CE1: Norms of Systems and Model Uncertainty

Advanced control systems ME-524

Group A: Taras Pavliv 276861, Alexandre Guerra De Oliveira 300636 and Valentin Roch 284935

March 26, 2022

1 CE1: Norms of Systems and Model Uncertainty

The complete matlab code used to solve the exercises is given in the annex.

1.1 Norms of SISO systems

1.1.1 2-Norm of SISO systems

1. Using the residue method :

$$G(s) = \frac{10 - 2s}{s^2 + 0.1s + 16}$$

$$G(-s) = \frac{10 + 2s}{s^2 - 0.1s + 16}$$

$$f(s) = G(s) \cdot G(-s) = \frac{10 - 2s}{s^2 + 0.1s + 16} \frac{10 + 2s}{s^2 - 0.1s + 16}$$

$$||G(s)||_2^2 = \frac{1}{2\pi j} \oint G(-s)G(s)ds = \sum_{p_i \in RHP} Res(G(s)G(-s), p_i)$$

To calculate the poles of G(s), one have to find the zeros of its denominator:

$$s^2 + 0.1s + 16 = 0$$

$$p_{1,2} = \frac{-0.1 \pm \sqrt{0.1^2 - 4 \cdot 16}}{2} = \frac{-0.1 \pm j\sqrt{63.99}}{2}$$

Same for the poles of G(-s):

$$s^2 - 0.1s + 16 = 0$$

$$p_{3,4} = \frac{0.1 \pm \sqrt{0.1^2 - 4 \cdot 16}}{2} = \frac{0.1 \pm j\sqrt{63.99}}{2}$$

As we only need the poles with a negative real part to calculate the two-norm of G(s), we only have to calculate the residues of p_1 and p_2.

$$Res(f(s), p_1) = \lim_{s \to p_1} (s - p_1) \cdot f(s) = \lim_{s \to p_1} \frac{(10 - 2s)(10 + 2s)}{(s - p_2)(s - p_3)(s - p_4)} = \frac{(10 - 2p_1)(10 + 2p_1)}{(p_1 - p_2)(p_1 - p_3)(p_1 - p_4)}$$

$$\cong 25.6250 - 0.0703j$$

$$Res(f(s), p_2) = \lim_{s\to p_2}(s-p_2)\cdot f(s) = \lim_{s\to p_2}\frac{(10-2s)(10+2s)}{(s-p_1)(s-p_3)(s-p_4)} = \frac{(10-2p_2)(10+2p_2)}{(p_2-p_1)(p_2-p_3)(p_2-p_4)}$$
$$\cong 25.6250 + 0.0703j$$

$$||G(s)||_2 = \sqrt{\sum_{i=1}^{2} Res(G(s)G(-s), s_i)} = \sqrt{25.625 + 0.0703j + 25.625 - 0.0703j} \cong 7.1589$$

2. Using the frequency response : $||G(s)||_2 = \sqrt{\frac{1}{2\pi}\int_{-\infty}^{\infty}|G(j\omega)|^2 d\omega}$ and by approximating the integral on matlab :

```
fun = @(om) (1/(2*pi))*(abs((10-2*(1i*om))./((((1i*om)).^2)+0.1*(1i*om)+16)).^2);
norm_2_2 = sqrt(integral(fun,-Inf,Inf))
```

Which gives : $norm\_2\_2 = 7.1589$. We aproximated the integral using matlab's integrate function that uses a global adaptive quadrature method to chose the discretisation step, thus the answer is quite precise.

3. Using the frequency response, for a stable system the Parseval rule applies and we can calculate the two-norm with the impulse response:

$||G(s)||_2 = \sqrt{\int_{-\infty}^{\infty}|g(t)|^2 dt}$

Using matlab, we can compute the norm of the impulse of G with:

```
G = tf([-2 10],[1 0.1 16]);
[impulse_g,impulse_tt] = impulse(G);
norm_2_3 = sqrt(trapz(impulse_tt,impulse_g.*impulse_g))
```

We get : $norm\_2\_3 = 7.1603$. Because the integral is calculated with the trapezoidal rule, it is not as precise as other methods, and thus the result is a bit worse than the other methods tested. Additionaly, matlab automatically calculates the length of the time and impulse response vector, which might cause additional error.

4. The state space method states: $||G||_2 = \sqrt{trace(CLC^T)}$

Where L solves the following Riccati equation: $AL + LA^T + BB^T = 0$

We can find the state space model by setting: $G(s) = \frac{10-2s}{s^2+0.1s+16} = G_1G_2$.
Where G_1 contains all the poles and G_2 contains all the zeros.

$$G_1 = \frac{X(s)}{U(s)} = \frac{1}{s^2+0.1s+16} \Leftrightarrow X(s)\cdot(s^2+0.1s+16) = U(s)$$
$$\Rightarrow \ddot{x} + 0.1\dot{x} + 16x = u \Rightarrow \ddot{x} = -0.1\dot{x} - 16x + u$$

and

$$G_2 = \frac{Y(s)}{X(s)} = 10 - 2s \Leftrightarrow Y(s) = (10-2s)\cdot X(s)$$
$$\Rightarrow y = 10x - 2\dot{x} = \begin{pmatrix}10 & -2\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}$$

$$\begin{cases} x_1 = x \\ \dot{x_1} = x_2 \\ \dot{x_2} = -0.1x_2 - 16x_1 + u \end{cases}$$

Setting $X = \begin{pmatrix}x_1\\x_2\end{pmatrix}$ leads to : $\dot{X} = AX + Bu$ and $y = CX + Du$

with $A = \begin{pmatrix}0 & 1\\-16 & -0.1\end{pmatrix}$ and $B = \begin{pmatrix}0\\1\end{pmatrix}$

$C = \begin{pmatrix} 10 & -2 \end{pmatrix}$
$D = 0$
We solve the algebraic Riccati equation using the matlab command: $L = are(A^T, 0, BB^T)$
Using the following matlab code :
`L = are(A', zeros(2,2), B*(B'));`
`norm_2_4 = sqrt(trace(C*L*(C')))` We get : $norm\_2\_4 = 7.1589$

5. Using matlab `norm_2_5 = norm(G,2)` to validate our result, we get $norm\_2\_5 = 7.1589$.

1.1.2 ∞-Norm of SISO systems

1. Because G(s) has no pole on the imaginary axis, we can measure the norm using $||G||_\infty = \sup_\omega |G(j\omega)|$. We plot the Bode diagram of $|G(j\omega)|$, we get the figure 1.

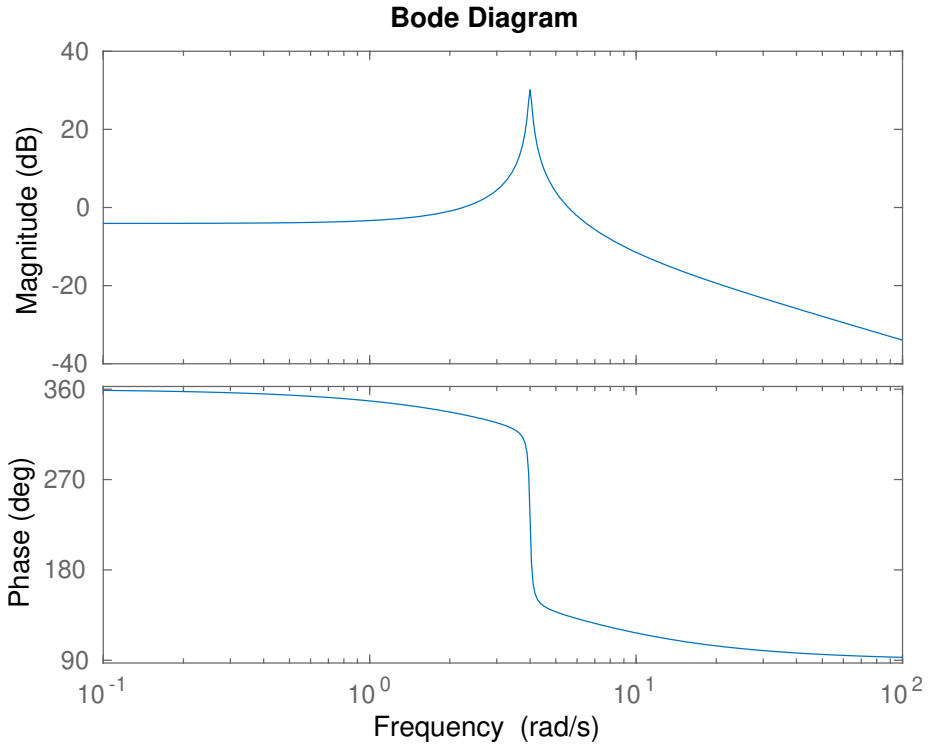

Figure 1: Bode diagram of $|G(j\omega)|$

With this we know that the supremum is before 10^2. So we make a grid with `linspace(0,100,10000)`, calculate the norm on each point of the grid, and keep the maximum value. With this method, we find $norm\_inf\_1 = 32.0126$. However it should be noted that this method gives an approximation, as it is unlikely that a point of the grid exists at the peak value of $|G(j\omega)|$. It should still be quite close if the time vector is discretised fine enough.

2. To use the *bounded real lemma* we proceed as follows (iterative method):

   - We first make a hypothesis on the range of the infinity norm of $G(s)$. With the previous method we know it is around 32.0126, so we choose $\gamma_{min} = 0$ and $\gamma_{max} = 100$.
   - We choose $\gamma = \frac{\gamma_{min} + \gamma_{max}}{2}$ as a test value to divide the interval by two at each itteration.
   - We define the matrix
   $$H = \begin{pmatrix} A & \gamma^{-2} \cdot BB^T \\ -C^T C & -A^T \end{pmatrix}$$
   . The A, B and C matrices are those found in 1.1.1.4..

- If this matrix has an eigenvalue on the imaginary axis, it means $\gamma < ||G||_\infty$, so we can set $\gamma_{min} = \gamma$ and repeat from step 2. If it has no value on the imaginary axis, we set $\gamma_{max} = \gamma$ and repeat from step 2.
- When $\frac{\gamma_{max} - \gamma_{min}}{\gamma_{min}}$ is inferior to a specified tolerance, the gamma value is our result for $||G||_\infty$.

With this method we get $norm\_inf\_2 = 32.0166$. As we use the bissection method, we expect this to converge quickly to the result, so we can decrease the tolerance a lot to get a precise result.

The code to calculate the bisection algorithm is:

```
while abs((gam1-gam2)/gam1)>tol_gam
    % solve for the eigen values

    gam = (gam2+gam1)/2;
    eig_gam = eig(H(gam));

    up = 0; % up = 0 <=> H has no eigenvalue in the imag axis for the current gamma

    for k = 1:length(eig_gam)
       if (abs(real(eig_gam(k)))<=tol_eigs)
          up = 1; % H has a eigenvalue on the imaginary axis (or very close) for this gamma
       end
    end

    if (up==0)
        gam2 = gam; % if H has no an eigenvalue on the imaginary axis set gam2 = gam
    else
        gam1 = gam; % if H has an eigenvalue on the imaginary axis set gam1 = gam
    end
end
```

3. Using matlab `norm_inf_3 = norm(G,Inf)` to validate our result, we get $norm\_inf\_3 = 32.0166$.

1.2 Norms of MIMO systems

1.2.1 2-Norm of MIMO systems

For this part we work with a multi-input multi-output control. For this we will need a matlab state space, that we define with `sys_cts = ss(A,B,C,D)`, where A,B,C and D are given, and have the values:

$$A = \begin{pmatrix} -13.098 & -5.8441 & 5.7893 \\ -5.8441 & -16.2606 & -5.3504 \\ 5.7893 & -5.3504 & -7.6497 \end{pmatrix} \quad B = \begin{pmatrix} -1.2075 & 0.4889 \\ 0 & 1.0347 \\ 1.6302 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & -0.7873 & -1.1471 \\ 0.2939 & 0.8884 & -1.0689 \end{pmatrix} \quad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

1. We calculate the norm using $||G||_2^2 = \frac{1}{2\pi} \int_{-\infty}^{\infty} trace[G^*(j\omega)G(j\omega)]\, d\omega$, but to avoid to integrate between $-\infty$ and ∞ we will try to find a range $[a, b]$ such that $\int_a^b trace[G^*(j\omega)G(j\omega)]\, d\omega \approx \int_{-\infty}^{+\infty} trace[G^*(j\omega)G(j\omega)]\, d\omega$. To find an appropriate range, we first plot the system, on figure 2 with `sigma(sys_cts)`.

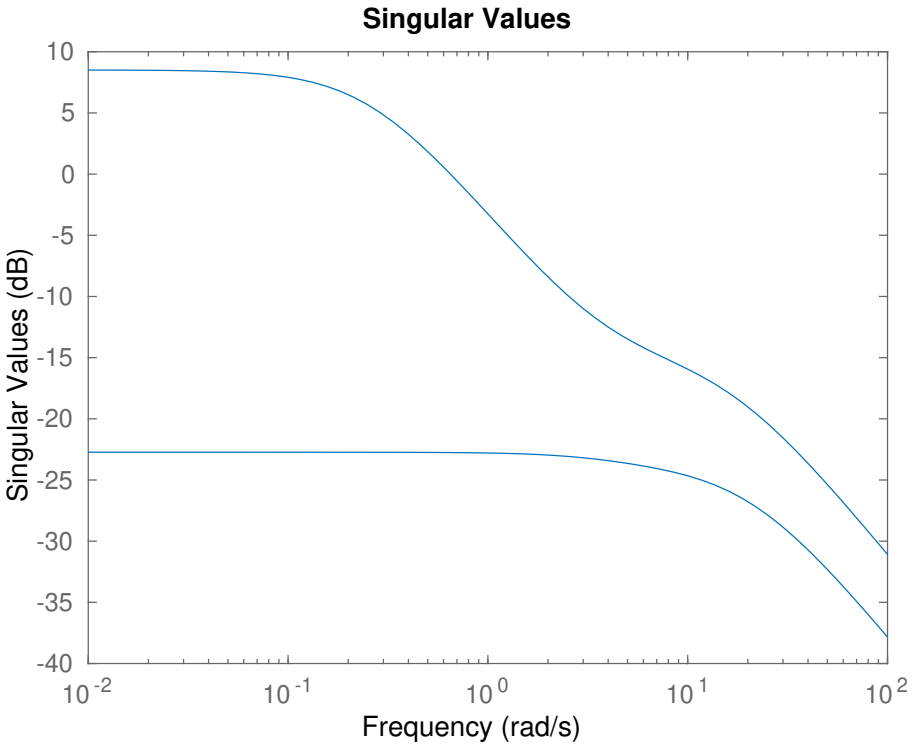

Figure 2: Sigma diagram of `sys_cts`

With the figure 2 we see that the values will already be below -30 dB with $\omega > 10^2$ so anything above that should be ok. We decide to integrate between $a = 10^{-4}$ and $b = 10^4$ using the fact that the function is even. We set `omega_grid = logspace(-4,4,50000)`. At each frequency ω we calculate $trace[G^*(j\omega)G(j\omega)]$, and store it in the vector `integrate_function_trace(k)`. We then get an approximation of the integral (using the trapezoidal rule) with the matlab `trapz(...)` function. We get the result with

```
estimated_integral = 2*trapz(omega_grid,integrate_function_trace)
norm_2_m1 = sqrt((1/(2*pi))*(estimated_integral))
```

and find $norm\_2\_m1 = 1.0912$. It should be noted that it is an approximation of the real norm, because of the inexactitudes in the integral (limited range and trapezoidal integration scheme).

2. Here we calculate the 2-norm using $||G||_2 = \sqrt{trace(CLC^T)}$, where L is the solution of the equation

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

L can be found using a matlab function `L=are(A',zeros(size(A)),B*(B'))` (the solver for an Algebraic Riccati Equation), and we find the norm with:
`norm_2_m2 = sqrt(trace(C*L*(C')))`
The result is $norm\_2\_m2 = 1.0915$, and should be quite close to the real value.

3. Using matlab `norm_2_m3 = norm(sys_cts,2)` to validate our result, we get $norm\_2\_m3 = 1.0915$.

1.2.2 ∞-Norm of MIMO systems

1. To calculate the infinity-norm of the MIMO system using the frequency response and using the same model as before, we start by computing the sigma function for the whole frequency domain:

`[sig_frequency_response, ∼] = sigma(sys_cts);` we then find the maximum sigma value for each frequency: `max_sig_frequency_response = max(sig_frequency_response,[],1);`. Finally, the infinity-norm is given by the maximum value over the array : `norm_inf_q1 = max(max_sig_frequency_response)` which gives the value 2.6627.

2. To calculate the infinity-norm of the MIMO system using the bounded real lemma, the exact same method as in section 1.1.2.2 is used.

   The bounded real lemma (iterative bisection algorithm) gives a result for the norm of 2.6647.

3. The matlab command gives `norm_inf_3 = norm(sys_cts, inf) = 2.6647` which validates the results in the first and second part of the 1.2.2 section.

1.3 Uncertainty modeling

First we define the parametric uncertain model in matlab with:

```
k = ureal('k',10,'PlusMinus',[-2 2]);
tau1 = ureal('tau1',2,'PlusMinus',[-0.5 0.5]);
tau2 = ureal('tau2',4,'PlusMinus',[-1 1]);
G1 = tf([k],[tau1 1]);
G2 = tf([1],[tau2 1]);
G = G1*G2;
```

Then we create N (in our case N1 = 20 an N2 = 200) random samples of the uncertain parametric model to convert it into multi-model uncertainty using usample().
Using ucover() we convert it from multi-model uncertainty into multiplicative uncertainty:

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

Here is the implementation:

```
g_sample_n1 = usample(G,N1);
[usys_n1,info_n1] = ucover(g_sample_n1,G.NominalValue,order_n1);
```

We have chosen the nominal model to be the same as the parametric nominal model and the chosen order of W_2 is 2.

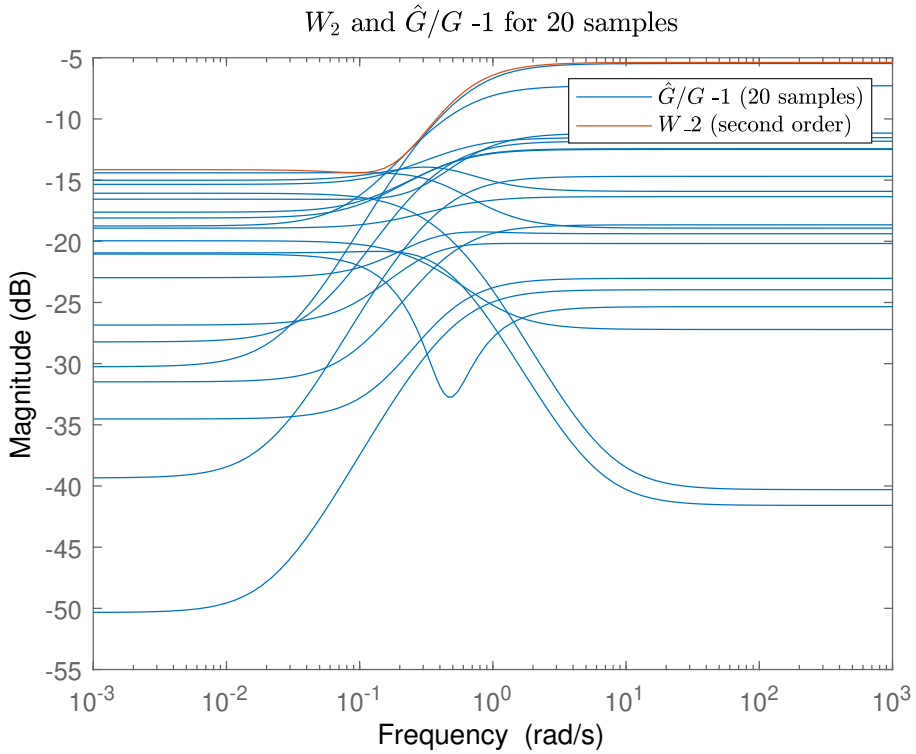

Figure 3: Uncertainty filter bode magnitude for 20 samples

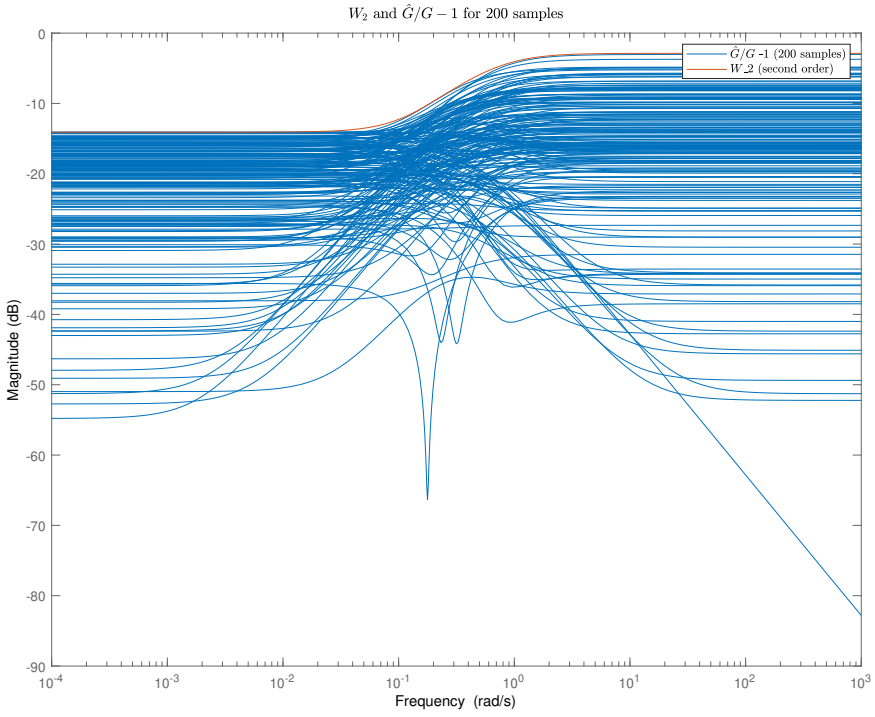

Figure 4: Uncertainty filter bode magnitude for 200 samples

We can see that in both cases, W_2 magnitude is bigger than the realisation of uncertainty of each of the models of the multi-model uncertainty case $\frac{\tilde{G}}{G} - 1$.
We can also see that an order two uncertainty filter can already provide a not very conservative estimate of the multiplicative uncertainty here. We normally try to have the smallest order W_2 possible while still keeping an order that does not give a very conservative W_2 estimate.
We can see that using N = 20 samples or N = 200 samples, the W_2 did not change a lot. But in the plot with 20 samples we can see that the weighing filter is quite close to one of the realisations while for the 200 samples we are a little further. This is due to the fact that the order of the filter did not change but we have way more samples in the 200 samples realisation and the weighing filter must be greater in magnitude for all of the realisations, so for N = 200 samples, the weighting filter is slightly more conservative then the one with 20 samples.

2 Annexe

The complete matlab code is given here:

2.1 Exercise 1

```
clc,close all,clear all
%% exo 1.1
%1.1.1
a = 1;
b = 0.1;
c = 16;

delta = (b^2)-4*a*c;

s1 = (-b+sqrt(delta))/(2*a);
s2 = (-b-sqrt(delta))/(2*a);
s3 = (b+sqrt(delta))/(2*a);
s4 = (b-sqrt(delta))/(2*a);

Res_s1 = (10-2*s1)*(10+2*s1)/((s1-s2)*(s1-s3)*(s1-s4));
Res_s2 = (10-2*s2)*(10+2*s2)/((s2-s1)*(s2-s3)*(s2-s4));
Res_s3 = (10-2*s3)*(10+2*s3)/((s3-s1)*(s3-s2)*(s3-s4));
Res_s4 = (10-2*s4)*(10+2*s4)/((s4-s1)*(s4-s2)*(s4-s3));

norm_2_1 = sqrt(Res_s1+Res_s2)

%1.1.2
fun = @(om) (1/(2*pi))*(abs((10-2*(1i*om))./( (((1i*om)).^2)+0.1*(1i*om)+16  )).^2);

norm_2_2 = sqrt(integral(fun,-Inf,Inf))

%1.1.3

G = tf([-2 10],[1 0.1 16]);
[impulse_g,impulse_tt] = impulse(G);

norm_2_3 = sqrt(trapz(impulse_tt,impulse_g.*impulse_g))

%1.1.4
A = [0,1;-16,-0.1];
B = [0;1];
C = [10,-2];
```

```
L = are(A',zeros(2,2),B*(B'));

norm_2_4 = sqrt(trace(C*L*(C')))

%1.1.5

norm_2_5 = norm(G,2)

%% exo 1.2
%1.2.1
fun2 = @(om) abs((10-2*(1i*om))./( (((1i*om)).^2)+0.1*(1i*om)+16  ));

figure
bode(G)

omega_grid = linspace(0,100,10000);
freq_response = fun2(omega_grid);

norm_inf_1 = max(freq_response)

%1.2.2
H = @(gam) [A,(gam^(-2))*B*(B');-C'*C,-A'];

tol_gam = 10^(-10);
tol_eigs = 10^(-10);

gam1 = 0.1;
gam2 = 100;

while abs((gam1-gam2)/gam1)>tol_gam
    % solve for the eigen values

    gam = (gam2+gam1)/2;
    eig_gam = eig(H(gam));

    up = 0; % up = 0 <=> H has no eigenvalue in the imag axis for the current gamma

    for k = 1:length(eig_gam)
       if (abs(real(eig_gam(k)))<=tol_eigs)
          up = 1; % H has a eigenvalue on the imaginary axis (or very close) for this gamma
       end
    end

    if (up==0)
        gam2 = gam; % if H has no an eigenvalue on the imaginary axis set gam2 = gam
    else
        gam1 = gam; % if H has an eigenvalue on the imaginary axis set gam1 = gam
    end
end

norm_inf_2 = gam
%1.2.3
norm_inf_3 = norm(G,Inf)
```

2.2 Exercise 1.2

```
%% ex 2
clc,close all,clear all
data = importdata('G_mimo.mat');


%% continous time state space model
A = data.A;
B = data.B;
C = data.C;
D = data.D;

sys_cts = ss(A,B,C,D);


%% exercice 1.2.1

% 1. 2-norm with frequency responce
sigma(sys_cts) % plot the sigma to know where to stop and end the grid in omega

N = 50000;
omega_grid = logspace(-4,4,N);
G_grided = freqresp(sys_cts,omega_grid);



for k = 1:N
    integrate_function(:,:,k)= (G_grided(:,:,k)')*(G_grided(:,:,k));
    integrate_function_trace(k) = trace(integrate_function(:,:,k));
end
 % function is symetric so integrate from 0 to inf then multiply by 2
estimated_integral = 2*trapz(omega_grid,integrate_function_trace);

norm_2_m1 = sqrt((1/(2*pi))*(estimated_integral))

% 2. 2-norm with state space method

L = are(A',zeros(size(A)),B*(B'));

norm_2_m2 = sqrt(trace(C*L*(C')))


% 3. 2-norm with matlab


norm_2_m3 = norm(sys_cts,2)

%% exercice 1.2.2

% 1. inf-norm with frequency response

[sig_frequency_response,~] = sigma(sys_cts);

max_sig_frequency_response = max(sig_frequency_response,[],1);

norm_inf_m1 = max(max_sig_frequency_response)
```

```
% 2. inf-norm with the bonded real lemma
H = @(gam) [A,(gam^(-2))*B*(B');-C'*C,-A'];

tol_gam = 10^(-10);
tol_eigs = 10^(-10);

gam1 = 0.1;
gam2 = 100;

while abs((gam1-gam2)/gam1)>tol_gam
    % solve for the eigen values

    gam = (gam2+gam1)/2;
    eig_gam = eig(H(gam));

    up = 0; % up = 0 <=> H has no eigenvalue in the imag axis for the current gamma

    for k = 1:length(eig_gam)
       if (abs(real(eig_gam(k)))<=tol_eigs)
          up = 1; % H has a eigenvalue on the imaginary axis (or very close) for this gamma
       end
    end

    if (up==0)
        gam2 = gam; % if H has no an eigenvalue on the imaginary axis set gam2 = gam
    else
        gam1 = gam; % if H has an eigenvalue on the imaginary axis set gam1 = gam
    end
end

norm_inf_m2 = gam

% 3. inf-norm with matlab command

norm_inf_m3 = norm(sys_cts,inf)
```

2.3 Exercise 1.3

```
%% ex 3
clc,close all,clear all

k = ureal('k',10,'PlusMinus',[-2 2]);
tau1 = ureal('tau1',2,'PlusMinus',[-0.5 0.5]);
tau2 = ureal('tau2',4,'PlusMinus',[-1 1]);


G1 = tf([k],[tau1 1]);
G2 = tf([1],[tau2 1]);
G = G1*G2;

%nyquist(G);


N1 = 20;
N2 = 200;
```

```
% convert to multi-model uncertanty with 20 random samples
g_sample_n1 = usample(G,N1);

order_n1 = 2;
% convert to multiplicative uncertanty
[usys_n1,info_n1] = ucover(g_sample_n1,G.NominalValue,order_n1);

figure
hold on
bodemag(g_sample_n1/G.NominalValue - 1)
bodemag(info_n1.W1)
title("W2 and Ghat/G -1 for 20 samples")
legend("Ghat/G -1 (20 samples)","W2 (second order)")

% convert to multiplicative uncertanty with 200 samples
g_sample_n2 = usample(G,N2);

order_n2 = 2;

[usys_n2,info_n2] = ucover(g_sample_n2,G.NominalValue,order_n2);

figure
hold on
bodemag(g_sample_n2/G.NominalValue - 1)
bodemag(info_n2.W1)
title("W2 and Ghat/G -1 for 200 samples")
legend("Ghat/G -1 (200 samples)","W2 (second order)")
```