System Identification Project

Part 2: Nonlinear ARX Identification

Data index:
22/10

Students	Matei-Alexandru Blaj Joelle-Mirela Danciu Mihai-Antonio Dîcă	Group 30333

Guiding professors Zoltan Nagy, Lucian Bușoniu

1. Introduction

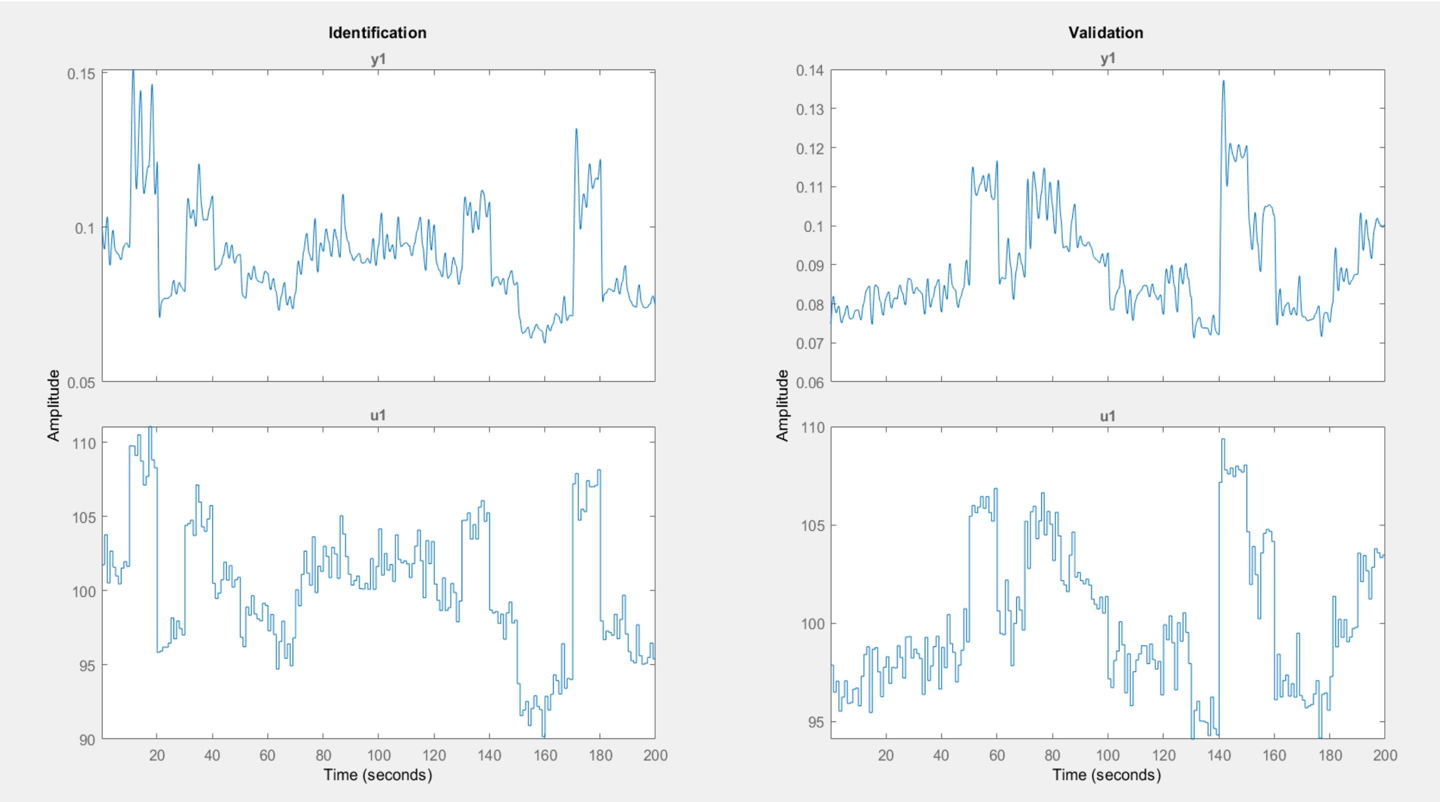
1.1. The data set and the system

1.2. A motivating example

1.3. ARX model

1.1 The data set and system

We were given two data sets (one for identification & one for validation), representing the input-output values of an unknown dynamic SISO system with the goal of finding a model.



Because we know that the system is dynamic we can develop an ARX model.

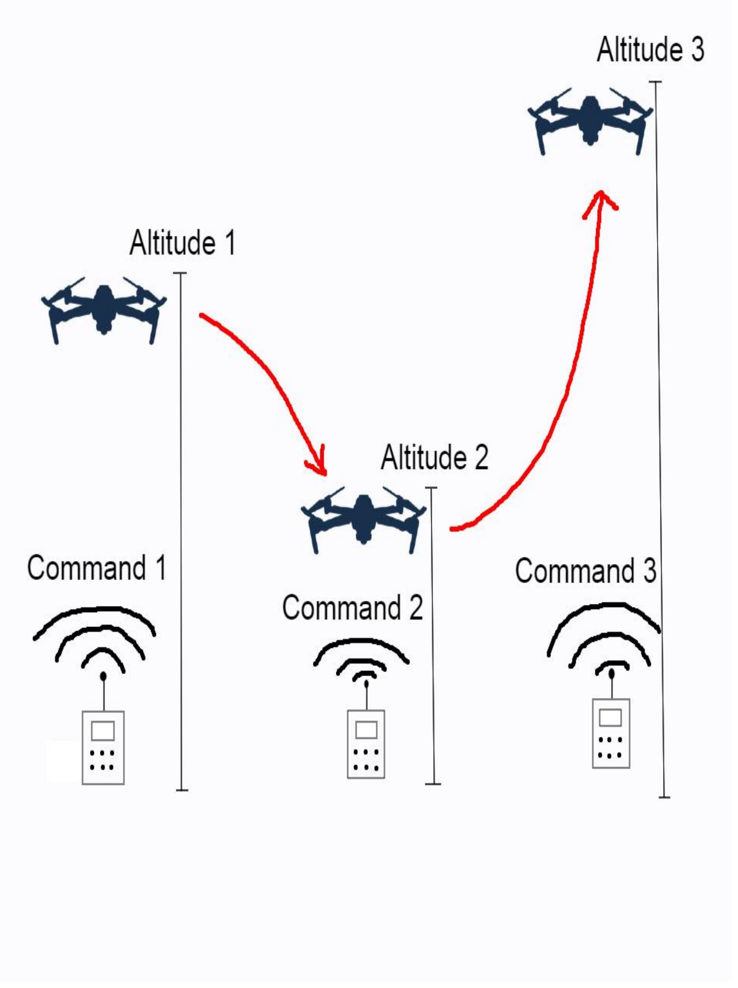
1.2. A motivating example

- Imagine that the system is a drone:
  - **Input**: command signal that controls the thrusters
  - **Output**: altitude

- We do not know the inner workings of the drone, but we wish to build a model for it.

- Our drone is constantly fighting against gravity:
  - **Thrust > Gravity**: the altitude increases
  - **Thrust < Gravity**: the altitude decreases

- Estimating the next altitude of the drone cannot solely be based on the current input, as the previous altitudes and the commands that produced them are also crucial.



1.3. ARX model

ARX stands for:

- **autoregressive**(the current output depends on the previous ones)
- with **exogenous** input(the current output depends on the previous inputs)

The order of the systems dynamics:

- dictates how many previous inputs & outputs influence the current output
- In our case is **not larger than 3** (no more than 3 previous inputs & outputs influence the current output)

Another piece of information that we were given is that the **dynamics are nonlinear**.

- Linear dynamics: the current output is a sum of weighted previous inputs & outputs
- Nonlinear dynamics: the current output is a weighted polynomial of previous inputs & outputs

The ARX assumes zero initial conditions. The given identification data set does not satisfy this assumption.

2. Algorithm

2.1. Polynomial regressor structure
2.2. The unknown parameter vector
2.3. Developing the regressor
2.4. Prediction VS Simulation
2.5. Simulated model instability

2.1. Polynomial regressor structure

First step in order to find the polynomial regressor is forming a vector of previous inputs & outputs:

$$r_e(k) = [-y(k-1) \quad -y(k-2) \quad \cdots \quad -y(k-n_a) \quad u(k-n_k) \quad \cdots \quad u(k-n_b-n_k+1)]$$

Where:
$n_a - \textit{the number of the previous outputs}$
$n_b - \textit{the number of the previous inputs}$
$n_k - \textit{the input delay (dead time of the system)}$

The terms of r_e are used to form a polynomial of degree m.

As an example, let's take m=2, n_a= n_b=1, n_k=1 and the vector $r_e = [-y(k-1) \;\; u(k-1)]$
The polynomial regressor, denoted φ, would contain:
$\varphi(r_e(k)) = [1 \quad -y\,(k-1) \quad -y(k-1)^2 \quad \mathrm{u(k-1)} \quad u(k-1)^2 \quad y(k-1) * u(k-1)]$
Each other output can be computed as:

$$y(k) = \sum_{i=1}^{6} \varphi(r_e(k))_i * \theta(i)$$

$\theta \;- \textit{the unknown parameters/weights vector}$

2.2. The unknown parameter vector

Finding the ARX model boils down to finding the parameters vector, which can be easily done by writing everything in matrix form and than solving the system of equations(linear regression)

$$\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(k+1) \\ \vdots \\ y(N+1) \end{bmatrix} = \begin{bmatrix} -y(0) & -y(0)^2 & u(0) & u(0)^2 & -y(0)*u(0) \\ -y(1) & -y(1)^2 & u(1) & u(1)^2 & -y(1)*u(1) \\ & & \vdots & & \\ -y(k) & -y(k)^2 & u(k) & u(k)^2 & -y(k)*u(k) \\ & & \vdots & & \\ -y(N) & -y(N)^2 & u(N) & u(N)^2 & -y(N)*u(N) \end{bmatrix} * \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix}$$

$$\Downarrow \qquad\qquad \Downarrow \qquad\qquad \Downarrow$$

Y $\qquad$ ϕ (the regressor) $\qquad$ θ

Note: if the index of y or u is smaller than 1 we assume the value those elements to be 0.

To compute this we use MATLAB's left matrix division:

$$\boldsymbol{\theta} = \boldsymbol{\phi} \setminus \boldsymbol{Y}$$

2.3. Developing the regressor

In order to find an algorithm for the polynomial regressor, we first tried a hard-coded and very long approach. The following is a small part of the function.

```
%pair of 6 (again just for the algorithm's sake):
    for a=1:m
        for b=1:m
            for c=1:m
                for d=1:m
                    for e=1:m
                        for f=1:m
                            for i=1:6-5
                                for j=i+1:6-4
                                    for k=j+1:6-3
                                        for p=k+1:6-2
                                            for q=p+1:6-1
                                                for s=q+1:6-0
                                                    if(a+b+c+d+e+f<=m)
                                                        ind=ind+1;
                                                        r(ind)=x(i)^a*x(j)^b*x(k)^c*x(p)^d*x(q)^e*x(s)^f;
                                                    end
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
```

Then, we came up with a recursive solution.

This creates pairs

```
Algorithm: recursive_pair
Input: prevIO,length_prevIO,pairTermsN0,m,exponents,prevIO_indices,prev_index
Output: regressorElements
|for a <-- prev_index+1 to (pairTermsN0-1)
|   do Save current element index in 'prevIO_indices' and in 'prev_index'
|   |for p <-- 1 to m
|   |   do Save current exponent in 'exponents'
|   |   |if the pair doesn't have 'pairTermsN0' chosen terms from prevIO
|   |   |   then
|   |   |     pairTerms <-- pairTermsN0-1
|   |   |     call recursive_pair
|   |   |   else
|   |   |     |if sum(exponents)<=m
|   |   |     |   then
|   |   |     |     do Multiply all chosen elements from prevIO
|   |   |     |     at their respective exponent and save the
|   |   |     |     product to 'regressorElements'
|   |   |     |endif
|   |   |endif
|   |   do Eliminate current exponent from 'exponents'
|   |endfor
|   do Eliminate current element index from 'prevIO_indices'
|endfor
```

This plugs the pairs into the polynomial

```
Algorithm: regressor_poly
Input: prevIO,length_prevIO,m
Output: regressorPolynomial
regressorPolynomial(1) <-- 1  %the free term
|for pairTermsN0 <-- 1:length_prevIO
|   |if pairTermsN0 <= m
|   |  then
|   |    call recursive_pair
|   |endif
|endfor
do transfer global values created in 'recursive_pair'
to 'regressorPolynomial'
return regressorPolynomial
```

This creates the regressor matrix

```
Algorithm: regressor
Input: na,nb,nk,y,u
Output: PHI
do Extract previous na outputs and nb inputs into 'r_e'
|for i <-- 1 to length(y)
|   do Assign the values returned by regressor_poly
|      for the elements of 'r_e' to the line 'i' of the
|      regressor matrix 'PHI'
|endfor
return PHI
```

2.4. Prediction VS Simulation

What does an ARX model actually do with a new set of inputs?

- It can **predict** outputs based on the previous, **real** ones
- It can **simulate** outputs based on the previous **simulated** ones

Recall the drone example. Considering it is a very expensive one, we would like to know how it would behave when somebody yanks the control sticks. This could be a threat to the drone as it could hit the ground, so instead of doing it on the real thing(prediction), we simply simulate its behavior.

One of the challenges that we faced when we tried to simulate different models was that almost all of the simulations were unstable, because of the non-zero initial conditions of our identification data set.

2.5. Simulated model instability

Because the ARX model assumes zero initial conditions, the first value of the output data is translated by the algorithm into an amount by which all other values are raised, which is false. Because of this $\theta(1)$ will be equal to that amount, causing a large initial error for the estimated outputs on validation data sets.

- Prediction: this large error quickly disappears as the algorithm gains access to previous real outputs
- Simulation: the error gets passed on exponentially until it becomes infinite, causing an unstable simulation

The only workaround that we found for this problem was starting the simulation from $n_b + n_k$, so that the algorithm would make a better guess for the first value of the simulation using the first $n_b + n_k$ inputs.

3. Tuning results

3.1. MSE

3.2. Best model for prediction

3.3. Best model for simulation

3.1. MSE

To validate the output of our computed model and reach a good degree m for our polynomial, we need a precise, numerical indicator of how good our approximated data fits the real data. This is why we use the mean-squared error, as it computes the average difference between the predicted/simulated and the actual data.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \widehat{y_i})^2$$

Where:

$$N - the\ number\ of\ values\ in\ y/\hat{y}$$
$$y\ - the\ real\ output\ data\ vector$$
$$\hat{y} - the\ predicted\ or\ simulated\ data\ vector$$

Note: because of the large initial errors the MSE might be compromised, so we offset the starting point for computing it by $n_a + n_b + n_k$.

The way we found the best models for prediction & simulation was by trying out multiple combinations of orders and parameters.

Note: the best model for prediction might be different from the one for simulation

```
|for m <-- 1 to 8
|    |for na <-- 1 to 3
|    |    |for nb <-- 1 to 3
|    |    |    |for nk <-- 1 to 3
|    |    |    |    do Compute ARX model with given m,na,nb,nk
|    |    |    |    do Compute ypred, ysim
|    |    |    |    do Compute MSEpred, MSEsim
|    |    |    |    |if (MSEpred<best_MSEpred)
|    |    |    |    |    then Save parameters and prediction output
|    |    |    |    |         best_MSEpred <-- MSEpred
|    |    |    |    |endif
|    |    |    |
|    |    |    |    |if (MSEsim<best_MSEsim)
|    |    |    |    |    then Save parameters and simulation output
|    |    |    |    |         best_MSEsim <-- MSEsim
|    |    |    |    |endif
|    |    |    |endfor
|    |    |endfor
|    |endfor
|endfor
```

3.2. Best model for prediction

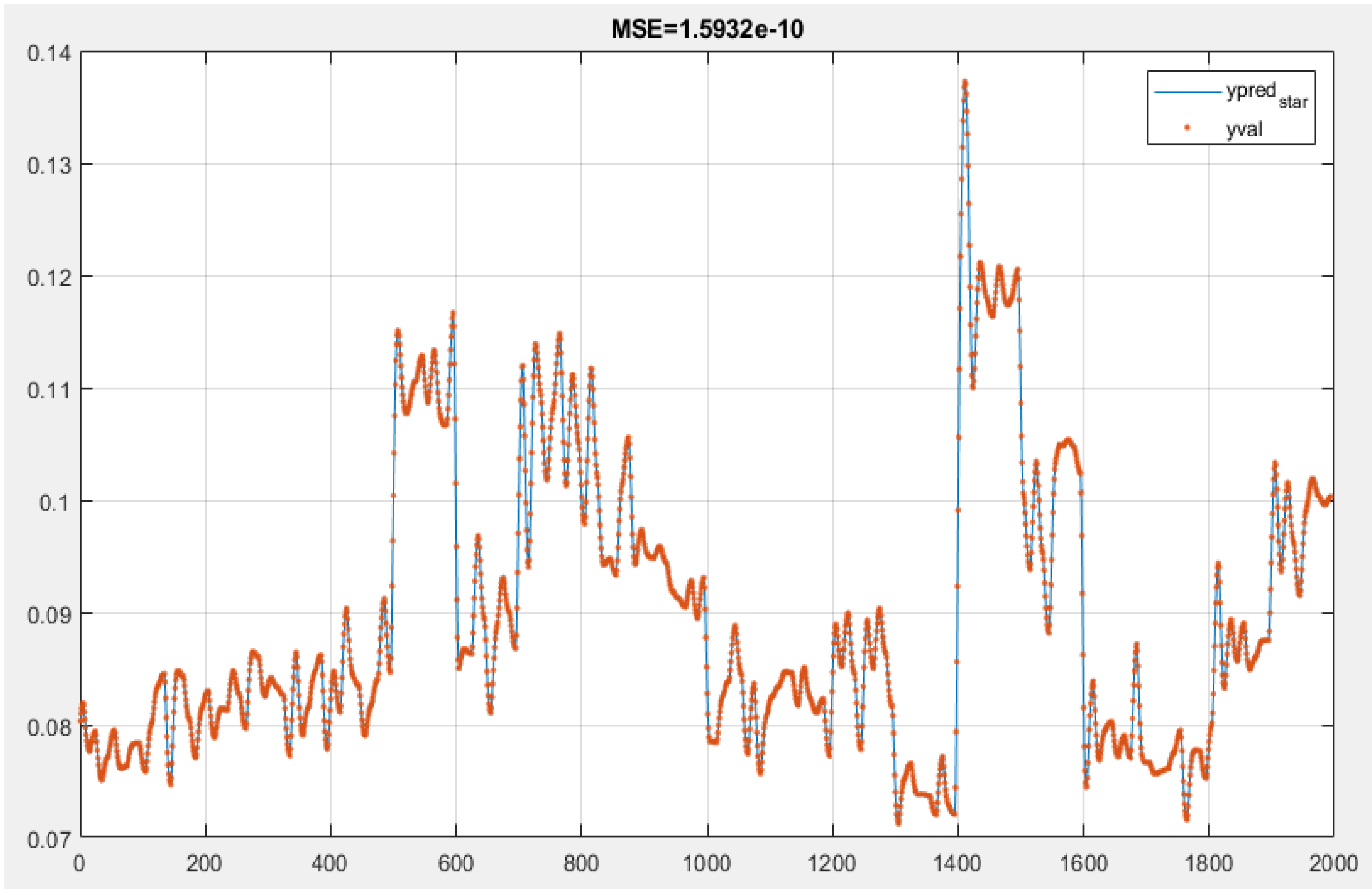
The following parameters yielded the best model for prediction:

- $n_a = 3$
- $n_b = 3$
- $n_k = 1$
- $m = 3$

ypredstar

2000x1 double

	1
1	0.1000
2	0.2628
3	0.1404
4	0.0774
5	0.0785
6	0.0795
7	0.0804
8	0.0810
9	0.0815
10	0.0818
11	0.0820
12	0.0817
13	0.0812
14	0.0805
15	0.0797



3.3. Best model for simulation

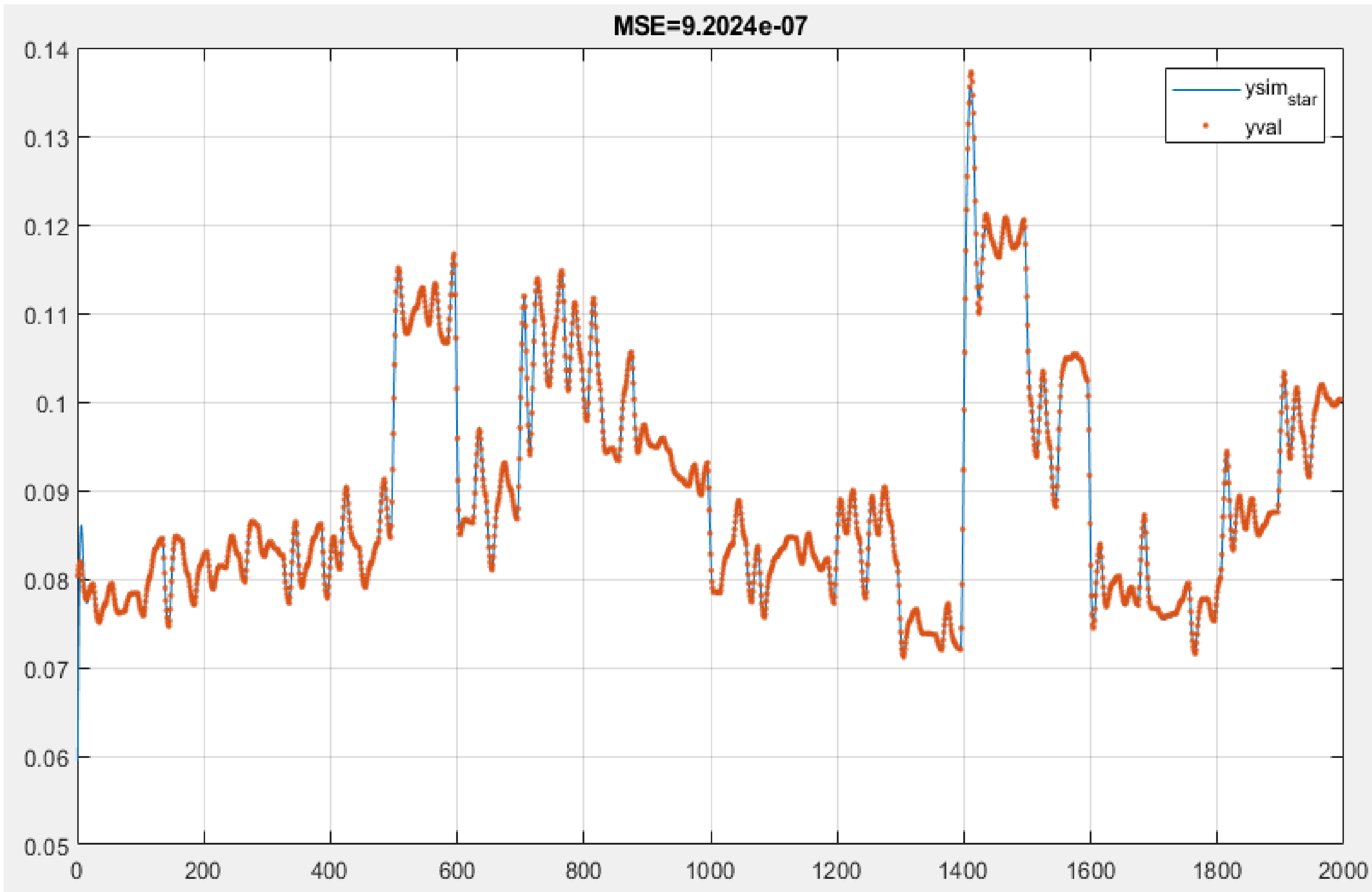
The following parameters yielded the best model for simulation:

- $n_a = 3$
- $n_b = 3$
- $n_k = 1$
- $m = 2$

ysimstar

2000x1 double

	1
1	0
2	0
3	0
4	0.0134
5	0.0219
6	0.0374
7	0.0489
8	0.0602
9	0.0693
10	0.0764
11	0.0816
12	0.0845
13	0.0855
14	0.0852
15	0.0840



Conclusion

The ARX modelling method is a powerful tool for predicting and simulating the behavior of a dynamic system with very good confidence, which would otherwise be impossible to describe with a fixed function.