SYSTEM IDENTIFICATION

Nonlinear ARX identification

By Peter Alexia, Vilau Paul and Tivadar Razvan
12/8

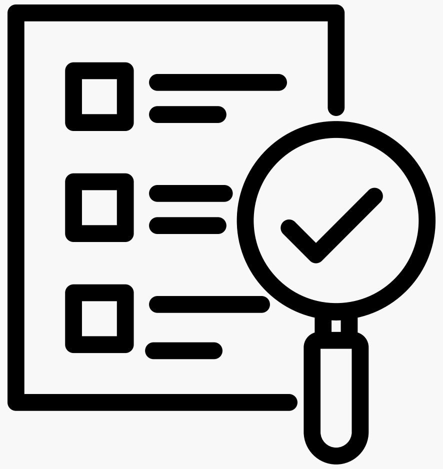
Problem statement

GOAL

Develop a nonlinear ARX model to approximate an unknown, nonlinear static function.

DATA PROVIDED

Identification data and validation data for I/O



REQUIREMENTS

Finding the best fit for the unknown nonlinear function

NARX Structure

ARX form:

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) - \cdots - a_{na} y(k-na) + b_1 u(k-1) + b_2 u(k-2) + \cdots + b_{nb} u(k-nb) + e(k)$$

$$\Downarrow$$

$$y(k) = p(-a_1 y(k-1) - a_2 y(k-2) - \cdots - a_{na} y(k-na) + b_1 u(k-1) + b_2 u(k-2) + \cdots + b_{nb} u(k-nb); \theta) + e(k)$$

NARX Structure

$$\hat{y}(k) = p\big(y(k-1), \ldots, y(k-n_a),\ u(k-n_k), u(k-n_k-1), \ldots, u(k-n_k-n_b+1)\big)$$

$$d(k) = \big[y(k-1), \ldots, y(k-n_a),\ u(k-n_k), u(k-n_k-1), \ldots, u(k-n_k-n_b+1)\big]^T$$

For $n_a = n_b = n_k = 1$,

$$d(k) = \big[y(k-1), u(k-1)\big]^T$$

$$y(k) = ay(k-1) + bu(k-1)\ +\ cy(k-1)^2 + vu(k-1)^2\ +\ wu(k-1)y(k-1) + z$$

How to find θ parameters

1. Calculate regressor matrix with polynomial expansion using the lagged inputs and outputs as terms

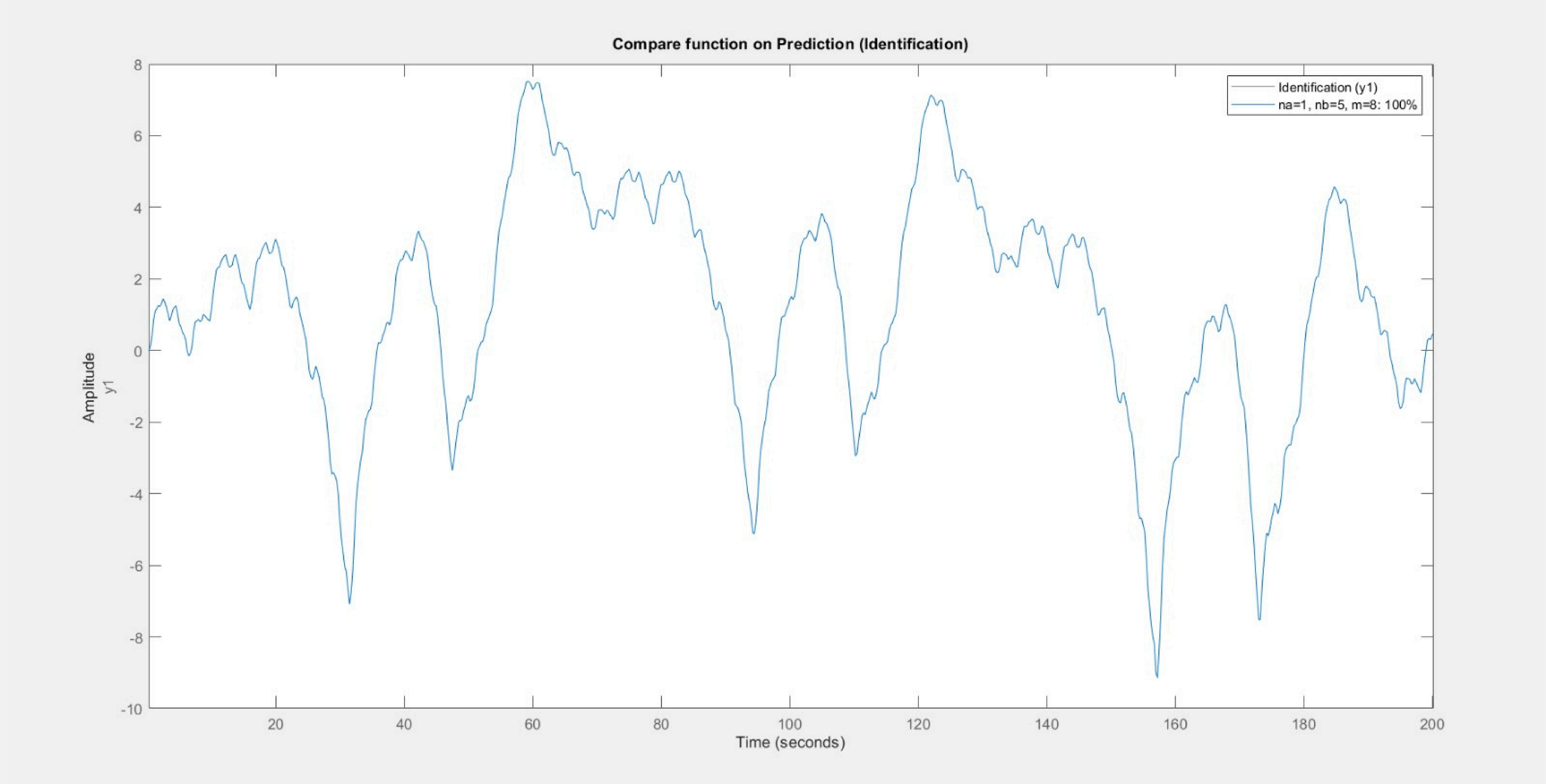
$$\phi = \phi(x_1, x_2, x_3)$$

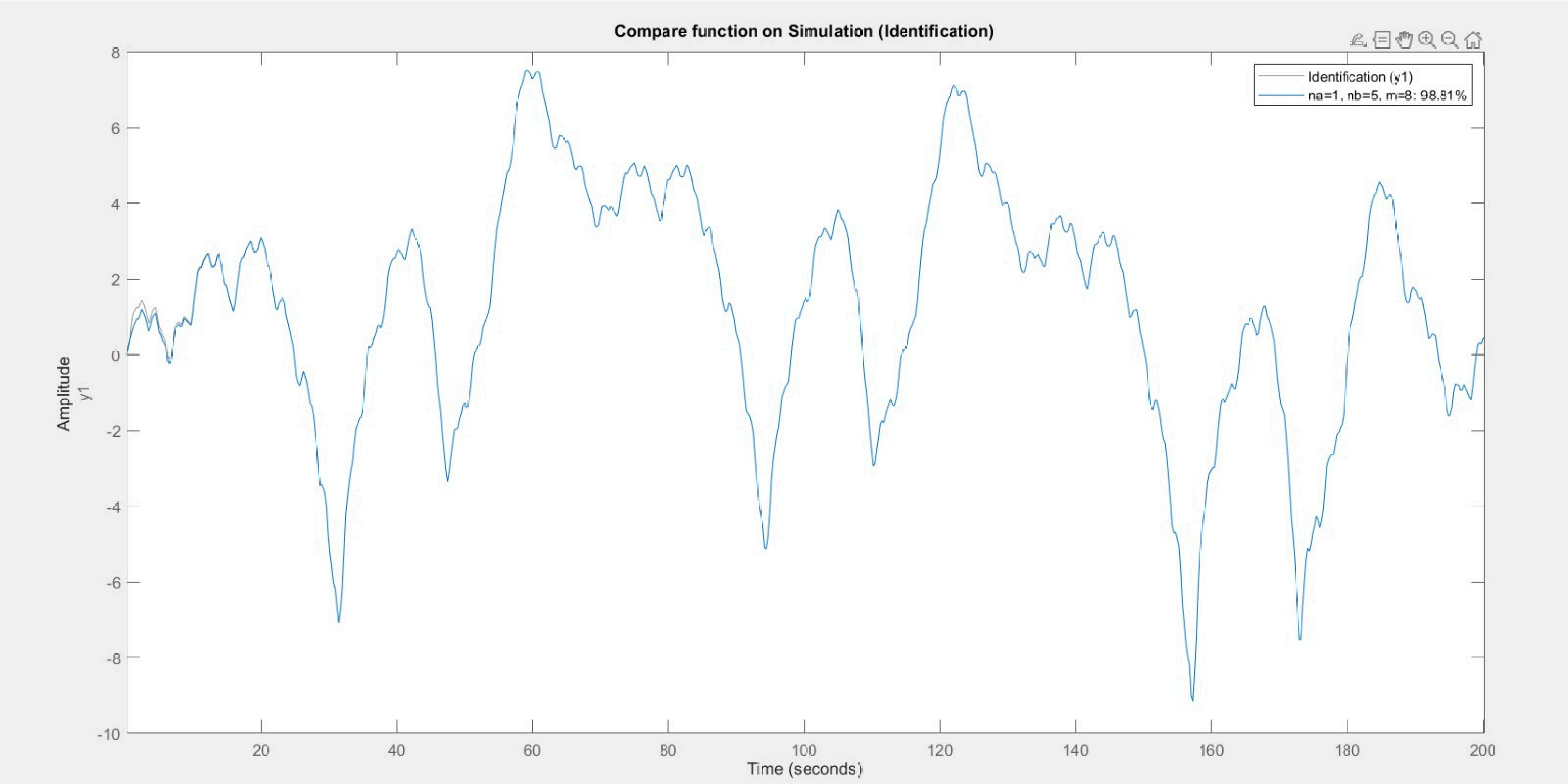
2. Calculate θ column vector

$$\theta = (\phi^\top \phi)^{-1} \phi^\top Y \quad \text{or} \quad \theta = \phi \backslash Y$$

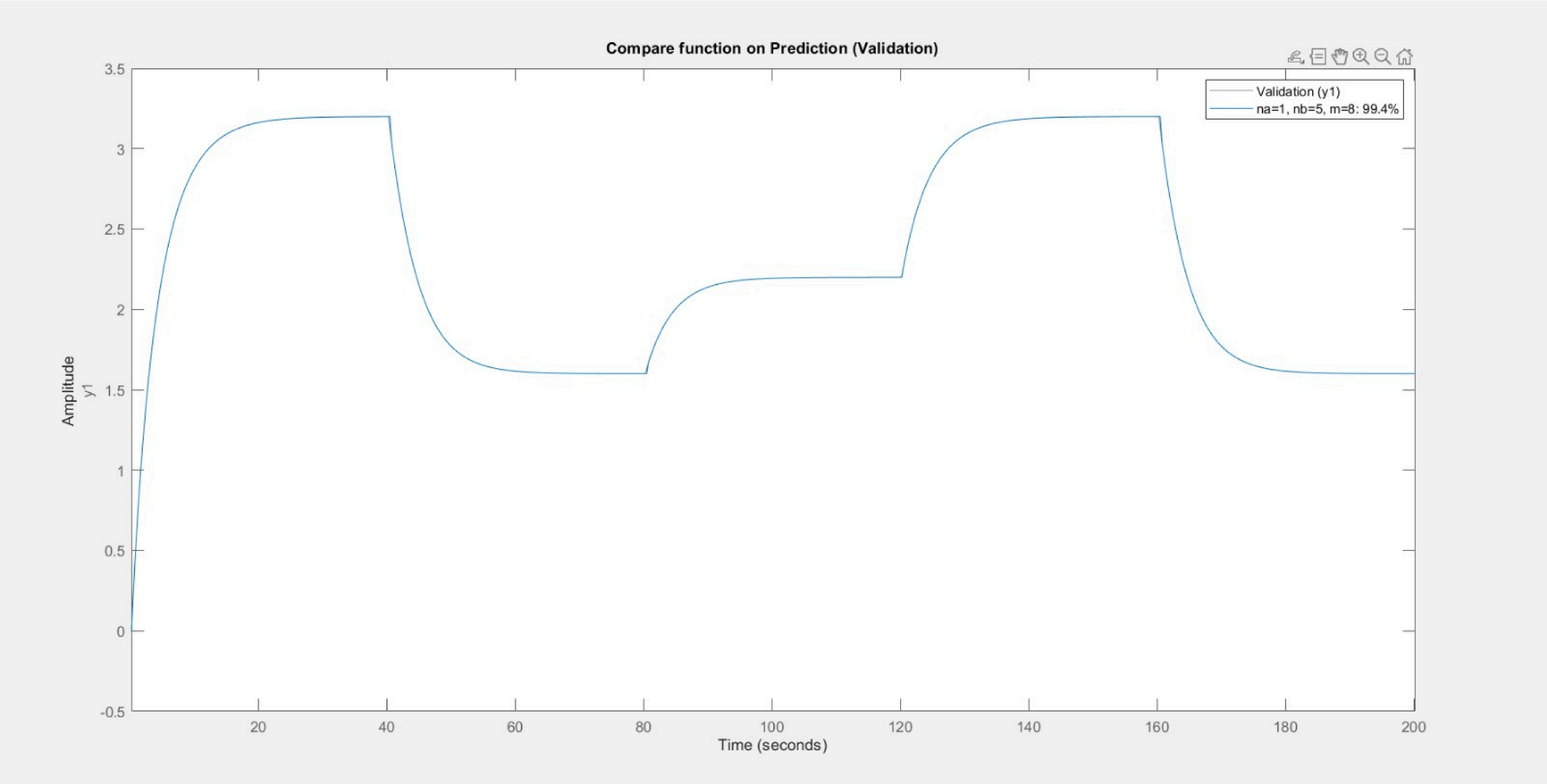
Key features

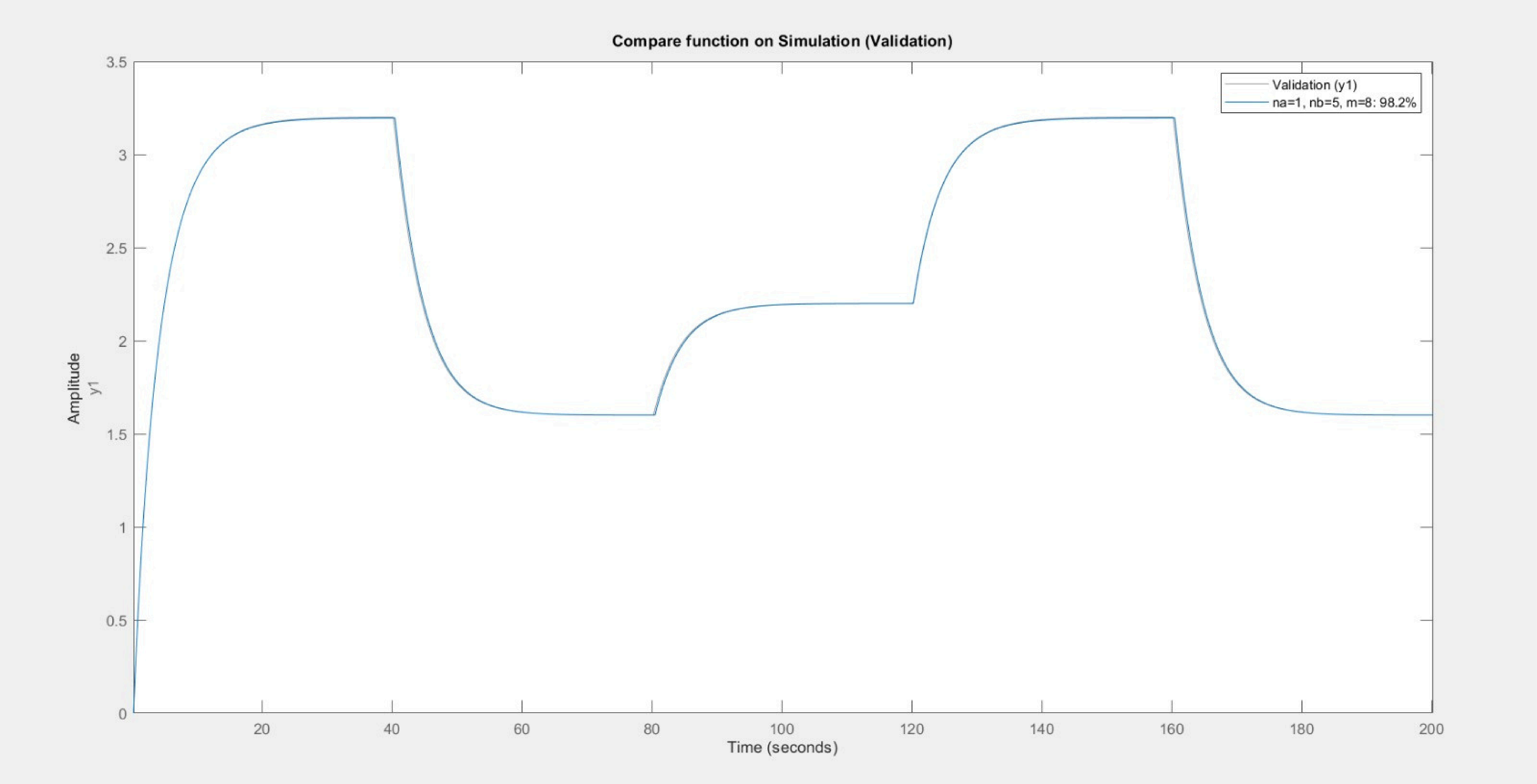
- Polynomial terms generator function
- Configurable model parameters
- Performance metrics


Compare function on Prediction (Identification)
Identification (y1)
na=1, nb=5, m=8: 100%
Amplitude
y1
Time (seconds)
8
6
4
2
0
-2
-4
-6
-8
-10
20
40
60
80
100
120
140
160
180
200



Compare function on Simulation (Identification)
Identification (y1)
na=1, nb=5, m=8: 98.81%
Amplitude
y1
Time (seconds)
8
6
4
2
0
-2
-4
-6
-8
-10
20
40
60
80
100
120
140
160
180
200


Compare function on Prediction (Validation)
Validation (y1)
na=1, nb=5, m=8: 99.4%
Amplitude
y1
Time (seconds)
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
20
40
60
80
100
120
140
160
180
200

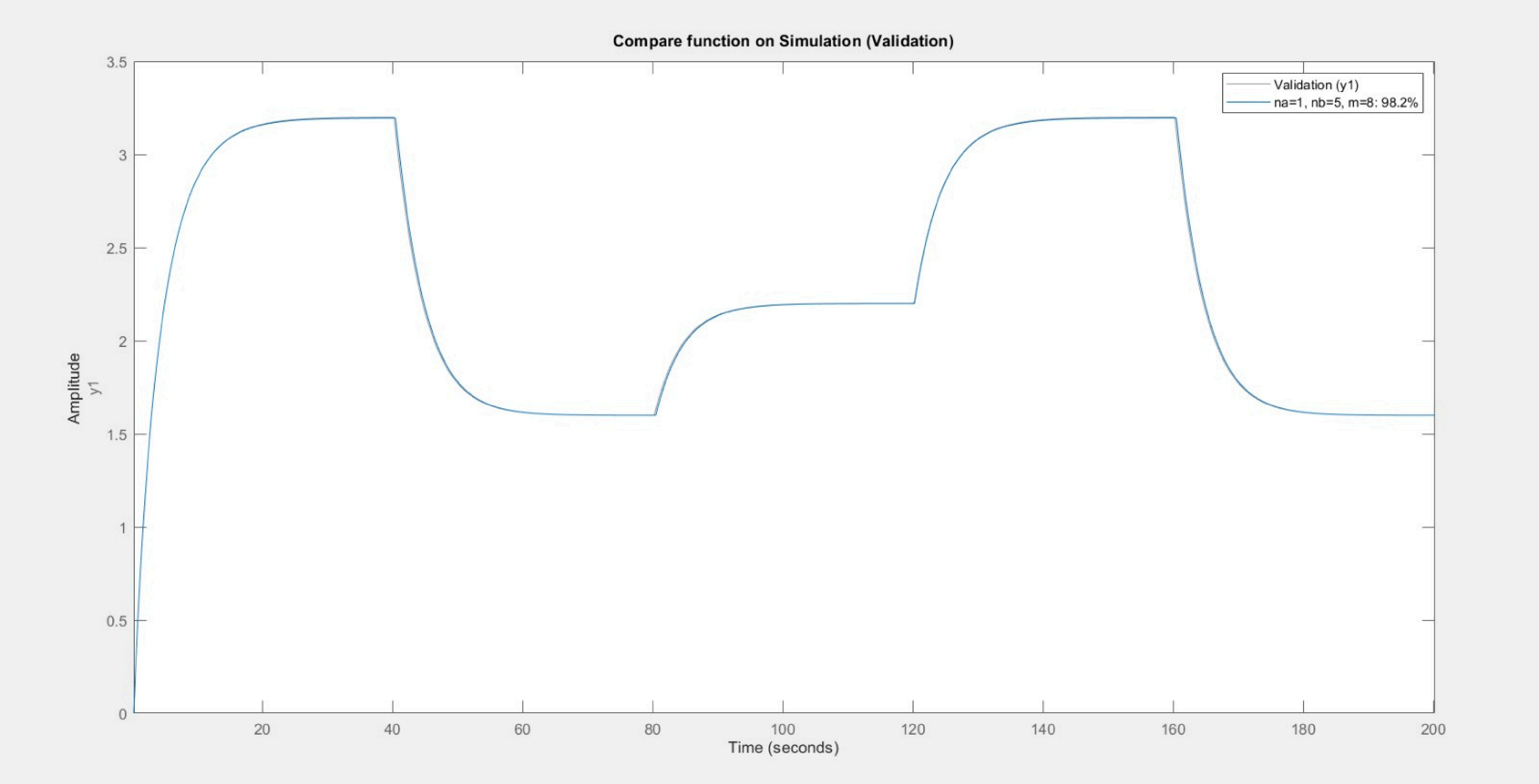

Compare function on Simulation (Validation)
Validation (y1)
na=1, nb=5, m=8: 98.2%
Amplitude
y1
Time (seconds)
3.5
3
2.5
2
1.5
1
0.5
0
20
40
60
80
100
120
140
160
180
200

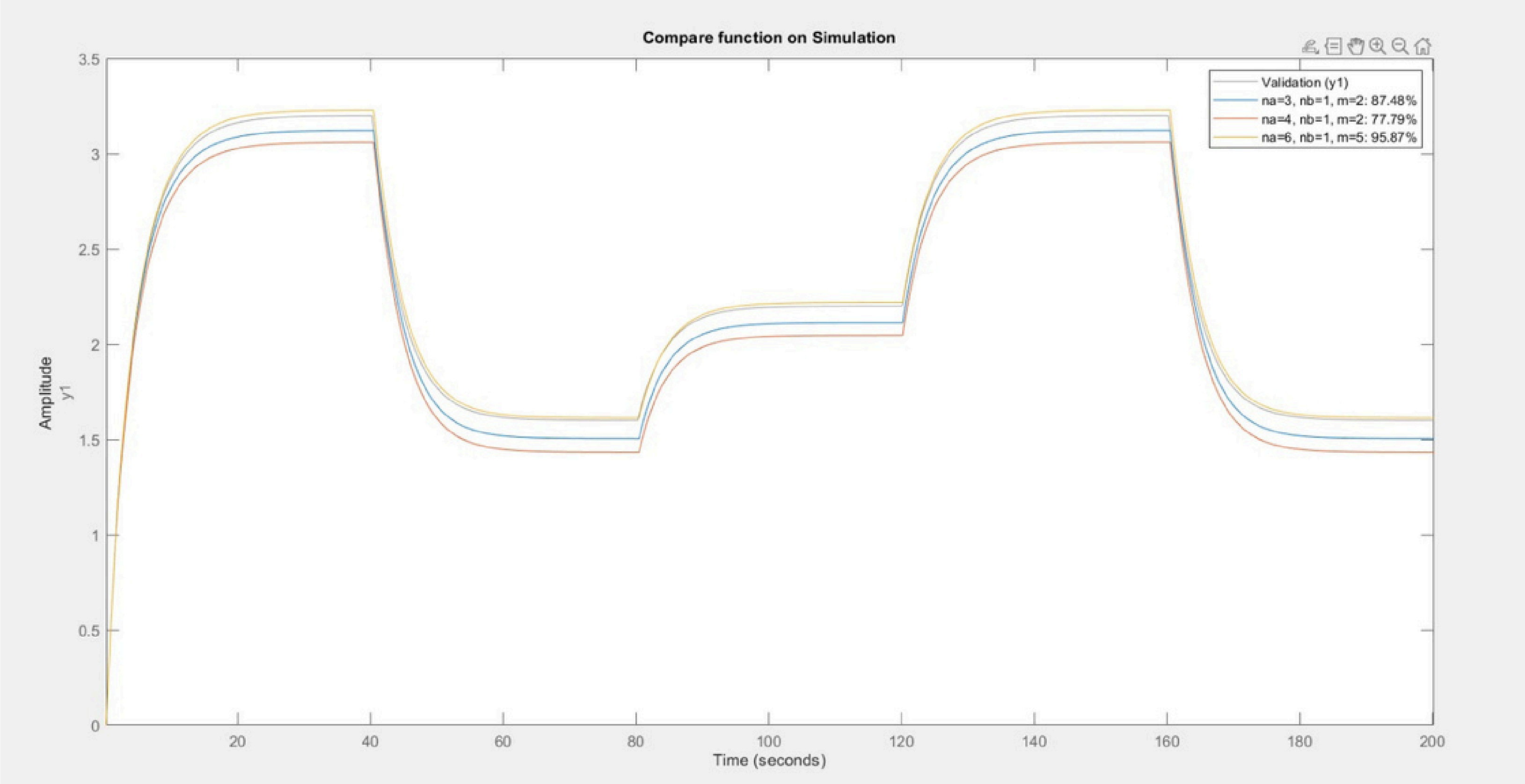
MSE Chart

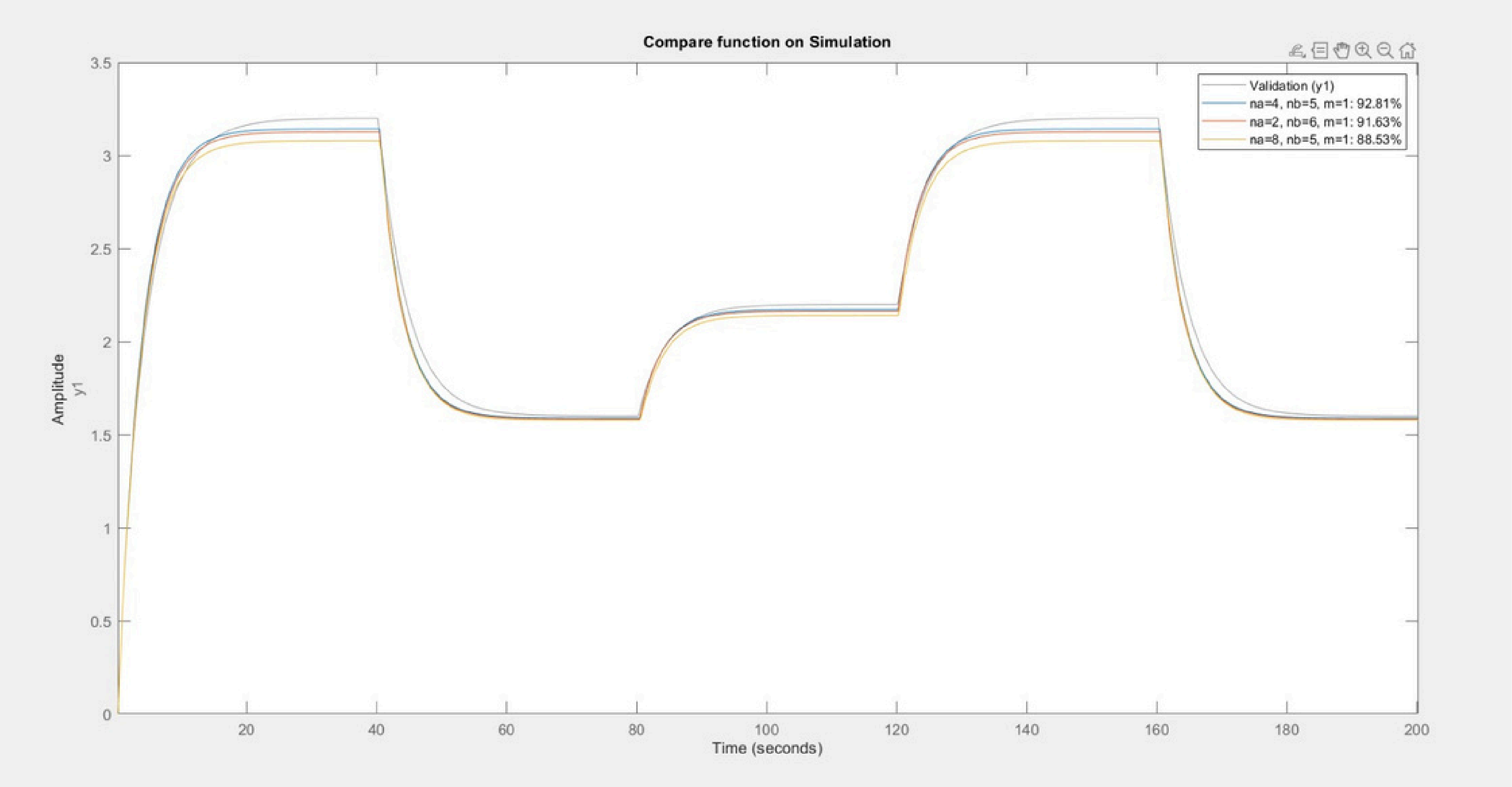
	BEST	1	2	3	4	5	6	7	8	9
na	1	3	4	6	4	2	8	2	1	3
nb	5	1	1	1	5	6	5	2	1	3
m	8	2	2	5	1	1	1	2	1	3
OSA MSE id	2.60E-29	8.00E-04	7.00E-04	1.40E-11	2.00E-03	3.00E-03	1.00E-03	4.00E-04	7.00E-03	1.20E-05
Simulation MSE id	1.50E-05	4.07E-05	9.10E-05	1.70E-05	4.50E-05	3.40E-05	1.00E-04	3.50E-01	2.80E-05	7.71E+05
OSA MSE val	8.70E-28	0.01	0.01	1.40E-09	0.16	0.17	0.2	NaN	0.16	NaN
Simulation MSE val	1.00E-04	0.006	0.02	7.00E-04	0.002	0.003	0.005	NaN	0.001	NaN

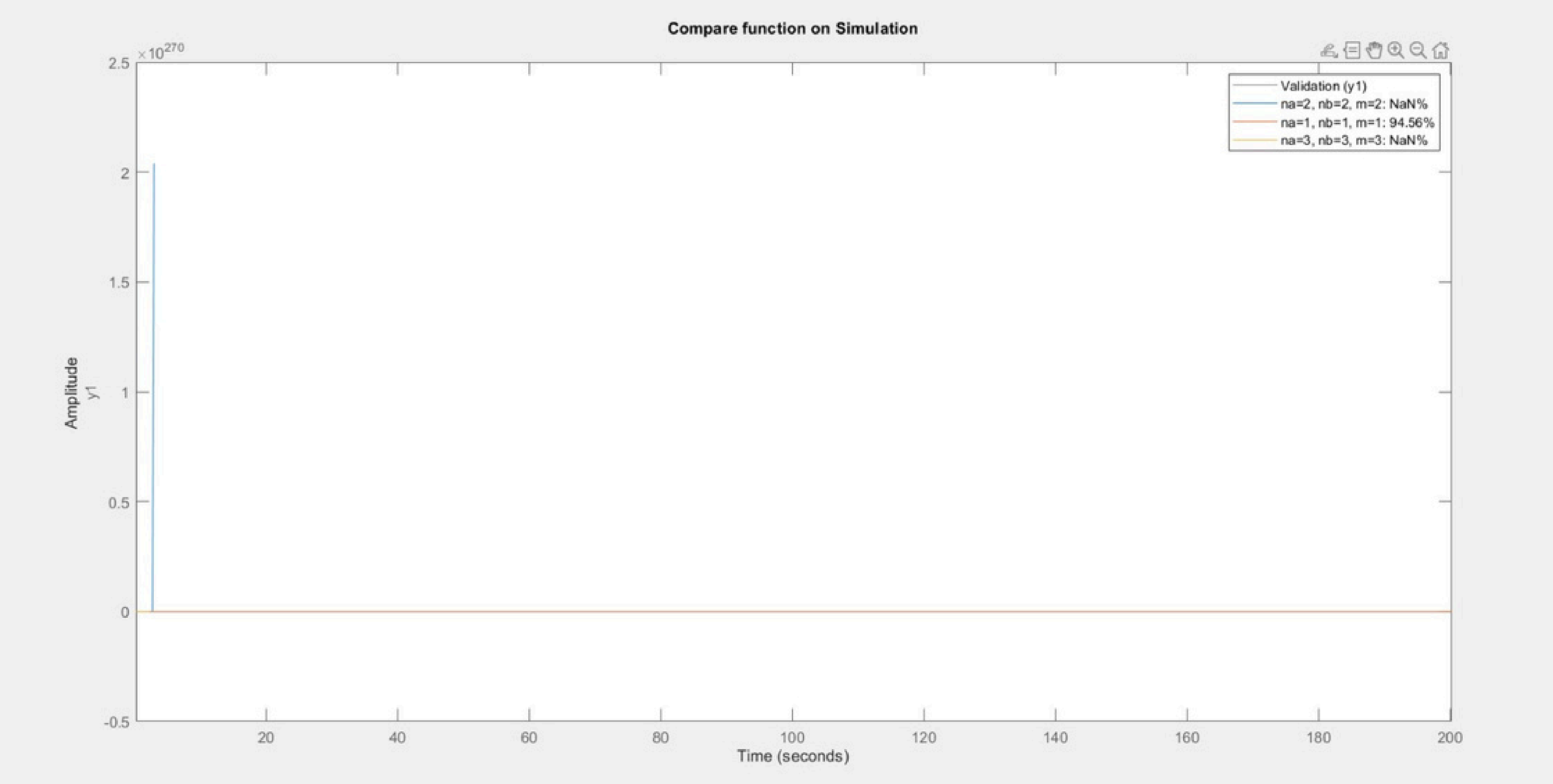
CONCLUSIONS

Our particular dataset made the algorithm follow a particular behavior where one parameter always had to be equal to 1 (e.g. na = 1, nb = 5, m = 8, optimal fit)




Compare function on Simulation
Validation (y1)
na=3, nb=1, m=2: 87.48%
na=4, nb=1, m=2: 77.79%
na=6, nb=1, m=5: 95.87%
Amplitude
y1
Time (seconds)


Compare function on Simulation
Validation (y1)
na=4, nb=5, m=1: 92.81%
na=2, nb=6, m=1: 91.63%
na=8, nb=5, m=1: 88.53%
Amplitude
y1
Time (seconds)


Compare function on Simulation
×10^270
Validation (y1)
na=2, nb=2, m=2: NaN%
na=1, nb=1, m=1: 94.56%
na=3, nb=3, m=3: NaN%
Amplitude
y1
Time (seconds)

Thank you

Code

```
function phi = construct_regressor(y, u, na, nb)
    N = length(y);
    phi = zeros(N, na + nb);
    for i = 1:N
        for j = 1:na
            if i - j > 0
                phi(i, j) = -y(i - j);
            else
                phi(i, j) = 0;
            end
        end
        for j = 1:nb
            if i - j  > 0
                phi(i, j+na) = u(i - j);
            else
                phi(i, j+na) = 0;
            end
        end
    end
end
```

```
function combinations = generate_combinations(N, total_degree)

    if N == 1
        combinations = total_degree;
    else
        combinations = [];
        for k = 0:total_degree
(total_degree - k)
            sub_combinations = generate_combinations(N - 1, total_degree - k);
            combinations = [combinations; [k * ones(size(sub_combinations, 1), 1), su
        end
    end
end
```

```
function power_combinations = generate_power_combinations(num_vars, degree
    power_combinations = [];
    for total_degree = 0:degree
        combinations = generate_combinations(num_vars, total_degree);
        power_combinations = [power_combinations; combinations];
    end
end
```

Code

```
function phi = generate_polynomial_terms(d, m)
    n = size(d, 1);
    num_vars = size(d, 2);
    combinations = generate_power_combinations(num_vars, m);
    phi = ones(n, size(combinations, 1));
    for i = 1:size(combinations, 1)
        term = ones(n, 1);
        for j = 1:num_vars
            term = term .* d(:, j).^combinations(i, j);
        end
        phi(:, i) = term;
    end
end
```

```
mse_sim = mean((yid - ysim_id).^2);
```