Force estimation in uncertain mechanical system: mu-synthesis, comparison with H-∞ and H2 observer

This code was executed on Matlab **R2022b**

Watch a presentation in video at:

http://personales.upv.es/asala/YT/V/obsmu1EN.html (modelling)

http://personales.upv.es/asala/YT/V/obsmu2EN.html (optimization)

http://personales.upv.es/asala/YT/V/obsmu3EN.html (order reduction, freq. domain performance)

http://personales.upv.es/asala/YT/V/obsmu4EN.html (time domain simulation)

Objective: in systems with possible model uncertainty, design the "optimal" observer for robust performance (musyn). Compare with hinfsyn and h2syn (observer) designs.

Table of Contents

Model

A mass-spring-damper will nominally be a 2nd order system $p(s)=\frac{2}{s^2+s+5}(u(s)+F(s))$, subject to a known force (manipulated variable, u) and an unknown disturbance force, $F(s)$. Our problem will be set in continuous-time.

Our model will have two refinements:

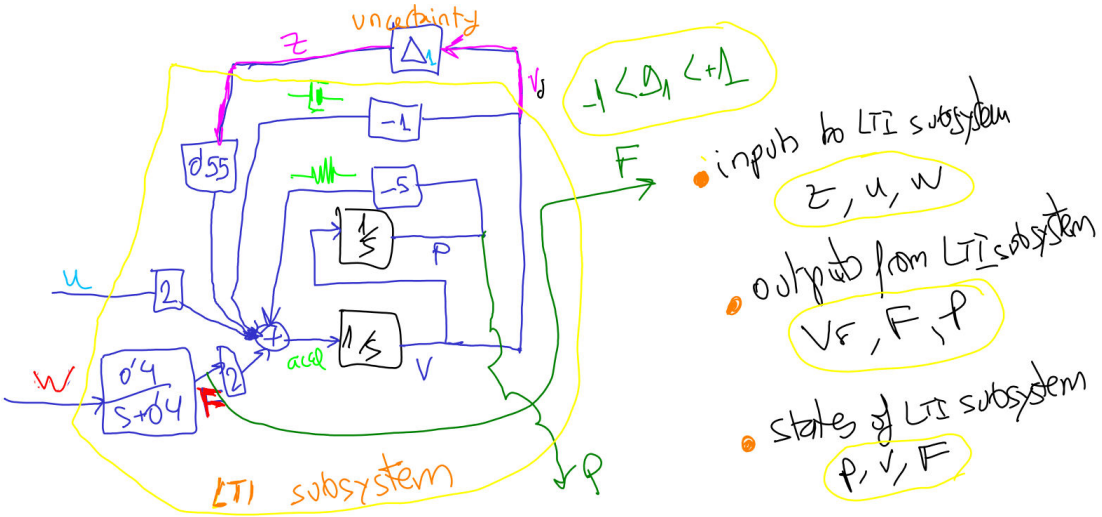
- First, we will limit the disturbance bandwitdh to 0.4 rad/s. We'll assume that its energy (power spectral density for H2, worst-case amplitude for Hinfinity) progressively decreases from that frequency onwards, considering $F(s)$ to be the output of a filter 0.4/(s+0.4) whose input is a unit PSD (when thinking in H2) or unit amplitude (if targeting Hinf) signal $w(s)$:

```
polepert=.4; %band-limited disturbance force
```

- Second, we will consider that damping coefficient is uncertain. Actually, we will consider friction to be "$\frac{b}{M} \cdot vel$", but with uncertain "b" we will get "$-1 \cdot vel + \Delta \cdot vel$", with Δ being a scalar (real uncertainty, unnormalized) of maximum size (absolute value) < 0.55;

Hence we will extract speeed as an "LFT" output and it will enter the system again after being multiplied by "Delta". Robust control toolbox allows other forms of entering this uncertainty, but this is the one closest to the "theoretical" LFT framework for robust control.

Thus, the underlying model has the block-diagram form:



And we can write down the normalised state-space equations as

$$\dot{p} = vel,$$

$$\dot{vel} = -5p - vel + 2(u + F) + 0.55z, \text{ [acceleration, force balance]}$$

$$\dot{F} = -0.4F + 0.4w \text{ [low-pass characterization of F]}$$

$$z = \Delta_1 \cdot vel, \text{ [uncertainty in friction]}$$

where Δ_1 is normalized to $|\Delta_1| < 1$.

Let's code that in Matlab:

```
delt=ureal("Delta",0)
```

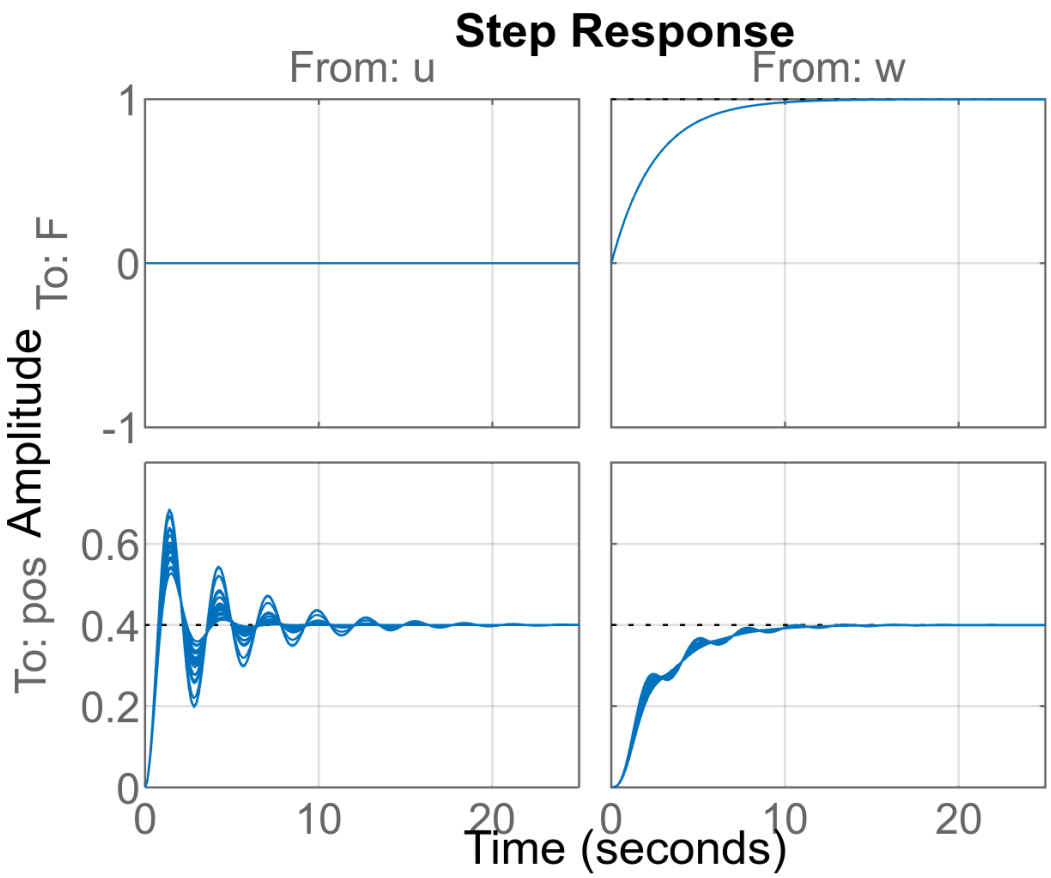
```
delt =
  Uncertain real parameter "Delta" with nominal value 0 and variability [-1,1].
```

```
A=[0 1 0;-5 -1 2;0 0 -polepert];
Gs1=ss(A,[0 0 0;0.55 2 0;0 0 polepert],[0 1 0; 0 0 1;1 0 0],0);
Gs1.InputName={'z_unc','u','w'};
Gs1.OutputName={'vel_delta','F','pos'};
```

```
Gs1.StateName={'pos','vel','F'};
GsUnc=lft(delt,Gs1);
```

Note: uncertain real parameters may be also directly plugged into state-space matrices ... the LFT formalism is close to the "theory" and would allow for dynamic uncertainty (say, damper with internal resonances), delta of type ultidyn... But, well, there are alternative (and possibly simpler) ways to carry out this uncertain damper modelling in the Robust Control Toolbox. I kept it "the LFT way" for didactic purposes.

```
step(GsUnc), grid on
```



Let us analyse the open-loop plant stability margin:

```
robstab(GsUnc)
```

```
ans = struct with fields:
           LowerBound: 1.8181
           UpperBound: 1.8182
    CriticalFrequency: 2.2361
```

```
1/0.55 %as intuitively expected
```

```
ans = 1.8182
```

Indeed, 1.82*0.55 gives 1, so the system's damping coefficient would be zero, as uncertainty might cancel the nominal damping of -1.

Let us have a look to the nominal model:

```
Gs=GsUnc.NominalValue;
zpk(Gs)
```

```
ans =

  From input "u" to output...
   F:  0

               2
    pos:  -------------
          (s^2 + s + 5)

  From input "w" to output...
          0.4
    F:  -------
        (s+0.4)

                   0.8
    pos:  ---------------------
          (s+0.4) (s^2 + s + 5)

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

```
norm(Gs(1,2),'inf')
```

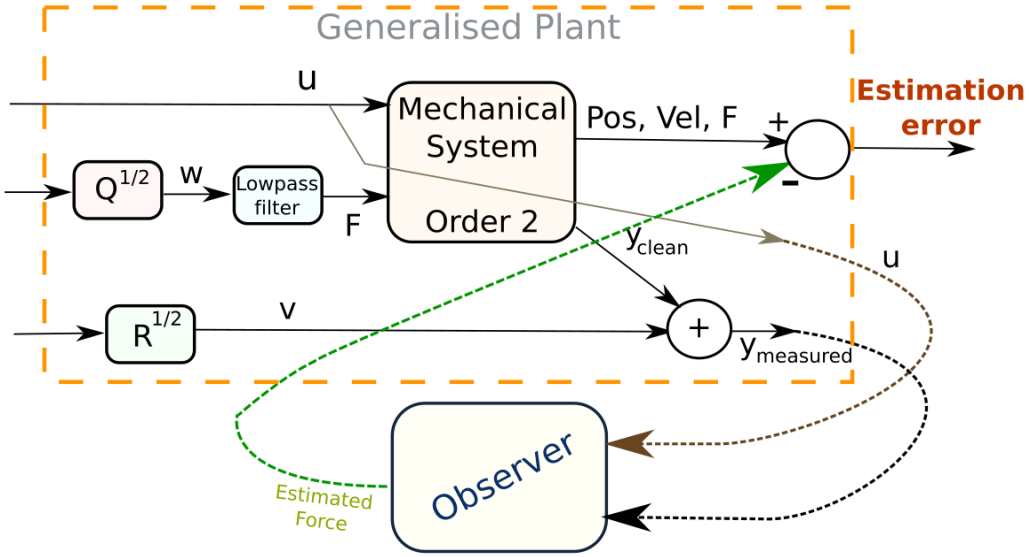
```
ans = 1
```

We will now introduce a "white noise square root of PSD" parameter (for H2; in discrete-time this would be the standard deviation of input sequence) or the "worst-case amplitude at all frequencies" for later manipulations:

```
sqrtPSD_measnoise=0.025;
sqrtPSD_procnoise=4;
```

Building the generalised plant to encode the problem to be solved

In control engineering, we usually think of "observers" as estimators of the whole state: this may be considered as the generalised plant below (well, Q and R and the lowpass filter may also be considered to be in the "weighted" generalised plant). Note that the "weighted" versus "non-weighted" is something useful to us as engineers to understand what we are doing, but the optimizer will just operate on the "weighted" generalised plant.



***NOTE:** We will just output "F" the third state, not the whole state because we decided it that way when stating the problem to be solved. Outputs "Pos, Vel, F" in the above block diagram would be the ones in an observer of the "full" state which we are not addressing here.

Also, weights Q, R will be deferred to the weighted generalized plant.

```
sens_noise=sumblk("ymeasured = pos+v"); %sum adding measurement noise
errs3=sumblk("obserrF = F-Fest"); %sum block computing F estimation error
GenPlant=connect(GsUnc,sens_noise,errs3,{'u','w','v','Fest'},
{'obserrF','u','ymeasured'});
zpk(GenPlant(1,2).NominalValue)
```

```
ans =

  From input "w" to output "obserrF":
    0.4
  -------
  (s+0.4)

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

```
norm(GenPlant(1,2).NominalValue,inf)
```

```
ans = 1
```

Design of an estimator of the third state (F), weighted generalised plant and optimization

1) H2SYN Observer on nominal model, no output weights

We'll start with an $\mathscr{H}_2$, design using the nominal plant, in continuous-time. This has close relationship with classical observers $\dot{\hat{x}} = (A - LC)x + \ldots$ and Kalman filtering.

```
Win1=tf(blkdiag(1,sqrtPSD_procnoise,sqrtPSD_measnoise));
Win1.InputName={'u_esc','w_esc','v_esc'};
Win1.OutputName={'u','w','v'};

Win=blkdiag(Win1,1);
PGNomWin=GenPlant.NominalValue*Win;
```

```
norm(PGNomWin(1,2),2) %this would be the std. dev. of "F"
```

```
ans = 1.7889
```

We will not add output frequency weighting... just wishing to minimise estimator variance.

```
[OO2,CL2,GAM2]=h2syn(PGNomWin,2,1); GAM2 %closed loop error standard
deviation
```

```
GAM2 = 0.9230
```

```
size(OO2) %same order as the plant (we have constant weights in the
weighted generalised plant)
```

```
State-space model with 1 outputs, 2 inputs, and 3 states.
```

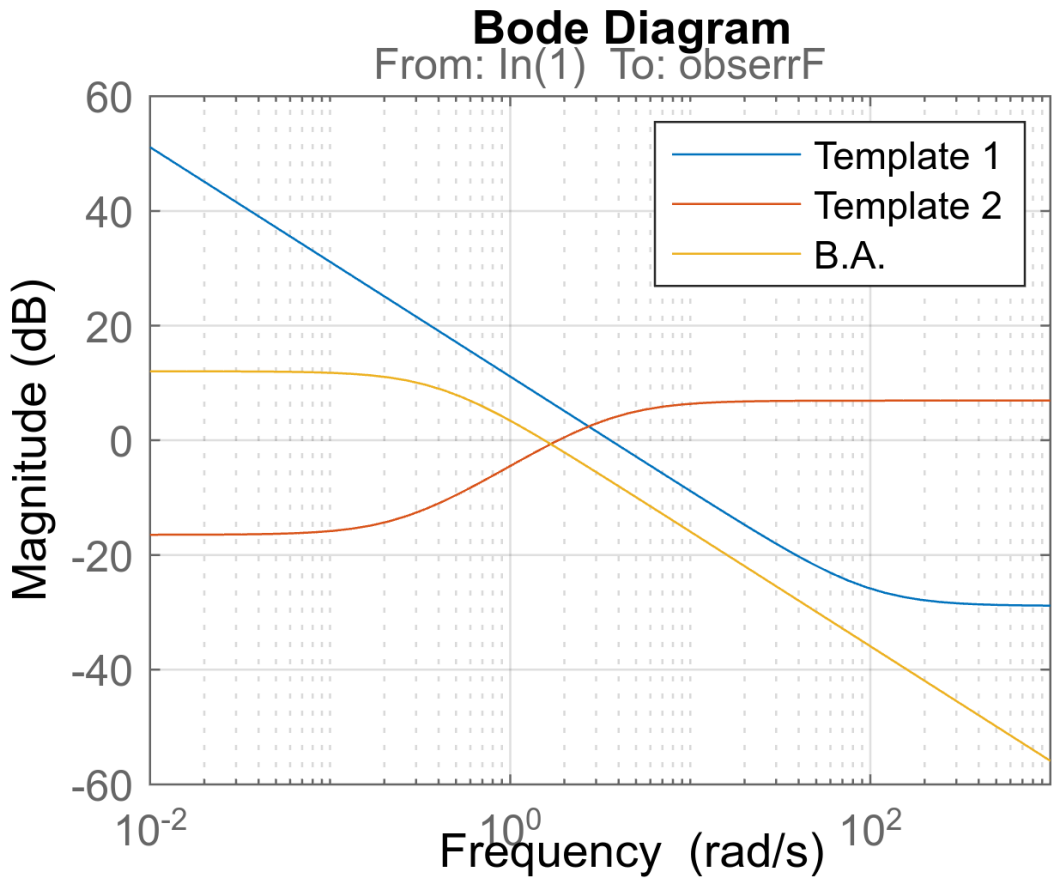
```
OO2.InputName={'u','ymeasured'};
OO2.OutputName={'Fest'};
```

2) HINFSYN design, nominal model, output weights

We'll now pursue and $\mathscr{H}_\infty$ design, over the nominal plant. White noise isn't the same as "amplitude bound" but we'll keep the same input weights as in the H2 design.

We wish some frequency-dependent bounds on estimation error, so we'll add some output weights to the generalised plant.

```
s=tf('s');
TemplateLim1=3.6*(1+0.01*s)/s; %high-frequency limitation
TemplateLim2=.15*(1+s/0.25)/(1+s/3.7); %low-frequency limitation
bodemag(TemplateLim1,TemplateLim2,GenPlant(1,2)*sqrtPSD_procnoise,logspac
e(-2,3)),grid on, legend("Template 1","Template 2","B.A.")
```




```
Filt1=1/TemplateLim1;
Filt2=1/TemplateLim2;
Wout1=[Filt1;Filt2]; % TWO filters, we augment the number of outputs.
Wout=blkdiag(Wout1,1,1);
size(Wout)
```

```
Transfer function with 4 outputs and 3 inputs.
```

```
GenPlant_Weighted=Wout*GenPlant*Win;
size(GenPlant_Weighted)
```

```
Uncertain state-space model with 4 outputs, 4 inputs, 5 states, and 1 blocks.
```

```
GenPlant_Weighted.OutputName={'Estim Err pond1','Estim Err
pond2','u','y'};
```

With no observer, these would be the performance (sort of "open loop") using the naive estimate $\widehat{F} = 0$:

```
norm(GenPlant_Weighted(1:2,2).NominalValue,'inf')
```

```
ans = 26.6667
```

We are 27 times above the required accuracy at some frequency.

We carry out the optimization with:

```
[OOi,CL,GAMi]=hinfsyn(GenPlant_Weighted.NominalValue,2,1);
GAMi %lower than 1 -> Nominal Performance OK
```

```
GAMi = 0.7616
```

```
size(OOi) % order of the plant + that of the weights
```

```
State-space model with 1 outputs, 2 inputs, and 5 states.
```

```
OOi.InputName={'u','ymeasured'};
OOi.OutputName={'Fest'};
```

If we look at the worst-case performance confronted to model error

```
BCipond=lft(GenPlant_Weighted,OOi);
robstab(BCipond) %this margin is not relevant for our problem
```

```
ans = struct with fields:
           LowerBound: 1.8181
           UpperBound: 1.8182
    CriticalFrequency: 2.2361
```

Robust stability margin is identical to the plant's margin, because our observer is not actually closing any "control loop" on the actual plant.

```
[wcg_i, wcu_i]=wcgain(BCipond) %we don't achieve robust performance
```

```
wcg_i = struct with fields:
           LowerBound: 2.3376
           UpperBound: 2.3424
    CriticalFrequency: 2.2530
wcu_i = struct with fields:
    Delta: 1
```

```
robgain(BCipond,1)
```

```
ans = struct with fields:
           LowerBound: 0.3754
           UpperBound: 0.3762
    CriticalFrequency: 2.3297
```

3) MUSYN design

Now we will get robust performance (I fiddled with the weights a bit)

```
tic
[OOmu,GAMmu]=musyn(GenPlant_Weighted,2,1,musynOptions(MixedMu="on"));
```

```
DG-K ITERATION SUMMARY:
-----------------------------------------------------------------------
                      Robust performance                 Fit order
-----------------------------------------------------------------------
  Iter          K Step        Peak MU        DG Fit           D       G
    1           2.515          1.47         1.481           10       6
    2           1.134         1.193         1.204            8       8
    3           1.044         1.087         1.112            8       8
    4          0.9989         1.036         1.038           10       8
    5          0.9902         1.019         1.014           10       8
    6          0.9875         1.007         1.008           10       8
    7          0.9847        0.9992         1.019           10       8
    8           1.003         1.001         1.063           10       8

Best achieved robust performance: 0.999
```

```
toc
```

```
Elapsed time is 4.286088 seconds.
```

```
GAMmu %1, i.e., robust performance
```

```
GAMmu = 0.9992
```

```
BCmupond=lft(GenPlant_Weighted,minreal(OOmu));
```

```
2 states removed.
```

```
robstab(BCmupond) %not relevant with "observer" problems
```

```
ans = struct with fields:
           LowerBound: 1.8181
           UpperBound: 1.8182
    CriticalFrequency: 2.2361
```

```
wcgain(BCmupond)
```

```
ans = struct with fields:
           LowerBound: 0.9975
           UpperBound: 0.9999
    CriticalFrequency: 2.2919
```

```
size(OOmu)
```

```
State-space model with 1 outputs, 2 inputs, and 23 states.
```

```
OOmu.InputName={'u','ymeasured'};
OOmu.OutputName={'Fest'};
```

Order reduction of musyn result

```
size(OOmu) %musyn produces high-order results
```

```
State-space model with 1 outputs, 2 inputs, and 23 states.
```

```
hsvd(OOmu)
```

```
ans = 23×1
    6.9032
    5.9651
    2.0476
    0.4567
    0.3798
    0.1985
    0.1377
    0.0191
    0.0097
    0.0034
     ⋮
```

```
OOmured=balred(OOmu,7);
eig(OOmured)
```

```
ans = 7×1 complex
10^2 ×
  -3.4132 + 0.0000i
  -0.0443 + 0.0462i
  -0.0443 - 0.0462i
  -0.0058 + 0.0205i
  -0.0058 - 0.0205i
```

```
  -0.0195 + 0.0191i
  -0.0195 - 0.0191i
```

We'll forget the pole at -341: out of our range of interesting frequencies and maybe requiring fast sampling rate in discrete implementation...

```
[slow,fast]=freqsep(OOmured,200);
OOmured=slow+dcgain(fast);
eig(OOmured)
```

```
ans = 6×1 complex
  -4.4316 + 4.6179i
  -4.4316 - 4.6179i
  -0.5778 + 2.0479i
  -0.5778 - 2.0479i
  -1.9482 + 1.9065i
  -1.9482 - 1.9065i
```

```
size(OOmured)
```

```
State-space model with 1 outputs, 2 inputs, and 6 states.
```

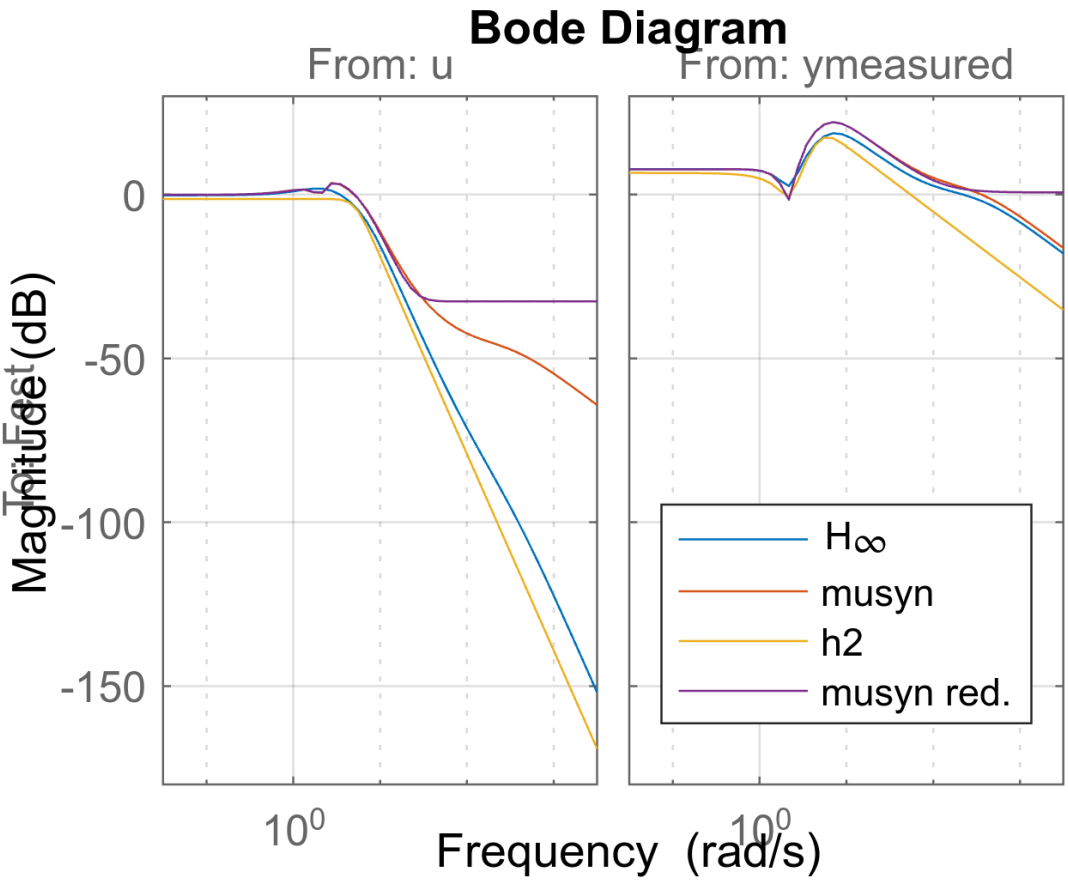
```
BCmupondred=lft(GenPlant_Weighted,OOmured);
wcgain(BCmupondred)
```

```
ans = struct with fields:
           LowerBound: 0.9944
           UpperBound: 0.9965
    CriticalFrequency: Inf
```

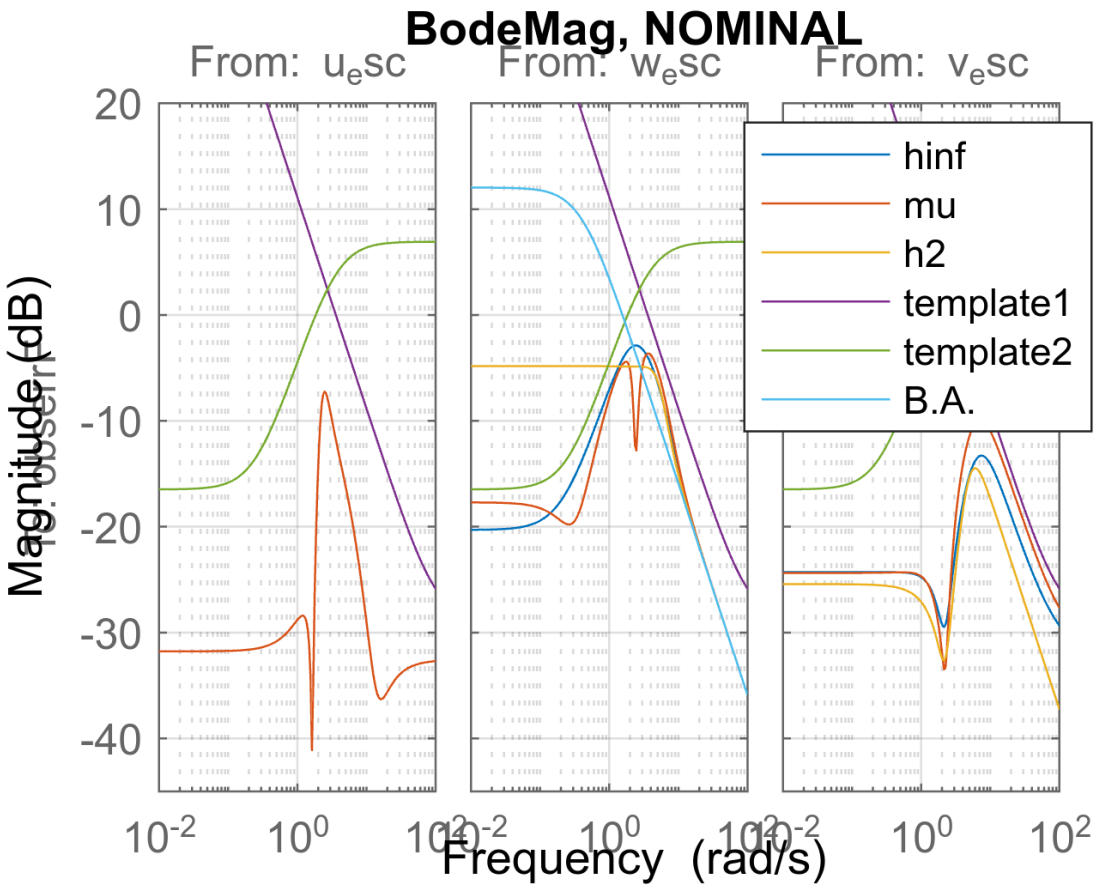
```
OOmured.InputName={'u','ymeasured'};
OOmured.OutputName={'Fest'};
```

Performance analysis (comparative) in frequency domain

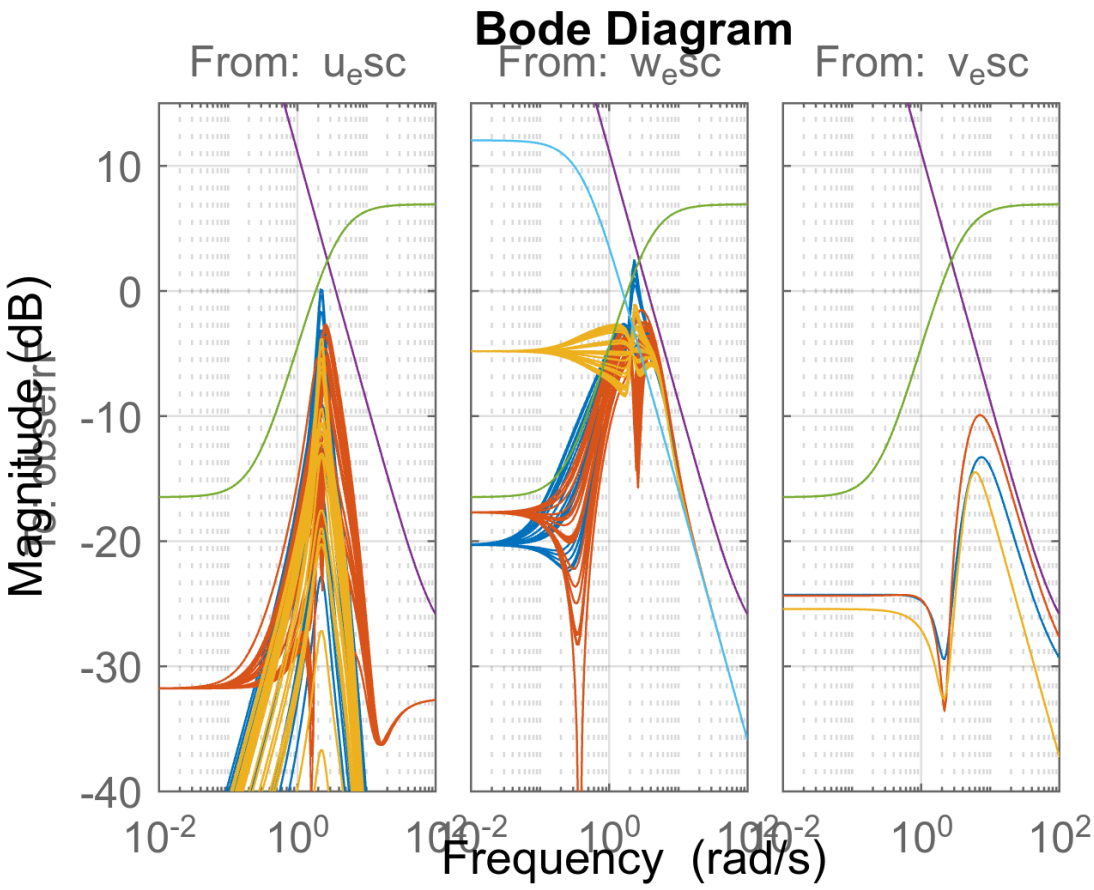
```
bodemag(OOi,OOmu,OO2,OOmured,logspace(-1.5,3.5)),
legend('H\infty','musyn','h2','musyn red.',Location='best'), grid on
```




```
BC2Wi=lft(GenPlant,OO2)*Win1;
BCiWi=lft(GenPlant,OOi)*Win1;
BCmuWi=lft(GenPlant,OOmured)*Win1;
bodemag(BCiWi.NominalValue,BCmuWi.NominalValue,BC2Wi.NominalValue,Template
Lim1*[1 1 1],TemplateLim2*[1 1 1],GenPlant(1,2)*sqrtPSD_procnoise*[0 1
0],logspace(-2,2,200)), grid on
ylim([-45 20]), title("BodeMag, NOMINAL")
legend("hinf","mu","h2","template1","template2","B.A.",Location="best")
```




```
bodemag(BCiWi,BCmuWi,BC2Wi,TemplateLim1*[1 1 1],TemplateLim2*[1 1
1],GenPlant(1,2)*sqrtPSD_procnoise*[0 1 0],logspace(-2,2,150)), grid on
ylim([-40 15])
```



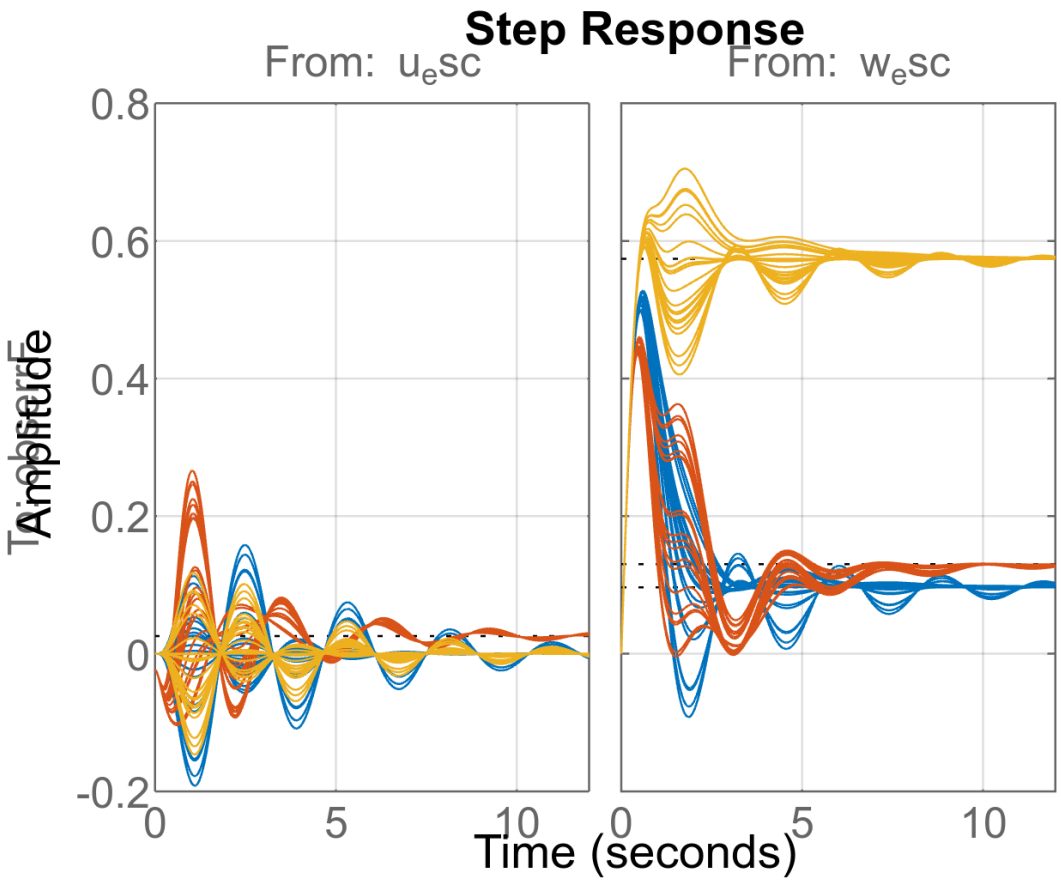
Note: although we carried out "`bodemag`" to better understand the effect of each input, what musyn and hinfsyn minimise is, of course, the combined effect ("`sigma`"), i.e., square root of sum of the squares of each individual bodemag.

Performance analysis in time domain

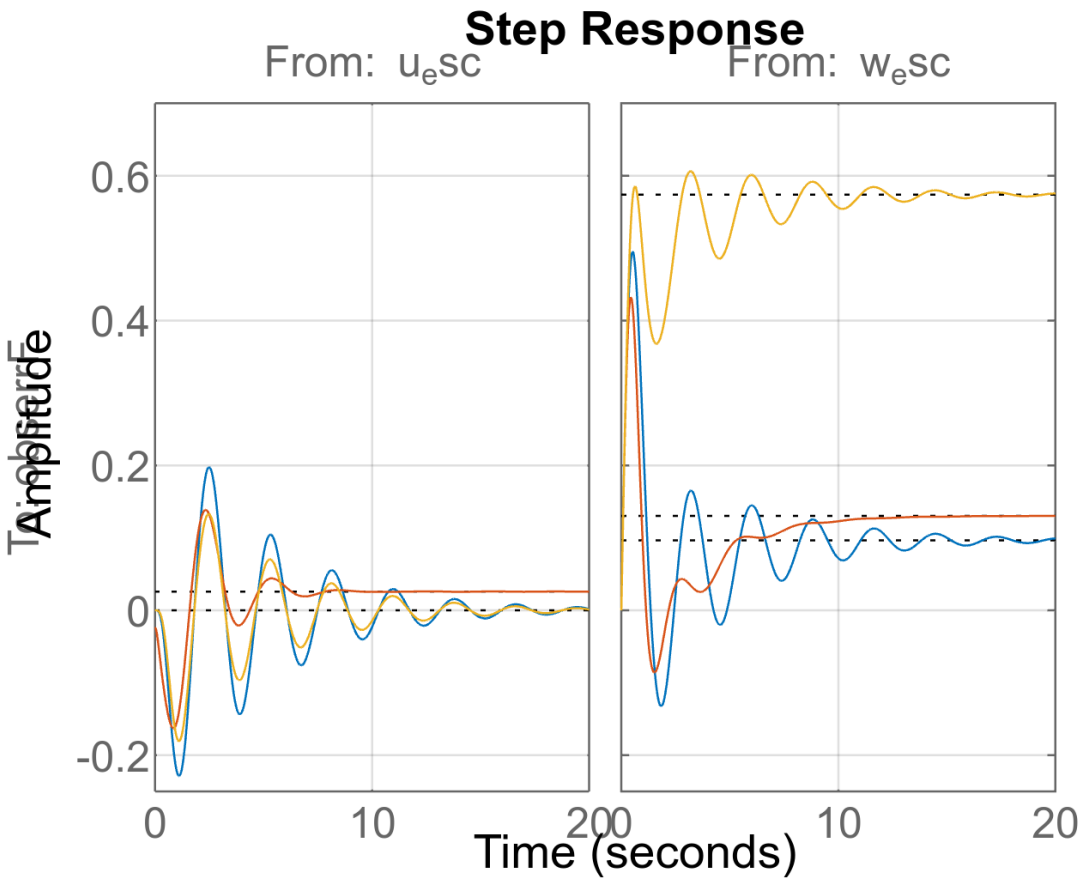
NOTE: time-domain is NOT the primary objective of "hinfinity" or "mu" synthesis... well, it might be thought as minimising ratios of infinite-integral-of-squares but that's not too intelligible: it's frequency and its templates what we use as design variables and performance measure... H2 is more related to "variance" in time domain, anyway... So, the conclusions from the analysis below are "approximate"...

Step response

```
step(BCiWi(:,1:2),BCmuWi(:,1:2),BC2Wi(:,1:2),12), grid on
```




```
%PGSTst=PlantaGen.NominalValue;
PGSTst=usubs(GenPlant,wcu_i);
%PGSTst=usubs(PlantaGen,"Delta",-1); %extremo máximo rozamiento
KK2_2=lft(PGSTst,OO2)*Win1;
KK2i=lft(PGSTst,OOi)*Win1;
KK2m=lft(PGSTst,OOmured)*Win1;
step(KK2i(:,1:2),KK2m(:,1:2),KK2_2(:,1:2),20), grid on
```



Linear Simulation with measurement noise

Generalised plant outputs the estimation error, but we will rebuild things to output the "actual F" and the "measurement" so we can better understand what is happening. So, we'll reconnect some stuff...

```
% different plants to simulate with
Gtst=usubs(GsUnc,wcu_i); % extreme delta=1, as discussed above.
%Gtst=GsUnc.NominalValue; % nominal plant
%Gtst=usubs(GsUnc,"Delta",-1); % maximum damping extreme
Gtst.InputName'
```

```
ans = 1×2 cell
'u'         'w'
```

```
Gtst.OutputName'
```

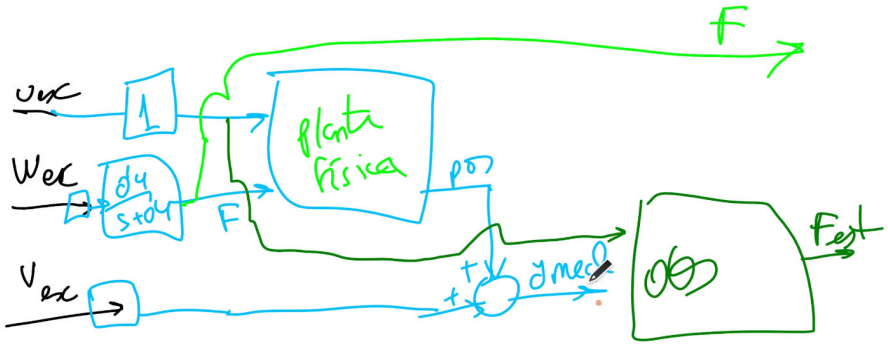
```
ans = 1×2 cell
'F'         'pos'
```

```
%sens_noise was "ymeasured=pos+v", sumblk already defined
Plant_Experiment=connect(Gtst,Win1,sens_noise,{'u_esc','w_esc','v_esc'},
{'F','u','ymeasured'});

% We'll cascade connect the observer, for each of the designs.
Thing_i=connect(Plant_Experiment(2:3,:), OOi,{'u_esc','w_esc','v_esc'},
{'Fest'});
Thing_2=connect(Plant_Experiment(2:3,:), OO2,{'u_esc','w_esc','v_esc'},
{'Fest'});
```

```
Thing_mu=connect(Plant_Experiment(2:3,:), OOmured,
{'u_esc','w_esc','v_esc'},{'Fest'});
```

In block diagram, we are doing this (Planta física means "physical plant", and y_med is y_measured):



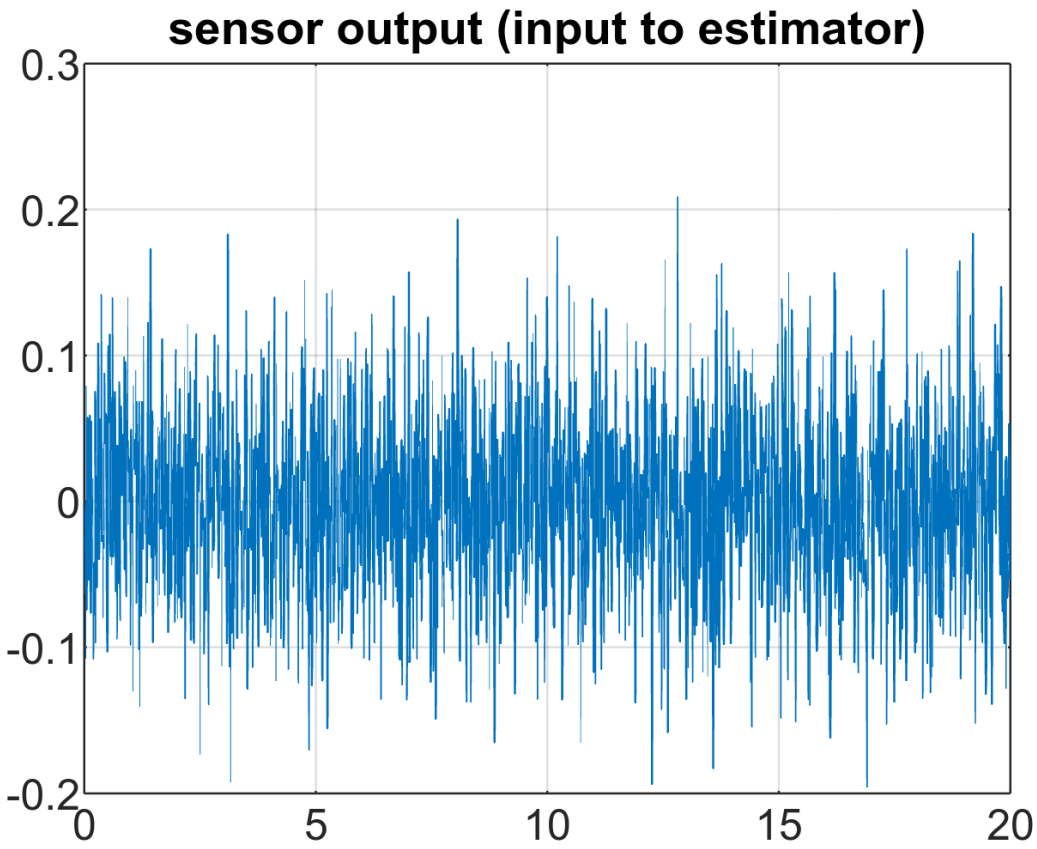
We will now set up the simulation inputs and time range:

```
Ts=0.01; % pi/0.01 = 314 Nyq. Frequency
T=0:Ts:20;Ns=length(T);
%INPUTS to simulate (change at will, download code in description)
utst=ones(Ns,1)*0; % manipulated input (known)
Wtst=cos(2.25e1*T'); % process noise
vtst=randn(Ns,1)*1/sqrt(Ts)/4; % measurement noise, see note below
```

*About measurement noise: we are going to use the PSD of the weighted plant divided by 4^2... Let's say that infinity/mu do not optimize variance, but the "worst case" value... The sensor will also have a certain finite bandwidth, or/and an antialias if implementation is discretized (actually in continuous time it would be infinite variance if the measurement noise were "white noise")... the template could be changed at "very" high frequency so that we forced the resulting observer to "filter more"... Sometimes deciding whether noise sensitivity is acceptable or not is done by experimenting on the "actual" thing...

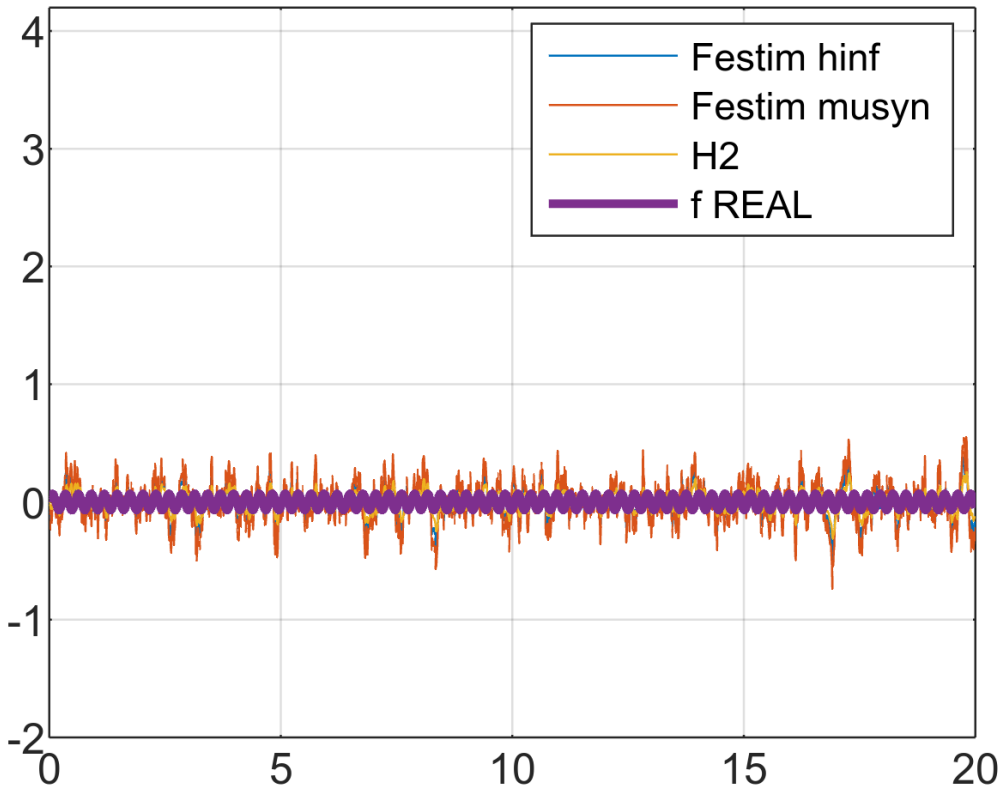
In short, we will simulate a "reasonable" sampling period such that the noise we see is "sensible", without formally claiming to be too exact (in the sense that the H2 optimizer in continuous time expects).

```
ExperimentSim=lsim(Plant_Experiment,[utst Wtst vtst],T);
Fsimreal=ExperimentSim(:,1);
plot(T,ExperimentSim(:,3)), title("sensor output (input to estimator)"),
grid on
```

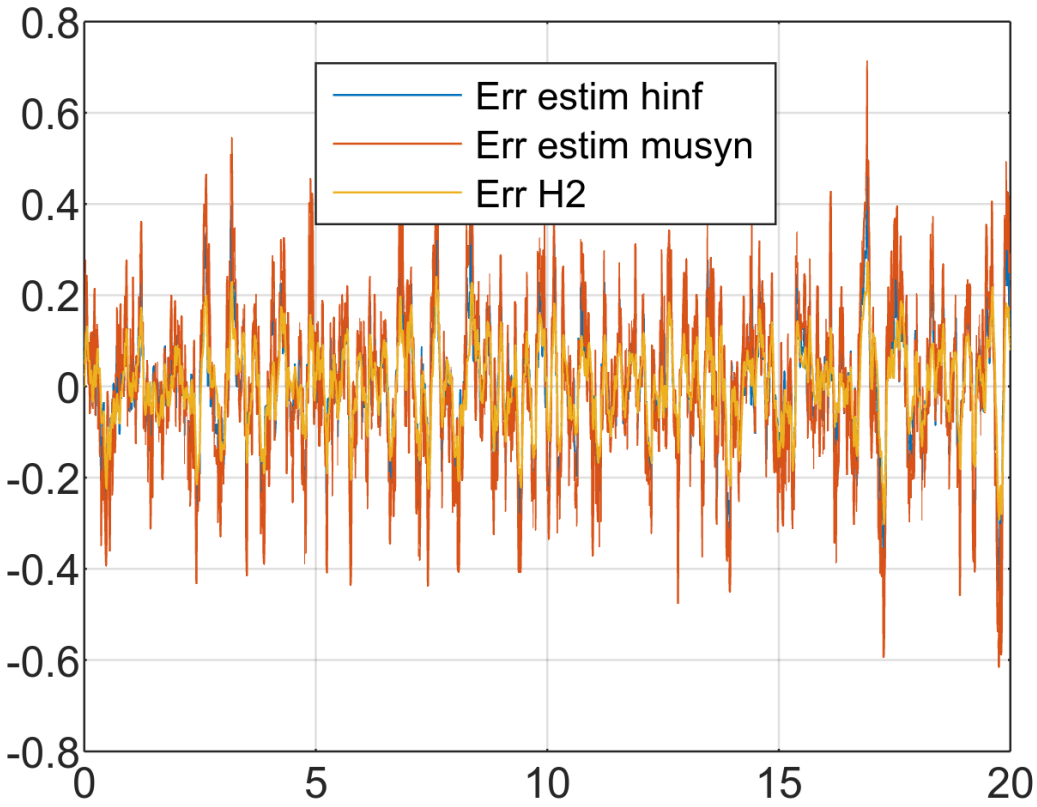



```
FYhF_i=lsim(Thing_i,[utst Wtst vtst],T);
FYhF_mu=lsim(Thing_mu,[utst Wtst vtst],T);
FYhF_2=lsim(Thing_2,[utst Wtst vtst],T);

plot(T,FYhF_i)
hold on
plot(T, FYhF_mu), grid on,
plot(T,FYhF_2),
plot(T,Fsimreal,LineWidth=2)
hold off
legend("Festim hinf","Festim musyn","H2","f REAL",Location="best"),
ylim([-2 4.2])
```




```
plot(T,Fsimreal-FYhF_i)
hold on
plot(T,Fsimreal-FYhF_mu)
plot(T,Fsimreal-FYhF_2), grid on
hold off
legend("Err estim hinf","Err estim musyn","Err H2",Location="best")
```



The pole that we elliminated with freqsep might filter a bit the high-frequency noise. Nevertheless, the templates didn't explicitly request that high-frequency gain should be zero. You may download the source file and change them at will, of course.

Conclusions

Estimating a certain "state" does not always have to be solved with an "observer" as classically understood. The observer problem can be expressed in terms of a "generalized plant" and solved with $\mathscr{H}_2$ or $\mathscr{H}_\infty$, or if there are uncertainties, as a mu-synthesis problem. The order of the observer can be that of the plant (as in the case of constant weights tested here), that plant + weights, or "very" large to adjust frequency-dependent multipliers in `musyn`.

"Musyn" is better than other options "when all the assumptions of the theory are met: the Bode peak under worst case uncertainty is lower"... What about in practice? Well, the input signals in practice are neither "sinusoidal at the worst case frequency" nor "white noise"... There are also "other modeling errors" not included in the plant description... Bode plots have to be analyzed, the standard deviation in simulation (in the face of different possible plants such as "nominal" and "extremes" and "wcu" returned by Matlab), we must consider consider possible problems in discretization, we should experiment on a prototype, we should compare the performance obtained versus complexity of the methodology and the resulting observer... Practice has many nuances...