Nonlinear ARX

Alexandru Zigler

December 2021

Contents

Introduction

Structure and implementation

Results and errors

Conclusions

Introduction: ARX method

Linear ARX:

$$y(k) = \big[-y(k-1), -y(k-2), \ldots, -y(k-na), u(k-1), u(k-2), \ldots, u(k-nb)\big] \cdot \theta + e(k)$$

The output at step k depends linearly on previous inputs and outputs.
θ is the parameter vector and $e(k)$ denotes the noise.

Nonlinear ARX:

$$y(k) = g(y(k-1), y(k-2), \ldots, y(k-na), u(k-1), u(k-2), \ldots, u(k-nb)) + e(k)$$

The polynomial g has $na + nb$ variables i.e. previous inputs and outputs. The degree is noted by m and the coefficients of g are parameters θ.

Problem specification

We are provided an identification dataset - *Fig. 1a*. Based on the dataset we can determine NARX models, depending on the selected values *na*, *nb* and *m* (optionally *nk*).

We can validate our model using an additional provided dataset - *Fig. 1b*. For both identification and validation, the model will be used in prediction and simulation mode.

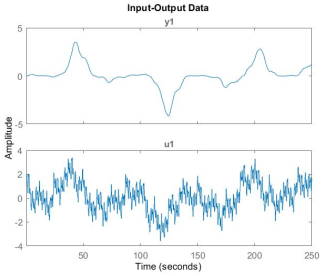


Figure: 1a Identification

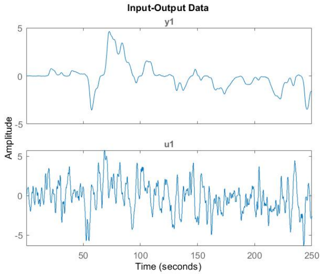


Figure: 1b Validation

Objectives

- designing a code with *na*, *nb*, *m* - configurable;
- plotting graphs for prediction and simulation, comparing them with initial datasets (id & val);
- calculate the mean square errors;
- for *na*,*nb* and *m* in predefined ranges[1], compare the approximation performance;
- plotting graphs showing the evolution of errors as functions of *na*, *nb* and *m*;
- From this set of models, we select the best one by minimizing the mean square error;

[1] $na \leq 5$, $nb \leq 5$, $m \leq 5$ and $na = nb$

Structure: Polynomial form

$$x(k) = \left[y(k-1), y(k-2), \dots, y(k-na), u(k-1), u(k-2), \dots, u(k-nb)\right]$$

We rewrite

$$x(k) = \left[x_1(k), x_2(k), \dots, x_{na}(k), x_{na+1}(k), x_{na+2}(k), \dots, x_N(k)\right], N = na+nb$$

$$g(x(k)) = \theta_1 \cdot 1 + \theta_2 \cdot x_1 + \theta_3 \cdot x_2 + \dots + \theta_{N+1} \cdot x_N +$$
$$+\theta_{N+2} \cdot x_1^2 + \theta_{N+3} \cdot x_2^2 + \dots + \theta_{2N+1} \cdot x_N^2 + \theta_{2N+2} \cdot x_1 x_2 + \theta_{2N+3} \cdot x_1 x_3 + \dots +$$
$$+\theta_{3N} x_1 x_N + \dots + \theta_? \cdot x_{N-1} x_N + \dots$$

Matrix form

$$\begin{bmatrix} 1 & x_1(1) & x_2(1) & \dots & x_N(1) & \dots \\ 1 & x_1(2) & x_2(2) & \dots & x_N(2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \\ 1 & x_1(n) & x_2(n) & \dots & x_N(n) & \dots \end{bmatrix} \cdot \theta = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(n) \end{bmatrix}$$

$\Phi \cdot \theta = Y$, where n is the size of the identification dataset.
To construct the matrix ϕ, we generate all combinations $\{p_1(j), p_2(j), \dots, p_N(j)\}, \quad \sum_{i=1}^{N} p_i \leq m, \quad 0 \leq p_i \leq m,$
where j - combination index

We note the line k of Φ with $\left[\varphi_1(k), \varphi_2(k), \dots\right]$.

Then $\varphi_j(k) = \prod_{i=1}^{N} x_i(k)^{p_i(j)}$.

Implementation

- "proiect2arx.m"
- "generate_generalized_v1.m"
- "generate_generalized_v2.m"
- "mlen.m"
- "thesearch.m"

Prediction and simulation

We construct the matrix X with vectors $x(k), k = \overline{1, \mathrm{n}}$

$$x(k) = \left[y(k-1), y(k-2), \dots, y(k-na), u(k-1), u(k-2), \dots, u(k-nb)\right]$$

We construct matrix Φ line by line based on the corresponding line from X. One of the generating functions is used.
We determine the parameters θ using linear regression.
We calculate matrices X and Φ for validation dataset.
We validate the model using the previously identified parameter vector θ.

For simulation, matrix X will be constructed line by line. We use simulated outputs from the previous steps instead of the real ones. The vector y is initialized to zero (zero initial conditions). It will be updated for each line in X.

$$\hat{y}_{sim}(k) = \varphi(k) \cdot \theta$$

where $\varphi(k)$ denotes the regressors corresponding to line k from X.

Number of regressors

Combinations with repetition

$$\left(\!\binom{n}{k}\!\right) = C^{k}_{n+k-1} = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!(n-1)!}$$

Example We have two sets of objects: type A and type B. We want to choose 3 objects. In how many ways can we make the choice?

3 x objects A + 0 x objects B
2 x objects A + 1 x object B
1 x object A + 2 x objects B
0 x objects A + 3 x objects B

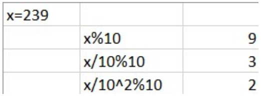
$$\left(\!\binom{2}{3}\!\right) = \binom{3+2-1}{3} = \frac{4!}{3! \cdot 1!} = 4$$

The function "mlen.m" iterates through the numbers from 0 to m (fixed) and calculates the total number of regressors by applying this formula for each iteration.

Generating regressors method 1

$f : \{1, 2, \ldots, N\} \longrightarrow \{0, 1, \ldots, m\}$

There are $(m+1)^N$ such functions. So we can associate each natural number from 1 to $(m+1)^N$ a function f.



x=239		
	x%10	9
	x/10%10	3
	x/10^2%10	2

Figure: Extracting digits of a number in base 10

We use the concept of extracting digits of a number in base 10 and we adapt it for base $m+1$.

$f_k(i) = (k-1)/(m+1)^{(N-i)}\%(m+1)$

$k = \overline{1, (m+1)^N}$ $\qquad N - i = \overline{0, N-1}$

We store in a matrix only the functions f_k that have the sum of values less than or equal to m.

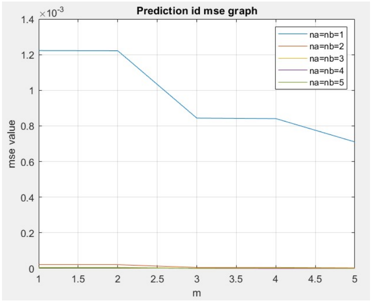
Generate regressors method 2

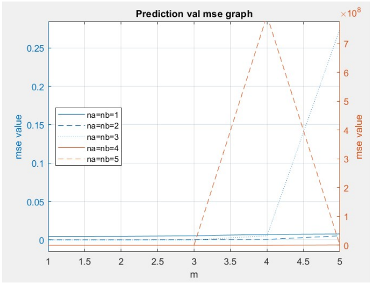
Consider 2 vectors of size N.
-*vector1*:initialized to 0, we use it to generate the current line with exponentials
-*vector2*: boolean vector, initialized with 0; a value from *vector2* becomes 1 when the corresponding value from *vector1* equals m
For each iteration, we increment the last position from *vector1*. When the last position reaches the value m, the corresponding value from *vector2* becomes 1. We reset the last position and we increment the neighbor position from the left.
A position k will be incremented when all the positions to its right have value 1 in *vector2*.
We memorize only those vectors *vector1* that have the sum of elements less than or equal to m.
The algorithm stops when all the values from *vector2* become 1.

Note: Method 1 is more efficient.

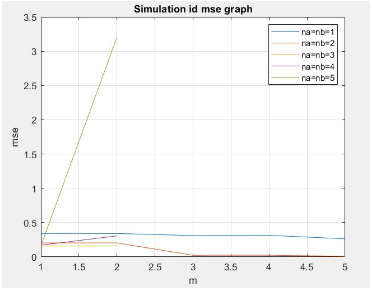
Tuning results

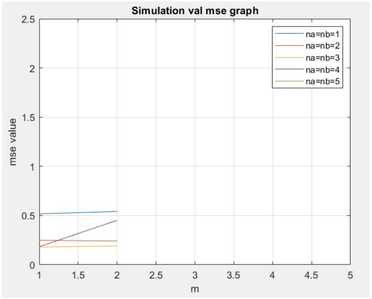
We impose *na*,*nb*, *m* to be in the range 1 to 5, with the goal of obtaining the best model (minimize **m**ean **s**quare **e**rror). We store the *mse* values in 4 matrices (id prediction, val prediction, id simulation, val simulation) and plot their evolution as functions of *m*,*na*,*nb*.





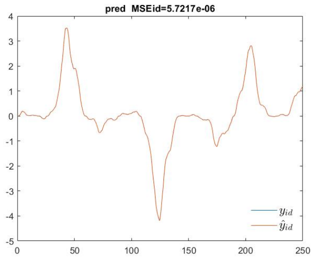
Tuning results

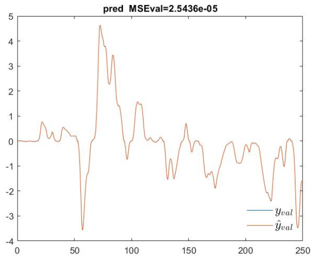


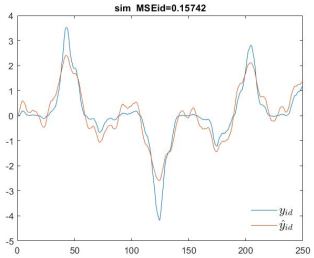


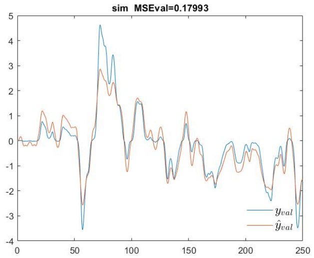
We note that for simulation graphs we have NaN values. They occur because the simulated output has very high values.

Optimum model: $na = nb = 3, m = 1$









Conclusions

- for prediction, the identification errors decrease as we increase variables m,na,nb, which is to be expected;
- the simulation errors are initially small(< 1); when the error reaches value 1, it grows exponentially; after a few samples we get NaN values; this phenomenon occurs for sufficiently large m;
- the optimum model has prediction error $< 10^{-5}$; for simulation, the error is < 0.2 and we see that the signals $\hat{y}_{id,val}$ have similar shape to $y_{id,val}$;

To conclude, we successfully fulfilled all the established objectives. We found an optimum model, with great performance both for prediction and simulation.