Polynomial solution of the SISO mixed sensitivity *H*-infinity problem

Synopsis

Mixed sensitivity optimization is a powerful design tool for linear single-degree-of-freedom feedback systems. It allows simultaneous design for performance and robustness, and relies on shaping the two critical closed-loop sensitivity functions with frequency dependent weights.

To obtain satisfactory high-frequency roll-off nonproper weighting functions may be needed. These cannot be directly handled in the conventional state space solution of the *H*-infinity problem. Nonproper weighting functions present no problems in the frequency domain solution of the *H*-infinity problem. The frequency domain solution may be implemented in terms of polynomial matrix manipulations.

We present the mixed sensitivity problem and its solution for single-input-single-output plants. Step by step the Toolbox function `mixeds` is developed that implements the algorithm. A simple but relevant example is used for illustration.

Introduction

To demonstrate the capabilities of the Polynomial Toolbox we implement the polynomial solution of a mixed sensitivity *H*-infinity problem. Consider the single-degree-of-freedom feedback loop of Fig. 1. *P* is the plant transfer matrix and *C* the compensator transfer matrix.

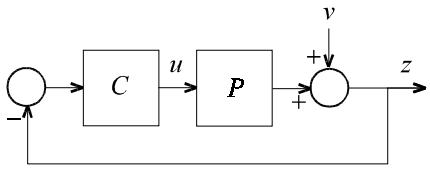

Fig. 1. Single-degree-of-freedom feedback loop

The sensitivity matrix *S* and input sensitivity matrix *U* of the feedback system are defined as

$$S = (I + PC)^{-1}, \quad U = C(I + PC)^{-1}$$

The mixed sensitivity problem is the problem of minimizing the infinity norm $\|H\|_\infty$ of

$$H = \begin{bmatrix} W_1 SV \\ W_2 UV \end{bmatrix}$$

with suitably chosen weighting matrices W_1 and W_2, and V a suitably chosen shaping matrix. In the SISO case the infinity norm is given by

$$\|H\|_\infty^2 = \sup_{-\infty<\omega<\infty} \left(|W_1(j\omega)S(j\omega)V(j\omega)|^2 + |W_1(j\omega)U(j\omega)V(j\omega)|^2 \right)$$

The problem may be reduced to a "standard" H-infinity problem by considering the block diagram of Fig. 2, which includes the weighting and shaping filters.

The diagram of Fig. 2 defines a standard problem whose "generalized plant" has the transfer matrix

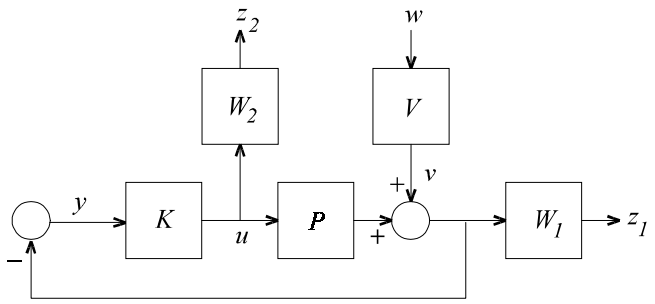

Fig. 2. Mixed sensitivity configuration

$$G = \left[\begin{array}{c|c} W_1V & W_1P \\ \hline 0 & W_2 \\ \hline -V & -P \end{array}\right]$$

The closed-loop transfer matrix from w to

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

is precisely the function H whose infinity-norm we wish to minimize.

There are many ways to solve this problem, but only the frequency domain solution (Kwakernaak, 1996) allows the generalized plant to have a nonproper transfer matrix G. To enhance robustness at high frequencies it usually is necessary to make the weighting filter W_2 nonproper, which in turn makes G nonproper.

For simplicity we develop the solution for the SISO case only. Suppose that

$$P = \frac{n}{d}, \quad V = \frac{m}{d}, \quad W_1 = \frac{a_1}{b_1}, \quad W_2 = \frac{a_2}{b_2}$$

where the numerators and denominators are (scalar) polynomials. Note that P and V have the same denominators d — this makes partial pole placement possible (Kwakernaak, 1993).

Numerical example

We study in particular the following numerical example (Kwakernaak, 1963):

$$P(s) = \frac{1}{s^2}, \quad V(s) = \frac{1 + s\sqrt{2} + s^2}{s^2}, \quad W_1(s) = 1, \quad W_2(s) = c(1 + rs)$$

where we let $c = r = 1$. Note that W_2 is nonproper and, hence, the generalized plant G is nonproper. The various polynomials are defined by the following command lines:

```
% Define the data

n = 1; d = s^2; m = s^2+s*sqrt(2)+1;

c = 1; r = 1;  a1 = 1; b1 = 1; a2 = c*(1+r*s); b2 = 1;
```

Left coprime polynomial matrix fraction representation

The frequency domain solution of the H_∞ problem of Kwakernaak (1996) is based on polynomial matrix manipulations. It requires G to be represented in left coprime polynomial matrix fraction form. The desired left coprime factorization is

$$G = \left[\begin{array}{c|c} \frac{a_1 m}{b_1 d} & \frac{a_1 n}{b_1 d} \\ 0 & \frac{a_2}{b_2} \\ \hline -\frac{m}{d} & -\frac{m}{d} \end{array}\right] = \underbrace{\left[\begin{array}{cc|c} b_1 & 0 & a_1 \\ 0 & b_2 & 0 \\ 0 & 0 & d \end{array}\right]}_{[D_1 \quad D_2]}^{-1} \underbrace{\left[\begin{array}{c|c} 0 & 0 \\ 0 & a_2 \\ -m & n \end{array}\right]}_{[N_1 \quad N_2]}$$

The partitioning is needed for the solution of the H_∞ problem. The following code lines define the various polynomial matrices:

```
% Define the various polynomial matrices

D1 = [b1 0; 0 b2; 0 0];

D2 = [a1  ; 0   ; d  ];
```

```
N1 = [ 0;  0; -m ];
N2 = [ 0; a2; -n ];
```

These code lines are actually taken from the macro `mixeds`, which automates the entire computation.

Frequency domain solution of the *H*-infinity problem

As usual in the solution of the H_∞ problem we consider the problem of finding a compensator that stabilizes the system and makes the infinity norm of the closed-loop transfer matrix less than or equal to a given number γ. Define the rational matrix

$$\Pi_\gamma^{-1} = \begin{bmatrix} N_2^* \\ D_2^* \end{bmatrix} (D_1 D_1^* - \frac{1}{\gamma^2} N_1 N_1^*)^{-1} \begin{bmatrix} N_2 & D_2 \end{bmatrix}$$

The * denotes conjugation, that is, $A^*(s) = A^T(-s)$. In the next subsection it is seen how a spectral factorization

$$\Pi_\gamma^{-1} = M_\gamma^* J M_\gamma$$

of Π_γ^{-1} may be obtained. The spectral factor M_γ is a square rational matrix such that both M_γ and M_γ^{-1} have all their poles in the open left-half plane. The matrix J is a signature matrix of the form

$$J = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}$$

where the two unit matrices do not necessarily have the same dimensions.

Given this spectral factorization, *all* compensators whose norm is less than the number γ are of the form $K = X^{-1}Y$, where

$$\begin{bmatrix} X & Y \end{bmatrix} = \begin{bmatrix} I & U \end{bmatrix} M_\gamma$$

U is a rational stable matrix such that $\|U\|_\infty \le 1$. In particular one may chose $U = 0$.

Spectral factorization

We consider the spectral factorization

$$\Pi_\gamma^{-1} = M_\gamma^* J M_\gamma$$

It requires the following steps.

1. Do the polynomial spectral cofactorization

$$D_1 D_1^* - \frac{1}{\gamma^2} N_1 N_1^* = Q_\gamma J_o Q_\gamma^{-1}$$

2. Perform the "left-to-right" conversion

$$Q_\gamma^{-1}\begin{bmatrix} N_2 & D_2 \end{bmatrix} = \Delta_\gamma \Lambda_\gamma^{-1}$$

3. Do the polynomial factorization

$$\Delta_\gamma^* J_o \Delta_\gamma = \Gamma_\gamma^* J \Gamma_\gamma$$

Then the desired spectral factor is

$$M_\gamma = \Gamma_\gamma \Delta_\gamma^{-1}$$

The first polynomial spectral factorization

The first spectral factorization may actually be done analytically, because we have

$$D_1 D_1^* - \frac{1}{\gamma^2} N_1 N_1^* = \begin{bmatrix} D_1 & N_1 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -\frac{1}{\gamma^2} I \end{bmatrix} \begin{bmatrix} D_1 & N_1 \end{bmatrix}^*$$

$$= \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & m \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{\gamma^2} \end{bmatrix} \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & m \end{bmatrix}^*$$

Inspection shows that if each of the polynomials b_1, b_2 and m is strictly Hurwitz then the desired spectral factorization may be rendered in slightly modified form as

$$D_1 D_1^* - \frac{1}{\gamma^2} N_1 N_1^* = Q J_\gamma^{-1} Q^*, \qquad Q = \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & m \end{bmatrix}, \qquad J_\gamma = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -\gamma^2 \end{bmatrix}$$

Left-to-right conversion

The computation of the left-to-right conversion

$$Q^{-1}\begin{bmatrix} N_2 & D_2 \end{bmatrix} = \Delta \Lambda^{-1}$$

may easily be coded.

```
% Left-to-right conversion
```

```
Q = diag([b1 b2 m]);
[Del,Lam] = lmf2rmf([N2 D2],Q)
```

The result is

```
Del =

    -0.71                    0.71 + s

     0.71 + 1.7s + s^2      0.71 + 0.71s

    -0.71                   -0.71 + s

Lam =

     0.71 + s      0.71

    -0.71          0.71 + s
```

Second polynomial factorization

The second spectral factorization now takes the form

$$\Delta^* J_\gamma \Delta = \Gamma_\gamma^* J \Gamma_\gamma$$

It is not difficult to write the necessary code lines

```
% Define gamma
gamma = 4;
% Spectral factorization
Jgamma = eye(3); Jgamma(3,3)= -gamma^2;
DelDel = Del'*Jgamma*Del;
[Gam,J] = spf(DelDel)
Gam =

    -1.3e+002 - 50s - s^2         -2.2e+002 - 0.046s

     1.3e+002 + 48s + 0.17s^2      2.2e+002 - 3.8s

J =

     1     0
```

```
     0    -1
```

Computation of the compensator

To determine the numerator Y and denominator X of the compensator we need to compute

$$\begin{bmatrix} X & Y \end{bmatrix} = \begin{bmatrix} I & U \end{bmatrix} M_\gamma = \begin{bmatrix} I & U \end{bmatrix} \Gamma_\gamma \Delta^{-1}$$

where for simplicity we choose $U = 0$. It is advantageous to implement this computation as a right-to-left conversion

$$\begin{bmatrix} I & U \end{bmatrix} \Gamma_\gamma \Delta^{-1} = z^{-1} \begin{bmatrix} x & y \end{bmatrix}$$

so that the compensator transfer function is

$$K = X^{-1} Y = x^{-1} y = \frac{y}{x}$$

Again this may be coded straightforwardly:

```
% Computation of the compensator
xy = rmf2lmf([1 0]*Gam,Lam);
x = xy(1,1), y = xy(1,2)
```

The output is

```
x =
     2.4e+002 + 1.6e+002s + 50s^2 + s^3
y =
     63 + 1.8e+002s - 0.66s^2
```

Computation of the closed-loop poles

The closed-loop characteristic polynomial is

$$\phi = dx + ny$$

We use it to test closed-loop stability and to compute the closed-loop poles.

```
% Computation of the closed-loop characteristic polynomial
% and closed-loop poles
phi = d*x+n*y;
```

```
clpoles = roots(phi)
clpoles =
 -46.7980
  -0.9727 + 0.6271i
  -0.9727 - 0.6271i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
```

Approaching the optimal solution

It is easy to collect the command lines listed so far in an m-script and to run the script repeatedly for different values of γ.

We first run the script with `gamma` = 4. The closed-loop poles all have negative real parts, and, hence, the closed-loop system is stable.

Next, we run the macro with `gamma` = 3.5. The script returns

```
clpoles =
   4.9995
  -0.9590 + 0.5600i
  -0.9590 - 0.5600i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
```

The closed-loop system is unstable. Hence, `gamma` has been chosen too small.

Note that four of the five closed-loop poles do not change much with `gamma`. The fifth pole is very sensitive to changes in `gamma`.

We test this dependence by running the script several times for different values of `gamma` without showing the output. Table 1 shows the results. Apparently as `gamma` decreases the fifth pole crosses over from the left- to the right-half complex plane, but does so through infinity.

For the final run we take gamma = 3.9515, close to the optimal value but such that the closed-loop system is stable. The corresponding numerator polynomial y and the denominator polynomial x of the compensator are

```
y =

     4e+003 + 1.1e+004s - 0.66s^2

x =

   1.5e+004 + 1e+004s + 3e+003s^2 + s^3
```

Table 1. Dependence of the fifth pole on gamma

gamma	the fifth pole
4	–46.798
3.5	4.9995
3.75	11.369
3.875	30.297
3.9375	173.86
3.9688	–127.80
3.95	315.38
3.951	3153.8
3.9515	–2987.6

Calculation of the optimal compensator

Inspection of the numerator and denominator polynomials y and x of the compensator obtained for gamma = 3.9515 shows that their coefficients are large, except for the leading coefficients.

In fact, we may cancel the leading coefficients and simplify y and x. This amounts to cancelling the compensator pole-zero pair and eliminating the corresponding closed-loop pole that pass through infinity as gamma passes through the optimal value.

Recalculation of the closed-loop poles after this cancellation confirms that the large closed-loop pole has disappeared. These calculations are performed by typing a few simple command lines

```
y{2} = 0; x{3} = 0;
y = y/x{2}, x = x/x{2}
y =
     1.3 + 3.8s
x =
      5.1 + 3.4s + s^2
phi = d*x+n*y;
clpoles = roots(phi)
clpoles =
  -0.9703 + 0.6201i
  -0.9703 - 0.6201i
  -0.7075 + 0.7072i
  -0.7075 - 0.7072i
```

Assessment of the design

To assess the design we calculate and plot the sensitivity function S and the complementary sensitivity function T of the closed-loop system. They are given by

$$S = \frac{dx}{\phi}, \quad T = \frac{ny}{\phi}$$

We type in the command lines

```
omega = logspace(-2,2); j = sqrt(-1);
S = bode(pol2mat(d*x),pol2mat(phi),omega);
T = bode(pol2mat(n*y),pol2mat(phi),omega);
figure(1);
```

```
loglog(omega,abs(S),'k'); hold on
loglog(omega,abs(T),'b'); grid on
text(.1,.01,'S'), text(10,.01,'T')
```

Fig. 3 shows the plots of S and T. For a discussion of the design and the mixed sensitivity design methodology see Kwakernaak (1993).

Alternative spectral factorization

As we have seen, the spectral factorization behaves poorly as the optimal solution is approached. The reason is that the factorization becomes "noncanonical" (Kwakernaak, 1996). This difficulty may be remedied by using an alternative form for the second polynomial spectral factorization. Instead of factoring

$$\Delta^* J_\gamma \Delta = \Gamma_\gamma^* J \Gamma_\gamma$$

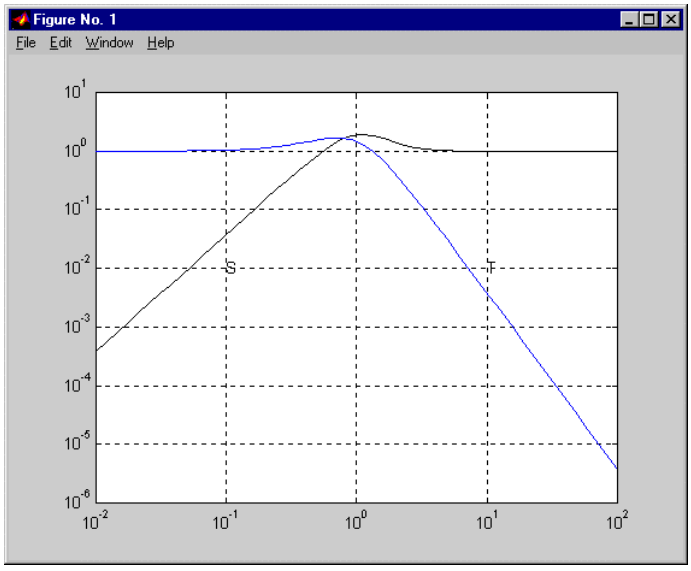

Fig. 3. Magnitude plots of the sensitivity functions

we rearrange the factorization as

$$\Delta^* J_\gamma \Delta = \Gamma_\gamma^* L^{-1} \Gamma_\gamma$$

L is diagonal in the form

$$L = \mathrm{diag}(L_1, -L_2)$$

where L_1 and L_2 are diagonal nonnegative-definite but not unit matrices. If the factorization is close to noncanonical then L is close to nonsingular. The large numbers disappear.

The alternative factorization is obtained by an option in the `spf` command. The computation of the compensator is not affected, so we only need to change the code line that contains the `spf` command to

```
[Gam,J] = spf(DelDel,'nnc');
```

Rerunning the script with this modification for `gamma` = 3.9515 shows that the large numbers have disappeared. We also see that instead of the large closed-loop poles and corresponding large pole and zero of the compensator we now have a closed-loop pole, compensator pole and zero at –1:

```
y,x

y =

     1.3 + 5.1s + 3.8s^2

x =

     5.1 + 8.4s + 4.4s^2 + s^3

rootsx = roots(x), rootsy = roots(y), clpoles

rootsx =

  -1.6787 + 1.5027i

  -1.6787 - 1.5027i

  -1.0000

rootsy =

   -1.0000

   -0.3476

clpoles =
```

```
-0.7071 + 0.7071i

-0.7071 - 0.7071i

-1.0002

-0.9715 + 0.6197i

-0.9715 - 0.6197i
```

Cancellation of the common root of y and x leads to the same optimal compensator that was previously obtained:

```
x/(s-rootsx(3)), y/(s-rootsy(1))

ans =

     5.1 + 3.4s + s^2

ans =

     1.3 + 3.8s
```

Automating the search

Now that the numerical instability in the computation of the compensator has been removed it is simple to automate the search process. We implement a binary search that involves the following steps:

1. Specify a minimal value `gmin` and a maximal value `gmax` for `gamma`.
2. Test if a stabilizing compensator is found at `gamma` = `gmax`. If not then stop.
3. Test if a stabilizing compensator is found at `gamma` = `gmin`. If yes then stop.
4. Let `gamma` = (`gmin`+`gmax`)/2. If a stabilizing compensator is found then let `gmax` = `gamma`, otherwise let `gmin` = `gamma`.
5. If `gmax-gmin` is greater than a prespecified accuracy then return to 4.
6. Retain the solution for `gamma` = `gmax` and stop.

This search algorithm has been implemented in the macro `mixeds`. Calling the routine in the form

```
gmin = 3.5; gmax = 4; accuracy = 1e-4;

[y,x,gopt] = mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,'show')
```

executes the search while showing the intermediate results. The search stops if `gmax-gmin` is less than the input parameter `accuracy.` This is the output:

```
    gamma  test result
    -----  -----------
    4      stable
    3.5    unstable
    3.75   unstable
    3.875  unstable
    3.9375 unstable
    3.96875     stable
    3.95313     stable
    3.94531     unstable
    3.94922     unstable
    3.95117     stable
    3.9502      unstable
    3.95068     unstable
    3.95093     stable
    3.95081     stable
    3.95074     stable
Cancel root at -0.999999
y =
     1.3 + 3.8s
x =
     5.1 + 3.4s + s^2
```

```
gopt =

    3.9507
```

Cancelling coinciding pole-zero pairs

The numerator x and the denominator y of the optimal compensator turn out to have a common root. This often happens. The precise location of this spurious pole-zero pair is unpredictable. It needs to be cancelled in the compensator transfer function $C = y / x$.

Rather than relying on one of the polynomial division routines we write a few dedicated code lines. Suppose that the numerator y has a root z. Then by polynomial division we have for the denominator x

$$x(s) = q(s)(s - z) + r$$

where the remainder r is a constant. By substituting $s = z$ we see that actually $r = x(z)$. Expanding the polynomials x and q as

$$x(s) = x_n s^n + x_{n-1} s^{n-1} + \cdots + x_0, \qquad q(s) = q_{n-1} s^{n-1} + q_{n-2} s^{n-2} + \cdots + q_0$$

it follows that the quotient q and the remainder r may recursively be computed as

$$\begin{aligned} q_{n-1} &= x_n \\ q_{k-1} &= x_k + q_k z, \quad k = n, n-1, \cdots, 1 \\ r &= x_0 + q_0 z \end{aligned}$$

Note that this way of computing $r = x(z)$ is nothing else than Horner's algorithm. If the remainder r is small then we cancel the factor $s–z$. By the same algorithm the factor $s–z$ may be cancelled from the numerator y.

These are the necessary code lines. A tolerance `tolcncl` is used to test if the remainder is small.

```
% Cancel any common roots of xopt and yopt

rootsy = roots(yopt);

xo = xopt{:}; degxo = deg(xopt);

yo = yopt{:}; degyo = deg(yopt);

for i = 1:length(rootsy)

   z = rootsy(i);
```

```
      % Divide xo(s) by s-z
      q = zeros(1,degxo);
      q(degxo) = xo(degxo+1);
      for j = degxo-1:-1:1
         q(j) = xo(j+1)+z*q(j+1);
      end
      % If the remainder is small then cancel
      % the factor s-z both in xo and in yo
      if abs(xo(1)+z*q(1)) < tolcncl*norm(xo,1)
         xo = q; degxo = degxo-1;
         p = zeros(1,degyo);
         p(degyo) = yo(degyo+1);
         for j = degyo-1:-1:1
            p(j) = yo(j+1)+z*p(j+1);
         end
         yo = p; degyo = degyo-1;
         if show
           disp(sprintf('\nCancel root at %g\n',z));
         end
      end
   end
```

The function mixeds

The full command

```
[y,x,gopt] = ...
    mixeds(n,m,d,a1,b1,a2,b2,gmin,gmax,accuracy,tol,'show')
```

includes a four-dimensional optional tolerance parameter

```
tol = [tolcncl tolstable tolspf tollr]
```

which allows to fine tune the macro. For a description of the various tolerances consult the manual page for `mixeds`.