dmax*Ad*X;`
`M_32=zeros(nx);`
`M_33=-dmax*(1-beta)*eye(nx);`
`M=[M_11,M_12,M_13;M_21,M_22,M_23;M_31,M_32,M_33];`
`eig(P)`
`eig(M)`
	`>> eig(P)`
	
	`ans =`
	
	`1.1540`
	`1.2312`
	`2.5015`
	
	`>> eig(M)`
	
	`ans =`
	
	`-7.2064`
	`-4.7101`
	`-0.8968`
	`-0.0069`
	`-0.0220`
	`-0.0613`
	`-0.1716`
	`-0.1226`
	`-0.1453`
	
	All eigenvalues of P are positive therefore it is a pos-def-matrix
	
	All eigenvalues of M are negative therefore it is a neg-def-matrix

Simulating the system using dde23 Function Phase Plane

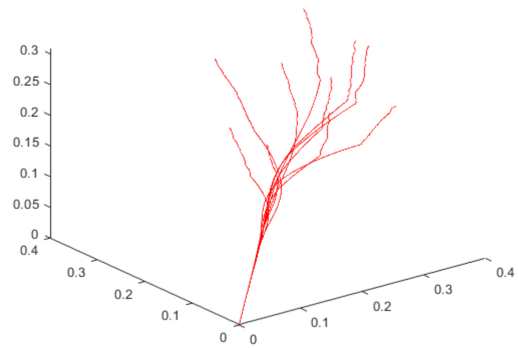
Matlab code

```
%% matlab delayed-diff system simulation
clear all,close all,clc;yalmip('clear');

lags=[0.2];
t_vec=[0:1e-4:20]';
fig1=figure(1);
fig1.Color=[1,1,1];
for ii=1:1:10
    sol1=dde23(@dde_func,lags,@x_history,t_vec);
    x_vec = deval(sol1,t_vec);
    x1_trajectory=x_vec(1,:);
    x2_trajectory=x_vec(2,:);
    x3_trajectory=x_vec(3,:);
    plot3(x1_trajectory,x2_trajectory,x3_trajectory,...
        'LineStyle','-',...
        'LineWidth',[1],...
        'Color','r'); hold on;
%     plot(t_vec,x1_trajectory,'LineStyle','-
','LineWidth',[1],'Color','r'); hold on;
%     plot(t_vec,x2_trajectory,'LineStyle','-
','LineWidth',[1],'Color','g');
%     plot(t_vec,x3_trajectory,'LineStyle','-
','LineWidth',[1],'Color','b');
%     legend('x1','x2','x3');
end
function xdot=dde_func(t,x,x_delayed)
x1=x(1);
x2=x(2);
x3=x(3);
xdot=zeros(3,1);
x1_delayed=x_delayed(1);
x2_delayed=x_delayed(2);
x3_delayed=x_delayed(3);
A=[-2,0,1;0,-3,0;1,0,-2];
Ad=[-1,1,1;2,-1,1;0,0,-1];

xdot=A*[x1;x2;x3]+Ad*[x1_delayed;x2_delayed;x3_delayed];
end
function x=x_history(t)
%     x=ones(2,1);
    x=0.1*ones(3,1)+rand(3,1)*0.3;
end
```

Code output



Notice that all the states converge to the origin regardless of the initial conditions.

Simulating the system using dde23 Function State Signals

Matlab code

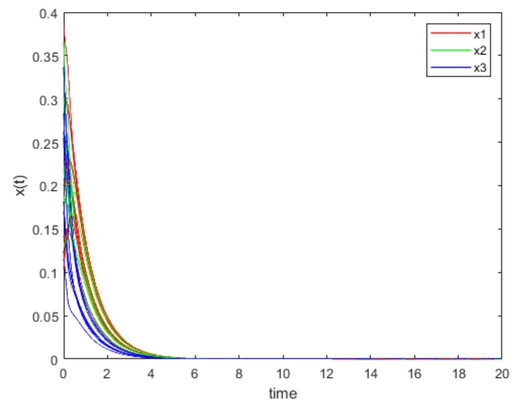
```
%% matlab delayed-diff system simulation
clear all,close all,clc;yalmip('clear');

lags=[0.2];
t_vec=[0:1e-4:20]';
fig1=figure(1);
fig1.Color=[1,1,1];
for ii=1:1:10
    sol1=dde23(@dde_func,lags,@x_history,t_vec);
    x_vec = deval(sol1,t_vec);
    x1_trajectory=x_vec(1,:);
    x2_trajectory=x_vec(2,:);
    x3_trajectory=x_vec(3,:);
%
plot3(x1_trajectory,x2_trajectory,x3_trajectory,...
%          'LineStyle','-',...
%          'LineWidth',[1],...
%          'Color','r'); hold on;
    plot(t_vec,x1_trajectory,'LineStyle','-
','LineWidth',[1],'Color','r'); hold on;
    plot(t_vec,x2_trajectory,'LineStyle','-
','LineWidth',[1],'Color','g');
    plot(t_vec,x3_trajectory,'LineStyle','-
','LineWidth',[1],'Color','b');

legend('x1','x2','x3');xlabel('time');ylabel('x(t)');
end
function xdot=dde_func(t,x,x_delayed)
x1=x(1);
x2=x(2);
x3=x(3);
xdot=zeros(3,1);
x1_delayed=x_delayed(1);
x2_delayed=x_delayed(2);
x3_delayed=x_delayed(3);
A=[-2,0,1;0,-3,0;1,0,-2];
Ad=[-1,1,1;2,-1,1;0,0,-1];

xdot=A*[x1;x2;x3]+Ad*[x1_delayed;x2_delayed;x3_delayed]
;
end
function x=x_history(t)
%     x=ones(2,1);
    x=0.1*ones(3,1)+rand(3,1)*0.3;
end
```

Code output



Notice that all the states converge to the origin regardless of the initial conditions.