

$\mathcal{H}_\infty$ -control in Matlab

1 Introduction

Typically the concept of $\mathcal{H}_\infty$ controller design is fairly easy to grasp. However, as controller synthesis is done numerically, a major problem for people new to the subject is *how to write the Matlab code*. We will here try to give a short overview of some useful Matlab functions. Hopefully this will help you when trying to design your first $\mathcal{H}_\infty$ -controller.

There are *many* $\mathcal{H}_\infty$ related functions available in Matlab and its toolboxes. The important toolbox is, in addition to the Control System Toolbox, the Robust Control Toolbox (RCT).

We have also prepared an m-file where we have tried to use as many of the functions discussed here as possible. The m-file is included in the appendix and can also be downloaded from Ilias.

A mixed S/KS synthesis problem will be used to illustrate the use of a handful of useful functions. Let's take a look at this problem first.

2 Shaping closed loop transfer functions

The mixed S/KS problem can be illustrated with the block diagram shown in Figure 1. The closed loop transfer function $T = F_l(P, K)$ from w to z can be found by visual inspection as

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} W_s S \\ W_{ks} K S \end{bmatrix} r. \quad (1)$$

The generalized plant $P(s)$ (see Figure 2) is

$$\begin{bmatrix} z_1 \\ z_2 \\ e \end{bmatrix} = \begin{bmatrix} W_s & -W_s G \\ 0 & W_{ks} \\ I & -G \end{bmatrix} \begin{bmatrix} r \\ u \end{bmatrix}. \quad (2)$$

If we have the state space realizations

$$G \stackrel{s}{=} \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right], \quad W_s \stackrel{s}{=} \left[\begin{array}{c|c} A_s & B_s \\ \hline C_s & D_s \end{array} \right], \quad W_{ks} \stackrel{s}{=} \left[\begin{array}{c|c} A_{ks} & B_{ks} \\ \hline C_{ks} & D_{ks} \end{array} \right],$$

it can be shown that a possible state space realization for $P(s)$ is given by

$$P \stackrel{s}{=} \left[\begin{array}{ccc|cc} A_s & 0 & -B_s C & B_s & -B_s D \\ 0 & A_{ks} & 0 & 0 & B_{ks} \\ 0 & 0 & A & 0 & B \\ \hline C_s & 0 & -D_s C & D_s & -D_s D \\ 0 & C_{ks} & 0 & 0 & D_{ks} \\ 0 & 0 & -C & I & -D \end{array} \right]. \quad (3)$$

(We leave this as an exercise for you.)

The weights W_s and W_{ks} are your tuning parameters, and it typically requires some iterations to obtain weights which will yield a good controller. That being said, a good starting point is to choose

$$W_s = \frac{s/M + \omega_0}{s + \omega_0 A}; \quad W_{ks} = \text{const.} \quad (4)$$

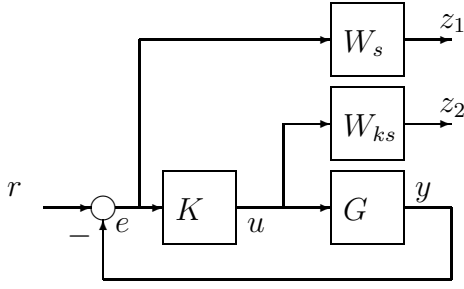


Figure 1: Mixed S/KS problem.

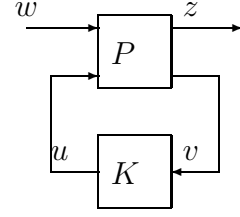


Figure 2: General control problem.

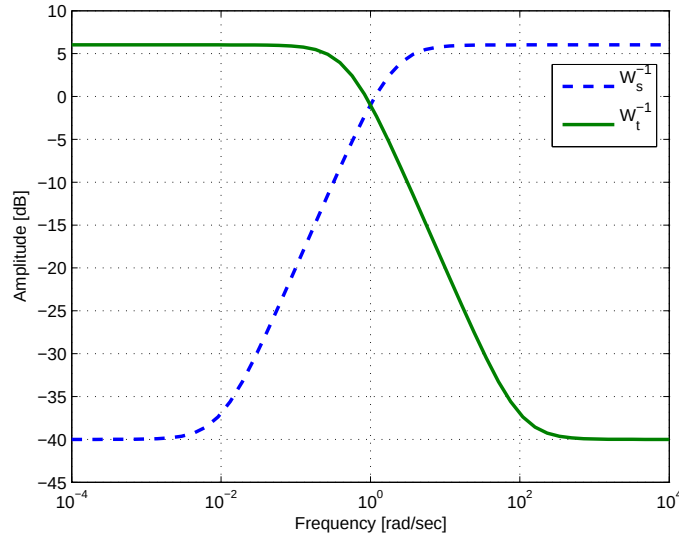


Figure 3: Inverse of weighting functions W_s and W_t

where $A < 1$ is the maximal allowed steady state offset, w_0 is the desired bandwidth and M is the sensitivity peak (typically $A = 0.01$ and $M = 2$). For the controller synthesis, the inverse of W_s is an upper bound on the desired sensitivity loop shape, and W_{ks}^{-1} will effectively limit the controller output u .

In some cases, you would also like to shape the complementary sensitivity function $T = GK(I + GK)^{-1}$ (done by adding an extra output $z_3 = W_t y$ in Figure 1). A starting point is to choose

$$W_t = \frac{s + \omega_0/M}{As + \omega_0}, \quad (5)$$

which is symmetric to W_s around the line $\omega = \omega_0$. The two weighting functions are shown in Figure for the parameter values $A = 0.01 (= -40\text{dB})$, $M = 2 (= 6\text{dB})$ and $\omega_0 = 1 \text{ rad/sec}$.

3 Obtaining the subsystems

There are several ways to obtain the dynamical systems G , W_s and W_{ks} in Matlab. Methods you probably already have heard about are `ss`, `tf` and `zpk` in Control System Toolbox. RCT offer a variety of similar possibilities like `pck`, `nd2sys` and `zp2sys`. Other methods are `mksys` and `tree`. Which one to choose is a matter of convenience, you can transfer back and forth between the different representations quite easily. One possibility is to write `[A,B,C,D]=ssdata(Gcst); Gmu=pck(A,B,C,D)`. The opposite

way would be `[A,B,C,D]=unpck(Gmu); Gcst = ss(A,B,C,D)`. Take a look at the documentation to see other options.

4 Obtaining the generalized plant P

Also in creating P you have many options. We list five:

- a) Write down the transfer function matrix in (2) directly. If you afterwards convert to state-space, you should use e.g. `minreal` to obtain a minimal realization. Useful commands: `sbs` (side-by-side), `abv` (above), `mmult` (multiply), `minv` (inverse).
- b) Write down the state space matrices A,B,C,D in (3) and use `P = pck(A,B,C,D)`.
- c) Use `sysic` (system interconnect), where you specify your subsystems and the interconnection between them.
- d) Use `sconnect`, a function where subsystems, inputs and outputs are passed as parameters, and `sconnct` returns the connected system.
- e) Use `iconnect`, functionally similar to `sysic`.

Of these methods we personally prefer `sysic` and `iconnect` because they are flexible and easy to use also for more complex systems where method 1 and 2 are no longer feasible.

Generally it is a good idea to use a balanced realization to avoid numerical problems. A balanced realization can be obtained e.g. with `balreal` in Control System Toolbox.

5 Synthesizing controller

The $\mathcal{H}_\infty$ S/KS synthesis problem is to find a controller K which stabilizes G and minimizes the $\mathcal{H}_\infty$ cost function

$$\|F_l(P, K)\|_\infty = \left\| \begin{bmatrix} W_s S \\ W_{ks} K S \end{bmatrix} \right\|_\infty.$$

We guess by now you are not surprised to hear that there are several methods available to synthesize $\mathcal{H}_\infty$ controllers. Typically you would use `hinfsyn`, `hinfri` or `hinflmi` which all have P in the system representation as an input. In RCT, there is the function `mixsyn` with G , W_s , W_{ks} (and W_t , a weight for the complementary sensitivity function) as inputs, that is, you do not need the generalized plant P at all. The main difference between the methods is whether they use Riccati equations and gamma-iteration or linear matrix equalities to solve the optimization problem. The LMI approach does not require all of the technical assumptions needed when using Riccati equation based solvers.

There are a variety of other commands like `ncfsyn` and `loopsyn` (for $\mathcal{H}_\infty$ loop shaping of the open loop transfer function $L = GK$), `hinfmix` and `msfsyn` (multi-objective). Check out the manual.

6 Analysing the results

After the controller has been synthesized, it is time to analyse the results. This can be done using Control System Toolbox commands like `lsim`, `step` (step response), `bode` (bode plot), `sigma` (singular value plot) and `freqresp` (frequency response) on typical transfer matrices like S , KS , T , K and GK .

7 Conclusions

As you have seen, there are many options. If you know what you want to do but cannot remember the command, the functions by category part of the matlab manual is a good reference.

Hopefully this short introduction to Matlab and $\mathcal{H}_\infty$ will make it a little easier for you to synthesize your first $\mathcal{H}_\infty$ controller, good luck!

A Matlab code illustrating the commands

```
%This script is to illustrate how one can design H-infinity controllers in
%Matlab. The example shows some different ways to synthesize a controller
%for the mixed S/KS weighted sensitivity problem. The plant and the weights
%were found in (Skogestad and Postlethwaite, 1996 ed.1 p.60) and the
%weights are not necessarily the "best".
%
%Most of the commands are available in mu-tools, some are from the
%lmi-toolbox. Both mu-tools and lmi-tools are included in the robust
%control toolbox v.3.0.1 in matlab 7 and higher
%-Jorgen Johnsen 14.12.06

%-----
%Defining the subsystems
%-----
%Plant G=200/((10s+1)(0.05s+1)^2)
%Alternative 1, mu-tools directly:
G = nd2sys(1,conv([10,1],conv([0.05 1],[0.05 1])),200);
%Alternative 2, indirectly via cst:
s = tf('s');
Gcst = 200/((10*s+1)*(0.05*s+1)^2);
[a,b,c,d] = ssdata( balreal(Gcst) );
G = pck(a,b,c,d);

%Weights Ws = (s/M+w0)/(s+w0*A), Wks=1
M = 1.5; w0 = 10; A=1.e-4;
Ws = nd2sys([1/M w0],[1 w0*A]);
Wks = 1;

%-----
%Creating the generalized plant P
%-----
%Alternative 0, direct approach:
% /z1\   /Ws -Ws*G\ /r\
% |z2| = |0   Wks | | |
% \ v/   \I  -G   / \u/

%Transfer matrix representation
Z1 = sbs(Ws,mmult(-1,Ws,G));
Z2 = sbs(0,Wks);
V = sbs(1,mmult(-1,G));
P0 = abv(Z1,Z2,V);

%P0 is generally not a minimal realization, so we use available reduction methods
[a,b,c,d] = unpck(P0);
[ab,bb,cb,db] = ssdata( balreal( minreal( ss(a,b,c,d) ) ) );
P0 = pck(ab,bb,cb,db); %now we have a System description

%-----
%Creating the generalized plant P
%-----
%Alternative 1, direct approach:
% /z1\   /W1 -W1*G\ /r\
```

```

% |z2| = |0   W2 | | |
% \ v/  \I  -G  / \u/

%ss realization of the subsystems:
[A,B,C,D]      = unpck(G);
[A1,B1,C1,D1] = unpck(Ws);
[A2,B2,C2,D2] = unpck(Wks);

%number of inputs and outputs to the different subsystems:
n1 = size(A1,1); [q1, p1] = size(D1);
n2 = size(A2,1); [q2, p2] = size(D2);
n  = size(A,1) ; [p , q ] = size(D);

%ss realization of the whole thing:
Ap = [   A1      , zeros(n1,n2) ,   -B1*C      ;
      zeros(n2,n1) ,   A2      , zeros(n2,n) ;
      zeros(n ,n1) , zeros(n ,n2) ,   A      ];

Bp = [   B1      , -B1*D;
      zeros(n2,p) , B2  ;
      zeros(n ,p) , B  ];

Cp = [   C1      , zeros(q1,n2) ,   -D1*C      ;
      zeros(q2,n1) ,   C2      , zeros(q2,n) ;
      zeros(q ,n1) , zeros(q ,n2) ,   -C      ];

Dp = [   D1      , -D1*D;
      zeros(q2,p ) , D2  ;
      eye(p)      , -D  ];

%making a balanced realization reduces the likelihood of numerical problems
[Apb,Bpb,Cpb,Dpb] = ssdata( balreal( ss(Ap,Bp,Cp,Dp) ) );
P1 = pck(Apb,Bpb,Cpb,Dpb);

%-----
%Creating the generalized plant P
%-----
%Alternative 2, using sysic:
systemnames = 'G Ws Wks';
inputvar    = '[r(1); u(1)]'; %all inputs are scalar, r(2) would be a 2dim signal
outputvar    = '[Ws; Wks; r-G]';
input_to_G   = '[u]';
input_to_Ws  = '[r-G]';
input_to_Wks = '[u]';
sysoutname   = 'P2';
cleanupsysic = 'yes';
sysic

%-----
%Creating the generalized plant P
%-----
%Alternative 3, using sconnect:
inputs = 'r(1); u(1)';
outputs = 'Ws; Wks; e=r-G';
K_in   = []; %no controller present
G_in   = 'G:u';
Ws_in  = 'Ws:e';
Wks_in = 'Wks:u';
[P3,r] = sconnect(inputs,outputs,K_in,G_in,G,Ws_in,Ws,Wks_in,Wks);

%-----
%Creating the generalized plant P
%-----

```

```

%Alternative 4, using iconnect:
r = icsignal(1);
u = icsignal(1);
ws = icsignal(1);
wks = icsignal(1);
e = icsignal(1);
y = icsignal(1);
M = iconnect;
M.Input = [r;u];
M.Output = [ws;wks;e];
M.Equation{1} = equate(e,r-y);
M.Equation{2} = equate(y,ss(A,B,C,D)*u);
M.Equation{3} = equate(ws,ss(A1,B1,C1,D1)*e);
M.Equation{4} = equate(wks,ss(A2,B2,C2,D2)*u);
[ab,bb,cb,db] = ssdata( balreal(M.System) );
P4 = pck(ab,bb,cb,db);

%-----
%Synthesizing the controller
%-----
%All the methods presented here use the System
%matrix representation of the generalized plant P
%Choose your favourite method and your favourite P

%Choose plant
P = P1; %(0-4)

%then some parameters (number of measurements and inputs, and bounds on gamma )
nmeas = 1; nu = 1; gmn=0.5; gmx=20; tol = 0.001;

%uncomment your favourite controller
[K,CL,gopt] = hinfsyn(P,nmeas,nu,gmn,gmx,tol);
[gopt,K] = hinflmi(P,[nmeas, nu],0,tol); CL = starp(P,K,nmeas,nu);
[gopt,K] = hinfric(P,[nmeas, nu],gmn,gmx); CL = starp(P,K,nmeas,nu);

%Alternative for RCT v3.0.1.
%Normally you would of course not do all the transfers between the system
%representations, but rather do everything using standard ss objects
[a,b,c,d] = unpck(G); Gcst = ss(a,b,c,d);
[a,b,c,d] = unpck(Ws); Wscst = ss(a,b,c,d);
[a,b,c,d] = unpck(Wks); Wkscst = ss(a,b,c,d);
[K,CL,gopt] = mixsyn(Gcst,Wscst,Wkscst,[]);
[a,b,c,d] = ssdata( balreal(K) ); K = pck(a,b,c,d);
[a,b,c,d] = ssdata( balreal(CL) ); CL = pck(a,b,c,d);

%-----
%Analysis of the result
%-----
%Note: mu-tools commands are used here. All of this can be done
%using control toolbox commands instead, e.g. series and
%feedback for interconnecting elements, sigma or freqresp, svd and bode
%for singular value plots, and step or lsim for time responses

%plot singular values of (weighted) closed loop system
w = logspace(-4,6,50);
CLw = vsvd(frsp(CL,w));
figure(1); vplot('liv,m',CLw);
title('singular values of weighted closed loop system');

%generate typical transfer matrices
[type,out,in,n] = minfo(G);
I = eye(out);

```

```

S = minv(madd(I,mmult(G,K))); %sensitivity
T = msub(I,S); %complementary sensitivity
KS = mmult(K,S); %input to G
GK = mmult(G,K); %loop transfer function

%singular values as a function of frequency
Sw = vsvd(frsp(S,w));
Tw = vsvd(frsp(T,w));
Kw = vsvd(frsp(K,w));
KSw = vsvd(frsp(KS,w));
GKw = vsvd(frsp(GK,w));

%Plot singular value plots
%Note: if desired, you can change vplot to plot the amplitude in dB. Type
%edit vplot and uncomment the appropriate lines in the code
figure(2); vplot('liv,lm',Sw,'-',Tw,'--',GKw,'-.');
title('\sigma(S(jw)) (solid), \sigma(T(jw)) (dashed) and \sigma(GK(jw)) (dashdot)');
xlabel('Frequency [rad/sec]'); ylabel('Amplitude')
figure(3); vplot('liv,lm',Kw);
title('\sigma(K(jw))');
xlabel('Frequency [rad/sec]'); ylabel('Amplitude')

%Did we get what we asked for?
Sd = minv(Ws); Sdw = vsvd(frsp(Sd,w)); %"desired" sensitivity
KSd = minv(Wks); KSdw = vsvd(frsp(KSd,w)); %"desired" output
figure(4); vplot('liv,lm',Sw,'-',Sdw,'--');
title('\sigma(S(jw)) (solid) and \sigma(Ws^{-1}(jw)) (dashed)');
xlabel('Frequency [rad/sec]'); ylabel('Amplitude')
figure(5); vplot('liv,lm',KSw,'-',KSdw,'--');
title('\sigma(KS(jw)) (solid) and \sigma(Wks^{-1}(jw)) (dashed)');
xlabel('Frequency [rad/sec]'); ylabel('Amplitude')

%Finally the step response
reference = 1; tfinal = 1; step = 0.01;
y = trsp(T,reference,tfinal,step);
u = trsp(KS,reference,tfinal,step);
figure(6); subplot(2,1,1); vplot('iv,d',y);
title('Step response'); ylabel('y');
subplot(2,1,2); vplot('iv,d',u);
ylabel('u'); xlabel('time');

```