# Prestaciones robustas, caso de estudio Matlab 5/(s+1)^2



Este código ejecutó sin errores en Matlab **R2022a**

Presentaciones en vídeo:

http://personales.upv.es/asala/YT/V/cerp1.html http://personales.upv.es/asala/YT/V/cerp2.html http://personales.upv.es/asala/YT/V/cerp3.html .

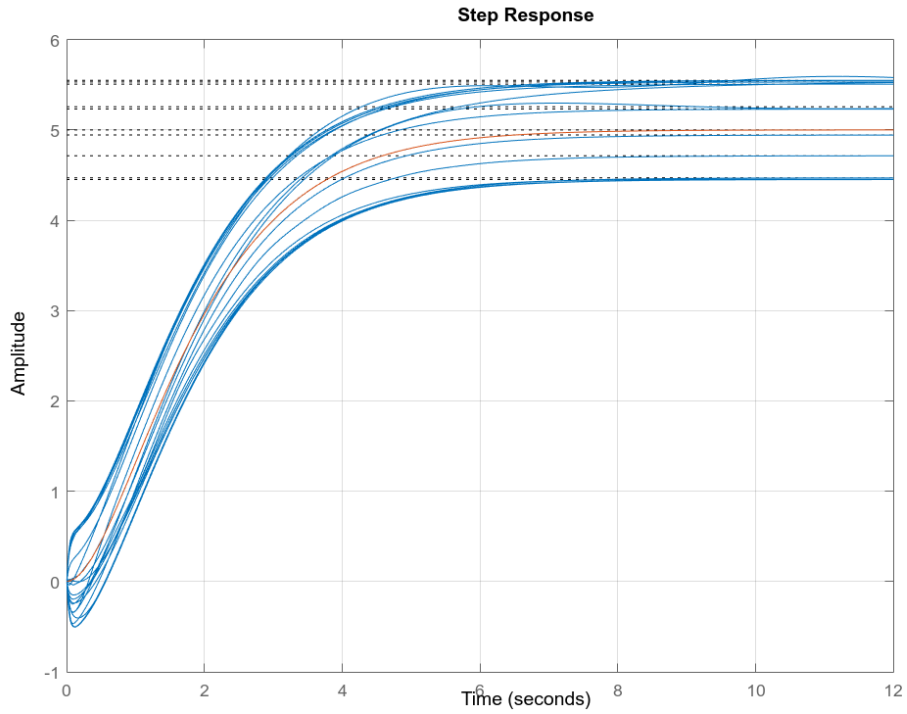
**Objetivos:** Comprender las ideas subyacentes a la teoría sobre prestaciones robustas y pequeña ganancia escalado con un ejemplo monovariable de segundo orden.

**Tabla de Contenidos**

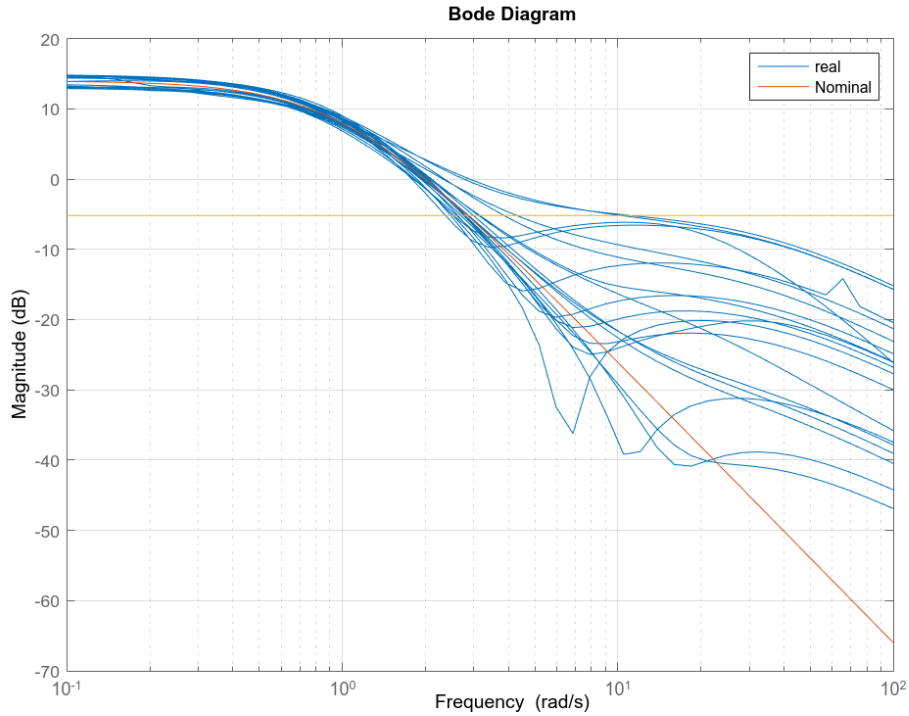


## Modelo de planta a controlar

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

BoundDelta=0.55; %tamaño error modelado "Delta" aditivo no estructurado
Greal=G+BoundDelta*ultidyn("DeltaEscalado")*1/(0.03*s+1);
%añado "matar" Greal a partir de 33 rad/s. No usado en análisis.

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




```
bodemag(Greal,logspace(-1,2)), grid on, hold on
bodemag(G,tf(BoundDelta),logspace(-1,2)), hold off, legend("real", "Nominal")
```

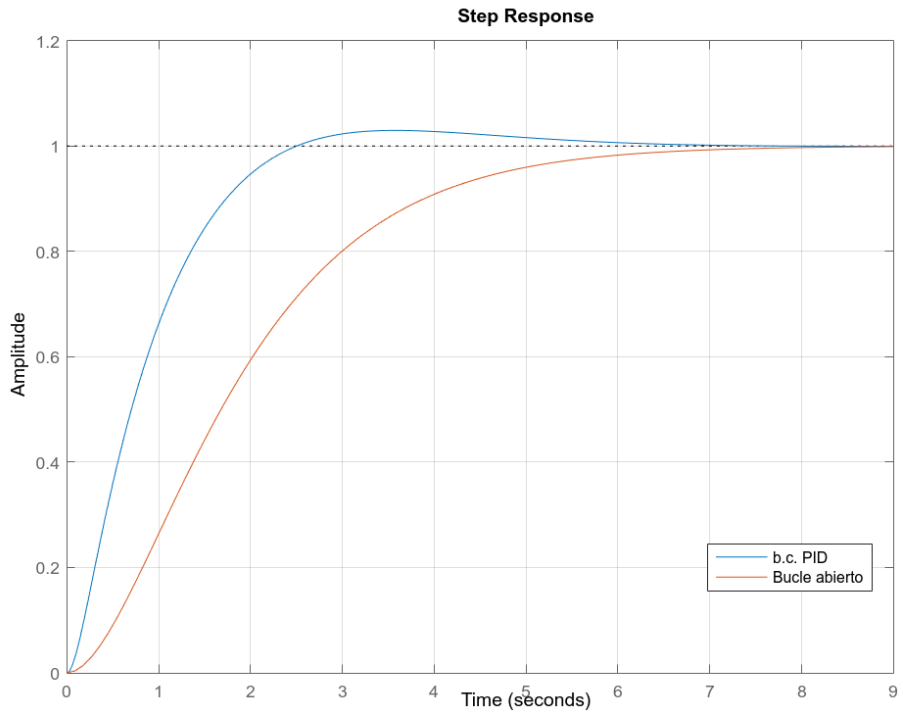


## Análisis de un controlador ya diseñado

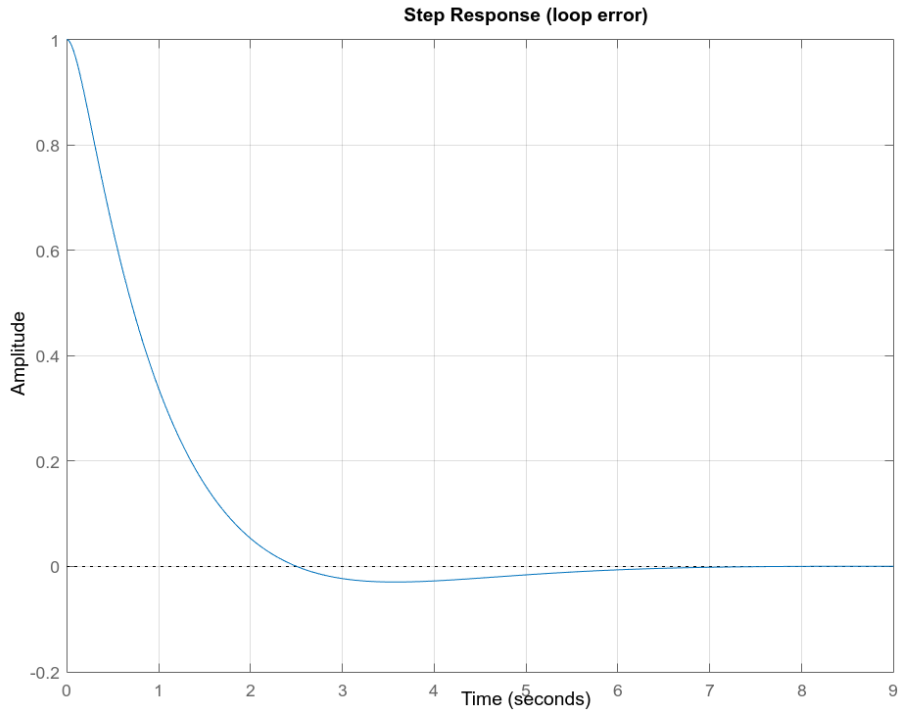
```
K=0.4*(1+0.65/s+0.4*s/(0.1*s+1));
%PID, sintonizado a mano, por "prueba y error"
```

### Prestaciones nominales

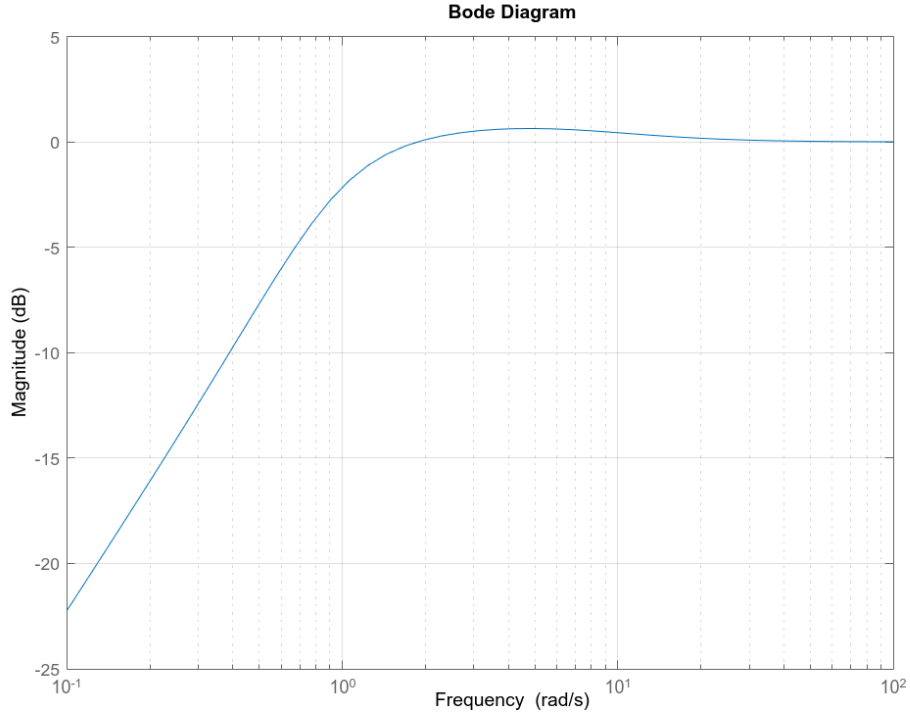
```
BC=feedback(G*K,1); %ref->output
step(BC,G/dcgain(G),9), grid on, legend("b.c. PID","Bucle abierto",Location="best")
```




```
BCerr=feedback(1,G*K); %ref->err
step(BCerr,9), grid on, title("Step Response (loop error)")
```

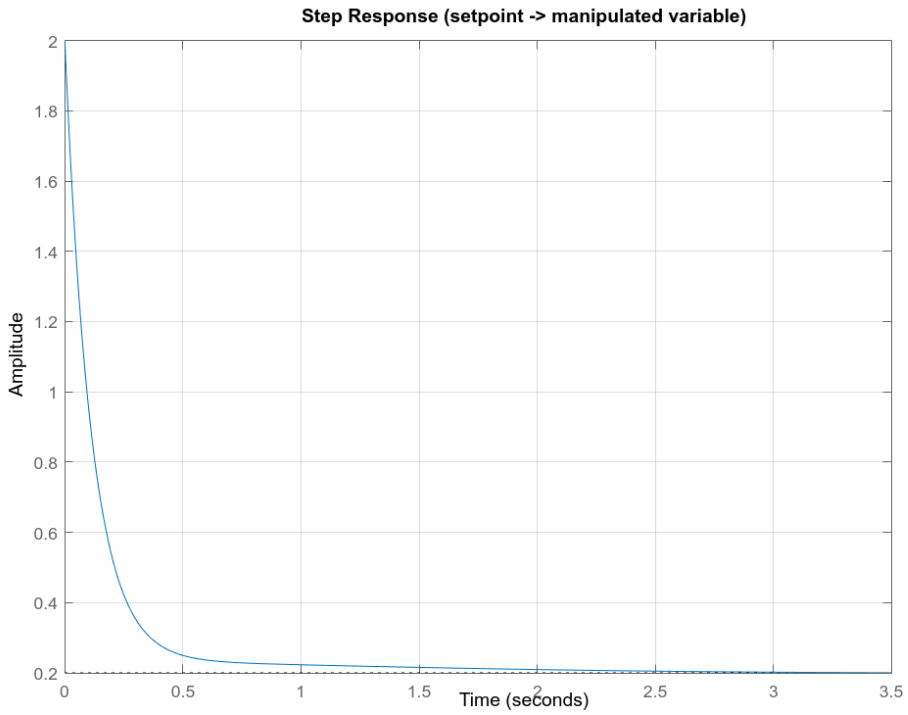



```
bodemag(BCerr), grid on
```

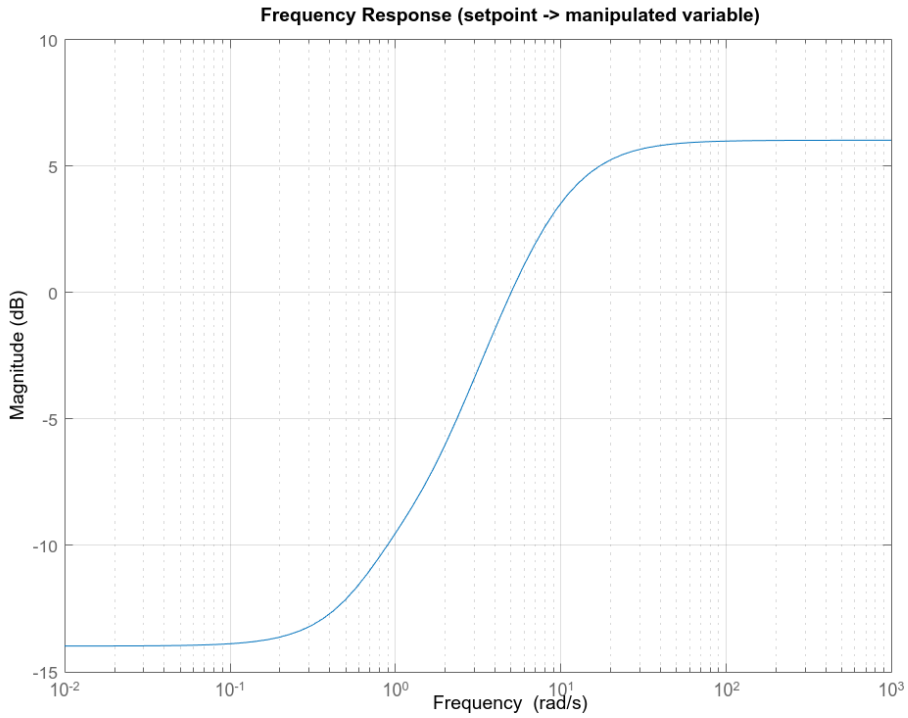



```
BCu=feedback(K,G);
```

```
step(BCu), grid on, title("Step Response (setpoint -> manipulated variable)")
```




```
bodemag(BCu), grid on, title("Frequency Response (setpoint -> manipulated variable)")
```



Nos parecen satisfactorias en el "tiempo", sobre la planta nominal. Podrían no serlo si tuviéramos problemas de saturación de acción de control o de excesiva amplificación de perturbaciones o ruido de medida a alguna frecuencia, pero las damos por buenas.

Obviamente, analizar prestaciones robustas podría resumirse simplemente a "simular" tirando el dado con Greal y comprobar que no hay nada raro... de hecho, lo haremos al final.

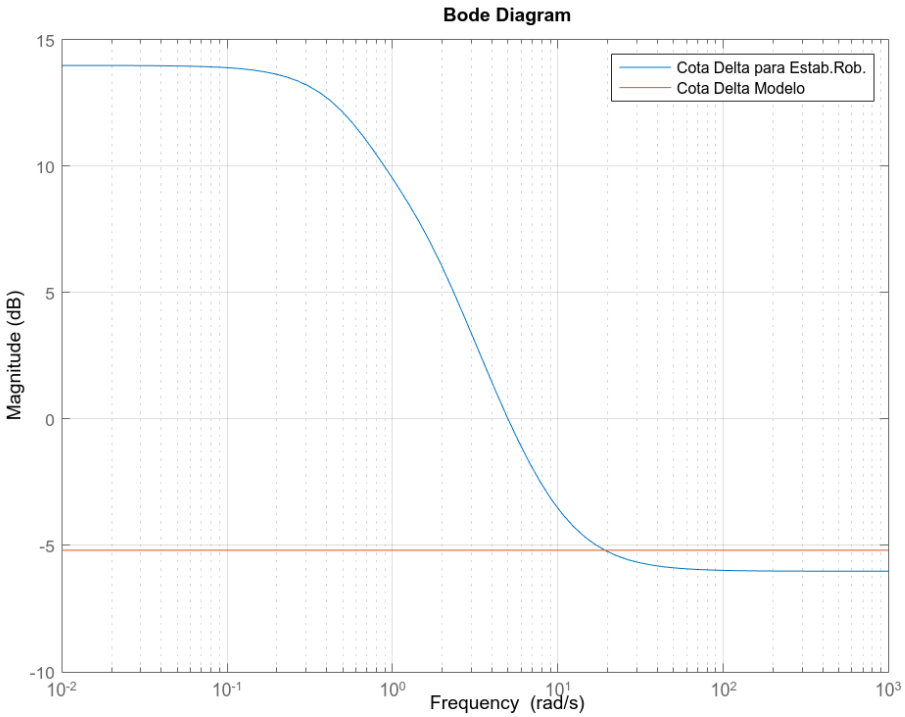
Pero nuestro objetivo es entender la justificación teórica de los márgenes, para garantizar que no vamos a tener "mala suerte" y alguna tirada de dado pudiera salir mal.

Aparte, la teoría justificará metodologías de diseño si este PID no tuviera prestaciones satisfactorias.

### Estabilidad robusta

Ante incertidumbre aditiva no estructurada.

```
cosaDeltaadd=K/(1+G*K); %input->error for small gain stability margins.
bodemag(1/cosaDeltaadd,tf(BoundDelta)),grid on, legend("Cota Delta para Estab.Rob.","Co
```




Conseguido Estabilidad Robusta. El BC no será inestable si el error experimental está por debajo de la cota propuesta (y realmente tenemos más márgen).

```
cotamaxDeltaAddEstRob=1/norm(cosaDeltaadd,inf)
```

```
cotamaxDeltaAddEstRob = 0.5000
```

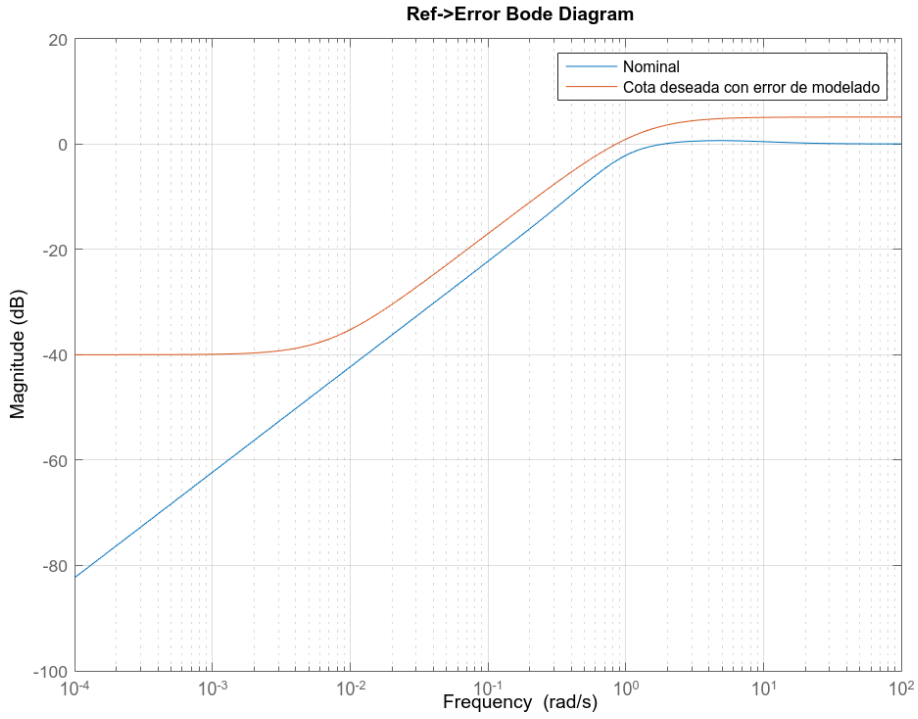
Podemos crecer el tamaño de $\Delta$, manteniendo estabilidad, en un factor de:

```
cotamaxDeltaAddEstRob/BoundDelta
```

```
ans = 0.9091
```

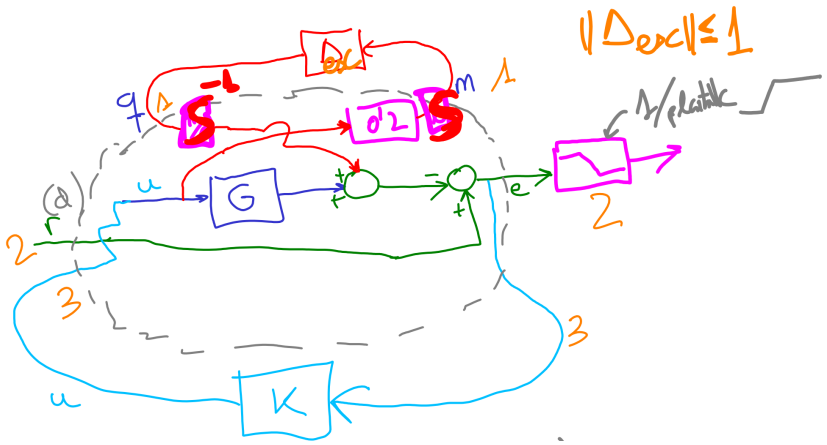
## Cota deseada de prestaciones robustas (en frecuencia)

```
bodemag(BCerr), grid on, hold on
TargetBandwidth=0.85; %aprox. inversamente prop. tiempo de subida
templateErr=makeweight(0.01,TargetBandwidth,1.8);
bodemag(templateErr), hold off, legend("Nominal","Cota deseada con error de modelado")
title("Ref->Error Bode Diagram")
```




**Nota:** por simplicidad no limitamos la acción de control por tema "prestaciones" (saturación, etc.), aunque en aplicaciones podría ser recomendable. De todos modos, la robustez ante incertidumbre aditiva se consigue limitando "u", con lo que implícitamente sí estamos en cierto modo limitando "u".

## Análisis de Prestaciones Robustas




```
PlantaGen=minreal(ss([0 0 1;-1 1 -G;-1 1 -G]));
S=1.6;
Win=blkdiag(1/S,1,1);
Wout=minreal(blkdiag(S*BoundDelta,1/templateErr,1));
```

```
1 state removed.
```

```
PlantaGenPond=Wout*PlantaGen*Win;
```

Con el PID tendríamos:

```
BCPond=lft(PlantaGenPond,K); norm(BCPond,inf)
```

```
ans = 2.1764
```

## Diseño H-infinito para prestaciones robustas

Aunque, claro, minimizar la norma infinito lo haríamos con:

```
[Khinf,Cl,GAM]=hinfsyn(PlantaGenPond,1,1); GAM
```

```
GAM = 0.9827
```

Sería una opción "teóricamente mejor"... pero no es un PID...

```
zpk(Khinf)
```

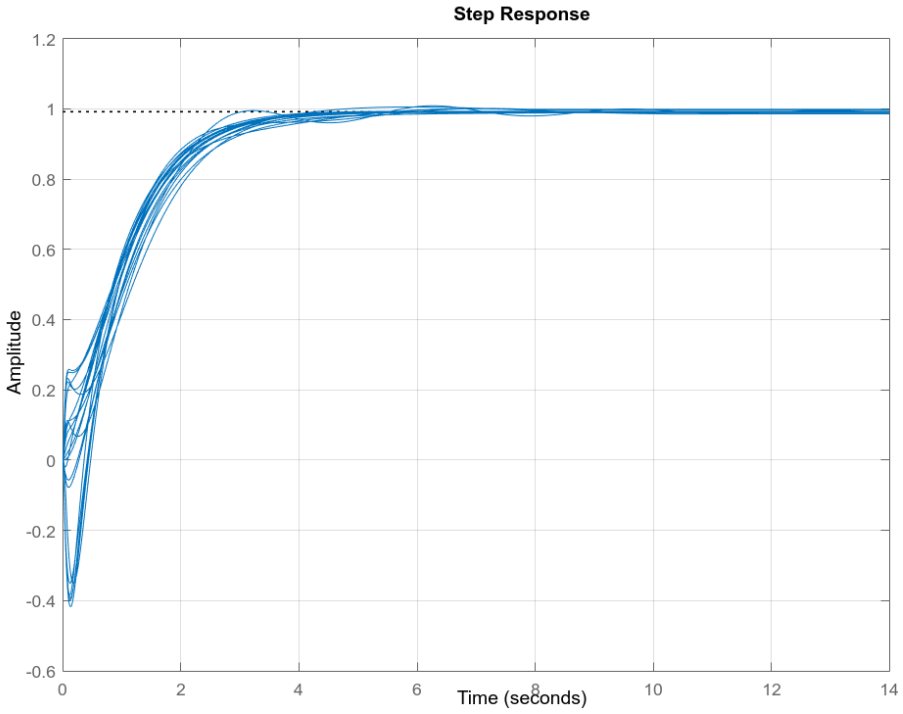
```
ans =

           43.186 (s+1)^2
  --------------------------------
  (s+4.111) (s+61.05) (s+0.007068)

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

***NOTA 1:** nótese la "cancelación" de la planta que podría dar lugar a problemas ante perturbaciones a la entrada y requisitos de especificaciones rápidas ante ellas (pero como no están en la planta generalizada, el optimizador no se preocupa de ello).

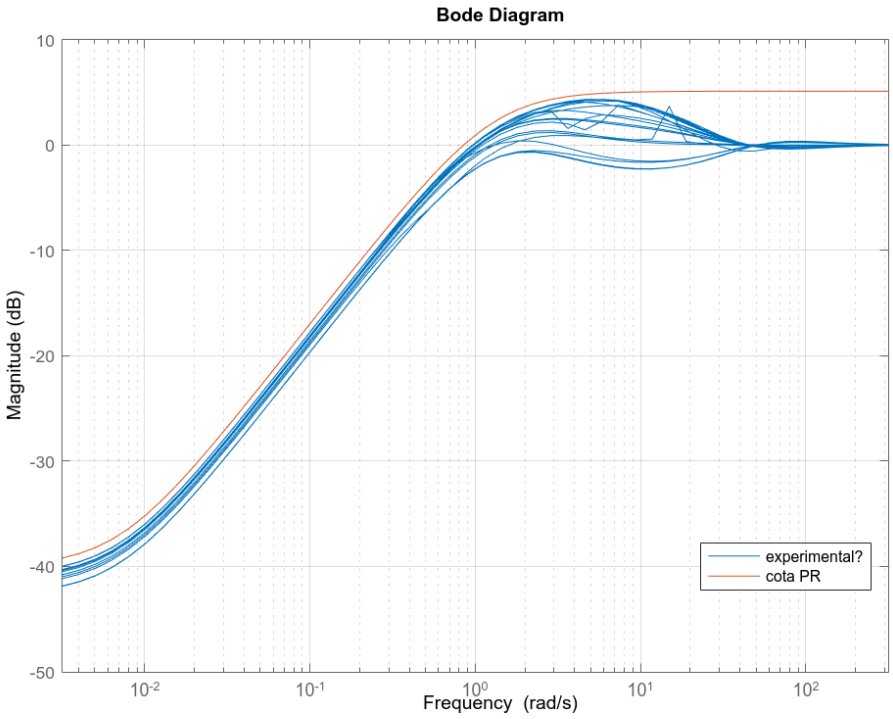
***NOTA 2:** buscar por "prueba y error" el multiplicador "S" es extremadamente ineficiente; podríamos hacer un bucle, hacer "fminsearch" o, incluso, para controlador fijo, buscar el multiplicador que minimiza la norma infinito es un problema convexo, fácilmente resoluble con ciertas técnicas (p.ej., LMI). Fuera de los objetivos de este ejemplo introductorio.

## Simulación prest. robustas en tiempo y frecuencia

```
K=Khinf;
step(feedback(Greal*K,1)), grid on
```

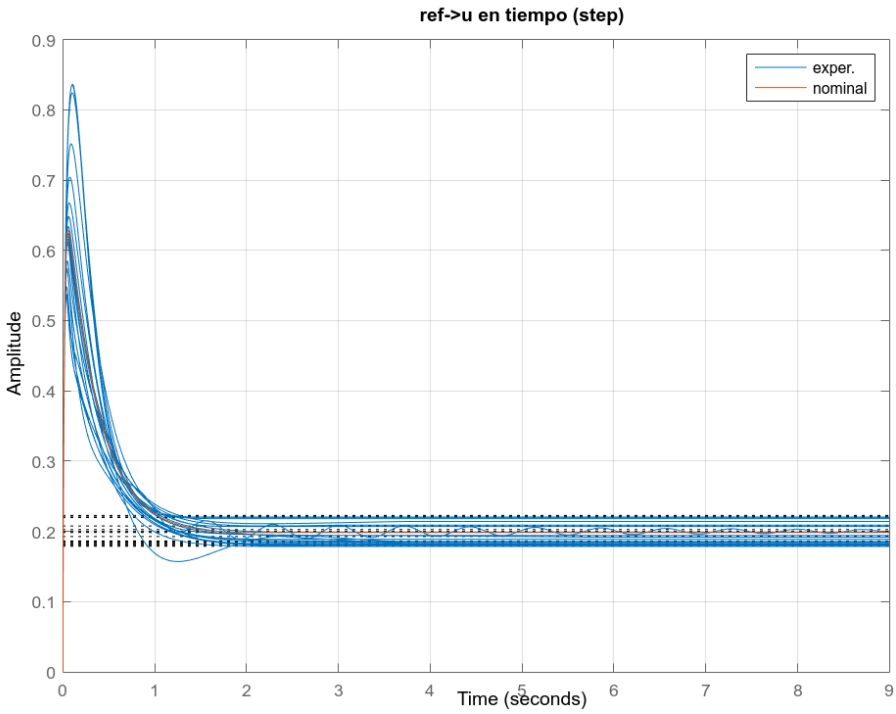



```
bodemag(feedback(1,Greal*K),templateErr,logspace(-2.5,2.5)), grid on, legend("experimer
```




Todo control "final" a un cliente debe tener una acción de control "razonable". No lo hemos limitado teóricamente, pero vamos a ver cuál es:

```
step(feedback(K,Greal),feedback(K,G)), grid on
title("ref->u en tiempo (step)"),legend("exper.","nominal")
```




```
bodemag(feedback(K,Greal),feedback(K,G),logspace(-2.5,2.5))
grid on, title("ref->u en frecuencia"),legend("exper.","nominal")
```