Michael Biselx (283812)
Grégoire Lacroix (288605)
Xavier Nal (288275)

Computer Exercice 2: Parametric Identification Methods

Goal

The second computer exercise is meant to demonstrate the use of parametric methods for identifying system models.
Two systems are analyzed : a laser beam stabilization mechanism and a flexible link.

Exercise 1 : Beam Stabilizing Mechanism

Exercise 1.1 : FIR model identification

Give the code for computing the parameters.
Plot the measured and predicted output in the same figure and give the value of loss function.
Give the code for computing the covariance of the parameters. Plot the impulse response together with confidence intervals.

The code for computing the parameters θ of the model is as follows :

```
% make the Toeplitz matrix
Phi     = toeplitz(u, [u(1), zeros(1, K-1)]);

% get the FIR predictor via LS algorithm
theta   = (Phi.' * Phi)\(Phi.' * y);

% predict the output
y_hat   = Phi * theta;
```

This yields a loss function value of J = 7.56. Using this, the covariance of the parameters may be estimated as follows, yielding the impulse response estimate visible in figure 1 :

```
% estimation of the noise variance
sigma_noise = sqrt(J/(N-K));

% covariance of the estimate
sigma_theta = sigma_noise * sqrt(diag(inv(Phi.'*Phi)));
```

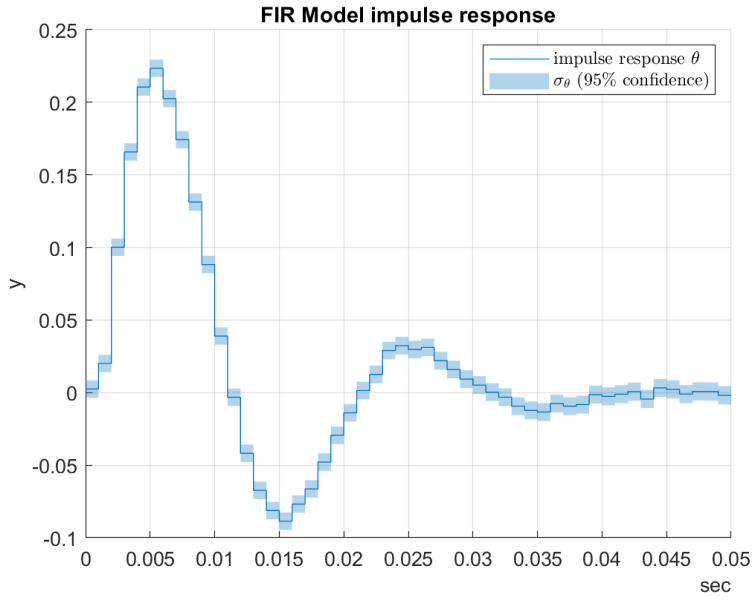


Fig. 1 : FIR model impulse response, for K = 50 parameters.

Exercise 1.2 : ARX model identification

Give the code for computing the parameters. Give the parameters.
Plot the measured and predicted output in the same figure and give the two-norm of the prediction error.
Give the code for computing the output of the identified model. Plot the measured output and the model output in the same figure. Compute the two-norm of the output error.
Give the code for the IV method. Give the parameters. Compare the output of the ARX model with that of IV model and the measured output. Comment on the results.

The ARX model identified from the data is the following :

$$G_{ARX}(z) = \frac{0.018\,z^{-1} + 0.073\,z^{-2}}{1 - 1.534\,z^{-1} + 0.635\,z^{-2}}$$

The code for computing its parameters and its output is :

```
% make the I/O matrix
Phi     = [[0; -y(1:end-1)], [ 0; 0; -y(1:end-2)], ...
           [0;  u(1:end-1)], [ 0; 0;  u(1:end-2)] ];

% get the ARX predictor via LS algorithm
theta   = (Phi.' * Phi)\(Phi.' * y);

% predict the output using the I/O matrix
y_hat   = Phi * theta;
```

```
% predict the output only using the input
G = tf([0 theta(3), theta(4)], [1 theta(1), theta(2)], 1/fs,
'Variable','z^-1');
ym = lsim(G, u);
```

This gives a decent match when using the I/O matrix (i.e. we get a loss of J=23.6), however when applying the model to the input, the loss rises to J=123.71.

The IV model identified based on the ARX model is as follows :

$$G_{IV}(z) = \frac{0.017\,z^{-1} + 0.068\,z^{-2}}{1 - 1.781\,z^{-1} + 0.877\,z^{-2}}$$

This model gives a loss function value of J=29.46. Its parameters and output were calculated using the following code :

```
% make the I/O matrix based on the ARX model
Phi_iv   = [[0; -ym(1:end-1)], [ 0; 0; -ym(1:end-2)], ...
            [0;  u(1:end-1)],  [ 0; 0;  u(1:end-2)] ];

% get the IV predictor via LS algorithm
theta_iv = (Phi_iv.' * Phi)\(Phi_iv.' * y);

% predict the output only using the input
G_iv = tf([0 theta_iv(3), theta_iv(4)], [1 theta_iv(1),
theta_iv(2)], 1/fs, 'Variable','z^-1');
y_m_iv = lsim(G_iv, u);
```

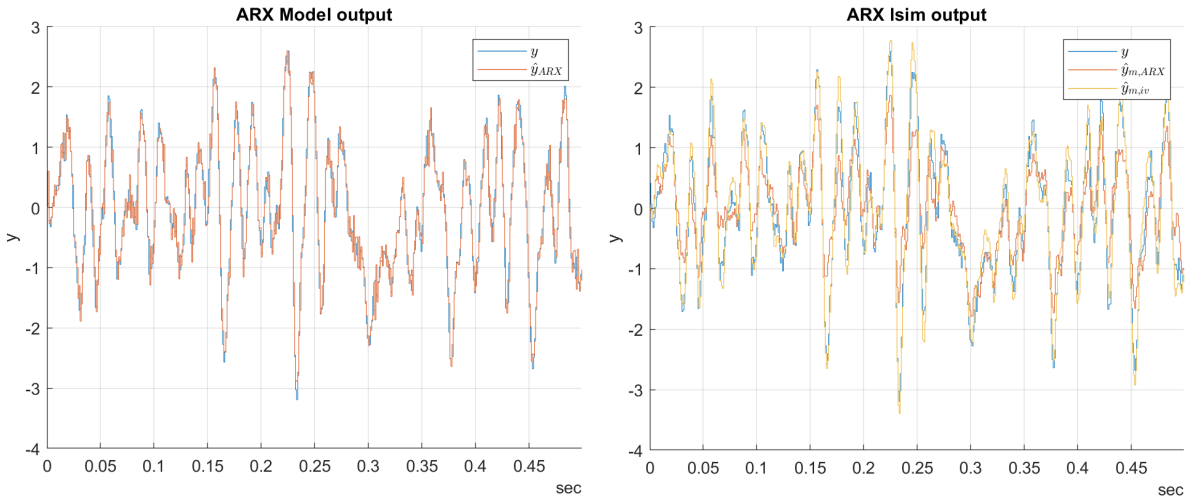


*Fig. 2 : **Left** : plot of the output of the ARX model using the input/output matrix Φ, which yields a relatively low loss function.*
***Right**: However, when simulating the model using lsim(), we find a different situation. The ARX model gives a high loss function. Therefore, we apply the IV method, which gives a good result.*

Exercise 1.3 : State-space model identification

Give the code for computing Q.
Plot the singular values of Q and find the number of states.
Give the code for computing A and C. Give A and C.
Give the code for computing B. Give B.
Give the code for computing the output of the model. Plot the measured output and the model output in the same figure. Compute the two-norm of the output error

The state-space model of the system is built using the Q matrix, which is set up using the following bit of Matlab code :

```
r  = 10; % initial rank assumption
K  = length(u)+1-r;
Y  = zeros(r, K);
U  = zeros(r, K);
for k = 1:K
    Y(:,k) = y(k + (0:r-1));
    U(:,k) = u(k + (0:r-1));
end

U_orth = eye(K) - U.'* ((U*U.') \ U);
Q = Y*U_orth;
```

Using the singular values of Q (figure 3), we determine the rank of the system, and thus the number of states to compute, to be n=2. Attributing a higher rank would reduce the loss function, but at the risk of overfitting.

Computing the state space matrices can be done with the following code :

```
% extended observability matrix
O = Q(:,1:n);

% compute state-space matrices
C = O(1:ny,:);
A = O(1:((r-1)*ny),:) \ O(ny+1:end,:);
```

The input matrices can be computed using this code :

```
% compute input matrix
q = tf('q', 1/fs);
f = C/(q*eye(size(A)) - A);
uf = zeros(length(time), n);
for i = 1:n
    uf(:,i) = lsim(f(i), u, seconds(time));
end
Phi = [uf, u].';

% control model matrices
theta = (y.'/Phi).';
B = theta(1 : (end-size(u,2)));
D = theta((end-size(u,2)+1) : end);
```

The identified system is listed below. It yields a loss function of J=42.47.

```
A =
           x1        x2
 x1    0.2603   -0.9923
 x2    0.3732     1.377

B =
           u1
 x1   0.04605
 x2   -0.2959

C =
            x1         x2
 y1     0.4024   0.006652

D =
             u1
 y1   -0.001092
```

Calculating the model output is as simple as running the following two lines. Its output can be found in figure 4.

```
G_SS   = ss(A, B, C, D, 1/fs);
y_m_SS = lsim(G_SS, u);
```

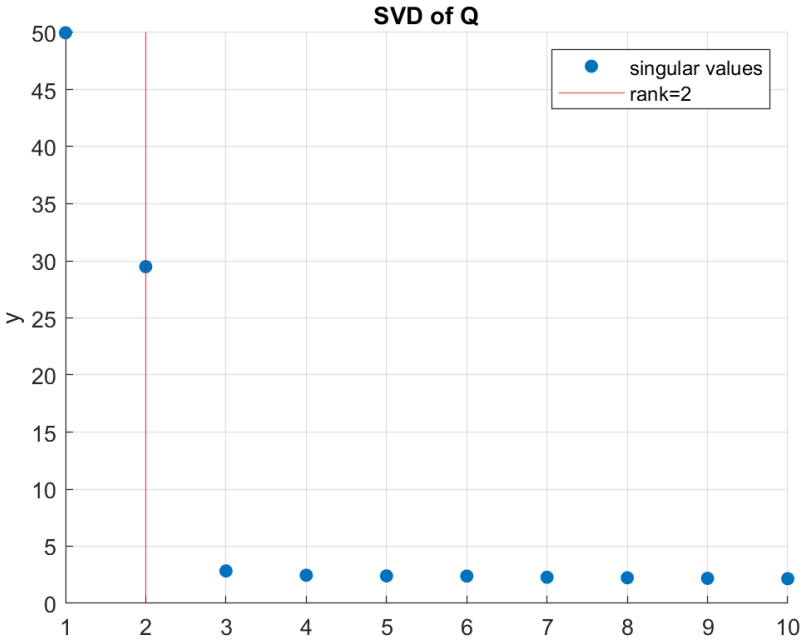


Fig. 3 : Singular values of Q.

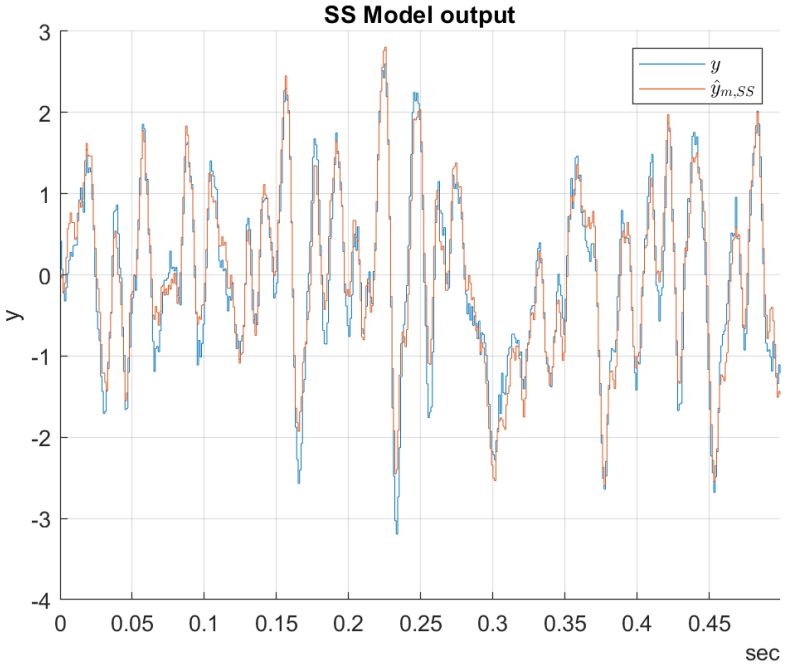


Fig. 4 : Output of the state-space model compared to the ground truth.

Exercise 2 : Flexible Link System

Exercise 2.1 : Order estimation

Plot the Bode diagram and estimate the minimum order of the system.
Plot the loss function evolution and estimate the model order using the loss function.
Plot the zero/pole maps of several models and conclude the order of the model.
Estimate the delay (give the code). Compute the number of parameters in the numerator.
Compare your final results with those proposed by Matlab

Using the evolution of the loss function over the order of ARX models (figure 5), we estimate the true order of the system to be 6. We validate this assumption by plotting the Zero/Pole maps for a model above this order and a model below this order (figure 6), expecting to see pole cancellation in the model of higher order, but in none of the others. This is the case.

We then estimate the delay of the model to be 0, using the following lines of code :

```
% estimation of nk
nk = find(abs(model_oe.b(2:end)) > .1*max(abs(model_oe.b)), ...
          true, 'first');
delay = nk - 1;
```

This then allows us to compute the number of parameters in the ARX model by trial-and-error :

```
% estimation of nb
model_arx_nb = cell(n-1+1, 1);
for nb = 1:n-1+1
    model_arx_nb{nb} = arx(io_data, [n, nb, 1]);
end
loss = cellfun(@(M) (M.EstimationInfo.LossFcn), model_arx_nb);
[~, nb] = min(loss);

% estimation of na
na = n;
```

This gives us an ARX model with orders `[na, nb, nk] = [6, 6, 1]`. Using `selstruc()`, Matlab proposes an ARX model with orders `[na, nb, nk] = [10, 10, 2]` (figure 7), which seems a little bit excessive, but does have a very low misfit.

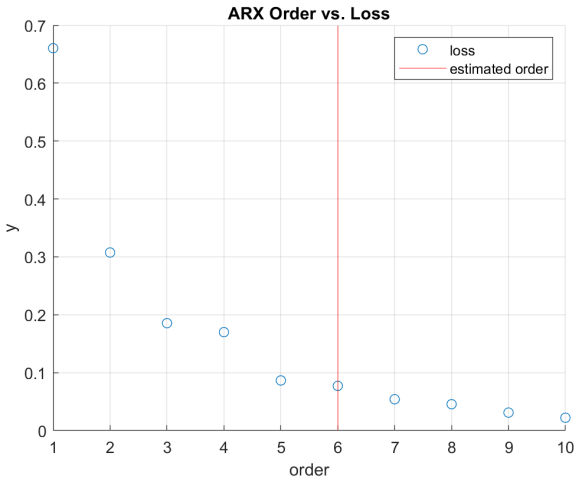


Fig. 5 : evolution of the loss function for ARX models vs. their order

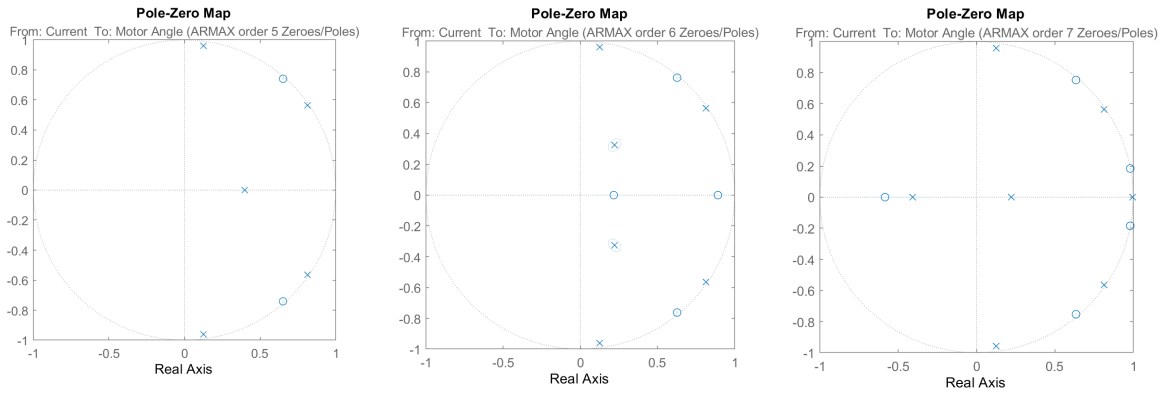


*Fig. 6 : **Left** : Zero/Pole map for a model of order 5. The order is too low.*
***Center** : Zero/Pole map for a model of order 6. New poles do not cancel out*
***Right** : Zero/Pole map for a model of order 7. The order is too high, and some pole cancellation occurs in the left half-plane.*

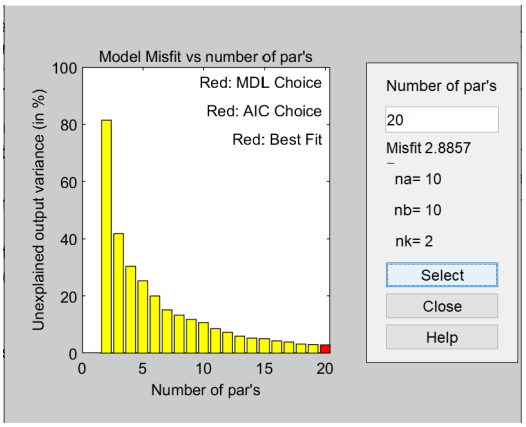


Fig. 7 : The 'best' model proposed by Matlab's `selstruc()` function has a whopping 20 parameters.

Exercise 2.2 : Parametric identification

Give the code for data partitioning.
Give the code for identification with different model structures

The code below divides data in two equal parts. One part is for identification and the other is for the model validation.

```
% divide data into equal identification / validation portions
split_idx = round(1/2*length(y));

idt_data = iddata(y(1:split_idx), u(1:split_idx), Ts, ...
              'Name', 'Flexible Link Identification', ...
              'InputName', 'Current', 'OutputName', 'Motor Angle');
val_data = iddata(y(split_idx+1:end), u(split_idx+1:end), Ts, ...
              'Name', 'Flexible Link Validation', ...
              'InputName', 'Current', 'OutputName', 'Motor Angle');
```

We then use the identification data and orders to create a number of different models, using different model structures. The code for this is as follows :

```
% set unset orders for parametric models
nc = na;
nd = na;
nf = na;

% create parametric models
model_arx    = arx(idt_data, [na, nb, nk]);
model_iv4    = iv4(idt_data, [na, nb, nk]);
model_armax  = armax(idt_data, [na, nb, nc, nk]);
model_oe     = oe(idt_data, [nb, nf, nk]);
model_bj     = bj(idt_data, [nb, nc, nd, nf, nk]);
model_n4sid  = n4sid(idt_data, n);

% prepare for plotting
models = {model_arx, model_iv4, model_armax, model_oe, ...
          model_bj, model_n4sid};
titles = ["model_arx", "model_iv4", "model_armax", "model_oe", ...
          "model_bj", "model_n4sid"];
```

Exercise 2.3 : Model validation

Compare the output of the identified models with the measured output using the command compare. What is the best model ?
Compare the frequency response of the models with a nonparametric frequency-domain identified model. What is the best model ?
Validate the identified models by the whiteness test of the residuals as well as the cross-correlation of the residuals and past inputs using the command resid. Which models are validated ?

The following code compares the models identified using the different model structures against the validation data in the time domain (fig. 8) :

```
% compare models in the time domain
fig = figure();
compare(val_data, models{:})
for i = 1:length(models)
    fig.Children(3).String{1+i} = titles(i);
end
```

The results show that the best models are the ARMAX model with 78.39%, the Output Error model with 73.65% and the Box-Jenkins model with 73.75% of fit.

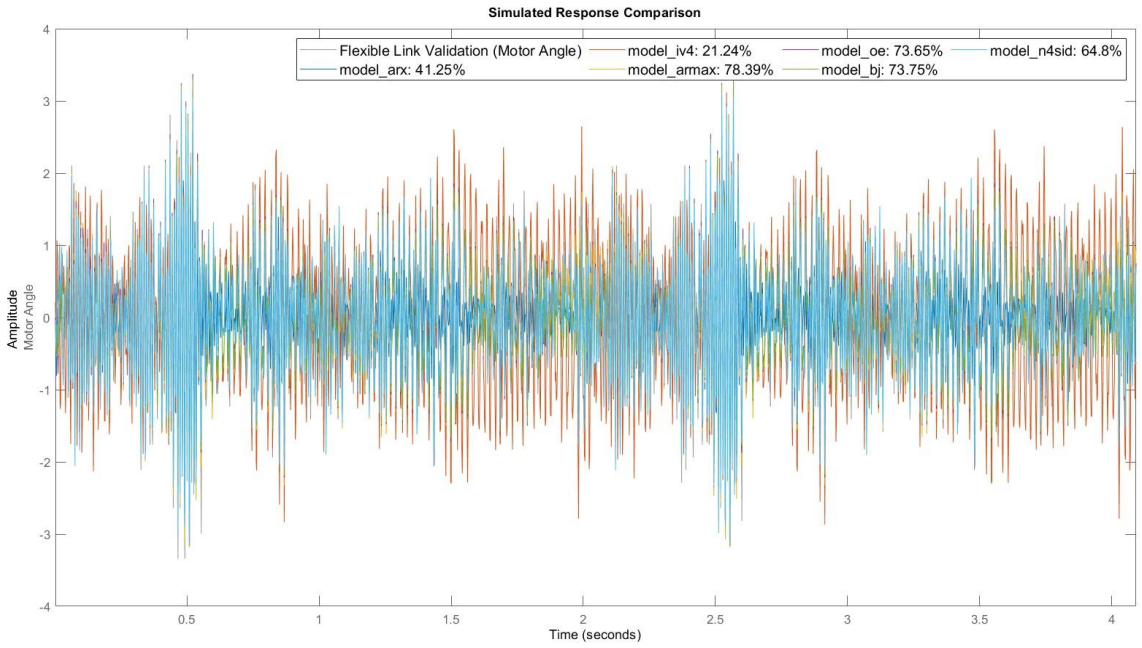


Fig. 8 : Comparison of the different model outputs and their fit to the validation data.

The code below compares the frequency response of the different models to the validation data. The results (fig. 9) show that the best frequency responses are for the ARMAX model with 83.47% fit, the Box-Jenkins model with 82.74% fit and the Output Error model with 82.4% fit. The fitting is better at higher frequencies, near the resonant modes of the system.

```
% spectrum of the validation data using a large Hann window
model_spa = spa(val_data, 1000);

% compare models in the frequency domain
fig = figure();
compare(model_spa, models{:})
fig.Children(3).String{1} = 'model\_spa';
for i = 1:length(models)
    fig.Children(3).String{1+i} = titles(i);
end
```

The code below was used to plot the whiteness test of the residuals as well as the cross-correlation of the residuals and past inputs.

```
for i = 1:length(models)
      figure('Name', titles{i});
      resid(val_data, models{i});
end
```

The results (fig. 10) show that the whiteness test for the Output Error model is not good, as almost all the peaks are above 0.2 (if the noise was truly white, only the instantaneous autocorrelation should rise above the noise floor). For the ARMAX and Box-Jenkins models, the autocorrelation shows peaks between -0.2 and 0.2, except for autocorrelation with 0. We can consider that both models are valid.

Considering all these validation tests we conclude that the ARMAX model structure best captures the dynamics of this system.

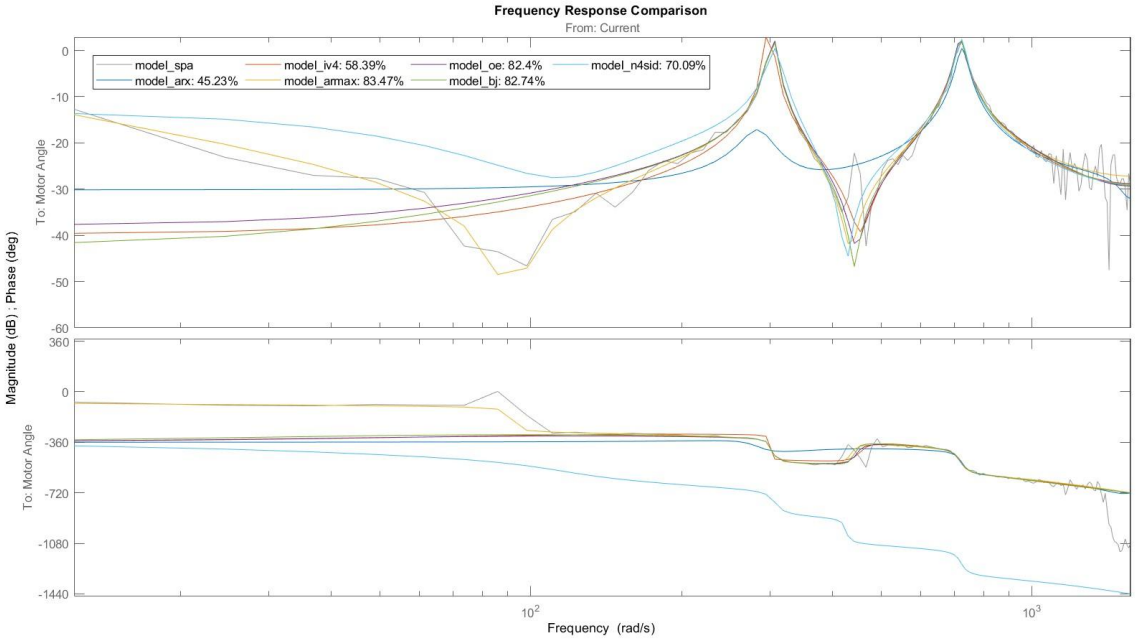
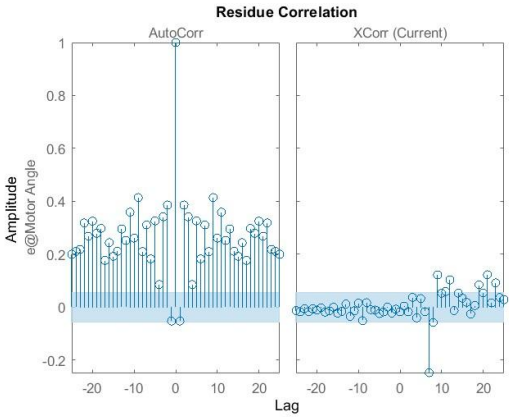
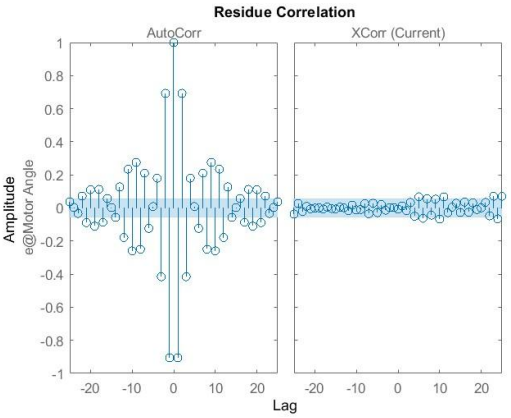


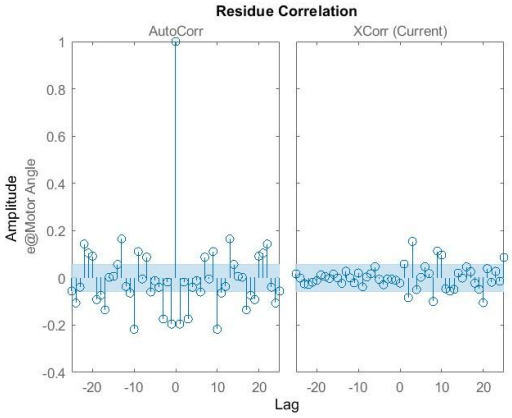
Fig. 9 : Comparison to the validation data of the frequency responses of the models.



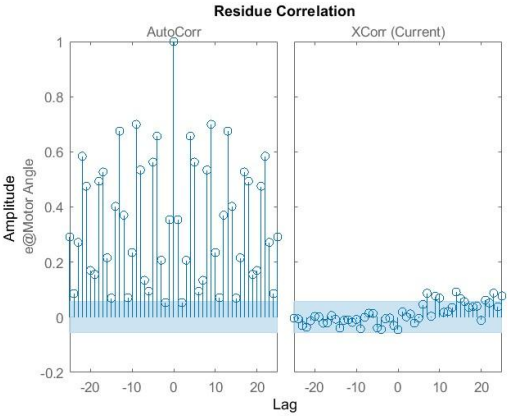
Output computed with ARX



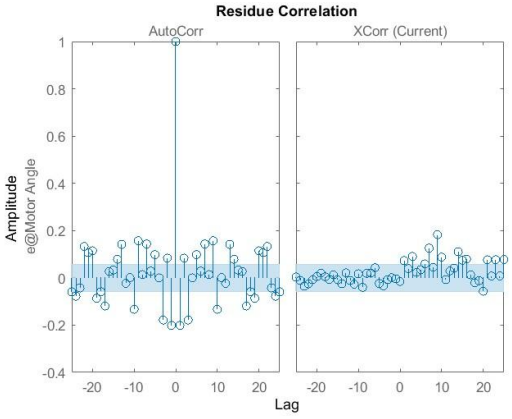
Output computed with iv4



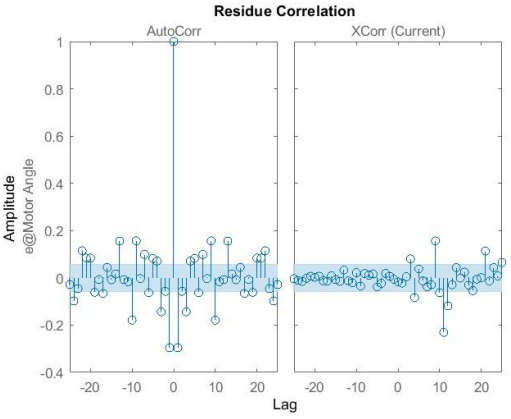
Output computed with ARMAX



Output computed with output error



Output computed with model bj



Output computed with model n4sid

Fig. 10 : whiteness tests for the identified models

ANNEX

The full code for this computer exercise can be found on our GitHub.

Full output for CE2.1

```
1. FIR model
    The value of the loss function J for the FIR is   7.562
2. ARX model
2.1. 2nd order ARX
    The ARX model is : (0.018 z^-1 +0.073 z^-2)/(1 -1.534 z^-1 +0.635 z^-2)
    When using the data, the loss function J is      23.601
    When simulating ARX, the loss function J is     123.708
2.2. IV model based on ARX
    The IV model is : (0.017 z^-1 +0.068 z^-2)/(1 -1.781 z^-1 +0.877 z^-2)
    When simulating IV, the loss function J is       29.458
3. State Space Model
    The estimated rank is 2
    When simulating SS, the loss function J is       42.474
>>
```

Full output for CE2.2

```
1. Order Estimation
1.1. ARX order estimation
    ARX order estimated as 6
1.2. ARMAX validation
    please check the figures for Zero/Pole cancellation
1.3. estimate delay
    estimated delay is 0 sample(s)
    order of nb for minimum loss is 6
1.4. comparison using selstruc
    selected ARX model is: [10, 10, 2]
2. Parametric Identification
2.1. divide data into identification / validation
    done
2.2. create other parametric models
    done
3. Model Validation
3.1. compare models in time domain
    ARX : 41.3%, IV4 : 21.2%, ARMAX : 78.4%, OE : 73.7%, BJ : 73.5%, N4SID : 64.8%,
3.2. compare models in frequency domain
    ARX : 45.2%, IV4 : 58.4%, ARMAX : 83.5%, OE : 82.4%, BJ : 82.8%, N4SID : 70.1%,
3.3. check whiteness of residuals
    ARX is not OK, IV4 is not OK, ARMAX is OK, OE is not OK, BJ is OK, N4SID is OK,
>>
```