École Polytechnique Fédérale de Lausanne

CE-1 : Nonparametric Methods

Practical Exercises for System Identification

ME-421 : System identification

Alexandre Guerra de Oliveira
Estelle Richard
Alicia Mauroux

April 20, 2022

Contents

1 CE-1 : Non-parametric Methods

In this first computer exercise we will implement different algorithms learned during the lectures about system identifications and more specifically about non-parametric methods.

The given plant transfer function for different exercises is the following:

$$G(s) = \frac{2 - s}{s^2 + 1.85s + 4} \tag{1}$$

The simulink model adds saturation and noise to this linear model.

1.1 Step response

First, we composed the simulink scheme shown in the Figure 1. It is composed of the transfer function G(s), of a noise with a variance of 0.01 and a sampling time $T_e = 0.1$ [s], and also of a saturation of the input with an upper limit of 0.5 and a lower limit of - 0.5. This scheme will be used for all the exercises of this document.

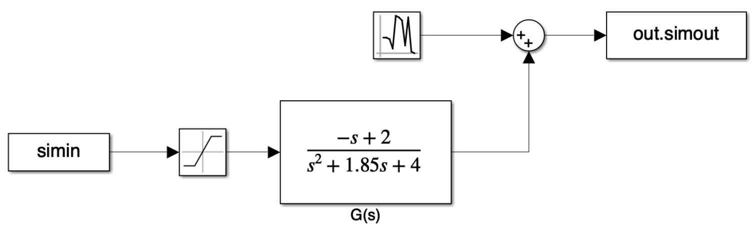

Figure 1
Simulink model

The sample time, $T_e = 0.1[s]$ was imposed on us. We could confirm it was a reasonable choice by using the Nyquist theorem. The Shannon theorem says that the sampling frequency should be at least twice as fast as the nyquist frequency of the system. We use the heuristic that the nyquist frequency is 10 times the bandwidth of the system, and we find a sample time of 0.0848[s] using the Matlab code below in the Listing 1. It is very close to the 0.1[s] given and as taking 10 times the bandwidth is a heuristic, we just want a response of the same order of magnitude. It means $T_e = 0.1[s]$ is a good choice.

```
%% Is Te enough ?
G = tf([-1 2],[1 1.85 4]);

fb = bandwidth(G) ; % rad/s
fb = fb/(2*pi); % Hz

f_N = 10*fb; % Nyquist frequency is 10 times the bandwidth
fe = 2*f_N; % Nyquist theorem

Te_th = 1/fe; % sampling time theoric
disp(Te_th);

% we can see that the natural frequency of the system is way smaller than
% than half of the sampling frequency, so Nyquists/Shannon theorem is
% verified. Thus the sampling time is reasonable.
```

Listing 1
Matlab code about sample time Te.

The plots of Figure 2 and Figure 3 display the step and Kronecker delta impulse response of the saturated and noisy system that we computed on Simulink as well as system without saturation or noise.

It is very hard to distinguish the Kronecker impulse response from the noise. Because of the saturation of the system, we cannot just apply a dirac function, we have to use a Kronecker delta function scaled with the saturation limit of the system.

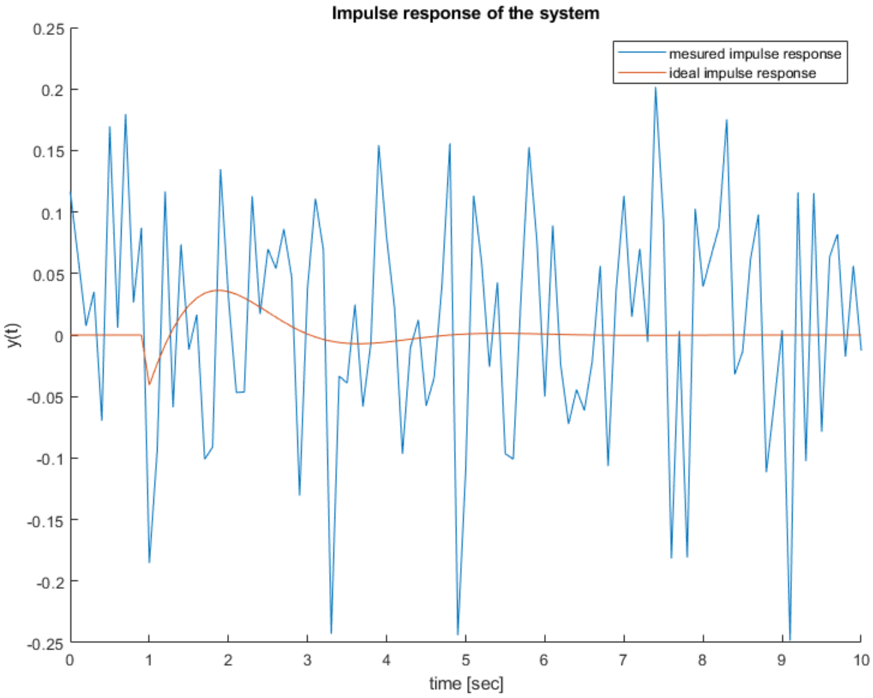

Figure 2
Impulse Response

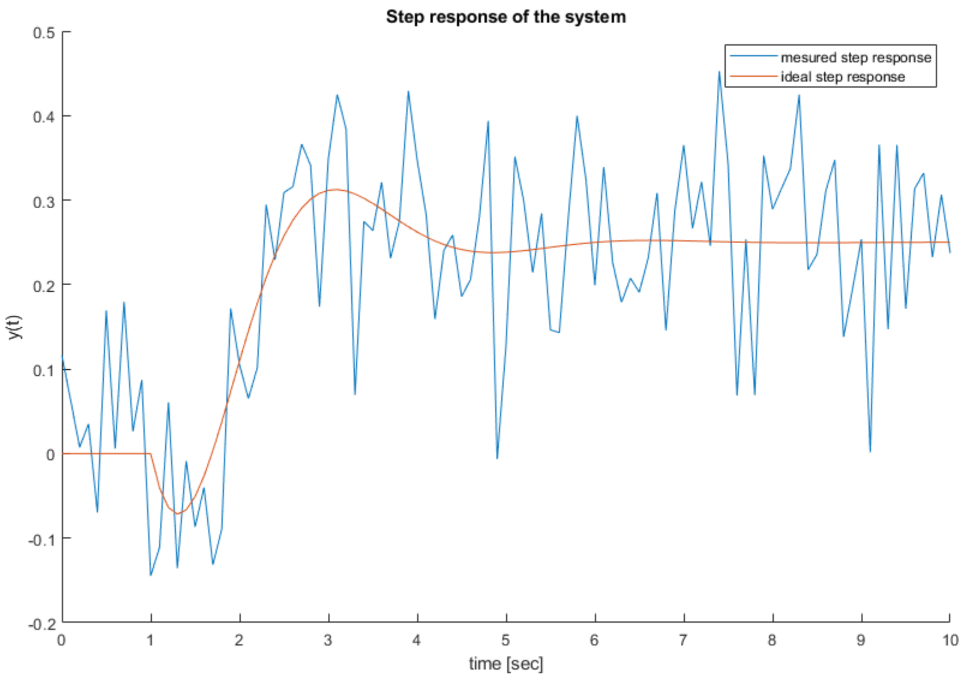

Figure 3
Step Response

In order to obtain these 2 graphs, the Matlab codes in Listing 2 and Listing 3 below have been computed. In the Listing 2, the initialisation of some parameters has been done. In Listing 3, the input vectors for the step and the impulse response are built as well as the graphs.

```
    sat_up = 0.5; % Upper saturation value
    Te = 0.1; % Sampling frequency
```

```

    % Time vector
    tt = (0:Te:10)'; % is a column vector as the exercise demands
```

Listing 2

Matlab code of parameters initialisation

```

    % create values vector
values = zeros(size(tt));
t_start = 1; % sec (when the step should start)
n_start = t_start/Te; % calculate in which sample of the discrete signal to start
    the step
values(n_start+1:end) = ones(1,size(tt,1)-n_start)*sat_up;

% pass to simulink stuct
simin.time = tt;
simin.signals.values = values;

% call the simulation
out_step = sim('exo1.slx',tt(end));


% ideal response

G = tf([-1 2],[1 1.85 4]);

y_step = lsim(G,values,tt);


% plot data
hold on

plot(out_step.simout.Time,out_step.simout.Data);
plot(tt,y_step);

title("Step response of the system");
legend("mesured step response","ideal step response")
xlabel("time [sec]")
ylabel("y(t)")



%% do the same for the impulse response

%create input vector
values_dirac = zeros(size(tt));
values_dirac(n_start) = sat_up;

simin.signals.values = values_dirac;

y_imp = lsim(G,values_dirac,tt);
% [y_imp2,tt2] = impulse(G*Te*sat_up);

% simulate in simulink
out_impulse = sim('exo1.slx',tt(end));

% plot data
figure
hold on
plot(out_impulse.simout.Time,out_impulse.simout.Data);
plot(tt,y_imp);
% plot(tt2,y_imp2);




title("Impulse response of the system");
legend("mesured impulse response","ideal impulse response")
xlabel("time [sec]")
ylabel("y(t)")
```

Listing 3

Matlab code to generate step and impulse inputs and responses.

1.2 Auto Correlation of a PRBS signal

In this section, we computed the auto correlation function for a periodic signal in order to use it in the section 1.4 to find the impulse response by correlation approach. You can find below in the Listing 4 the Matlab code of our function to compute the correlation between two periodic signals u and y. Note that it only works if the two signals have the same length.

```
function [R,h] = intcor(u,y)
    % Find M
    M =size(u,1);
    if(size(u,1)~=size(y,1))
        error("u and y do not have the same size");
    end

    % Calculate h vector
    h = 0:M-1;

    % Calculate R(h)
    R = zeros(size(h));
    for j = h
       y_shifted = [y(M-j+1:M);y(1:M-j)]; % We shift y by h: y(k-h)
       components = u.*y_shifted; % Components to be summed over to get R(h): u(k)*y(k-h)

       R(j+1) = sum(components)/M;
    end
end
```

Listing 4

Matlab code about the intercor function

To check this function, we use it to calculate the auto correlation for the signal given by the function PRBS(n=6,p=4). The code that has been used is the one in the Listing 5. When using the function, we get the following graph in Figure 4:

```
u=prbs(6,4);
y=u;

[R,h] = intcor(u,y);

plot(h,R,'*')
title("Autocorrelation function for PRBS(n=6,p=4)")
xlabel("h")
ylabel("R_{uu}(h)")
```

Listing 5

Matlab code to test intcor function

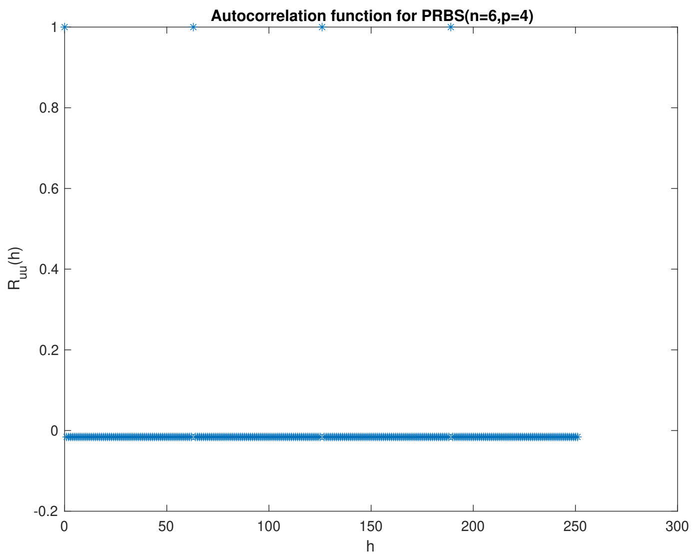

Figure 4

Auto-correlation of a PRBS signal with n = 6 (shift register length) and p = 4 (number of periods in signal)

We can see that we get the same result as we were to get using the formula 2 given in the system identification course. This formula corresponds to the auto-correlation function of a PRBS signal.

$$R_{uu}(h) = \begin{cases} a^2 & \text{h} = 0, \pm M, \pm 2M... \\ \frac{-a^2}{M} & \text{elsewhere} \end{cases} \tag{2}$$

Here $M = 2^n - 1$

Indeed, we can see in Figure 4 that we have the value $a^2 = 1$ at each one of the 4 periods and a very small negative value for the rest of the data which is coherent with the formula.

1.3 Impulse response by deconvolution method

In this section we want to compute the Impulse response of the system in simulink using the numerical deconvolution method.

The matrix U is a asymmetric toeplitz matrix created using the vector of inputs. This matrix can be singular, so it is better to suppose that the length of the impulse response is equal to K, with K ¡ N, so we can solve the problem in the least squares sense.
We have found that K = 70 worked well for this system.
For this purpose, we used the formula 3 where Θ_K and U_K are the truncated version of Θ and U.

$$\Theta_K = (U_K^T U_K)^{-1} U_K^T Y \tag{3}$$

```

Te = 0.1; %[s]
N = 1001; %[-] number of points

tt = (0:Te:(N-1)*Te)';

sat_up = 0.5;
u = rand(size(tt))*sat_up;
%no noise system as comparison

G = tf([-1 2],[1 1.85 4]);
y_nonoise = lsim(G,u,tt);

%simulate system with simulink
simin.time = tt;
simin.signals.values = u;
out_sim = sim('exo3.slx',tt(end));
y = out_sim.simout.Data;

%% deconvolution method (finite impulse response)
K = 70;

% create asymetric toeplitz matrix
r = zeros(1,K);
r(1) = u(1);
T = toeplitz(u,r);

% solve least squares
g_fir = inv((T')*(T))*((T')*y);

%% ideal system (no noise, no saturation)
sys_d = c2d(G,Te);
[g_d,t_d] = impulse(sys_d*Te);
```

Listing 6

Matlab code about the Finite impulse response

This problem can also be solved using the regulatiration aproach, this method adds a weight λ to the parameters to ensure that we do not try to invert a singular matrix in the least squares algorithm. We have found that $\lambda = 2$ worked well for this system. Increasing λ makes so the optimisation will try to minimise more Θ

The solution to this problem is given by the formula 4 where Φ is the full asymetric Toeplitz matrix.

We can see the matlab code of linear regression applied to our system in the Listing 7 below.

$$\Theta = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T Y \tag{4}$$

```
    lambda = 2;

r = zeros(size(u));
r(1) = u(1);
T_full = toeplitz(u,r); % full N*N asymetric toeplitz matrix

g_reg = inv(T_full'*T_full+lambda*eye(size(T_full)))*(T_full')*y; % solve the least squares problem

%% plots
figure
hold on
```

```

plot(tt(1:K),g_fir)
plot(t_d,g_d)
plot(tt(1:K),g_reg(1:K))

err_fir = g_d-g_fir;
err_reg = g_d-g_reg(1:K);

norm_err_fir = norm(err_fir,2)
norm_err_reg = norm(err_reg,2)


title("Impulse response")
legend("Indentified response (FIR)","Ideal response (no saturation, no noise)","Indentified response (Regularisation)")
xlabel("time [s]")
ylabel("impulse response [arbitrary units]")
```

Listing 7

Matlab code about the regularisation approach and plots

We can see the comparison of both method: Finite Impulse Response and Regularisation compared to the ideal response, on the Figure 5 below. The 2-norm error of the FIR is 0.1746 and the 2-norm error of the regularisation response is 0.1934. We can conclude that both metods have similar result, however, for this system, FIR is slightly better in term of error minimization and tuning K is more intuitive than tunning λ. To compare to the ideal impulse response, we had to multiply matlab's impulse(G) by the sampling time T_e since the function impulse returns the response to a Kronecker delta of amplitude $\frac{1}{T_e}$.

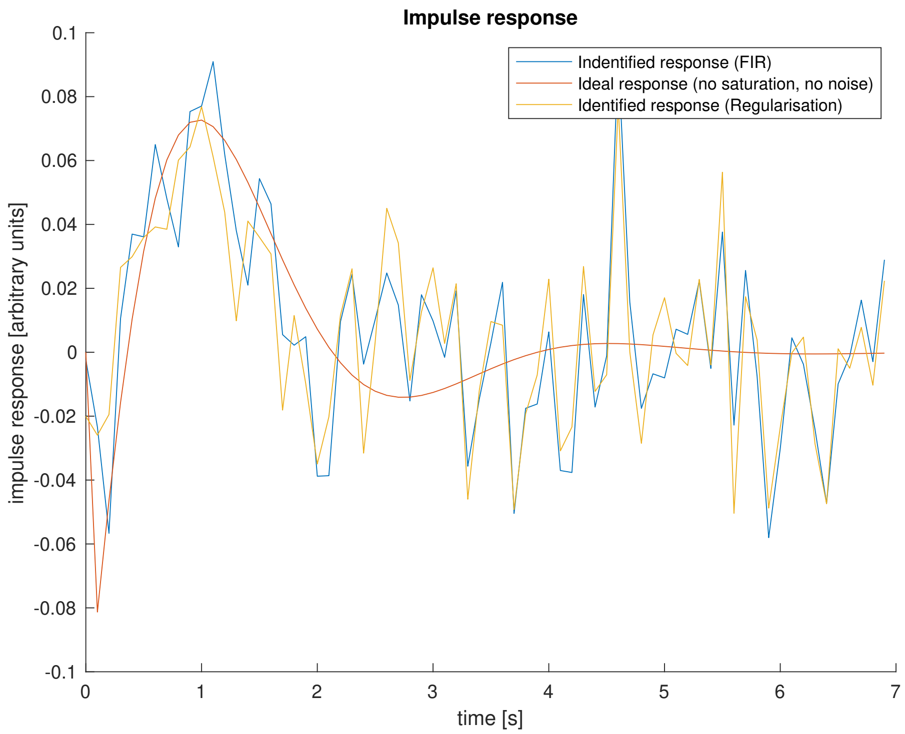

Figure 5

Identified impulse response compared with the true one using two different algorithms (finite impulse response in blue and regularisation in yellow)

1.4 Impulse response by correlation approach

In this part, we want to compute again the Impulse response but this time by using the correlation approach.

Now we do not use the inputs, but the correlation and autocorelation function to create the toeplitz matrix. The formula is:

$$R_{yu} = U\Theta \tag{5}$$

Where U is now a symetric toeplitz matrix created with the vector of the auto-correlation of the input R_{uu}, and R_{yu} is the vector with the cross-correlation of the output and input:

$$R_{yu} = \begin{bmatrix} R_{yu}(0) \\ . \\ . \\ . \\ R_{yu}(M-1) \end{bmatrix} \tag{6}$$

$$R_{uu} = \begin{bmatrix} R_{uu}(0) \\ . \\ . \\ . \\ R_{uu}(M-1) \end{bmatrix} \tag{7}$$

The measurement noise is way smaller for the correlation approach since in open-loop u and y (the input and output of the plant $G(s)$) are uncorrelated.

Since the signals are periodic, the correlation function can be exactly calculated. This is not exactly the case because of noise, but it is already way better than the normal convolution approach.

We only used the last period of the PRPS signal to ensure that the transient phase was already gone. We have found that including more than a period of the PRBS signal made the toeplitz matrix singular. The results could have been inproved by averaging the correlation functions over several periods, but the results obtained seamed sufficient without the need of averaging.

The 2-norm of the error using our correlation function was 0.1996. Using matlab's function, it was 0.1452. So we can conclude that both our function, and matlab's inplementation give good results.

```
    %%
sat_up = 0.5; % must fix the maximum amplitude as to not saturate the input
Uprbs = prbs(8,8)*sat_up;
Te = 0.1;
tt = (0:Te:(size(Uprbs,1)-1)*Te)';

% pass to simulink stuct
simin.time = tt;
simin.signals.values = Uprbs;

% call the simulation
out_step = sim('exo4.slx',tt(end));
tt_sim = out_step.simout.Time;
y_sim = out_step.simout.Data;

% pick only the last period of the signal
M = 2^(8)-1;
Uprbs = Uprbs(7*M + 1:8*M);
y_sim = y_sim(7*M + 1:8*M);
tt_sim = tt_sim(1:M);

R_uu_intcor = intcor(Uprbs,Uprbs);
R_yu_intcor = intcor(y_sim,Uprbs);

U_toeplitz_intcor = toeplitz(R_uu_intcor);

g_k_intcor = pinv(U_toeplitz_intcor)*(R_yu_intcor');

%% now with matlab

R_uu_matlab = xcorr(Uprbs,Uprbs);
R_yu_matlab = xcorr(y_sim,Uprbs);

%keep only the positive part of the correlations functions

```

```
R_yu_matlab = R_yu_matlab((end+1)/2:end);
R_uu_matlab = R_uu_matlab((end+1)/2:end);

U_toeplitz_matlab =  toeplitz(R_uu_matlab);
g_k_matlab =  pinv(U_toeplitz_matlab)*R_yu_matlab;

plot(R_uu_matlab)

figure
% figure


%% exact reponse of the system
G = tf([-1 2],[1 1.85 4]);
G = c2d(G,Te);
g_theory = impulse(G,tt_sim)*Te;

%% plots
hold on
plot(tt_sim,g_k_intcor)
plot(tt_sim,g_k_matlab)
plot(tt_sim,g_theory)

err_intcor = g_theory-g_k_intcor;
err_matlab = g_theory-g_k_matlab;

err_norm_intcor = norm(err_intcor,2)
err_norm_matlab = norm(err_matlab,2)

title("Impulse response g(t)")
xlabel("discrete time k")
legend("Impulse reponse with intcor()","Impulse reponse with xcorr()","Theoretical
    impulse (no noise, no saturation)")
```

Listing 8

Matlab code about computing the impulse response using intcor and xcorr

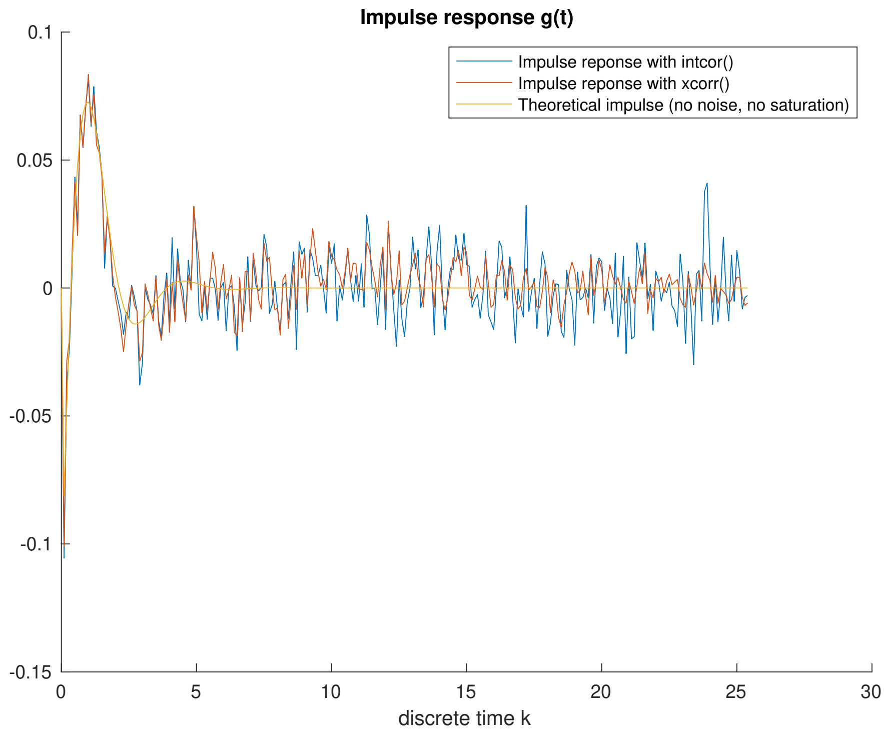

Figure 6

Identified impulse responses by correlation approach compared with the true one.

1.5 Frequency domain Identification (Periodic signal)

For the PRBS, we chose $p = 16$ and to get more than 2000 points, we chose $n = 7$, that gives us 2032 points in the PRBS signal. We average the fourier coefficients over the multiple periods, but skiping the first period to ignore the transient response of the system.

We can see that the bode diagram of the identified system is quite similar to the ideal bode diagram. This is because we averaged the fourier coefficients over several periods and this reduces the mesurement errors. We have no truncation error since the signals are periodic, and we respect the Shannon's theorem so we should not get any considerable sampling error.

```
clc,close all,clear all
%% 1

sat_up = 0.5; % must fix the maximum amplitude as to not saturate the input
p = 16;

N_wanted = 2000;
n_min = ceil(log2((N_wanted+1)/p)); %minimum shift register length

Uprbs = prbs(n_min,p)*sat_up;
size(Uprbs)
Te = 0.1;
tt = (0:Te:(size(Uprbs,1)-1)*Te)';

% pass to simulink stuct
simin.time = tt;
simin.signals.values = Uprbs;

% call the simulation
out_step = sim('exo5.slx',tt(end));
tt_sim = out_step.simout.Time;
y_sim = out_step.simout.Data;

%% 2

N = size(Uprbs,1);

Uprbs_fft_avg = 0;%initialiation
y_fft_avg = 0;%initialiation
N_one_period = N/p;

number_periods_to_ignore = 1;

for i = (1+number_periods_to_ignore*N_one_period):N_one_period:N
    Uprbs_period = Uprbs(i:i+N_one_period-1);
    Uprbs_fft_avg = (fft(Uprbs_period) + Uprbs_fft_avg);

    y_period = y_sim(i:i+N_one_period-1);
    y_fft_avg = (fft(y_period) + y_fft_avg);
end
Uprbs_fft_avg = Uprbs_fft_avg/(p-number_periods_to_ignore);
y_fft_avg = y_fft_avg/(p-number_periods_to_ignore);


%% 3

f_s = 1/Te;
omega_s = 2*pi*f_s;

omega_vec = (omega_s./N_one_period).*(0:(N_one_period-1));

G = y_fft_avg./Uprbs_fft_avg;

%% 4
sys_f = frd(G(1:floor(end/2)),omega_vec(1:floor(end/2)),Te);

%%
G = tf([-1 2],[1 1.85 4]);
G = c2d(G,Te);


bode(sys_f,G)
legend("identified system","ideal system")
```

Listing 9

Matlab code about Fourier analysis

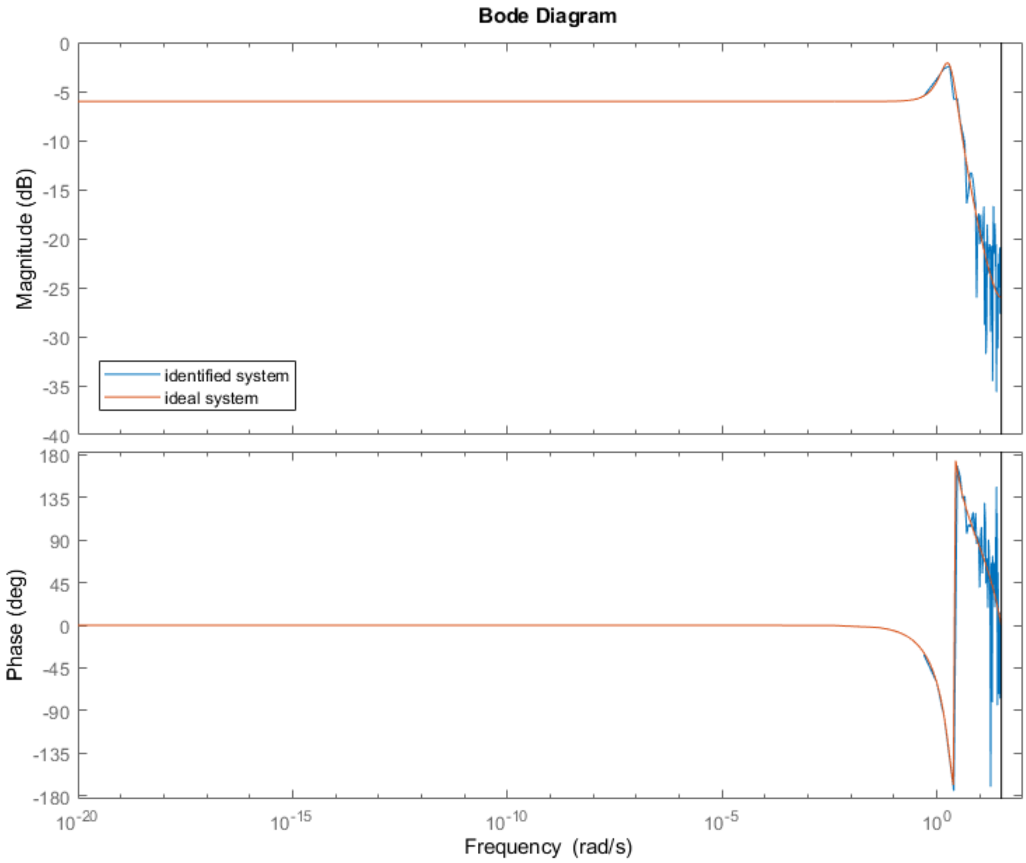

Figure 7
Bode diagram of the identified model [blue] compared with the true one [orange].

1.6 Frequency domain Identification (Random signal)

We have chosen to apply binary random signal to the system to deliver the maximum amount of energy on the signal and thus increase the signal to noise ratio.

We have chosen uniform white noise. The important part is that its white noise, so that the we have a flat spectrum to excite as much frequencies as possible.

For the first identification, we use a biased correlation estimate because the unbiased estimate has a high variance for large $|k|$. Using a biased estimate is equivalent as using an unbiased one with a triangular window of size N.

To reduce the truncation error, we do a second identification using a Hann window. It is important to note that we only use half of the window for the positive part or the correlation functions.

Because we are using the window, we can use the unbiased estimate of the correlation functions.

We have found that a window size of $M = 300$ gave us good results (the actual entire window has size $2M$). Increasing M increases the resolution but also increases the spectrum leakage problem. For the Hann window, the secondary lobes are very small so this is not a problem.

Using a $M < N$ is equivalent of padding the rest of the window with zeros. And in our case, $M < N$ gave a better result than for $M = N$. This is again because of the high variance for large $|k|$ of the unbiased correlation estimate.

For the last identification, we have split the data in $m = 10$ groups and averaged the power spectrum functions ϕ_{uu} and ϕ_{yu} to reduce the variance of the estimated parameters. We have also used the Hann window on each segment to reduce the truncation error.

We can see from our results that while the first identification has worked, it has a lot of truncation error that is significantly removed by using the Hann window. The second identification using the window is thus way better. The last identification averaging the power spectrum functions seams to help a bit more but most of the noise was coming from truncation so the second and third identification are quite similar.

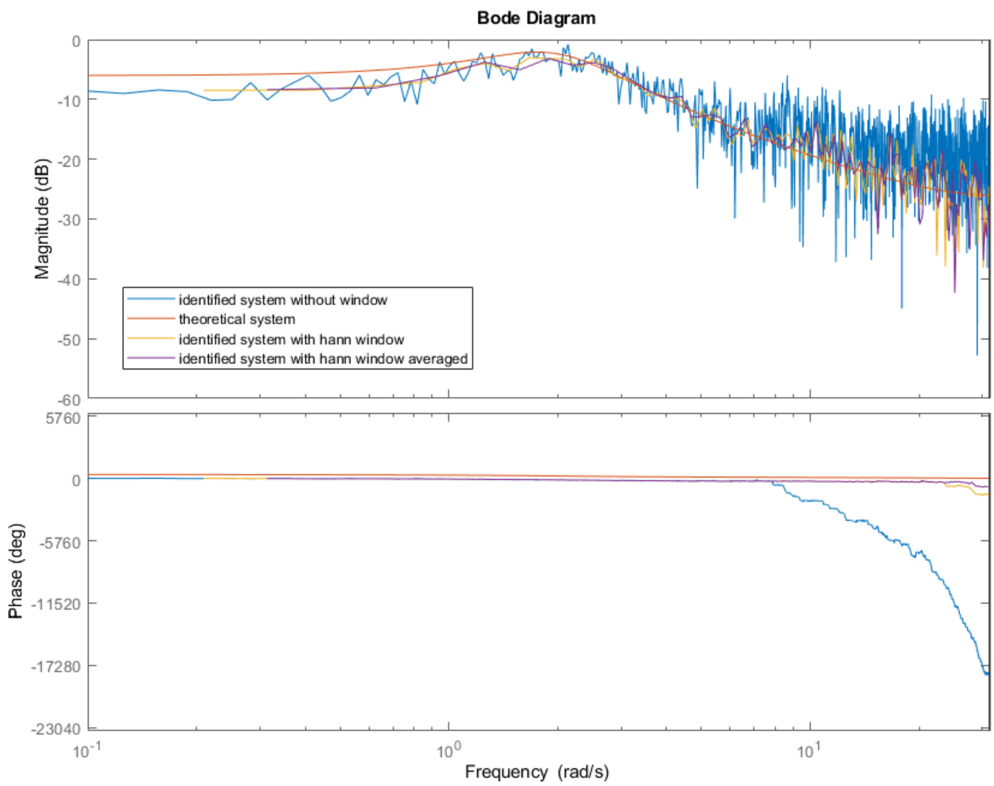

Figure 8

Bode diagram of the identified models and the ideal model.

```
clc,close all,clear all
%% exo 6
```

```matlab
%% 1
N = 2000;
sat_up = 0.5;
Te = 0.1;

U = rand(N,1)-0.5;

U(U<0)=-sat_up;
U(U>0)=sat_up;

tt = (0:Te:(size(U,1)-1)*Te)';

% call simulink
simin.time = tt;
simin.signals.values = U;
out_step = sim('exo6.slx',tt(end));
tt_sim = out_step.simout.Time;
y_sim = out_step.simout.Data;


%% 2

Ruu = xcorr(U,U,'biased');
Ryu = xcorr(y_sim,U,'biased');

%keep only the positive part of the correlations functions
Ruu =  Ruu((end+1)/2:end);
Ryu =  Ryu((end+1)/2:end);

phi_uu = fft(Ruu);
phi_yu = fft(Ryu);

f_s = 1/Te;
omega_s = 2*pi*f_s;
omega_vec = (omega_s./N).*(0:(N-1));


G_id = phi_yu./phi_uu;

sys_id = frd(G_id(1:floor(end/2)),omega_vec(1:floor(end/2)),Te);

% real system (no noise, no saturation)
G = tf([-1 2],[1 1.85 4]);
G = c2d(G,Te);
%% 3

M = 300;
window = hann(2*M);
window = window((end)/2 + 1:end);

Ruu = xcorr(U,U,'unbiased');
Ryu = xcorr(y_sim,U,'unbiased');

%keep only the positive part of the correlations functions
Ruu =  Ruu((end+1)/2:end);
Ryu =  Ryu((end+1)/2:end);


phi_uu_windowed = fft(Ruu(1:M).*window);
phi_yu_windowed = fft(Ryu(1:M).*window);
G_windowed = phi_yu_windowed./phi_uu_windowed;

omega_vec_windowed = (omega_s./M).*(0:(M-1));

sys_id_windowed = frd(G_windowed(1:floor(end/2)),omega_vec_windowed(1:floor(end/2))
    ,Te);

%% 4

m = 10;
samples_per_group = N/m;

window = hann(2*samples_per_group);
window = window((end)/2 + 1:end);

phi_uu_avg = zeros(samples_per_group,1);%initialiation
```

```
phi_yu_avg = zeros(samples_per_group,1);%initialiation

for i = 1:samples_per_group:N
    U_period = U(i:i+samples_per_group-1);
    y_period = y_sim(i:i+samples_per_group-1);


    Ruu = xcorr(U_period,U_period,'unbiased');
    Ryu = xcorr(y_period,U_period,'unbiased');

    Ruu =  Ruu((end+1)/2:end);
    Ryu =  Ryu((end+1)/2:end);

    phi_uu_avg = fft(Ruu.*window) + phi_uu_avg;
    phi_yu_avg = fft(Ryu.*window) + phi_yu_avg;
end

phi_uu_avg = phi_uu_avg/m;
phi_yu_avg = phi_yu_avg/m;

G_windowed_avg = phi_yu_avg./phi_uu_avg;
omega_vec_windowed_avg = (omega_s./samples_per_group).*(0:(samples_per_group-1));

sys_id_windowed_avg = frd(G_windowed_avg(1:floor(end/2)),omega_vec_windowed_avg(1:
    floor(end/2)),Te);


%% 5

figure
bode(sys_id,G,sys_id_windowed,sys_id_windowed_avg);
legend("identified system without window","theoretical system","identified system
    with hann window","identified system with hann window averaged")
xlim([10^-1 inf])
```

Listing 10

Matlab code about frequency domain identification