System Identification: Nonlinear ARX identification

Project made by: Brezae Tudor, Diaconu Paul and Dutkas Alexis

Index: 15

Mission Statement

- The Second Project oversees a given dataset, measured on an unknown dynamic system with one input and one output. We developed a black-box model for this system, using a polynomial, nonlinear ARX model, then validating it on a second data set measured on the same system.
- In order to achieve that goal, the team had to make use of various mathematical, computational and algorithmic methods which will be presented more explicitly in the following slides.

Constructing the Phi Matrix

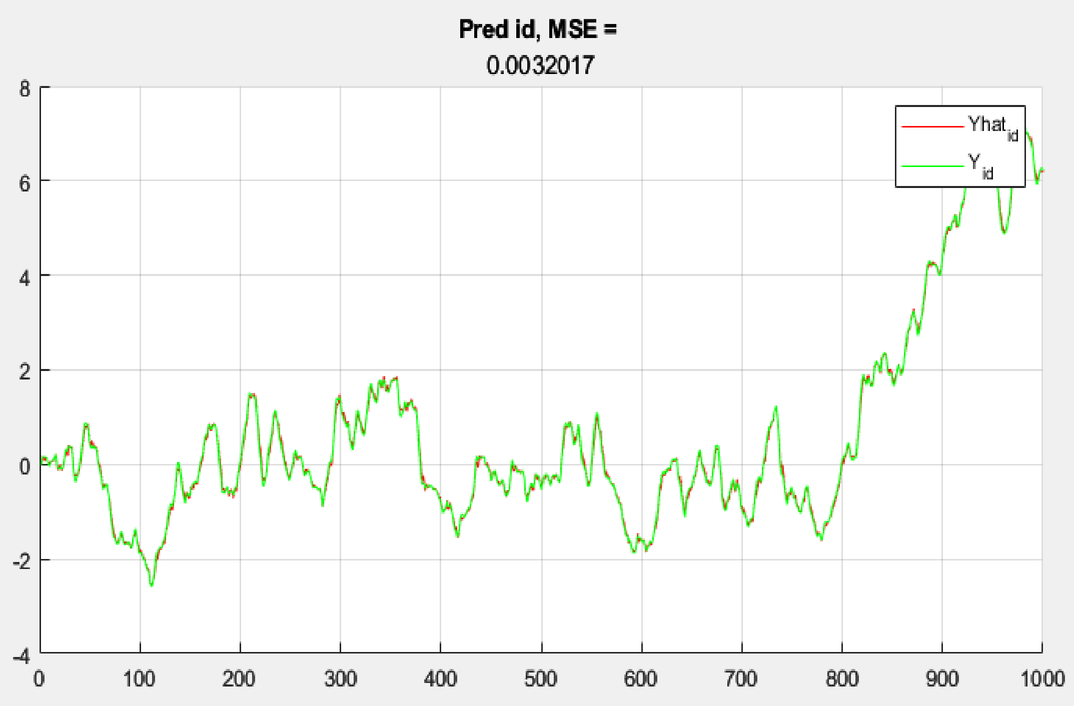
- To start the project off, we first had to construct the polynomial matrix Phi of order '*m*' and '*na + nb*' terms. For us to achieve that goal, we created two functions:
- The first function's role is to create a matrix of all possible exponent combinations for a certain polynomial.
- The subsequent function was used to construct the Phi matrix, line by line, by raising the terms to their corresponding exponents and multiplying them.

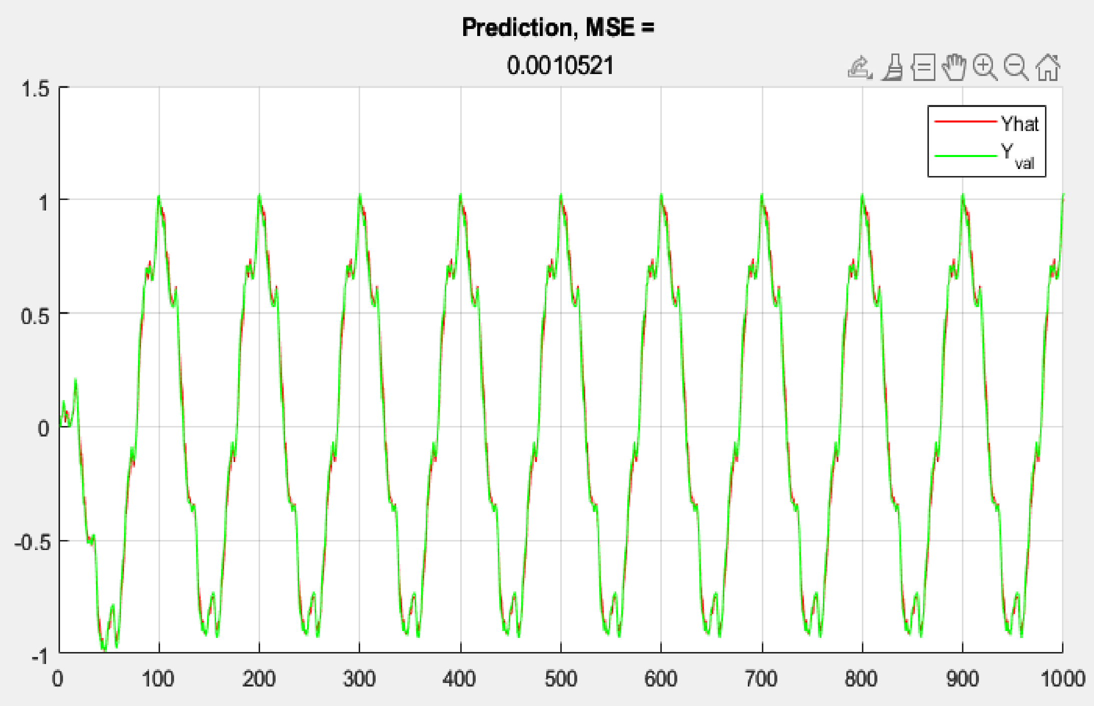
Tuning results

- After testing different values for na, nb and the degree m, we came to the conclusion that the best parameters are *na* = 1, *nb* = 3 and m = 5.
- For bigger *na*, *nb* and m the MSEs were very low (10e-8), resulting in an overfitting function.

Representative plots

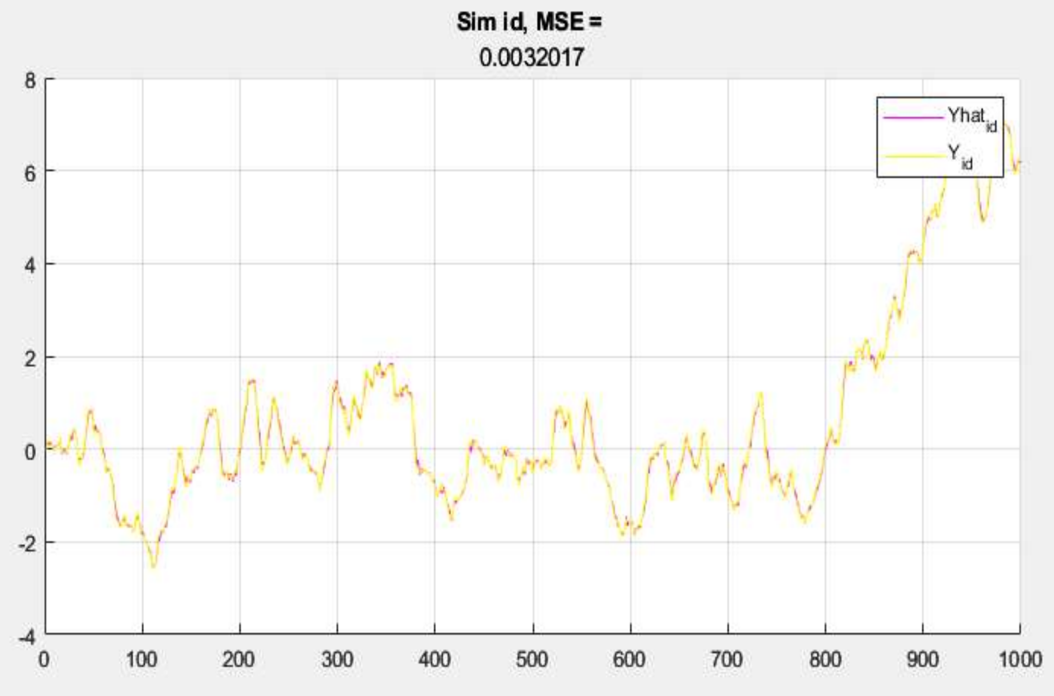
The representative plot for one-step-ahead prediction:

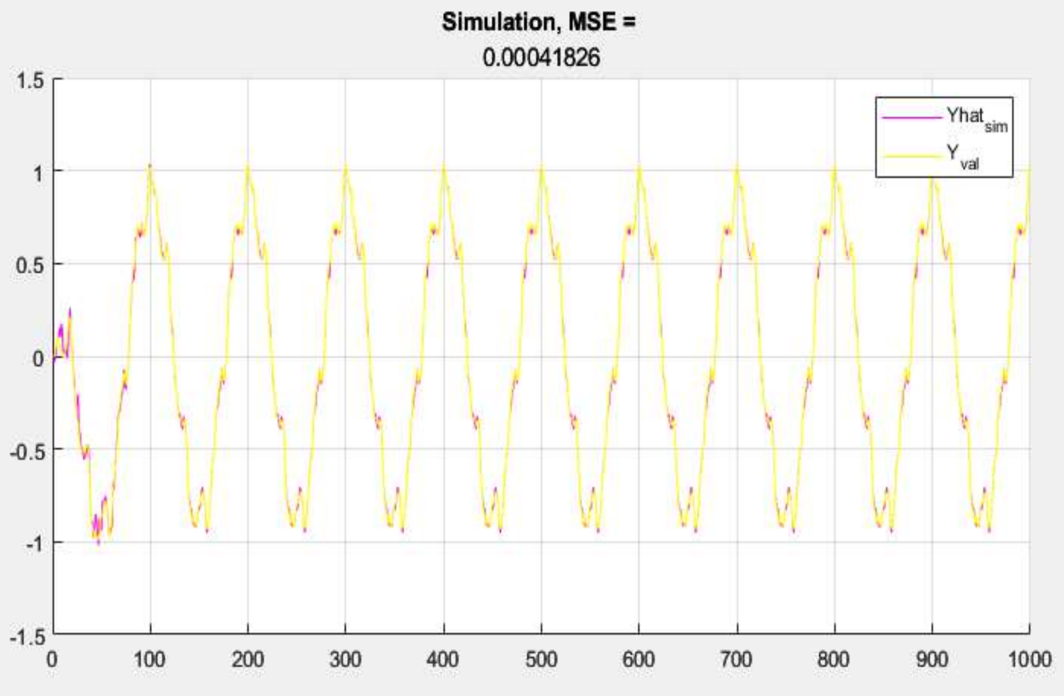




Representative plots

The representative plot for simulation:





To Conclude

- Mission Statement
- Constructing the Phi Matrix
- Tuning results
- Representative plots for *na* = 1, *nb* = 3, m = 5
- ***To Conclude***
- Code appendix

A final conclusion:

After choosing our optimal *na*, *nb* and degree m it led us to a good solution such that the model is not too specific for the training data and will not fail on generalizing to new data.

As we saw, the one-step-ahead prediction and simulation MSEs are both low implying an adequate response to new data.

Code appendix

```
clear all;
close all;
clc;

data = load('iddata-15.mat');

% Taking the data for the validation and identification
uId = data.id.InputData;
yId = data.id.OutputData;

uVal = data.val.InputData;
yVal = data.val.OutputData;

% Plotting the identification and validation data
subplot 221
plot(uId);grid;shg
title("Identification Input Data")
subplot 222
plot(yId);grid;shg
title("Identification Output Data")

subplot 223
plot(uVal);grid;shg
title("Validation Input Data")
subplot 224
plot(yVal);grid;shg
title("Validation Output Data")
```

```
% Prediction and Simulation for Identification
phi = [];
for i = 1:length(uId)
    phia = [];
    phib = [];
    for j = 1:na
        if i-j>0
            phia = [phia yId(i-j)];
        else
            phia = [phia 0];
        end
    end

    for j = 1:nb
        if i-j>0
            phib = [phib uId(i-j)];
        else
            phib = [phib 0];
        end
    end
    phi = [phi; phia phib];
end

Phi = [];
for I = 1:length(phi)
    Phi = [Phi; Poly(phi(I,:),m)]; % generating the phi matrix
end
```

Code appendix

```
% Prediction for validation
theta = Phi \ yId;
yhatId = Phi* theta;

phipred = [];
for i = 1:length(uVal)
    phia = [];
    phib = [];
    for j = 1:na
        if i-j>0
            phia = [phia yVal(i-j)];
        else
            phia = [phia 0];
        end
    end

    for j = 1:nb
        if i-j>0
            phib = [phib uVal(i-j)];
        else
            phib = [phib 0];
        end
    end
    phipred = [phipred; phia phib];
end

Phi = [];
for I = 1:length(phi)
    Phi = [Phi; Poly(phipred(I,:),m)]; % generating the phi matrix
end
yhat = Phi * theta;
```

```
% Simulation for validation
ytilde = [];
phisim = [];
for i = 1:length(uVal)
    phia = [];
    phib = [];
    for j = 1:na
        if i-j>0
            phia = [phia -ytilde(i-j)];
        else
            phia = [phia 0];
        end
    end
    for j = 1:nb
        if i-j>0
            phib = [phib uVal(i-j)];
        else
            phib = [phib 0];
        end
    end
    phisim = [phisim; phia phib];
    ytilde = [ytilde Phi*theta];
end

Phi = [];
for I = 1:length(phi)
    Phi = [Phi; Poly(phisim(I,:),m)]; % generating the phi matrix
end
thetaH = linsolve(Phi,yVal);
yHat2 = Phi * thetaH;
```

Code appendix

```
% Calculating MSEs for every case
MSE = 0;
mse_id = 0;
for i = 1:length(yhatId)
    MSE = MSE + (yhatId(i)-yId(i)).^2;
end
mse_id = 1/length(yhatId) * MSE;

MSE = 0;
mse_pred = 0;
for i = 1:length(yhat)
    MSE = MSE + (yhat(i)-yVal(i)).^2;
end
mse_pred = 1/length(yhat) * MSE;

MSE2 = 0;
for i = 1:length(yHat2)
    MSE2 = MSE2 + (yHat2(i)-yVal(i)).^2;
end
mse_sim = 1/length(yHat2) * MSE2;

 % Subplots
 % Prediction identification
 figure
 subplot 221
 hold on
 plot(yhatId,'r')
 plot(yId,'g')
 grid;shg
 hold off
 legend('Yhat_{id}','Y_{id}')
 title('Pred id, MSE = ', mse_id)
```

```
% Prediction Validation
subplot 222
hold on
plot(yhat,'r')
plot(yVal,'g')
hold off
grid;shg
legend('Yhat','Y_{val}')
title('Prediction, MSE = ', mse_pred)

% Simulation Identification
subplot 223
hold on
plot(yhatId,'m')
plot(yId,'y')
grid;shg
hold off
legend('Yhat_{id}','Y_{id}')
title('Sim id, MSE = ', mse_id)

% Simulation Validation
subplot 224
hold on
plot(yHat2,'m')
plot(yVal,'y')
hold off
grid;shg
legend('Yhat_{sim}','Y_{val}')
title('Simulation, MSE = ', mse_sim)
```

Code appendix

```
% All the functions that we need in order to run
function [phipol] = Poly(l1,m)
phipol = [];
L = [];
exp = [];
for i=1:m
    % With the help of the recursive function, we construct the exponent
    % matrix which is comprised of concatenated matrices whose line sum is
    % equal to i
    exp = [exp; generateExponents(length(l1),i,[])];
end
i = 1;
% We create the polynomial matrix which contains the terms of the
% polynomials with order <= m
while(i<=length(exp(:,1)))
    Prod = 1;
    for j = 1:length(l1)
        Prod = Prod * l1(j)^exp(i,j);
    end
    phipol = [phipol Prod];
    i = i+1;
end
end
```

```
% We use a function to construct a matrix which contains all possible
% exponents from a multinomial with n terms and order m
function exponents = generateExponents(n, rest, currExp)
% We give the base case condition, where we introduce the rest
% (A value which is decremented from m order), into a new line of exponents
if n == 1
    exponents = [currExp, rest];
else
    exponents = [];
    for i = 0:rest
        % We generate through recursivity, all possible combinations of
        % exponents by continuously decrementing the number of term
        % till we get to the base case and also decrementing the rest
        % (rest-i + i = rest)
        subExp = generateExponents(n-1,rest-i,[currExp i]);
        % We construct a new line
        exponents = [exponents; subExp];
    end
end
end
```