Team #01

Capătă Andrei-Bogdan
Rizoiu Tudor
Căstrase Sergiu-Gabriel

Nonlinear ARX Identification

System Identification Project - Part 2

Table of Contents

Introduction / Problem Statement

Developing a black-box model for an unknown dynamic system characterized by a single input and a single output, with dynamics that are not larger than order three.

How can we do it?

Using **Nonlinear ARX Identification**

What is Nonlinear ARX Identification ?

Nonlinear ARX Identification = a process used to develop a model for a system using a nonlinear ARX model.

A nonlinear ARX model extends the linear ARX models to the nonlinear case. The structure of these models enables us to model complex nonlinear behavior using flexible nonlinear functions.

Approximator Structure

- **building 'PHI_id'** - regressors matrix, for the identification dataset;
- **finding 'theta'** - the coefficients of the unknown function, by using matrix left division method between 'PHI id' and the output of the identification data set 'Y id';
- **building 'PHI_val'** - regressors matrix, for the validation dataset;

Approximator Structure

- **building the one-step-ahead prediction vector 'yhat_id'** - the approximated function by multiplying 'PHI_id' and 'theta';

- **building the one-step-ahead prediction vector 'yhat_val'** - the approximated function by multiplying 'PHI_val' and 'theta';

- **building the simulation vectors 'ysim_id' and 'ysim_val'** using the previously simulated outputs to construct the approximation and 'u_id' and 'u_val' respectively;

Key Features

Algorithm for building ‘t’ matrix

Cartesian product of na+nb sets, each set containing numbers from 0 to m.

The dimensions of the matrix are na+nb columns and (m+1)^(na+nb) rows.

Based on the index of the current column, we can determine that each number from 0 to m repeats at a certain number of steps. The formula for computing this step is (m+1)^(*column index*-1).

Key Features

Algorithm for building ‘PHI’

Each value y(k-1) ... y(k-na) and u(k-1) ... u(k-nb) has its own column in the ‘t’ matrix, thus creating all the possible combinations of powers.

Algorithm for building ‘y_sim’

Creating a line by line matrix similar to ‘PHI’ using the previously simulated outputs, ysim(k-1) ... ysim(k-na) and u(k-1) ... u(k-nb). On the ysim(k) position -> *constructed line* * theta.

Tuning Results - MSE

MSE measures how well a prediction or estimation method performs by calculating the average of the squared differences between predicted values and actual observations.

Lower MSE values indicate better performance.

The plot in *Fig. 1* was made so that we could clearly see how the MSE evolves in comparison with na, nb and m.

Graphic representation of MSE

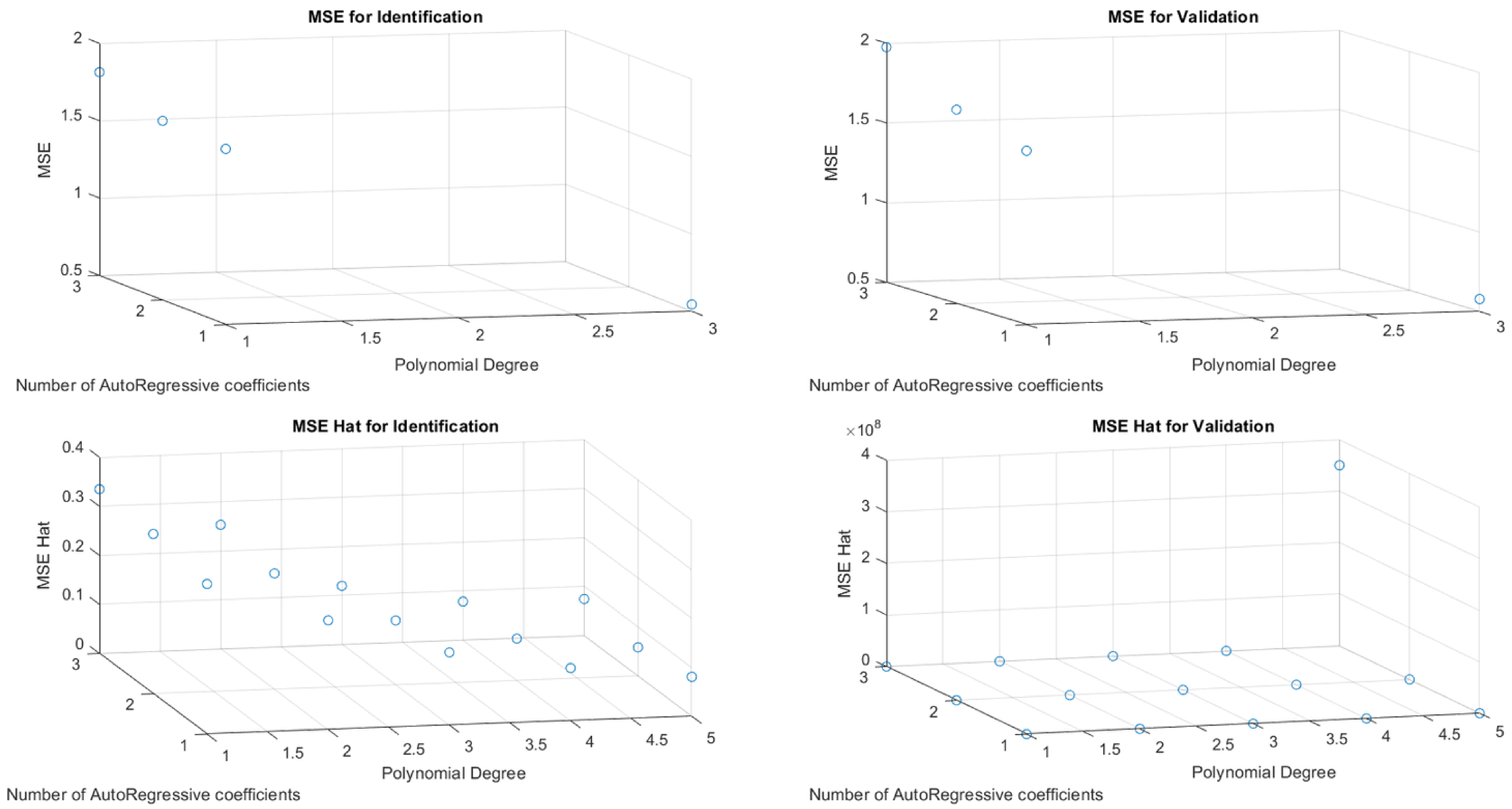

Fig. 1

Representative plots

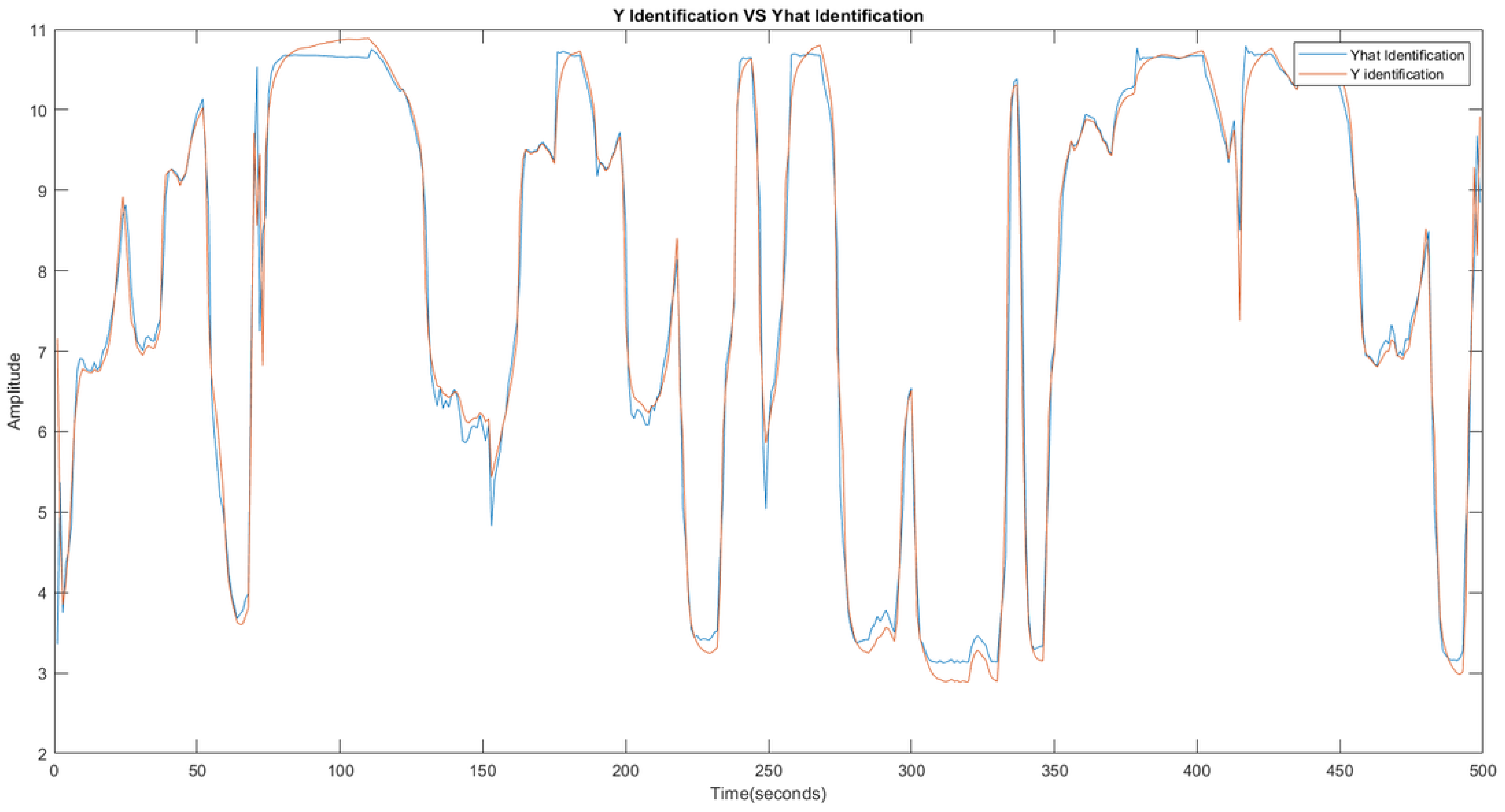

Fig. 2

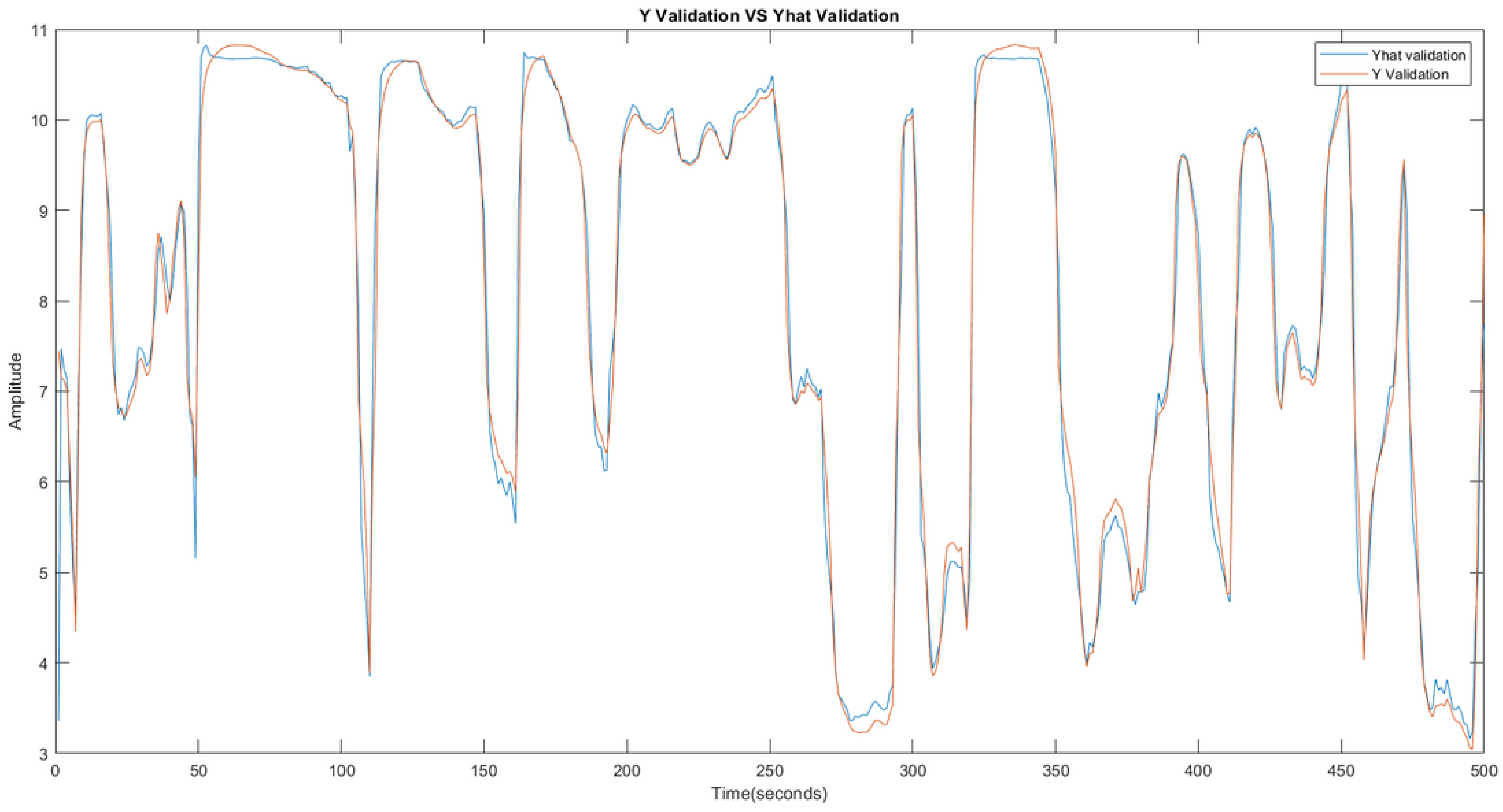

Fig. 3

One-step-ahead prediction error and the simulation error

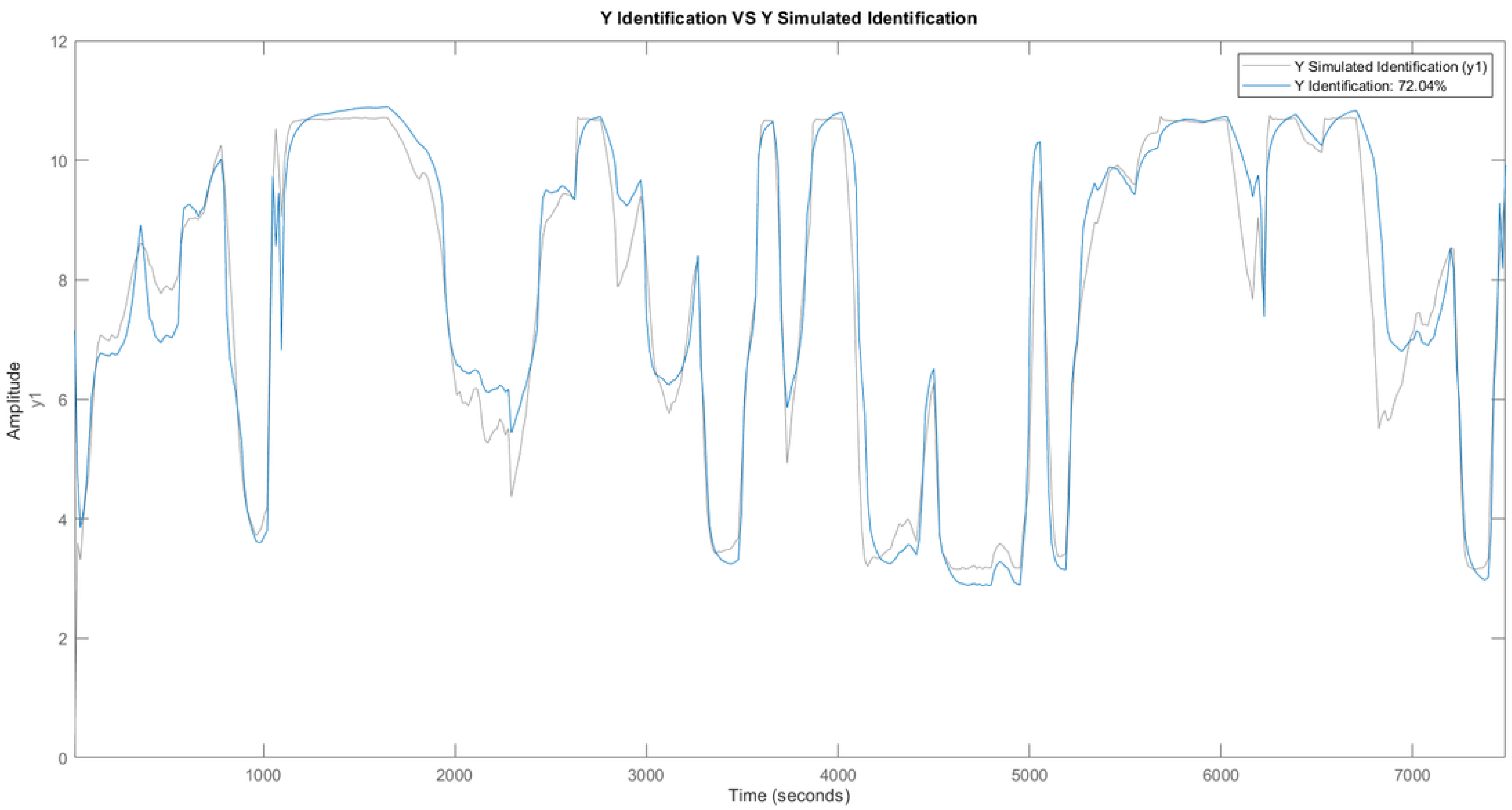

Fig. 4

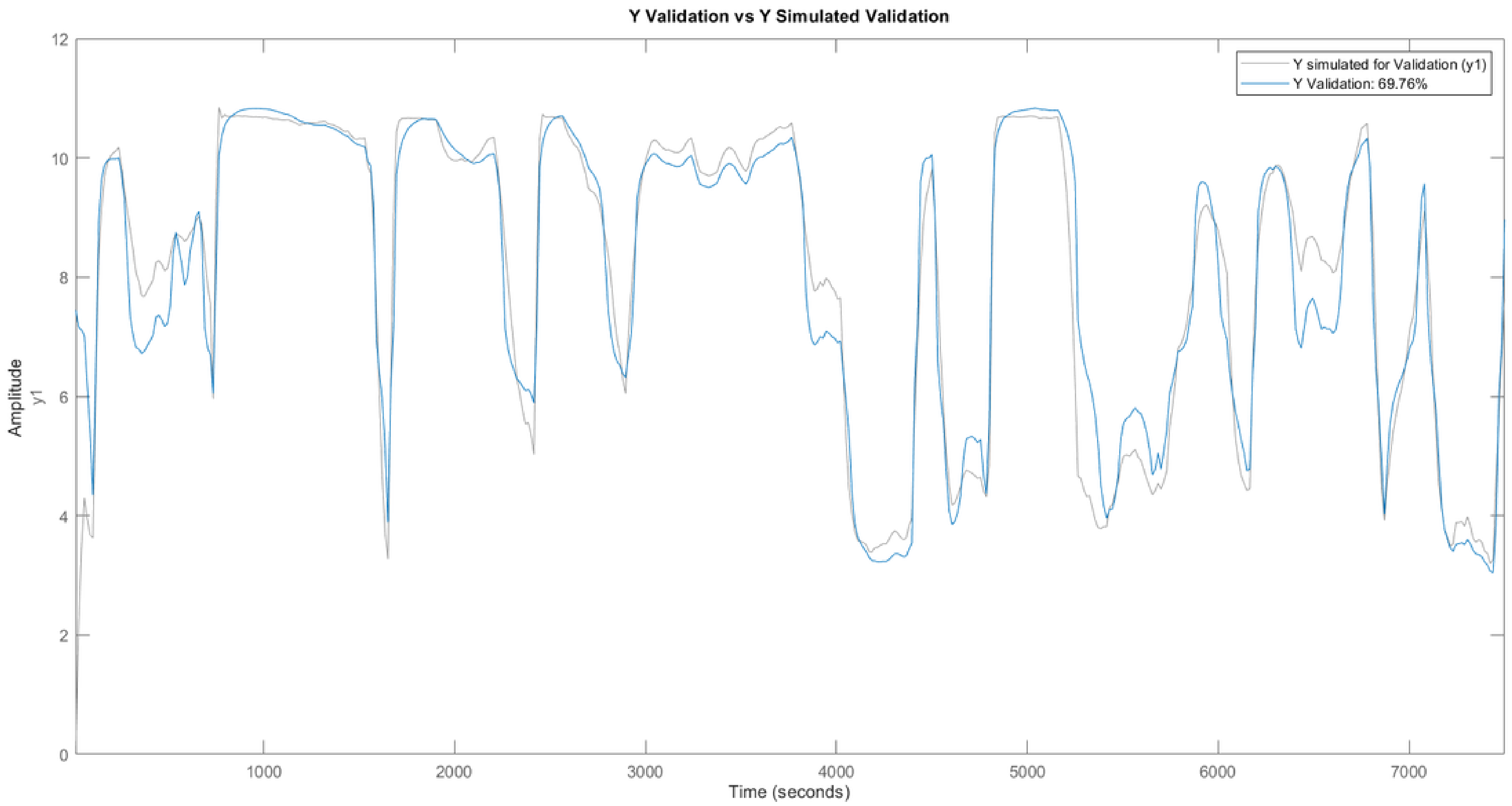

Fig. 5

Discussion of the results

We chose:
na (AutoRegressive coefficient) = 1;
nb (Moving Average coefficient) = 1
m (Polynomial degree) = 3

Why?
Because the Mean Squared Error has the minimum value, using this values.

Code Snippets

Algorithm for ‘PHI’ for the identification data set

```
%% Build the phi matrix for identification
N = length(y_id);
phi = ones(N, length(t));
a = length(t);

for k = 1:N
    for j = 1:na
        for i = 1:a
            if (k - j) <= 0
                phi(k, i) = 0;
            else
                phi(k, i) = phi(k, i) * ((y_id(k - j))^t(i, j)) * (u_id(k - j)^t(i, j + na));
            end
        end
    end
end

phi(1, :) = 1;
theta = phi \ y_id;

y_hat_id = phi * theta;
```

Algorithm for ‘PHI’ for the validation data set

```
%% Build the phi matrix for validation
Nval = length(y_val);
phiv = ones(Nval, a);

for k = 1:Nval
    for j = 1:na
        for i = 1:a
            if (k - j) <= 0
                phiv(k, i) = phiv(k, i);
            else
                phiv(k, i) = phiv(k, i) * ((y_val(k - j))^t(i, j)) * (u_val(k - j)^t(i, j + na));
            end
        end
    end
end
phiv(1,:) = 1;
y_hat_val = phiv * theta;
```

Algorithm for building the simulation vector ‘ysim_id’

```
%% Simulate the identified model for identification data
ysimid = zeros(N, 1);
for k = 1:N
    d = ones(1, length(t));
    for j = 1:na
        for i = 1:a
            if (k - j) <= 0
                d(i) = ysimid(1);
            else
                d(i) = d(i) * ((ysimid(k - j))^t(i, j)) * (u_id(k - j)^t(i, j + na));
            end
        end
    end
    ysimid(k) = d * theta;
    clear d;
end
```

Algorithm for building the simulation vector ‘ysim_val’

```
%% Simulate the identified model for validation data
ysimval = zeros(Nval, 1);
for k = 1:Nval
    d = ones(1, length(t));
    for j = 1:na
        for i = 1:a
            if (k - j) <= 0
                d(i) = ysimval(1);
            else
                d(i) = d(i) * ((ysimval(k - j))^t(i, j)) * (u_val(k - j)^t(i, j + na));
            end
        end
    end
    ysimval(k) = d * theta;
    clear d;
end
```

Algorithm for building the 't' matrix

```
% Generate the t matrix representing all possible combinations of coefficients
t = zeros((m + 1)^(na + nb), na + nb);

for c = 1:(na + nb)
    line = 1;
    step = (m + 1)^(c - 1);
    for a = 1:(m + 1)^(na + nb - 1) / step
        for k = 0:m
            for b = line:(line + step)
                t(b, c) = k;
            end
            line = line + step;
        end
    end
end

t(end, :) = [];

j = (m + 1)^(na + nb);

% Remove rows from t where the sum of coefficients exceeds the maximum degree
while (j >= 1)
    if (sum(t(j, :)) > m)
        t(j, :) = [];
        j = j - 1;
    else
        j = j - 1;
    end
end
```

Thank you for your attention!

Please address any questions regarding our presentation.