Robust Performance Matlab case study, 5/(s+1)^2

This code was successfully run on Matlab **`R2022a`**

Presentations in video:

http://personales.upv.es/asala/YT/V/cerp1EN.html *http://personales.upv.es/asala/YT/V/cerp2EN.html* *http://personales.upv.es/asala/YT/V/cerp3EN.html* .

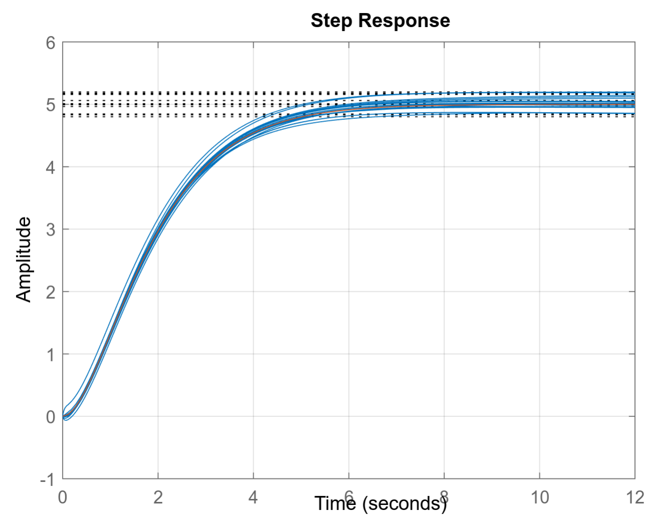
Objectives: understanding the underlying ideas in robust performance and scaled small gain theorem, and the role of H-infinity synthesis, with a simple SISO 2nd order example.

Table of Contents

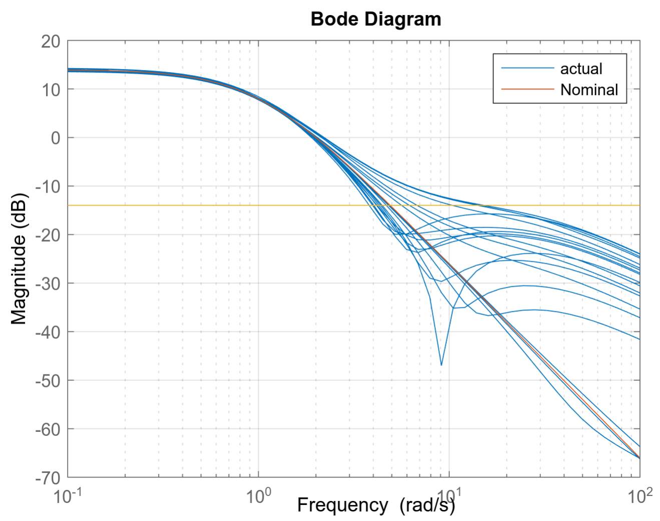
Plant Model

```
s=tf('s');
G=5/(s+1)^2; %NOMINAL PLANT MODEL
BoundDelta=0.2; %Additive unstructured error bound
Gtrue=G+BoundDelta*ultidyn("DeltaScaled")*1/(0.03*s+1);
% we "kill" Greal from 33 rad/s onwards for simulation.
% Not used in robust perf. analysis.

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



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

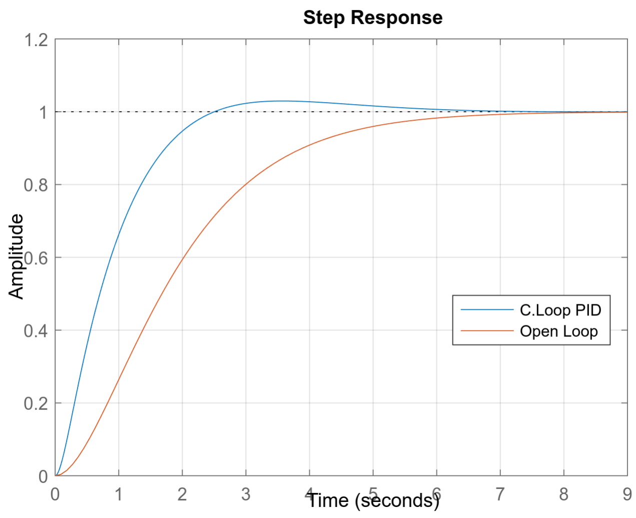


Robustness analysis of a pre-designed controller by simulation

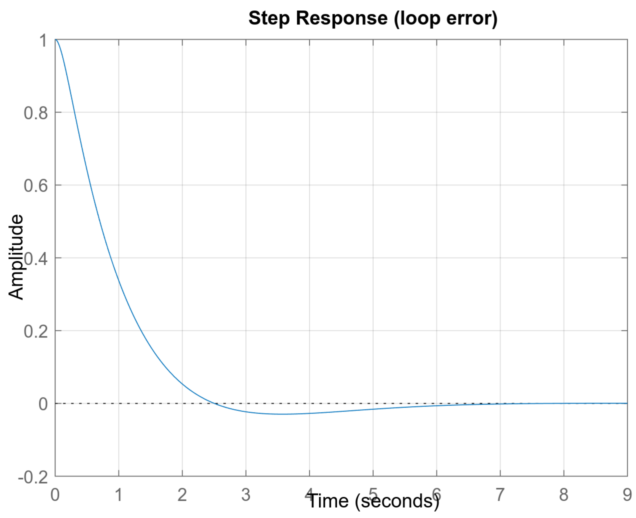
```
K=0.4*(1+0.65/s+0.4*s/(0.1*s+1));
%PID, "trial and error" hand tuning.
```

Nominal performance

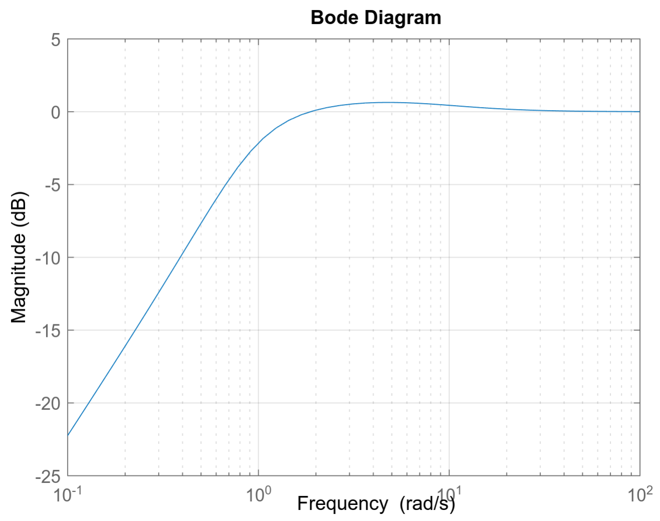
```
CLnom=feedback(G*K,1); %ref->output
step(CLnom,G/dcgain(G),9), grid on, legend("C.Loop PID","Open Loop",Location="best")
```



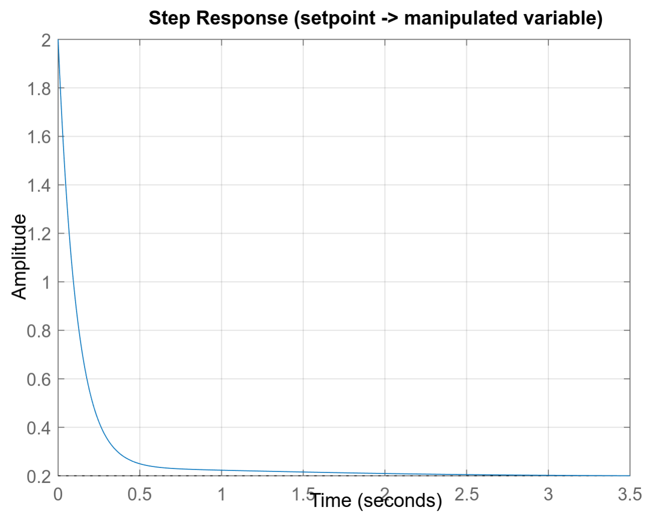
```
CLerr=feedback(1,G*K); %ref->err
step(CLerr,9), grid on, title("Step Response (loop error)")
```



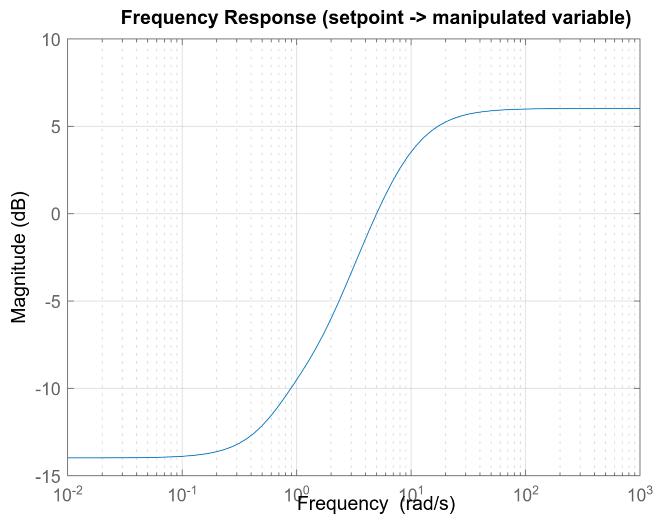
```
bodemag(CLerr), grid on
```



```
CLu=feedback(K,G);
step(CLu), grid on, title("Step Response (setpoint -> manipulated variable)")
```



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

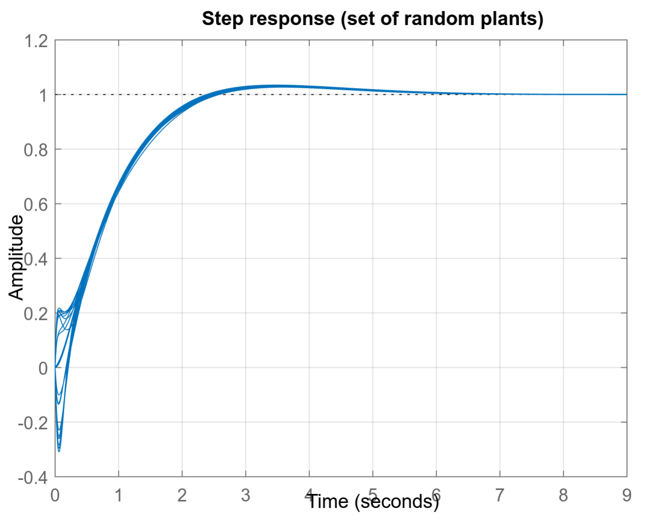


Performance seems satisfactory to us in the "time domain", on the nominal plant. It might not be if we had problems of control action saturation, or excessive amplification of disturbances, or measurement noise at high frequency, but we take them as acceptable in this simple case study.

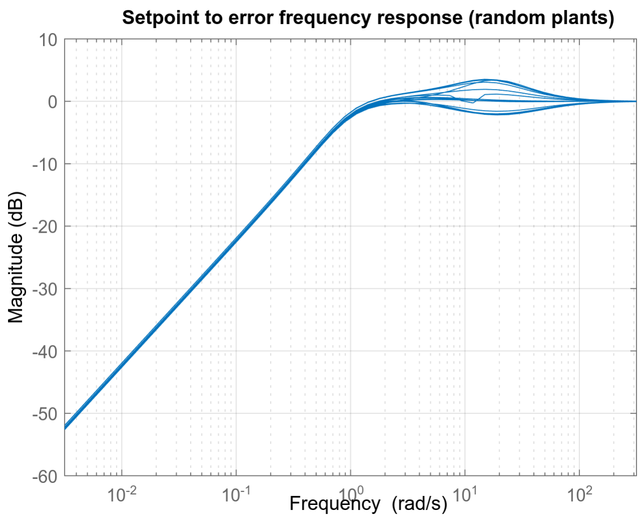
Simulation-based robustness analysis

Obviously, analyzing robust features could simply be carried out via "simulating" by rolling the dice with Gtrue and checking that there is nothing unacceptable... in fact, we will do it just now.

```
step(feedback(Gtrue*K,1)), grid on, title("Step response (set of random plants)")
```

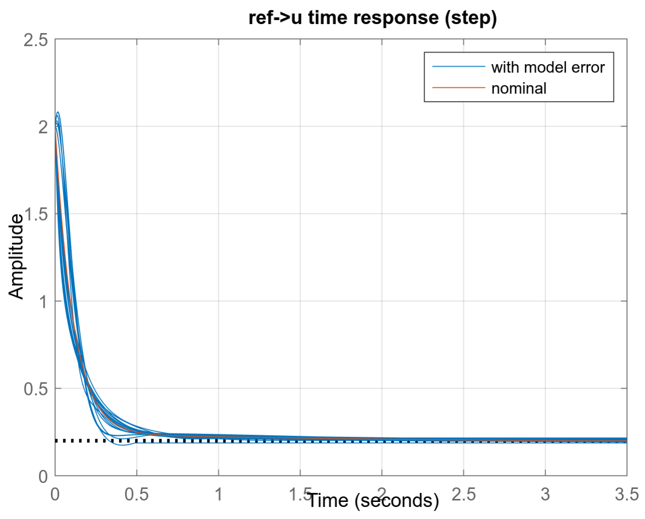


```
bodemag(feedback(1,Gtrue*K),logspace(-2.5,2.5)), grid on, title("Setpoint to error frec
```

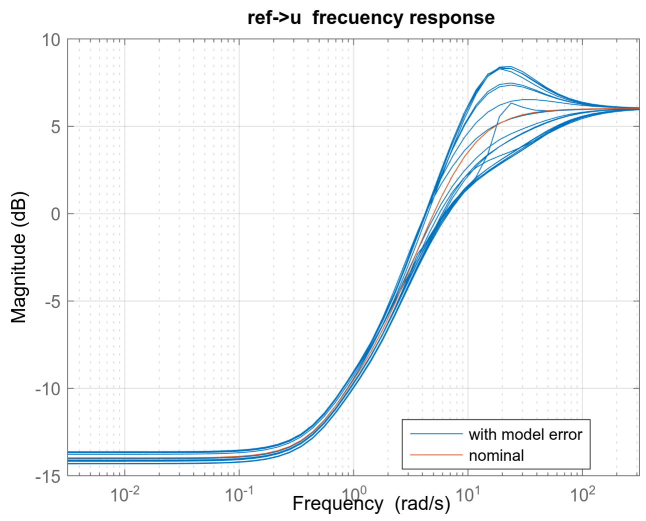


Let us have a look to the control action (manipulated variable) to ensure nothing "bad" skips our radar:

```
step(feedback(K,Gtrue),feedback(K,G)), grid on
title("ref->u time response (step)"),legend("with model error","nominal")
```



```
bodemag(feedback(K,Gtrue),feedback(K,G),logspace(-2.5,2.5))
grid on, title("ref->u  frecuency response"),legend("with model error","nominal",Locati
```



Conclusions and pending tasks

Our objective is to understand the theoretical justification of the robust performance margins, to guarantee that we are not going to have "bad luck" and some roll of the dice is surprisingly wrong.

Besides, the theory will justify design methodologies if this PID does not have satisfactory performance.

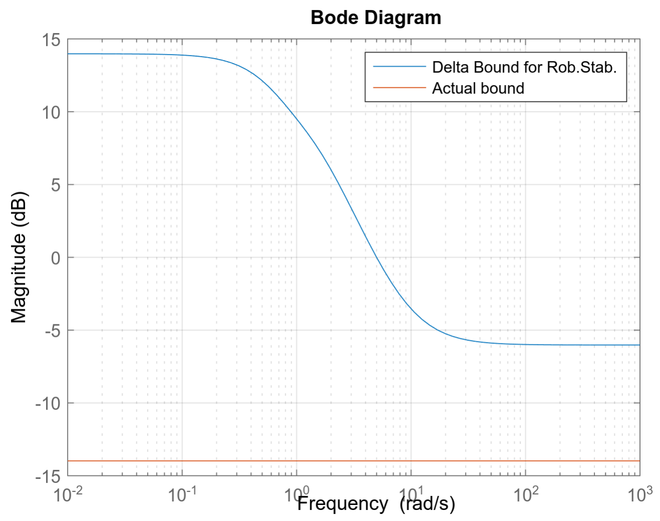
Nevertheless, "simulation-based performance validation" is of paramount importance in practice, because we can model nonlinearities, high-order ODEs or PDEs, faults, etc. without the need of extracting a Δ and bounding its size, and without the possible conservatism of small-gain theorem and its variations.

Robustness analysis by performance certificates (margins)

Robust stability margin

Additive unstrucured uncertainty margin formulae.

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



RS is granted. In fact, we can increase our uncertainty.

```
maxBoundForDeltaAddRS=1/norm(ThingForDeltaadd,inf)
```

```
maxBoundForDeltaAddRS = 0.5000
```

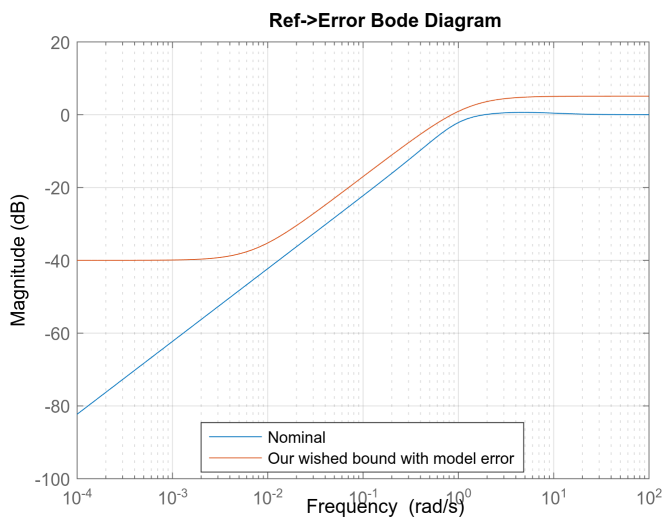
Sizer of Δ can increase by a factor of:

```
maxBoundForDeltaAddRS/BoundDelta
```

```
ans = 2.5000
```

Robust performance frequency bound definition

```
bodemag(CLerr), grid on, hold on
TargetBandwidth=0.85; %inverse prop. to rise time, roughly
templateErr=makeweight(0.01,TargetBandwidth,1.8);
bodemag(templateErr), hold off, legend("Nominal","Our wished bound with model error",Lo
title("Ref->Error Bode Diagram")
```



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

Robust Performance Analysis

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

```
1 state removed.
```

```
WeightedGenPlant=Wout*GenPlant*Win; %encodes the 3x3 robust performance problem
```

This should be less than 1 with the PID:

```
BCPond=lft(WeightedGenPlant,K); %This is "P" in the theory slides
norm(BCPond,inf)
```

```
ans = 0.9998
```

```
%bodemag(feedback(1,Gtrue*K),templateErr,logspace(-2.5,2.5)), grid on, title("Setpoint
```

H-infinity design for robust performance

If we minimize the infinity norm in closed loop, we get:

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

```
GAM = 0.9387
```

So it would be better, at least in theory. Here we have the resulting regulator:

```
zpk(Khinf)
```

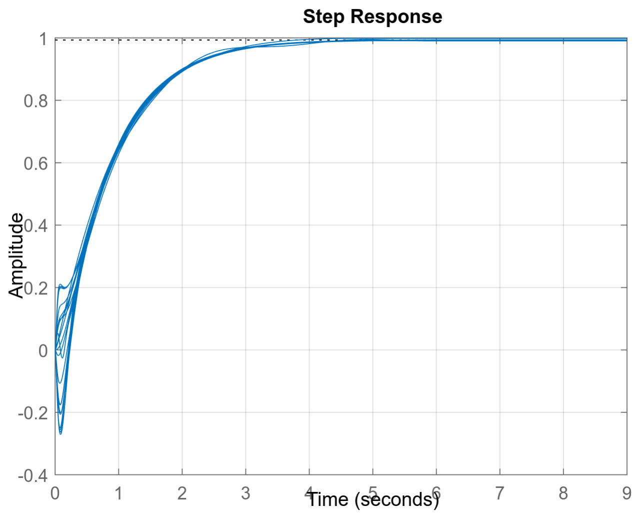
```
ans =
 
          126.23 (s+1)^2
  --------------------------------
  (s+0.007068) (s+8.646) (s+68.89)
 
Continuous-time zero/pole/gain model.
```

***REMARK 1:** no input disturbance entails cancellation of the plant in this h-infinity controller, so performance when subject to input disturbances may be disappointing (this is a caveat of mixed-sensitivity design; we are doing similar stuff here). If these disturbances were expected, do include them in the generalised plant.

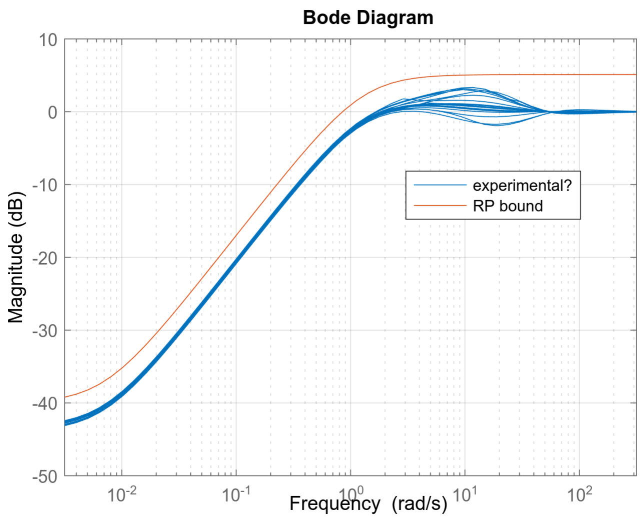
***REMARK 2:** trial-and-error search for the multiplier is blatantly inefficient. We might just craft a loop for searching it, or doing "fminsearch", for instance. Furthermore, for fixed controller, finding the multiplier that minimises the H-infinity norm is a convex problem. The command "musyn" exploits that fact.

Simulation in time and frequency of the achieved performance level

```
Ksim=Khinf;
step(feedback(Gtrue*Ksim,1)), grid on
```

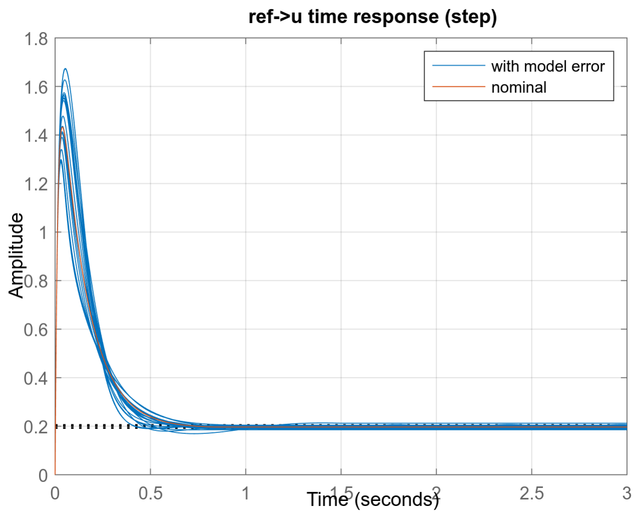


```
bodemag(feedback(1,Gtrue*Ksim),templateErr,logspace(-2.5,2.5)), grid on, legend("experi
```

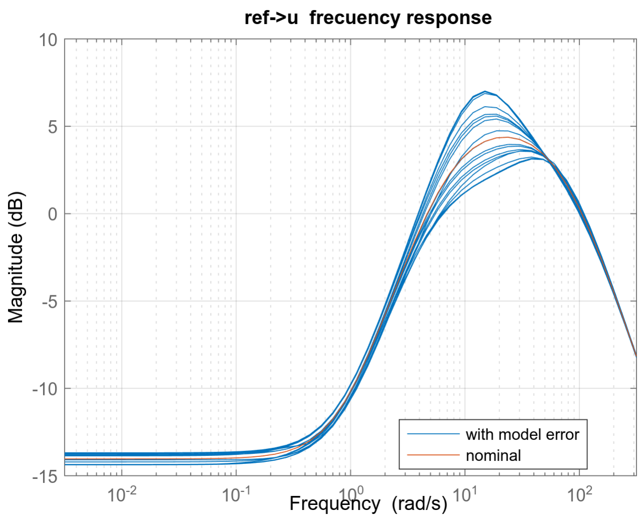


Let us have a look to the control action (manipulated variable) to ensure nothing "bad" skips our radar:

```
step(feedback(Ksim,Gtrue),feedback(Ksim,G)), grid on
title("ref->u time response (step)"),legend("with model error","nominal")
```



```
bodemag(feedback(Ksim,Gtrue),feedback(Ksim,G),logspace(-2.5,2.5))
grid on, title("ref->u  frecuency response"),legend("with model error","nominal",Locati
```


ref->u frecuency response
Magnitude (dB)
Frequency (rad/s)
10
5
0
-5
-10
-15
10^-2
10^-1
10^0
10^1
10^2
with model error
nominal