In The Name of God

Fourth Homework

Nonlinear Dynamic Systems identification

Student:

Mohamad Alikhani

Professor:

Dr. Mehdi Aliyari Shoorehdeli

Winter of 2024

Contents

Analytical Question

1)

$$\hat{y}(t) = f(u(t-1), u(t-2), \hat{y}(t-1))$$

$$\rightarrow a_1 u(t-1) + a_2 u(t-2) + a_3 \hat{y}(t-1) + a_4 u^2(t-1) + a_5 u^2(t-2) + a_6 \hat{y}^2(t-1) + a_7 u(t-1)u(t-2) + a_8 u(t-1)\hat{y}(t-1) + a_9 u(t-2)\hat{y}(t-1)$$

The cost function:

$$j = \frac{1}{2} e^2$$

Second power coefficient:

$$a_4(t) = a_4(t-1) - \eta \frac{\partial j}{\partial a_4(t-1)} = a_4(k) - \eta \frac{\partial j}{\partial e} \times \frac{\partial e}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial a_4} = a_4(k) + \eta e u^2(t-1)$$

$$a_5(t) = a_5(t-1) - \eta \frac{\partial j}{\partial a_5(t-1)} = a_5(k) - \eta \frac{\partial j}{\partial e} \times \frac{\partial e}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial a_5} = a_5(k) + \eta e u^2(t-2)$$

$$a_6(t) = a_6(t-1) - \eta \frac{\partial j}{\partial a_6(t-1)} = a_5(k) - \eta \frac{\partial j}{\partial e} \times \frac{\partial e}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial a_6}$$

$$\frac{\partial \hat{y}}{\partial a_6} = \frac{\partial f}{\partial a_6} + \underbrace{\frac{\partial f}{\partial \hat{y}(t-1)} \times \frac{\partial \hat{y}(t-1)}{\partial a_6}}_{BPTT}$$

The BPTT (Backpropagation through time) depends on the responses of the past samples, in another word, it is recursive. And the final equation for the algorithm is:

$$\frac{\partial \hat{y}(t-i)}{\partial a_6} = \frac{\partial f}{\partial a_6} + \frac{\partial f}{\partial \hat{y}(t-i-1)} \times \frac{\partial \hat{y}(t-i-1)}{\partial a_6}$$

And we continue upon reaching:

$$\hat{y}(0) = 0 \text{ or } \frac{\partial \hat{y}(0)}{\partial a_6} = 0$$

As is shown, “Backpropagation Through Time” happens, and updating parameters requires the previous steps’ values. This happens in recurrent networks, and we do the propagation for a few steps back and ignore the rest of the steps. This method of countering this issue is called truncated BPTT.

2)

The Hammerstein model is a static nonlinear model coupled with a linear dynamic system. In Figure 1 we can see the Hammerstein model:

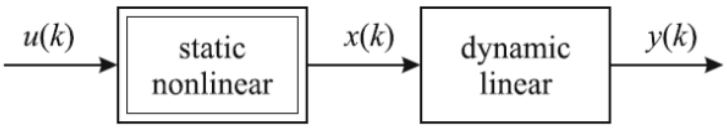


Figure 1 Hammerstein model

The system dynamics are as below:

$$x(k) = g(u(k))$$

$$y(k) = b_1 x(k-1) + \cdots + b_m x(k-m) - a_1 y(k-1) - \cdots - a_m y(k-m)$$

In this system, in order to prevent redundancy, we regularize the linear model system gain to be one, which constrains the system as below:

$$\frac{\sum_{i=1}^{m} b_i}{1 + \sum_{i=1}^{m} a_i} = 1$$

This structure alongside with other system structures, is suitable for systems with actuator nonlinearity and other nonlinear properties and aspects are dismissible. And this is the exact reason that these models are so useful in control engineering. And the nonlinear behavior can be compensated for with a controller in form of $g^{-1}(.)$. In these systems the stability is only determined by the linear component of the model.

The wiener model is demonstrated in Figure 2, which is the inversed Hammerstein model. Only a few processes are modeled using this model. One very prominent example of these systems is the PH titration. This structure is good for systems with sensor nonlinearity.

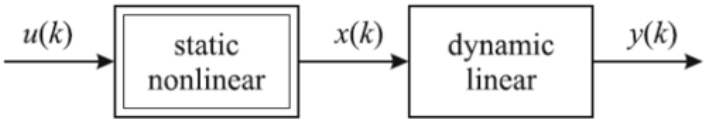


Figure 2 Wiener model

The equations of this model are:

$$x(k) = b_1 u(k-1) + \cdots + b_m u(k-m) - a_1 x(k-1) - \cdots - a_m x(k-m)$$

$$y(k) = g(x(k))$$

In a general sense, we can say that the Weiner model is used for systems that change in system dynamics happen as the operating point changes. And Hammerstein model is used when the operating point and consequently does the gain of the system.

3)

AIC (Akaike Information Criterion) is a measure of the relative quality of a statistical model for a given set of data. In essence, it is a tool used for model selection. AIC describes a trade-off between the goodness of fit and the complexity of the model. This statistic is based on the information dispersion.

When the selected model is used for generating data, it suggests an estimate dependent on lost information. AIC is not a test for the accuracy of the null hypothesis, for example, it cannot say anything about the absolute quality of the model. If all selected models have poor fits, AIC will not provide a warning in this regard.

Definition: In general, AIC is written as $AIC = 2K - 2\ln L$, where K is the number of parameters in the statistical model, and L is the maximum likelihood function of the model.

When comparing multiple candidate models for a set of data, AIC is a statistic that refers to having the lowest AIC value. However, AIC alone does not indicate the goodness of fit of the model. It includes an error term that increases with the number of estimated parameters, serving as a kind of barrier against overfitting.

Increasing the number of free parameters in the model improves the goodness of fit, regardless of the number of free parameters in the data generation processes.

Execution Method:

To execute AIC, we start with a set of candidate models and then find the AIC values for each model. In the process of selecting a model, some amount of information is typically lost in describing the true model, and our preference is to choose a model that minimizes the lost information. While we cannot make a certain choice, we can minimize the loss of information.

Let $AIC_1, AIC_2, \dots, AIC_i, \dots AIC_n$ be the AIC values for n selected models. The goal is to minimize these values. Then, the model with the minimum AIC value represents the minimum information loss for that specific model.

For example, suppose we have three models with AIC statistics as follows: 102, 100, and 110. Therefore, for the second model, $e^{\left(\frac{100-102}{2}\right)} = 0.368$, indicating a higher probability of minimizing information loss compared to the first model. For the third model, $e^{\left(\left(\frac{100-110}{2}\right)\right)} = 0.007$, suggesting a lower probability compared to the first model for minimizing information loss.

In this example, we can exclude the third model from further consideration. Now, we have three possibilities:

1. We can collect more data and hope that this may help clarify the difference between the first two models.
2. We can easily conclude that the existing data is sufficient to support model selection.

3. We can choose a weighted average for the first two models, e.g., 1 and 0.368, and perform statistical inference based on a weighted multiple-model approach.

The relative likelihood for model i represents the probability associated with that model. If all models under consideration have the same number of parameters, AIC may appear very similar to the likelihood ratio test. In particular, the likelihood ratio test is valuable only for nested models, while AIC does not have this limitation. When dealing with small sample sizes (n) or large numbers of parameters (K), the following correction is often applied.

This correction is preferred when the sample size (n) is small, or the number of parameters (K) is large. The connection with the Kullback-Leibler divergence is commonly utilized.

Regarding the choice of models, we often prefer a model that assumes the errors to be normally distributed (with a mean of zero) and independent of each other among several models. This assumption facilitates the fit of the model. For fitting the Kullback-Leibler divergence, the likelihood function is applied.

Bayesian Information Criterion (BIC)

The Bayesian Information Criterion (BIC) is one of the tools and metrics used for evaluating models created based on statistical inference. There are various tools and metrics for this purpose. BIC is an evaluation criterion that determines the amount of information lost by the model. BIC, which stands for Bayesian Information Criterion, is calculated based on the likelihood function and has a close relationship with the Akaike Information Criterion (AIC).

Usually, by increasing the parameters of a model and making it more complex, we can increase the likelihood function, indicating a better fit to the data and validation of the presented model. However, this may lead to overfitting issues, where the credibility of the model is compromised. Therefore, the use of criteria such as BIC, which considers both the likelihood and the number of parameters and observations, becomes crucial. Thus, considering the impact of both the number of parameters and observations, along with the likelihood function, BIC evaluation criterion becomes a significant indicator for model adequacy.

As we will see later, the BIC evaluation criterion, like AIC, reflects the amount of information lost by the model. Therefore, a smaller BIC value suggests that the model is better and more suitable compared to other models. Gideon Schwarz introduced the BIC evaluation criterion in articles published in 1978.

The calculation of BIC is similar to AIC, with the difference that the penalty for the number of parameters in the model is higher. Suppose we denote the maximum likelihood function for a statistical model as L, the number of parameters as k, and the number of observations as n. In this case, the BIC criterion for this model is calculated as follows.

$$BIC = kln(n) - 2ln(\hat{L})$$

Characteristics of the BIC Evaluation Criterion:

Although the calculation of the BIC evaluation criterion may seem complex, its characteristics can be listed as follows:

- BIC is independent of the prior distribution. Therefore, evaluation using BIC is possible without considering the prior distribution for parameters.
- BIC evaluates the model's performance considering both the number of parameters and the predictive power of the data.
- The number of parameters and observations play a role in BIC calculation, penalizing the likelihood function based on these two characteristics to prevent overfitting.
- BIC can be used to determine the appropriate number of clusters in partitional clustering algorithms.
- BIC has a significant relationship with the AIC evaluation criterion.
- If the number of observations is much larger than the number of parameters, using BIC is appropriate, and the approximation mentioned in the calculations above will be more effective.
- BIC cannot be used for feature analysis and selection in large-dimensional datasets.

4)

The first method for identifying time delays is presented in the context of a paper titled:

"TIME-VARYING TIME-DELAY ESTIMATION FOR NONLINEAR SYSTEMS USING NEURAL NETWORKS."

Most industrial systems exhibit time delays, making the identification of time delays one of the crucial topics in system modeling and identification. Over the past decade, it has been established that neural networks provide a very suitable approach for modeling systems and approximating the response function. In this context, a dynamic neural network is considered, which has an input layer, a recurrent layer representing external recurrent connections, and outputs from the network.

Time delays in some industrial processes are time-varying. In such cases, online identification of time delays becomes necessary. For modeling such nonlinear processes, two methods based on neural networks are used. These two methods are:

1. Indirect Time-Delay Identification Method
2. Direct Time-Delay Identification Method

The following, details each of these methods.

Indirect Time-Delay Identification Method:

Consider a process described as a mapping $f: R^{m+n} \to R$, for example:

$$y_k = f\left(Y_{k-1}, U_{k-\tau_k}\right)$$

That $f(\cdot) \in C^2, Y_{k-1} = [y_{k-1}, \dots, y_{k-n}]^T \in R^n$ and $U_{k-\tau_k} = \left[u_{k-\tau_k-1}, u_{k-\tau_k-2}, \dots, u_{k-\tau_k-m}\right] \in R^m$ are output and input vectors and τ_k is time delay. Take the model as a Neural network and the process is formulized as below:

$$\hat{y}_k = W^{2T} S(x)$$

The output of the model is delayed and $W^2 = [w_1^2, w_2^2, \dots, w_h^2]^T \in R^h$ is weighted vector that connects the output of middle layer to the output layer. $S(x) = [s(x_1), \dots, s(x_h)]^T \in R^h$ is the output of the middle layer. In each neuron of the middle layer the below function is used:

$$s(x_i) = \frac{1 - e^{-x_i}}{1 + e^{-x_i}}$$

Which is a sigmoid function, and its inputs in the intermediate layer are expressed as follows:

$$x_i = \sum_{j=1}^{n_a} w_{ij}^1 \hat{y}_{k-j} + \sum_{j=1}^{n_b} w_{i,n_a+j}^1 u_{k-\tau_k-j} \; j = 1, \dots, h$$

In the above equation, $\hat{\tau}_k$ represents the estimated time delay, and n_a and n_a are the respective orders of the input and output time delays. ω_{ij} denotes the weights connecting inputs to nodes in the intermediate layer. Introducing auto-regression of the model output to the network can be useful in simulating the dynamics of the process.

Now, let's assume a scenario where the time delay is time-varying. It is assumed that the time delay can be decomposed into two parts, an integer part and a fractional part. For example:

$$\hat{\tau}_k = \hat{d}_k + \delta\hat{\tau}_k$$

Where $\hat{d}_k$ represents the integer part of the time delay, and $\delta\hat{\tau}_k$ denotes the fractional part of the time delay. It is assumed that the fractional part of the time delay should fall within the range of one sampling period. The assumption is made that the rate of change of the time delay is much smaller than the sampling rate, so the time delay remains constant during one sampling period.

To identify the time delay, the fractional part is considered as a parameter that needs to be estimated. Consequently, the integer part of the time delay can be refined based on the estimation of the fractional part. To estimate the fractional part of the time delay, the gradient of the neural network output with respect to $\delta\hat{\tau}_k$ needs to be calculated.

$$\frac{\partial\hat{y}_k}{\partial\hat{\tau}} = \frac{\partial\hat{y}_k}{\partial\delta\hat{\tau}} = \sum_{i=1}^{h} w_i^2 s'(x_i)\left(\sum_{j=1}^{n_a} w_{ij}^1 \frac{\partial\hat{y}_{k-j}}{\partial\delta\hat{\tau}} + \sum_{i=1}^{n_a} w_{i,n_a+j}^2 \frac{\partial u_{k-\hat{\tau}_k-j}}{\partial\delta\hat{\tau}}\right)$$

Which s' denotes:

$$s'(x) = \frac{ds(x)}{dx} = 0.5(1 - s(x)^2)$$

By using first-order interpolation, we can estimate $\frac{\partial u_{k-\hat{\tau}_k-j}}{\partial\hat{\tau}}$ Taylor expansion leads to:

$$u_{k-\hat{\tau}_k} \approx u_{k-\hat{d}_k-1} + \delta\hat{\tau}\frac{u_{k-\hat{d}_k} - u_{k-\hat{d}_k-1}}{k - \hat{d}_k - \left(k - \hat{d}_k - 1\right)} = u_{k-\hat{d}_{k-1}} + \delta\hat{\tau}\left(u_{k-\hat{d}_k} - u_{k-\hat{d}_k-1}\right)$$

Then we have:

$$\frac{\partial u_{k-\hat{t}_k-j}}{\partial\delta\hat{\tau}} = u_{k-\hat{d}_k-j} - u_{k-\hat{d}_k-j-1}$$

Additionally, the gradient of the neural network output with respect to the weights is equal to:

$$\frac{\partial\hat{y}_k}{\partial w_i^2} = s(x_i),\ i = 1, \dots, h$$

$$\frac{\partial\hat{y}_k}{\partial w_i^2} = s(x_i),\ i = 1, \dots, h$$

$$\frac{\partial\hat{y}_k}{\partial w_{ij}^1} = \sum_{i=1}^{h} w_i^2 s'(x_i)\left(\sum_{j=1}^{n_a} w_{ij}^1 \frac{\partial\hat{y}_{k-j}}{\partial w_{ij}^1} + \hat{y}_{k-j}\right),\ j = 1, \dots, n_a$$

$$\frac{\partial\hat{y}_k}{\partial w_{ij}^1} = \sum_{i=1}^{h} w_i^2 s'(x_i)\left(\sum_{j=1}^{n_{ij}} w_{yj}^1 \frac{\partial\hat{y}_{k-j}}{\partial w_{ij}^1} + \hat{u}_{k-i_k-j}\right),\ j = n_a + 1, \dots, n_b$$

The cost function is defined as follows:

$$Q = 0.5e_k^2 = 0.5(y_k - \hat{y}_k)^2$$

The parameter matrix is expressed as follows:

$$\theta = [w_1^2, w_2^2, \dots, w_h^2, \delta\hat{\tau}]^T$$

$$\omega = \left[w_{ij}^1\right]_{(h)\times(n_a+m_b+q)}^T$$

The estimation of these matrices will be as follows:

$$\theta_k = \theta_{k-1} - \lambda_1 \frac{\partial Q}{\partial \theta_{k-1}}$$

$$\omega_k = \omega_{k-1} - \lambda_2 \frac{\partial Q}{\partial \omega_{k-1}}$$

Which $\lambda > 0$ and $i = 1, 2$ are optimization steps. If we utilize a second-order optimization algorithm, such as the Modified Levenberg-Marquardt method, the parameter matrices will be updated as follows:

$$\psi(k) = \psi(k-1) - \lambda[H_e + \alpha I]^{-1} \frac{\partial Q}{\partial \psi} + \beta(\psi(k-1) - \psi(k-2))$$

Which ψ is equal to generalized parameter of the neural network and H_e is the Hessian matrix:

$$H_e = \left(\frac{\partial \hat{y}_k}{\partial \psi}\right)\left(\frac{\partial \hat{y}_k}{\partial \psi}\right)_{\psi=\theta}^T = \left(\frac{\partial \hat{y}_k}{\partial \theta}\right)\left(\frac{\partial \hat{y}_k}{\partial \theta}\right)^T$$

Or

$$H_e = \left(\frac{\partial \hat{y}_k}{\partial \psi}\right)\left(\frac{\partial \hat{y}_k}{\partial \psi}\right)^T{}_{\Psi=\omega} = \left(\frac{\partial \hat{y}_k}{\partial \omega}\right)\left(\frac{\partial \hat{y}_k}{\partial \omega}\right)^T$$

Where $\alpha > 0$, a tunable factor within the range of $(0,)$. In the beginning, it is chosen to be sufficiently large to ensure the positivity of the approximation with the Hessian. In this case, the algorithm transforms into the Steepest-Descent method. Then, the value of α must decrease towards zero at each stage. If alpha becomes zero, the Gauss-Newton algorithm will be obtained. Additionally, alpha has the capability of stabilization in such a way that the algorithm converges towards the saddle point. In this case, the Hessian matrix becomes zero and $\alpha > 0$ improves numerical stability. To enhance the ability to escape from local minima, a gradient term has been added to the algorithm, where $\beta > 0$ is the gradient factor. When the estimate is obtained, both the integer and fractional parts will be updated with a delay as follows:

$$\begin{cases} \hat{d}_{k+1} = \hat{d}_k - 1 \\ \delta\hat{\tau}_{k+1} = \delta\hat{\tau}_k + 1 \end{cases} \text{ for } \quad \delta\hat{\tau}_k \in (-\infty, k + \eta]$$

Or

$$\begin{cases} \hat{d}_{k+1} = \hat{d}_k + 1 \\ \delta\hat{\tau}_{k+1} = \delta\hat{\tau}_k - 1 \end{cases} \text{ for } \quad \delta\hat{\tau}_k \in [k+1+\eta, \infty)$$

Or

$$\begin{cases} \hat{d}_{k+1} = \hat{d}_k \\ \delta\hat{\tau}_{k+1} = \delta\hat{\tau}_k \end{cases} \text{ for } \quad \delta\hat{\tau}_k \in (k+\eta, k+1+\eta), 0 < \eta < 1$$

η is a very small number. Since the indirect method of delay estimation for a non-linear online scheduling process, it will have a high computational load for this reason.

A direct time delay estimation method:

A dynamic neural network is designed directly for identifying delays. The performance of the estimator depends on the characteristics of the weights $\boldsymbol{V}$, the order of the inputs, and the number of neurons in the intermediate layer of the network. In this section, time delay estimation is formulated as an identification process. Similar to before, it is assumed that the error can be divided into two parts, integer and fractional.

$$\hat{\tau}_k = \hat{d}_k + \delta\hat{\tau}_k$$

For time delay estimation, the fractional part of the estimation is considered as follows:

$$\delta\hat{\tau}_k = g(V, I_k)$$

Which function $g(.) \in C^2$ and apply the mapping $g: R^q \to R$. $q = q_1 + q_2$, q_1 and q_2 are equal to the orders of the fractional part of the time delay sequence $\{\delta\hat{\tau}_k\}$ and the sequence of differences between the system outputs and the model is $\{e_k\}$. V is the weight matrix, and I_k is the input vector of the time delay neural estimator

$$I_k = \left[\delta\hat{\tau}_{k-1}, \dots, \delta\hat{\tau}_{k-q_1}, e_{k-1}, \dots, e_{k-q_2}\right]^T$$

Which $e_k = y_k - \hat{y_k}$. The equation below by using a neural network:

$$\delta\hat{\tau}_k = \sum_{i=1}^{H} v_i^2 s(\bar{z}_i) = \sum_{i=1}^{H} v_i^2 s\left(\sum_{j=1}^{q} v_{ij}^1 I_{k-j}\right)$$

Where H is the number of neurons in the intermediate layer, v_i^2 represents the weight connecting the ith neuron in the intermediate layer to the output of the neural network. Meanwhile, v_{ij}^1 represents the weight connecting the jth input of the network to the ith neuron in the intermediate layer. To find the mentioned weights, the derivative of $\delta\hat{\tau}_k$ with respect to v_i^2 and v_{ij}^1 is calculated as follows

$$\frac{\partial\delta\hat{\tau}_k}{\partial v_i^2} = s(z_i) \quad i = 1, \dots, H$$

$$\frac{\partial\delta\hat{t}_k}{\partial v_{ij}^1} = \sum_{i=1}^{H} v_i^2 s'(z_i)\left(I_{k-j} + \sum_{j=1}^{q_1} v_{ij}^1 \frac{\partial\delta\hat{\tau}_{k-j}}{\partial v_{ij}}\right) \quad j = 1, \dots q$$

The gradient of the output with respect to the neural network weights is calculated using equations provided before. The weights are adjusted using the Levenberg-Marquardt method. The matrix of corresponding parameters for the neural network is as follows:

$$\theta = [v_1^2, \dots, v_H^2, \delta\hat{\tau}]^T, \qquad \omega = \left[v_{ij}^1\right]_{H\times q}^T$$

Then, based on the results of estimating the fractional part of the time delay, both the integer and fractional parts of the time delay are adjusted separately.

Usually, the order of input variables is determined based on empirical knowledge. Then, the number of neurons in the intermediate layer of the neural network is determined based on a criterion function. The criterion function should be at least dependent on the weight matrix.

$$J(H, V_H) = \sum_{t=1}^{N} [y(\mathrm{t}) - \hat{y}\left(t, \hat{d} + \delta\hat{\tau}\right)]\text{^}2$$
$$J(H) = \min_{V_H} \sum_{t=1}^{N} [y(t) - \hat{y}(t, \hat{d} + g(H, I))]^2$$

While N represents the number of optimization epochs, based on the theory of feedforward neural networks with multiple layers, it is quite clear that:

$$J(H) \geq J(H+1)$$

In the theoretical scenario, with an increase in H, the number of optimal neurons in the intermediate layer of the neural network time-delay estimator is obtained. For example, if $H = H^*$ then we have:

$$J(H-1) \geq J(H^*) \approx J(H^*+1)$$

From all the discussions and practical examples, it can be observed that the computational burden of the direct estimation method is heavier than the indirect method. While the direct method can be used in both online and offline scenarios, the training process can be halted when the neural network has been trained well. Considering the online scenario, the direct method has a lower computational cost compared to the indirect method.

Second Delay Identification Method:

Here, delay identification is specialized in finding the parameter T in the following equation:

$$y(t) = u(t-T)$$

u(t) and y(t) represent the input and output, respectively.

The analyses we want to perform are based on two assumptions:

$$(1) \quad u(t) = 0,\ t < 0$$

Taking assumption (1) into account and applying the Laplace transform, we have:

$$Y(s) = e^{-sT} U(s)$$

This means that $U(s)\ and\ Y(s)$ are the Laplace transforms of $y(t)$ and $u(t)$, respectively.

Assumption (2) is formulated as follows: the transform function is equivalent to:

$$(2) \quad Y(s) \cong \frac{1}{\left(1+\frac{sT}{N}\right)^N} U(s)$$

The right-hand side of the above equation is a well-known approximation for time delay.

Specifically, for each constant value of s, if $N \to \infty$, the right-hand side of the equation tends to e^{-sT}. And for constant N, the left-hand side of (2), if $sT \to \infty$, tends to e^{-sT}.

$$C(s) = \sum_{i=0}^{N} c_i s^i$$

The approximated model for the system becomes as follows:

$$\frac{\left(1+\frac{sT}{N}\right)^N}{C(s)} Y(s) \cong \frac{1}{C(s)} U(s)$$

Lemma: With the approximations of (1) and (2) hold, the above system can be written in the following implicit form:

$$g(t,T) = e(t)$$

Where $e(t)$ is the approximation error and

$$g(t,T) = \sum_{i=0}^{N} \alpha_i T^i y_i(t) - u_0(t)$$

$$y_i(t) = L^{-1}\left[\frac{s^i}{C(s)} Y(s)\right] \ (i = 0, \dots, N)$$

$$u_0(t) = L^{-1}\left[\frac{1}{C(s)} U(s)\right]$$

$$\alpha_i = \frac{N!}{i!\,(N-i)!\,N!}$$

The values of y_i can be generated using the following state-space filter equations.

$$y_N = \frac{1}{c_N}\left[y - \sum_{i=0}^{N-1} c_i y_i\right]$$

$$\frac{dy_i}{dt} = y_{i+1} \ (i = 0, \dots, N-1)$$

The identification problem can be formulated as follows.

For each final time in the interval $[0, t_e]$, and for each input-output pair in the interval $[0, t]$, we estimate the optimal parameter θ, denoted as $\hat{\theta}$ in such a way that the weighted exponential cost function is minimized.

$$J(t,\theta) = \frac{1}{2}e^{-\beta t}\bar{J}(\theta - \theta_0)^2 + \frac{1}{2}\int_0^t e^{-\beta(t-\tau)}g(\tau,\theta)^2 d\tau$$

$\beta > 0$ Exponential weighting coefficient, θ_0 is the initial estimation and $\bar{J}$ is the initial positive weight. This problem is solvable in the following way:

The theorem states: If the optimal estimate T, i.e. $\bar{\theta}$, exists and is unique for each T, then the following system of differential equations holds:

$$J_2 \frac{d\hat{\theta}}{dt} + h_1 = 0$$

$$\frac{dJ_i}{dt} + \beta J_i = h_i + J_{i+1}\frac{d\hat{\theta}}{dt} \ (i = 2, \dots, 2N)$$

$$h_i = \sum_{j=0}^{i} \frac{i!}{j!\,(i-j)!} g_i g_{i-j}$$

$$g_i = \sum_{j=i}^{N} \alpha_j \frac{i!}{(i-j)!} y_i \hat{\theta}^{j-i}$$

$$J_{2N+1} = 0$$

And the only non-zero initial values are:

$$J_2(0) = \bar{J}$$

$$\hat{\theta}(0) = \theta_0$$

Scaler values J_i, g_i and h_i are:

$$J_i = \frac{\partial^i J}{\partial \theta^i}$$

$$g_i = \frac{\partial^i g}{\partial \theta^i}$$

$$h_i = \frac{1}{2}\frac{\partial^i g^2}{\partial \theta^i}$$

This is to check the existence of a local minimum, in addition to$\hat{\theta}(0) = \theta$ (where θ is the same as T).

Choosing algorithm parameters:

To work with this algorithm, you need to first choose five parameters:

- Order N of the approximation
- Polynomial degree of filter $C(s)$
- Exponential weighting factor β
- Initial value of the cost function J
- Initial value of the time delay estimation θ_0

Since this algorithm has been recently developed, there are few quantitative guidelines for selecting these parameters.

Identification of nonlinear dynamics:

"Determining the Model Order of Nonlinear Input/Output Systems Directly From Data"

The method used here for direct determination of the dynamics of nonlinear systems is the False Nearest Neighbors (FNN) method. A unique aspect of this method is that it does not make any assumptions about the structure of the system model.

Modeling Time Delay Coordinates:

Let's assume that we have a nonlinear dynamical system described by the following equations:

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x})$$

$$y(t) = g(\mathbf{x}(t))$$

So that $y \in R$ is measurable, and $x \in R^n$

According to a theory that has been proven in numerous articles, there exists an equality relationship in another coordinate set as follows:

$$y(t) = F[y(t-\tau), y(t-2\tau), \dots, y(t-l\tau)]$$

This new set of coordinates is often referred to as time delay coordinates if the number of time delays, denoted by l, is sufficiently large to ensure the reconstruction of the dynamics. Here, we consider the general case of the input-output system in the following form:

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, u)$$

$$y(t) = g(\mathbf{x}(t))$$

$$u \in R$$

If u changes only at the sampling time, we assume the following relationship:

$$y(t) = F[y(t-\tau), \dots, y(t-l\tau), u(t-\tau), \dots, u(t-m\tau)]$$

From this relationship, we define two distinct spaces: the prediction space y(t) and the time-delay space.

$$(y(t-\tau), \dots, y(t-l\tau), u(t-\tau), \dots, u(t-m\tau)) \in \Re^l \times \Re^m.$$

False Nearest Neighbors Algorithm

The FNN algorithm for input-output systems is presented below. This algorithm relies on the following reality: if the dimension of the time-delay space is very small to represent the system's dynamics, points that are close in the time-delay space may be far in the prediction space. On the other hand, if the dimension of the time-delay space is sufficiently large, points close in the time-delay space will always be close in the prediction space.

FNN algorithm:

1. Find the point within the dataset in such a way that the distance is minimized.

$$\mathbf{r}_{l,m}(k) = [y(k-\tau), \dots, y(k-l\tau), u(k-\tau), \dots, u(k-m\tau)]$$

Find the point $r_{i,m}(k)$ from the set of the points in a way that minimizes the distanced.

$$d = \|\mathbf{r}_{l,m}(k) - \mathbf{r}_{l,m}(j)\|_2$$

2. Determine if the equation below is true or false.

$$\frac{|y(k) - y(j)|}{\|\mathbf{r}_{l,m}(k) - \mathbf{r}_{l,m}(j)\|_2} \leq R$$

R is a predetermined threshold. If true then “true neighbor”, if false “false neighbor”.

3. Continue the algorithm for the points k in the dataset and calculate the percentage of points in the dataset that have false nearest neighbors.
4.
5. Continue the algorithm by increasing l and m until the percentage of false nearest neighbors approaches zero.

The method starts by calculating the Lyapanuv exponents. The number of dynamics before the curve breaks, D, is determined. If the system has a delay, D, is not indicative of the correct number of input dynamics. Therefore, irrelevant dynamics need to be removed to obtain the appropriate number. Since our concern is estimating the delay in the chosen models, for this purpose, by having a set of D input dynamics, calculations are inverted. In fact, the inputs are swept from the beginning to the end, and one by one is removed. (For a more detailed study, refer to the article.) Based on the idea of Lyapunov, if an input is not related to the system, removing it does not significantly change the Lyapunov exponent. However, if the removed input is one of the main dynamics of the

model, this value increases significantly. As a result, by identifying the first significant increase in these values, the model's delay is easily determined.

To clarify the example, let's assume you have the following system with time delays of 5.2 and 5.9:

$$\dot{y}(t) = -y(t) - y^3(t) + \frac{1}{9}x^2(t-5.2) - \sin(x(t-5.9))$$

First, we calculate the Lipschitz number for the first 30 dynamics and plot the graph.

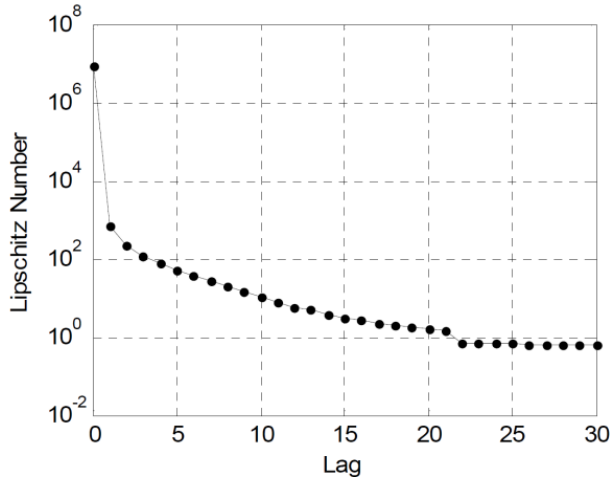


Figure 3 Lyapunov exponents for the first 30 dynamics

It is observed that, from the 20th dynamic onward, adding dynamics does not significantly change the Lipschitz number values. Therefore, the value of D is set to 20. In the next step, to ensure the existence of delay for the selected 20 dynamics, we calculate the Lipschitz number from the end to the beginning and plot the corresponding graph.

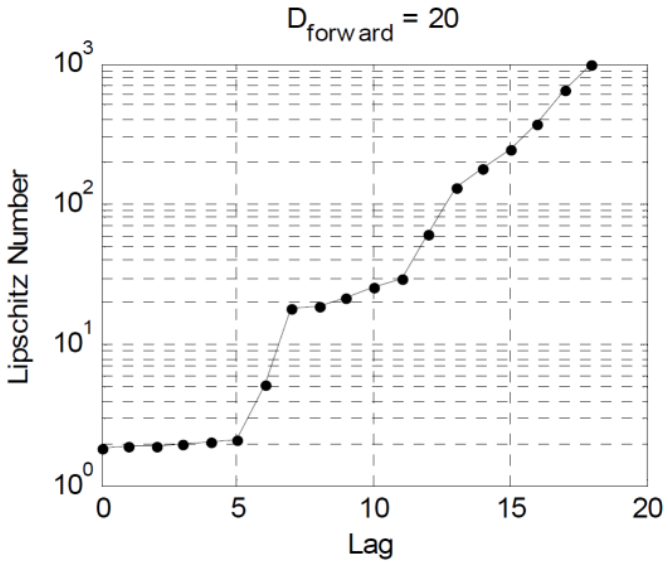


Figure 4 Lipschitz number backwards

Given the jump in the graph above in the fifth sample, the estimated delay is approximately 5, which is close to the actual delay of the system, which is 5.2.

Simulation Questions

Question 1

In this first section, we will examine the second system, the steam generator, and then analyze the first system, the ball-and-beam setup.

It has been mentioned that some hyperparameters have been obtained through trial and error, and if not specified, it means the same.

The percentage fit plots, drawn with the compare command, and the correlation of errors are related to the test data.

1.1) system 2

1.1) The input and output data plot is as follows.

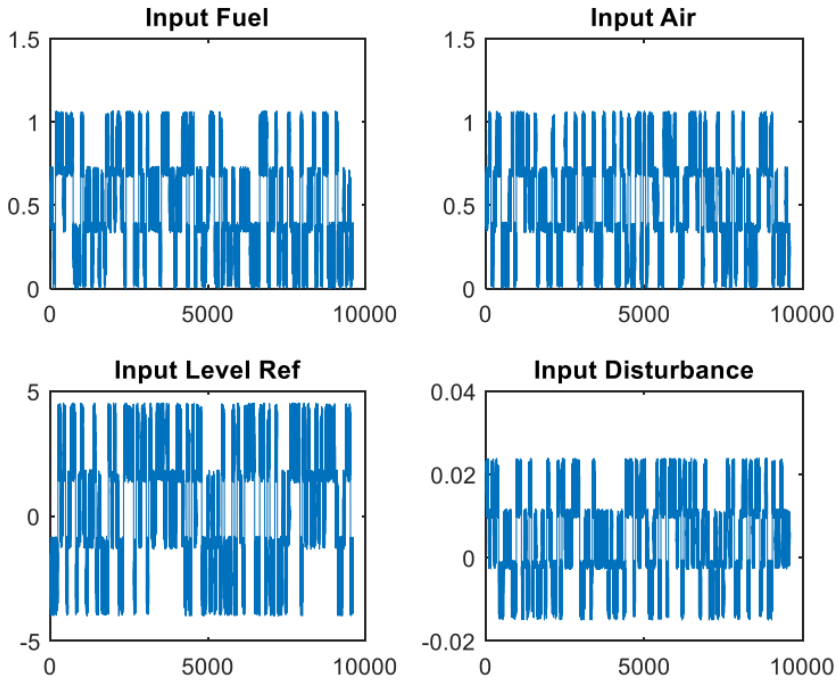


Figure 5 Input data of the system

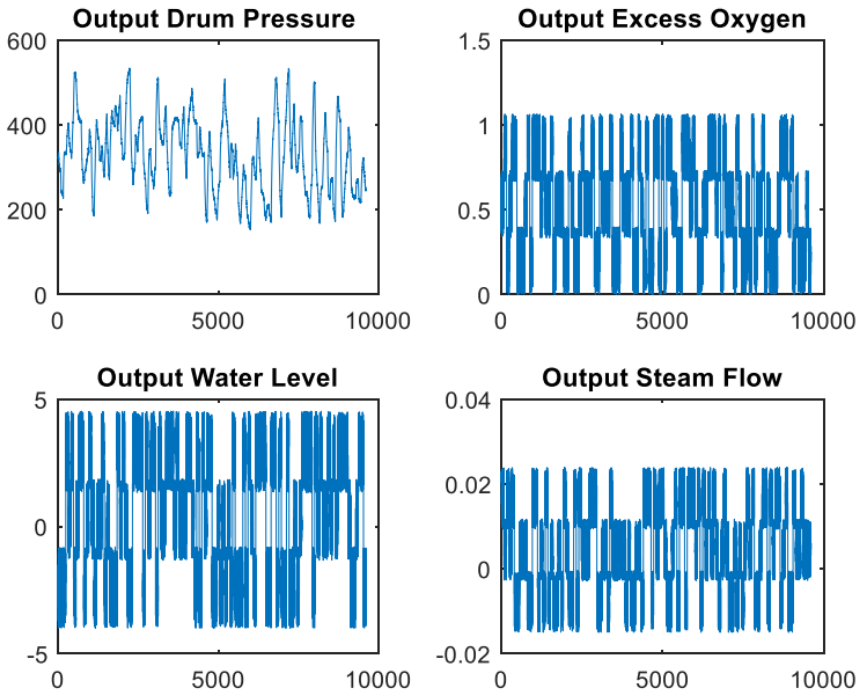


Figure 6 the output plot

As evident from Figures 5 and 6, the data is not in the same range, and by normalizing them, we bring all the data between 0 and 1. We allocate 15% of the data for validation (1440 data points), 20% of the data for testing (1920 data points), and the rest is considered as training data (6240 data points).

Lipschitz method

The Lipschitz method is a direct approach to obtaining a first-order model based on the input and output data of a system. This method relies on the Lipschitz theory, which states that a continuous mapping has a bounded gradient band that can be maximized at known points. Therefore, when the relationship between input and output in a mapping is smooth, an assumption that is mandatory in identifying nonlinear systems as black-box models, this idea can be applied.

In the Lipschitz method, numbers known as Lipschitz constants are calculated based on the following coefficient:

$$L_{ij}^{n} = \frac{|y(i) - y(j)|}{\sqrt{(u_1(i) - u_1(j))^2 + \cdots + (u_n(i) - u_n(j))^2}}$$

In this equation, y is the output of the system, and u_l's are its potential inputs with dynamics up to the $l - th$ order. Lipschitz constants are calculated from the L_{ij}^{n} coefficients:

$$L^n = \left(\prod_{b=1}^{p} \sqrt{n} L^n(k)\right)^{\frac{1}{p}}$$

In this equation, the $k-th$ coefficient is the largest among the L_{ij}'s. The parameter p is approximately one to two percent of the data size. In cases where the data is not very noisy, choosing p = 1 is a suitable option. As long as the number of dynamics is very small, and hence not all main inputs are included, the Lipschitz value decreases as more inputs are added. Once all relevant inputs have been taken into account, this decreasing trend will be interrupted, and adding more inputs will not cause a significant reduction. This point is referred to as the "breakpoint."

In this method, the output dynamics are not considered because the goal is to find the input delay and its associated dynamics, and ignoring the output dynamics does not introduce any error in reaching the correct answer.

In this approach, Lipschitz constants are calculated first, and the number of dynamics before the breakpoint is determined (denoted by D_0). If the system has a delay, D_0 does not represent the integer number of input dynamics, so irrelevant dynamics are removed to find the appropriate number. After determining the breakpoint, dynamics before it are discarded, and an inverse operation is performed. Dynamics are reduced one by one, and the Lipschitz constant is calculated for the remaining set. For example, the Lipschitz constant is calculated as follows:

$$LB_{ij}^{d} = \frac{|y(i) - y(j)|}{\sqrt{(u_d(i) - u_d(j))^2 + \cdots + \left(u_{D_0}(i) - u_{D_0}(j)\right)^2}}$$

$$LB^{d} = \left(\prod_{k=1}^{p} \sqrt{D_0 + d - 1} LB^{d}(k)\right)^{\frac{1}{p}}$$

According to the Lipschitz idea, if an input is not suitable, removing it does not significantly change the Lipschitz number. However, if the removed input is one of the main dynamics of the model, this value increases noticeably. Therefore, by identifying the first sharp increase in these values, the model delay can be easily determined.

The system inputs are such that in some samples, the difference between two inputs is zero. The presence of input in the denominator causes the Lipschitz numbers to become infinite. To address this issue, either a very small noise should be applied to the inputs, or the data should be resampled. In this case, a resampling ratio of 1 to 7 has been considered based on trial and error.

For example, initially, additional dynamics are removed with the help of Figure 8. Referring to this plot, for the first input and output, the Lipschitz number for dynamics 22 and beyond is almost constant. By excluding additional dynamics, the process proceeds by decrementing one dynamic at a time, and the Lipschitz number is calculated. Figure 24 illustrates the delay between the first input and output. Based on these plots, it is observed that the first sharp increase for the first input and output is equal to 14. Therefore, according to this method, the delay between the first input and output is 14. The dynamic depth is also obtained from the difference between these two values, which is 8 for the first input and output. The delay and dynamic depth values for different inputs and outputs are shown in Table 1.

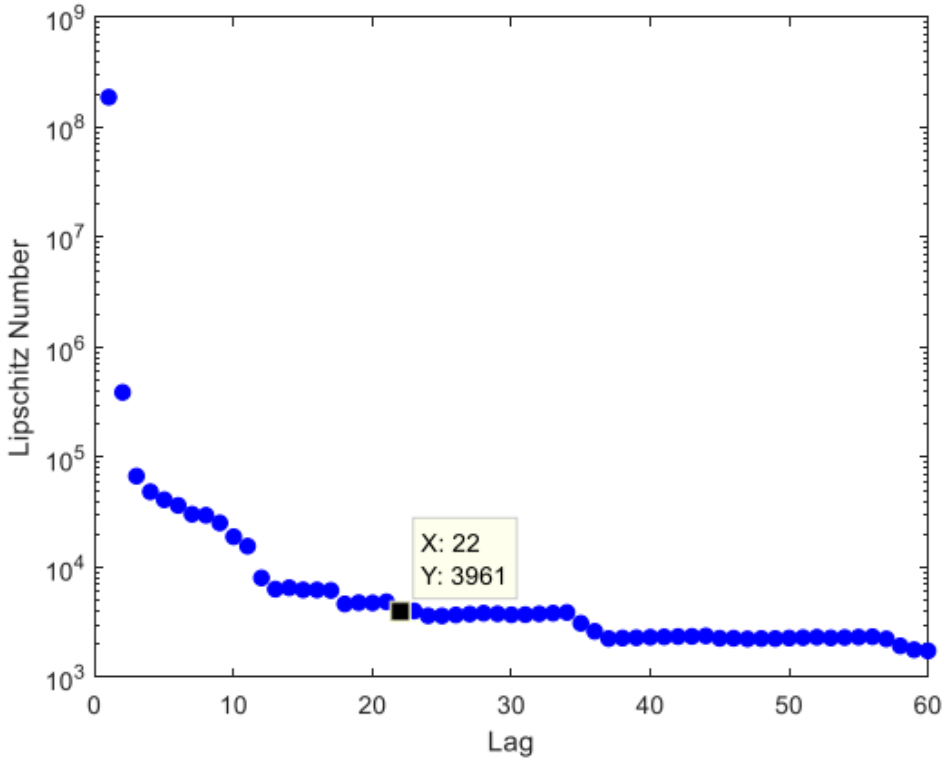


Figure 7 Lipschitz for 60 dynamics from the first input and the first output (steamgen)

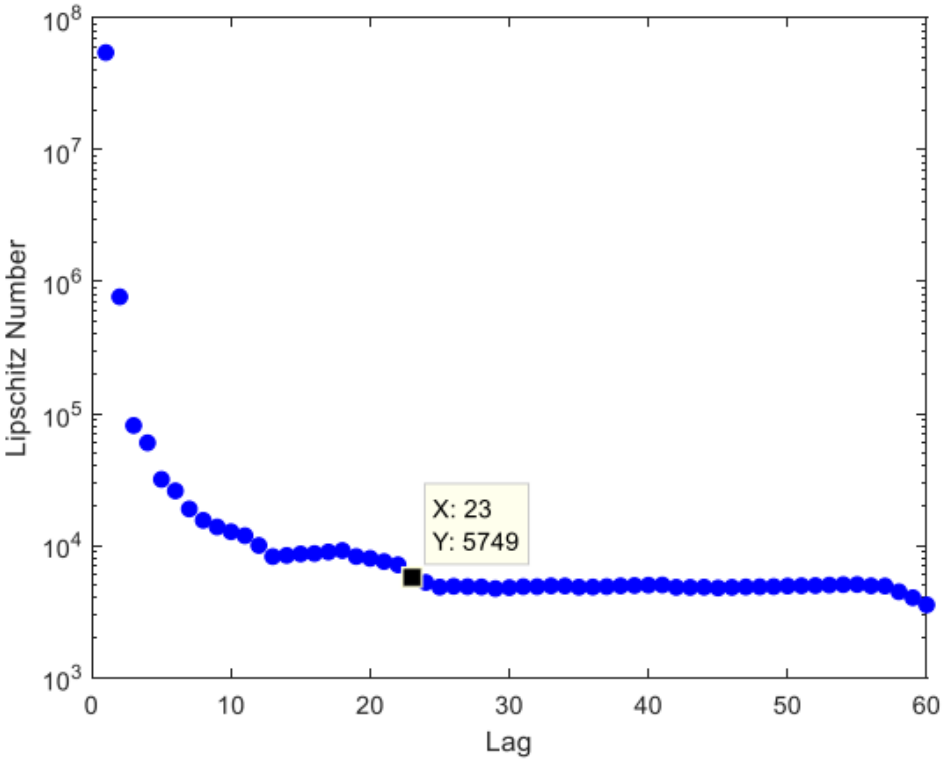


Figure 8 Lipschitz for 60 dynamics from the second input and the first output (steamgen)

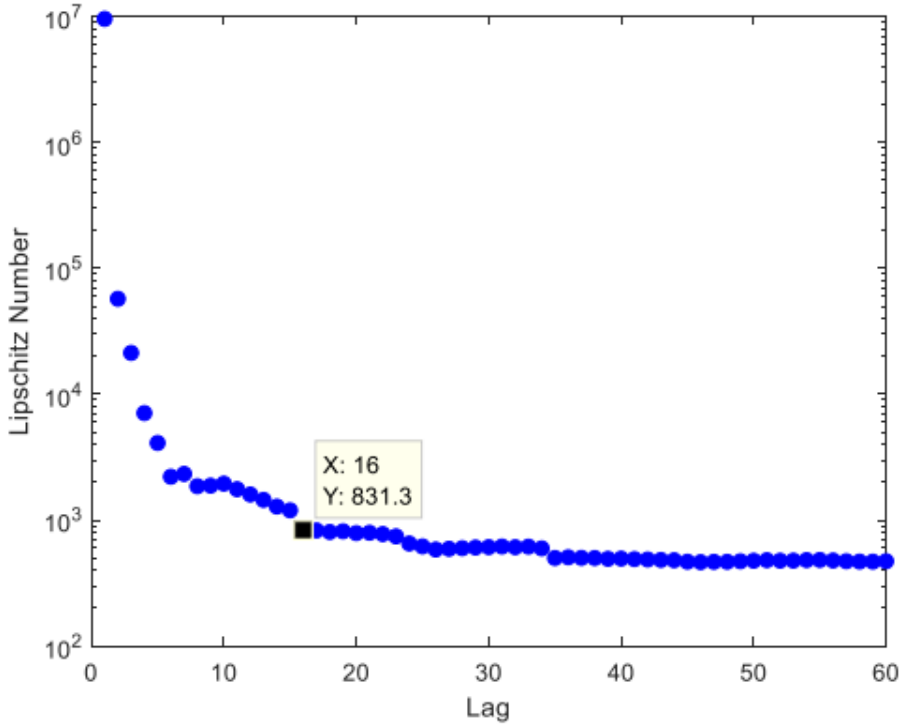


Figure 9 Lipschitz for 60 dynamics from the third input and the first output (steamgen)

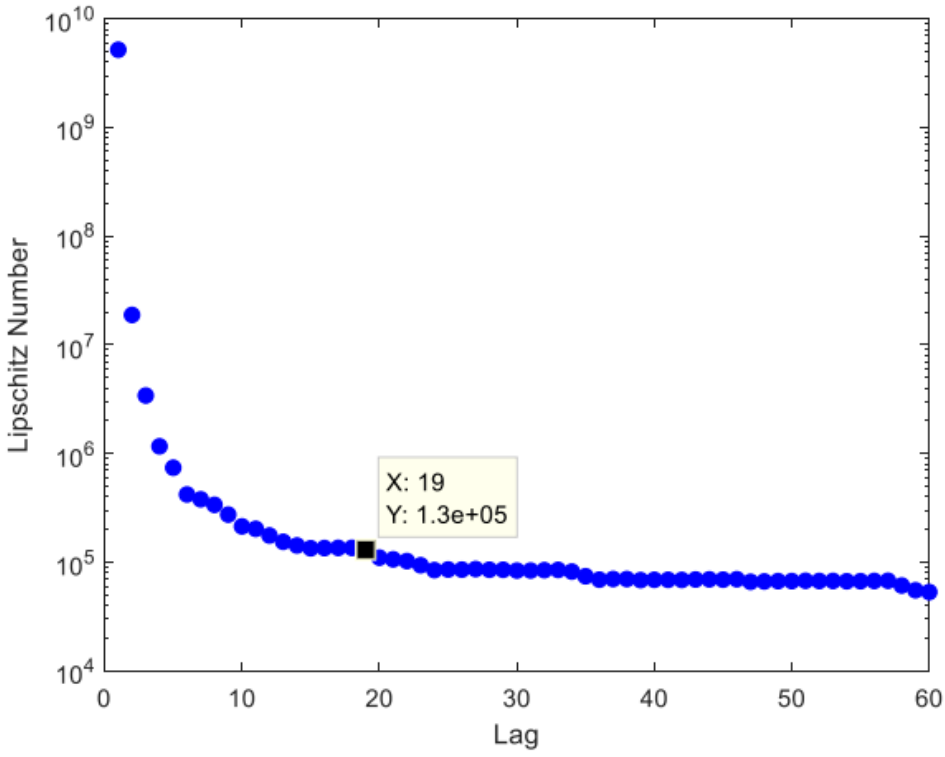


Figure 10 Lipschitz for 60 dynamics from the fourth input and the first output (steamgen)

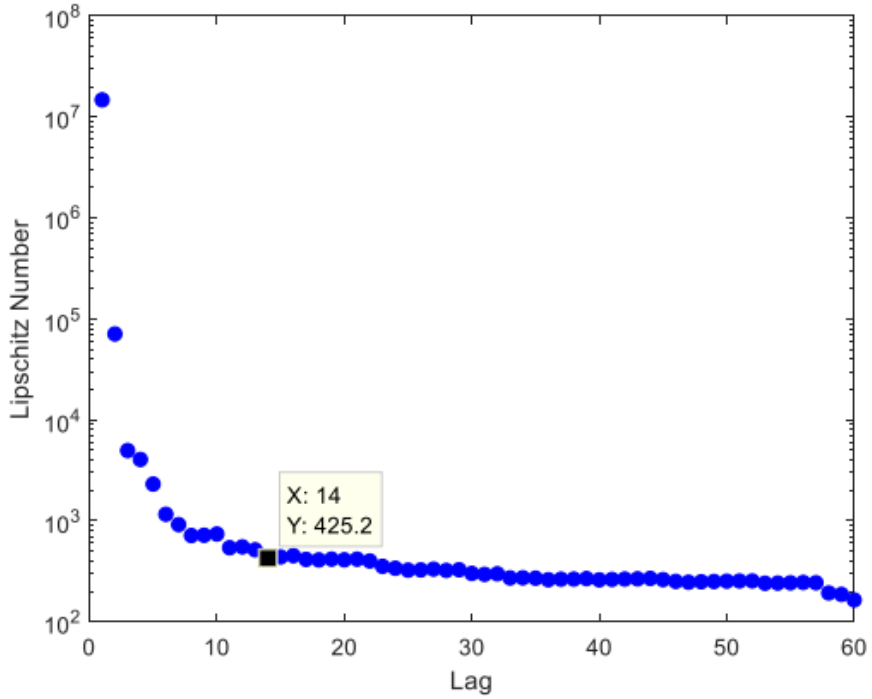


Figure 11 Lipschitz for 60 dynamics from the first input and the second output (steamgen)

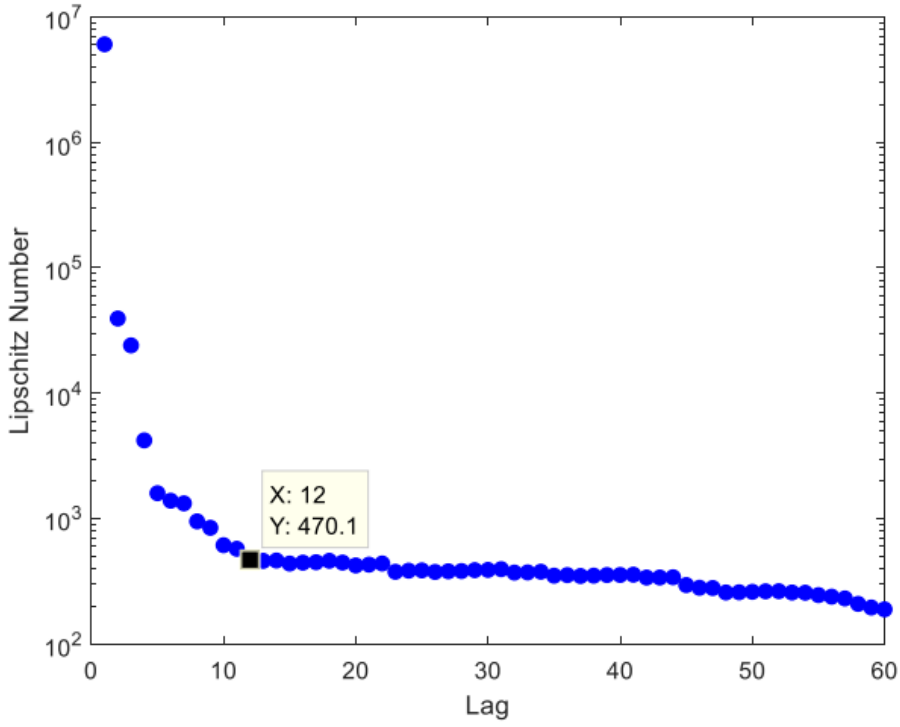


Figure 12 Lipschitz for 60 dynamics from the second input and the second output (steamgen)

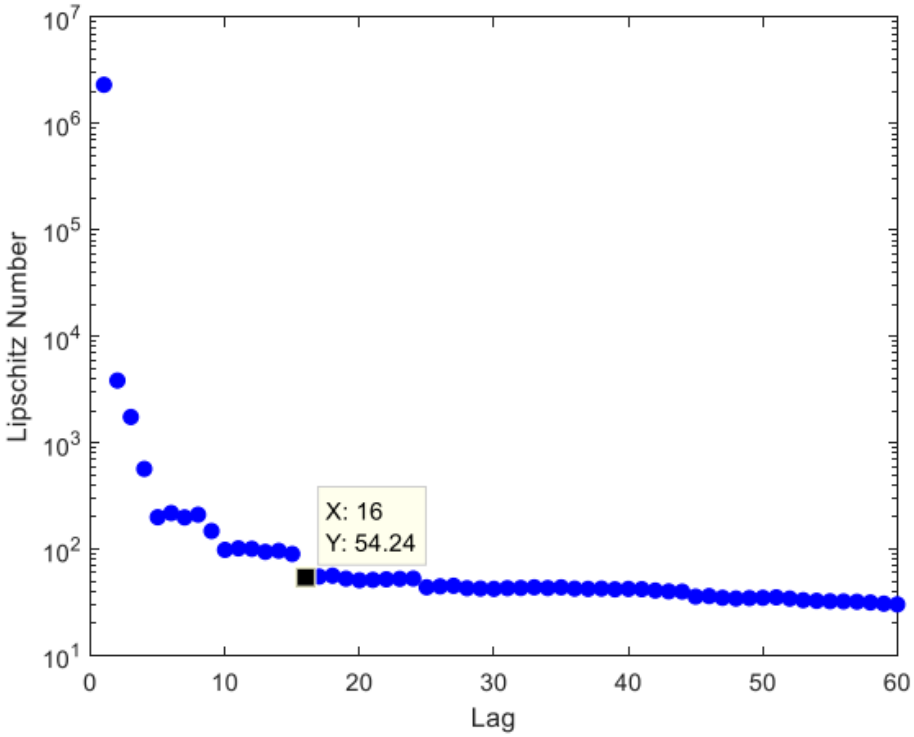


Figure 13 Lipschitz for 60 dynamics from the third input and the second output (steamgen)

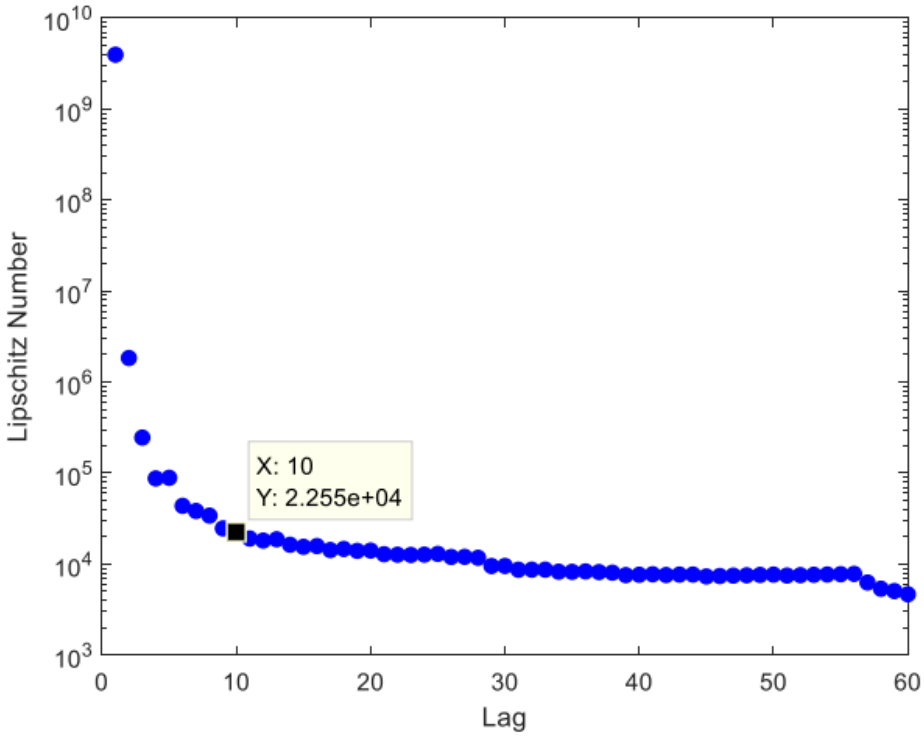


Figure 14 Lipschitz for 60 dynamics from the fourth input and the second output (steamgen)

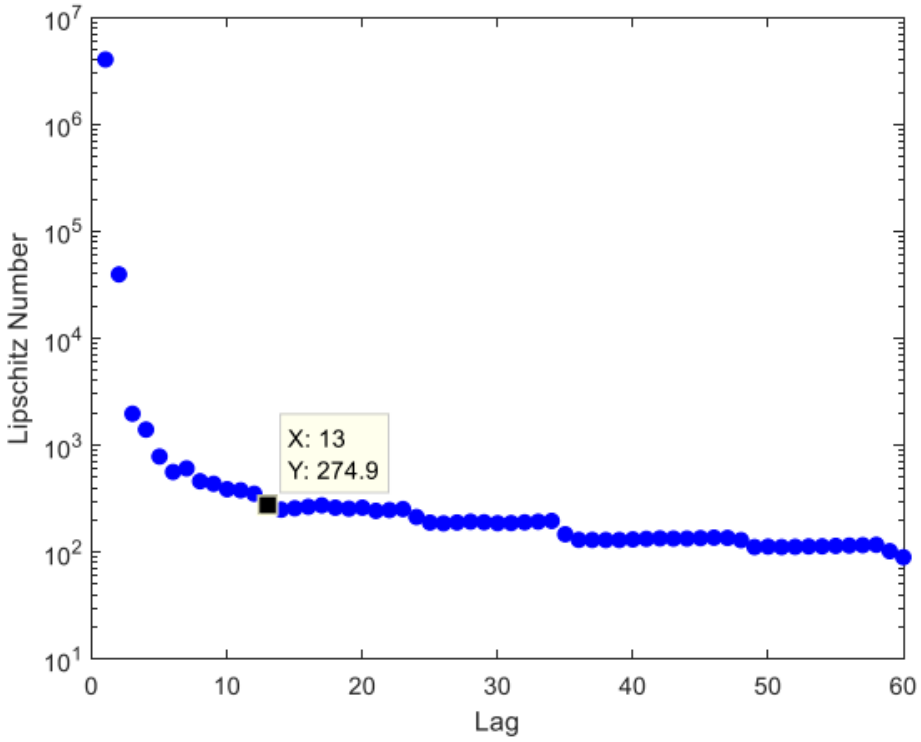


Figure 15 Lipschitz for 60 dynamics from the first input and the third output (steamgen)

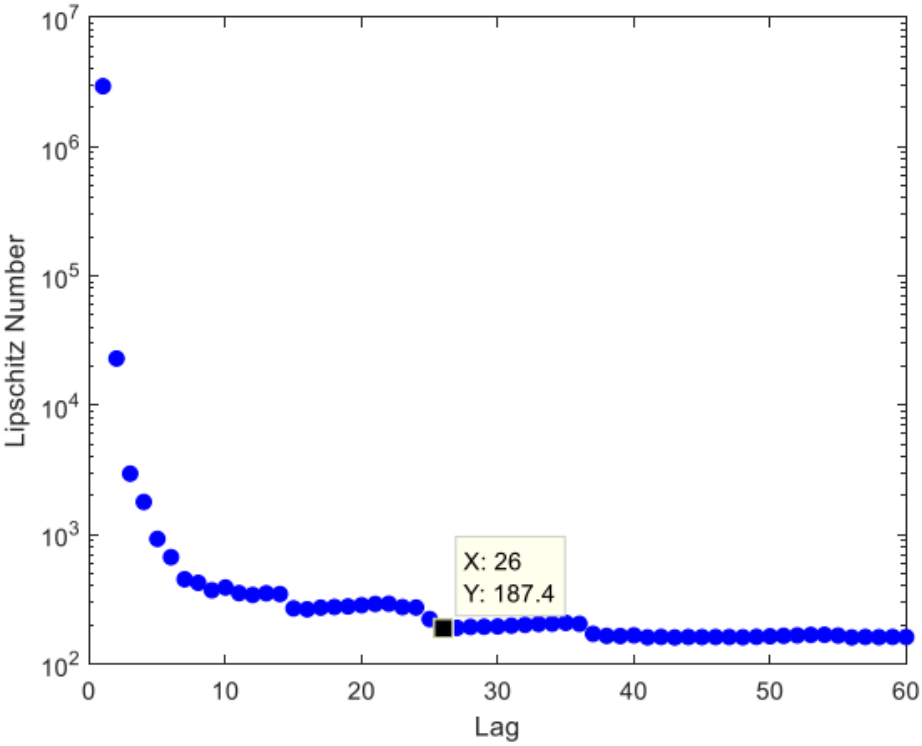


Figure 16 Lipschitz for 60 dynamics from the second input and the third output (steamgen)

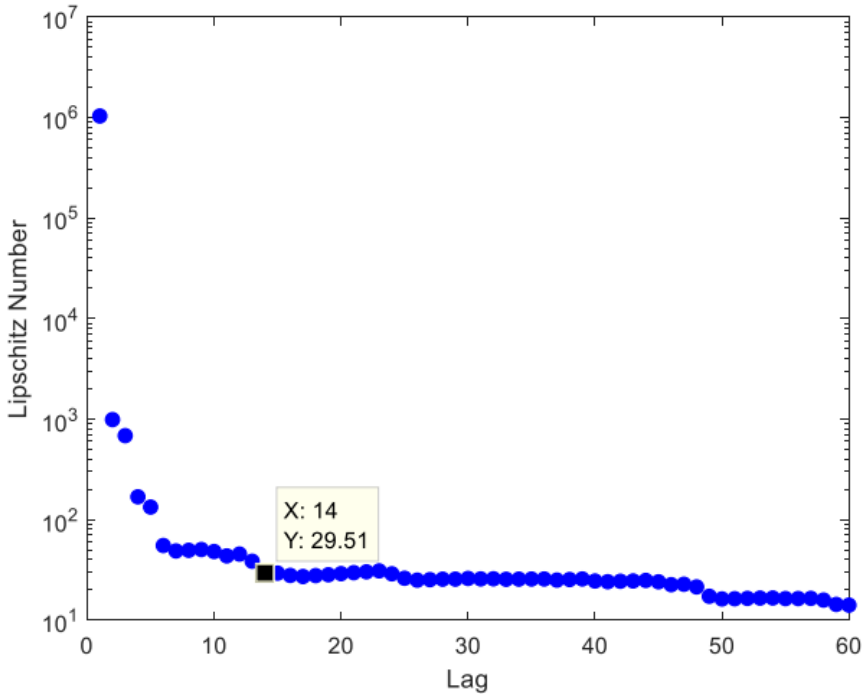


Figure 17 Lipschitz for 60 dynamics from the third input and the third output (steamgen)

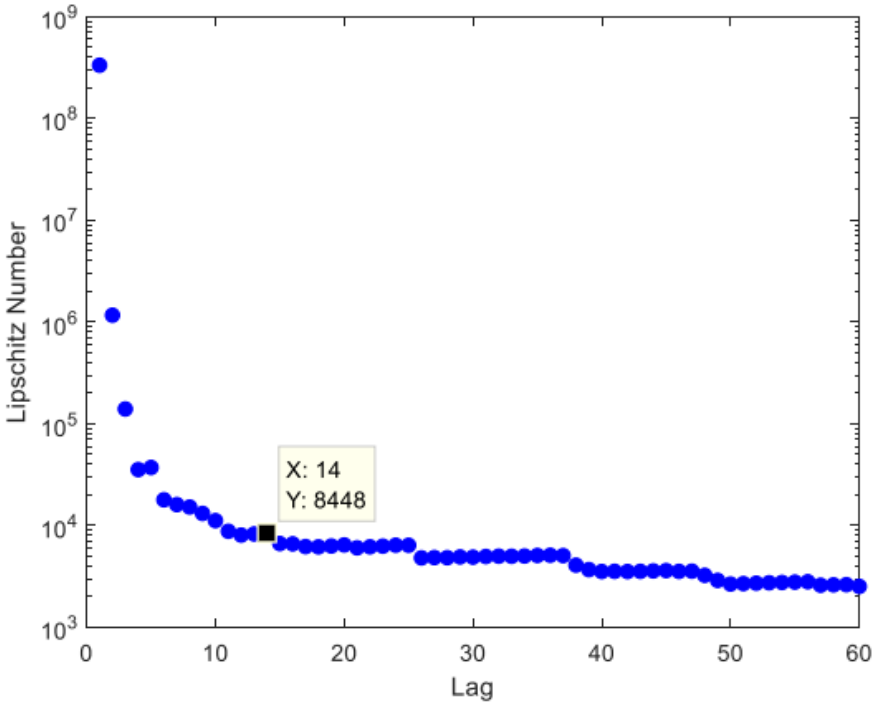


Figure 18 Lipschitz for 60 dynamics from the fourth input and the third output (steamgen)

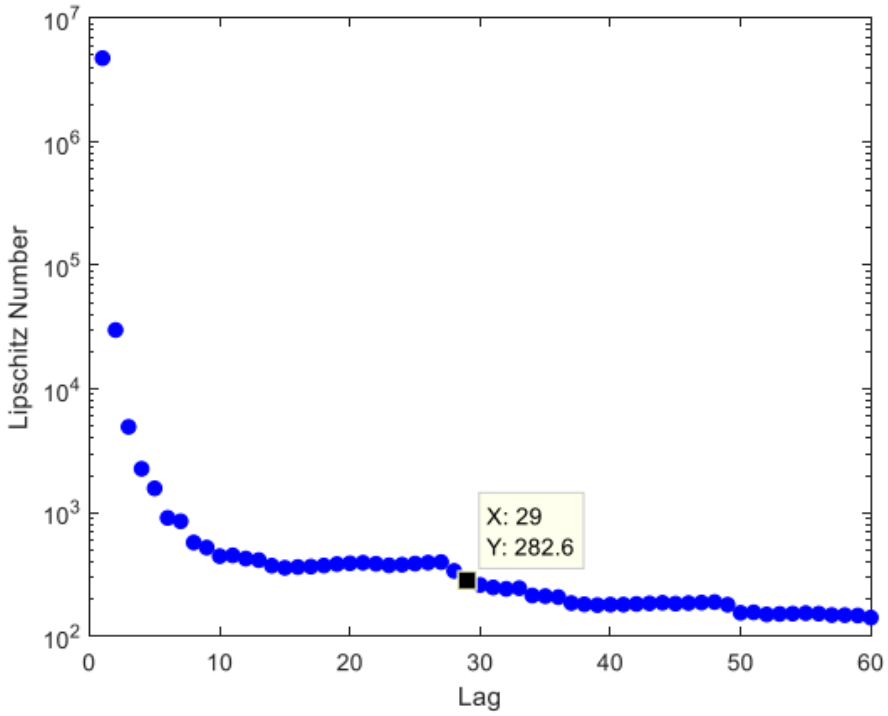


Figure 19 Lipschitz for 60 dynamics from the first input and the fourth output (steamgen)

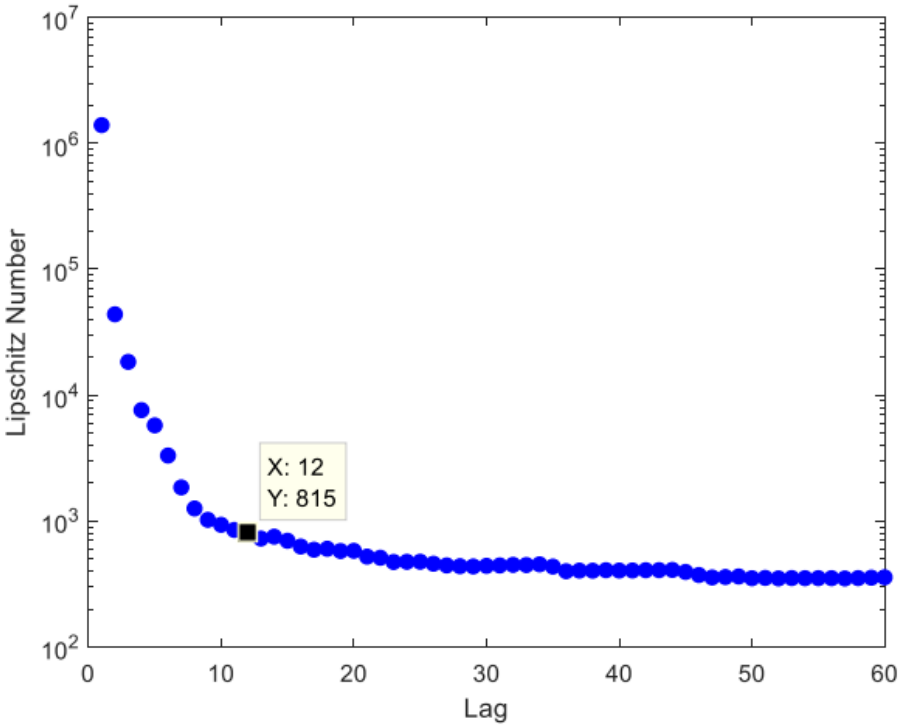


Figure 20 Lipschitz for 60 dynamics from the second input and the fourth output (steamgen)

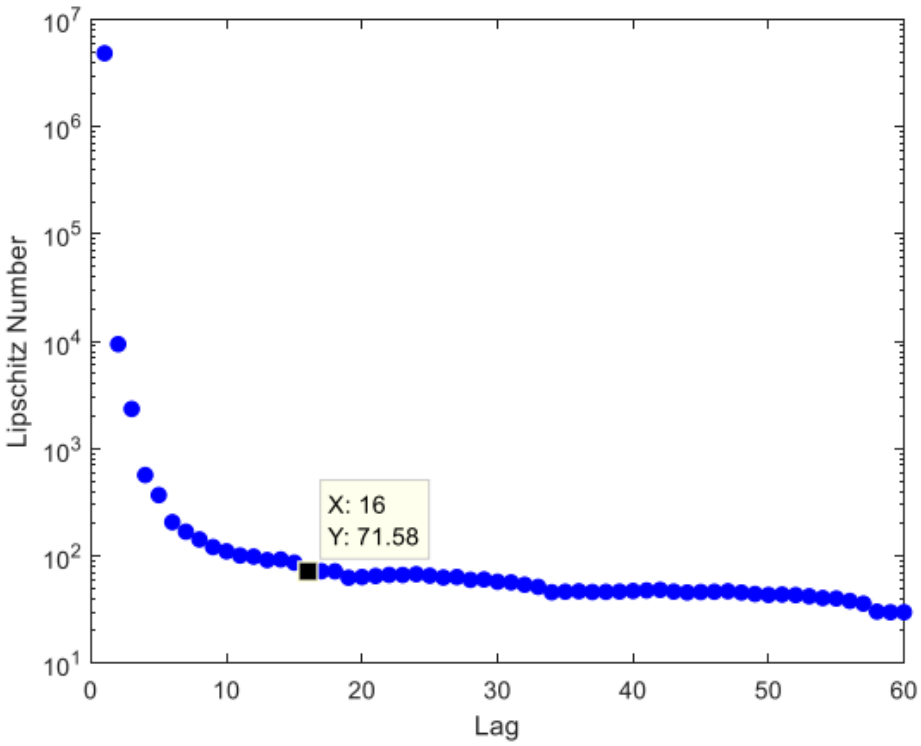


Figure 21 Lipschitz for 60 dynamics from the third input and the fourth output (steamgen)

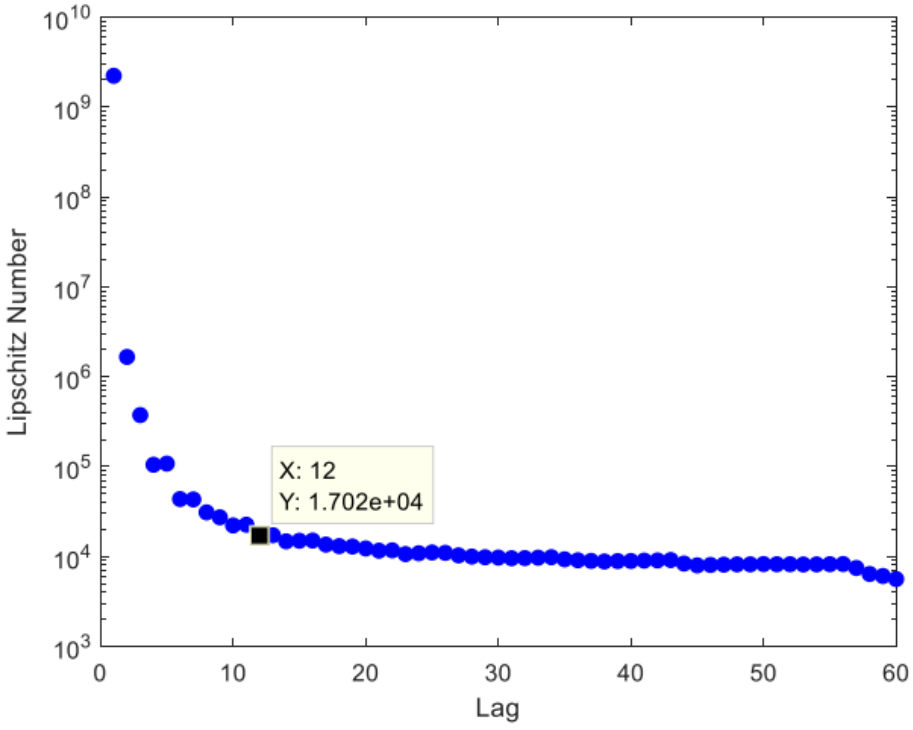


Figure 22 Lipschitz for 60 dynamics from the fourth input and the fourth output (steamgen)

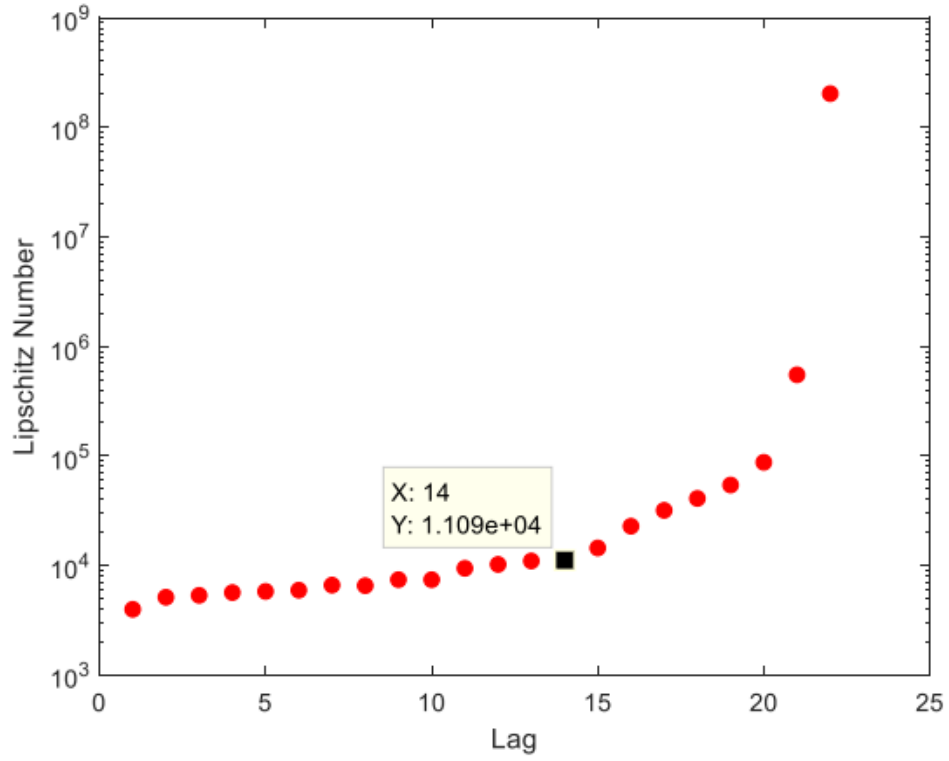


Figure 23 The delay between the first input and first output (steamgen)

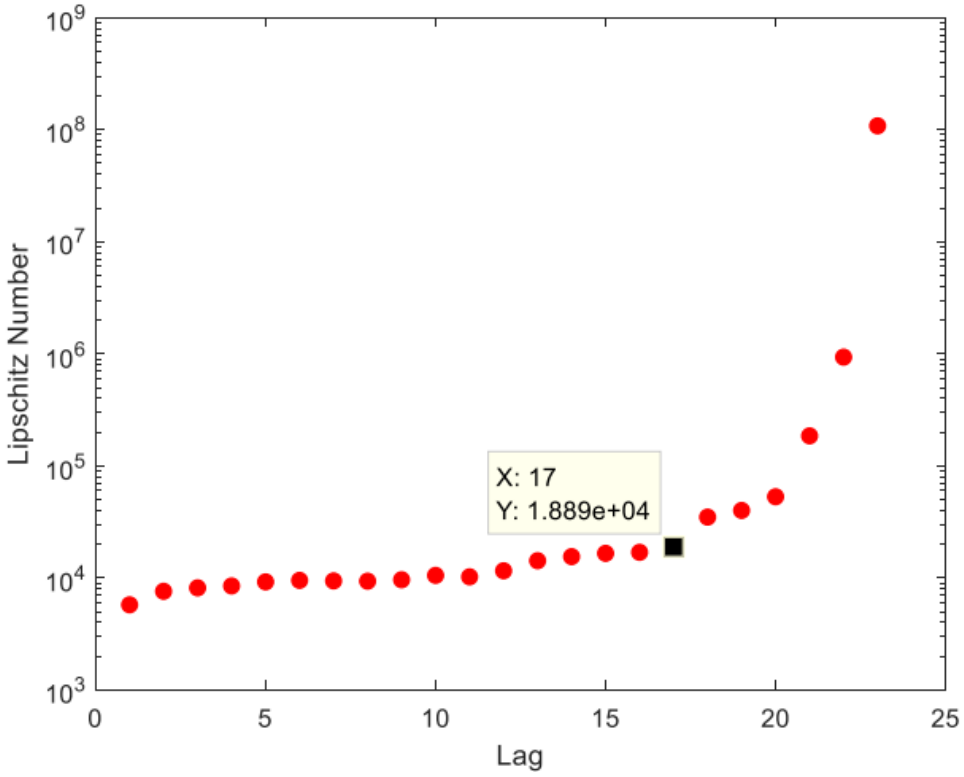


Figure 24 The delay between the second input and first output (steamgen)

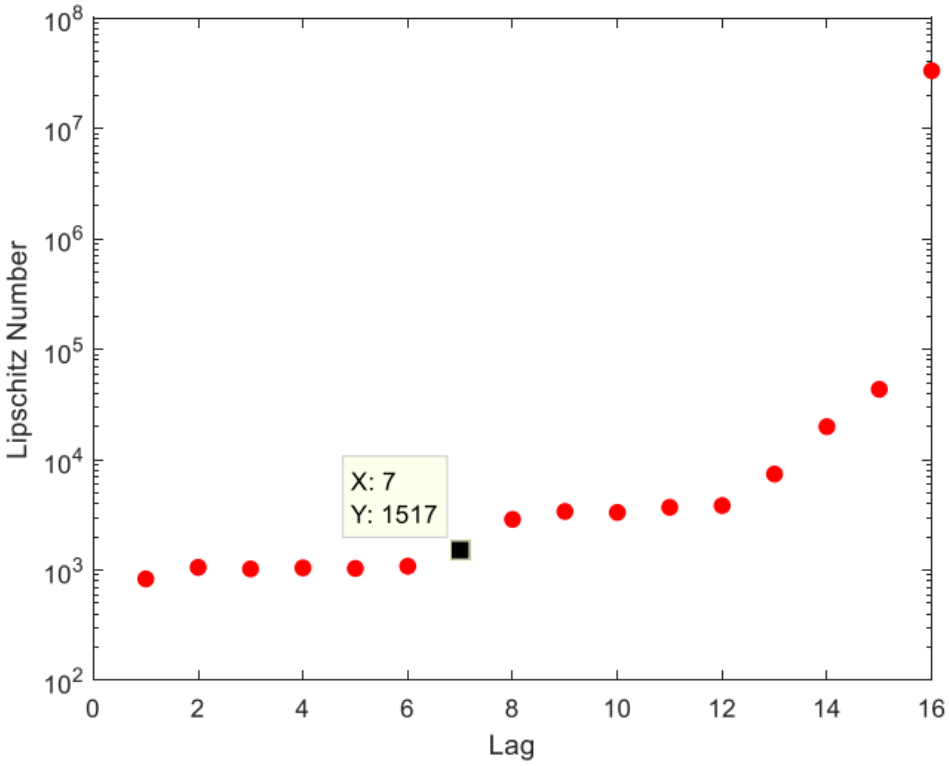


Figure 25 The delay between the third input and first output (steamgen)

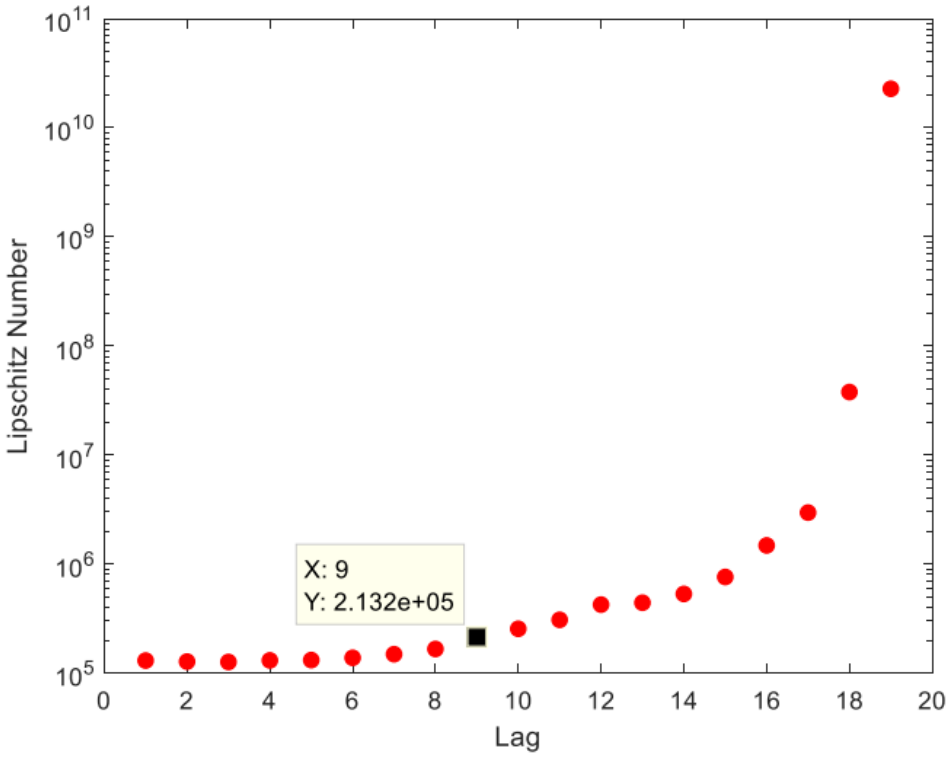


Figure 26 The delay between the fourth input and first output (steamgen)

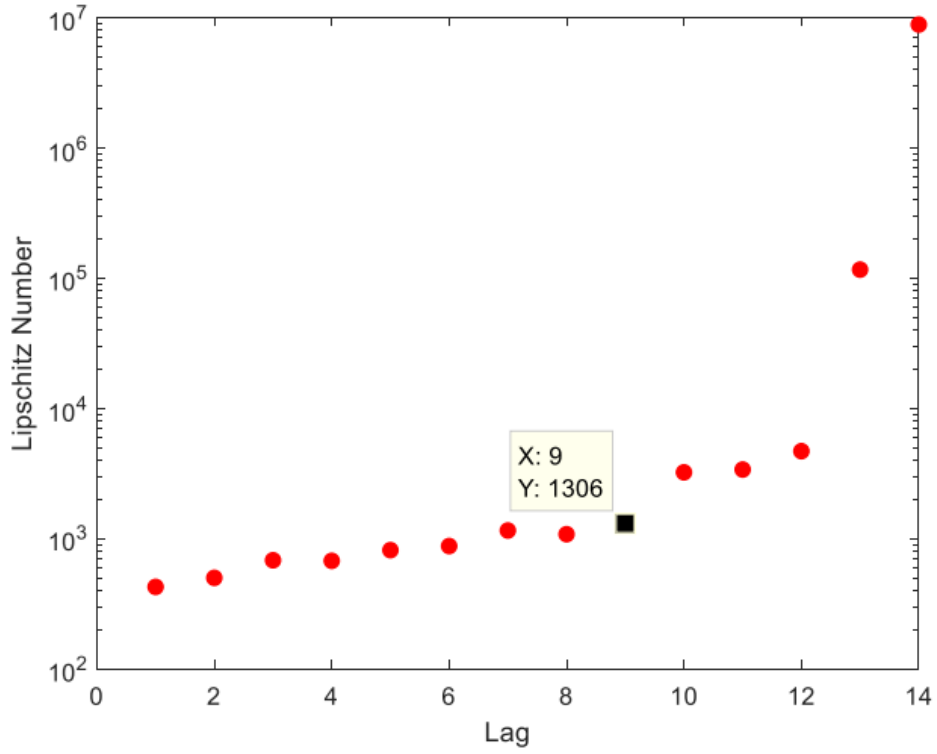


Figure 27 The delay between the first input and second output (steamgen)

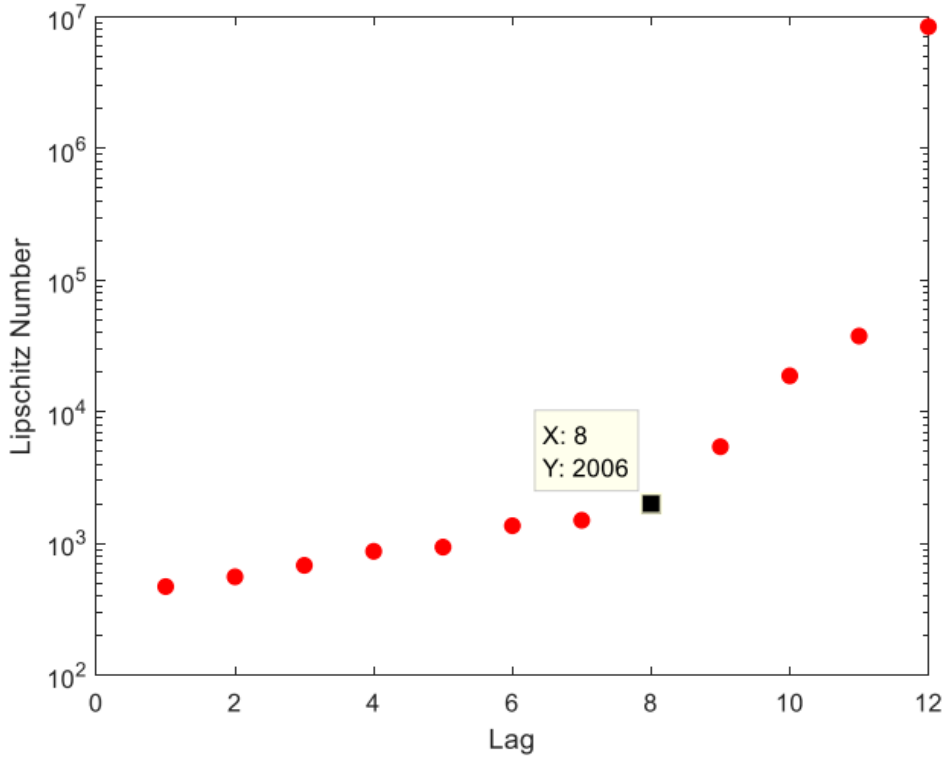


Figure 28 The delay between the second input and second output (steamgen)

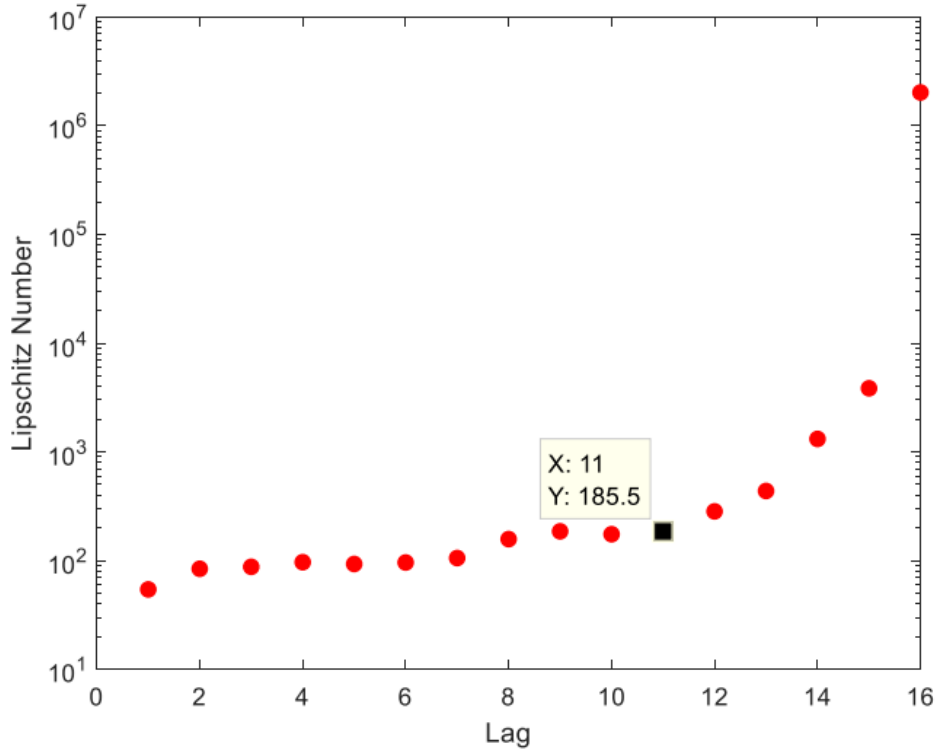


Figure 29 The delay between the third input and second output (steamgen)

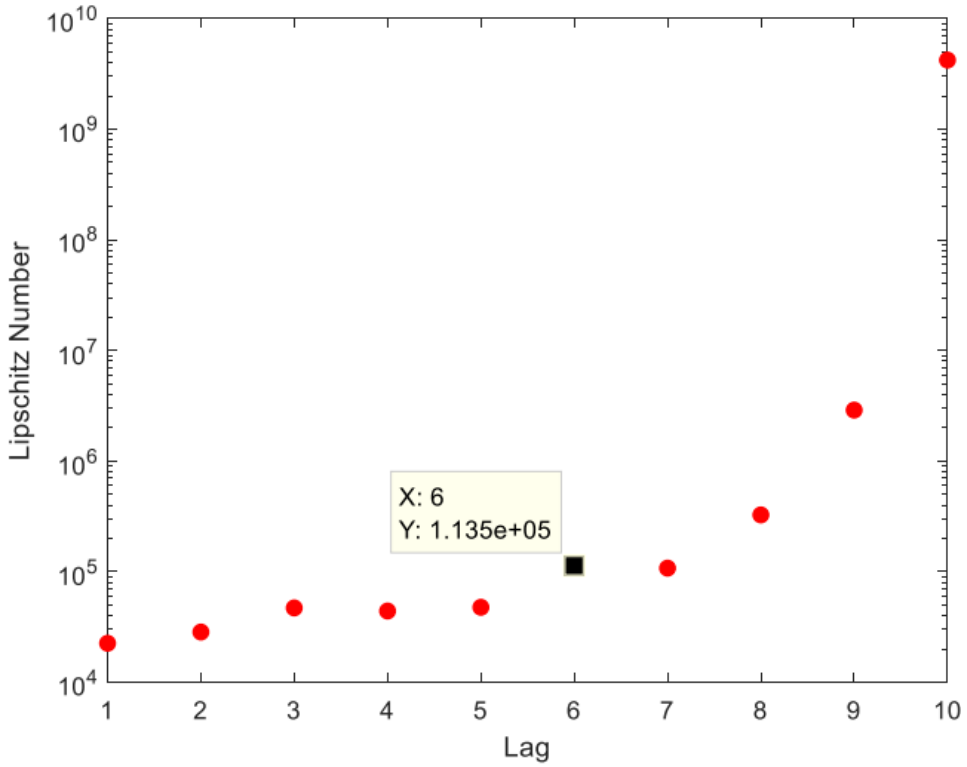


Figure 30 The delay between the fourth input and second output (steamgen)

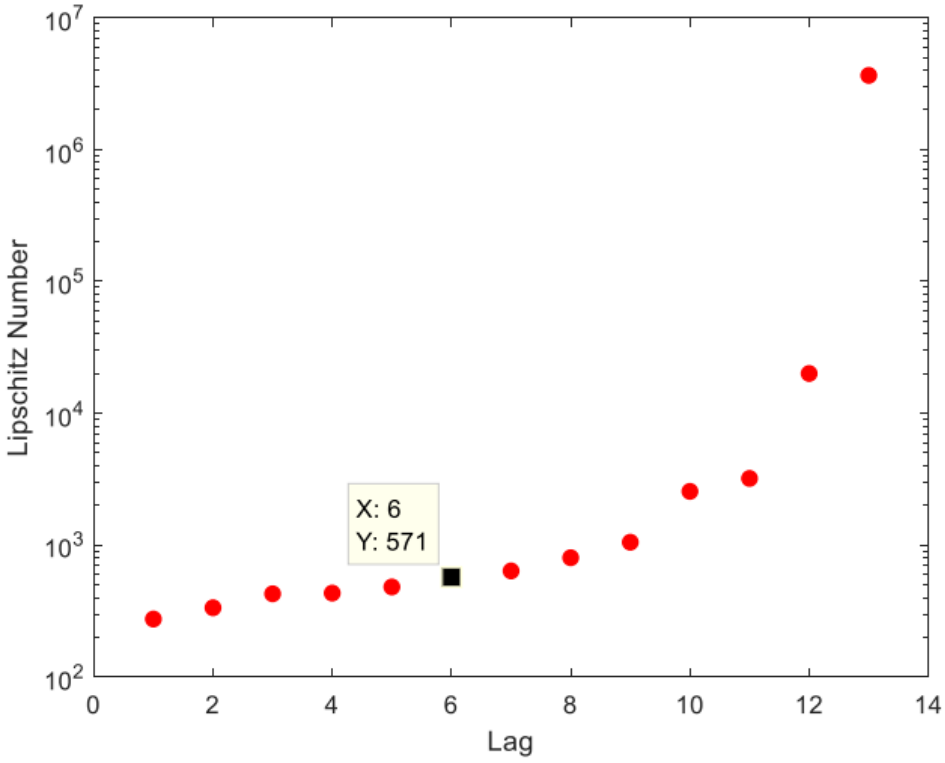


Figure 31 The delay between the first input and third output (steamgen)

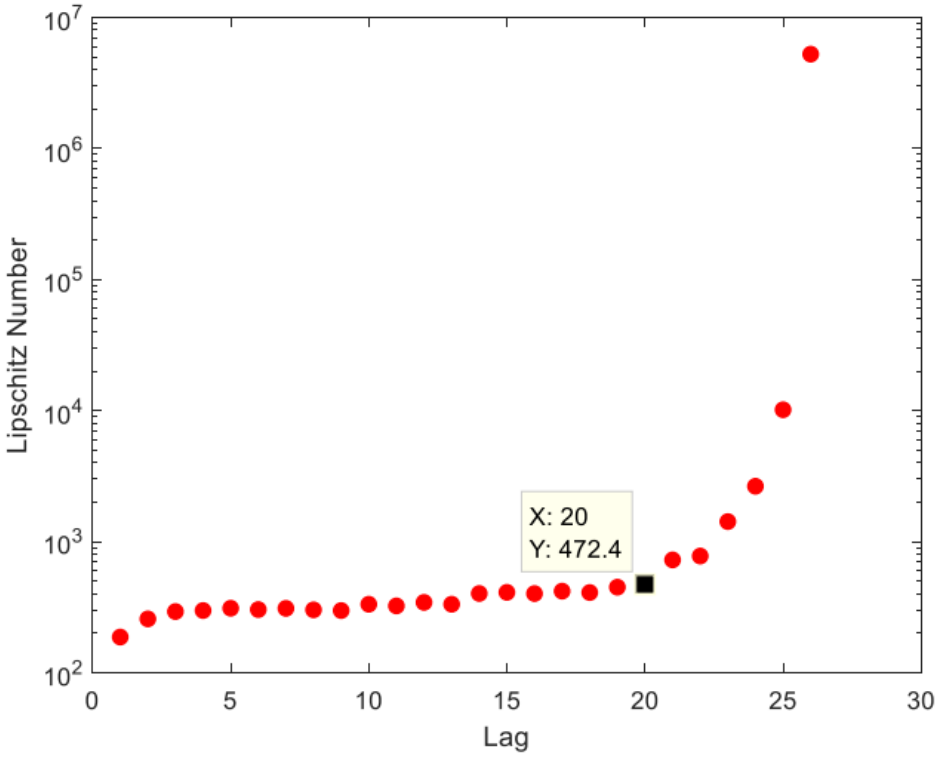


Figure 32 The delay between the second input and third output (steamgen)

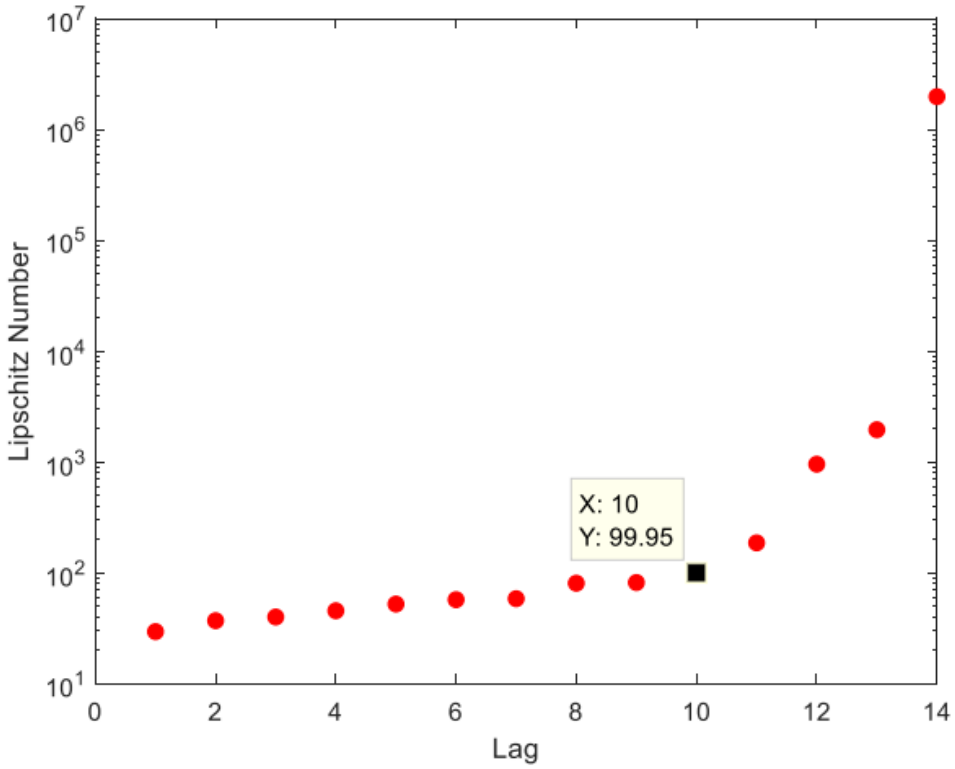


Figure 33 The delay between the third input and third output (steamgen)

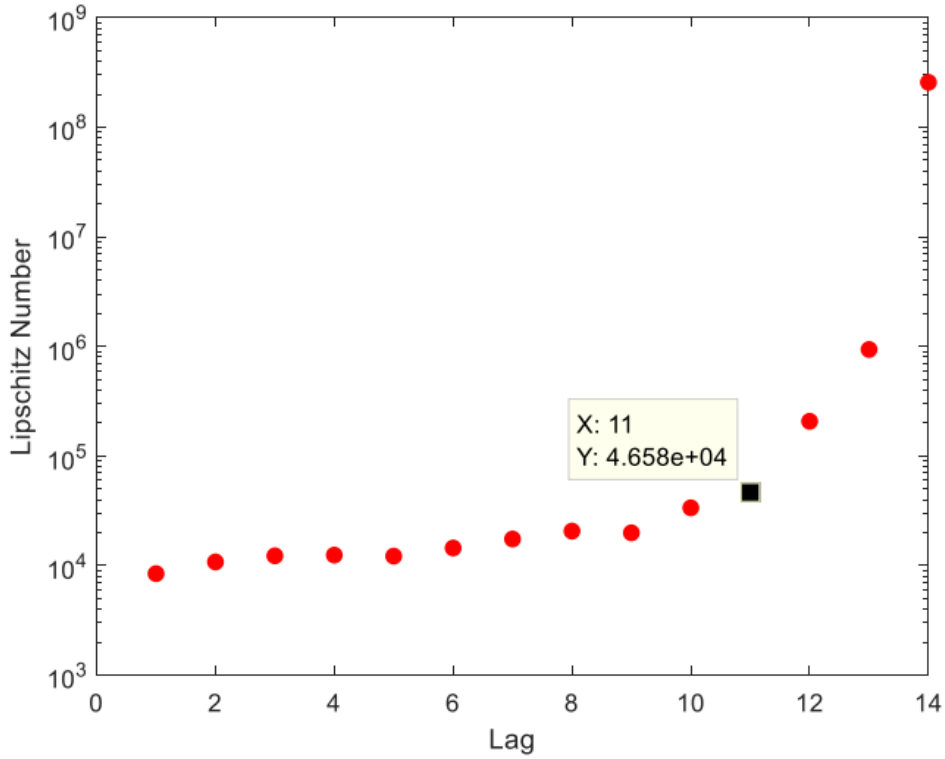


Figure 34 The delay between the fourth input and third output (steamgen)

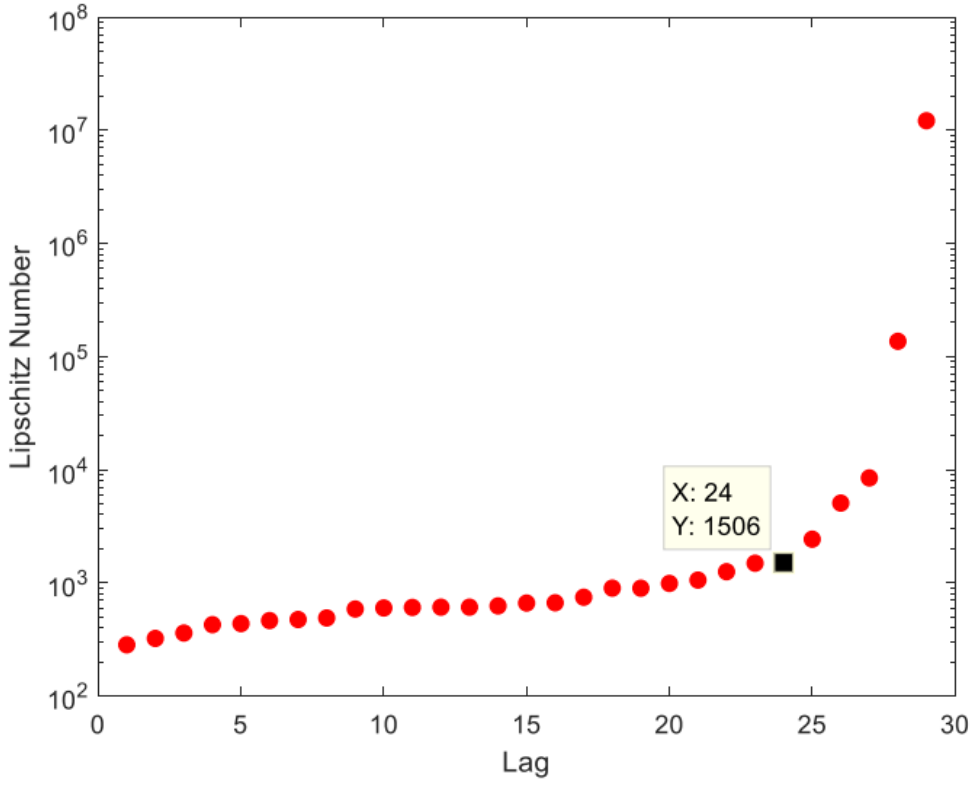


Figure 35 The delay between the first input and fourth output (steamgen)

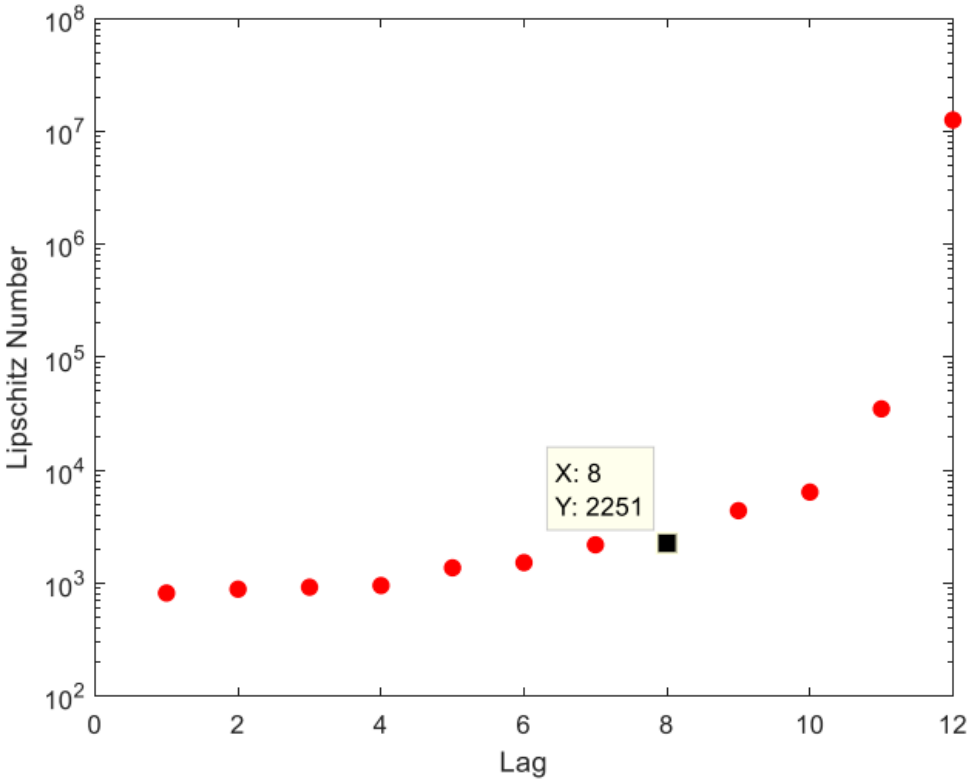


Figure 36 The delay between the second input and fourth output (steamgen)

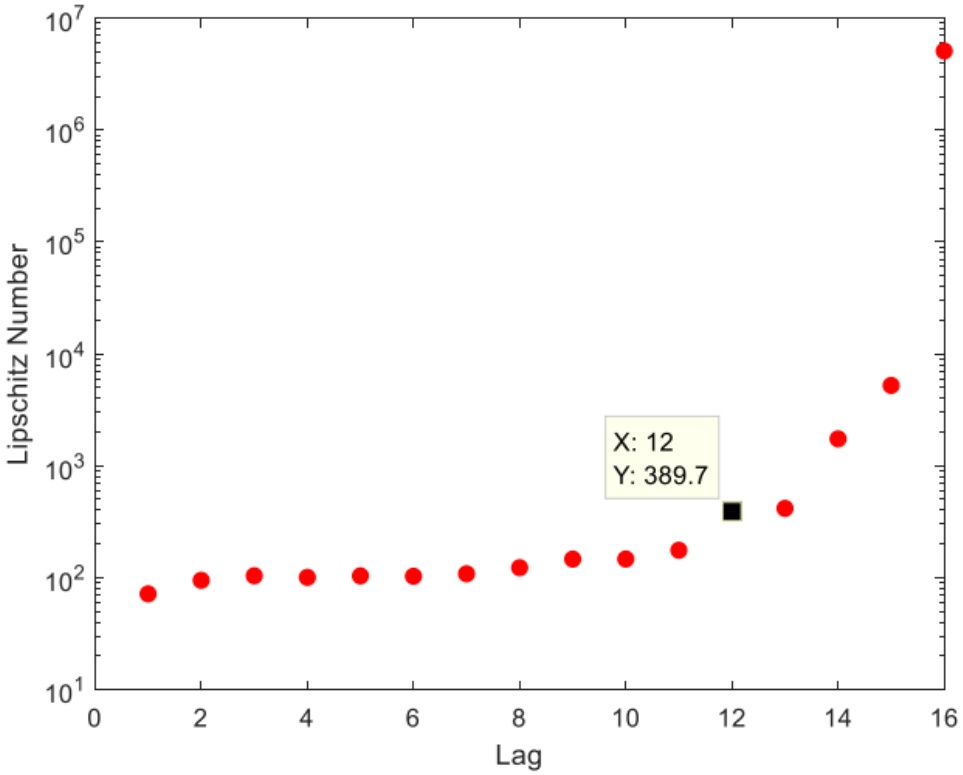


Figure 37 The delay between the third input and fourth output (steamgen)

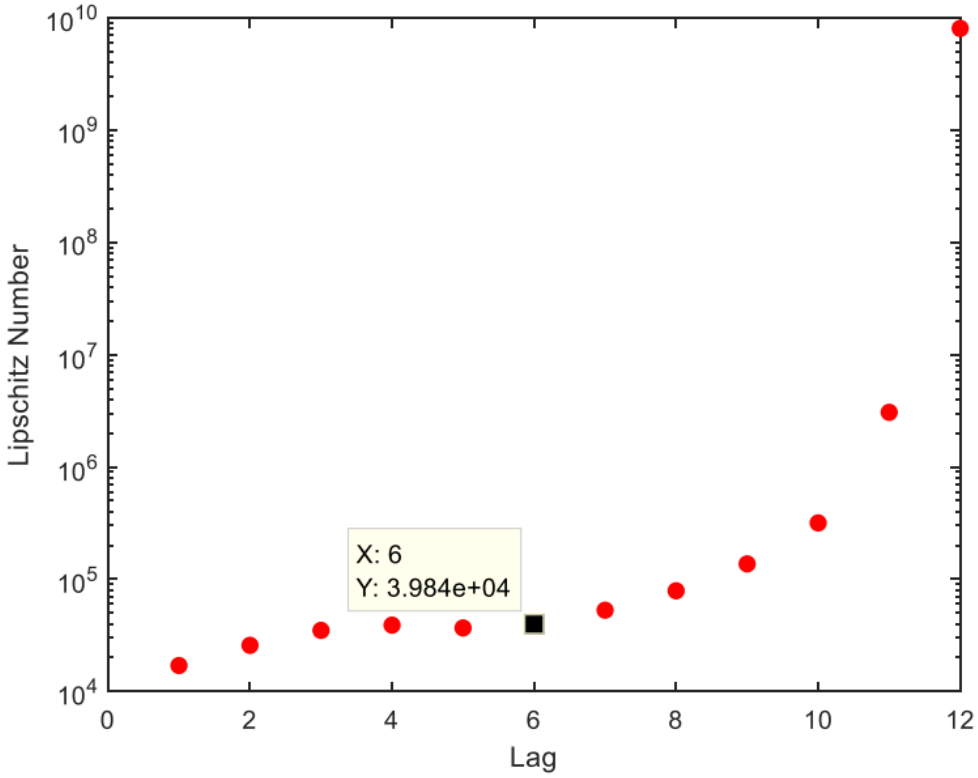


Figure 38 The delay between the fourth input and fourth output (steamgen)

Table 1 the delay between inputs and output pairs (steamgem)

Input-output pair	Delay	Dynamic Depth
1st input and 1st output	14	8
2nd input and 1st output	17	6
3rd input and 1st output	7	9
4th input and 1st output	9	10
1st input and 2nd output	9	5
2nd input and 2nd output	8	4
3rd input and 2nd output	11	5
4th input and 2nd output	6	4
1st input and 3rd output	6	7
2nd input and 3rd output	20	6
3rd input and 3rd output	10	4
4th input and 3rd output	11	3
1st input and 4th output	24	5
2nd input and 4th output	8	4
3rd input and 4th output	12	4
4th input and 4th output	6	6

In the next step, we intend to estimate the system delay using statistical methods such as the correlation method, which is related to linear algebra. Systems 1 and 2 are nonlinear systems; therefore, it is not expected that linear algebra would be an appropriate measure for estimating the delay of these systems. However, to calculate the correlation function's estimate, we plot the correlation between inputs and outputs. For increased accuracy, the correlation diagram is drawn for the first 200 dynamics.

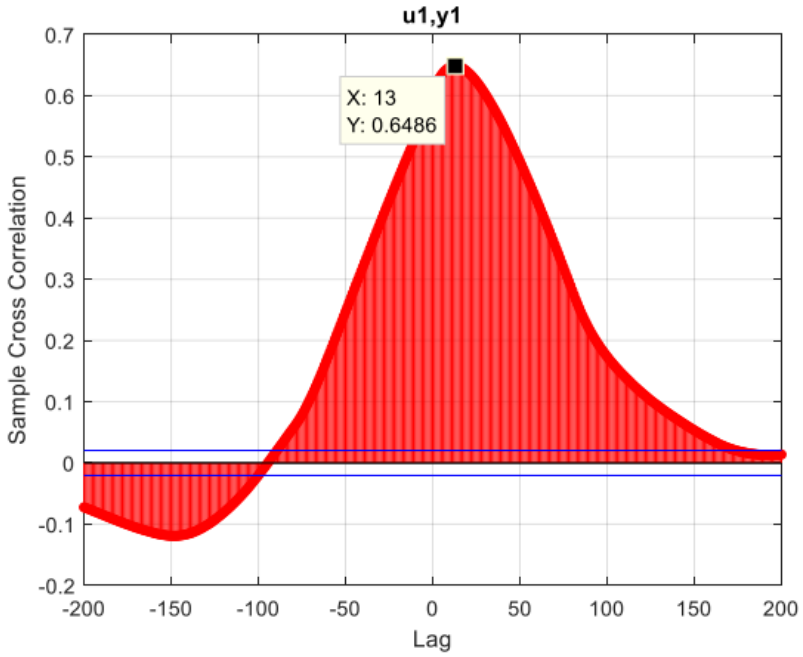


Figure 39 Correlation between first input and first output (streamgen)

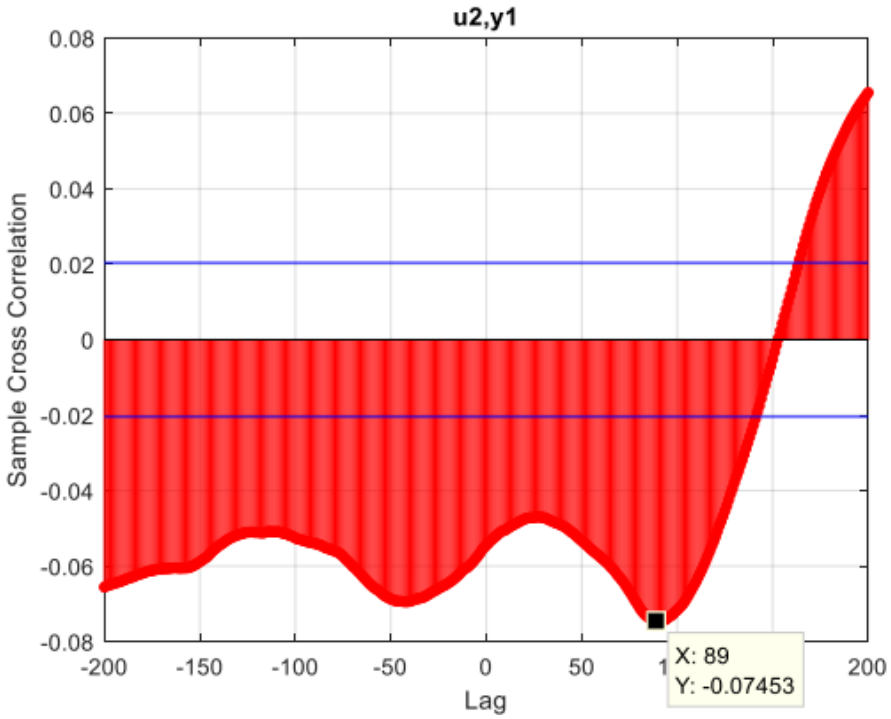


Figure 40 Correlation between second input and first output (streamgen)

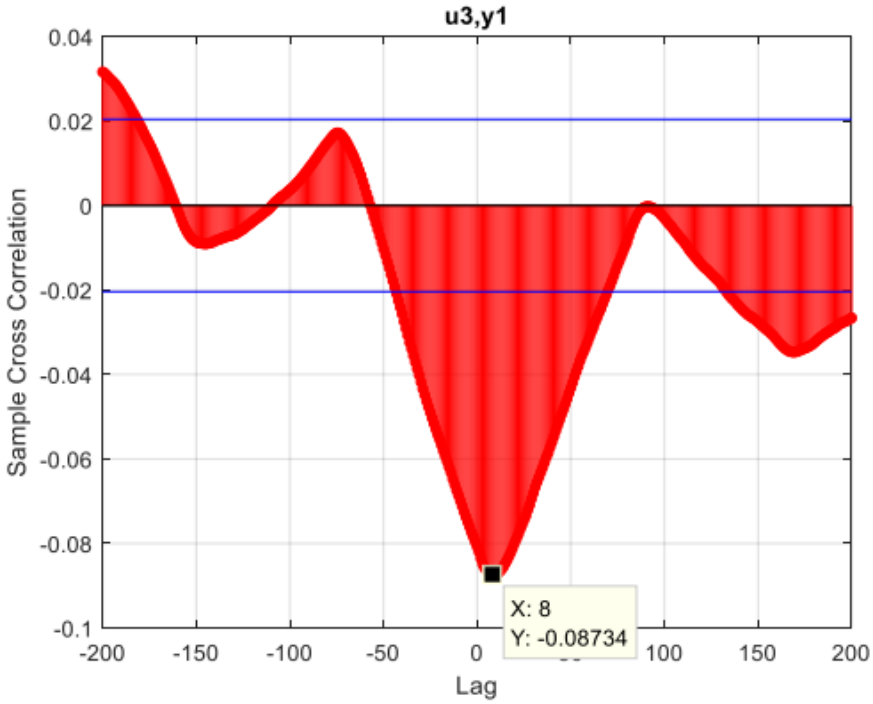


Figure 41 Correlation between third input and first output (streamgen)

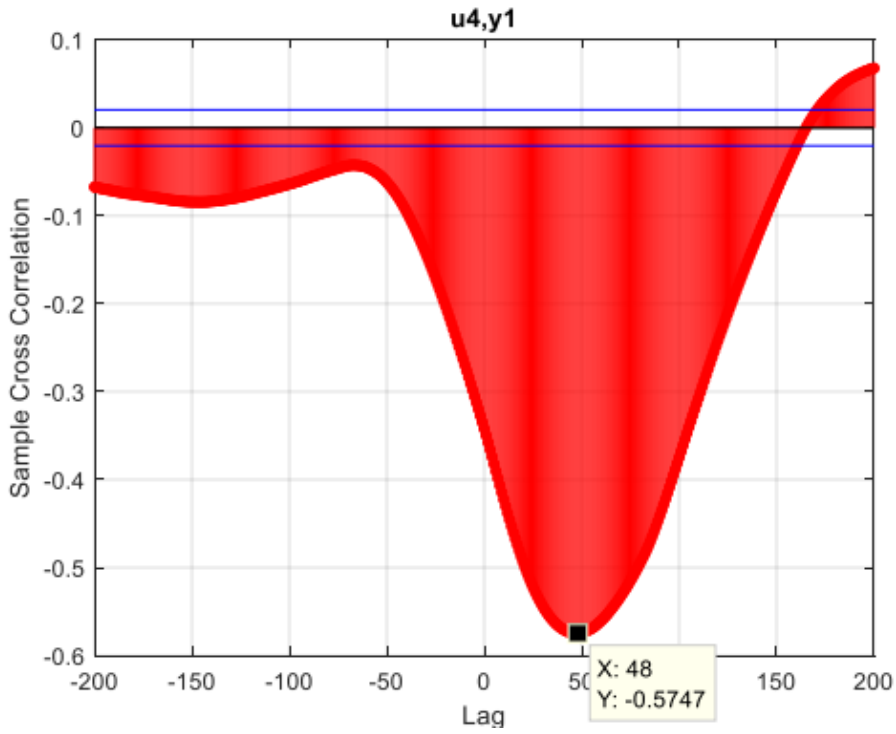


Figure 42 Correlation between fourth input and first output (streamgen)

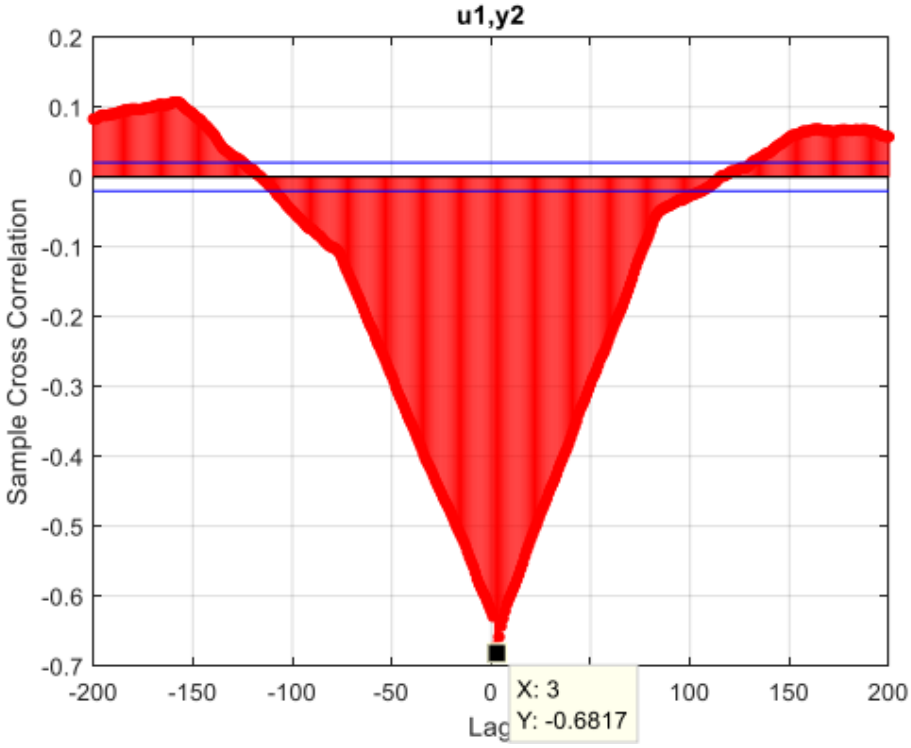


Figure 43 Correlation between fourth input and first output (streamgen)

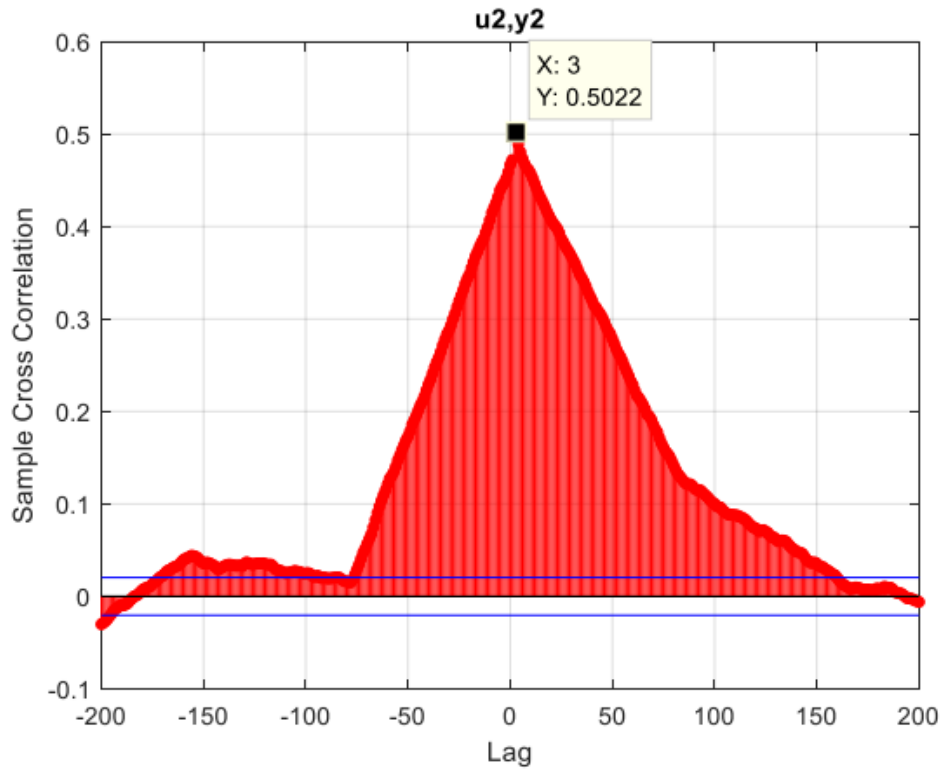


Figure 44 Correlation between second input and second output (streamgen)

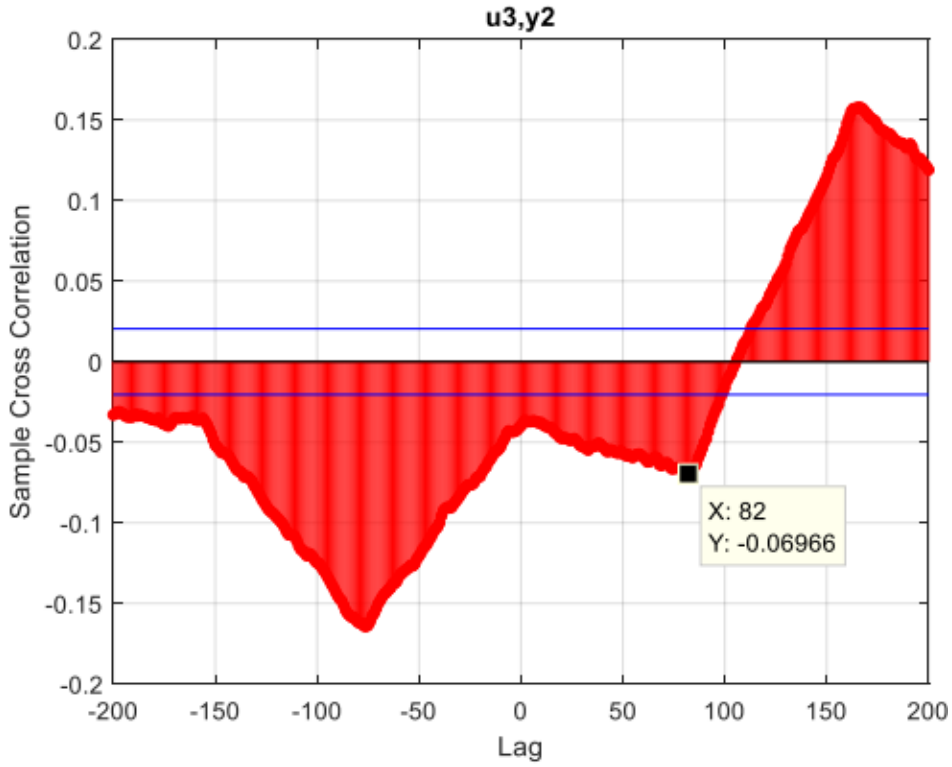


Figure 45 Correlation between third input and second output (streamgen)

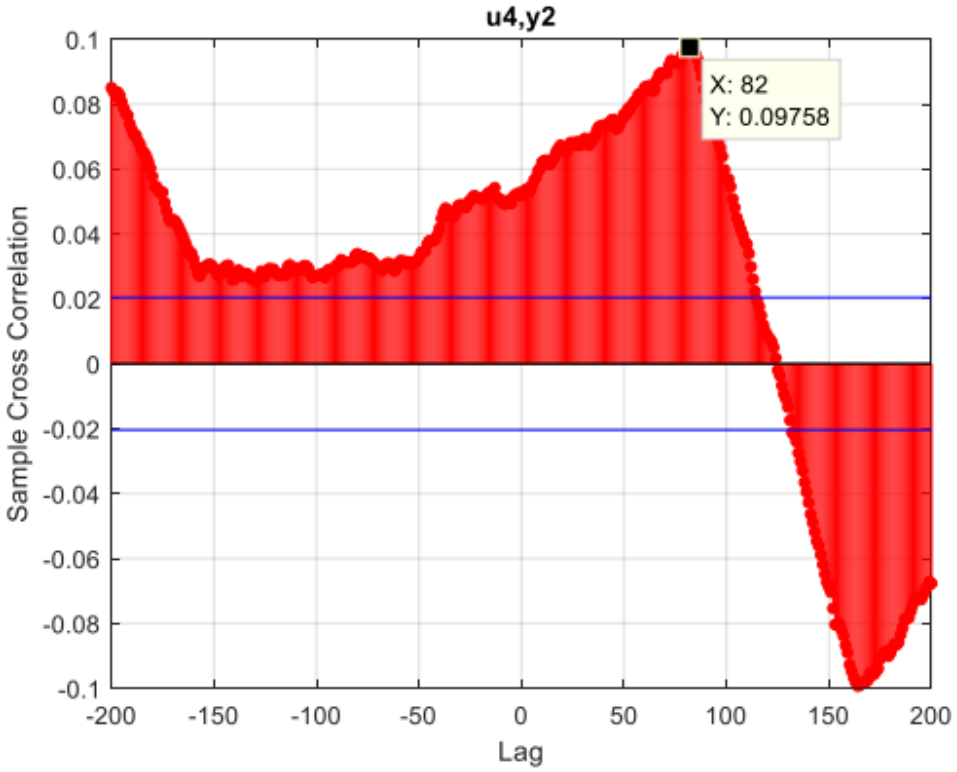


Figure 46 Correlation between fourth input and second output (streamgen)

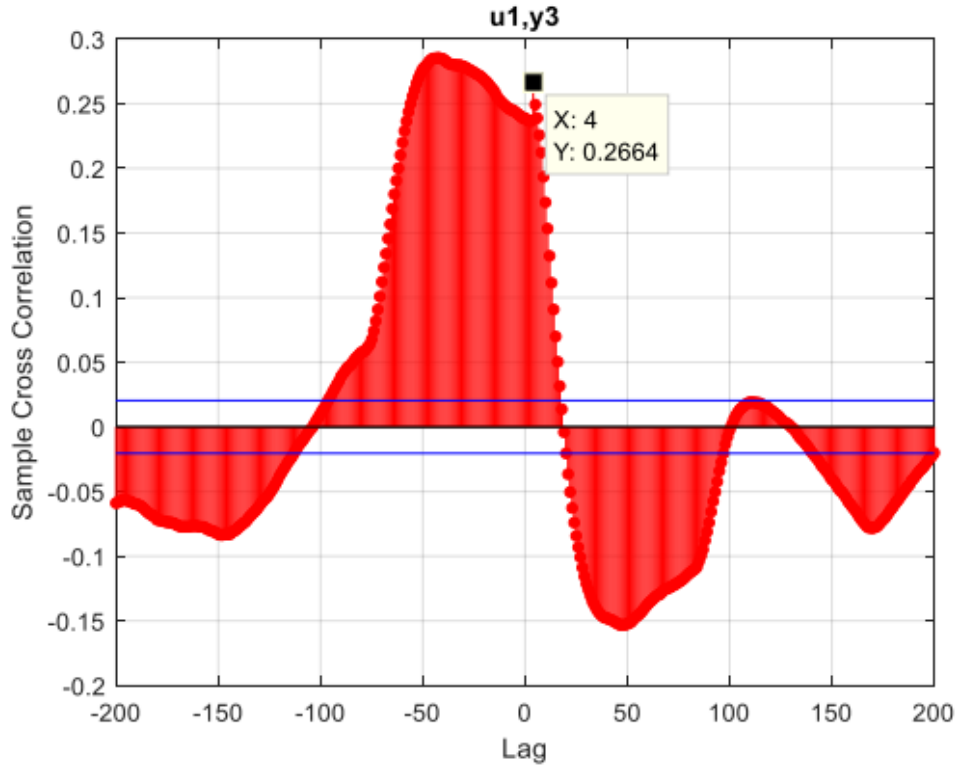


Figure 47 Correlation between first input and third output (streamgen)

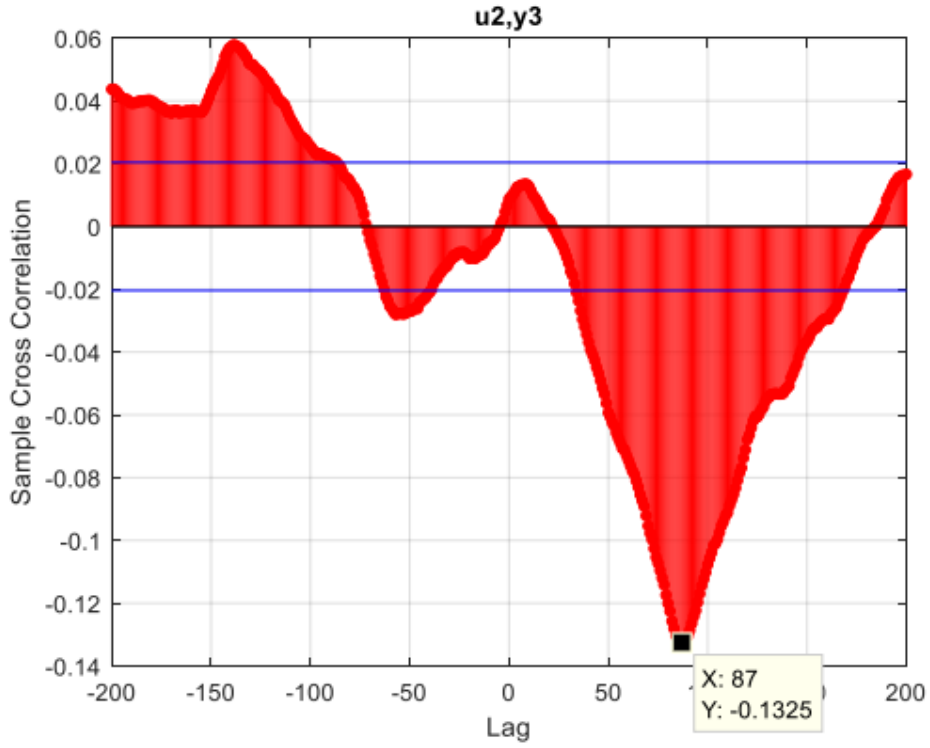


Figure 48 Correlation between second input and third output (streamgen)

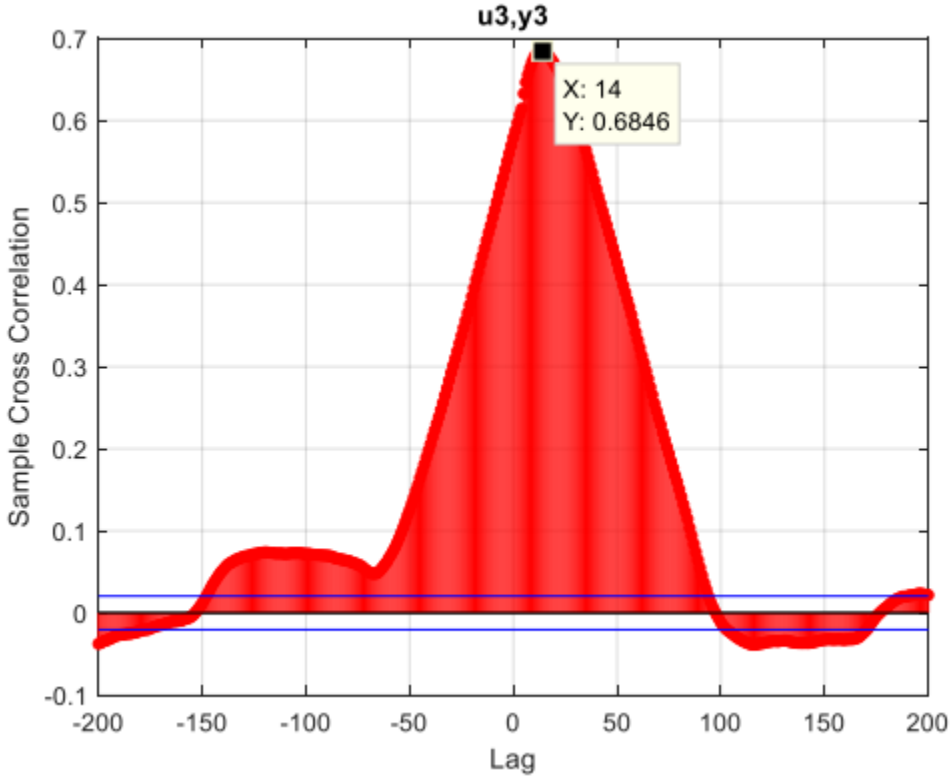


Figure 49 Correlation between third input and third output (streamgen)

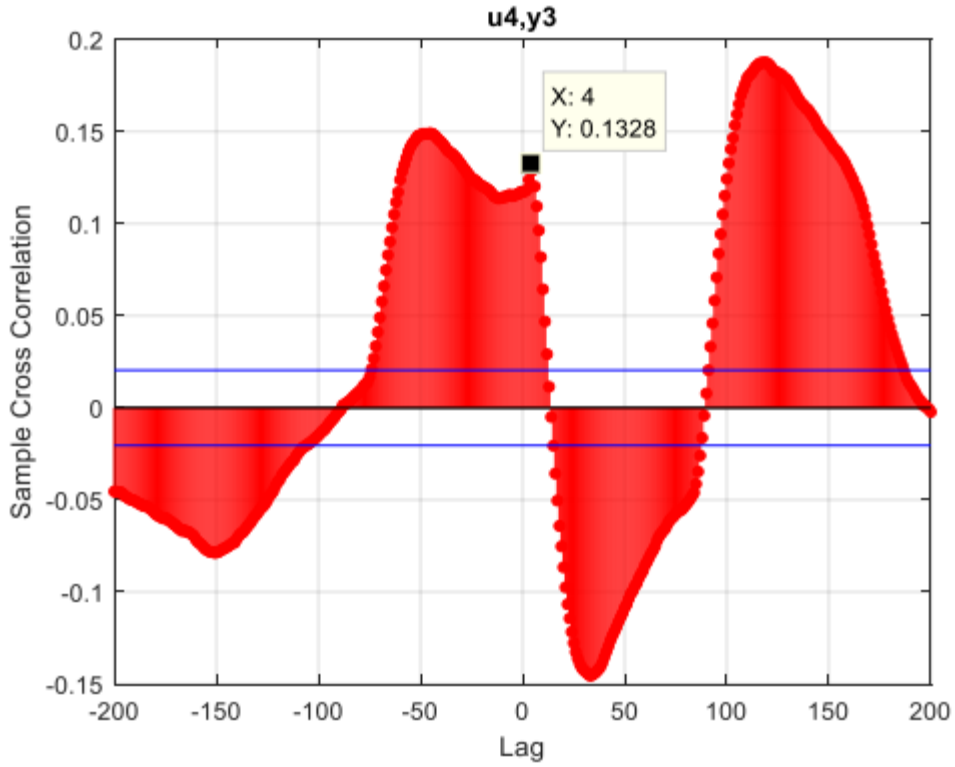


Figure 50 Correlation between fourth input and third output (streamgen)

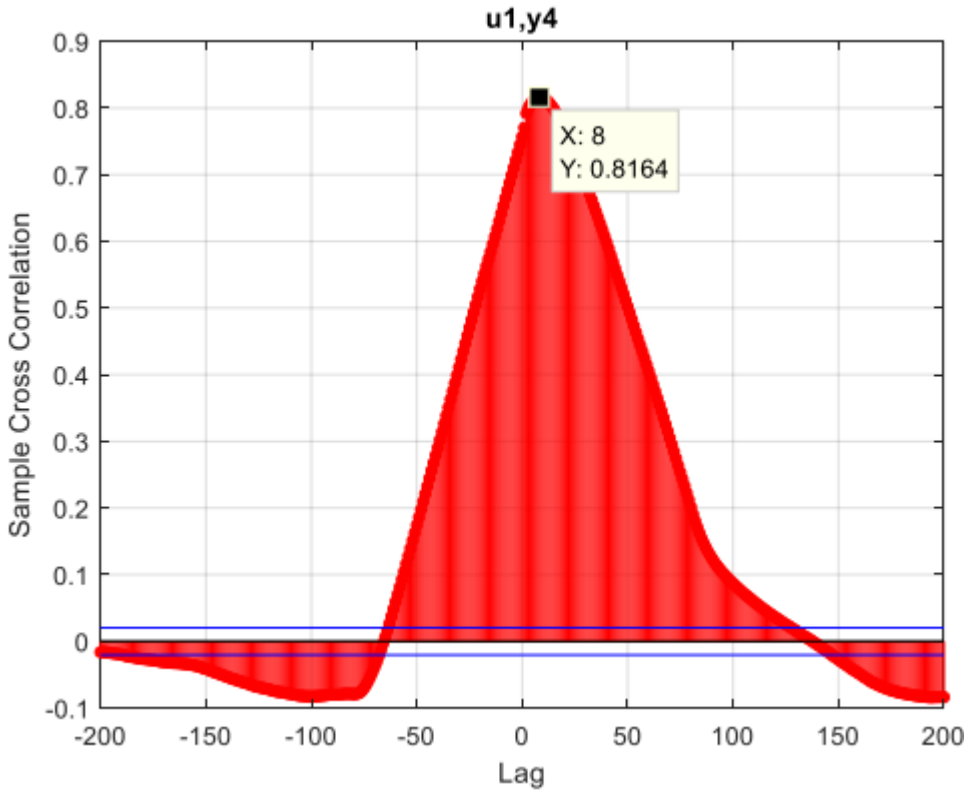


Figure 51 Correlation between first input and fourth output (streamgen)

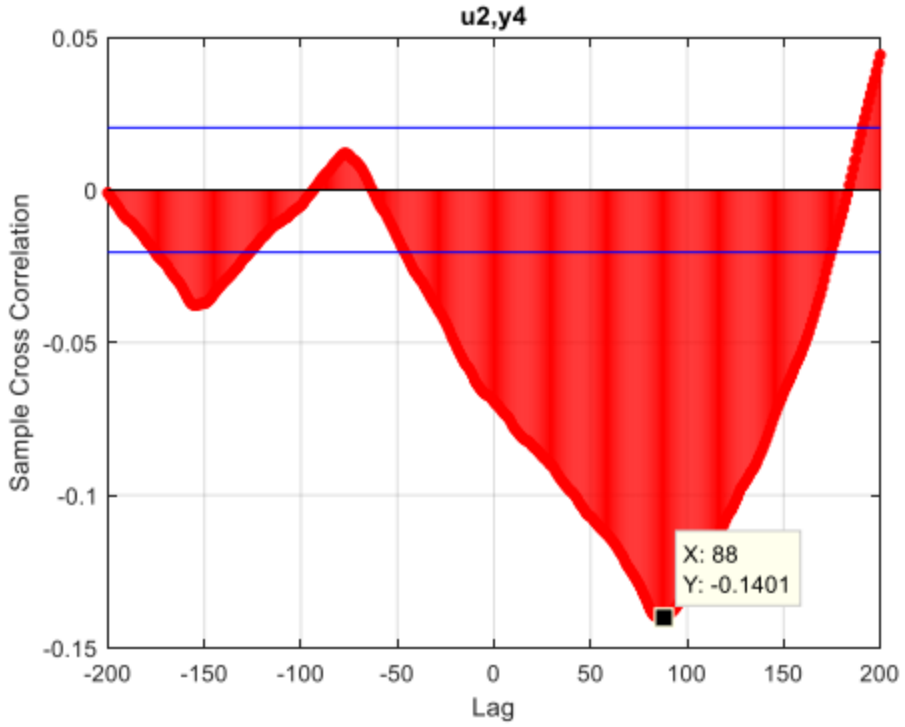


Figure 52 Correlation between second input and fourth output (streamgen)

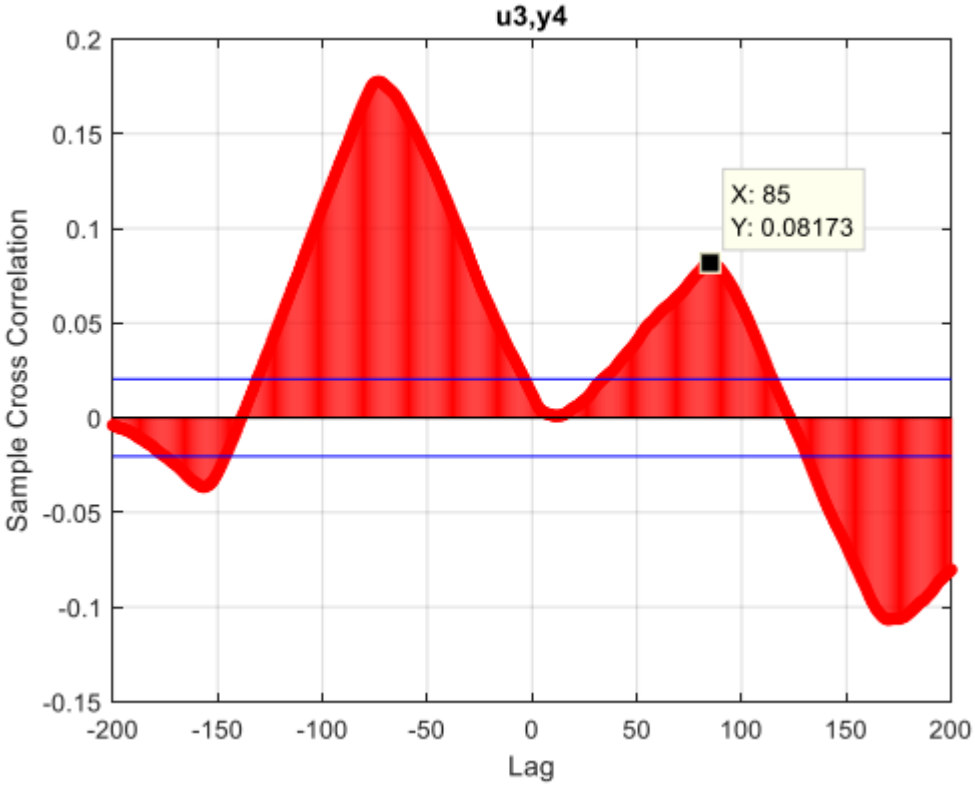


Figure 53 Correlation between third input and fourth output (streamgen)

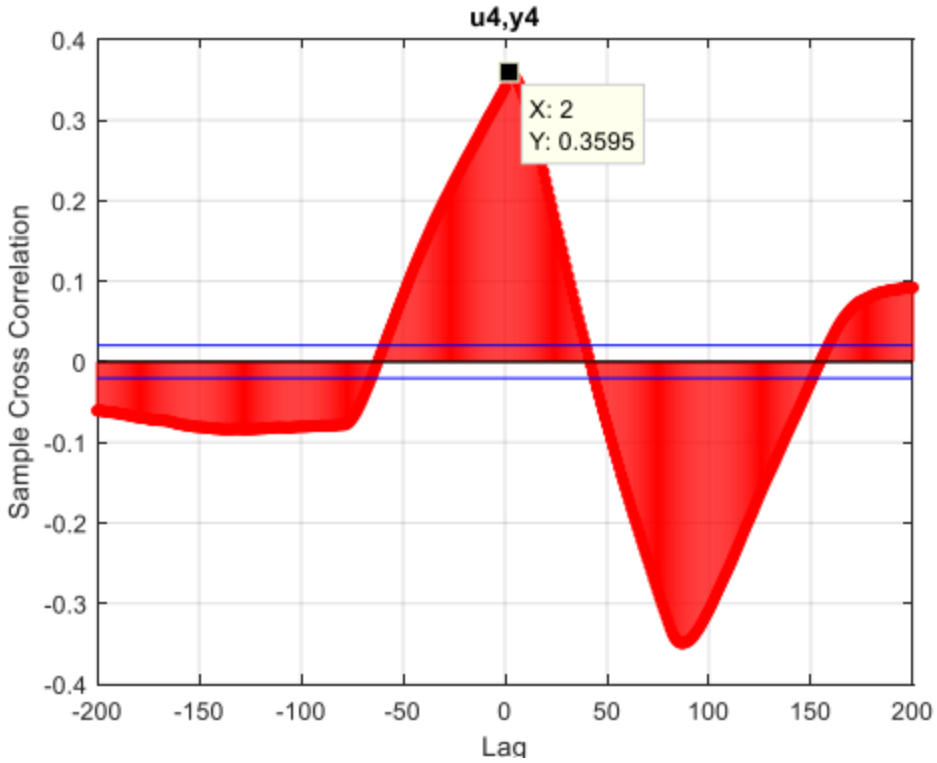


Figure 54 Correlation between fourth input and fourth output (streamgen)

To estimate the delay using the correlation method, we need to consider the dynamics where the correlation diagram exceeds the thresholds. As observed in Figure 39, using the correlation method between the second input and the first output in System 1 indicates a delay of approximately 89. However, considering that the first dynamics already have a significant value, it can be inferred that there is only one delay, mainly due to zero-order hold (ZOH).

The remaining delays for other inputs and outputs are shown in Table 2. By comparing the results of the correlation method, which is a linear system delay estimation technique, and the Levenshtein distance, which is a method for estimating dynamics and delays in nonlinear systems, it is evident that the results obtained from the two methods are entirely different.

Figure 55 The delay values for System 2 calculated using the correlation method

Input-output pair	delay
1st input and 1st output	1
2nd input and 1st output	1
3rd input and 1st output	1
4th input and 1st output	1
1st input and 2nd output	1
2nd input and 2nd output	1
3rd input and 2nd output	1
4th input and 2nd output	1
1st input and 3rd output	1

2nd input and 3rd output	87
3rd input and 3rd output	1
4th input and 3rd output	1
1st input and 4th output	1
2nd input and 4th output	1
3rd input and 4th output	85
4th input and 4th output	1

1.1) system 1

The input output plot is:

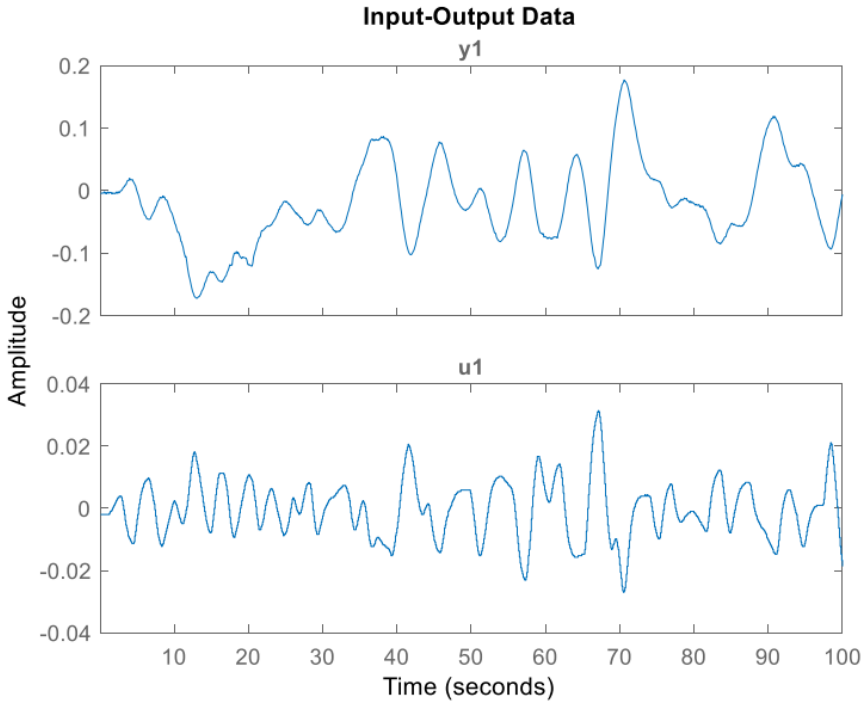


Figure 56 Input-output plot of system 1

Take the resample ratio 1 to 1:

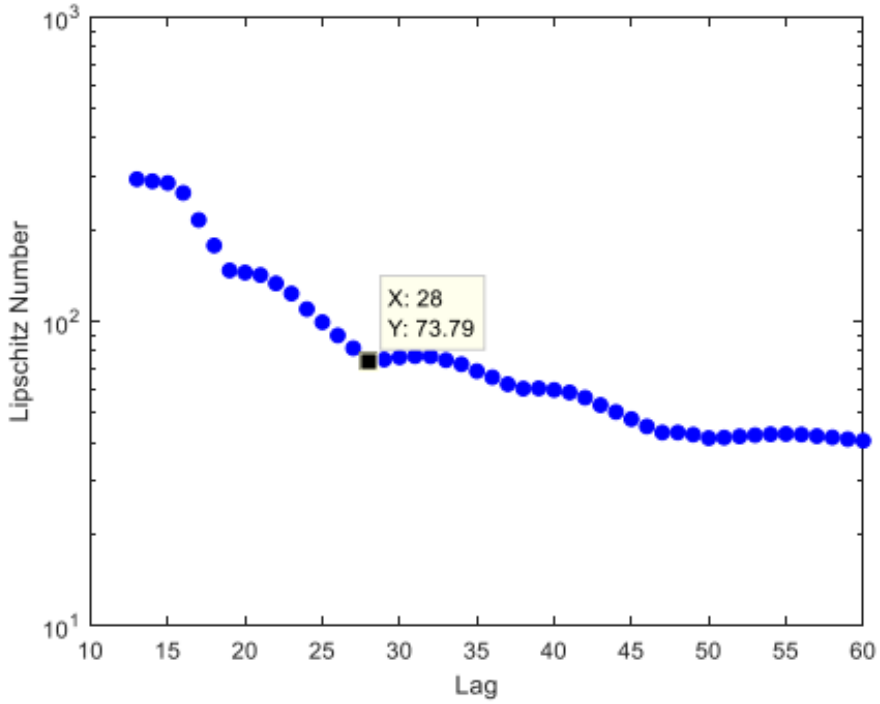


Figure 57 Lipschitz plot for 60 dynamics of the first system

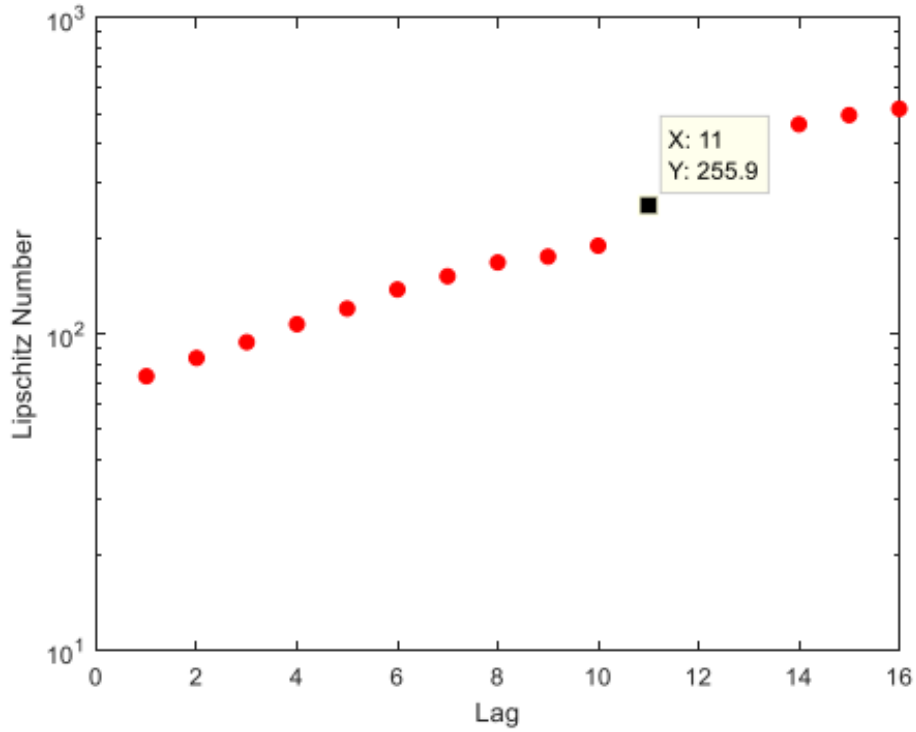


Figure 58 The delay plot of system 1

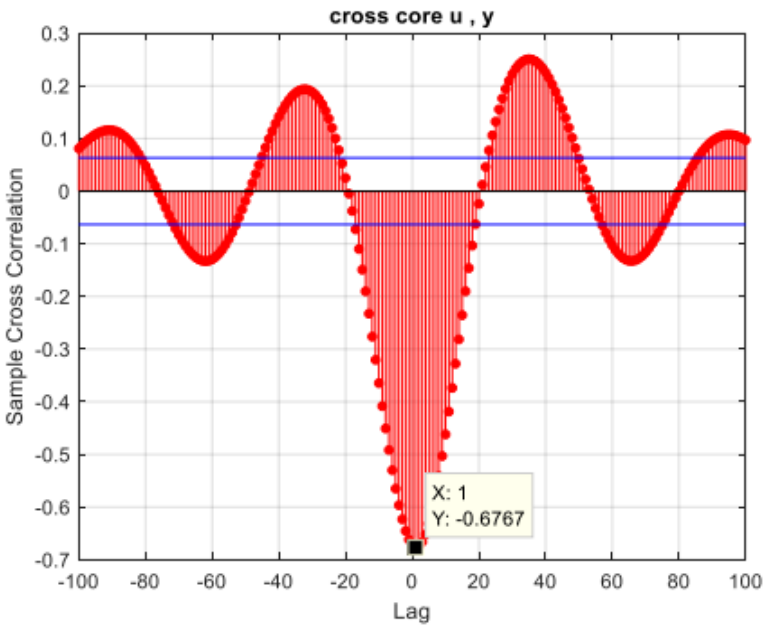
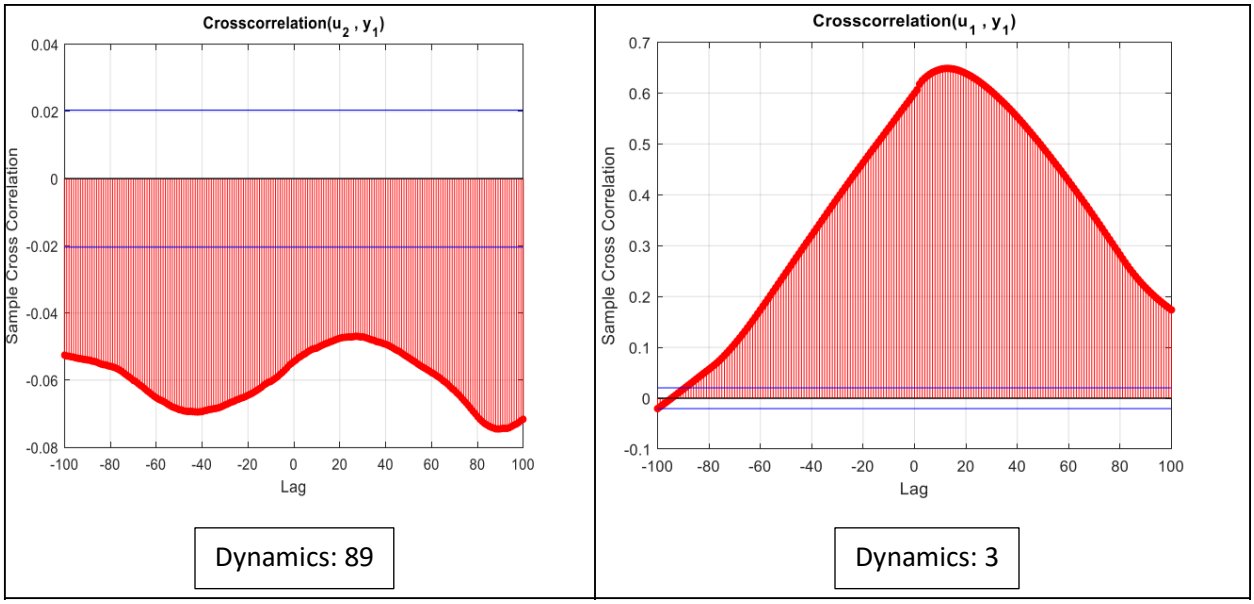


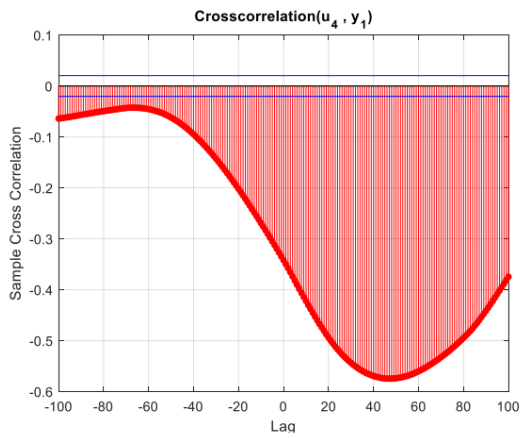
Figure 59 Correlation between input and output second system

The delay is almost 1.

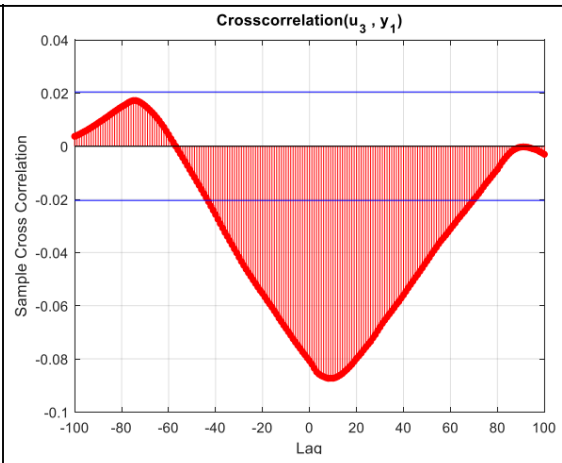
1.2) System 2

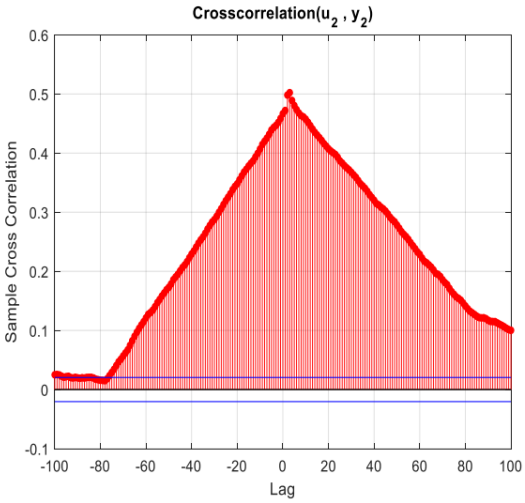
To determine the dynamics using the correlation method, the correlation plot of the output is analyzed to find the output dynamics, and the correlation plot between the input and output is examined to identify the input dynamics. Correlation is valid for linear systems. While it is not entirely accurate for nonlinear systems, we use this method to obtain an approximate view of the system dynamics. The results are as follows:



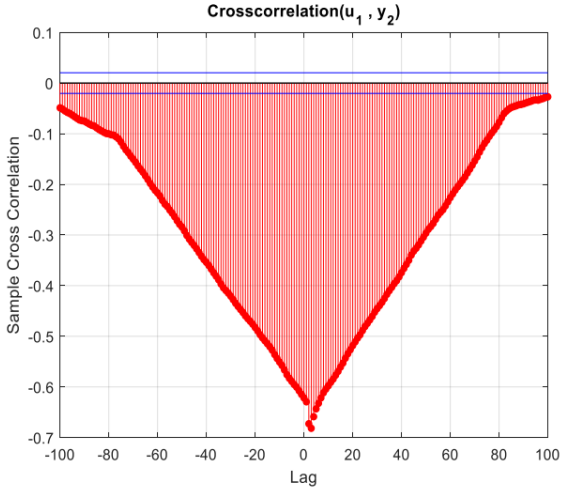


Dynamics: 47

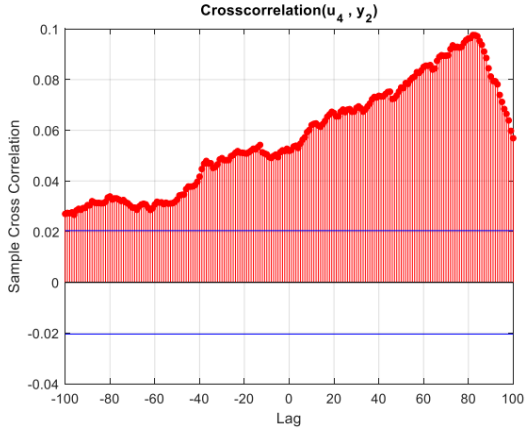




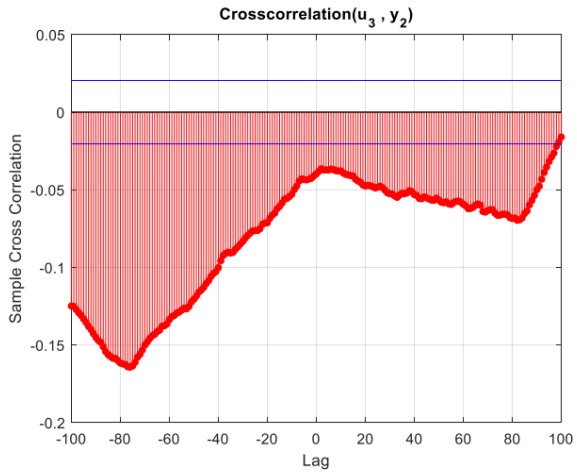
Dynamics: 3



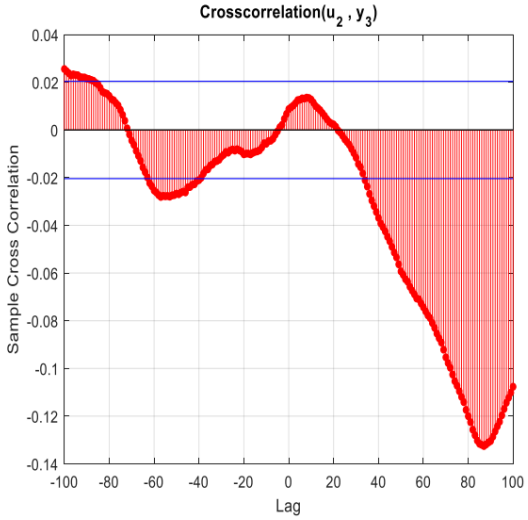
Dynamics: 3



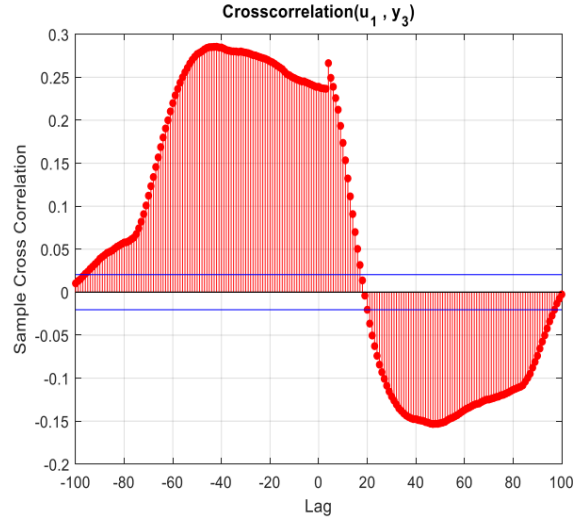
Dynamics: 82



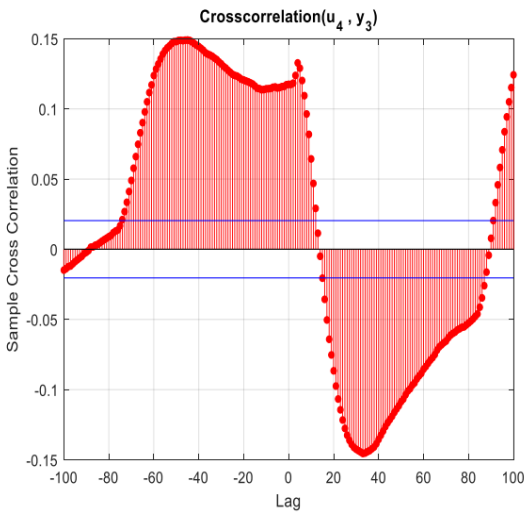
Dynamics: 82



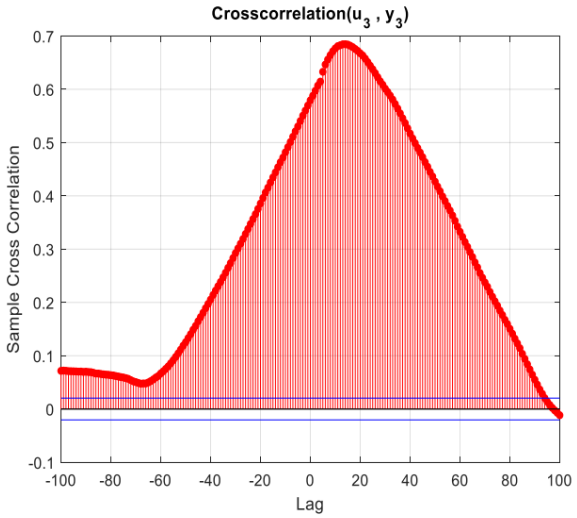
Dynamics: 87



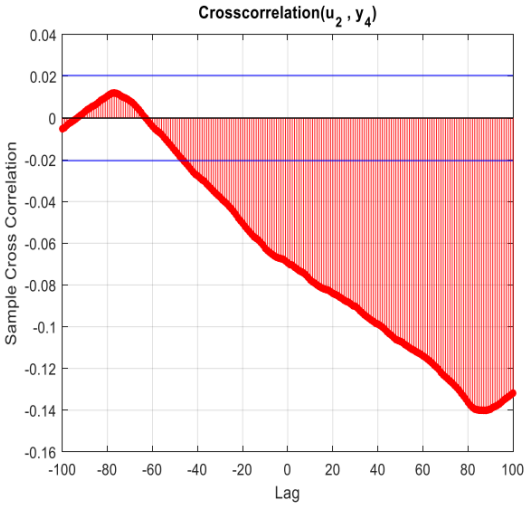
Dynamics: 47-4



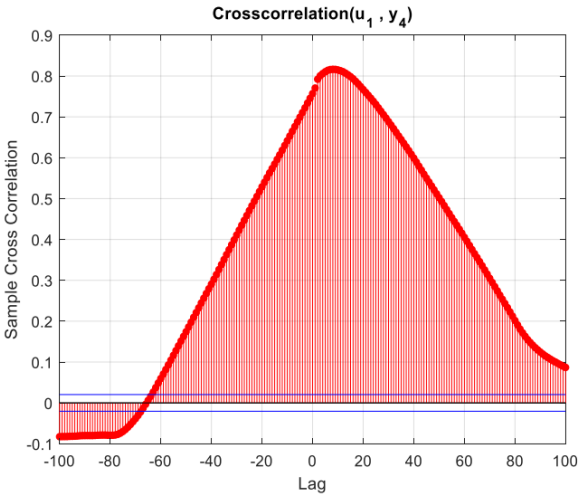
Dynamics: 33-4



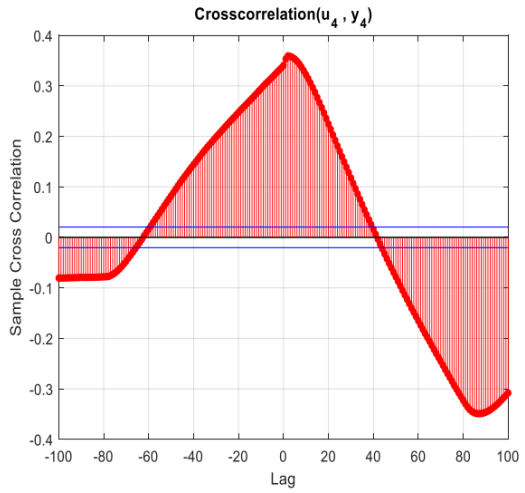
Dynamics: 14



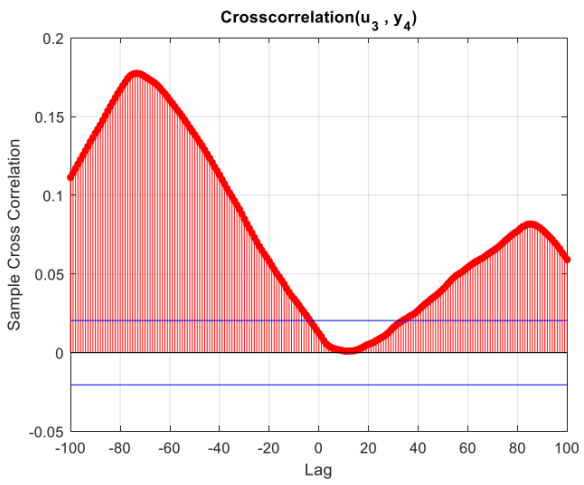
Dynamics: 88



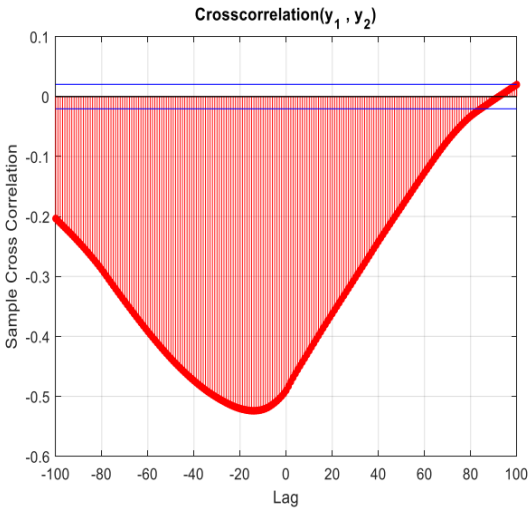
Dynamics: 8



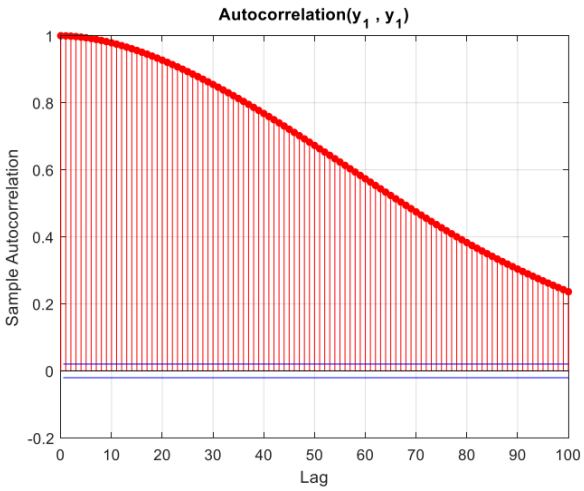
Dynamics: 87-2



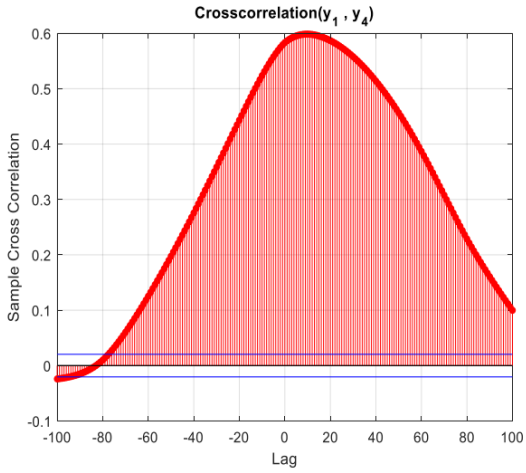
Dynamics: 85



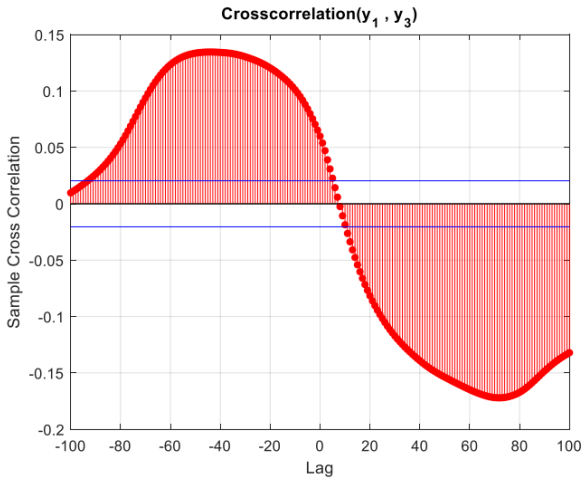
Dynamics: no dynamics

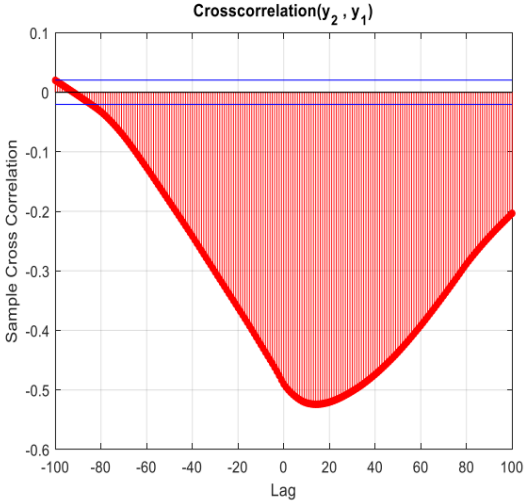


Dynamics: 1

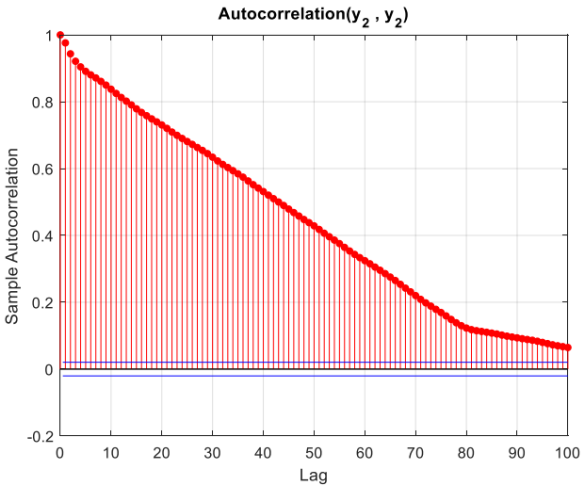


Dynamics: 10

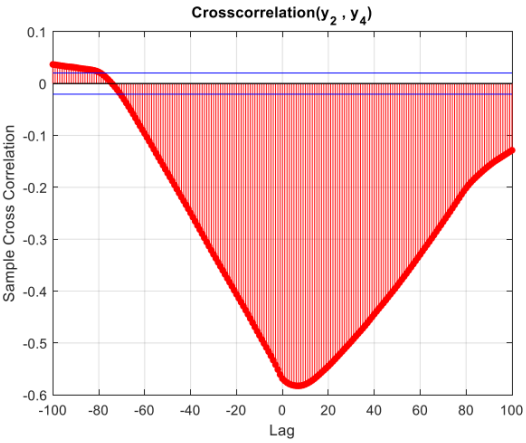




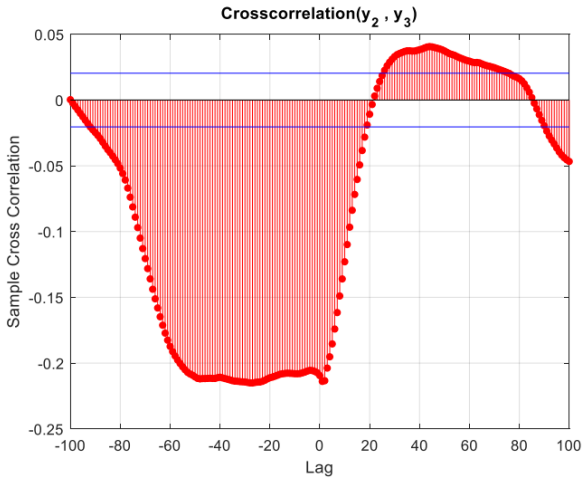
Dynamics: 14



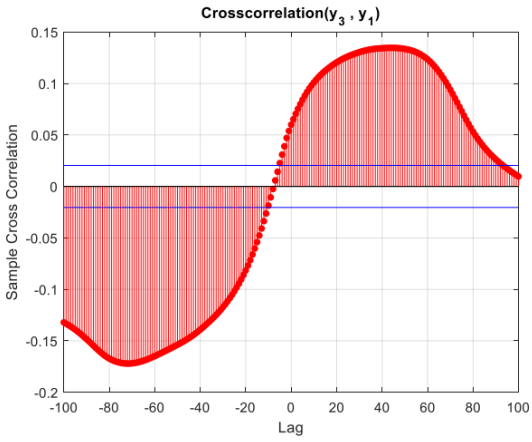
Dynamics: 1



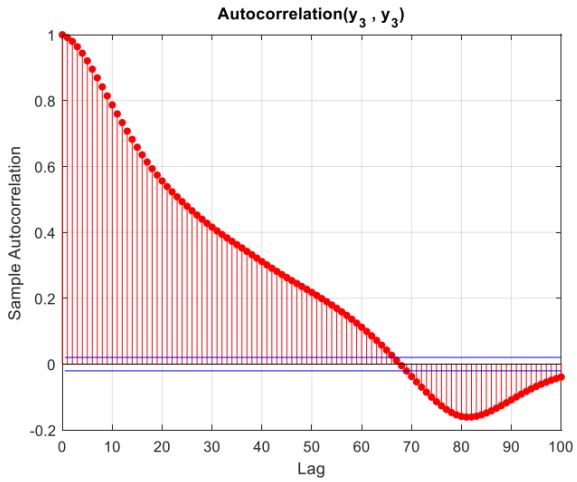
Dynamics: 7



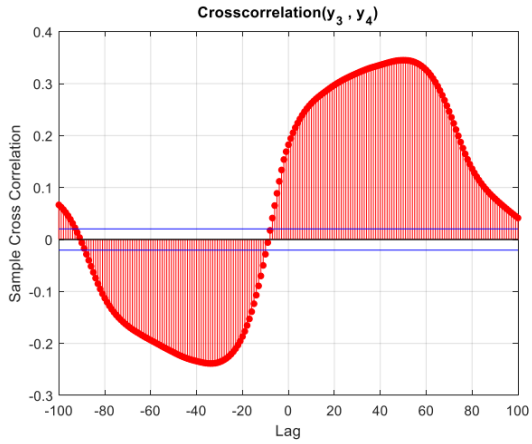
Dynamics: 44



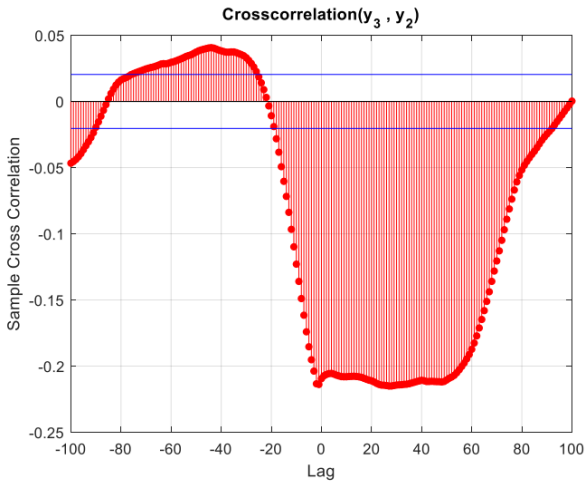
Dynamics: 44



Dynamics: 81-1



Dynamics: 50



Dynamics: 48-27

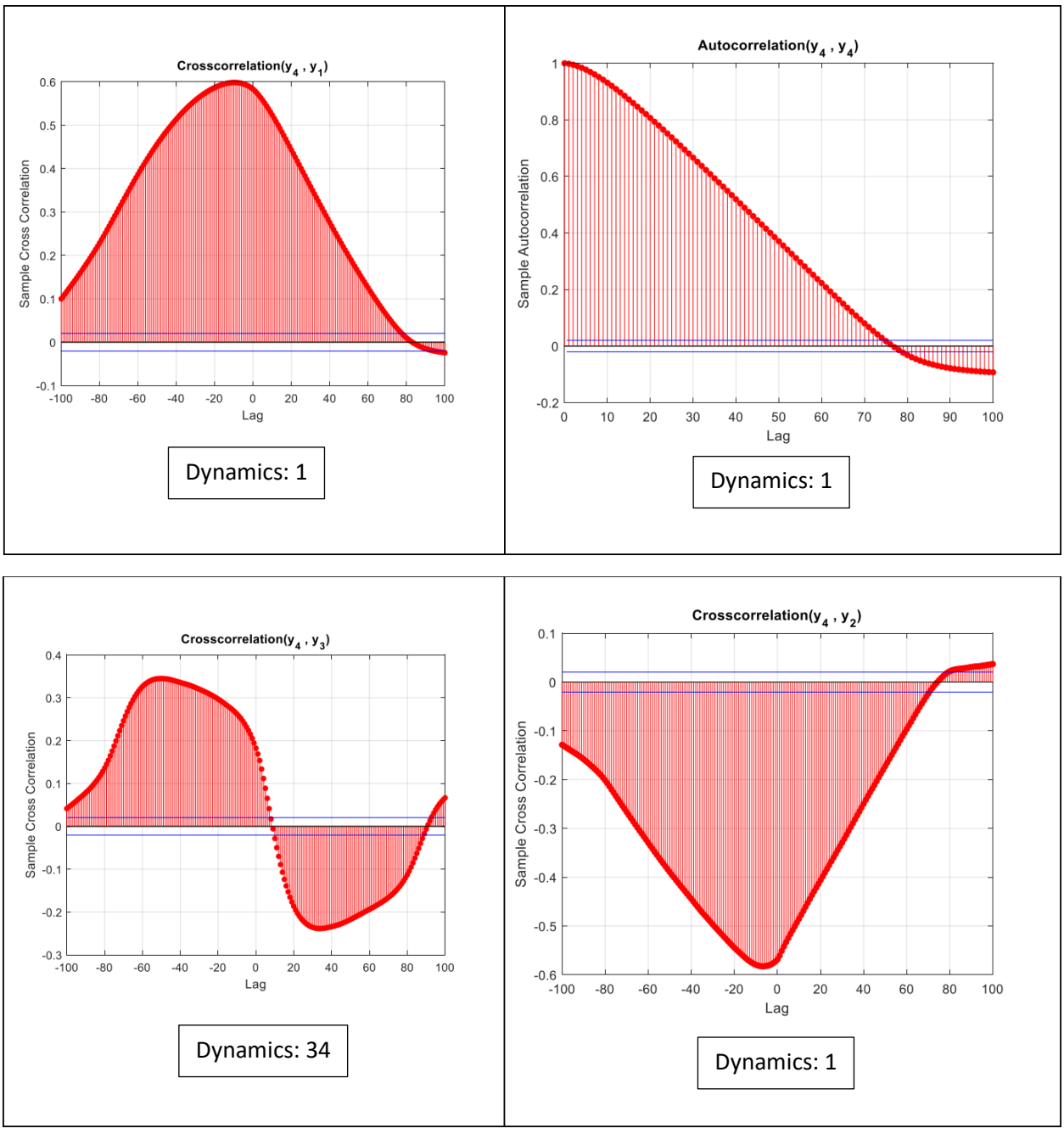


Figure 60 The dynamics of input and output pairs using correlation plot (steamgen)

The translation of the provided text to English is as follows:

Dynamics for u1: 3-4-8-13-47

Dynamics for u2: 3-87-88-89

Dynamics for u3: 8-14-82-85

Dynamics for u4: 2-4-33-47-82-87

Dynamics for y1: 1-10-72

Dynamics for y2: 1-7-14-44

Dynamics for y3: 1-27-44-48-50-81

Dynamics for y4: 1-34

With the help of these dynamics, we estimate a neural network with 6 neurons and 30 epochs for the Steamgen system. The responses are as follows. The activation function "tansig" and the Levenberg-Marquardt training method have been selected. The learning rate is set to 0.01, and for simplicity, a single-layer network has been chosen.

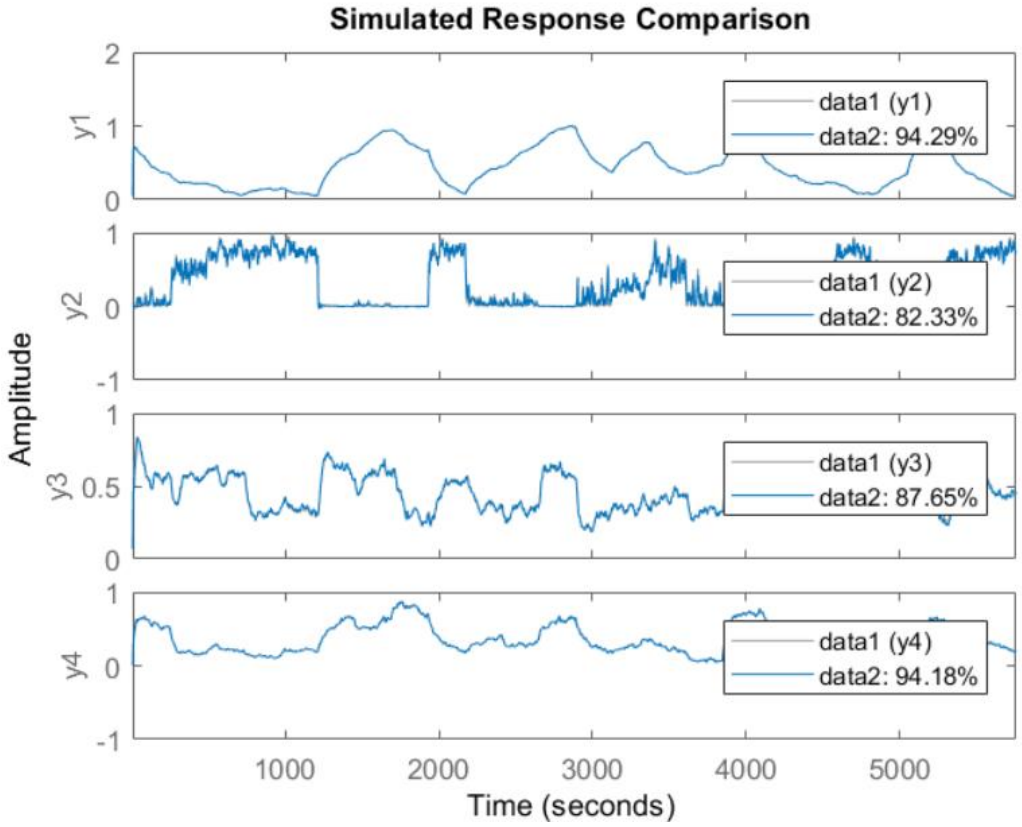


Figure 61 The output of the actual system and the estimated system (steamgen)

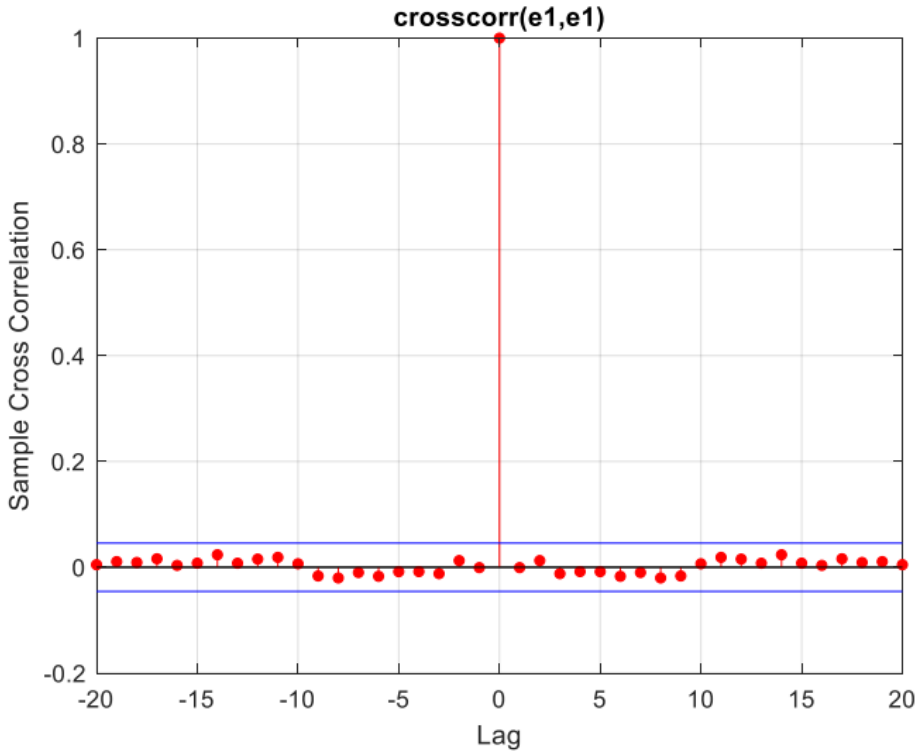


Figure 62 The error auto-correlation plot for the first output (steamgen)

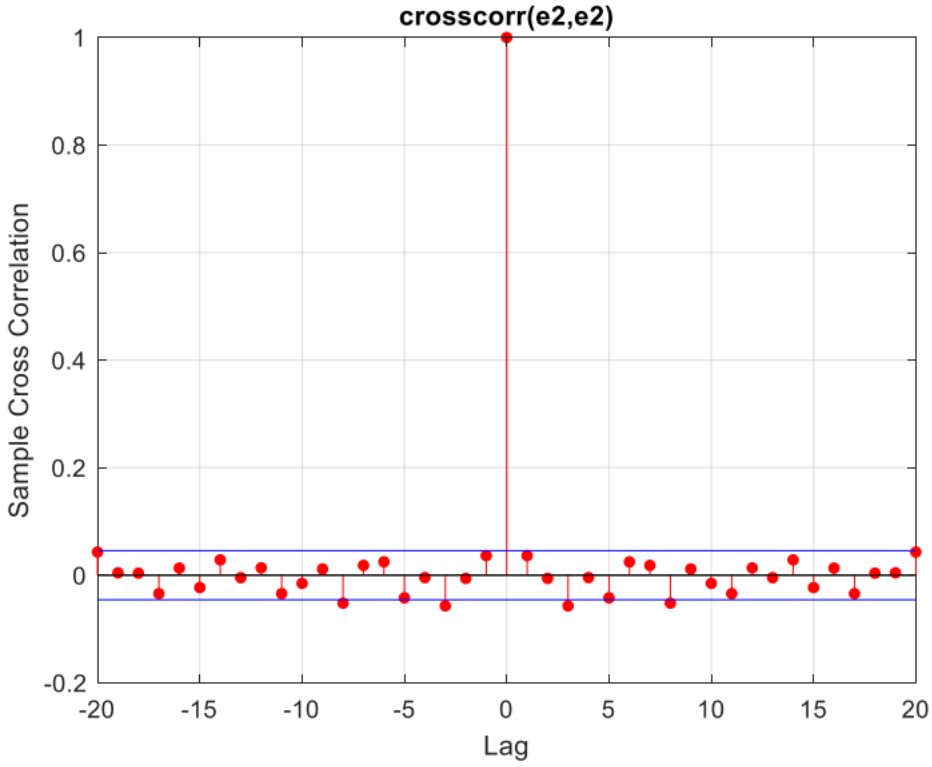


Figure 63 The error auto-correlation plot for the second output (steamgen)

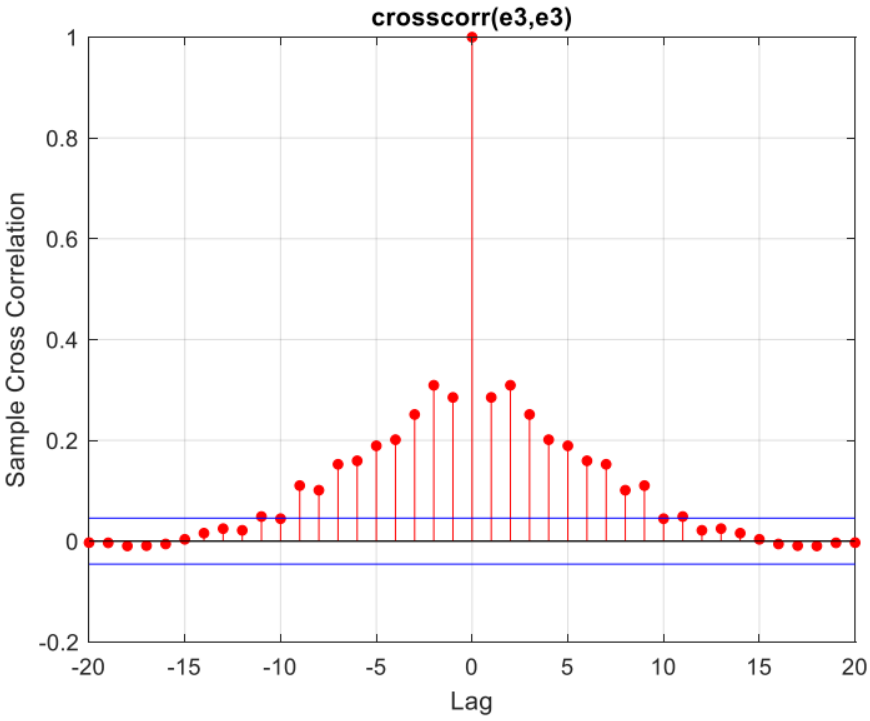


Figure 64 The error auto-correlation plot for the third output (steamgen)

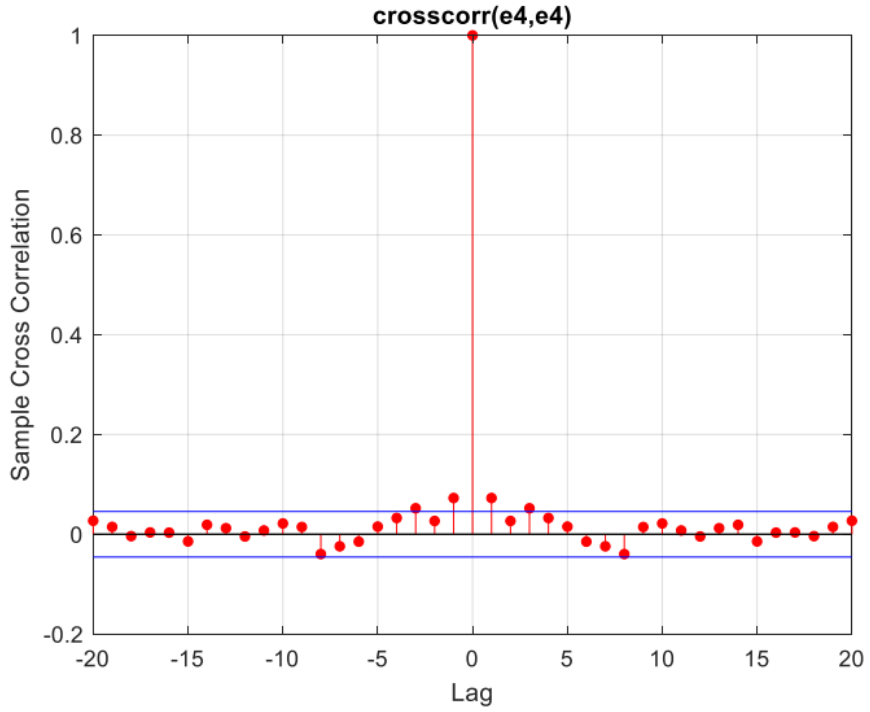


Figure 65 The error auto-correlation plot for the forth output (steamgen)

It can be observed that using the extracted dynamics, we have achieved a suitable fit percentage for each output. The error in the third output deviates slightly from whitening, but the other outputs exhibit acceptable errors (in terms of correlation and whitening).

Determining the dynamics of the system using FS

Using statistical analysis and correlation tests between the data, 33 dynamics for System 1 and 9 dynamics for System 2 have been identified. Now, using the Forward Selection (FS) method, we choose important dynamics among them. In this method, all subsets of effective dynamics are input into the network for training, and the best subset of dynamics, which minimizes the Sum of Squared Errors (SSE), is selected. The steps are explained for System 1, and they are performed identically for System 2.

Step 1: We individually input each effective dynamic into a single-layer neural network (MLP) with 6 neurons, 30 training epochs, one input, and four outputs. Then, we plot the Mean Squared Error (MSE) for the test data. The dynamic with the lowest MSE is selected as the first effective dynamic.

Step 2: Now, from the remaining dynamics, one is selected, and alongside the previously chosen effective dynamic, a network with two inputs and four outputs, with the same number of neurons as the previous step, is designed. This process is repeated for all remaining dynamics. The network with the lowest MSE for the test data is the winner, and its second input is the second effective dynamic.

Step 3 to n-1: This process is continued until the point where we have a network with the same number of inputs and four outputs as all candidate dynamics, and the number of neurons is the same as in the previous steps.

Step n: We examine the MSE error plot for the entire process at each step. We select the dynamics chosen up to that point as the main dynamics of the system wherever there is no significant change.

Here, we have gone through all the steps, and in the last step, we observe that the final stage shows the same trend as the initial one. Let's consider how many steps we need to monitor the process. Therefore, in the last step, where the minimum MSE is plotted for each stage, we start by examining it from the beginning.

According to Figure 62, for System 1, the minimum error doesn't change significantly from stage 11 onwards, and it even increases in higher stages. Therefore, it is sufficient to run the algorithm up to stage 11 and consider the selected dynamics up to stage 11. The order of regression selection is also provided based on the algorithm stages.

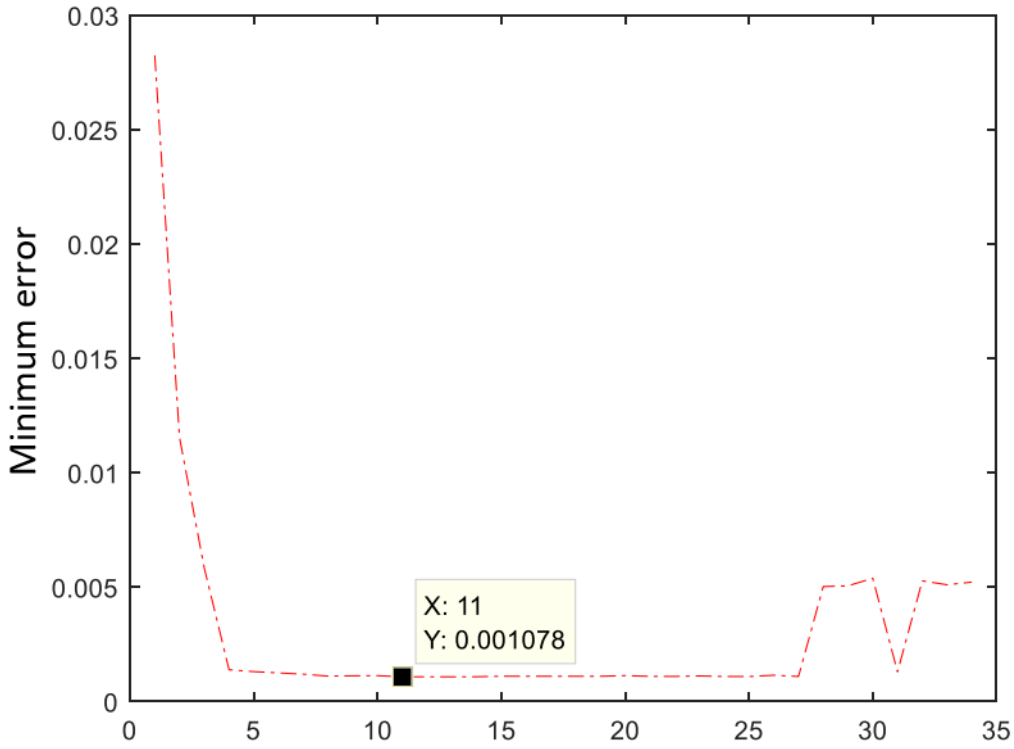


Figure 66 The MSE change in every step of FS algorithm (ssteamgen)

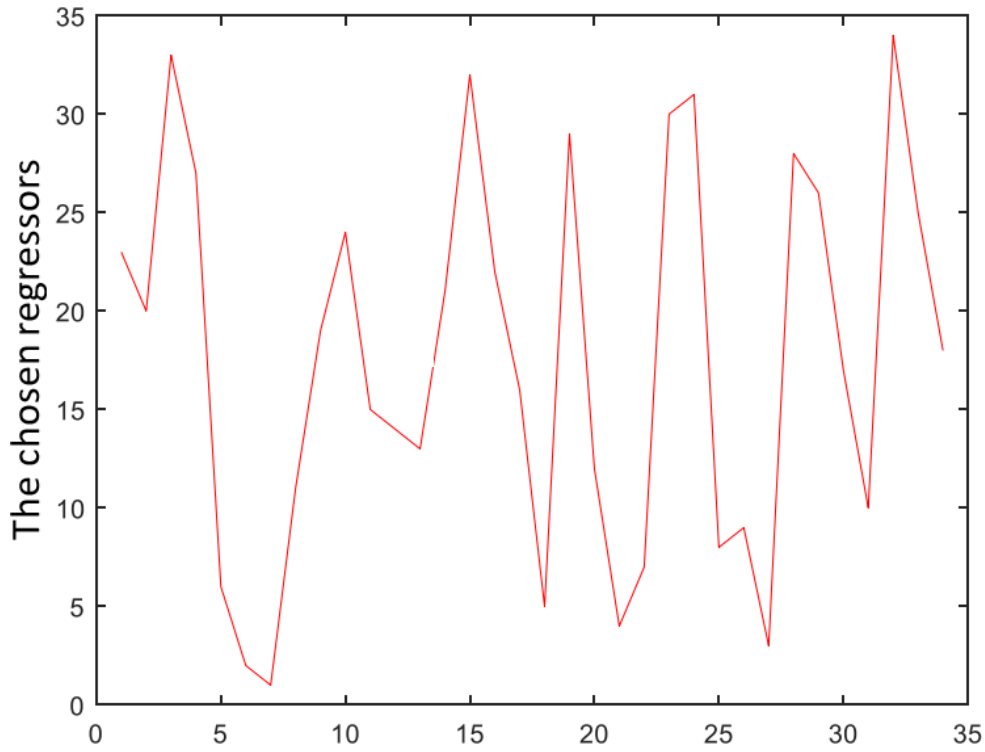


Figure 67 The chosen regressors based on the iterations using FS algorithm

The important regressors up until step 11, are as follows:

23,20,33,27,6,2,1,11,19,24 and 15, which represent:

$$u_1(t-3), u_1(t-4), u_2(t-3), u_3(t-14), u_4(t-4), u_4(t-87), y_1(t-1),$$

$$y_2(t-1), y_2(t-7), y_3(t-1), y_4(t-1)$$

Considering Figure 66, for System 2, the minimum error does not change significantly from stage 4 onwards, and it even increases in higher stages. Therefore, it is sufficient to run the algorithm up to stage 4 and consider the selected dynamics up to stage 4. The order of regression selection is also provided based on the algorithm stages.

Now, we input these selected regressors into a neural network for the system and compare the response with the previous state. The plot of the actual output versus the predicted output for the system is shown in the figure below.

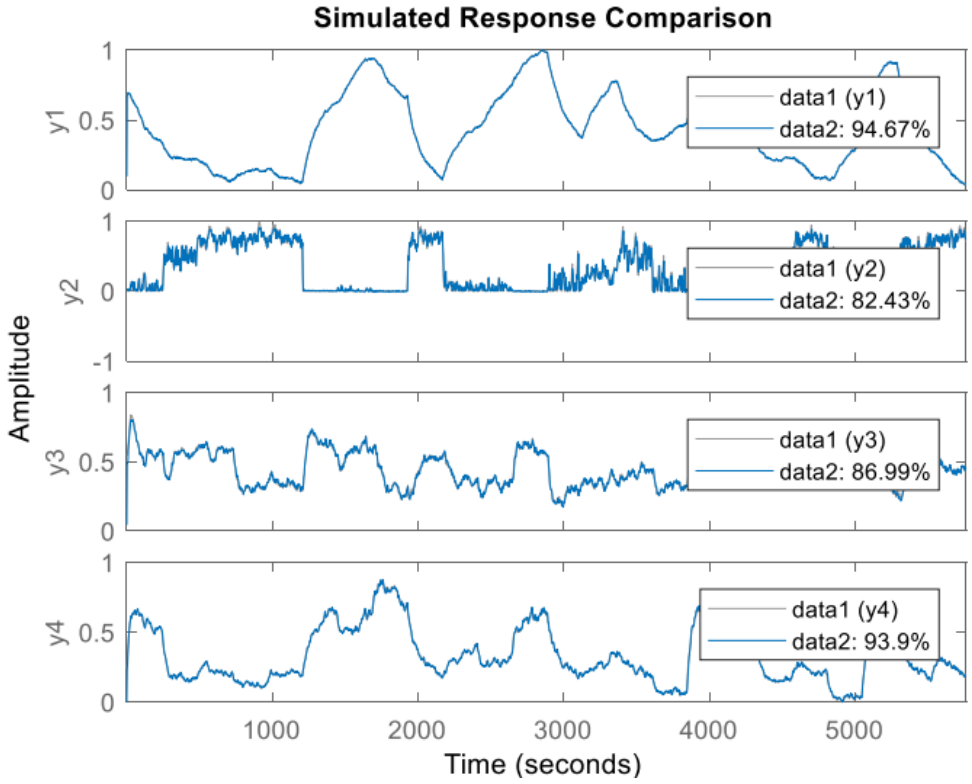


Figure 68 The output if the actual system vs estimated system using FS method (steamgen)

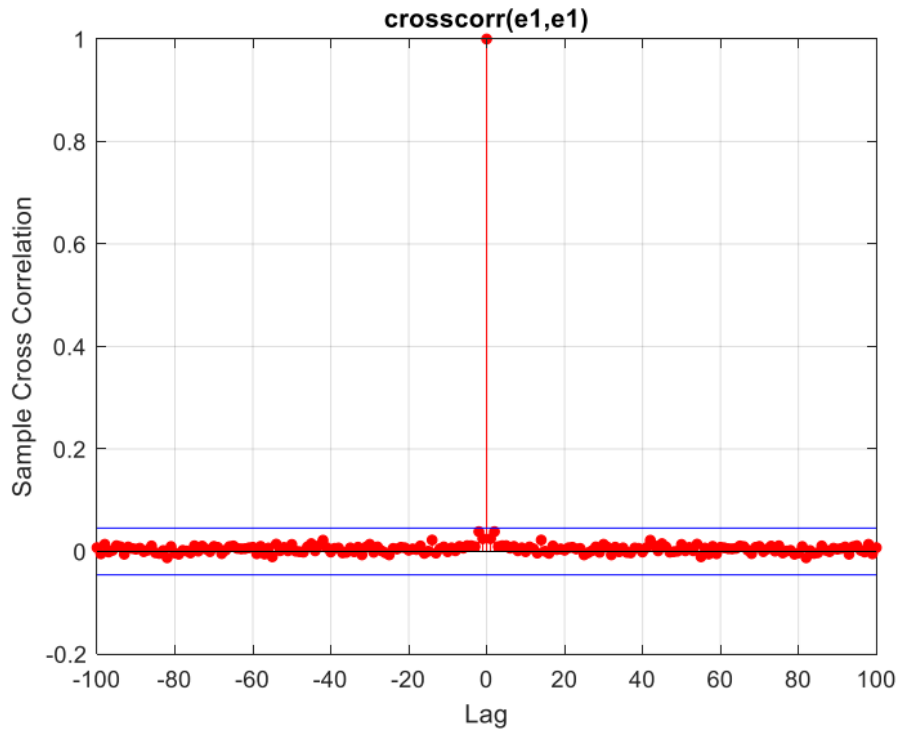


Figure 69 auto-correlation between first output error (steamgen)

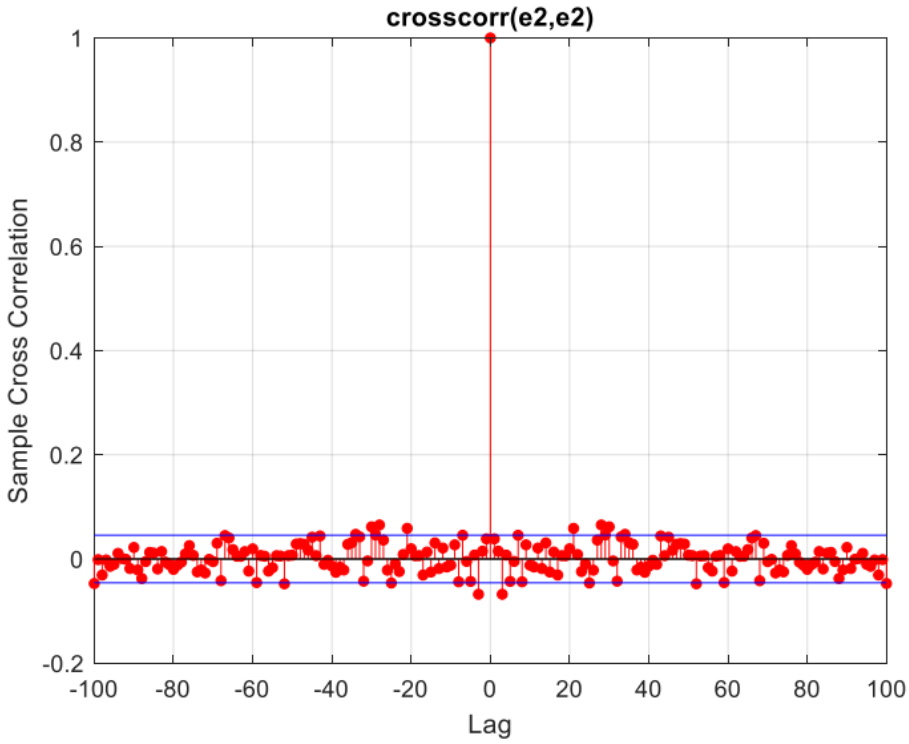


Figure 70 auto-correlation between second output error (steamgen)

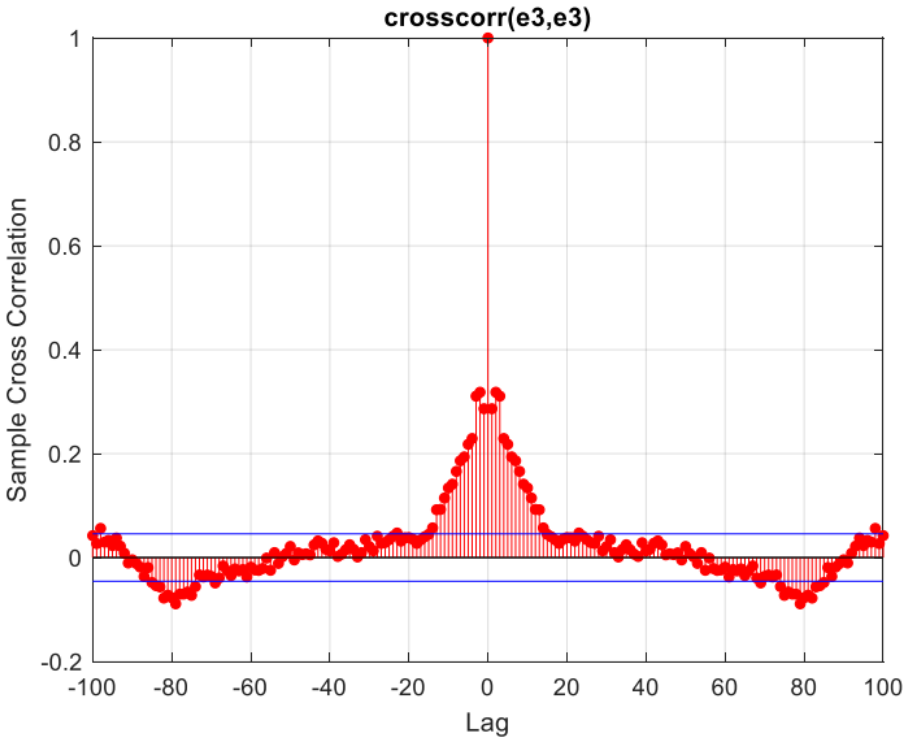


Figure 71 auto-correlation between second output error (steamgen)

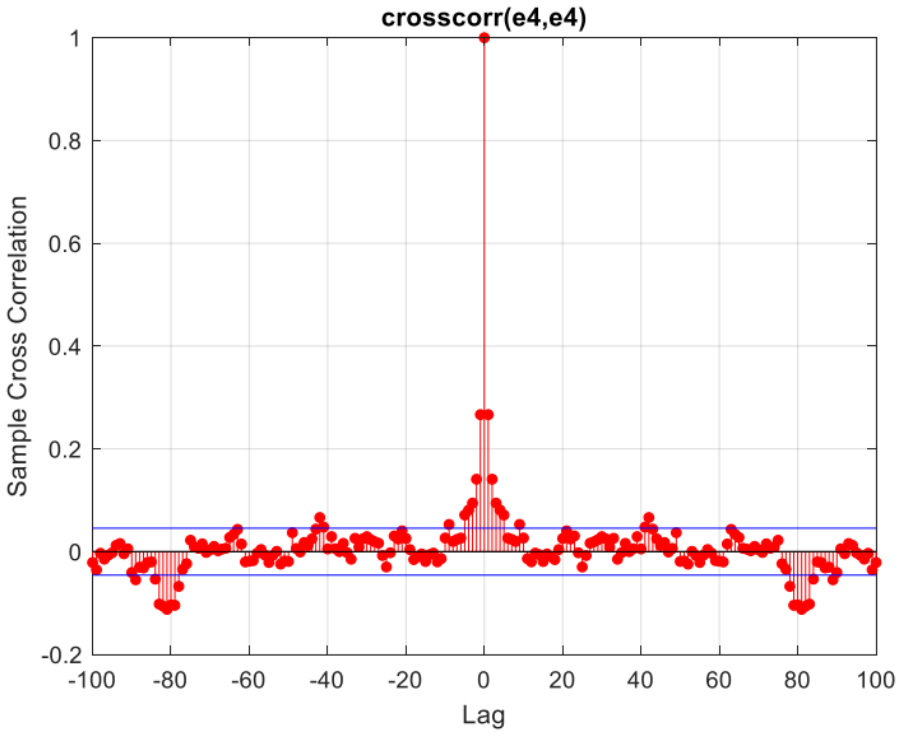


Figure 72 auto-correlation between second output error (steamgen)

The percentages obtained are similar to the previous case, indicating that the reduced regressors, or in other words, the results obtained with the F.S method, are suitable. However, overall, this method is suboptimal unless the regressors are orthogonal. It is possible that better regressors could be obtained to reduce errors and improve fits further. The errors are also similar to the previous state in terms of whitening, but for the third output, there is a slight deviation from whitening.

The "Feature Selection (GA)" algorithm has several key differences from other methods, making it more reliable. These differences include:

It uses encoded decision variables rather than the variables themselves.

It employs random sampling rather than deterministic rules.

It starts with a set of random points.

It only utilizes the output information of the fitness function, not derivatives or other auxiliary information.

In this method, the **ga** command is used with the following syntax:

x = ga(fun, nvars, A, b, [], [], lb, ub, nonlcon, IntCon, options)

The parameters used in this algorithm, based on MATLAB's help function, are:

fun: Instead of using an MLP network similar to the structure used in the FS method, a comparable network is employed for a meaningful comparison between methods.

This algorithm is applied to perform feature selection using the ga command in MATLAB.

The Nvars parameter represents the total number of inputs and outputs, which is 8 in this case.

The A and b parameters, as explained in MATLAB's help documentation, should be set in the form of A and b arrays since there is no linear relationship.

The LB and UB parameters determine the lower and upper bounds of the dynamics depths, respectively. In this case, the lower and upper bounds are set to 0 and 40, respectively.

The IntCon parameter is an array specifying that depths should be integers, so an array from 1 to 4 is entered.

The generation option is set to 4 to calculate 4 generations.

The order of dynamics for inputs and selected outputs is given by the Dynamics array as follows:

Dynamics = [41 41 2 41 1 1 1 1];

These dynamics correspond, in order, to the first four inputs and then the first four outputs. It is important to note that in this method, only 8 dynamics are considered, one for each input and output, whereas in the FS method, more dynamics were used.

The identification of the MLP structure using these dynamics is performed similarly to the previous structure. However, it is crucial to emphasize that in the FS method, more dynamics were employed, while in this method, only 8 dynamics are considered for each input and output.

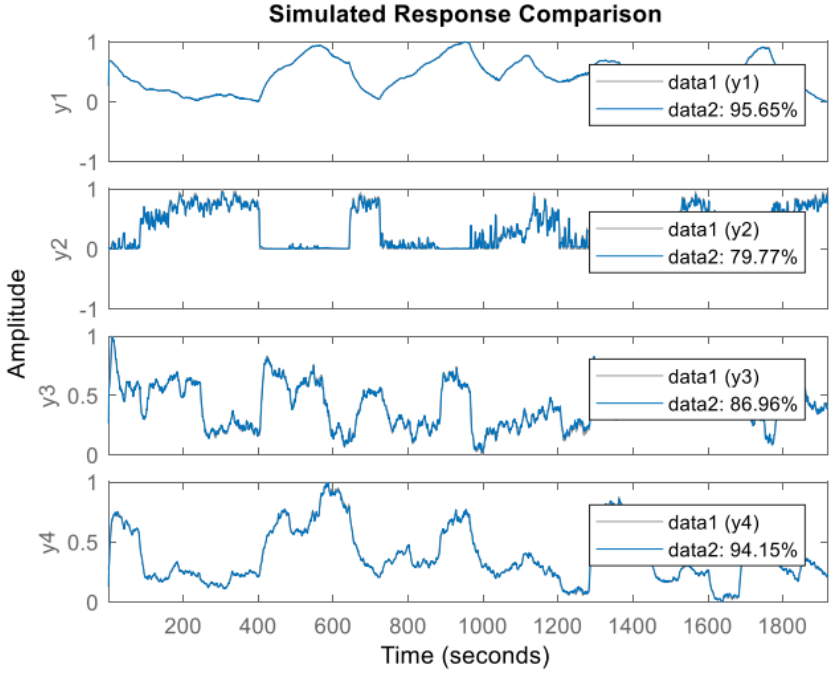


Figure 73 The actual system output and estimated system output (GA regressors)

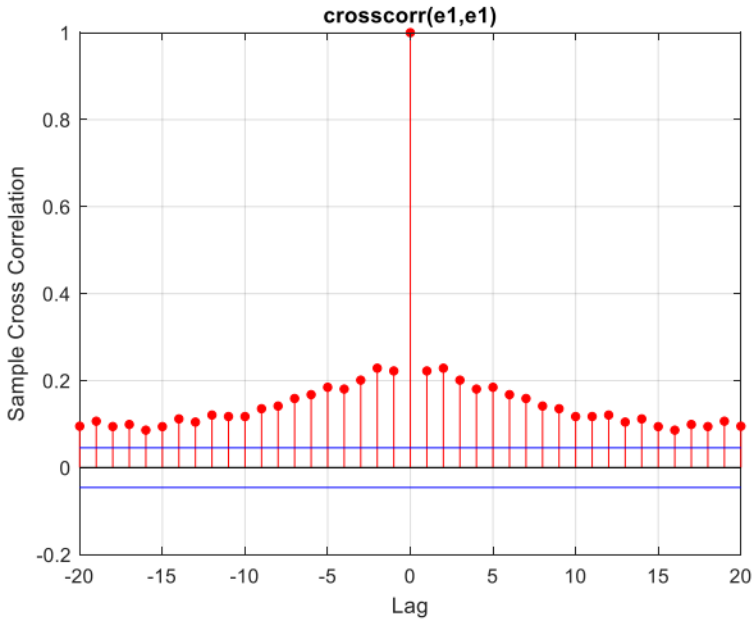


Figure 74 Correlation of first input error (GA)

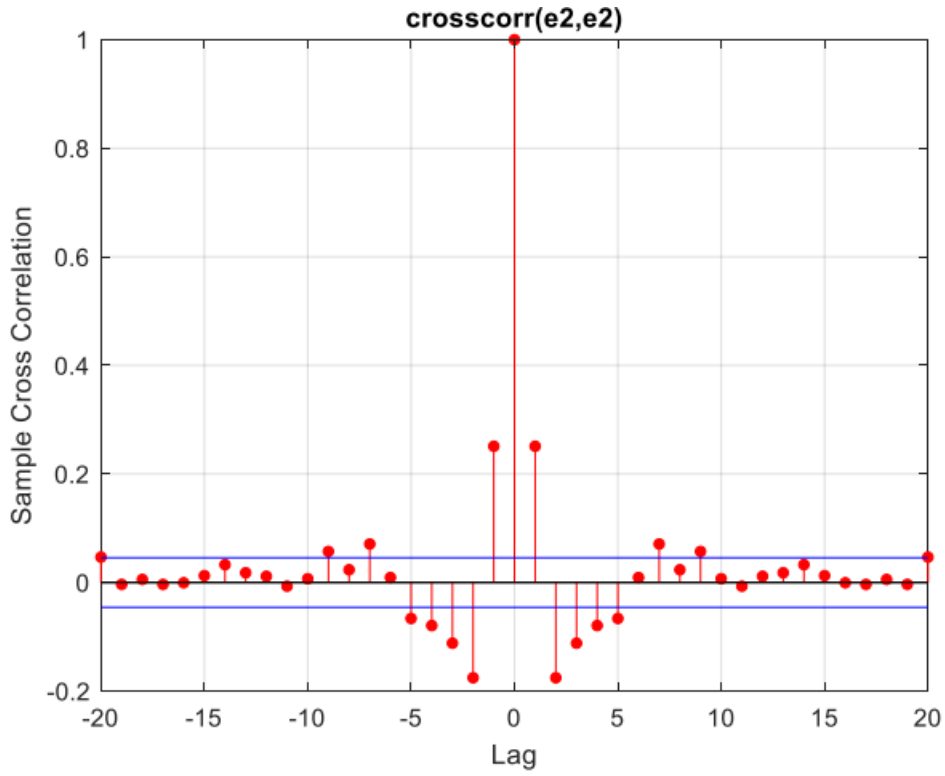


Figure 75 Correlation of second input error (GA)

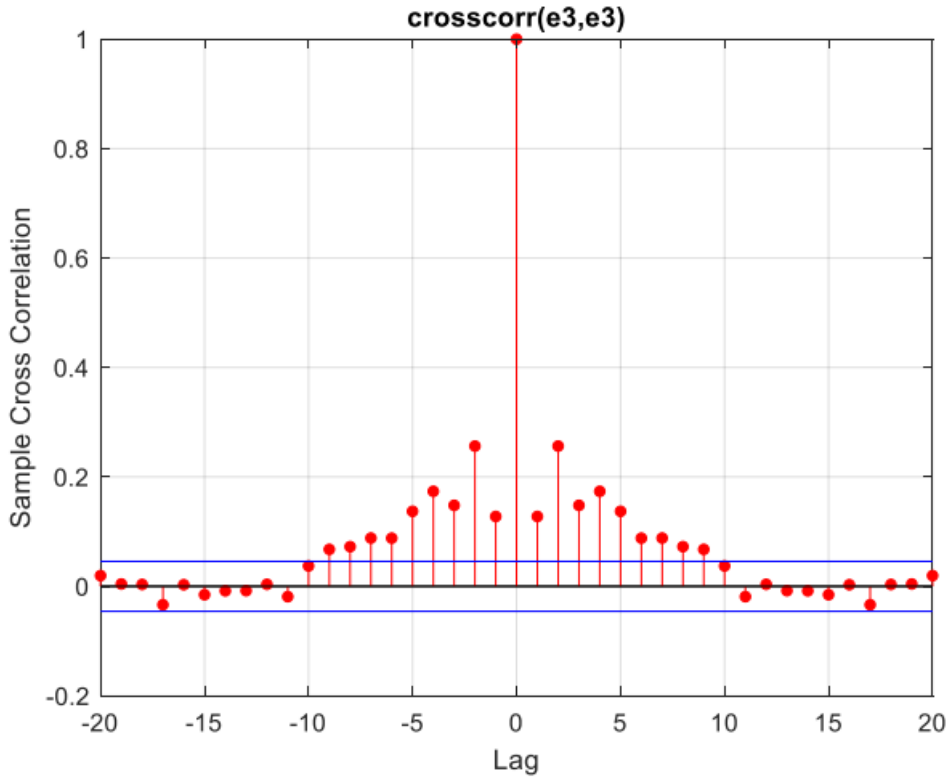


Figure 76 Correlation of third input error (GA)

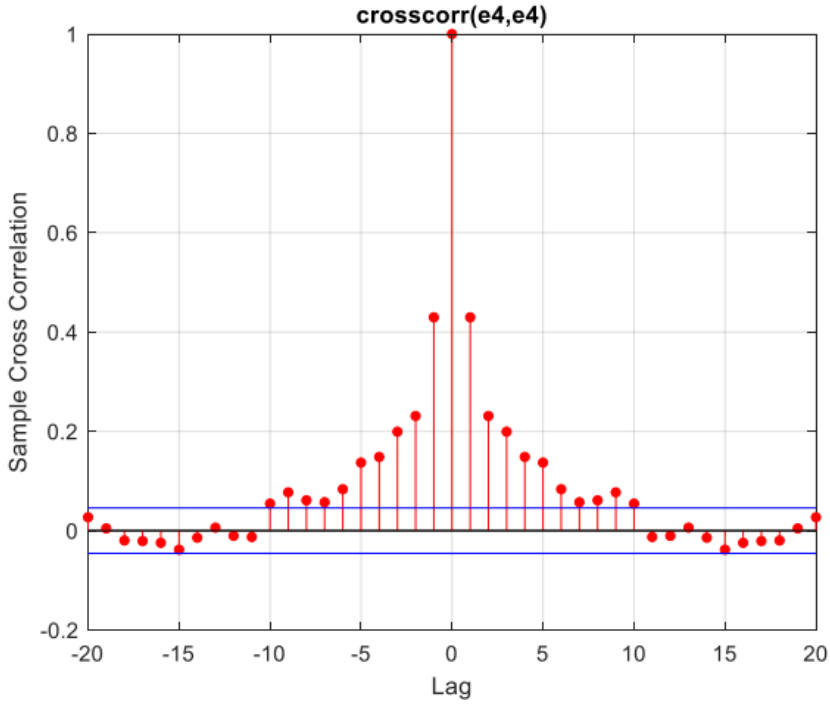


Figure 77 Correlation of fourth input error (GA)

It can be observed that similar fitness percentages have been obtained compared to the two previous methods. However, the errors at the end for each output have slightly deviated from whitening, indicating that the dynamics are not optimal. The reason for this is explained before; GA is suboptimal.

1.2) System 1

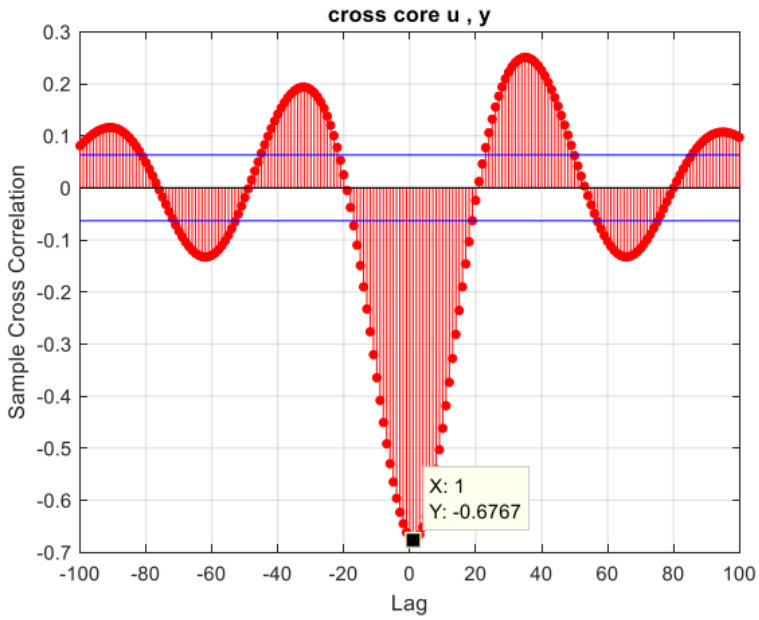


Figure 78 correlation of input and output (system 1)

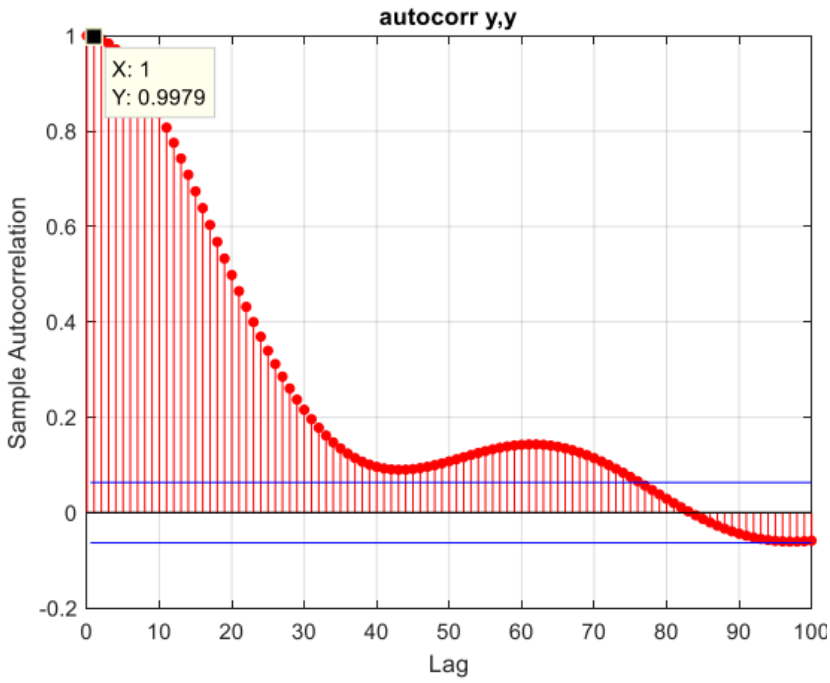


Figure 79 Auto-correlation the output (system 1)

If we consider the dynamics u(t−1),u(t−2),u(t−3),y(t−1),y(t−2),y(t−3), and use a neural network with 4 neurons in the hidden layer, employing the hyperbolic tangent sigmoid activation function (tansig), and utilizing the Levenberg-Marquardt training method, we have the following configuration.

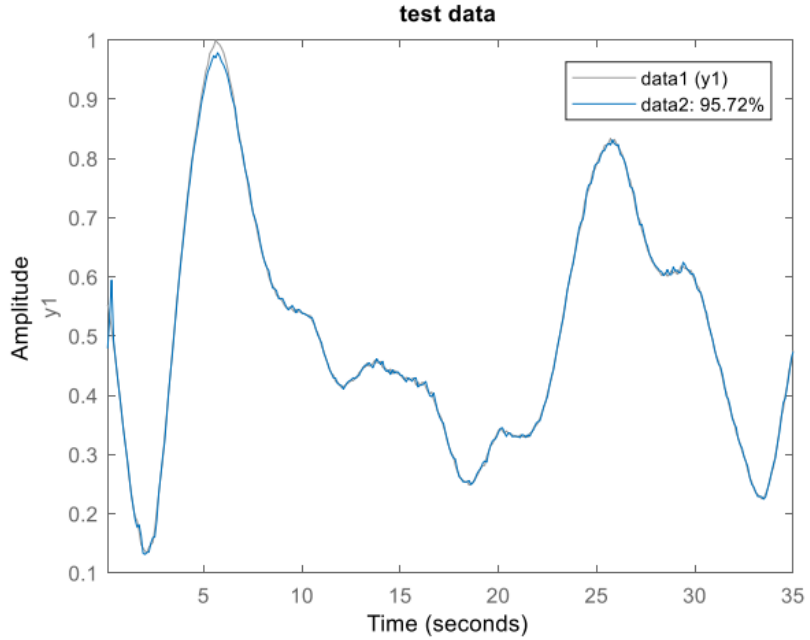


Figure 80 The actual system and the estimated system output (system 1)

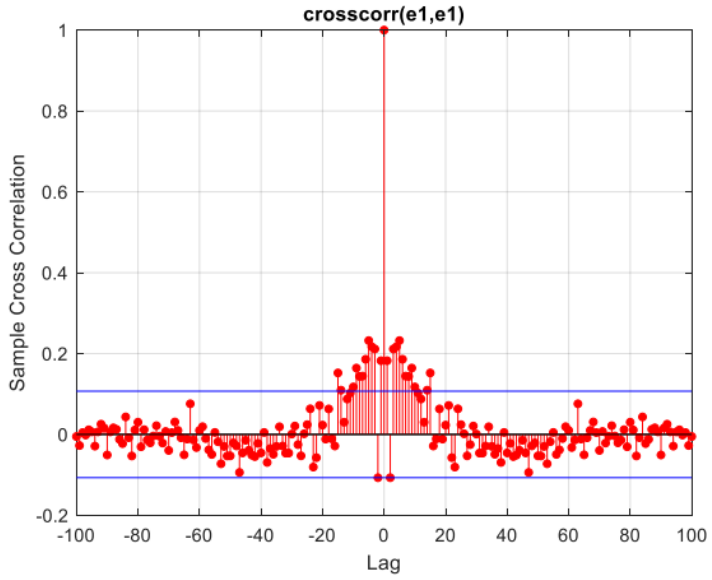


Figure 81 The auto-correlation of error (system 1)

Now we use the FS algorithm:

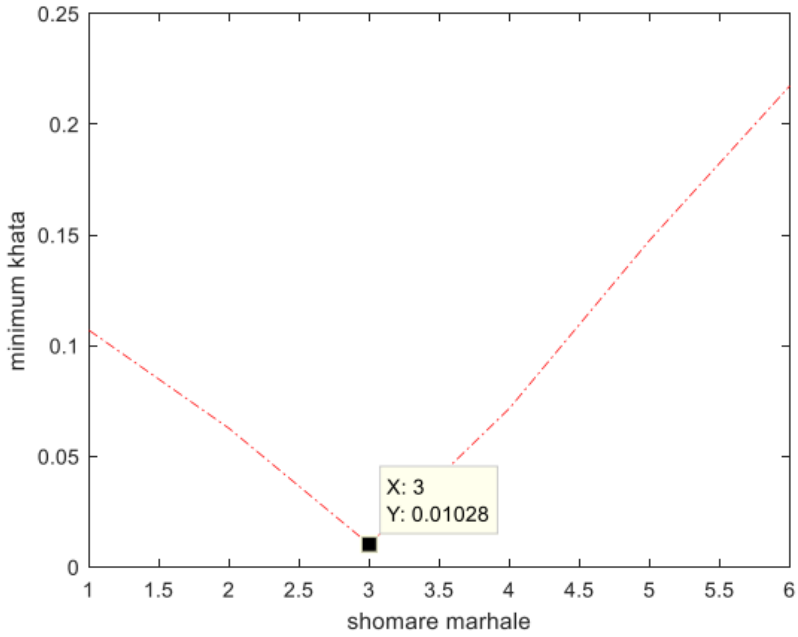


Figure 82 MSE changes in different steps (system 1)

Choose 3 regressors would suffice:

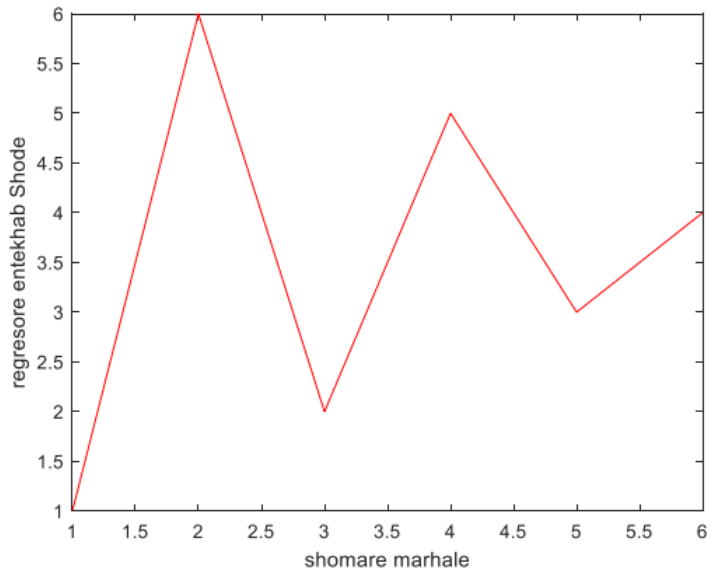


Figure 83 Regressors chosen from FS algorithm (system 1)

Regressors 1, 6, and 2 are selected, corresponding to $u(t-1), u(t-2)$, and $y(t-3)$, respectively. Now, with these regressors, we train a neural network with the previous structure.

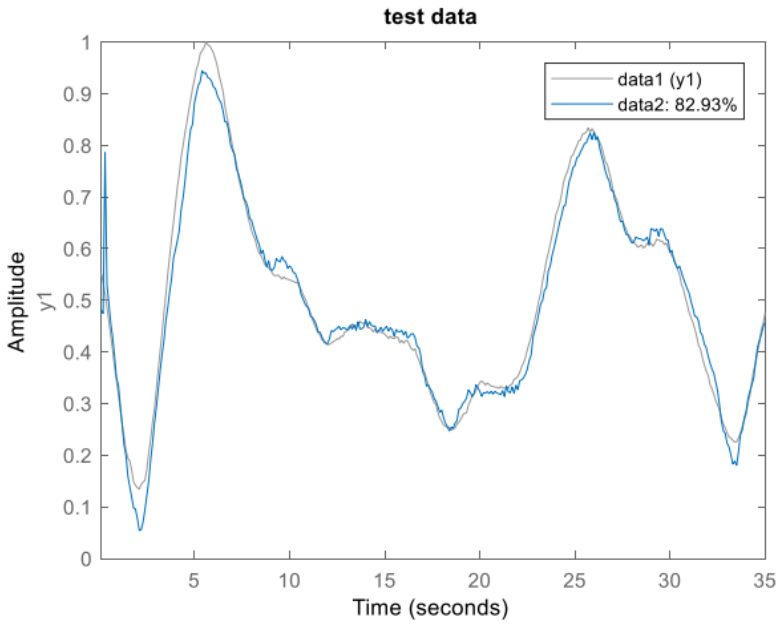


Figure 84 The output of the actual system and estimated system (system 1)

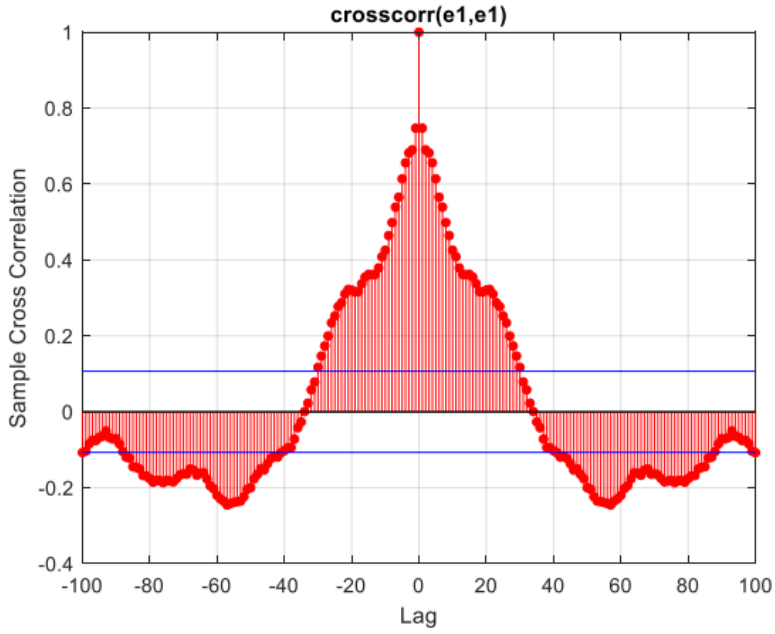


Figure 85 The auto-correlation of error using FS algorithm (system 1)

It seems that based on the fitness percentage and error correlation, the first regressors perform better than F.S. Now, let's find regressors using the genetic algorithm.

Considering the mentioned explanations, we proceed.

```
ga_opts = gaoptimset('Generations',4,'display','iter');
[x_ga_opt, err_ga] = ga(fitnessfcn, 2,
[],[],[],[],zeros(1,2),5*ones(1,2),[],1:2,ga_opts);
```

the dynamics will be:

Dynamics=[5 2]

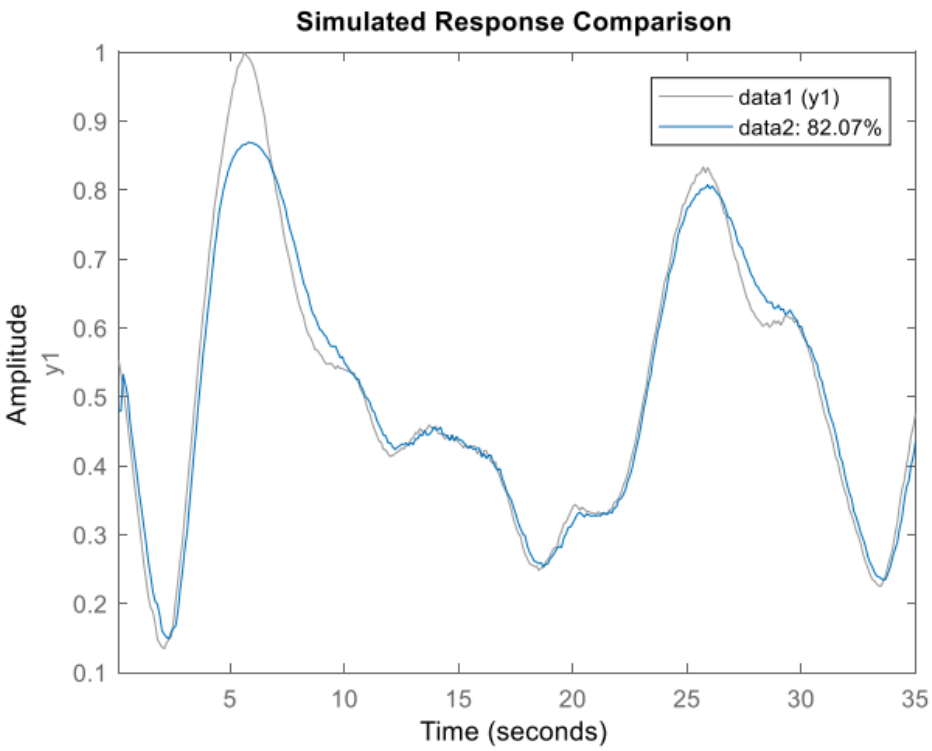


Figure 86 The output of the actual system and the estimated system with regressors acquired by GA (system 1)

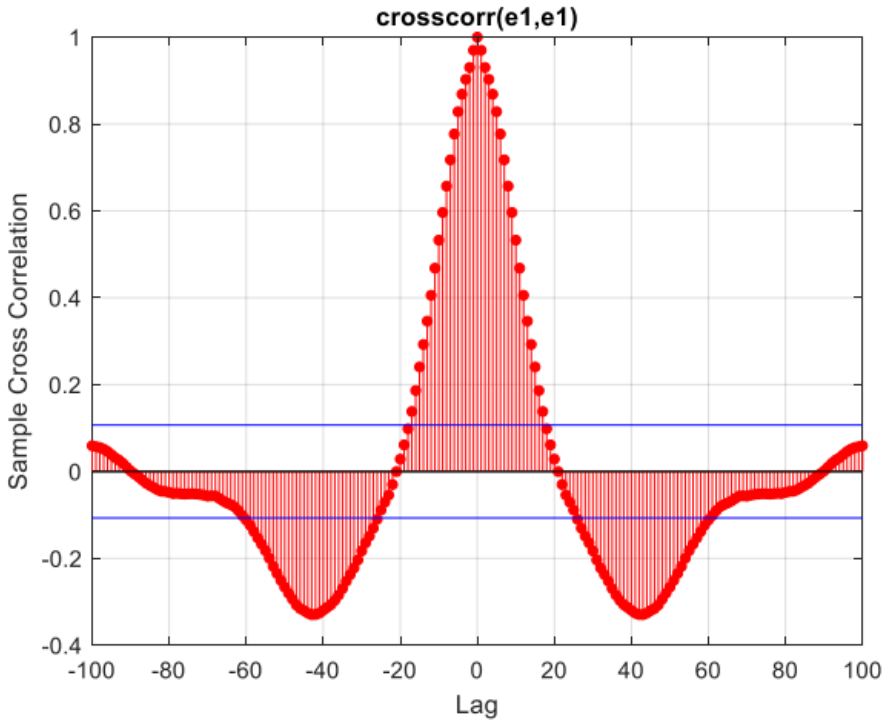


Figure 87 Correlation of error

It appears that the initially selected regressors are more suitable considering the fitness percentage and error whitening.

1.3) System 2

Initially, an explanation about Identification and Validation is provided to clarify the difference between these two methods. Several models for identifying nonlinear dynamic systems have been introduced, and we focus on examining the NOE and NARX models. These two models are schematically represented as follows:

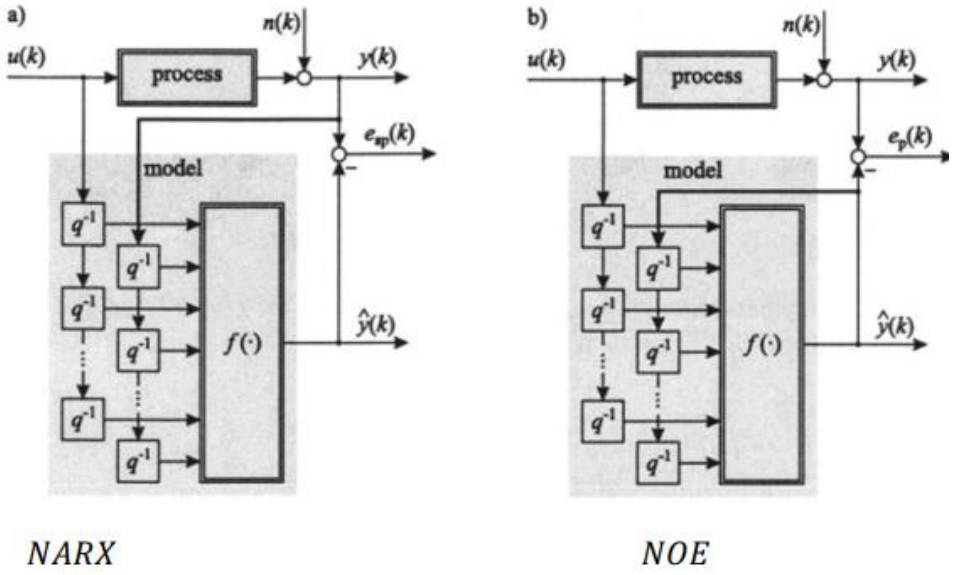


Figure 88 The schematics of NARX and NOE models

Models that utilize the past actual outputs as inputs are referred to as "predictor" models. The NARX model falls into this category.

Models that use the past predicted model outputs as inputs are called "simulator" models. The NOE model is an example of this category.

To train a model in "predictor" fashion, the predict command is used. Conversely, for a model trained in "simulator" fashion, the sim command is employed.

It is worth noting that as the prediction horizon increases in the predict command, it behaves equivalently to the sim command with the test data size. Therefore, careful consideration should be given to choosing an appropriate prediction horizon in the predict command, avoiding overly large values for efficient operation.

For the NOE model, the sim command is used, while for the NARX model, the predict command is utilized. The block diagram of the NOE network is depicted as follows, and its equations are given as:

$$y(t) = f(\hat{y}(t-1), \dots, \hat{y}(t-n_y), u(t-1), \dots, u(t-n_u))$$

In the NOE model, the output at time t is a function of past dynamics of inputs and past dynamics of estimated outputs $y(t)$ is a function of past dynamics of inputs and past dynamics of estimated outputs). This means that, unlike the NARX model, which used past dynamics of the output as inputs, here we use past dynamics of the estimated output as inputs. In other words, our model is not like the NARX model; it resembles a Predictor model and finds itself in a Simulator model:

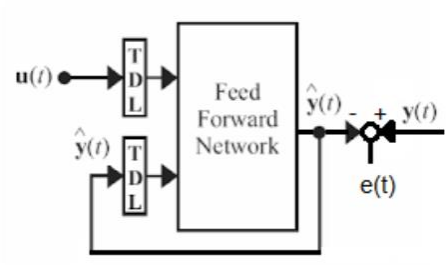


Figure 89 simulator-like model

For implementing the NOE model, we used the **narxnet** command, which, by default, provides a NARX network. We need to convert it to a NOE network using the **closeloop** command.

To select regressors, we use the results from the previous section. In the first section, we obtained delays using both statistical and Lipschitz methods.

The simulation results for systems 1 and 2 in both NARX and NOE modes are as follows:

We consider the nonlinear network as the default.

```
sys.Nonlinearity = 'wavenet' ;
```

Through trial and error, it is observed that this selection provides a good solution.

Applying NARX on the entire dataset in a simulator mode:

Through trial and error, it is observed that increasing the values of **na** and **nb** improves simulation accuracy and fitness percentages. However, changing the value of **nk** does not have a significant impact on the output. Therefore, we slightly increase **na** and **nb** values while keeping them relatively low to maintain simplicity.

```
na=3*ones(4,4);

nb=3*[1 2 1 1; 1 2 1 1; 1 2 1 1; 1 2 1 1];

nk=ones(4,4);
```

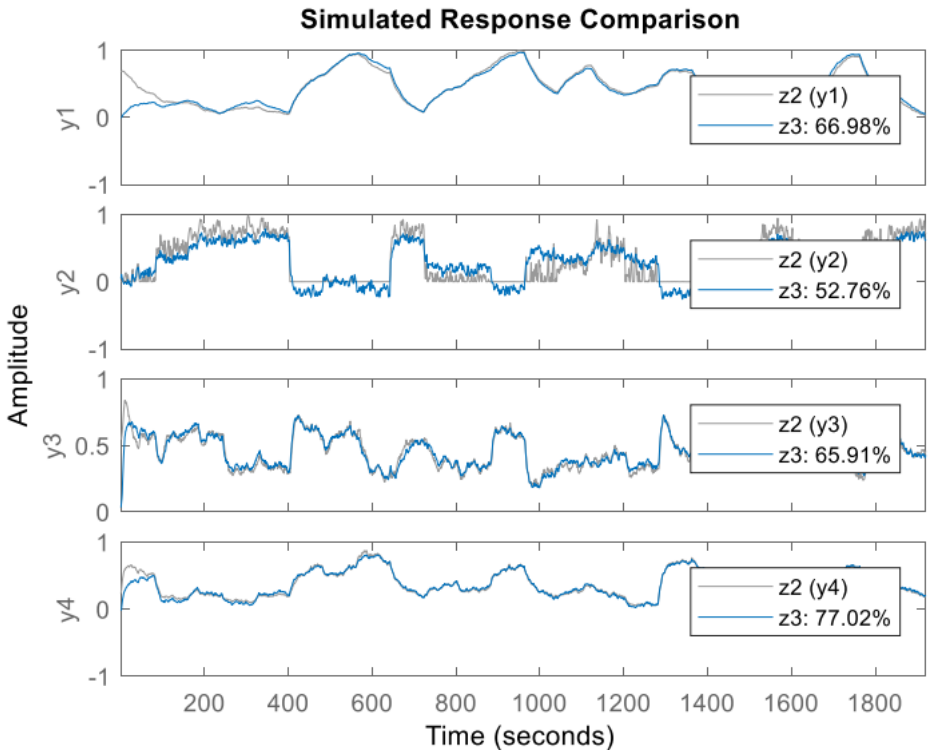


Figure 90 The NARX output on the entire data (simulator mode) (system 2)

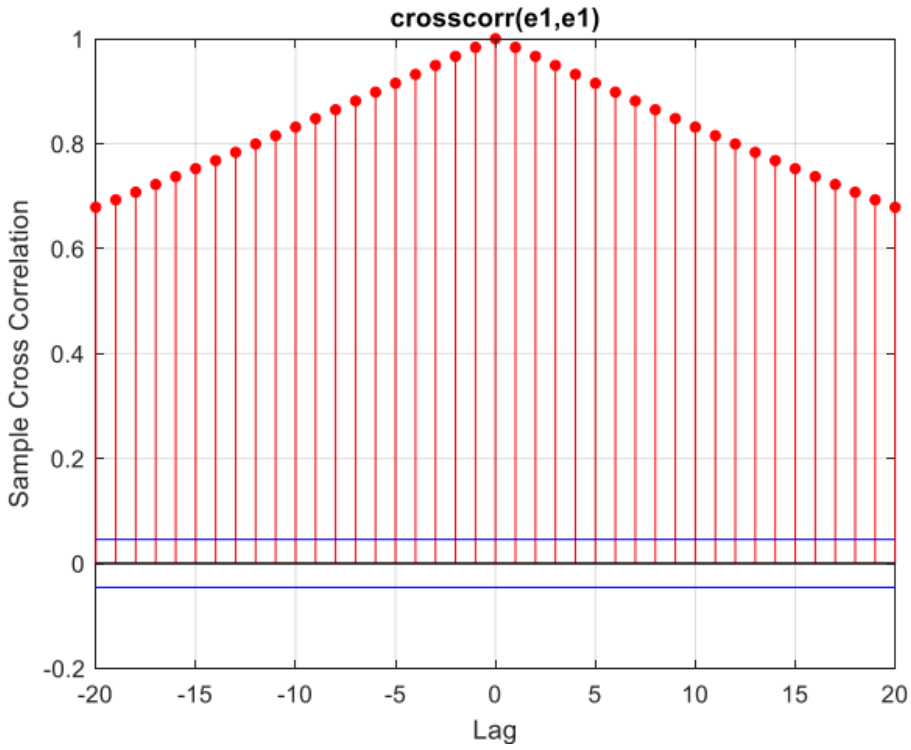


Figure 91 Crosscorrelation on error from first output

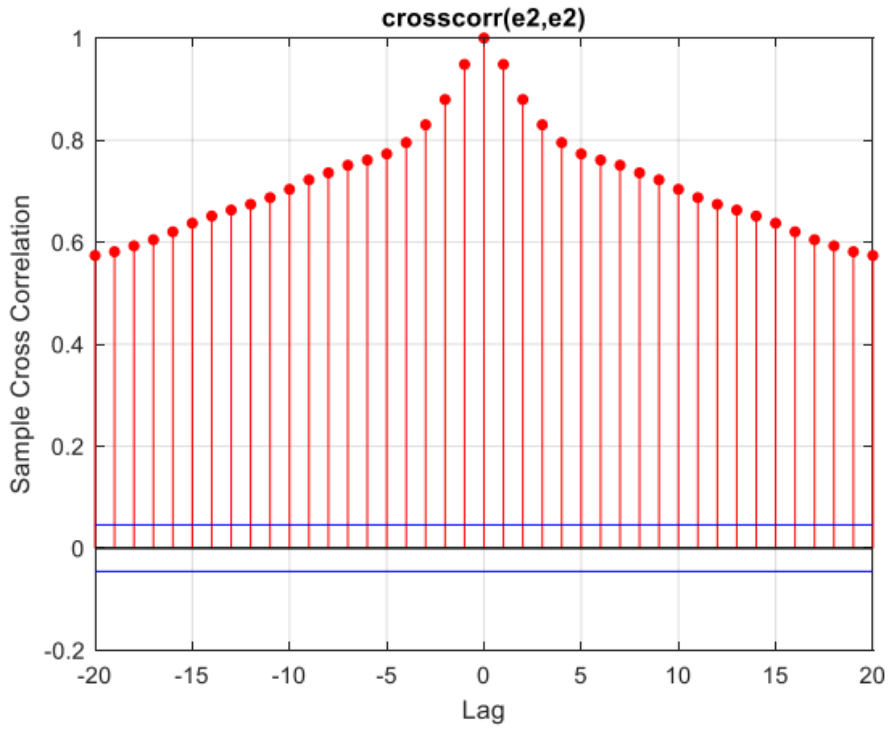


Figure 92 Crosscorrelation on error from second output

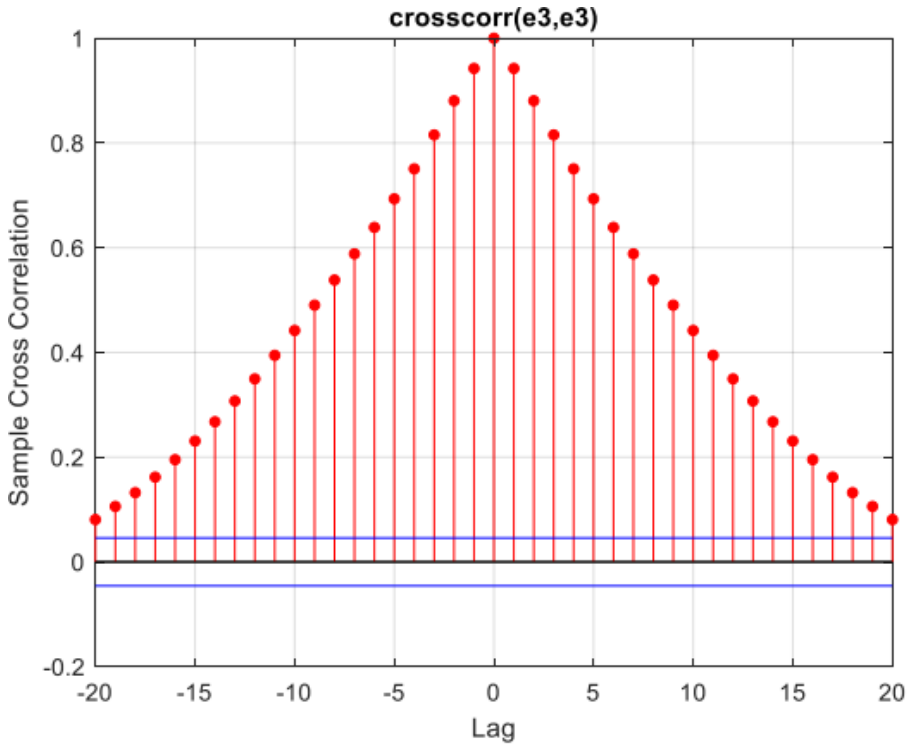


Figure 93 Crosscorrelation on error from third output

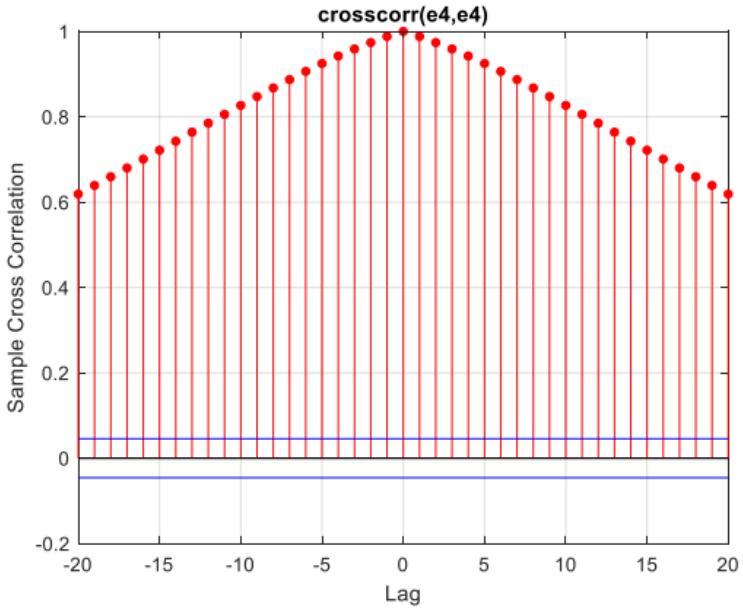


Figure 94 Crosscorrelation on error from third output

Dynamic data on simulator mode:

We use the dynamics obtained from the F.S method since they are relatively few and the error for each output has been whitened to a good extent. We use the same values for na and nb as before, as explained earlier.

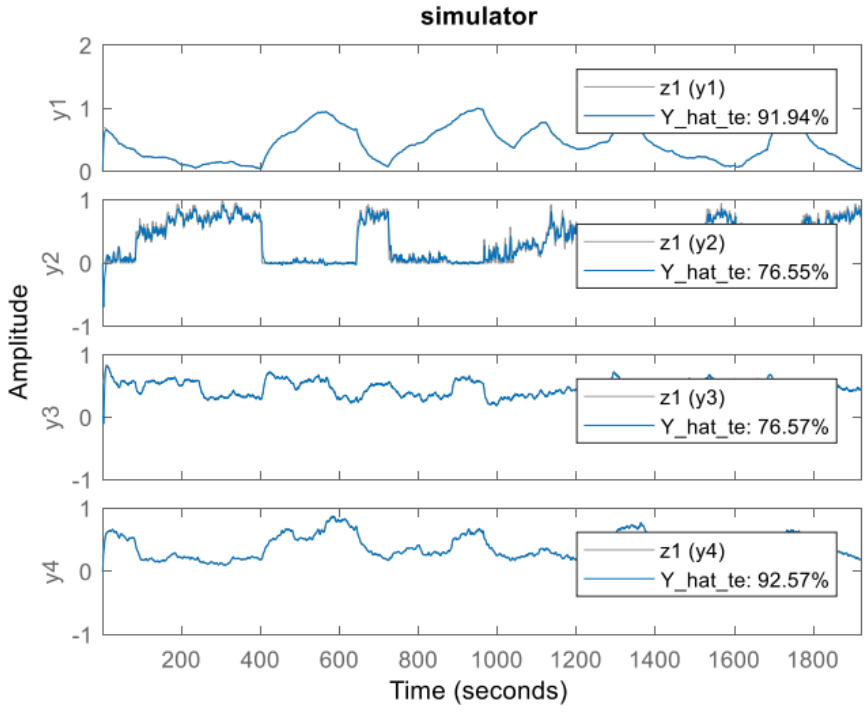


Figure 95 The output of NARX on dynamic data (simulator mode) (system 2)

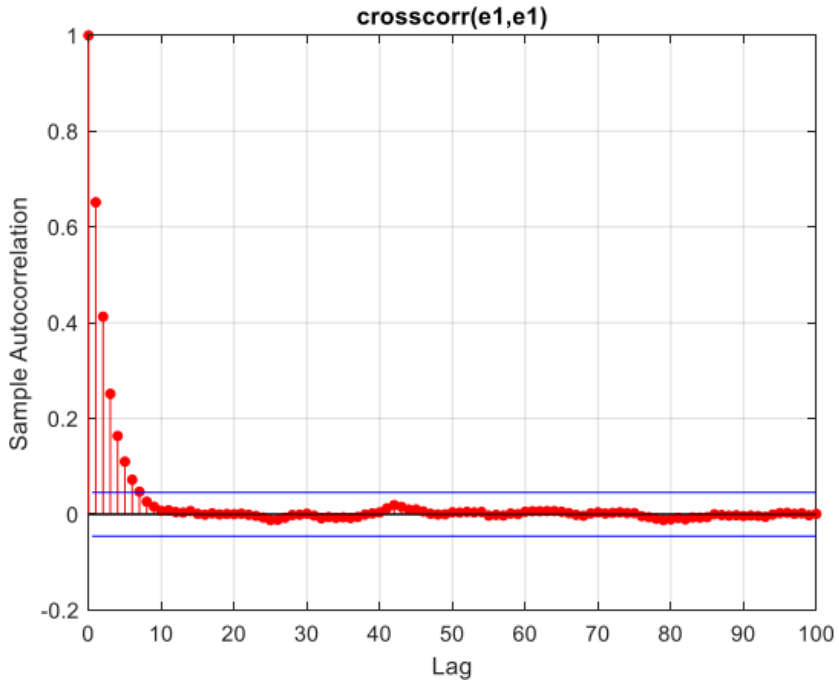


Figure 96 Cross-correlation on the error of the first output (system 2)

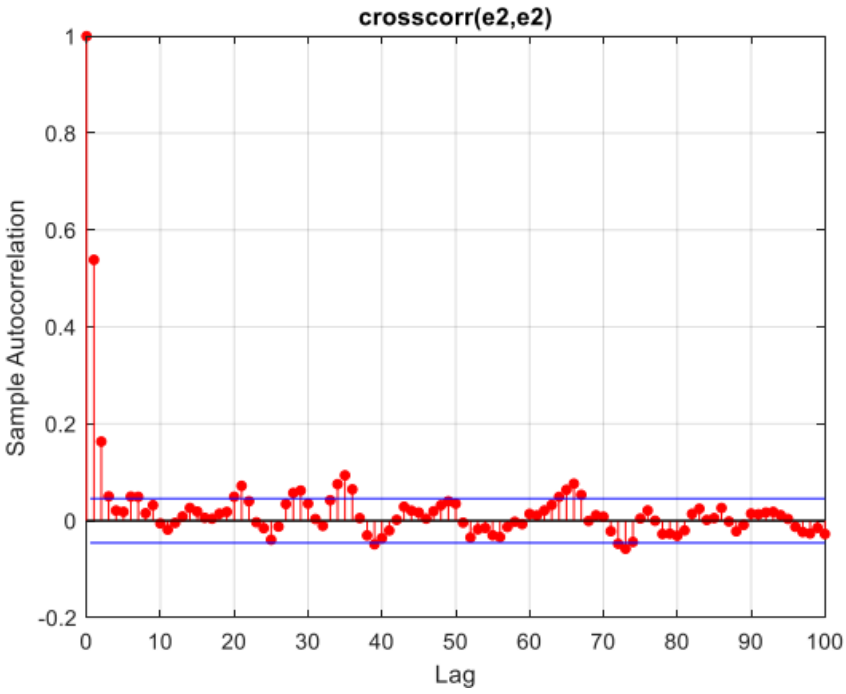


Figure 97 Cross-correlation on the error of the second output (system 2)

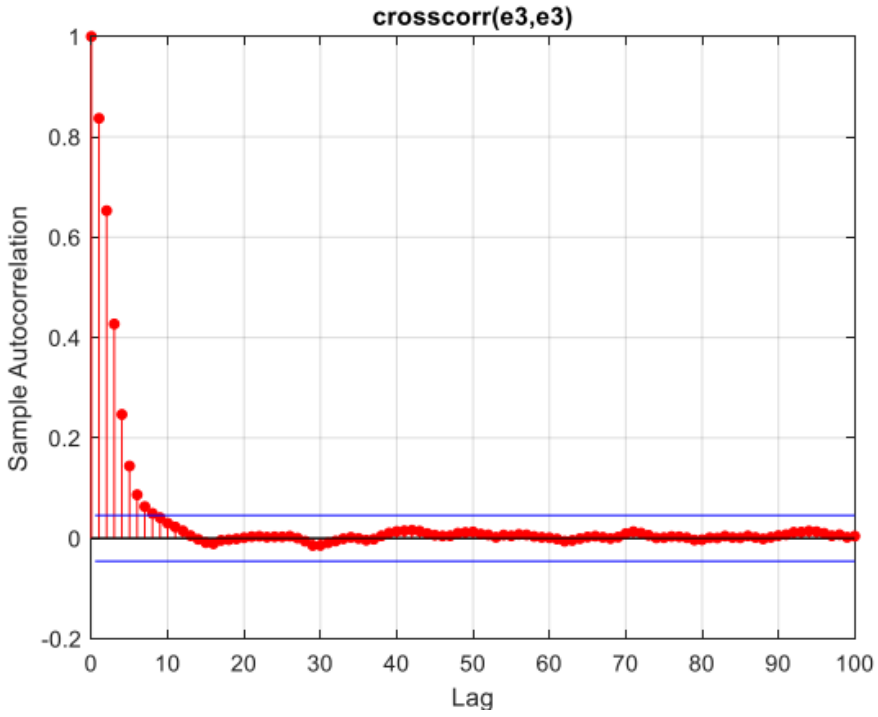


Figure 98 Cross-correlation on the error of the third output (system 2)

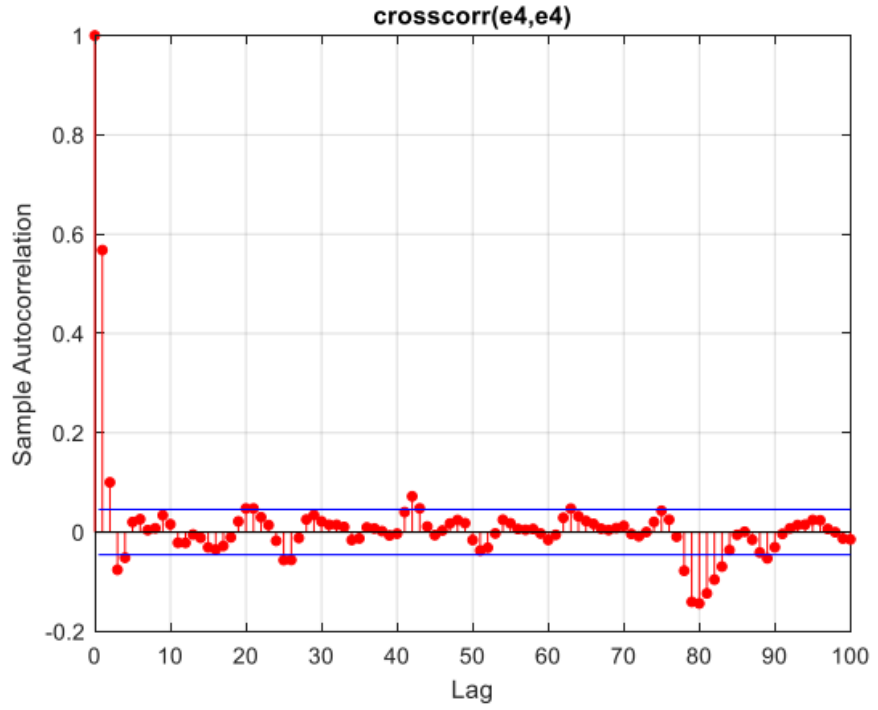


Figure 99 Cross-correlation on the error of the fourth output (system 2)

It is observed that in the simulator mode, whitening of errors is better achieved by dynamic data, resulting in improved fitness percentages.

On the entire dataset in a prediction mode:

Through trial and error, it is observed that increasing the values of **na** and **nb** improves simulation accuracy and fitness percentages. However, changing the value of **nk** does not have a significant impact on the output. Therefore, we slightly increase **na** and **nb** values while keeping them relatively low to maintain simplicity.

```
na=3*ones(4,4); %%ziad beshe=kami behtare
nb=4*ones(4,4); %%ziad beshe=kami behtare
nk=1*ones(4,4); %%taghiresh kheili asari nadasht
```

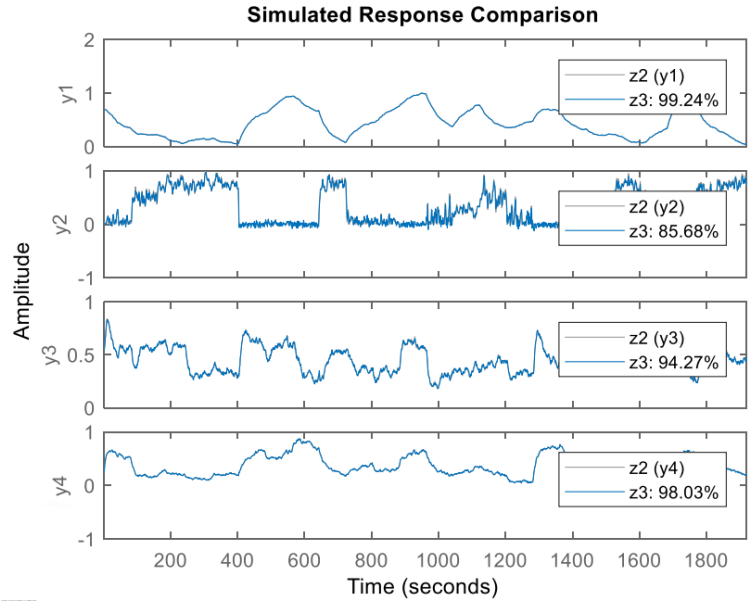


Figure 100 The NARX model on all data (predictor mode) (system 2)

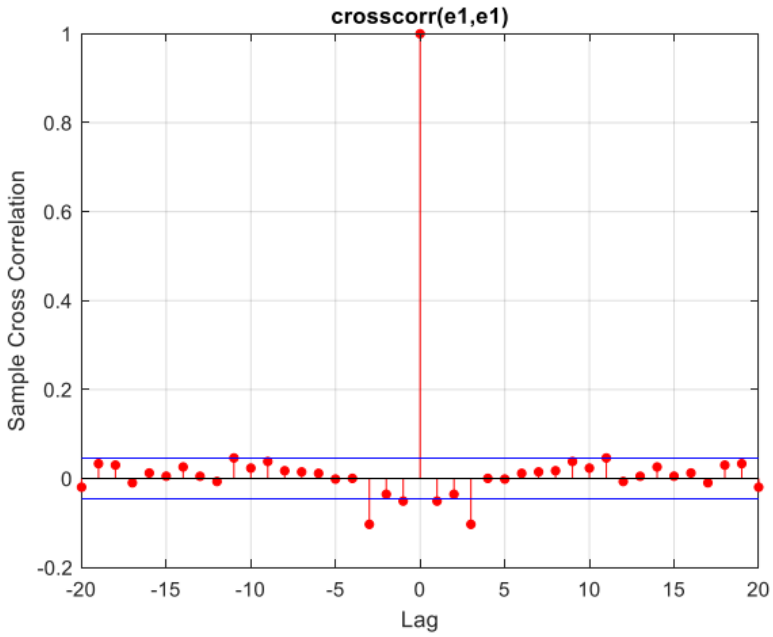


Figure 101 Cross-correlation on the error of the first output (system 2)

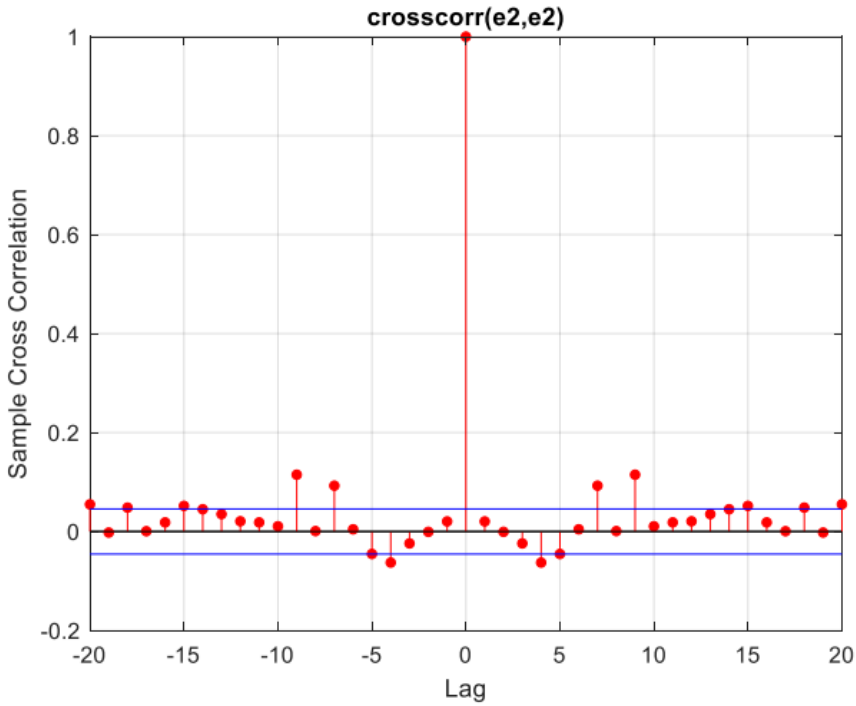


Figure 102 Cross-correlation on the error of the second output (system 2)

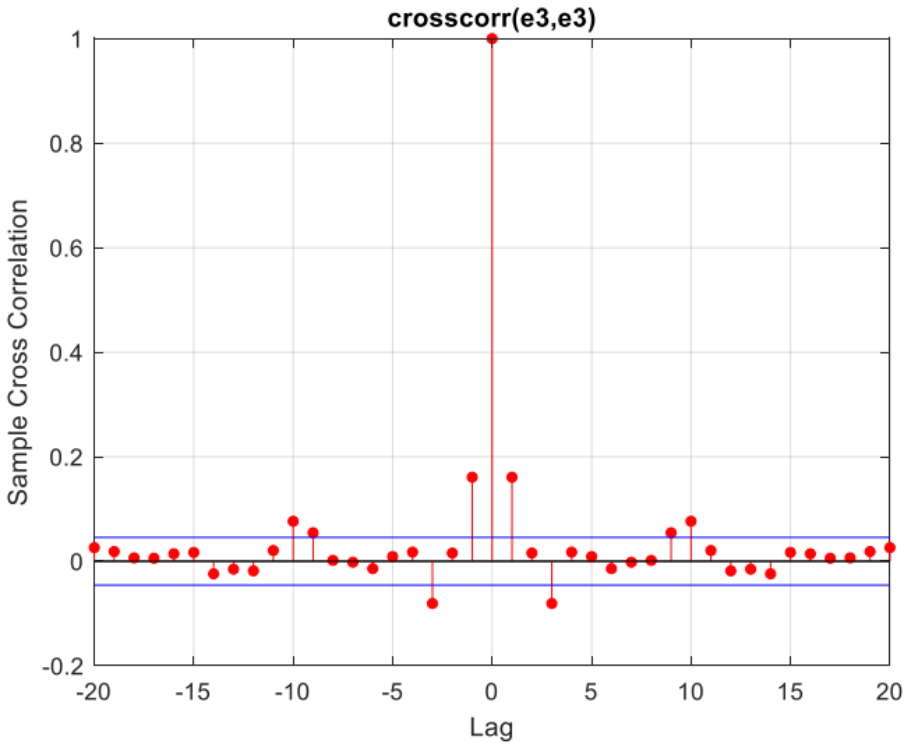


Figure 103 Cross-correlation on the error of the third output (system 2)

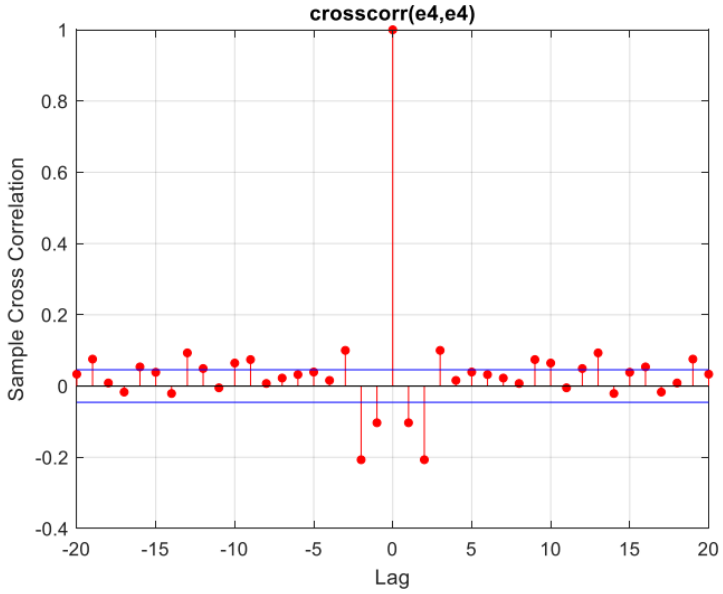


Figure 104 Cross-correlation on the error of the fourth output (system 2)

It is observed that the fitness percentages on the test data are very good, and error whitening has been achieved remarkably well.

On dynamic data in a prediction mode:

We use the dynamics obtained from the F.S method since they are relatively few, and the error for each output has been whitened to a good extent. We use the same values for **na** and **nb** as before, as explained earlier.

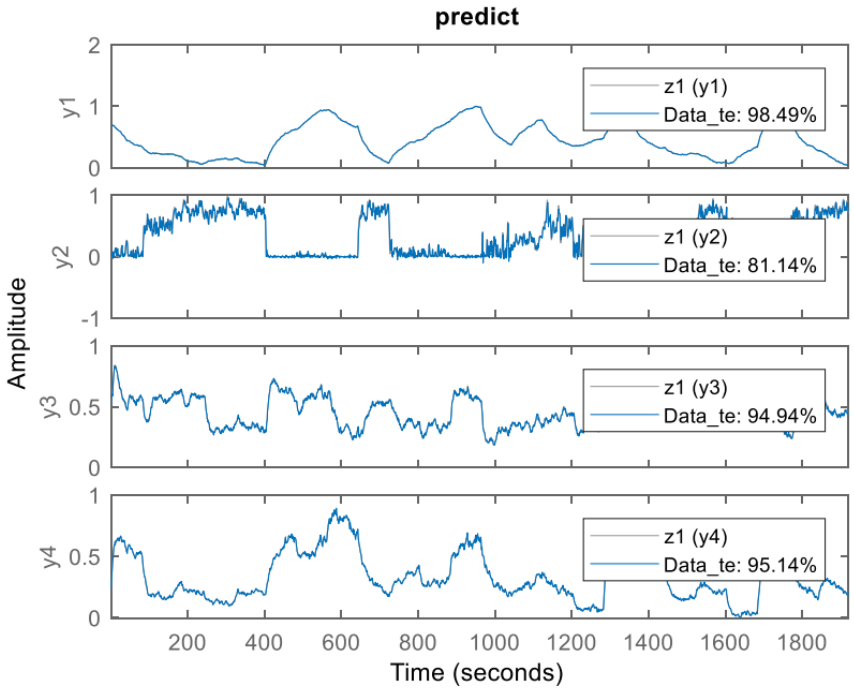


Figure 105 The output of NARX on dynamic data (predictor mode) (system 2)

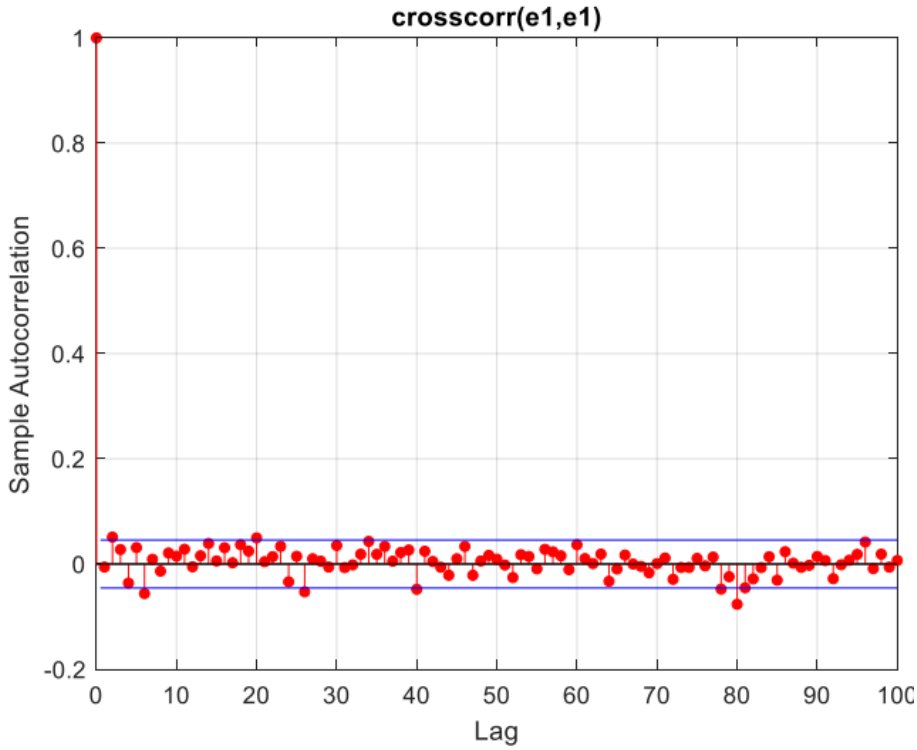


Figure 106 Cross-correlation of error of first output NARX on dynamic data (predictor mode) (system 2)

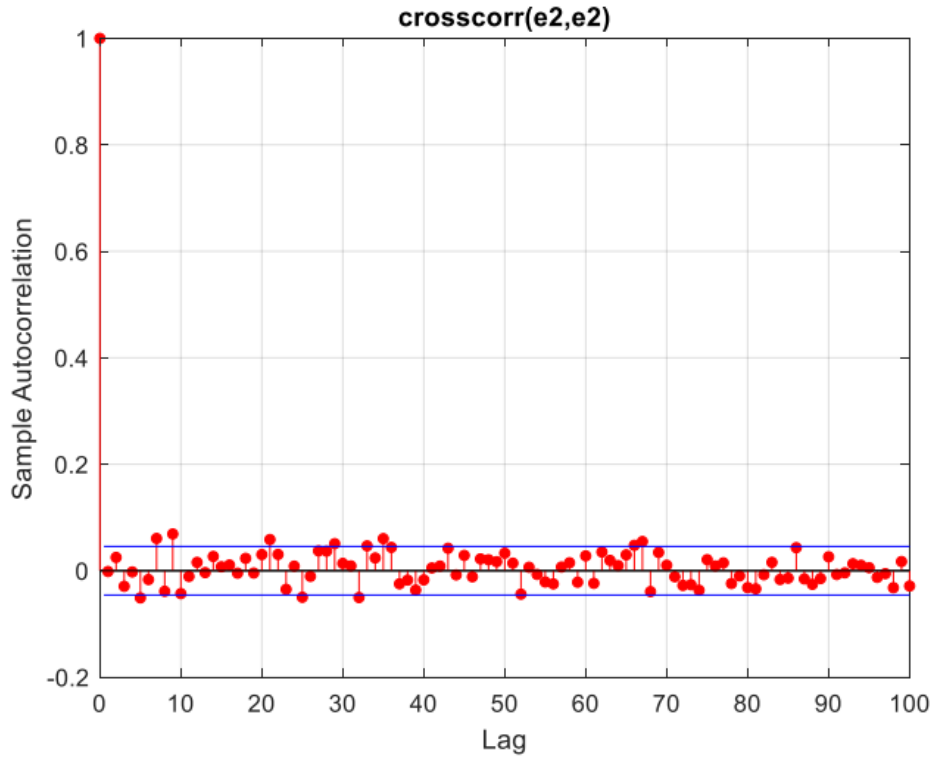


Figure 107 Cross-correlation of error of second output NARX on dynamic data (predictor mode) (system 2)

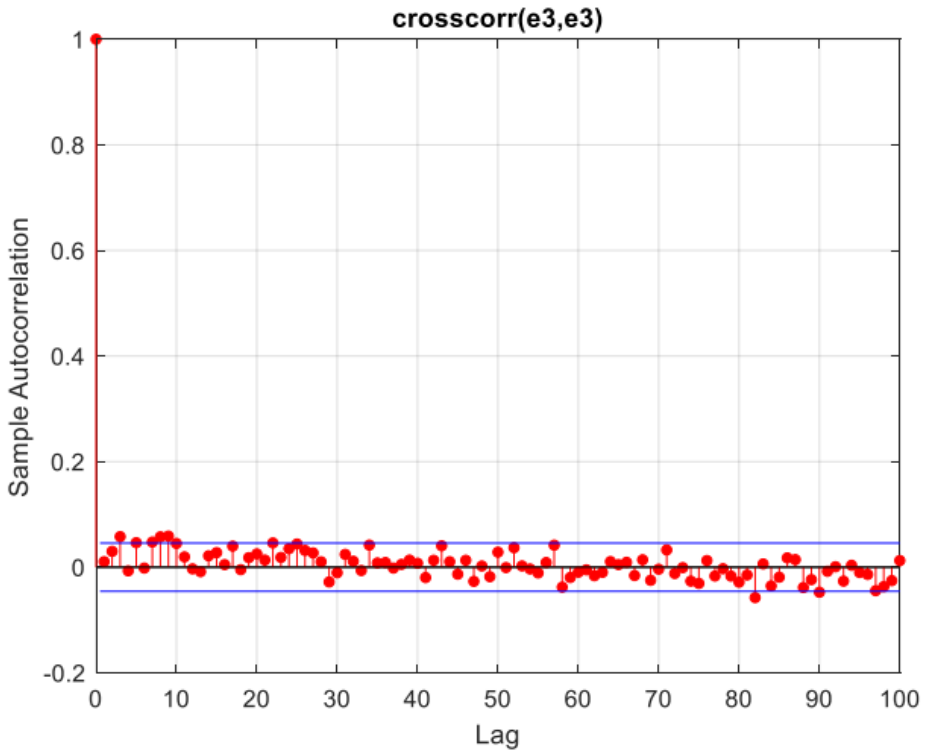


Figure 108 Cross-correlation of error of third output NARX on dynamic data (predictor mode) (system 2)

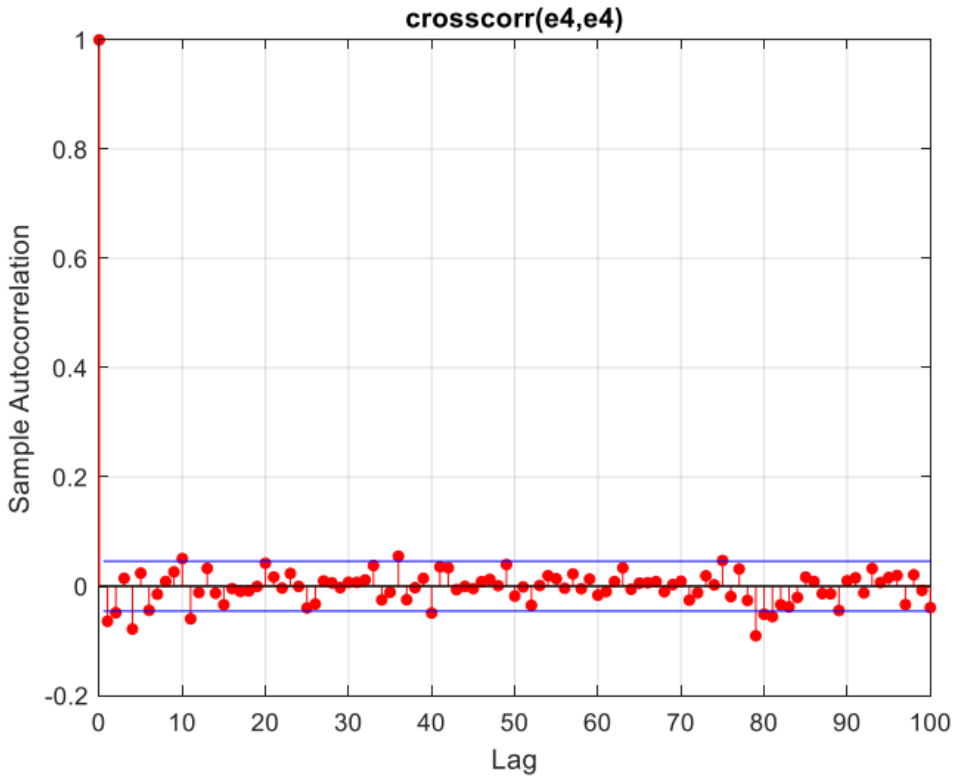


Figure 109 Cross-correlation of error of fourth output NARX on dynamic data (predictor mode) (system 2)

It is observed that the fitness percentages on the test data are very good, and error whitening has been achieved remarkably well. It is evident that well-chosen dynamics lead to high accuracy and effective error whitening.

In the prediction mode, both on the entire dataset and dynamic data, the results are significantly better than the simulator mode. In the Predictor mode, considering the autocorrelation plots of errors, the error is closer to white noise.

We have set the prediction horizon to 1. If we increase it to 5, the results may improve.

```
Y_hat_te = predict(NARX_model_p,Data_te,5); %% afzayesh
ofogh pishbini=badtar
```

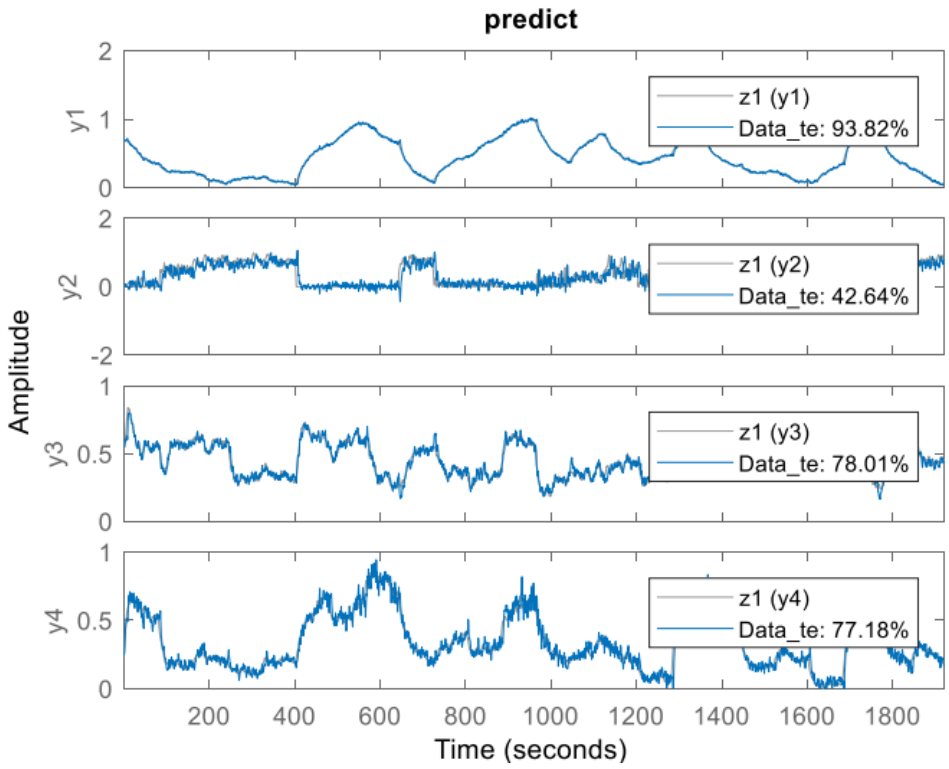


Figure 110 The output of NARX on dynamic data (predictor mode, increasing prediction horizon) (system 2)

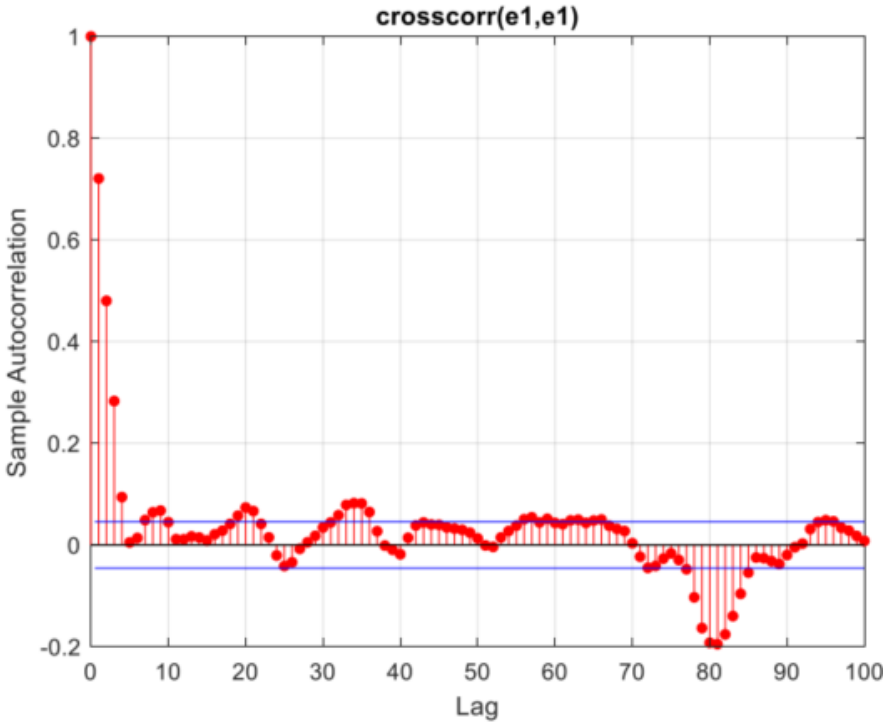


Figure 111 Cross-correlation of error of first output NARX on dynamic data (predictor mode, increasing prediction horizon) (system 2)

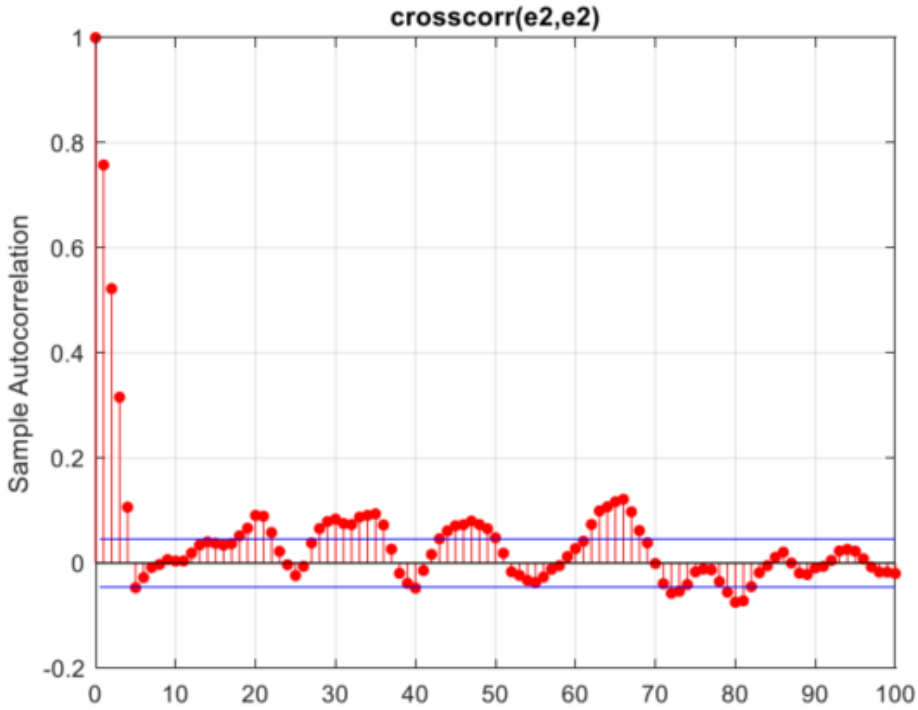


Figure 112 Cross-correlation of error of second output NARX on dynamic data (predictor mode, increasing prediction horizon) (system 2)

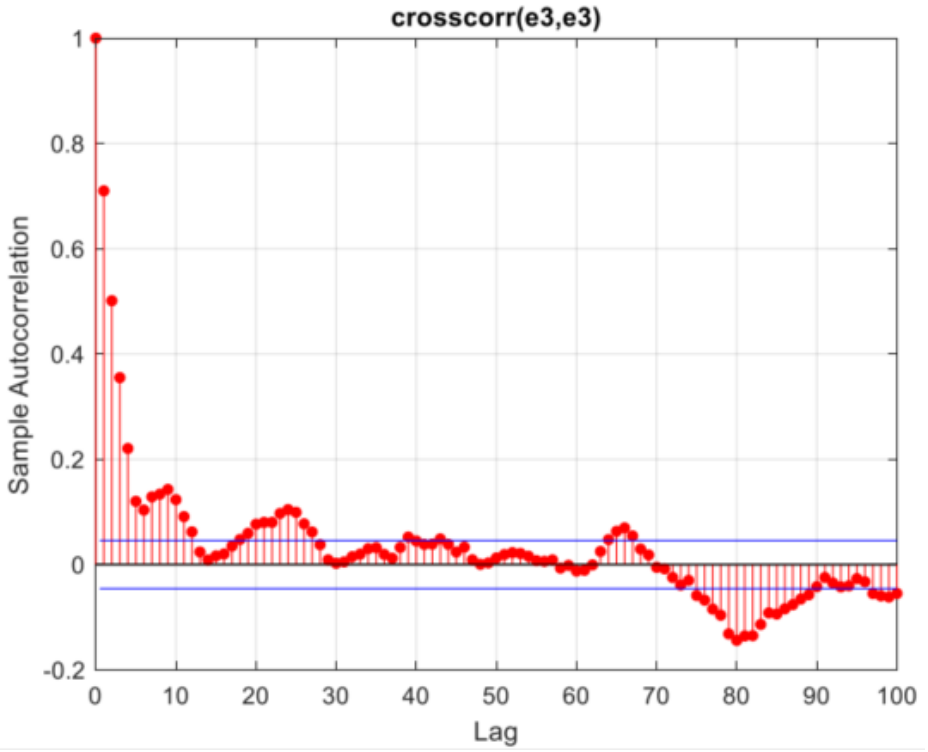


Figure 113 Cross-correlation of error of third output NARX on dynamic data (predictor mode, increasing prediction horizon) (system 2)

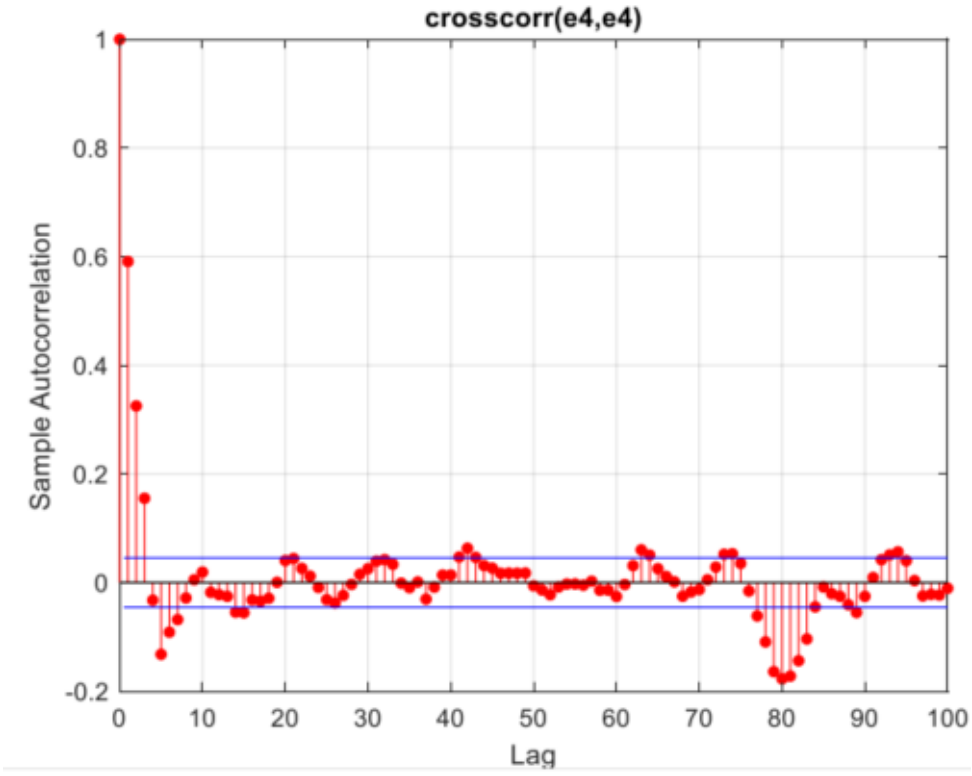


Figure 114 Cross-correlation of error of fourth output NARX on dynamic data (predictor mode, increasing prediction horizon) (system 2)

It is observed that the errors have deviated slightly from whitening, and fitness percentages have also decreased. Therefore, a small prediction horizon is recommended.

Applying NOE on the entire dataset:

The parameters for NOE are adjusted as follows (through trial and error).

```
a=1;
 d1 = [1:a];
 d2 = [1:1];                %%ziad=deghat kam mishe
neuron = 5; %%nabayad ziad kard kheili
narx_net = narxnet(d1,d2,neuron);
```

With an increase in the parameter **d1**, the accuracy and fitness percentage decrease. Similarly, increasing the number of neurons leads to the same effect, and the above values are chosen through trial and error. Both **tansig** and **logsig** activation functions performed well, but **tansig** was found to be better and was selected.

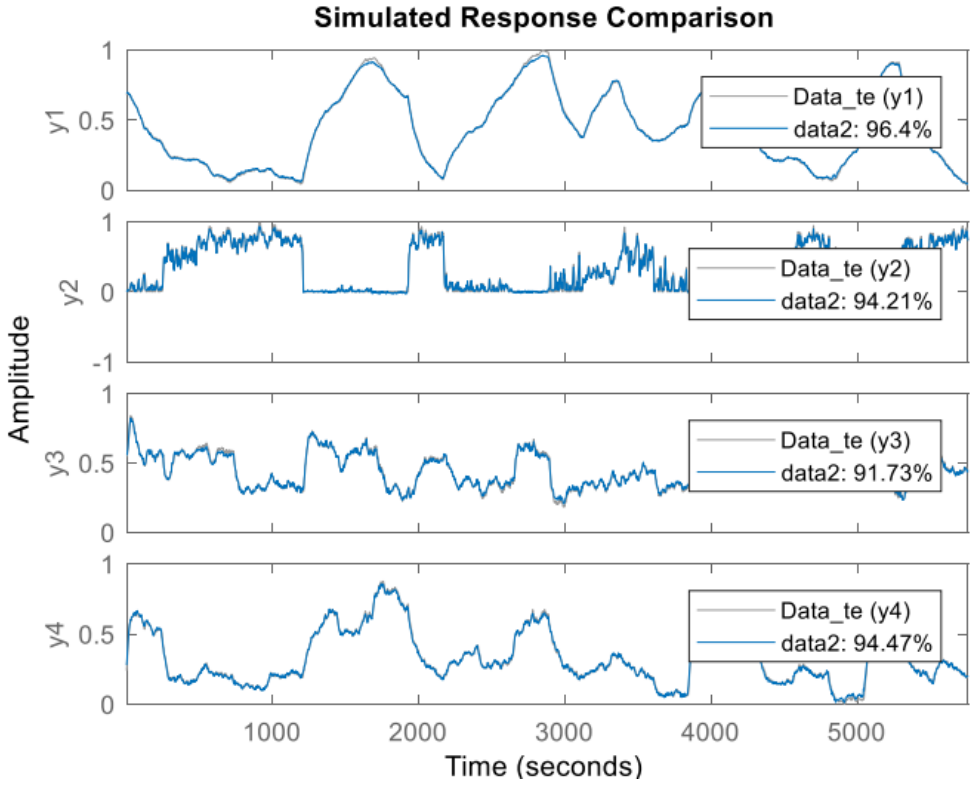


Figure 115 The output of the actual system and the estimated system (entire data NOE) (system 2)

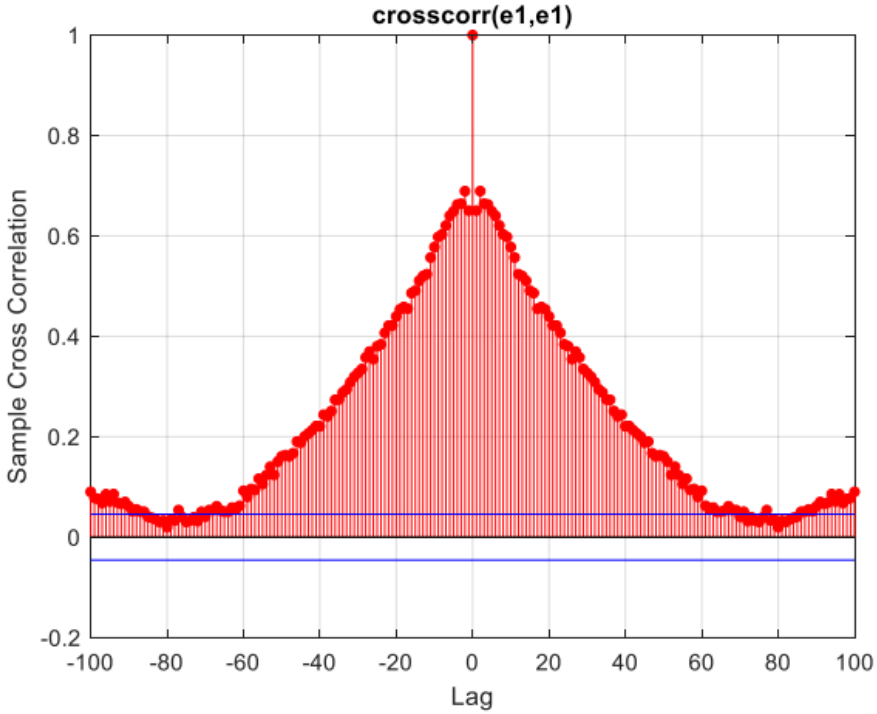


Figure 116 Cross-correlation on error of the first output NOE (steamgen)

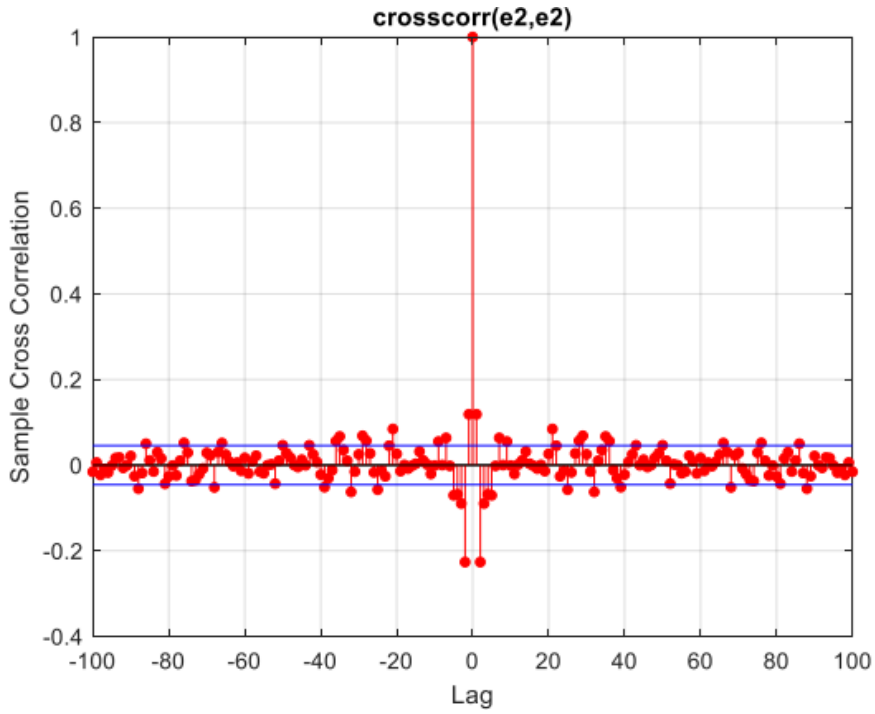


Figure 117 Cross-correlation on error of the second output NOE (steamgen)

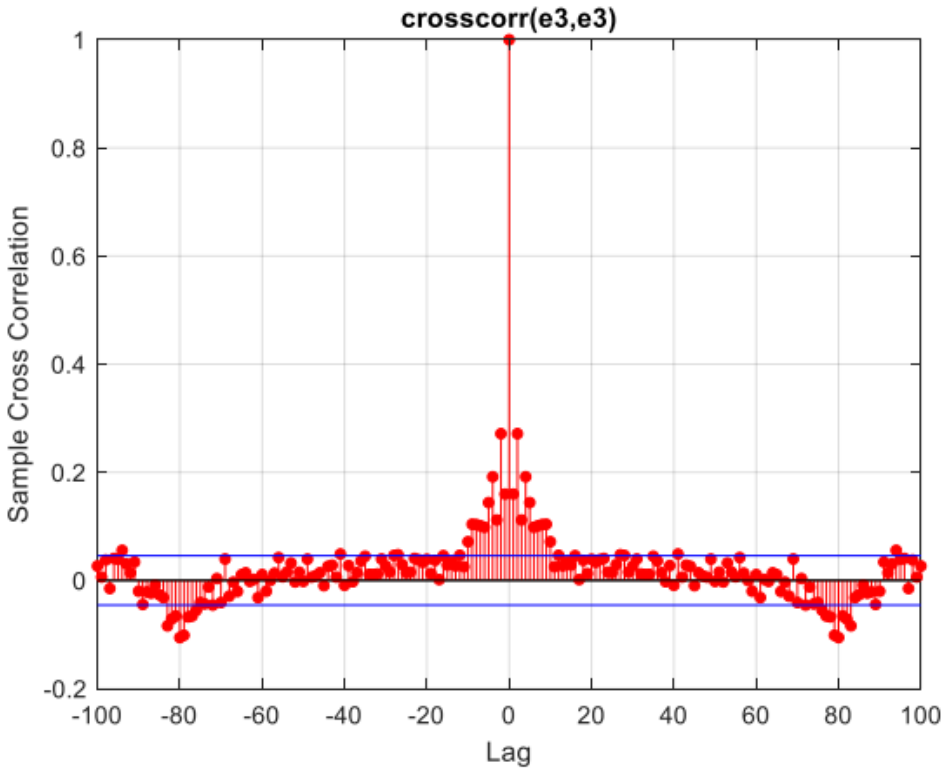


Figure 118 Cross-correlation on error of the third output NOE (steamgen)

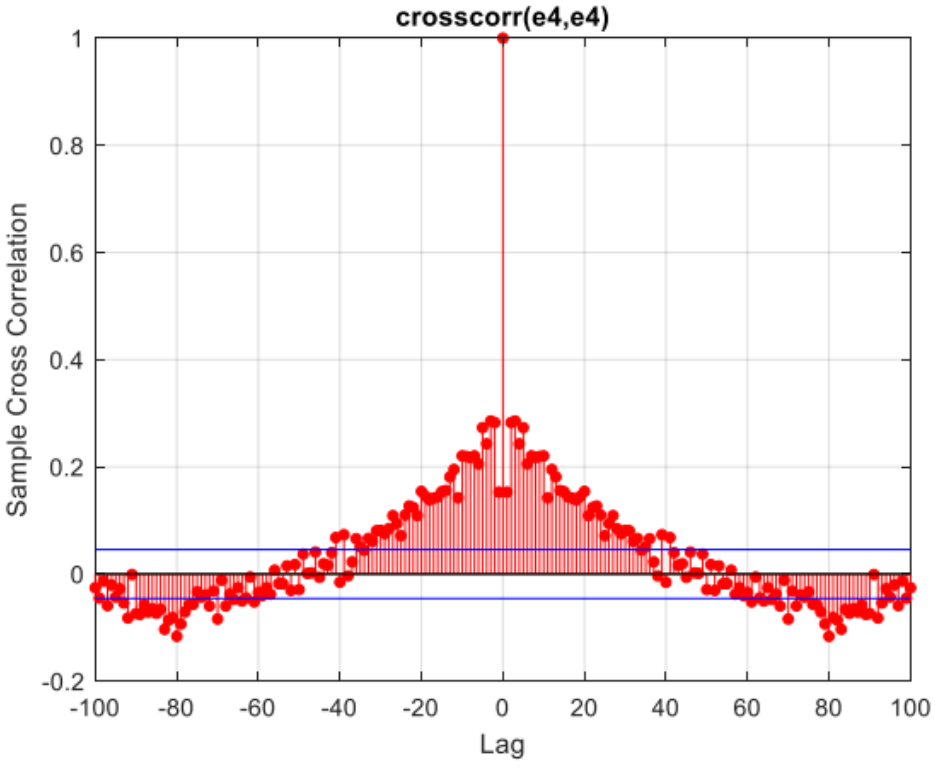


Figure 119 Cross-correlation on error of the fourth output NOE (steamgen)

The fitness percentage is very suitable, but the error has not been minimized.

Testing dynamic data: choosing the same previous parameters (as we have observed improvement in results through trial and error).

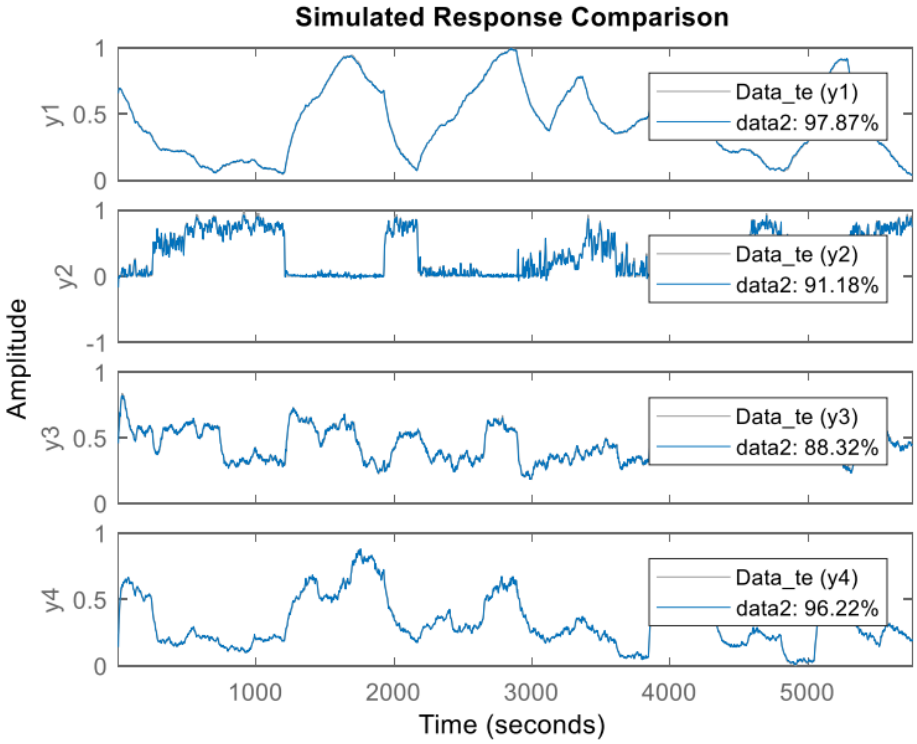


Figure 120 The real system output and the estimated system output for NOE (steamgen)

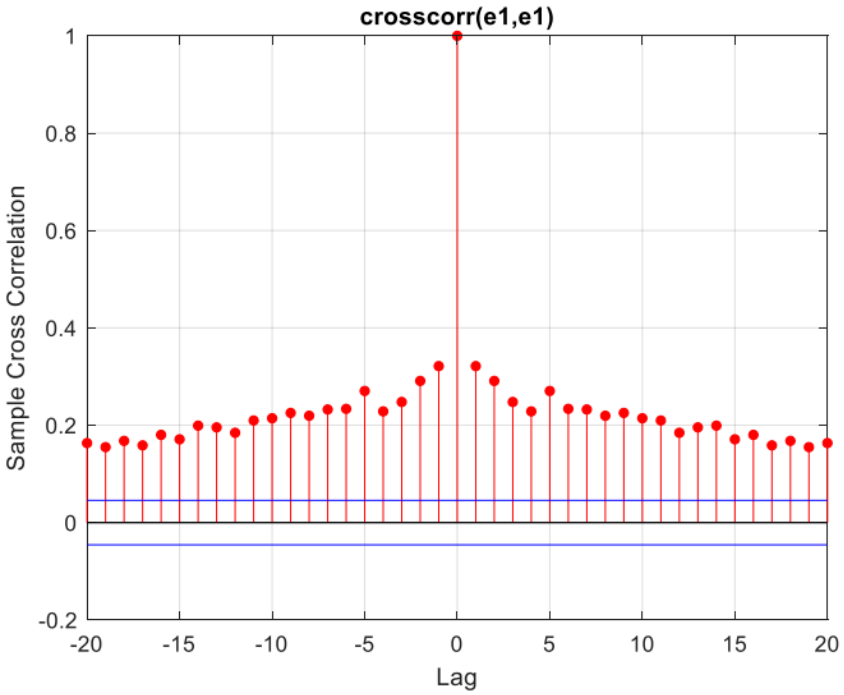


Figure 121 The cross-correlation for first output for NOE (dynamic data) (steamgen)

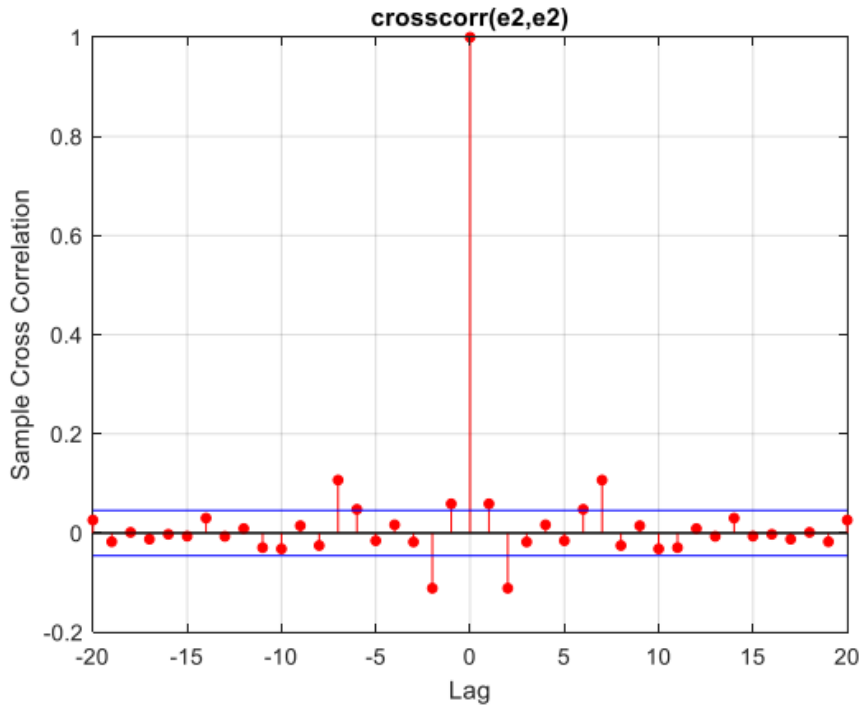


Figure 122 The cross-correlation for second output for NOE (dynamic data) (steamgen)

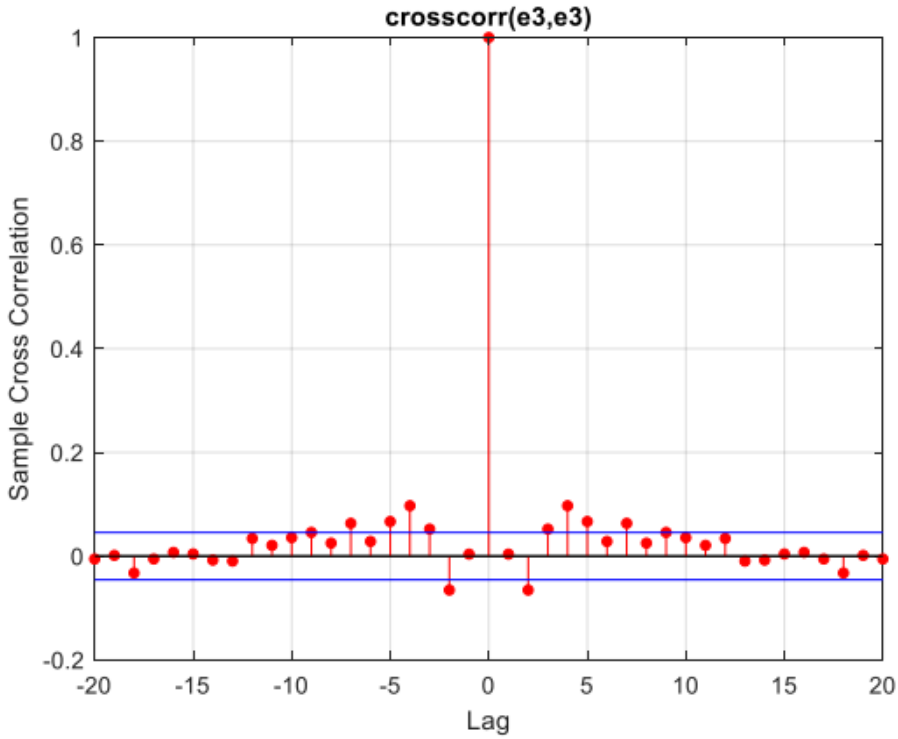


Figure 123 The cross-correlation for third output for NOE (dynamic data) (steamgen)

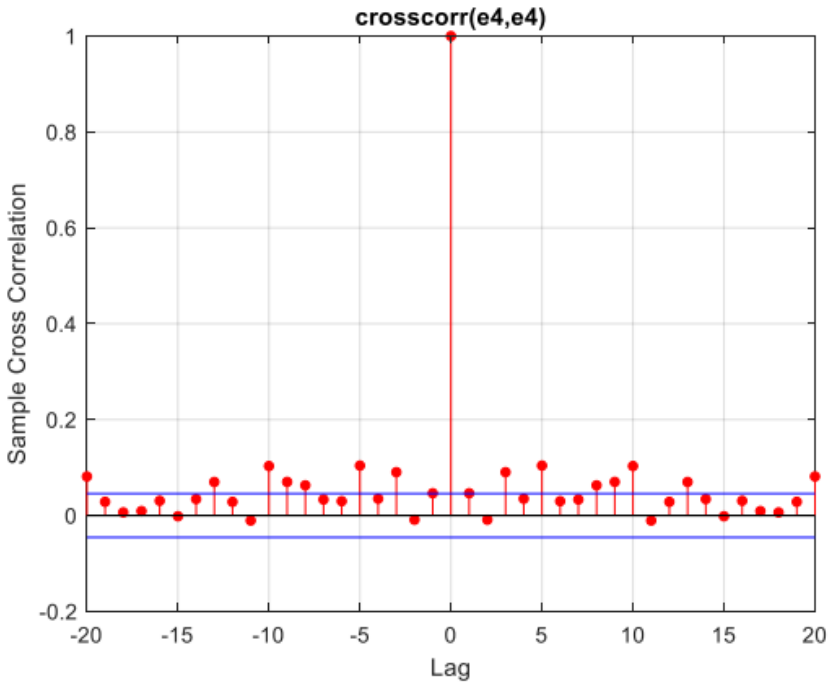


Figure 124 The cross-correlation for third output for NOE (dynamic data) (steamgen)

It can be observed that compared to the non-dynamic case, we have better fitness percentages and the errors have become whiter. However, in comparison between NOE and NARX, it can be said that the errors are whiter in NARX, and the fitness percentages are also close; therefore, NARX is preferred.

1.3) System 1

The selected hyperparameters for the second system (such as the delay value, activation functions, etc.) were also suitable and utilized for this system.

NARX on entire dataset in the 'Predictor' mode.

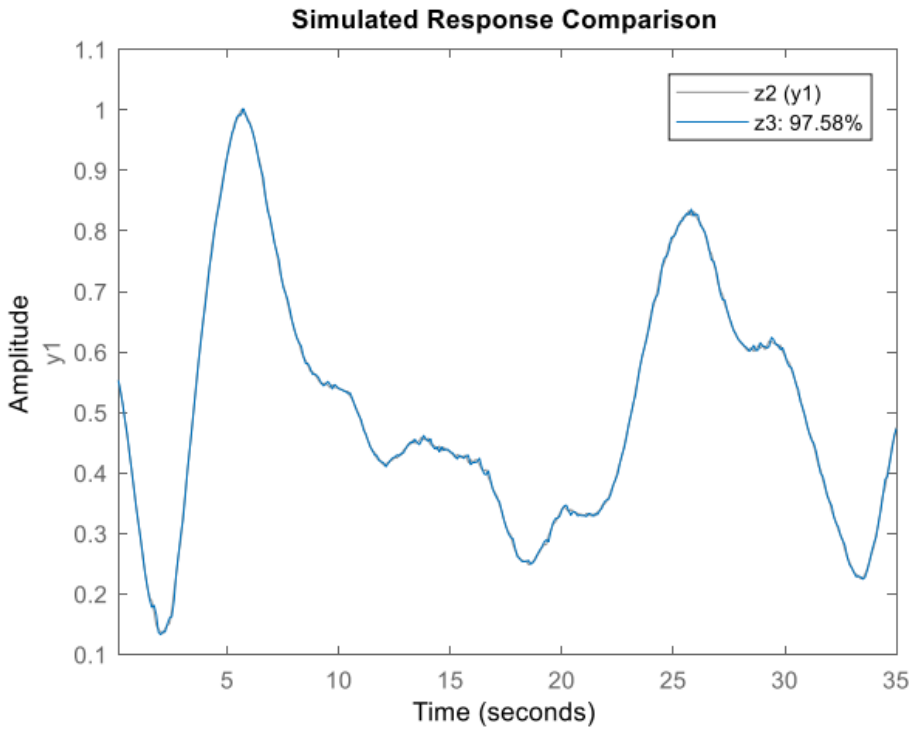


Figure 125 The output plot of NARX on the entire data (predictor) (system 1)

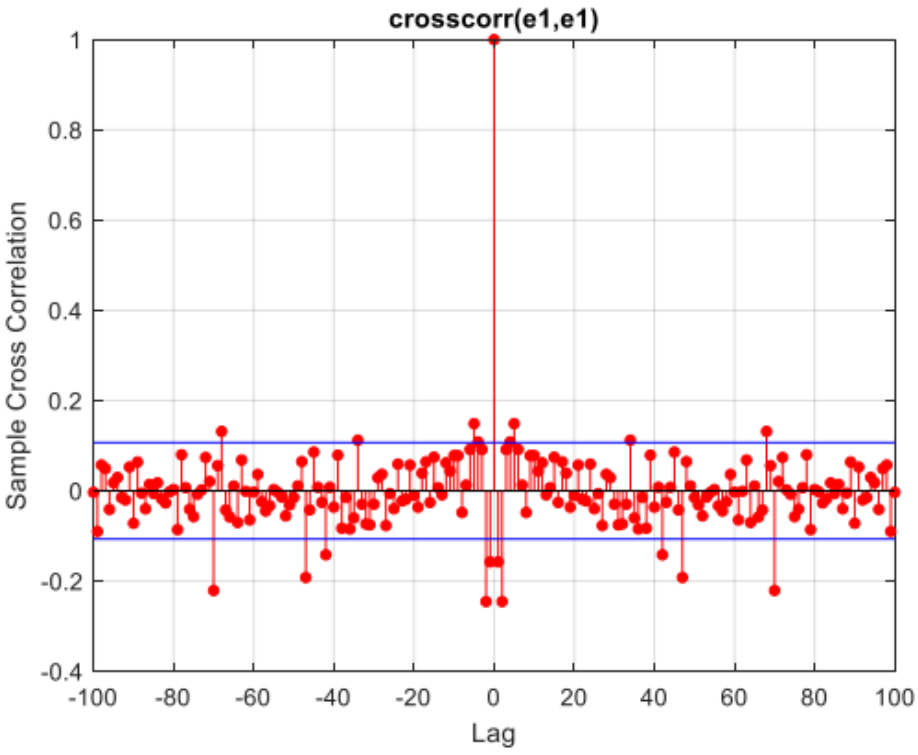


Figure 126 Auto-correlation of error

NARX on the dynamic data in predictor mode:

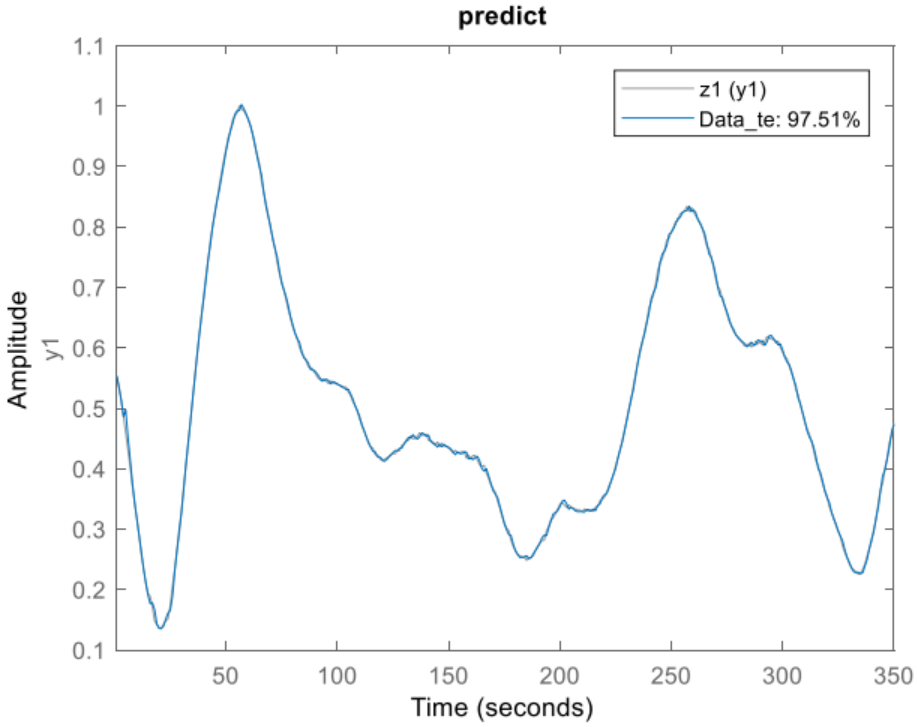


Figure 127 The NARX output on dynamic data in predictor mode (system 1)

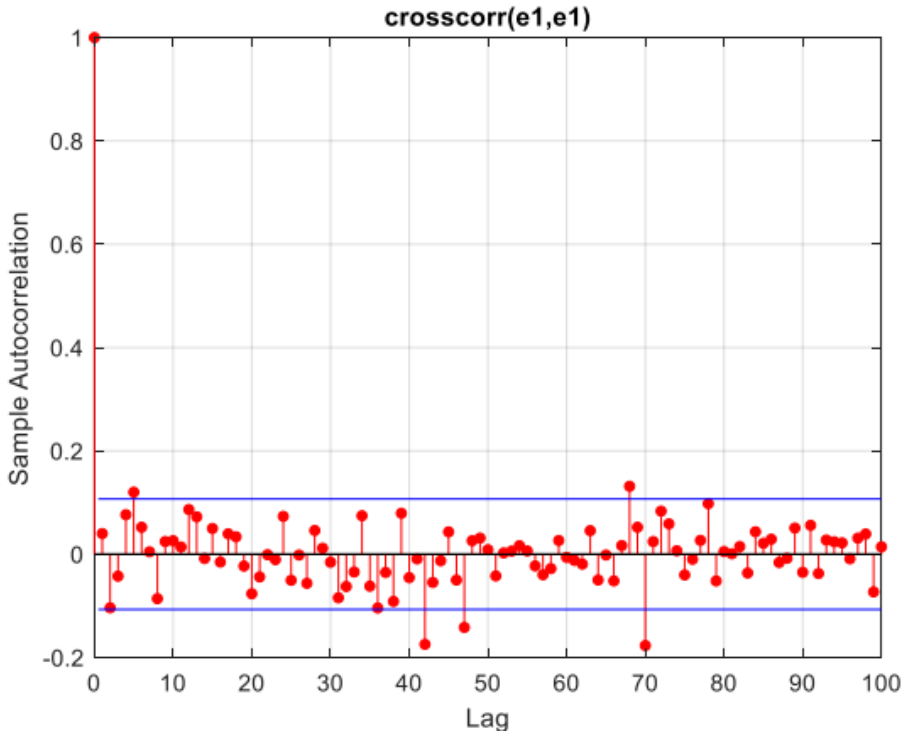


Figure 128 Error auto-correlation

In each case, the noise whitening has happened and the fit is around 97%.

Increasing prediction horizon:

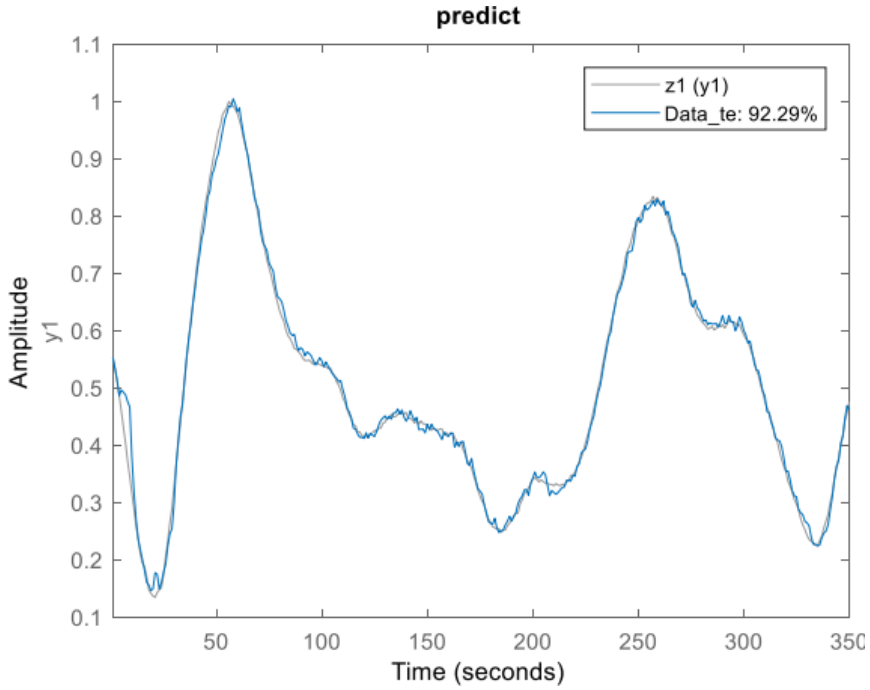


Figure 129 NARX output on the dynamic data in predictor mode (increased prediction horizon) (system1)

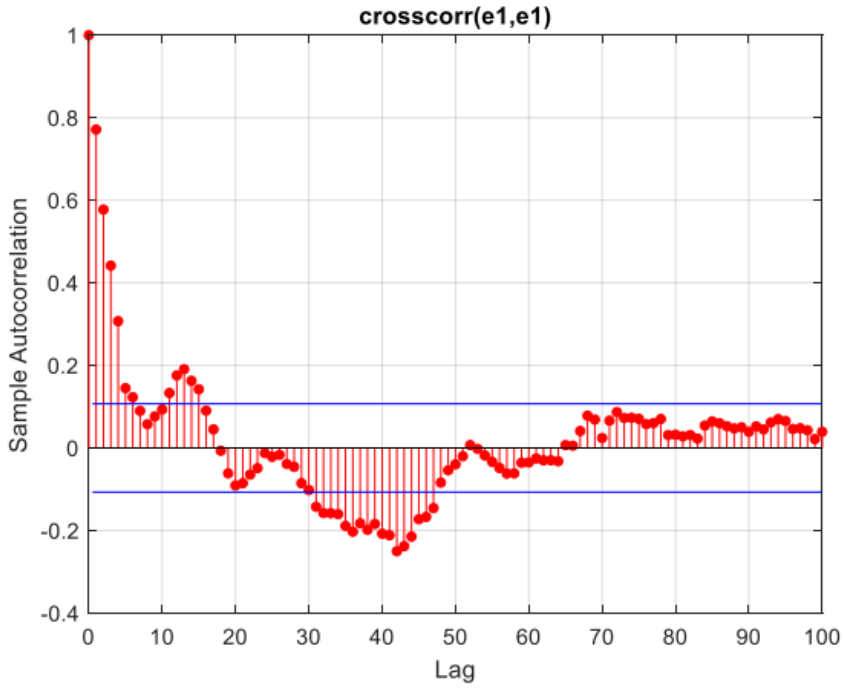


Figure 130 error auto-correlation (system 1)

It is observed that the fitness percentage slightly decreases, the correlation of errors increases, and it deviates slightly from whitening. In the simulator mode,

NARX in simulator mode:

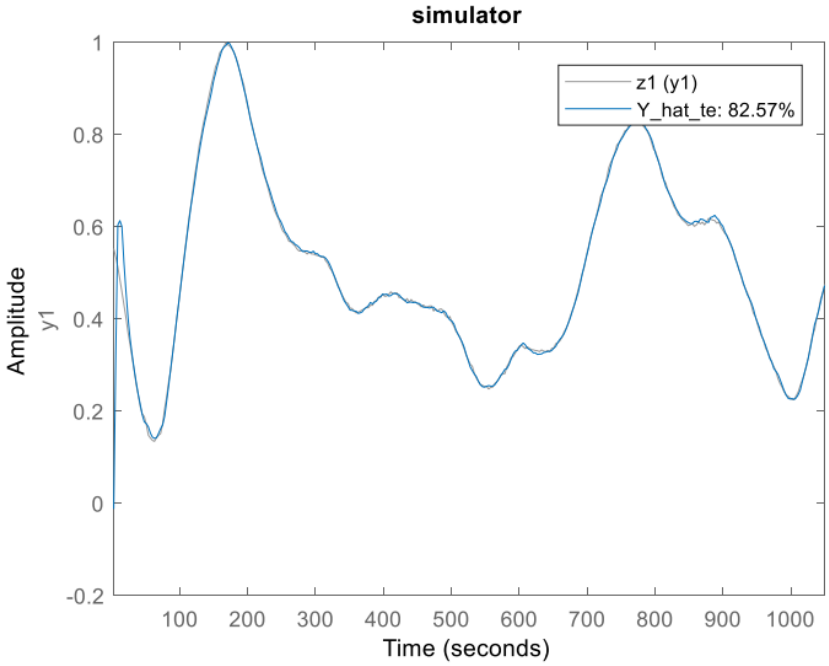


Figure 131 NARX output in simulator mode (system 1)

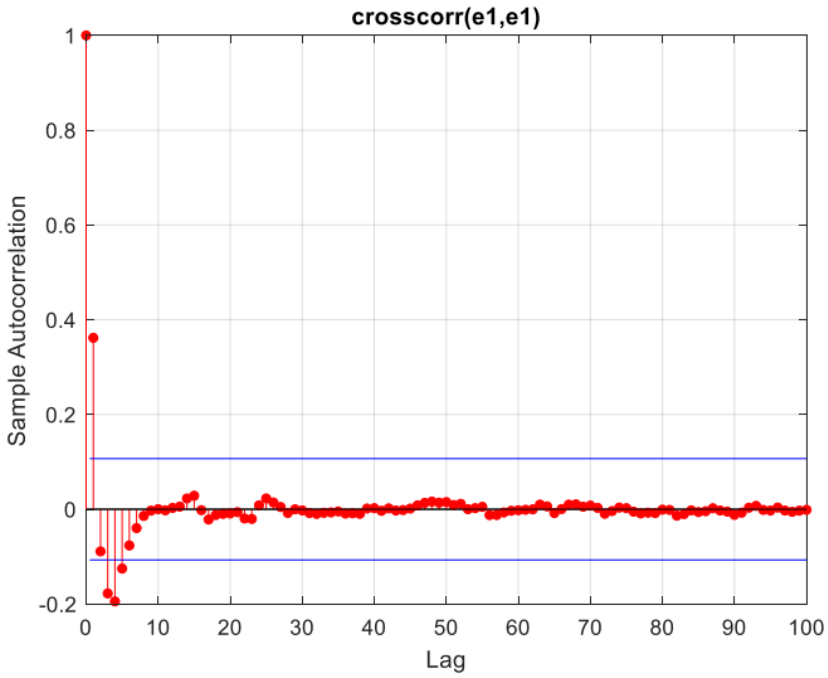


Figure 132 Error cross-correlation plot (system 1)

It is observed that error whitening has been performed, but the fitness percentage is lower than the predictor mode. (Similar to the previous system) NOE on the entire dataset: Through trial and error, the number of 5 neurons is suitable.

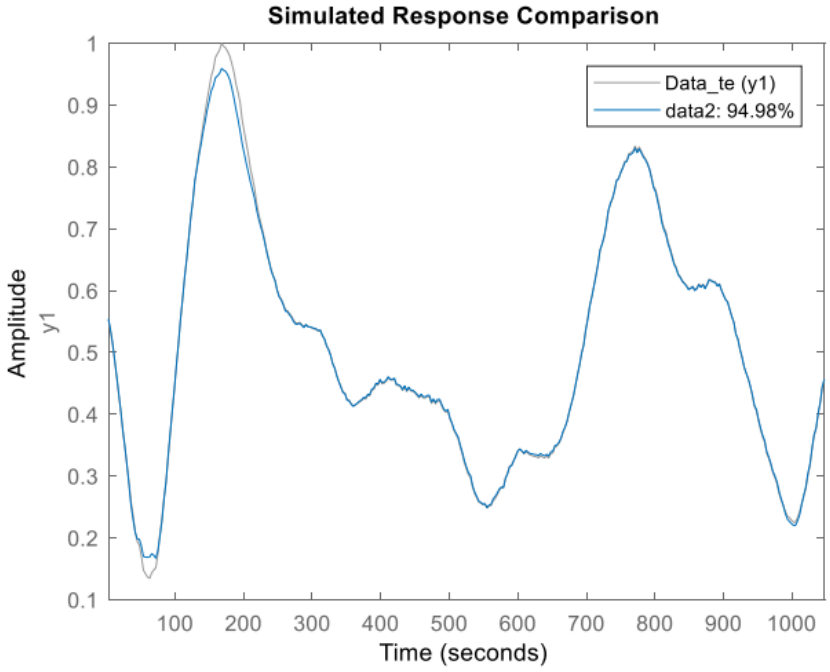


Figure 133 The actual and estimated outputs of NOE (system 1)

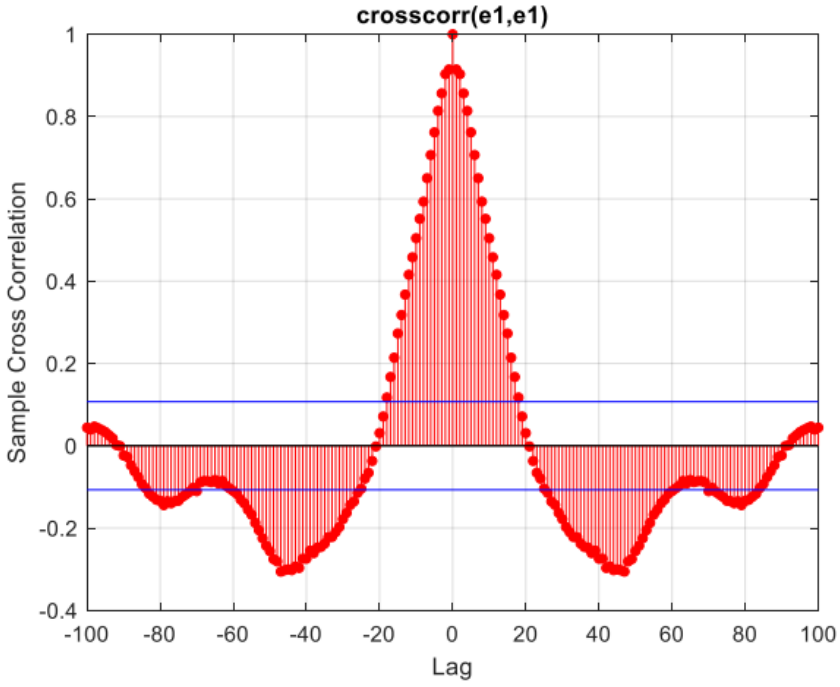


Figure 134 The cross-correlation of error (system 1)

It is observed that we have suitable fitness, but the error has not been whitened.

NOE on dynamic data:

Through trial and error, the number of 10 neurons is suitable.

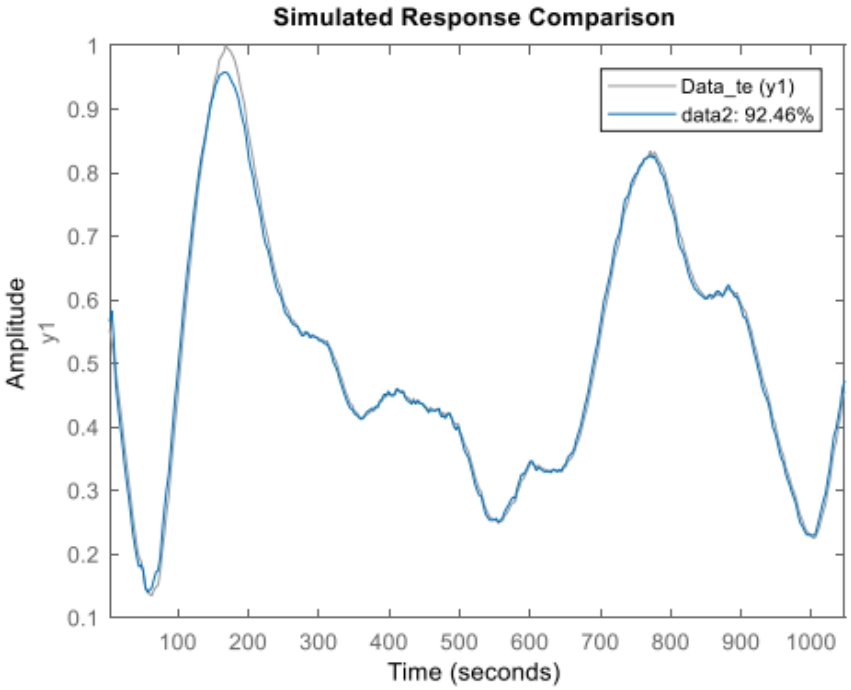


Figure 135 The actual and estimated system outputs of NOE dynamic data (system 1)

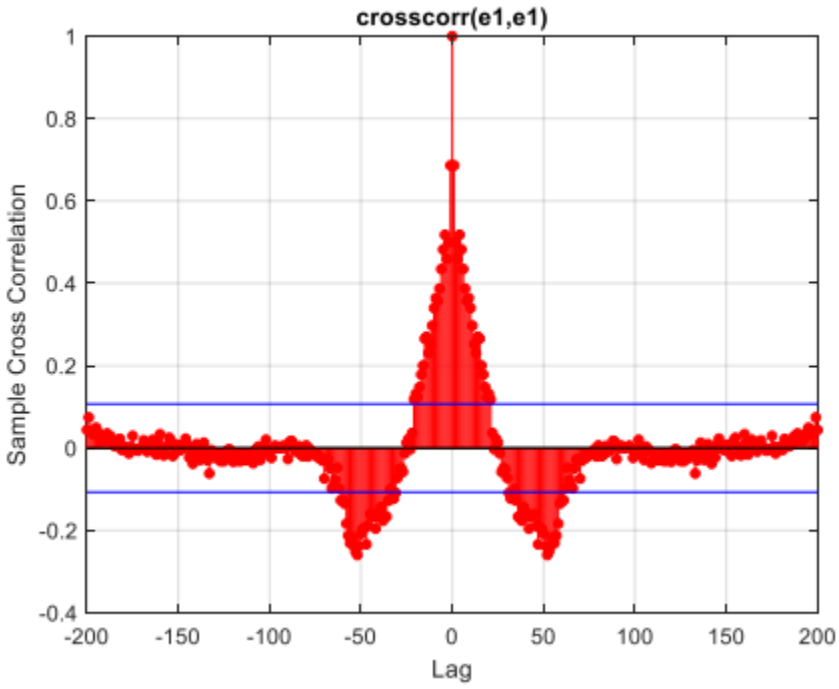


Figure 136 error cross-correlation (system 1)

The fitness percentage is suitable, and the error tends to whiten. Overall, NARX seems to have better fitness percentages, and error whitening is also better in that case.

1.4) System 2

In this question, we identify two systems using three methods: RBF, MLP, and NRBF. In this section, it should be noted that since the systems have nonlinear dynamics, effective dynamics need to be applied to the system. The effective dynamics have been calculated for both systems using the FS method in all cases. In all cases, through trial and error, we have chosen the tanh activation function for the hidden layer and the Levenberg-Marquardt weight training method. The learning rate is set to 0.01, and for simplicity, a single-layer network is selected.

MLP model

This section has been covered in question 2, and the results of the MLP neural network with the selected dynamics are presented. We refrain from repeating the results. We use the static data and the network structure extracted in that section.

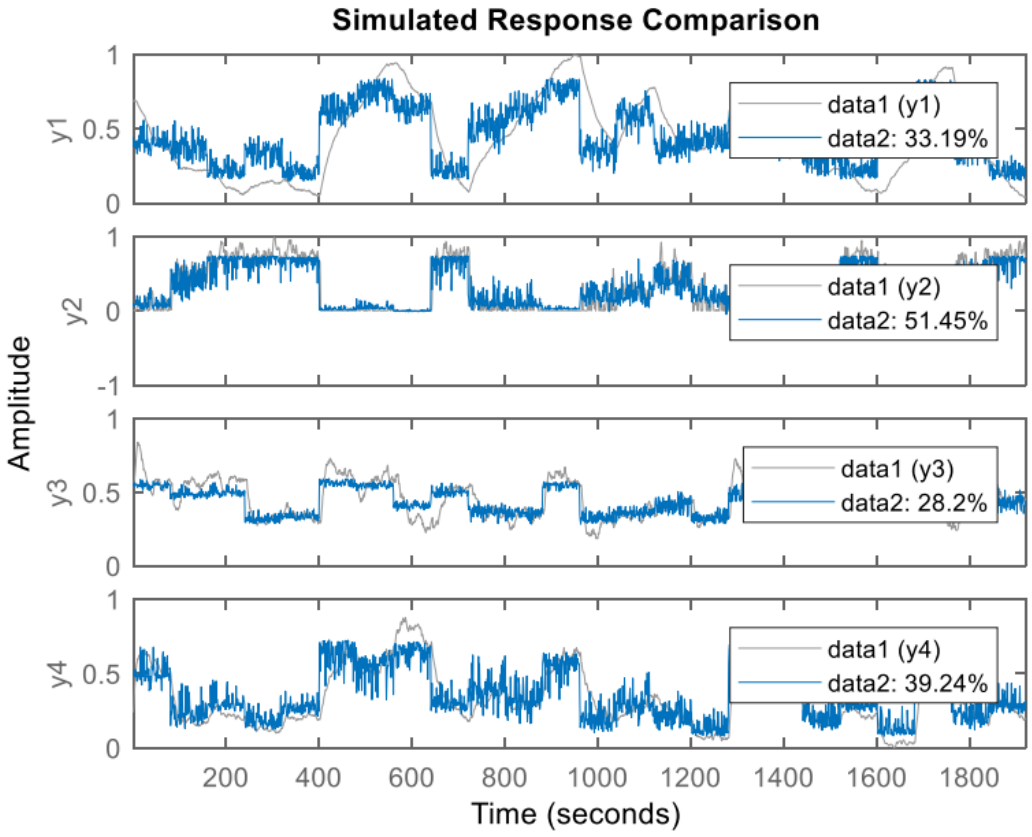


Figure 137 The output of the estimated and the real system using MLP- static (system 2)

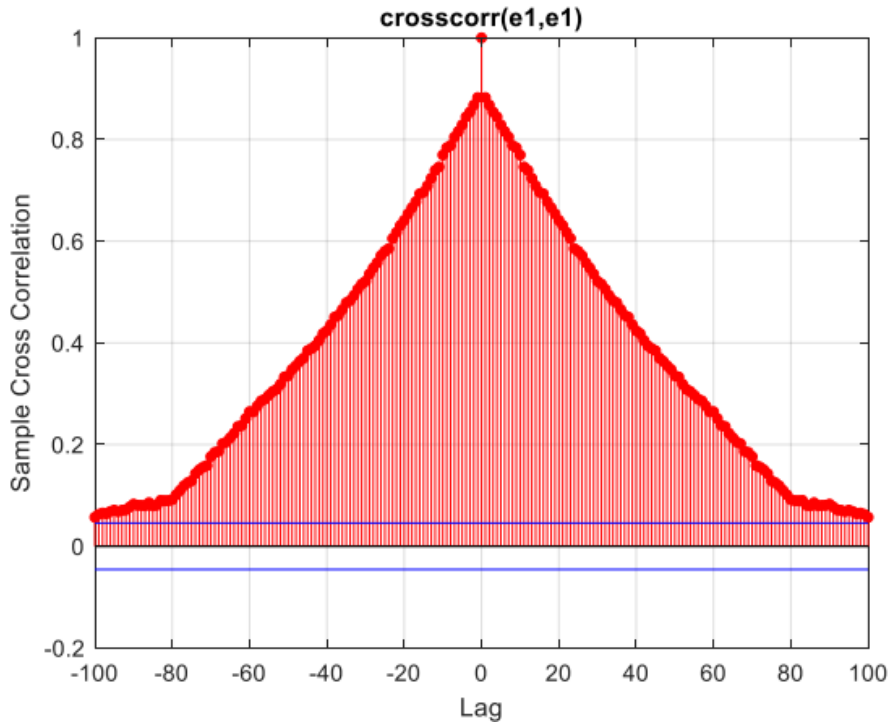


Figure 138 Error cross-correlation for first output

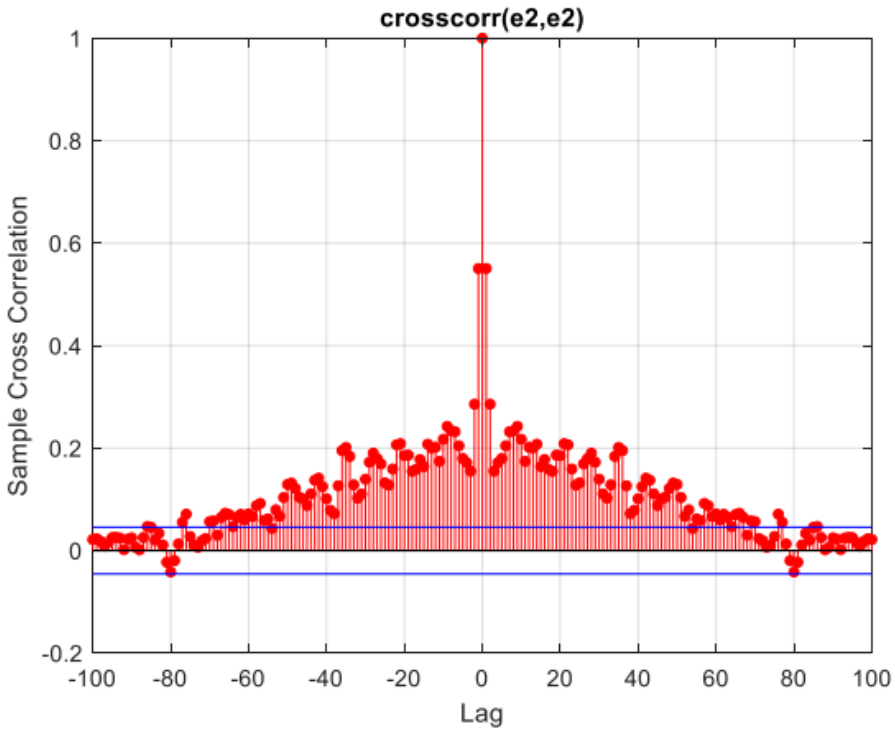


Figure 139 Error cross-correlation for second output

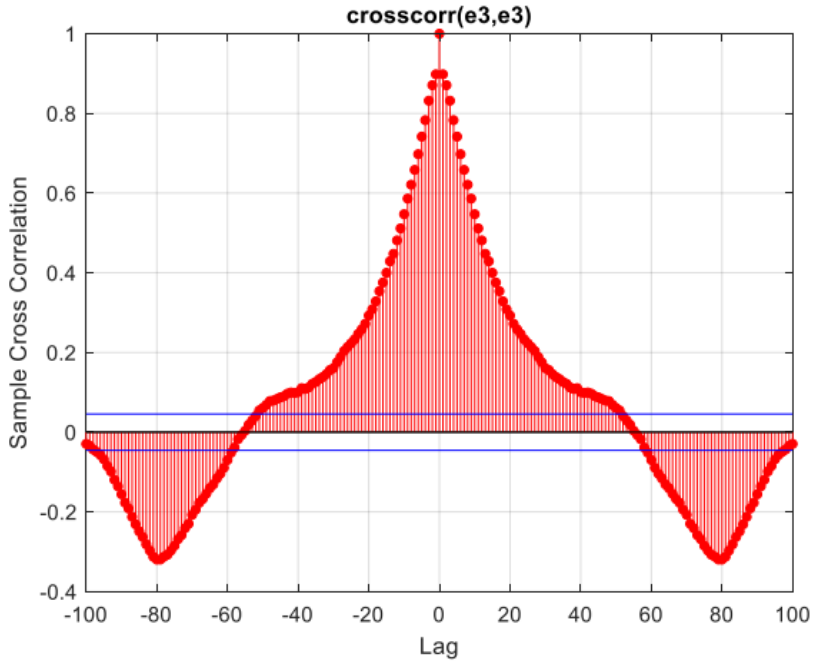


Figure 140 Error cross-correlation for third output

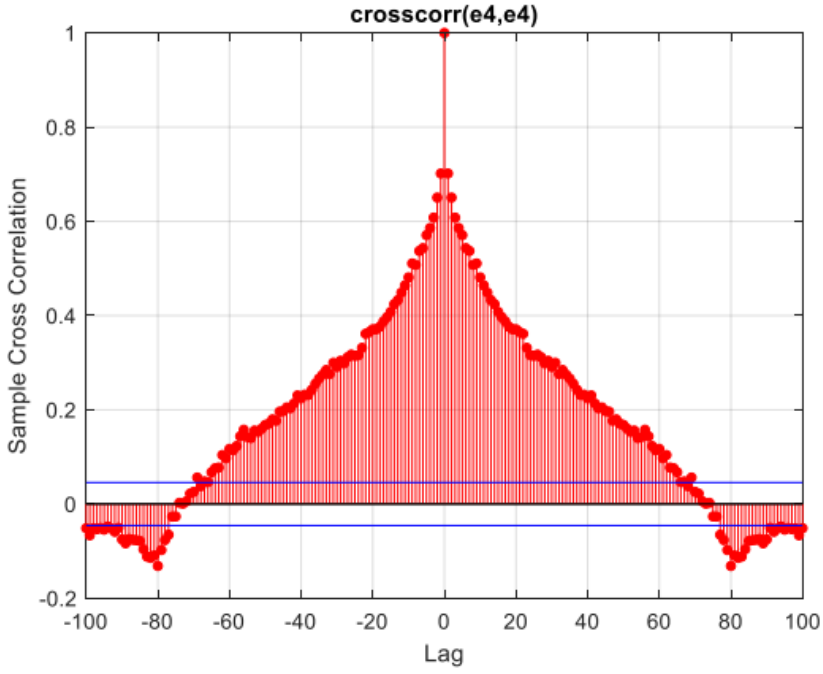


Figure 141 Error cross-correlation for the fourth output

It is observed that with static data, the fitness percentages are low, and error whitening has not occurred. However, with dynamic data, we have both error whitening and high fitness percentages.

RBF

Through trial and error (based on fitness percentage and error whitening), we placed 6 neurons in the hidden layer for.

Static data:

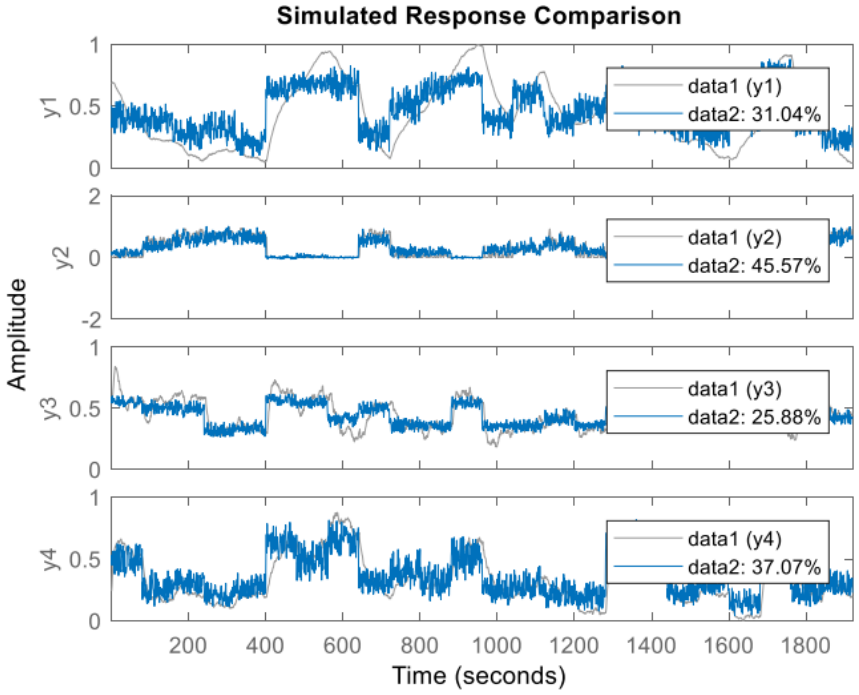


Figure 142 The actual and estimated outputs of the system-RBF-Static (steamgen)

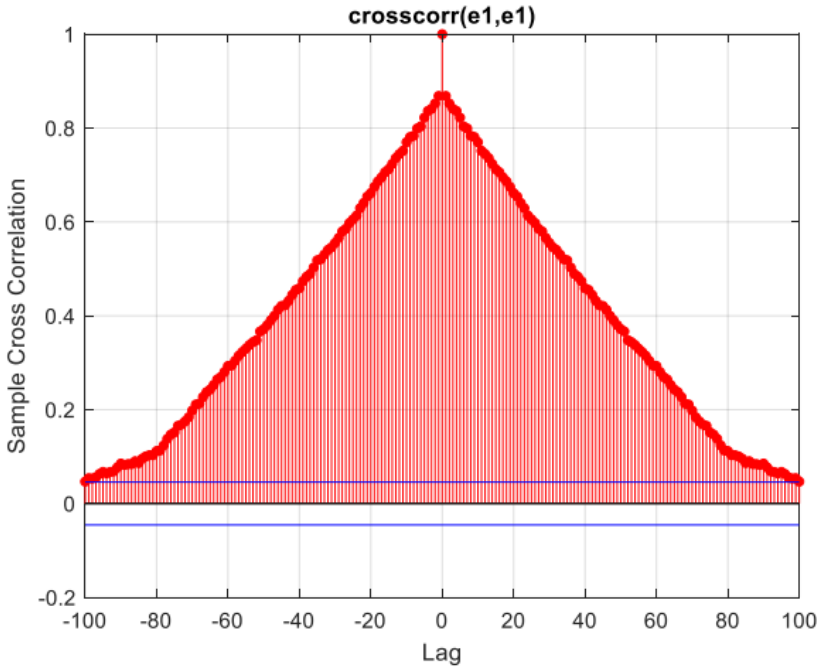


Figure 143 First outptu error cross-correlation

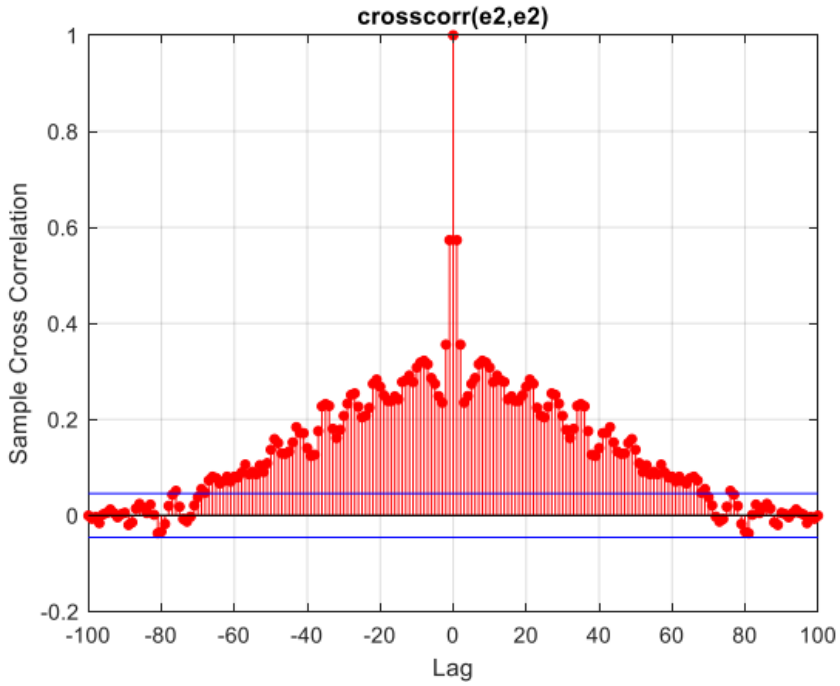


Figure 144 Second output error cross-correlation (steamgen)

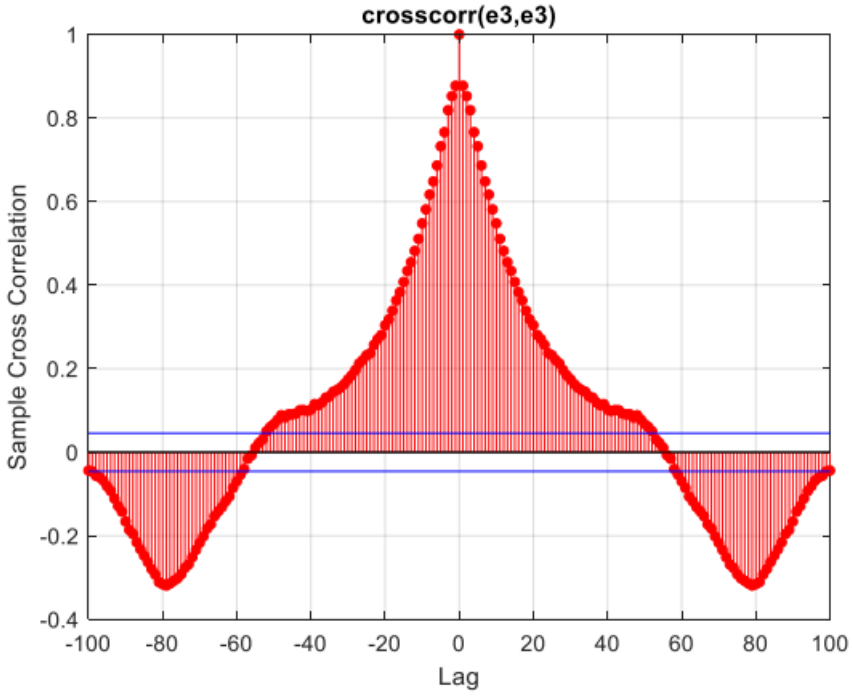


Figure 145 Third output error cross-correlation (steamgen)

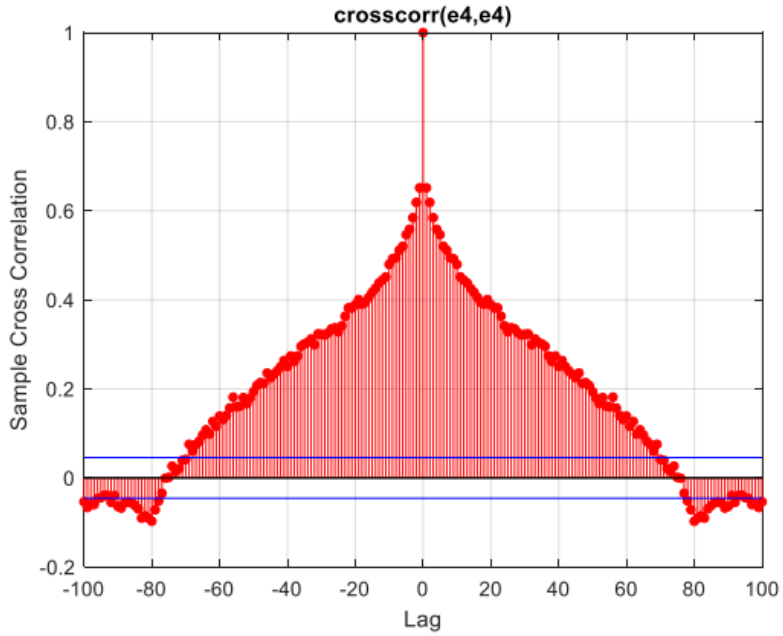


Figure 146 Fourth output error cross-correlation (steamgen)

It appears that the fitness percentages are not satisfactory, and error whitening has not occurred. Now, let's examine the dynamic mode:

Dynamic data:

The network structure is as follows, selected through trial and error on fitness percentage and error whitening:

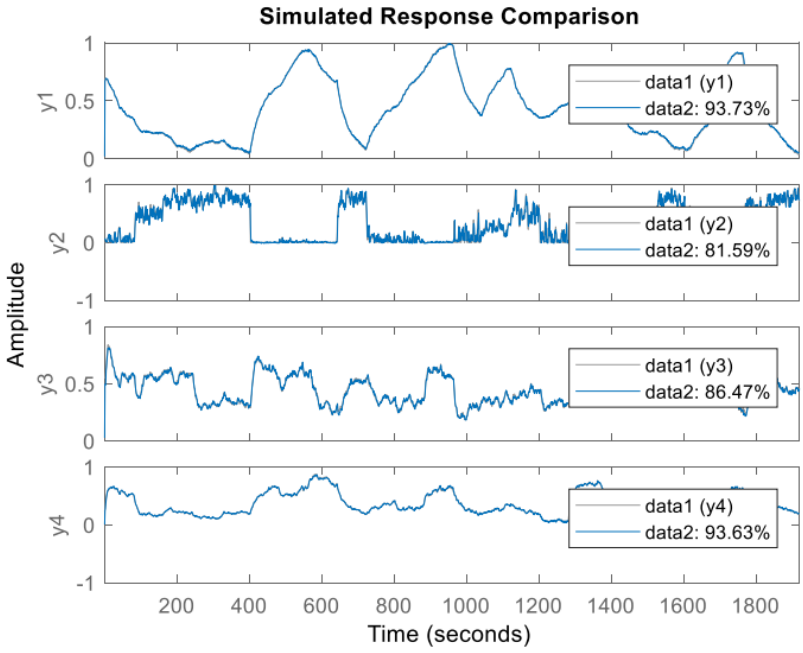


Figure 147 The actual and estimated outputs of the system-RBF-dynamic (steamgen)

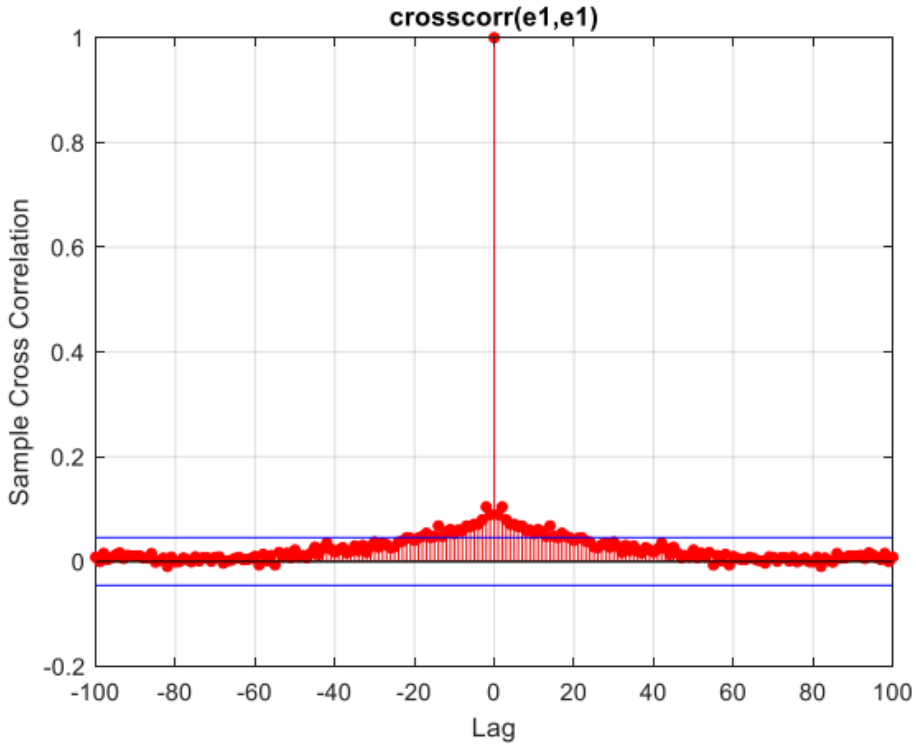


Figure 148 Error cross-correlation first output-RBF model-(steamgen)

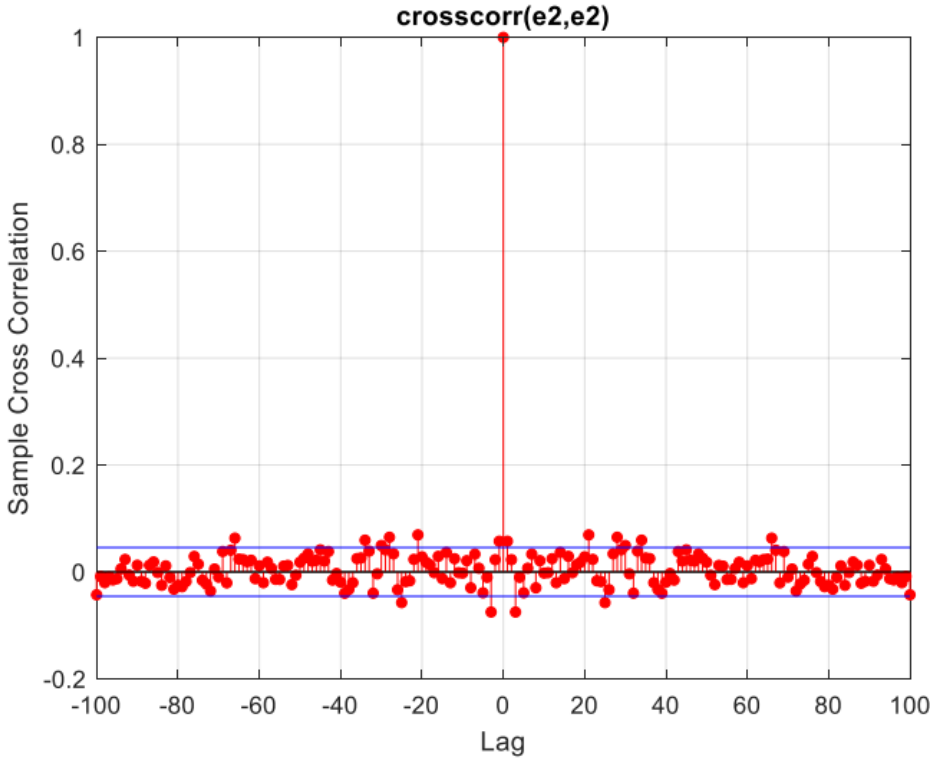


Figure 149 Error cross-correlation second output-RBF model-(steamgen)

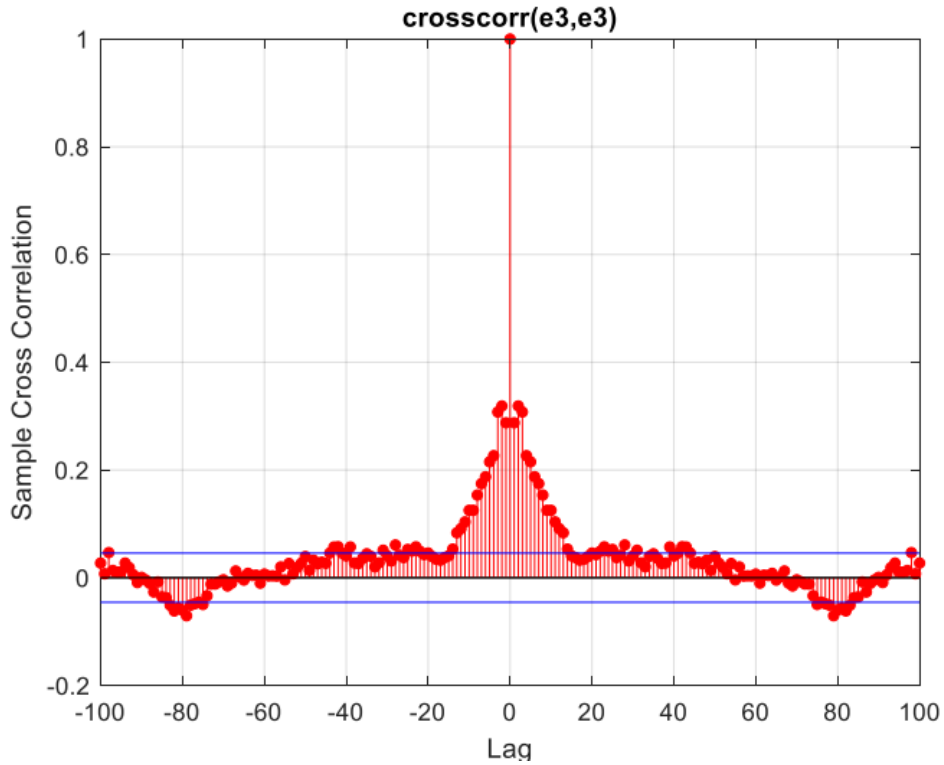


Figure 150 Error cross-correlation third output-RBF model-(steamgen)

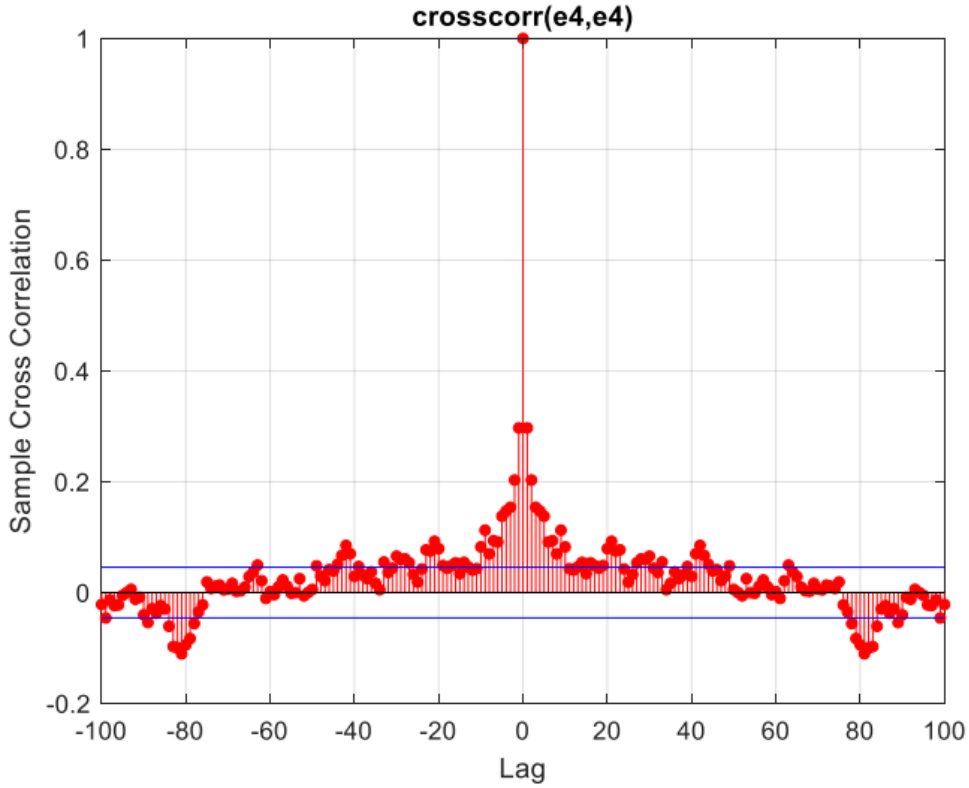


Figure 151 Error cross-correlation fourth output-RBF model-(steamgen)

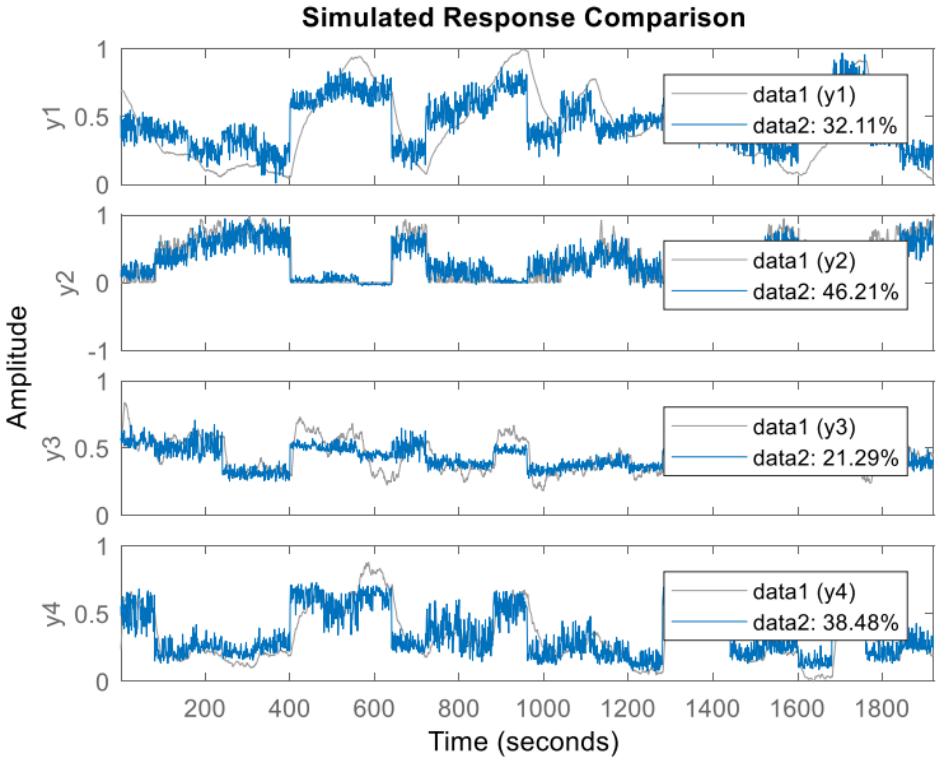


Figure 152 actual and estimated outputs pf NRBF static model (system 2)

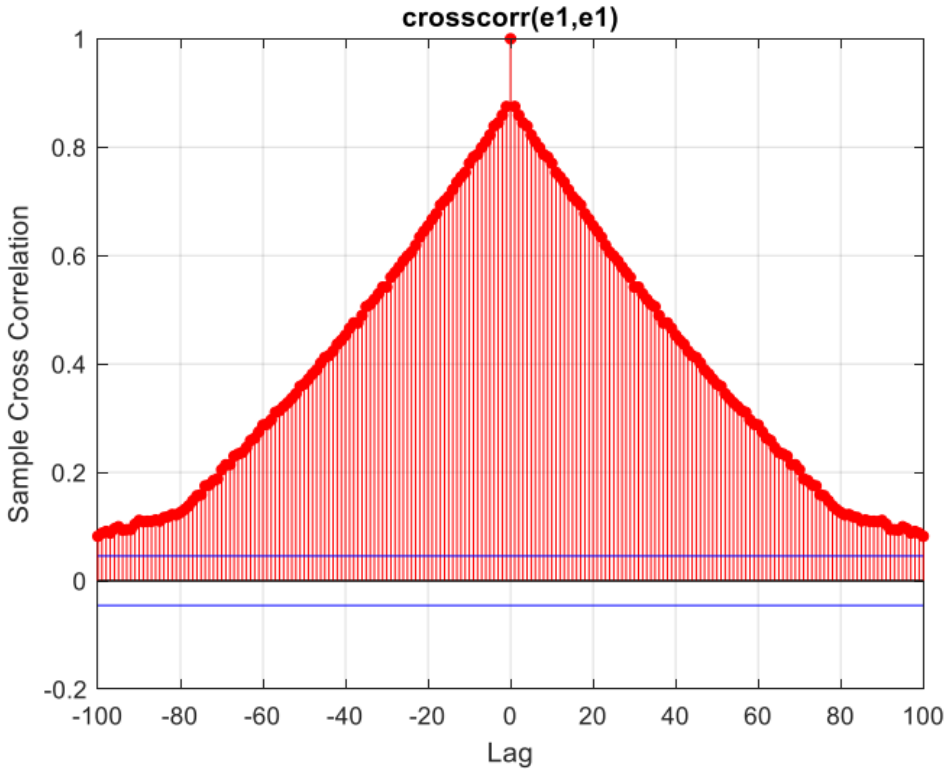


Figure 153 First output cross-correlation

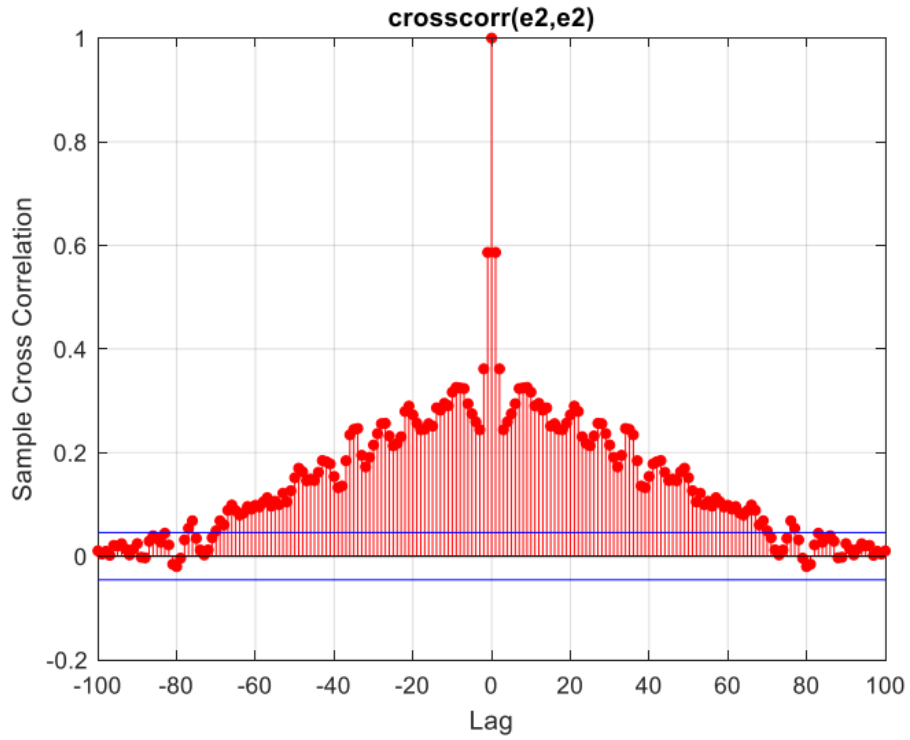


Figure 154 Second output cross-correlation

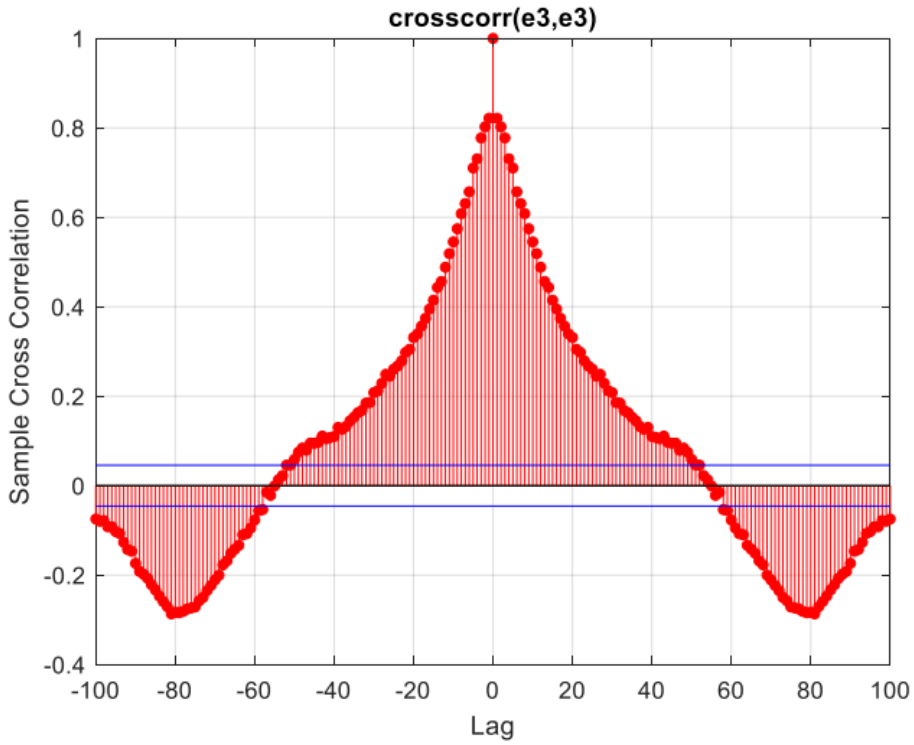


Figure 155 Third output cross-correlation

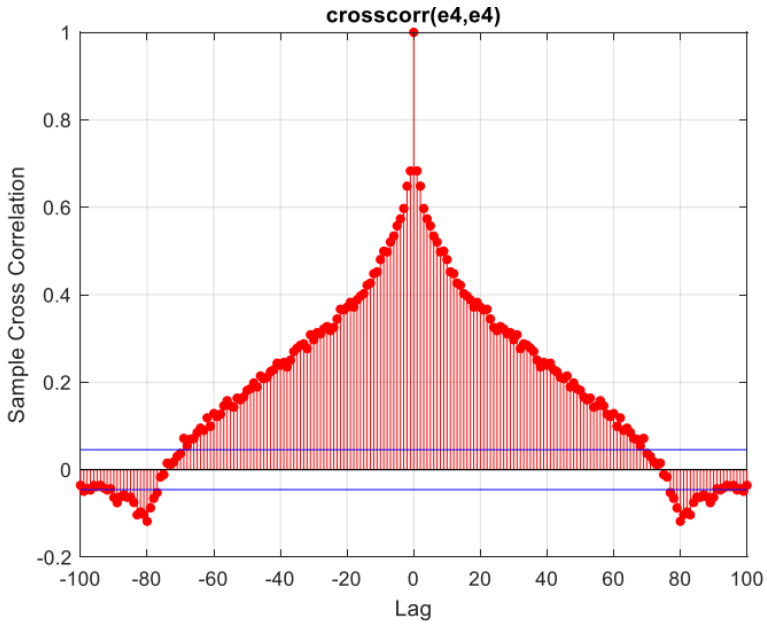


Figure 156 Fourth output cross-correlation

It is observed that the fitness percentage is low, and error whitening has not occurred. Now, let's examine the dynamic mode:

Dynamic data:

Through trial and error, we choose the same RBF network structure and normalized the activation functions. The results of this approach are as follows:

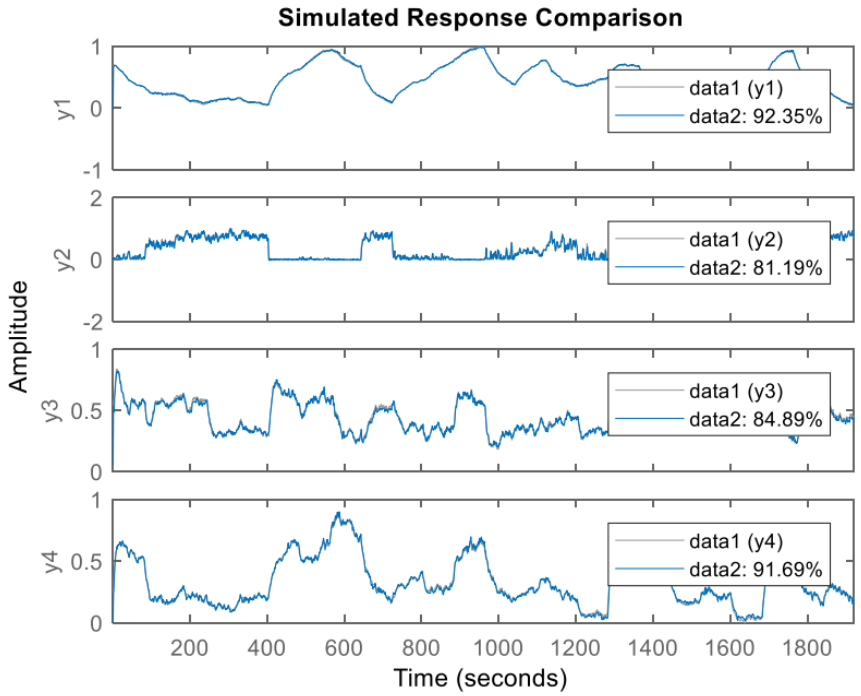


Figure 157 actual and estimated outputs pf NRBF dynamic model (system 2)

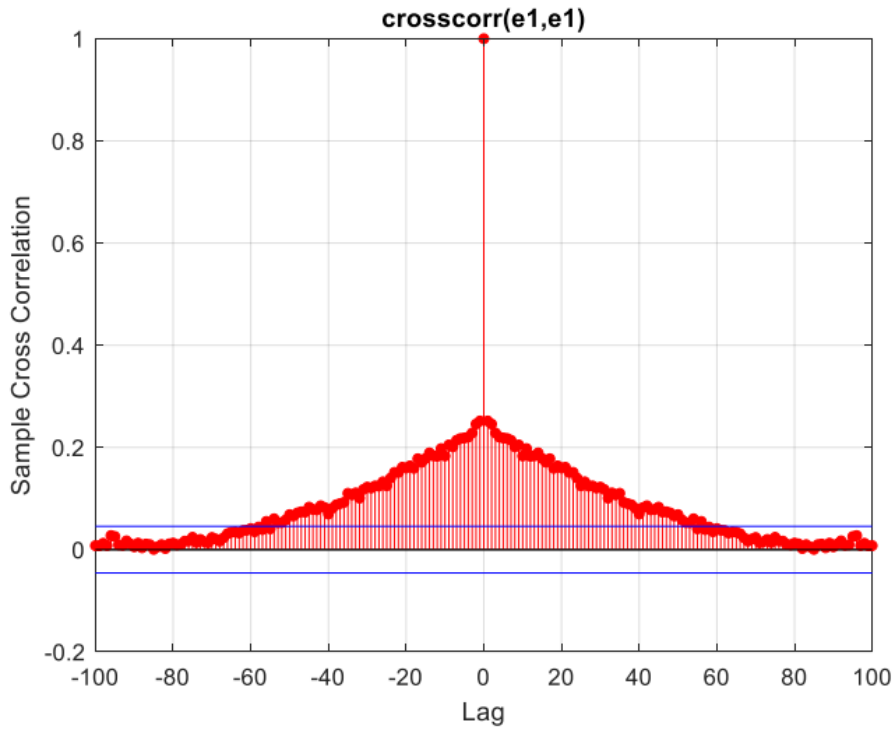


Figure 158 cross-correlation of first output error for NRBF (steamgen)

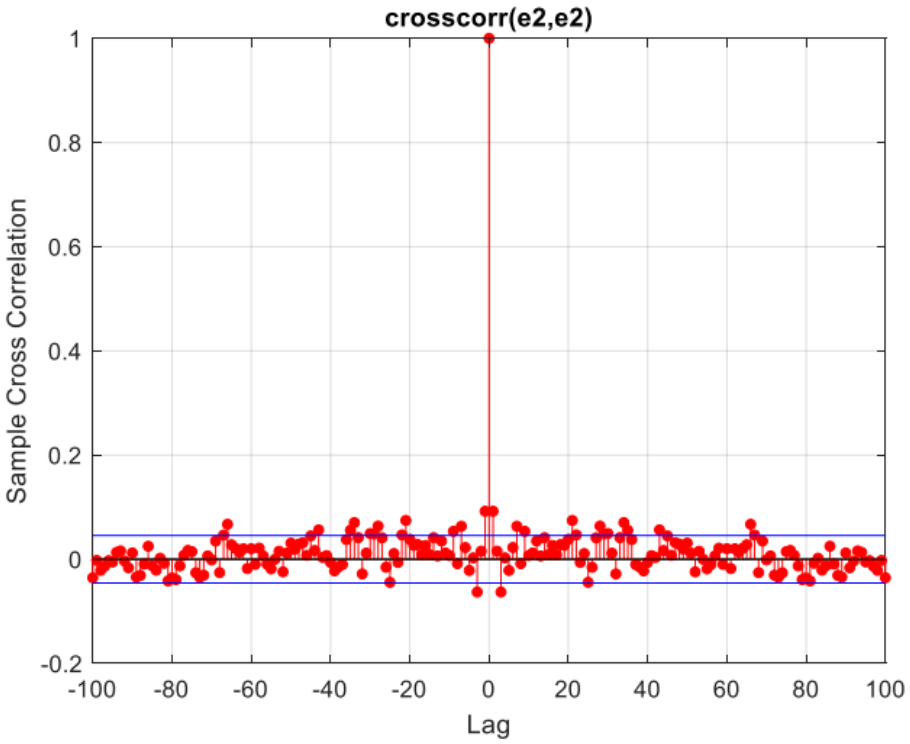


Figure 159 cross-correlation of second output error for NRBF (steamgen)

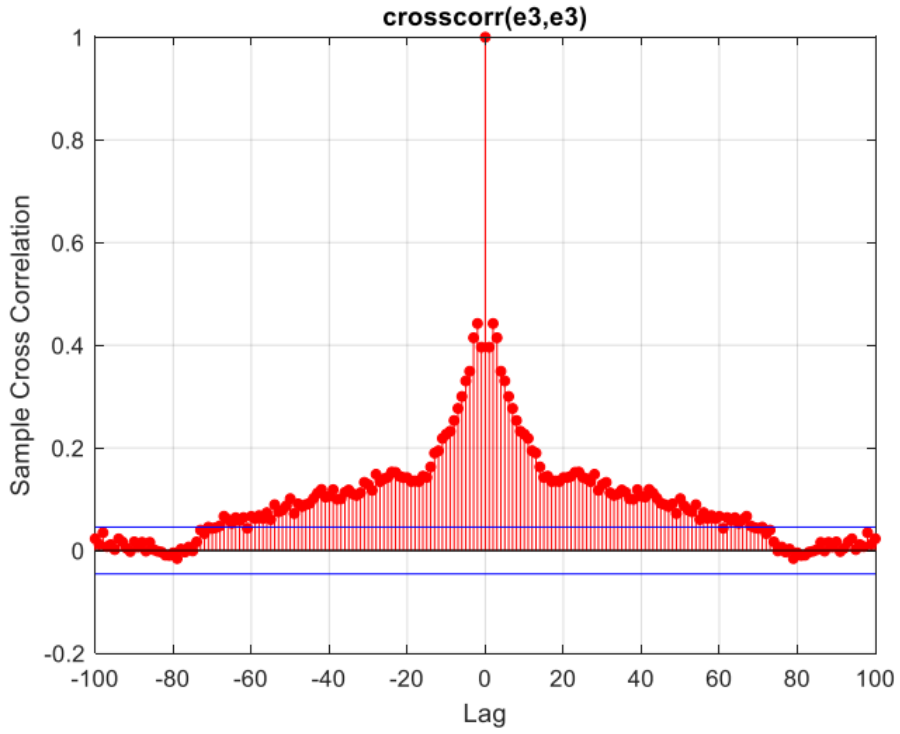


Figure 160 cross-correlation of third output error for NRBF (steamgen)

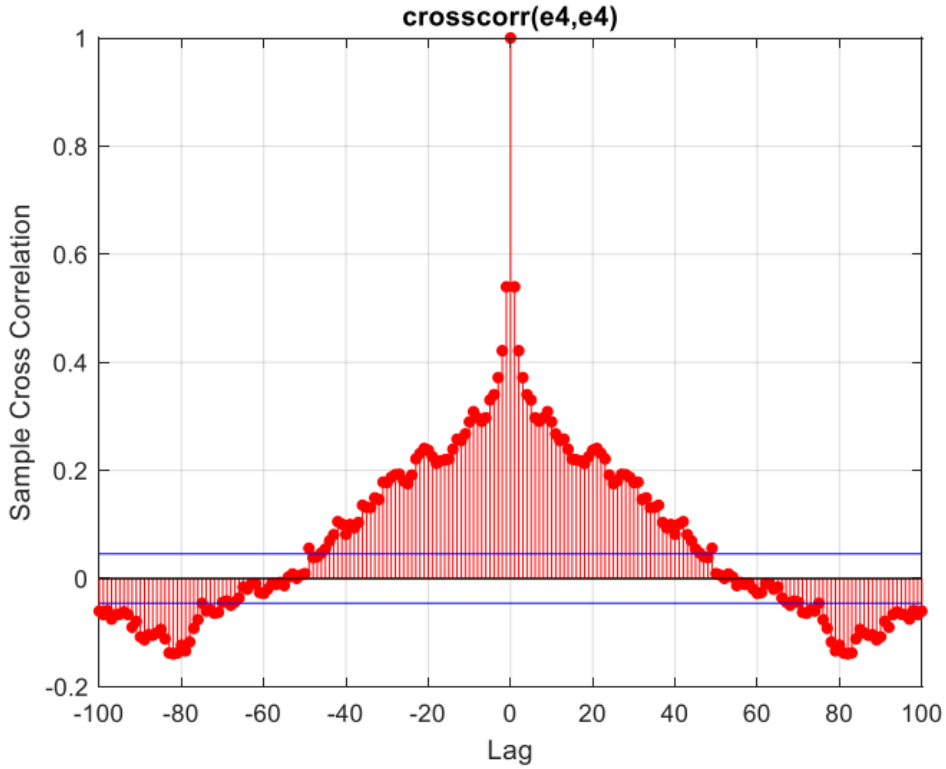


Figure 161 cross-correlation of fourth output error for NRBF (steamgen)

In terms of fitness percentage, MLP, RBF, and NRBF are ranked in that order. MLP has better error whitening compared to the others, but to elaborate further; for the third output, error whitening in RBF is better than MLP, while for the fourth output, MLP is better. In NRBF, error whitening is undesirable for all outputs.

1.4) System 1

4 neurons have been selected in the hidden layer of the network through trial and error (considering fitness percentage and error whitening). The MLP network has been examined with dynamic data in previous section

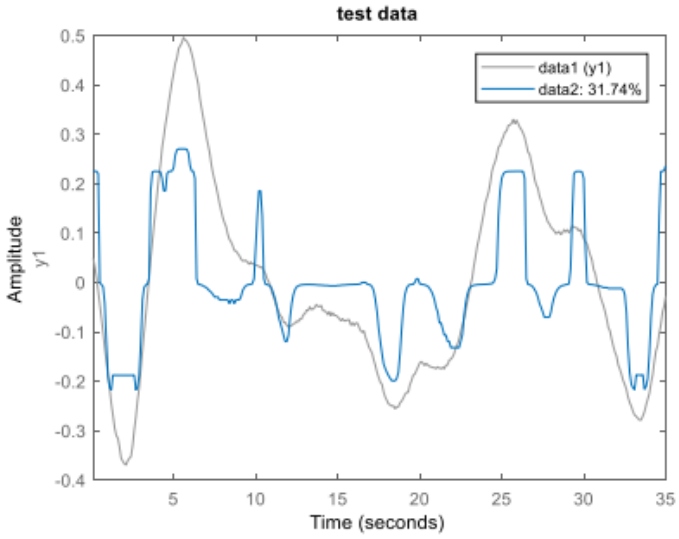


Figure 162 The actual output and the estimated output using MLP static (system 1)

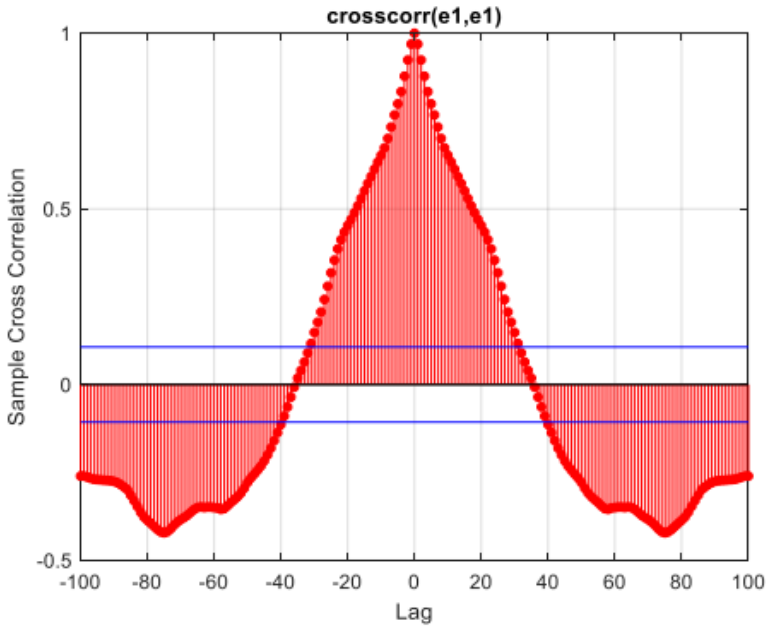


Figure 163 Error cross-correlation MLP static (system 1)

The fitness is not good and error whitening did not happened.

RBF

Static data:

2 neurons chosen

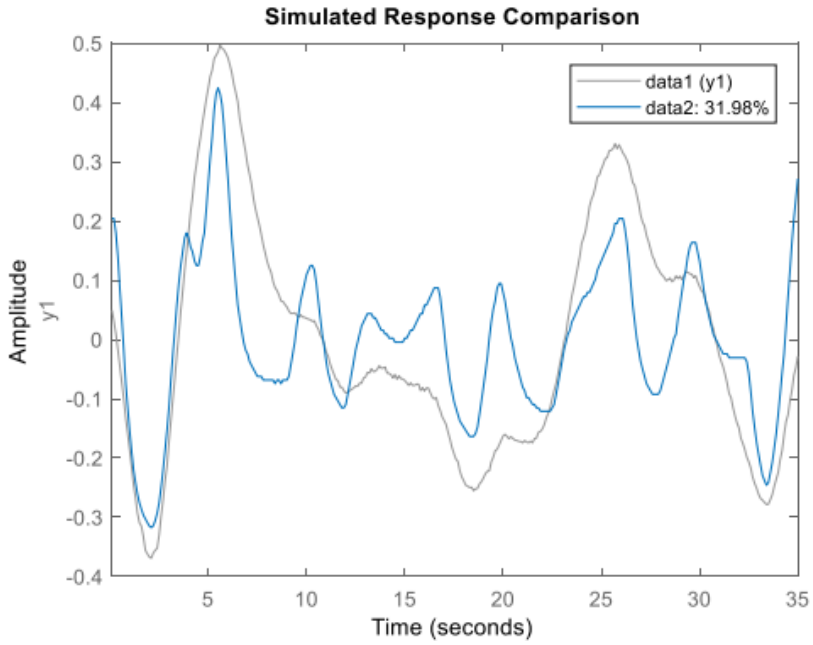


Figure 164 Actual and estimated system outputs static RBF model (system 1)

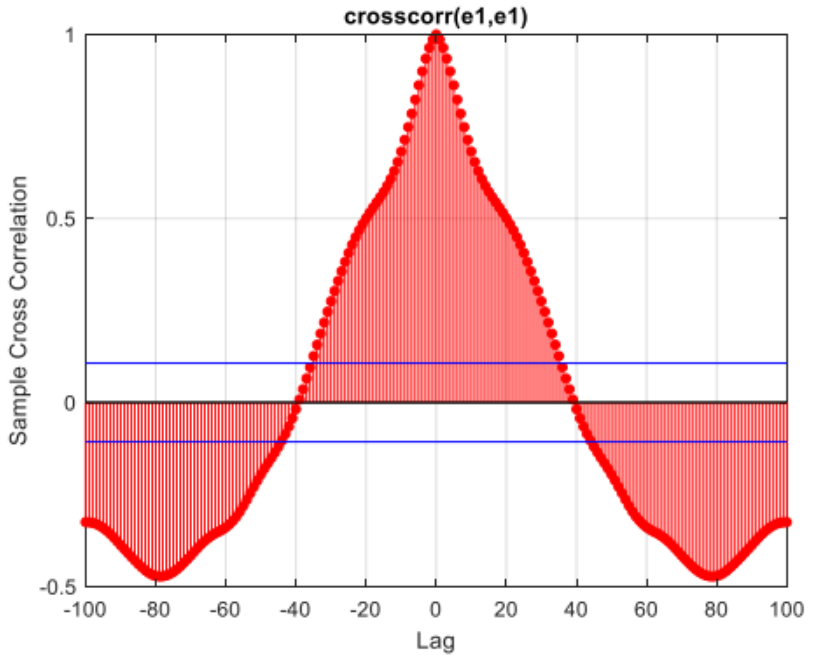


Figure 165 Error cross-correlation (System 1)

Fitness is not good and whitening has not happened.

Dynamic data:

The number of 5 neurons is appropriate (determined through trial and error).

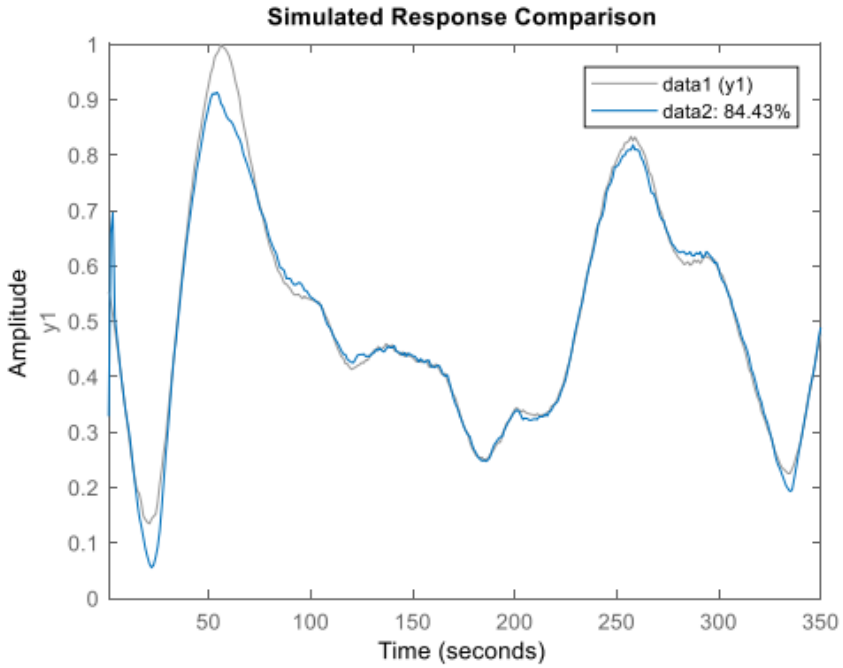


Figure 166 Actual and estimated system outputs dynamic RBF model (system 1)

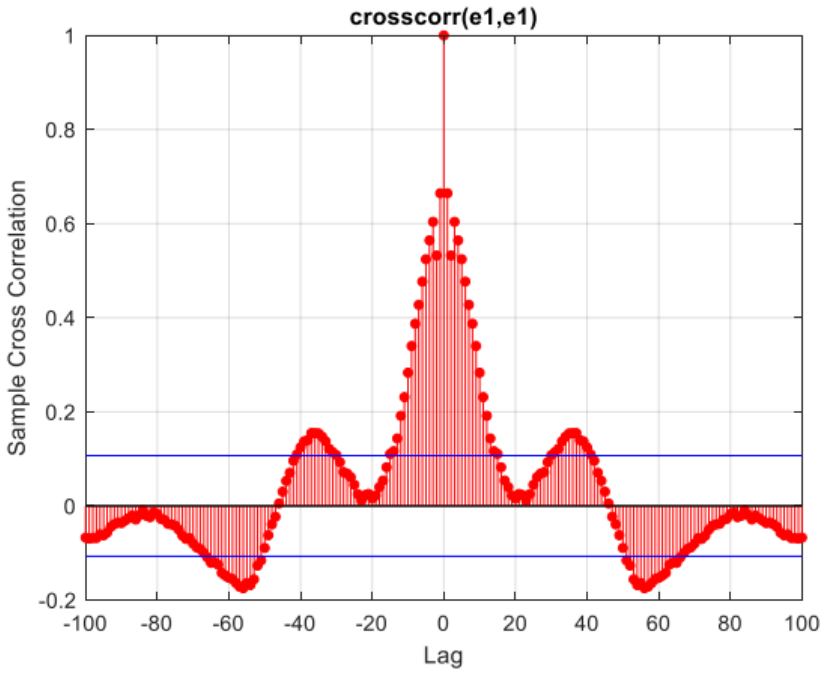


Figure 167 Error cross-correlation (System 1)

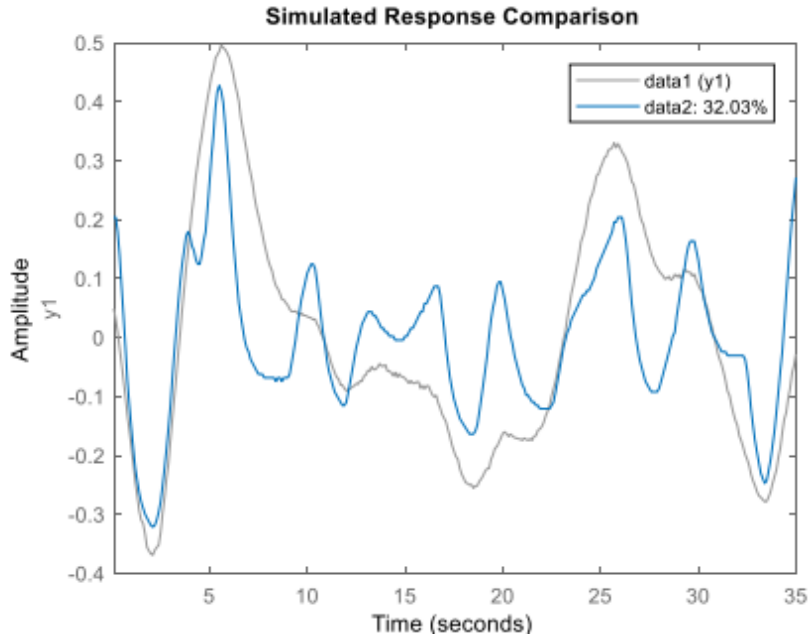


Figure 168 Actual and estimated system outputs dynamic NRBF model (system 1)

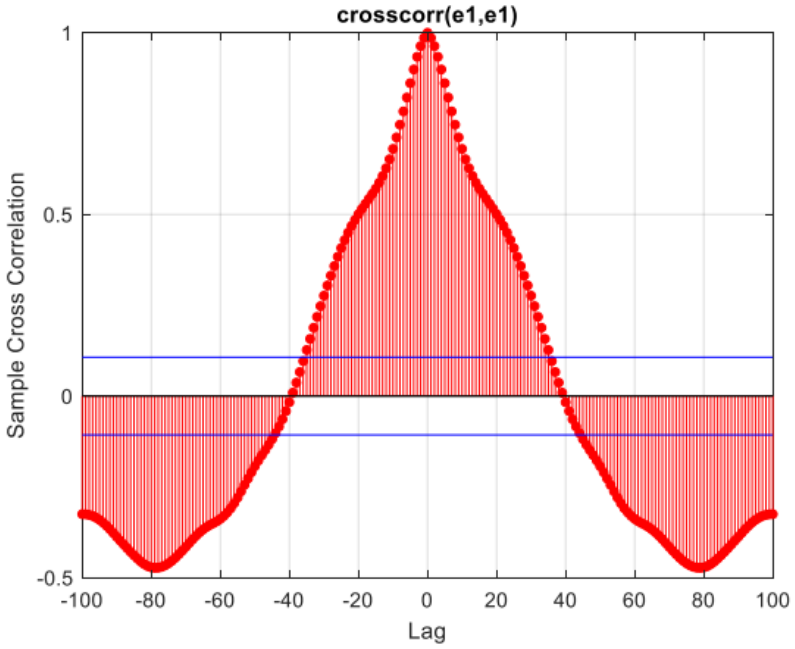


Figure 169 Error cross-correlation

Dynamic data:

The number of 5 neurons is appropriate and obtained through trial and error.

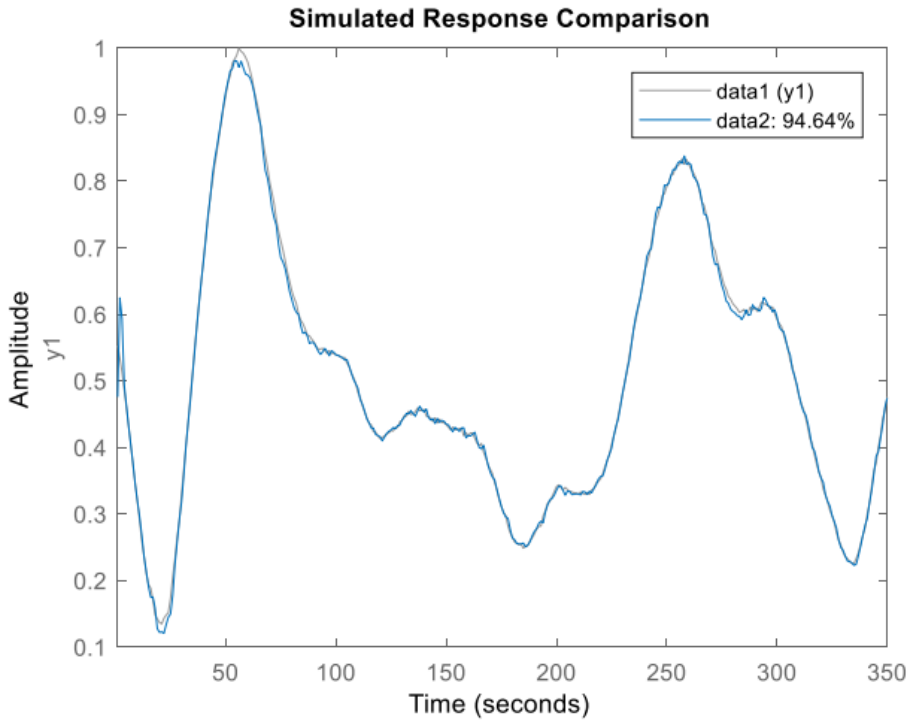


Figure 170 Actual and estimated system outputs dynamic NRBF model (system 1)

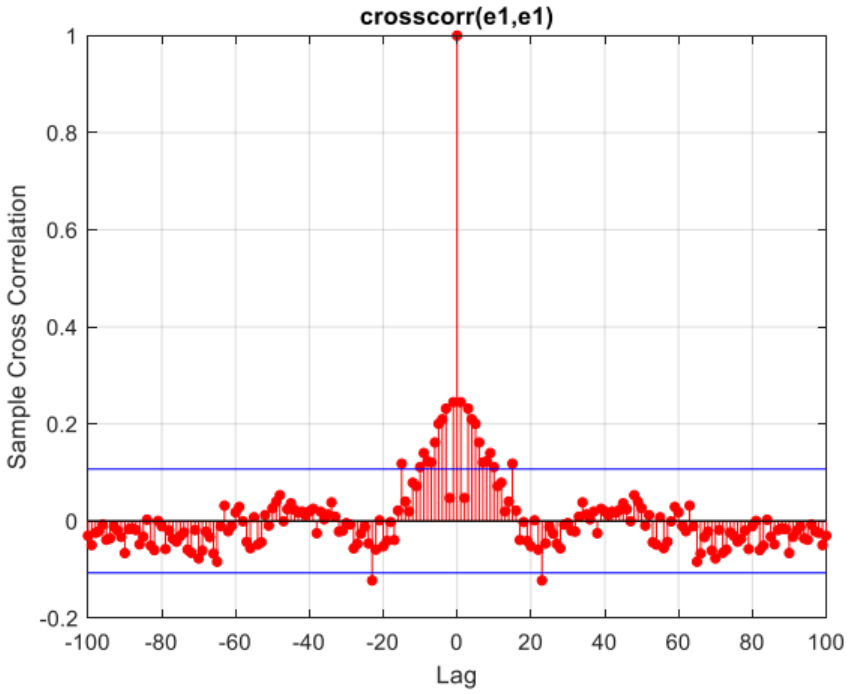


Figure 171 Error cross correlation

It can be observed that whitening error is better than RBF, and the fitting percentage is also higher. The fitting percentage is close to MLP.

1.5) System 2

In this section, we identify two systems using the RNN method. We use the **newelm** command to create an RNN network. The structure is chosen as follows after trial and error:

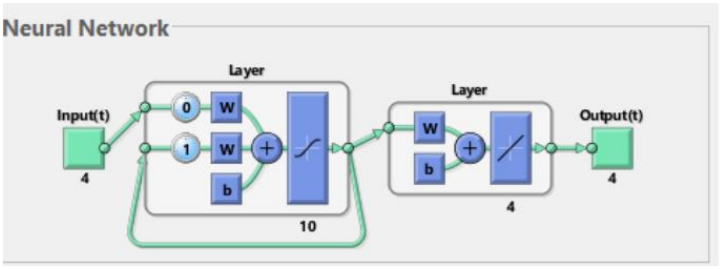


Figure 172 The RNN GUI of matlab

The activation function tansig and the Levenberg-Marquardt training method have been selected. We have chosen a learning rate of 0.01 (through trial and error). The results are as follows:

On the entire data:

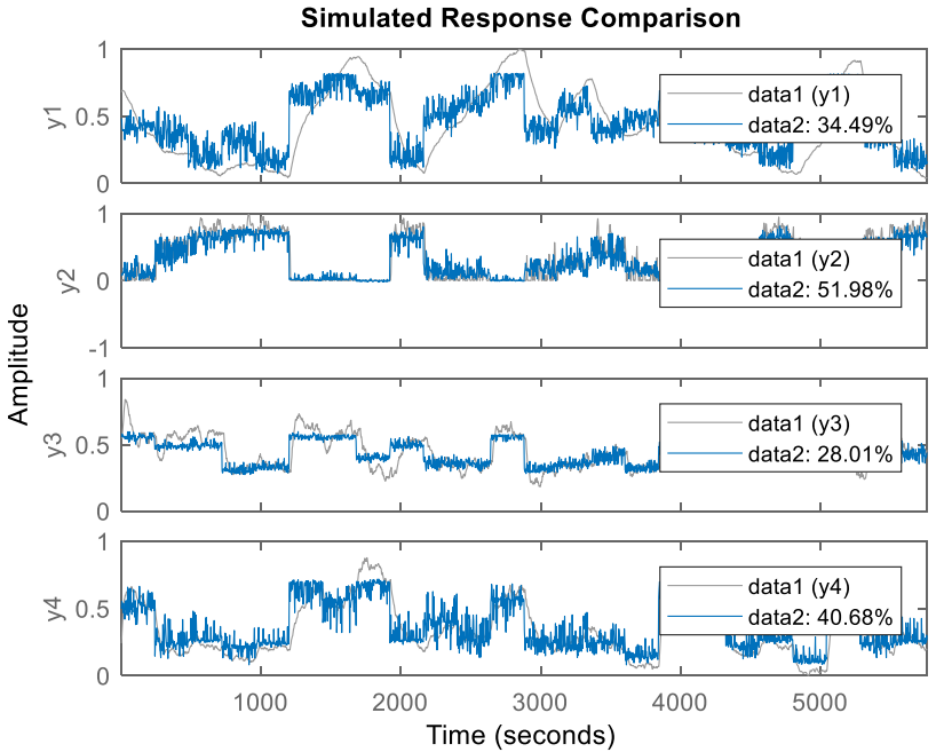


Figure 173 the output plot of the real system and the estimated system (using RNN) (static)

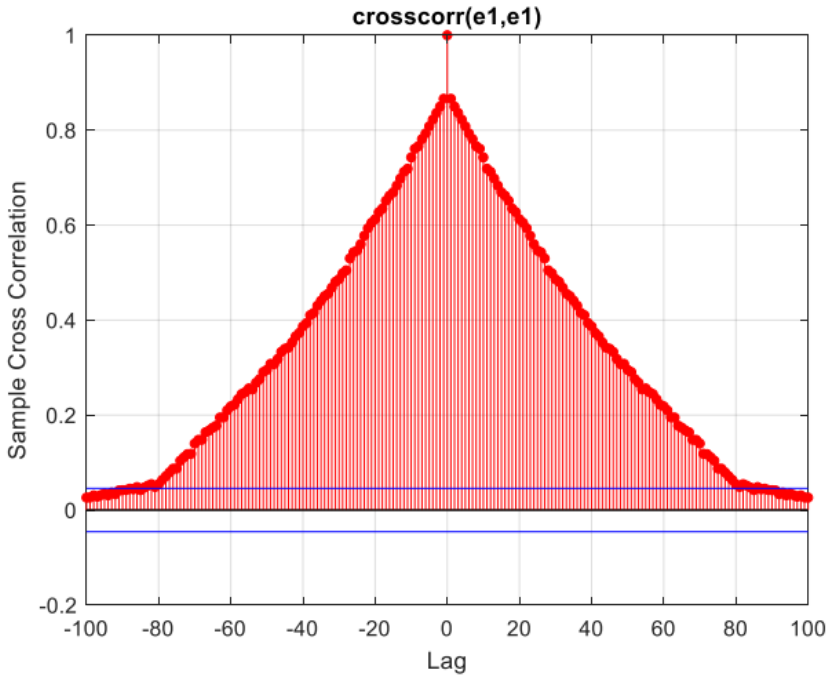


Figure 174 error cross-correlation first output (steamgen, using RNN)

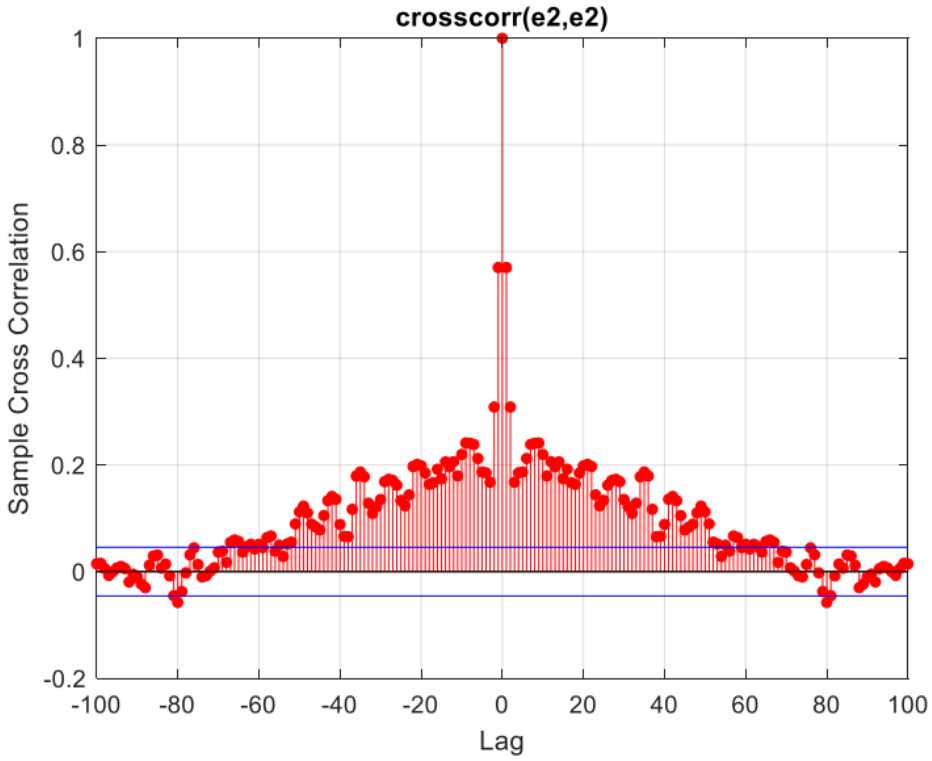


Figure 175 error cross-correlation second output (steamgen, using RNN)

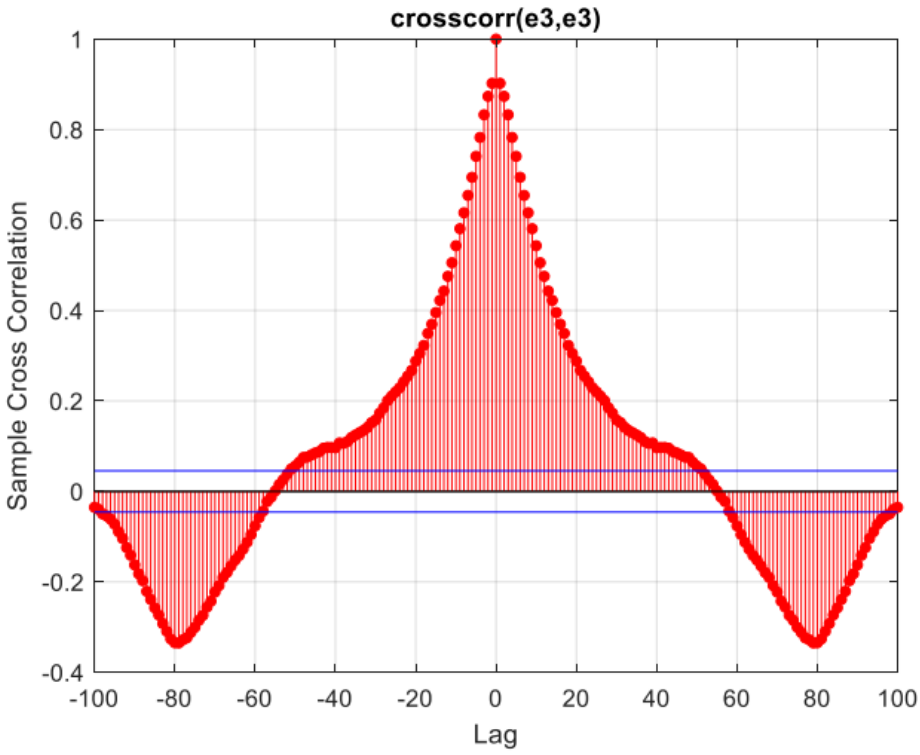


Figure 176 error cross-correlation third output (steamgen, using RNN)

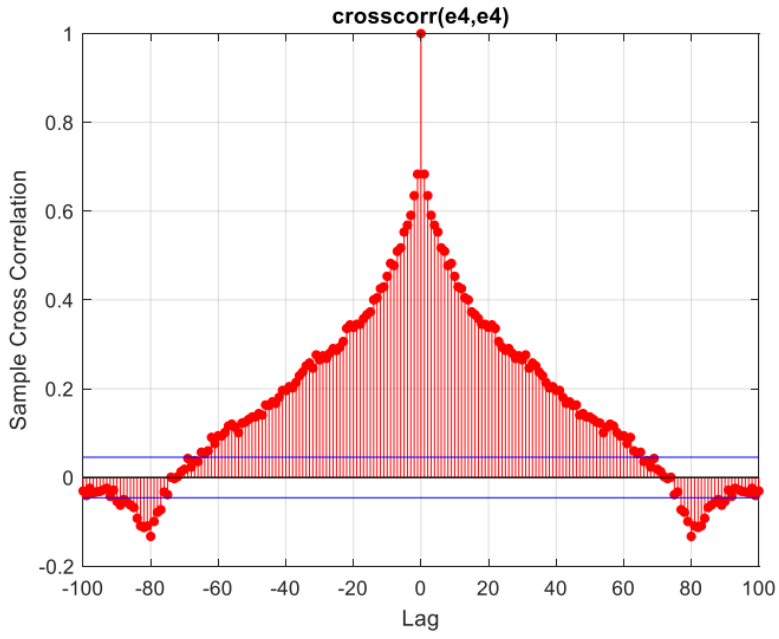


Figure 177 error cross-correlation fourth output (steamgen, using RNN)

It is observed that the fitting percentages are unsatisfactory, and there is a high correlation between errors. Regarding dynamic data: the parameters were mentioned above.

On dynamic data:

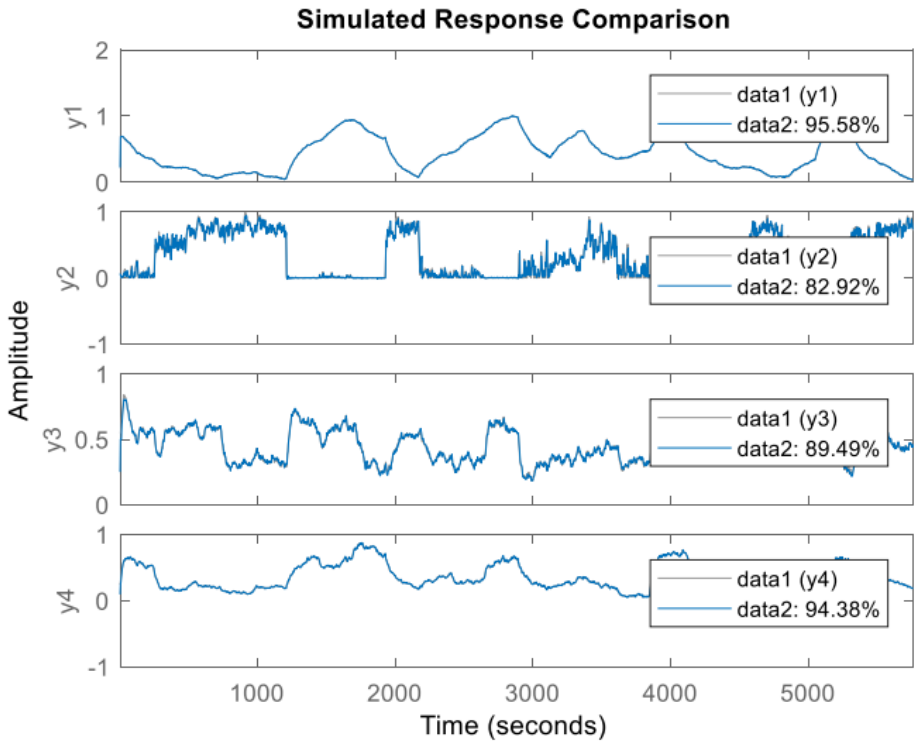


Figure 178 error cross-correlation first output (steamgen, using RNN) (dynamic data)

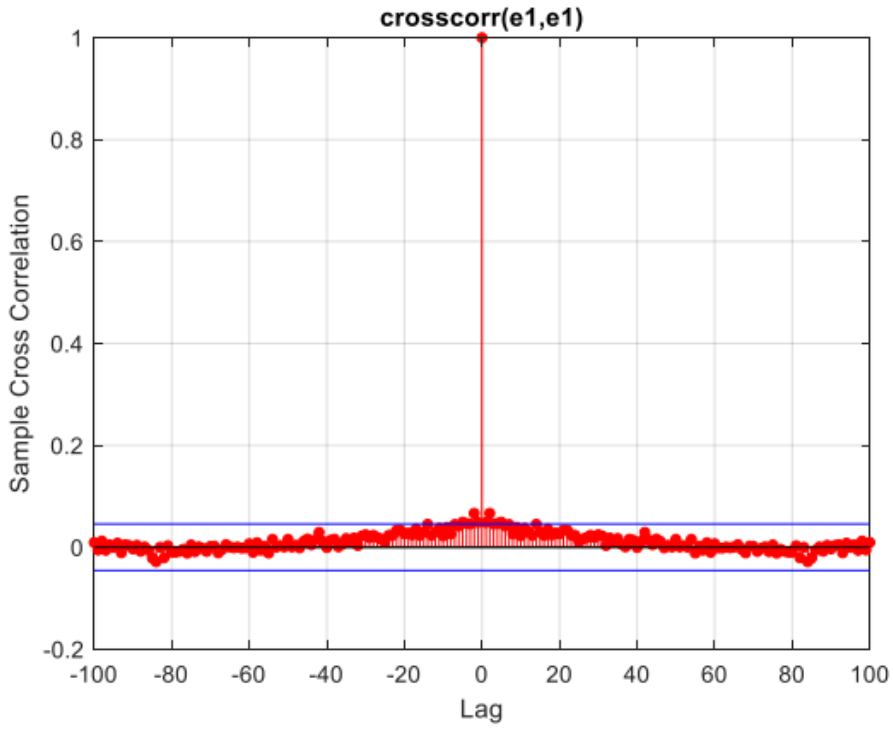


Figure 179 error cross-correlation first output (steamgen, dynamic data, using RNN)

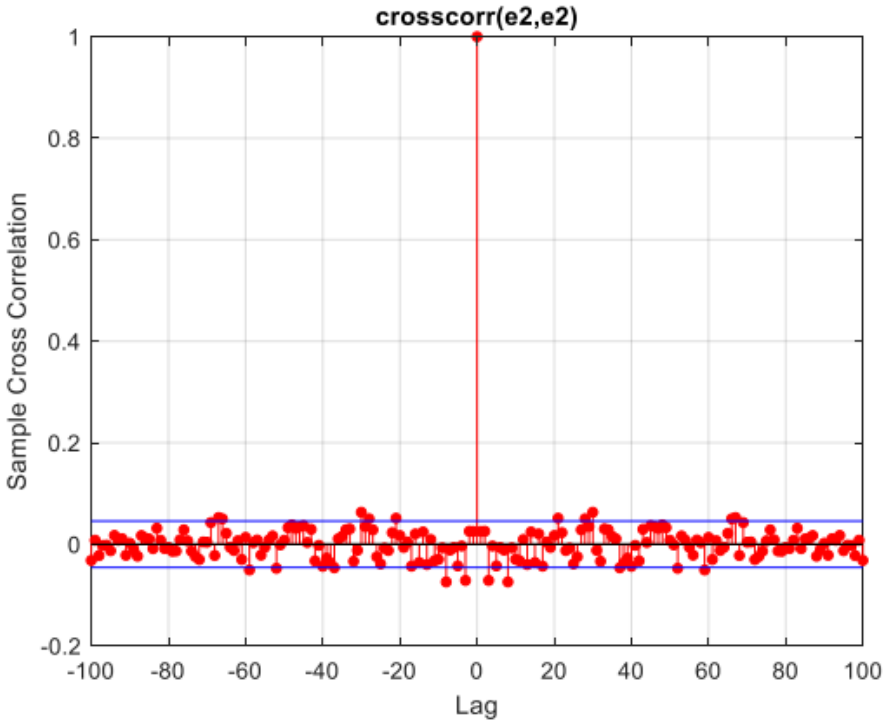


Figure 180 error cross-correlation second output (steamgen, dynamic data, using RNN)

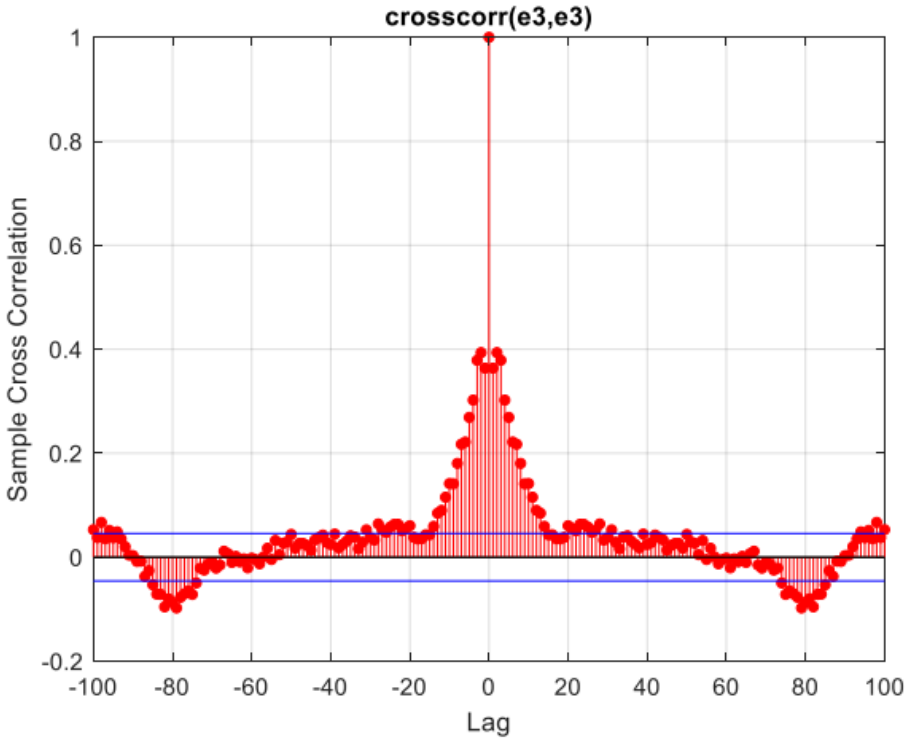


Figure 181 error cross-correlation third output (steamgen, dynamic data, using RNN)

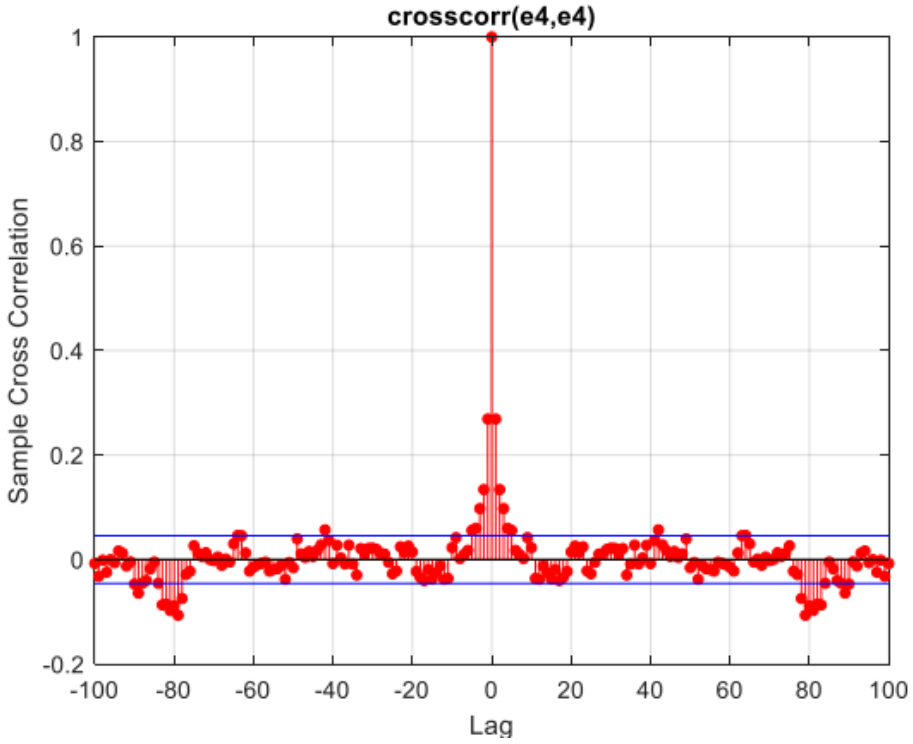


Figure 182 error cross-correlation fourth output (steamgen, dynamic data, using RNN)

It is observed that very satisfactory fitting percentages and whitening of errors are achieved. However, similar to previous cases, there is still room for improvement in whitening the error for the third output.

1.5) System 1

10 neurons and 50 epochs are chosen with trail and error. The activation function tansig and the Levenberg-Marquardt training method have been selected. We have chosen a learning rate of 0.01 (through trial and error).

On the entire data:

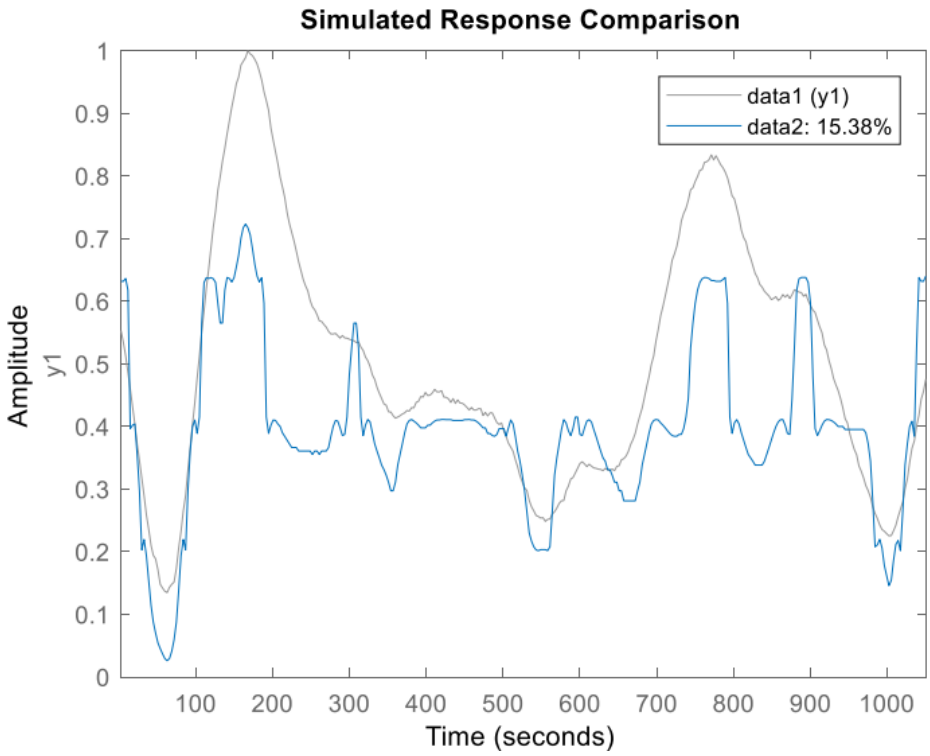


Figure 183 Actual and estimated system outputs RNN model (system 1)

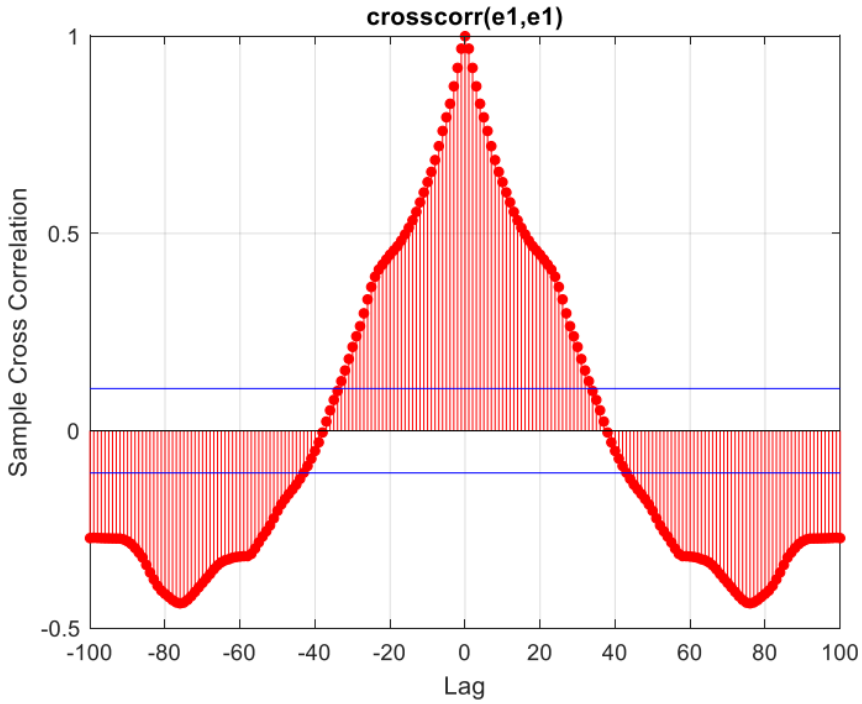


Figure 184 Error cross-correlation

On dynamic data:

5 neurons and 100 epochs:

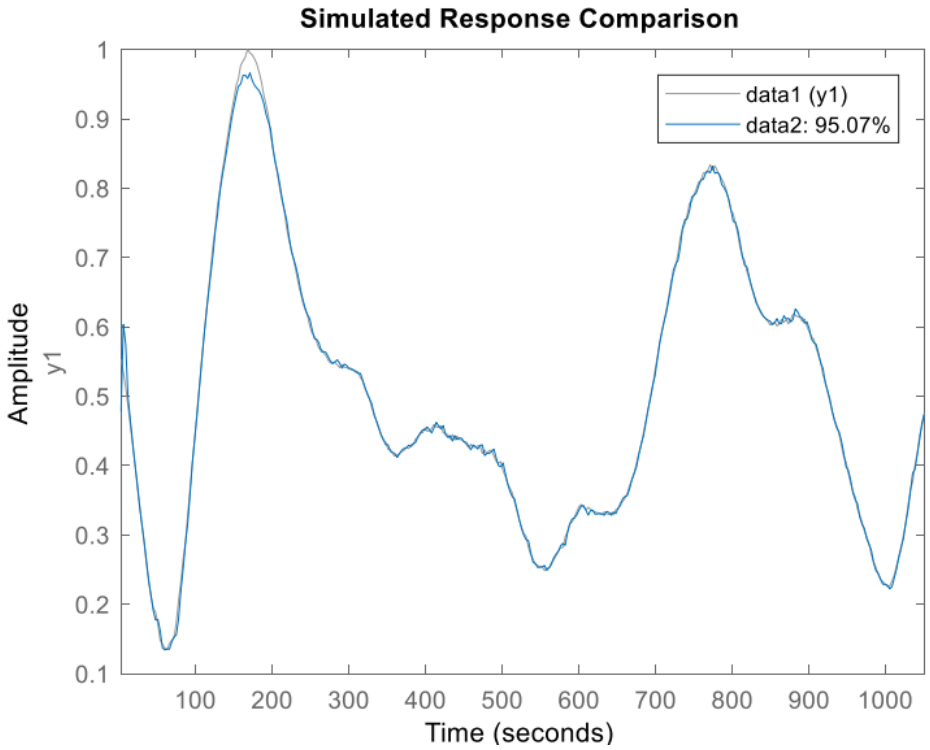


Figure 185 Actual and estimated system outputs RNN model dynamic data (system 1)

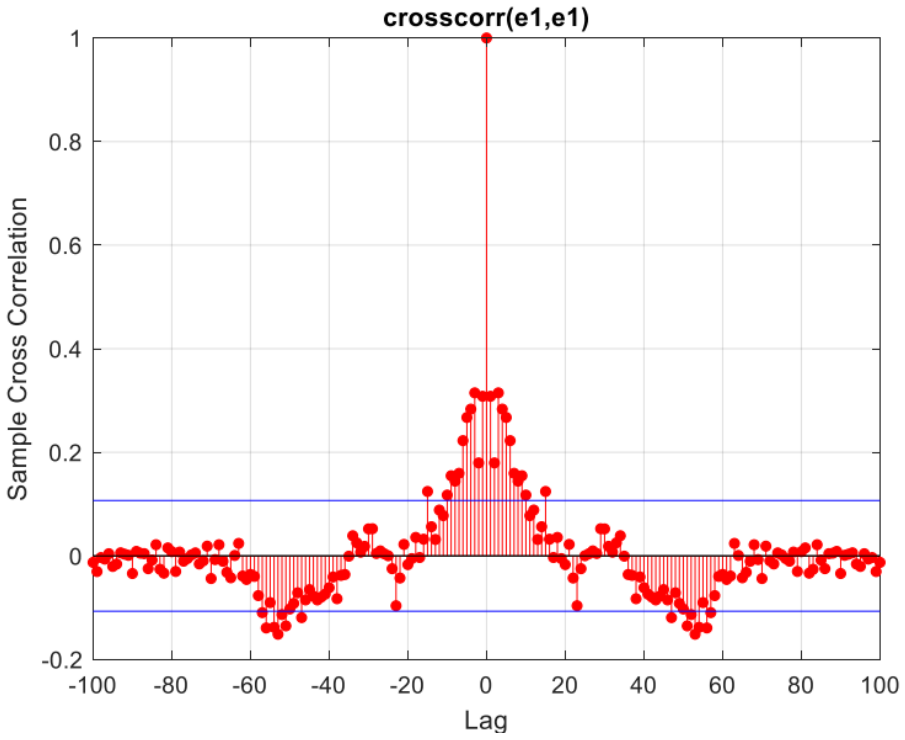


Figure 186 Error cross-correlation

It is observed that the fitting percentage is above 95%, and the error has moved towards whitening. One of the best solutions has been obtained for the first system so far.

1.6) System 2

For identification using the ***Wiener–Hmerstein*** method, the ***nlhw*** command is used in MATLAB. The general structure of this method is as follows:

$sys = nlhw(Train\ Data\ ,\ Orders)$

$Orders$ = [nes(4 ,4) $ones$(4 ,4) $zeros$(4 ,4)]

Through trial and error, it was observed that increasing **na** and **nb** for our system is not desirable, and fitting percentages decrease. Also, if the delay is a zero vector, better estimates are obtained. Among different nonlinearities for this model, incorporating saturation in nonlinear sections provides a better response, and we use it.

According to above:

m = nlhw(z1,Orders, saturation([-1.5 1.5]),
saturation([-3 3]), opt)

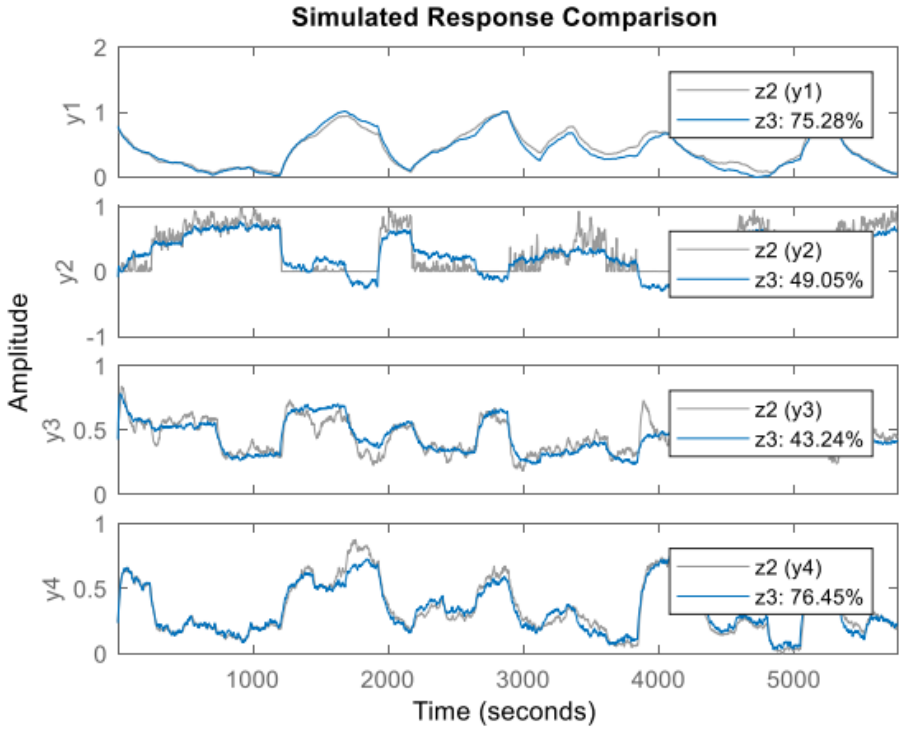


Figure 187 the estimated and actual outputs in the Hammerstein-Wiener model for the (system 2)

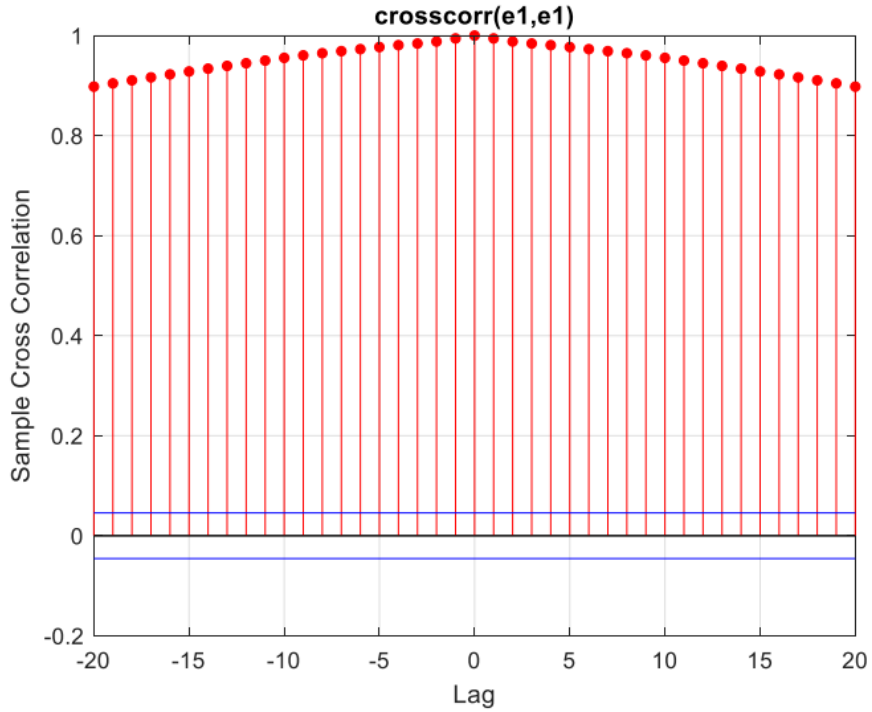


Figure 188 Error cross-correlation first output

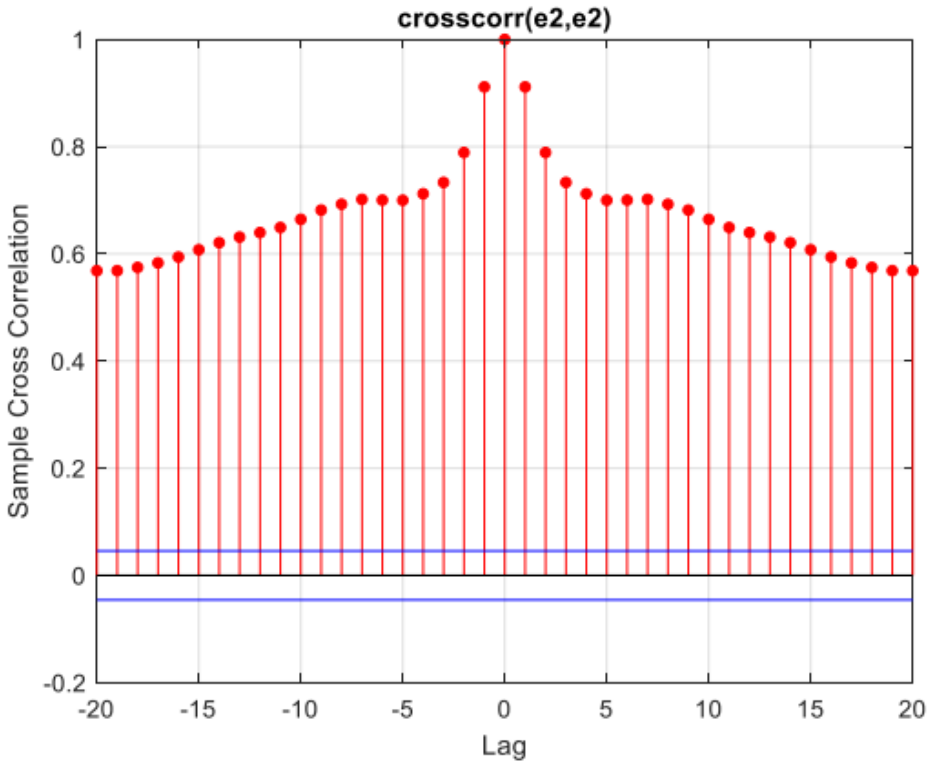


Figure 189 Error cross-correlation second output

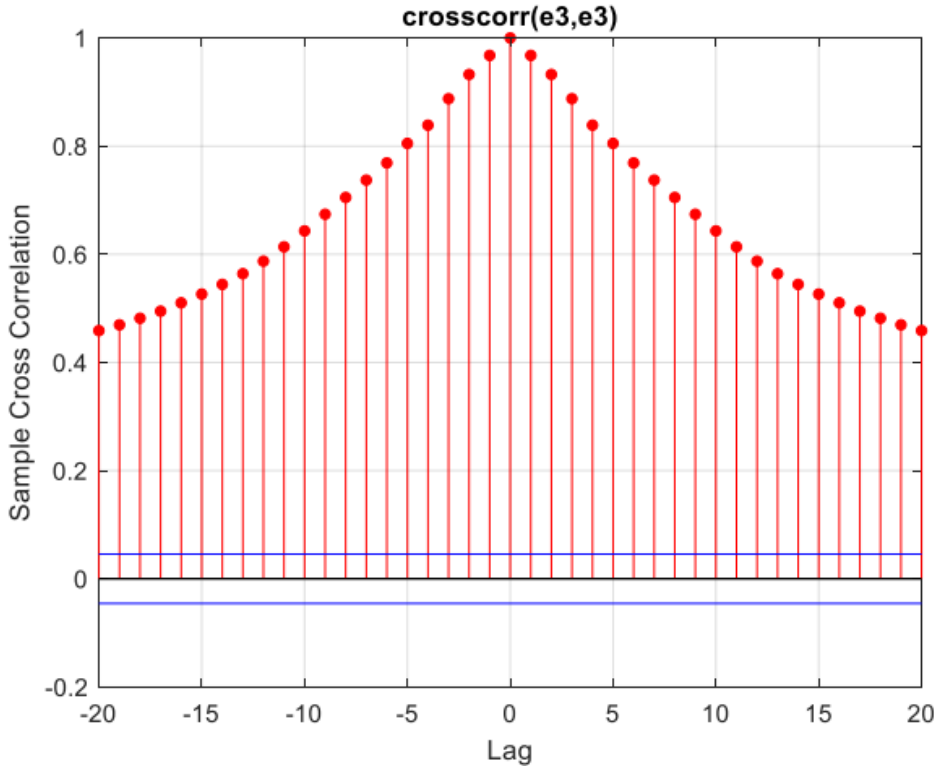


Figure 190 Error cross-correlation third output

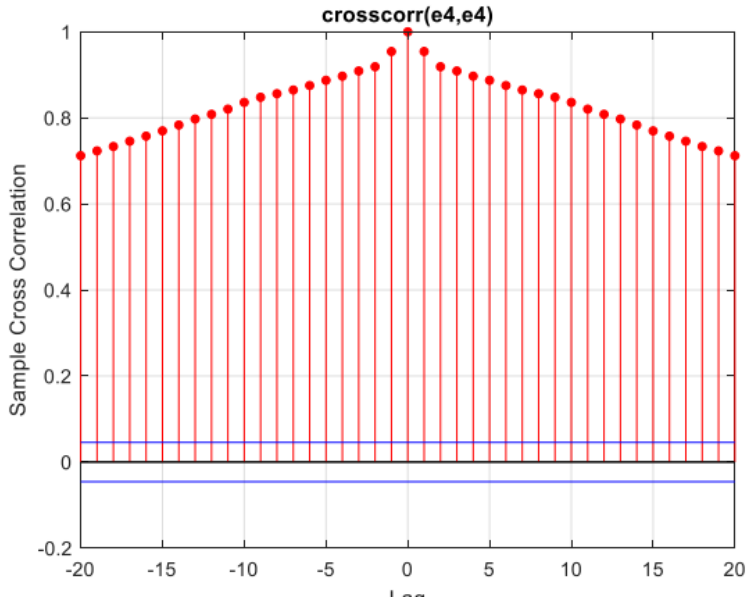


Figure 191 Error cross-correlation fourth output

On dynamic data:

$Orders$ = [nes(4 ,11) $ones$(4 ,11) $zeros$(4 ,11)]

The reason for its selection was mentioned earlier. In this case, the errors are not whitened. We made the following changes, and the errors became whitened.

$Orders$ = [$3 * ones(4\ ,11)\ zeros(4\ ,11)\ zeros(4\ ,11)$]

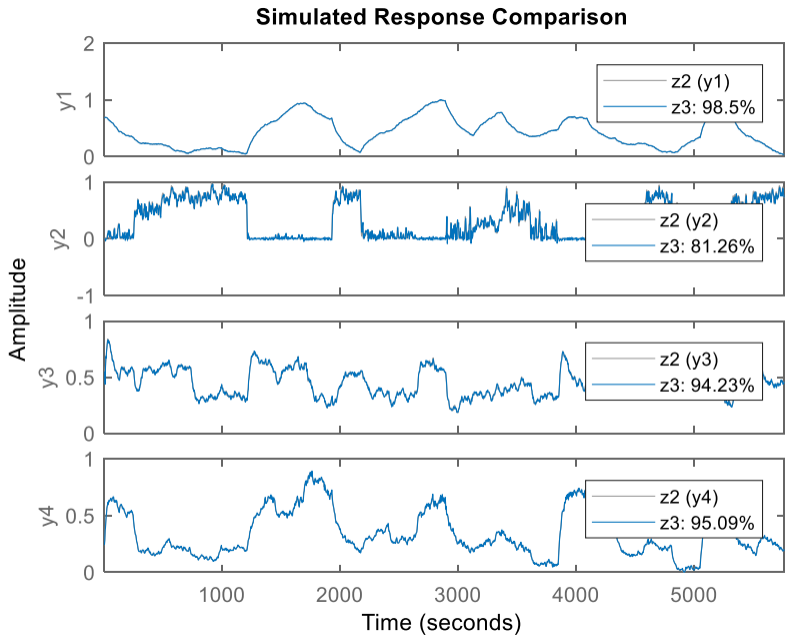


Figure 192 the estimated and actual outputs in the Hammerstein-Wiener model for the (dynamic data) (system 2)

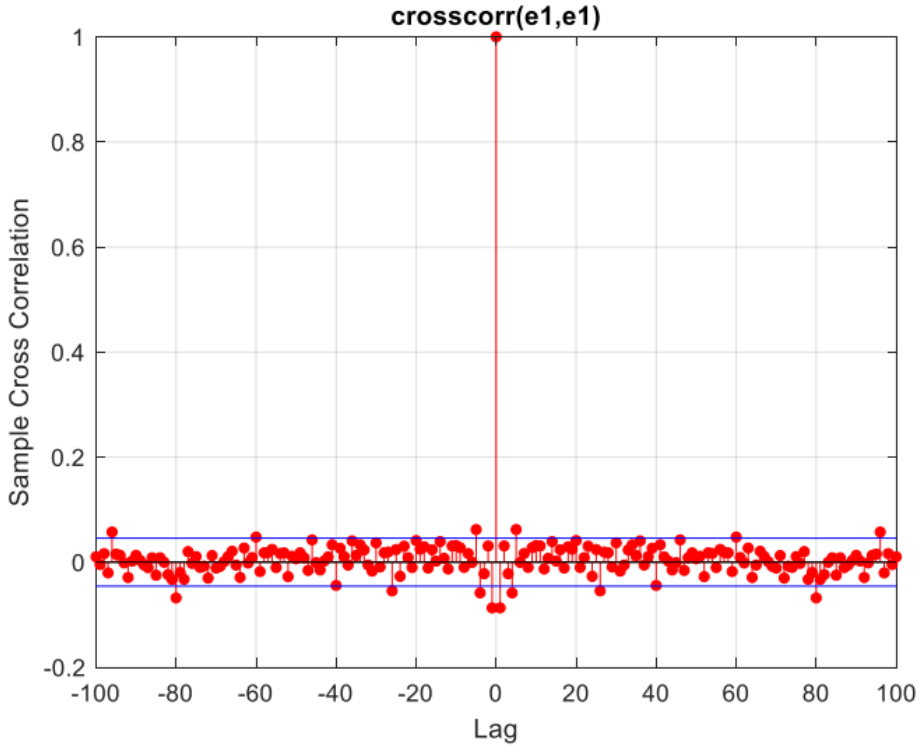


Figure 193 Error cross-correlation first output

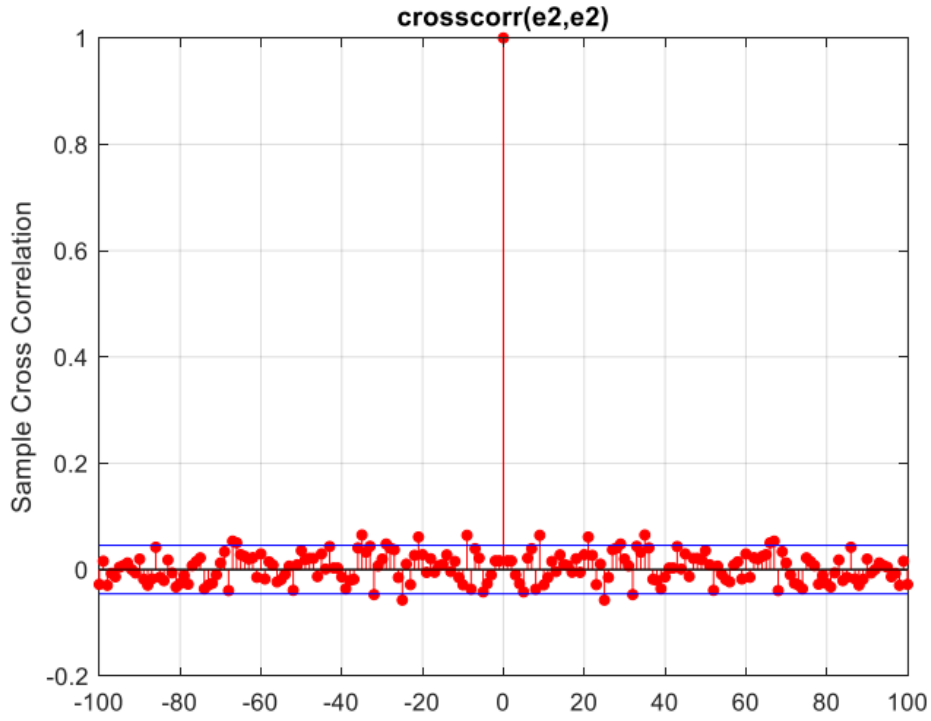


Figure 194 Error cross-correlation second output

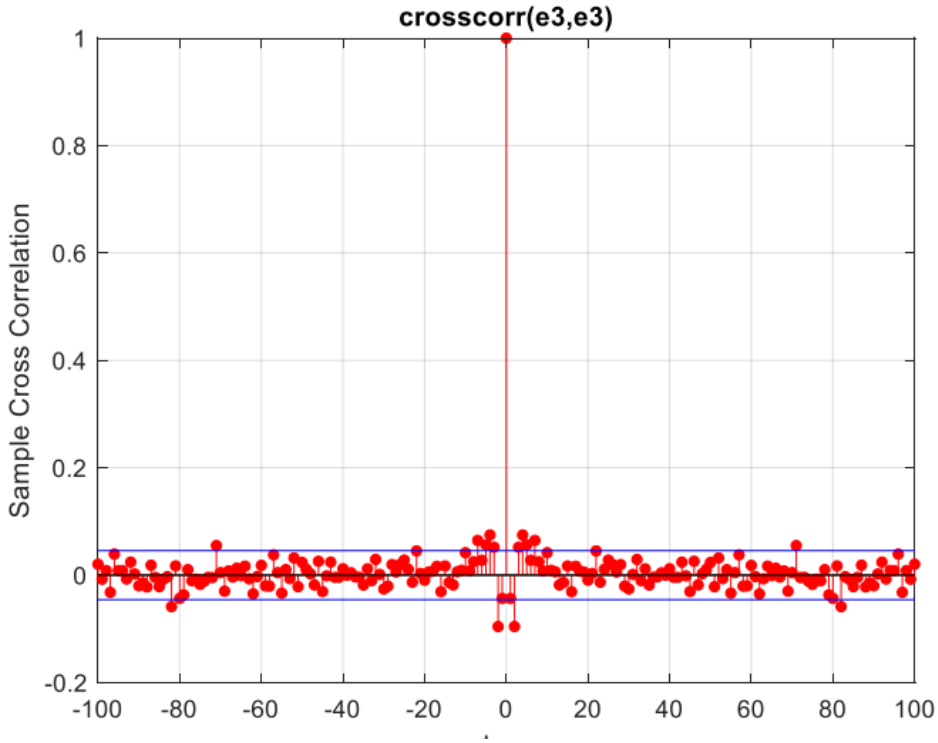


Figure 195 Error cross-correlation third output

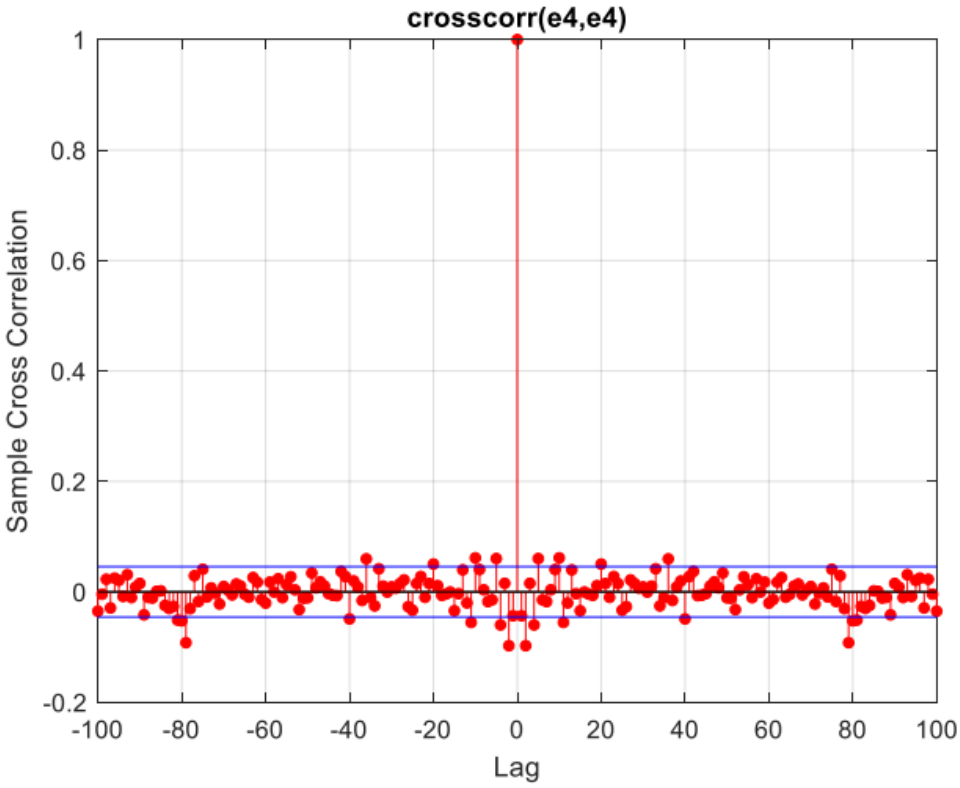


Figure 196 Error cross-correlation fourth output

It can be observed that we have very good fitting percentages, and even the output errors (including the third output) are whitened. The model is suitable.

1.6) System 1

With the same logic as mentioned for the second system, we also determine the parameters for the first system.

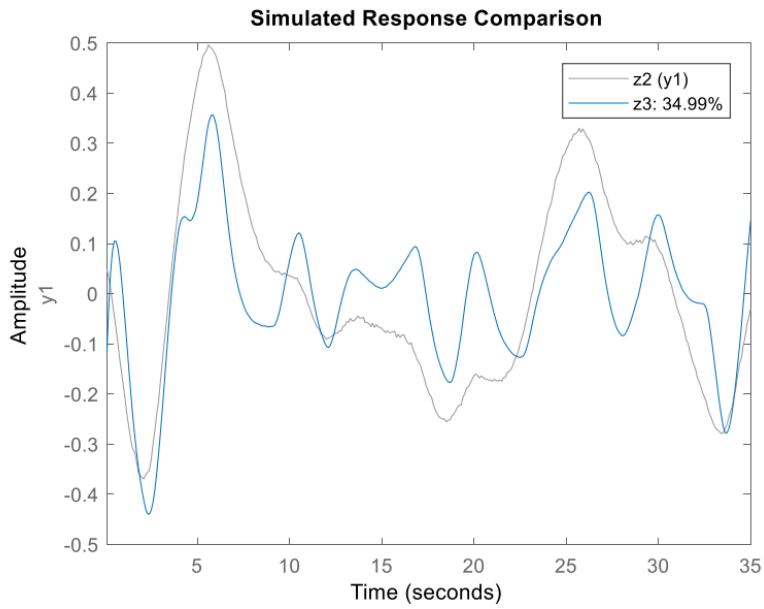


Figure 197 the estimated and actual outputs in the Hammerstein-Wiener model for the (system 1)

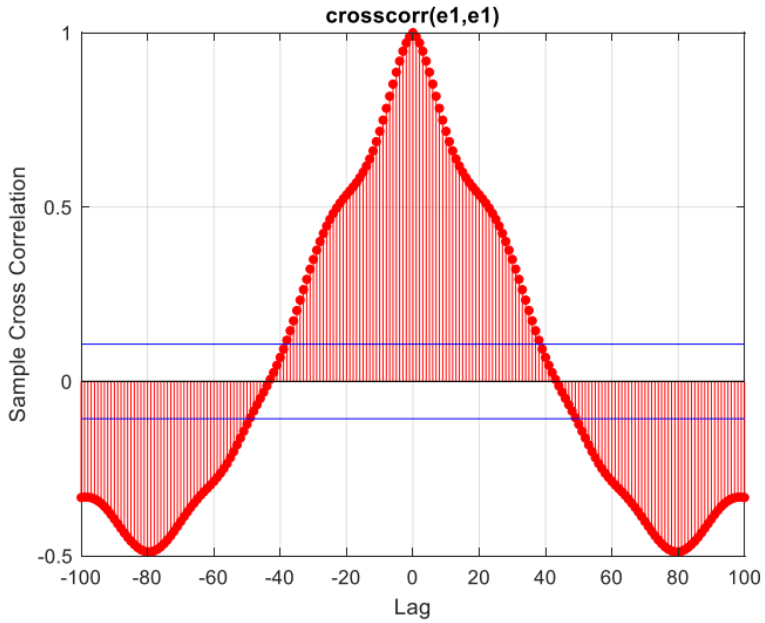


Figure 198 Error cross-correlation

Dynamic data:

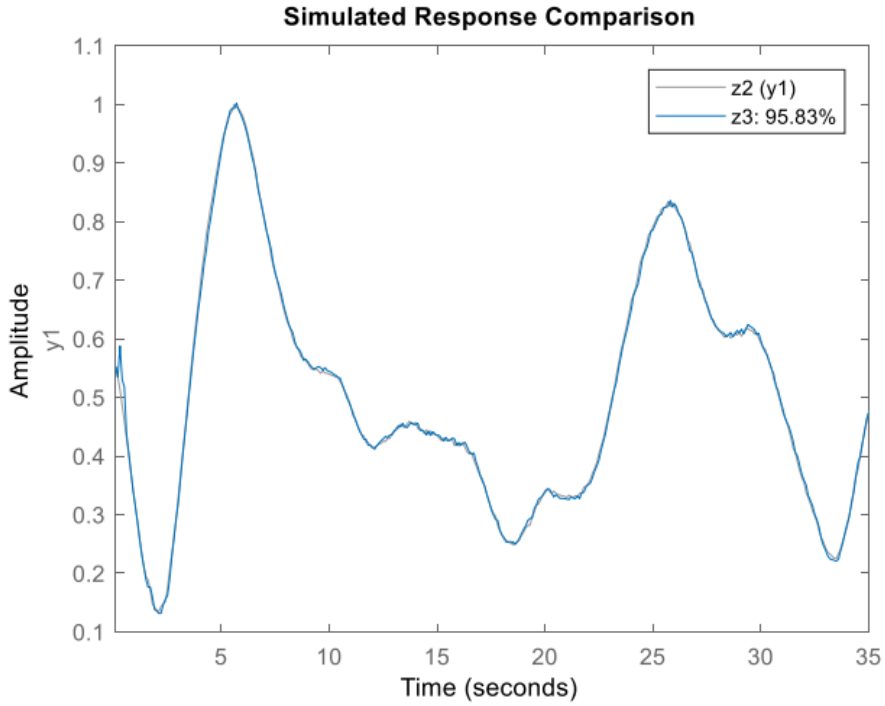


Figure 199 the estimated and actual outputs in the Hammerstein-Wiener model for dynamic data (system 1)

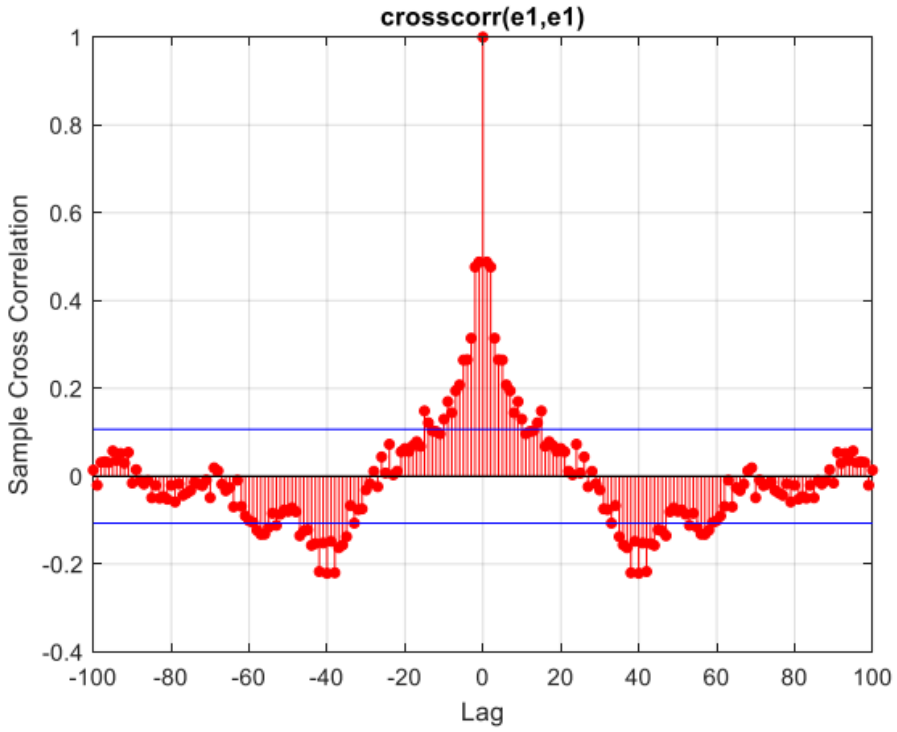


Figure 200 error correlation (system 1)

It can be observed that we have obtained one of the best fitting percentages so far, and the error has also moved towards whitening.

1.7) System 2

In this section, we proceed with the ANFIS method. ANFIS in MATLAB is designed for a single output, and it does not work for multiple outputs. To address this issue, there is a tool called Canfis, but it is not available for use in MATLAB. For the 4th system with 1 output, we obtain a MISO (Multiple Input, Single Output) model. Neuro-fuzzy systems are systems that utilize the learning capability of neural networks and fuzzy logic. The parameters that need to be adjusted for this system include the number of epochs, the number of membership functions for each input, the type of membership function, and the learning method, which is considered in a hybrid form. For this network, only one dynamic is applied as input and output. Gaussian membership functions are used. We have taken the default options for the epoch and utilized a grid partition for the second system.

On the entire data

First option

```
opt2 = genfisOptions('SubtractiveClustering');

fis=genfis([u_trn, y_trn1],y_trn,opt2,'FISType','mamdani');
```

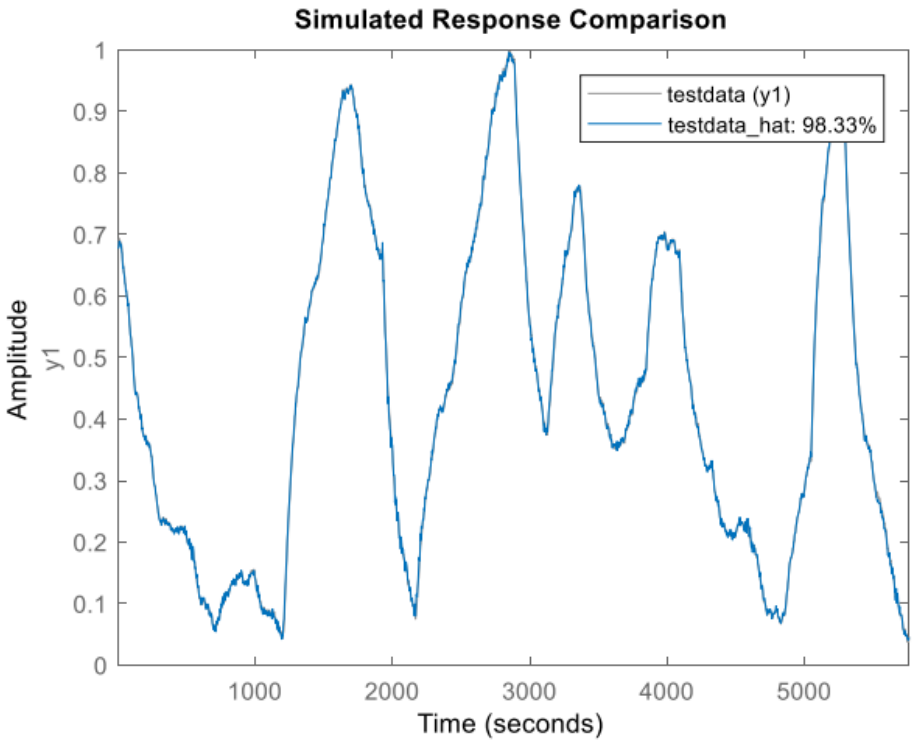


Figure 201 the estimated and actual outputs first output(steamgen)

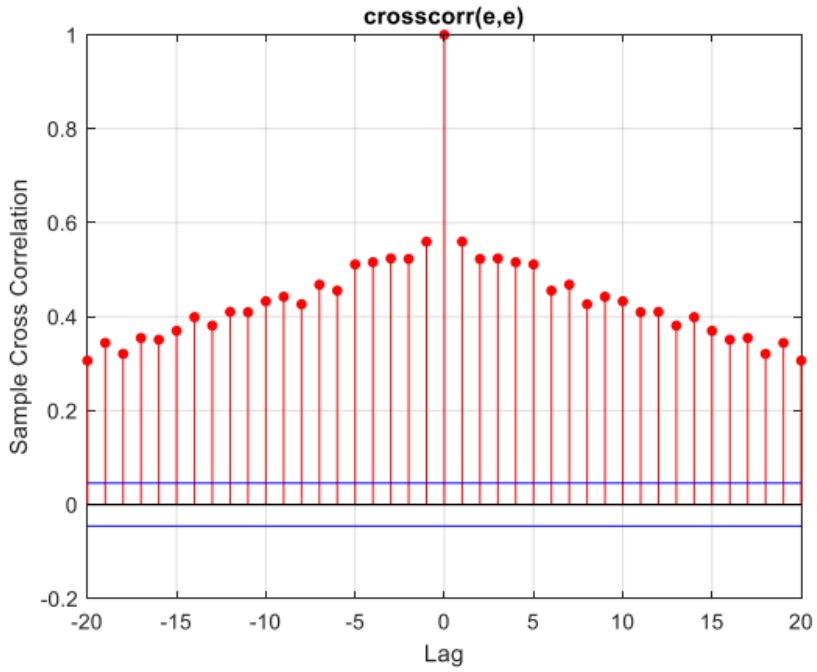


Figure 202 the cross-correlation of first output error (steamgen0

Second Option

The number of clusters has been determined through trial and error.

opt3 = genfisOptions('FCMClustering','NumClusters',10);

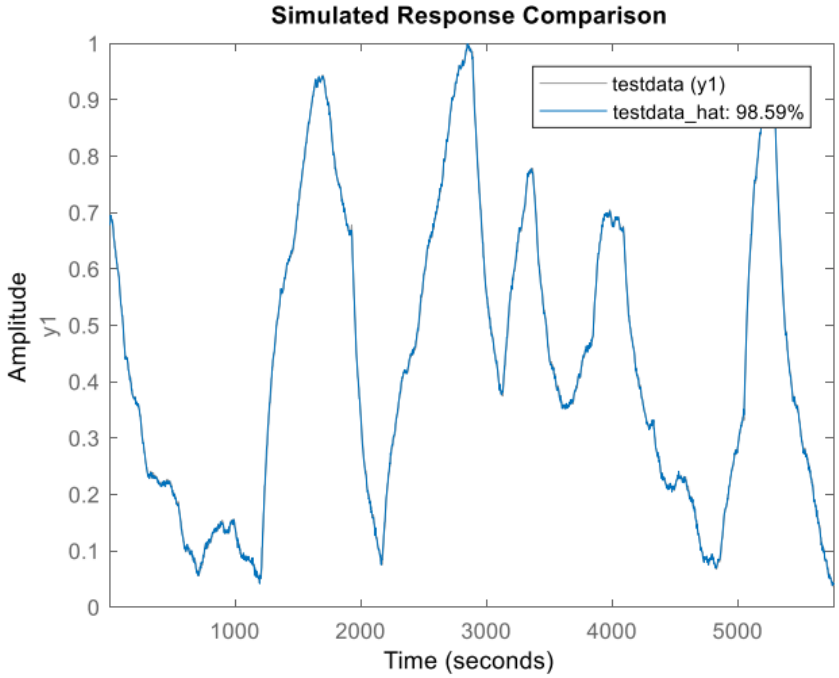


Figure 203 the estimated and actual outputs first output (steamgen)

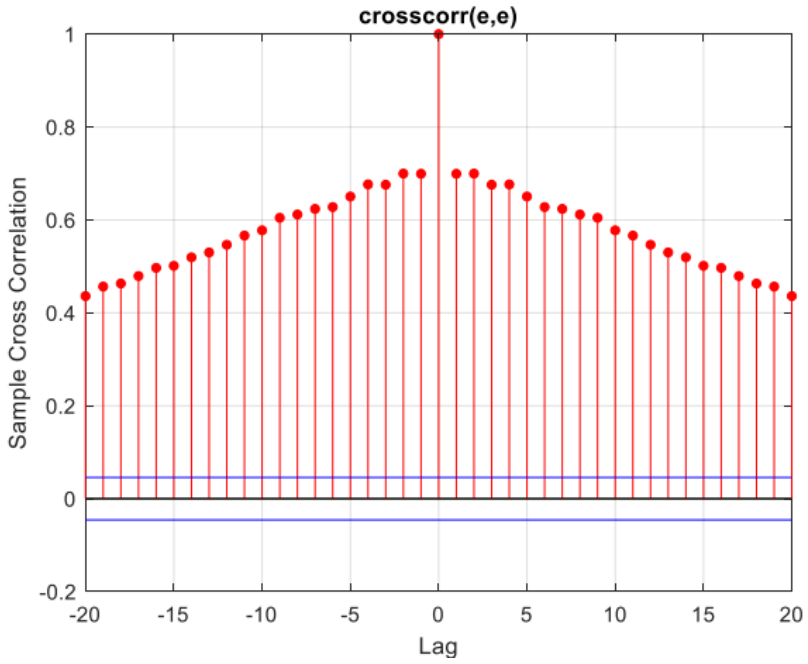


Figure 204 the cross-correlation of first output error (steamgen)

Third option

opt4 = genfisOptions('GridPartition');

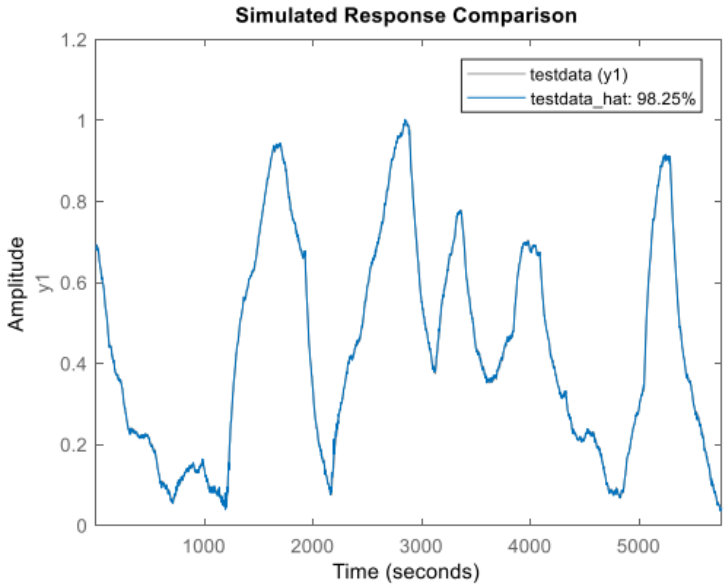


Figure 205 the estimated and actual outputs first output (steamgen)

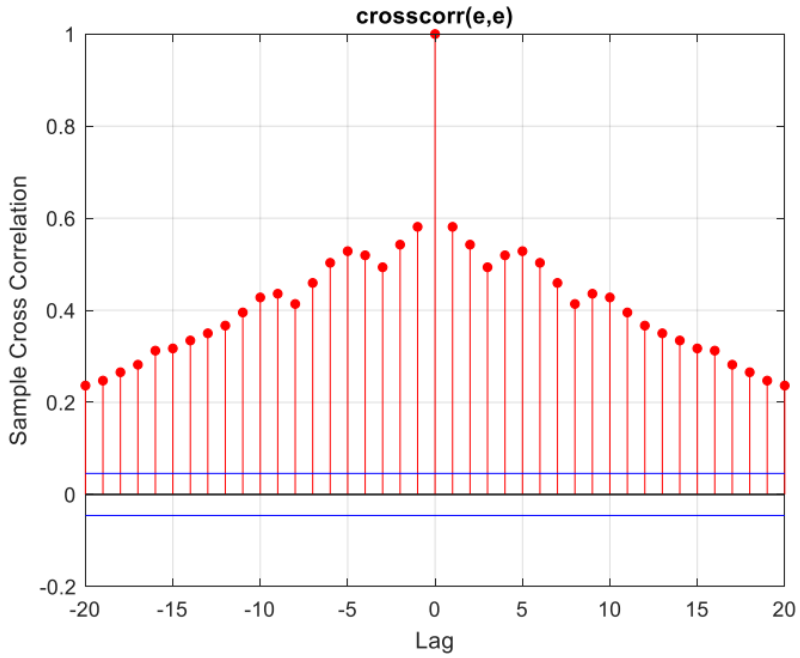


Figure 206 the cross-correlation of first output error (steamgen)

Considering whitening, we use the third option.

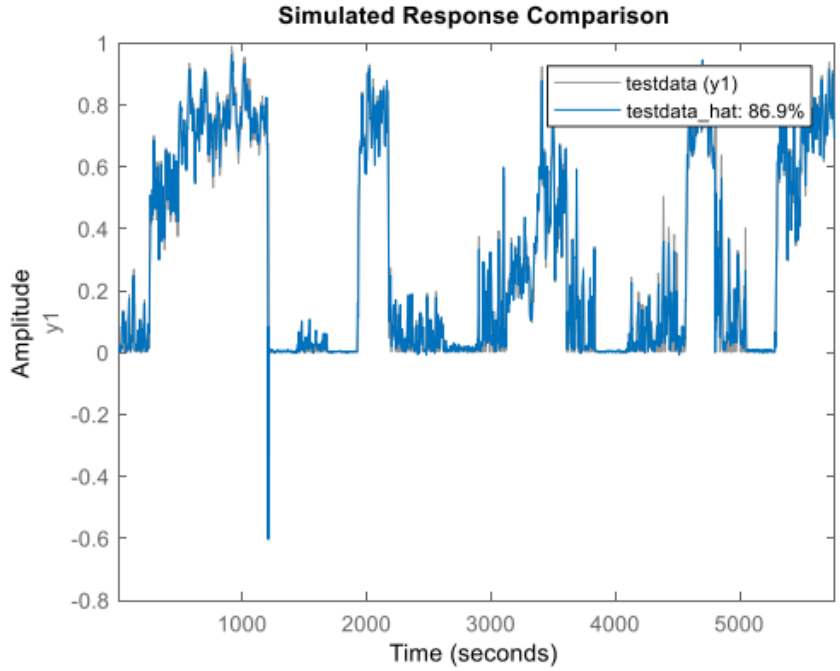


Figure 207 the estimated and actual outputs second output (steamgen)

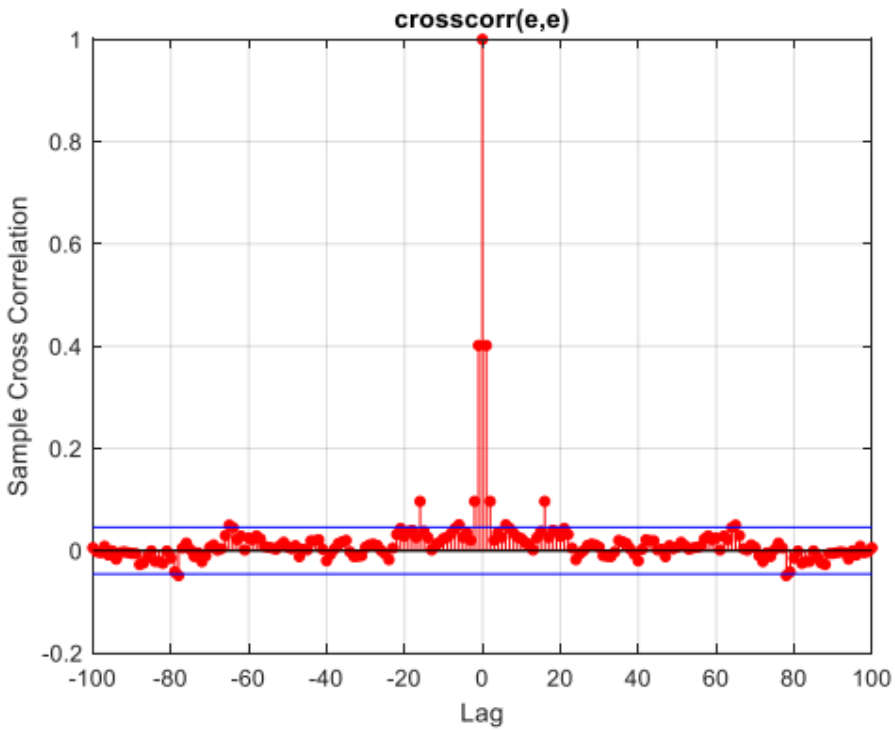


Figure 208 the cross-correlation of second output error (steamgen)

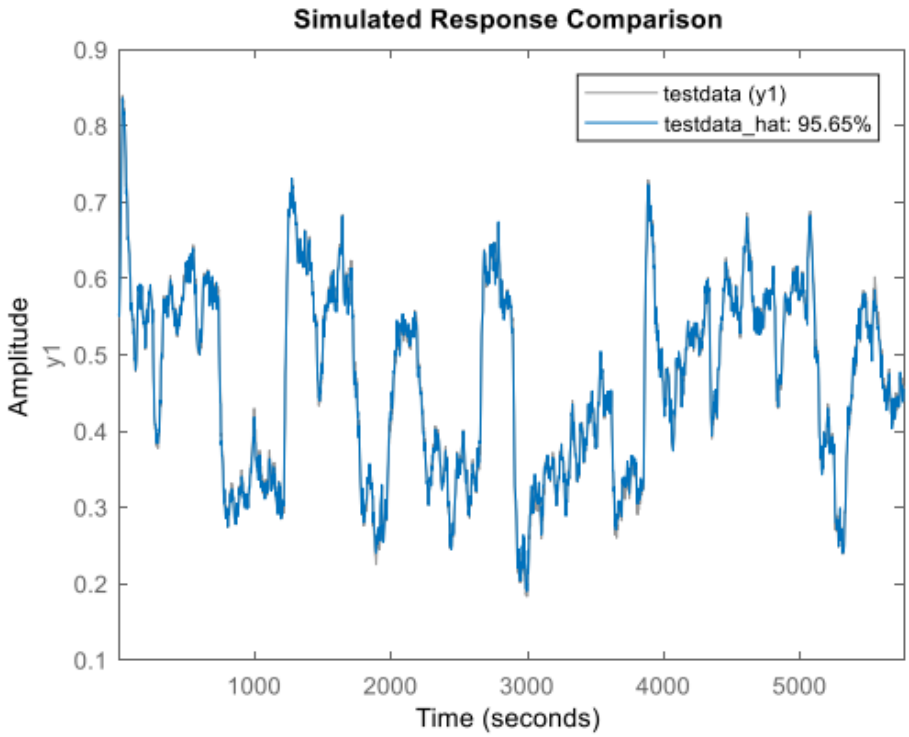


Figure 209 the estimated and actual outputs third output (steamgen)

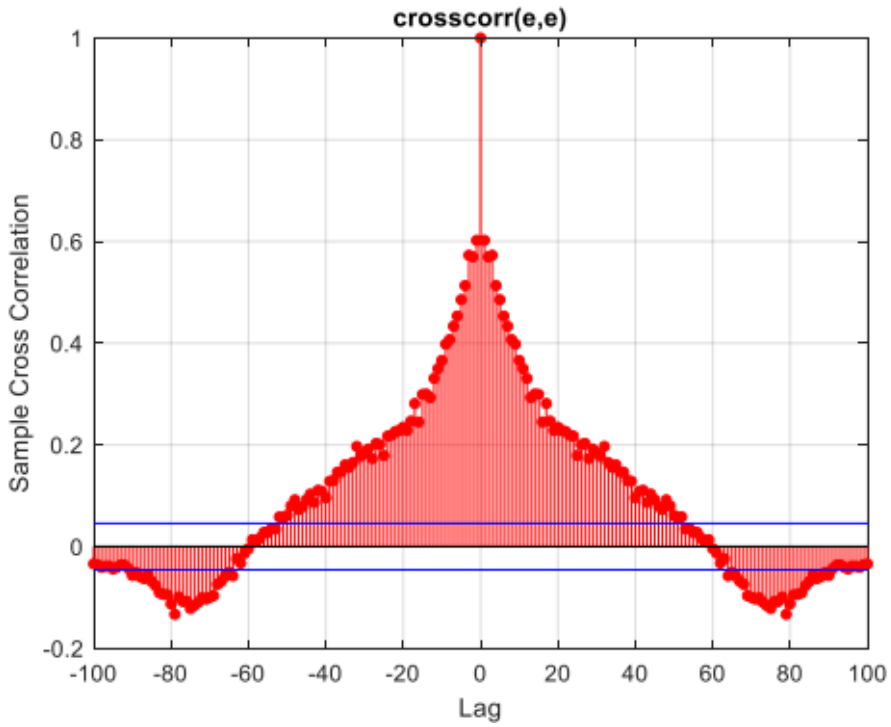


Figure 210 the cross-correlation of third output error (steamgen)

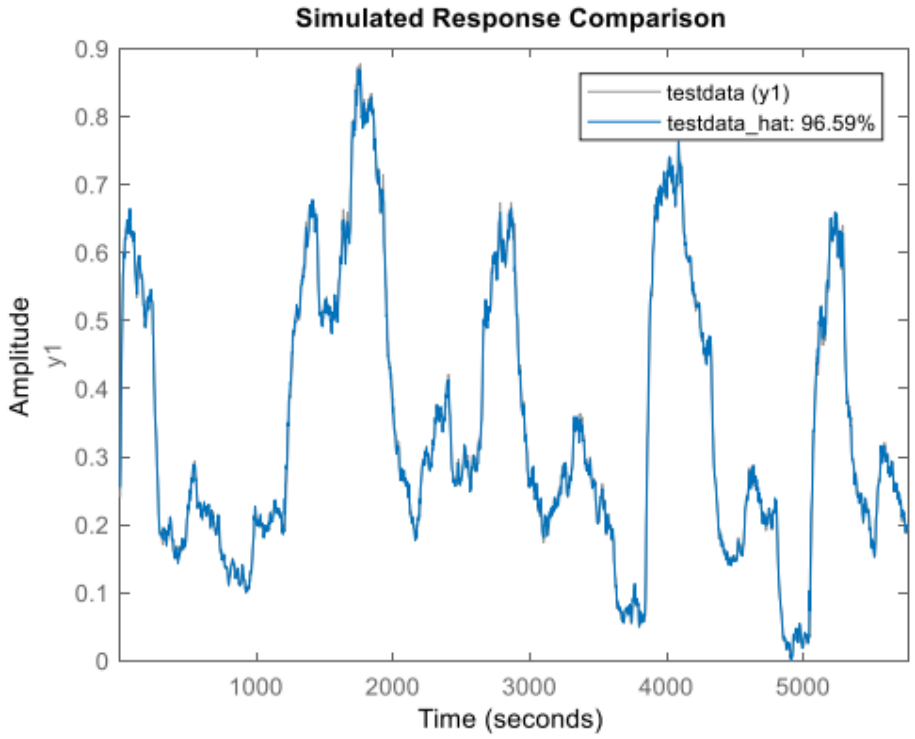


Figure 211 the estimated and actual outputs fourth output (steamgen)

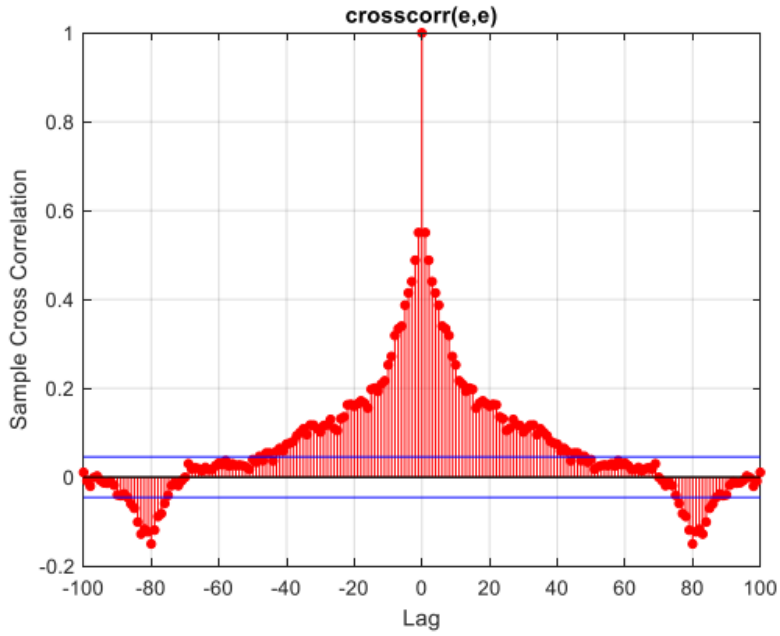


Figure 212 the cross-correlation of fourth output error (steamgen)

It is observed that the fitting percentages are very satisfactory, but whitening of the error has only occurred for the second output.

Now, we test the dynamic data with the structure obtained in this section:

Dynamic Data:

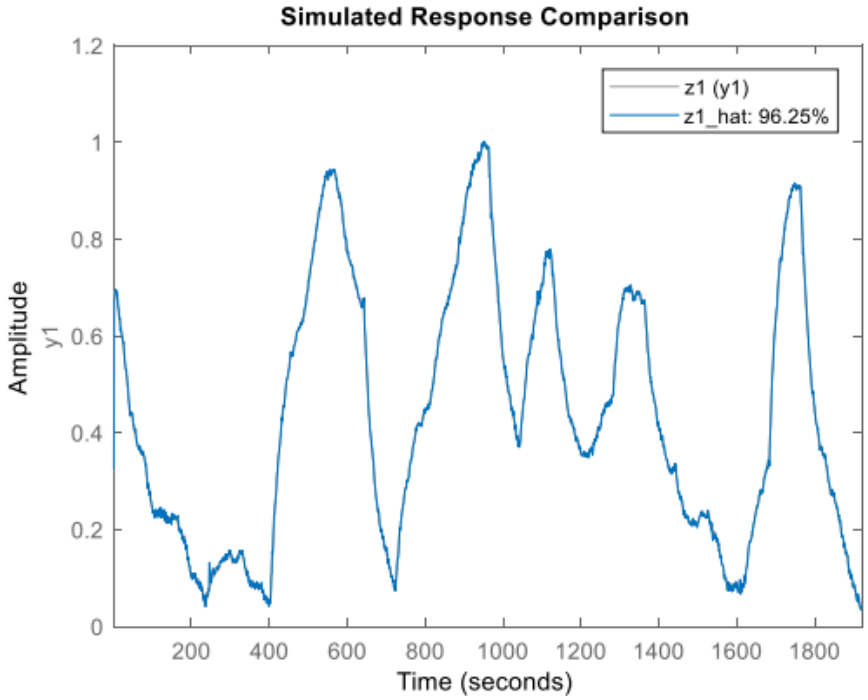


Figure 213 the estimated and actual outputs first output (steamgen)

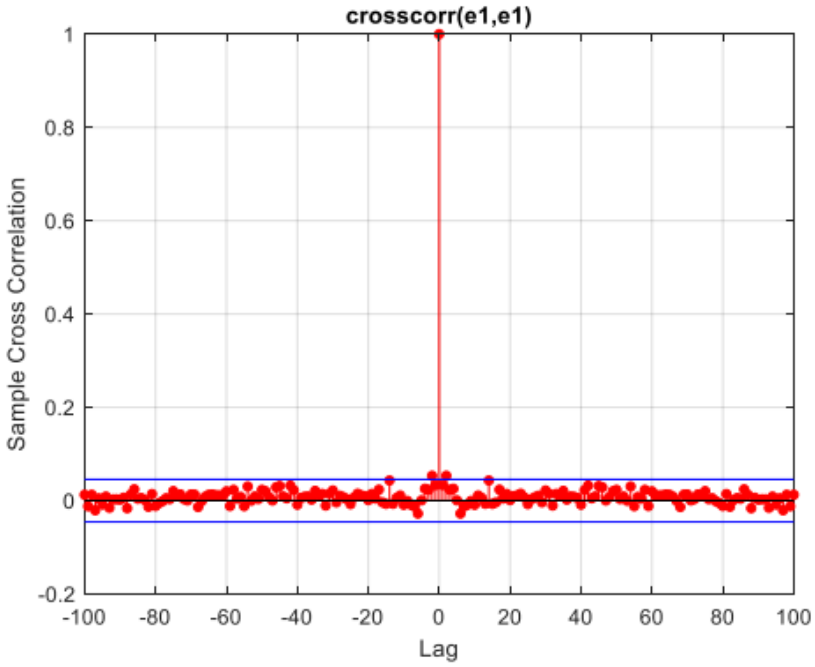


Figure 214 the cross-correlation of first output error (steamgen)

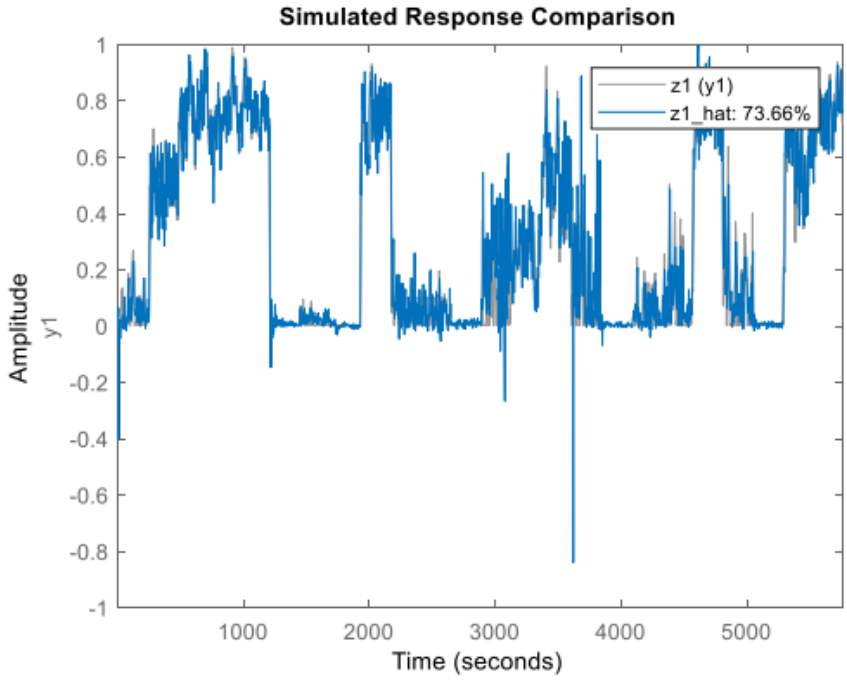


Figure 215 the estimated and actual outputs second output (steamgen)

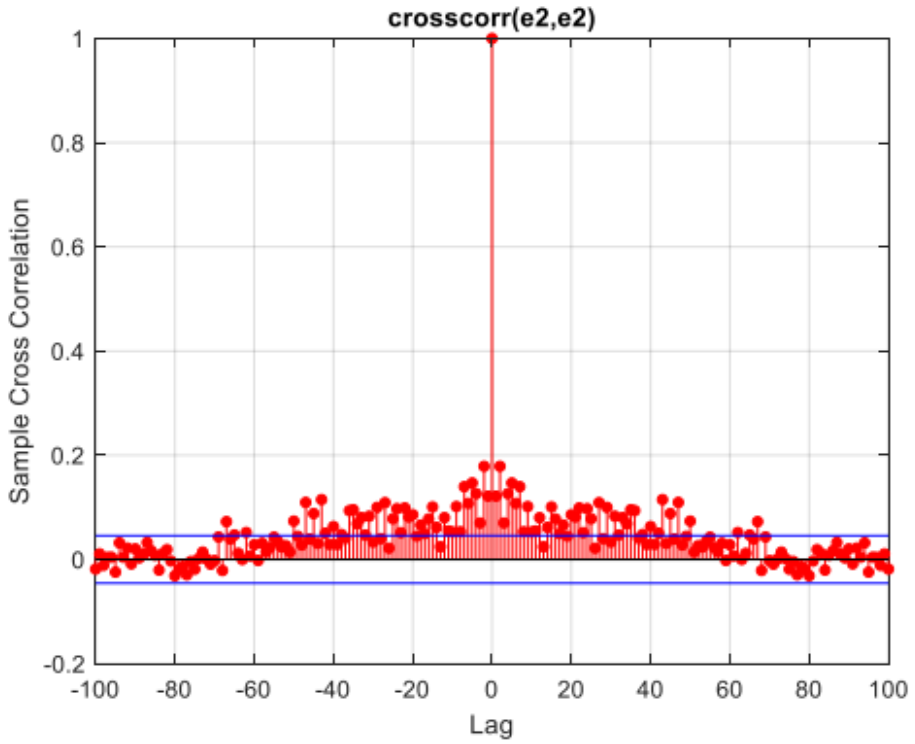


Figure 216 the cross-correlation of second output error (steamgen)

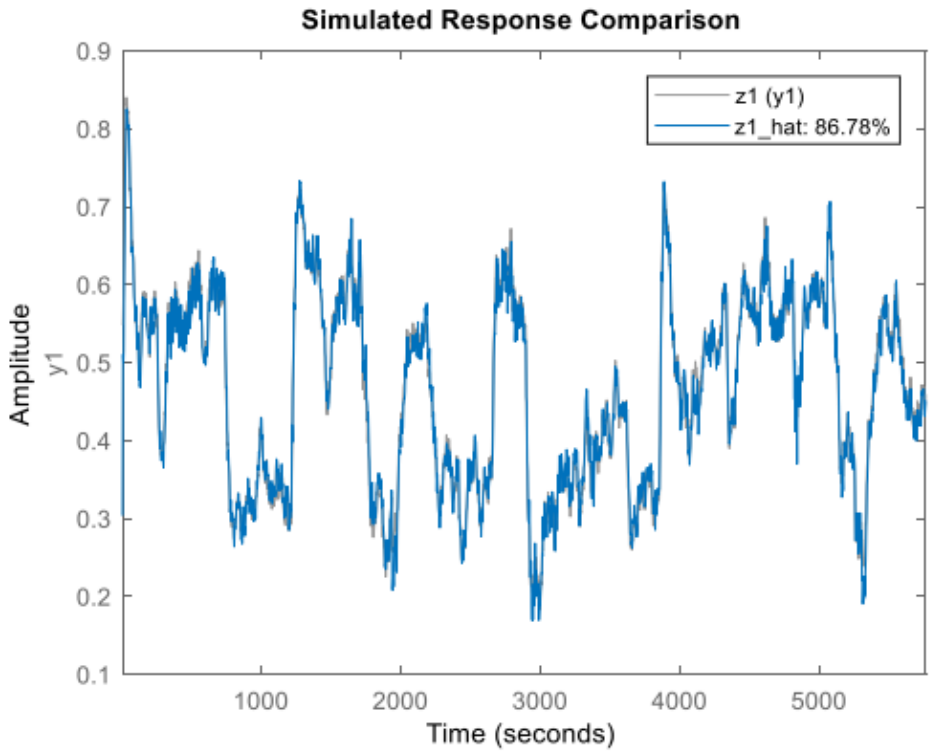


Figure 217 the estimated and actual outputs third output (steamgen)

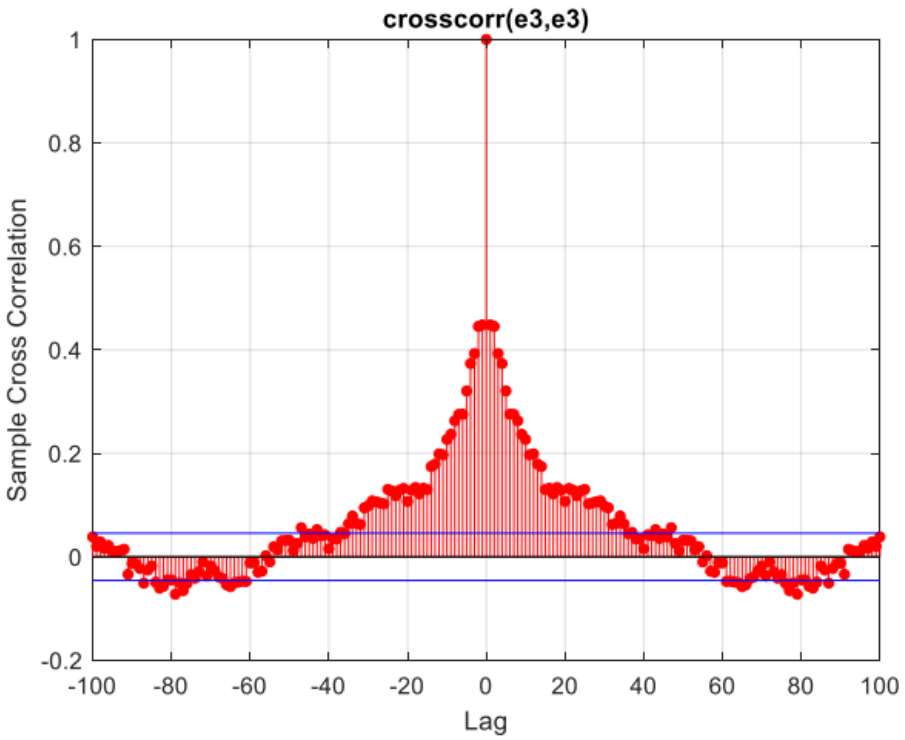


Figure 218 the cross-correlation of third output error (steamgen)

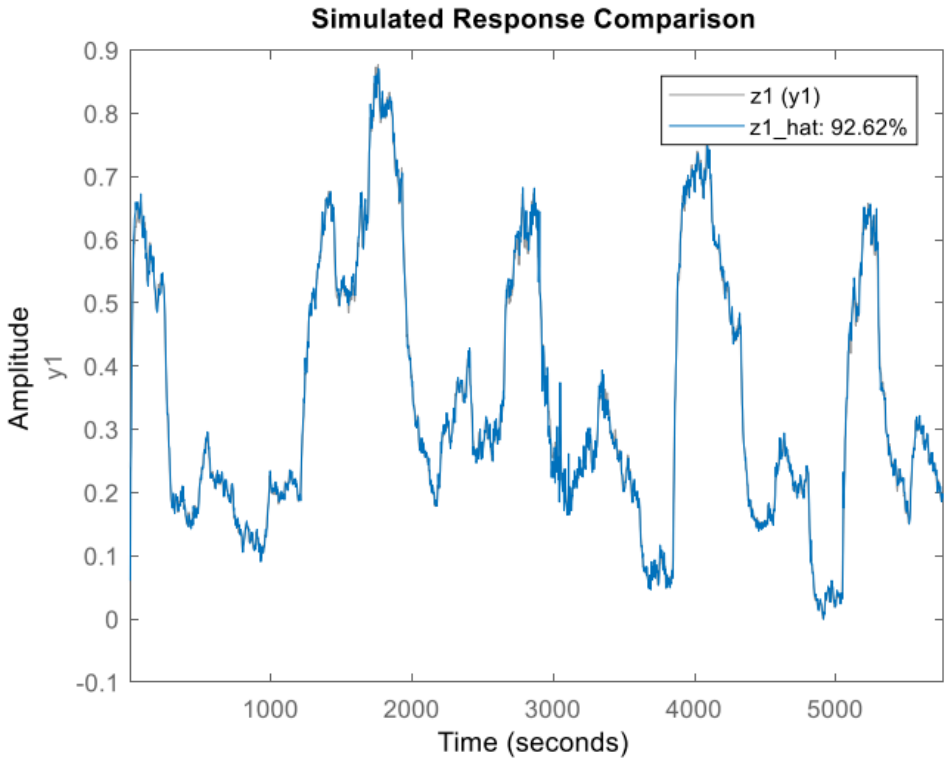


Figure 219 the estimated and actual outputs fourth output (steamgen)

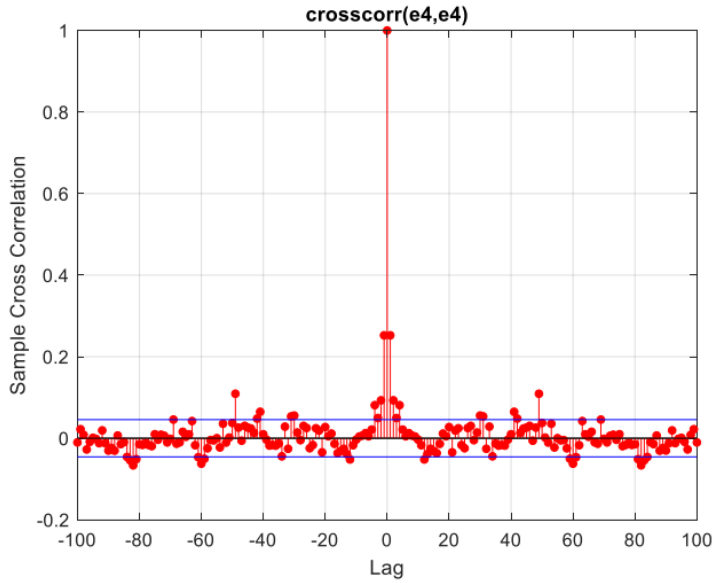


Figure 220 the cross-correlation of fourth output error (steamgen)

The fitting percentages are suitable. However, the fitting percentage for the second output has slightly decreased compared to previous methods, and the error of the third output is somewhat distant from whitening.

1.7) System 1

opt3 = genfisOptions('FCMClustering','NumClusters',3);

Let's use it (the number of clusters has been chosen through trial and error).

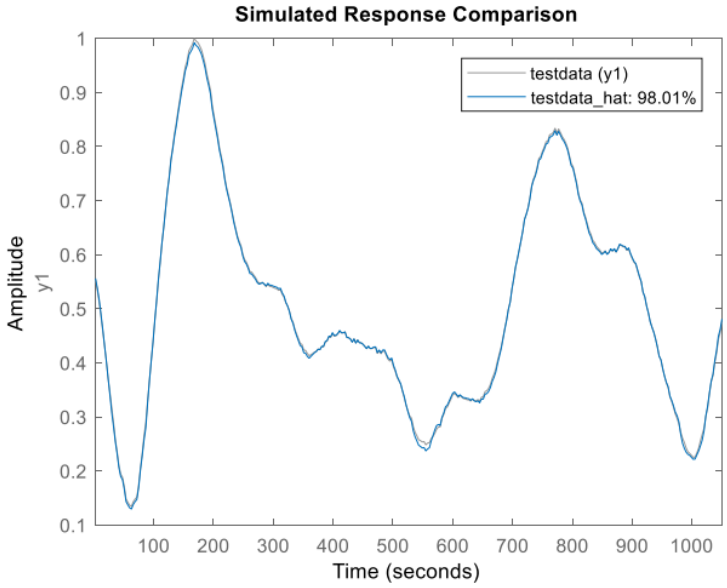


Figure 221 the estimated and actual outputs (steamgen)

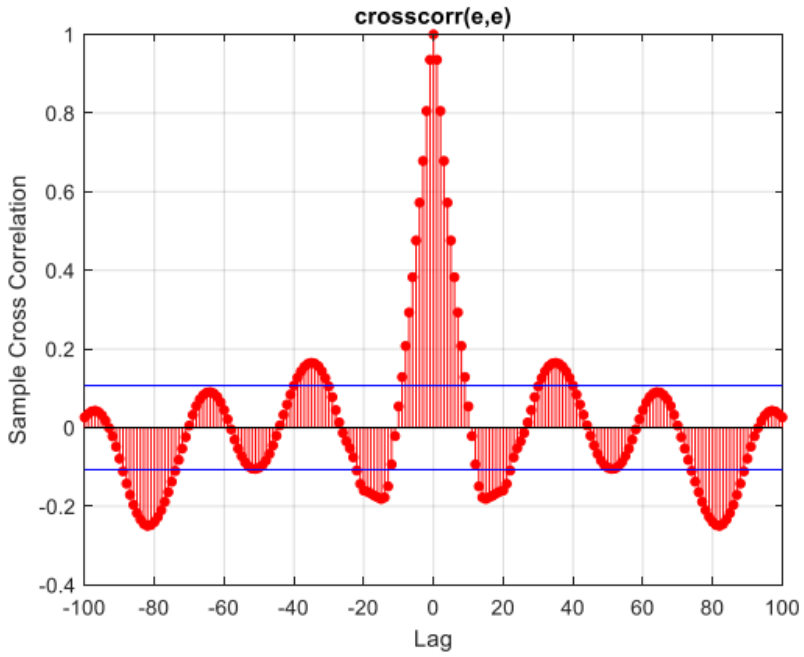


Figure 222 the cross-correlation of error (steamgen)

We increase the number of clusters by one.

opt3 = genfisOptions('FCMClustering','NumClusters',4);

For an epoch of 10, one of the best whitening of errors occurs.

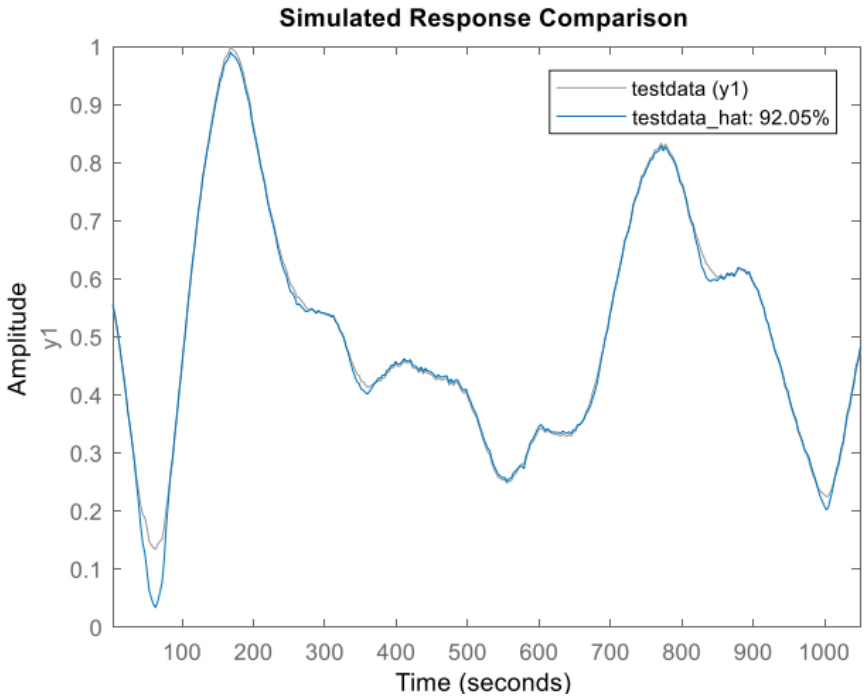


Figure 223 the estimated and actual outputs (steamgen)

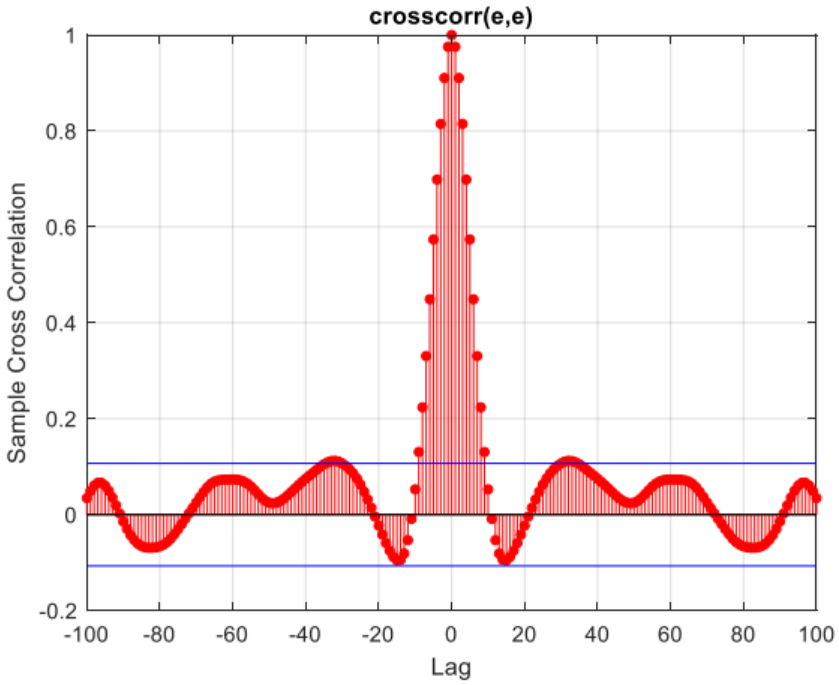


Figure 224 the cross-correlation of error (steamgen)

A lower fitting percentage but a whiter error is observed.

With the option:

```
opt4 = genfisOptions('GridPartition');
```

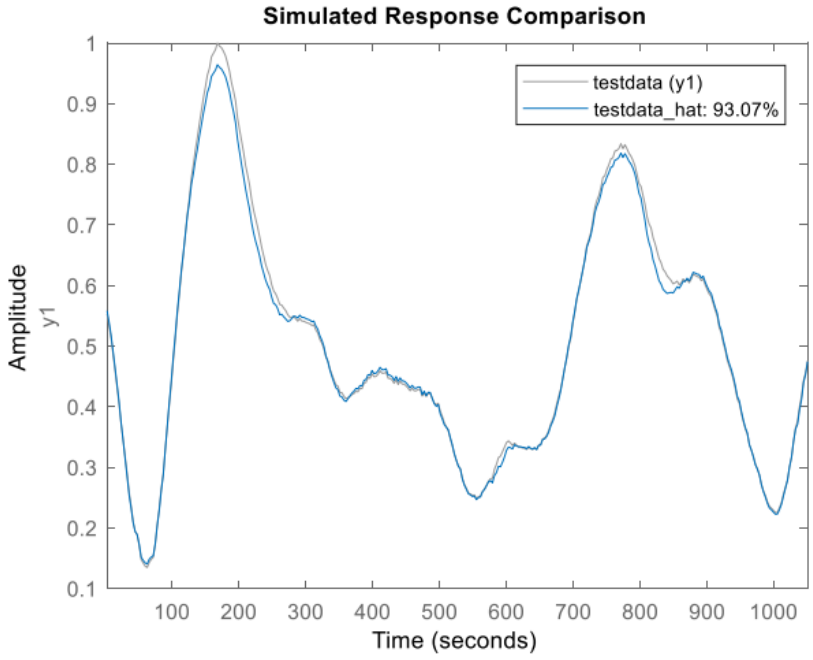


Figure 225 the estimated and actual outputs (steamgen)

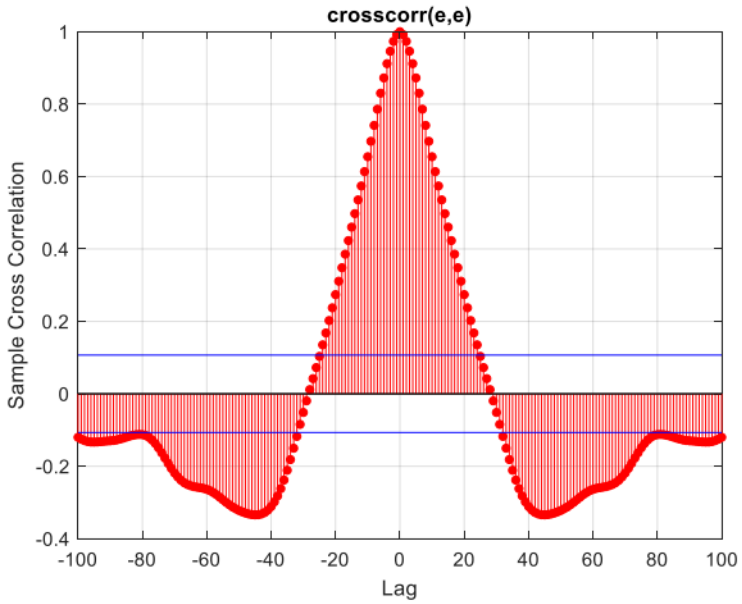


Figure 226 the cross-correlation of error (steamgen)

For Dynamic data:

With trial and error:

opt3 = genfisOptions('FCMClustering','NumClusters',2);

two clusters moves us closer to error whitening.

With 200 epochs, we have:

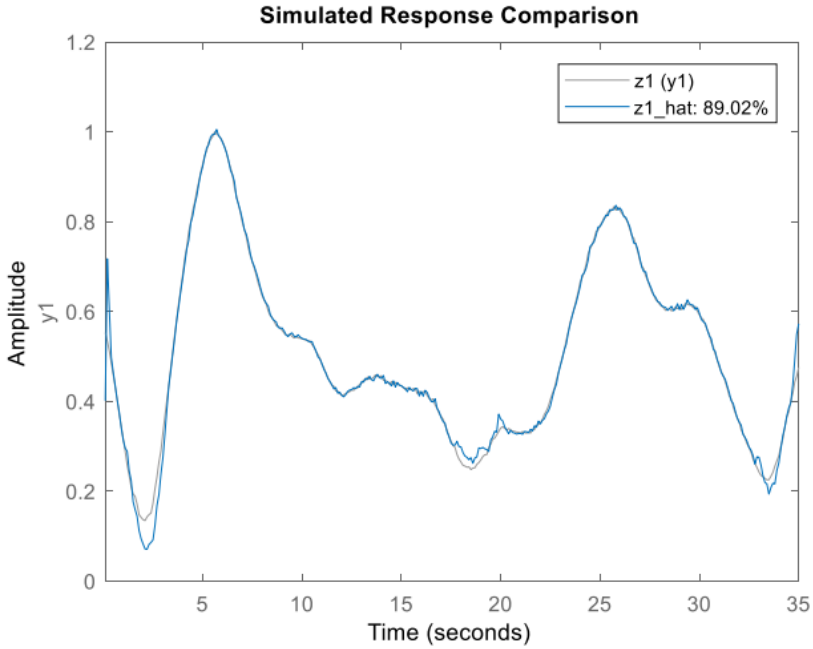


Figure 227 the estimated and actual outputs (steamgen)

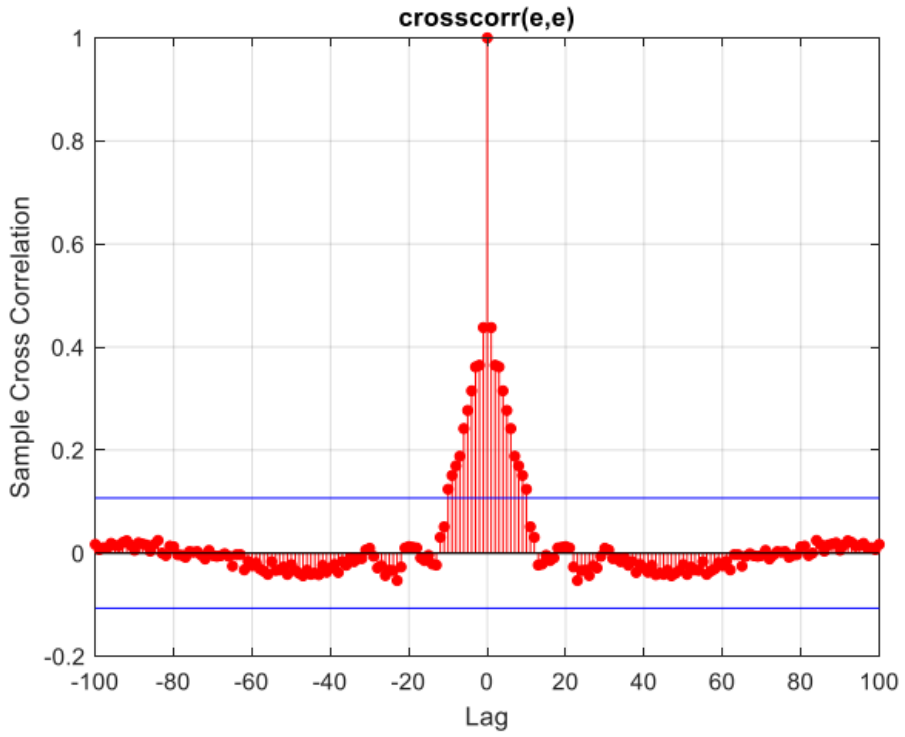


Figure 228 the cross-correlation of error (steamgen)

The fitting percentage is satisfactory, and the error has also whitened.

General Comparison of Models:

In each section, relevant explanations have been provided. Here, we provide a general comparison. In each code, we have applied **rand('state',0)** to compare the results using the specified hyperparameters.

For the **system 1**, it was observed that the fitting percentage and whitening of the error were nearly the same for regression and F.S methods. The fitting percentage with genetic algorithm regressors was also suitable, but the error was slightly distant from whitening, indicating that the dynamics are not optimal.

The NARX method did not provide a satisfactory answer when applied on the entire dataset in the simulator mode, but with dynamic data, the error whitened, and the fitting percentage was suitable. However, the fitting percentage for the second and third outputs decreased slightly compared to MLP. In the prediction horizon mode, better results were obtained in terms of fit and whitening.

In the case of NOE, if applied on the entire dataset, we get suitable fitting percentages, but the errors have deviated from whitening for different outputs. Now, if applied on dynamic data, the errors have become whiter, and the fitting percentage is close to the previous state and similar to other suitable algorithms.

Overall, whitening of the error has been better on NARX.

MLP, RBF, and NRBF networks show that they do not provide a suitable response on the entire dataset, but with dynamic data, we observe that the fitting percentages are similar to MLP and others. However, whitening of the error in NRBF is somewhat undesirable.

The RNN network did not provide a suitable answer on the entire dataset but showed suitable fitting percentages and whitening of the error with dynamic data.

- It is worth mentioning that in most of the above algorithms, for the third output, there is a slight deviation in error whitening.

In the Hammerstein-Wiener model, we do not have a suitable response on the entire dataset, but on dynamic data, the fitting percentages are especially suitable for the third output, and whitening of the error has also occurred (even the error of the third output has whitened).

In the ANFIS network, whether on the entire dataset or on dynamic data, the fitting percentage is suitable, and high fitting percentages are obtained. However, whitening of the error is not observed on the entire dataset, but on dynamic data, whitening of the error has occurred for the outputs. Although the error of the third output has deviated slightly from whitening.

System 1:

It is observed that on dynamics obtained from the linear method, whitening and better fitting percentages are achieved compared to the F.S and G.A methods. However, with the G.A method, whitening has progressed, but the first dynamics have a slight distance from whitening.

In the NARX method, in the simulation mode, a desirable response was not obtained on the entire dataset, but with dynamic data, a suitable fitting percentage and whitening of the output error occurred. In the prediction mode, a much higher fitting percentage and whitening of the error have occurred (both on the entire dataset and dynamic data), and the error has also whitened. With an increase in the prediction horizon, the fitting percentage decreased slightly, but it still remained higher than the simulation fit percentage. However, the error deviated slightly from whitening. The fit percentage of the prediction mode for this method was higher than MLP.

The NOE method also had high fitting percentages on both static and dynamic data (similar to NARX in prediction mode). Whitening of the error did not occur on the entire dataset, and generally, whitening was better in NARX on dynamic data.

MLP, RBF, and NRBF networks did not provide satisfactory responses on the entire dataset, but when using dynamic data, NRBF showed a higher fitting percentage close to MLP, while the RBF fitting percentage was lower than MLP and NRBF. Additionally, the output error of the NRBF method was whiter than RBF.

In the RNN method, considering dynamics, a high fitting percentage was achieved, and the error tended towards whitening. This fitting percentage was close to the MLP method.

In the Hammerstein-Wiener model, a suitable response was not obtained on the entire dataset, but on dynamic data, especially for the third output, the fitting percentages were suitable, and whitening of the output error had also occurred (even for the third output).

In the ANFIS network, whether on the entire dataset or dynamic data, the fitting percentage is suitable, and high fitting percentages are obtained. However, whitening of the error is not observed on the entire dataset, but on dynamic data, whitening of the error has occurred for the outputs. Although the error of the third output has deviated slightly from whitening.

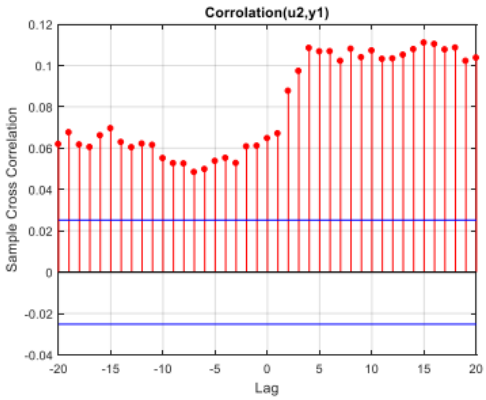
Question 2

Industrial Evaporator

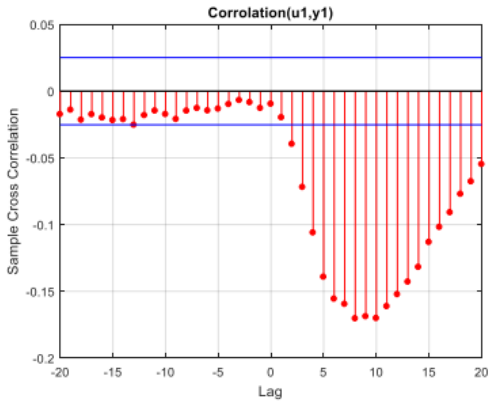
Step 1)

Testing the correlation between system data.

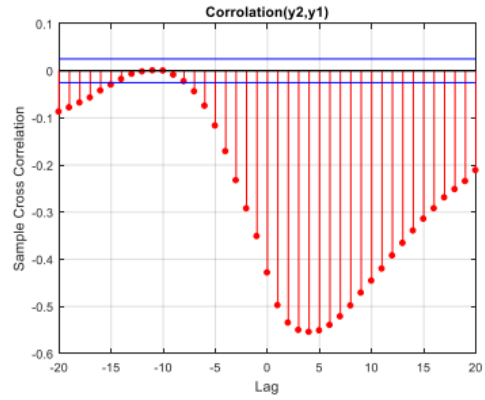
Correlation is valid for linear systems. It is not a completely accurate method for nonlinear systems, but we use this approach to gain an approximate insight into the system's dynamics.



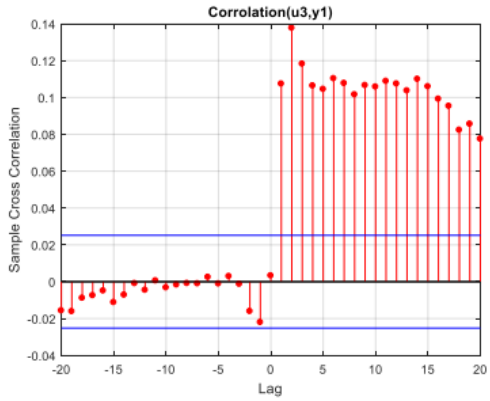
Dynamics: 4-8-10-15-18-20



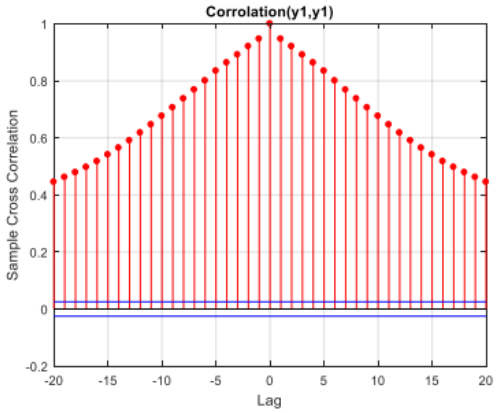
Dynamics: 8-10



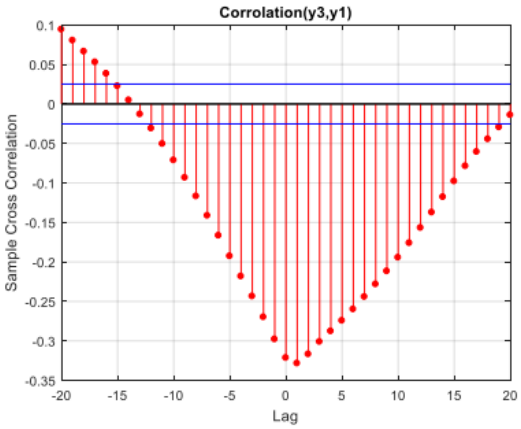
Dynamics: 4



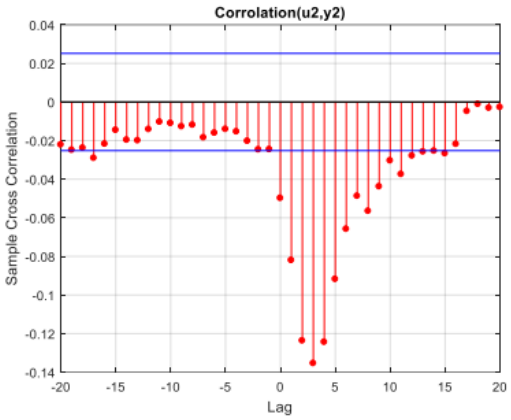
Dynamics: 2-8-10



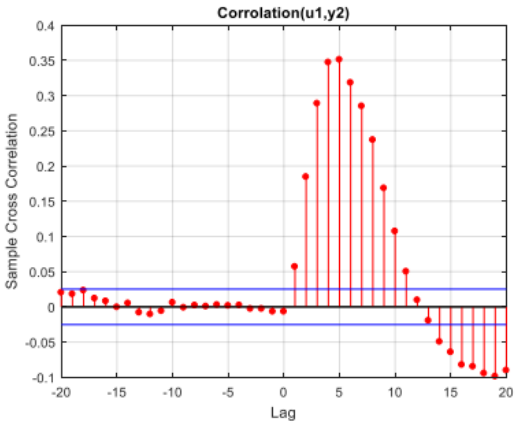
Dynamics: 2-1



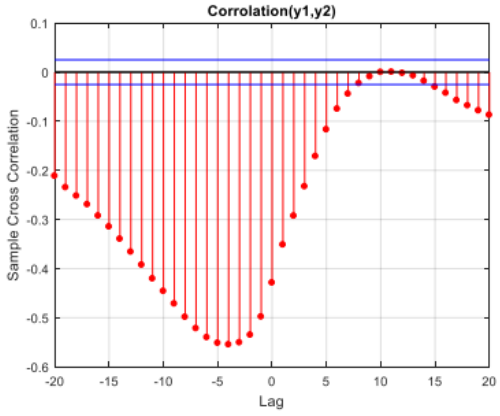
Dynamics: 1



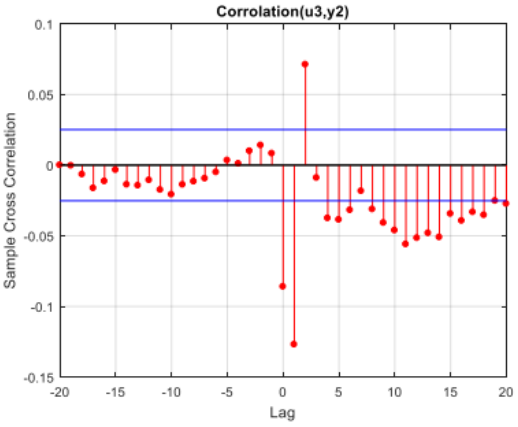
Dynamics: 3-8-11-15



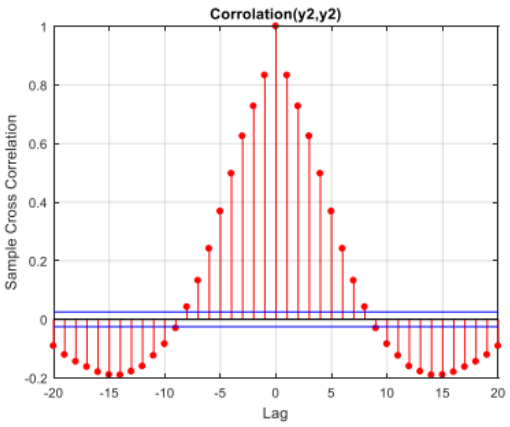
Dynamics: 5-19



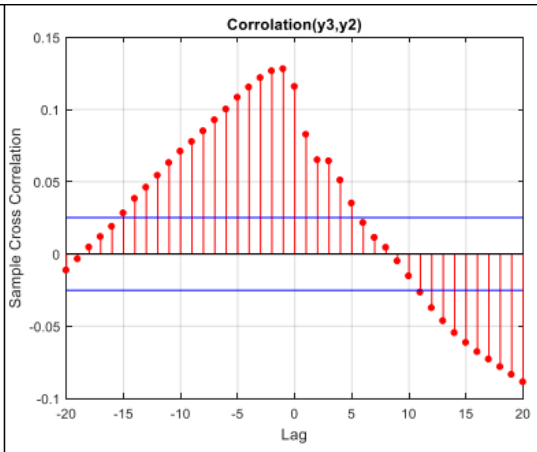
Dynamics: 20



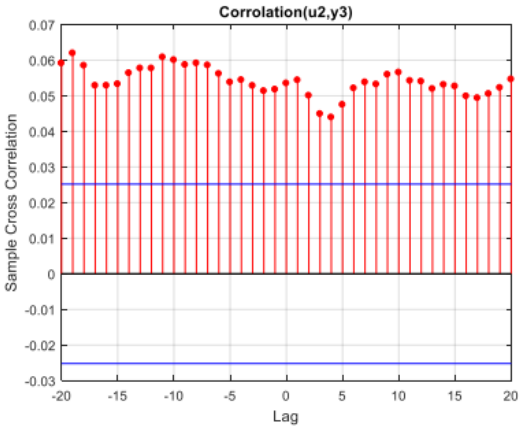
Dynamics: 1- 2



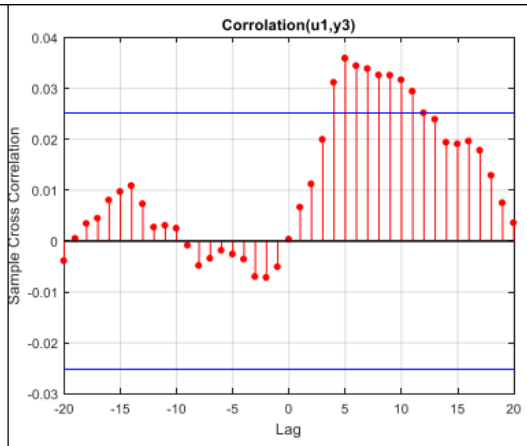
Dynamics: 15



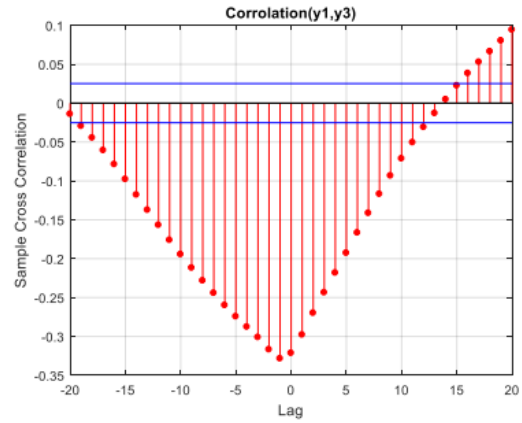
Dynamics: 20



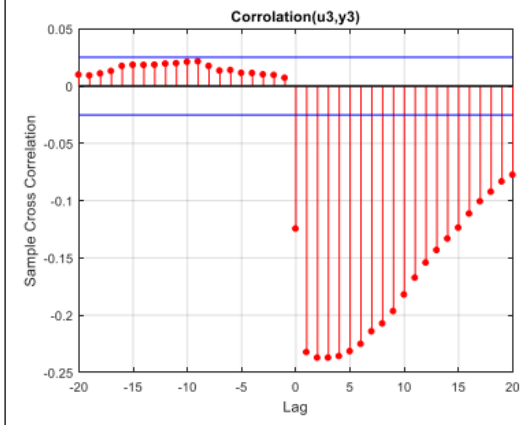
Dynamics: 1-7-10-14-20



Dynamics: 5-9-16



Dynamics: 20



Dynamics: 2-3-4

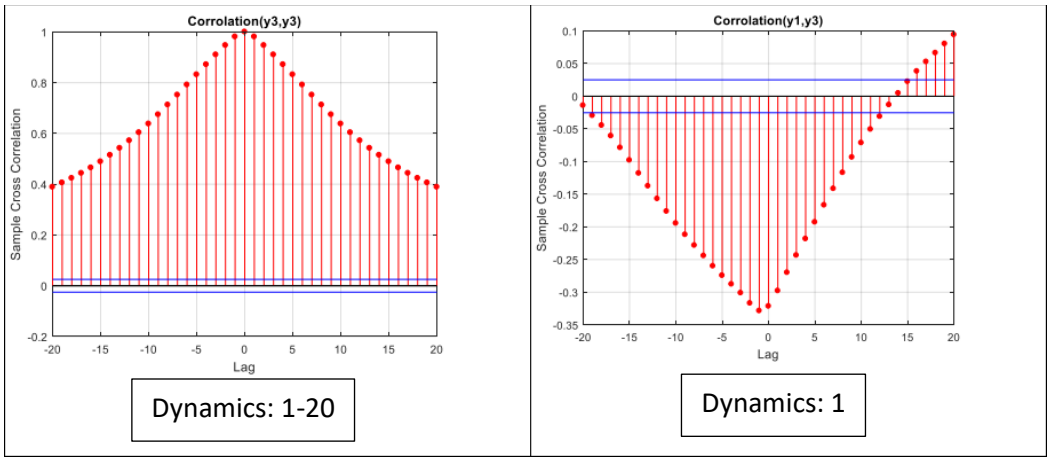


Figure 229 The correlation plot of the outputs and inputs

Dynamics for u_1: 19-16-10-5-8

Dynamics for u_2: 1-3-4-7-8-10-14-15-18-20

Dynamics for u_3: 2-5-6-8-9-10-14-16-18-19

Dynamics for y_1: 1-2

Dynamics for y_2: 1-4-15

Dynamics for y_3: 1-20

We have divided the data into a 70-30 ratio to create training and testing datasets. We then plot the output results on the test data. Using these dynamics, we estimate a neural network response, which is as follows.

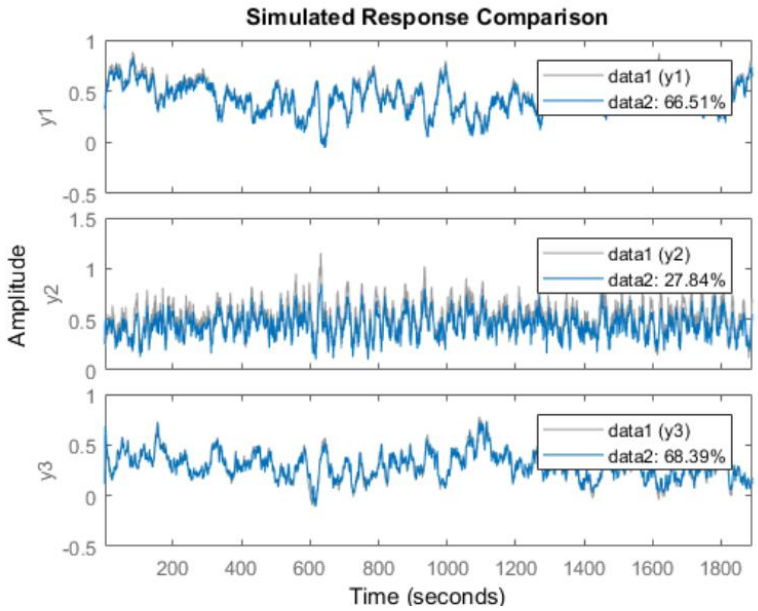


*Figure 230 Actual and estimated **MLP** system output- **test data***

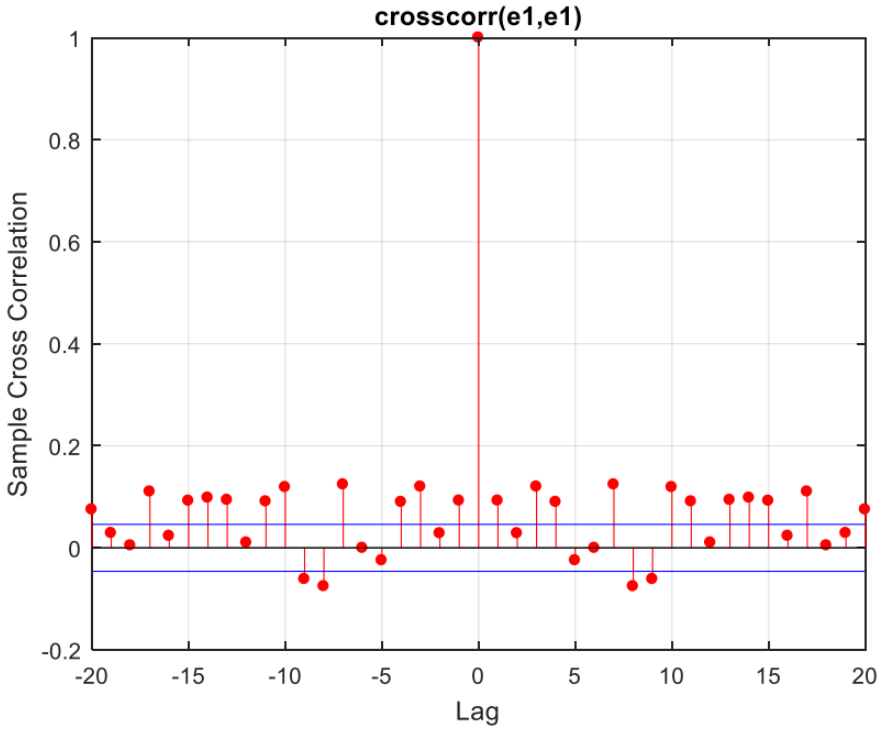


Figure 231 Error correlation first output ***on test data***

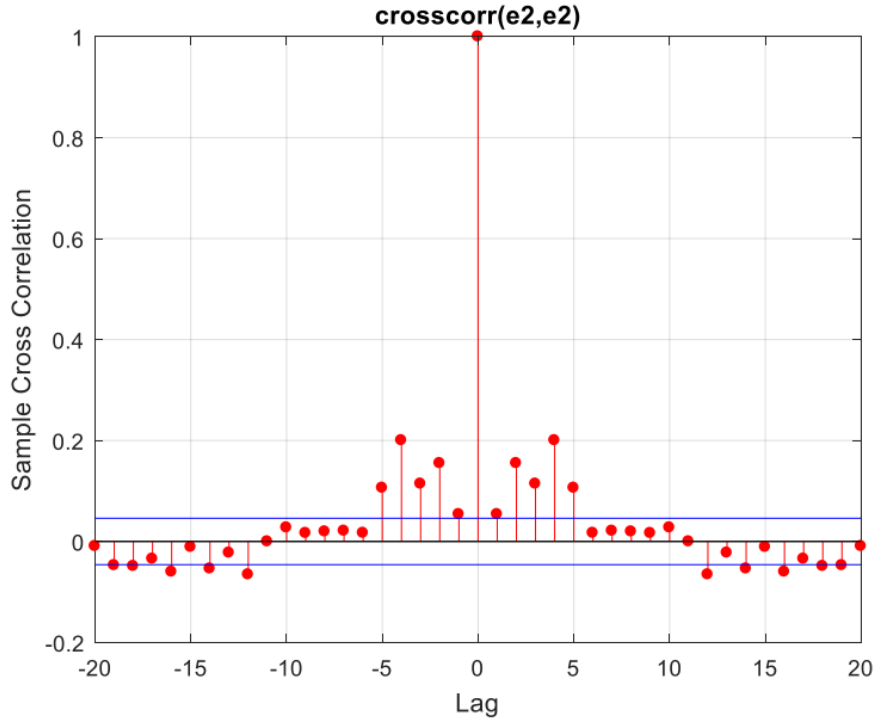


Figure 232 correlation second output ***on test data***

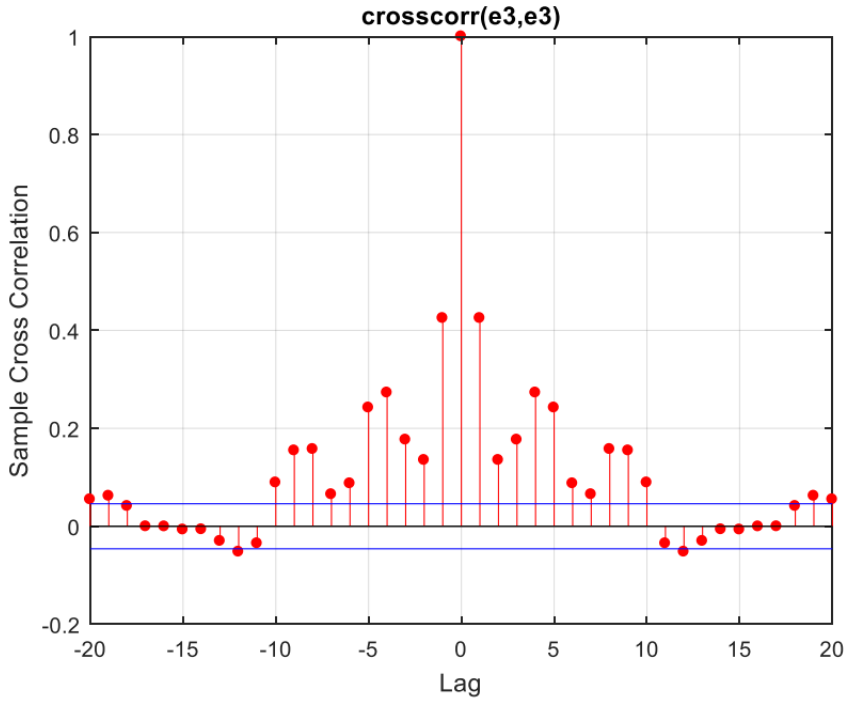


Figure 233 correlation third output ***on test data***

We can see that the estimated neural network model is not very acceptable, and the fitting percentage is low.

Second step)

We identified 32 dynamics using statistical analysis and correlation tests among the data. Now, we want to select important dynamics from them using the FS method. We try to distinguish important dynamics from unimportant ones. We have to feed all the subsets of effective dynamics as input to the network for training and find the best subset of dynamics that minimizes the SSE.

Stage one: We feed each effective dynamic individually to a single-layer MLP neural network designed with 10 neurons, trained 10 times, and one input and three outputs. Then, we plot the MSE error for the test data. The dynamic with the lowest MSE is selected as the first effective dynamic.

Stage two: Now, among the remaining dynamics, we select one and put it alongside the previously chosen first effective dynamic. We design a network with two inputs, three outputs, and the same number of neurons as in the previous stage. We repeat this process for all remaining dynamics. The network with the lowest MSE for the test data is the winner, and its second input is the second effective dynamic.

Stage n-1: We continue this process until the n-1 stage, creating a network with the same number of neurons as the previous stage. At this point, we have a network with the same number of neurons as the number of candidate input dynamics and 3 outputs.

Stage n: We examine the MSE error chart for the entire process of each stage and identify the point where significant changes no longer occur. Up to that point (n-1), we consider the selected dynamics as the main dynamics of the system.

In this case, we have completed all stages, and in the final stage, we observe that the MSE error does not significantly change from stage 12 onwards. Therefore, it is sufficient to run the algorithm up to stage 12 and consider the selected dynamics up to stage 12. The order of selecting regressors is also presented according to the stages of the algorithm:

$$y_1(t-1), y_3(t-1), y_2(t-1), u_3(t-2), y_2(t-4), y_1(t-2), u_1(t-19), u_3(t-10), y_3(t-20), u_1(t-5), u_2(t-3), u_3(t-8)$$

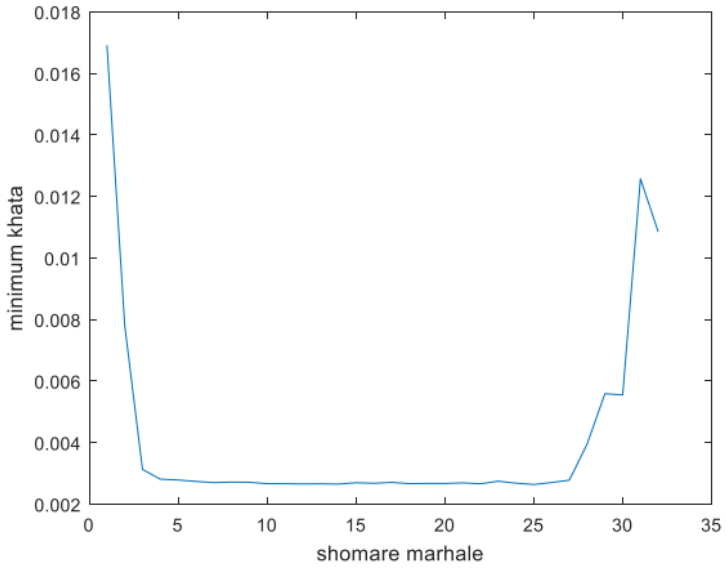


Figure 234 Variations in MSE in different steps of FS algorithm

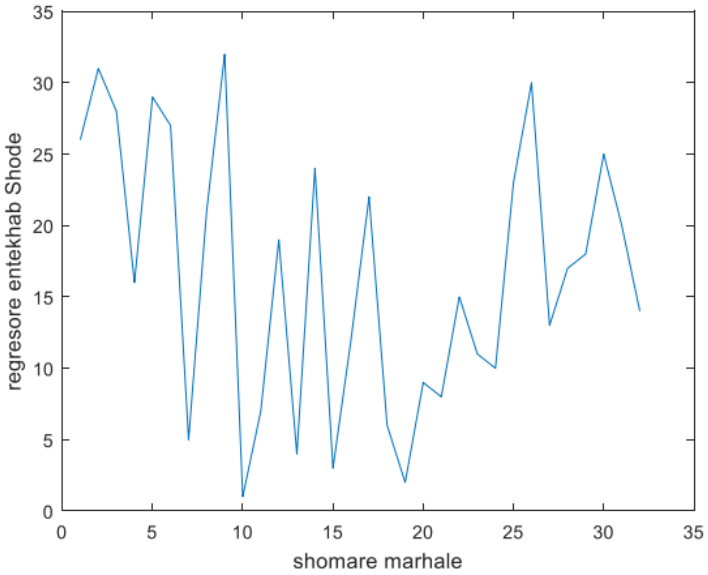


Figure 235 plot o different regressors chosen in every step of FS algorithm

It seems that you intended to provide further details or data, but the information is incomplete. Could you please provide more details or clarify your request?

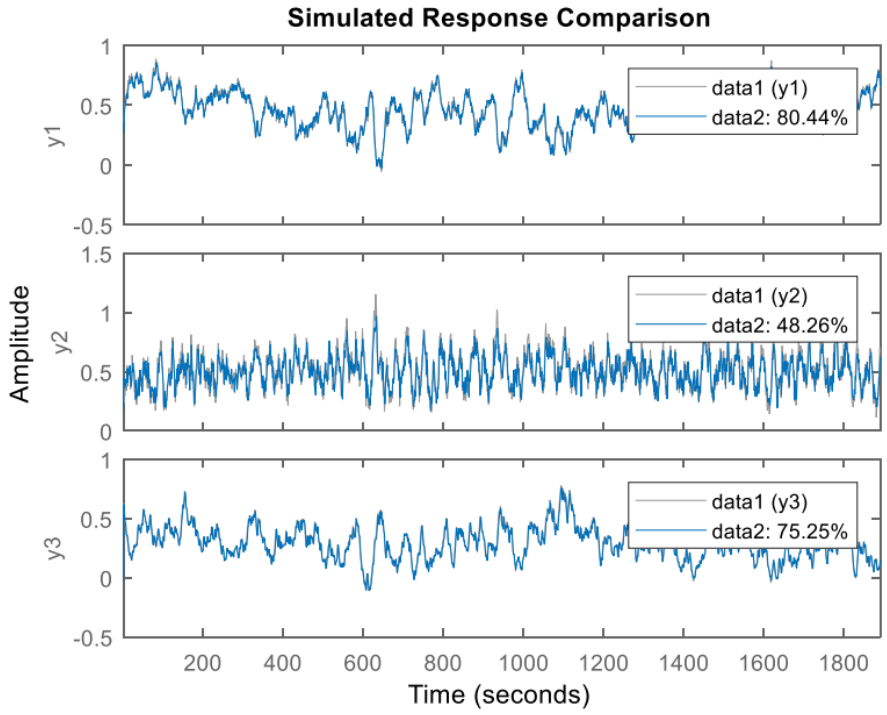


Figure 236 the actual and estimated system outputs and the fitting percentage acquired by FS algorithm

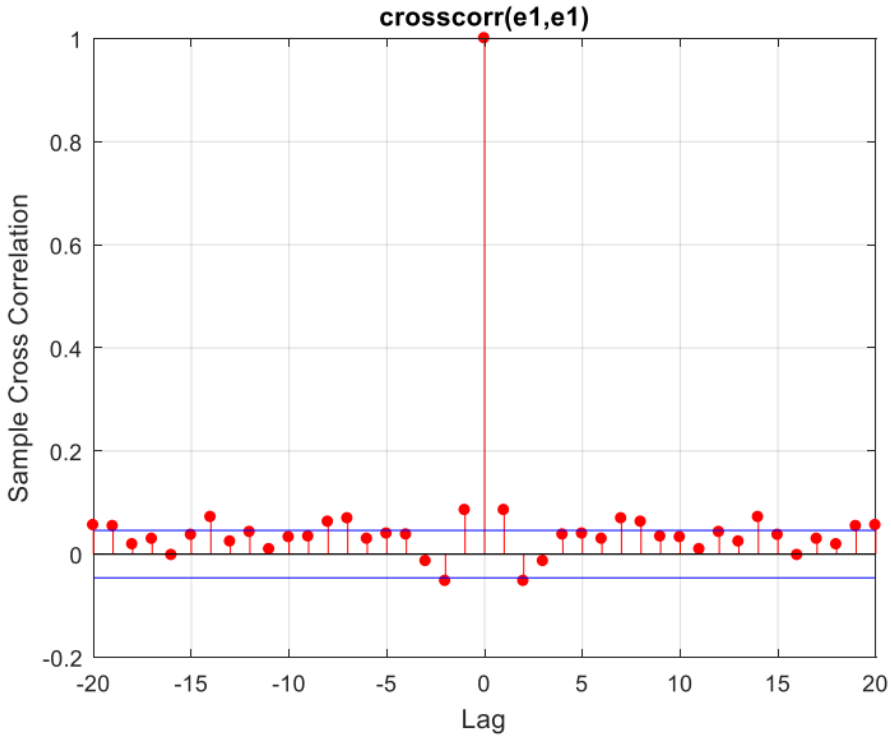


Figure 237 Error correlation of first output-on test data-FS algorithm

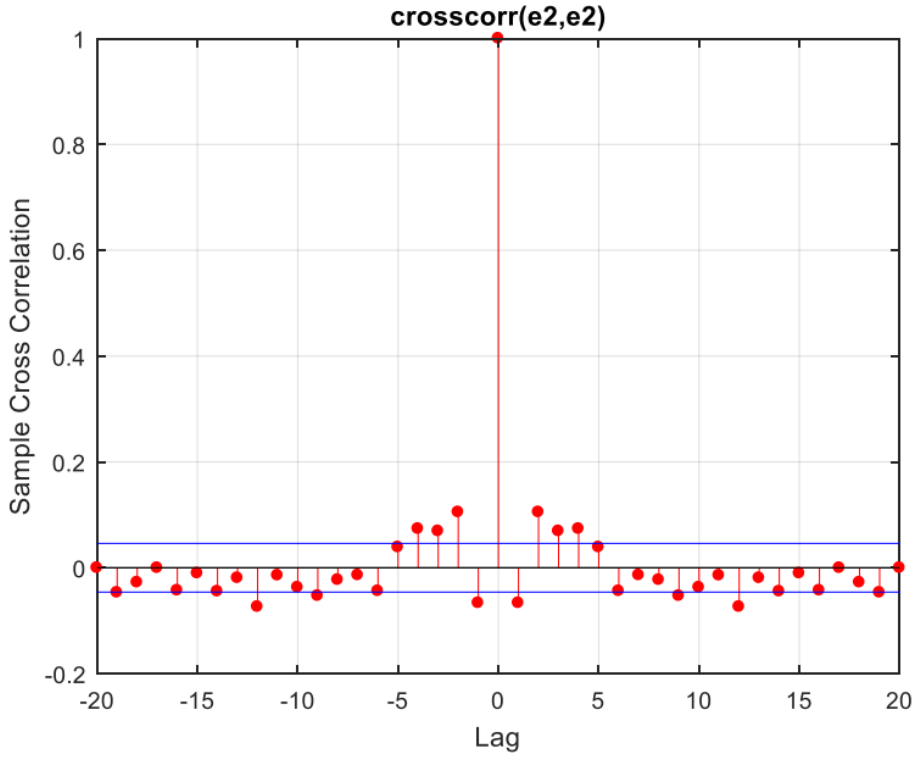


Figure 238 Error correlation of second output-on test data-FS algorithm

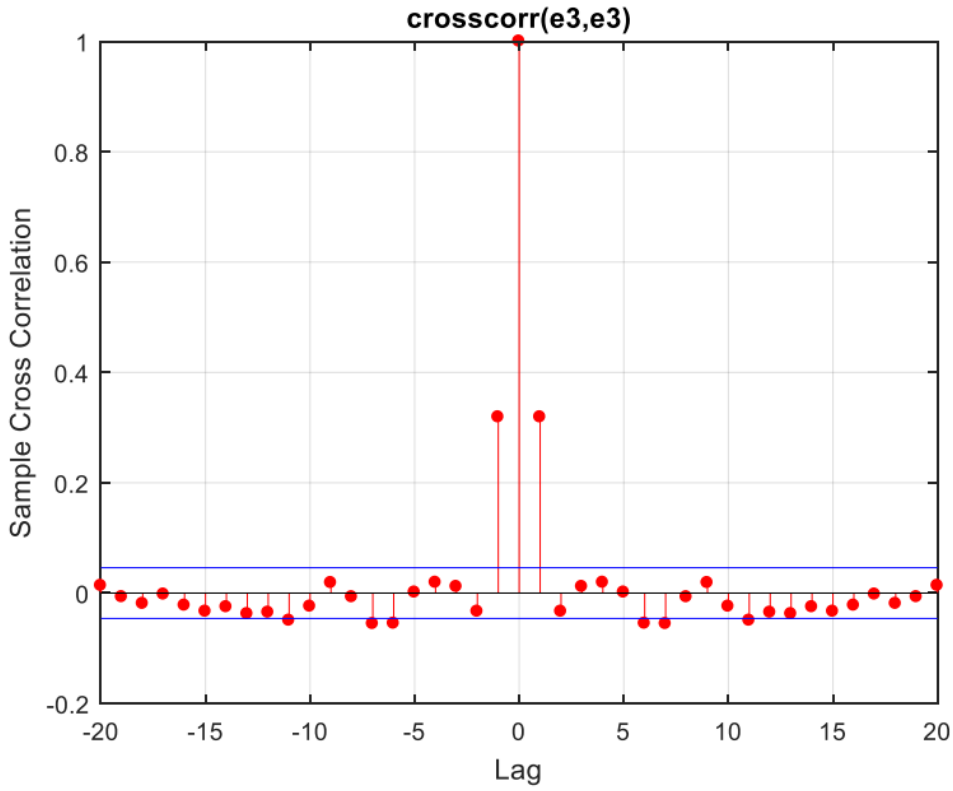


Figure 239 Error correlation of third output-on test data-FS algorithm

It looks like we have obtained a much better response compared to the previous stage. The status of the cross-correlation of the error with itself has also significantly improved.

In this stage, we turn to the genetic algorithm to identify the dynamics of the system. To use the genetic algorithm in MATLAB, we utilize the "ga" command, which has the following general form:

```
x = ga(fitnessfcn,nvars,A,b,[],[],LB,UB,nonlcon,IntCon,options)
```

For **fitnessfcn**, we used a single-layer neural network similar to the one in the feature selection (FS) to make a fair comparison between the methods. Since we have 3 inputs and 3 outputs, we set **nvars** to 6. Additionally, according to MATLAB's documentation, as we do not have a linear relationship to calculate the local minimum based on it, we need to set A and b as [] []. LB and UB determine the lower and upper bounds of the depth of the dynamics, respectively. We set the lower and upper bounds to 0 and 40, respectively. In the options section, we considered the number of generations to be 3 to avoid extensive calculations. Finally, we used the following command:

```
fitnessfcn = @(x) MSE_GA(x);
ga_opts = gaoptimset('Generations',3,'display','iter');
[x_ga_opt, err_ga] = ga(fitnessfcn, 6, [],[],[],[],zeros(1,6),40*ones(1,6),[],1:6,ga_opts);
```

The function MSE_GA takes the depth of dynamics as input, feeds inputs with the received depth to the neural network, and provides the Mean Squared Error (MSE) related to the model's estimated data for the test data as output.

After applying this algorithm, the following effective dynamics were identified:

```
Dynamics =

    10    14     1     6     1     1
```

These dynamics, in order from left to right, correspond to the first input, the second input, the third input, the first output, the second output, and the third output. In other words, using the genetic algorithm, the identified effective dynamics are as follows:

$$y_3(t-1), y_2(t-1), y_1(t-6), u_3(t-1), u_2(t-14), u_1(t-10)$$

After applying these dynamics to the neural network with the specifications mentioned earlier, the obtained results are as follows:

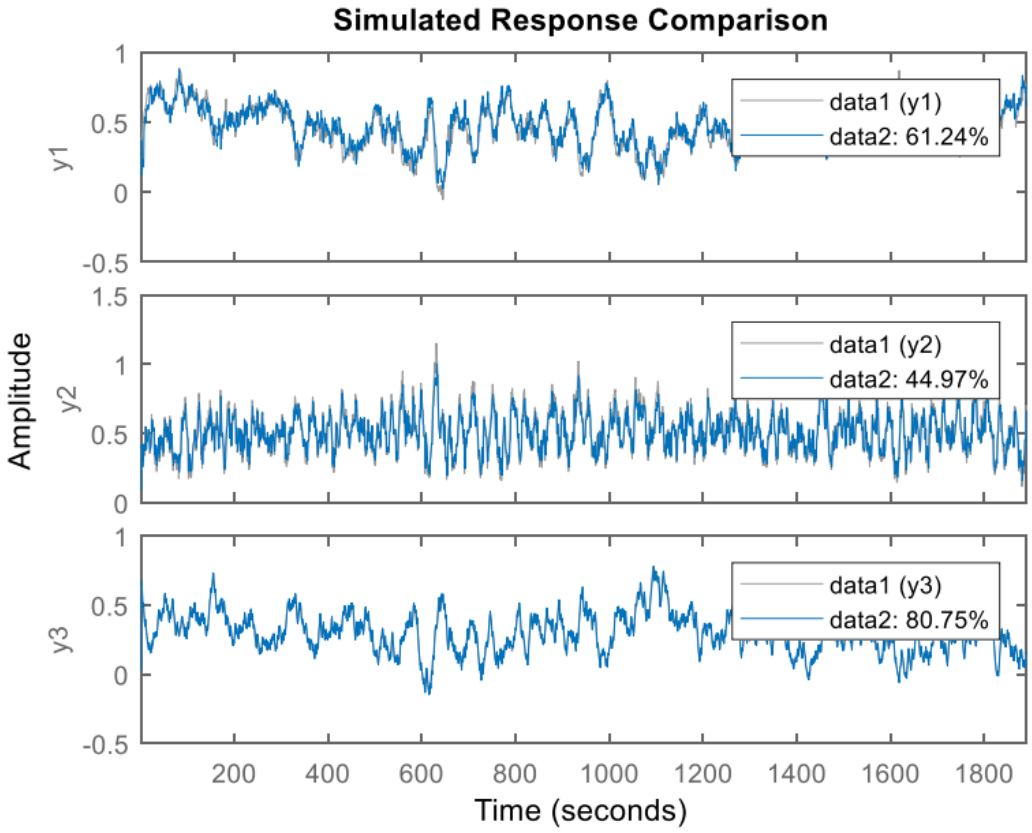


*Figure 240 the actual and estimated system outputs and the fitting percentage acquired by GA algorithm-**test data***

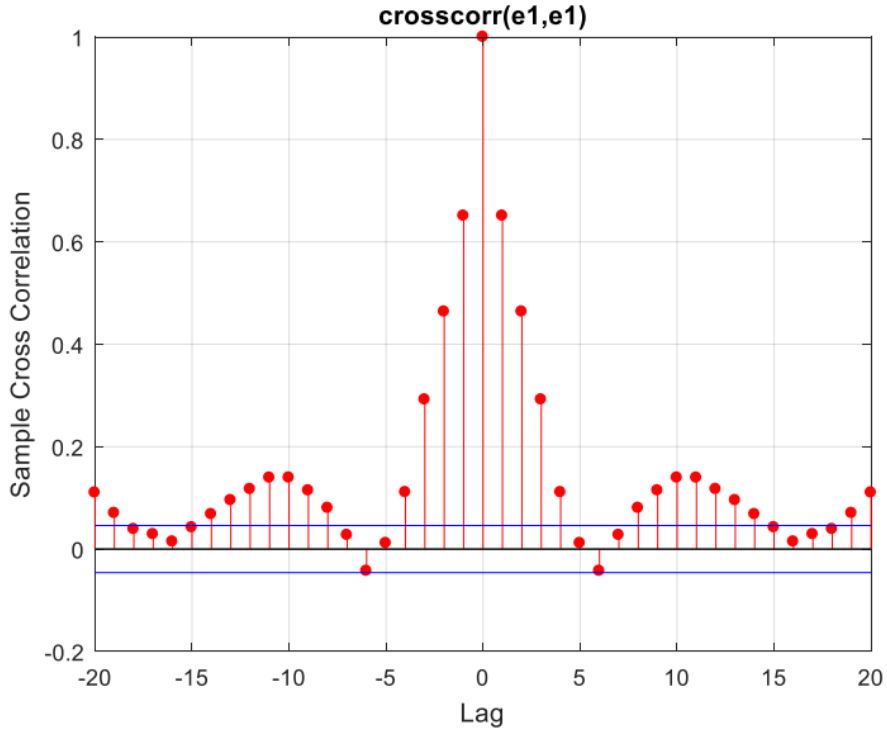


Figure 241 Error correlation of first output-on test data-GA algorithm

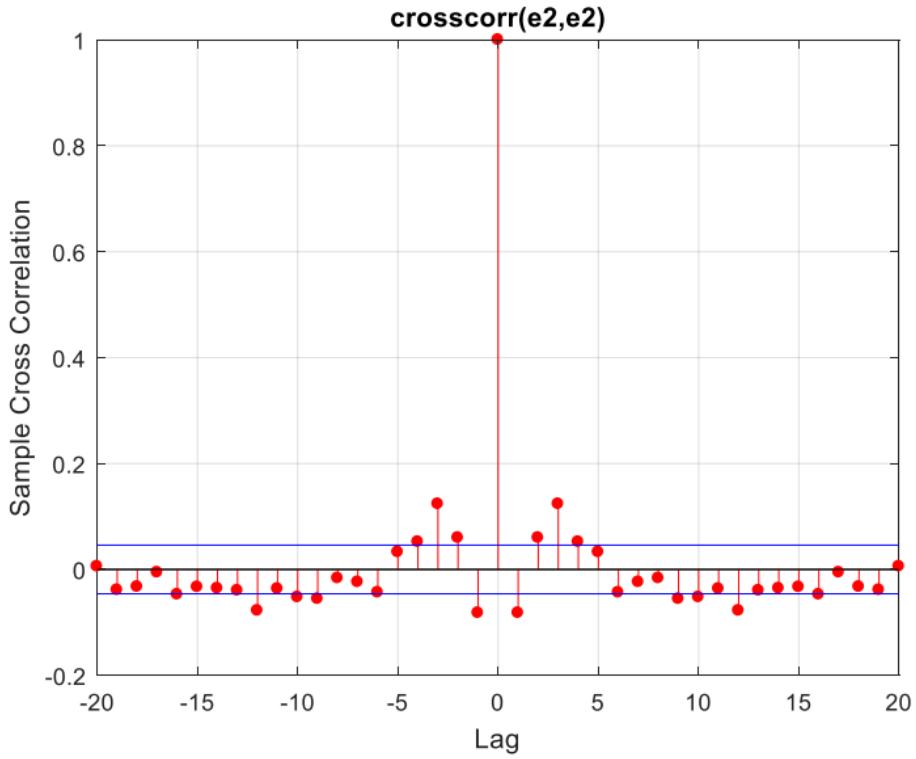


Figure 242 Error correlation of second output-on test data-GA algorithm

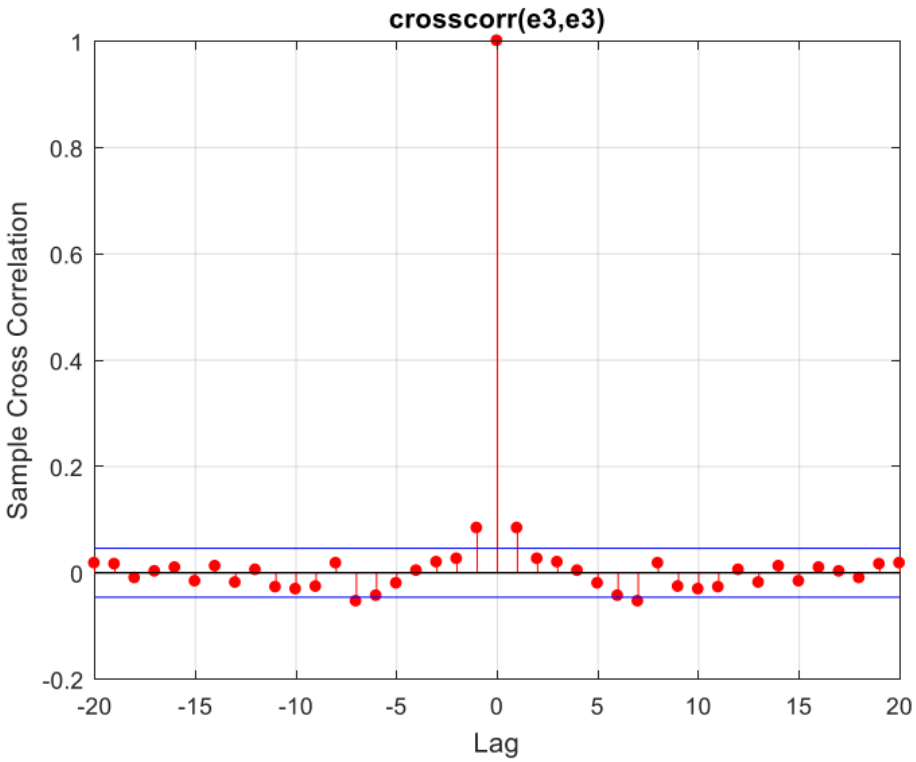


Figure 243 Error correlation of third output-on test data-GA algorithm

As observed in the figure, the error for the first output is not well whitened, and the percentage shown in the figure for the first output cannot be trusted. However, based on the figures, the error for the second and third outputs is well whitened, and the observed percentages for these two outputs are reliable. In summary, considering the whitening of errors and the percentage of matching between the actual and estimated outputs, it can be stated that the genetic algorithm performed better than the correlation test method, but it was weaker than the feature selection (FS) method.

With careful observation of the autocorrelation figures for errors, it was noticed that we only have an issue with the first output. Through trial and error, an attempt was made to modify the dynamics of the first output in the following manner: Instead of the dynamics y1(t-6), we considered dynamics y1(t-1) similar to the dynamics of the other two outputs. The results were as follows:

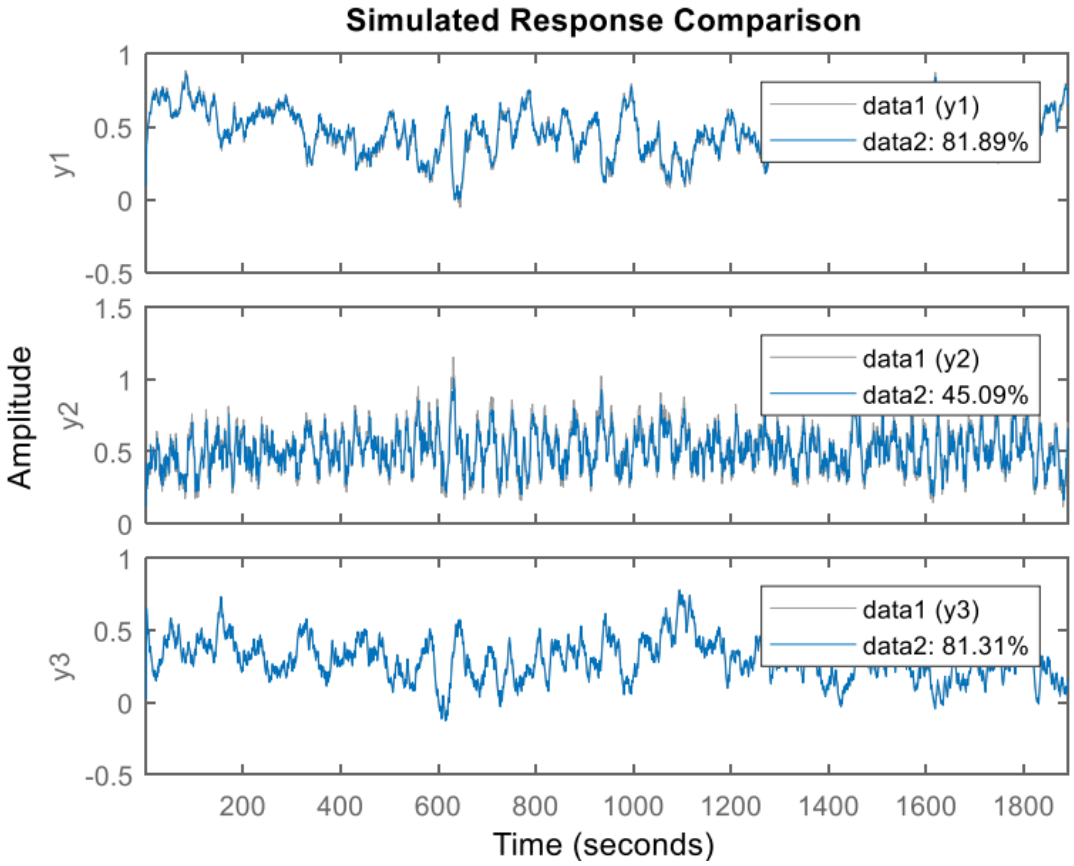


Figure 244 the actual and estimated system outputs and the fitting percentage with new regressors

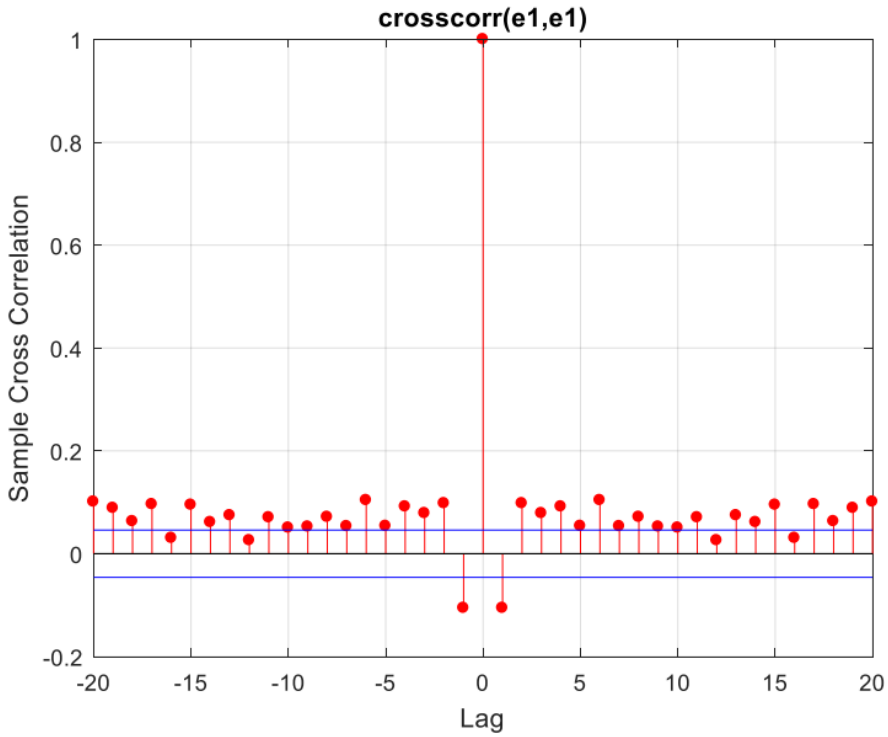


Figure 245 error correlation first output on test data-new regressors

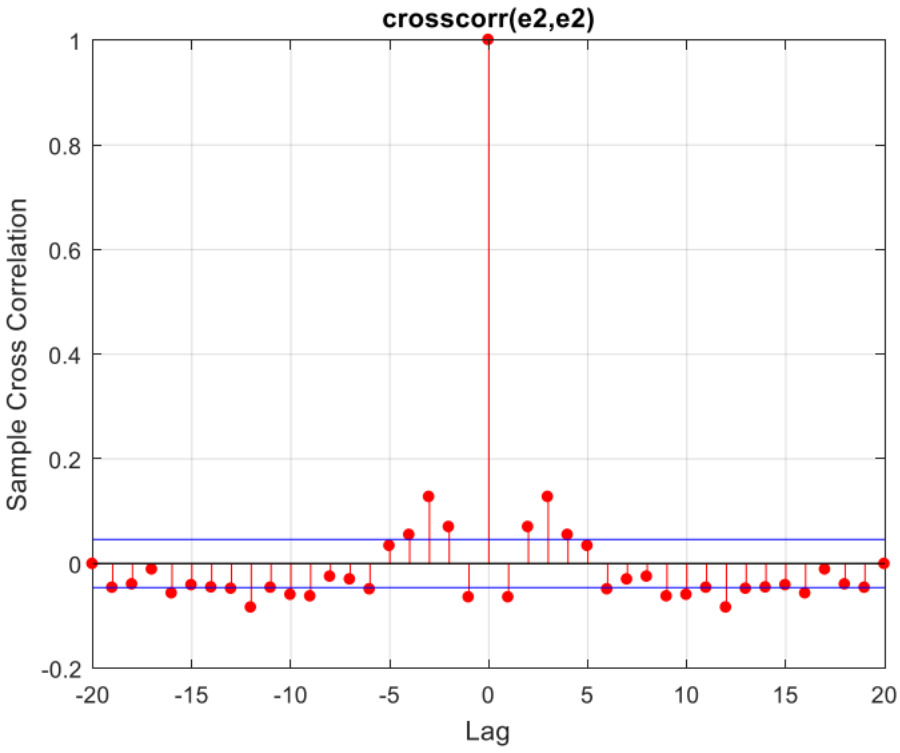


Figure 246 error correlation second output on test data-new regressors

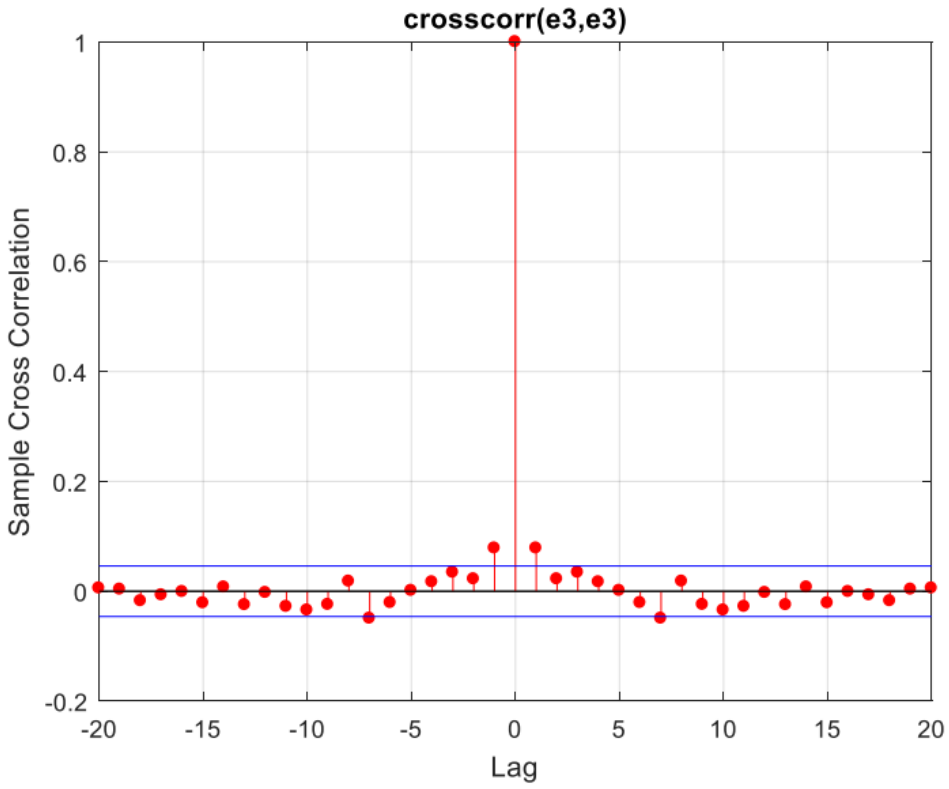
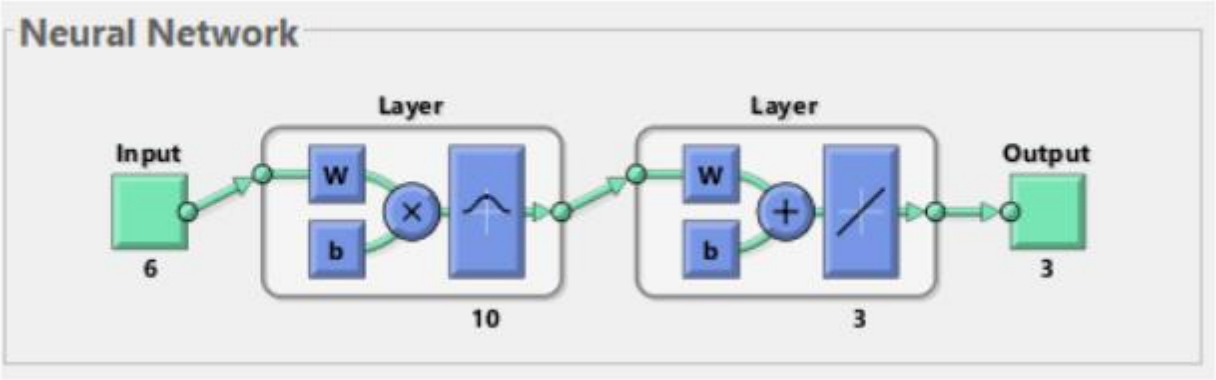


Figure 247 error correlation third output on test data-new regressors

As observed in the figures, by modifying the dynamics of the first output, the error has been effectively whitened. The percentage of match for all three outputs has improved compared to the previous state, with a significant enhancement for the first output. Additionally, as evident in the figures, this improvement did not introduce any disturbances to the whitening of errors in the second and third outputs. Comparing these results with the FS method, it can be concluded that the identified dynamics in this section provided better responses.

Now, with the obtained dynamics, we proceed to train RBF and NRBF networks.

We train an RBF network with 10 neurons after trial and error. The network structure is as follows:



The results:

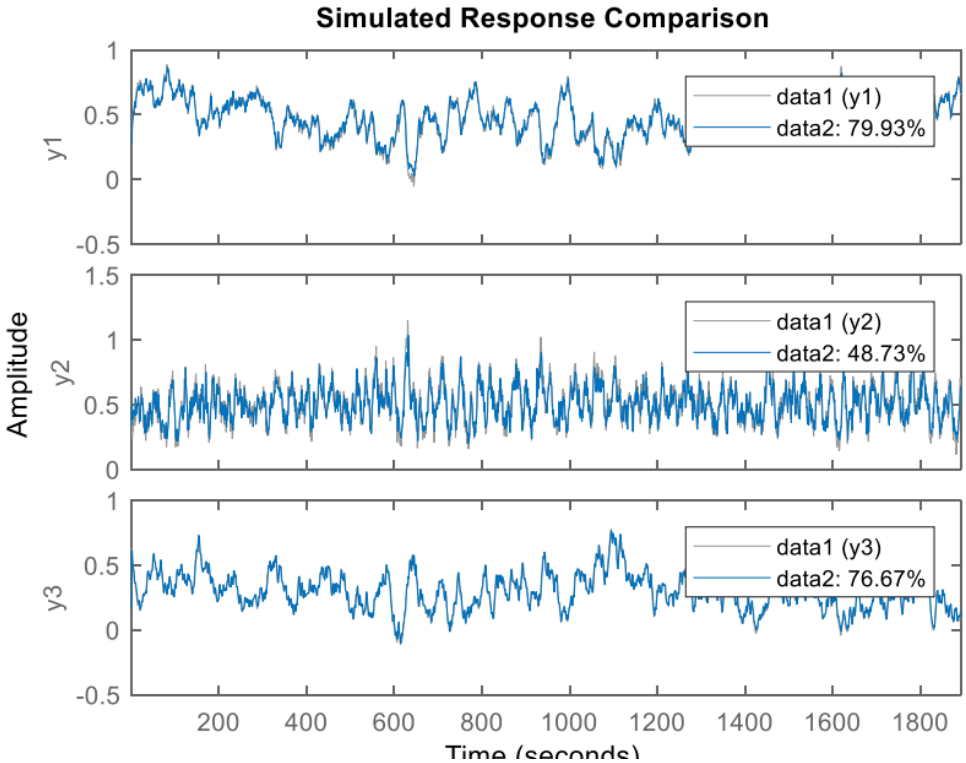


Figure 248 Actual and estimated system outputs using RBF

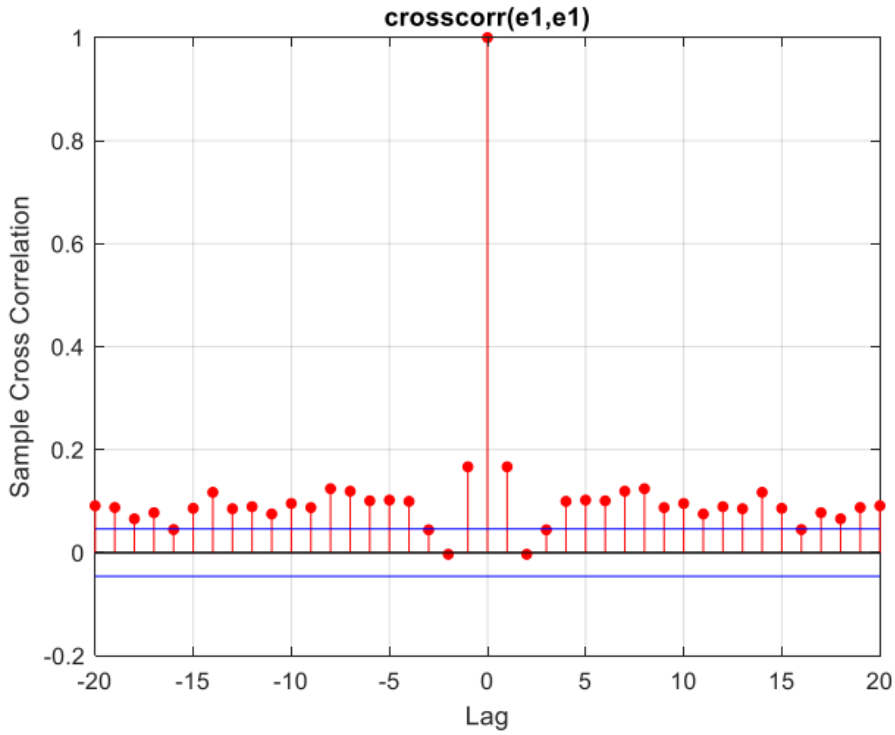


Figure 249 first output error correlation RBF network

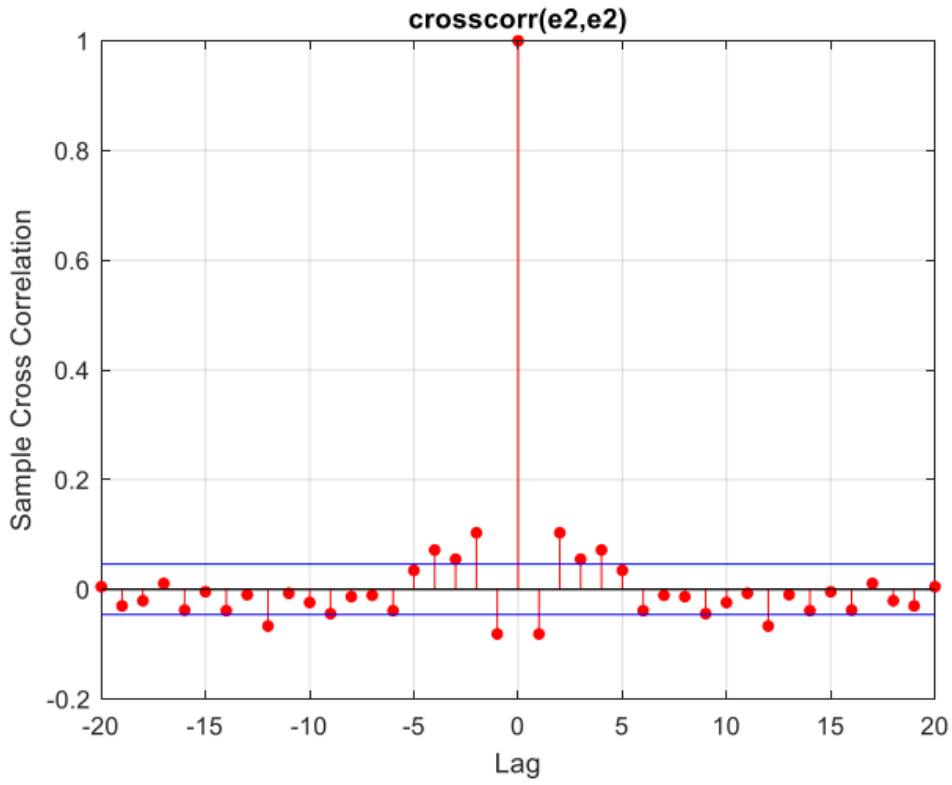


Figure 250 second output error correlation RBF network

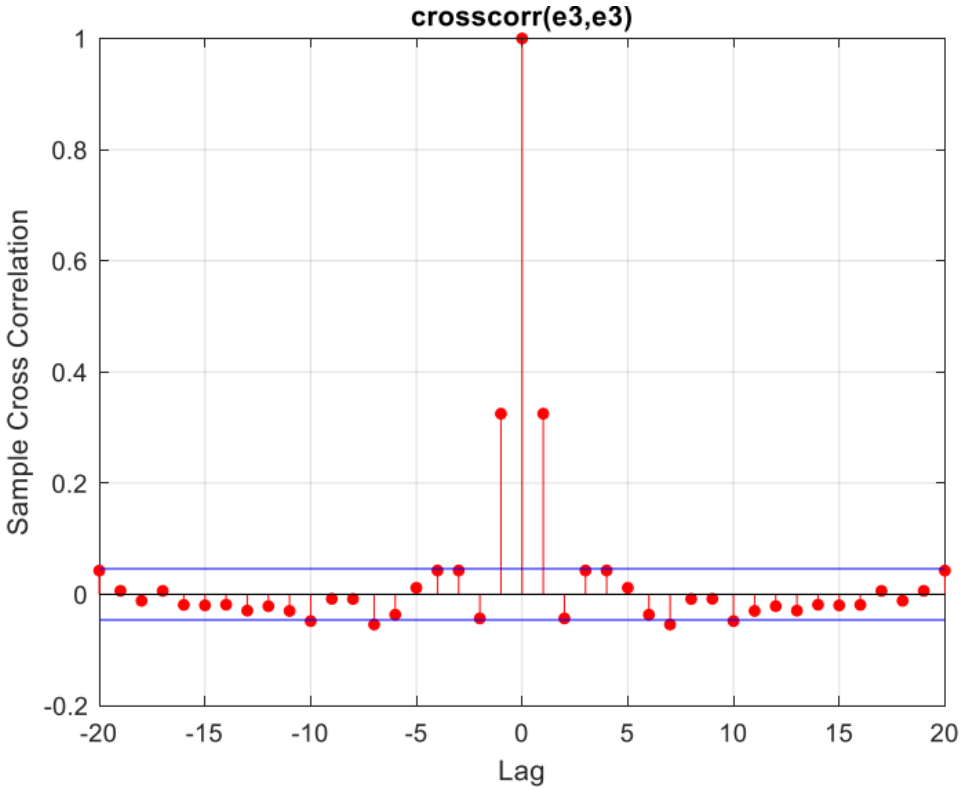


Figure 251 third output error correlation RBF network

NRBF:

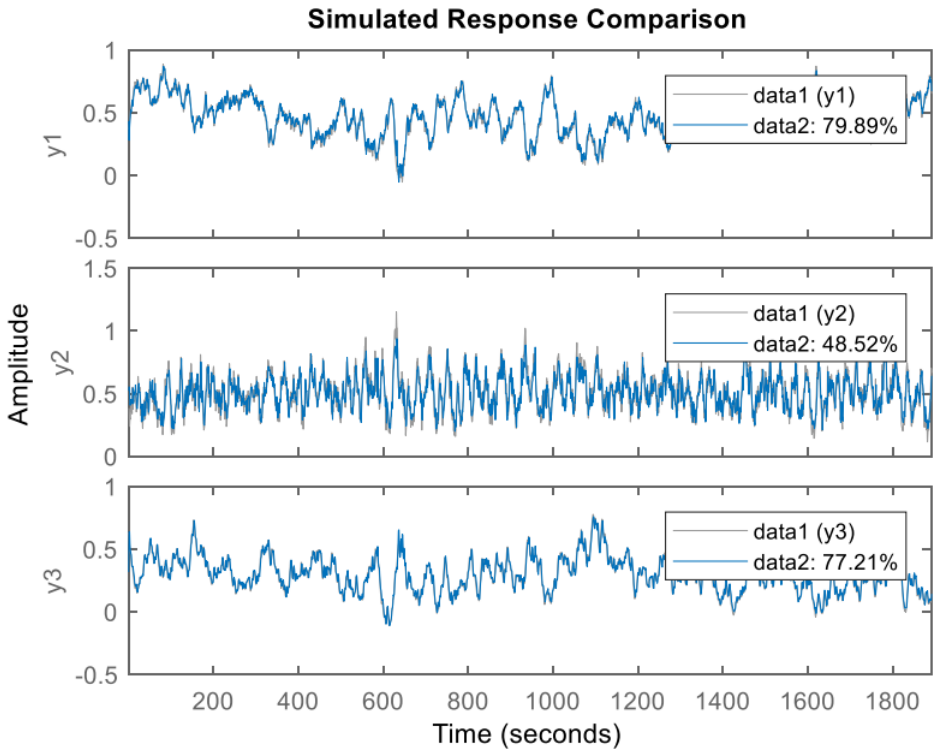


*Figure 252 the output of the actual and estimated system using **NRBF***

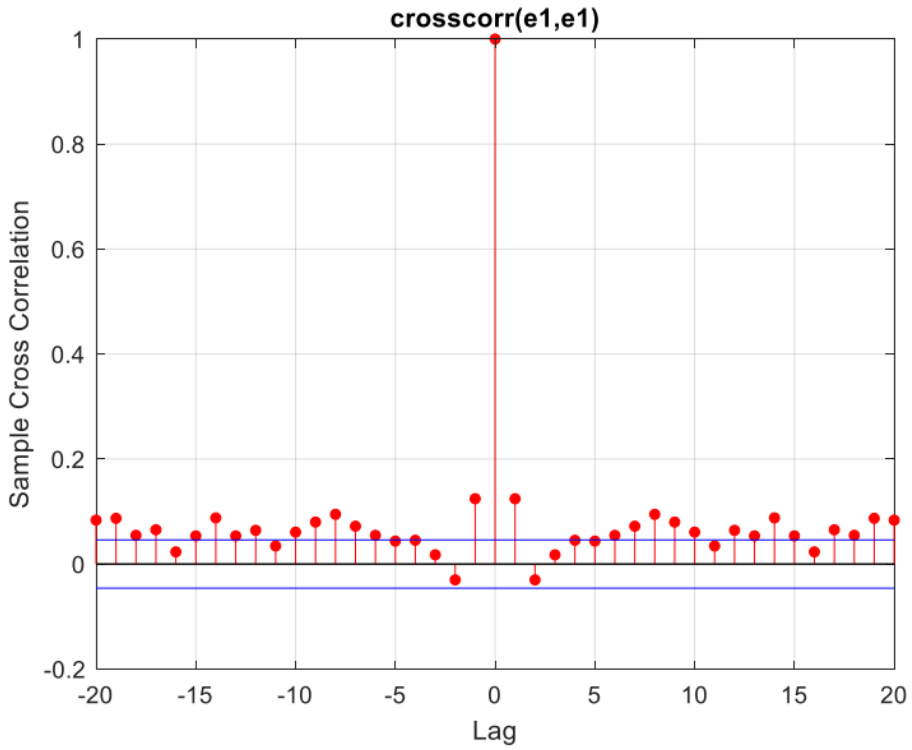


Figure 253 first output error correlation NRBF network

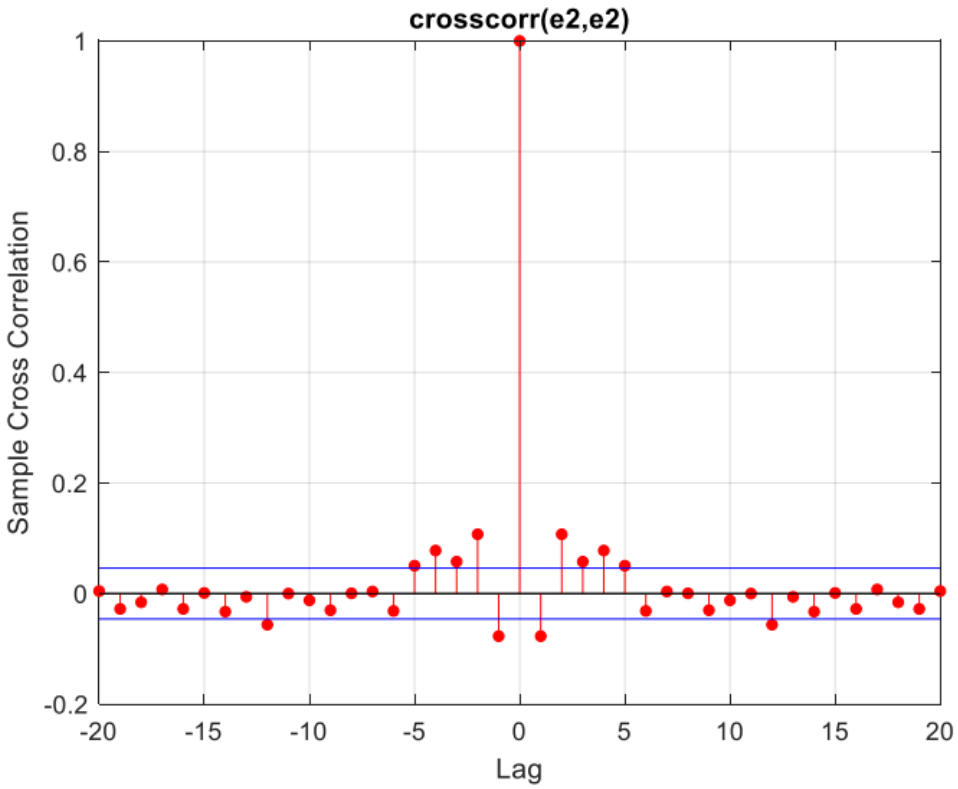


Figure 254 second output error correlation NRBF network

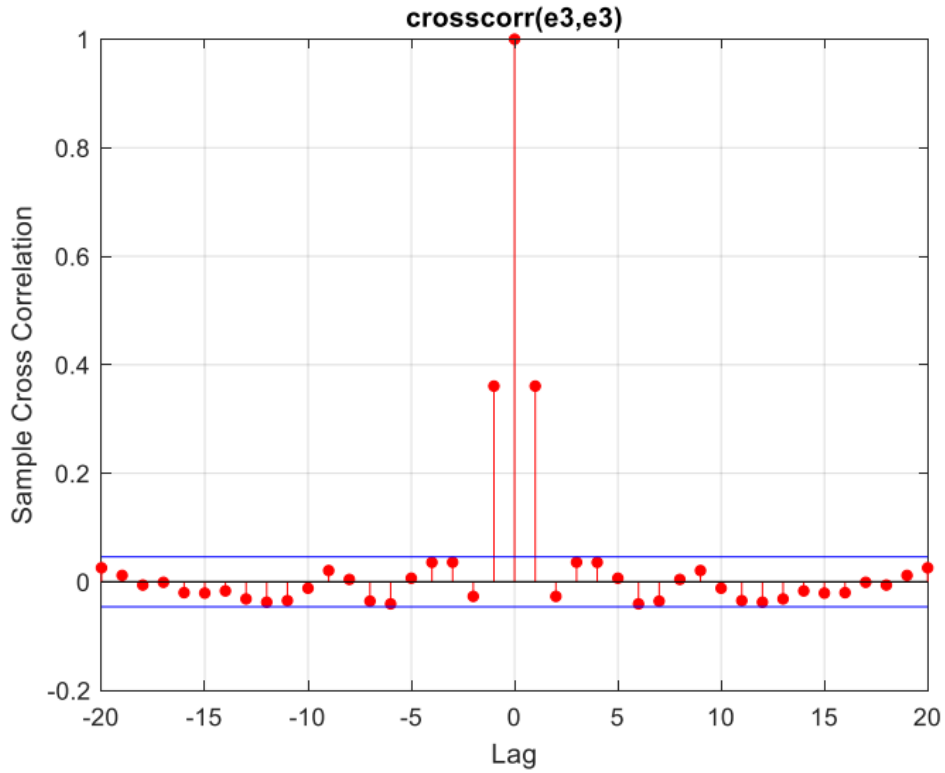


Figure 255 third output error correlation NRBF network

In both networks, the errors have approached whiteness. In NRBF, the error has subtly whitened, and the fitting percentages of both methods are almost close and similar to the MLP method.

LoLiMoT

Next, we turn to the lolimot method using the toolbox. In this toolbox, each method in the Lolimot domain is individually selected. The network is trained with the training data, and both the MSE Train Data and MSE Test Data are calculated. Finally, the method with the lower MSE Test Data is chosen.

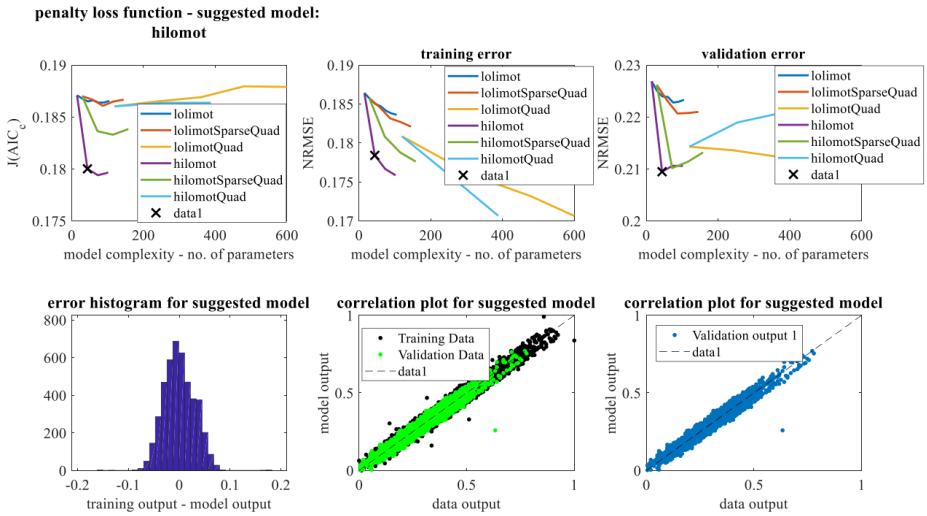


Figure 256 the statistics of the LoLiMoT method

If we want to see the output of this model for the test data and the percentage of conformity of the model with this data, we will have:

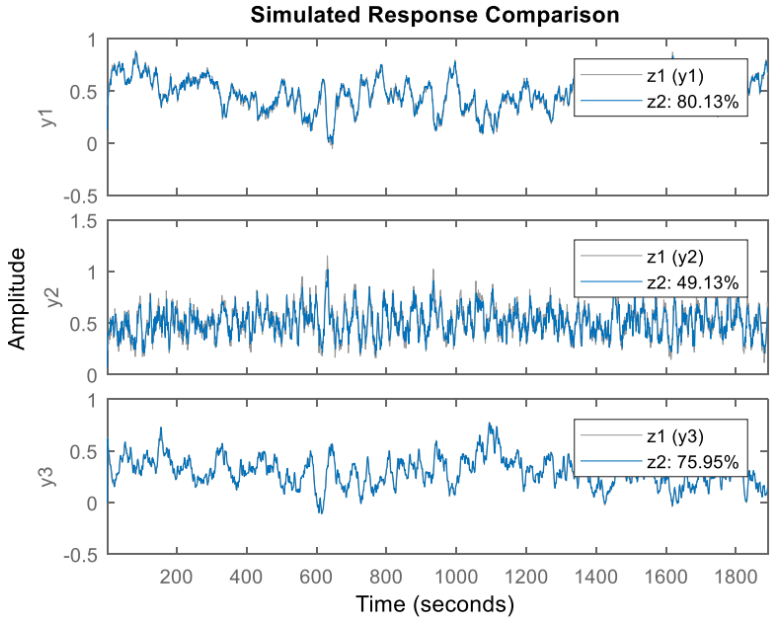


*Figure 257 The output of actual system and estimated system using **LoLiMoT algorithm***

Again, similar to before, we will plot the autocorrelation of the error to ensure the credibility of the percentage of conformity.

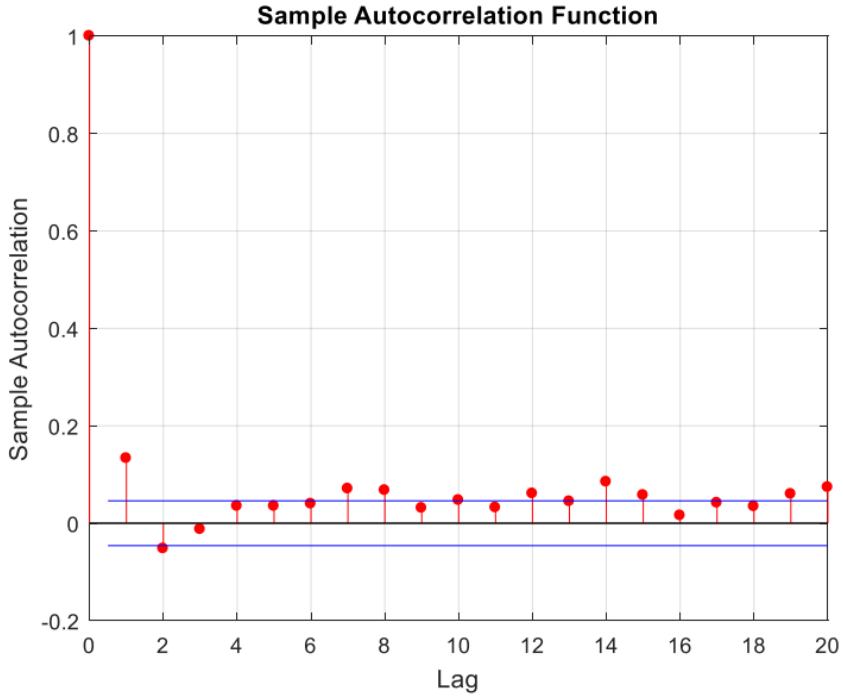


*Figure 258 Auto-correlation plot of first output error-**LoLiMoT***

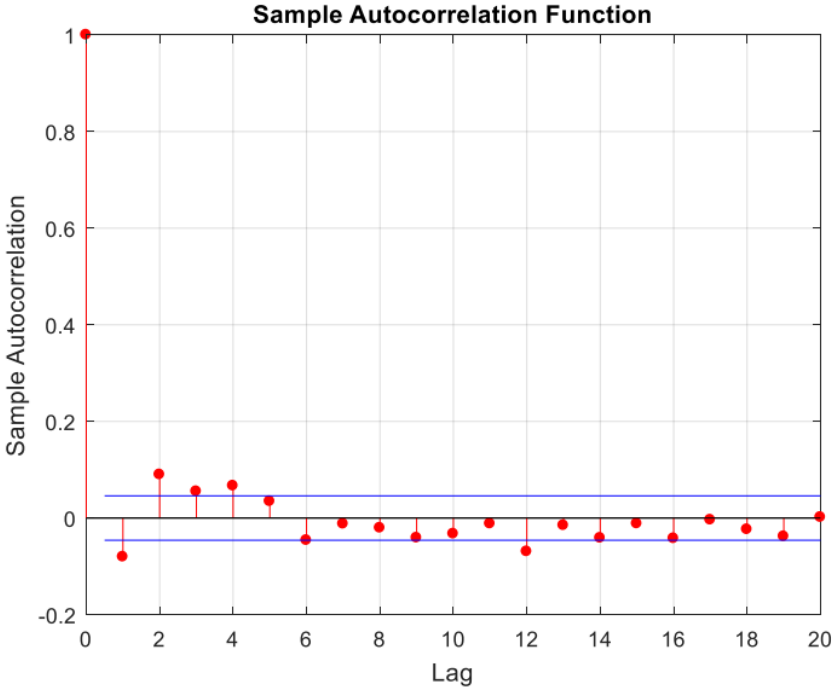


*Figure 259 Auto-correlation plot of second output error-**LoLiMoT***

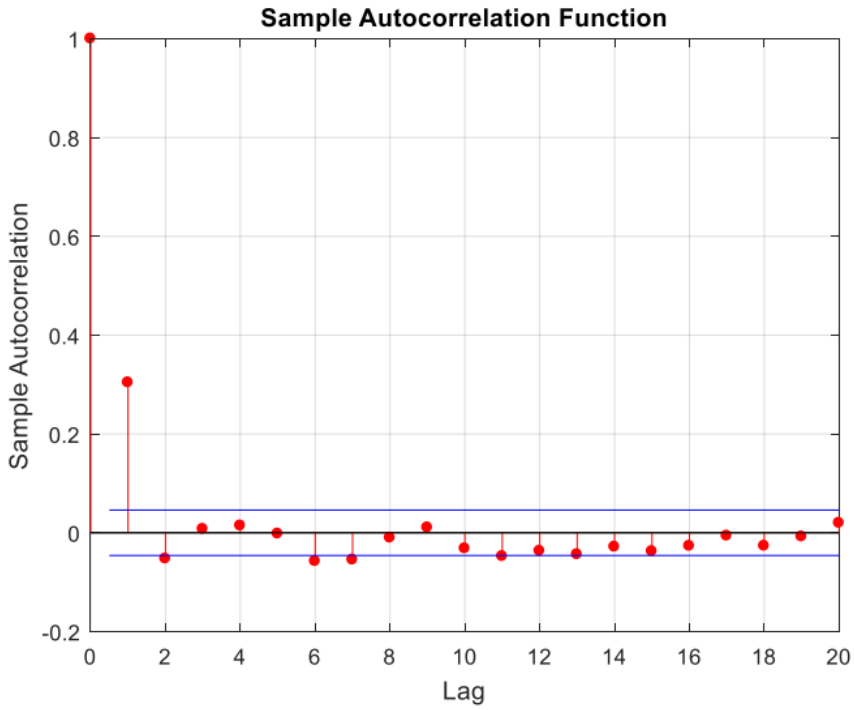


*Figure 260 Auto-correlation plot of third output error-**LoLiMoT***

It can be observed that the errors have approached whiteness significantly. The desired characteristics of the network can also be interpreted from the toolbox.

```
Estimated model complexity limit reached. The improvement of
the loss function (penaltyLossFunction) was 2 times less than
1.000000e-12 on TRAINING data.


Final net has 4 local models and 192 parameters: J = 0.340679

Net 2 with 2 LMs and 90 parameters is suggested as the model with the best complexity trade-off.>>
```

Analyses were provided for each of the methods in their respective sections. It was observed that the fitness percentages of these three methods were very close, and the outputs of all three methods were close to whitening the error.