Control Robusto de un sistema masa-muelle-amortiguador incierto

Este código ejecutó sin errores con Matlab **R2022b**

Objetivos: Modelar un sistema incierto con la Robust Control Toolbox, simplificar la incertidumbre, y comparar técnicas mixed sensitivity, mu-síntesis con reguladores arbitrarios o con reguladores fijando la estructura a tipo PID.

Presentaciones en vídeo en:

http://personales.upv.es/asala/YT/V/rcml1.html [modelado ureal, ultidyn]

http://personales.upv.es/asala/YT/V/rcml2.html [simplificación incertidumbre ucover]

http://personales.upv.es/asala/YT/V/rcml3.html [mixed sensitivity]

http://personales.upv.es/asala/YT/V/rcml4.html [hinfstruct PID]

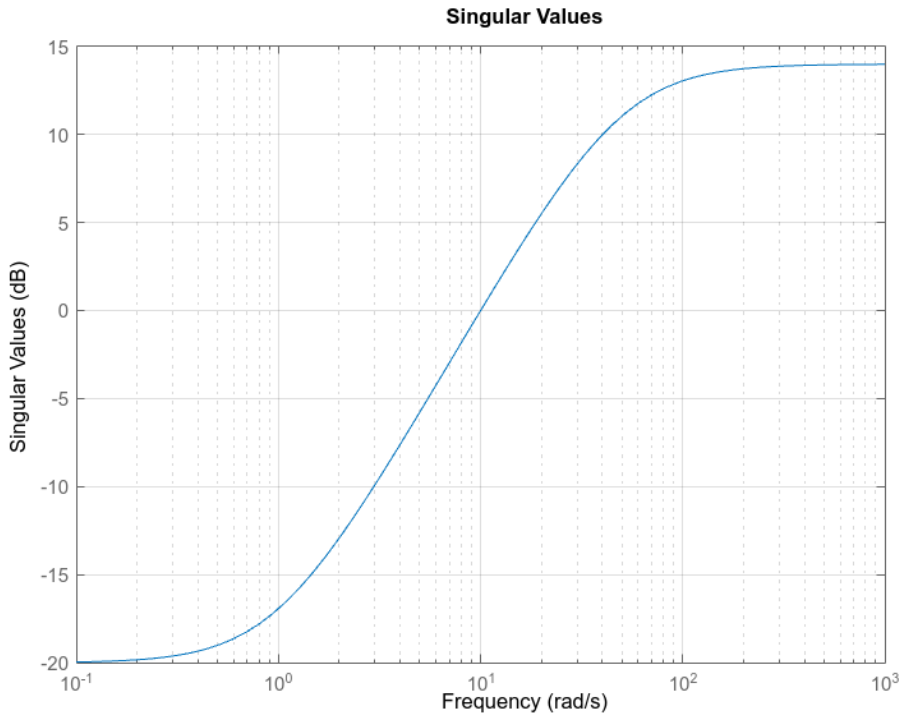
http://personales.upv.es/asala/YT/V/rcml5.html [musyn (dksyn obsoleto)

http://personales.upv.es/asala/YT/V/rcml6.html [PID musyn]

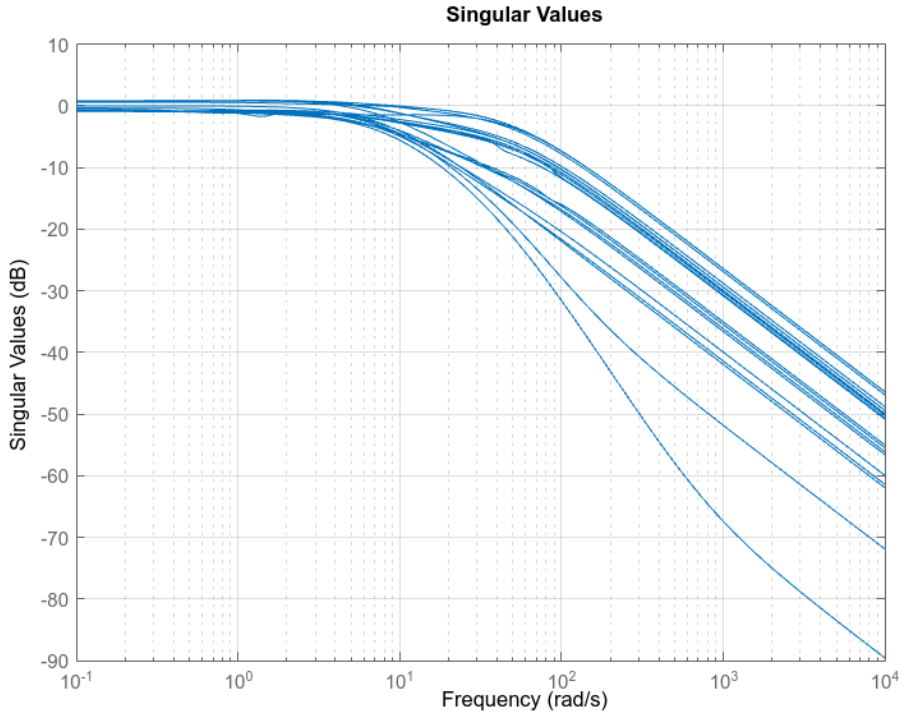
Tabla de Contenidos

1.- Modelado de sistema incierto

```
%masa muelle amortiguador
M=2.5;
k=ureal('kmuelle',5,'Perc',20);
f=ureal('famortiguador',1,'Perc',30);
s=tf('s');
G=1/(M*s^2+f*s+k);
deltaact=ultidyn('deltaact',[1 1]);
Wdelta=makeweight(0.1,10,5);
sigma(Wdelta), grid on
```




```
act=8/(s+8)*(1+Wdelta*deltaact);
sigma(act), grid on
```




```
P=G*act;
```

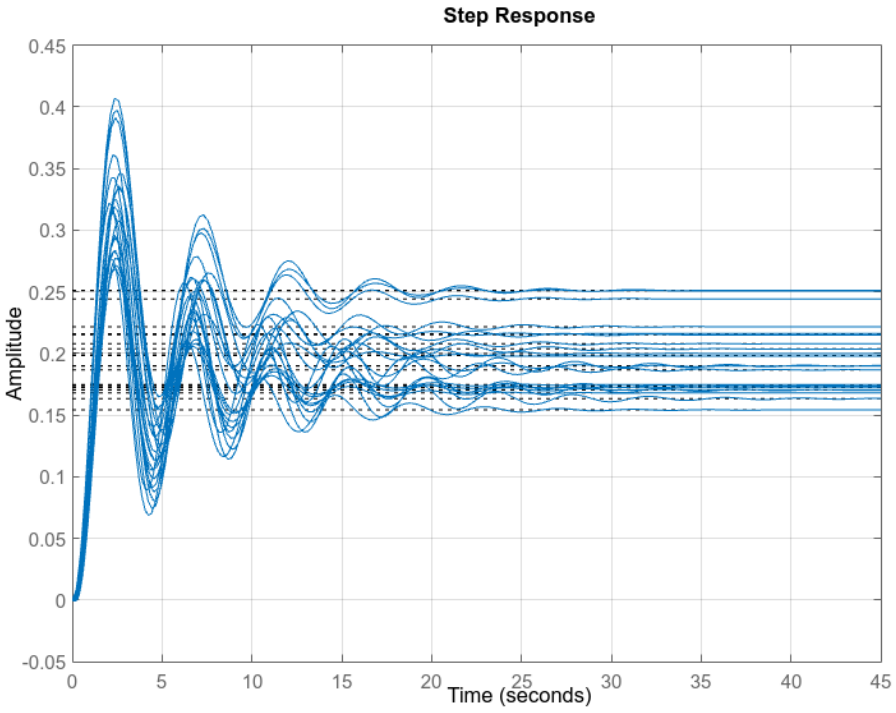
```
P.Uncertainty
```

```
ans = struct with fields:
          deltaact: [1×1 ultidyn]
     famortiguador: [1×1 ureal]
           kmuelle: [1×1 ureal]
```

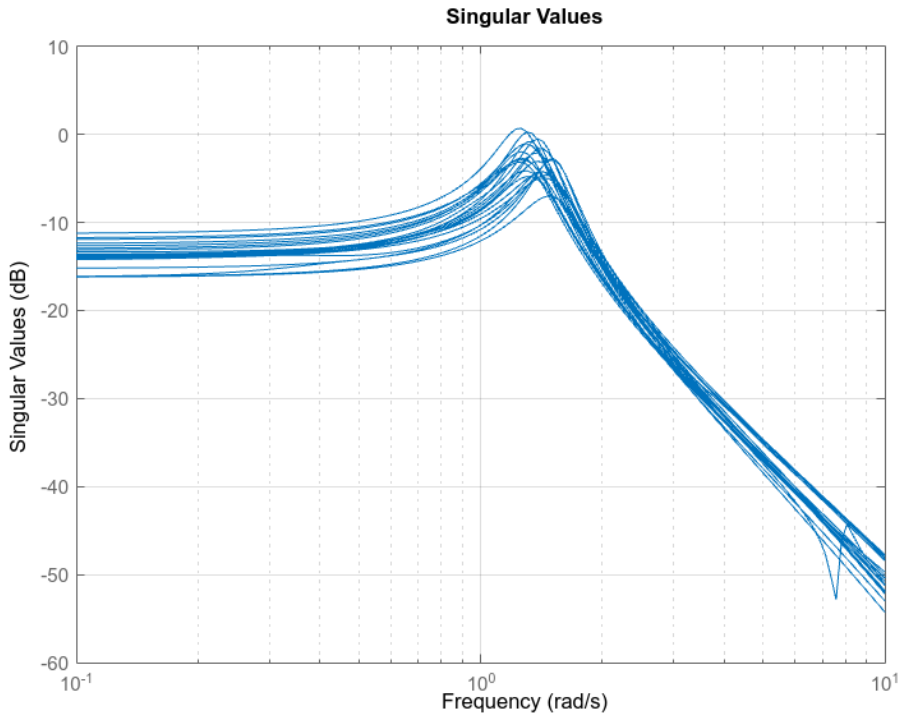
```
Pnominal=P.NominalValue;
zpk(Pnominal)
```

```
ans =

           3.2
  ----------------------
  (s+8) (s^2 + 0.4s + 2)

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

```
step(P), grid on
```




```
sigma(P,logspace(-1,1,150)), grid on
```

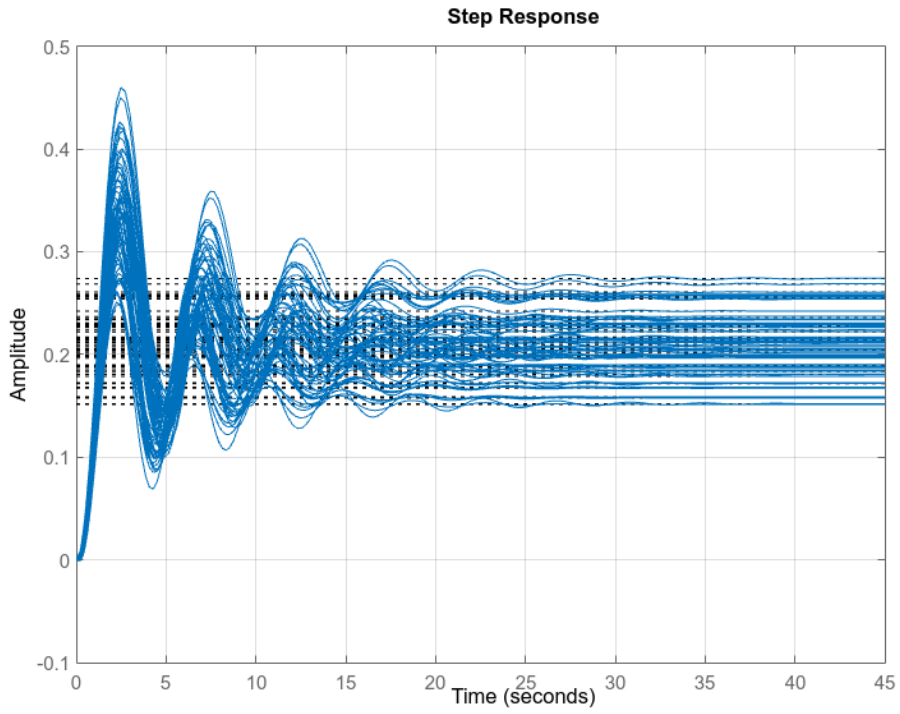



```
wcgain(P)
```

```
ans = struct with fields:
           LowerBound: 1.3041
           UpperBound: 1.3068
    CriticalFrequency: 1.2503
```

2.- Simplificación de la incertidumbre a un único Delta

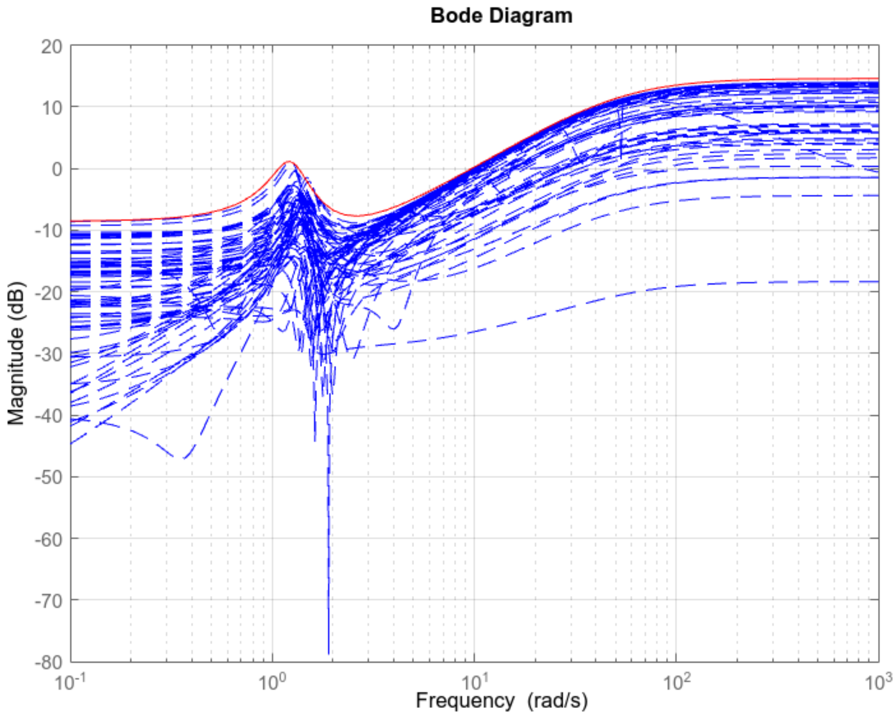
```
rng('default');% the random number generator is seeded for repeatability
NumSistemasPrueba=70;
muchosP=usample(P,NumSistemasPrueba);
step(muchosP), grid on
```




```
mPf=frd(muchosP,logspace(-1,3,60)); %frequency-response data model
[Pr,Info]=ucover(mPf,Pnominal,3);
Pr.Uncertainty
```

```
ans = struct with fields:
    mPf_InputMultDelta: [1×1 ultidyn]
```

```
relerr = (Pnominal-muchosP)/Pnominal;

bodemag(relerr,'b--',Info.W1,'r',{0.1,1000}); grid on
```



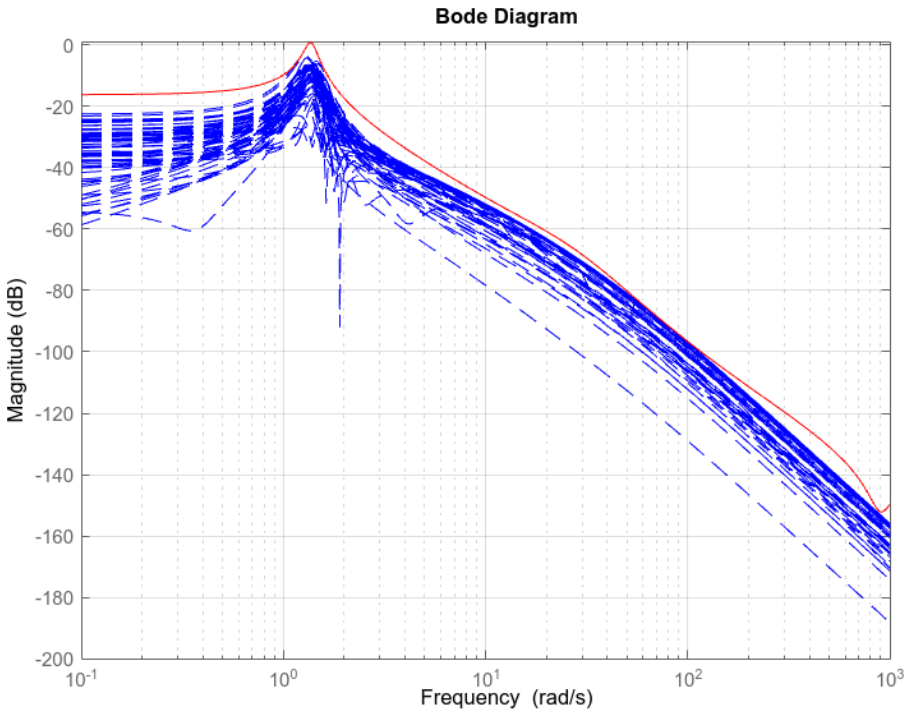

```
TamanyoincertMultiplicativa=Info.W1;

[Pra,Infoa]=ucover(mPf,Pnominal,4,'Additive');
```

```
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND =
NaN.
```

```
Pra.Uncertainty
```

```
ans = struct with fields:
    mPf_AddDelta: [1×1 ultidyn]
```

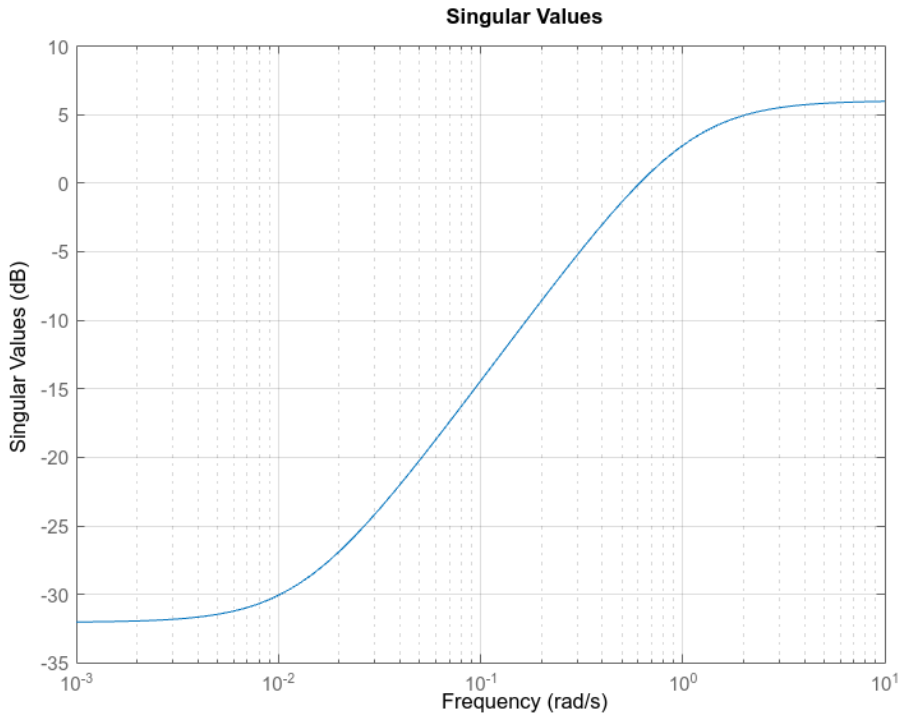
```
adderr=Pnominal-muchosP;
bodemag(adderr,'b--',Infoa.W1,'r',{0.1,1000});, grid on
```




```
TamanyoincertAditiva=Infoa.W1;
```

3.1- Diseño Hinfinito mixed sensitivity (sólo garantiza estabilidad robusta)

```
%queremos que el control tenga un ancho de banda objetivo, lo más rápido...
w_objetivo=0.61;
Werr=makeweight(40,w_objetivo,0.5);
sigma(Werr,[],1), grid on %inversa de Werr es plantilla de error (debe estar por debajo
```




```
Wu=1/30; %que control no sature a 30 ante referencia 1.

%opción incertidumbre MULTIPLICATIVA
GPNW=[1 0;0 1;0 0;1 0]+[0 -1;0 0;0 1;0 -1]*P;
GPW=blkdiag(Werr,Wu,TamanyoincertMultiplicativa,1)*GPNW*blkdiag(1,1);
tic
[K,CL,GAM,INFO]=hinfsyn(minreal(GPW.NominalValue),1,1);%robust stab garantizada si GAM<
```

```
13 states removed.
```

```
toc
```

```
Elapsed time is 0.072338 seconds.
```

```
GAM
```

```
GAM = 0.7455
```

```
size(K)
```

```
State-space model with 1 outputs, 1 inputs, and 7 states.
```

```
CLinciertoHinfW=lft(GPW([1:2 4],:),K); %medida robust performance, lft con planta ponde
%eliminamos fila 3 (salida r->y mixed sensitivity, porque GPW ya tiene la incertidumbre
[stabmarg,wcu] = robstab(CLinciertoHinfW); %margen mayor de 1 --> estable. Mixed sens e
stabmarg
```

```
stabmarg = struct with fields:
```

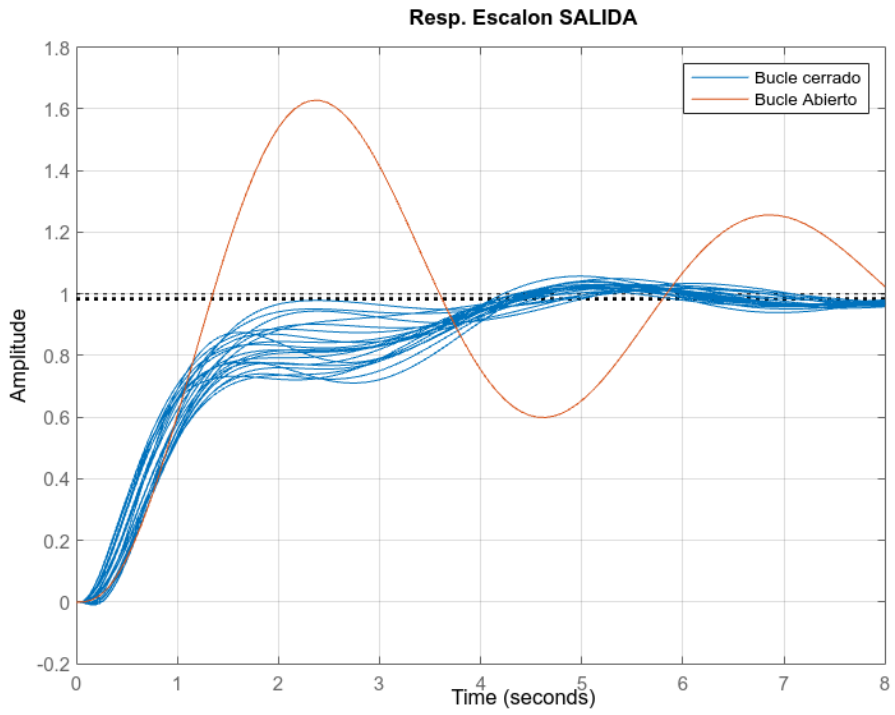
```
            LowerBound: 1.8933
            UpperBound: 2.4164
    CriticalFrequency: 1.2662
```

```
[peorganancia,peorincert]=wcgain(CLinciertoHinfW);%peor caso de norma >1 --> no hay per
peorganancia
```

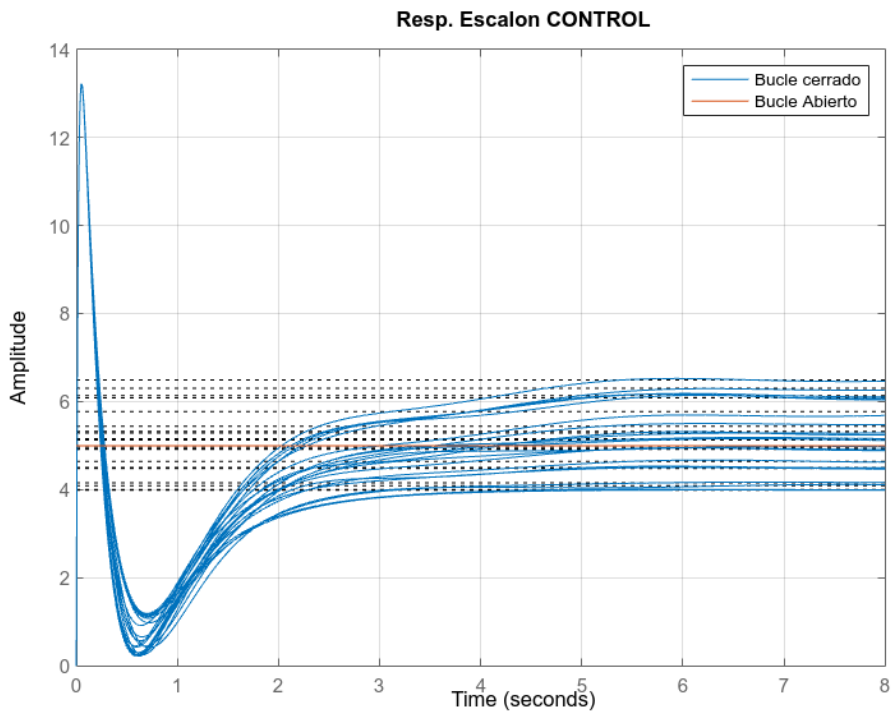
```
peorganancia = struct with fields:
            LowerBound: 0.9907
            UpperBound: 0.9929
    CriticalFrequency: 1.5255
```

Simulemos ahora el bucle diseñado.

```
CLinciertoHinf=feedback(P*K,1);%P es un modelo incierto
step(CLinciertoHinf,Pnominal/dcgain(Pnominal),8), grid on
title('Resp. Escalon SALIDA')
legend('Bucle cerrado','Bucle Abierto')
```




```
CL_u_incierto=feedback(K,P);
step(CL_u_incierto,tf(1/dcgain(Pnominal)), 8), grid on
title('Resp. Escalon CONTROL')
legend('Bucle cerrado','Bucle Abierto')
```



3.2- Diseño Hinfinito mixed sensitivity fijando estructura del regulador (PID)

```
w_objetivo=0.69;
Werr=makeweight(40,w_objetivo,0.5);

GPW2=blkdiag(Werr,Wu,TamanyoincertMultiplicativa,1)*GPNW*blkdiag(1,1);
%misma planta generalizada ponderada

tic
C0=tunablePID('ReguladorPID','pid')
```

```
C0 =
  Tunable continuous-time PID controller "ReguladorPID" with formula:

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

  and tunable parameters Kp, Ki, Kd, Tf.

Type "pid(C0)" to see the current value and "get(C0)" to see all properties.
```

```
TunableCL=lft(GPW2.NominalValue,C0);
opt = hinfstructOptions('Display','final','RandomStart',5);
[CL_PID,GAM,INFO] = hinfstruct(TunableCL,opt);
```

```
Final: Peak gain = 0.785, Iterations = 34
Final: Peak gain = 0.785, Iterations = 37
Final: Peak gain = 0.785, Iterations = 46
Final: Peak gain = 0.785, Iterations = 36
Final: Peak gain = 0.785, Iterations = 39
Final: Peak gain = 0.785, Iterations = 37
```

```
GAM
```

```
GAM = 0.7850
```

```
toc
```

```
Elapsed time is 0.367176 seconds.
```

```
PID_num=getValue(C0,CL_PID.Blocks);
tf(PID_num)
```

```
ans =

  18.15 s^2 + 9.29 s + 28.21
  --------------------------
        s^2 + 6.306 s

Name: ReguladorPID
Continuous-time transfer function.
```

```
CLinciertoPIDW=lft(GPW2([1:2 4],:),PID_num); %medida robust performance, lft con planta
[stabmarg,wcu] = robstab(CLinciertoPIDW); %margen mayor de 1 --> estable.
stabmarg
```

```
stabmarg = struct with fields:
           LowerBound: 1.8891
           UpperBound: 2.8475
    CriticalFrequency: 2.1466
```

```
[peorganancia,~]=wcgain(CLinciertoPIDW);
peorganancia
```

```
peorganancia = struct with fields:
           LowerBound: 0.9907
           UpperBound: 0.9926
    CriticalFrequency: 1.6317
```

Pasamos a simular el bucle conseguido.

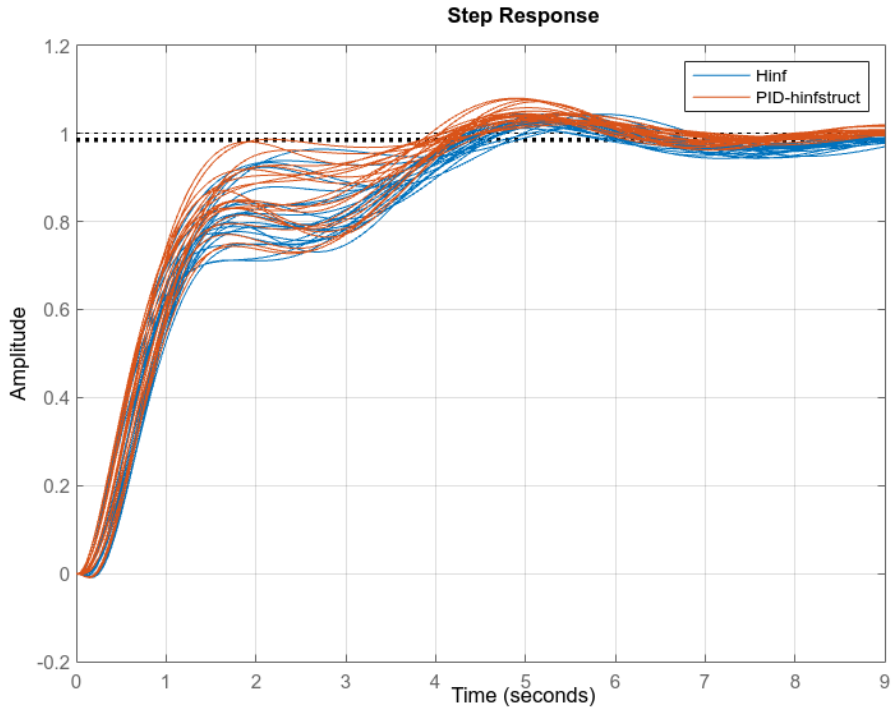
```
PIDfinal=getBlockValue(CL_PID,'ReguladorPID')
```

```
PIDfinal =

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

  with Kp = 0.764, Ki = 4.47, Kd = 2.76, Tf = 0.159

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

```
tf(PID_num)
```

```
ans =

  18.15 s^2 + 9.29 s + 28.21
  --------------------------
        s^2 + 6.306 s

Name: ReguladorPID
Continuous-time transfer function.
```

```
CLinciertoPID=feedback(P*PID_num,1);
step(CLinciertoHinf,CLinciertoPID,9), grid on, legend("Hinf","PID-hinfstruct")
```



4.1- Diseño iterativo multiplicador-controlador (mu-síntesis)

```
w_objetivo=0.68;
Werr=makeweight(40,w_objetivo,0.5);
GPNW=[1 0;0 1;1 0]+[0 -1;0 0;0 -1]*P;%No ponemos peso de incert. mulitiplicativa, ya es
GPNW.InputName={'ref','u'};
GPNW.OutputName={'err','u_arbitro','err_copia'};
GPW=blkdiag(Werr,Wu,1)*GPNW*blkdiag(1,1);
GPW.Uncertainty
```

```
ans = struct with fields:
         deltaact: [1×1 ultidyn]
    famortiguador: [1×1 ureal]
          kmuelle: [1×1 ureal]
```

```
if(1) %esto tarda un monton y se raya... al menos en versiones antiguas (2017) de Matla
tic
opt = musynOptions('MixedMu','on');
[Kmu_nored,wcGAM]=musyn(GPW,1,1,opt); %no necesito peso deltamultiplicativa, ya está to
toc
wcGAM
size(Kmu_nored)
wcgain(lft(GPW,Kmu_nored))
end
```

```
DG-K ITERATION SUMMARY:
-----------------------------------------------------------------------
                    Robust performance                 Fit order
-----------------------------------------------------------------------
  Iter         K Step       Peak MU       DG Fit           D      G
    1          5.302        2.091        2.145           56     36
    2          1.888        1.583         1.79           60     34
    3          1.661        1.394        1.453           60     34
    4          1.251        1.183        1.217           50     40
    5          1.122        1.075        1.112           60     34
    6          1.087        1.046        1.211           66     32
    7          1.024       0.9984        1.107           58     32
    8          1.055        1.025        1.264           64     34
    9          1.006       0.9918        1.091           64     32

Best achieved robust performance: 0.992
Elapsed time is 100.097026 seconds.
wcGAM = 0.9918
State-space model with 1 outputs, 1 inputs, and 107 states.
ans = struct with fields:
           LowerBound: 0.9898
           UpperBound: 0.9918
    CriticalFrequency: 0.2788
```

```
%con incertidumbre simplificada, más rápido en tiempo de cómputo
w_objetivo=0.65;
Werr=makeweight(40,w_objetivo,0.5);
```

Gastamos Pr en vez de P. No ponemos peso de incert. mulitiplicativa porque ya está en Pr y musyn debe estabilizarlo:

```
GPNWr=[1 0;0 1;1 0]+[0 -1;0 0;0 -1]*Pr;
GPWr=blkdiag(Werr,Wu,1)*GPNWr*blkdiag(1,1);
GPWr.Uncertainty
```

```
ans = struct with fields:
    mPf_InputMultDelta: [1×1 ultidyn]
```

```
tic
[Kmu,wcGAM]=musyn(GPWr,1,1);
```

```
D-K ITERATION SUMMARY:
-----------------------------------------------------------------
                    Robust performance              Fit order
```

```
-----------------------------------------------------------------
  Iter         K Step       Peak MU       D Fit            D
    1            5.1        2.338        2.358           12
    2          1.227        1.224        1.236           22
    3          1.044        1.041        1.053           22
    4           1.01        1.006        1.017           22
    5          1.004            1        1.007           22
    6          1.006        1.004        1.009           24

Best achieved robust performance: 1
```

```
toc
```

```
Elapsed time is 4.338912 seconds.
```

```
wcGAM %es un escalado conjunto de incertidumbre y planta,
```

```
wcGAM = 1.0001
```

```
% no es la medida directa de robust performance (sí se garantiza si wcGAM<1).
size(Kmu)
```

```
State-space model with 1 outputs, 1 inputs, and 28 states.
```

```
CLinciertoW3r=lft(GPWr,Kmu);%con la planta "simplificada" de una incertidumbre.
wcgain(CLinciertoW3r)
```

```
ans = struct with fields:
           LowerBound: 1.0002
           UpperBound: 1.0019
    CriticalFrequency: 0.3767
```

```
GPNW=[1 0;0 1;1 0]+[0 -1;0 0;0 -1]*P;
GPW=blkdiag(Werr,Wu,1)*GPNW*blkdiag(1,1);
CLinciertoW3=lft(GPW,Kmu);%con la planta "original" de muelles, etc... 3 bloques de inc
robstab(CLinciertoW3)
```

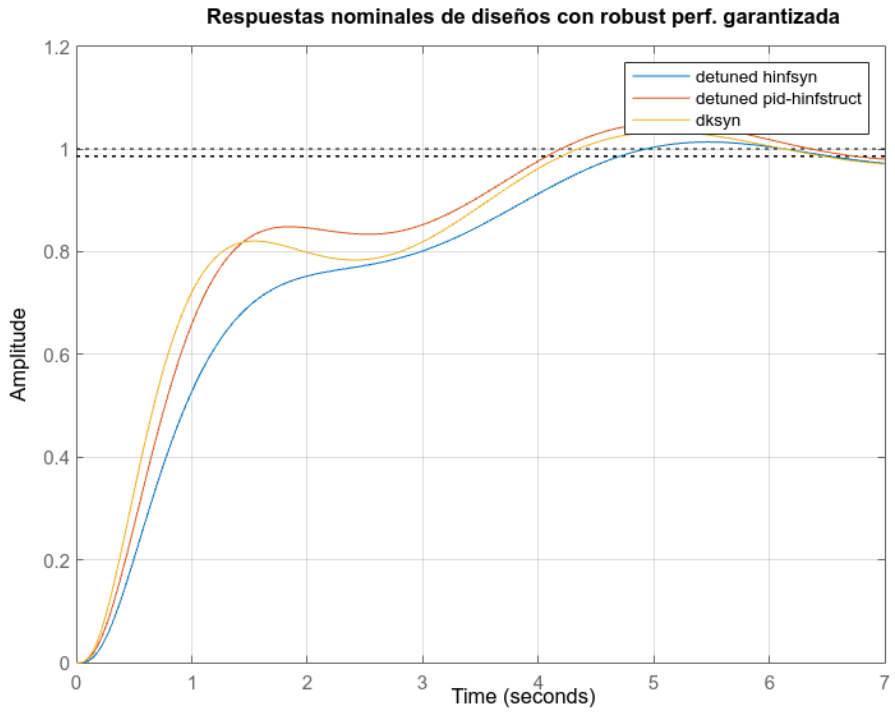
```
ans = struct with fields:
           LowerBound: 2.2853
           UpperBound: 2.9823
    CriticalFrequency: 2.2179
```

```
wcgain(CLinciertoW3)
```

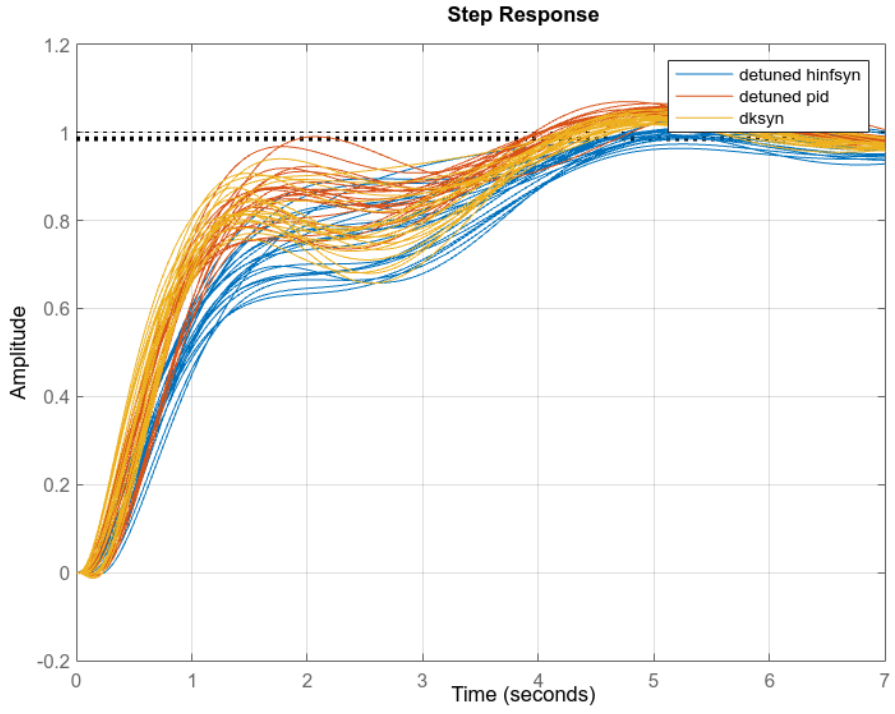
```
ans = struct with fields:
           LowerBound: 0.9922
           UpperBound: 0.9942
    CriticalFrequency: 12.2768
```

```

```

```
CLincierto3=feedback(P*Kmu,1);
step(CLinciertoHinf.NominalValue,CLinciertoPID.NominalValue,CLincierto3.NominalValue,7)
title('Respuestas nominales de diseños con robust perf. garantizada')
legend('detuned hinfsyn','detuned pid-hinfstruct','dksyn')
```




```
step(CLinciertoHinf,CLinciertoPID,CLincierto3,7), grid on
legend('detuned hinfsyn','detuned pid','musyn')
```



4.2- Diseño Musyn con PID estructurado

Dado que μ-síntesis es una metodología que requiere optimización no convexa (contrariamente a `hinfsyn`, que sí era optimización convexa), también su código puede incorporar plantas generalizadas + controlador con estructura prefijada

```
w_objetivo=0.82;
Werr=makeweight(40,w_objetivo,0.5);
GPW=blkdiag(Werr,Wu,1)*GPNW*blkdiag(1,1);
tic
TunableCL2=lft(GPW,C0); %la misma que para el ejemplo hinfstruc, y musyn
opt = musynOptions('MixedMu','on');
[CL,WcGAM2]=musyn(TunableCL2,opt)
```

```
DG-K ITERATION SUMMARY:
-----------------------------------------------------------------
                      Robust performance                 Fit order
-----------------------------------------------------------------
  Iter         K Step       Peak MU       DG Fit           D      G
    1          5.332        3.305        3.515           54     30
    2          3.173        2.387        2.565           54     30
    3          2.346        1.908        1.935           58     30
    4          1.597        1.566        1.601           64     32
    5          1.205        1.194        1.211           68     34
    6          1.097        1.082        1.144           58     32
    7          1.049        1.038         1.05           62     34
    8          1.039        1.027        1.039           66     34
    9         0.9972       0.9866       0.9975           68     34
   10         0.9911       0.9803            1           66     34

Best achieved robust performance: 0.98


CL =

  Generalized continuous-time state-space model with 2 outputs, 1 inputs, 19 states, and the following blo
    ReguladorPID: Tunable PID controller, 1 occurrences.
    deltaact: Uncertain 1x1 LTI, peak gain = 1, 2 occurrences
    famortiguador: Uncertain real, nominal = 1, variability = [-30,30]%, 2 occurrences
    kmuelle: Uncertain real, nominal = 5, variability = [-20,20]%, 2 occurrences

Type "ss(CL)" to see the current value, "get(CL)" to see all properties, and "CL.Blocks" to interact with
WcGAM2 = 0.9803
```

```
toc
```

```
Elapsed time is 94.073230 seconds.
```

```
wcgain(CL)
```

```
ans = struct with fields:
           LowerBound: 0.9803
           UpperBound: 0.9823
    CriticalFrequency: Inf
```

```
regefinal=pid(CL.Blocks.ReguladorPID)
```

```
regefinal =

            1           s
```

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

  with Kp = 0.771, Ki = 5.13, Kd = 3.14, Tf = 0.128

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

```
w_objetivo=0.6;
Werr=makeweight(40,w_objetivo,0.5);
GPW=blkdiag(Werr,Wu,1)*GPNWr*blkdiag(1,1); %con incertidumbre reducida
tic
TunableCL2=lft(GPW,C0); %la misma que para el ejemplo hinfstruc
opt = musynOptions('MixedMu','on');
[CLr,WcGAM2]=musyn(TunableCL2,opt)
```

```
D-K ITERATION SUMMARY:
-----------------------------------------------------------------
                      Robust performance               Fit order
-----------------------------------------------------------------
  Iter          K Step       Peak MU       D Fit             D
    1           5.149        4.521        4.624             8
    2            1.16          1.1        1.111            16
    3           1.021        1.018        1.027            24
    4           1.006       0.9987        1.006            20
    5           1.002       0.9942         1.01            20
    6           1.004       0.9968        1.005            20

Best achieved robust performance: 0.994


CLr =

  Generalized continuous-time state-space model with 2 outputs, 1 inputs, 25 states, and the following blc
    ReguladorPID: Tunable PID controller, 1 occurrences.
    mPf_InputMultDelta: Uncertain 1x1 LTI, peak gain = 1, 2 occurrences

Type "ss(CLr)" to see the current value, "get(CLr)" to see all properties, and "CLr.Blocks" to interact wi
WcGAM2 = 0.9942
```

```
toc
```

```
Elapsed time is 4.972441 seconds.
```

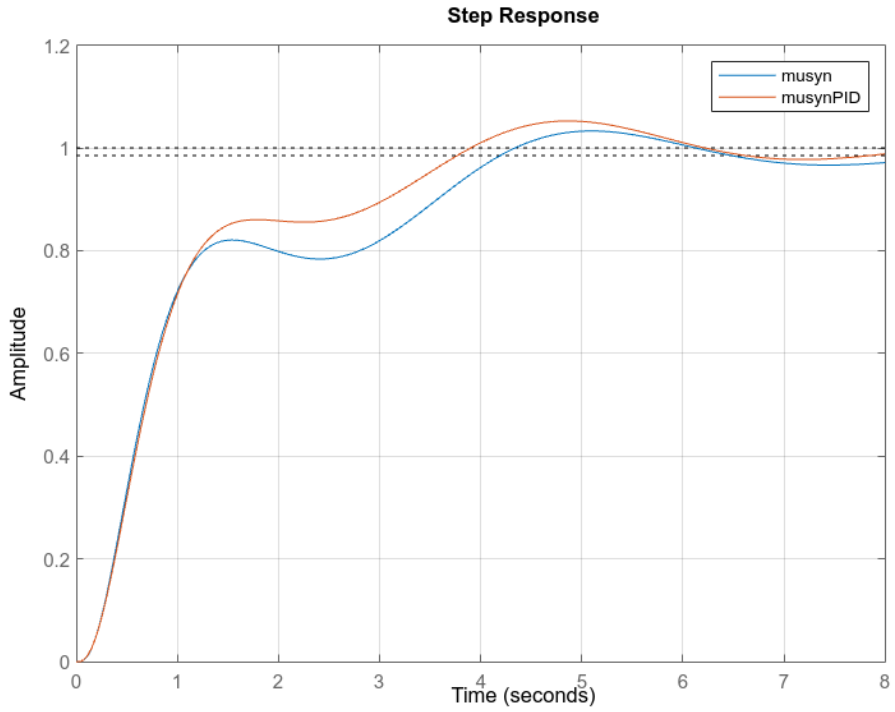
```
regefinal_r=pid(CLr.Blocks.ReguladorPID)
```

```
regefinal_r =

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

  with Kp = 1.04, Ki = 4.42, Kd = 2.99, Tf = 0.122

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

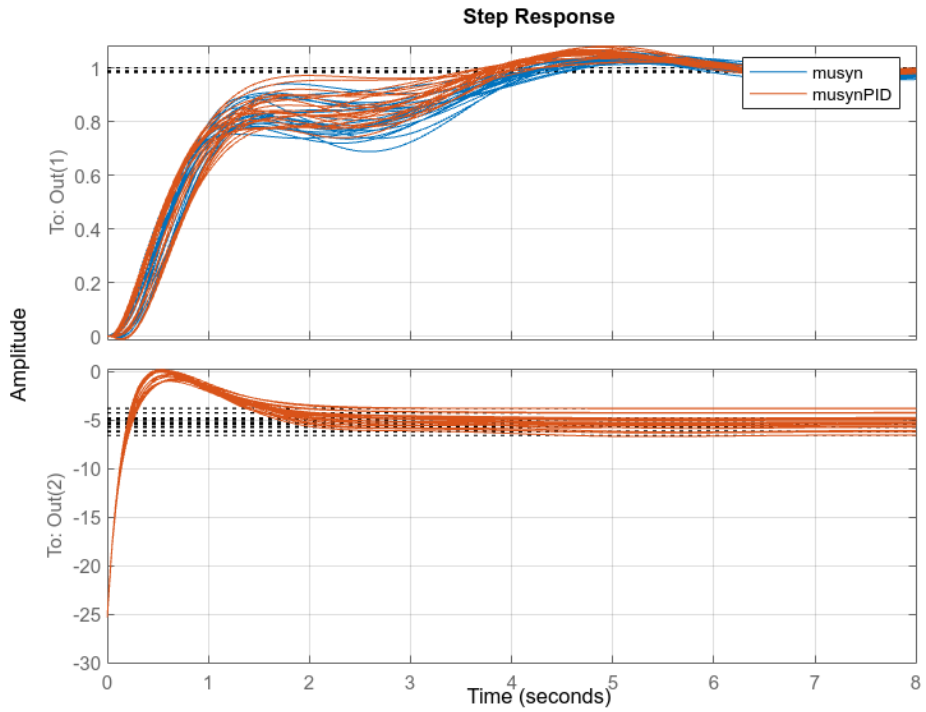
```
wcgain(CLr)
```

```
ans = struct with fields:
           LowerBound: 0.9911
           UpperBound: 0.9931
    CriticalFrequency: Inf
```

```
bcms=lft(GPNW,regefinal);
step(CLincierto3.Nominalvalue,1-bcms.NominalValue(1),8), grid on, legend('musyn','musyn
```




```
step(CLincierto3,[1;0]-bcms,8), grid on, legend('musyn','musynPID')
```




```
bodemag(1-CLincierto3,bcms,logspace(-2,2,200)), grid on
```

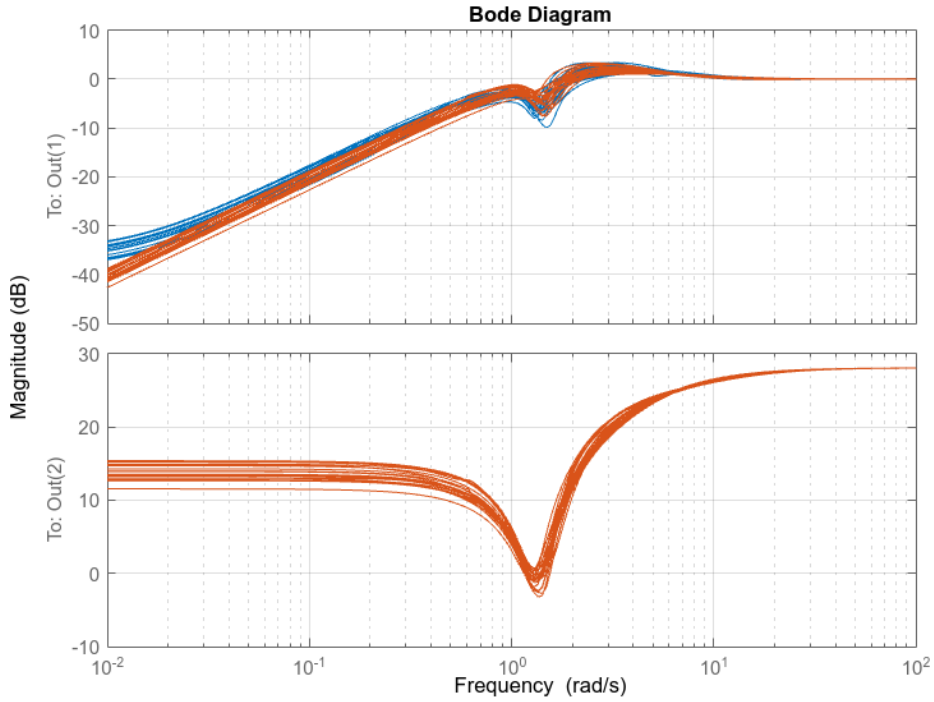


Tabla resumen de resultados con diferentes opciones

Resumen ancho banda para garantizar prestaciones robustas (`wcgain` ponderado < 1, incertidumbre no simplificada)

Detuned hinf: 0.43 [en video rcml3, Matlab 2018a] **0.61** [Matlab R2022b]

Detuned PID hinfstruct: 0.66 [en video rcml4, Matlab 2018a] **0.69** [Matlab R2022b]

Musyn (orden 107, incert. **completa**): **0.68**

Musyn (orden 24, incert. **simplificada**): **0.65**

Musyn PID (incert. **completa**): ***0.82***

Musyn PID (incert. **simplificada**): **0.6**

¿Diferencias entre versiones? Por corrección de errores o que haya tocado en planta o planta ponderada yo algo que no me acuerde.

Como musyn no es "optimización convexa" (mínimo único, eficiente de calcular), pues dependiendo de opciones, inicializaciones, etc. se llega a unos mínimos locales u otros.

Lo mejor ha acabado siendo el musyn PID (**0.82**). De todas formas, no hay ninguna diferencia "espectacular" en respuesta temporal. Subir mucho orden de multiplicadores parece que lía/baja condicionamiento numérico, todo es opciones por defecto, no hay cambios en orden de multiplicadores, tolerancias, etc.