Possibility of large, coupled sensitivities in MIMO problems

See also: `mimoMotivateResolveMU.m` and `mimoSolve.m`.

MUSYNID

Contents

Nominal Model

Constant N-by-N real matrix. The plant is **not** diagonal. It represents an idealized manufacturing process, such as paper pulp spraying, where many nearly identical actuators contribute in a spatially invariant manner to a final result.

```
N = 5;
if N>=3
   G = toeplitz([1;.75;.40;zeros(N-3,1)],[1 .75 .40 zeros(1,N-3)]);
   tLoc = [2 2;N 2;3 N;N N];
elseif N==2
   G = toeplitz([1;.40],[1 .40]);
   tLoc = [1 1;1 2;2 1;2 2];
elseif N==1
   G = 1;
   tLoc = [1 1];
end
G = ss(G)
```

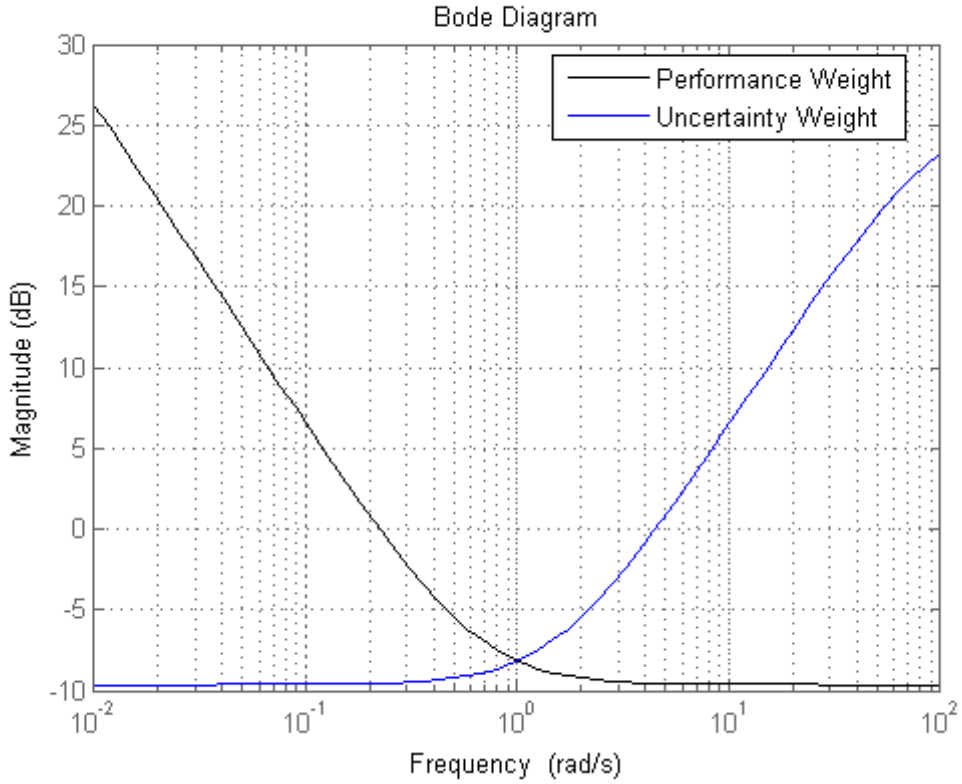
```
G =

  d =
         u1     u2     u3     u4     u5
   y1     1   0.75    0.4      0      0
   y2  0.75      1   0.75    0.4      0
   y3   0.4   0.75      1   0.75    0.4
   y4     0    0.4   0.75      1   0.75
   y5     0      0    0.4   0.75      1

Static gain.
```

Performance/Uncertainty interaction

Since the plant `G` is nondynamic, any/all frequency-dependence will arise in the interaction between the uncertainty weightings (`Wunc`) and the performance weightings (`Wp`). A parameter `BWsep` will be used to quantify the separation between:

- the desired bandwidth of sensitivity reduction, [0 1/BWsep], and
- the frequency range for which plant uncertainty exceeds 100%, [BWsep inf).

```
BWsep = 4.5;
wp = makeweight(100,1/BWsep,0.33);
wu = makeweight(0.33,BWsep,20);
bodemag(wp,'k',wu,'b',{1e-2 1e2})
grid
legend('Performance Weight','Uncertainty Weight','Location','Best');
```

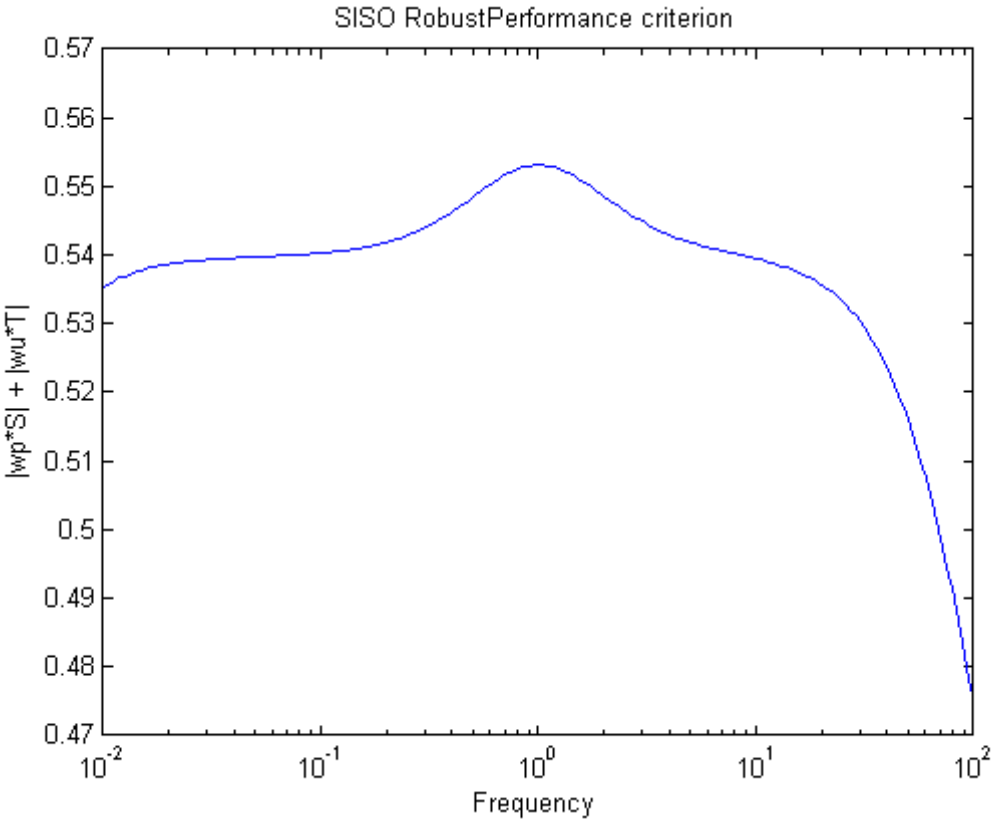


Performance/Uncertainty tradeoff for a Scalar, constant plant

Note that if `Gsiso` is a scalar, then a very simple integral controller, based on inversion, `Kinv = inv(Gsiso)*tf(1,[1 0])`, solves the problem nicely. The crossover-frequency is chosen to be 1, since the Uncertainty and Performance weights are "balanced" around 1.

```
Gsiso = ss(randn);
Kinv = 1/Gsiso*tf(1,[1 0]);
SF = loopsens(Gsiso,Kinv);
wgrid = logspace(-2,2,200);
RPtest = abs(wp*frd(SF.So,wgrid)) + abs(wu*frd(SF.Ti,wgrid));
semilogx(RPtest)
xlabel('Frequency')
ylabel('|wp*S| + |wu*T|')
title('SISO RobustPerformance criterion')
```

```
Warning: Ignoring all input names because of name conflicts.
```



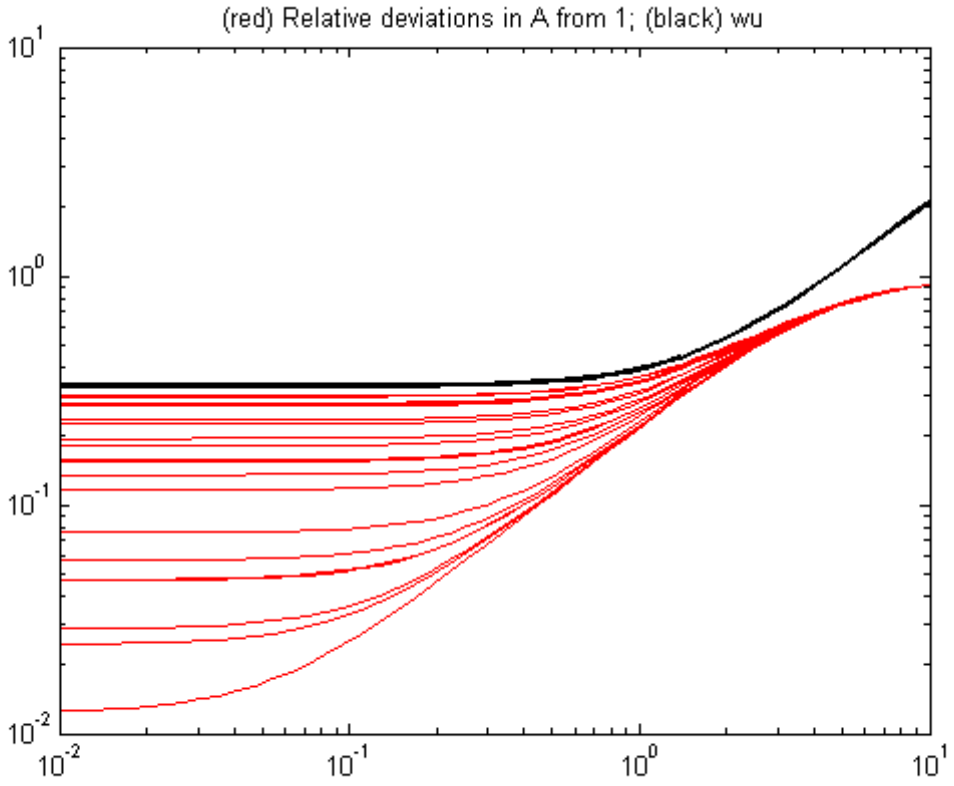
This shows that a SISO problem easily achieves RobustPerformance for this pairing of uncertainty/performance objectives.

Performance/Uncertainty Time-domain verification

Create array of nonideal actuators, consistent with uncertainty model It is easy to verify that 1st-order actuator, `gamma/(tau*s + 1)`, is consistent with the uncertainty model if `0.67<gamma<1.33` and `tau = 1/BWsep`. Construct an array of 20-by-1 array of actuator models with these properties.

```
clf
kMax = 20;
A = ss(zeros(1,1,kMax));  % 20-by-1 SS array of actuator models
for k=1:kMax
   % Create different non-Ideal actuators that satisfy uncertainty model
   gamma = 1 + 0.33*(2*(rand-0.5));
   tau = 1/BWsep;
   A(:,:,k) = ss(tf(gamma,[tau 1]));
   % relative difference from ideal actuator, AIdeal = 1
   loglog(abs(frd(A(:,:,k)-1,logspace(-2,1,50))),'r');
   hold on
end
% Compare the relative difference to the Uncertainty weight
H = loglog(abs(frd(wu,logspace(-2,1,50))),'k');
set(H,'linewidth',2)
title('(red) Relative deviations in A from 1; (black) wu')
```

```
hold off
```

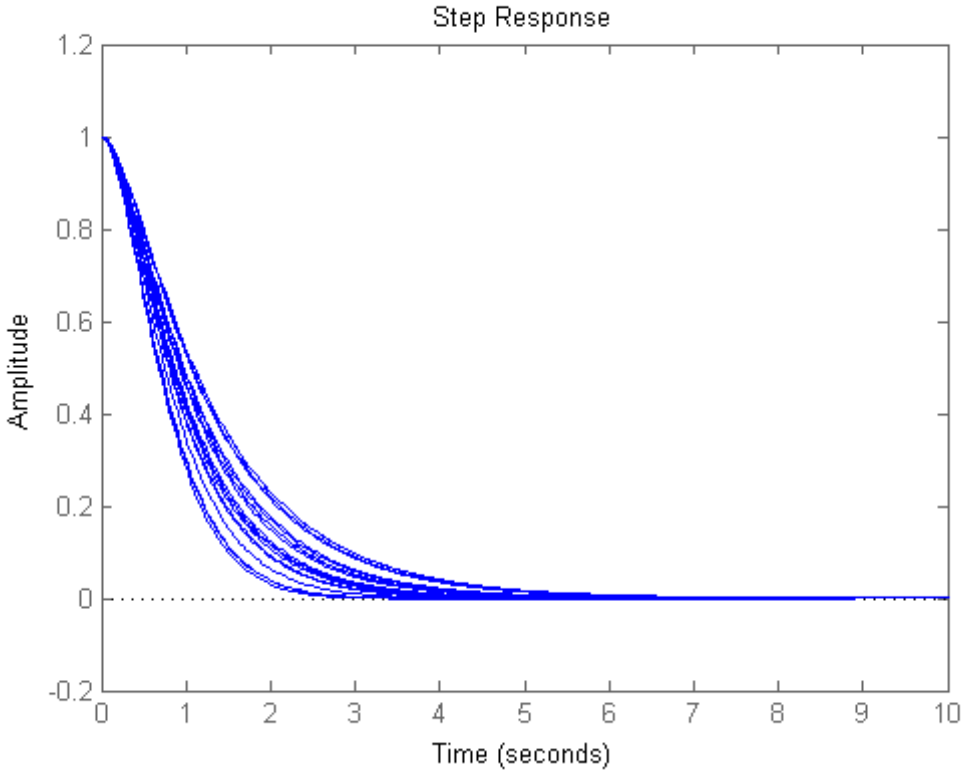


This plot confirms that each actuator model is with the set of models defined by a nominal model of 1, coupled with the uncertainty weight `wu`.

Time-domain simulations using non-ideal actuators

Using the nonideal actuators, which are consistent with the modeled uncertainty, perform time-domain simulations, and examine the output disturbance rejection. Create G_NonIdeal

```
G_NonIdeal = Gsiso*A;
% Output Sensitivity
S = feedback(1,G_NonIdeal*Kinv);
% Disturbance Step-response at output
step(S,10);
```



All responses are similar to the nominal response, as expected. Next steps are to see what happens back in the MIMO case.

Uncertainty in each InputChannel

The plant model will have independent uncertainty in all input channels. This would be typical in most situations. Use ULTIDYN atoms along with the uncertainty weighting functions to add uncertainty to the nominal plant. Note that in this specific example, the level of uncertainty in each of the channels is identical.

```
Wunc = ss(zeros(N,N));
Delta = uss(ss(zeros(N,N)));
for i=1:N
   Wunc(i,i) = wu;
   UName = ['delta' int2str(i)];
   Delta(i,i) = ultidyn(UName,[1 1]);
end
Gunc = G*(eye(N) + Wunc*Delta)
```

```
Gunc =

  Uncertain continuous-time state-space model with 5 outputs, 5 inputs, 5 states.
  The model uncertainty consists of the following blocks:
    delta1: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences
```

```
    delta2: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences
    delta3: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences
    delta4: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences
    delta5: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences

Type "Gunc.NominalValue" to see the nominal value, "get(Gunc)" to see all
properties, and "Gunc.Uncertainty" to interact with the uncertain elements.
```

Output Sensitivity Performance Objective

The scalar weight `wp` will be used to weight the output sensitivity function, so that the goal is `norm(S(jw)` <= 1/|wp(jw)| for all w. Create an N-by-N system which is just this scalar weight times the identity matrix.

```
Wp = wp*eye(N);
```

Inverse-Based Controller

Make a simple integral controller, with gain matrix equal to the inverse of the nominal plant's constant transfer function matrix.

```
Kinv = ss(0,1,1,0)*inv(G.d);
```

Closed-loop Sensitivity functions

Use the command `loopsens` to form all various closed-loop sensitivity and complementary sensitivity functions, by interconnection of the uncertain plant model `Gunc` along with the inverse-based controller `Kinv`.

```
SF = loopsens(Gunc, Kinv);
```

Check Nominal Performance (output sensitivity)

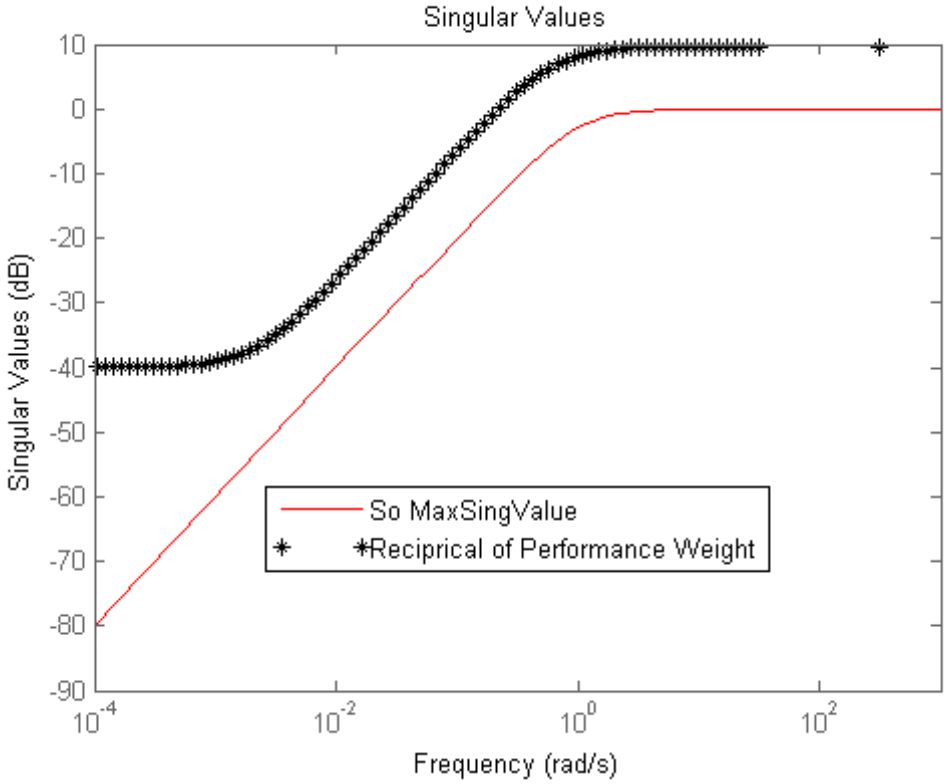
As a weighted norm, with Norm()<1, this confirms **nominal performance**

```
norm(Wp*SF.So.NominalValue, inf)
```

```
ans =
   3.3000e-01
```

SIGMA plot, with 1/Wp shown, confirming nominal performance

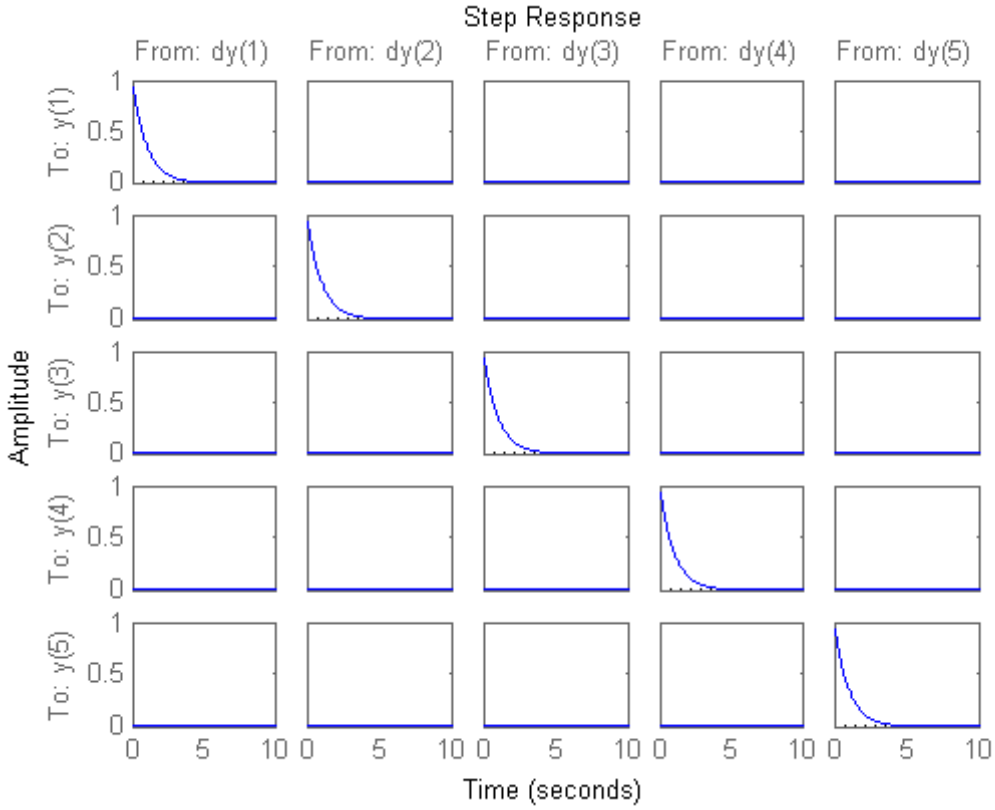
```
sigma(SF.So.NominalValue,'r', 1/Wp(1,1),'k*');
legend('So MaxSingValue','Reciprical of Performance Weight','Location','Best')
```



Nominal StepResponse at Output

This clear illustrates how the inverse-based controller completely decouples the closed-loop response of the plant to output disturbances. All responses have a bandwidth of 1, which should be expected since `Lo` is simply `Lo(s) = (1/s) eye(N)`.

```
step(SF.So.NominalValue, 10)
```



Robust Stability (to a full 5-by-5 block)

Weighted Input Complimentary Sensitivity norm, confirming Robust Stability for 5-by-5, unstructured unmodeled dynamics block at plant input. Again, as a weighted norm, with Norm()<1, this confirms **robust stability**.

```
norm(Wunc*SF.Ti.NominalValue, inf)
```

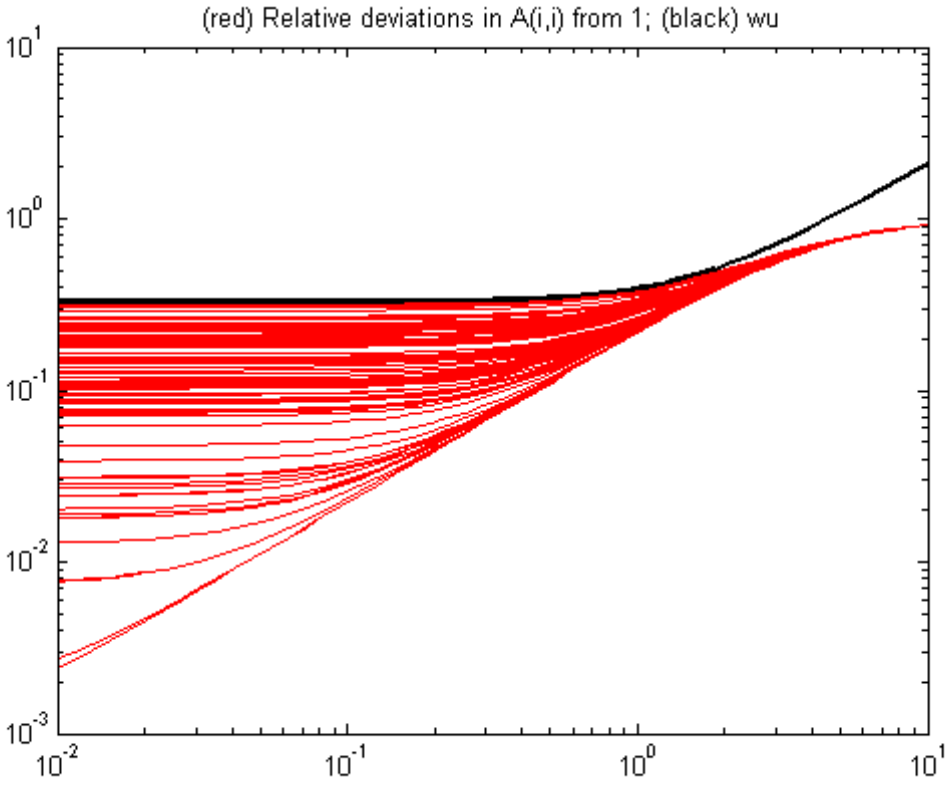
```
ans =
   3.3000e-01
```

Create array of nonideal actuators, consistent with uncertainty model

It is easy to verify that 1st-order actuator, `gamma/(tau*s + 1)`, is consistent with the uncertainty model if `0.67<gamma<1.33` and `tau = 1/BWsep`. Construct an array of N-by-N diagonal actuator models with these properties.

```
clf
kMax = 20;
A = ss(zeros(N,N,kMax));  % 20-by-1 array of N-by-N diagonal actuator models
for k=1:kMax
   % Create N different non-Ideal actuators that satisfy uncertainty model
   for i=1:N
```

```
      gamma = 1 + 0.33*(2*(rand-0.5));
      tau = 1/BWsep;
      A(i,i,k) = ss(tf(gamma,[tau 1]));
      % relative difference from ideal actuator, AIdeal = 1
      loglog(abs(frd(A(i,i,k)-1,logspace(-2,1,50))),'r');
      hold on
   end
end
% Compare the relative difference to the Uncertainty weight
H = loglog(abs(frd(wu,logspace(-2,1,50))),'k');
set(H,'linewidth',2)
title('(red) Relative deviations in A(i,i) from 1; (black) wu')
hold off
```

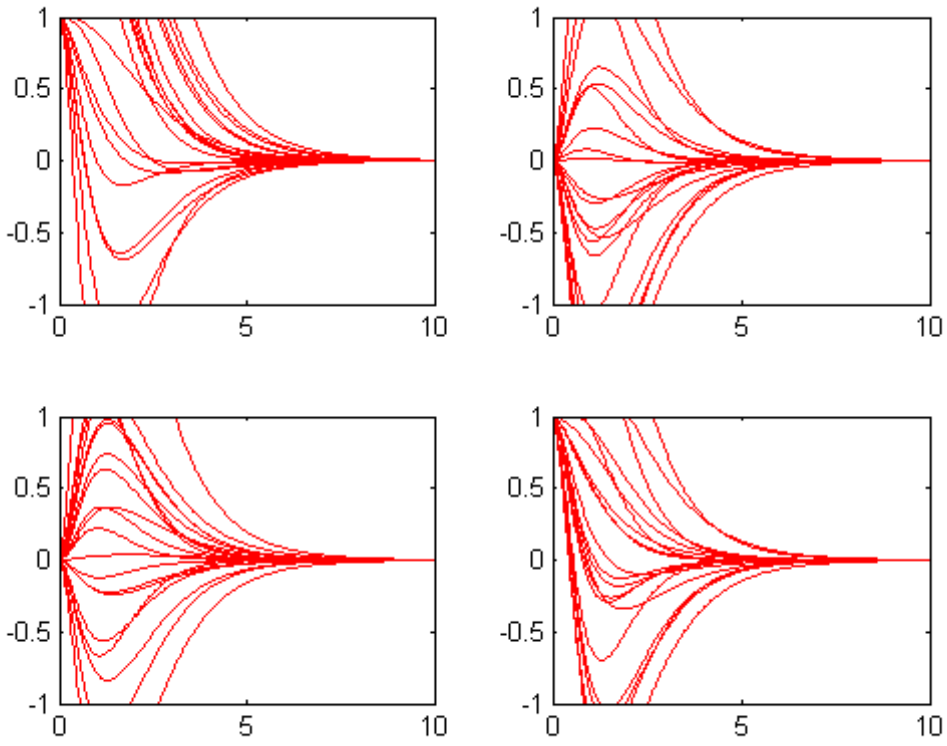


This plot confirms that each actuator model is with the set of models defined by a nominal model of 1, coupled with the uncertainty weight `wu`.

Time-domain simulations using non-ideal actuators

Using the nonideal actuators, which are consistent with the modeled uncertainty, and hence cannot cause instability, perform time-domain simulations, and examine the output disturbance rejection. Only 4 specific responses are shown. The code can easily be modified to examine other responses.

```
for k=1:kMax
   % Create G_NonIdeal
   G_NonIdeal = G*A(:,:,k);
   % Output Sensitivity
   S = feedback(eye(N),G_NonIdeal*Kinv);
   % Disturbance Step-response at output
   [Y,T] = step(S,10);
   if size(tLoc,1)==4
      sb = [2 2];
   elseif size(tLoc,1)==1
      sb = [1 1];
   end
   for m=1:size(tLoc,1)
      % Look at 4 typical responses
      subplot(sb(1),sb(2),m)
      plot(T,Y(:,tLoc(m,1),tLoc(m,2)),'r'); ylim([-1 1]); hold on % (d2->y2)
   end
end
```



Conclusions

In this MIMO example, an inversion-based integral controller performs very well with regard to **robust stability** and **nominal performance** objectives, However, the Robust Performance properties are quite poor, as evidenced by a series of MonteCarlo simulations. In contrast, a SISO problem with identical uncertanty and performance objectives easily achieves robust performance using an inversion-based controller.

Published with MATLAB® R2013b