Advanced Control System

Robust Control of an Active Suspension System

ME-524

Bastien Vergnet - 245633
Willem Suter - 246977

Report date: May 1^{st}, 2019

Contents

1 Norms of Systems and Model Uncertainty

1.1 Norms of systems

```
clear variables;
close all;

%define transfer function and negative transfer function
s=tf('s');
a=2;
G=(s+a)/(s^2+2*s+5);
Gneg=(-s+a)/(s^2-2*s+5);
```

Listing 1: 2-norm Matlab script: defining transfer function

1.1.1 2-Norm

1. Residue theorem method

```

%% 1.1.1. 2-Norm
%1. Residue theorem
Gres=G*Gneg;
[r,p] = residue(Gres.numerator{1},Gres.denominator{1});
Norm_2_res = sqrt(r(1)+r(2));
```

Listing 2: 2-norm Matlab script: residue theorem method

We want to compute $||G_2||$ using the residue theorem. Since

$$||G||_2^2 = \frac{1}{2\pi}\int_{-\infty}^{+\infty} |G(j\omega)|^2 d\omega = \frac{1}{2\pi j}\int_{-j\infty}^{+j\infty} G(-s)\cdot G(s) ds$$

We can use the residue theorem, which states that the $||G||_2^2$ equals the sum of the residues of $G(-s)G(s)$ at its poles in the left half plane. Thus, the 2-norm is given by:

$$||G||_2 = \sqrt{p_1 + p_2} = 0.67082039325$$

2. Frequency response method

```

%2. Frequency response of G
b=bandwidth(G); % -3dB bandwidth
w=-1000*b:0.1:1000*b;
H = freqresp(G,w);
H=abs(H(:)).^2;
Norm_2_freq = (1/(2*pi)*trapz(w,H))^(1/2);
```

Listing 3: 2-norm Matlab script: frequency response method

By definition, for a stable transfer function G(s), the 2-two norm of the transfer function is given by:

$$||G||_2 = (\frac{1}{2\pi}\int_{-\infty}^{\infty} |G(jw)|^2 dw)^{\frac{1}{2}}$$

Therefore, the boundaries of the integral $\pm\infty$ are simulated for $\pm 1000 \cdot (bandwidth\ of\ G) = \pm 1000 \cdot [4.42dB]$:

$$||G||_2 = (\frac{1}{2\pi} \int_{-4420}^{+4420} |G(jw)|^2 dw)^{\frac{1}{2}} = 0.67076697955$$

3. Impulse response method

```
22
23 %3. Impulse response of G
24 t=0:0.01:8; % Tfinal= 8 --> little more than the settling time
25 g=impulse(G,t);
26 Norm_2_imp = (trapz(t,g.^2))^(1/2);
```

Listing 4: 2-norm Matlab script: Impulse response method

Using the Parseval theorem, we have that:

$$||G||_2 = (\frac{1}{2\pi}\int_{-\infty}^{\infty}|G(jw)|^2 dw)^{\frac{1}{2}} = (\int_{-\infty}^{\infty} g^2(t)dt)^{\frac{1}{2}} = 0.67082035878$$

Where g(t) is the time domain impulse response of the system.

4. State-space method

```
27
28 %4. State space method
29 sys=ss(G);
30 A=sys.A;
31 B=sys.B;
32 C=sys.C;
33 L=are(A',zeros(2,2),B*B');
34 Norm_2_ss = sqrt(trace(C*L*C'));
```

Listing 5: 2-norm Matlab script: State-space method

Considering a State-Space expression of the system, one can use the following expression for the 2-norm :

$$||G||_2 = \sqrt{trace[CLC^T]}$$

Using that L is given by the following equation (corresponding to line 33 of Listing 5):

$$AL + LA + BB^T = 0$$

Finally, we get that:

$$||G||_2 = 0.67082039325$$

5. Matlab norm command

```
35
36 %5. Matlab norm
37 Norm_2=norm(G);
```

Listing 6: 2-norm Matlab script: matlab norm command

Comparison of the results

Matlab	Resiude theorem	Frequency response	Impulse response	State-space
0.67082039325	0.67082039325	0.67076697955	0.67082035878	0.67082039325

Table 1: 2-Norm values comparison

As can be seen in Table 1 the frequency response and the impulse response methods are a little less precise than the residue theorem and the state-space methods (when compared to maltab norm command). This is due to discretization that is necessary for numerical solving. Taking larger boundaries with more discrete points for time and frequency would lead to more precise results, but at the price of more computation time, which in this case in unnecessary.

1.1.2 Infinity-Norm

1. Frequency response method

```
38
39 %% 1.1.2. inf-Norm
40 %1. Frequency response
41 w = 0:0.01:4; % Magnitude peak at approx 2 rad/s
42 H = freqresp(G,w);
43 Norm_inf_freq=max(abs(H));
```

Listing 7: 2-norm Matlab script: Frequency repsonse method

By definition of the stable transfer function, we have that the maximum of the magnitude of the transfert function in frequency domain gives the infinity norm:

$$||G||_\infty = \sup_w |G(jw)| = 0.68605830418$$

Where the set of ω to look into is first observed graphically, then it is more finely chosen in order to precisely approximate the value of the maximum of the magnitude of the transfer function.

2. Bounded real lemma method

```
44
45 %2. Bounded real
46 y=1;
47 error=1;
48 while error>10^-5
49     H = [A y^(-2)*(B*B');
50     -C'*C -A'];
51     e=eig(H);
52     error=abs(real(e(1)));
53     y=y-0.00001;
54 end
55 Norm_inf_br=y;
```

Listing 8: 2-norm Matlab script: Bounded real lemma method

Considering a State-Space expression of the system, one can use the first part of the Lemma 1.2 from chapter 1.3 of the class:

Consider a strictly proper stable LTI system and $\gamma > 0$. Then $||G||_\infty < \gamma$, if and only if (1) the Hamiltonian matrix H has no eigenvalue on the imaginary axis; where:

$$H = \begin{bmatrix} A & \gamma^2 BB^T \\ -C^T C & -A^T \end{bmatrix}$$

Therefore, H was iteratively computed for a set of different values of γ. The eigen values of H were checked; if it was too far to the imaginary axis (equivalently, the real part of the eigen value is not yet small enough), then gamma was decreased, and H computed again. Once the real part of the eigen value reached a value that is small enough (i.e. it reaches the stop criterion), -but not yet 0-, the algorithm stops. The result found was given to be:

$$||G||_\infty = 0.68606$$

3. Matlab norm command

```
56
57 %3. Matlab norm
58 Norm_inf=norm(G,inf);
```

Listing 9: 2-norm Matlab script: Matlab norm command

Matlab	Frequency response	Bounded real lemma
0.68605830418	0.68606598092	0.68606

Table 2: Infinity-Norm values comparison

As can be seen in Table 2 the bounded real lemma has a slightly closer value to that computed by matlab. This is due to the ability to set the precision one wants (see line 53). The frequency response suffers from the same impreciseness as in the 2-norm estimation due to discretization.

1.2 Uncertainty modelling

1.2.1 Deterministic Uncertainty

```
1 %% 1.2 Uncertainty modeling
2 clear all;
3 close all;
4 clc;
5 s=tf('s');
6 a_0=2;
7 G=(s+a_0)/(s^2+2*s+5);  % Define the transfer function
8
9 %% 1.2.1 Deterministic uncertainty
10
11 % Weighting Filter (Analytical expression)
12
13 Delta=-1:0.01:1;
14 a=a_0+Delta;
15 W2=1/(s+a_0);        % Define the weighting filter
16 h0=0;                % Criterion of maximum peak starts at 0
17
18 for i=1:length(a)
19     Gk=(s+a(i))/(s^2+2*s+5); % G is parametrized with the values of "a"
20     H=Gk/G-1; % The remaining "filter" is given by H
21     hpeak = getPeakGain(H); % Compute the maximum amplitude of each G
22     if hpeak>h0     % Find the highest value of magnitude of the filters
23         h0=hpeak;  % Store the highest value of amplitude
24         Hmax=H;   % Store the transfer function with max amplitude
25         a_max=a(i); % Find which value of "a" gives the highest amplitude
26     end
27 end
28
29 % For the multi plot purpose
30 aa=1:0.35:3;
31 for j=1:length(aa)              % Plot for several values of "a"
32     GGk=(s+aa(j))/(s^2+2*s+5); % GGk is parametrized with the values of "a"
33     HH=GGk/G-1;
34     figure(1)
35     hold on
36     bodemag(HH)
37     legend('off')
38 end
39
40 figure(1)
41 hold on
42 bodemag(Hmax, 'k--') % plot H which has the highest amplitude
43 bodemag(W2, 'r--')   % plot weighting filter (analytical)
44
45 % Weighting Filter (using Matlab)
46
```

```
aa=ureal('aa',2);
G_uncertain=tf([1,aa],[1 2 5]);
samples=usample(G_uncertain,100);
[usys,Info]=ucover(samples,G);
W2_matlab=tf(Info.W1)

figure(2)
hold on
bodemag(W2_matlab)  % plot Matlab weighting filter
bodemag(W2, 'r--')  % plot weighting filter (analytical)
legend('Matlab weighting filter','Maximum theoretical filter W2')
hold off
```

Listing 10: Deterministic Uncertainty Matlab script

1. Weighting filter for multiplicative uncertainty, analytical solution

We can compute the analytical formulation of the multiplicative uncertainty. In order to do so, we want to express the true model $\tilde{G}(s)$ as a function of the nominal model $G(s)$, and the weighting filter $W_2(s)$ for a multiplicative uncertainty, as given below:

$$\tilde{G} = G(s)(1 + \Delta W_2(s))$$

First, we introduce the uncertainty of the parameter "a" as $a = a_0 + \Delta = 2 + \Delta$, and $-1 < \Delta < 1$, which gives the following expression:

$$\tilde{G}(s) = \frac{s + a}{s^2 + 2s + 5} = \frac{s + a_0 + \Delta}{s^2 + 2s + 5}$$

Thus, by setting that:

$$\frac{\Delta}{s^2 + 2s + 5} = G(s) \cdot \Delta W_2(s)$$

It comes that:

$$\frac{\Delta}{s^2 + 2s + 5} = \frac{s + a_0}{s^2 + 2s + 5} \cdot \Delta \cdot \frac{1}{s + a_0}$$

Finally, the expression of the weighting filter of the multiplicative parametric uncertainty is identified:

$$W_2(s) = \frac{1}{s + a_0} = \frac{1}{s + 2}$$

Using the following formula, one can compute the Bode Diagram on Matlab. However, it is interesting to see how the theory can be verified using Matlab. Therefore, we propose to simulate the uncertain multiplicative model for all values of a with a step of 0.05, compute the filter H for each set of a, and find an upper bound of H. This is done through lines 13 to 34 of listing 10.

Using the function "**getPeakGain**", we find that the residual filter H that has the highest magnitude among all possible values of a is for $a = 1$, thus $\Delta = -1$. Finally, it can be observed on Figure 1 that this maximum of H (in black) is still bounded in magnitude by the analytical expression of $W_2(s)$ that we found previously (in dashed red), which was expected by the theory. The rest of the curves represent

the filter for different set of parameter. The value of a is not indicated since we only want to show the upper bound of the analytical solution of the weighting filter.

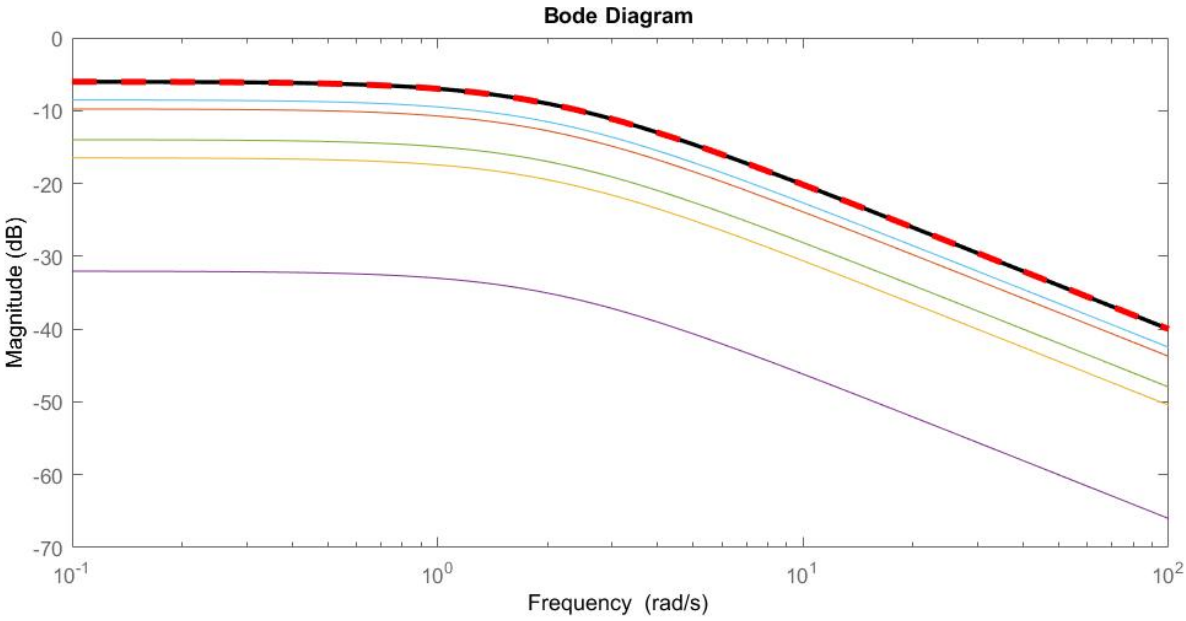

Figure 1: Comparison of W2 found analytically with maximum filter found within the set of "a". In black, the maximum H reach within rage of a; In red dashed line, the maximum weighting filter (found analytical)

2. Weighting filter for multiplicative uncertainty, using matlab

Lines 38 to 42 of 10 show how the filter is computed using matlab. The filter computed by matlab is the following:

$$W_{2,\ matlab} = \frac{0.0005501s + 0.9904}{s + 2}$$

3. Comparison of the weighting filters

The bode diagram of $W_2(s)$ found both analytically and using matlab is displayed on Figure 2.

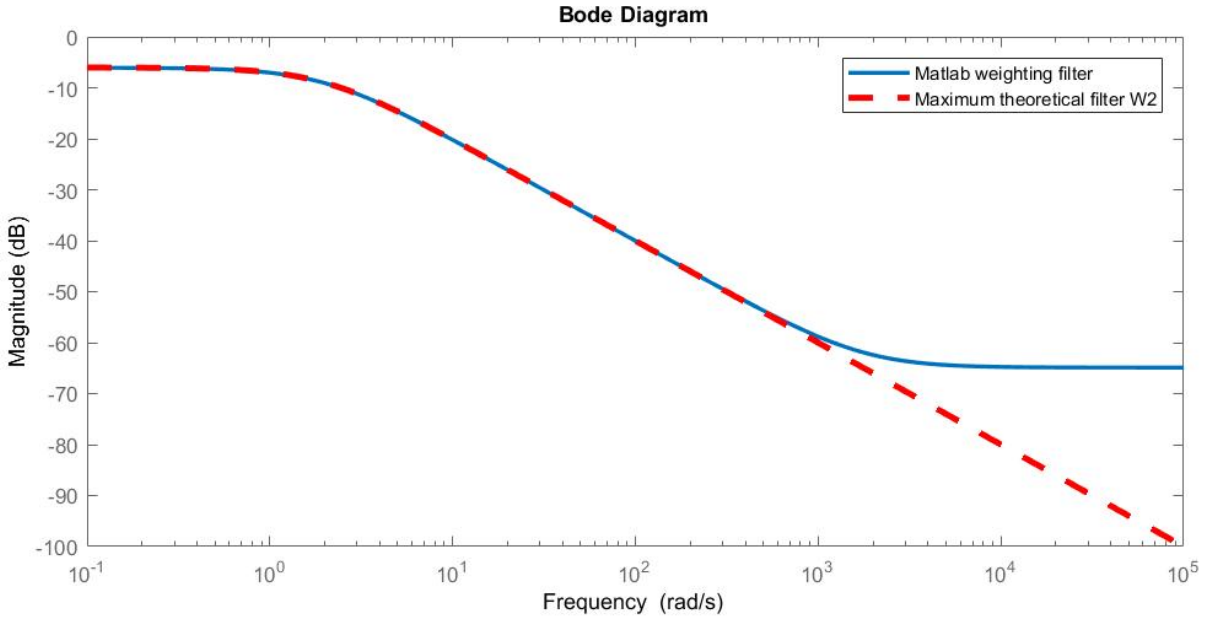

Figure 2: Comparison of W2 found analytically with W2 computed by Matlab

As we can see, Matlab tends to overestimate the magnitude of the filter at high frequencies, but converges to the same filter as the analytical one for low frequencies. Therefore, regarding results from Figure 1 as well, it is clear that both filter are correct since they give an upper bound to the magnitude of the uncertainty. Furthermore, and not surprisingly, computing the filters analytically seem to lead to upper bounds that are lower the numerical ones, so it is likely that it is preferable to use analytical expression whenever it is available.

Apart from that, it is observed that the order of the numerator is higher than the analytical one (1 degree higher). This might probably be due to numerical approximation (the coefficient of the higher order being very small, equal to 0.0005501). Matlab still gives accurate results sinceit approximates the zero order member of the numerator as equal to 0.9904, against 1 for the analytical model.

1.2.2 Stochastic Uncertainty

1. Mean value and variance

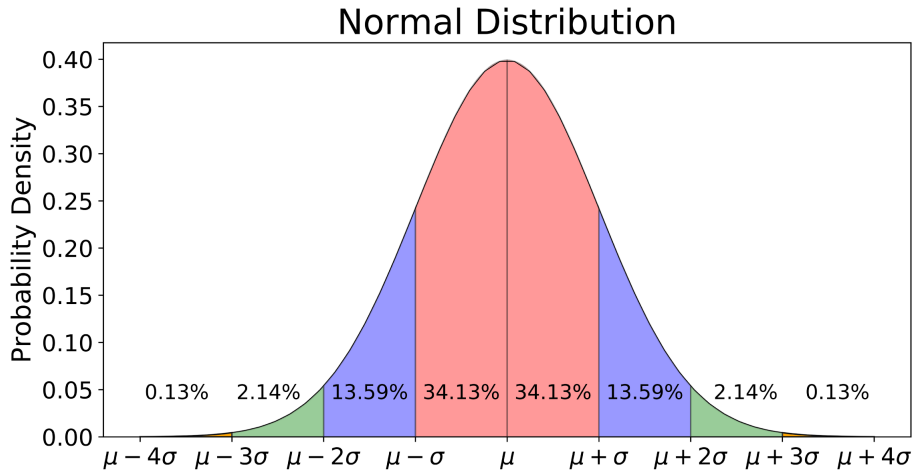

Figure 3: Normal (Gaussian) distribution

To compute the mean value of a we can use the following simple equation:

$$a_{mean} = \frac{a_{max} + a_{min}}{2} \tag{1}$$

Which in our case equals **2**. For calculating the variance, we must take into account that a has a Gaussian distribution and the probability of $a \in [1, 3]$ is 99%. The 99% confidence interval can be found using the following equation:

$$0.99 = erf(\frac{x}{\sqrt{2}}) \tag{2}$$

Where x indicates the amount of times the standard deviation is multiplied so that the surface under the Gaussian probability density function between $a_{mean} \pm x\sigma$ is 99% of the total surface. We find that x equals 2.57583. This allows us to calculate the standard deviation by using the following relation:

$$a_{max} - a_{mean} = x\sigma \tag{3}$$

We find that the standard deviation is therefore 0.38822. By taking the square of σ we find the variance of a is **0.1507**.

2. Theoretical size of multiplicative uncertainty

The co-variance of the parametric model is defined as follows:

$$C_G(w) = \left(\frac{\partial \hat{G}_{vec}(jw)}{\partial a}\right) \cdot cov(a) \cdot \left(\frac{\partial \hat{G}_{vec}(jw)}{\partial a}\right)^T \tag{4}$$

Where $\hat{G}_{vec}(jw)$ is given by:

$$\hat{G}_{vec}(jw) = [R_e\{\hat{G}(jw)\}I_m\{\hat{G}(jw)\}] \tag{5}$$

In the case of our model, $\hat{G}(jw)$ is defined as:

$$\hat{G}(jw) = \frac{jw + a}{(5 - w^2) + 2jw} \tag{6}$$

Decomposing the relation above, we can find $R_e\{\hat{G}(jw)\}$ and $I_m\{\hat{G}(jw)\}$ as follows:

$$R_e\{\hat{G}(jw)\} = \frac{2w^2 + a(5 - w^2)}{(5 - w^2)^2 + 4w^2} \tag{7}$$

$$I_m\{\hat{G}(jw)\} = \frac{w(5 - w^2) - 2wa}{(5 - w^2)^2 + 4w^2} \tag{8}$$

Computing the derivative of the imaginary and the real part with respect to a we find that:

$$\frac{\partial R_e\{\hat{G}(jw)\}}{\partial a} = \frac{5 - w^2}{(5 - w^2)^2 + 4w^2} \tag{9}$$

$$\frac{\partial I_m\{\hat{G}(jw)\}}{\partial a} = -\frac{2w}{(5 - w^2)^2 + 4w^2} \tag{10}$$

The co-variance can finally be written in its fully developed form as:

$$C_G(w) = \frac{cov(a)}{((5 - w^2)^2 + 4w^2)^2} \begin{pmatrix} (5 - w^2)^2 & -2w(5 - w^2) \\ -2w(5 - w^2) & 4w^2 \end{pmatrix}$$

$cov(a)$ is the covariance of a, which is nothing other than the variance calculated in the question 1. The magnitude of the uncertainty ellipses is defined by the eigenvalues of C_G. Respectively the maximum eigenvalue represents the semi-major axis and the minimum eigenvalue represents the semi-minor axis of the ellipse. Therefore, we can calculate the eigenvalues and eigenvectors (defining the orientation in the plane of the ellipse) to later plot the uncertainty ellipses. The eigenvalues of the co-variance are as follows:

$$\lambda_1 = 0 \tag{11}$$

$$\lambda_2 = \frac{cov(a)}{(5 - w^2)^2 + 4w^2} \tag{12}$$

The size of uncertainty is given by the radius of the smallest circle that covers an uncertainty ellipse for a probability of 0.95% with a χ_2 distribution. It can be computed by multiplying the square-root of the largest eigenvalue (λ_2) by $\sqrt{5.99}$ at selected frequencies.

ω [rad/s]	2	3	4	5	6	7
Size of uncertainty	0.2305	0.1318	0.0699	0.0425	0.0286	0.0206

Table 3: Size of multiplicative stochastic uncertainty for different frequencies

The computation of the uncertainty ellipse and circle are done using the Matlab function *plotellipse* in Listing 11 and plotted in Figure 4.

```
1
2  function [ellipse,circle]=plotellipse(Sigma)
3
4      s=-2*log(1-0.95); % 95% confidence interval
5
6      [V, D] = eig(Sigma*s);
7
8      t = linspace(0, 2 * pi);
9      ellipse = (V *sqrt(D)) * [cos(t(:))'; sin(t(:))'];
10     circle = (V * eye(2)*sqrt(D(4))) * [cos(t(:))'; sin(t(:))'];
11 end
```

Listing 11: Matlab function plotellipse

3. Stochastic and deterministic comparison

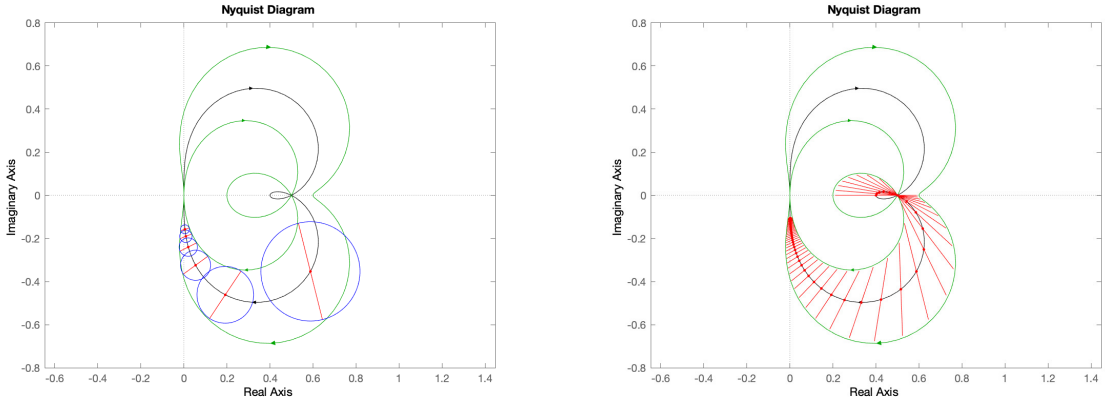

Figure 4: Nyquist plot with W2 and W3 uncertainty (left figure with W3 uncertainty circle)

In Figure 4 the nominal model is plotted in black, the nominal model with the deterministic uncertainty is plotted in green and the radius of the stochastic uncertainty is plotted in blue with the uncertainty ellipse (a line given the zero eigenvalue) in red. As can be clearly seen the stochastic uncertainty is slightly more conservative, especially at low frequencies, than the deterministic uncertainty. However, the conservatism of the stochastic uncertainty is based on the confidence interval for the magnitude of uncertainty we chose (95% probability). By choosing a lower probability we will lower the confidence interval, which consequently reduces the size of the W3 uncertainty.

```
clear variables;
close all;
%% 1.2.2. Stochastic uncertainty

%Model transfer function
s=tf('s');
a0=2;
G=(s+a0)/(s^2+2*s+5);

%Theoretical W2 deterministic weighting filter
W2_mult=1/(s+2);

%Compute covariance matrix
syms omega a real

cov_a=0.1507; %varible covariance
Ghat=((1i*omega)+a)/((1i*omega)^2+2*(1i*omega)+5);
Gvec=[real(Ghat),imag(Ghat)];
Cg=jacobian(Gvec,a)*cov_a*jacobian(Gvec,a)';

% plot model and deterministic uncertainty
nyquist(G,'k')
hold on
nyquist(G*(1+W2_mult),'g')
hold on
nyquist(G*(1-W2_mult),'g')

W3=[];
for w=2:1:7

    %Define points on the model nyquist plot at specific frequencies
    Ghat_value=double(subs(Ghat,[omega,a],[w,a0]));
    Greal=real(Ghat_value);
    Gimag=imag(Ghat_value);

    %Define covariance matrix at specific frequencies
    Cg_value=double(subs(Cg,omega,w));

    %Compute the size of multiplicative uncertainty
    e=eig(Cg_value);
```

```
    W3=[W3, sqrt(e(2))*sqrt(5.9915)];

    %Calculate and plot W3 uncertainty ellipse and circle
    [ellipse,circle]=plotellipse(Cg_value);
    hold on
    plot(Greal,Gimag,'r.');
    hold on
    plot(ellipse(1,:) + Greal, ellipse(2,:) + Gimag,'r');
    hold on
    plot(circle(1,:) + Greal, circle(2,:) + Gimag,'Color','Blue');
end

axis([-0.2,1,-0.8,0.8])
axis equal
```

Listing 12: Matlab script for stochastic uncertainty estimation and comparison

2 Robust Control of an Active Suspension System

The aim of this report is to convert multi-model uncertainty to multiplicative uncertainty, and design several controllers: model-based H_∞ controller, model-based H_2 controller, and data-driven H_∞ controller.

2.1 Multiplicative uncertainty

```
clear all;
close all;
clc;

load ASdata.mat;

%% 2.1.1 Plot freq. responses

% Ts=0.02 %sec
Zd1=detrend(ASdata{1});
G1 = oe(Zd1,[4 4 1]);
G1f=spa(Zd1,100);

Zd2=detrend(ASdata{2});
G2 = oe(Zd2,[4 4 1]);
G2f=spa(Zd2,100);

Zd3=detrend(ASdata{3});
G3 = oe(Zd3,[4 4 1]);
G3f=spa(Zd3,100);

Zd4=detrend(ASdata{4});
G4 = oe(Zd4,[4 4 1]);
G4f=spa(Zd4,100);

figure(1)   % Plot the frequency response of the non-parametric (spa) ID
bodemag(G1f,G2f,G3f,G4f)
legend('G1f','G2f','G3f','G4f')

figure(2)   % Plot the frequency response of the parametric (OE) ID
bodemag(G1,G2,G3,G4)
legend('G1','G2','G3','G4')

% We present the result using both models (parametric and non-parametric)

%%  2.1.2 Compute the weighting filter using parametric ID

Gf=stack(1,G1,G2,G3,G4,G1f,G2f,G3f,G4f);
orderw2=10;

% Change the "Nominal model", (consider it to be G1, G2, G3 or G4)
[Gu1, Info1]=ucover(Gf,G1,orderw2,'InputMult');
W2_1=Info1.W1;
[Gu2, Info2]=ucover(Gf,G2,orderw2,'InputMult');
W2_2=Info2.W1;
```

```
[Gu3, Info3]=ucover(Gf,G3,orderw2,'InputMult');
W2_3=Info3.W1;
[Gu4, Info4]=ucover(Gf,G4,orderw2,'InputMult');
W2_4=Info4.W1;

%%  2.1.2 Compute the weighting filter using spectral analysis ID

% Change the "Nominal model", (consider it to be G1f, G2f, G3f or G4f)
[Gu1f, Info1f]=ucover(Gf,G1f,orderw2,'InputMult');
W2_1f=Info1f.W1;
[Gu2f, Info2f]=ucover(Gf,G2f,orderw2,'InputMult');
W2_2f=Info2f.W1;
[Gu3f, Info3f]=ucover(Gf,G3f,orderw2,'InputMult');
W2_3f=Info3f.W1;
[Gu4f, Info4f]=ucover(Gf,G4f,orderw2,'InputMult');
W2_4f=Info4f.W1;

%% Graphical Observations
% comparison of Weghting filters using parametric models
figure(3)
hold on
title('Weighting filters using OE parametric models')
bodemag(W2_1, 'r', W2_2, 'g', W2_3, 'k', W2_4, 'm');
legend('W2 using G1 as nom','W2 using G2 as nom', 'W2 using G3 as nom', 'W2 using G4 as
    nom')
hold off

% Graphical comparison of Weghting filters using non-parametric models
figure(4)
hold on
title('Weighting filters using Spectral Analysis identification')
bodemag(W2_1f, 'r', W2_2f, 'g', W2_3f, 'k', W2_4f, 'm');
legend('W2 using G1 as nom','W2 using G2 as nom', 'W2 using G3 as nom', 'W2 using G4 as
    nom')
hold off

%% Method of selection of the best model using norms:

% Loop the order of the weighting filter and compute 2-norms and
% infinity-norms iteratively. Then, compute the average of the norms for
% each model. The model with smallest average 2-norm and/or infinity norm
% will be the less conservative, i.e. the best choice.

order_loop=5:15;    % Orders to be tested

norm1=[];
norm2=[];
norm3=[];
norm4=[];
norm1_inf=[];
norm2_inf=[];
norm3_inf=[];
norm4_inf=[];
norm1f=[];
norm2f=[];
norm3f=[];
norm4f=[];
norm1f_inf=[];
norm2f_inf=[];
norm3f_inf=[];
norm4f_inf=[];

% Compute W2 filter for each model and for each order
for k=1:length(order_loop)
    order=order_loop(k);

    [Gu1, Info1]=ucover(Gf,G1,order,'InputMult');
    W2_1=Info1.W1;
    [Gu2, Info2]=ucover(Gf,G2,order,'InputMult');
    W2_2=Info2.W1;
    [Gu3, Info3]=ucover(Gf,G3,order,'InputMult');
    W2_3=Info3.W1;
    [Gu4, Info4]=ucover(Gf,G4,order,'InputMult');
    W2_4=Info4.W1;

    [Gu1f, Info1f]=ucover(Gf,G1f,order,'InputMult');
```

```
    W2_1f=Info1f.W1;
    [Gu2f, Info2f]=ucover(Gf,G2f,order,'InputMult');
    W2_2f=Info2f.W1;
    [Gu3f, Info3f]=ucover(Gf,G3f,order,'InputMult');
    W2_3f=Info3f.W1;
    [Gu4f, Info4f]=ucover(Gf,G4f,order,'InputMult');
    W2_4f=Info4f.W1;

    % Store the value of the norms in vectors
    norm1=[norm1;norm(W2_1)];
    norm2=[norm2;norm(W2_2)];
    norm3=[norm3;norm(W2_3)];
    norm4=[norm4;norm(W2_4)];
    norm1_inf=[norm1_inf;norm(W2_1,inf)];
    norm2_inf=[norm2_inf;norm(W2_2,inf)];
    norm3_inf=[norm3_inf;norm(W2_3,inf)];
    norm4_inf=[norm4_inf;norm(W2_4,inf)];
    norm1f=[norm1f;norm(W2_1f)];
    norm2f=[norm2f;norm(W2_2f)];
    norm3f=[norm3f;norm(W2_3f)];
    norm4f=[norm4f;norm(W2_4f)];
    norm1f_inf=[norm1f_inf;norm(W2_1f,inf)];
    norm2f_inf=[norm2f_inf;norm(W2_2f,inf)];
    norm3f_inf=[norm3f_inf;norm(W2_3f,inf)];
    norm4f_inf=[norm4f_inf;norm(W2_4f,inf)];
end

%% Choice of the model

% 1/ Compute average the 2-NORMS and INF-NORM of the OE and SPA models
Av_norm_OE=[];
Av_norm_SPA=[];
for i=1:8
    if i<5  % Concatenate the average of 2-norm values
        Av_norm_OE=[Av_norm_OE; mean(eval(sprintf('norm%d',i)))];
        Av_norm_SPA=[Av_norm_SPA; mean(eval(sprintf('norm%df',i)))];

    else    % Concatenate the average inf-norm values
        Av_norm_OE=[Av_norm_OE; mean(eval(sprintf('norm%d_inf',(i-4))))];
        Av_norm_SPA=[Av_norm_SPA; mean(eval(sprintf('norm%df_inf',(i-4))))];
    end
end

% 2/ Find the minimum values of 2-norm and inf-norms for OE and SPA models
% 2-Norm average minimums:
[Av_min_OE2, pos_OE2]=min(Av_norm_OE(1:4))
[Av_min_SPA2, pos_SPA2]=min(Av_norm_SPA(1:4))

% Infinity-Norm average minimums:
[Av_min_OEinf, pos_OEinf]=min(Av_norm_OE(5:8))
[Av_min_SPAinf, pos_SPAinf]=min(Av_norm_SPA(5:8))

% >> Model 2 - SPA as nominal is chosen, it gives minimum 2-norm <<

% 3/ Choose the final order of the filter within the interval [5:15]
% Plot the changes in norm depending on the filter order
figure(10)
hold on
title('Norms of the filter of the 2nd model VS order of the filter')
plot(order_loop, norm2f, '-*m')
plot(order_loop, norm2f_inf, '--b')
legend('2-norm of SPA','Inf-norm of SPA')
hold off

% >> Order 10 is selected graphically (it reduces 2 norm and inf-norms) <<

% 4/ Check that for an order of 10, the chosen model is still the best:
N=[];
for i=1:8   % Concatenate 2-norm of all models for order 10 in an array
    if i<5
        N2nom=eval(sprintf('norm%df',i));
        N=[N;N2nom(6)];
    else
        N2nom=eval(sprintf('norm%df_inf',(i-4)));
        N=[N;N2nom(6)];
    end
```

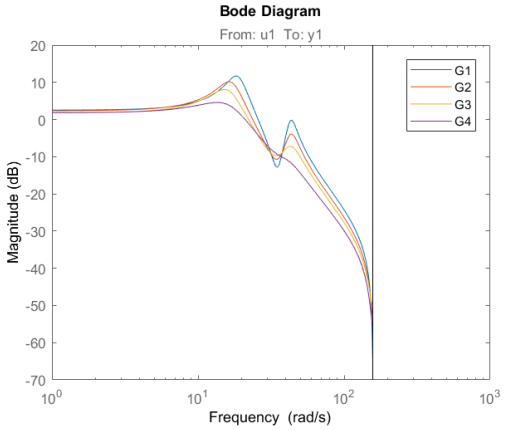

Figure 5: Frequency response of the 4 models using parametric OE models

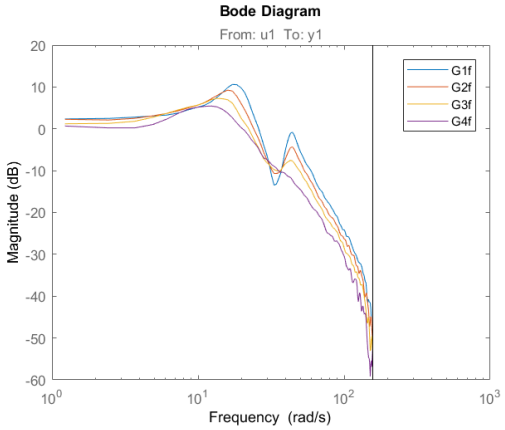

Figure 6: Frequency response of the 4 models using Spectral Analysis

```
196 end
197 % Check for minimum value of this array
198 [N2min, whichNominal2]=min(N(1:4))
199 [Ninfmin, whichNominalf]=min(N(5:8))
200
201 %% Final Choice:
202 [Gu2f, Info2]=ucover(Gf,G2f,10,'InputMult');
203 W2_2=Info2.W1;
204
205 [Tf_W2a, Tf_W2b]=ss2tf(W2_2.A,W2_2.B,W2_2.C,W2_2.D)
```

Listing 13: Matlab script for conversion from Multi-model uncertainty to Multiplicative uncertainty, and Weighting filter computation

2.1.1 Conversion from multi-model uncertainty to multiplicative uncertainty

All references in this subsection relate to Listing 13.

There are 4 sets of data available. First, each set of data (input/output of the plant) is used to fit a parametric Output-Error model, and a non-parametric frequency domain model using Spectral Analysis (lines 10-24). The two models are more or less equivalent, but still have several differences (Output Error provides a smooth frequency response model, as shown in Figure 5, so it might underestimate the order of the system; whereas the spectral analysis provides a model that captures all dynamics of the system at all frequencies, but resulting in a noisier model, as shown in Figure 6).

Then, all the models are stacked together to build a multi-model (line 38). Using the matlab command **ucover**, a nominal model is chosen (i.e. a model considered as the "true model"), and the multi-model is converted to multiplicative uncertainty around it. This is done with every available model (resulting from either data set 1, 2, 3 or 4, and using OE or SPA) (lines 43-62). Finally, this results in 8 different filters for multiplicative uncertainty, which are shown on Figures 7 using OE models, and 8 using SPA models. The problem is now to select the filter that gives the best performance.

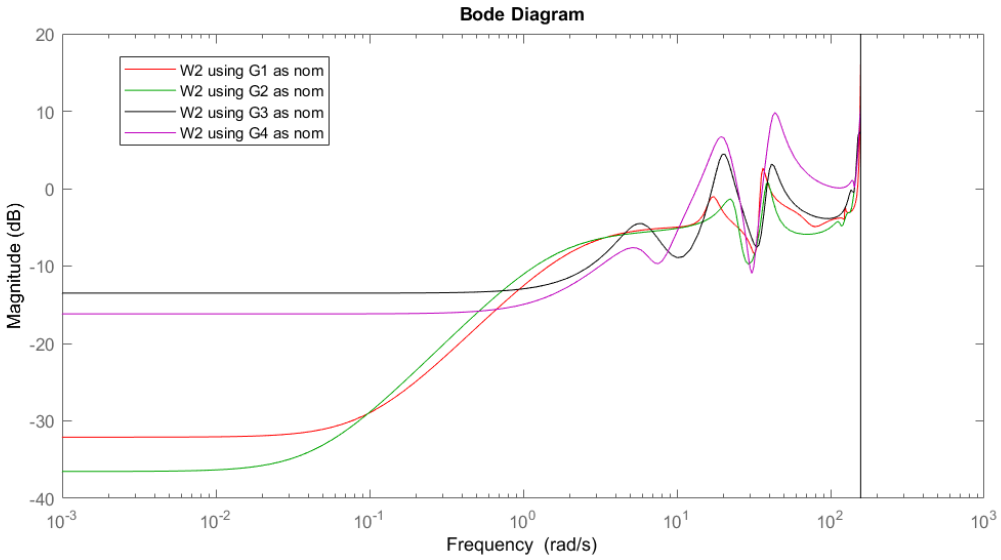

Figure 7: Weighting filter for Multiplicative Uncertainty using the OE models

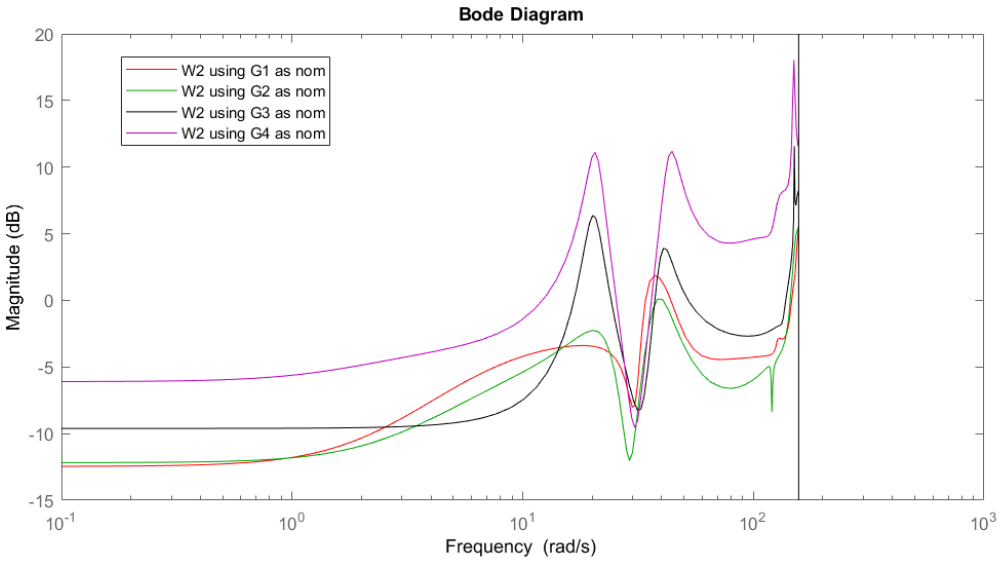

Figure 8: Weighting filter for Multiplicative Uncertainty using the Spectral Analysis models

Looking at the above filters, it seems rather complex to find out which one is the least conservative, i.e. which one has the lowest amplitude in the bode diagram. As a consequence, we decided to design a method to select the "best nominal model".

2.1.2 Choice of the best model

In order to choose the best model as nominal model, and the best order for the multiplicative filter, we chose to compute the average of the $2-norm$ and $\infty-norm$ over all the nominal models, for a range of order of the multiplicative weighting filters. In matlab, we proceed as follows:

1. lines 88-105: Choose a range for the order of the weighting filter (here 5 to 15), and create empty vectors to store values of norms for each model and each norm type;

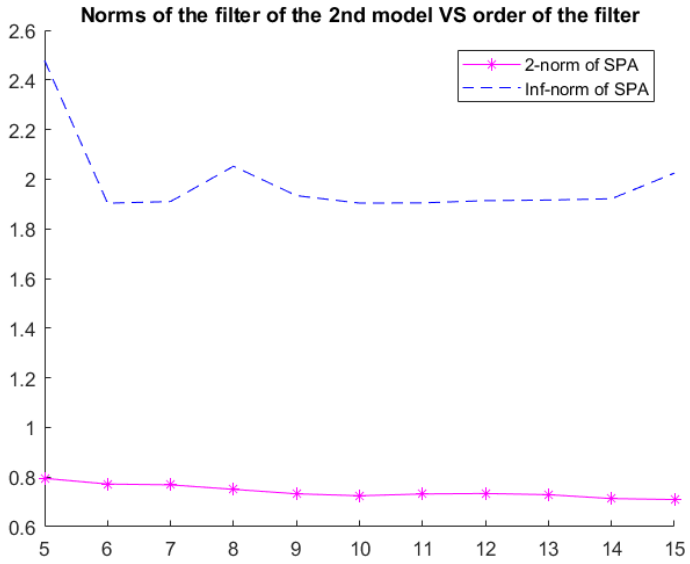

Figure 9: Norm values of W_2 depending on its order

2. lines 108-146: Loop over the filter order, and compute iteratively the filter W2 for each model, each norm and each order, and store it;
3. lines 150-162: Compute the average of the $2-norm$ and $\infty-norm$ for each model and put all the averages in a vector (here it is separated for OE and SPA);
4. lines 166-170: Find the minimum of the average for each type of norm, for OE and SPA;
5. line 172: Choose the result that gives the lowest $2-norm$ and/or $\infty-norm$, i.e. that is be the least conservative.

Note: The $2-norm$ and $\infty-norm$ represent different measures of distance. In a way, the $2-norm$ gives an average of the variation of the signal (here, the signal is the filter W_2), equally for all frequencies. On the other hand, the $\infty-norm$ will rather characterize the signal's variation depending on its peak values, i.e. at given frequencies. In our case, the average of the $2-norm$, which gives a more general measure of distance, seems appropriate to find which model is the least conservative. However, if we were to minimize amplification in a frequency range of the filter, the $\infty-norm$ would be more appropriate.

	G1 - OE	G2 - OE	G3 - OE	G4 - OE	G1 - SPA	G2 - SPA	G3 - SPA	G4 - SPA
$2-norm$	0.9116	0.829	1.0171	1.4735	0.7634	0.7419	1.1226	2.4905
$\infty-norm$	6.7422	2.8706	2.4071	3.3486	2.009	1.9874	7.9295	7.8366

Table 4: Average norms of the weighting filters for each nominal model

Table 4 gives the value of the $2-norm$ and $\infty-norm$ for all nominal models. The least $2-norm$ is given by the second model for an SPA model. It also gives the smallest $\infty-norm$.

Finally, the order of the weighting filter has to be selected. Figure 9 shows the variations of the $2-norm$ and $\infty-norm$ depending on the order of the filter. As can be seen, an order of 10 gives (graphically) the minimum 2-norm, and a very low value for the $\infty-norm$. Therefore, the 2nd model using spa and a 10th order weighting filter is chosen.

lines 188-198 are a check to verify that model 2-SPA is the best among all models for an order of 10 (it is the case for the $2-norm$, and second best for $\infty-norm$). It gives the final following values for the norms of W_2 of the model 2, SPA, order 10: $2-norm = 0.7247$; $\infty-norm = 1.9035$. The final function is shown on Figure 10. Line 205 shows gives the continuous-time transfer function coefficients for W_2.

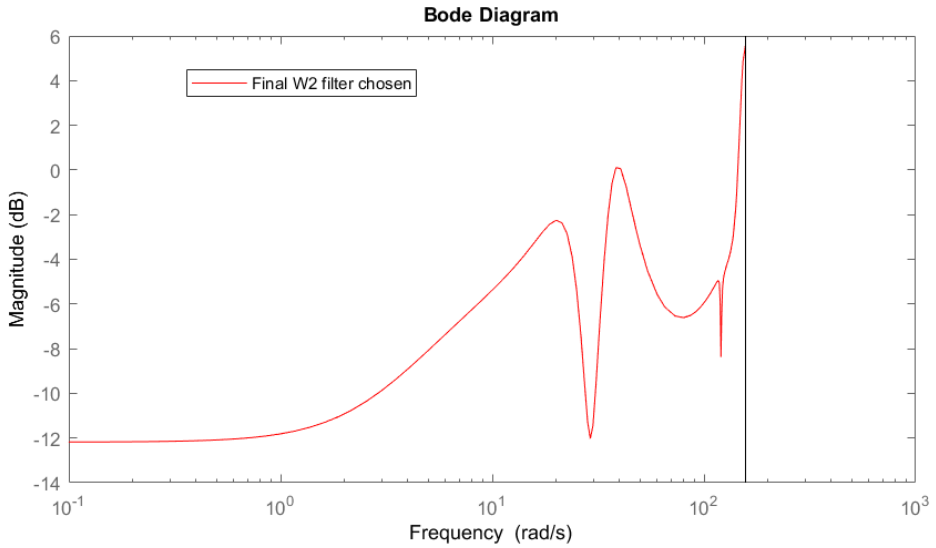

Figure 10: Chosen weighting filter

2.2 Model-based $\mathcal{H}_\infty$ control design

```
clear variables;
close all;

%% load data and compute models ==========================================
load ASdata.mat;

Ts=0.02; % sampling time [s]

Zd1=detrend(ASdata{1});
G1=oe(Zd1,[4 4 1]);
G1f=spa(Zd1,100);

Zd2=detrend(ASdata{2});
G2=oe(Zd2,[4 4 1]);

Zd3=detrend(ASdata{3});
G3=oe(Zd3,[4 4 1]);

Zd4=detrend(ASdata{4});
G4=oe(Zd4,[4 4 1]);

%% 1. Design the weighting filter W1 ======================================
wb=20; % desired closed-loop bandwidth
m=0.5; % modulus margin

s=tf('s');
W1 = (s+wb)*m/(s+0.0001); % add a stable "integrator" (with pole very close to 0)
W1d=c2d(W1,Ts); % discretize W1

%% 2. Retrieve multiplicative uncertainty from ex 1 =======================
load W2.mat;

W2=W2_2; % Multiplicative uncertainty

%% 3. & 4. Closed-loop system dynamics ====================================
W3=1/20;

K=mixsyn(G2,W1d,W3,W2); % Hinf controlled using mixed sensitivity approach

G=stack(1,G1,G2,G3,G4);

T=feedback(G*K,1); % closed-loop system
U=feedback(K,G); % input sensitivity function
S=feedback(1,G*K); % sensitivity function

figure(1)
subplot(2,2,1);
step(T)
title('Step response')

subplot(2,2,2);
step(U);
title('Control signal')

subplot(2,2,3);
bodemag(U)
hold on
plot([1 157],20*log10([1/W3 1/W3]),'-r')
title('Sensitivity function U')

subplot(2,2,4);
bodemag(S)
hold on
bodemag(1/W1d)
title('Sensitivity function S')

%% 5. Reduce controller order =============================================

figure(2)
pzmap(K) % look at zero-pole cancellation

Kred=reduce(K,4); % reduce controller order

Tred=feedback(G*Kred,1);
```

```
Ured=feedback(Kred,G);
Sred=feedback(1,G*Kred);

figure(3)
subplot(2,2,1);
step(Tred)
title('Step response')

subplot(2,2,2);
step(Ured);
title('Control signal')

subplot(2,2,3);
bodemag(Ured)
hold on
plot([0.01 157],20*log10([1/W3 1/W3]),'-r')
title('Sensitivity function U')

subplot(2,2,4);
bodemag(Sred)
hold on
bodemag(1/W1d)
title('Sensitivity function S')
```

Listing 14: Matlab script for Model-based $\mathcal{H}_\infty$ control design

Weighting filter W1 design

In order to design a discrete-time $\mathcal{H}_\infty$ controller for the active suspension system using the mixed sensitivity method, it is necessary to first design an appropriate W1 weighting filter. The desired specifications are a closed-loop bandwidth of 20 rad/s, zero steady-state tracking error for a step response and a modulus margin of at least 0.5. A continuous time weighting filter is therefore designed as follows:

$$W_1(s)^{-1} = \frac{s}{m(s+\omega_b)} \tag{13}$$

Where m is the modulus margin and ω_b the desired closed-loop bandwidth. An additional integrator is placed in order to have the zero steady-state tracking error for a step response. However, given that W1 must be stable, the pole of the integrator must be slightly displaced from zero. Which finally gives:

$$W_1(s) = \frac{0.5s + 10}{s + 0.0001} \tag{14}$$

The amplitude of the inverse of W1 at high frequencies is $20log(1/m)$, which equals 6.02 dB.

Weighting filter W3 design

W3 is implemented in order to reduce the magnitude of $U(e^{j\omega})$ and consequently the magnitude of u(t) since the control signal u(t) should be less than 3v to avoid saturation in real-time implementation and the magnitude of $U(e^{j\omega})$ should be less than 20dB in high frequencies in order to avoid the amplification of the measurement noise on the actuator. The final value obtained for W3 is **1/20**. With this value u(t) is maximum 2.97 v and the magnitude of $U(e^{j\omega})$ is maximum 18.7 dB.

Controller order reduction

The order of the controller obtained from the mixed sensitivity method is 20. This order is extremely large and therefore reducing it is good idea. Looking at the pole-zero map of K, it is clear that there are plenty of zero-pole cancellations. Therefore the order of K can be reduced by removing these zero-pole

cancellations. Taking into account the plot of the Hankel singular values, which define the "energy" of each state, as well as the visible zero-pole cancellations, the final controller order 4 is chosen.

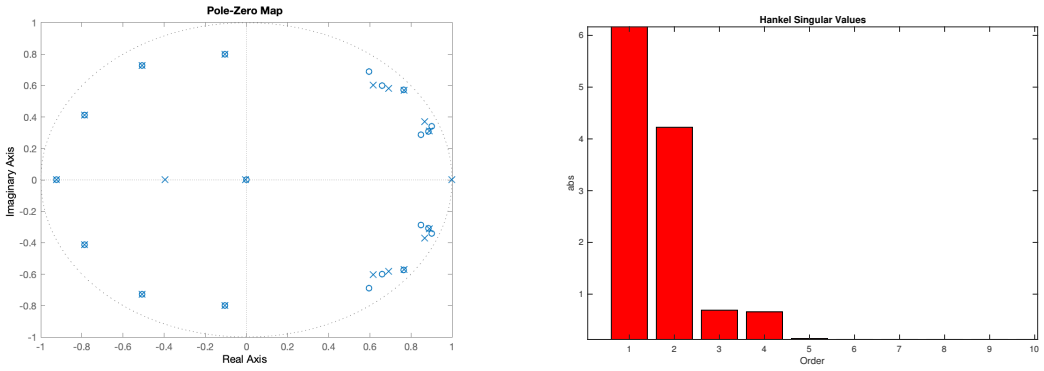

Figure 11: Pole-zero map of K (left) and Hankel singular values of K (right)

Using function *ss2tf* in matlab, the continuous time transfer function for K was computed:

$$K(s) = \frac{2.9743s^4 - 8.9860s^3 + 11.5808s^2 - 7.5633s + 2.1357}{s^4 - 1.9698s^3 + 1.2029s^2 + 0.0742s - 0.3072}$$

Sensitivity plots and step responses

Finally, after reducing the controller, the step response, the control signal and the bode plots of the sensitivity functions U and S can be plotted.

The stability and performance of the closed-loop system is good and not overly conservative. The steady-state error is zero as the output of the step response of all systems goes to 1. It is clear that the control input is not too large and that the magnitude of sensitivity function U is lower than 20 dB. Also the closed-loop bandwidths of the different models are all close to the desired 20 rad/s as can be seen in Table 5. Finally, the closed-loop system is stabilized for all models and the performance specifications are met.

G1	G2	G3	G4
12.4460	16.8731	16.3006	14.5228

Table 5: Closed-loop bandwidth (rad/s)

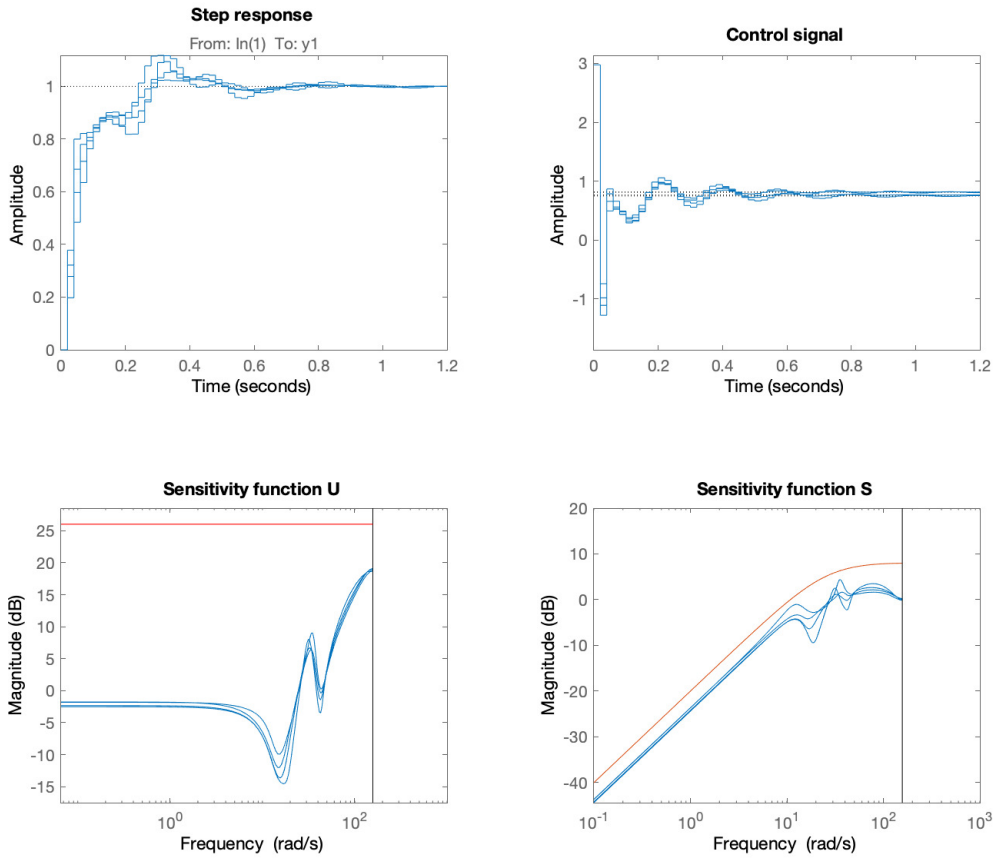

Figure 12: Matlab plot of the sensitivity functions and step responses

2.3 Model-Based $\mathcal{H}_2$ Controller Design

```
clear variables;
close all;

%% 1. load data and compute model ========================================
load ASdata.mat;

Zd1=detrend(ASdata{1});
G1=oe(Zd1,[4 4 1]);
G1c=d2c(G1);

%% 2. Retrieve state-space matrices of G1 ================================
G1ss=ss(G1c);
A=G1ss.A;
B=G1ss.B;
C=G1ss.C;

%% 3. Define LMI convex optimization problem =============================
n=4;
Y=sdpvar(1,n);
L=sdpvar(n,n,'symmetric');
alpha=sdpvar(1,1);
beta=sdpvar(1,1);
lmi=A*L+L*A'-B*Y-Y'*B'+B*B'<=0;
lmi=[lmi, C*L*C' <= alpha];
lmi=[lmi, [beta Y;Y' L] >= 0];
options=sdpsettings('solver','mosek'); %lmilab

%% 4. Compute controller solving the SDP problem =========================

optimize(lmi,alpha+beta,options);
L=value(L);
Y=value(Y);
K=Y*L^(-1);

%% 5. Plot step response of the CL system ================================

sys_cl = ss(A-B*K,B,[C;-K],0);
figure
step(sys_cl)

%% 6. Compute LQR controller =============================================

R=1;
Q=C'*C;
Klqr=lqr(A,B,Q,R);

sys_cl_lqr = ss(A-B*Klqr,B,[C;-Klqr],0);
figure
step(sys_cl_lqr)
```

Listing 15: Matlab script for Model-Based $\mathcal{H}_2$ Controller Design

State-space equations of the close-loop system

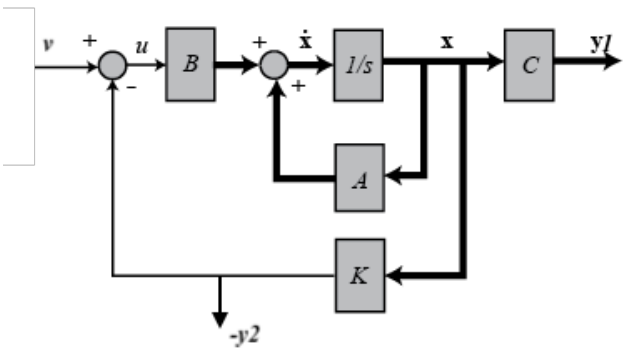

Figure 13: Steady-state with controller feedback block model

The block model of the system with a state feedback controller is displayed in Figure 13. As can be seen, two outputs are of interest, namely y1 and y2. The state-space equations of the closed-loop system can be written as follows:

$$\dot{x} = Ax + Bu = (A - BK)x + Bv \tag{15}$$
$$y1 = Cx \tag{16}$$
$$y2 = -Kx \tag{17}$$

Convex optimization problem

The minimization of the sum of the two-norm of the closed loop transfer functions from the input disturbance to the output (y1) and to the input of the system (y2) can be converted to a convex optimization problem with the following LMIs:

$$AL + LA^T - BY - Y^T B^T + BB^T < 0 \tag{18}$$
$$\begin{bmatrix} \beta & Y \\ Y^T & L \end{bmatrix} > 0 \tag{19}$$
$$CLC^T < \alpha \tag{20}$$

Where $\alpha + \beta$ needs to be minimized. Equation (18) flows from the state-space method to compute the two-norm of a transfer function with a subtle variable substitution for K: $Y = KL$. Equation (19) flows from the necessity to minimize the trace of KLK^T. Given that this is not an LMI, Schur compliments are used to transform the inequality into an appropriate format. First, noticing that KLK^T is a scalar, the trace can be replaced by the matrix multiplication. Secondly, the minimization can be written in an alternative way as follows:

$$trace(KLK^T) = KLK^T < \beta \tag{21}$$

Where β must be minimized. This equation can be rearranged to obtain:

$$\beta - KLK^T > 0 \tag{22}$$

Taking into account that $L > 0$, the two equations can be combined into an LMI as in Equation (19). Finally, Equation (20) flows from the necessity to minimize the trace of CLC^T which is once more a scalar. Now that 4 LMIs have been identified with 4 unknowns, YALMIP can be used to solve them with the minimization of $\alpha + \beta$ using MOSEK as a solver.

Note: $L > 0$ can additionally be added to the list of LMIs. However, given that it is already inexplicitly defined in Equation (19)*, this would mean having 5 LMIs for 4 unknowns, producing an over-determined system. The difference in the final K matrix is extremely slight and the system without the extra LMI returns the same controller as the LQR method. Therefore it was not added.*

State-feedback controller

The controller obtained from solving the SDP problem is:

$$K = \begin{bmatrix} 2.0929 & 0.6993 & 3.4086 & 1.6750 \end{bmatrix} \tag{23}$$

Step response of the closed-loop system

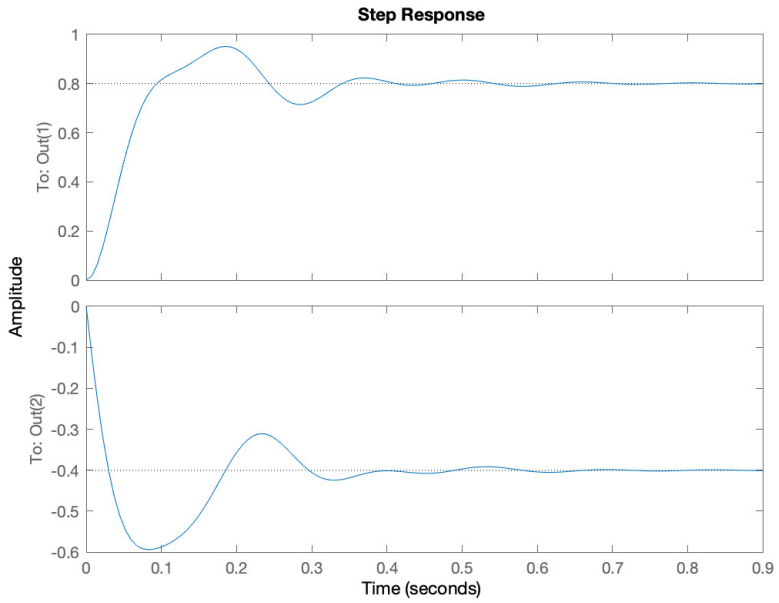

Figure 14: Step response of the closed-loop system (out(1) = y1, out(2) = y2)

LQR method

In order to compute an LQR controller, the weighting matrices Q and R must be defined. The output vector of the system can be written as follows:

$$Y = \begin{bmatrix} y1 \\ y2 \end{bmatrix} = \begin{bmatrix} Cx \\ -Kx \end{bmatrix} \tag{24}$$

Taking the square of the norm of the output vector, the following equation is derived:

$$Y^T Y = (y1^T y1 + y2^T y2) = x^T C^T C x + x^T K^T K x \tag{25}$$

The LQR method calculates the optimal gain matrix K needed to minimize the quadratic cost function. It is possible to make use of this definition and Equation (25) to identify the weight matrix. Minimizing the sum of the norm of the output at each time we obtain:

$$min \int_0^\infty (Y^T Y)dt = min \int_0^\infty (x^T C^T C x + x^T K^T K x)dt = min \int_0^\infty (x^T Q x + u^T R u)dt = min\ J(u) \tag{26}$$

Where u=-Kx. It is clear from the Equation (26) that the weigth matrix Q is therefore $C^T C$ and that R is 1. Implementing this in Matlab gives the same controller matrix:

$$K_{lqr} = \begin{bmatrix} 2.0929 & 0.6993 & 3.4086 & 1.6750 \end{bmatrix} \tag{27}$$

2.4 Data-driven control design

```
clear all;
close all;
clc;

%% load path and files
load ASdata.mat;
javaaddpath 'C:\Program Files\MATLAB\R2017a\Mosek\8\tools\platform\win64x86\bin\
    mosekmatlab.jar'
addpath 'C:\Program Files\MATLAB\R2017a\Mosek\8\toolbox\r2014a'

%% Retrieve data and create weighting filters & K_init
Ts=0.02;
m=0.5;
wb=20;

% Create all models from available data
Zd1=detrend(ASdata{1});
G1 = oe(Zd1,[4 4 1]);
G1f=spa(Zd1,100);

Zd2=detrend(ASdata{2});
G2 = oe(Zd2,[4 4 1]);
G2f=spa(Zd2,100);

Zd3=detrend(ASdata{3});
G3 = oe(Zd3,[4 4 1]);
G3f=spa(Zd3,100);

Zd4=detrend(ASdata{4});
G4 = oe(Zd4,[4 4 1]);
G4f=spa(Zd4,100);

G1f=spa(Zd1,100);
G=stack(1,G2,G1,G3,G4,G1f,G2f,G3f,G4f);

% Create weighting filters
w=G1f.frequency;
s=tf('s');
objW1=1/s;       % optimized for Step input
cW1=(s+wb)*m/(s);
cW3=tf(1,6);

% Create initial controller Kinit with fixed pole & K0=0.1 (low init.gain)
z=tf('z', Ts);
Kinit=0.01/(z-1);
orderW=4;
Gx=stack(1,G2,G1,G3,G4) % Create a smaller stack to test initial controller
T_init=feedback(Kinit*Gx,1); % Closed-loop transfer function to check
                             % stability of the initial controller
figure(7)
pzmap(T_init)

%% Run solver to compute the controller
P= data_driven('G', G, 'W', w, 'Kinit', Kinit, 'order', orderW,'cinfW1', cW1, 'cinfW3',
    cW3, 'o2W1', objW1, 'Ts', Ts , 'tol', 0.0005, 'maxiter', 10, 'exit', 1e-5)
[K,obj] = solve(P)

%% Plot same figures as in 2.2.3

G=stack(1,G2,G1,G3,G4);
T=feedback(G*K,1);
U=feedback(K,G); % input sensitivity function
S=feedback(1,G*K); % sensitivity function
t=0:0.02:1.2;

figure(2)
subplot(2,2,1);
step(T,t)
title('Step response')

subplot(2,2,2);
step(U,t);
title('Control signal')

```

```
subplot(2,2,3);
bodemag(U)
hold on
bodemag(1/cW3)
title('Sensitivity function U')

subplot(2,2,4);
bodemag(S)
hold on
bodemag(1/cW1)
title('Sensitivity function S')
```

Listing 16: Matlab script for Data-driven Controller Design

2.4.1 Constraints and minimization criteria

As written in the code, there are several inputs, constraints and objective in the optimization problem.

Inputs
First, the order of the controller is set to 4, as it was a reasonable choice for the model-based H_∞ controller design. An initial controller is given as:

$$K_{init} = K_0 \cdot \frac{1}{(z-1)} = \frac{0.01}{(z-1)}$$

The value of $K_0 = 0.01$ is due to the fact that the initial controller needs to stabilize the system. Therefore, a low value for initial gain provides stability in most cases. On the other hand, we force the controller to have an integrator (given by the term $\frac{1}{(z-1)}$), leading to the zero steady state error* demanded in the requirements specifications of the controller. The validity of the initial controller is ensured by checking the closed-loop stability of the system using the initial controller. For this purpose, Figure 15 shows the zero-pole cancellation of the transfer function with initial controller. Since there are no pole lying outside the unit circle, the closed-loop system is initially stable, and can be used as an input for the data-driven algorithm.

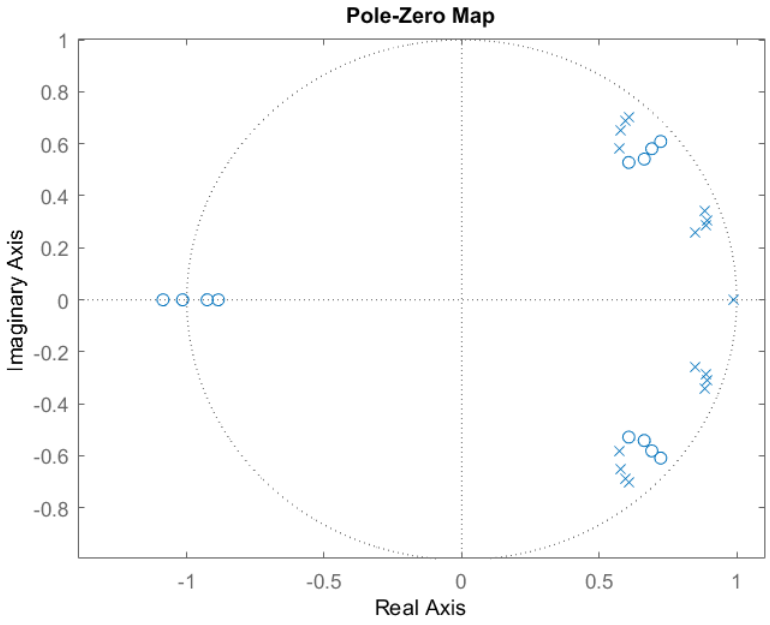

Figure 15: Zero-pole cancellation of the closed-loop transfer function using the initial controller

**Note: From theorem 1.3 (page 17 of Robust and Adaptive Control, march 2016) it is known that if the*

feedback system is internally stable then: "if r(t) is a step, e(t) → 0 as t → ∞ if and only if S has at least one zero at the origin". By forcing an integrator in the controller K, the sensitivity function has automatically a forced zero at the origin.

Constraints
The constraints on the optimization problem are given by the weighting filters that define:

1. the closed-loop bandwidth (using W_1), as required (around 20rad/s);
2. the amplitude of the input (using W_3) was tuned-up iteratively to $W_3 = \frac{1}{6}$ in order to bound the amplitude of the input ($3V$) and avoid saturation;

These constraints are considered to be achievable, but if it was not the case, the algorithm would have to violate them, and might not converge to a solution.

Objective function
There is a specification on improving the tracking performance by minimizing the 2-norm of the tracking error. Thus, the second objective function is given by minimizing the $2 - norm$ of $\frac{1}{s}S$. The reason for this is that the sensitivity function is the transfer function from the input to the tracking error. The aim is to increase the system performance for a step input, which is defined in the Laplace domain as $\frac{1}{s}$, therefore the tracking error becomes the multiplication of the step input and the sensitivity function.

$$||e||_2 = ||rS||_2 = ||\frac{1}{s}S||_2 \tag{28}$$

2.4.2 Results

Results of Sensitivity functions, step response and control signal for the chosen controller are shown on figure 16. The resulting controller K(s) is given by the following expression:

$$K(z) = \frac{3.119z^4 - 7.189z^3 + 5.707z^2 - 1.622z + 0.08257}{z^4 - 1.445z^3 + 0.2229z^2 + 0.1452z + 0.07735}$$

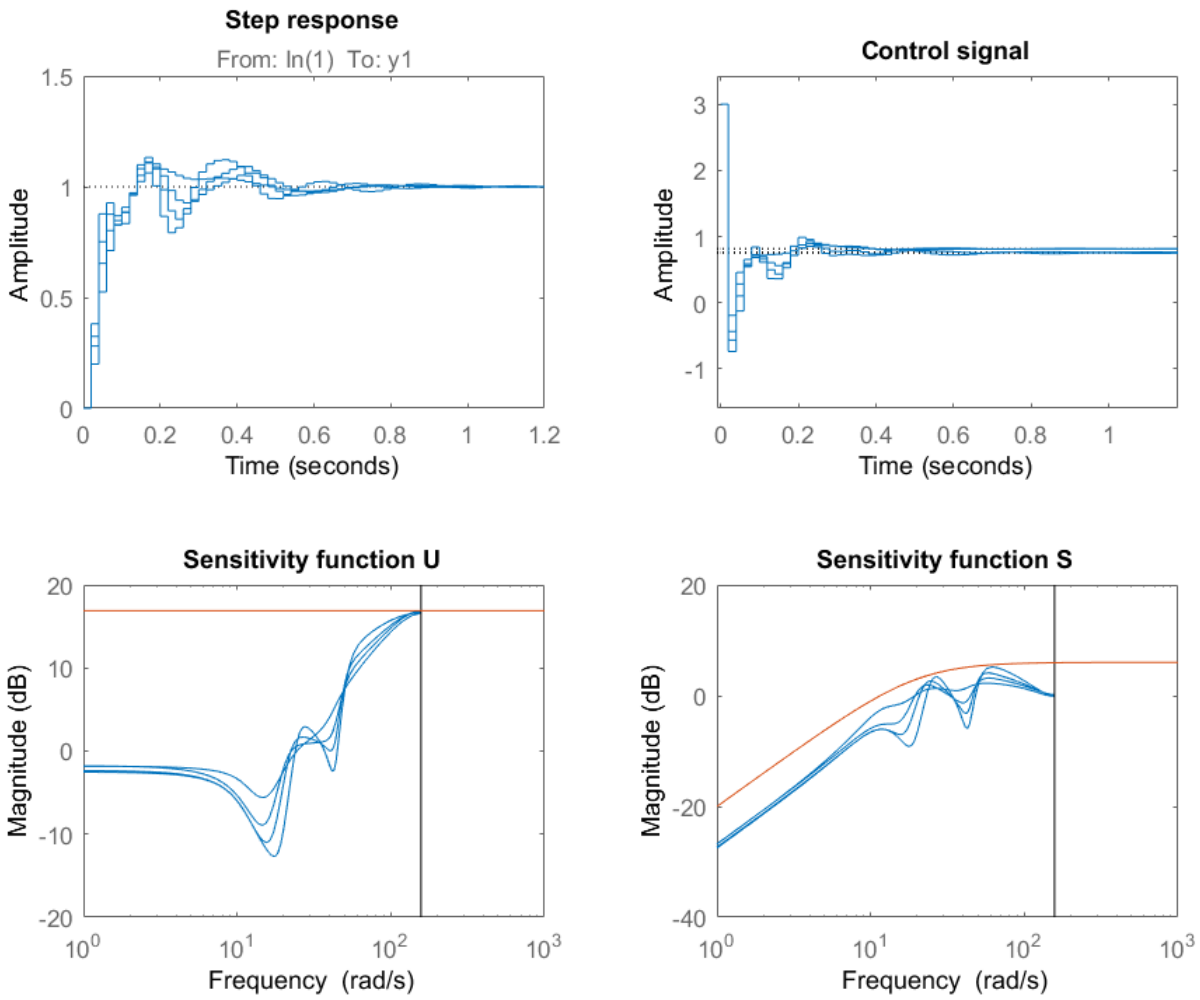

Figure 16: Results of the data-driven controller design, Sensitivity functions and step responses

2.4.3 Comparison with model-based controller

First of all, the control signal of the data-driven controller converges faster to steady state than that of the H_∞ controller. Graphically, the data-driven controller seems to be less conservative because the distance between W_1 and S is almost 0 in Figure 16, compared to Figure 12 for the H_∞ controller. Finally, the objective functions reaches $H_2 = 0.1477$ and a minimization value of the ∞-norm of $H_\infty = 0$, which seems perfect for a minimization problem. The main drawback could be to access only too few data of the plant, possibly leading to a poorer controller performance.

3 RST controller design

3.1 Pole placement

3.1.1 Orders of A, B and d

For designing the primary RST controller, the G1 nominal model is considered. The vectors containing the polynomial parameters of A and B are displayed below as well as their respective orders.

$$A = (1.0000, -2.9613, 3.8152, -2.5124, 0.7274) \tag{29}$$
$$B = (0, 0.1081, -0.0493, -0.0502, 0.0815) \tag{30}$$

n_A	n_B	d
4	4	1

3.1.2 RST Pole placement function

```
function [R,S]=poleplace(B,A,Hr,Hs,P)

Aprime = conv(A,Hs);
naprime=length(Aprime)-1;

Bprime = conv(B,Hr);
nbprime=length(Bprime)-1;

nsprime=nbprime-1;

M=zeros(naprime+nbprime);
for i=1:nbprime
    M(i:i+naprime,i)=Aprime';
end

for i=nbprime+1:nbprime+naprime
    M(i-nbprime:i,i)=Bprime';
end

p=zeros(naprime+nbprime,1);
p(1:length(P),1)=P';
x=M^-1*p;

Sprime=x(1:nsprime+1,1)';
Rprime=x(2+nsprime:end,1)';

S=conv(Sprime,Hs);
R=conv(Rprime,Hr);

end
```

Listing 17: Matlab Pole placement function

The above function computes the coefficients of polynomial $R(q^{-1})$ and $S(q^{-1})$ that places the closed-loop poles at $P(q^{-1})$, while taking into account the fixed parts $H_S(q^{-1})$ and $H_R(q^{-1})$.

The inputs of the function the vectors A, B, Hr, Hs and P. A and B are the coefficients of the polynomials of the plant transfert operator (denominator and numerator respectively). $H_S(q^{-1})$ and $H_R(q^{-1})$ are the filters ensuring: (1) zero steady state error thanks to $H_S = (1, -1)$; (2) operate in open loop at $f_{Nyquist}$ by using $H_R = (1, 1)$. P is the polynomial specifying the closed loop poles. To summarize, the function constructs the Sylvester matrix M in order to solve for R and S, considering all the inputs. The equation solved can be described by:

$$A(q^{-1})S'(q^{-1})H_S(q^{-1}) + q^{-d}B(q^{-1})R'(q^{-1})H_R(q^{-1}) = P(q^{-1}) \tag{31}$$

and

$$R(q^{-1}) = R'(q^{-1})H_R(q^{-1})$$
$$S(q^{-1}) = S'(q^{-1})H_S(q^{-1})$$

The only difference is that it is solved in the form of a (Sylvester) matrix (for each order of q^{-1}).

3.1.3 Closed-loop characteristic polynomial

```
function P=closedlooppoles(Ts,Tr,OS)

xi = sqrt(log(OS)^2 / (pi^2 + log(OS)^2)); % damping ratio
wn = (2.16*xi + 0.6) / Tr; % natural frequency
p1 = -2*exp(-xi*wn*Ts)*cos(wn*Ts*sqrt(1-xi^2));
p2 = exp(-2*xi*wn*Ts);

P=[1,p1,p2];
end
```

Listing 18: Matlab function to compute closed-loop characteristic polynomial coefficients

The desired closed-loop performance is defined by the polynomial $P = 1 + p_1 + p_2$, usually chosen to be of second order. The function above allows to compute the polynomial coefficients of this closed-loop characteristic polynomial in order to place the dominant closed-loop poles in such a way to obtain a rise time of Tr and an overshoot of less than OS%. Ts is the sampling time.

The system being chosen as the discretzation of a second order continuous-time system, it can be expressed as:

$$H(s) = \frac{{\omega_n}^2}{s^2 + 2\xi\omega_n s + {\omega_n}^2}$$

Then, recalling that for a step input, the overshoot can be given as:

$$e^{\frac{-\xi\pi}{\sqrt{1-\xi^2}}} = OS$$

We obtained an expressed for ξ for a given OS:

$$\xi = \sqrt{\frac{ln(OS)^2}{\pi^2 + ln(OS)^2}}$$

We also have that the rise time to go from 10% to 90% of the final value is given by:

$$T_r = \frac{2.16\xi + 0.6}{\omega_n}$$

Thus, solving for ω_n, we obtain:

$$\omega_n = \frac{2.16\xi + 0.6}{T_r}$$

By defining that the rise time be 0.1 seconds and that the overshoot be less than 5%, the following damping and natural frequency are obtained:

$$\omega_n = 20.9063 \quad [rad/s] \tag{32}$$
$$\xi = 0.6901 \tag{33}$$

The resulting coefficients are therefore given by:

$$p_1 = -2e^{-\xi\omega_{ns}} \cdot cos(\omega_n T_s\sqrt{1-\xi^2})$$

$$p_2 = e^{-2\xi\omega_{ns}}$$

Finally, the following values are found:

$$P = (1, p_1, p_2) = (1, -1.4306, 0.5615) \tag{34}$$

3.1.4 R and S polynomials

The polynomials $R(q^{-1})$ and $S(q^{-1})$ obtained for given the desired closed-loop poles, the fixed integrator and the nominal model G1 are as follows:

$$R = (12.3967, -29.1007, 42.7839, -40.4036, 15.7761) \tag{35}$$
$$S = (1.0000, 0.1905, 1.0683, -0.4906, -1.7681) \tag{36}$$

If additionally a fixed part in R is added to reduce the noise at high frequencies ($H_R = 1 + q^{-1}$), then the R and S polynomials become:

$$R = (6.0216, -3.8468, -0.4171, -0.0646, -4.8783, 4.6376) \tag{37}$$
$$S = (1.0000, 0.8797, 0.0644, -0.4957, -0.9286, -0.5198) \tag{38}$$

3.1.5 Achieved closed-loop poles

As desired, the closed-loop polynomial obtained is the one prescribed in Equation (34). This results in the following poles and zeros (the p indicates the poles and not the polynomial coefficients):

$$p_1 = 0.7153 + 0.2233i \tag{39}$$
$$p_2 = 0.7153 - 0.2233i \tag{40}$$
$$z_1 = 0 \tag{41}$$
$$z_2 = 0 \tag{42}$$

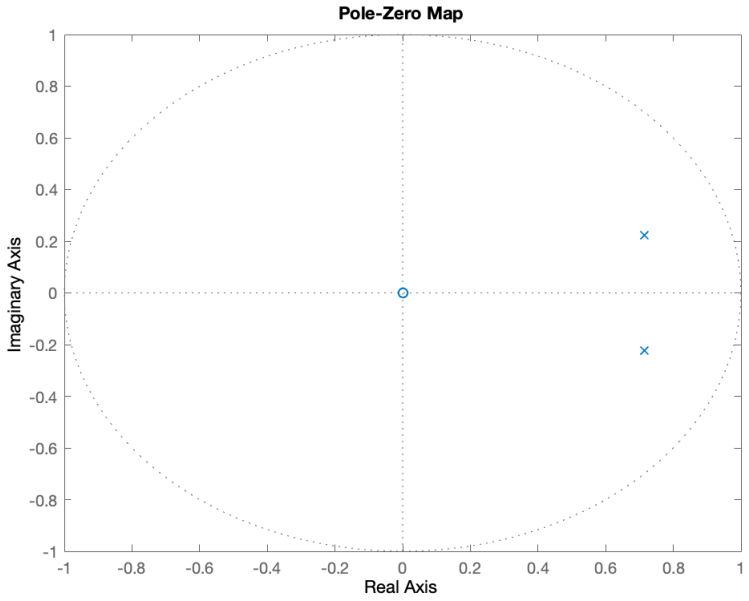

Figure 17: pole-zero map of 1/P

It is clear that they are within the unit circle and therefore the closed-loop system with the nominal model is stable.

3.1.6 T polynomial

Given that an integrator is present in the S polynomial and that the same dynamics for tracking and regulation is wished, the T polynomial can easily be computed by summing all coefficients in R from Equation (35), which results in the following constant:

$$T = 1.4524 \tag{43}$$

T remains the same with and without the fixed term in R.

3.2 Analysis of the closed-loop system

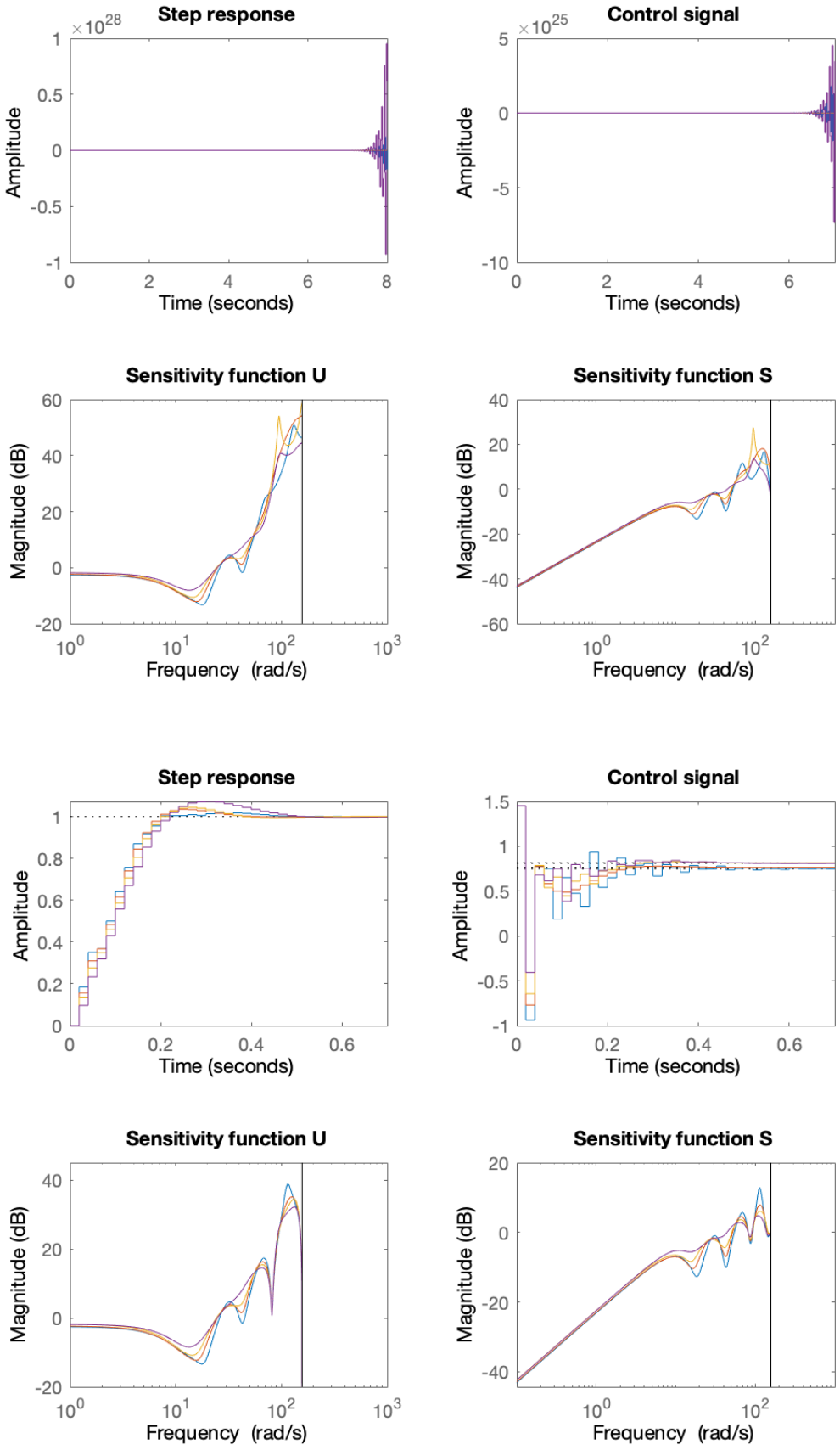

Figure 18: RST controller performance comparison without (top) and with (bottom) H_r

Table 6: Performance parameters comparison between RST controllers

	H_r	**Rise time [s]**	**Overshoot [%]**	**Settling time [s]**	M_m	$\|\|U\|\|_\infty$
G0	without	NaN	NaN	NaN	0.1456	50.8813
	with	0.1400	1.6972	0.2000	0.2271	38.8792
G1	without	0.1400	6.3918	0.4200	0.1245	54.2723
	with	0.1400	3.4267	0.3200	0.3981	35.1402
G2	without	0.1400	3.3207	0.3000	0.0422	58.7787
	with	0.1600	4.3953	0.3400	0.4929	34.4631
G3	without	NaN	NaN	NaN	0.2103	44.4979
	with	0.1400	7.1866	0.4400	0.5791	32.2253

In Figure 18, the step response of the system is described without the fixed term H_R (top four plots), and with the fixed term (four bottom plots). From the figure, it is immediately clear that inserting a fixed term in H_R in order to reduce the sensitivity function at high frequencies is necessary as the infinity norm of U reaches dangerous levels for all models without the loop opening. Moreover, for model G0 and G3, the system becomes highly unstable. By adding the term, the infinity norm of U is considerably decreased, while the modulus margin increases, assuring a more robust control and overall stability.

All the combinations of models performances (for $\{G0, G1, G2, G3\}$), with and w/o H_R, are described in Table 6. However, both controllers lead to results that are way off from the necessary safe values prescribed, namely a maximum 20 dB for the infinity norm of U and a minimum 0.5 for the modulus margin. Therefore, this controller cannot be implemented on the real system as is. In terms of the performance with H_R, all plant models are stabilized and converge to a stable steady state. The rise times are very close to the desired 0.1 second, while most models have lower overshoots than 5%. The difference from the desired performance is due to the additional zeros in the closed-loop transfer function provided by B and T as well as the slightly different poles due to the different dynamics of the off nominal models. These zeros have the effect of increasing slightly the rising time, while decreasing the overshoot. The control signal also remains way below the maximum 3 v for all models, ensuring controller compliance with the system.

3.3 Robust RST controller with Q-parameterization

3.3.1 RST controller design by Q-parameterization

```
%% 3.2 Analysis of the closed-loop system ==========================
% 1. Design the weighting filter W1 ==============================

wb=16; % desired closed-loop bandwidth
m=0.5; % modulus margin
s=tf('s');
W1=c2d((s+wb)*m/(s+0.0001),Ts); % discretize W1

% Multiplicative uncertainty filter W2
load W2.mat;
W2=W2_2;

B=G1.b;
A=G1.f;

n=6; % Q order
Q0=zeros(1,n);

fun = @(x) norm(W2*tf(conv(B,[R,zeros(1,n)])+conv(A,conv(Hr,conv(Hs,x)))),P_desired,Ts,'variable','z^-1'),'inf')...
          +norm(W1*tf(conv(A,[S,zeros(1,n)])-conv(B,conv(Hr,conv(Hs,x)))),P_desired,Ts,'variable','z^-1'),'inf');

options.MaxFunctionEvaluations = 3000;
options.MaxIterations = 1000;
[Q,fval]=fmincon(fun,Q0,[],[],[],[],[],[],@(x) mycon(x,n,A,B,S,R,Hr,Hs,P_desired,Ts),options);

Rq=[R,zeros(1,n)]+conv(A,conv(Hr,conv(Hs,Q)));
```

```
Sq=[S, zeros(1,n)]-conv(B, conv(Hr, conv(Hs,Q)));
Tq=sum(Rq);
```

Listing 19: Part of CE_3 Matlab script to compute the coefficients of Q

After an initial controller with desired closed loop poles has been computed, Q-parametrization allows to change the expression of S and R, which will then depend linearly on Q. The main advantage of this method is that it allows the sensitivity function $\boldsymbol{S}$ and input sensitivity function $\boldsymbol{U}$ to be convex functions of Q, while keeping the desired closed-loop poles. On the other hand, the solutions to the Bezout equation will not necessarily be of minimal order. The expression of S and R are expressed below; where S_0 and R_0 are the R and S polynomials found to solve the initial controller (i.e. w/o Q-parametrization).

$$R(q^{-1}) = R_0(q^{-1}) + A(q^{-1})H_R(q^{-1})H_S(q^{-1})Q(q^{-1})$$
$$S(q^{-1}) = S_0(q^{-1}) - q^{-d}B(q^{-1})H_S(q^{-1})H_R(q^{-1})Q(q^{-1})$$

Now that the problem is convex in Q, it is possible to solve a minimization problem and find the global minimum of an objective function expressed in terms of Q. As there are requirements on the modulus margin, as well as on the infinity norm of the input sensitivity function $\boldsymbol{U}$, two constraints on both the infinity norm of U and the modulus margin are put in place (see equation 17 and 18). Then, the objective function can be expressed in terms of "filtered" transfer functions, using the performance filter on $W_1(s)$ and the uncertainty filter $W_2(s)$ computed in a previous assignment. The objective function was chosen as the sum of the infinity norms of both filtered sensitivity functions. From the triangular inequality, it is known that minimizing the sum of the infinity norms leads to the same or better result than minimizing the infinity norm of the sums. Also minimizing on the infinity norms rather than on the two norms is chosen as to lower the maximum magnitudes of all sensitivity functions. This leads to equation 44, which is equivalent to **theorem 1.5** of the class notebook, on the necessary and sufficient condition for robust performance. The order of Q is found by iteratively solving the objective function until a feasible point is reached.

In a nutshell, The objective to be minimized is:

$$||W_1 S||_\infty + ||W_2 T||_\infty \tag{44}$$

While the constraints are:

$$M_m > 0.5 \tag{45}$$
$$||U||_\infty < 20 \quad [dB] \tag{46}$$

The parameters obtained with Q-parametrization are as follows:

$$R_{\,Q-parametrization} = (3.1313, 0.8791, -2.2099, -1.9823, -0.6408, 0.8147, 1.0711, 0.8088, -0.2545,$$
$$-0.4619, -0.3710, 0.6678)$$
$$S_{\,Q-parametrization} = (1.0000, 1.1922, 0.3362, -0.6010, -0.9006, -0.6078, -0.1571, 0.0817, 0.0282,$$
$$-0.1261, -0.1709, -0.0748)$$
$$T_{\,Q-parametrization} = 1.4524$$

While the solution of the objective to be minimized, i.e. the sum of the infinity norms of $W_1 S$ and $W_2 T$, is 1.4867. This is slightly larger than the desired value of 1 or lower but still reasonably good.

```
function [c,ceq] = mycon(x,n,A,B,S,R,Hr,Hs,P,Ts)

c(1)=mag2db(norm(tf(conv(A,[R,zeros(1,n)]+conv(A,conv(Hr,conv(Hs,x)))),P,Ts,'variable','z
    ^-1'),'inf'))-20;
c(2)=0.5-1/norm(tf(conv(A,[S,zeros(1,n)]-conv(B,conv(Hr,conv(Hs,x)))),P,Ts,'variable','z
    ^-1'),'inf');

ceq = [];
end
```

Listing 20: Matlab constraint function to define the constraints of the Q-parameterization

3.3.2 Comparison with and without Q-parametrization

Finally, using Q-parametrization, the following controller was computed. Table 7 shows the performance of the Q-parametrized controller for G0, G1 and G3; and Figure 19 shows the behaviour of all four models with the Q-parametrization.

Table 7: Performance and robustness comparison of different controllers

Model	Rise time [s]	Overshoot [%]	Settling time [s]	M_m	$\|\|U\|\|_\infty$
G0	0.1400	1.4018	0.2000	0.4478	20.7372
G1	0.1400	3.4267	0.3200	0.5371	19.2354
G3	0.1600	7.1134	0.5000	0.7061	18.9092

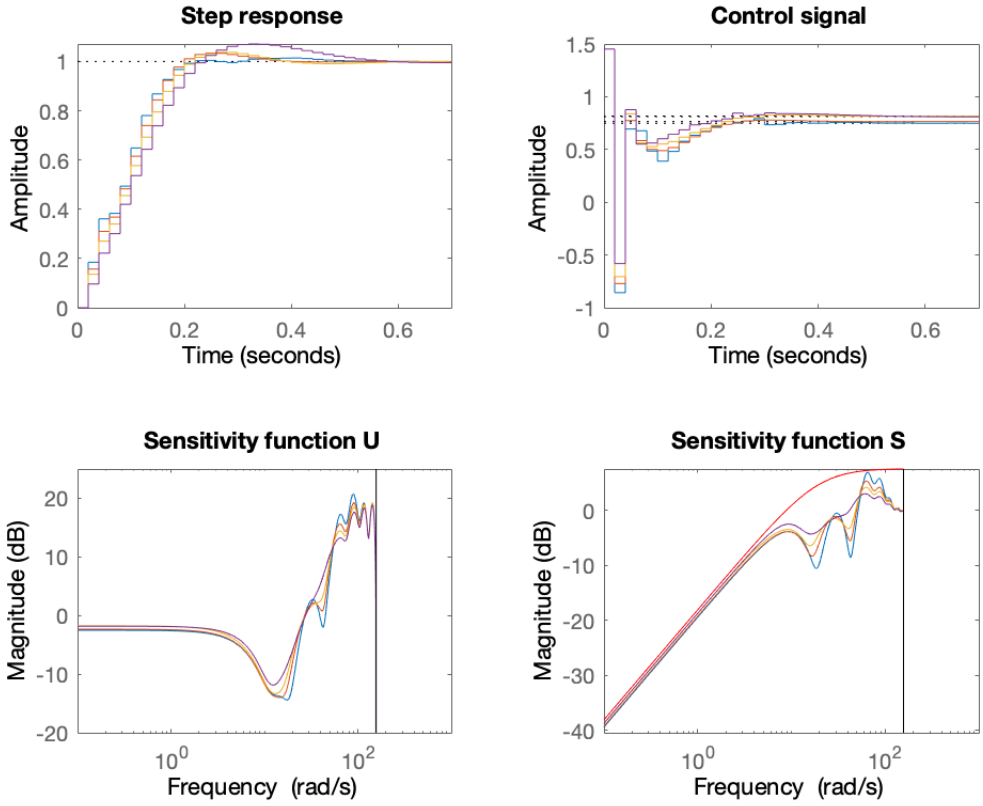

Figure 19: RST controller performance with Q-parametrization

Figure 20 shows a comparison of the controller performances with and without Q-parametrization, both with the nominal model G1. The comparison is based on Table 6 and Table 7. In the Q-RST controller,

G3 has the highest rise time of 0.16sec, whereas in the initial RST-controller, G2 holds the highest rise time of 0.16sec. In fact, G2 has a rise time of 0.14sec for the Q parametrization, but as the extreme values only are shown in Table 7, the value of rise time for G2 for the Q-parametrization is not shown. Thus, it seems like Q parametrization is not much of an improvement for rise time performance. Regarding overshoot, it seems like there is an improvement in the Q-parametrization, as only one model (G3) shows an overshoot of 7.11%. On the other hand, in the initial controller, G1 and G3 showed overshoot to 6.39% and 7.18% respectively. In all the cases, the settling times are reasonable. One can observe however that once more, model G3 in the Q-RST controller has the highest value of settling time (of 0.50sec). Thus, it seems like there is a trade off in the performance of the models in the Q-parametrization: the algorithm seems to converge to an overall better minimization by letting one of the model with to poorer performances (in this case, G3).

However, in the initial RST controller, all input sensitivity functions lead to infinity norms of 32.2 to 38.9dB which are much higher than the specifications. In addition, the modulus margins for the initial controller range from 0.2271 to 0.5791, and models G0 and G1 do not fulfill robustness requirements. On the other hand, the Q-parametrization greatly influences the infinity norm of the input sensitivity function, as the highest value of $||U||_\infty$ reaches 20.74dB only for the G0 off nominal model, thus almost attaining the specifications. The same way, the lowest modulus margin is equal to 0.4478, almost reaching specification as well. The rest of the $||U||_\infty$ and modulus margins meet the requirements. Therefore, and as expected, the Q-parametrization greatly influences the $||U||_\infty$ and modulus margin of the controller for most models, as the objective function is defined in terms of robustness and performances as well as the fact that the solution satisfies all constraints for the nominal model (thus, it is expected to improve the modulus margin and decrease the input sensitivity). The higher value of $||U||_\infty$ for the initial controller can be observed on the bottom left subplot of the input sensitivity from Figure 20, where the Q-RST one does not overcome 20dB. The robustness is also observed in the sensitivity function bottom right subplot of Figure 20, where initial controller overcomes the filter $\frac{1}{W_1}$, whereas the Q-parametrization is still underneath the filter. Figure 19 also shows that the Q-RST is robust with the other models (G0, G2 and G3).

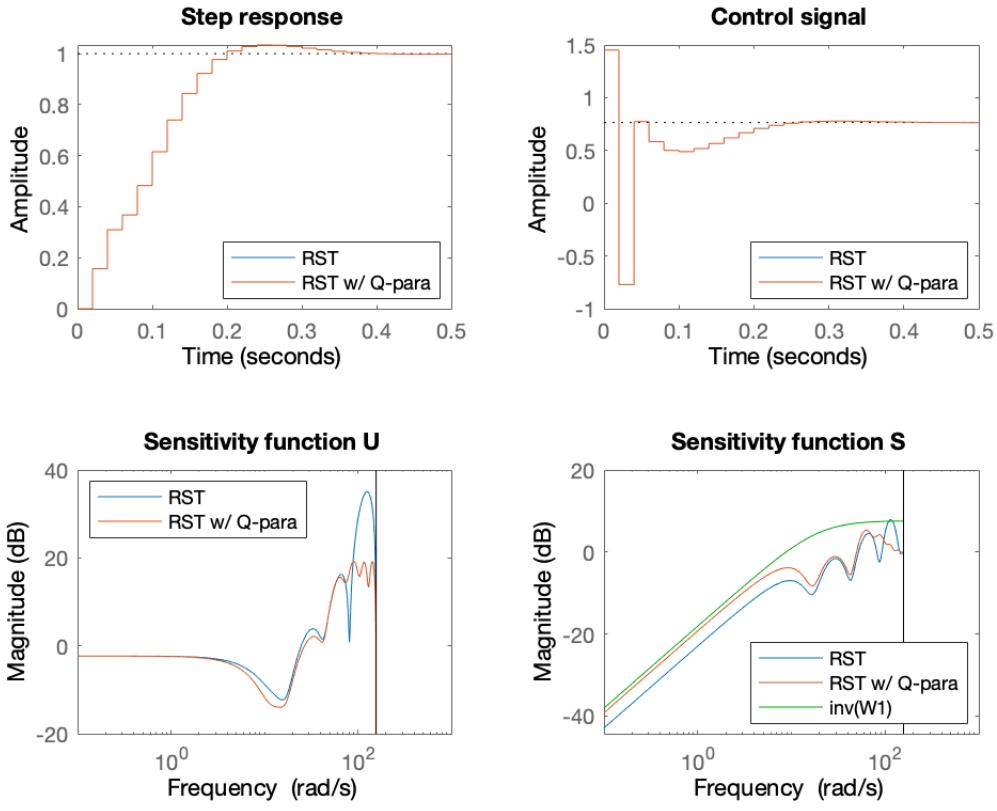

Figure 20: Comparison of the initial controller with the Q-parametrized RST controller

It is important to note that the order of Q (here equal to 6) influences the capabilities of the solver (a higher order allows more flexibility to converge to a feasible point). Finally, a trade-off between the modulus margin of one model and the $||U||_\infty$ of another has been observed: while changing iteratively the value of the bandwidth ω_0, it was possible to change the values of modulus margins and $||U||_\infty$ for all models, but a trade-off between good robustness and low $||U||_\infty$ was observed . The final solution is given for a value of $\omega_0 = 16$ rad/s as this gave much better performance than the initial 20 rad/s.

3.3.3 Controller comparison and experimental test

In the previous control exercise, two controllers were designed for the 4 plant models of the active suspension system. The first controller was designed using mixed sensitivity with a reduction of the controller order to remove zero-pole cancellations, while the second one was designed using the newer data driven approach. Before selecting a controller to be tested on the physical system, a comparison is first needed. For all controllers, the behavior of the nominal plant model G1 for which they were designed is reviewed as well as additional off nominal models to verify robustness against model uncertainty. Models G0 and G3 were chosen as they express the most radical off nominal performance.

Table 8: Performance and robustness comparison of different controllers

Controller	**Model**	**Rise time [s]**	**Overshoot [%]**	**Settling time [s]**	M_m	$\|\|U\|\|_\infty$
Mixed sensitivity	G0	0.2400	11.6324	0.6600	0.6677	18.8141
	G1	0.1200	5.5129	0.4600	0.7376	19.0374
	G3	0.2000	11.7869	0.6200	0.6009	18.7300
Data driven	G0	0.1200	5.2388	0.5800	0.6883	20.8630
	G1	0.1200	4.9842	0.6400	0.7504	21.1467
	G3	0.1400	0.4157	0.2000	0.8443	20.7567
RST	G0	0.1400	1.4018	0.2000	0.4478	20.7372
	G1	0.1400	3.4267	0.3200	0.5371	19.2354
	G3	0.1600	7.1134	0.5000	0.7061	18.9092

It can be observed that the mixed sensitivity controller has larger rise times and overshoots than the other models, but complies to the constraints on the physical bounds of the system ($||U||_\infty < 20dB$), and the requirement on the modulus margin. Using this controller could potentially be fatal to the real system due to the high overshoots making it a less appealing choice. On the other hand, RST and the data-driven controllers show better rise times (data driven fulfills the requirements, whereas RST shows a rise time of 0.16 sec and overshoot of 7.11% for model G3), but both of them have more weaknesses in the $||U||_\infty$ of U, which is slightly higher than 20dB. However, since the values of $||U||_\infty$ do not overcome values of 21.1dB for the data-driven, it seems like the overall performances of this controller are better (rise time, overshoot, settling time and modulus margin), and thus the data-driven controller is chosen for the experimentation with the real system. Once converted into an RST controller design, the data-driven controller gives the following parameters:

$$R = (2.9025,\ -7.94175,\ 8.17605,\ -3.6773,\ 0.5972)$$
$$S = (1.0000,\ -1.5315,\ 0.1889,\ 0.4836,\ -0.1410)$$
$$T = (2.9025,\ -7.9417,\ 8.1760,\ -3.6773,\ 0.5972)$$

As one can observe, the rise time of the experimental step response is much higher than that of the theoretical simulation ($T_{Rise,Exp} = 0.2908sec$ against $T_{Rise,Theory} = 0.1200sec$). It is also observed that there is a much higher overshoot in the experiment: 13.7931% against 4.9842% in theory. Since robustness of the data driven controller was one of its advantages, it seems likely that the Q-RST controller could have lead to even worse performances (such as overshoot), since the RST's modulus margin were lower than the data-driven's. Regarding the settling time however, the theory predicted 0.64sec, and the experiment lead to 0.6739sec; thus it seems like the prediction on settling time was rather accurate. It is interesting to observe that both systems lead to a zero steady state, but the control signal for the experiment is very close to the theoretical one, with a slight offset. Overall, the controller follows the same predicted

trend with slightly lower control outputs (plant inputs) and the output of the system reacts slightly more "violently" and has an odd dip between 0.05 and 0.3 seconds. This may be due to additional friction in the system or some other unmodeled external disturbances.

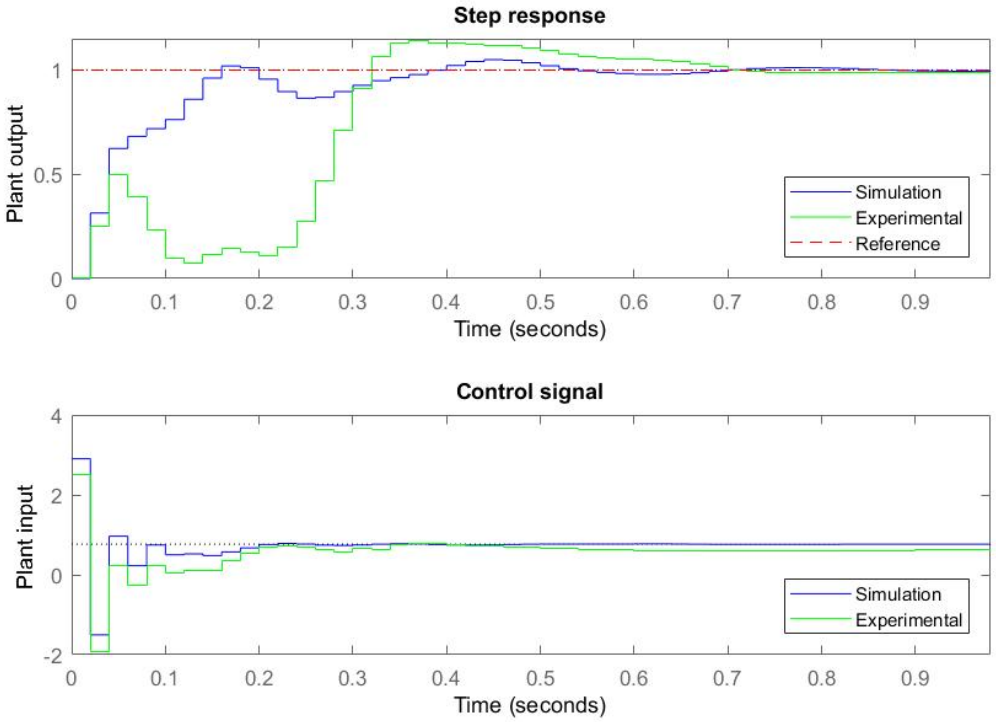

Figure 21: Comparison of simulated and experimental results for step reference input

```
clear variables;
close all;

%% 1. load data and compute model ==========================================
load ASdata.mat;
Zd0=detrend(ASdata{1});
G0=oe(Zd0,[4 4 1]);

Zd1=detrend(ASdata{2});
G1=oe(Zd1,[4 4 1]);

Zd2=detrend(ASdata{3});
G2=oe(Zd2,[4 4 1]);

Zd3=detrend(ASdata{4});
G3=oe(Zd3,[4 4 1]);

G={G0,G1,G2,G3};

%% Design closed loop poles
Ts = 0.02; % sampling time (s)
Tr = 0.1; % rise time (s)
OS = 0.05; % overshoot (5%)

P_desired = closedlooppoles(Ts,Tr,OS); % characteristic polynomial of 2nd order system
    which satisfies the requirements

%% Compute RST controller on nominal model ================================

B=G1.b;
A=G1.f;

Hr=[1 1]; % reduce noise at high frequencies
Hs=[1 -1]; % place integrator in controller

[R,S]=poleplace(B,A,Hr,Hs,P_desired);
```

```
36
37 T=sum(R);
38
39 P=conv(A,S)+conv(B,R); % verify position of closed-loop poles
40
41 figure
42 pzmap(tf(1,P,Ts,'variable','z^-1'))
43
44 %% Plot controller performance for all models ==========================
45
46 figure
47 win(1) = subplot(2, 2, 1);
48 win(2) = subplot(2, 2, 2);
49 win(3) = subplot(2, 2, 3);
50 win(4) = subplot(2, 2, 4);
51
52 set(win,'Nextplot','add')
53 U_norm_inf=zeros(1,4);
54 Mm=zeros(1,4);
55
56 for i=1:4
57 Gn=G{1,i};
58 B=Gn.b;
59 A=Gn.f;
60
61 P=conv(A,S)+conv(B,R); %compute closed-loop polynomial
62
63 CL=tf(conv(T,B),P,Ts,'variable','z^-1');
64 U=tf(conv(A,R),P,Ts,'variable','z^-1');
65 U_control=tf(conv(T,A),P,Ts,'variable','z^-1');
66 S_sensitivity=tf(conv(A,S),P,Ts,'variable','z^-1');
67 U_norm_inf(i)=mag2db(norm(U,'inf')); % [dB]
68 Mm(i)=1/norm(S_sensitivity,'inf'); %modulus margin
69
70 step(win(1),CL)
71 stepinfo(CL)
72 title('Step response')
73
74 step(win(2),U_control)
75 title('Control signal')
76
77 bodemag(win(3),U)
78 title('Sensitivity function U')
79 ylim([-20,60])
80
81 bodemag(win(4),S_sensitivity)
82 title('Sensitivity function S')
83 end
84
85 %% 3.2 Analysis of the closed-loop system ==============================
86 % 1. Design the weighting filter W1 ====================================
87
88 wb=16; % desired closed-loop bandwidth
89 m=0.5; % modulus margin
90 s=tf('s');
91 W1=c2d((s+wb)*m/(s+0.0001),Ts); % discretize W1
92
93 % Multiplicative uncertainty filter W2
94 load W2.mat;
95 W2=W2_2;
96
97 B=G1.b;
98 A=G1.f;
99
100 n=6; % Q order
101 Q0=zeros(1,n);
102
103 fun = @(x) norm(W2*tf(conv(B,[R,zeros(1,n)]+conv(A,conv(Hr,conv(Hs,x)))),P_desired,Ts,'
    variable','z^-1'),'inf')...
104             +norm(W1*tf(conv(A,[S,zeros(1,n)]-conv(B,conv(Hr,conv(Hs,x)))),P_desired,Ts,
    'variable','z^-1'),'inf');
105
106 options.MaxFunctionEvaluations = 3000;
107 options.MaxIterations = 1000;
108 [Q,fval]=fmincon(fun,Q0,[],[],[],[],[],[],@(x) mycon(x,n,A,B,S,R,Hr,Hs,P_desired,Ts),
    options);
```

```
109
110 Rq=[R,zeros(1,n)]+conv(A,conv(Hr,conv(Hs,Q)));
111 Sq=[S,zeros(1,n)]-conv(B,conv(Hr,conv(Hs,Q)));
112 Tq=sum(Rq);
113
114 %% Plot controller performance for all models ==========================
115
116 figure
117 win(1) = subplot(2, 2, 1);
118 win(2) = subplot(2, 2, 2);
119 win(3) = subplot(2, 2, 3);
120 win(4) = subplot(2, 2, 4);
121
122 set(win,'Nextplot','add')
123 U_norm_inf_Q=zeros(1,4);
124 Mm_Q=zeros(1,4);
125
126 for i=1:4
127 Gn=G{1,i};
128 B=Gn.b;
129 A=Gn.f;
130
131 P=conv(A,Sq)+conv(B,Rq); %compute closed-loop polynomial
132
133 CL=tf(conv(Tq,B),P,Ts,'variable','z^-1');
134 U=tf(conv(A,Rq),P,Ts,'variable','z^-1');
135 U_control=tf(conv(Tq,A),P,Ts,'variable','z^-1');
136 S_sensitivity=tf(conv(A,Sq),P,Ts,'variable','z^-1');
137
138 U_norm_inf_Q(i)=mag2db(norm(U,'inf')); % [dB]
139 Mm_Q(i)=1/norm(S_sensitivity,'inf'); %modulus margin
140
141 step(win(1),CL)
142 stepinfo(CL)
143 title('Step response')
144
145 step(win(2),U_control)
146 title('Control signal')
147
148 bodemag(win(3),U)
149 title('Sensitivity function U')
150 ylim([-20 25])
151 bodemag(win(4),S_sensitivity)
152 title('Sensitivity function S')
153 end
154
155 bodemag(win(4),1/W1,'r')
156
157 %% Compare RST with RST+Q-parametrization ===============================
158
159 B=G1.b;
160 A=G1.f;
161
162 Pq=conv(A,Sq)+conv(B,Rq);
163 P=conv(A,S)+conv(B,R);
164
165 CL=tf(conv(T,B),P,Ts,'variable','z^-1');
166 U=tf(conv(A,R),P,Ts,'variable','z^-1');
167 U_control=tf(conv(T,A),P,Ts,'variable','z^-1');
168 S_sensitivity=tf(conv(A,S),P,Ts,'variable','z^-1');
169
170 CLq=tf(conv(Tq,B),Pq,Ts,'variable','z^-1');
171 Uq=tf(conv(A,Rq),Pq,Ts,'variable','z^-1');
172 U_controlq=tf(conv(Tq,A),Pq,Ts,'variable','z^-1');
173 S_sensitivityq=tf(conv(A,Sq),Pq,Ts,'variable','z^-1');
174
175 figure
176 win(1) = subplot(2, 2, 1);
177 win(2) = subplot(2, 2, 2);
178 win(3) = subplot(2, 2, 3);
179 win(4) = subplot(2, 2, 4);
180
181 set(win,'Nextplot','add')
182
183 step(win(1),CL)
184 step(win(1),CLq)
```

```
legend(win(1),'RST','RST w/ Q-para','Location','SouthEast')
title('Step response')

step(win(2),U_control)
step(win(2),U_controlq)
legend(win(2),'RST','RST w/ Q-para','Location','SouthEast')
title('Control signal')

bodemag(win(3),U)
bodemag(win(3),Uq)
legend(win(3),'RST','RST w/ Q-para','Location','NorthWest')
title('Sensitivity function U')
ylim([-20 40])

bodemag(win(4),S_sensitivity)
bodemag(win(4),S_sensitivityq)
bodemag(win(4),1/W1,'g')
legend(win(4),'RST','RST w/ Q-para','inv(W1)','Location','SouthEast')
title('Sensitivity function S')
```

Listing 21: CE_3 complet Matlab script

```
clear variables;
close all;

%% 1. load data and compute model ==========================================
load ASdata.mat;

Zd1=detrend(ASdata{2});
G1=oe(Zd1,[4 4 1]);

load RST.mat;

M=load('BV.txt','-ascii');
u=M(:,3);
y=M(:,5);
r=M(:,7);

Ts = 0.02; % sampling time (s)

t=0:0.02:12.4;
t=t(118:end)-t(118);
%% Plot controller performance for all models ==============================

figure

B=G1.b;
A=G1.f;

P=conv(A,S)+conv(B,R); %compute closed-loop polynomial

CL=tf(conv(T,B),P,Ts,'variable','z^-1');
U_control=tf(conv(T,A),P,Ts,'variable','z^-1');

subplot(2, 1, 1);
hold on
%Simulated system
step(CL,t,'b')
stepinfo(CL)

%True system
stairs(t,y(118:end)./10,'g')
stepinfo(y(118:end)./10,t)
stairs(t,r(118:end)./10,'r--')

ylim([0 1.15]);
xlim([0 t(50)]);
title('Step response')
ylabel('Plant output ')
xlabel('Time')
legend('Simulation','Experimental','Reference','location','southEast')

subplot(2, 1, 2);
hold on
%Simulated system
step(U_control,t,'b')
```

```
55
56 %True system
57 stairs(t,u(118:end)./10,'g') % offset ?
58 xlim([0 t(50)]);
59 title('Control signal')
60 ylabel('Plant input')
61 xlabel('Time')
62 legend('Simulation','Experimental','location','SouthEast')
```

Listing 22: Matlab script to compare simulation and experimental performance