Controlador H-infinito discreto para proceso de 2º orden: implementación y antiwindup

Presentaciones en vídeo:

http://personales.upv.es/asala/YT/V/hiimpl1.html , http://personales.upv.es/asala/YT/V/hiimpl2.html .

Este código funcionó correctamente con Matlab **R2021b**

Objetivo: Diseñar un control $\mathscr{H}_\infty$ en tiempo discreto para rechazo de perturbaciones en entrada y seguimiento de referencias (con ciertos pesos).

Discutir su implementación (control por computador), y la estrategia antiwindup.

Table of Contents

Modelo

```
Ac=[0 1 ;-3 -0.2]; Bc=[0;3];
C=[1 0];
sysc=ss(Ac,Bc,C,0);
gan=dcgain(sysc)
```

```
gan = 1
```

```
tf(sysc)
```

```
ans =

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

Continuous-time transfer function.
```

```
limiteSatU=1;
```

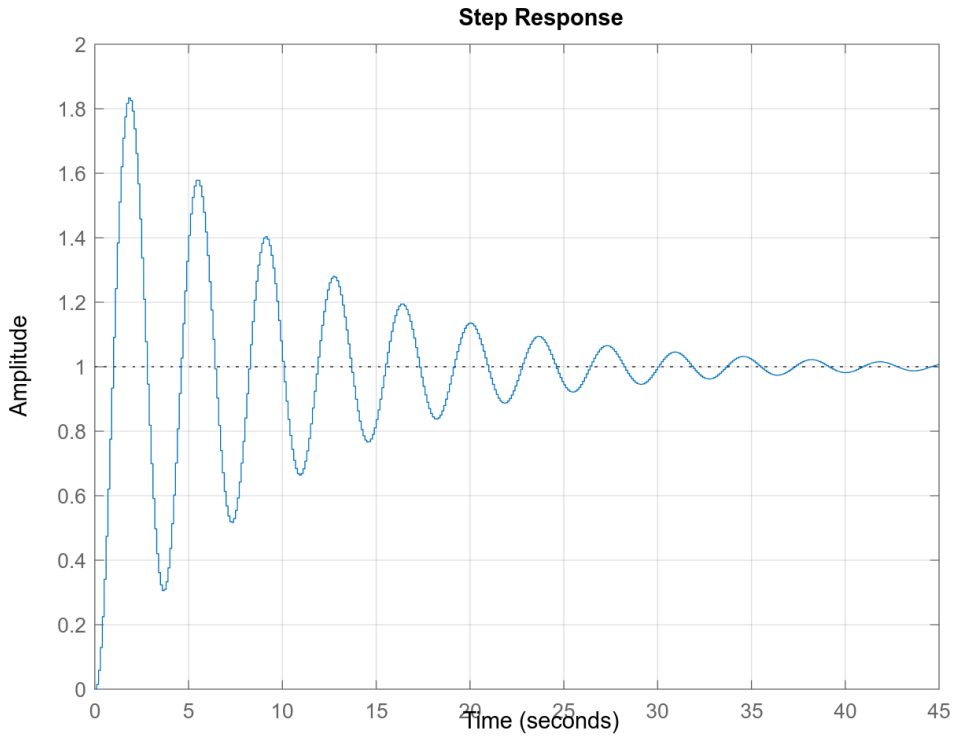
Vamos a discretizarlo:

```
Ts=0.1;
sysd=c2d(sysc,Ts,'zoh');
```

```
tf(sysd)
```

```
ans =

   0.01486 z + 0.01476
  ----------------------
  z^2 - 1.951 z + 0.9802

Sample time: 0.1 seconds
Discrete-time transfer function.
```

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



Control H-infinito (mixed sensitivity + pert entrada)

Planta Generalizada

El diseño discreto lo basaremos en la planta generalizada:

$$err = ref - G(d_u + u)$$

$$\begin{pmatrix} err \\ u \\ \dots \\ err_{copy} \end{pmatrix} = \begin{pmatrix} 1 & -G & -G \\ 0 & 0 & 1 \\ \dots & \dots & \dots \\ 1 & -G & -G \end{pmatrix} \cdot \begin{pmatrix} ref \\ d_u \\ \dots \\ u \end{pmatrix}$$

```
PGbuilder=@(G) [[1;0;1] [0 0;0 1;0 0]-[1;0;1]*G*[1 1]];
PGbuilder(sym('G'))
```

```
ans =
```

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

```
PG=PGbuilder(sysc); %luego discretizaremos
PG.INputName={'ref','du','u'};PG.OutputName={'err','u','err_copy'};
size(PG) %no hace falta "minreal" porque PGbuilder evita multiincidencia de G
```

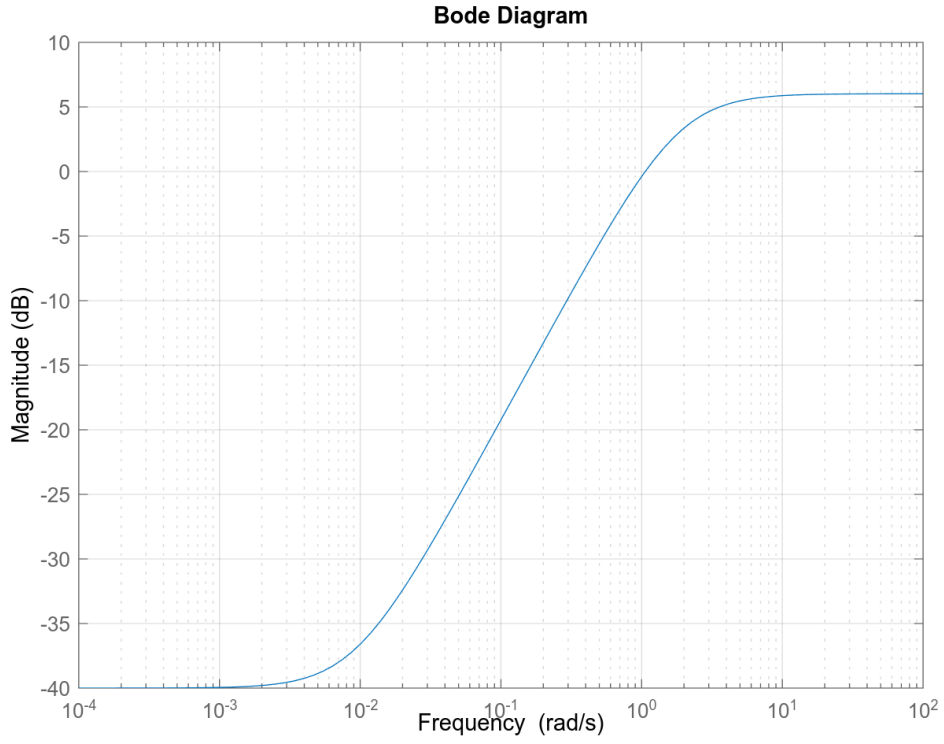
```
State-space model with 3 outputs, 3 inputs, and 2 states.
```

Planta Generalizada Ponderada y control Hinf discreto

```
bw=1.06;
PlantillaError=makeweight(0.01,bw,2);
ErrorWeight=minreal(1/PlantillaError);
```

```
1 state removed.
```

```
bodemag(PlantillaError), grid on
```




```
InputWeight=1/(limiteSatU*2); %con referencia "1/2" no saturará, con más... sí!
Wdu=0.4; %tamaño perturbación a la entrada
PGP=minreal(blkdiag(ErrorWeight,InputWeight,1)*PG)*blkdiag(1,Wdu,1);
```

Discretizamos la planta generalizada (y la planta ponderada):

```
PGd=c2d(PG,Ts,'zoh');
%perturbaciones no son "zoh"... ¿Tustin? bueno, si Ts "pequeño" son sutilezas poco rele
PGPd=c2d(PGP,Ts,'zoh');
[Kd,~,GAM]=hinfsyn(PGPd,1,1);GAM
```

```
GAM = 0.9962
```

```
eig(Kd)
```

```
ans = 3×1
    0.0188
    0.9991
    0.6951
```

Prestaciones sin considerar saturación (lft)

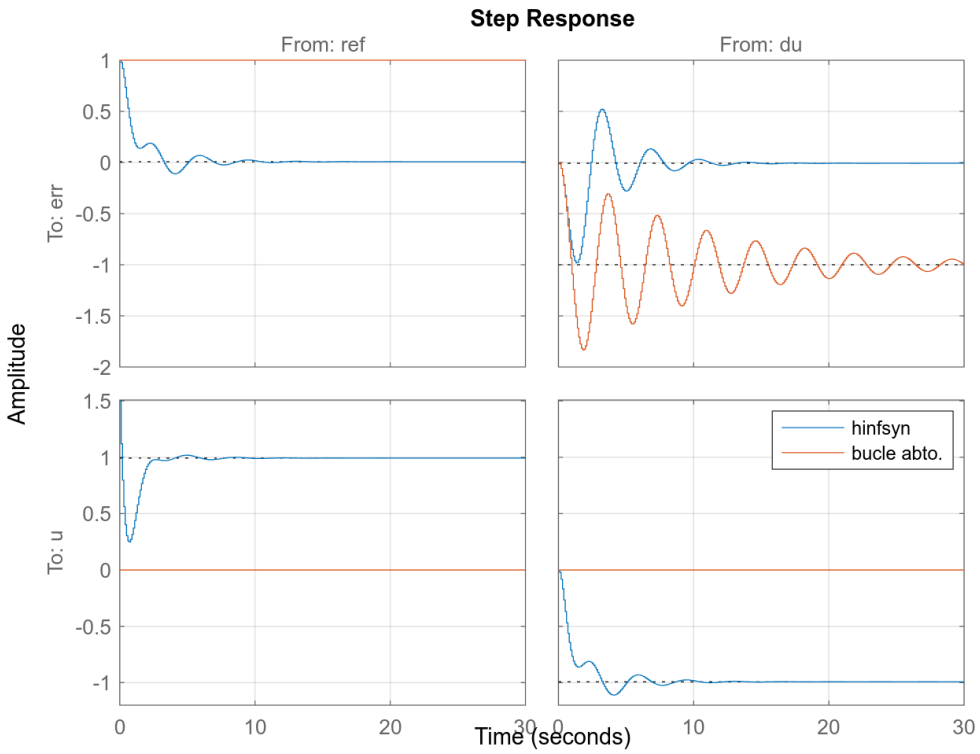
```
BC=lft(PGd,Kd);
eig(BC)
```

```
ans = 5×1 complex
   0.0182 + 0.0000i
   0.9490 + 0.1694i
   0.9490 - 0.1694i
   0.8625 + 0.1157i
   0.8625 - 0.1157i
```

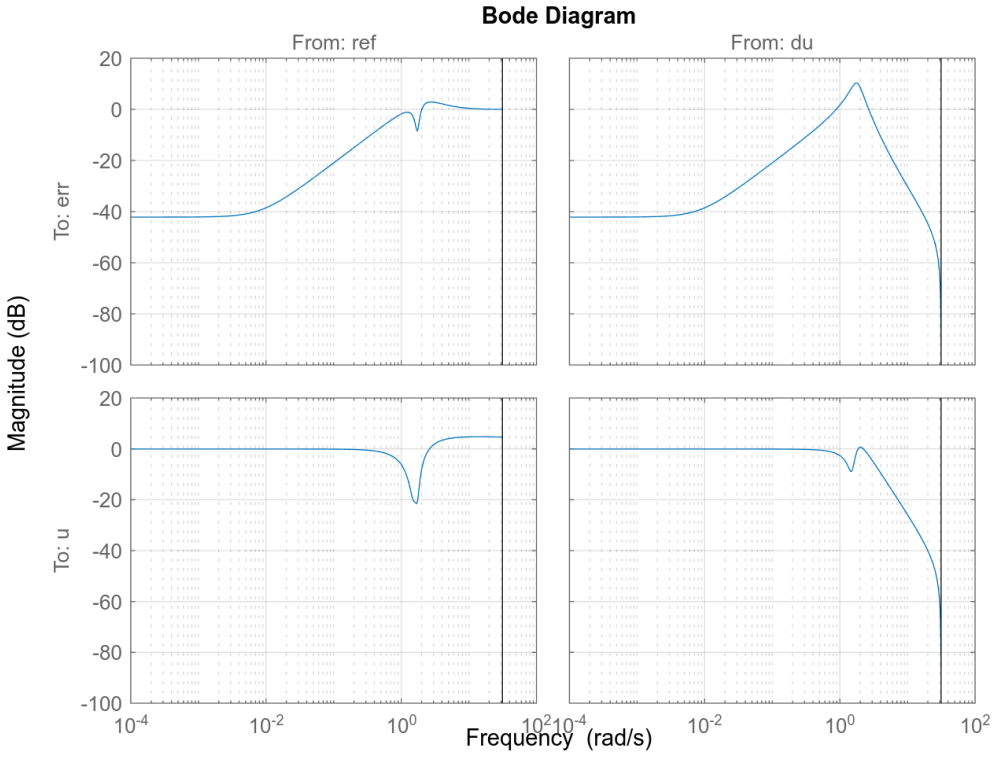
```
abs(ans)
```

```
ans = 5×1
    0.0182
    0.9640
    0.9640
    0.8702
    0.8702
```

```
step(BC,[1 -sysd;0 0],30), grid on, legend('hinfsyn','bucle abto.')
```




```
bodemag(BC), grid on
```

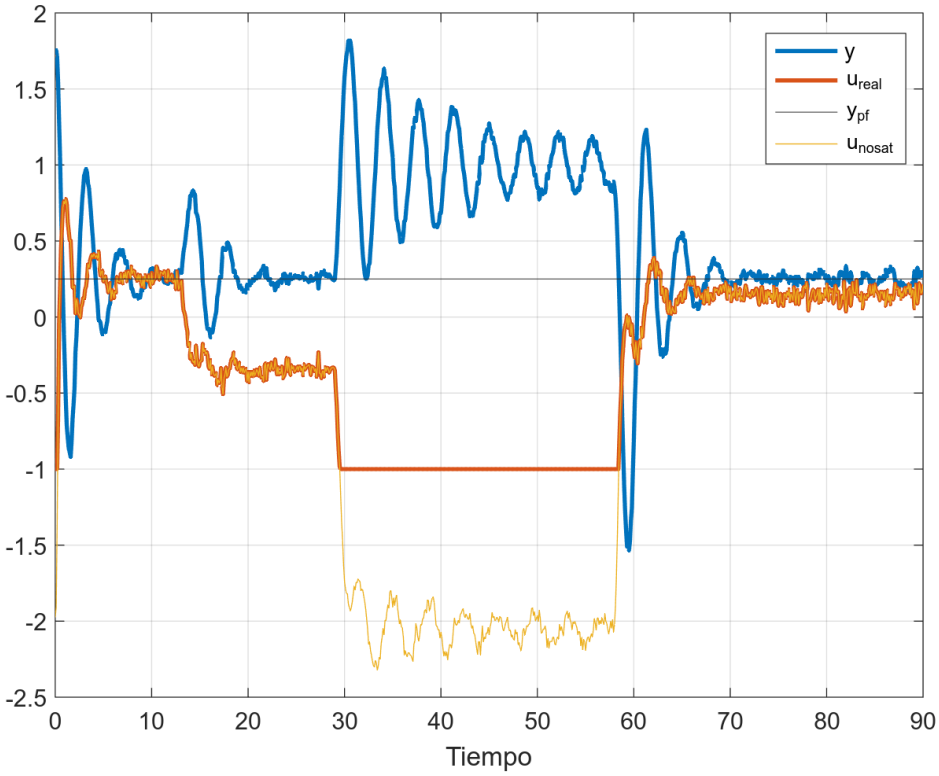


Simulación ante perturbación a la entrada y CI no nulas

```
ypf=0.25;
upf=inv(dcgain(sysd))*ypf;
Tsim=(0:900)*Ts;
Nsamples=length(Tsim);
x=[1.75;0.45]; x_control=[0;0;0]; %condiciones iniciales.

grfy=zeros(1,Nsamples); grfxhat=zeros(3,Nsamples);
grfu=zeros(1,Nsamples); grfuideal=zeros(1,Nsamples);
```

Elegiremos la ganancia antiwindup para que la dinámica del antiwindup sea "más rápida" que la dominante de bucle cerrado... Sin pasarse para evitar efectos de ruidos o overrides espúreos en caso de medidas "reales" (tracking-mode antiwindup), aunque aquí, a partir de límites conocidos internamente por el controlador (una especie de back-calculation antiwindup) no importa tanto... si los polos antiwindup son rápidos es casi un "stop integration".

```
L_antiwindup=place(Kd.A',Kd.C',[1 .95 .9]*0.75)';

for k=1:Nsamples
    pert=-0.6*(k<130)-1.4*(k<290)+1.9*(k<580)+0.1; %pert. a la entrada
    yabs=C*x+0.02*randn(); %LEER SENSORES, con ruido de medida distrib. normal
    y=yabs-ypf; %incrementales
    %CONTROLADOR
       u=Kd.C*x_control+Kd.D*(-y);
       usat=max(-limiteSatU,min(limiteSatU,u+upf));
       usatinc=usat-upf;
       x_control=Kd.A*x_control+Kd.B*(-y)+L_antiwindup*(usatinc-u); %actualizar estado
    % Gráficas de salidas
    grfy(k)=yabs;
    grfu(k)=usat;
    grfuideal(k)=u+upf;
    grfxhat(:,k)=x_control;
    %Actualizar estado proceso
    x=sysd.A*x+sysd.B*(usat+pert+0.055*randn()); %sumamos ruido proceso aleatorio

end
plot(Tsim,[grfy;grfu],LineWidth=2), yline(ypf), hold on
plot(Tsim,grfuideal), xlabel("Tiempo")
legend('y','u_{real}','y_{pf}','u_{nosat}', Location="best"), grid on, hold off
```




```
plot(Tsim,grfxhat,'-.'), xlabel("Tiempo"), grid on %hay "windup" con L_antiwindup=0.
```