System identification
Spring 2025

CE2 Report : Parametric Identification Methods

Risse Maxime (345491), Jules Streit (327732), Yo-Shiun Cheng (386249)
Group 25

Professor Alireza KARIMI

1 Identification of a DC servomotor

1.1 FIR and model identification

The regression problem is formulated as:

$$\mathbf{y} = \Phi\theta + \mathbf{e} \tag{1}$$

where:

- $\mathbf{y} \in R^N$ is the output vector
- $\Phi \in R^{N\times m}$ is the Toeplitz-structured regression matrix:

```
tpc = u; % First column
tpr = [u(1); zeros(m-1,1)]; % First row
phi = toeplitz(tpc,tpr); % Input delay matrix
```

$$\Phi = \begin{bmatrix} u(1) & 0 & \cdots & 0 \\ u(2) & u(1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u(N) & u(N-1) & \cdots & u(N-m+1) \end{bmatrix} \tag{2}$$

- $\theta \in R^m$ contains the FIR coefficients estimated via:

```
theta_hat = phi \ y; % Least squares solution
y_hat = phi * theta_hat;     % predicted output
J = (norm(y - y_hat))^2 % loss function is the sum of squares of the
    prediction error
```

The predicted output and loss function are computed as:

$$\hat{\mathbf{y}} = \Phi\hat{\theta}, \quad J(\hat{\theta}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = 477.6267 \tag{3}$$

The parameter covariance matrix is estimated accounting for noise variance:

$$\hat{\sigma}^2 = \frac{J(\hat{\theta})}{N-m}, \quad \mathrm{Cov}(\hat{\theta}) = \hat{\sigma}^2(\Phi^\top\Phi)^{-1} \tag{4}$$

```
var_hat = J/(N - m);
covar_hat = var_hat * inv(phi'*phi);
sigma = sqrt(diag(covar_hat)); % Parameter std deviations
```

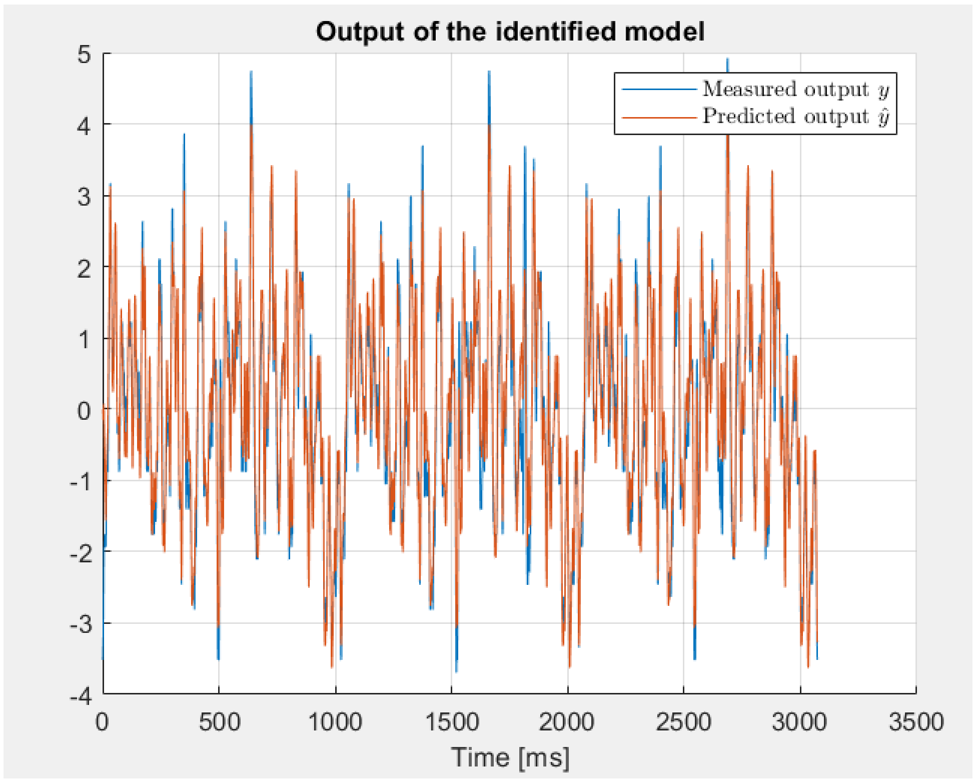


Figure 1: Measured output $y(k)$ vs predicted output $\hat{y}(k)$.

1.2 ARX model identification

We now assume a second order auto-regressive with external input (ARX) model for the same data as in the previous section. Consequently, we use the following predictor

$$\hat{y}(k,\theta) = -a_1 y(k-1) - a_2 y(k-2) + b_1 u(k-1) + b_2 u(k-2) = \phi^T(k)\theta \tag{5}$$

where

$$\begin{aligned} \phi^T(k) &= [-y(k-1), -y(k-2), u(k-1), u(k-2)] \qquad (6)\\ \theta^T &= [a_1, a_2, b_1, b_2]\,. \qquad (7) \end{aligned}$$

We observe that the system has no modeled delay as $d = 0$. Using the least squares algorithm and the Moore-Penrose pseudo inverse $\boldsymbol{\phi}^+$ of $\boldsymbol{\phi}$, we compute, like we already did in previous section about FIR, the parameter vector $\hat{\theta}$ according to

$$\hat{\theta} = \left[\sum_{k=1}^{N} \phi(k)\phi^T(k)\right]^{-1} \sum_{k=1}^{N} \phi(k)y(k) = \left(\boldsymbol{\phi}^T\boldsymbol{\phi}\right)^{-1} \boldsymbol{\phi}^T \mathbf{Y} = \boldsymbol{\phi}^+ \mathbf{Y}. \tag{8}$$

We construct the regression matrix based on the chosen predictor structure using past input and output values. The parameters are then estimated using the least squares method with the backslash operator.

Listing 1 on page 3 shows the MATLAB implementation.

```
path_ = "CE2_data.m";
[y, u, Ts] = GetExperimentData(path_);

N = size(u,1); % number of input samples

function [Big_Phi] = create_BigPhi_matrix(u, y, N)
    Big_Phi = zeros(N, 4);
```

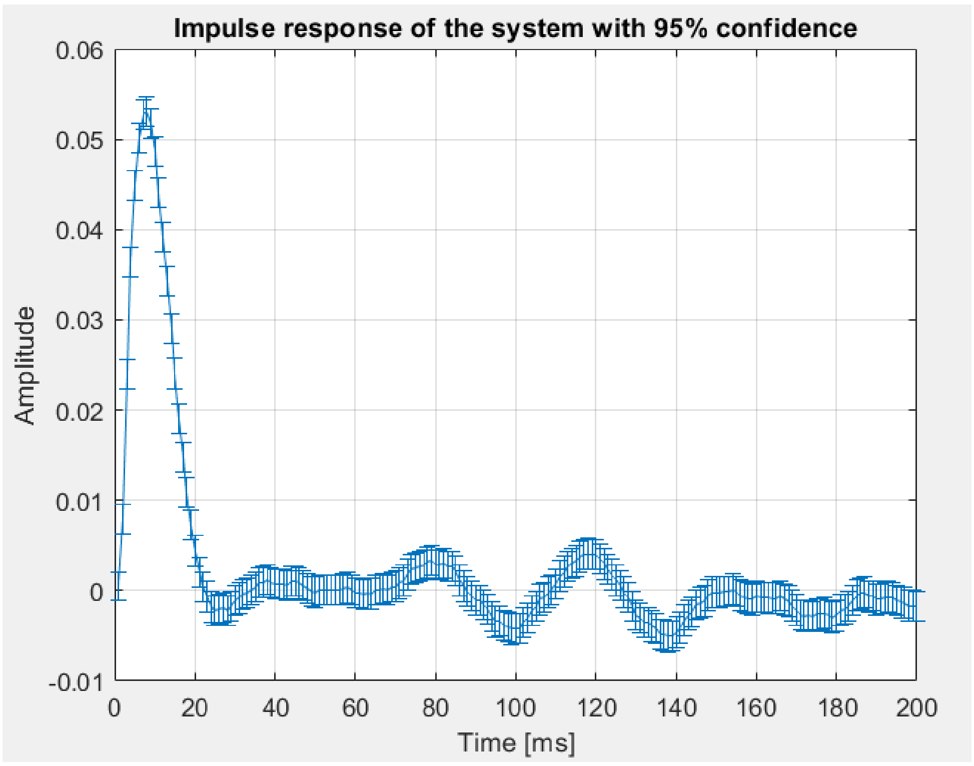


Figure 2: Impulse response with 95% confidence intervals.

```
    Big_Phi(2,:) = [-y(1), 0, u(1), 0];
    for k = 3:N
        Big_Phi(k,:) = [-y(k-1), -y(k-2), u(k-1), u(k-2)];
    end
end

BigPhi = create_BigPhi_matrix(u, y, N);

% Estimate the parameter vector using least squares
theta_ARX = BigPhi \ y;
```

Listing 1: ARX parameter estimation in MATLAB

The estimated parameter vector is:

$$\theta_{\text{ARX}} = \begin{bmatrix} -1.6522 \\ 0.6861 \\ 0.0073 \\ 0.0111 \end{bmatrix}$$

The predicted output of the identified model, denoted as $\hat{y}(k, \hat{\theta})$, is computed using the relation

$$\hat{y}(k, \hat{\theta}) = \phi^{\top}(k)\,\hat{\theta},$$

where $\phi(k)$ is the regression vector at time step k. In order to estimate the quality of our predictor, we need a fit criterion, which should be minimized. Hence, we consider a loss function as

$$J(\theta) = \sum_{k=1}^{N} [y(k) - \hat{y}(k, \theta)]^2 . \tag{9}$$

The resulting prediction error, quantified by the loss function, yields a value of $J_{\text{ARX}} = 61.8342$.

```
%Question 2
y_ARX = BigPhi * theta_ARX;
```

```
J_ARX = norm(y - y_ARX) ^ 2 % loss function is the sum of squares of the
    prediction error
```

The model's performance is illustrated in Figure 3, where the predicted output is compared to the measured data.

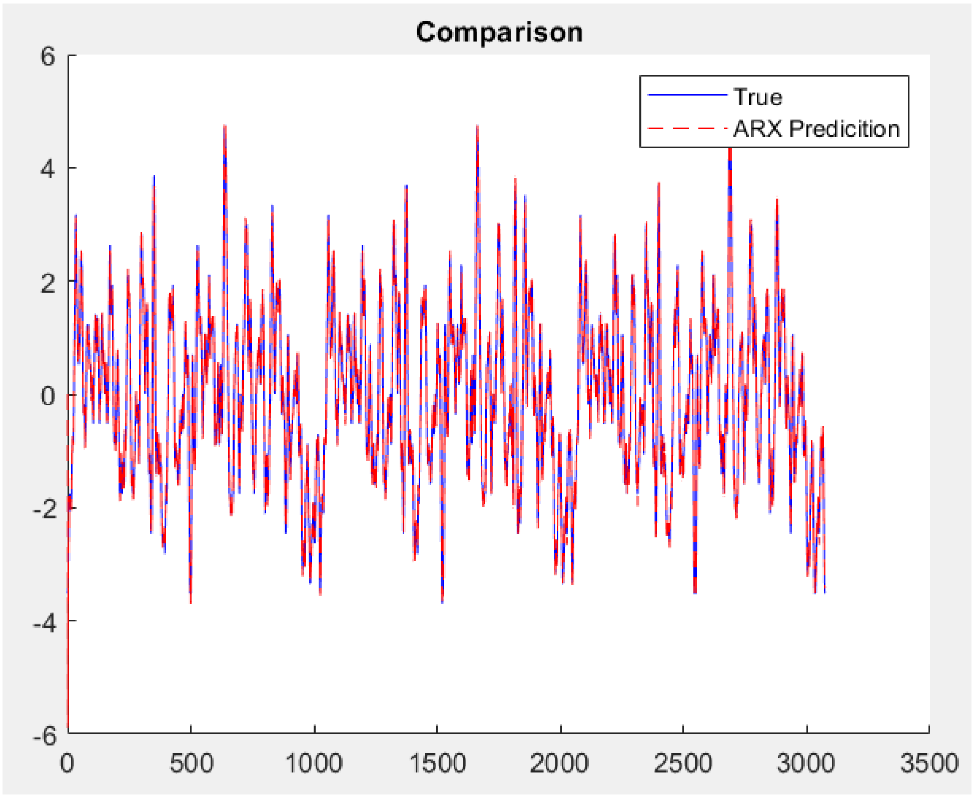


Figure 3: Comparison between measured output and ARX model prediction

Afterwards we compute the output of the identified model $y_m(k)$ using the MATLAB command `lsim` and the transfer function of the model

$$G_0(q) = \frac{b_1 q^{-1} + b_2 q^{-2}}{1 + a_1 q^{-1} + a_2 q^{-2}} = \frac{0.0073q^{-1} + 0.0111q^{-2}}{1 - 1.6522q^{-1} + 0.6861q^{-2}}\,. \tag{10}$$

```
%Question 3
a1 = theta_ARX(1);
a2 = theta_ARX(2);
b1 = theta_ARX(3);
b2 = theta_ARX(4);

f_sampling = 1e3;

q = tf('q', 1/f_sampling);
sys = (b1*(1/q) + b2*(1/q^2)) / ...
        (1 + a1*(1/q) + a2*(1/q^2));

% Simulate transfer function model
t = 0:(1/f_sampling):(N-1)*(1/f_sampling);
ym = lsim(sys, u, t);

J_tf = norm(ym - y)^2
```

We generate a transfer function object using the MATLAB command `tf` and simulate the output given the input $u(k)$ and we plot the simulated output ym resulting from the second order model with the measured output in Figure 4. The resulting sum of the squares of the error is then $J_{tf}(\hat{\theta}) \approx 965.0174$.

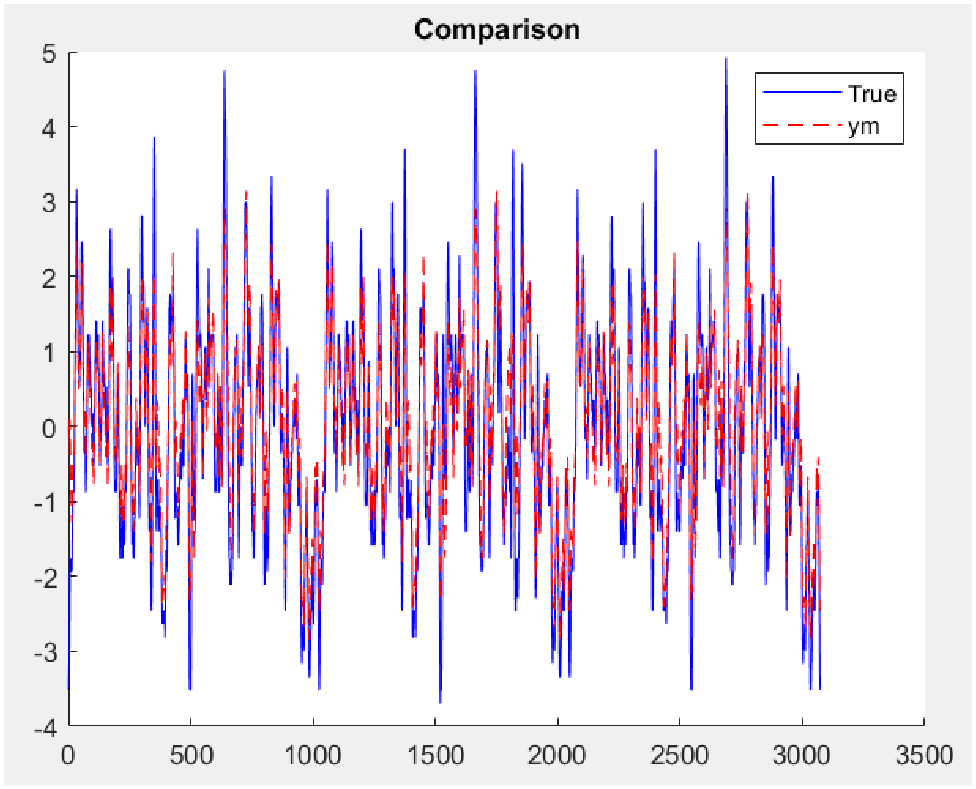


Figure 4: Comparison between measured output and output of the identified model ym

In order to obtain unbiased estimates the following two conditions must be satisfied:

1. $R_{\phi\phi}$ is not singular,
2. $R_{\phi e}(0) = 0$.

These conditions can be satisfied by replacing the non-transposed ϕ's in the estimator for $\hat{\theta}$ by a new vector ϕ_{iv} called the vector of instrumental variables (IV):

$$\hat{\theta}_{iv} = \left[\sum_{k=1}^{N} \phi_{iv}(k)\phi^T(k)\right]^{-1}\left[\sum_{k=1}^{N} \phi_{iv}(k)y(k)\right] \tag{11}$$

Therefore, the estimates are asymptotically unbiased, if

1. $R_{\phi_{iv}\phi}$ is not singular,
2. $R_{\phi_{iv}e}(0) = 0$.

That means, choosing an vector of instrumental variables, that is correlated with the regressor vector ϕ and uncorrelated with the noise e leads to asymptotically unbiased estimates. One choice of this vector is keeping the past, non-noisy inputs in the vector and replacing the past noisy outputs using the output of an already identified model $M(q^{-1})$. A noiseless output of this model can be computed by

$$y_m(k) = M(q^{-1})u(k) \tag{12}$$

and the vector ϕ_{iv} is defined as

$$\phi_{iv}^T(k) = [-y_M(k-1), \ldots, -y_M(k-n), u(k-d-1), \ldots, u(k-d-m)] \tag{13}$$

with $d = 0$. Therefore, we can first identify an ARX model without the use of instrumental variables. Afterwards, we use the identified parameters as instrumental variables and repeat the ARX fitting. In order to obtain better results, this process can be repeated for several iterations but we choose to do it one time. The resulting parameter vector is $\hat{\theta}_{iv}^T = [-1.7983, 0.8300, 0.0075, 0.0101]$. The result of the instrumental variables method compared to the measured output and the predicted output of the transfer function model can be seen in Figure 5 and the implementation can be viewed, again, in Listing 1 (page 3). We can observe, that the predicted output of the model identified using the IV method is closer to the measured output.

```
% Question 4 - Instrumental Variable (IV) Method

% Step 1: Create the IV matrix (using y_ARX as instrumental variables)
Big_Phi_IV = zeros(N,4);
Big_Phi_IV(2,:) = [-y_ARX(1), 0, u(1), 0];
for k=3:N
    Big_Phi_IV(k,:) = [-y_ARX(k-1), -y_ARX(k-2), u(k-1), u(k-2)];
end

% Step 2: Estimate parameters with IV
theta_IV = inv(Big_Phi_IV' * BigPhi) * Big_Phi_IV' * y;  % IV estimator formula

% Step 3: Compute predicted output with IV model
y_IV = BigPhi * theta_IV;
J_IV = norm(y - y_IV)^2  % Loss function for IV

% Step 4: Compare with ARX results
fprintf('ARX Loss: %.4f | IV Loss: %.4f\n', J_ARX, J_IV);

%Plot ARX vs IV in subplots
figure;
subplot(1,2,1); % Left subplot: ARX
hold on;
plot(y, 'b', 'LineWidth', 1.5);
plot(y_ARX, 'r--', 'LineWidth', 1.5);
legend("True", "ARX Prediction");
title("ARX Model");
xlabel("Time (samples)"); ylabel("Output");
grid on;

subplot(1,2,2); % Right subplot: IV
hold on;
plot(y, 'b', 'LineWidth', 1.5);
plot(y_IV, 'g:', 'LineWidth', 1.5);
legend("True", "IV Prediction");
title("IV Model");
xlabel("Time (samples)");
grid on;

% Step 5: Simulate IV model (like Q3 for ARX)
sys_IV = (theta_IV(3)*(1/q) + theta_IV(4)*(1/q^2)) / ...
         (1 + theta_IV(1)*(1/q) + theta_IV(2)*(1/q^2));
ym_IV = lsim(sys_IV, u, t);

% Compare errors
J_tf_IV = norm(ym_IV - y)^2;
fprintf('ARX Sim Error: %.4f | IV Sim Error: %.4f\n', J_tf, J_tf_IV);
```

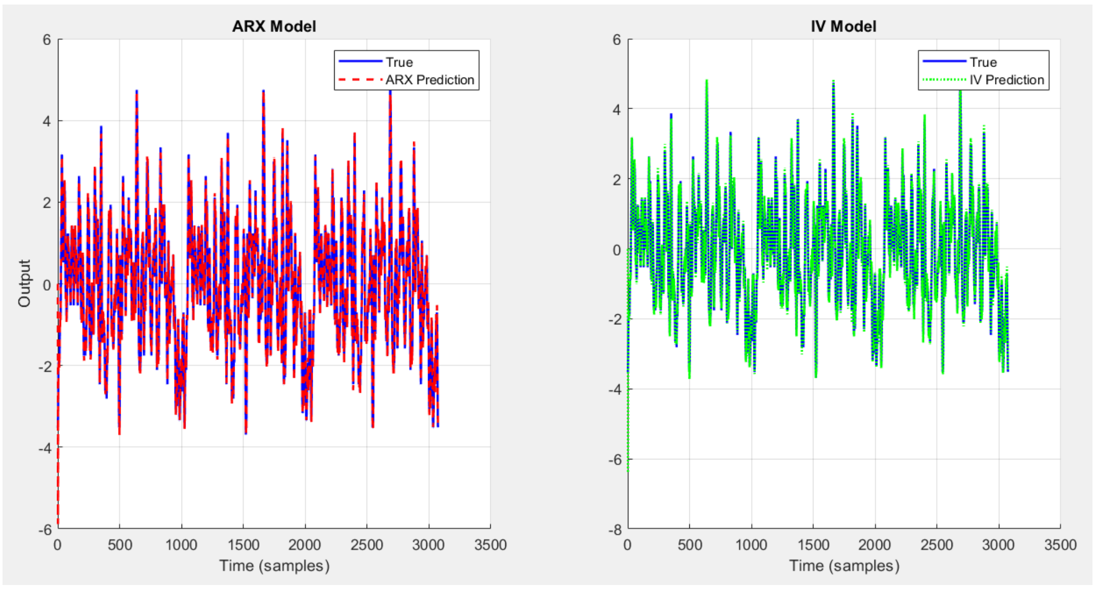


Figure 5: Measured output vs ARX model and IV Model

We obtain the following results :
ARX Loss: 61.8342 | IV Loss: 67.2347
ARX Sim Error: 965.0174 | IV Sim Error: 783.4593

The ARX method yields a prediction error of **61.8342** versus **67.2347** for IV, but IV reduces the simulation error from **965.0174** (ARX) to **783.4593**, demonstrating its effectiveness at estimating the true system dynamics.

1.3 State-space model identification

Introduction

The goal of state-space model identification is to represent a dynamic system using state-space equations. For a discrete-time linear system, these equations are:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned}$$

where:

- x_k is the state vector at time k,
- u_k is the input vector,
- y_k is the output vector,
- A, B, C, and D are the system matrices to be identified.

The objective is to estimate these matrices from measured input-output data.

Computation of Q

We compute Q as follows:

```matlab
path_ = "CE2_data.m";
[y, u, Ts] = GetExperimentData(path_);

r = 10; % maximal order of the system
N = size(u, 1); % input size

%Question 1

% Construct U and Y
Y = zeros(r, N);
U = zeros(r, N);
for i=1:N
    for j=1:r
        k = i+j-1;
        if k > N
            Y(j,i) = 0;
            U(j,i) = 0;
        else
            Y(j,i) = y(k);
            U(j,i) = u(k);
        end
    end
end

U_orthogonal = eye(N) - U' * inv(U*U') * U;
Q = Y * U_orthogonal;
```

Singular Value Decomposition (SVD) of Q

Performing the singular value decomposition (SVD) of matrix Q allows us to estimate the system's order, i.e., the number of its internal states. By examining the descending singular values, we deduce n = 3 since we judge that further values are too close to 0.

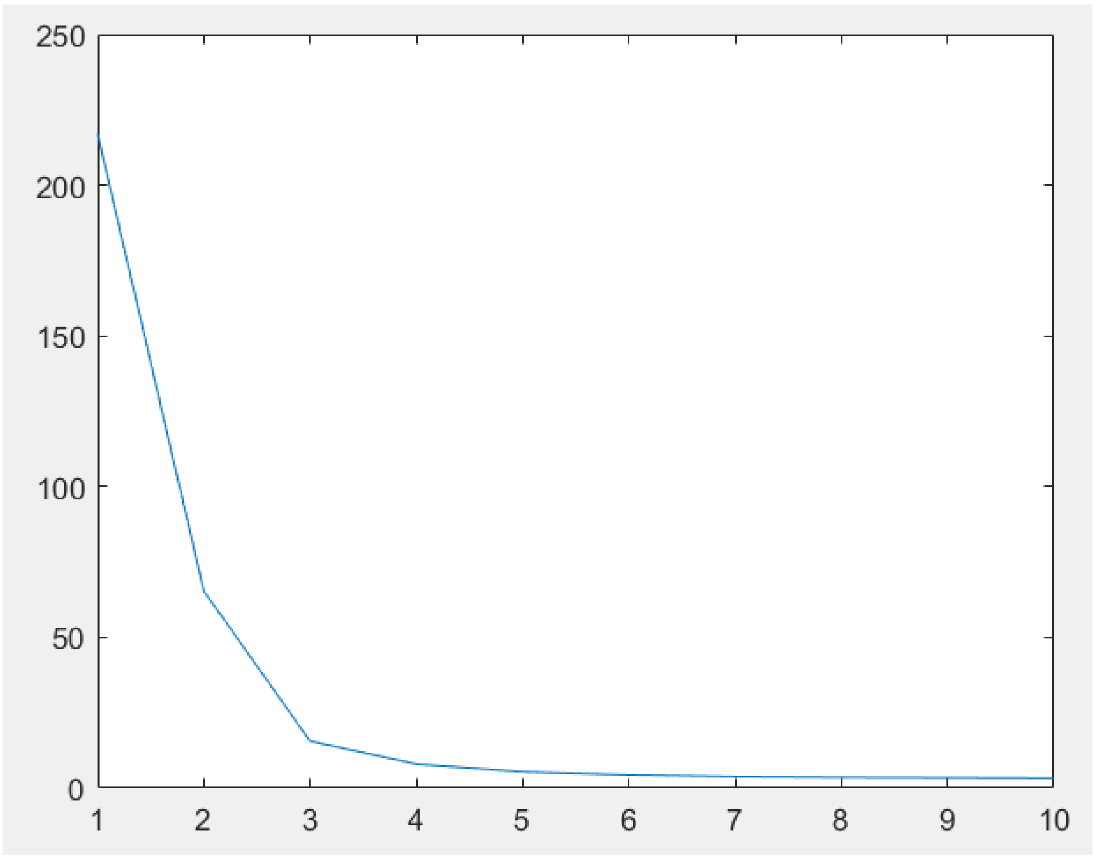


Figure 6: Singular values of Q

Estimation of Matrices A and C

Once the system order n is determined, we extract the first n singular vectors (n first columns of UU) to create the extended observability matrix Or. The matrix C is obtained from the first row of this matrix. To estimate the state transition matrix $\hat{A}$, we utilize the shift-invariance property of the observability matrix. Specifically, the relationship between consecutive rows of O_r can be expressed as:

$$O_r^{\text{lower}} = O_r^{\text{upper}} \cdot \hat{A}$$

where: - O_r^{upper} comprises the first $(r-1)$ rows of O_r, - O_r^{lower} comprises the last $(r-1)$ rows of O_r.

This equation reflects the principle that each subsequent block row of the observability matrix is obtained by multiplying the previous block row by the state transition matrix A. Solving this equation for $\hat{A}$ yields an estimate of the state transition dynamics of the system.

```
% from the plot we guess n = 3
n = 3;
Or = UU(:, 1:n);

%Question 3
C = Or(1, :);
A_hat = pinv(Or(1:(r-1),:)) * Or(2:r,:);
```

We find the following matrix :

$$C = \begin{bmatrix} 0.3371 & -0.5665 & 0.4452 \end{bmatrix}$$

$$\hat{A} = \begin{bmatrix} 0.9534 & 0.3249 & 0.0159 \\ -0.0680 & 0.8437 & 0.6440 \\ 0.0033 & -0.0152 & 0.9691 \end{bmatrix}$$

Estimation of Matrix B

To estimate the input matrix B, we utilize the system's transfer function, which relates the input $u(k)$ to the output $y(k)$ in the frequency domain. The transfer function is given by:

$$y(k) = C(qI - \hat{A})^{-1}Bu(k) + Du(k)$$

Here, q denotes the forward shift operator, D is equal to zero and $\hat{A}$, C are the previously estimated system matrices. The term $\hat{C}(qI - \hat{A})^{-1}$ represents the system's impulse response.

To facilitate the estimation, we define an auxiliary variable $u_f(k)$ as:

$$u_f(k) = C(qI - \hat{A})^{-1}u(k)$$

Solving this problem yields the estimate $\hat{B}$. Specifically, the solution is given by:

$$\hat{B} = (u_f)^{-1} \cdot y$$

```matlab
% compute uf and then B estimate
q = tf('z');
F = C * inv(q * eye(n) - A_hat);
uf = zeros(N, n);
for i = 1 : n
    uf(:, i) = lsim(F(i), u);
end

% assume D = 0
D = 0;
B_hat = pinv(uf) * y;
```

We find the following matrix :

$$\hat{B} = \begin{bmatrix} 0.1256 \\ 0.0617 \\ -0.0058 \end{bmatrix}$$

Model Simulation and Performance Evaluation

The identified model is simulated in response to input u, and its output is compared to the measured output y. The model's performance is evaluated by computing the 2-norm of the error between these two outputs, providing a quantitative measure of the model's fit to the data.

```matlab
sys = ss(A_hat, B_hat, C, D, Ts); %add 1e-3 as sapling time to have a discrete
    system
[y_sys, t_sys, x_sys] = lsim(sys, u);

fprintf('Loss function: %d\n', norm(y_sys - y)^2);
```

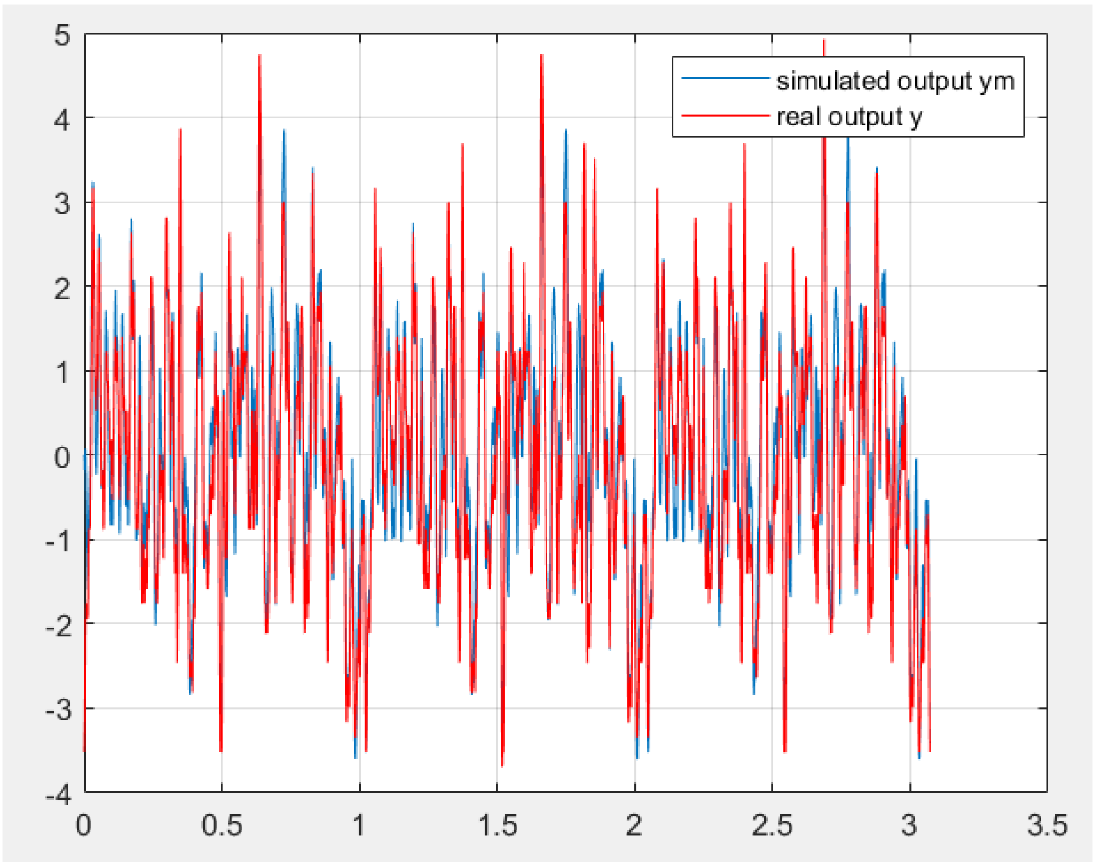


Figure 7: Comparison

Loss function: 8.395942e+02

2 Parametric Identification of an Electromechanical System

2.1 Order estimation

Bode diagram of nonparametric identified model

Here is the Bode plot with the real data that we have, if we take into account that we have 3 resonant peaks, which involves 2 conjugate poles each, implies that we have at least a 6th order system.

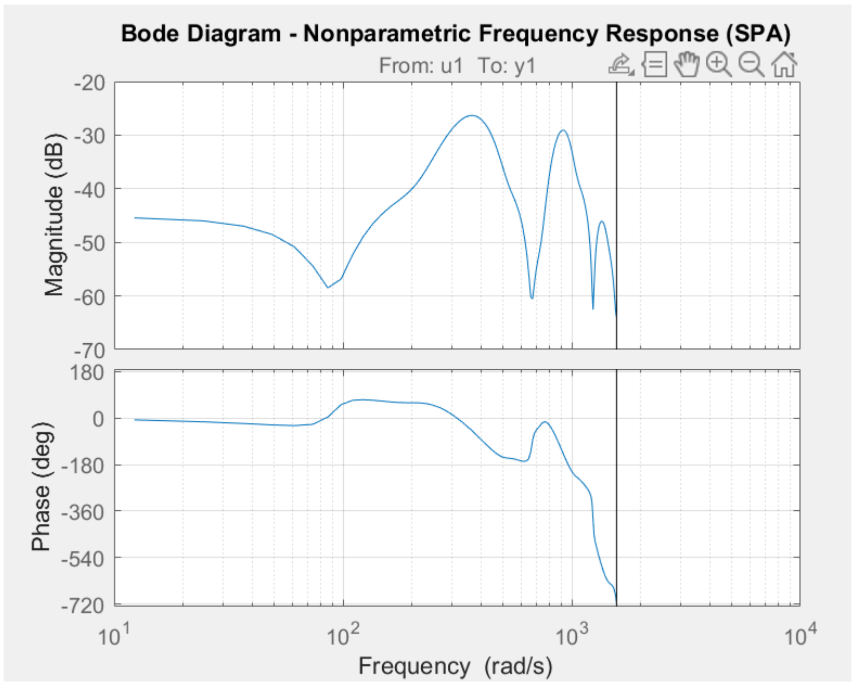


Figure 8: Bode diagram of the estimated ARX model

Loss Function Evolution and Model Order Selection

We plot the 'loss function' versus the model order. You can see the result below. If we look at the elbow of the curve we can estimate that the order of the system is 8.

Pole-Zero Map Analysis for Order Validation

When we look at zero/pole plots for consecutive orders from 5 to 10. We notice that zero pole cancellation begin to occur at n = 9. Indeed, we can observe that the new poles appearing at orders n=9n and n=10 are immediately canceled by zeros within their respective confidence intervals.

Delay Estimation and Numerator Parameter Count

From the course, we know that the first coefficients of $B(q^{-1})$ that are statistically close to zero (considering their standard deviation) indicate the time delay n_k. The vector `idx` is a binary array of the same size as `b`, where a value of 1 at index k indicates that 0 lies within the interval $[b_k - 2\sigma_k,\ b_k + 2\sigma_k]$; otherwise, the value is 0. We count the first values equal to 1, it is the delay that we seek.

```
%% Question 3

% Fix model order (based on previous steps)
na = 8;   % You can use the estimated order from the loss function curve
nb = 8;
nc = 8;

fprintf('\nEstimating nk by checking B coefficients:\n');

```

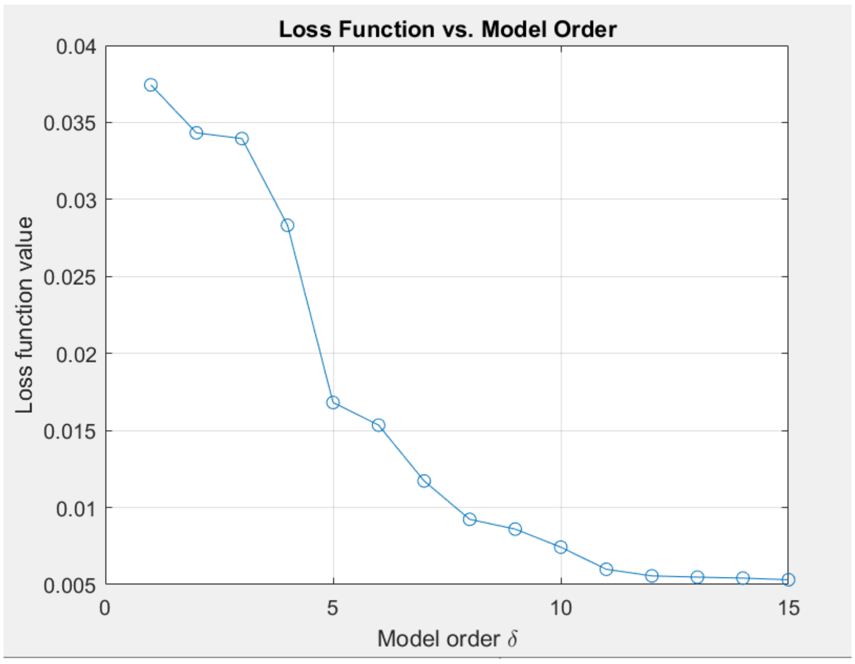


Figure 9: Loss function depending on order n

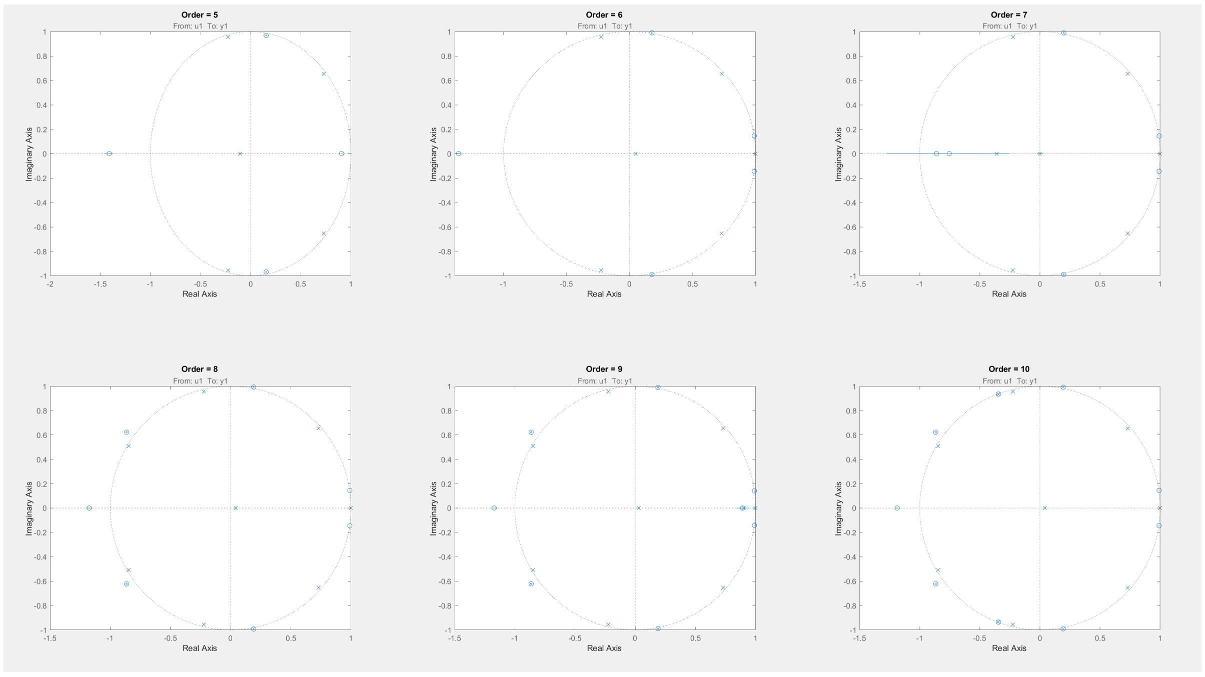


Figure 10: Zero poles plot for different orders

```
% Fit ARMAX model
model = armax(Z, [na nb nc 0]);

% Get numerator coefficients and their std deviations
b = model.b;
db = model.db;

lower = b - 2*db;
upper = b + 2*db;
idx = (lower <= 0) & (upper >= 0)  % 1 8  logical vector
%nk    = find(idx, 1);             % the indices k where 0   [ b _ k 2db_k , b_k+2
```

```
    db_k]
nk = 1; %TBD by looking at idx first values which are 1

% Display and analyze
fprintf('nk = %d | B = %s | std = %s\n', nk, mat2str(b, 3), mat2str(db, 3));

% Optional: plot coefficients vs. std deviation
figure;
errorbar(1:length(b), b, db, 'o');
title(['B Coefficients with Std Dev for nk = ' num2str(nk)]);
xlabel('Coefficient Index');
ylabel('Value');
grid on;
```

The resulting vector is:

$$\texttt{idx} = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]$$

Therefore, the delay is given by the index of the first 1 in `idx`, which gives $n_k = 1$. We can check that only the first coefficient of B is correct by looking at this plot:

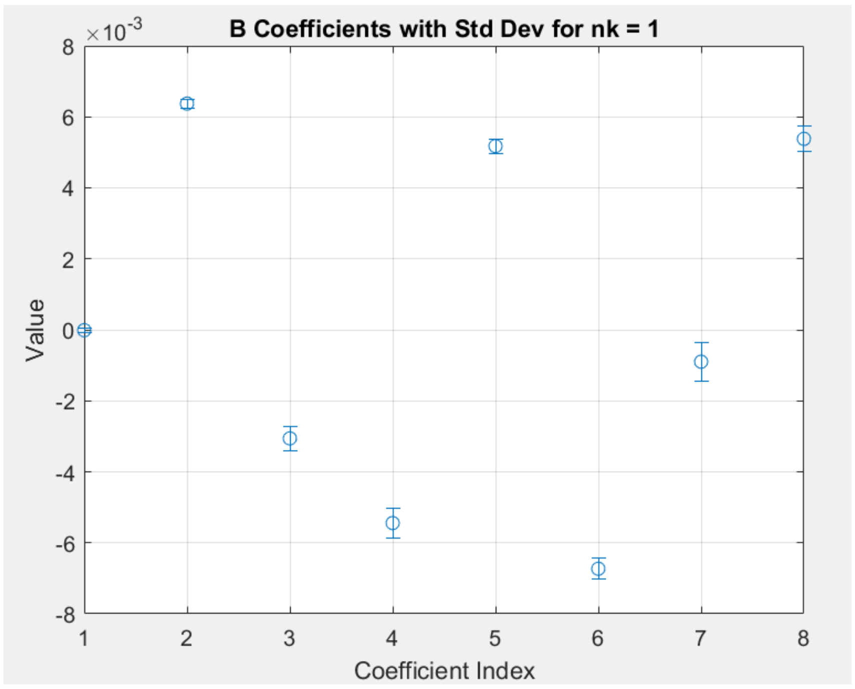


Figure 11: B coefficients

The number of parameters in the numerator is given by the formula $n_b = \delta - n_k + 1$. Substituting the values, we obtain $n_b = 8 - 1 + 1 = 8$.

Comparison with MATLAB Model Selection Tools

```
%% Question 4      Comparison with struc, arxstruc, selstruc

% Split the data into estimation and validation sets
N = length(Z.y);
Z_est = Z(1:round(2*N/3));        % 2/3 for estimation
Z_val = Z(round(2*N/3)+1:end);   % 1/3 for validation

% Define model structures to test: na = 1:15, nb = 1:15, nk = 1 (fixed)
orders = struc(1:15, 1:15, 1);

% Evaluate ARX models on the estimation and validation sets
V = arxstruc(Z_est, Z_val, orders);
```

```

% Automatically select the best model using FPE (default)
[best_order, best_index] = selstruc(V, 0);

% Display the selected model structure
fprintf('\nBest ARX structure according to selstruc:\n');
fprintf('na = %d, nb = %d, nk = %d\n', best_order);

% Estimate ARX model with the best selected structure
best_model = arx(Z, best_order);

% Print the loss function of this model
fprintf('LossFcn for selected model = %.4f\n', best_model.EstimationInfo.LossFcn
    );
```

Model Selection Using `struc`, `arxstruc`, and `selstruc`

In this step, we compare the results obtained previously with MATLAB's built-in model selection tools. First, the dataset `Z` is split into two subsets: two-thirds for estimation (`Z_est`) and one-third for validation (`Z_val`).

Next, a set of model structures is generated using the `struc` function, where both `na` and `nb` vary from 1 to 15, and the delay `nk` is fixed at 1:

```
orders = struc(1:15, 1:15, 1)
```

The `arxstruc` function evaluates each candidate model by estimating it using `Z_est` and validating it using `Z_val`. The resulting structure `V` contains performance metrics such as the loss function and FPE (Final Prediction Error).

Then, the `selstruc` function is used to automatically select the best model according to the FPE criterion:

$$[\texttt{best\_order}, \texttt{best\_index}] = \texttt{selstruc}(V,\ 0)$$

Finally, the best ARX model is re-estimated using the entire dataset `Z`, and its corresponding loss function is displayed. This provides a data-driven, objective validation of the previously selected model structure. The best ARX structure selected by `selstruc` is given by $n_a = 15$, $n_b = 15$, and $n_k = 1$, with a loss function value of 0.0053. This difference in model order compared to our manual selection is expected: while `selstruc` selects the model that minimizes the loss function, we chose a lower order at the "elbow" of the loss curve, favoring a more parsimonious model that still captures the system's dynamics effectively.

2.2 Parametric identification

Data Partitioning

```
load data_position1.mat

% Compute the derivative of the output signal
y_derivative = lsim(1 - tf('z', Ts)^-1, y);

% Create detrended IDDATA object
Z = detrend(iddata(y_derivative, u, Ts, "Period", 8191));

% Split data into estimation and validation sets
N = length(Z.y);
Z_est = Z(1:round(2*N/3));       % Use 2/3 of data for estimation
Z_val = Z(round(2*N/3)+1:end);   % Use 1/3 of data for validation
```

Identification with Different Model Structures

Here is our code for identification with different model structures.

```
% Define model orders based on previous identification
na = 8;    % Number of past outputs in the model A(q^-1)
nb = 8;    % Number of past inputs in the model B(q^-1)
nk = 1;    % Input delay (number of samples before input affects output)
nc = na;  % Order of C(q^-1) in ARMAX/BJ (noise model)
nd = na;  % Order of D(q^-1) in BJ (noise model)
nf = nb;  % Order of F(q^-1) in OE/BJ (denominator of dynamic model)

% Estimate models using different structures
models = struct();

models.arx   = arx(Z_est, [na nb nk]);
models.iv4   = iv4(Z_est, [na nb nk]);
models.armax = armax(Z_est, [na nb nc nk]);
models.oe    = oe(Z_est, [nb nf nk]);
models.bj    = bj(Z_est, [na nb nc nd nk]);
models.n4sid = n4sid(Z_est, na);
```

Model Output Comparison and Best Model Selection

Here we compare the output of the models with the measured output, as shown in Figure 12.

Unfortunately it is shard to distinguish major differences between plots in order to determine which one fits the best, it has to be computed by MATLAB:

```
model_names = fieldnames(models);
fit_values = zeros(length(model_names),1);

figure('Name', 'Model Comparisons on Validation Data');

for i = 1:length(model_names)
    name = model_names{i};
    model = models.(name);

    subplot(2, 3, i);  % 2 rows, 3 columns
    compare(Z_val, model);
    title(upper(name));

    % Compute fit
    [~, fit, ~] = compare(Z_val, model);
    fit_values(i) = fit;
end
```

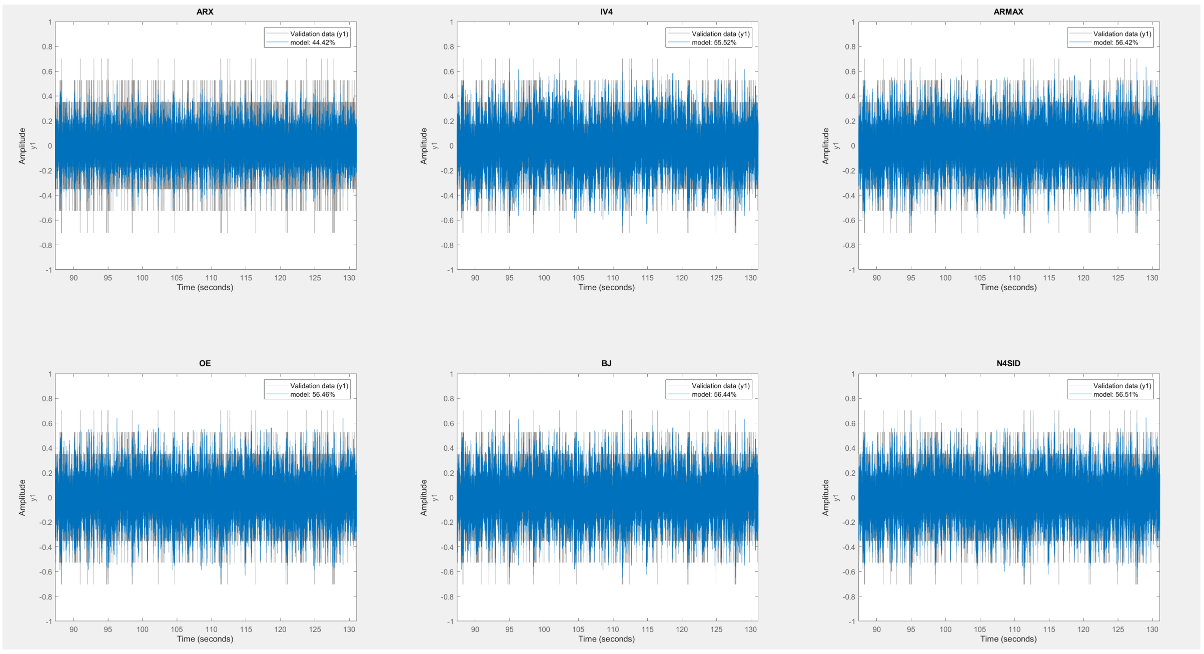


Figure 12: Comparison of the different models with real output

Table 1: Model fit (percentage) on validation data

Model	**Fit (%)**
ARX	44.42
IV4	55.52
ARMAX	56.42
OE	56.46
BJ	56.44
N4SID	56.51

The conclusion is that all models fits approximately the same way excepting ARX model. If we have to keep only one we take N4SID.

Frequency Response Comparison and Model Selection

The Bode diagram of the identified models are shown in Figure 13. Also, the frequency response of the test system is estimated using the validation data and the spectral analysis is implemented. The following is the code snippets of this part:

```
G_spa = spa(Z_est);
figure;
hold on;
bode(models.arx)
bode(models.iv4)
bode(models.armax)
bode(models.oe)
bode(models.bj)
bode(models.n4sid)
bode(G_spa)
hold off;
legend('arx', 'iv4', 'armax', 'oe', 'bj', 'n4sid', 'True model')
```

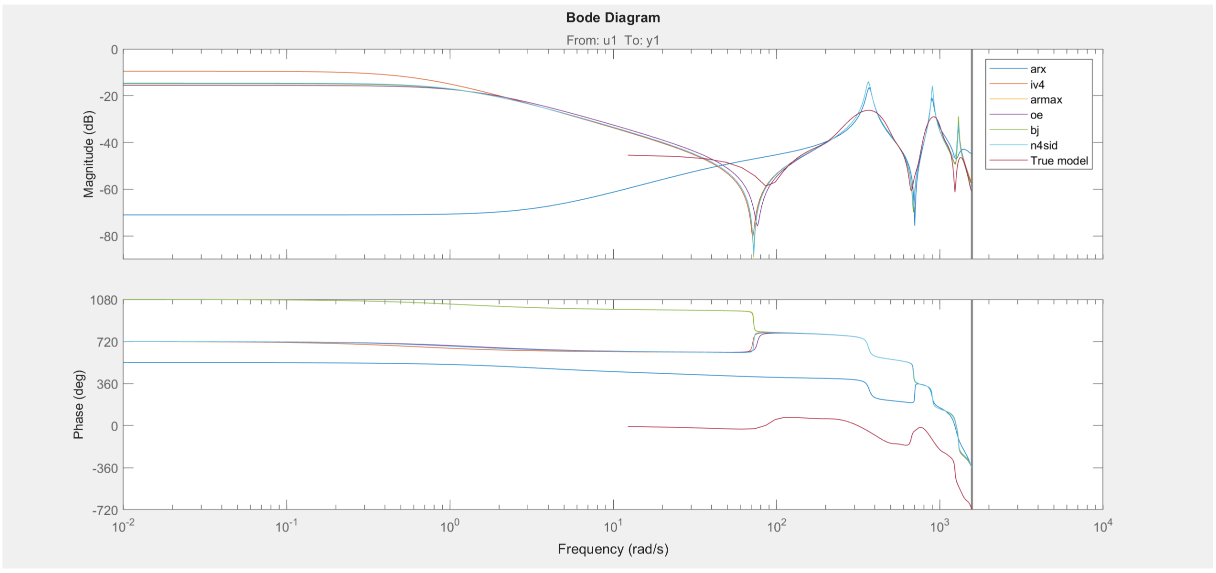


Figure 13: Bode diagram of the identified models compare with the true model

The most intuitive observation is that the ARX model behaves really differently in frequency below around 200 rad/s. Also, in low frequencies different systems converge to similar magnitudes except for the ARX structure. As for the middle frequencies, all models behave similarly and show the resonance modes.

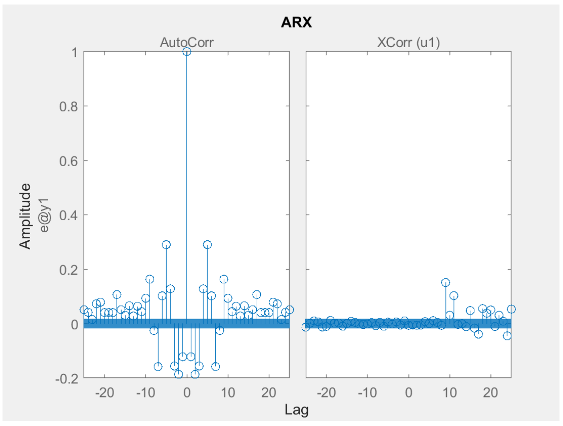
To sum up, based on the Bode diagram, it is quite hard to determine which model performs the best, but we can conclude that the ARX model is not a good fit.

Statistical Validation of Plant and Noise Models

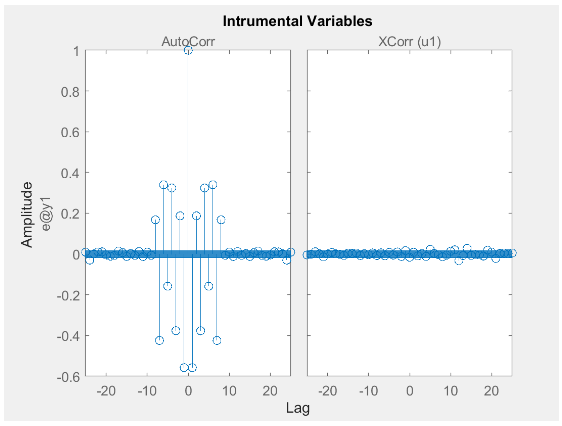
Next, we validate our models by the whiteness test of the residuals ϵ and the cross-correlation of the residuals and the past inputs u.

- Whiteness test: In models that include a noise component (ARX, ARMAX, and Box–Jenkins), we assume the system noise is white noise passed through some filter. If both the plant and noise model parameters are correctly estimated, the residuals should behave like white noise, meaning their autocorrelation is zero for all $h \neq 0$. In practice, we check this using a 99% confidence interval (via MATLAB's `resid` function); the autocorrelation values must lie within these bounds. Because of this, the whiteness test is only meaningful when a noise model is present. The test results are shown on the left side of Figures 14a through 14f.

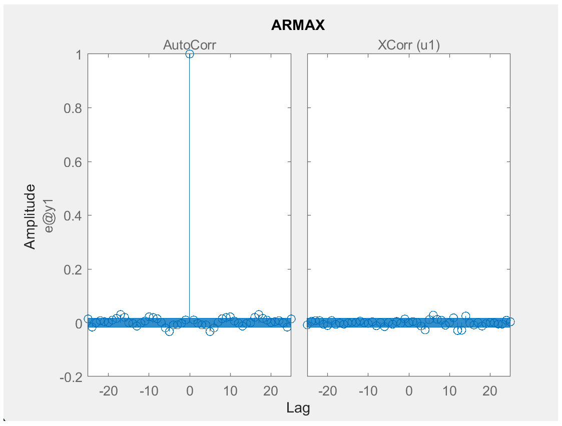
- Cross-correlation test: A correctly identified model will have residuals that contain no information from past inputs. In other words, the residuals should be uncorrelated with earlier input values, so their cross-correlation should be zero. Again, we use a 99% confidence interval (MATLAB's `resid` function) to verify this: all cross-correlation values must fall within those limits. This test applies to validating the plant model in every structure but does not assess the noise model. The results appear on the right side of Figures 14a through 14f.



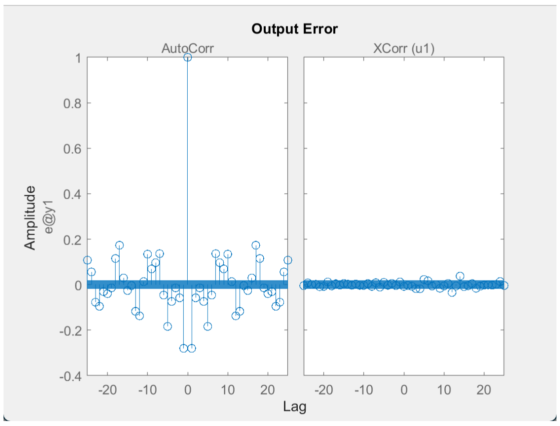
(a) Whiteness test and cross-correlation test of the ARX model



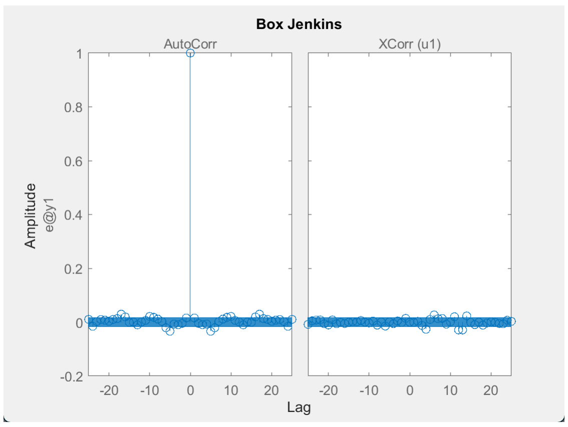
(b) Whiteness test and cross-correlation test of the Instrumental Variables method



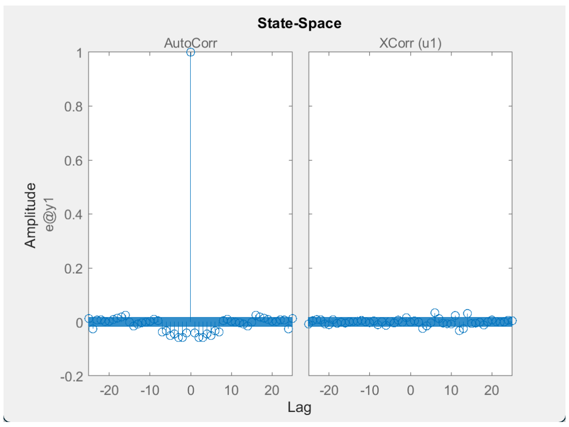
(c) Whiteness test and cross-correlation test of the ARMAX model



(d) Whiteness test and cross-correlation test of the Output Error model



(e) Whiteness test and cross-correlation test of the Box Jenkins model



(f) Whiteness test and cross-correlation test of the State Space model

Figure 14: Whiteness test of the residuals and cross-correlation of the residuals and past inputs of different models.

Based on the results of the statistical tests of the identified models, we discuss about their validation in this section. We observe that none of the models is validated in a strict sense as we have samples outside of the blue confidence interval for every model.

- ARX model (Figure 14a): The whiteness test indicates that the residuals are not white, and the cross-correlation test shows significant correlation between the input and the residuals. Both tests suggest that the ARX structure does not capture the system's dynamics adequately.
- Instrumental variables (Figure 14b): The statistical tests resemble those for the ARX model, except

the cross-correlation appears slightly improved. However, since IV4 also relies on an ARX structure, this improvement is minimal, indicating that using $\frac{1}{A(q^{-1})}$ as the plant model remains poorly suited.

- ARMAX model (Figure 14c): The whiteness test shows less deviation from white noise than with the ARX (and IV4) structures, thanks to including a noise model $\frac{C(q^{-1})}{A(q^{-1})}$. Nevertheless, the cross-correlation test still fails, so even though the noise model is better, the overall structure remains inappropriate.
- Output error (Figure 14d): Since there is no explicit noise model here, the whiteness test is irrelevant. Yet the cross-correlation of residuals with inputs still exceeds the confidence bounds in several samples, so the OE form cannot be validated.
- Box–Jenkins (Figure 14e): The whiteness test shows fewer violations than for the ARX-based methods. Still, the cross-correlation test is not satisfied—several lags lie outside the confidence interval—so the BJ structure fails validation.
- State-space (Figure 14f): Like the OE case, there's no noise model in this form, so the whiteness test does not apply. However, the residuals' cross-correlation remains significant at some lags, so this state-space realization also cannot be validated.

```
figure;
resid(Z_val, models.arx); title('ARX');
figure;
resid(Z_val, models.iv4); title('Intrumental Variables');
figure;
resid(Z_val, models.armax); title('ARMAX');
figure;
resid(Z_val, models.oe); title('Output Error');
figure;
resid(Z_val, models.bj); title('Box Jenkins');
figure;
resid(Z_val, models.n4sid); title('State-Space');
```

Add Integrator

To further validate the models, we add an integrator to all of the parametric models we have obtained and generate a new validation dataset using the second half of the position data y. We compare the output of the identified models using the `compare` function and compare the frequency response just like n the previous sections. The results are shown in Figure 15 and 16.

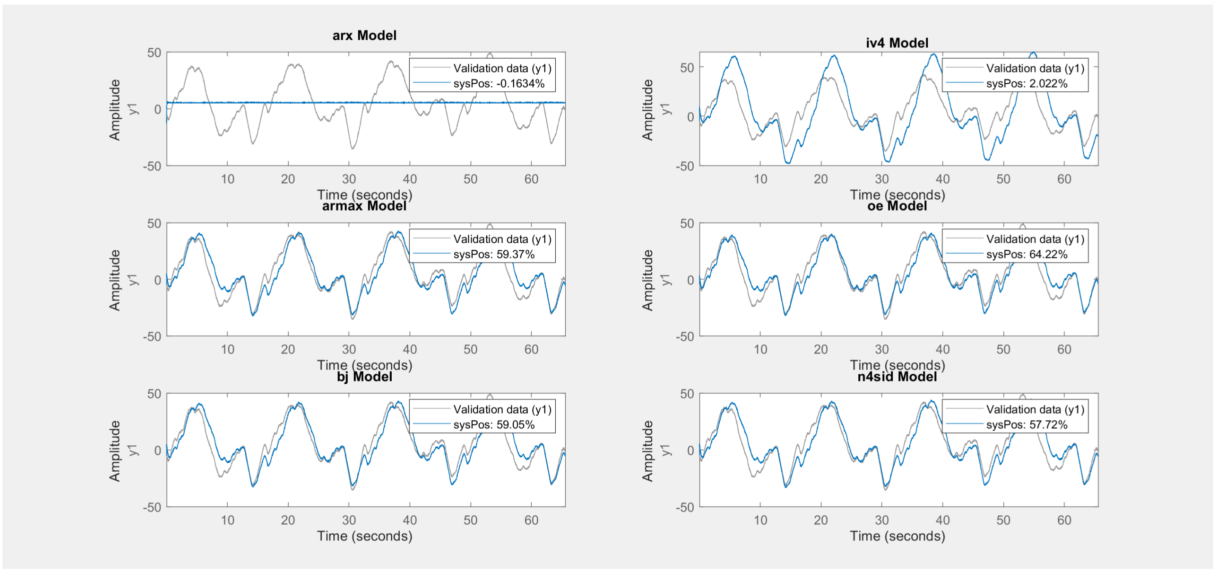


Figure 15: Comparison of the different models with real output. (Position data)

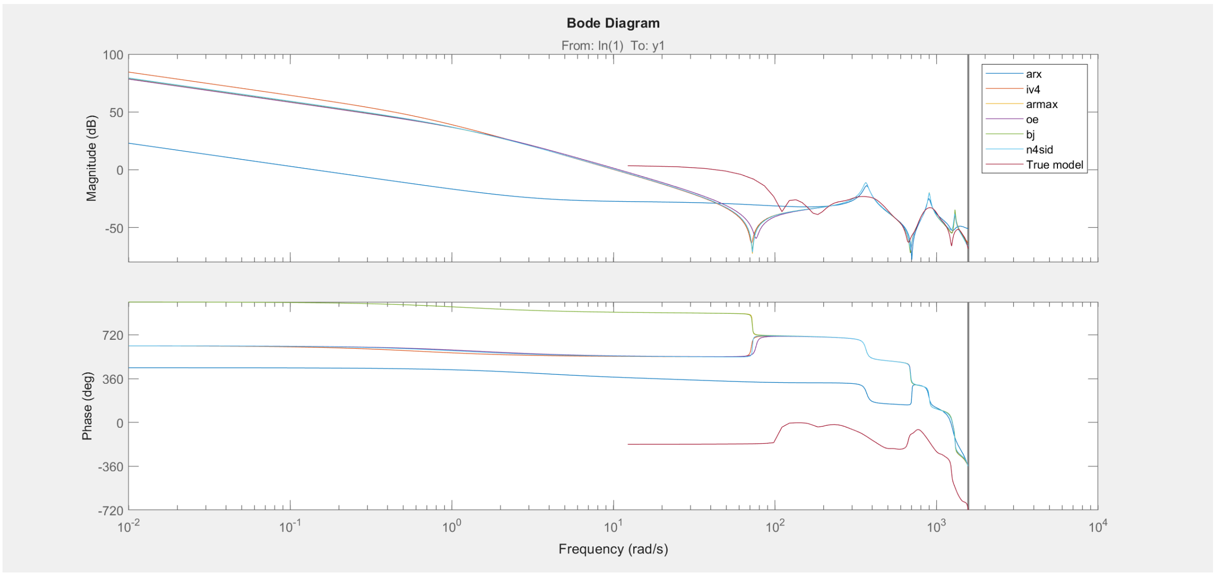


Figure 16: Bode diagram of the identified models compare with the true model

In Figure 15 we can see that the output error model fits the best, and in Figure 16 we can see that the results are quite similar to the previous section that, ARX model is not a good fit. Also, after adding the integrator, all the models did not fully show the resonance modes, which is invalid.

```
% Reload raw position data (y) and input (u)
load data_position1.mat   % brings in u, y, Ts

% Build iddata for the SECOND HALF of the raw position signal
Npos = length(y);
idx2 = round(Npos/2)+1 : Npos;
y_second = y(idx2);
u_second = u(idx2);
posValData = iddata(y_second, u_second, Ts);

% Build the discretetime integrator 1/(1 - z^{-1}) as an idtf
integrator = idtf(1, [1 -1], Ts);

% Loop over each derivativedomain model, append integrator, and compare
figure('Name','PositionDomain Validation','Units','normalized','Position',[.1
    .1 .8 .7]);
for k = 1:nModels
    sysDer = models.(modelNames{k});        % derivativedomain model
    sysPos = sysDer * integrator;           % now maps u     position

    subplot(3,2,k);
    compare(posValData, sysPos);
    title(sprintf('%s Model', modelNames{k}));
end
set(gcf,'Renderer','painters');

% Nonparametric FRF of  u position  using SPA on posValData
G_pos_spa = spa(posValData);

% Overlay Bode plots: six integrated models vs. nonparametric FRF
figure('Name','Bode: Parametric vs. Nonparametric (Position)','Units','
    normalized','Position',[.1 .1 .8 .7]);
hold on;
for k = 1:nModels
    sysDer = models.(modelNames{k});
    sysPos = sysDer * integrator;
    bode(sysPos);
```

```
end
bode(G_pos_spa);
hold off;
legend('arx', 'iv4', 'armax', 'oe', 'bj', 'n4sid', 'True model')
```

Final Model Selection Based on All Validation Results

In the initial time-domain validation, none of the models predict the output particularly well. Prediction accuracies range between 44% and 57%, with the ARX model performing the worst (44.42%) and the state-space model performing the best (56.51%). In the frequency-domain validation, the ARX structure deviates notably from the true frequency response at lower frequencies. Finally, the statistical tests (whiteness and cross-correlation) reveal that the residuals for all models are far from white. This indicates that the noise-model assumptions in ARX and ARMAX are inappropriate, and even the output-error structure confirms that the system noise is not white. As a result, none of the models passes all statistical tests. Depending on the intended use of the identified model, one might choose ARMAX or Box–Jenkins as the least unsuitable options.

There are several reasons why these linear methods cannot perfectly capture the system. For example, the actual system may be nonlinear, so a linear model will always be an approximation. Additionally, the measured signals could contain problematic noise—perhaps with a nonzero mean or time-dependent behavior—that these models do not account for.