Robust performance, Matlab SISO case study, 5/(s+1)^2: mu-synthesis

This code ran with no errors in Matlab **R2022a**

Presentations (video):

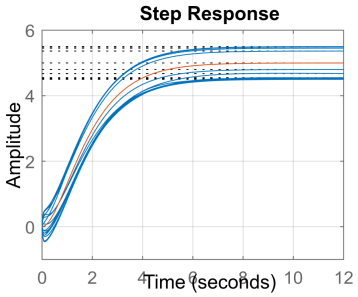
http://personales.upv.es/asala/YT/V/cerp4muEN.html , http://personales.upv.es/asala/YT/V/cerp5muEN.html .

Objectives: Understand the ideas behind the theory of robust performance and scaled small gain, with a second-order single-variable example, executed with μ-synthesis (looking for multipliers in an iterative automated way) instead of looking for multipliers "by hand" as in previous videos that were only of didactic/illustrative interest, but not advised as a go-to option in actual control design problems.

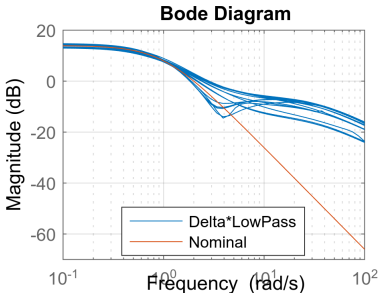
Table of Contents

Plant Model

```
s=tf('s');
G=5/(s+1)^2; %model, NOMINAL

BoundDelta=0.5; %Unstructured additive uncertainty size
Greal=G+BoundDelta*ultidyn("DeltaNormalized"); %uncertain model for mu-síntesis

Greal_simul=G+BoundDelta*ultidyn("DeltaNormalized")*1/(0.03*s+1);
%we kill things from 33 rad/s onwards in simulations.

step(Greal_simul,G,12), grid on
```

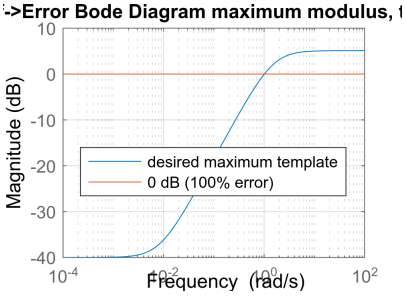



```
bodemag(Greal_simul,logspace(-1,2)), grid on, hold on
bodemag(G,logspace(-1,2)), hold off, legend("Delta*LowPass", "Nominal",Location="best")
```



Target robust performance bound (in frecuency domain)

```
TargetBandwidth=1.01; %approx. inverse prop. to rise time
templateErr=makeweight(0.01,TargetBandwidth,1.8);
bodemag(templateErr,tf(1)), grid on
title("Ref->Error Bode Diagram maximum modulus, template")
legend("desired maximum template","0 dB (100% error)",Location='best')
```

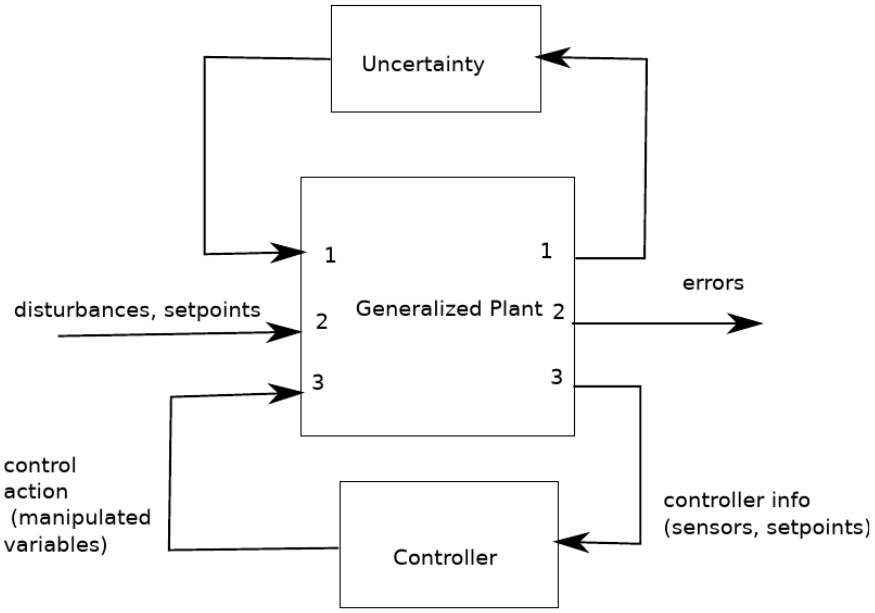


Note: for simplicity, we do not limit the control action by "performance" reasons (saturation, etc.), although in applications it would be recommended. In any case, the robustness in the face of additive uncertainty is achieved by limiting "u", so, actually, we are implicitly limiting "u" somehow.

Mu-Synthesis design for robust performance

Theoretical goal

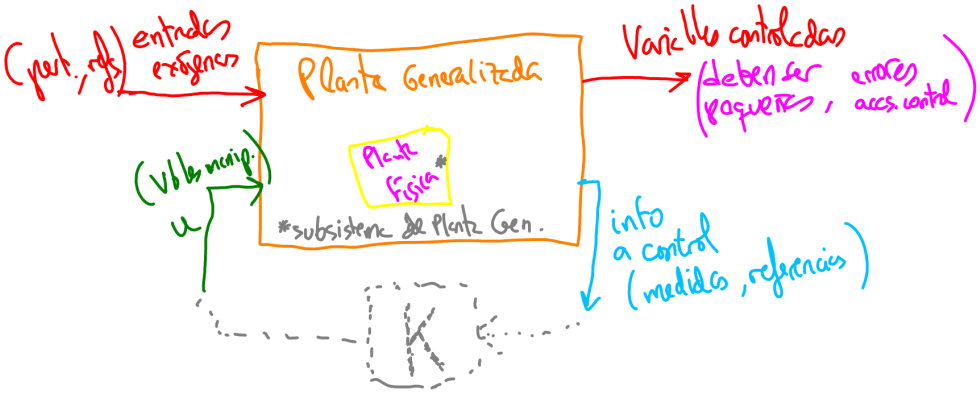
We would try to build a generalized 3x3 plant with uncertainty to apply (for example) scaled small gain theorem if we wanted to run the multiplier search on our own (as "manually" done in other previous videos).



If, however, we build with an uncertain model only the 2x2 part that refers to groups 2 and 3, Matlab's Robust Control Toolbox takes care of extracting the uncertainty and (internally) setting up the 3x3 plant. [When I say 2x2 or 3x3 I mean "groups" of signals, each group has different dimensions, depending on each specific problem to be solved].

*If we want to "explicitly" see the 3x3 plant that musyn uses, we can do it with the **`lftdata`** command (see below), but this explicit use is **not** necessary in most simple cases (**`musyn`** already takes care of it), and we'll pose the generalized plant with the uncertainty "inside".

Generalized Plant (2x2) for musyn



We will write `[err;err]=[1 -Greal;1 -Greal]*[ref;u]` avoiding multi-incidence of Greal:

```
PGenMu=minreal([[1;1] [-1;-1]*Greal] )%if we execute "ss" then uncertainty is destroyed
```

```
PGenMu =

  Uncertain continuous-time state-space model with 2 outputs, 2 inputs, 2 states.
  The model uncertainty consists of the following blocks:
    DeltaNormalized: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences

Type "PGenMu.NominalValue" to see the nominal value, "get(PGenMu)" to see all properties, and "PGenMu.Unce
```

Generalized Plant 3x3 from 2x2 + uncertainty inside (for comparison, this is NOT necessary)

We extract the uncertainty with **`lftdata`** as follows (it is really NOT necessary to do this; it is already done, internally, by **`musyn`**, transparent to us):

```
[M,Delta]=lftdata(PGenMu);
size(M)
```

```
State-space model with 3 outputs, 3 inputs, and 2 states.
```

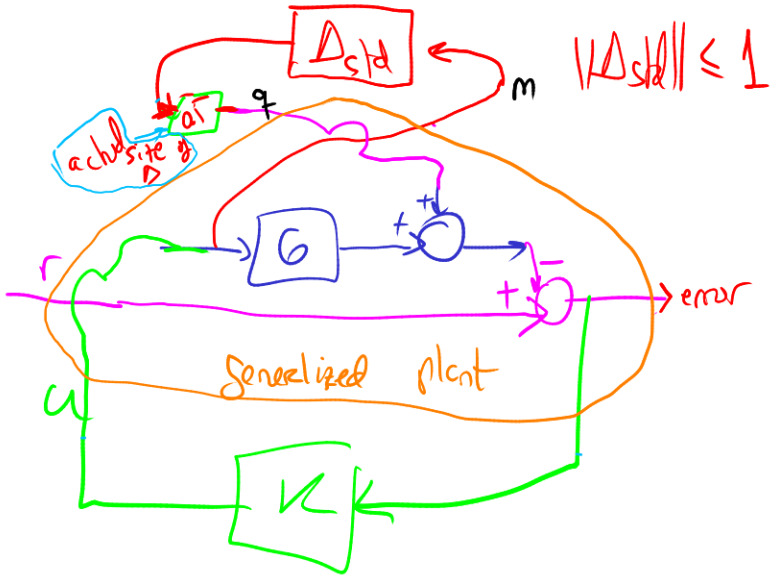
```
Delta
```

```
Delta =

  Uncertain continuous-time state-space model with 1 outputs, 1 inputs, 0 states.
  The model uncertainty consists of the following blocks:
    DeltaNormalized: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences

Type "Delta.NominalValue" to see the nominal value, "get(Delta)" to see all properties, and "Delta.Uncerta
```

```
zpk(M)
```

```
ans =

  From input 1 to output...
   1:  0

   2:  -0.5

   3:  -0.5

  From input 2 to output...
   1:  0

   2:  1

   3:  1

  From input 3 to output...
   1:  1

         -5
   2:  -------
       (s+1)^2

         -5
   3:  -------
       (s+1)^2

Continuous-time zero/pole/gain model.
```

It coincides with what we had proposed when searching "by hand" for the multiplier in other earlier videos. Indeed:




```
PGenOtherVideo=minreal(ss([0 0 1;-1 1 -G;-1 1 -G])); %NOT NEEDED when using musyn
zpk(PGenOtherVideo*blkdiag(BoundDelta,1,1)) %We add input weight with size of uncertain
```

```
ans =

  From input 1 to output...
   1:  0

   2:  -0.5

   3:  -0.5

  From input 2 to output...
   1:  0

   2:  1

   3:  1

  From input 3 to output...
   1:  1

         -5
   2:  -------
       (s+1)^2

         -5
   3:  -------
       (s+1)^2

Continuous-time zero/pole/gain model.
```

Weighted generalised plant (2x2) for musyn

```
Wref=1;
Win=blkdiag(Wref,1);
```

```
Wout=minreal(blkdiag(1/templateErr,1));
```

```
1 state removed.
```

```
PGenPond=Wout*PGenMu*Win %uncertainty is inside PGenMu
```

```
PGenPond =

  Uncertain continuous-time state-space model with 2 outputs, 2 inputs, 3 states.
  The model uncertainty consists of the following blocks:
    DeltaNormalized: Uncertain 1x1 LTI, peak gain = 1, 1 occurrences

Type "PGenPond.NominalValue" to see the nominal value, "get(PGenPond)" to see all properties, and "PGenPon
```

Mu-synthesis

Automated hinfsyn plus scaled-small-gain multiplier search (iterative) is done with:

```
tic, [Kmu,GAM]=musyn(PGenPond,1,1); toc, GAM %we do not explicitly use "M" "Delta" from
```

```
D-K ITERATION SUMMARY:
-----------------------------------------------------------------
                      Robust performance               Fit order
-----------------------------------------------------------------
  Iter         K Step       Peak MU       D Fit             D
    1          1.021        1.021        1.025             4
    2          1.001        1.001        1.005             8
    3         0.9986       0.9986       0.9987             8
    4         0.9981       0.9981       0.9984             8

Best achieved robust performance: 0.998

Elapsed time is 1.991494 seconds.
GAM = 0.9981
```

*The scaling for an ultidyn SISO is a dynamic one, $D(s)$, which improves the performance GAM that can be proven (with respect to a constant multiplier seen in other preliminary videos). See `D fit, Fit Order`.

It would be a "theoretically better" option if the uncertainty is linear and time invariant... but notice how the dynamic multiplier raises the order of the resulting regulator. However, it would not prove robustness against non-linearity with harmonic distortion or variation in time.

```
size(Kmu) %lti uncertainty permite pesos en frecuencia, por eso sube el orden.
```

```
State-space model with 1 outputs, 1 inputs, and 11 states.
```

Reduced-order controller

There are several options:

- reduce the order of the controller obtained with `musyn`,
- Tell `musyn` with some options to reduce the "fit order" of the "D" scaling
- Directly optimize a tunable fixed-structure controller of lower order (such as a PID)

We'll discuss first and third options.

Controller order reduction

```
hsvd(Kmu) %many small singular values is good news
```

```
ans = 11×1
   10.9823
    0.4618
    0.2640
    0.0316
    0.0006
    0.0005
    0.0003
    0.0001
    0.0001
    0.0000
     ⋮
```

```
Kmu_reduced=balred(Kmu,3);
wcgain(lft(PGenPond,Kmu_reduced))
```

```
ans = struct with fields:
           LowerBound: 0.9984
           UpperBound: 1.0005
    CriticalFrequency: 0.9415
```

```
zpk(Kmu_reduced)
```

```
ans =

  -0.063784 (s-9031) (s+1.182) (s+0.895)
  --------------------------------------
     (s+665.3) (s+4.892) (s+0.008398)

Continuous-time zero/pole/gain model.
```

There is a "very fast" pole we will remove with `freqsep`, to see what we can get:

```
[Kslow,Kfast]=freqsep(Kmu_reduced,150);
zpk(Kslow)
```

```
ans =

  -0.063784 (s-0.3137) (s+43.86)
  ------------------------------
      (s+0.008398) (s+4.892)

Continuous-time zero/pole/gain model.
```

```
zpk(Kfast)
```

```
ans =

   621.13
  ---------
  (s+665.3)

Continuous-time zero/pole/gain model.
```

```
Kmu_reduced2=Kslow+dcgain(Kfast);
wcgain(lft(PGenPond,Kmu_reduced2)) %we still have robust performance
```

```
ans = struct with fields:
           LowerBound: 0.9965
           UpperBound: 0.9983
    CriticalFrequency: 0.7940
```

```
zpk(Kmu_reduced2) %it's a PID+'noise filter' controller
```

```
ans =

  0.86976 (s+0.915) (s+1.151)
  ---------------------------
    (s+0.008398) (s+4.892)

Continuous-time zero/pole/gain model.
```

The resulting controller closely resembles a PID with a noise filter... this suggests direct optimization of a PID, which we'll see below.

Direct MUSYN optimization of a PID regulator

```
PID=tunablePID('PIDtest','PID');
ClosedLoopWithPID=lft(PGenPond,PID);tic
[TunedCL,GAM]=musyn(ClosedLoopWithPID); GAM,toc
```

```
D-K ITERATION SUMMARY:
-----------------------------------------------------------------
                      Robust performance               Fit order
-----------------------------------------------------------------
  Iter          K Step       Peak MU       D Fit             D
    1           1.022        1.022        1.024             4
    2           1.005        1.005        1.008             8
    3           1.002        1.001        1.002             8
    4           1.001        1.001        1.001             8

Best achieved robust performance: 1
GAM = 1.0009
Elapsed time is 2.996113 seconds.
```

Well, we haven't lost practically anything with respect to the generic "state space" regulator. Well, it would still be necessary to reduce the bandwidth a bit to be "below" 1 as conceptually required, but 0.001 excess norm is not going to matter in practice (the internal tolerance of the H-infinity solvers by default is already 1 percent)...

```
TunedPID=pid(TunedCL.Blocks.PIDtest)
```

```
TunedPID =

             1            s
  Kp + Ki * --- + Kd * --------
             s          Tf*s+1

  with Kp = 0.332, Ki = 0.189, Kd = 0.11, Tf = 0.206
```

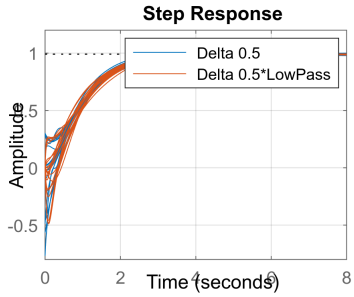
```
Name: PIDtest
Continuous-time PIDF controller in parallel form.
```

```
zpk(TunedPID) %quite similar to Kmu_reduced2 above, they achieve basically the same, ap
```

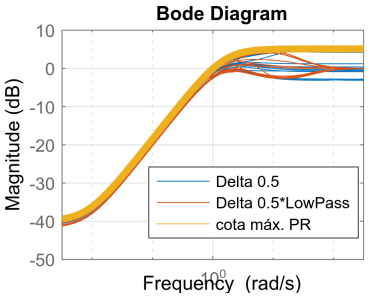
```
ans =

  0.86741 (s+1.19) (s+0.8884)
  ---------------------------
          s (s+4.863)

Name: PIDtest
Continuous-time zero/pole/gain model.
```

Time and frequency simulation

```
Kfinal=Kmu_reduced2;
%Kfinal=TunedPID;
step(feedback(Greal*Kfinal,1),feedback(Greal_simul*Kfinal,1),8), grid on
legend(" Delta 0.5"," Delta 0.5*LowPass")
```

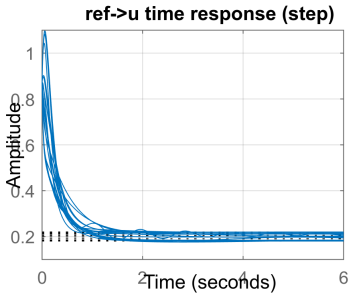



```
bodemag(feedback(1,Greal*Kfinal),feedback(1,Greal_simul*Kfinal),templateErr,logspace(-2
h_line=findobj(gcf,'type','line');
h_line(46).LineWidth=4; %trial and error to get the right line to make thicker
legend("Delta 0.5","Delta 0.5*LowPass","cota máx. PR",Location="best") %error
```

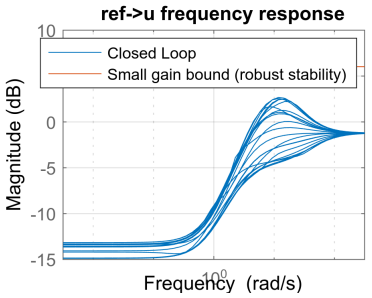


- Control action? Any "final" control design must have a "sensible" control action. We have not limited it theoretically, but it is important in practice, so let's see what it is:

```
step(feedback(Kfinal,Greal_simul),6), grid on
title("ref->u time response (step)")
```




```
bodemag(feedback(Kfinal,Greal_simul),tf(1/BoundDelta),logspace(-2.5,2.5))
grid on, title("ref->u frequency response"),legend("Closed Loop","Small gain bound (rob
```



Of course, it is below the robust stability bound (non-scaled small gain theorem), as robust performance is more stringent than robust stability. [robust stability requires the NOMINAL $K/(1+GK)$ being below the $1/\|Delta_{add}\|$ bound].