Nonlinear ARX Identification

SYSTEM IDENTIFICATION 2023-2024

Project part 2

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

Petcuț Adrian-Axente

Ilea Cosmin-Ionuț

Szakacs Armand-Antonio

Group Id: 11

Table of contents:

Problem statement

Given a set of data where outputs are measured on an unknown dynamic system with one input and one output, and the dynamics may be nonlinear while the output may be affected by noise. It is given two data sets: one to identify the model and another to validate it.

- We had to develop a black-box model for the system using a polynomial, nonlinear ARX model.
- Compute one step ahead prediction $\hat{y}$ and simulation $\tilde{y}$ for the output

Approximator Structure

- $\hat{y}(k) = p\big(y(k-1), \dots, y(k-na), u(k-nk), \dots, u(k-nk-nb+1)\big)$
  $= p\big(d(k)\big)$
- $d(k) = [y(k-1), \dots, y(k-na), u(k-nk), \dots, u(k-nk-nb+1)]^{\mathrm{T}}$
- Example:

  For $na = nb = 1, m = 2$ the polynomial expansion will be:
  - $\hat{y}(k) = \emptyset_0 + \emptyset_1 y(k-1) + \emptyset_2 u(k-1) + \emptyset_3 y(k-1)^2 + \emptyset_4 u(k-1)^2 + \emptyset_5 u(k-1) y(k-1)$

- d - delay vector
- p - polynomial of degree m in d

Finding the parameters

- The way that we generated the polynomial:
  1. Create a vector $power$ the same length l as the vector d, $\prod_1^l d(i)^{power(i)}$ this would correspond to an element in the sum $\hat{y}$
  2. Generate all the possible combinations of vector $power$, with the sum of elements less or equal than m, defining our polynomial $\hat{y}$
- Example for $na = nb = 1, m = 2$

the delay vector: $d(k) = [y(k-1), u(k-1)]$

$$d(k)^{power} = [y(k-1), u(k-1)].^{[2,0]} = y(k-1)^2$$

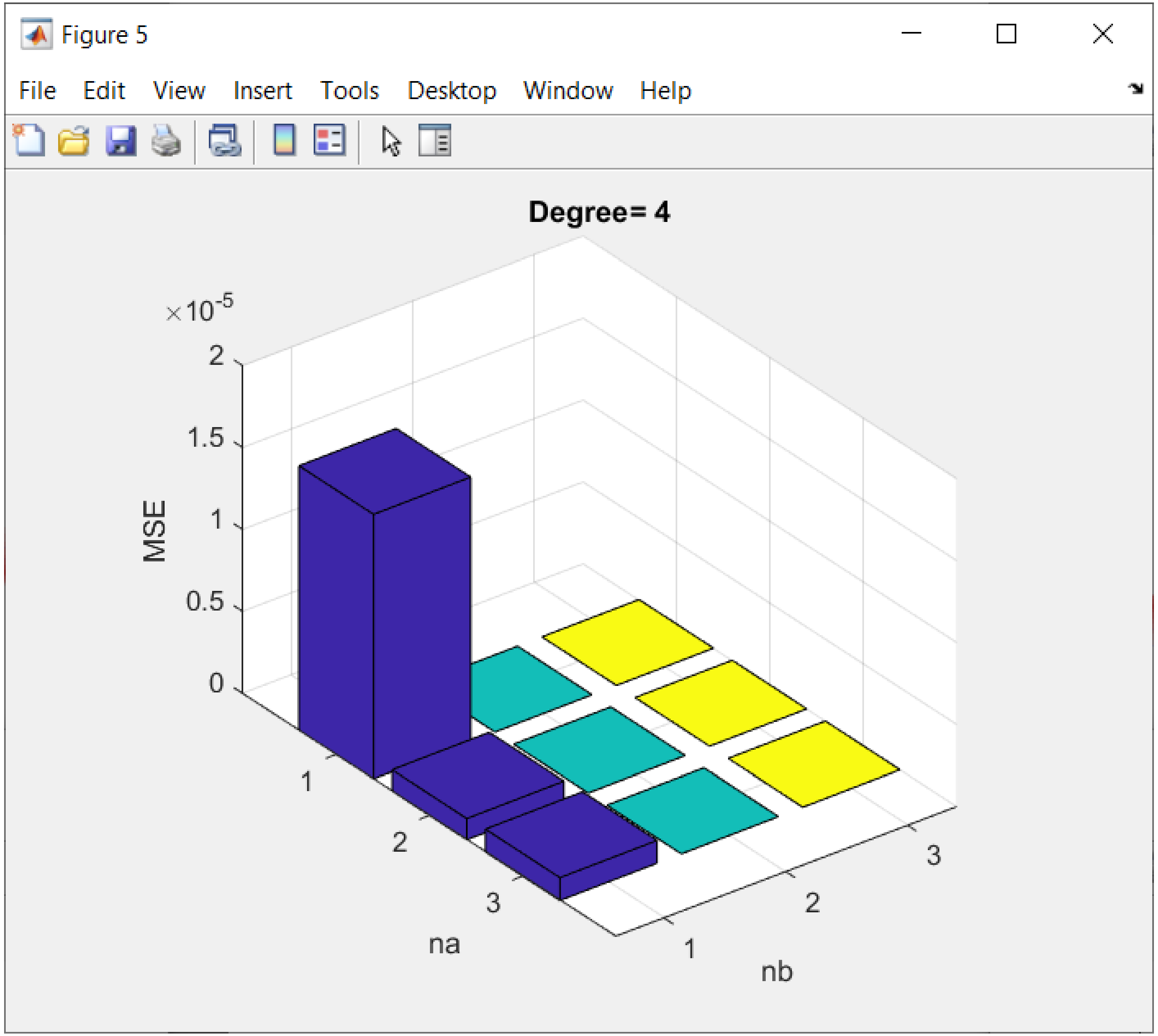
$$d(k)^{power} = [y(k-1), u(k-1)].^{[1,1]} = y(k-1)u(k-1)$$

Key features

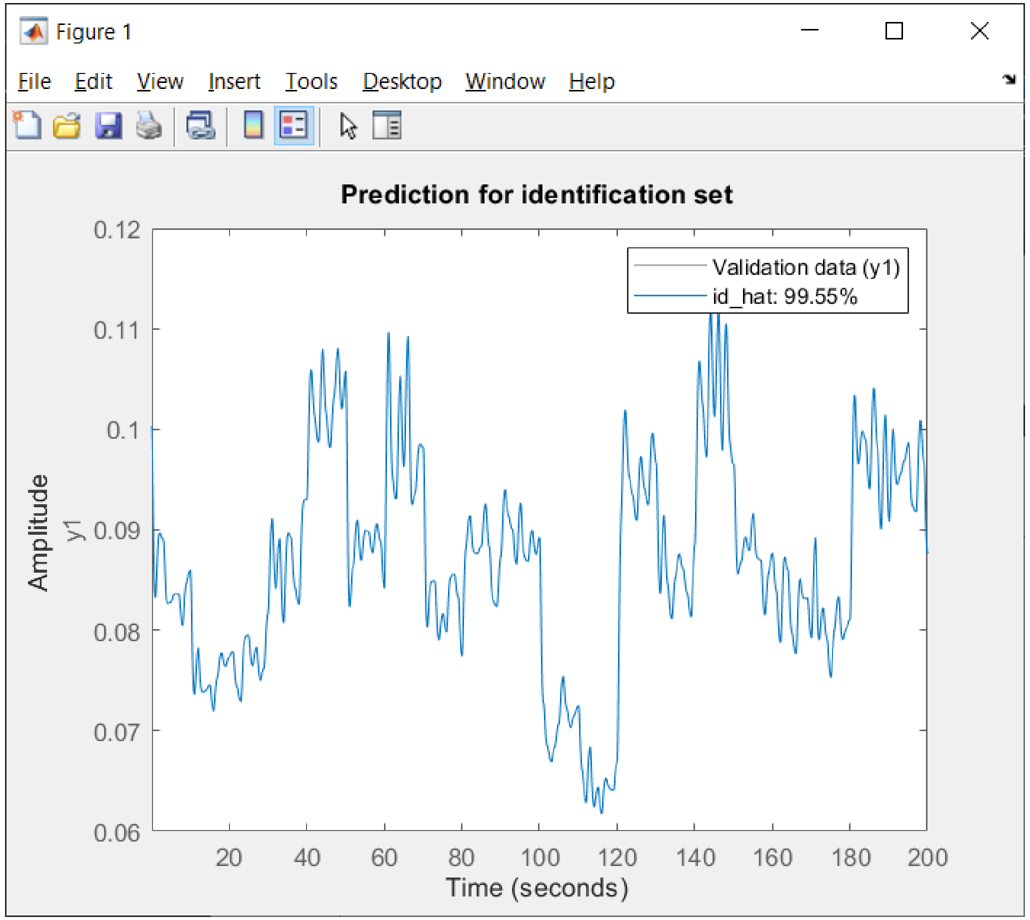
- We automated the process of choosing the best combination of na, nb, m, based on the lowest Mean Squared Error
- Transformed the outputs to Input-output data structures (iddata) to use the *compare* function to see the Normalized Root Mean Square Error
- Our algorithm works for any given data set

Tuning Results

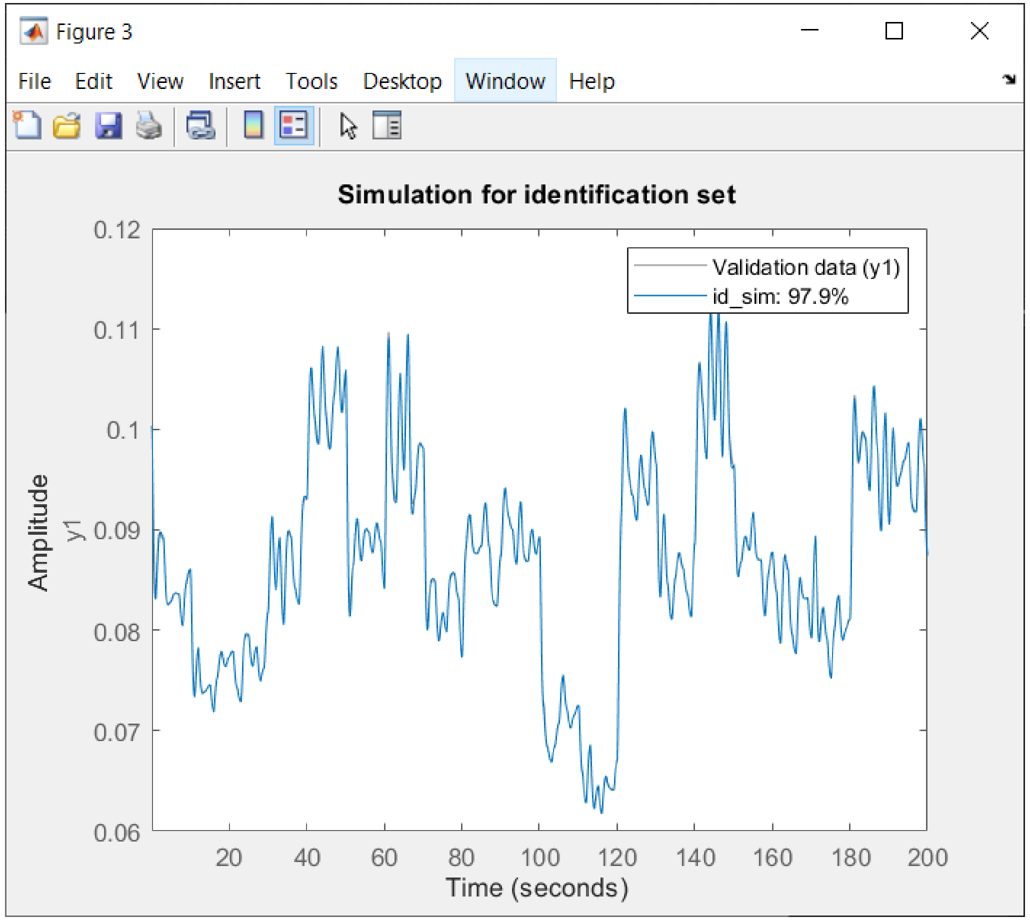
- Best orders and degree:
  - $na = 2$
  - $nb = 1$
  - $m = 4$



Plots for the optimal value
Identification data

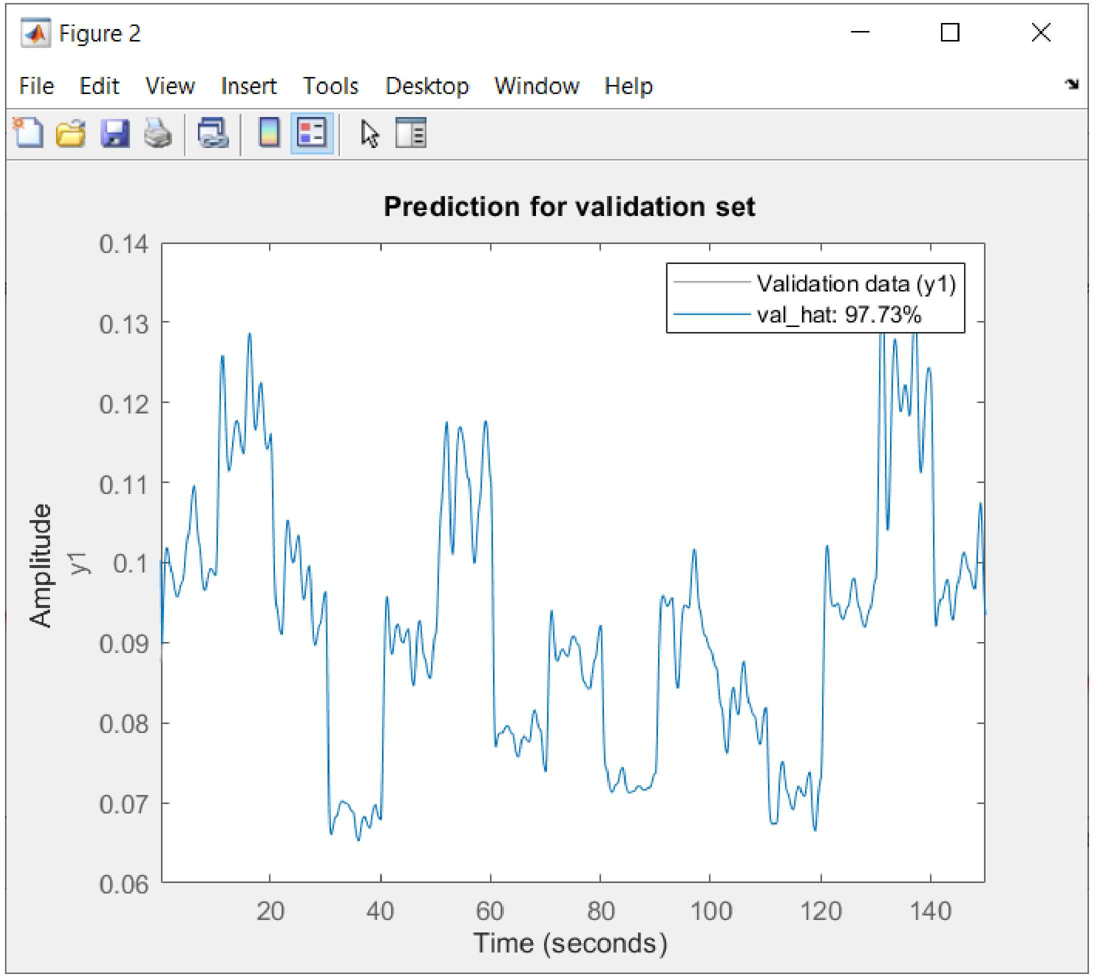


MSE = 2.187e-09

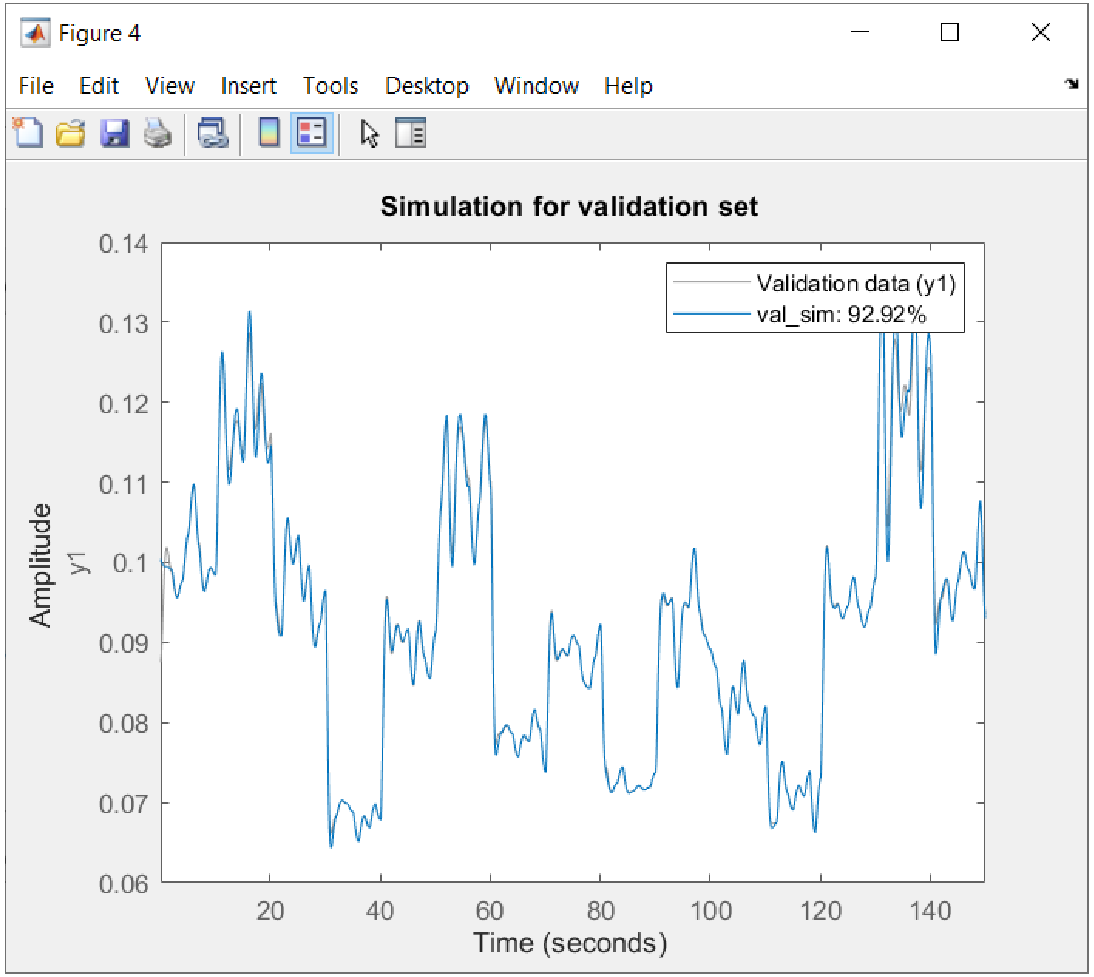


MSE = 4.786e-08

Validation data



MSE = 1.332e-07



MSE = 1.291e-06

Conclusions

- Using the NARX method we can identify easier black box systems concluding that NARX is a great method of finding the best model for the data set given
- We managed to find the model that is the closest to the real one
- We found an efficient algorithm to generate the polynomial p

List of code

```
function [MSE] = compMSE(approx,
trueVal)
MSE = 0;
for i=1:length(approx)
MSE = MSE + (approx(i) - trueVal(i))^2;
end
MSE = MSE / length(approx);
end
```

```
function [phi] = compPhi(na,nb,u,y,powerMatrix)
phi = [];
for k = 1:length(u)
elements = [];
for i = 1:na
if k - i > 0
elements = [elements y(k-i)];
else
elements = [elements 0];
end
end
for i = 1:nb
if k - i > 0
elements = [elements u(k-i)];
else
elements = [elements 0];
end
end
aux = [];
for i = 1:length(powerMatrix)
aux = [aux prod(elements .^ powerMatrix(i,:))];
end
phi = [phi; aux];
end
end
```

```
function [y_sim] = compYSim(na, nb, u,
PowerMatrix, theta)
y_sim = zeros(length(u),1);
for k = 1:length(u)
line = [];
for i = 1:na
if k - i > 0
line = [line y_sim(k-i)];
else
line = [line 0];
end
end
for i = 1:nb
if k - i > 0
line = [line u(k-i)];
else
line = [line 0];
end
end
aux = [];
for i = 1:length(PowerMatrix)
aux = [aux prod(line .^ PowerMatrix(i,:))];
end
y_sim(k) = aux * theta;
end
end
```

Function powerGen

```
function [power_mat] =
powerGen(no_inputs, power)
power_mat = [];
aux = zeros(1,no_inputs);
power_mat = aux;
aux(no_inputs) = 1;
power_mat = [power_mat; aux];
while aux(1) ~= power
sum_aux = sum(aux);
changed = 0;
i = length(aux);
while changed == 0
if sum_aux < power
aux(i) = aux(i) + 1;
changed = 1;
else
found = 0;
while found == 0
aux(i) = 0;
aux(i-1) = aux(i-1) + 1;
sum_aux = sum(aux);
if sum_aux <= power
found = 1;
else
i = i - 1;
end
end
changed = 1;
end
end
power_mat =
[power_mat; aux];
end
end
```

Main code

```
%| Nonlinear ARX Identification |
%|--------------------------------|
%| Petcuț Adrian-Axente |
%|--------------------------------|
%| Ilea Cosmin-Ionuț |
%|--------------------------------|
%| Szakacs Armand-Antonio |
%|--------------------------------|
%| SYSTEM IDENTIFICATION 2023-2024 |
%| TUCN |
%| Project part 2 |
%|--------------------------------|
clear all;
close all;
clc;
load("iddata-11.mat");
% choosing the parameters
na_max=3;
nb_max=3;
m_max=4;
% the time vectors
Ts=id.Ts;
t_id = id_array(:,1);
t_val = val_array(:,1);
u_id = id_array(:,2);
y_id = id_array(:,3);
% the input and output of the validation data
u_val = val_array(:,2);
y_val = val_array(:,3);
% declaring the MSE matrices
mse_prediction_matrix=zeros(na_max*nb_max, m_max);
mse_simulation_matrix=zeros(na_max*nb_max, m_max);
mse_simulation_min=1e200;
```

```
for m=1:m_max
for na=1:na_max
for nb=1:nb_max
% generation the matrix with powers
pow_matrix = powerGen(na+nb, m);
% computing phi for identification data
phi_id = compPhi(na, nb, u_id, y_id,
pow_matrix);
% computing theta
theta = phi_id\y_id;
% computing phi for validation data
phi_val = compPhi(na, nb, u_val, y_val,
pow_matrix);
% the one step ahead prediction for validation
y_hat_val = phi_val * theta;
% computing the simulation for validation
y_sim_val = compYSim(na, nb, u_val,
pow_matrix, theta);
% computing the MSE for prediction
mse_prediction = compMSE(y_hat_val, y_val);
% computing the MSE for simulation
mse_simulation = compMSE(y_sim_val, y_val);
% save the MSE values in matrices (each
collum represents a value
% of m)
mse_prediction_matrix((na-1)*nb_max +
nb, m) = mse_prediction;
mse_simulation_matrix((na-1)*nb_max +
nb, m) = mse_simulation;
if mse_simulation<mse_simulation_min
mse_prediction_min = mse_prediction;
mse_simulation_min = mse_simulation;
na_best_fit = na;
nb_best_fit = nb;
m_best_fit = m;
end
end
end
end
%get the best MSE depending on na and nb
na = na_best_fit;
nb = nb_best_fit;
m = m_best_fit;
```

```
% generation the matrix with powers
pow_matrix = powerGen(na+nb, m);
phi_id = compPhi(na, nb, u_id, y_id, pow_matrix);
% computing theta
theta = phi_id\y_id;
% computing phi for validation data
phi_val = compPhi(na, nb, u_val, y_val,
pow_matrix);
% the one step ahead prediction for validation
y_hat_val = phi_val * theta;
% computing the simulation for validation
y_sim_val = compYSim(na, nb, u_val, pow_matrix,
theta);
% computing the simulation for identification
y_sim_id = compYSim(na, nb, u_id, pow_matrix,
theta);
% the one step ahead prediction for identification
y_hat_id = phi_id * theta;
```

```
% computing the MSE for prediction
mse_prediction_id = compMSE(y_hat_id, y_id);
% computing the MSE for simulation
mse_simulation_id = compMSE(y_sim_id, y_id);
f1 = figure;
movegui(f1, 'northwest');
id_hat = iddata(y_hat_id,u_id,Ts);
compare(id,id_hat);
title("Prediction for identification set");
f2 = figure;
movegui(f2, 'southwest');
val_hat = iddata(y_hat_val,u_val,Ts);
compare(val,val_hat);
title("Prediction for validation set");
f3 = figure;
movegui(f3, 'northeast');
id_sim = iddata(y_sim_id,u_id,Ts);
compare(id,id_sim);
title("Simulation for identification set");
f4 = figure;
movegui(f4, 'southeast');
val_sim = iddata(y_sim_val,u_val,Ts);
compare(val,val_sim);
title("Simulation for validation set");
```

```
f5 = figure;
movegui(f5, 'north')
% taking the collum with the best MSE from
mse_simulation matrix and
% computing matrix z (rows represent
values for na and collums for nb
vect = mse_simulation_matrix(:,
m_best_fit)';
z = [];
for i = 1:na_max
z = [z; vect((i-1)*na_max+1:i*na_max)];
end
bar3(z);
xlabel('nb');
ylabel('na');
zlabel('MSE');
title(['Degree= ' num2str(m_best_fit)]);
```