# PID para sistema poco amortiguado: comparativa alternativas diseño



Presentaciones en vídeo:

http://personales.upv.es/asala/YT/V/mmimod.html , http://personales.upv.es/asala/YT/V/mmipid.html .

Este código funcionó correctamente con Matlab **R2021a**

**Objetivo:** sintonizar un PID teniendo en cuenta errores de modelado, respuesta en frecuencia ante perturbaciones, respuesta escalón, etc.

**Tabla de Contenidos**



## Modelo de proceso a controlar

```
p1=ureal('stiffness',0);
p2=ureal('damping',0);
p3=ureal('delay',1);
s=tf('s');
Ac=[0 1 ;-3 -0.2]; Bc=[0;3];
C=[1 0];
sysc_nominal=ss(Ac,Bc,C,0); %no incluiremos "retardo" en lo que vamos a suponer "nomina
tf(sysc_nominal)
```

```
ans =

        3
  ---------------
  s^2 + 0.2 s + 3

Continuous-time transfer function.
```

```
Ac=[0 1 ;-3+0.9*p1 -0.2+0.1*p2]; Bc=[0;3];
C=[1 -0.09*p3]; %p3 es "uncertain delay", puesto como un "cero"... hasta 0.18 retraso
sysc_novibra=ss(Ac,Bc,C,0);
sysc=ss(Ac,Bc,C,0)*700/(s^2+1.5*s+700) %para comprobar robustez: incierto + vibración a
```

```
sysc =

  Uncertain continuous-time state-space model with 1 outputs, 1 inputs, 4 states.
  The model uncertainty consists of the following blocks:
```

```
    damping: Uncertain real, nominal = 0, variability = [-1,1], 1 occurrences
    delay: Uncertain real, nominal = 1, variability = [-1,1], 1 occurrences
    stiffness: Uncertain real, nominal = 0, variability = [-1,1], 1 occurrences

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

```
gan=dcgain(sysc_nominal)
```

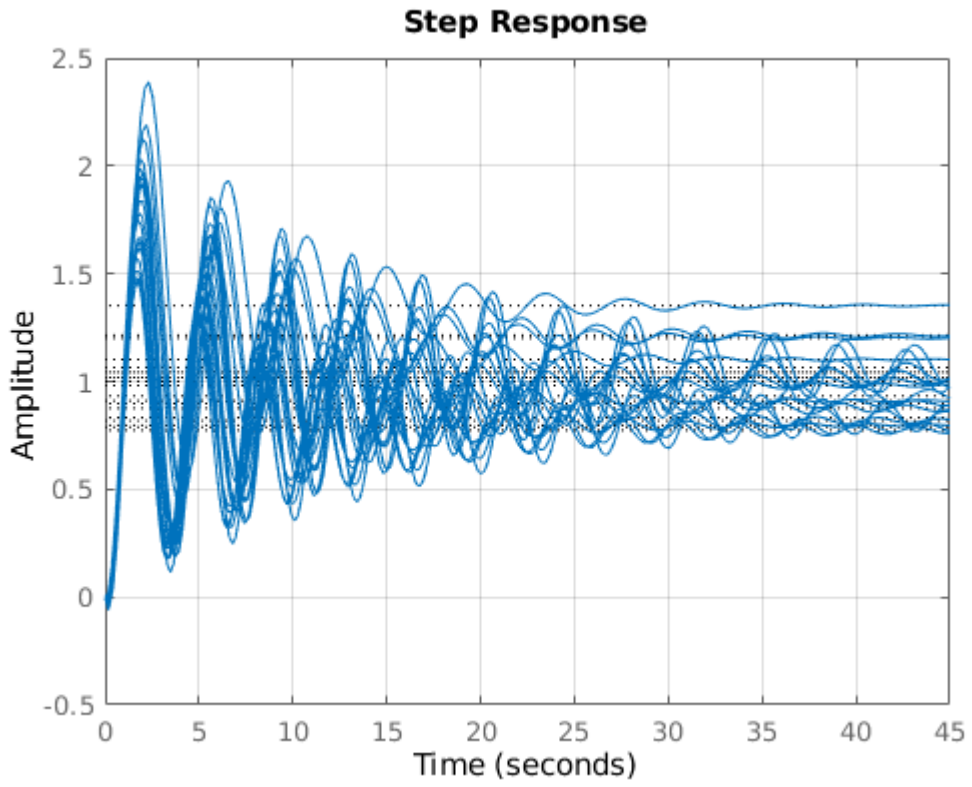
```
gan = 1
```

```
tf(sysc_nominal)
```

```
ans =

         3
  ---------------
  s^2 + 0.2 s + 3

Continuous-time transfer function.
```

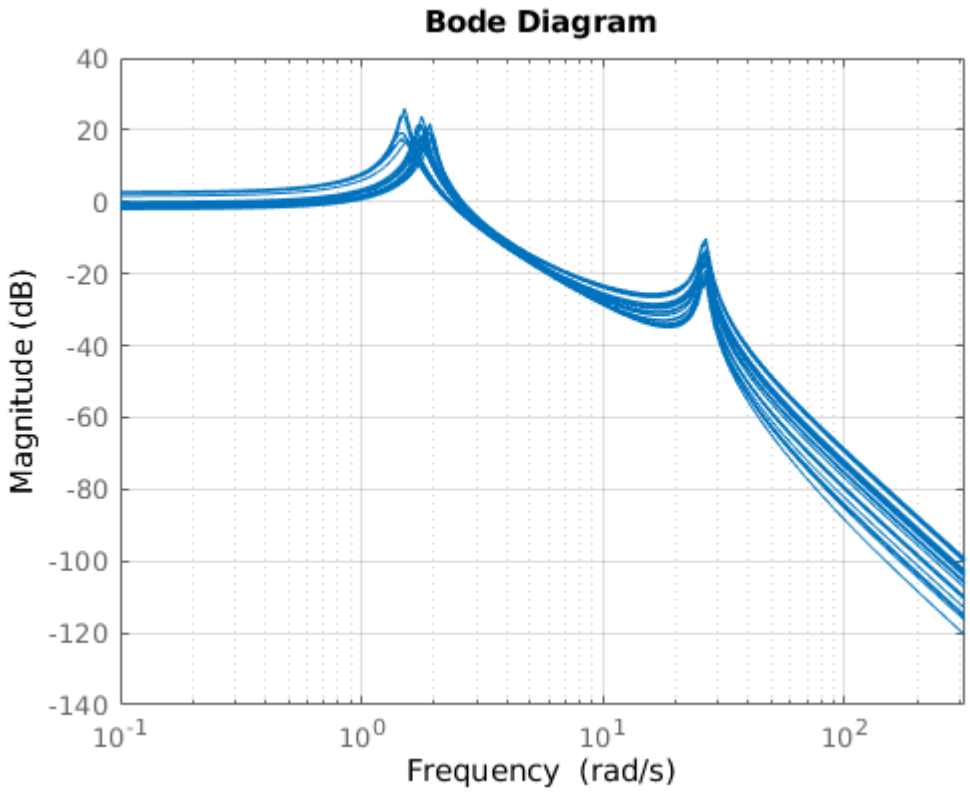
La respuesta en bucle abierto:

```
step(sysc,45), grid on
```

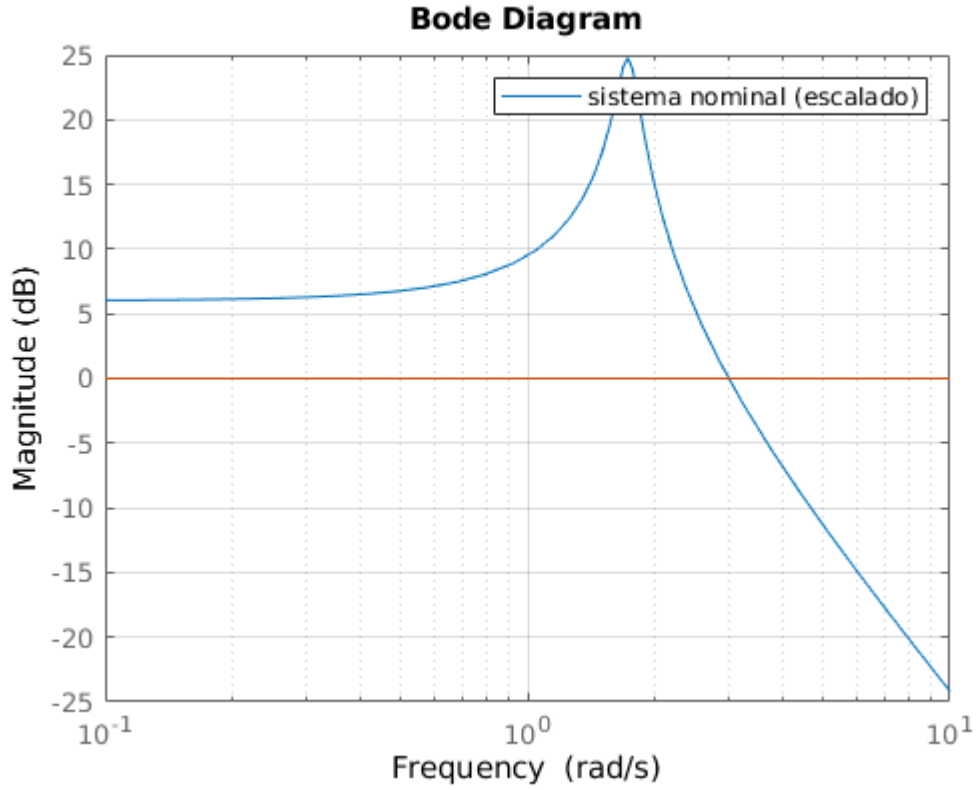



Respuesta en frecuencia, análisis controlabilidad entrada/salida:

```
bodemag(sysc,logspace(-1,2.5,200)), grid on
```

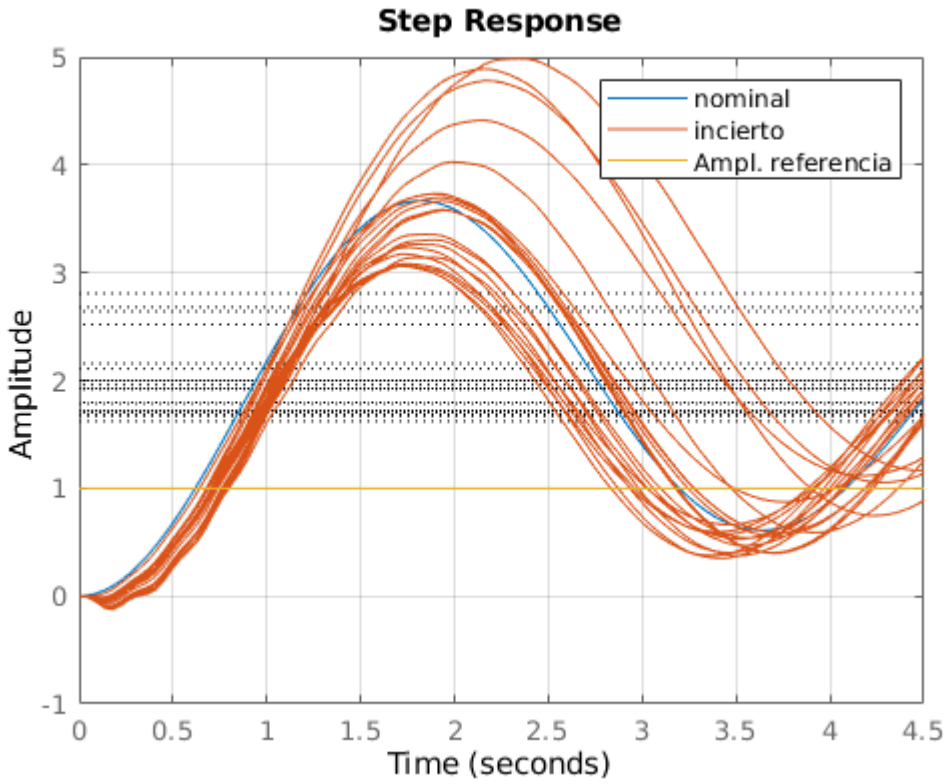



```
Gesc=sysc_nominal*2;
bodemag(Gesc,tf(1)), grid on, legend("sistema nominal (escalado)")
```



Con amplitud de entrada = 2, ancho de banda de aprox 3 rad/s... semiperíodo siguiendo senoides de pi/3... 1.1 segundos.

```
step(Gesc,sysc*2,tf(1),4.5), grid on, legend("nominal","incierto","Ampl. referencia")
```




Tiempo de subida 0.6-0.8 segundos, y luego lo que tardemos en "frenar"...

Bueno, pues un regulador que no sature deberá cumplir eso... y pensar que de natural frena en 40 segundos, y que no hay que amplificar mucho ruidos de medida y tener buenos márgenes de robustez...

```
bcanterior=[]; %para superponer con el diseño anterior al hacer pruebas
```

## Pruebas de sintonizado de un PID

```
%En el vídeo se hacen varias pruebas...
pid_test= 1; %Primera prueba... lugar raices ya nos dice que irá mal, pero por algo hay
pid_test = 1+s/(s/9+1); %Nominal amortigua mucho la dinámica, pero no funciona.
pid_test = 1+s/(s/9+1)+2.5/s; %Nominal "muy bueno", pero no funciona.
pid_test=inv(Gesc)*1/s/(s/6+1)*1.5; %cancelación (IMC-PID)
pid_test=pidtune(Gesc,'pidf',3) %continuamos con pidtune...
```

```
pid_test =

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

  with Kp = 0.579, Ki = 0.271, Kd = 0.302, Tf = 0.00292

Continuous-time PIDF controller in parallel form.
```

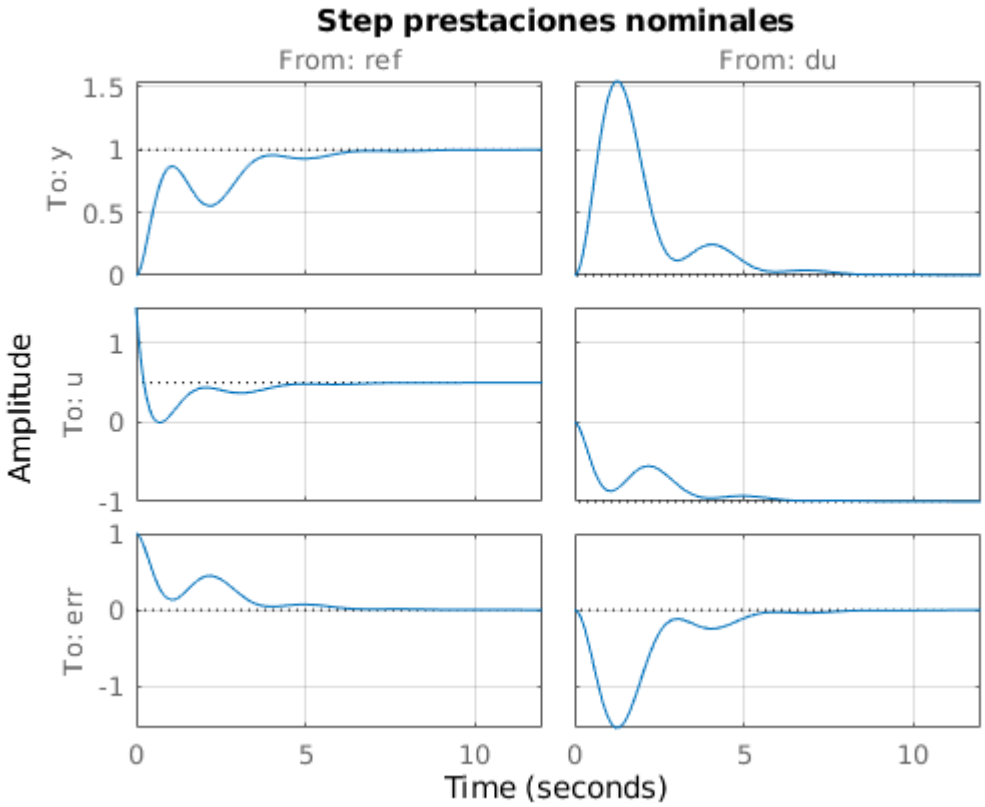
```
pid_test=0.2+0.35/s+0.25*s/(s/5+1); %tocamos "a ojo" filtrando más el de pidtune y baja
%esta es la que al final nos quedamos.

zpk(pid_test)
```

```
ans =

  1.45 (s^2 + 0.931s + 1.207)
  ---------------------------
            s (s+5)

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

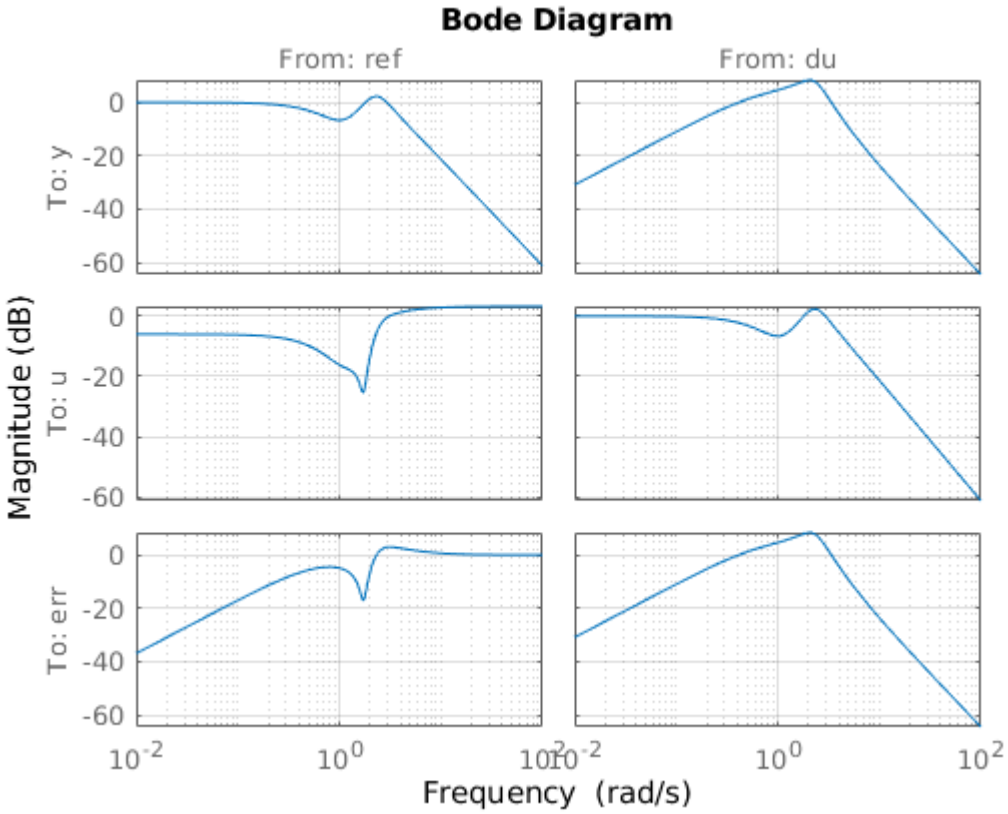
```
bc_nominal=minreal(cierrabucle(Gesc,pid_test));
pole(bc_nominal)
```

```
ans = 4×1 complex
  -3.1618 + 0.0000i
  -0.7104 + 2.2077i
  -0.7104 - 2.2077i
  -0.6174 + 0.0000i
```

```
step(bc_nominal,12), grid on, title("Step prestaciones nominales")
```




```
bodemag(bc_nominal), grid on
```



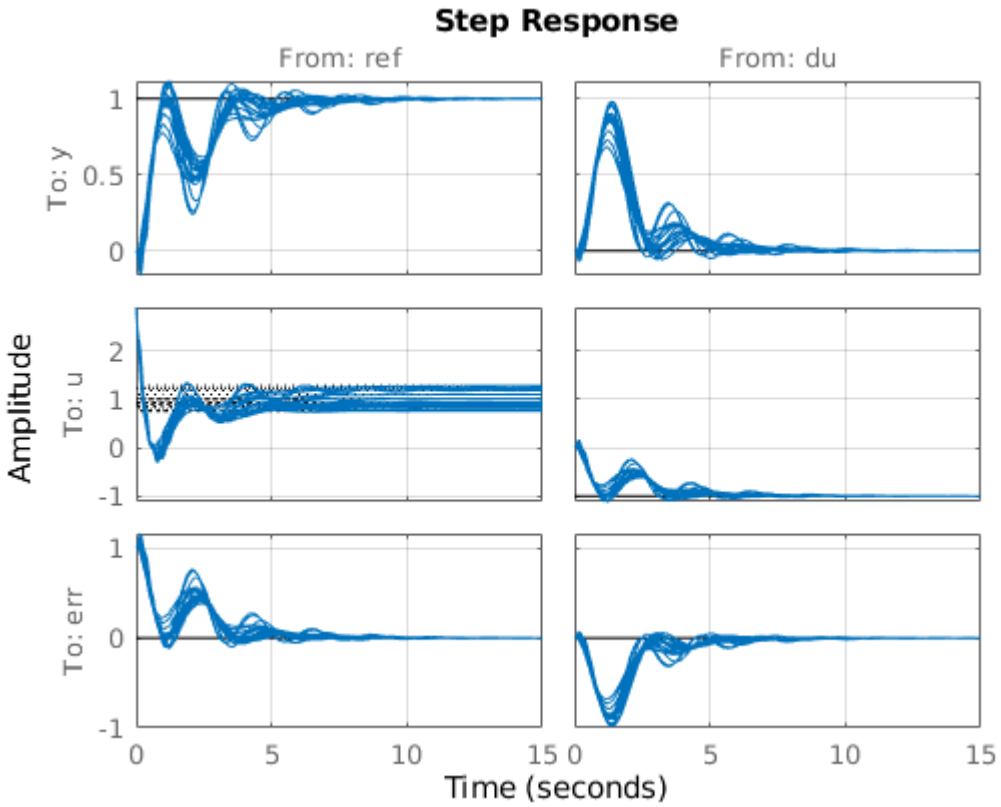
## Evaluación de prestaciones/robustez del diseño

```
bcpidfin=cierrabucle(sysc,2*pid_test) %deshacemos escalado para ver en unidades físicas
```

```
bcpidfin =

  Uncertain continuous-time state-space model with 3 outputs, 2 inputs, 6 states.
  The model uncertainty consists of the following blocks:
    damping: Uncertain real, nominal = 0, variability = [-1,1], 1 occurrences
    delay: Uncertain real, nominal = 1, variability = [-1,1], 1 occurrences
    stiffness: Uncertain real, nominal = 0, variability = [-1,1], 1 occurrences

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

Nada "mejor" para ver si hay estabilidad/prestaciones robustas que simular y ver qué pasa (bueno, los teoremas no engañan, pero tirar el dado puede no acertar con "lo peor"):

```
if isempty(bcanterior)
step(bcpidfin,15), grid on %el primer diseño sólo 1 gráfica.
else
step(bcanterior,bcpidfin,15), grid on, legend("anterior","nuevo")
end
```



### Márgenes de estabilidad

Estabilidad robusta márgenes ante incertidumbre estructurada (tma. pequeña ganancia escalado):

```
[mrg,wcu]=robstab(bcpidfin) %margen debe ser >1 para garantizar estabilidad de nuestro
```

```
mrg = struct with fields:
           LowerBound: 1.8157
           UpperBound: 1.8216
    CriticalFrequency: 3.2206
wcu = struct with fields:
      damping: 1.8191
        delay: 2.8216
    stiffness: -1.8100
```

Márgenes ante incertidumbre no estructurada quizás no contemplada en modelo ureal

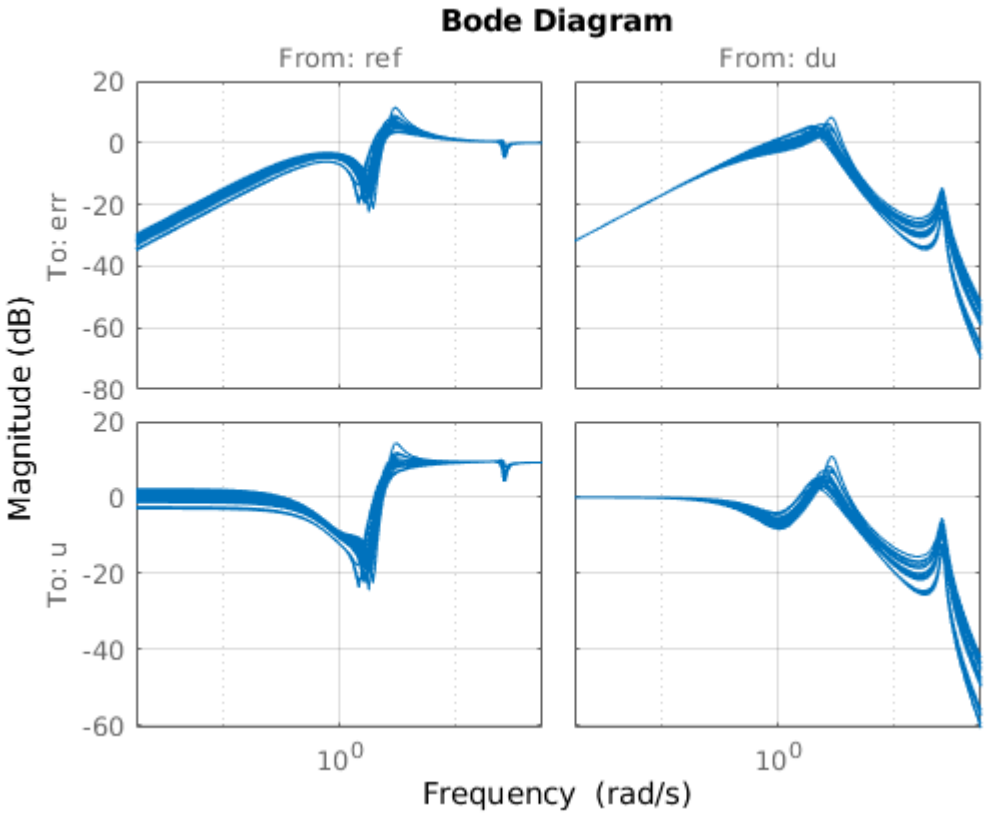
```
allmargin(Gesc*pid_test) %márgenes clásicos (1940-1950)
```

```
ans = struct with fields:
     GainMargin: Inf
    GMFrequency: Inf
    PhaseMargin: [118.2736 168.6920 47.1331]
    PMFrequency: [0.6627 1.1891 2.5468]
    DelayMargin: [3.1150 2.4759 0.3230]
    DMFrequency: [0.6627 1.1891 2.5468]
         Stable: 1
```

```
ncfmargin(Gesc,pid_test) %incertidumbre no estructurada en num. y denom. (factorizació
```

```
ans = 0.3294
```

```
bcanterior=bcpidfin;
bodemag(bcpidfin([3 2],:),logspace(-1.75,1.75,200)), grid on
```




## Conclusiones

Hay muchos requisitos contrapuestos en un sistema de control: rechazo de perturbaciones en entrada, seguimiento de refs (pert. salida), amortiguación y estabilización de la dinámica, no amplificar ruido de alta frecuencia, evitar picos de resonancia excesivos... y, obviamente, en fase de proyecto, intentar asegurar que funcione ante un previsible error de modelado. No es suficiente con un "step" ante referencia de un modelo nominal.

```
syms G real
[0 0 0;0 0 1;1 0 0;1 0 0] + [1;0;-1;-1]*G*[0 1 1]
```

```
ans =
```

$$\begin{pmatrix} 0 & G & G \\ 0 & 0 & 1 \\ 1 & -G & -G \\ 1 & -G & -G \end{pmatrix}$$

## Funciones auxiliares (cerrar bucle)

```
function bc=cierrabucle(planta,contr)
PG=minreal([0 0 0;0 0 1;1 0 0;1 0 0] + [1;0;-1;-1]*planta*[0 1 1]);
%realmente no hace falta "minreal" en este caso (planta sale 1 vez, si planta ya fuera
PG.InputName={'ref','du','u'};PG.OutputName={'y','u','err','err_to_controller'};
```

```
bc=lft(PG,contr);
end
```