In The Name of God

Third Homework

Nonlinear Static Systems Identification

Student:

Mohamad Alikhani

Professor:

Dr. Mehdi Aliyari Shoorehdeli

Winter of 2024

Contents

Analytical Questions

2)

Determining the appropriate kernel for Radial Basis Function (RBF) networks is a crucial step, as it significantly affects the network's performance. The choice of the kernel depends on the nature of the data and the problem you are trying to solve. Here are some common types of kernels used in RBF networks and considerations for choosing them:

1. **Gaussian (Radial) Kernel:**
   - The Gaussian kernel is the most commonly used in RBF networks.
   - It is defined as $K(x, c) = \exp(-\frac{||x-c||^2}{2\sigma^2})$ where x and c are data points, and σ is a parameter controlling the width of the kernel.
   - Choose σ carefully, as it determines the influence radius of each center. Too small values can lead to overfitting, while too large values can result in underfitting.

2. **Multiquadric Kernel:**
   - The Multiquadric kernel is another radial basis function kernel.
   - It is defined as $K(x, c) = \sqrt{||x - c||^2 + \sigma^2}$.
   - The σ parameter controls the shape of the kernel.

3. **Thin Plate Spline Kernel:**
   - The Thin Plate Spline kernel is a smoothing spline that can capture both local and global structures.
   - It is defined as $K(x, c) = ||x - c||^2 \log(||x - c||)$.
   - It is effective when dealing with smooth surfaces.

4. **Polynomial Kernel:**
   - The Polynomial kernel is also used in RBF networks.
   - It is defined as $K = (< x, c > +1)^d$, where d is the degree of the polynomial.
   - Polynomial kernels can capture non-linear relationships, but the choice of d is important.

5. **Neural Network-Based Kernels:**
   - Some researchers propose using neural network-based kernels that are learned along with the network.
   - These kernels can adapt to the specific characteristics of the data.

6. **Data-Driven Approaches:**
    - You can experiment with different kernels and parameters using cross-validation.
    - Grid search or random search can be employed to find the optimal hyperparameters.

The choice of the kernel is often empirical and depends on the characteristics of the dataset and the problem at hand. It's recommended to try different kernels, tune their parameters, and assess their performance using validation techniques.

3)

For MLP neural network:

Number of parameters of MLP network:

$$M(P+1)+M+1$$

$$M(P+1)+M+1 \leq Q \rightarrow M(3+1)+M+1 \leq 1000 \rightarrow M \leq 200$$

For RBF usually number of neurons is much smaller than the number of data points. Number of parameters in these networks are equal to: $2MP+M+1$. So:

$$2MP+M+1 \leq Q \rightarrow 6M+M+1 \leq 1000 \rightarrow M \leq 142$$

4)

The NBRF model:

$$\hat{y} = \sum_{i=1}^{M} w_i \frac{\varphi_i(u)}{\sum_{j=1}^{M} \varphi_j(u)}$$

It is guaranteed that the output of this model always falls within the following range.

$$\min_i(w_i) \leq \hat{y} \leq \max_i(w_i)$$

And of course, $\max_i(w_i)$ shall not be bigger than 2 or 3 times $\hat{y}$. Because it will result in parameter drift.

ANFIS model:

$$\hat{y} = \sum_{i=1}^{M} y_i \frac{\gamma_i(u)}{\sum_{j=1}^{M} \gamma_j(u)}$$

$$\gamma_i = \mu_{A_i}(x)$$

Comparing these two models, ANFIS and NRBF, they will be the same thing if the below condition is satisfied:

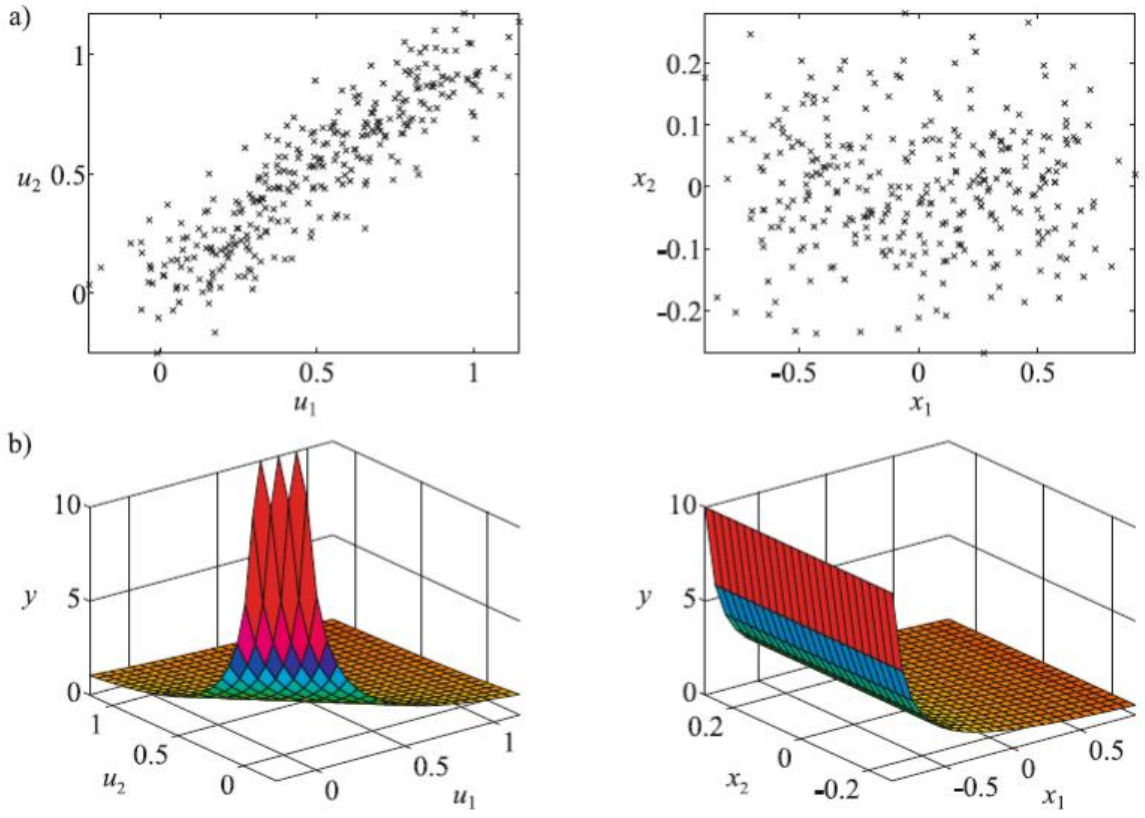
Membership functions γ_i used in ANFIS must be the same as nonlinear functions ϕ_i in NRBF model. Also w_i's and y_i's must be equal.

5)

The provided text discusses one of the limitations of the LOLIMOT algorithm, which is orthogonal axis-based partitioning in the input space. The limitations arising from orthogonal axis-based partitioning can be severe, especially when the initial dimensionality of the input space is significantly reduced. However, as the dimensionality increases, this limitation restricts the performance of LOLIMOT.

There are generally two approaches to address or mitigate this issue, but both options tend to reduce many strengths of LOLIMOT in interpretability. One solution involves incorporating an unsupervised preprocessing step. In the second approach, one can develop an algorithm for axis-oblique or non-orthogonal decomposition. The first solution has lower computational costs and is easier to implement.

Principal Component Analysis (PCA) is mentioned as an unsupervised tool for preprocessing and dimensionality reduction, which transforms the axes.



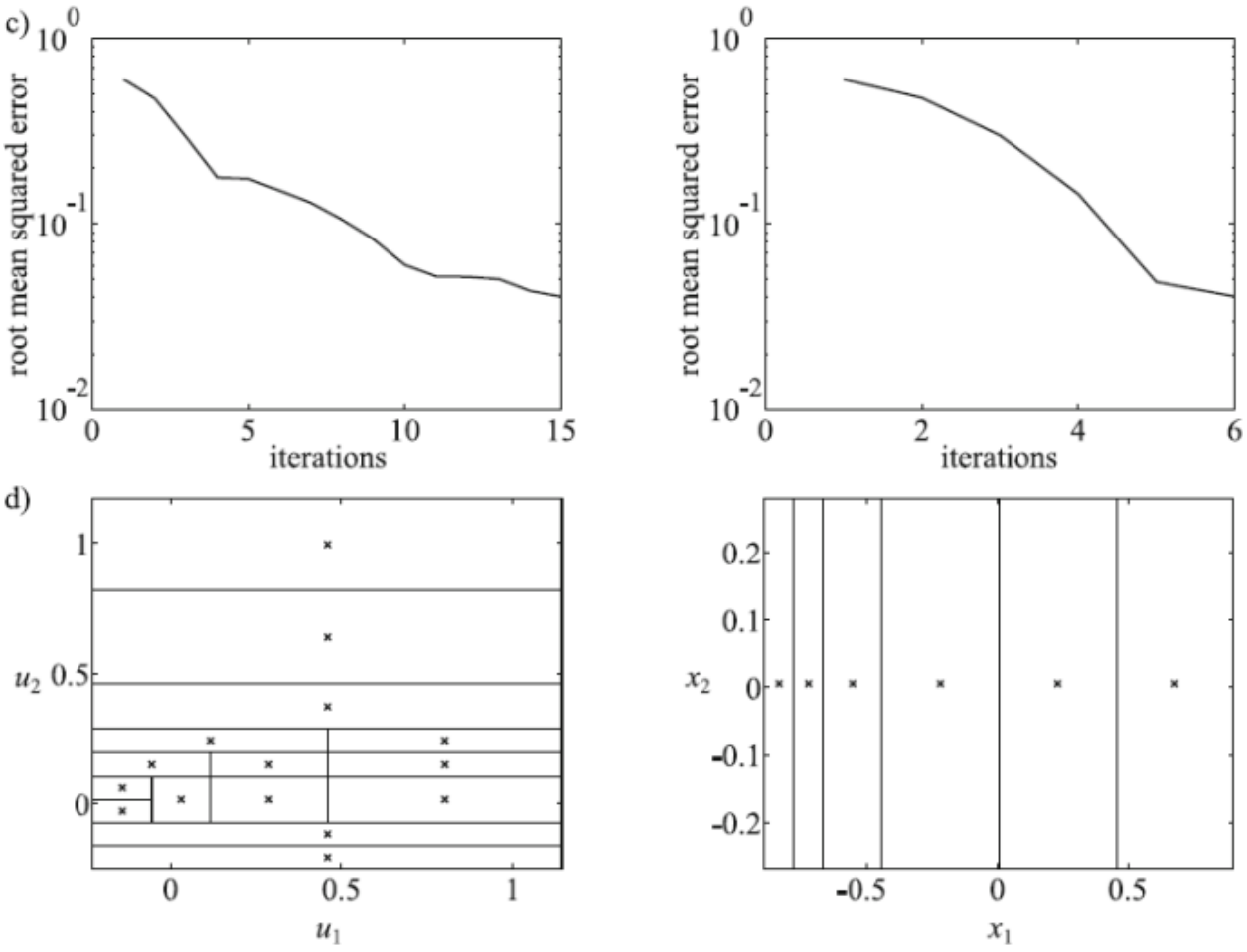


Figure 1 LOLIMOT Estimation, Left Side: Before applying PCA, Right Side: After applying PCA

a) Sample data

b) LOLIMOT estimation function

c) LOLIMOT convergence plot

d) Input space partitioning with LOLIMOT

Figure 1 provides a two-dimensional example to illustrate the impact of PCA. If the input variables u1 and u2 are linearly dependent, the data will be distributed diagonally (as shown in part a on the left side of the figure). By applying PCA, the data distribution will become diagonal, as depicted in part a on the right side. If the nonlinear behavior of the process is truly dependent on one of the input axes, the complexity of the problem can be significantly reduced using this method.

While nonlinear behavior in the original input space (see Figure, part b on the left) is stretched along a diagonal direction, in the transformed input space, it becomes orthogonal to the axes (part b on the right). In this example, the second input, x2, is interpreted as redundant or superfluous and can be discarded without losing information, reducing the dimensions of the problem.

However, it's essential to note that such favorable conditions are not typically expected in practice. Due to the orthogonal axis behavior in the transformed input space, a model with only six rules may be sufficient to achieve satisfactory performance.

Parts c and d of the right side of the figure show the convergence plot and the input space partitioning with the LOLIMOT algorithm. A comparable performance requires only 15 rules or partitions without PCA, and the input space is partitioned into 11 regions in both dimensions. The figures on the left side denote parts c and d of this description.

It is essential to note that the above example only demonstrates the effectiveness of PCA to a certain extent. While highly correlated inputs, such as u1 and u2, might sometimes be related to measurements that are nearly additive or redundant due to a physical cause (here, x1), this can only be inferred from the insight into the process (and not from the distribution of input data alone).

In the example above, nonlinearity was present in the x1 direction. If it had been along u1, all the advantages of PCA would turn into disadvantages and would, in fact, shift places in the right and left parts of figures b and d. Therefore, PCA is a good option, especially for high-dimensional problems, but its successful application cannot be guaranteed. Supervised approaches like Oblique Partitioning, despite having higher computational costs, overcome these challenges successfully, such as Hinging Hyperplanes and Smoothness Determination.

Simulation Questions

1.1)

Note: I have done whatever the question has asked, but the order is not preserved

First, specify the input and record samples of the output by applying them to the system. In total, 466 samples have been generated. Among these 466 samples, 216 samples are considered for training, 125 samples for testing, and 125 data for evaluating the network. Also, in estimating the parameters, the range and variance of the noise will have a significant impact.

To determine the noise variance, in general, if the noise range is greater than the regression range, it will prevent accurate identification of the corresponding parameters. In this system, sentences with lower powers become very small, and as a result, they are more influenced by noise, leading to higher errors for parameters corresponding to inputs with lower power, and consequently, the estimation error will be larger. The histogram below is plotted for the relevant regression.

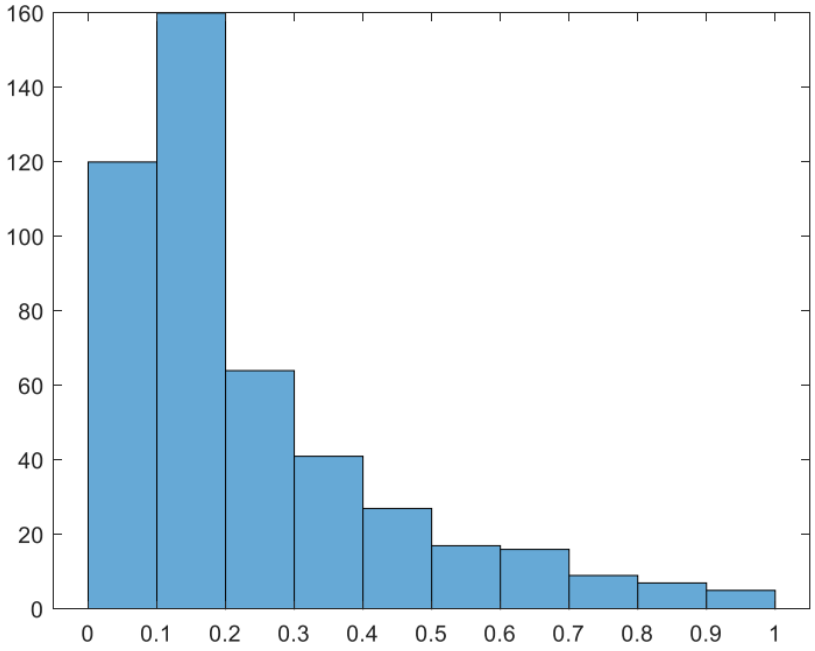


Figure 2 Regressors distribution histogram

Given the histogram and considering the Signal-to-Noise Ratio (SNR) of the input data, we will consider noise in cases with low, medium, and high variances, corresponding to variances of 0.005, 0.05, and 0.3, respectively.

$$\sigma(\text{ noise }) = 0.005 \Rightarrow \frac{\sigma_{y\,(\text{noise})}}{\sigma_{y(\text{ normal })}} = 1.0015$$

$$\sigma(\text{ noise }) = 0.05 \Rightarrow \frac{\sigma_{y_{(\text{noise})}}}{\sigma_{y_{(\text{normal})}}} = 1.03$$

$$\sigma(\text{ noise }) = 0.3 \Rightarrow \frac{\sigma_{y\,(\text{noise})}}{\sigma_{y(\text{ normal })}} = 1.33$$

Also, before training, it's essential to pay attention to the range of the input and output data. If their range is not suitable, normalization should be performed. Therefore, it's better to take a look at the range of training inputs and outputs. The plots of the generated output data without noise, with low noise, with medium noise, and with high noise are provided in the following figures.

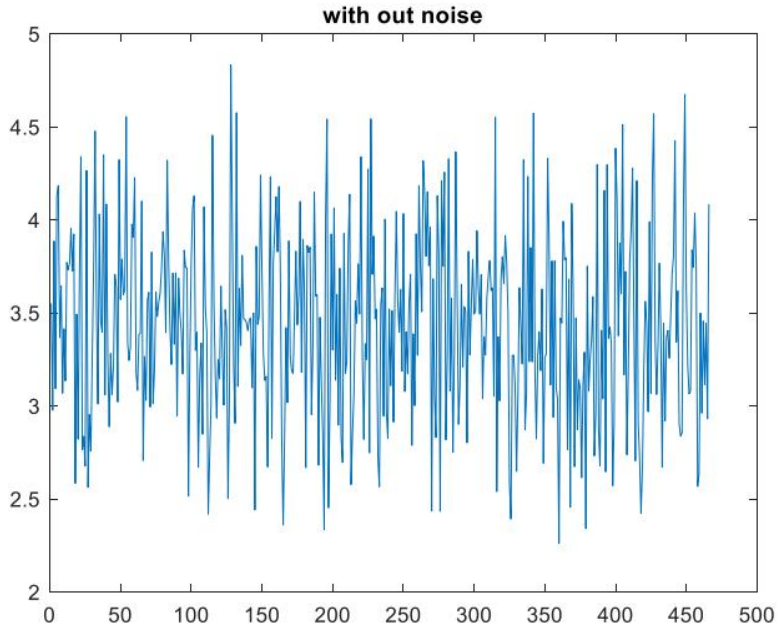


Figure 3 The output without noise

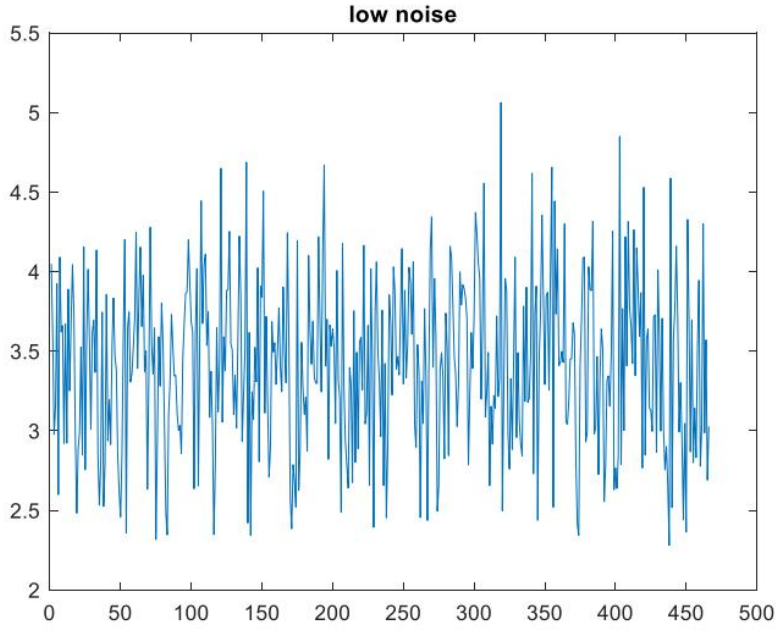


Figure 4 The output with low noise

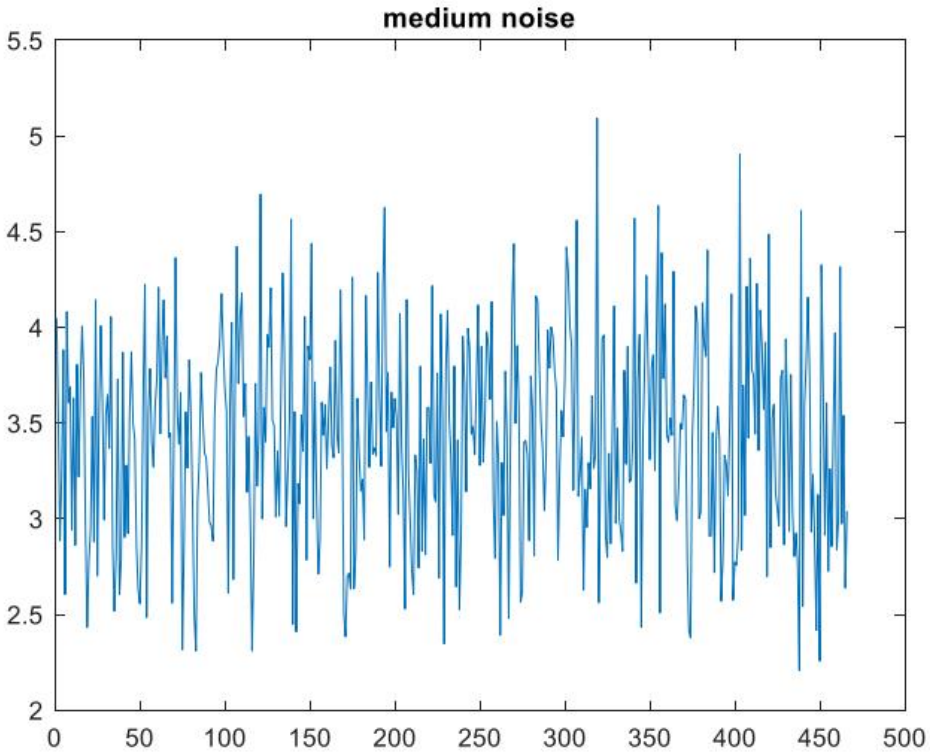


Figure 5 The output with medium noise

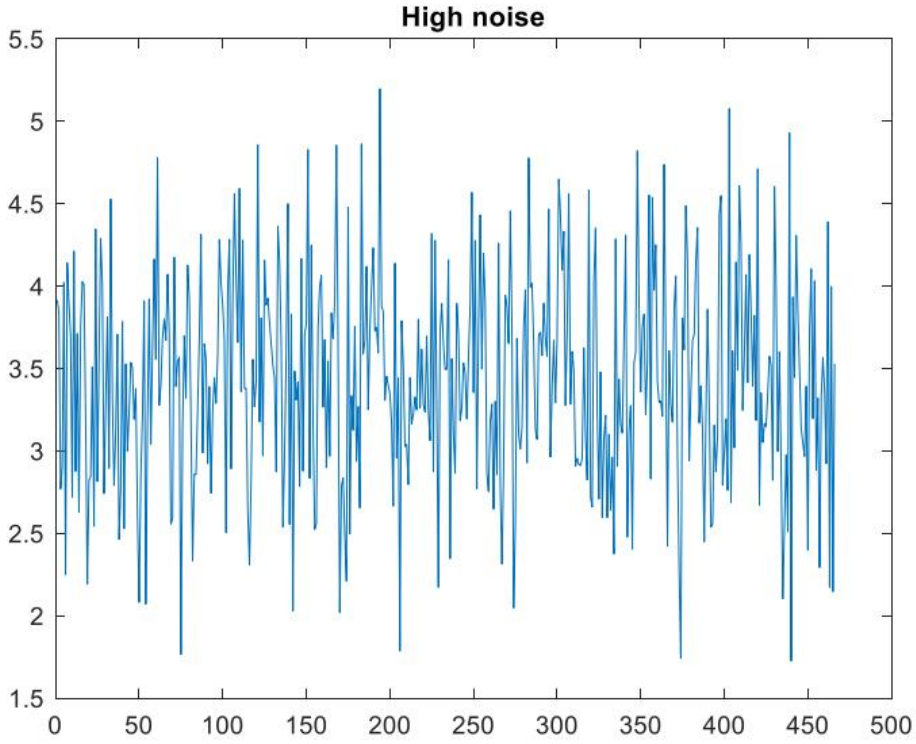


Figure 6 The output with high noise

Back propagation

This algorithm is an online learning method where data is presented to the network one by one, and the error for each data point is propagated back through the network. The weights are then updated using the gradient descent algorithm. An epoch is completed after going through all the data once.

During each iteration (epoch), the algorithm performs a feed-forward pass for each data point, calculates the estimation error, and then executes backpropagation for each data point to compute the weight and bias changes. Finally, the weights and biases are updated.

In this approach, the number of active layers, the number of neurons in each layer, the activation function for each neuron, and parameters such as the learning rate (η) are adjusted. The total number of epochs is also determined for training the network.

The optimal number of neurons in the hidden layer in neural networks is a crucial factor for achieving better performance. As the number of neurons increases, the training error gradually decreases, leading to better learning. However, having too many neurons can make the network overly complex, potentially reducing its ability to generalize to new, unseen data. A balance needs to be struck to avoid overfitting.

It's essential to note that with an increase in the number of neurons, the parameters of the network also increase. If the number of neurons becomes too large, resulting in more parameters than the available data points, the network may become unstable, leading to over-parameterization issues. In such cases, the test error may diverge from the training error, indicating instability.

The optimal number of neurons is the point at which further increases do not significantly improve test error or when the network becomes over-parameterized. It is crucial to ensure that the number of parameters doesn't exceed the number of data points to maintain the stability and generalization ability of the network. The maximum number of neurons in the hidden layer, considering 216 training data, can be calculated as follows.

$$M(P + 1) + M + 1 \leq Q \rightarrow 4M + M + 1 \leq 216 \rightarrow M < 43$$

The formula you provided seems to be incomplete. It mentions P, Q, and M as the number of network inputs, outputs, and neurons, respectively, but the relationship between them and the nature of the equation are not clear. Additionally, the text mentions that the number of neurons is unconventional and may lead to overfitting before the network reaches this neuron count.

$$M = 2 * (P + 1) + 1 = 2 * (3 + 1) + 1 = 9$$

The statement suggests that the suitable number of neurons should be between 9 and 43. To determine the appropriate number of neurons, the Mean Squared Error (MSE) is calculated by varying the number of neurons from 1 to 43. The Error Back-Propagation algorithm will be employed for learning, and the cost function for optimization is considered as follows:

$$MSE: J = \frac{\sum_{q=1}^{Q} J_q}{Q},\ J_q = \frac{1}{2}\left(y_q - \hat{y}_q\right)^2$$

The provided statement indicates that the optimization problem is solved using the gradient descent method. Therefore, the criterion for selecting the optimal number of neurons is the minimum average mean squared error (MSE) for each structure during the optimal epoch.

Based on the mean squared error (MSE) on the validation set, the optimal number of neurons can be determined. Typically, with an increase in the number of neurons, the training error decreases, but the validation and test errors may increase. To ensure that the network is not sensitive to the initial conditions, each network has been trained 10 times with the same number of neurons.

The optimal network is selected by varying the number of neurons from 1 to 43, considering a soft measure of test and training errors (SEM). The parameters considered are epochs = 100 and learning rate $(\eta) = 0.01$. The MSE plot for test and training data without noise for different numbers of neurons is shown in the figure below.

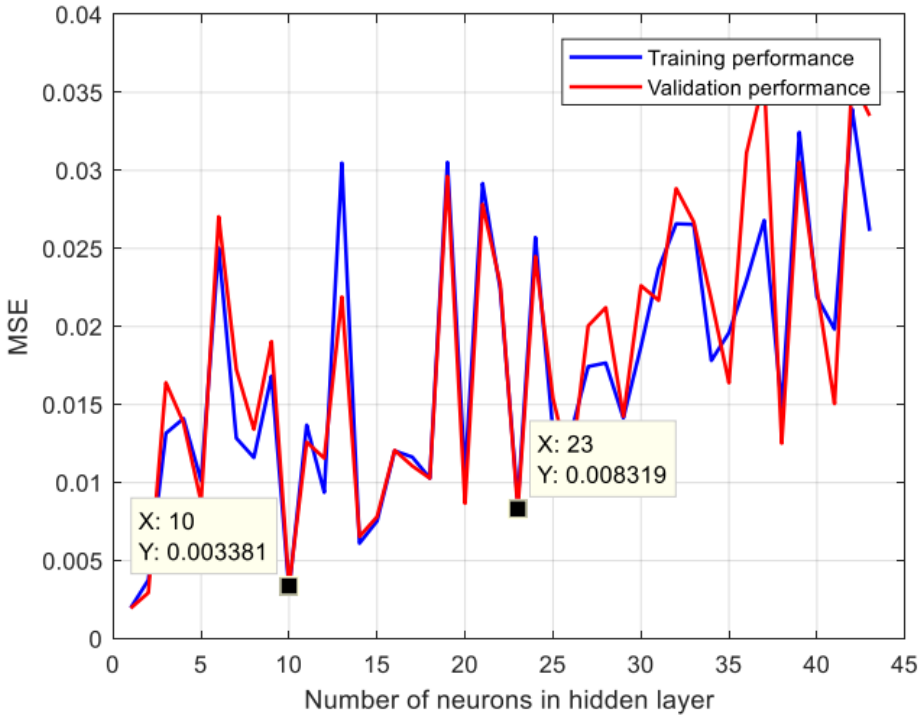


Figure 7 Error based on the count of neurons ***without noise***

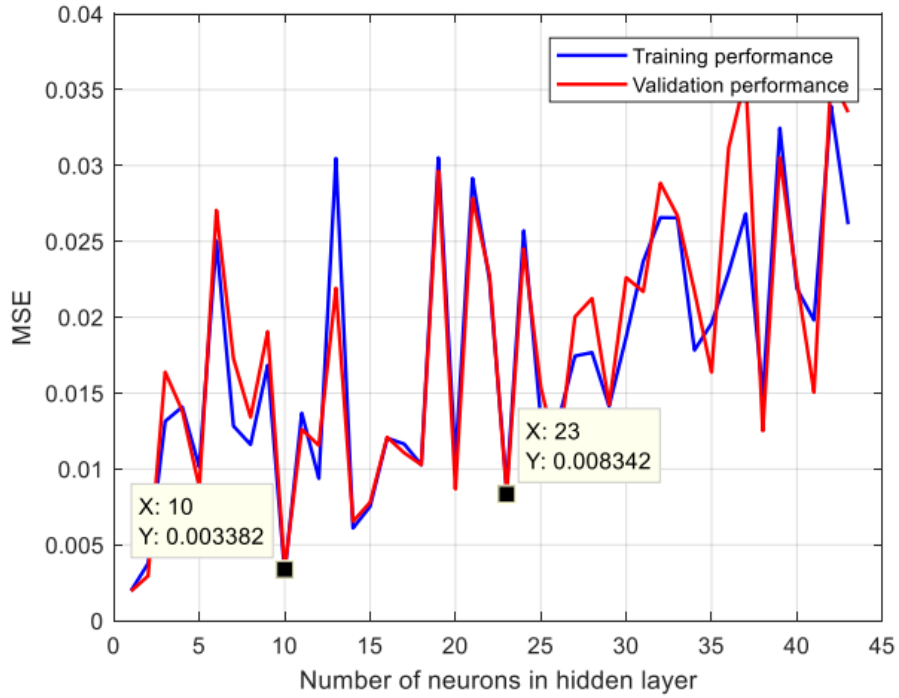


*Figure 8 Error based on the count of neurons **low noise***

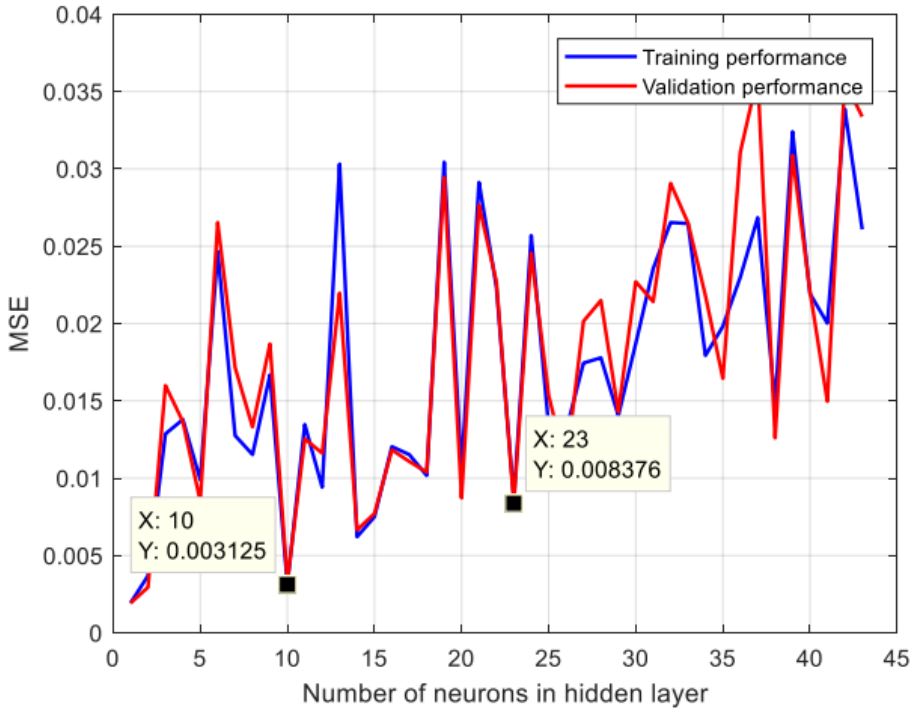


*Figure 9 Error based on the count of neurons **medium noise***

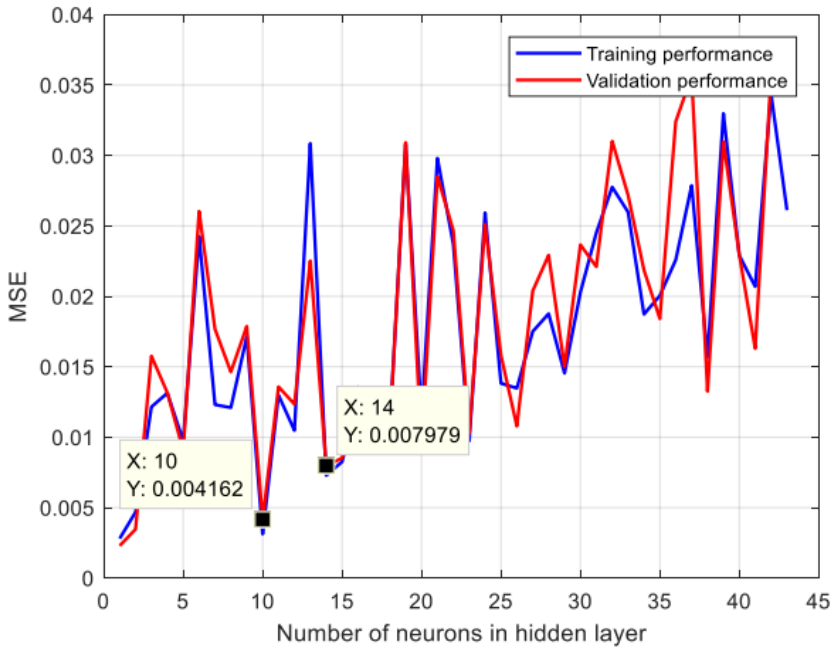


*Figure 10 Error based on the count of neurons **high noise***

Based on the figure and the mentioned points, it can be stated that the optimal number of neurons in all cases is approximately 10. The MSE plot for all cases is drawn, and the MSE value increases, indicating the learning of noise in the neural network, which is not desirable. In these cases, over-parameterization occurs from 23 neurons onward.

trainlm

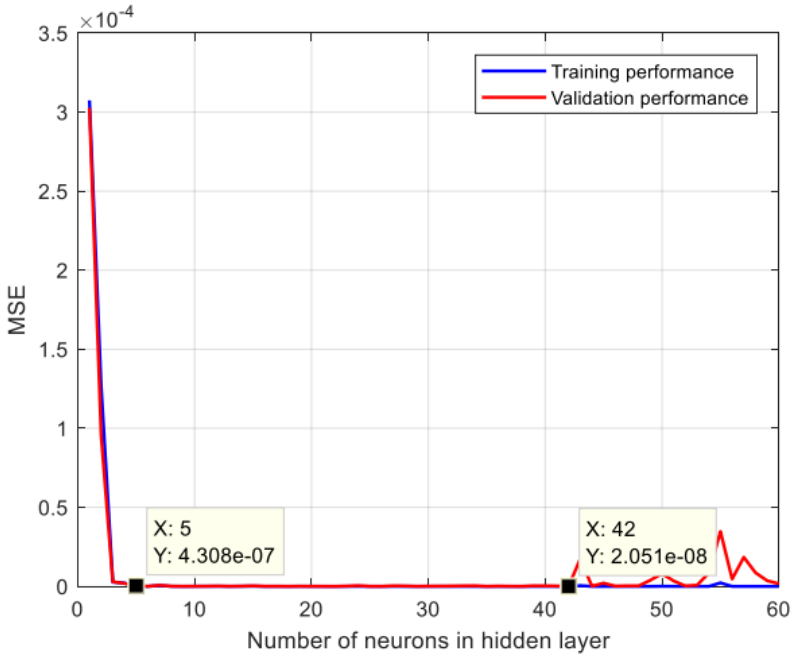


*Figure 11 error based on the count neurons in Levenberg–Marquardt algorithm **without noise***

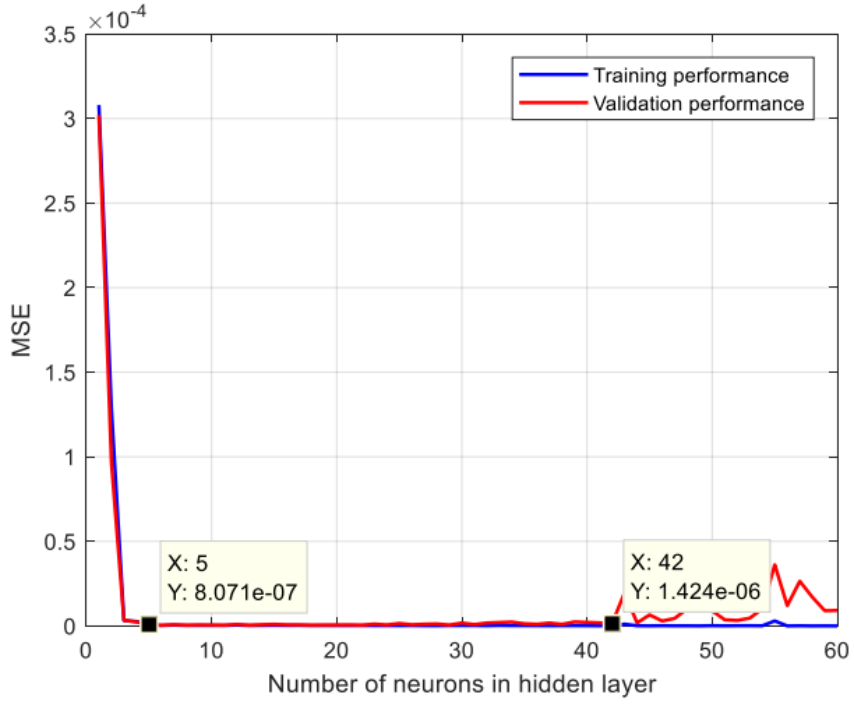


Figure 12 error based on the count neurons in Levenberg–Marquardt algorithm ***low noise***

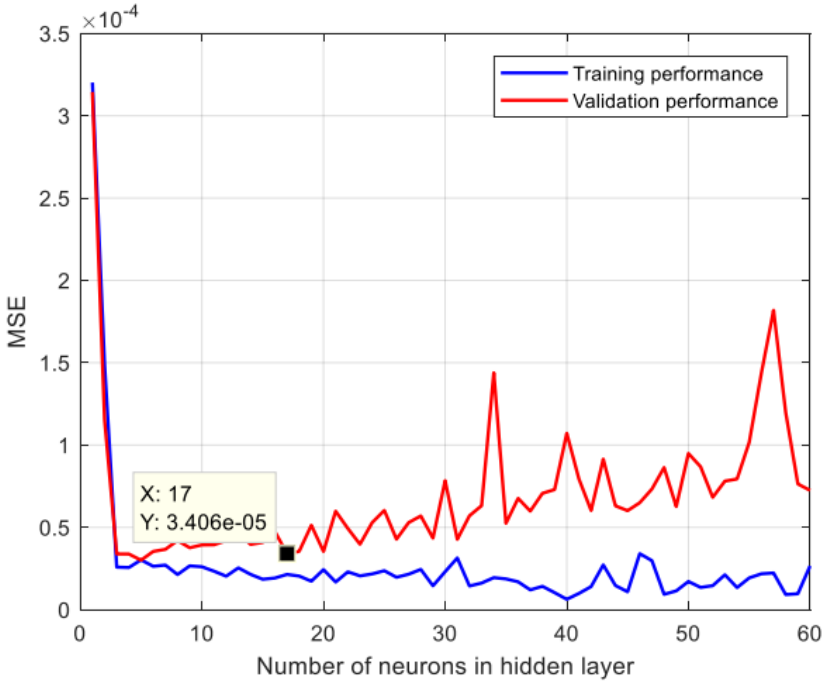


Figure 13 error based on the count neurons in Levenberg–Marquardt algorithm ***medium noise***

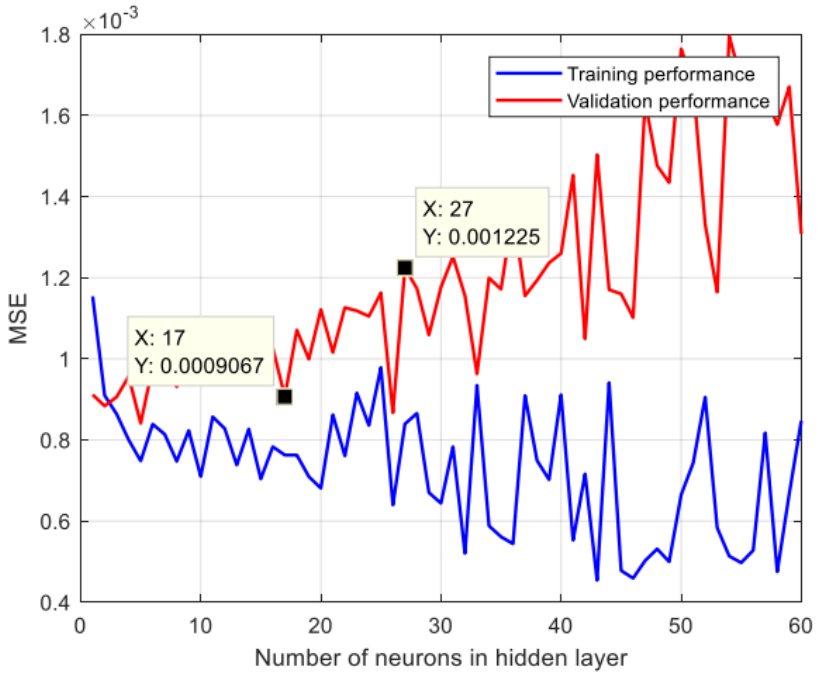


Figure 14 error based on the count neurons in Levenberg–Marquardt algorithm ***high noise***

Considering the figure and the mentioned points, it can be said that the optimal number of neurons in the first case is 5, in the second case is 5, and in the third and fourth cases is 17. The MSE plot for all cases is drawn, and the MSE value increases, indicating the learning of noise in the neural network, which is not desirable.

The phenomenon of "over-parametrization" refers to the situation where the number of parameters in a neural network is excessive relative to the available data. In the context of neural networks, it means having more parameters than needed, which can lead to issues such as increased training time, overfitting, and poor generalization.

As discussed earlier, when the number of parameters becomes too large, the network may become too complex, and it might start fitting the training data too closely, losing its ability to generalize well to unseen data. This is particularly evident in the over-parameterization phenomenon.

In the case of gradient descent, as the number of neurons increases (23, 23, 23, and then 14 in the mentioned cases), the test error also increases. This suggests that with an excessive number of neurons, the network becomes over-parametrized, leading to poorer performance on unseen data. It is crucial to strike a balance between having enough parameters to capture the underlying patterns in the data and avoiding excessive complexity that may hinder generalization.

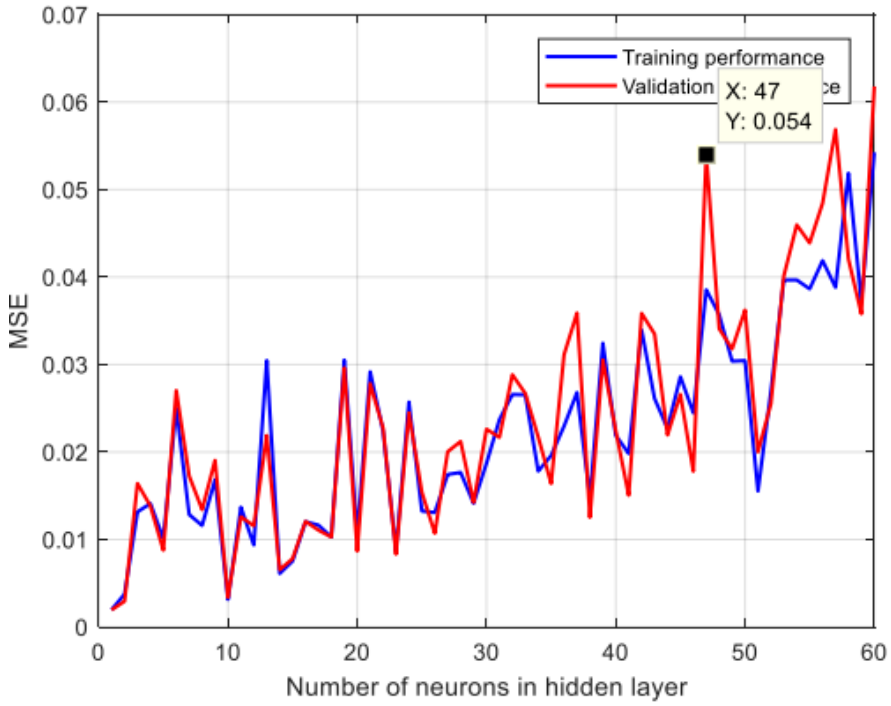


Figure 15 Number of neurons in hidden layers ***without noise***

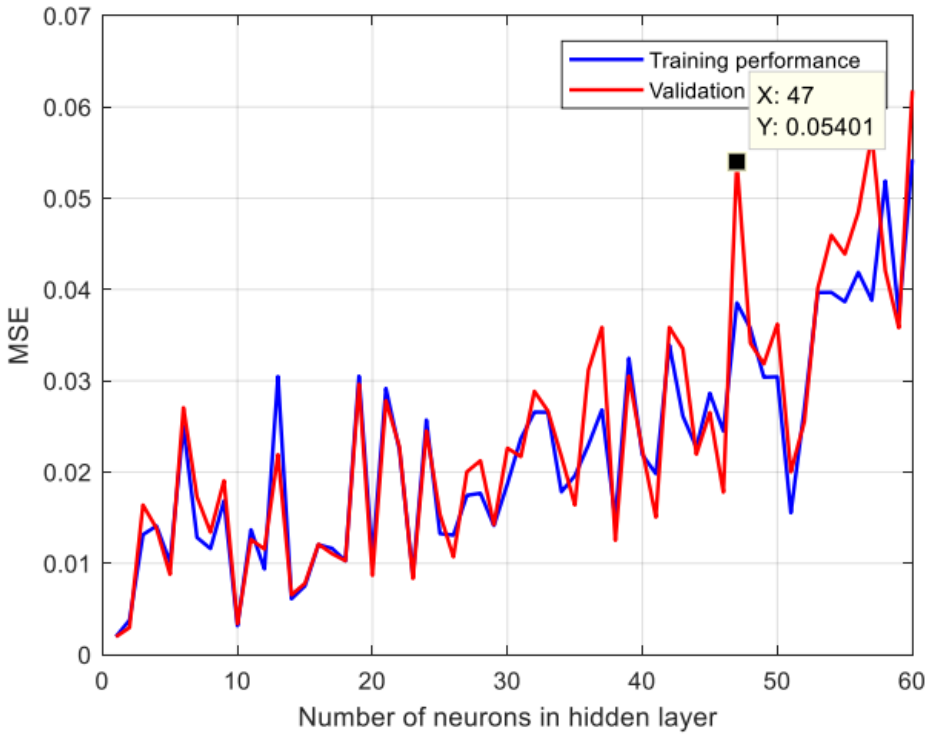


Figure 16 Number of neurons in hidden layers ***low noise***

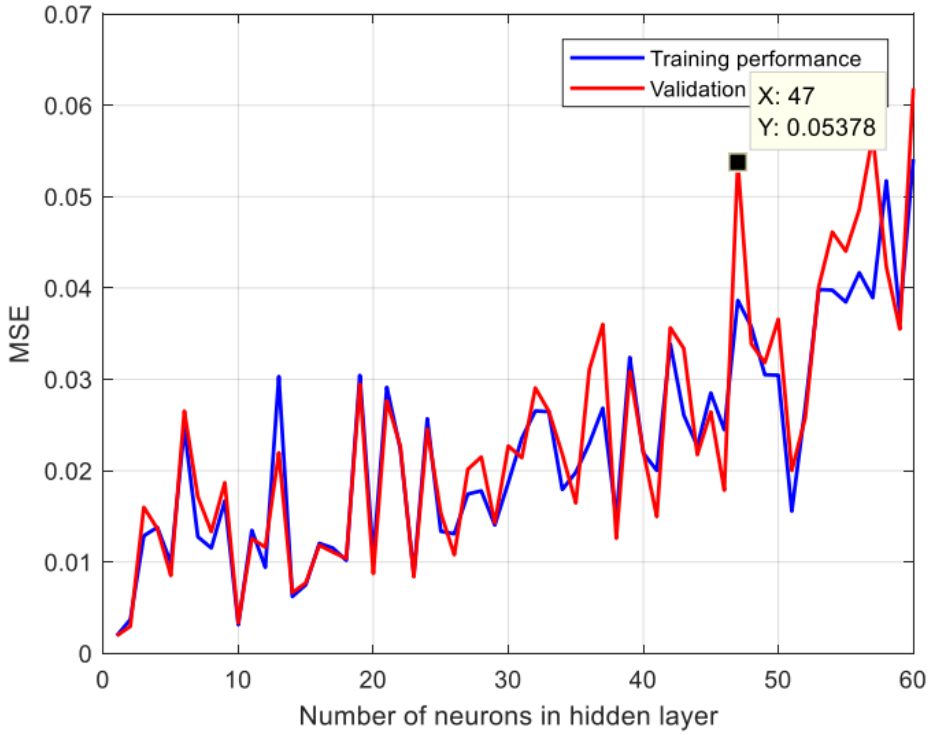


Figure 17 Number of neurons in hidden layers ***without noise***

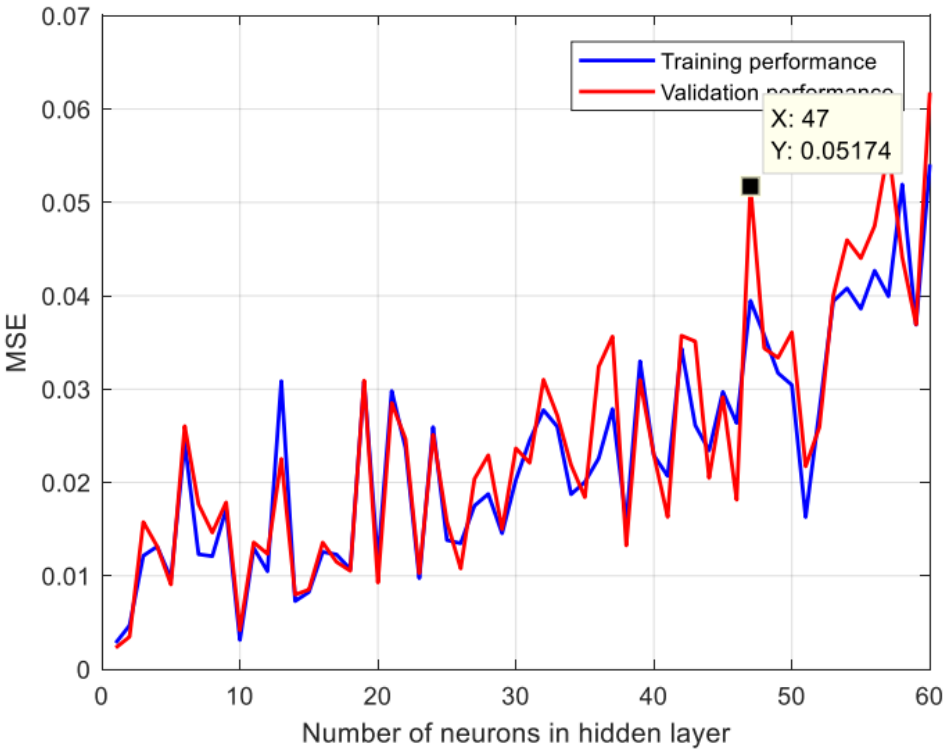


Figure 18 Number of neurons in hidden layers ***without noise***

As evident from the above figures, after the calculated permissible value, i.e., the number of 43 neurons, more pronounced peaks in the test error are observed, indicating the presence of the "over-parametrization" phenomenon.

In the Trainlm method, it is also observed that after the calculated permissible value of 43 neurons, the test error starts increasing with an upward trend, signaling the occurrence of over-parametrization

1.2, 1.3)

Overtraining and selection of optimal epoch:

Now we turn to the investigation of the overtraining phenomenon and the selection of the optimal epoch. With an increase in the number of training iterations in each stage, the parameter values are updated, and the training error decreases. However, it should be noted that when the network is trained excessively, it merely memorizes the training data and fails to generalize well to test data, leading to an increase in test error and a deviation from the training error. This phenomenon is referred to as overtraining. Therefore, the optimal epoch is the point before the overtraining phenomenon occurs, where the test error is minimized or undergoes minimal change.

Considering the optimal number of neurons obtained in the previous stage, we varied the number of epochs from 1 to 200. The MSE (Mean Squared Error) for both test and training data is plotted for different numbers of epochs, for datasets with no noise, moderate noise, and high noise, as shown in the following figures. (The number of neurons is set equal to the optimal number obtained in the previous stage.)

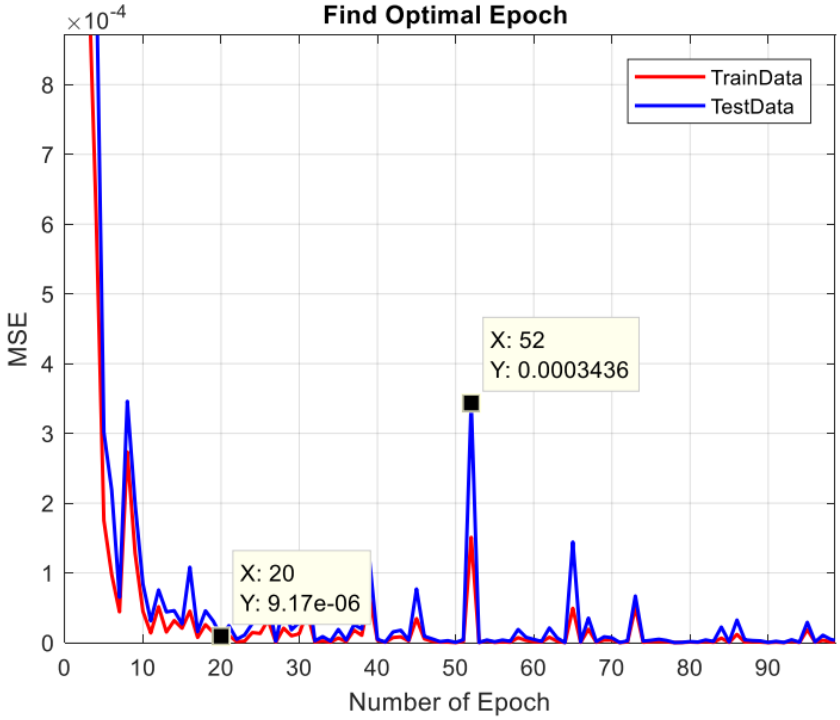


Figure 19 Error based on the count of neurons gradient descent method ***without noise***

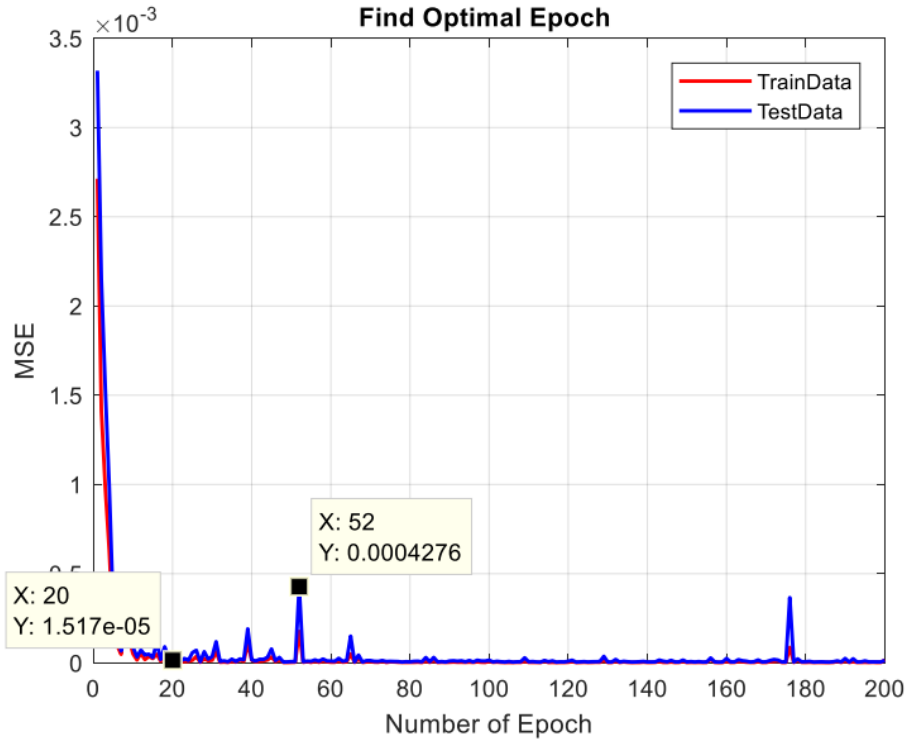


*Figure 20 Error based on the count of neurons gradient descent method **low noise***

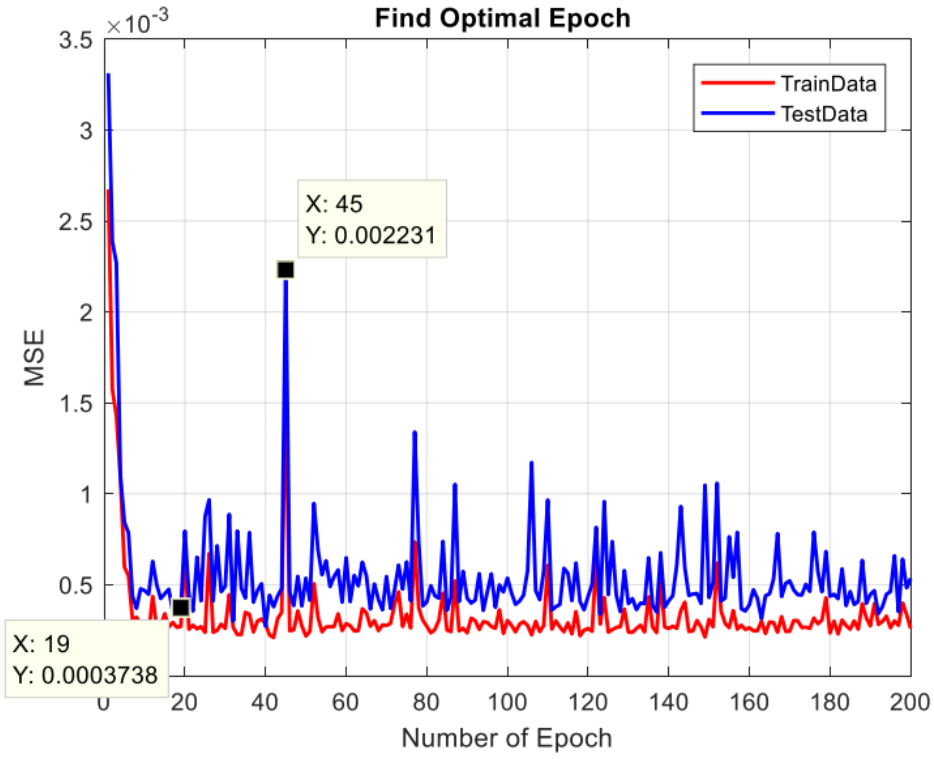


*Figure 21 Error based on the count of neurons gradient descent method **medium noise***

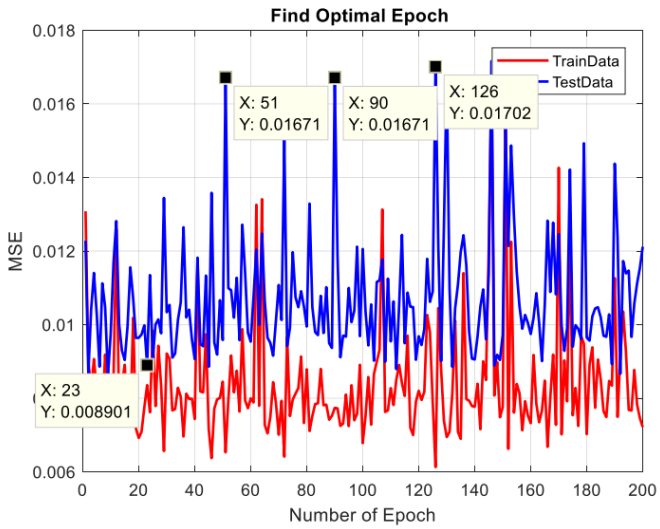


*Figure 22 Error based on the count of neurons gradient descent method **high noise***

As observed in the figures, with an increase in the number of epochs, the training error decreases. However, this reduction in error does not necessarily indicate the attainment of a better model. Despite the decrease in training data error, the test data error exhibits significant oscillations, reaching peaks at certain points. The most notable peaks occur at epoch numbers equal to 52 for datasets with no noise and weak noise, 45 for moderate noise, and various epoch numbers such as 51, 90, 26, 126, etc., for datasets with high noise. This is precisely where the overtraining phenomenon occurs.

Considering the points mentioned above and as indicated in the figures, the optimal number of training epochs for datasets with no noise and weak noise is 20, for datasets with moderate noise is 19, and for datasets with high noise is 23.

traingd

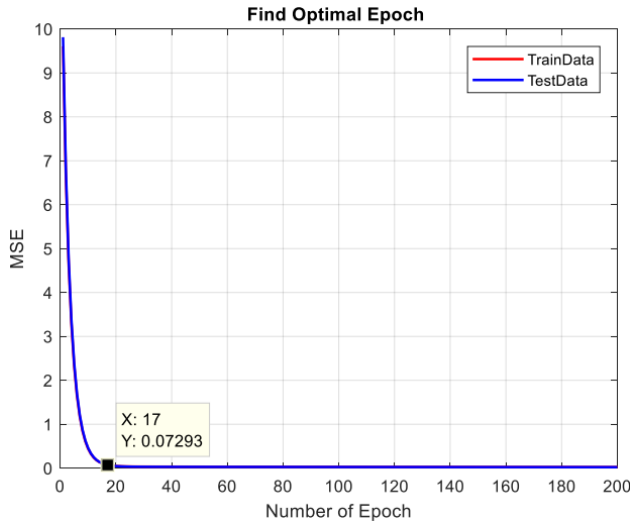


*Figure 23 Error-epoch plot-gradient descent **without noise***

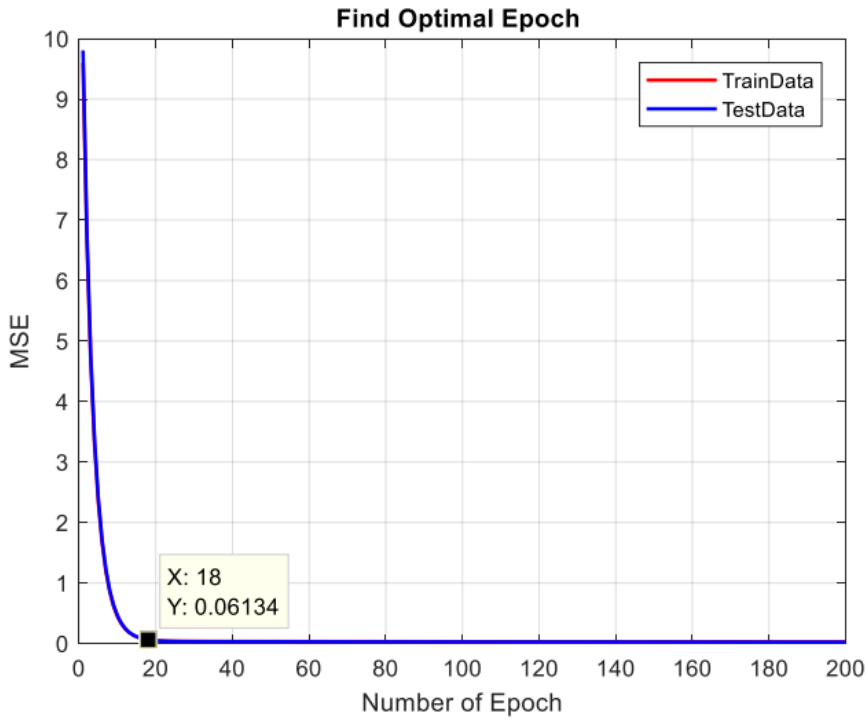


*Figure 24 Error-epoch plot-gradient descent **low noise***

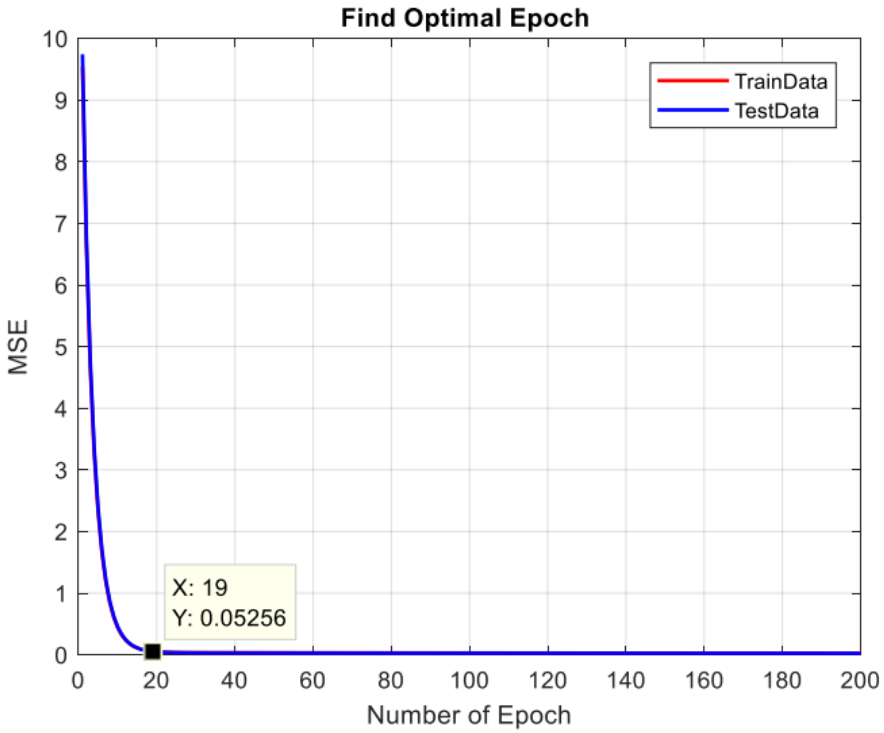


*Figure 25 Error-epoch plot-gradient descent **moderate noise***

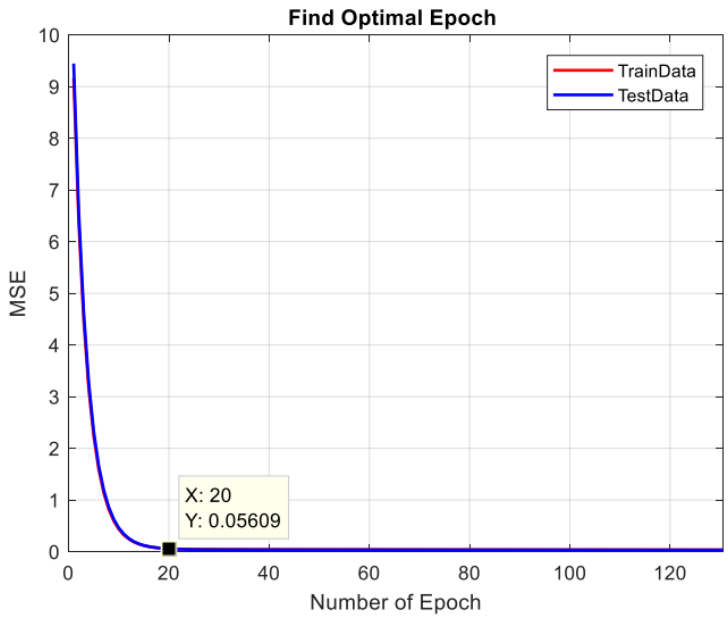


*Figure 26 Error-epoch plot-gradient descent **high noise***

Validation:

Finally, using the 125 data points that were set aside for validation, we evaluate the neural network. In the trainlm method, we display the output (as the results were not satisfactory with the gradient descent method). Additionally, we plot the error correlation on the validation data.

Without noise

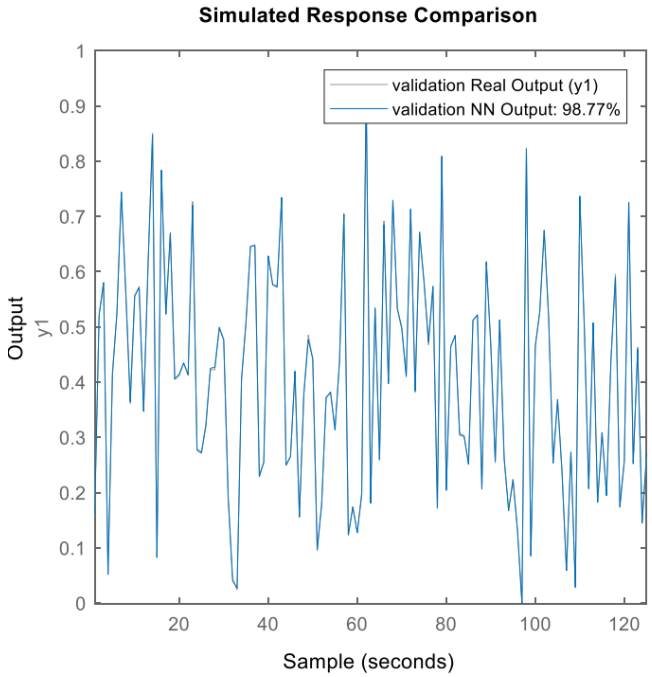


*Figure 27 the actual and estimated outputs, **without noise Validation***

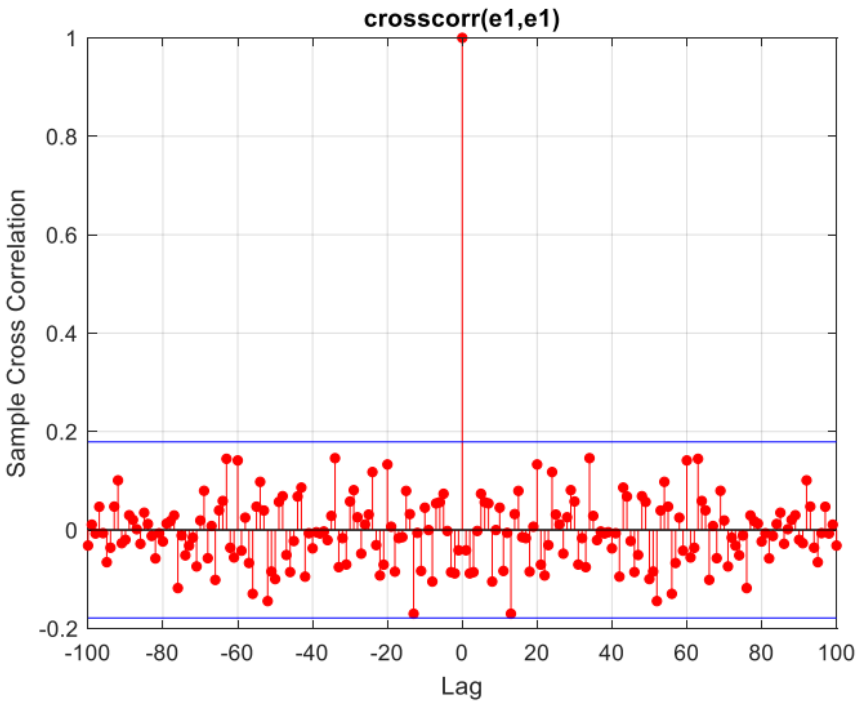


Figure 28 Error correlation ***without noise validation***

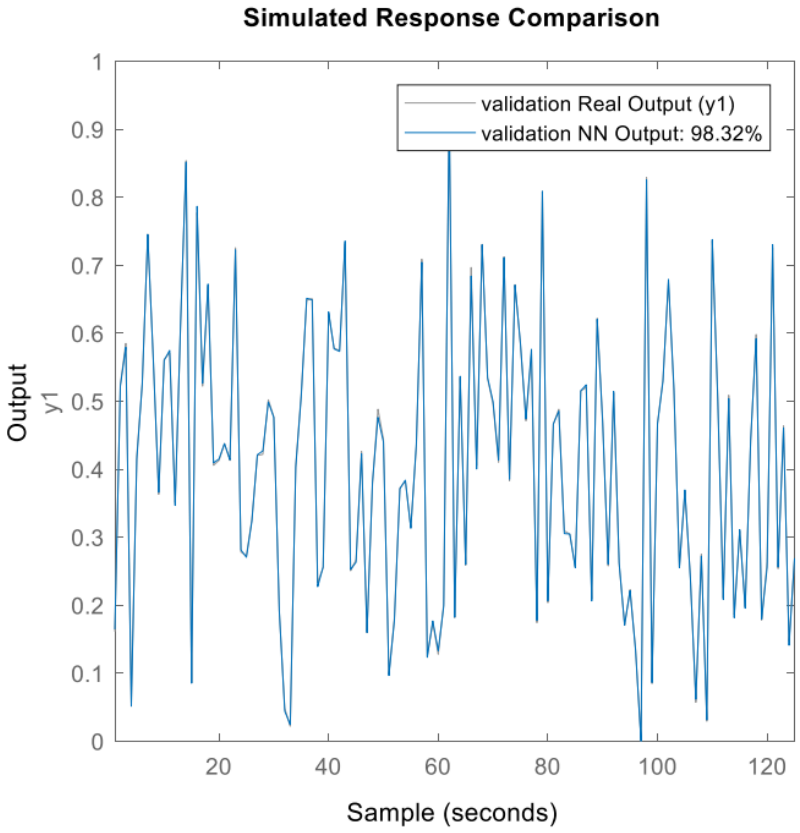


Figure 29 the actual and estimated outputs, ***low noise Validation***

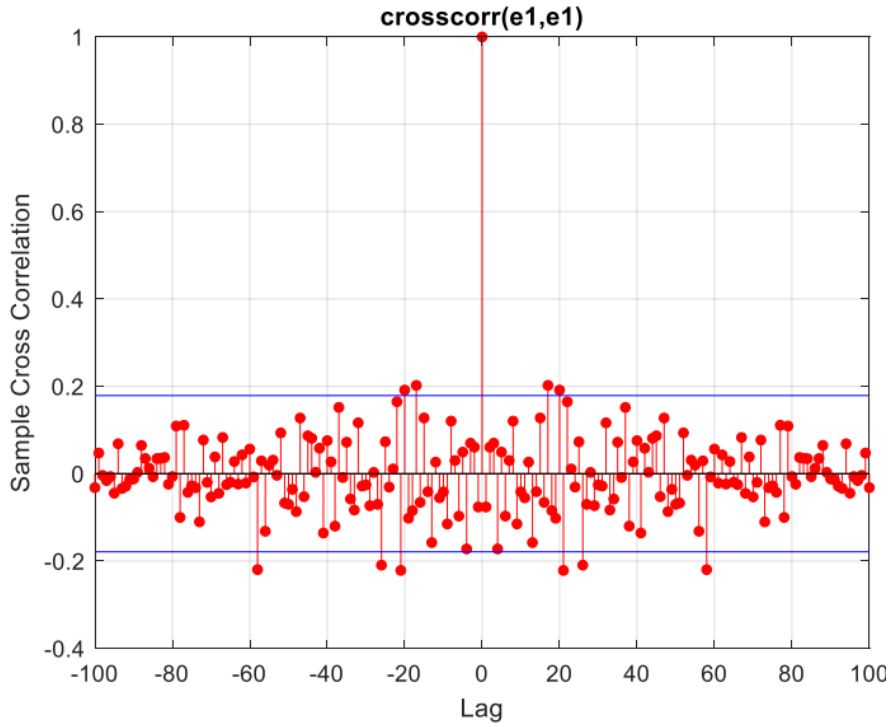


Figure 30 Error correlation ***low noise validation***

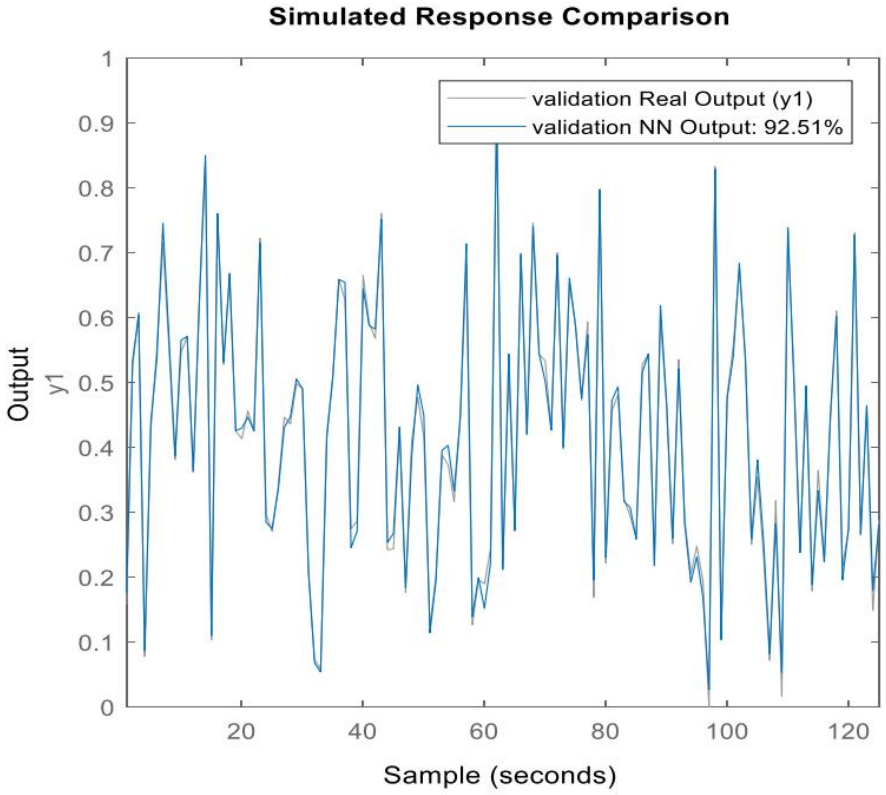


Figure 31 the actual and estimated outputs, ***moderate noise Validation***

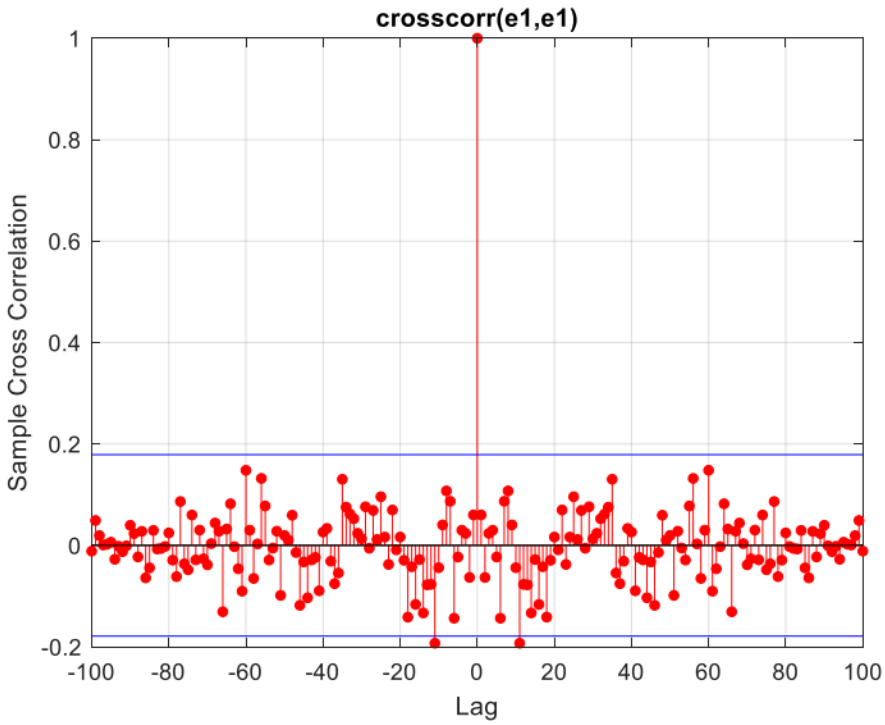


*Figure 32 Error correlation **moderate noise validation***

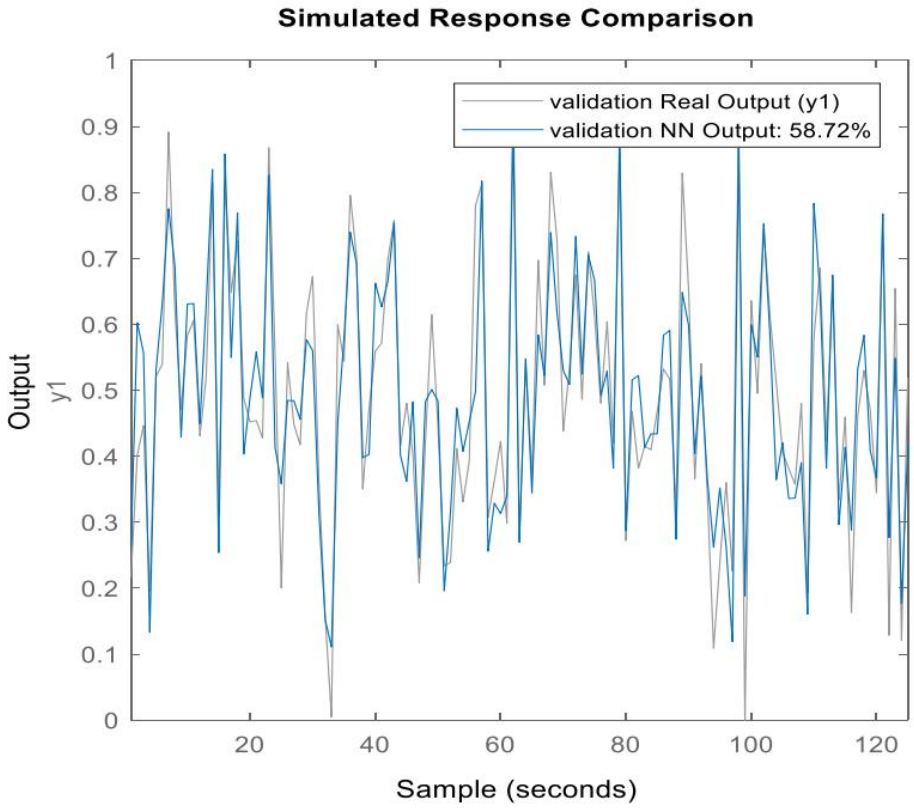


*Figure 33 the actual and estimated outputs, **high noise Validation***

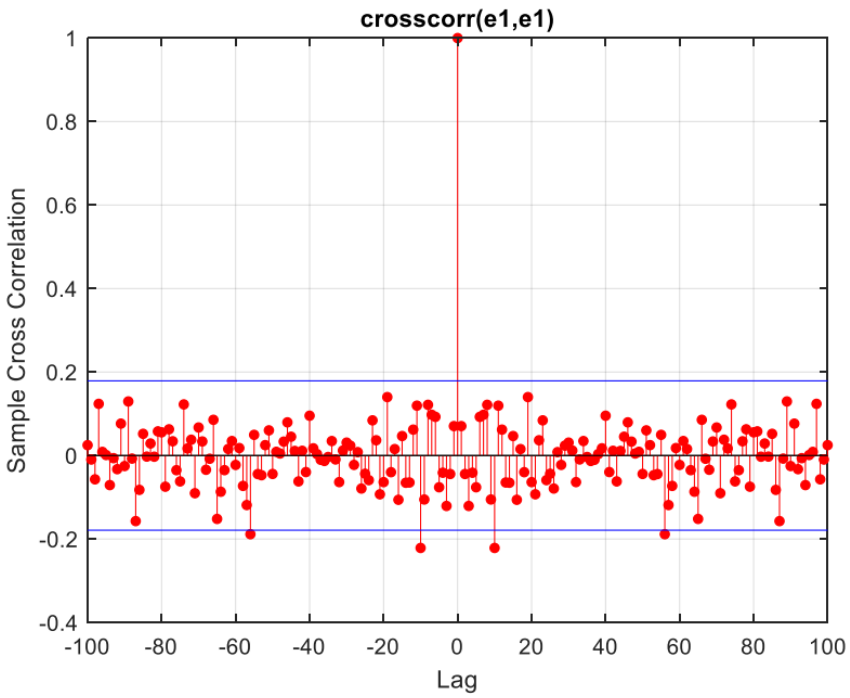


Figure 34 Error correlation ***high noise validation***

For each of the three noise levels, whitening of the error has occurred, and important dynamics have been captured by the model. It can be observed that as the noise increases, the fitting percentage decreases. In the case of noise, the larger the noise variance, the worse the estimation becomes, which is due to the network learning the noise.

1.7)

Investigating the optimal network with other training methods:

In this section, we will try to find the best optimization algorithm according to the problem's request. For this purpose, we need to plot the mean square error (MSE) for test and training data against the number of epochs. We will examine the convergence slope of the two plots. The optimization algorithm with a faster convergence indicates a more suitable optimization algorithm.

As observed, this algorithm converges much faster compared to the previous algorithm (standard gradient descent) and experiences a rapid reduction in the slope in the early epochs.

Gradient Descent (GD):

The Gradient Descent with Adaptive Learning Rate (GD) is a variation of the standard gradient descent algorithm. In this approach, the direction of movement towards the next point is opposite to the gradient, and there is a high probability of falling into local minima. The fundamental principle of the GD algorithm is similar to the standard gradient descent, with the key difference being that the learning rate is computed adaptively in this algorithm.

Quasi-Newton:

The quasi-Newton training algorithm employs an approximation to the Hessian matrix, making computations more manageable and speeding up the convergence process. It is closely related to the Levenberg-Marquardt algorithm, an extension of the Gauss-Newton training algorithm. The Levenberg-Marquardt algorithm introduces an additional term, similar to what is defined in ridge regression for linear least squares estimation, to prevent the Jacobian matrix from becoming singular.

When close to the optimal point, the value of the additional term is reduced, and the algorithm behaves similarly to the Gauss-Newton method. As the distance from the optimal point increases, the value of this term is increased, and the algorithm approaches the steepest descent method. Empirically, this training algorithm yields favorable results, and simulation results show that the MSE curve decreases rapidly from the early epochs.

Using the training methods TRAINLM, TRAINGD, and TRAINBFG, the network is trained, and you can observe the error results and fitness percentage on the validation data in Table 1.

Table 1 The statistics of the test data on different training method

TRAINBFG	TRAINLM	TRAINGD	Training Method
77.66	98.77	59.4	Fitness percentage
0.0022	0.0000066	0.0956	MSE_Test

As evident from the table, the LM training method has the least error and the highest fitness percentage. Therefore, LM is the best training method. The Gauss-Newton algorithm has performed better than the gradient descent on this data.

1.4)

Perceptron Two-Layer

In this case, the network structure is similar to the function structure with two inputs and one output. The network structure is considered as two layers. According to the question, the number of neurons in the first hidden layer is set to 5 and kept constant. To find the optimal number of neurons for the second hidden layer, we need to gradually increase the number of neurons and determine the optimal number based on the obtained error for training and testing data. The characteristics that this optimal network should have are the same as those mentioned in the problem statement. We will use the Backpropagation learning algorithm for training. Here, using the available functions such as newff in the Neural Network Toolbox of MATLAB, we first create a two-layer MLP neural network with 5 neurons in the first layer.

The optimal number of neurons in the second layer, assuming the number of neurons in the first layer is fixed at 5, is as follows:

As you may know, in neural networks, adding an intermediate layer will increase computations, but it will lead to better performance. In this section, since we are adding one extra layer, the total number of neurons should not exceed the number of neurons in a single-layer MLP network. The MSE (Mean Squared Error) plot for training and testing data without noise, with weak and

moderate noise, and with significant noise using the gradient descent method is shown in the following figures.

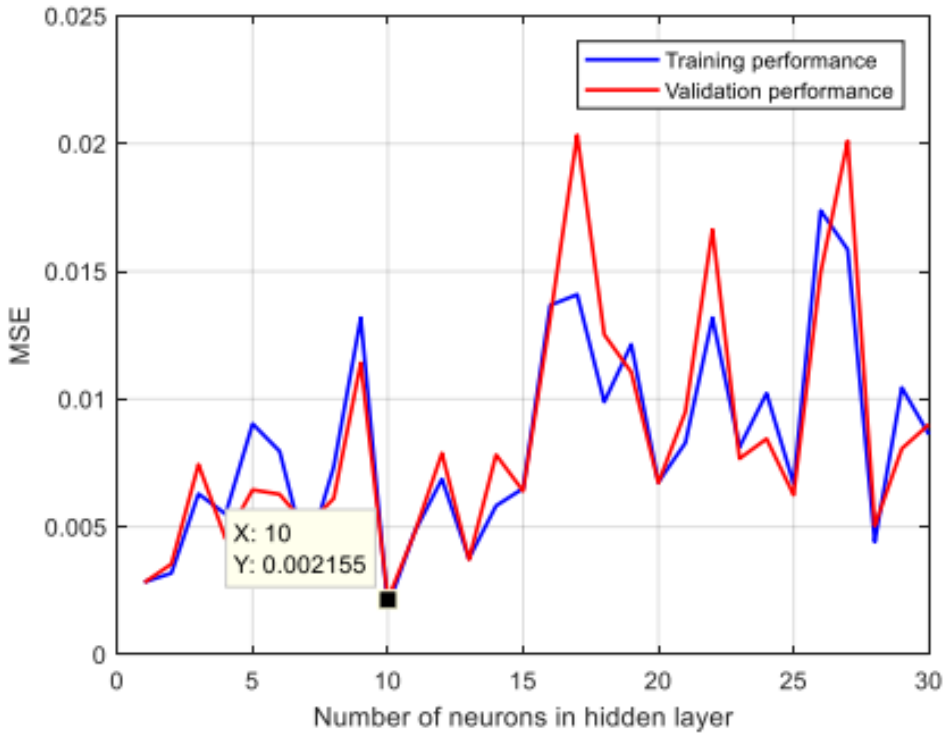


*Figure 35 error based on the number of neurons in hidden layer-**without noise***

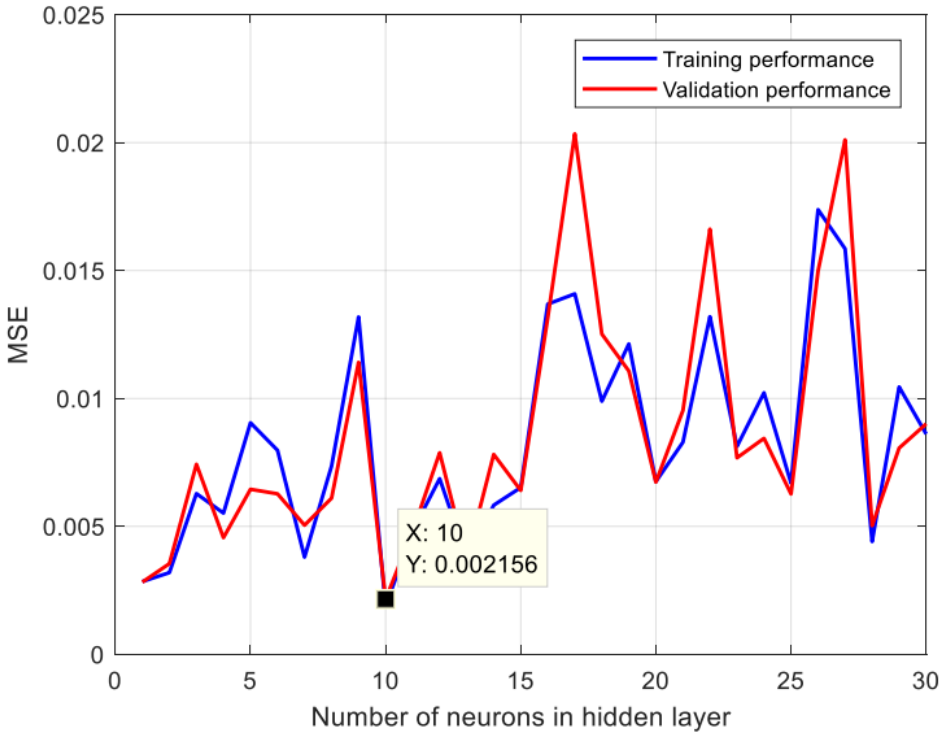


*Figure 36 error based on the number of neurons in hidden layer-**low noise***

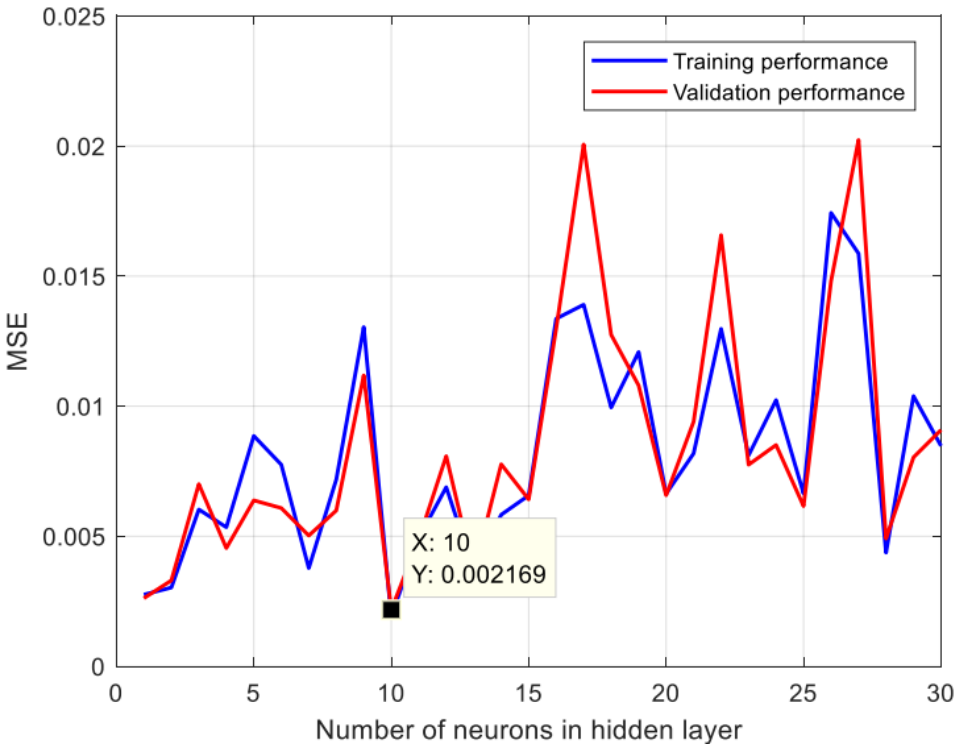


*Figure 37 error based on the number of neurons in hidden layer-**moderate noise***

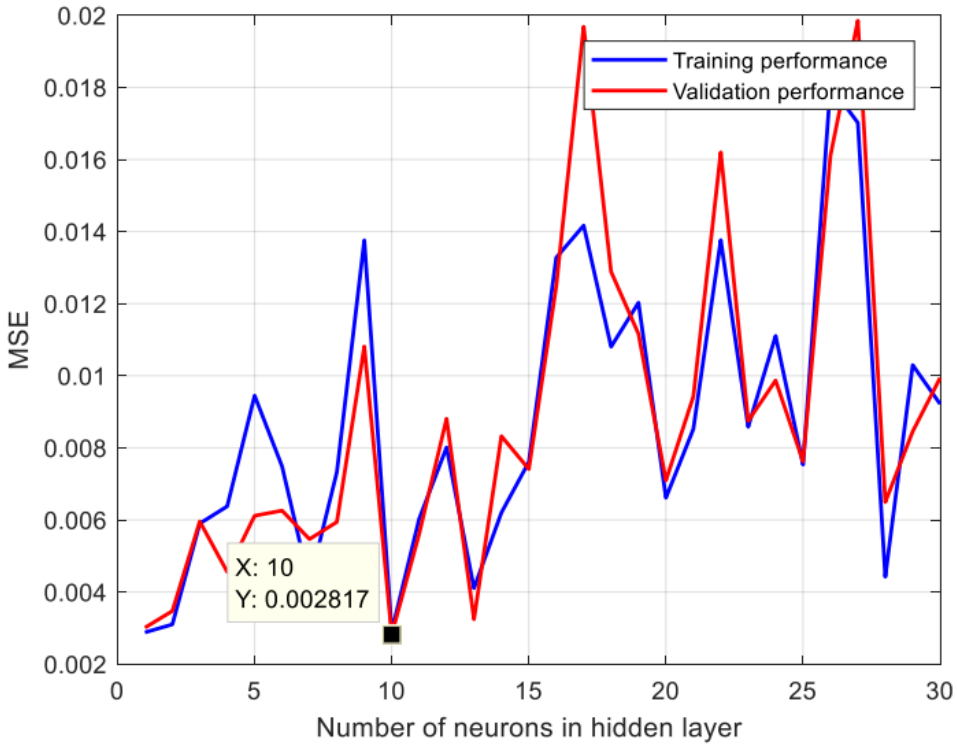


*Figure 38 error based on the number of neurons in hidden layer-**high noise***

In almost all cases, around 10 neurons have been optimal, and approximately beyond 15 neurons, the overparametrization phenomenon occurs.

Trainlm:

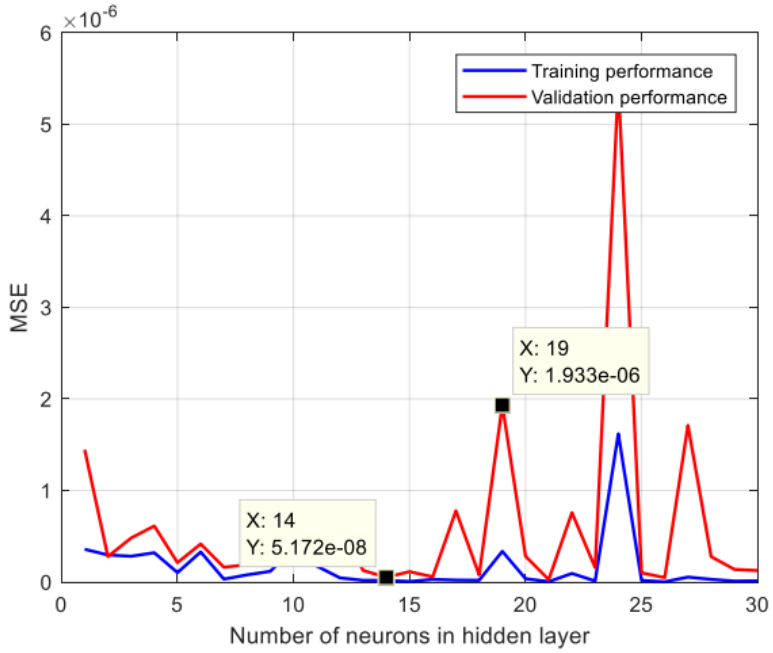


*Figure 39 error based on the number of neurons in hidden layer Levenberg–Marquardt **without noise***

To adhere to the simplicity principle, you can choose 5 neurons as well. Overparametrization phenomenon occurs approximately beyond 19 neurons.

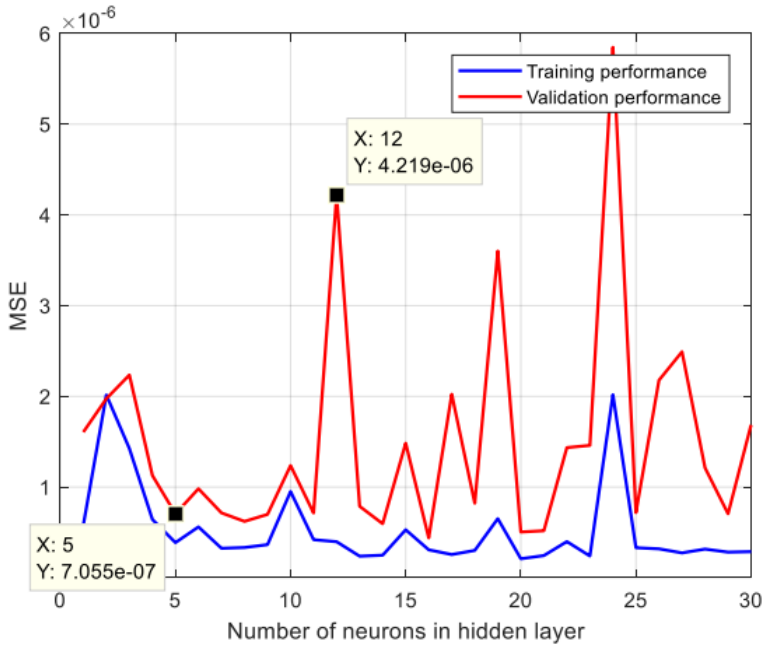


Figure 40 error based on the number of neurons in hidden layer Levenberg–Marquardt ***low noise***

You can choose 5 neurons as well. The overparametrization phenomenon occurs approximately beyond 12 neurons.

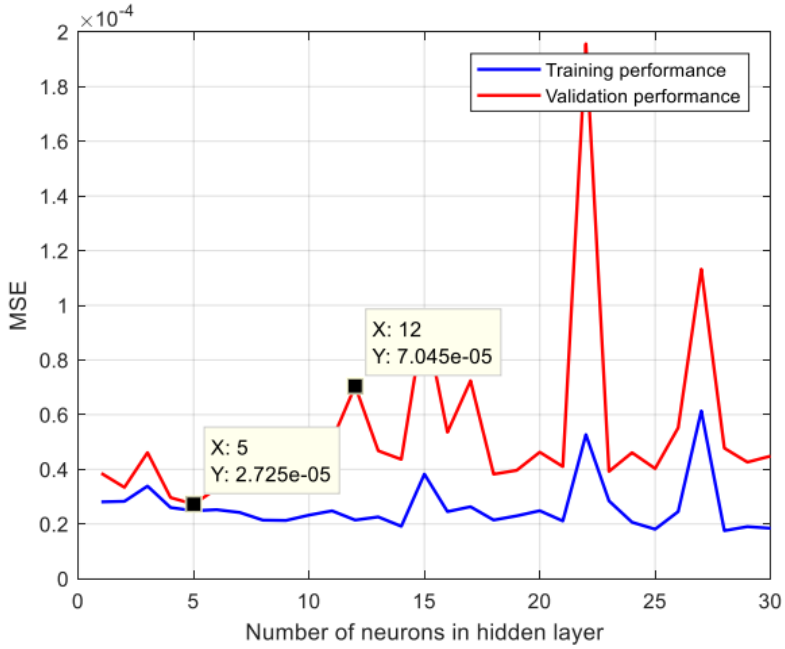


Figure 41 error based on the number of neurons in hidden layer Levenberg–Marquardt ***moderate noise***

You can also choose 5 neurons. The overparametrization phenomenon occurs approximately beyond 12 neurons.

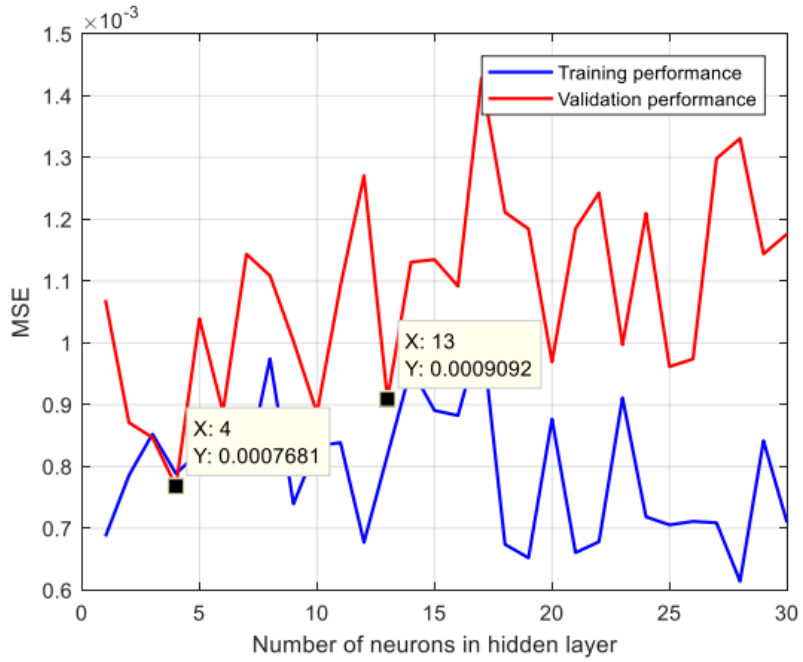


Figure 42 error based on the number of neurons in hidden layer Levenberg–Marquardt ***high noise***

You can also choose 4 neurons. The overparametrization phenomenon occurs approximately beyond 13 neurons.

1.5)

Some parts have been done in the previous sections, Like the effect of different noises and different training algorithms

Choosing optimized Epoch

The effect of different algorithms will be examined

To achieve this, the number of neurons in the first hidden layer is set to 5, the number of neurons in the second hidden layer is set to 10, and the number of training iterations is varied from 1 to 200. The MSE plots for testing and training data without noise, with moderate noise, and with significant noise for different numbers of training iterations are shown in the following figures.

Gradient descent method:

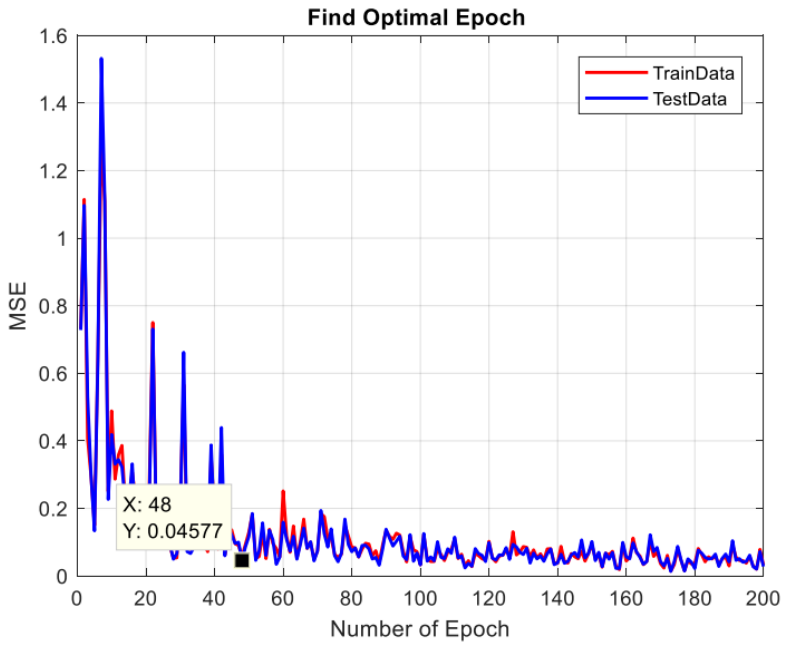


*Figure 43 Error-epoch two layer gradient descent **without noise***

Approximately, 50 epochs are optimal, and overtraining occurs beyond 60 epochs.

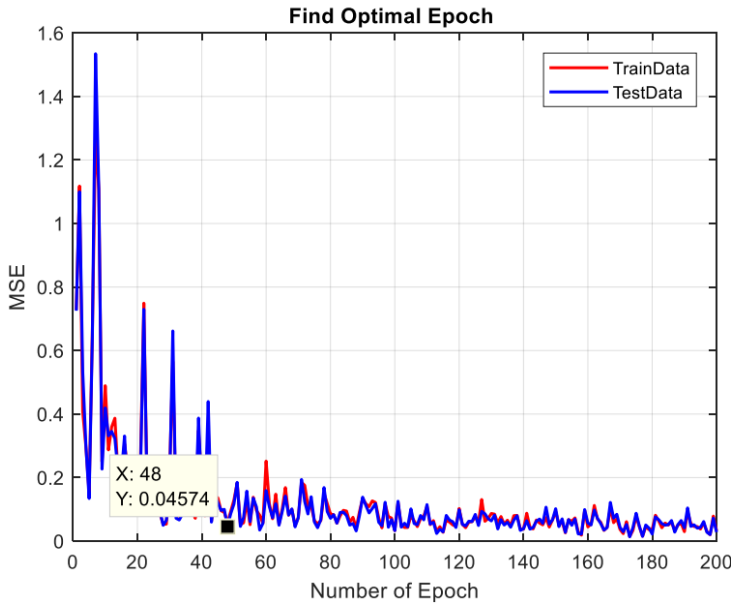


*Figure 44 Error-epoch two layer gradient descent **low noise***

Approximately, 50 epochs are optimal, and overtraining occurs beyond 60 epochs.

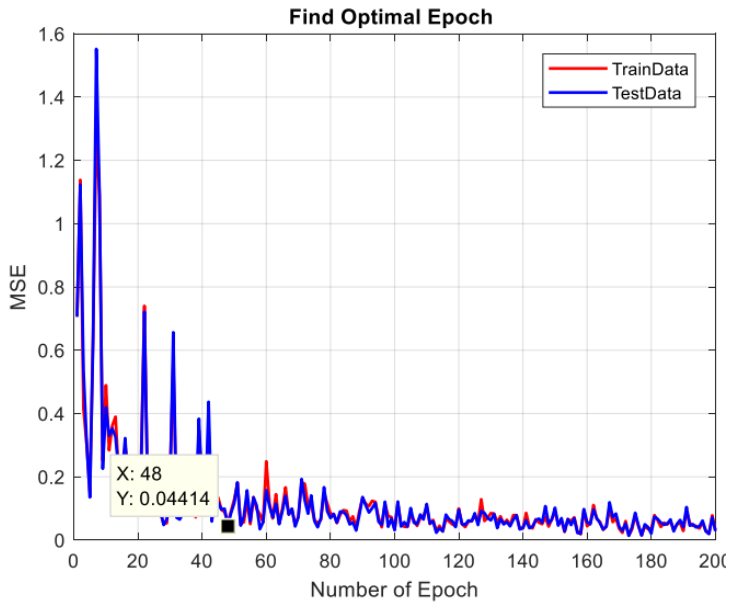


Figure 45 Error-epoch two layer gradient descent ***moderate noise***

Approximately, 50 epochs are optimal, and overtraining occurs beyond 60 epochs.

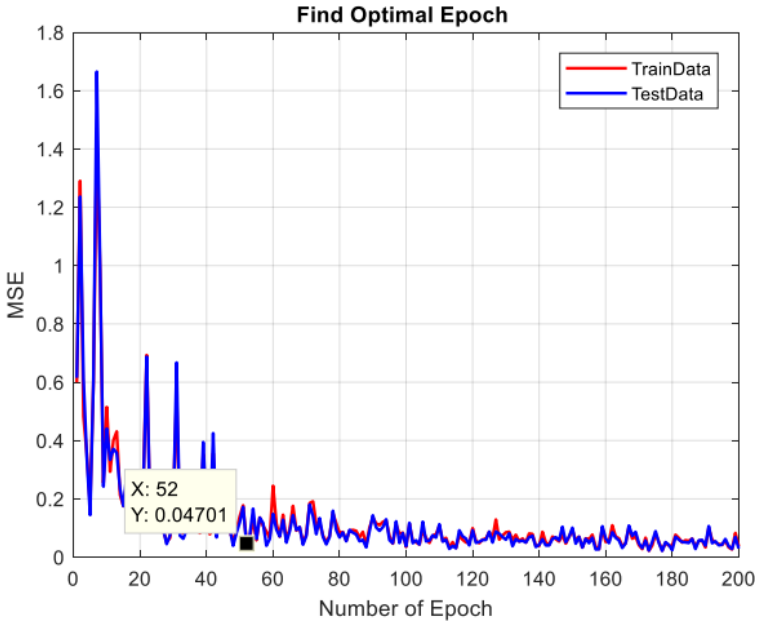


Figure 46 Error-epoch two layer gradient descent ***high noise***

Approximately, 52 epochs are optimal, and overtraining occurs beyond 60 epochs.

trainlm:

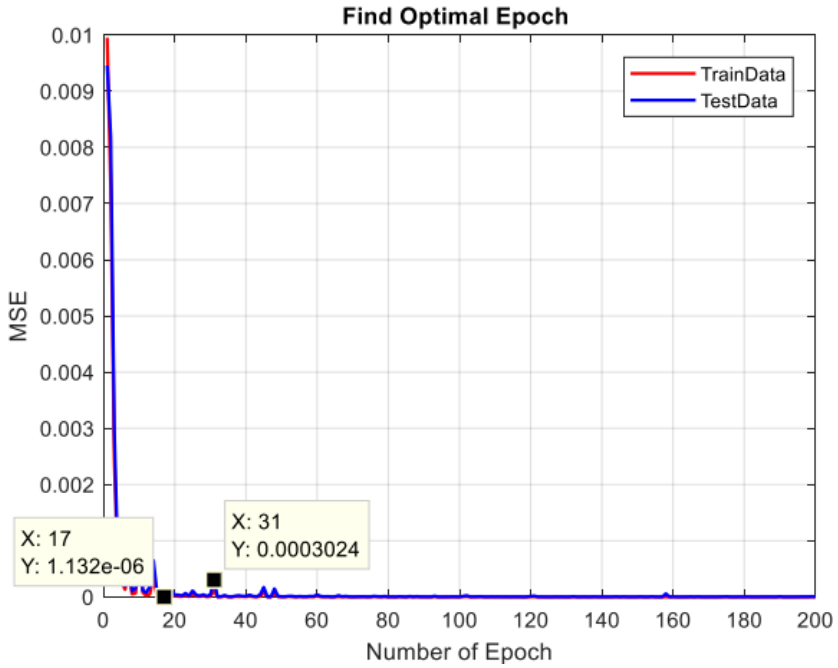


Figure 47 Error-epoch two-layer network Levenberg–Marquardt ***without noise***

Approximately, 17 epochs are optimal, and overtraining occurs beyond 31 epochs.

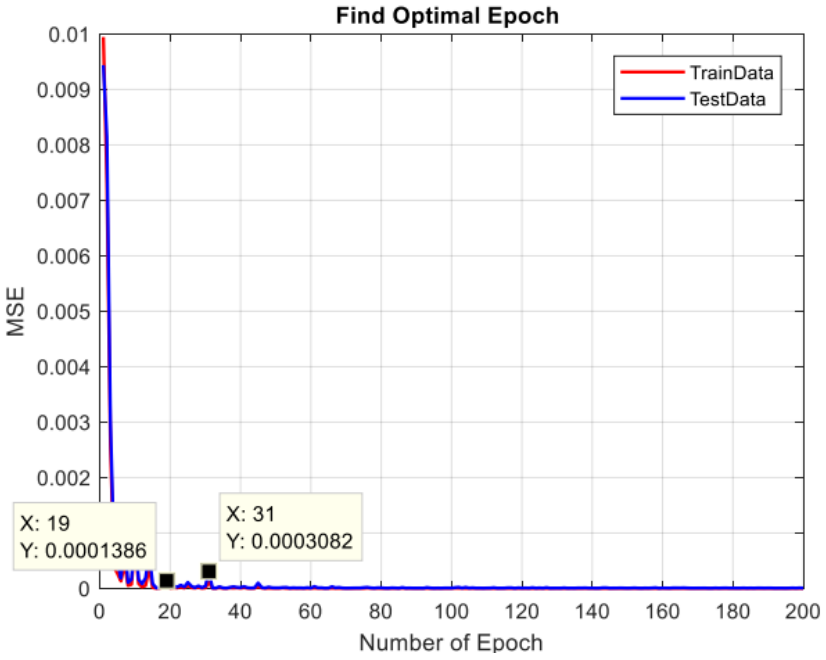


Figure 48 Error-epoch two-layer network Levenberg–Marquardt ***low noise***

Approximately, 19 epochs are optimal, and overtraining occurs beyond 31 epochs.

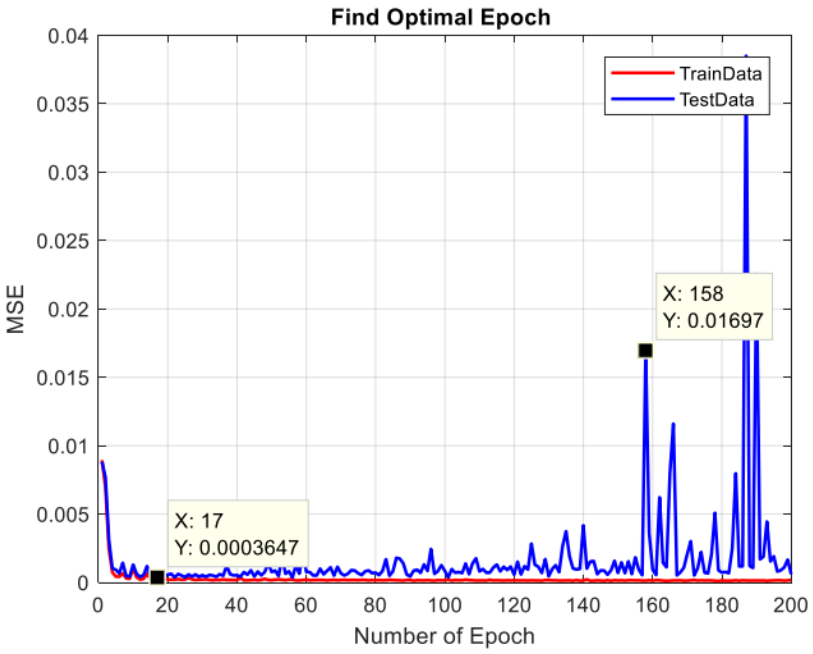


Figure 49 Error-epoch two-layer network Levenberg–Marquardt ***moderate noise***

Approximately, 17 epochs are optimal, and overtraining occurs beyond 158 epochs.

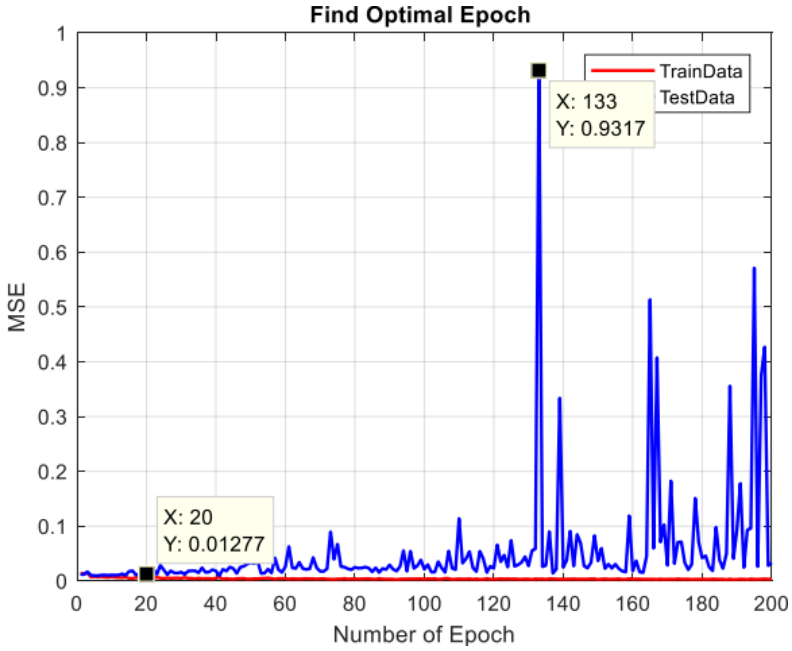


Figure 50 Error-epoch two-layer network Levenberg–Marquardt ***high noise***

Approximately, 20 epochs are optimal, and overtraining occurs beyond 133 epochs.

Validation:

Finally, using the 125 data points that we set aside as validation data, we perform the validation of the neural network. We show the output for the trainlm method (as the results were not satisfactory with the gradient descent method). We also plot the correlation of errors on the validation data.

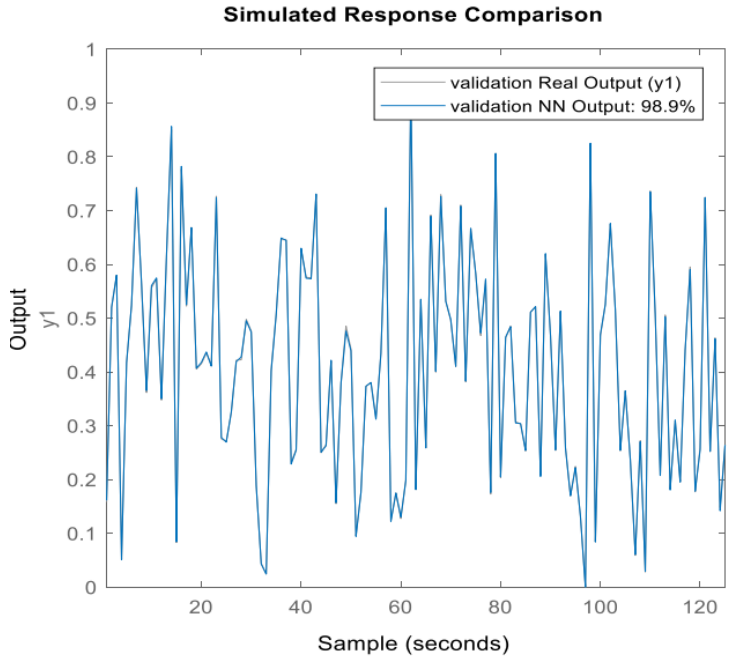


Figure 51 the actual system and estimated system outputs-validation data- ***without noise***

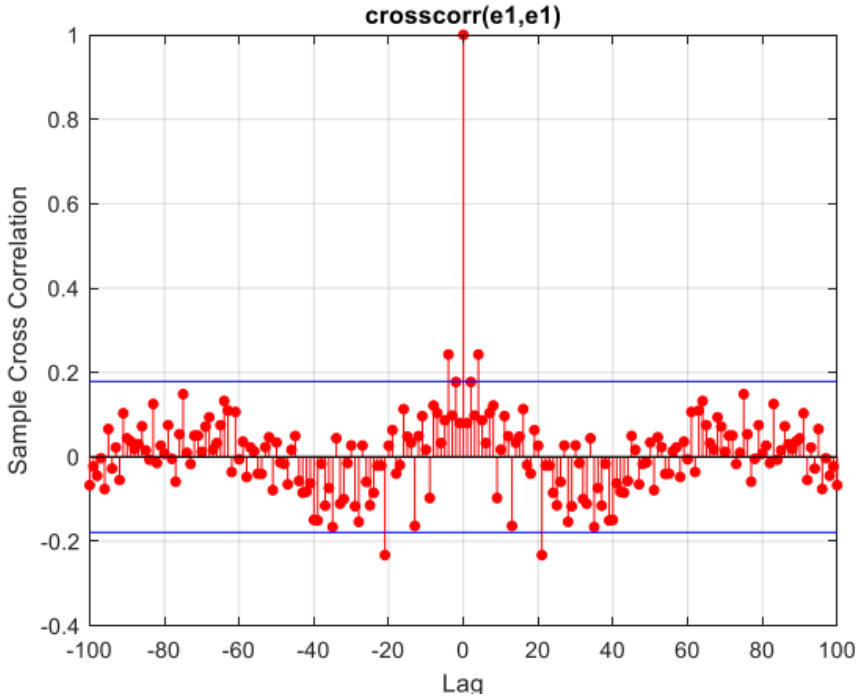


Figure 52 Error cross-correlation – ***without noise***

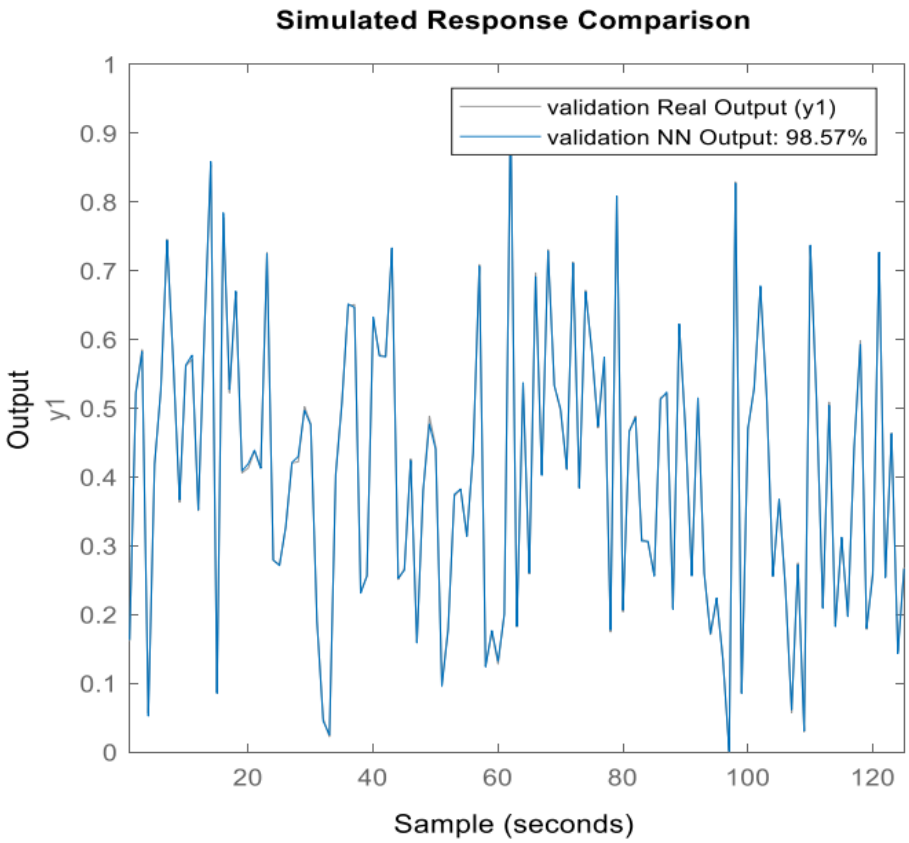


*Figure 53 Error cross-correlation – **low noise***

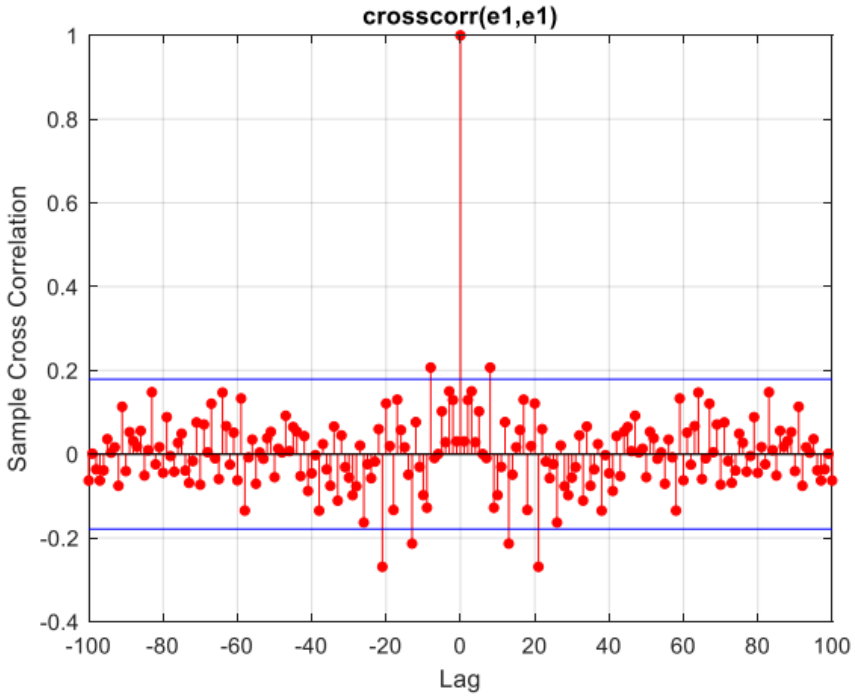


*Figure 54 Error cross-correlation – **low noise***

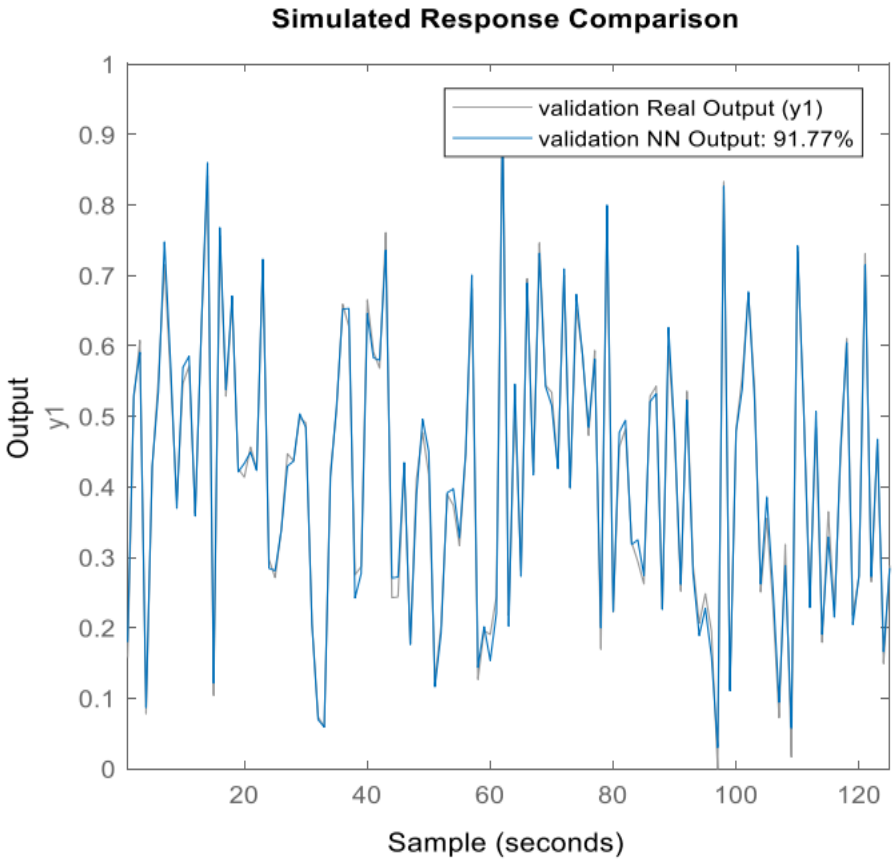


*Figure 55 Error cross-correlation – **moderate noise***

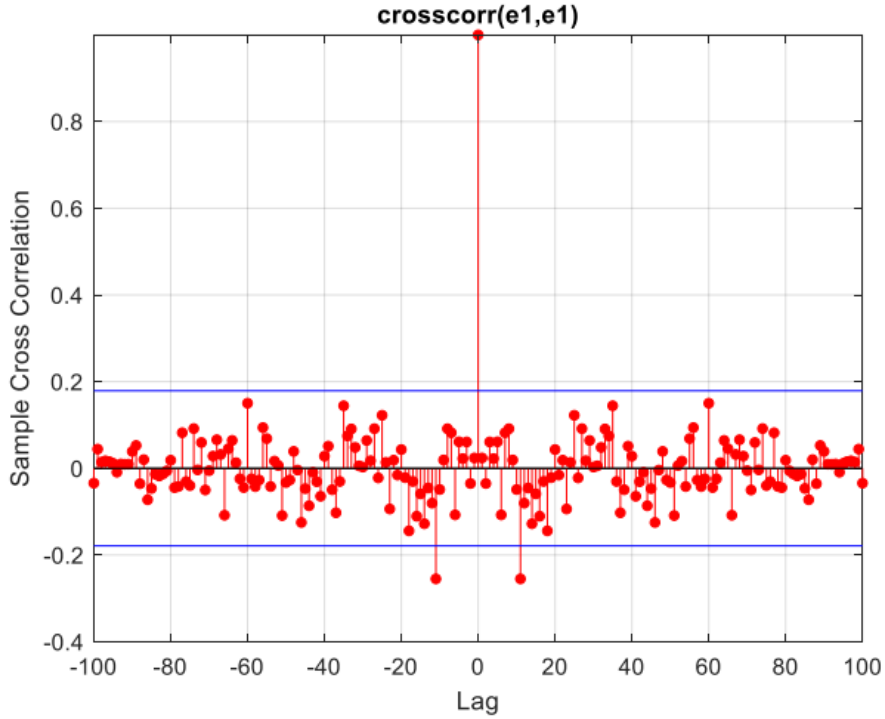


*Figure 56 Error cross-correlation – **moderate noise***

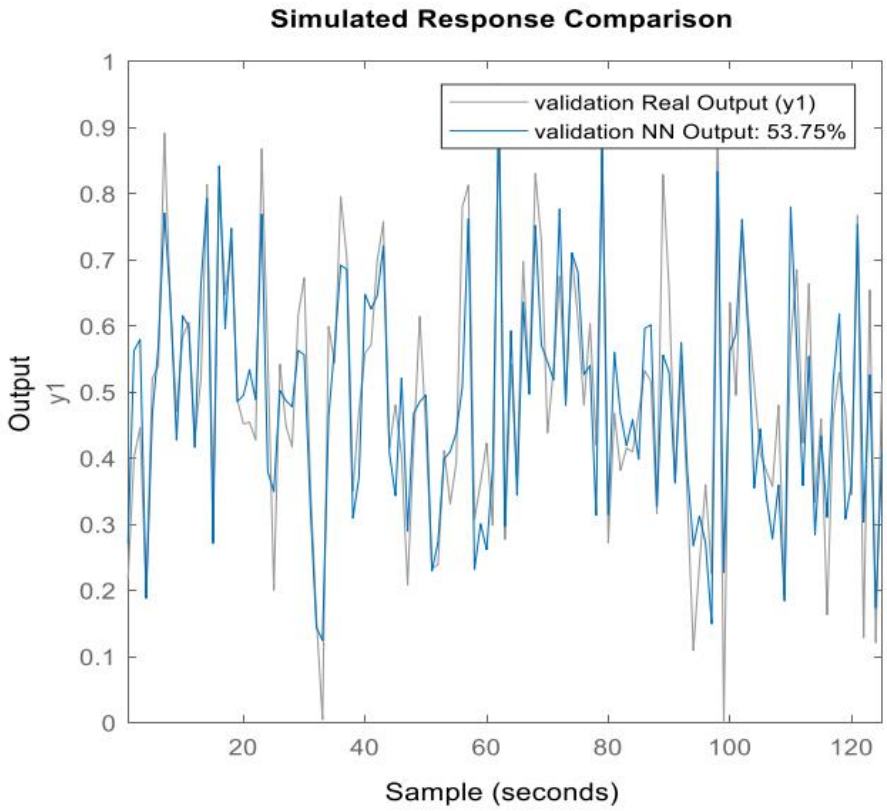


*Figure 57 Error cross-correlation – **high noise***

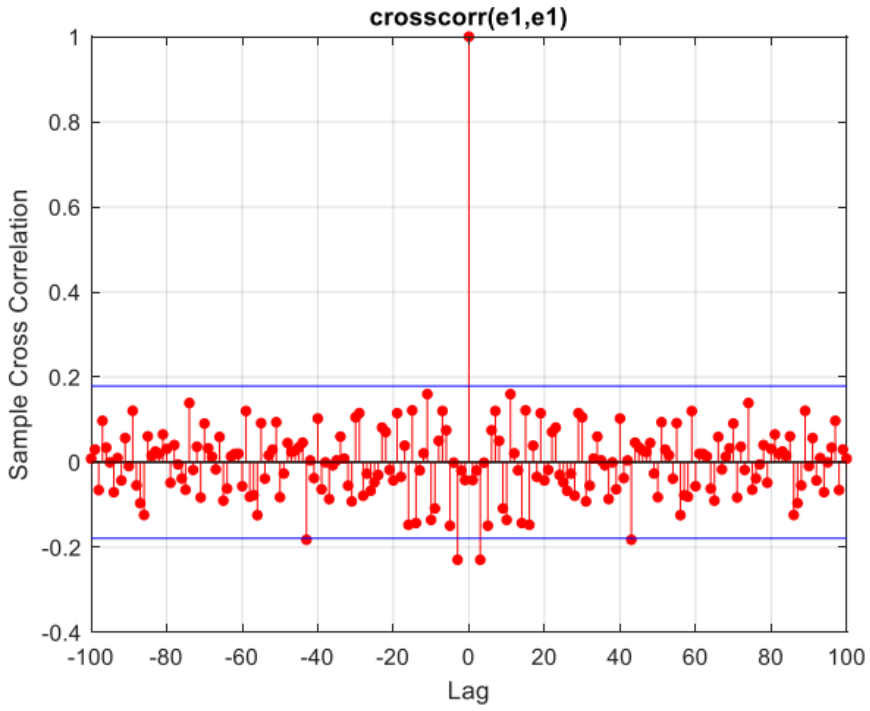


*Figure 58 Error cross-correlation – **high noise***

For each of the three types of noise, error whitening has occurred, and the essential dynamics of the model have been captured. The model demonstrates good validity. It can be observed that as the noise increases, the fitness percentage decreases. In the presence of any level of noise, increasing the noise variance leads to a worse estimation, indicating the network's learning of the noise.

Using the training methods TRAINLM, TRAINGD, and TRAINBFG, the network has been trained, and you can see the error results and fitness percentage on the validation data in Table 1.

Table 2 The statistics of the test data on different training method

TRAINBFG	TRAINLM	TRAINGD	Training Method
73.99	98.9	-0.08	Fitness percentage
0.0029	0.0000051	0.0433	MSE_Test

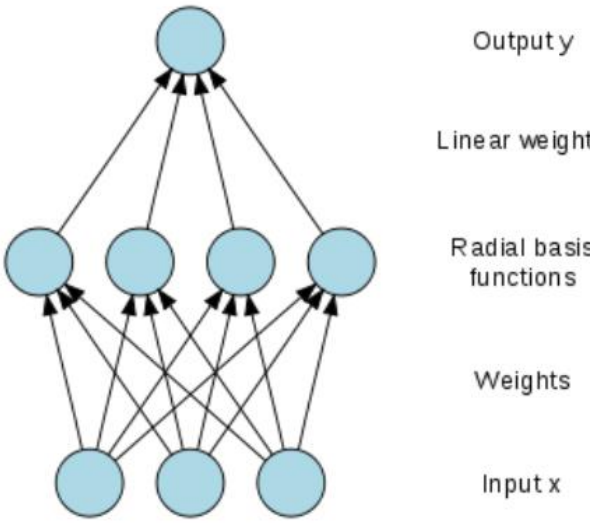
As evident from the table, the LM training method has the least error and the highest fitness percentage. Therefore, the LM training method is the best. The Gauss-Newton algorithm has performed better than the gradient descent on this data.

2.1, 2.2)

RBF

2. and 2.2 are written together.

In the RBF network, we transform the functions inside each neuron into Gaussian functions with a mean and variance. In the case of NRBF, Gaussian functions are also normalized, meaning that the sum of each input in these functions becomes 1. This requires each Gaussian neuron function to be divided by the sum of all neurons in that layer. This makes it a bit challenging for training as it requires a derivative of the sum, but after differentiation, the equations become beautifully formulated, as will be discussed later.

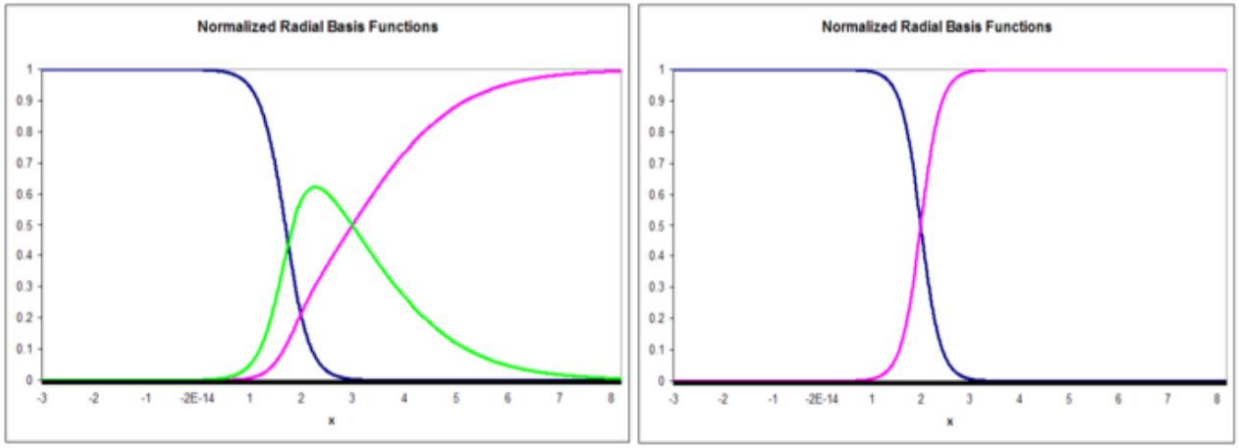


$$\rho(\|\mathbf{x}-\mathbf{c}_i\|) = \exp\left[-\beta\|\mathbf{x}-\mathbf{c}_i\|^2\right]$$

Normalized;

$$\varphi(\mathbf{x}) \overset{\text{def}}{=} \frac{\sum_{i=1}^{N} a_i \rho(\|\mathbf{x}-\mathbf{c}_i\|)}{\sum_{i=1}^{N} \rho(\|\mathbf{x}-\mathbf{c}_i\|)} = \sum_{i=1}^{N} a_i u(\|\mathbf{x}-\mathbf{c}_i\|)$$

$$u(\|\mathbf{x}-\mathbf{c}_i\|) \overset{\text{def}}{=} \frac{\rho(\|\mathbf{x}-\mathbf{c}_i\|)}{\sum_{i=1}^{N} \rho(\|\mathbf{x}-\mathbf{c}_i\|)}$$



RBF:

If we assume that the final output is as follows:

$$Y = w^T e^{-\frac{(K^T x-c)^T \Sigma^{-1} (K^T x-c)}{2}}$$

Additionally, the functions of each neuron are as follows:

$$f_i = \mathrm{e}^{-\frac{(\mathrm{x}-\mathrm{c_i})^2}{2\sigma_\mathrm{i}^2}}$$

First, we calculate the derivative of the function f with respect to the parameters:

$$\frac{\partial \mathrm{f_i}}{\partial \mathrm{c_i}} = \frac{(\mathrm{x}-\mathrm{c_i})}{\sigma_\mathrm{i}^2} \mathrm{e}^{-\frac{(\mathrm{x}-\mathrm{c_i})^2}{2\sigma_\mathrm{i}^2}} = \frac{(\mathrm{x}-\mathrm{c_i})}{\sigma_\mathrm{i}^2} f_i$$

$$\frac{\partial \mathrm{f_i}}{\partial \sigma_\mathrm{i}} = \frac{(\mathrm{x}-\mathrm{c_i})^2}{\sigma_\mathrm{i}^3} \mathrm{e}^{-\frac{(\mathrm{x}-\mathrm{c_i})^2}{2\sigma_\mathrm{i}^2}} = \frac{(\mathrm{x}-\mathrm{c_i})^2}{\sigma_\mathrm{i}^3} f_i$$

$$J = e^T e = (\mathrm{y}-\mathrm{t})^2$$

$$\frac{\partial \mathrm{J_i}}{\partial \mathrm{c_i}} = 2e\mathrm{w_i} \frac{\partial \mathrm{f_i}}{\partial \mathrm{c_i}}$$

$$\frac{\partial \mathrm{J_i}}{\partial \sigma_\mathrm{i}} = 2e\mathrm{w_i} \frac{\partial \mathrm{f_i}}{\partial \sigma_\mathrm{i}}$$

If we replace $\frac{\partial \mathrm{f_i}}{\partial \mathrm{c_i}}$ and $\frac{\partial \mathrm{f_i}}{\partial \sigma_\mathrm{i}}$ in the above relationship, the main equation for derivative will be the result and we use following equations in order to train the Gaussian parameters:

$$\mathrm{c_{i+}} = c_{i-} - \rho \frac{\partial \mathrm{J_i}}{\partial \mathrm{c_i}}$$

$$\sigma_{i+} = \sigma_{i-} - \rho \frac{\partial J_i}{\partial \sigma_i}$$

And for the parameter k we will have:

$$\frac{\partial J_i}{\partial K_i} = -2e w_i \frac{(k_i^T x - c_i)}{\sigma_i^2} x f_i$$

And finally we have;

$$k_{i+} = k_{i-} - \rho \frac{\partial J_i}{\partial k_i}$$

And in the output we have:

$$Y = \sum_{i=1}^{M} \tilde{\tilde{w}}_i \tilde{\tilde{f}}_i^{X}$$

So, it is clear that we can train all the weights using RLS.

2-2)

Optimal Number of Neurons:

In this section, the number of neurons is varied from 1 to 50, and the changes in MSE are plotted. The MSE plots for data without noise, with weak and moderate noise, and with significant noise are shown in the figures below.

Before determining the number of neurons, we select the optimal spread:

spread = [0.001 0.01 0.05 0.5 0.1 1]

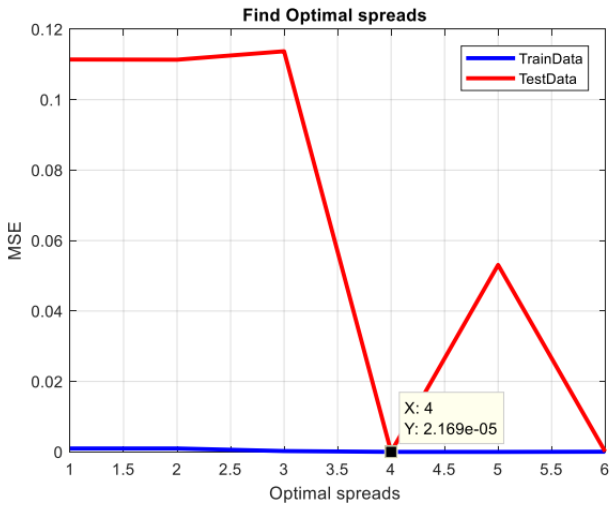


Figure 59 the error-spread plot

Therefore, spread=0.5 is chosen, and we will use it in the subsequent steps. We have it for the gradient descent method.

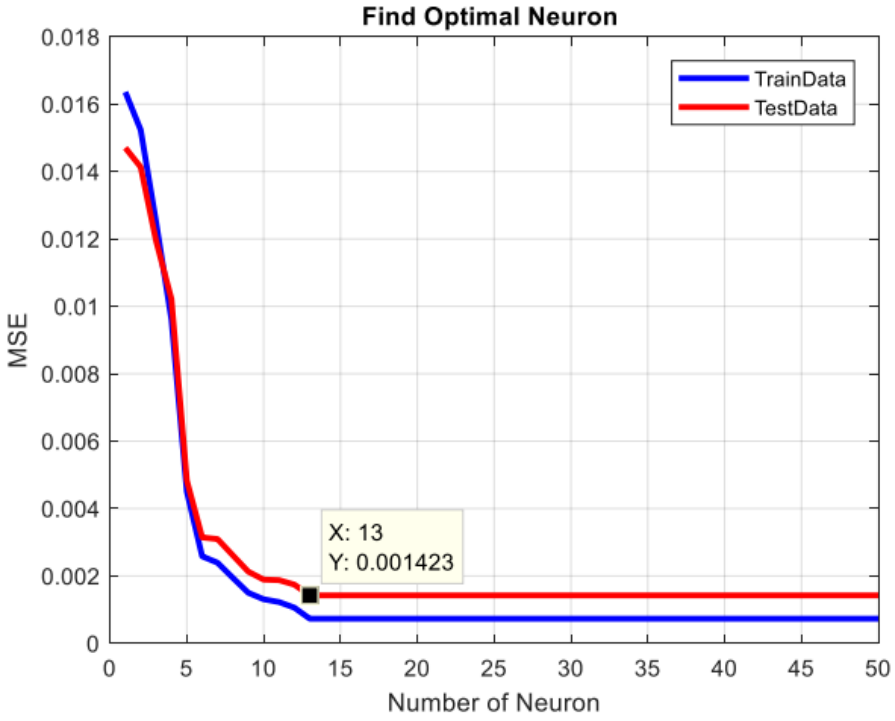


Figure 60 error-no. neurons in RBF model- ***without noise***

13 neurons is the optimal choice.

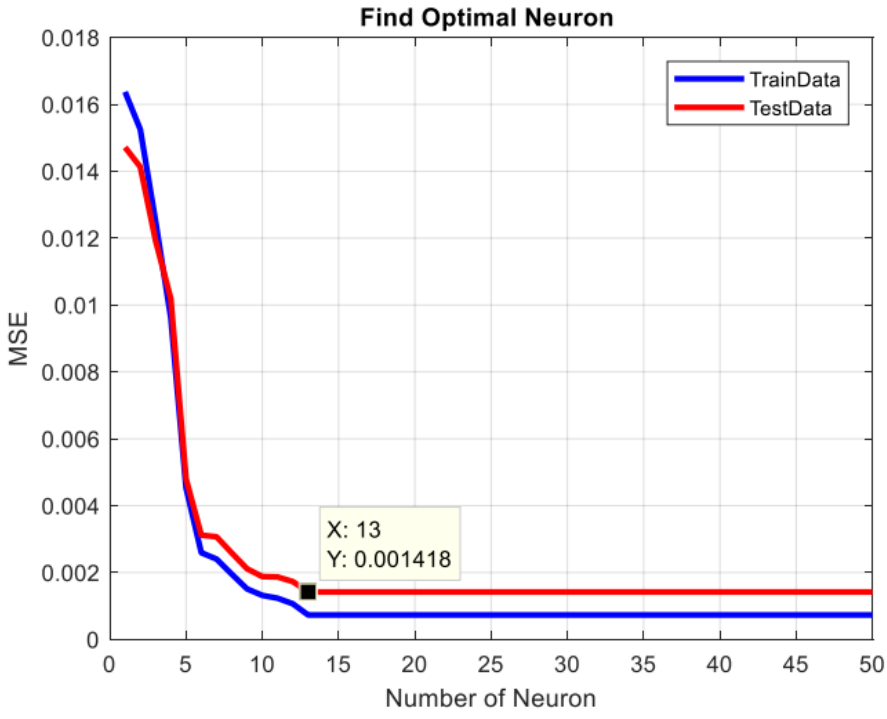


Figure 61 error-no. neurons in RBF model- ***low noise***

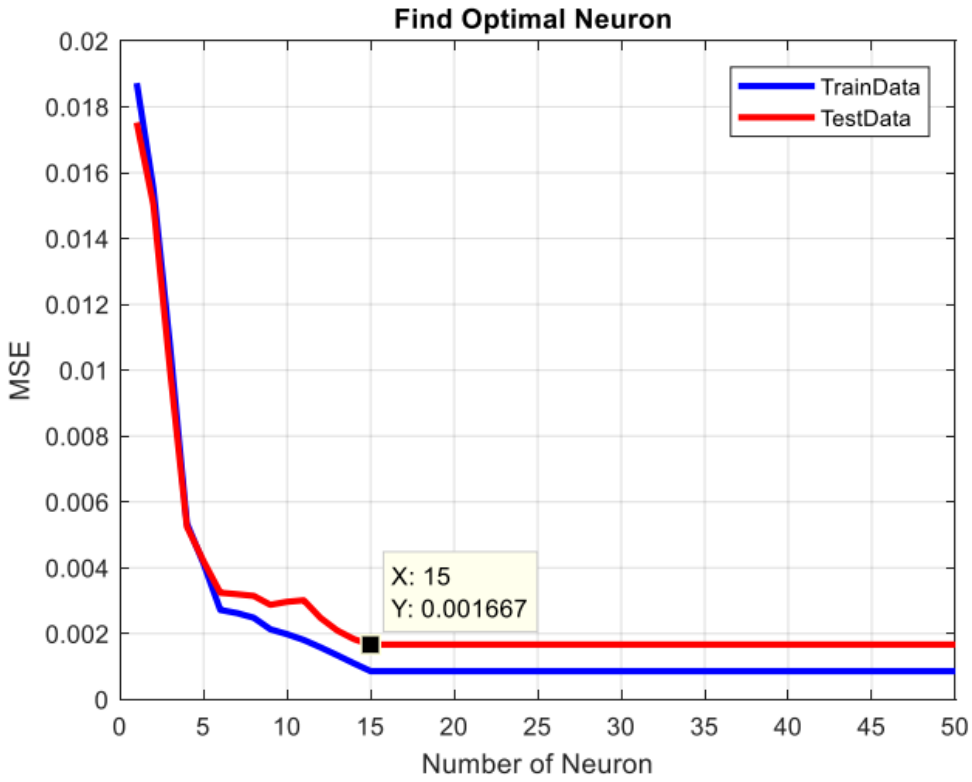


Figure 62 error-no. neurons in RBF model- ***moderate noise***

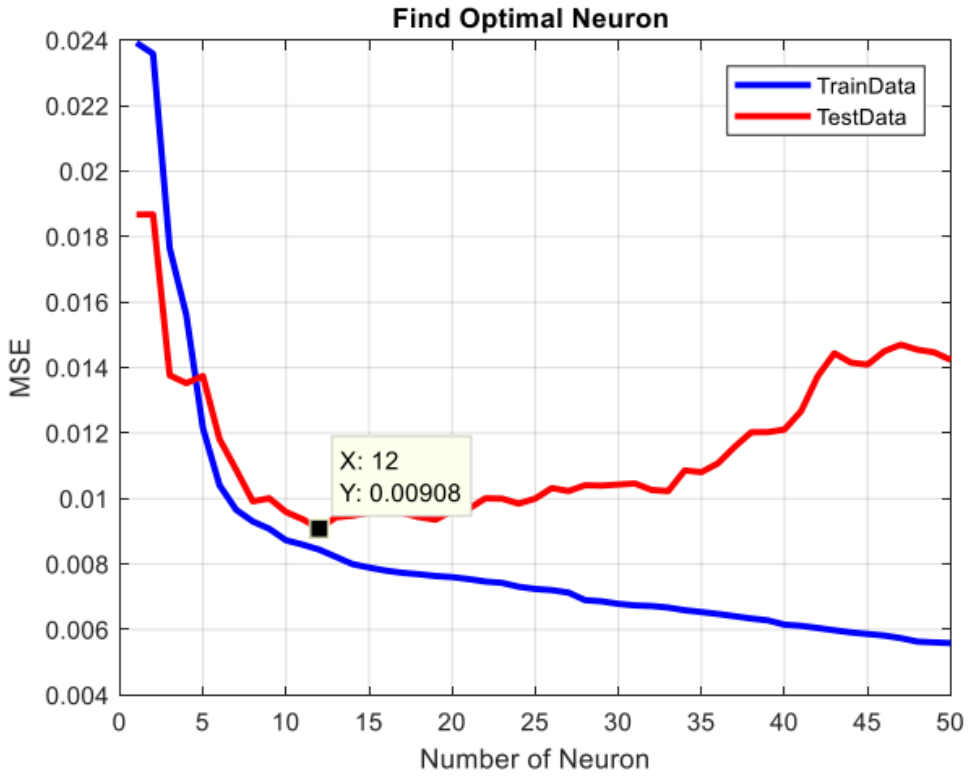


Figure 63 error-no neurons in RBF model- ***high noise***

In order, 13, 13, 15, and 12 neurons are chosen as the optimal neurons. The MSE error increases significantly with the increase in high noise, indicating that the network is learning the noise and is not desirable.

trainlm

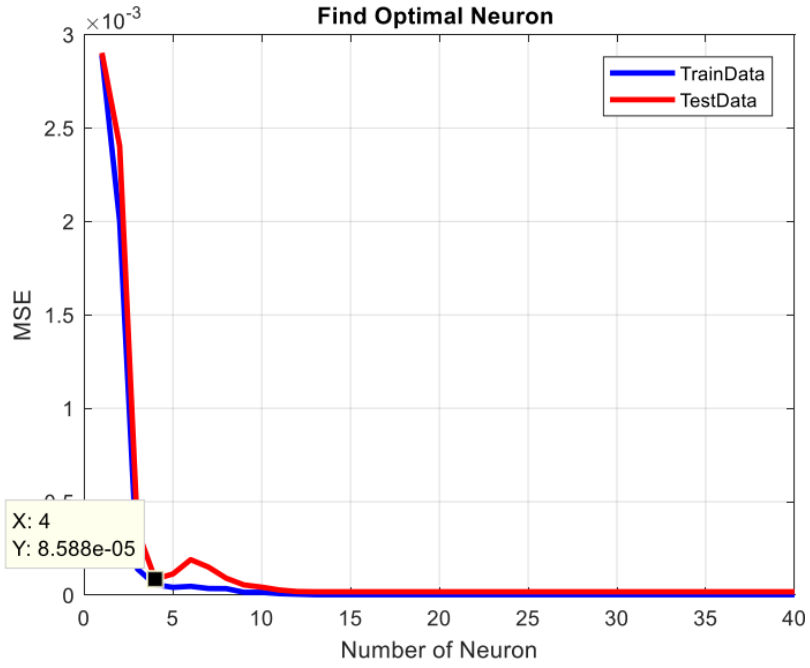


Figure 64 error-no neurons in RBF model- Levenberg–Marquardt- ***without noise***

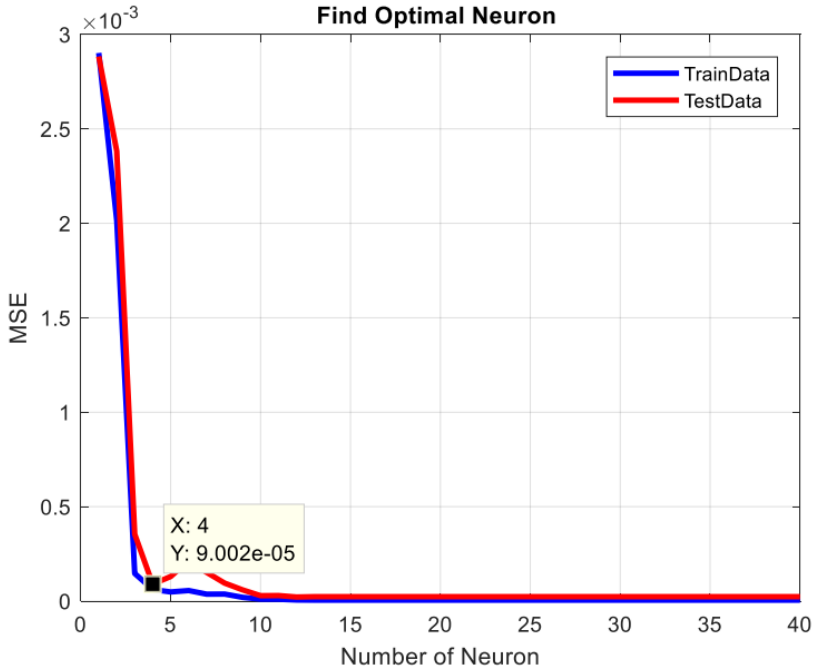


Figure 65 error-no neurons in RBF model- Levenberg–Marquardt- ***low noise***

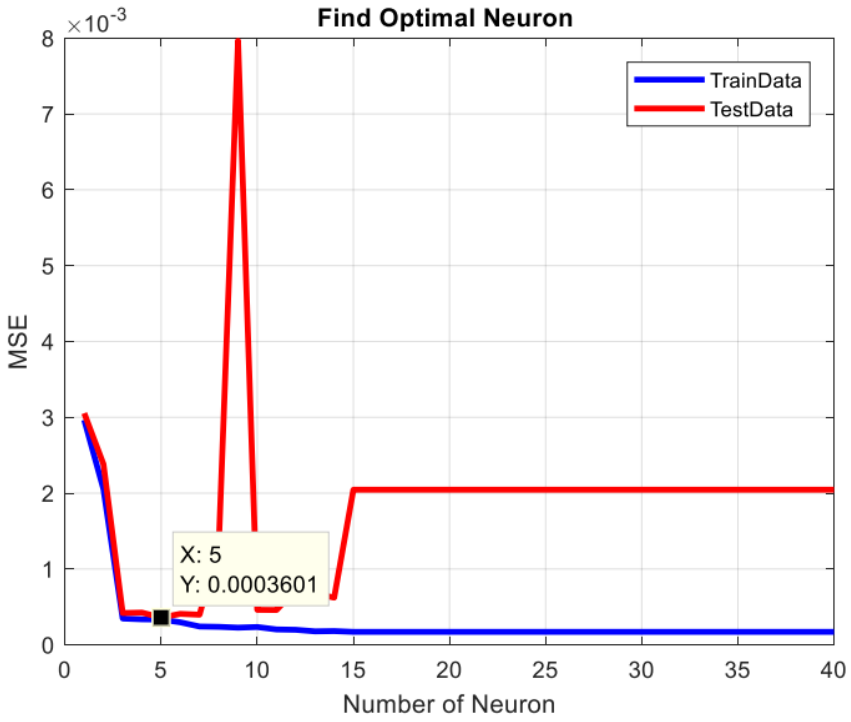


Figure 66 error-no. neurons in RBF model- Levenberg–Marquardt- ***moderate noise***

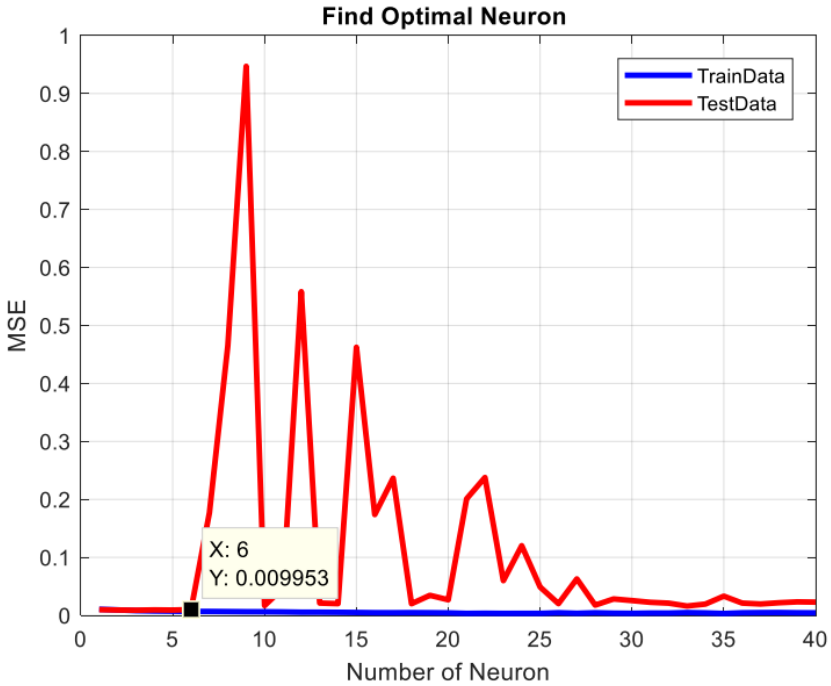


Figure 67 error-no. neurons in RBF model- Levenberg–Marquardt- ***high noise***

In order, 4, 4, 5, and 6 neurons are chosen as the optimal neurons. The MSE error increases significantly with the increase in high noise, indicating that the network is learning the noise and is not desirable.

Now we select the optimal number of epochs using the **trainlm** method.

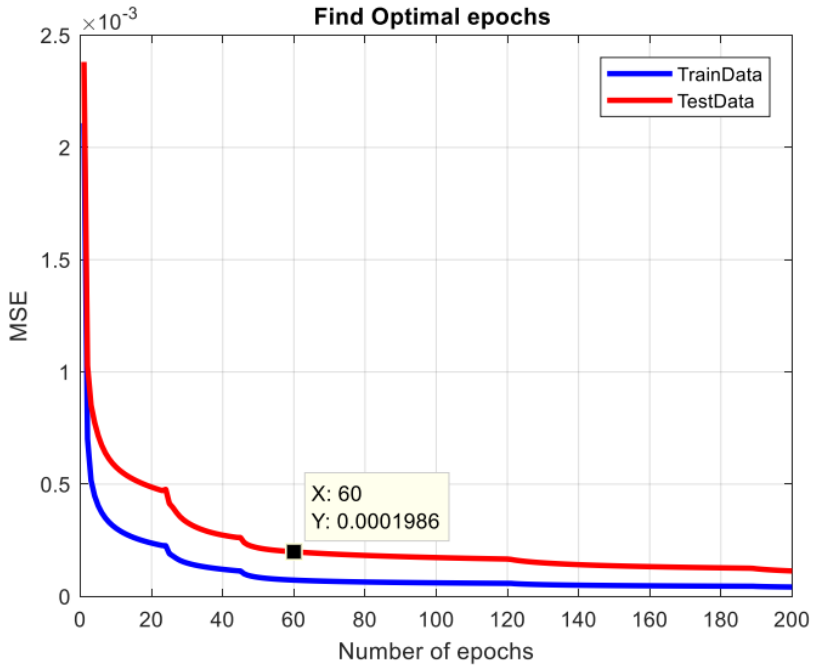


Figure 68 error-epoch in RBF model- Levenberg–Marquardt- ***without noise***

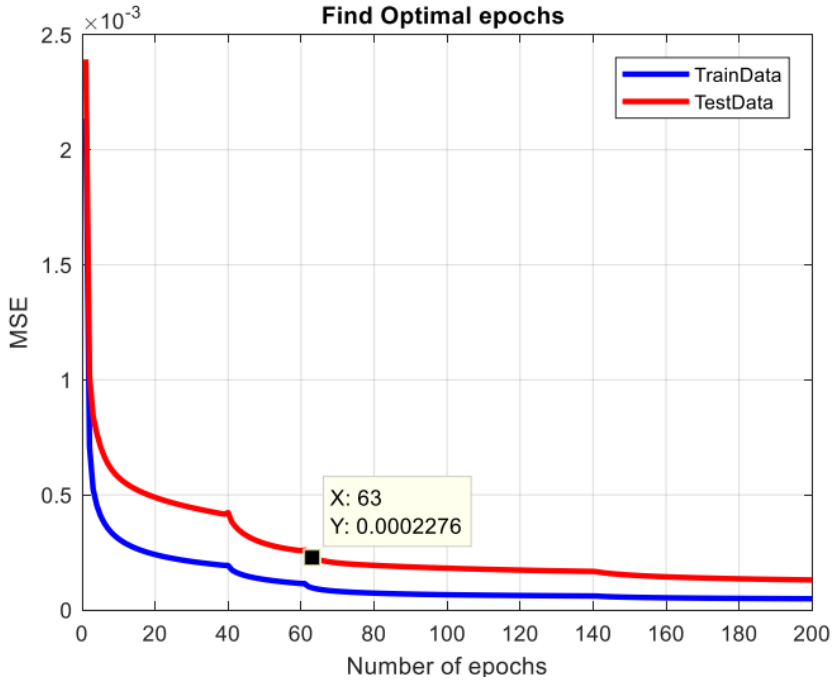


Figure 69 error-epoch in RBF model- Levenberg–Marquardt- ***low noise***

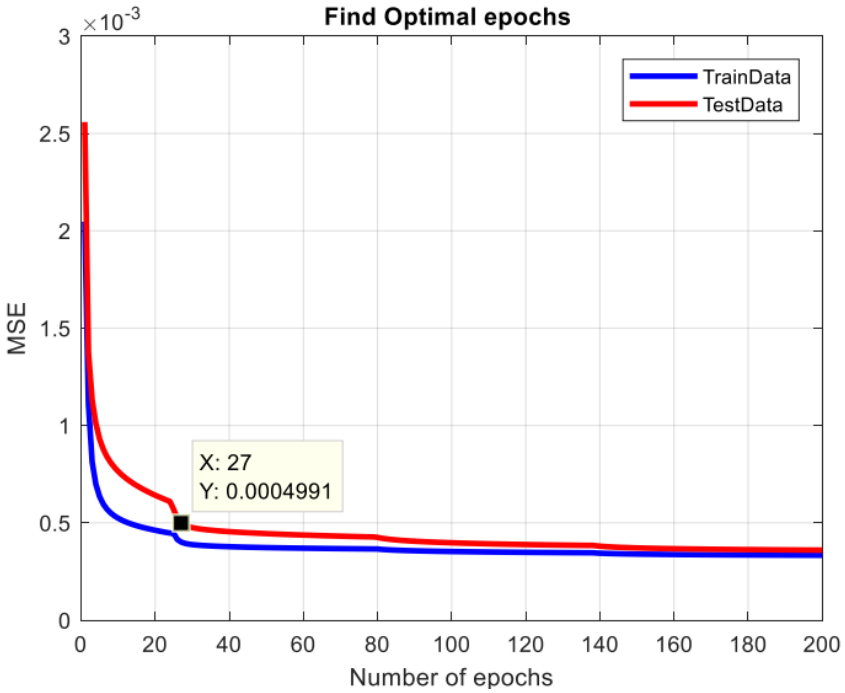


Figure 70 error-epoch in RBF model- Levenberg–Marquardt- **moderate noise**

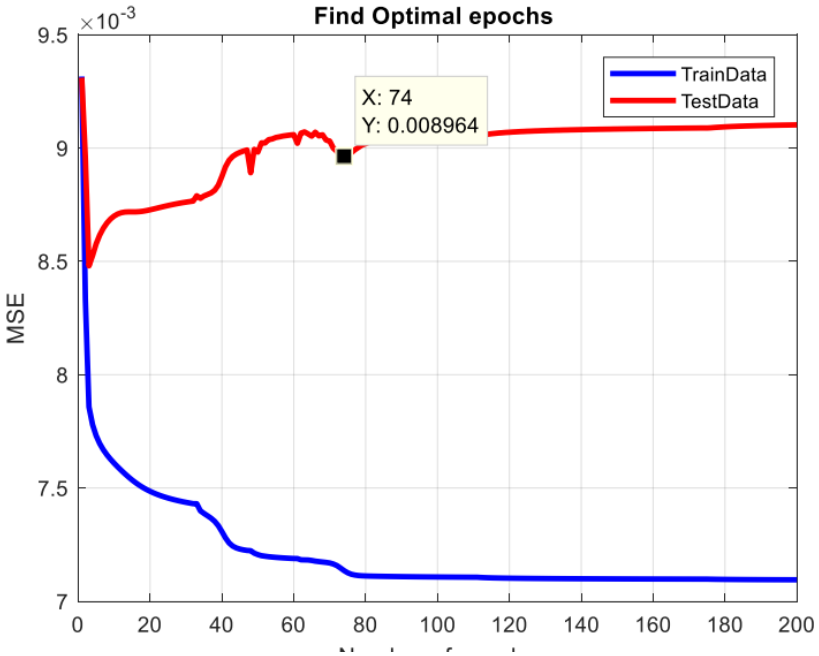


Figure 71 error-epoch in RBF model- Levenberg–Marquardt- ***high noise***

In order, 60, 63, 27, and 74 epochs can be considered as the optimal number of epochs.

Validation:

Finally, using the 125 data points that we set aside as validation data, we perform the validation of the neural network. We show the output for the trainlm method (as the results were not satisfactory with the gradient descent method).

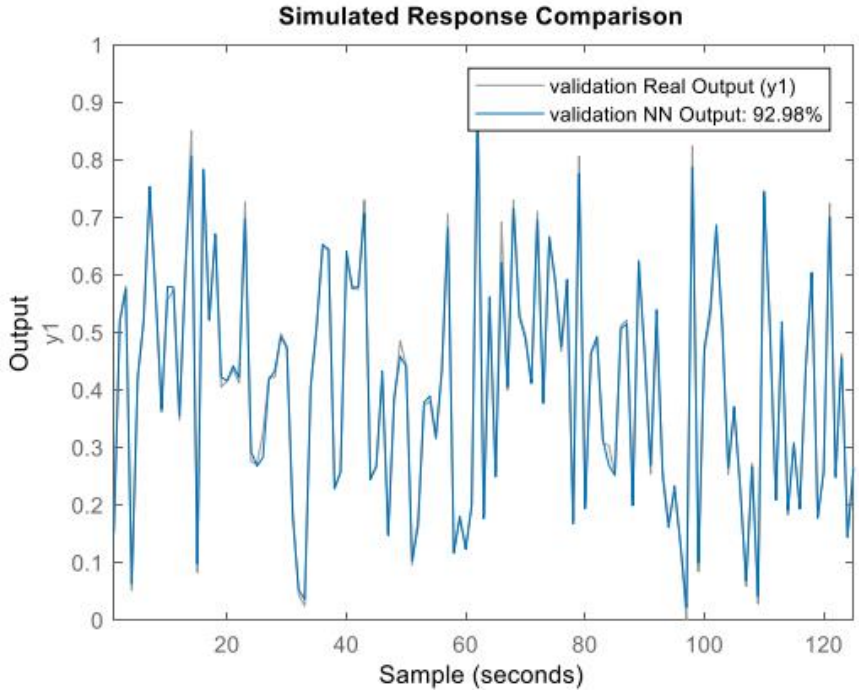


*Figure 72 The actual and the estimated system output-validation-**without noise***

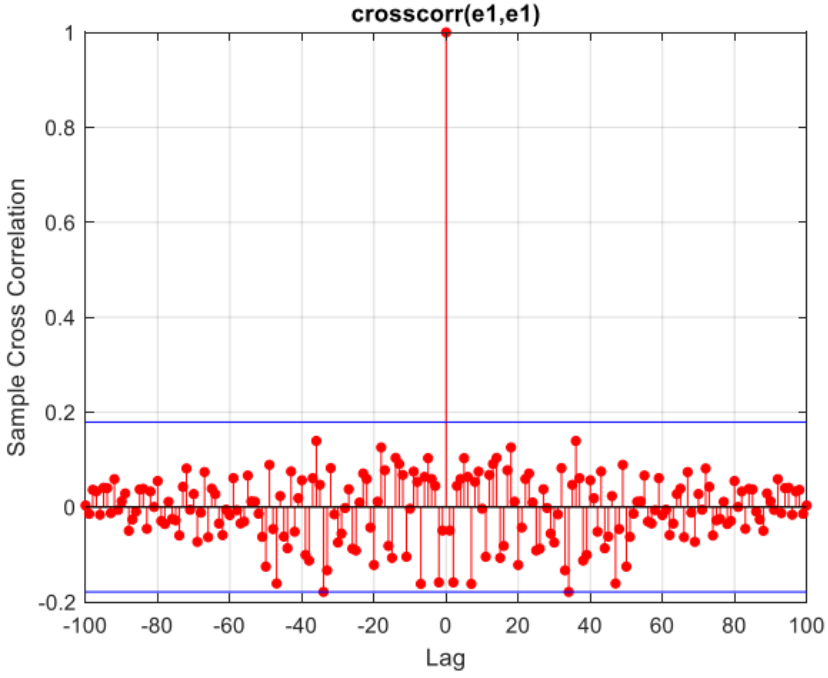


*Figure 73 error cross-correlation **without noise***

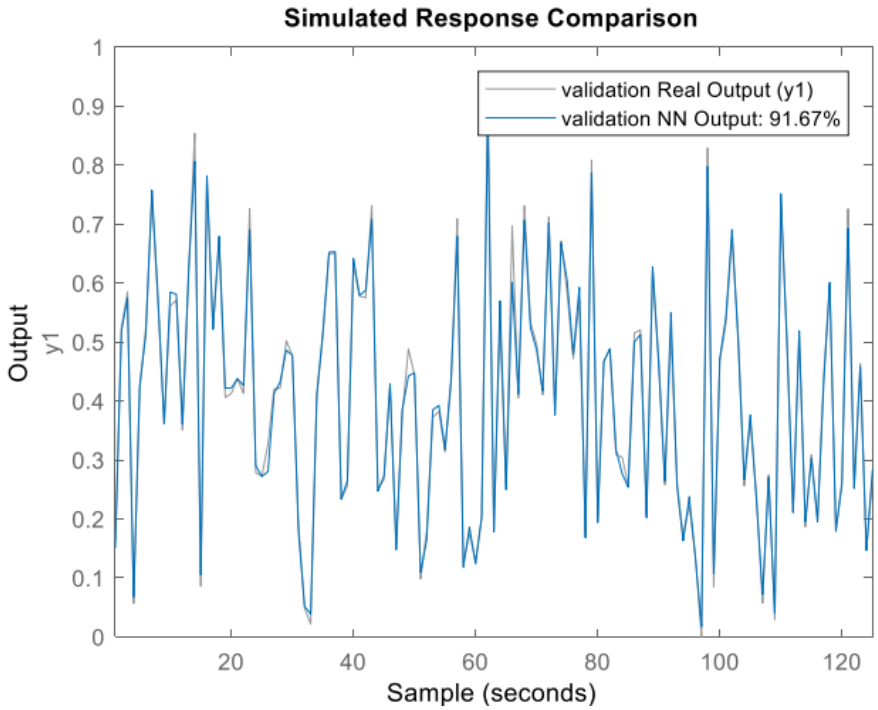


*Figure 74 The actual and the estimated system output-validation-**lwo noise***

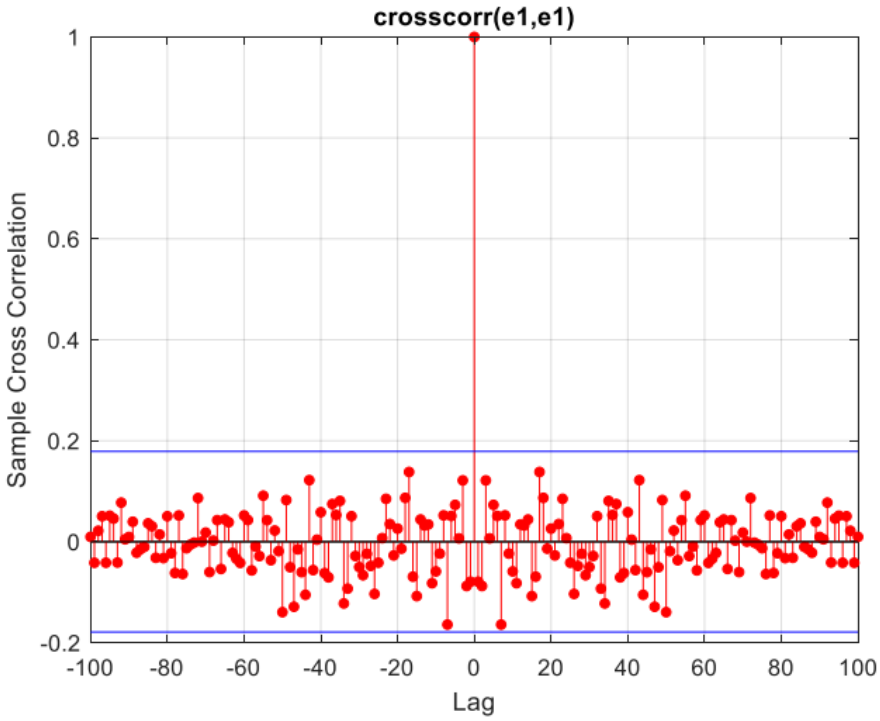


*Figure 75 error cross-correlation **low noise***

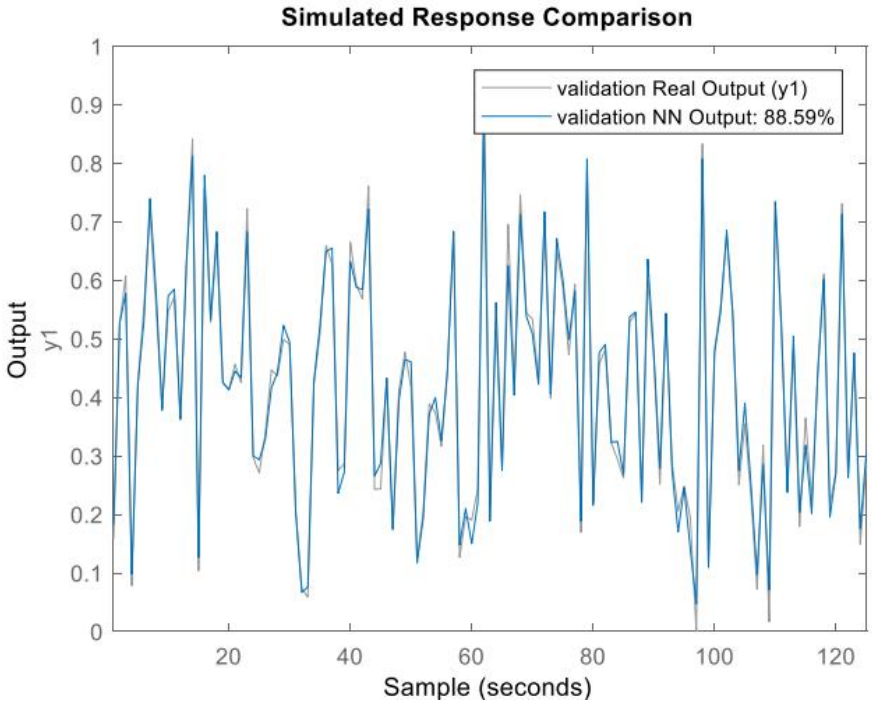


*Figure 76 The actual and the estimated system output-validation-**moderate noise***

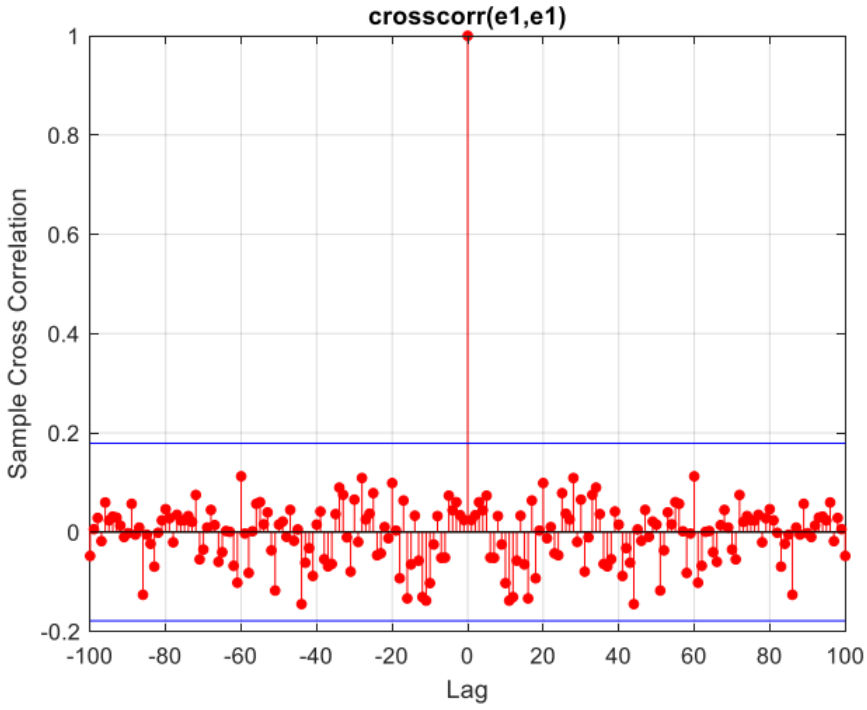


*Figure 77 error cross-correlation **moderate noise***

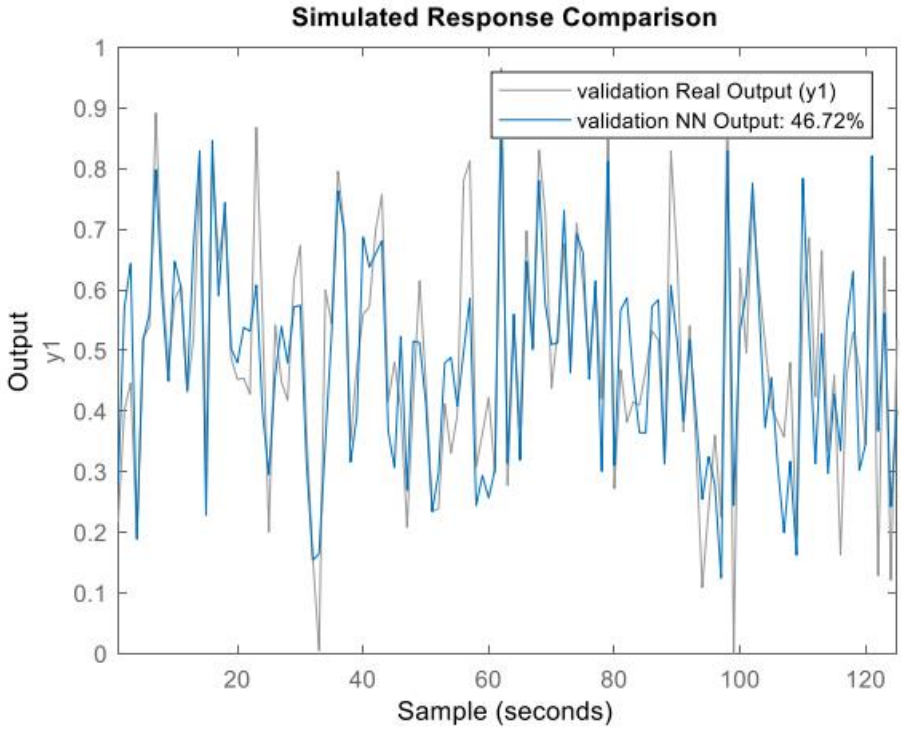


*Figure 78 The actual and the estimated system output-validation-**high noise***

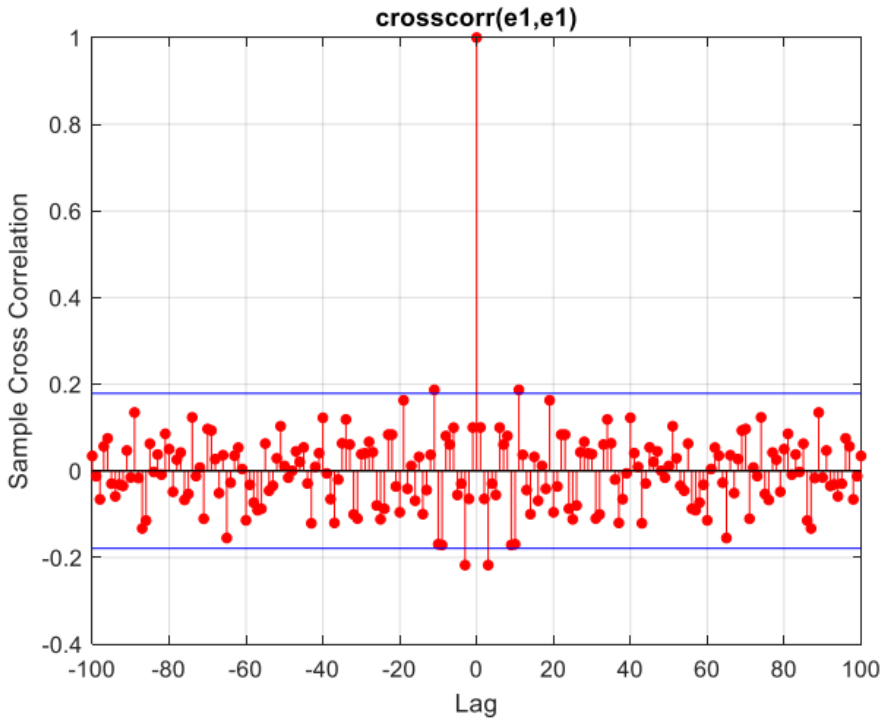


*Figure 79 error cross-correlation **high noise***

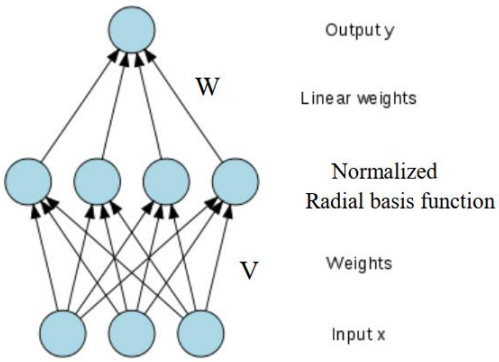
2.3)

For each of the three types of noise, error whitening has occurred, and the essential dynamics of the model have been captured. The model demonstrates good validity. It can be observed that as the noise increases, the fitness percentage decreases. In the presence of any level of noise, increasing the noise variance leads to a worse estimation, indicating the network's learning of the noise. Compared to a similar case, the accuracy of the single-layer and two-layer MLP networks is higher than that of the RBF.

2.4)

NRBF

As mentioned in the introduction to the question, an NRBF network differs only in the function inside each neuron, which in this network is Gaussian. When the sum of the Gaussian function for each neuron for all inputs is equal to 1, it means it has been normalized. This means that it is normalized, and this factor causes more gain in one input region and less gain in another input region. This normalization factor also increases the speed.



The relationships in this section are slightly more complex than the previous section due to a factor in the denominator, making differentiation more complicated. However, with simplification, it leads to a beautiful closed form. If the functions of each neuron are as follows:

$$f_i = \frac{e^{-\frac{(v_i^T x - c_i)^2}{2\sigma_i^2}}}{\sum_{i=1}^{M} e^{-\frac{(v_i^T x - c_i)^2}{2\sigma_i^2}}}$$

The output of the network:

$$Y = \sum_{i=1}^{M} \hat{\theta}_i \hat{X}_i$$

From these relationships, we can calculate the weights w_i using RLS. As mentioned before, the presence of a linear layer at the end of the network plays a crucial role. In fact, the learning speed of the network is the learning speed of its last linear layer.

$$J = e^T e = (y - t)^2$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{c_j}} = 2e\frac{\partial \mathrm{Y}}{\partial \mathrm{c_j}}$$

And for $\frac{\partial \mathrm{Y}}{\partial \mathrm{c_j}}$ we can write:

$$\frac{\partial \mathrm{Y}}{\partial \mathrm{c_j}} = \sum_{i=1}^{M} w_i \frac{\partial \mathrm{f_i}}{\partial \mathrm{c_j}}$$

After calculating the derivatives and simplification, we have:

$$\frac{\partial f_k}{\partial c_j} = \begin{cases} = \dfrac{-(v_k^T x - c_k)}{\sigma_k^2} f_k(1 - f_k)\,, (j = k) \\ = \dfrac{\left(v_j^T x - c_j\right)}{\sigma_j^2} f_k f_j\,, (j \neq k) \end{cases}$$

And for other parameters we have:

$$\frac{\partial f_k}{\partial \sigma_j} = \begin{cases} = \dfrac{-(v_k^T x - c_k)^2}{\sigma_k^3} f_k(1 - f_k)\,, (j = k) \\ = \dfrac{\left(v_j^T x - c_j\right)^2}{\sigma_j^3} f_k f_j\,, (j \neq k) \end{cases}$$

$$\frac{\partial f_k}{\partial v_j} = \begin{cases} = \dfrac{-(v_k^T x - c_k)}{\sigma_k^2} f_k(1 - f_k) \cdot x\,, (j = k) \\ = \dfrac{\left(v_j^T x - c_j\right)}{\sigma_j^2} f_k f_j \cdot x\,, (j \neq k) \end{cases}$$

w_i parameters are trained in RLS network.

Optimal Number of Neurons:

In this section, using the RBF network commands and the radbasn activation function, an NRBF neural network is formed. The number of neurons is varied from 1 to 40, and the changes in MSE are plotted. The MSE plots for data without noise, with weak and moderate noise, and with significant noise are shown in the figures below.

Before that, we determine the optimal **spread**.

spread = [0.001 0.01 0.05 0.5 0.1 1]

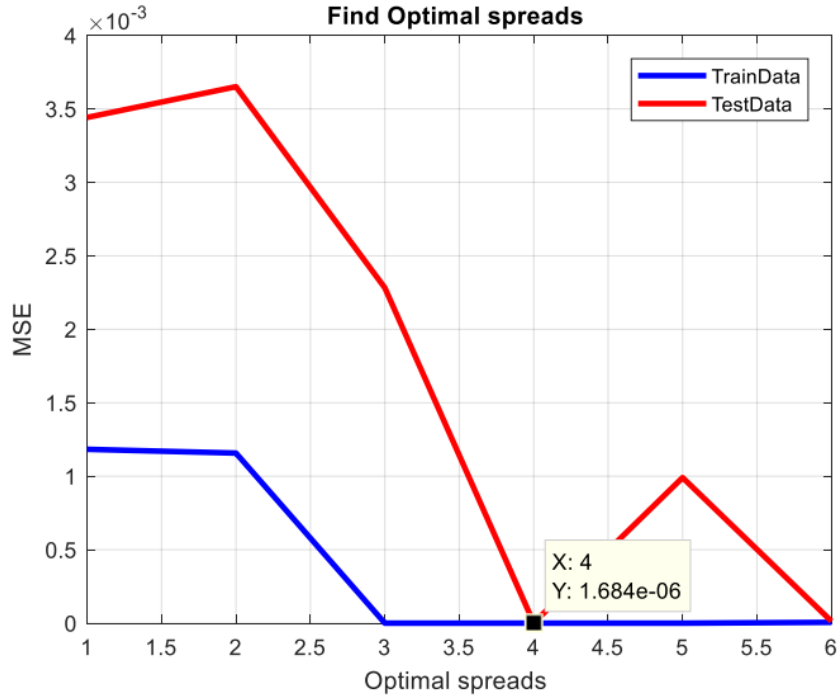


Figure 80 Error-spread plot

The optimal spread is 0.5.

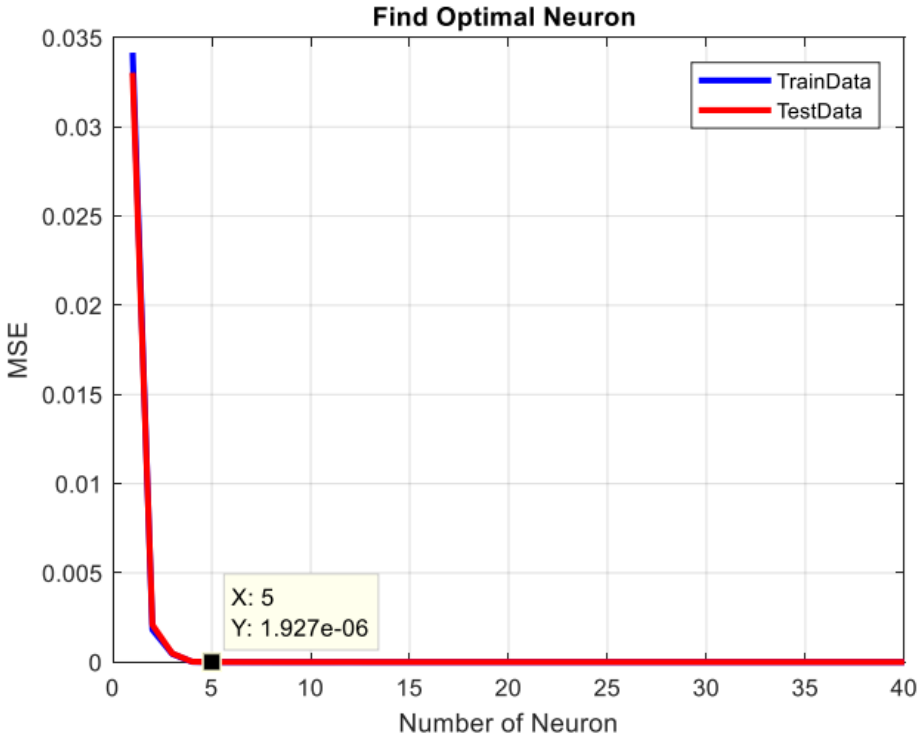


*Figure 81 The error-neurons-**without noise***

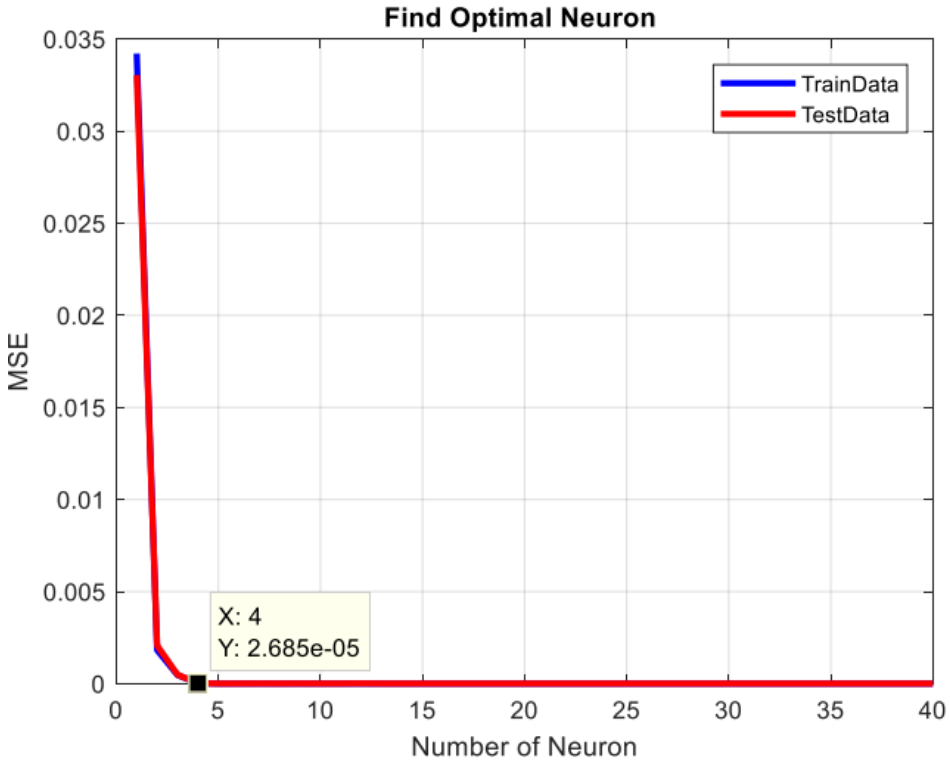


*Figure 82 The error-neurons-**low noise***

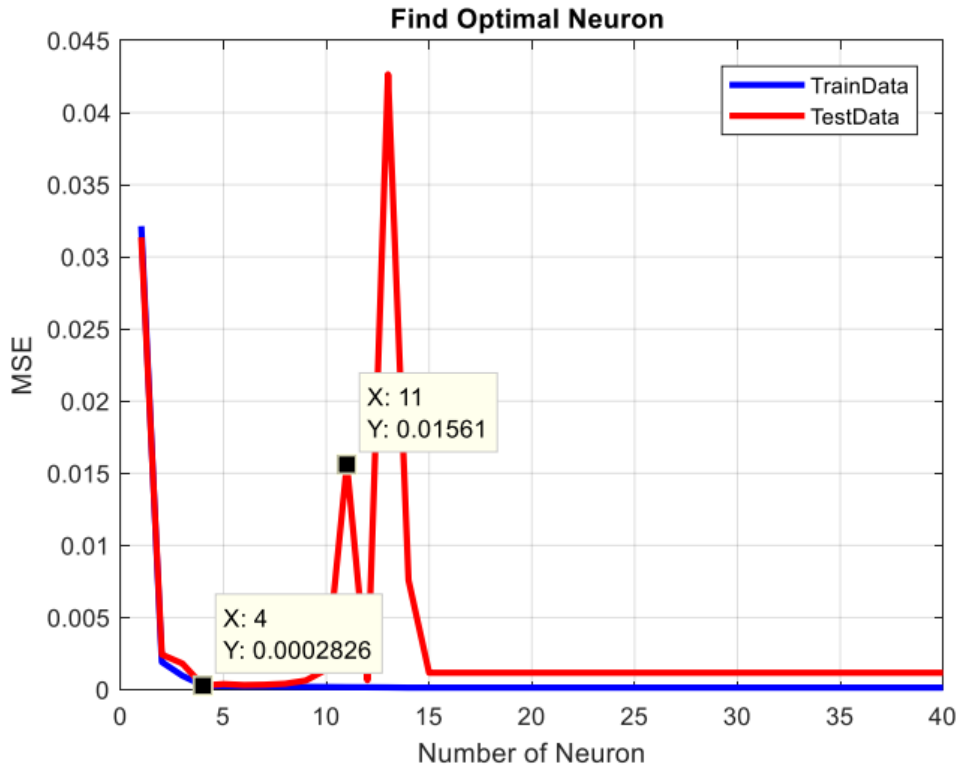


*Figure 83 The error-neurons-**moderate noise***

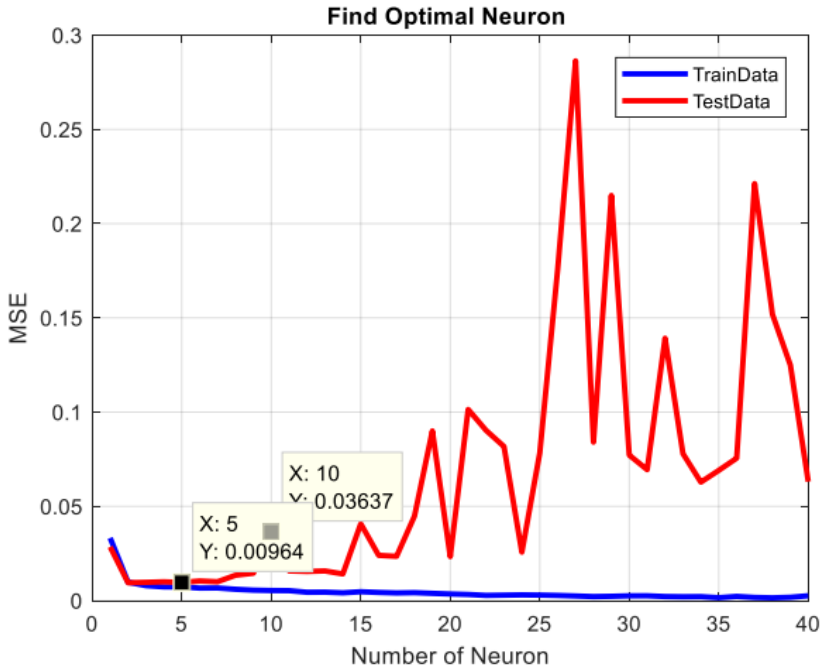


*Figure 84 The error-neurons-**high noise***

The optimal number of neurons is, in order, 5, 4, 4, and 5. The MSE error increases significantly with the increase in high noise, indicating that the network is learning the noise and is not desirable.

Now, we select the optimal number of epochs using the **trainlm** method:

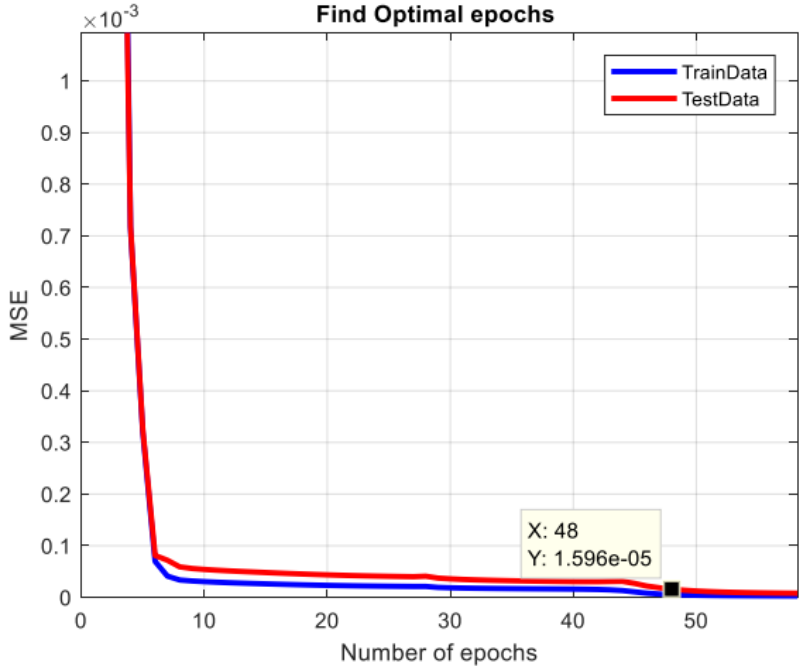


*Figure 85 Error-epoch plot-**without noise***

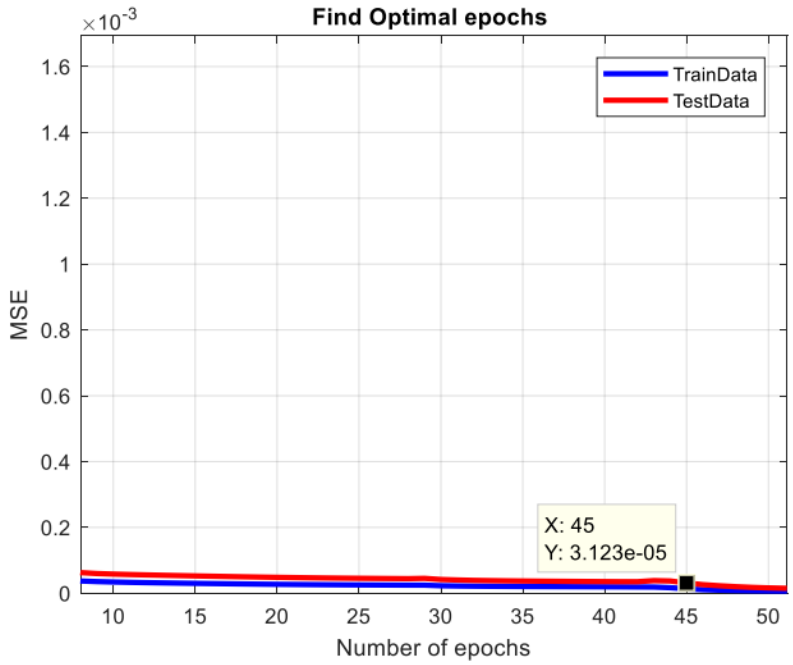


*Figure 86 Error-epoch plot-**low noise***

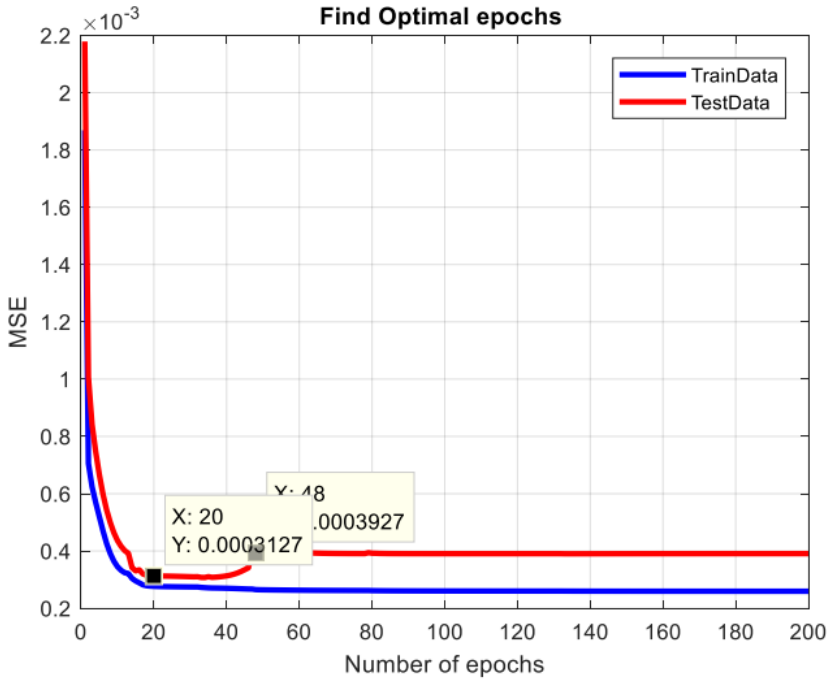


*Figure 87 Error-epoch plot-**moderate noise***

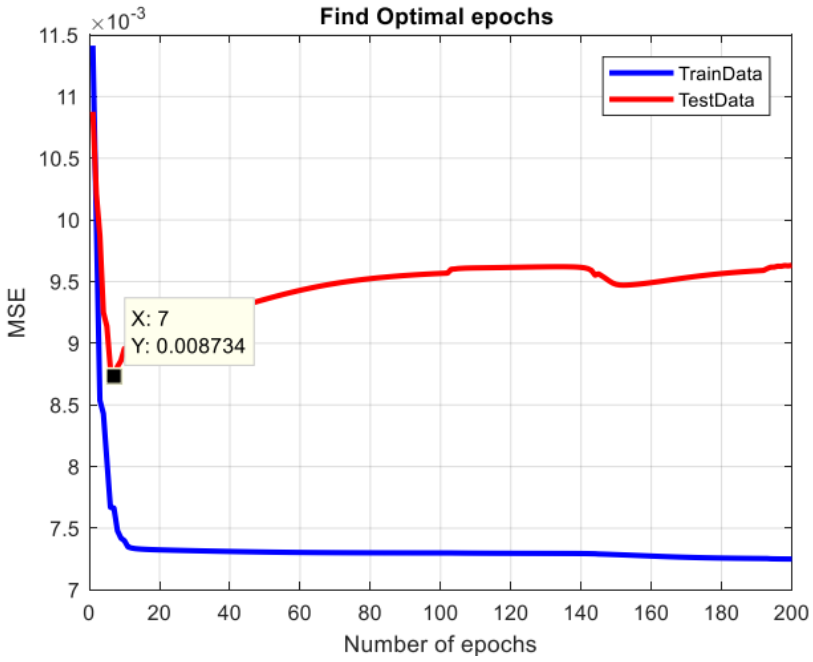


*Figure 88 Error-epoch plot-**high noise***

In order, 48, 45, 20, and 7 epochs can be considered as the optimal number of epochs. However, the last two plots may be less reliable.

Validation:

Finally, we perform validation on the neural network using the 125 data points set aside for validation. We display the output using the trainlm method (because the results with the gradient descent method were not satisfactory). The correlation error plot for this data is also shown.

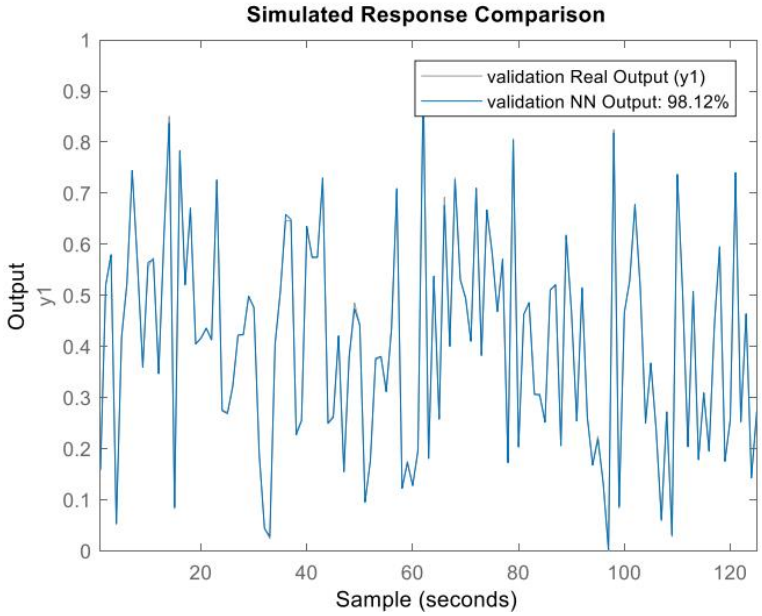


*Figure 89 The actual and estimated output-validation-**without noise***

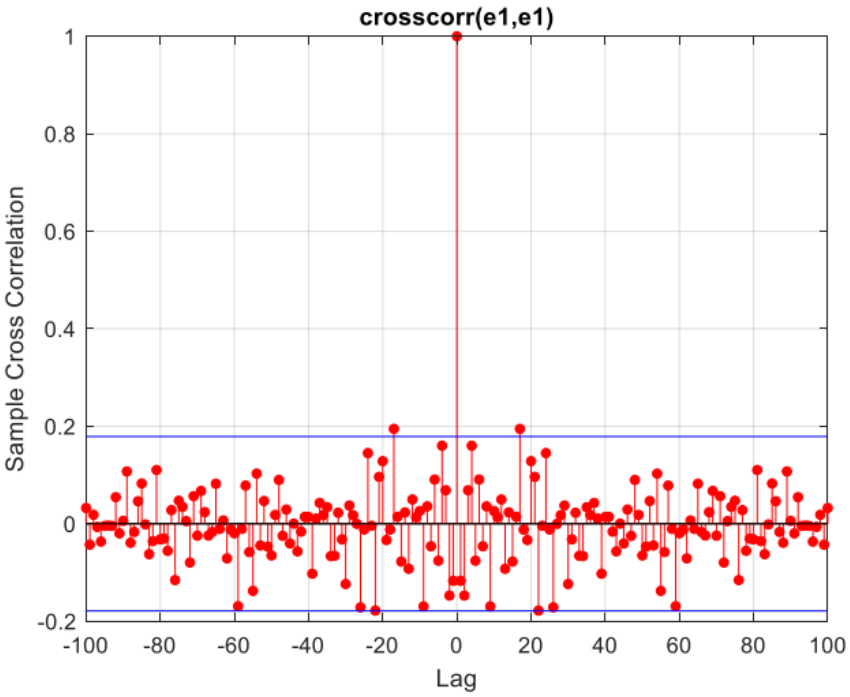


*Figure 90 cross-correlation of error **without noise***

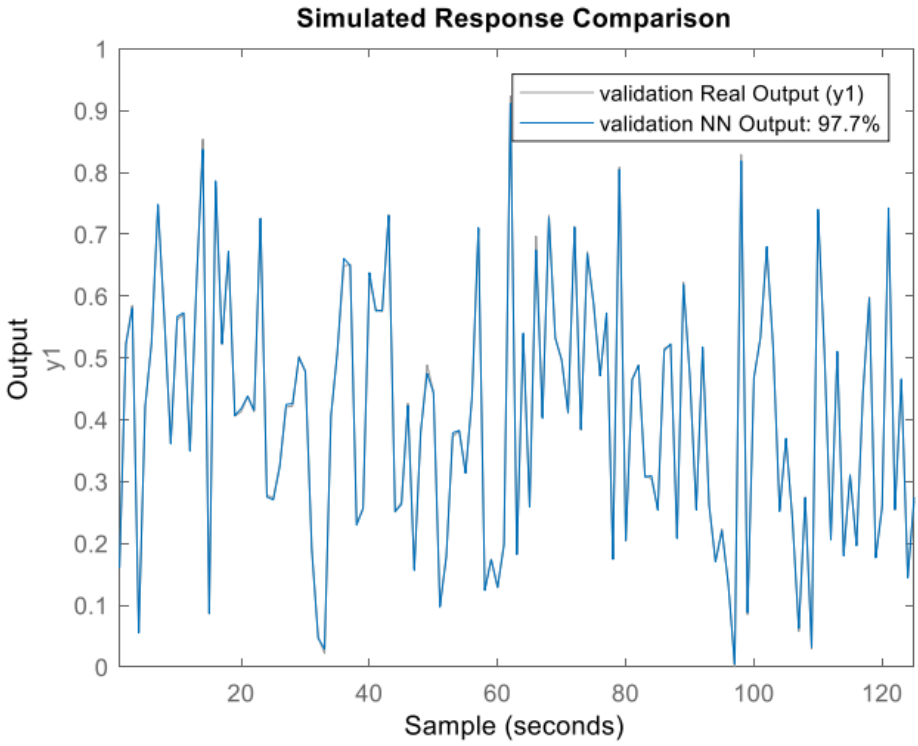


*Figure 91 The actual and estimated output-validation-**low noise***

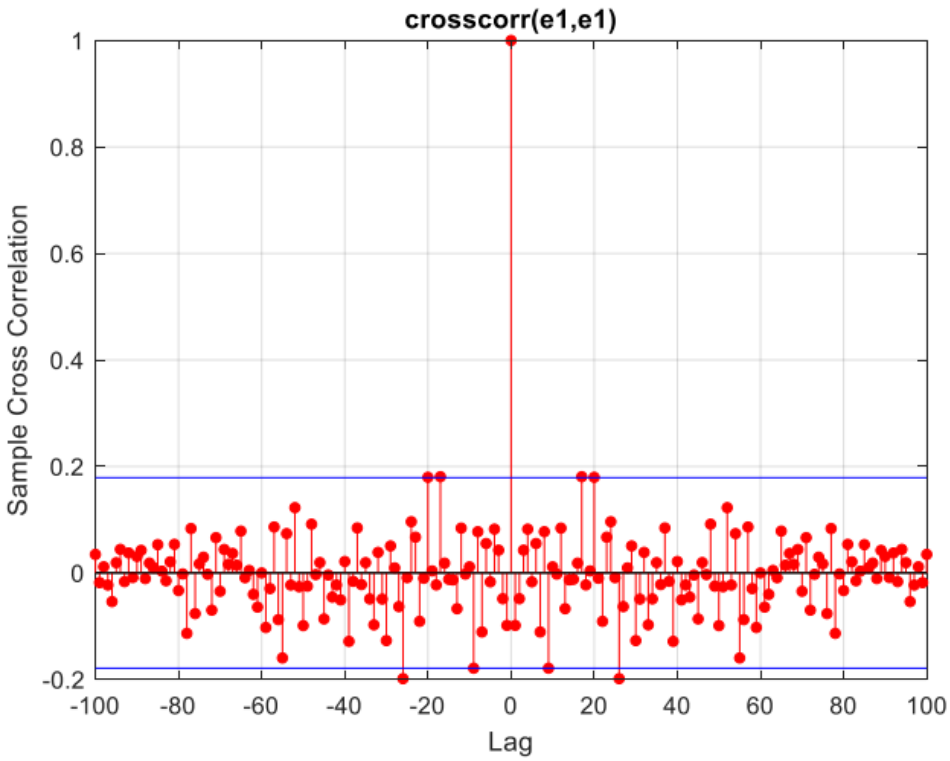


*Figure 92 cross-correlation of error **low noise***

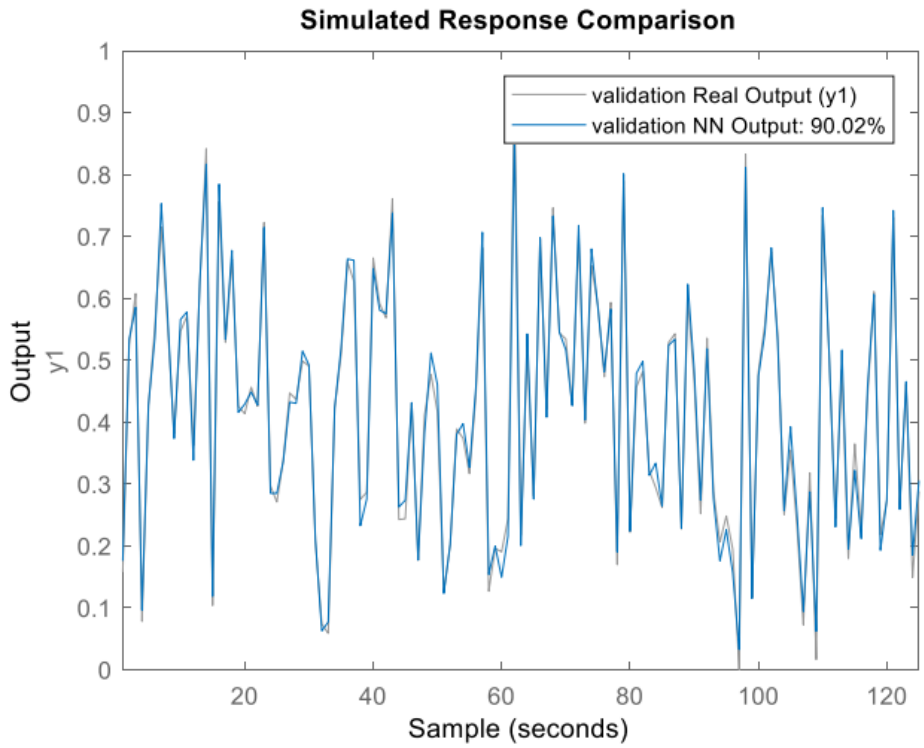


*Figure 93 The actual and estimated output-validation-**moderate noise***

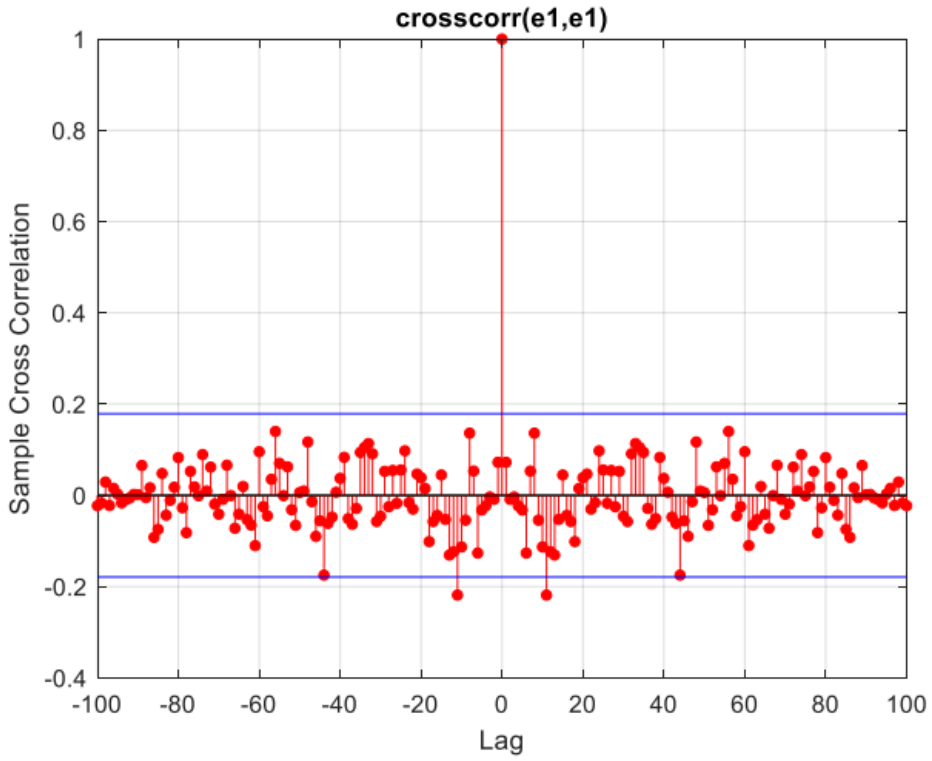


Figure 94 cross-correlation of error ***moderate noise***

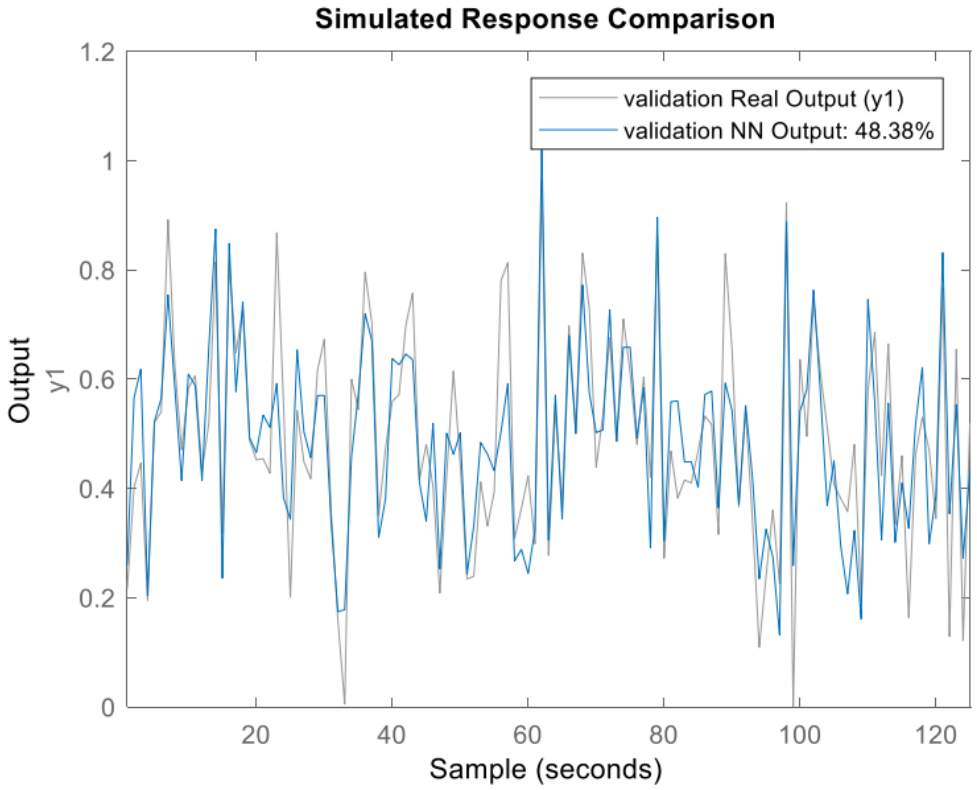


*Figure 95 The actual and estimated output-validation-**high noise***

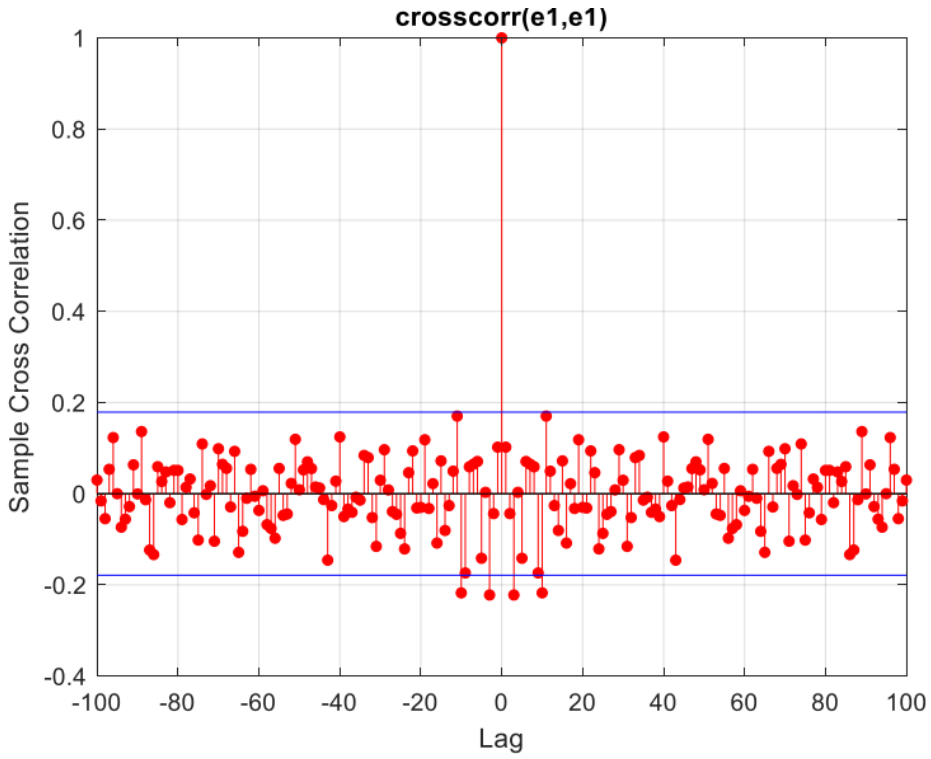


*Figure 96 cross-correlation of error **moderate noise***

For each type of noise, whitening of the error has occurred, indicating that important dynamics have been captured, and the model benefits from good generalization. It is observed that with an increase in noise, the fitting percentage decreases. In cases with higher noise levels, MLP performs slightly better than NRBF, and overall, the fitting percentage of NRBF is higher than RBF. In terms of error whitening, RBF performs slightly better than NRBF.

3.1)

Third part-LLM

In this section, the LOLIMOT algorithm is used to identify non-linear functions. This algorithm divides the input space into two equal parts parallel to the orthogonal axes, resulting in the subdivision of the input space into rectangles. A Gaussian function with a standard deviation equal to 1/3 of the size of each dimension is placed at the center of each rectangle. Consequently, the parameters of the non-linear model are obtained without the need for non-linear optimization methods.

The LOLIMOT algorithm divides the input space axis-orthogonally into two equal parts in a way that, in the first step, the worst region (the one having the maximum weighted sum of output squared errors) is selected. A neuron is placed at the center of each part, and the modeling error in each part is calculated. This process is performed for all possible directions for breaking the space, and the modeling error is calculated. The direction that placing a neuron in each of the two parts creates a lower modeling error is chosen. Once again, the worst LOLIMOT is selected and broken axis-orthogonally. This process continues until a stopping criterion is met, which can be the number of neurons or reaching an acceptable level of modeling error.

One important characteristic of this method is its very fast training, and the simplicity in the linear models used for each part.

In this section, the LMN toolbox is used. Finally, the validation data is simulated by the LMN model, and the results are presented in the following figures.

```
LMN.maxNumberOfLM = x;                    % Termination criterion
for maximal number of LLMs
```

```
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 24 local models and 96 parameters: J = 0.038613

Net 22 with 22 LMs and 88 parameters is suggested as the model with the best complexity trade-off.
```

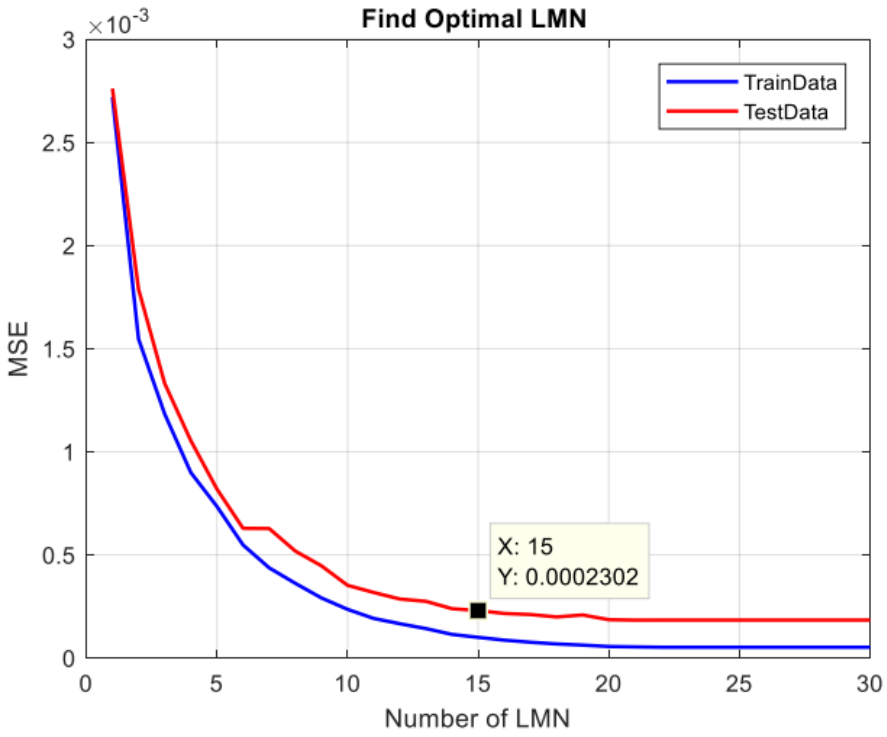


Figure 97 Error-neuron in LMN algorithm- ***without noise***

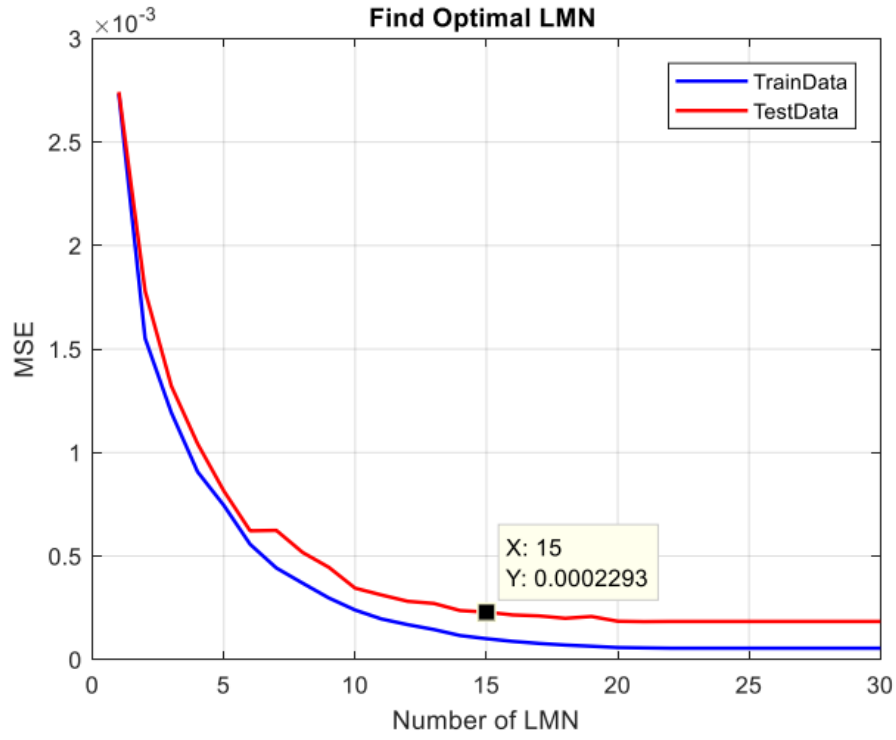


Figure 98 Error-neuron in LMN algorithm- ***low noise***

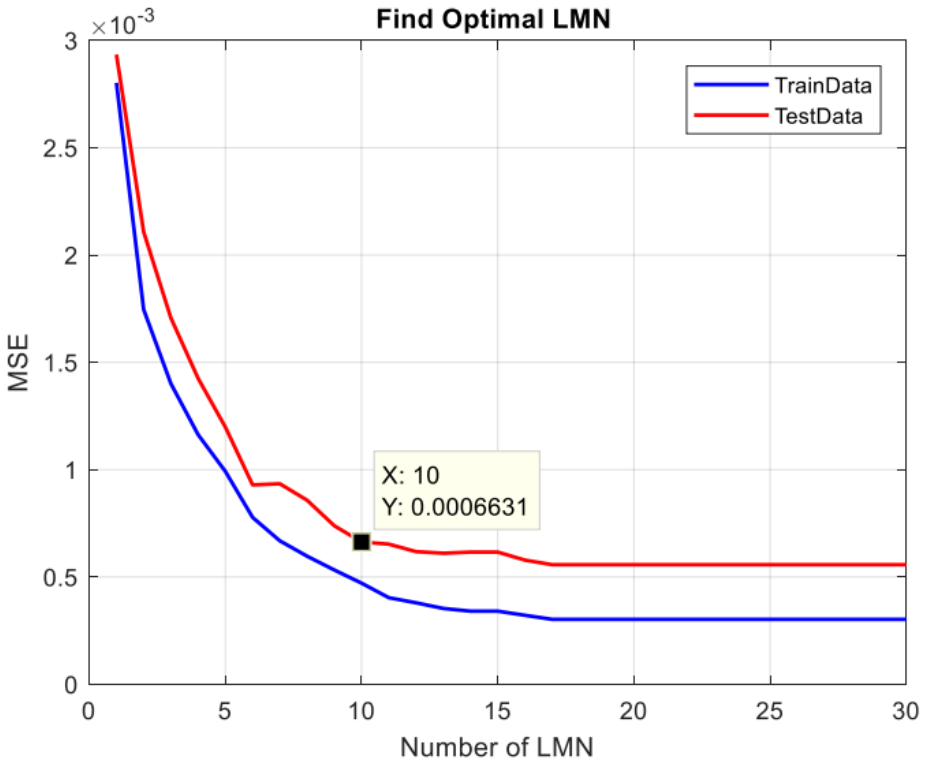


Figure 99 Error-neuron in LMN algorithm- ***moderate noise***

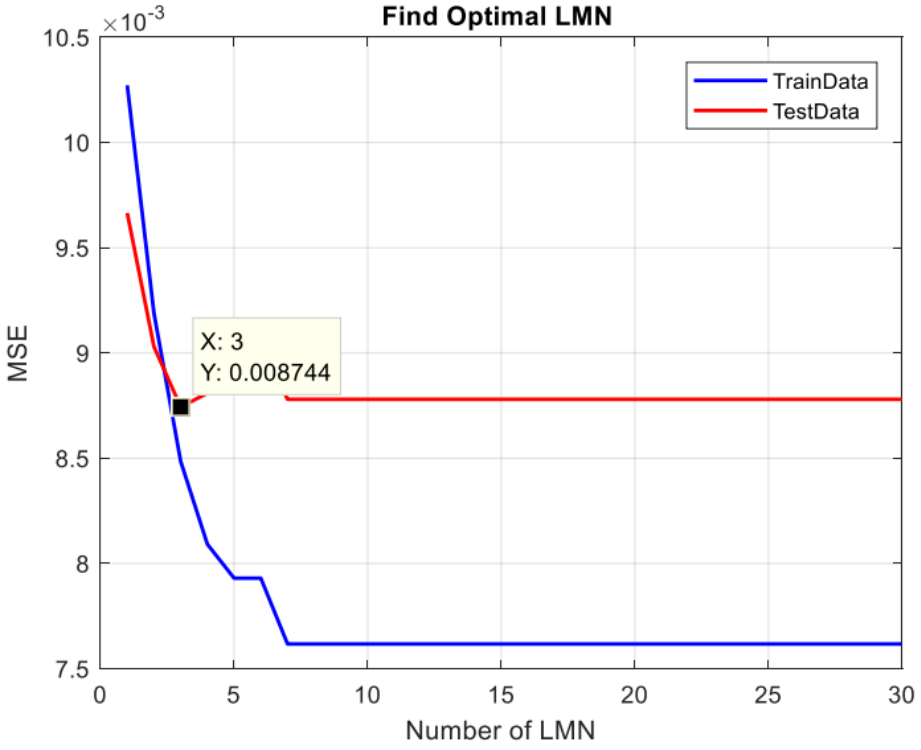


Figure 100 Error-neuron in LMN algorithm- ***high noise***

15, 15, 10, 3 or 10 neurons could be the optimal number of neurons.

Now the validation results as follows:

```
Final net has 15 local models and 60 parameters: J = 0.054215

Net 15 with 15 LMs and 60 parameters is suggested as the model with the best complexity trade-off.

MSE_Validation =

   3.0118e-04
```

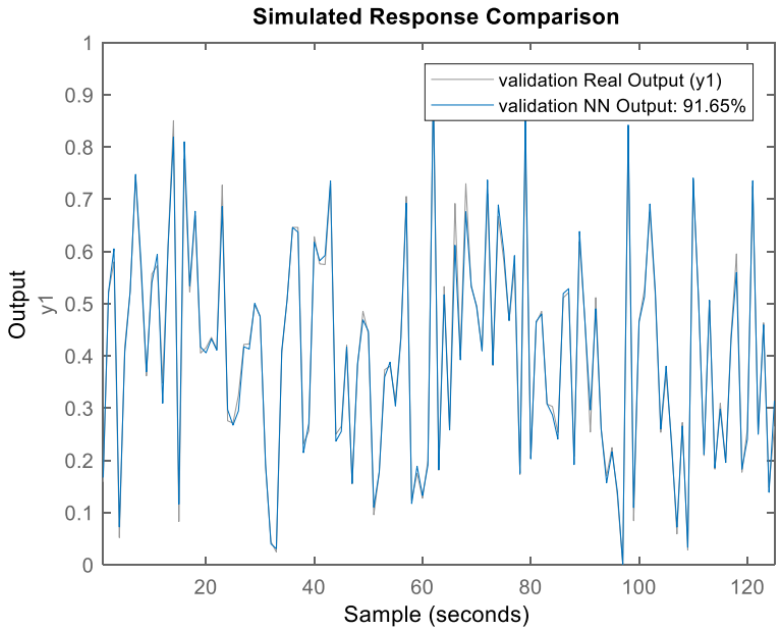


*Figure 101 The estimated and actual output- LMN algorithm-**without nosie***

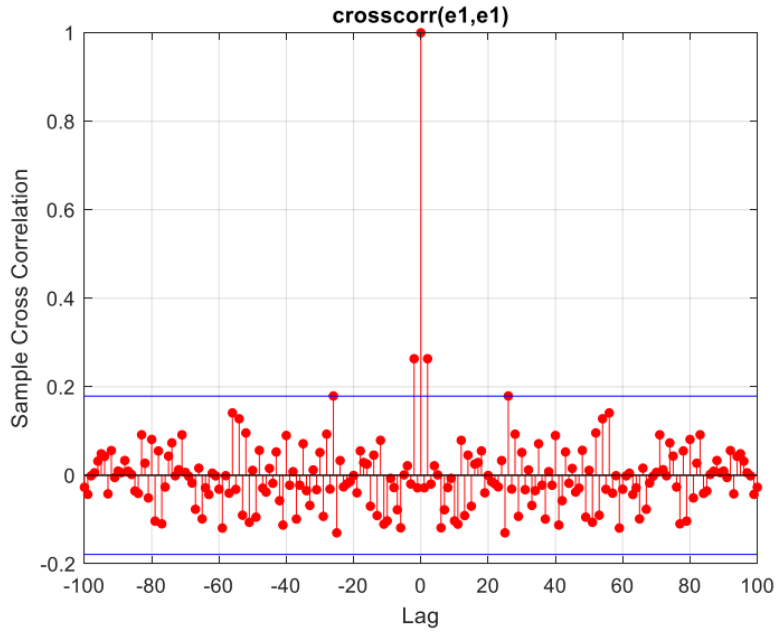


*Figure 102 Error cross-correlation **without nosie***

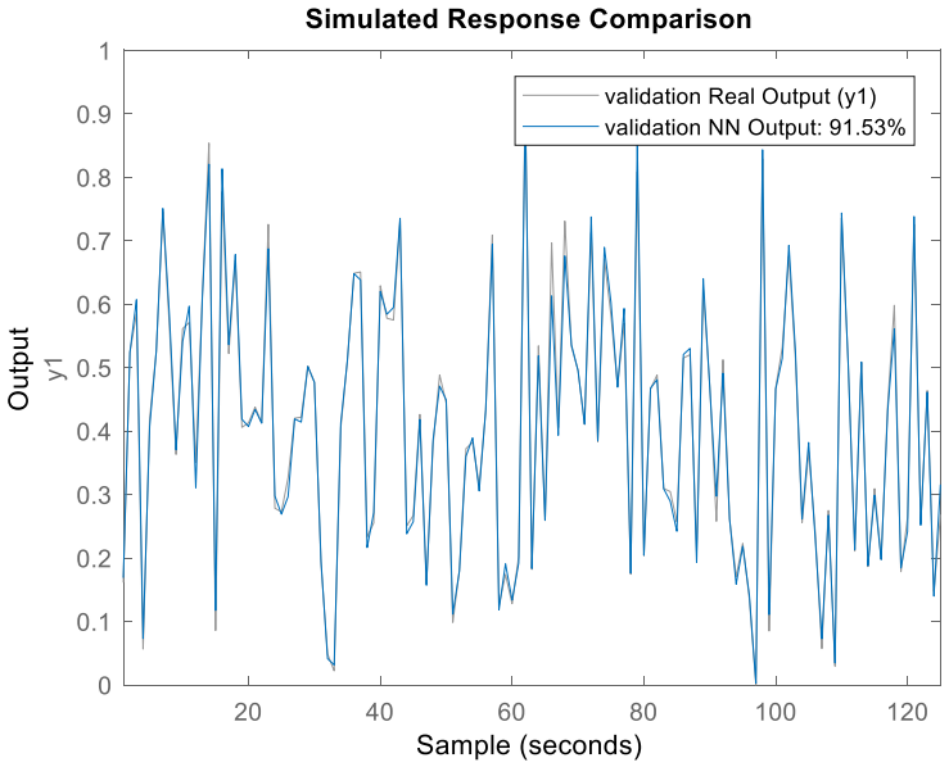


*Figure 103 The estimated and actual output- LMN algorithm-**lwo nosie***

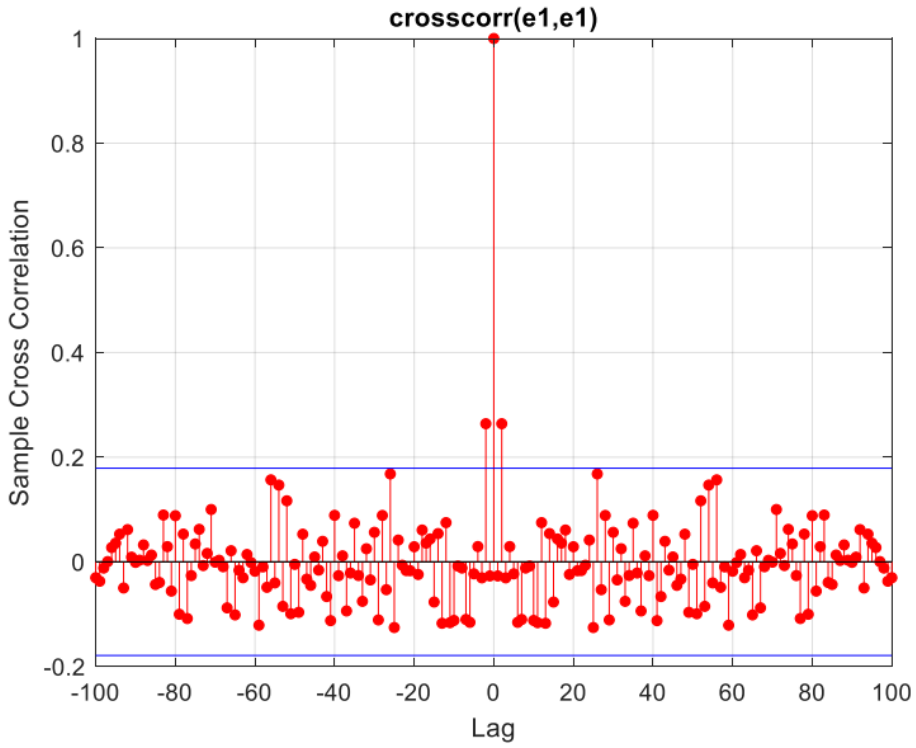


*Figure 104 Error cross-correlation **lwo nosie***

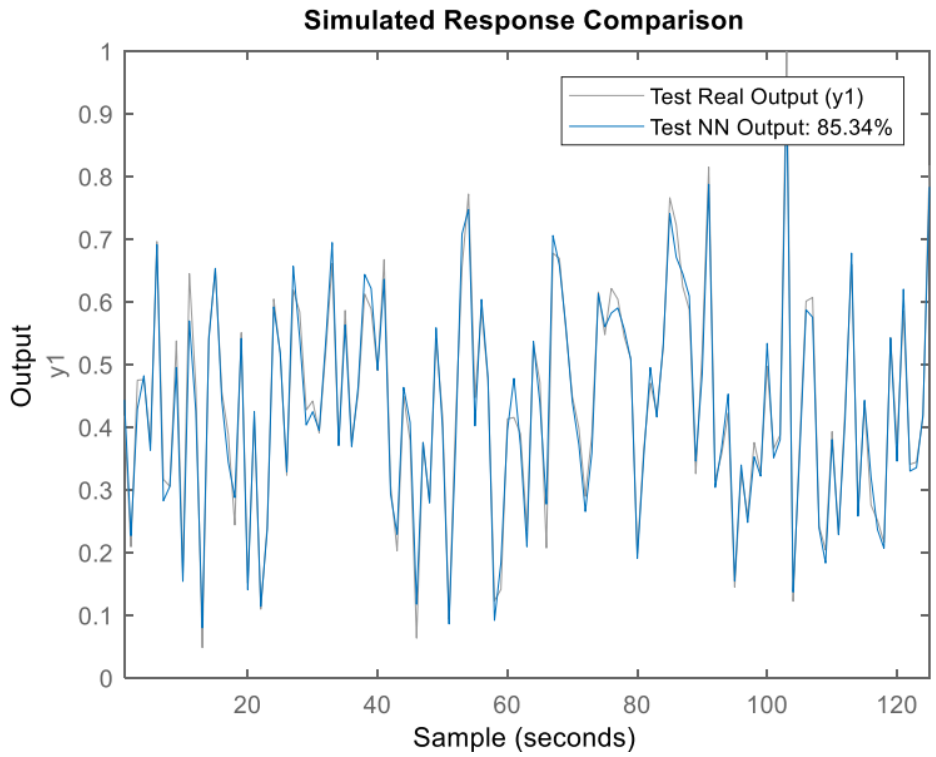


*Figure 105 The estimated and actual output- LMN algorithm-**moderate Nosie***

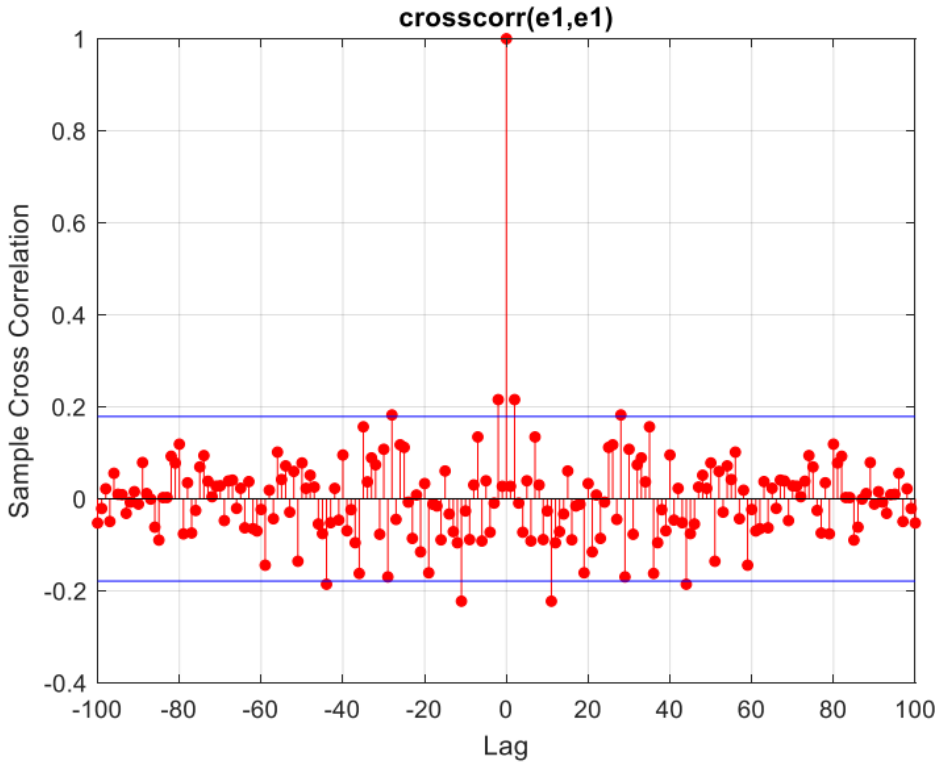


*Figure 106 Error cross-correlation **moderate nosie***

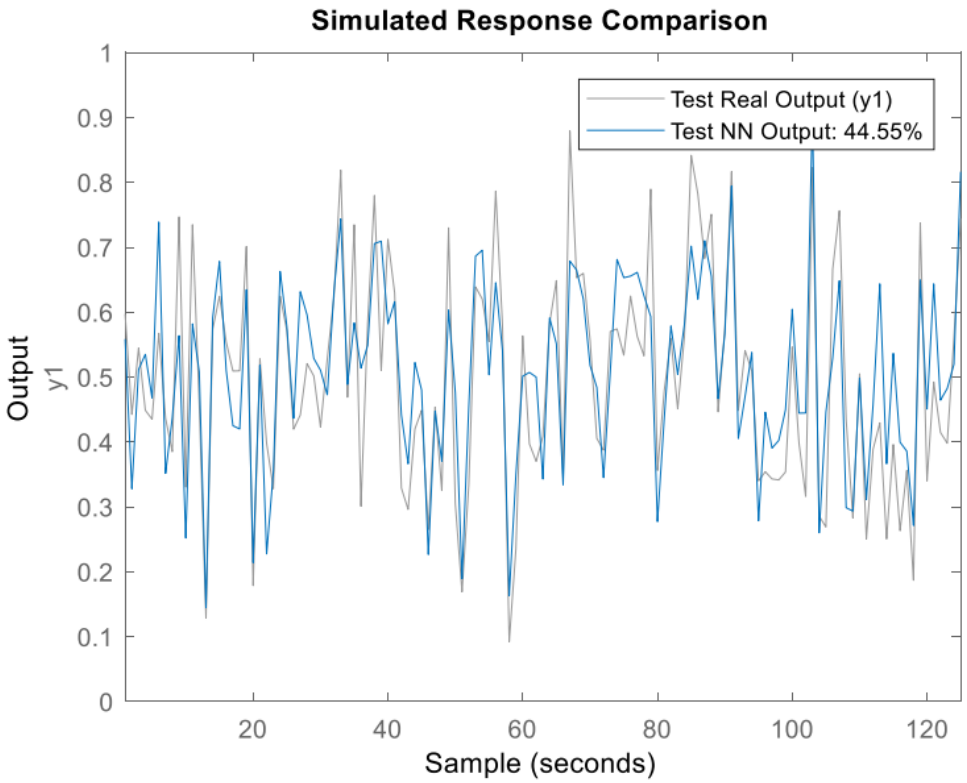


*Figure 107 The estimated and actual output- LMN algorithm-**high Nosie***

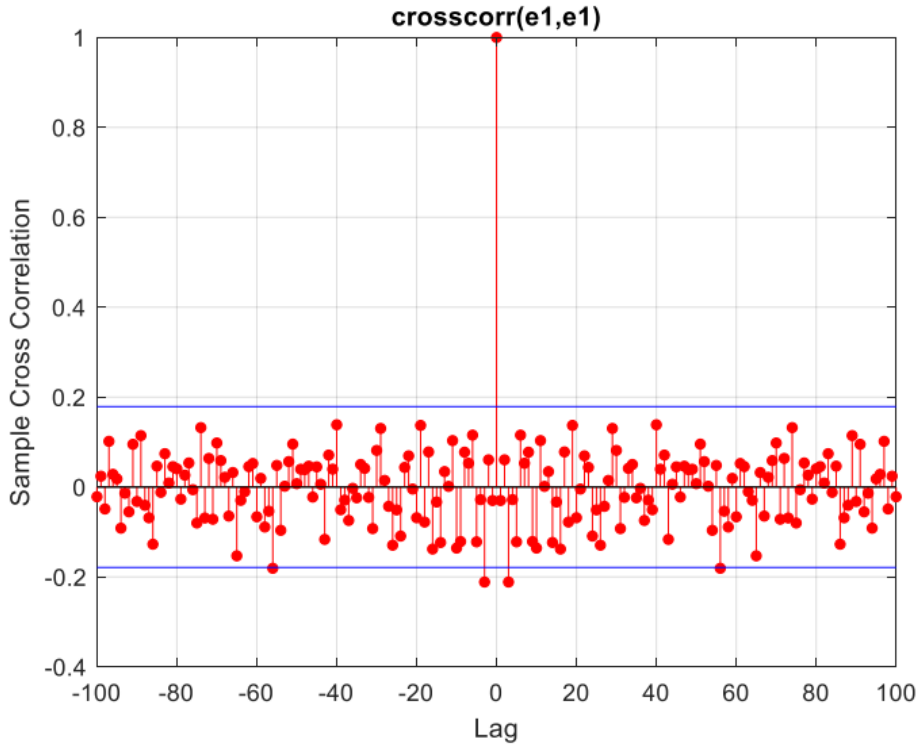


*Figure 108 Error cross-correlation **high nosie***

For each of the three noise levels, the error shows almost whitening, indicating that important dynamics are captured, and the model performs well in terms of credibility. It can be observed that as the noise increases, the percentage of fit decreases. In the case of noise, the higher variance of the noise leads to a worse estimation, and this is attributed to the network learning the noise.

Comparing this to similar cases, the accuracy of the MLP single-layer, MLP double-layer, and NRBF networks is higher. The fit percentages of these methods can be compared to the RBF method.

3.2)

(Unbiasedness Criteria) Sugeno & Kang

Calculations have been performed using MATLAB. The data is divided into 4 parts, and each time, one of these 4 parts is considered as the test data, and the others are considered as training data. Therefore, 4 values of UC are obtained, and we take their average to calculate the final UC. The command **rand('state',0)** is also used to ensure that the initial conditions of the networks are the same, and the results are comparable.

Details about these sections are explained as comments in the MATLAB environment.

Contents

```
clc
clear all
close all

load( 'Data_1' );
load( 'Data_noise_H' );
addpath('C:\Users\Ghestionline.com\Downloads\LMNtool_Version_1.5.2')

%load( 'Data_noise_M' );
%load( 'Data_noise_H' );
total_P=[trn_P tst_P vrf_P];
total_T=[trn_T tst_T vrf_T];
```

Normalize input data

```
total_P(1,:)=(total_P(1,:)-min(total_P(1,:)))/(max(total_P(1,:))-min(total_P(1,:)));
total_P(2,:)=(total_P(2,:)-min(total_P(2,:)))/(max(total_P(2,:))-min(total_P(2,:)));
total_P(3,:)=(total_P(3,:)-min(total_P(3,:)))/(max(total_P(3,:))-min(total_P(3,:)));

total_T=(total_T-min(total_T))/(max(total_T)-min(total_T));

total_P=total_P';
total_T=total_T';
```

folds test in 4th section

---

folds test in 4th section

```
total_P1=total_P(1:348,:);
total_T1=total_T(1:348,:);
u_test1=total_P(349:464,:);
y_test1=total_T(349:464,:);
```

test in 1th section

```
total_P2=total_P(117:464,:);
total_T2=total_T(117:464,:);
u_test2=total_P(1:116,:);
y_test2=total_T(1:116,:);
```

test in 2th section

```
total_P3=[total_P(1:116,:);total_P(233:464,:)];
total_T3=[total_T(1:116,:);total_T(233:464,:)]
u_test3=total_P(117:232,:);
y_test3=total_T(117:232,:);
```

test in 3th section

```
total_P4=[total_P(1:232,:);total_P(349:464,:)]
total_T4=[total_T(1:232,:);total_T(349:464,:)]
u_test4=total_P(233:348,:);
y_test4=total_T(233:348,:);
```

step1_1

```
rand('state',0)

x=3;
LMN = lolimot;
LMN.input = total_P1;
LMN.output = total_T1;

LMN.xRegressorDegree = 1;       % use 1st order polynoms for local models (default: 1)
LMN.maxNumberOfLM = x;         % Termination criterion for maximal number of LLMs
LMN.minError = 0.01;            % Termination criterion for minimal error
LMN.kStepPrediction = 0;        % Static model
LMN.smoothness = 1;             % Less overlap between the validity functions
LMN.history.displayMode = true; % display information

LMN = LMN.train;
y_aa = calculateModelOutput(LMN, total_P1, total_T1);

y_ba = calculateModelOutput(LMN, u_test1, y_test1);
```

step1_2

```
rand('state',0)

x=3;
LMN2 = lolimot;
LMN2.input = u_test1;
LMN2.output = y_test1;

LMN2.xRegressorDegree = 1;       % use 1st order polynoms for local models (default: 1)
LMN2.maxNumberOfLM = x;         % Termination criterion for maximal number of LLMs
LMN2.minError = 0.01;            % Termination criterion for minimal error
LMN2.kStepPrediction = 0;        % Static model
LMN2.smoothness = 1;             % Less overlap between the validity functions
LMN2.history.displayMode = true; % display information

LMN2 = LMN2.train;
y_ab = calculateModelOutput(LMN2, total_P1, total_T1);

y_bb = calculateModelOutput(LMN2, u_test1, y_test1);

uc1=sqrt(sum((y_ab-y_aa).^2)+sum((y_ba-y_bb).^2))
```

Carry out these steps on each fold:

Without noise:

$uc_1 = 0.6066$

$uc_2 = 0.5625$

$uc_3 = 0.5946$

$uc_4 = 0.6072$

$$UC = \frac{uc_1 + uc_2 + uc_3 + uc_4}{4} = 0.5927$$

Low noise:

$uc_1 = 0.6076$

$uc_2 = 0.5614$

$uc_3 = 0.6002$

$uc_4 = 0.6102$

$$UC^2 = \frac{uc_1 + uc_2 + uc_3 + uc_4}{4} = 0.5949$$

Moderate noise

$uc_1 = 0.6421$

$uc_2 = 0.8205$

$uc_3 = 0.2204$

$uc_4 = 0.6106$

$$UC = \frac{uc_1 + uc_2 + uc_3 + uc_4}{4} = 0.5734$$

High noise:

$uc_1 = 1.0894$

$uc_2 = 0.6602$

$uc_3 = 0.5706$

$uc_4 = 0.4598$

$$UC = \frac{uc_1 + uc_2 + uc_3 + uc_4}{4} = 0.6950$$

The smaller the UC value, the more suitable and general the modeling is.

3.3)

In comparison with the numbers from the article (although the system is different), it can be said that the value is relatively small, indicating that a suitable modeling has been achieved. As the noise becomes stronger, this criterion becomes larger, indicating that the model needs to be designed in a way that the increase in this value is not significant.

3.4)

Implementing LOLIMAT

Without noise:

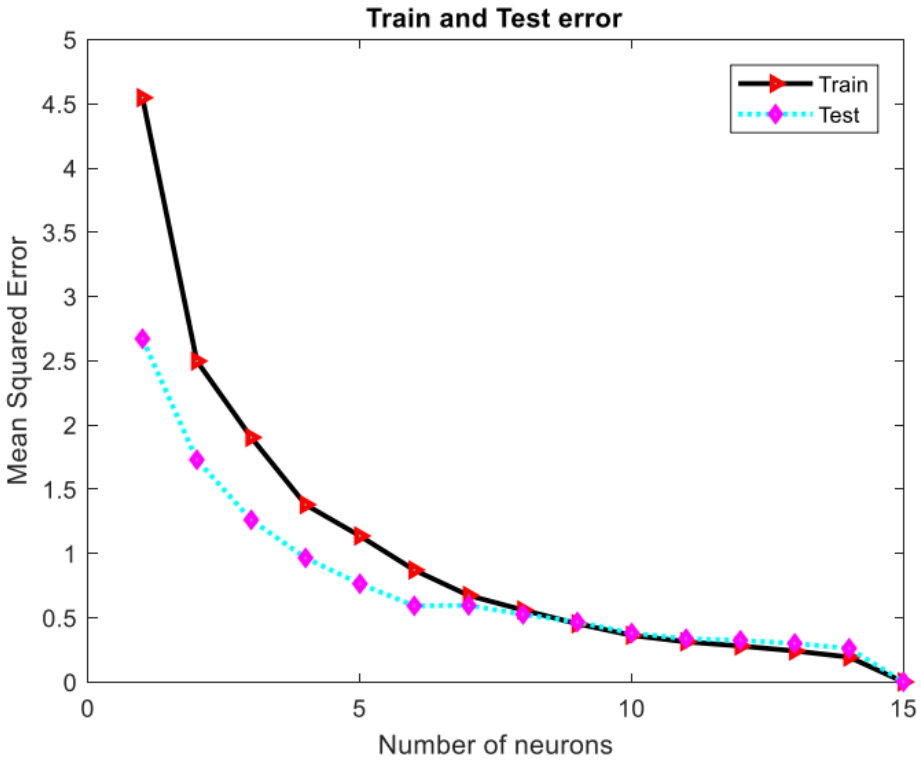


*Figure 109 test and train error in LOLIMOT implementation-**without noise***

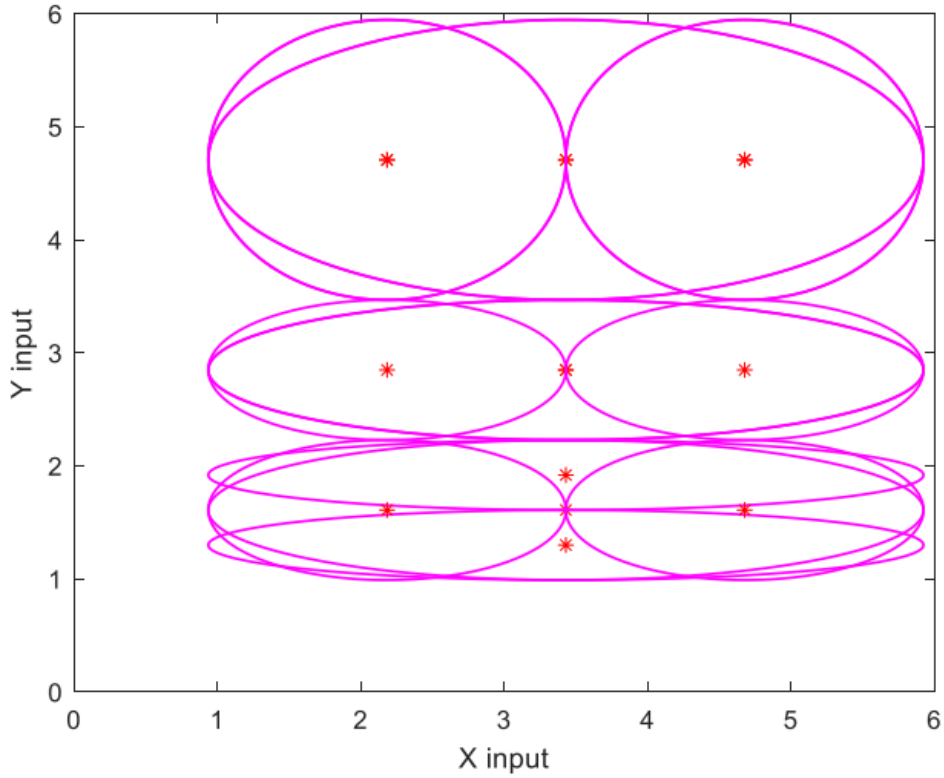


*Figure 110 How the functions break-**without noise***

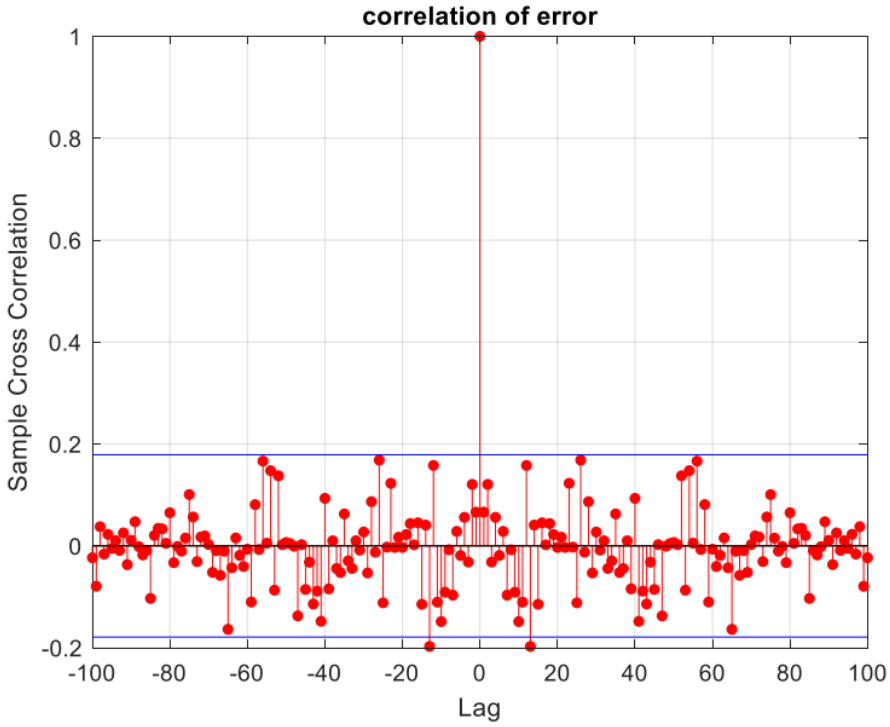


*Figure 111 Error Cross-correlation-**without noise***

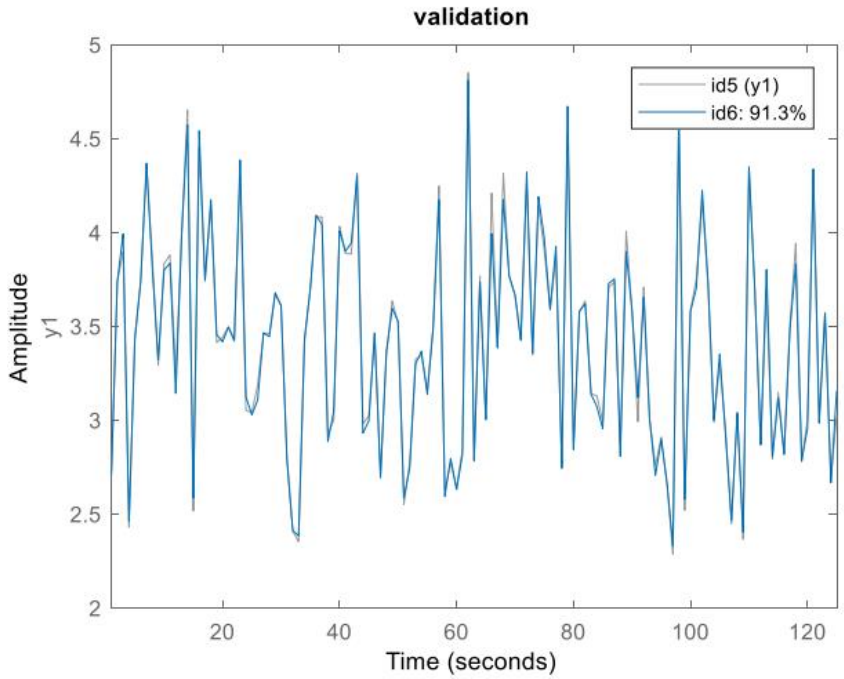


*Figure 112 The actual and estimated system output-LOLIMOT- **Without noise***

Low noise:

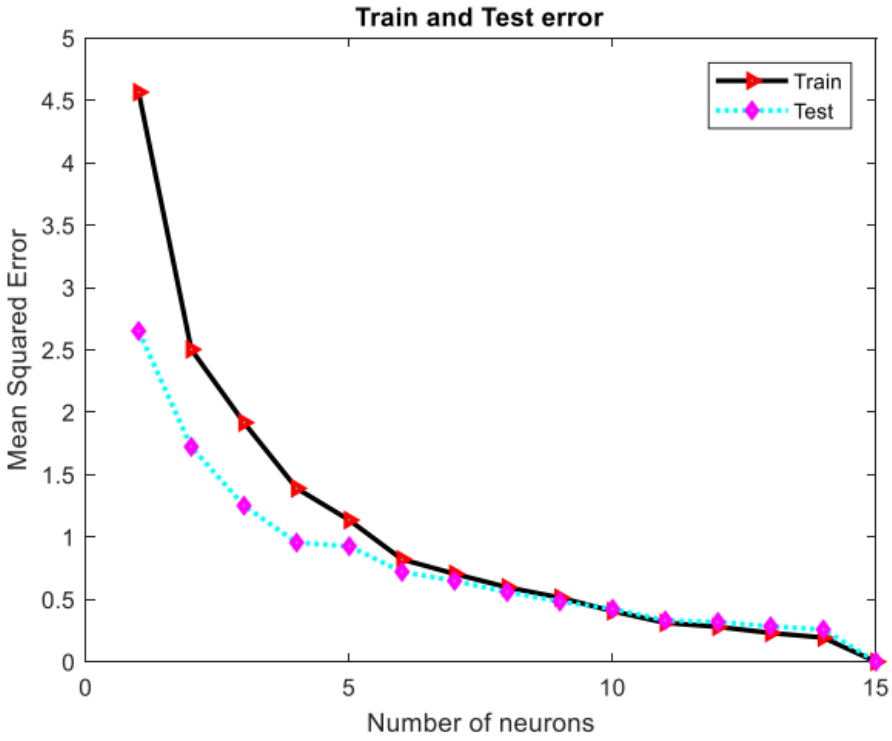


*Figure 113 test and train error in LOLIMOT implementation-**low noise***

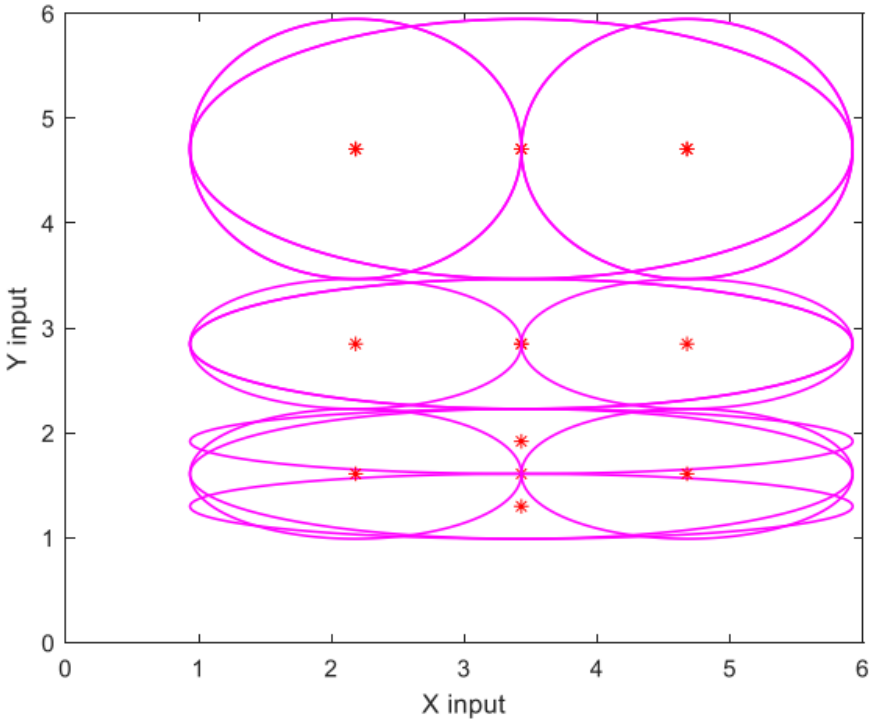


*Figure 114 How the functions break-**low noise***

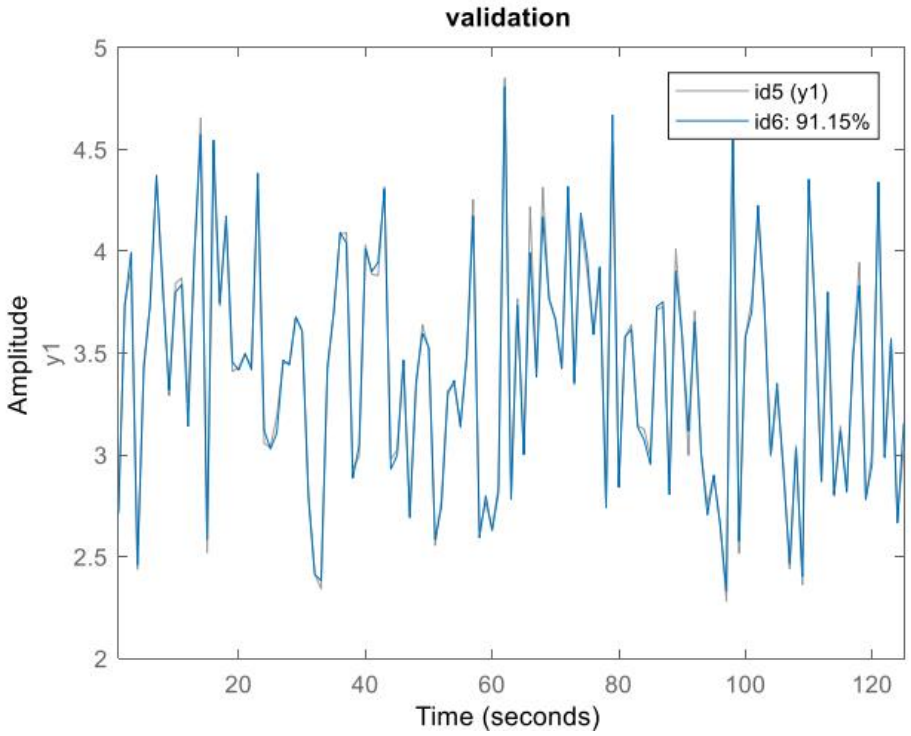


Figure 115 The actual and estimated system output-LOLIMOT- ***low noise***

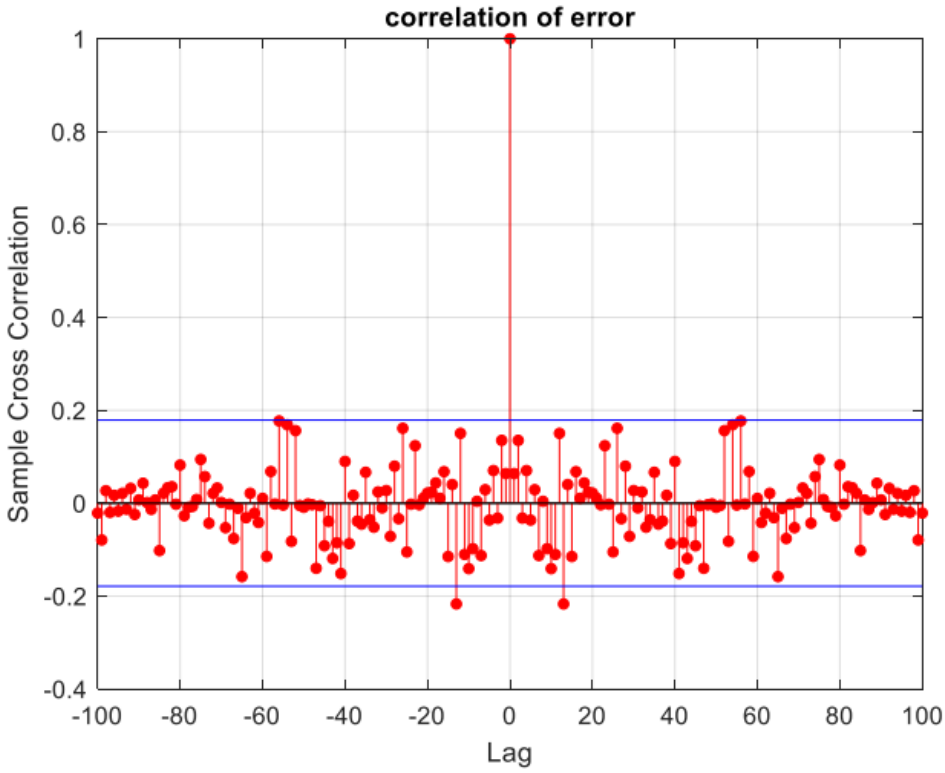


*Figure 116 Error Cross-correlation-****low noise***

Moderate Noise

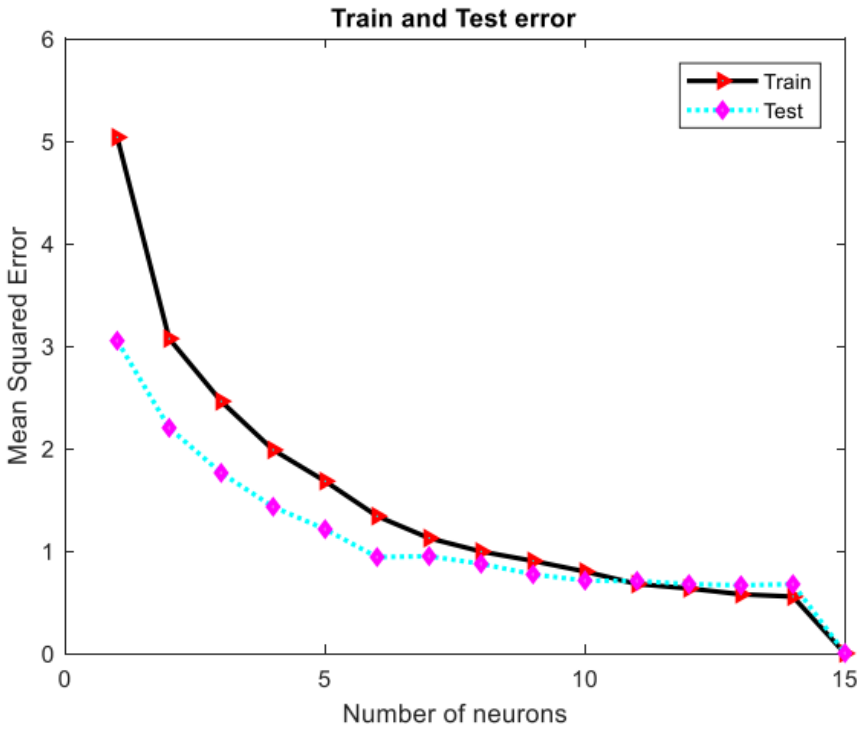


*Figure 117 test and train error in LOLIMOT implementation-**moderate noise***

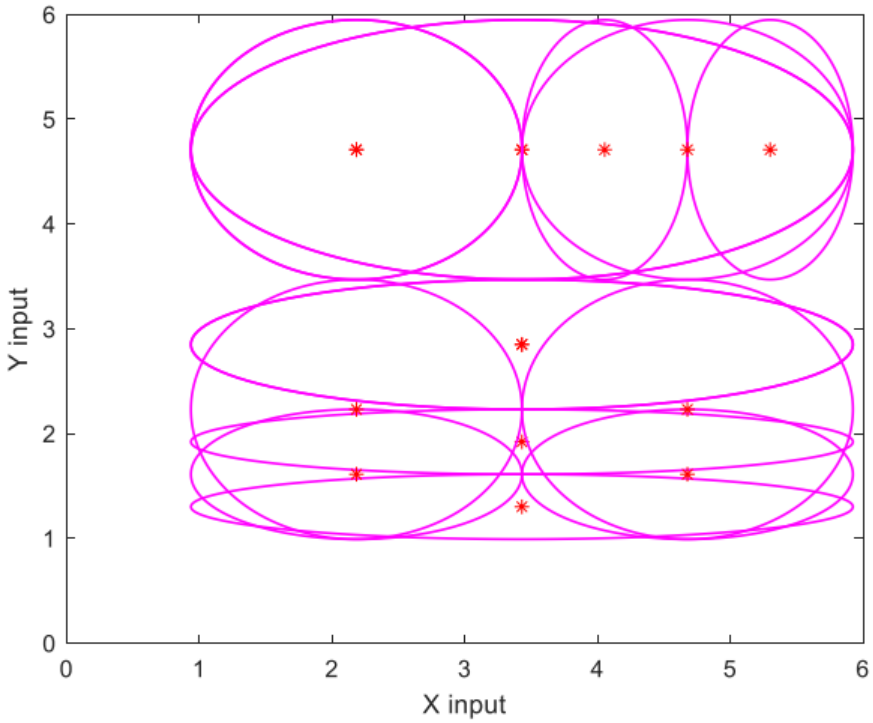


*Figure 118 How the functions break-**moderate noise***

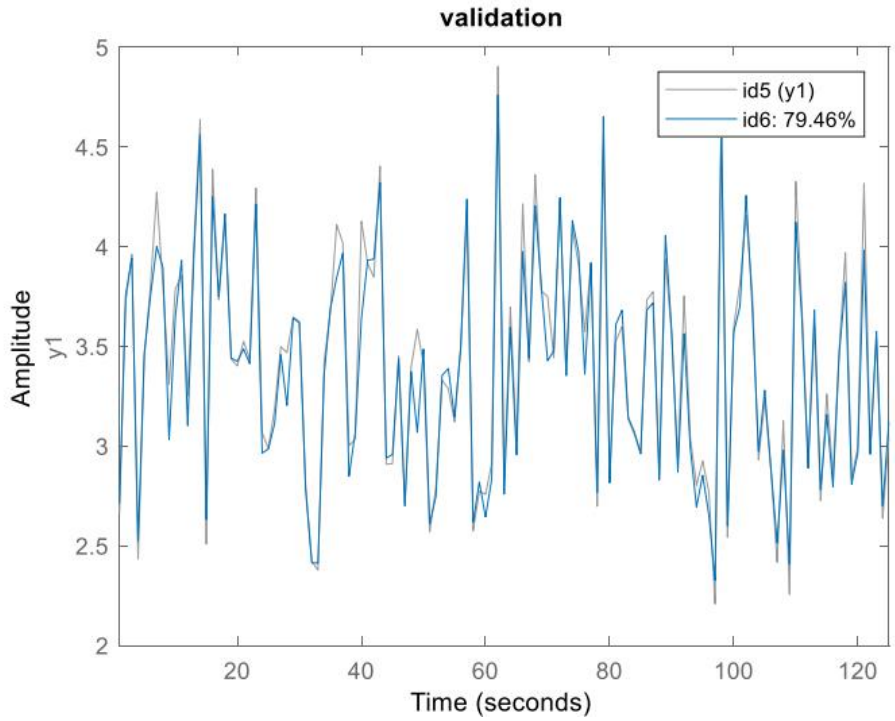


Figure 119 The actual and estimated system output-LOLIMOT- ***moderate noise***

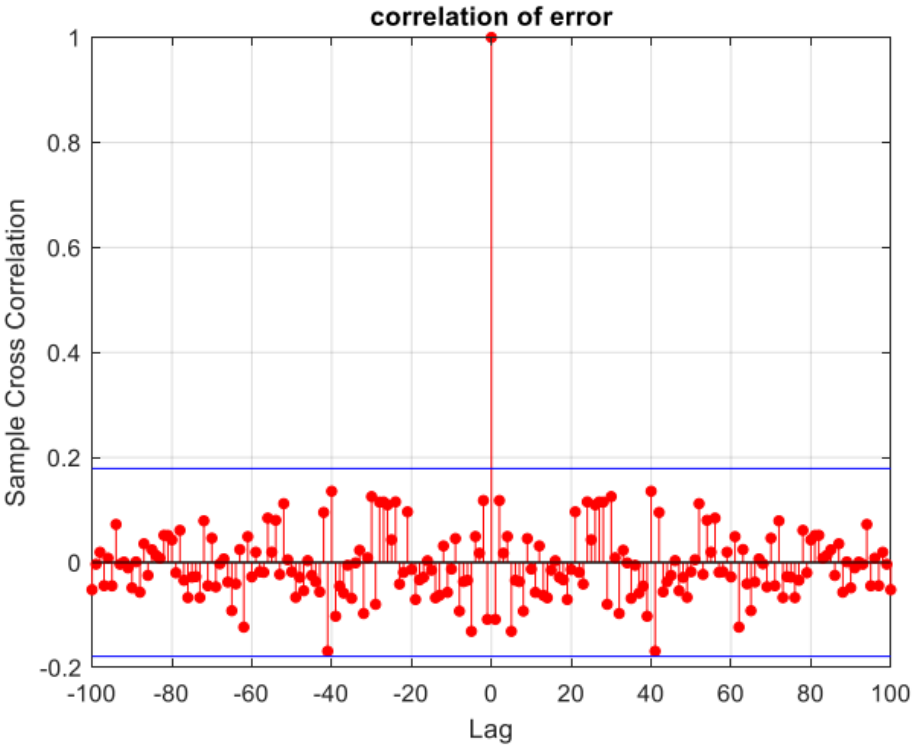


*Figure 120 Error Cross-correlation-****moderate noise***

High Noise

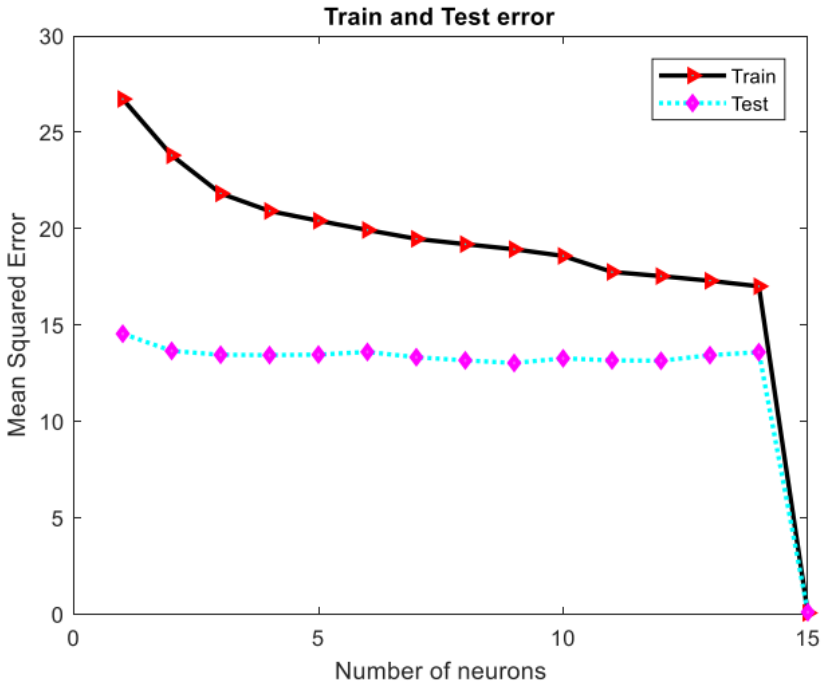


*Figure 121 test and train error in LOLIMOT implementation-**high noise***

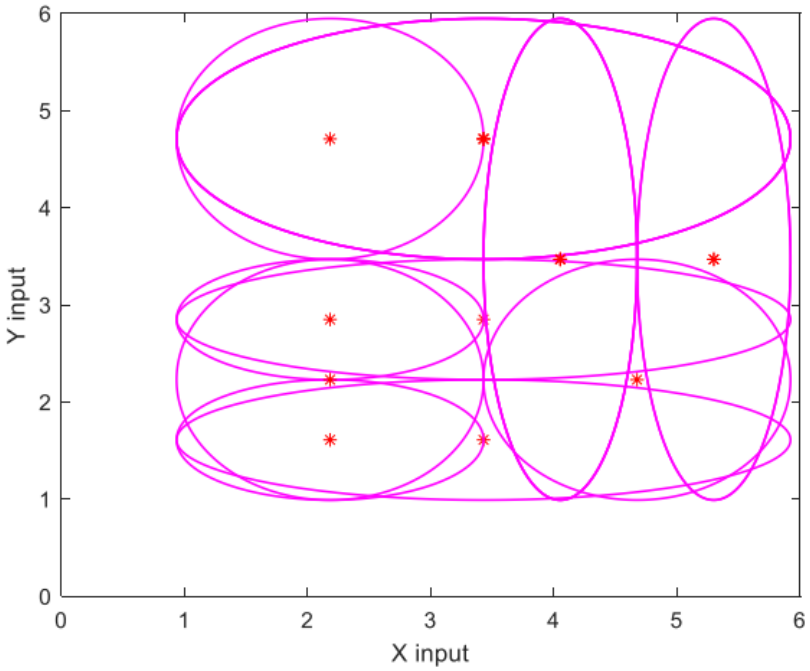


*Figure 122 How the functions break-**moderate noise***

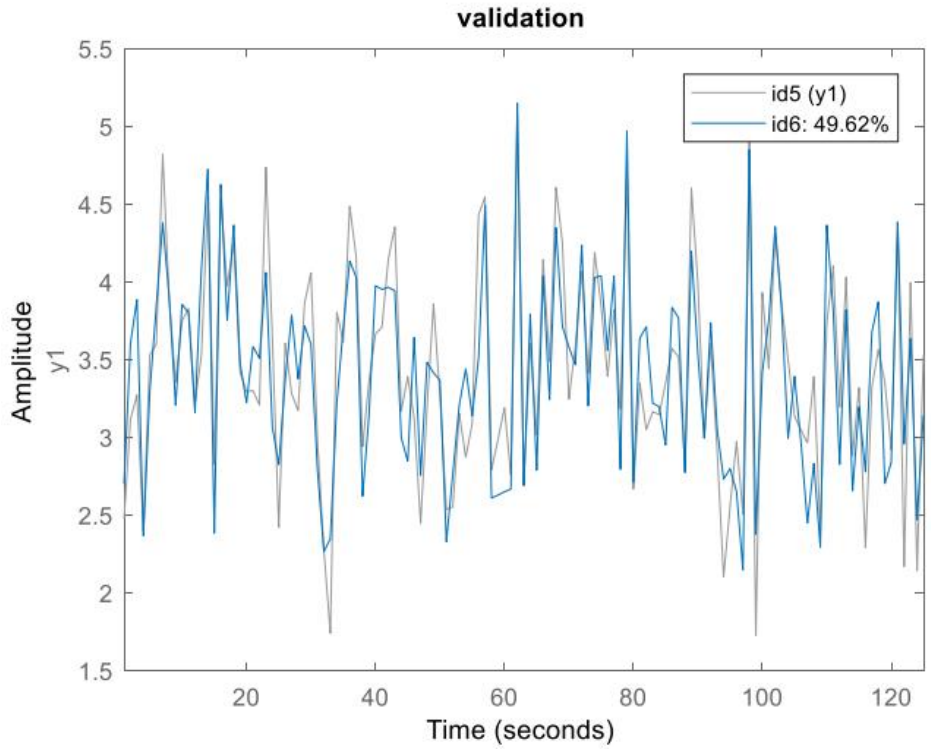


Figure 123 The actual and estimated system output-LOLIMOT- ***hogh noise***

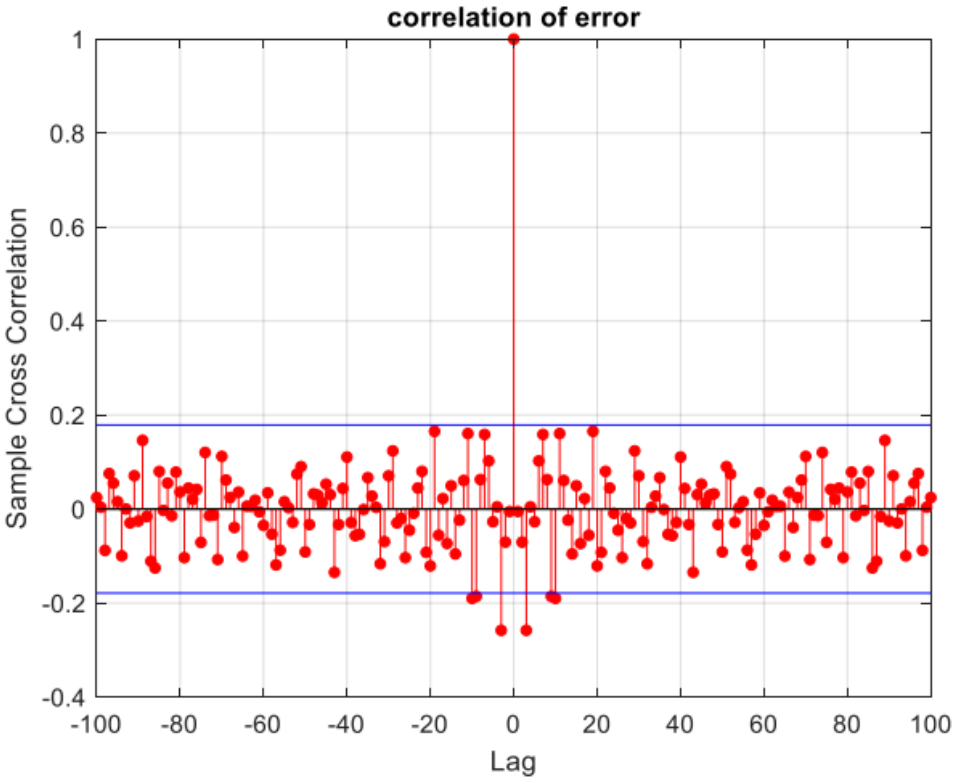


*Figure 124 Error Cross-correlation-****high noise***

For each of the three noises, whitening of the error has occurred, and the important dynamics of the model have been captured, resulting in a model with good credibility. It is observed that as the noise increases, the percentage of fit decreases. In the case of any amount of noise, as the noise variance increases, the estimation worsens, indicating that the network is learning the noise. Compared to a similar case, the accuracy of the MLP single-layer and two-layer network and NRBF is higher. The fit percentages of this method are comparable to the RBF method.

For the first two cases, the fit percentages for Toolbox and the implemented code are nearly equal. In the third case, the Toolbox code has a higher fit percentage, and in the fourth case, the implemented code has a higher fit percentage.

3-5)

ANFIS

We have considered one of the best options suitable for this data:

```
opt3 = genfisOptions('GridPartition');
opt3.InputMembershipFunctionType = ['gaussmf'];
opt3.OutputMembershipFunctionType= ['linear'];
fis=genfis([u_trn
y_trn1],y_trn,opt3,'FISType','mamdani');
```

We consider a Gaussian input function and a linear output function.

We adopt a Mamdani-type neural-fuzzy network. We are dealing with noise-free data.

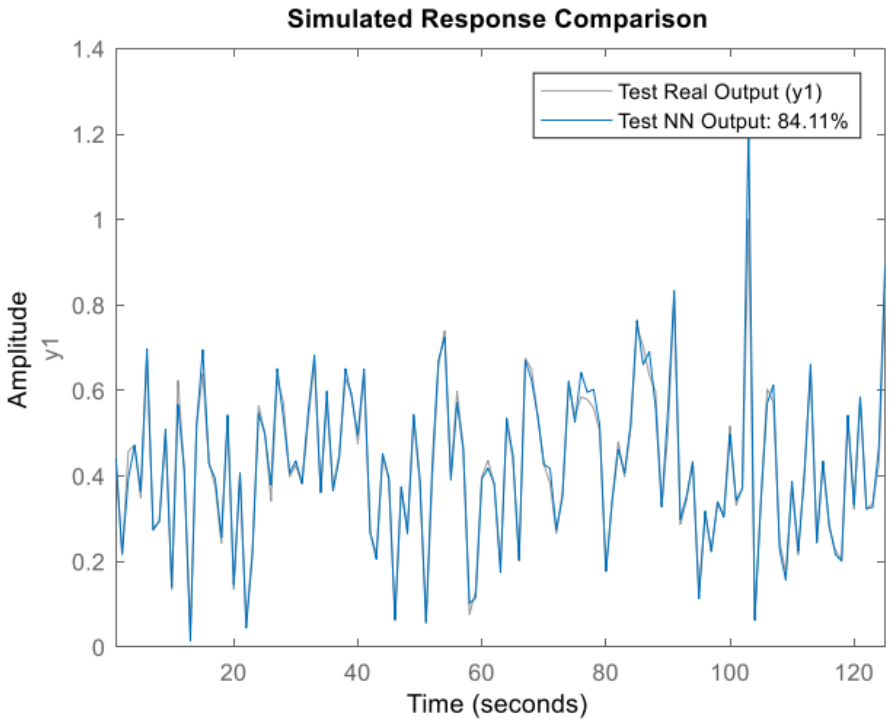


Figure 125 test plot- anfis first option

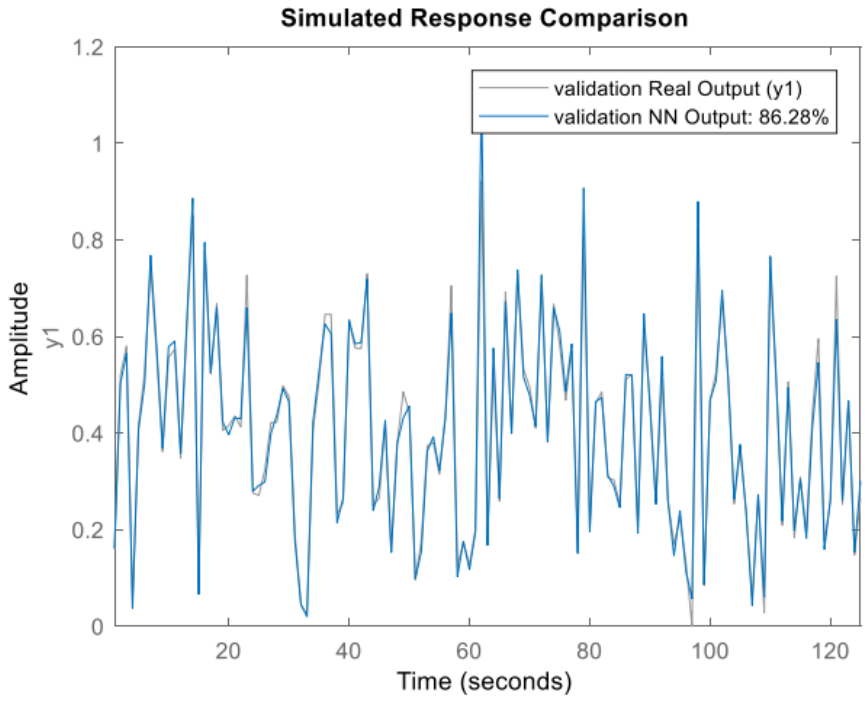


Figure 126 validation plot- ANFIS first option

Now we try the default option of the ANFIS toolbox:

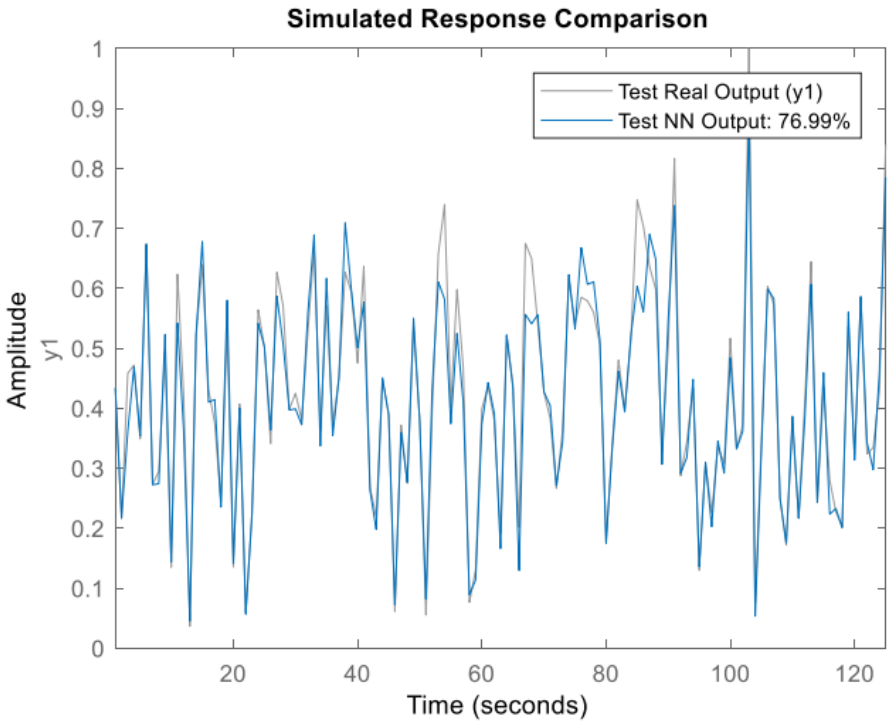


*Figure 127 Model output-default setting-on **test data***

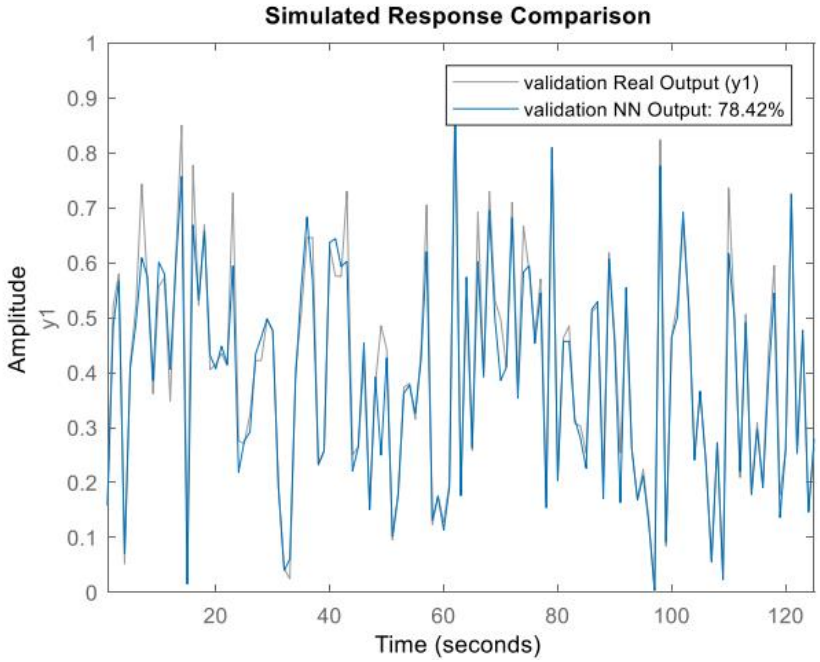


*Figure 128 the actual system and estimated system output-default setting-on **validation data***

If we try the next option:

opt2 = genfisOptions('FCMClustering','NumClusters',14);

If we use the above option the fit percentage will be more than eighty percent, but we cannot decide which model and setting to use before we examine the error whitening plot.

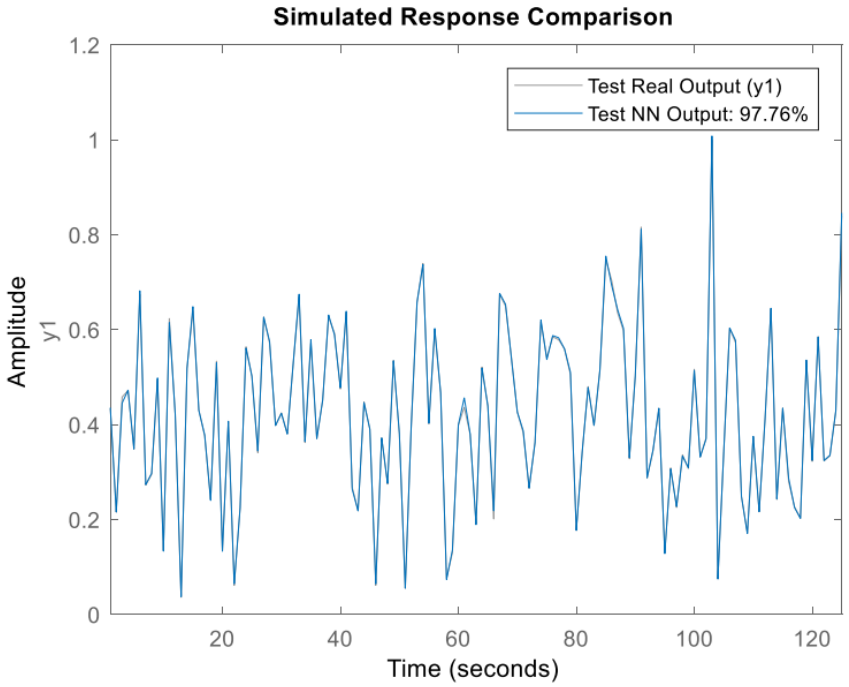


*Figure 129 the actual and estimated system output and fitness percentage-**Without noise-test data***

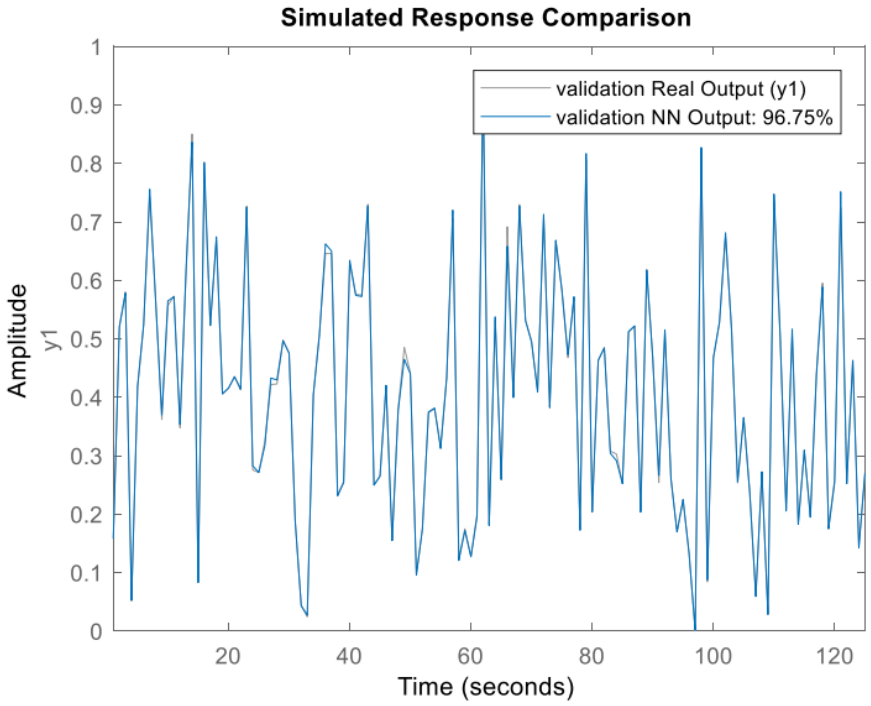


*Figure 130 the actual and estimated system output and fitness percentage-**Without noise-validation data***

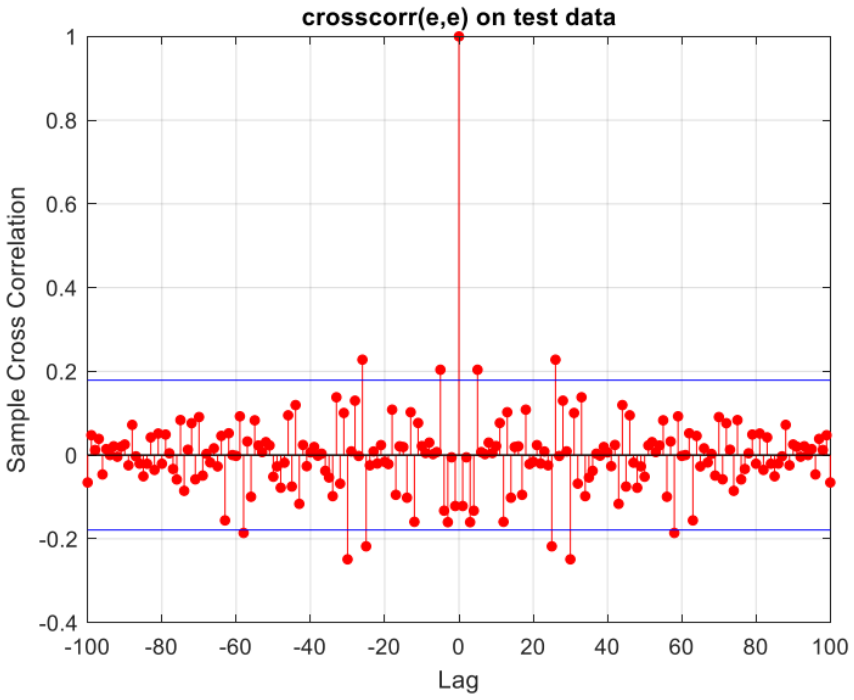


*Figure 131 error correlation **test data***

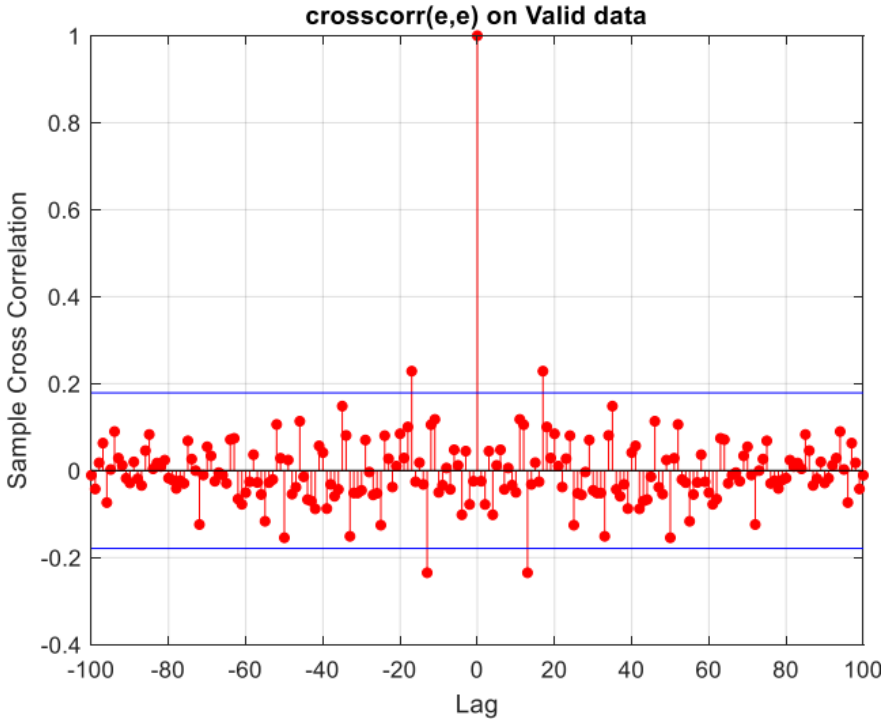


*Figure 132 error correlation **validation data***

Low noise:

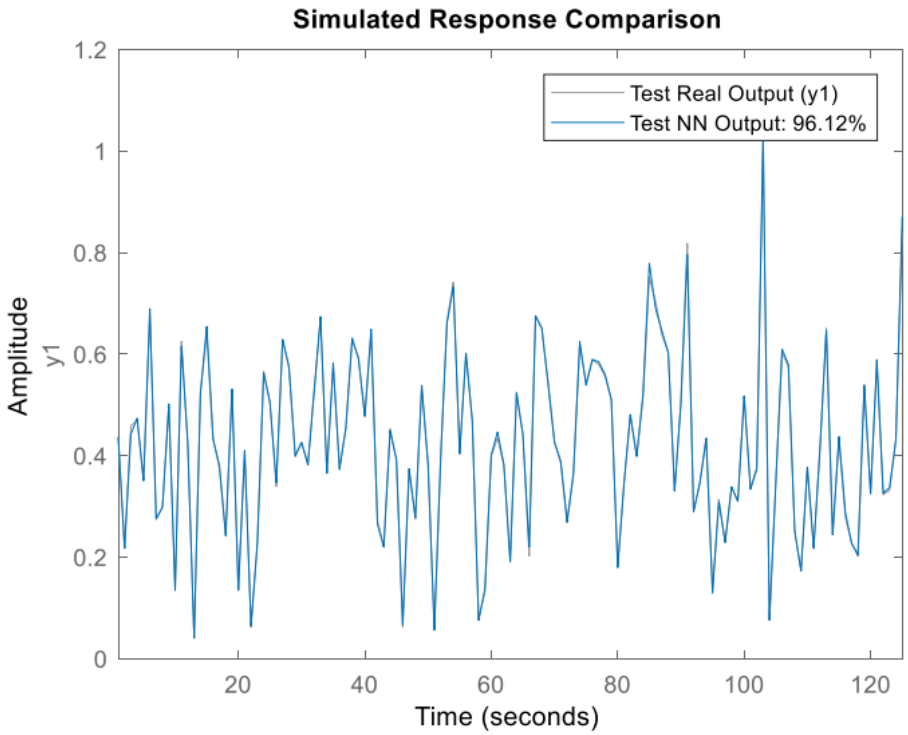


*Figure 133 the actual and estimated system output and fitness percentage-**low noise-test data***

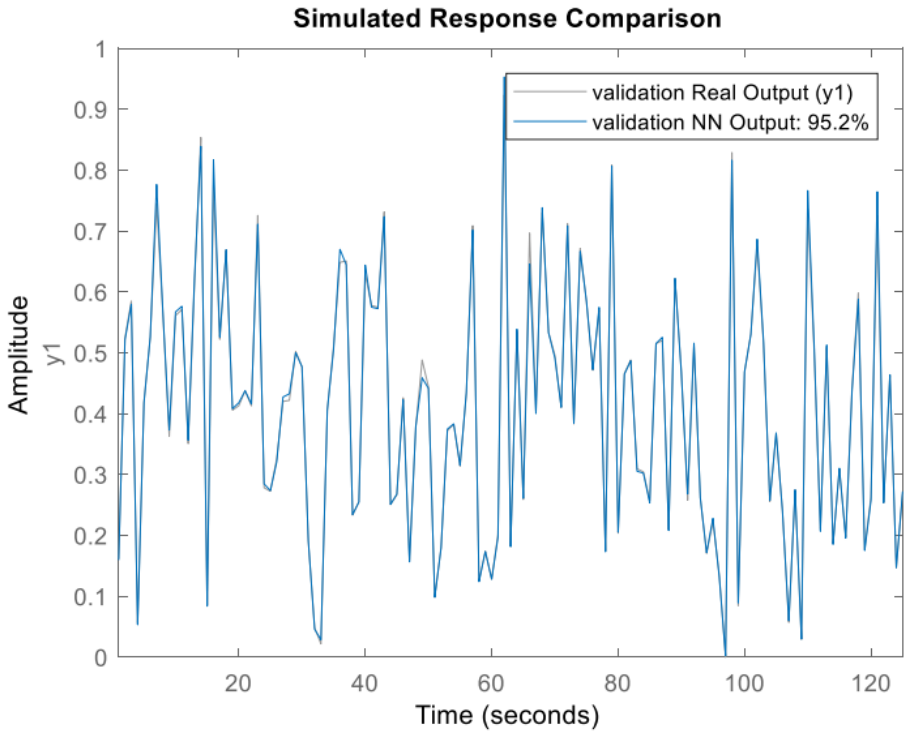


*Figure 134 the actual and estimated system output and fitness percentage-**low noise-validation data***

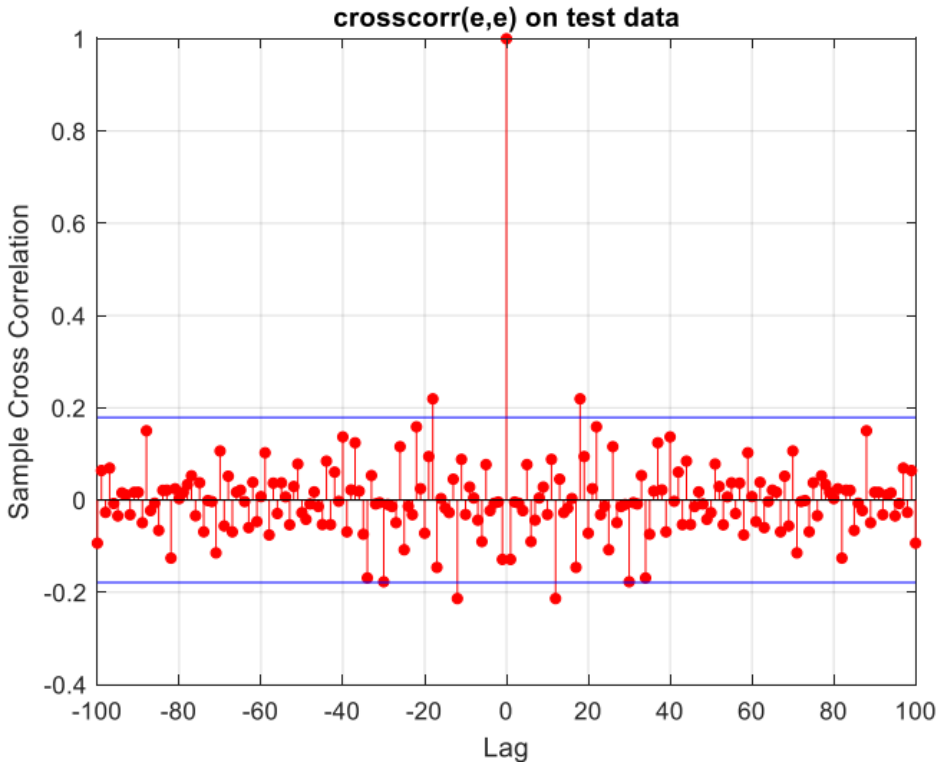


*Figure 135 Error correlation plot-**low noise test data***

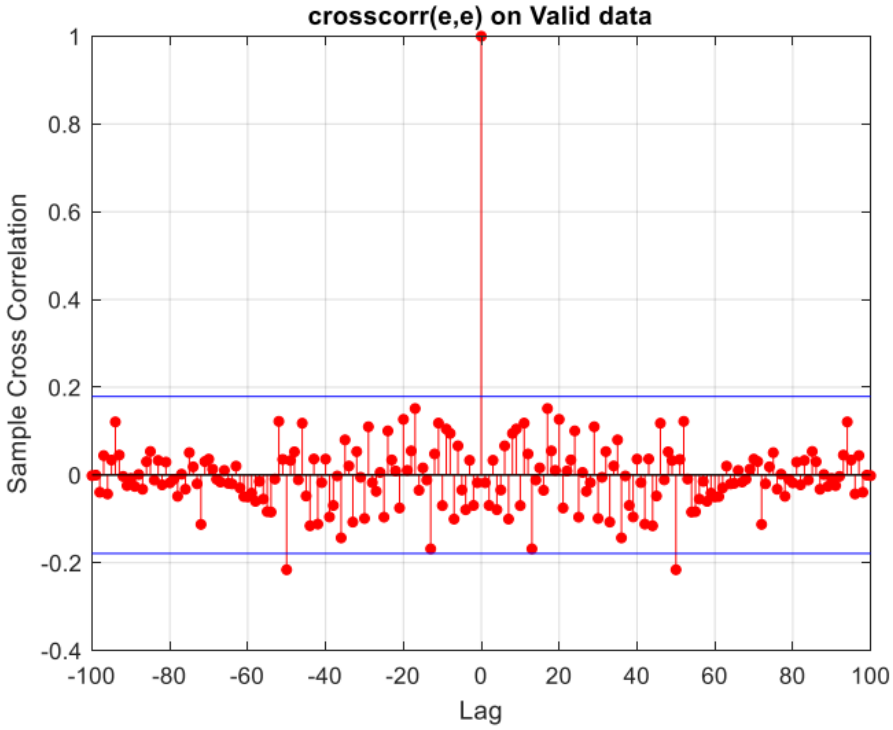


*Figure 136 Error correlation plot-**low noise validation data***

Moderate noise (30 epochs)

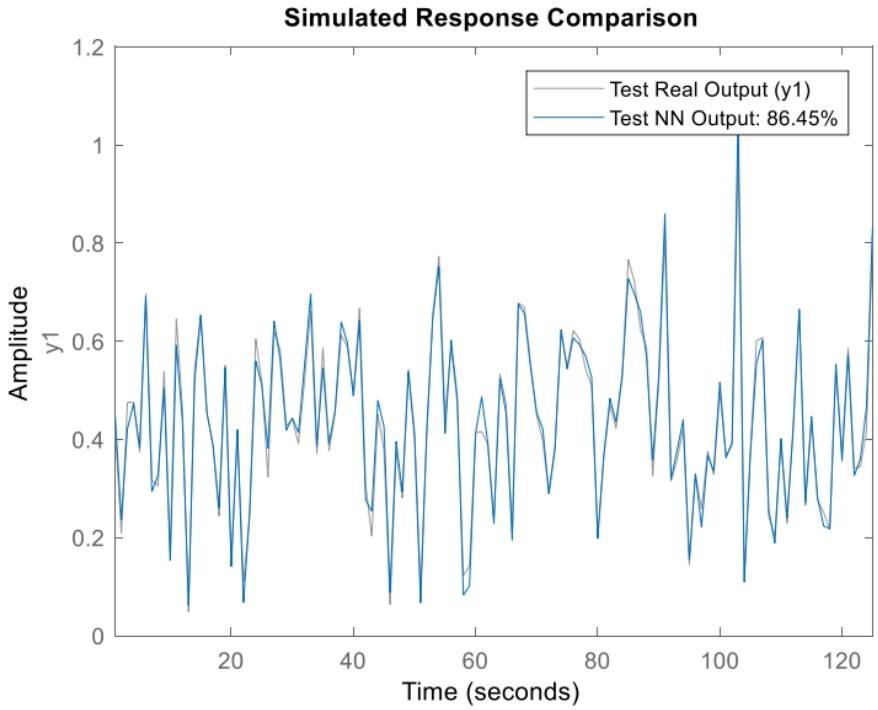


*Figure 137 the actual and estimated system output and fitness percentage-**moderate noise-test data***

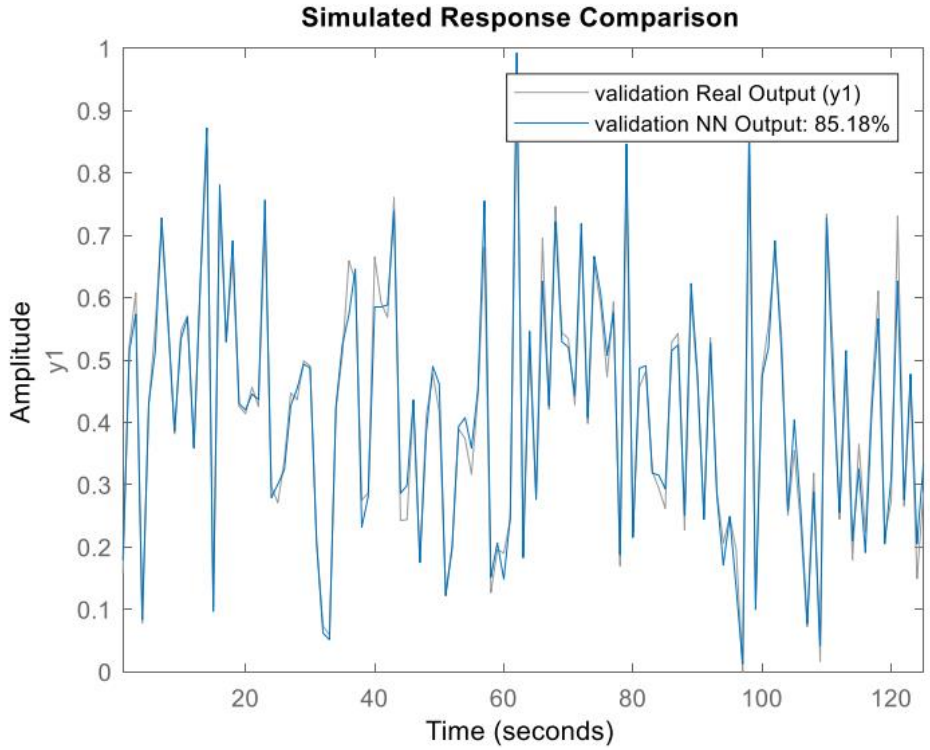


*Figure 138 the actual and estimated system output and fitness percentage-**moderate noise-validation data***

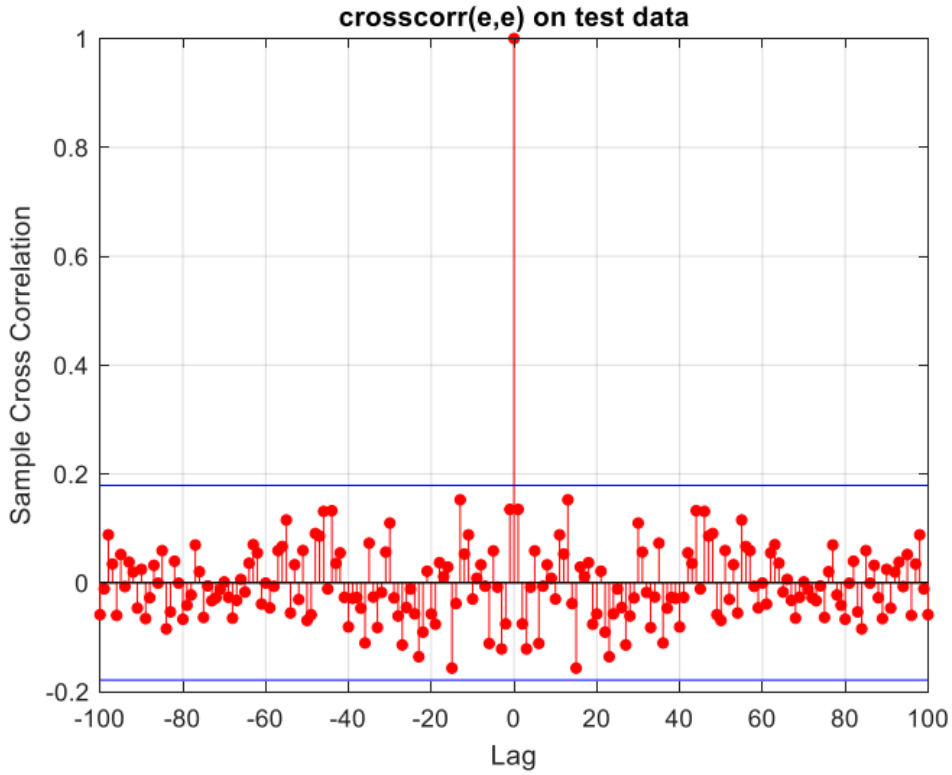


*Figure 139 Error correlation plot-**moderate noise test data***

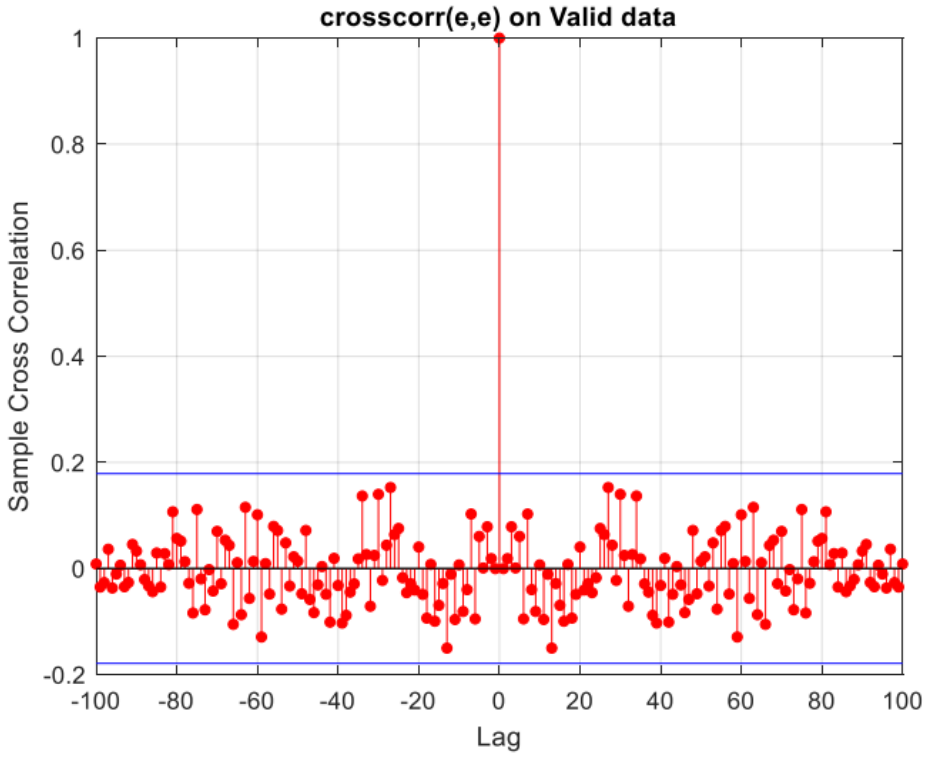


*Figure 140 Error correlation plot-**moderate noise validation data***

High noise:

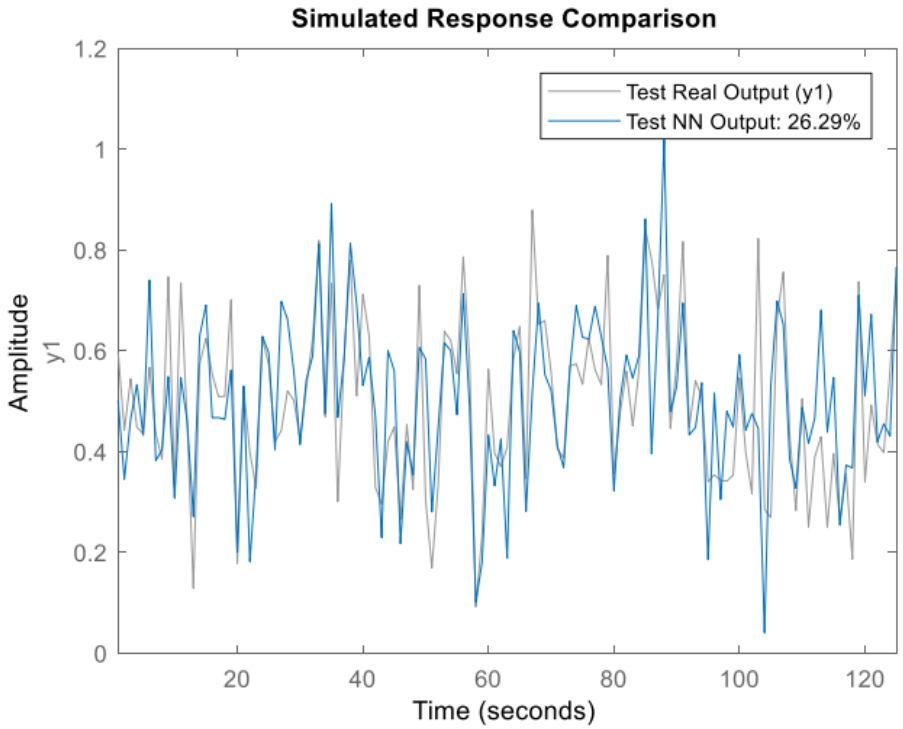


*Figure 141 the actual and estimated system output and fitness percentage-**high noise-test data***

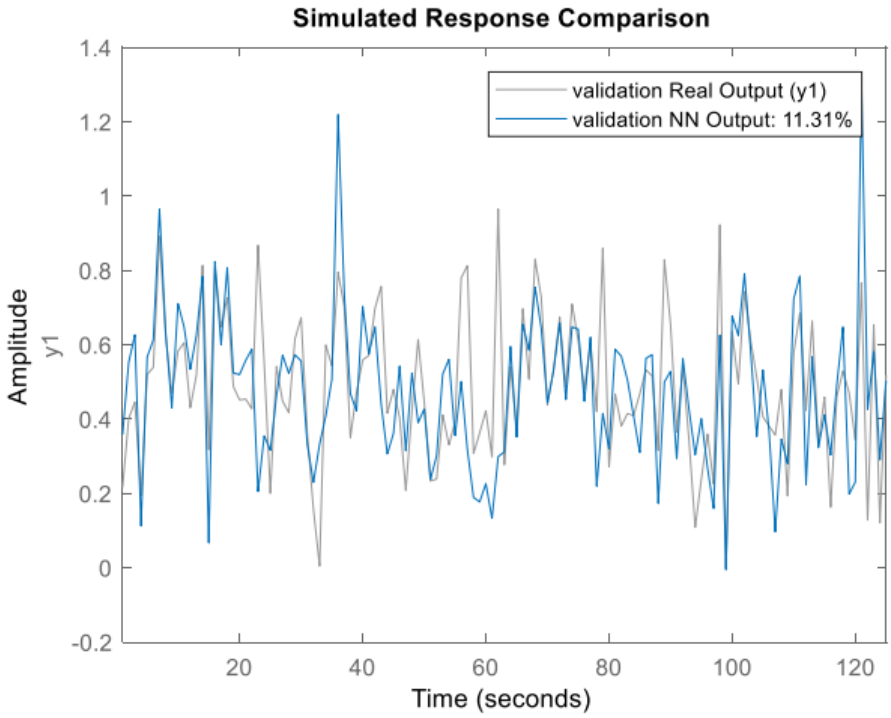


*Figure 142 the actual and estimated system output and fitness percentage-**high noise-validation data***

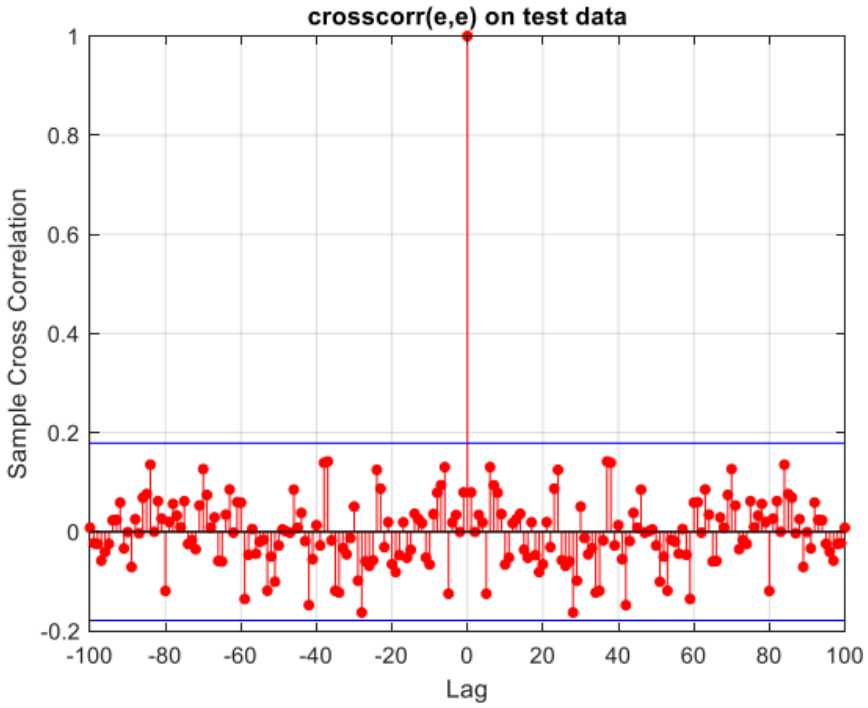


*Figure 143 Error correlation plot-**high noise test data***

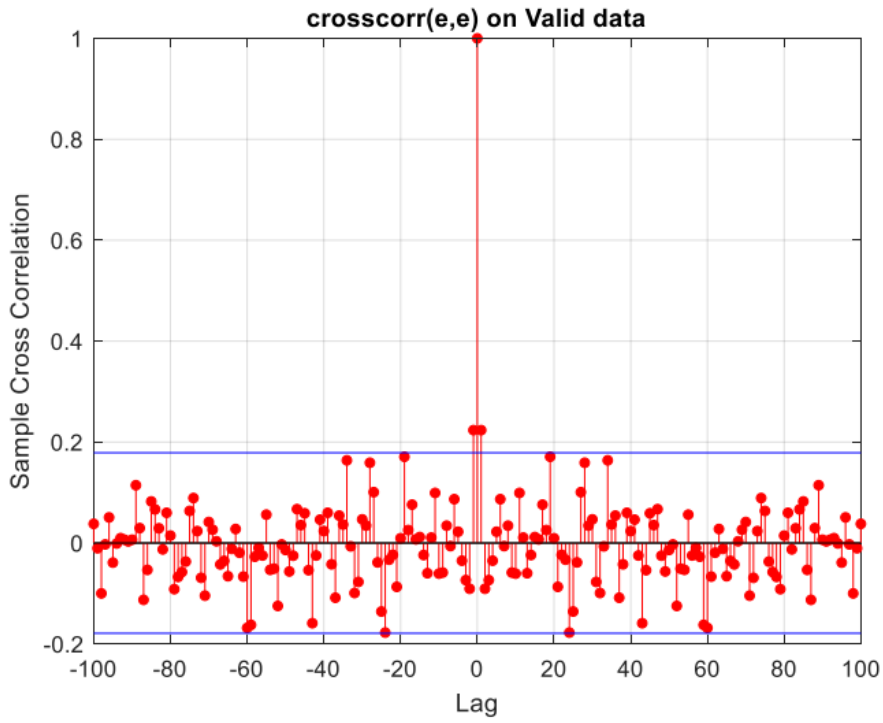


*Figure 144 Error correlation plot-**high noise validation data***

If we use this setting the fitness percentage is very low. Use second option once again:

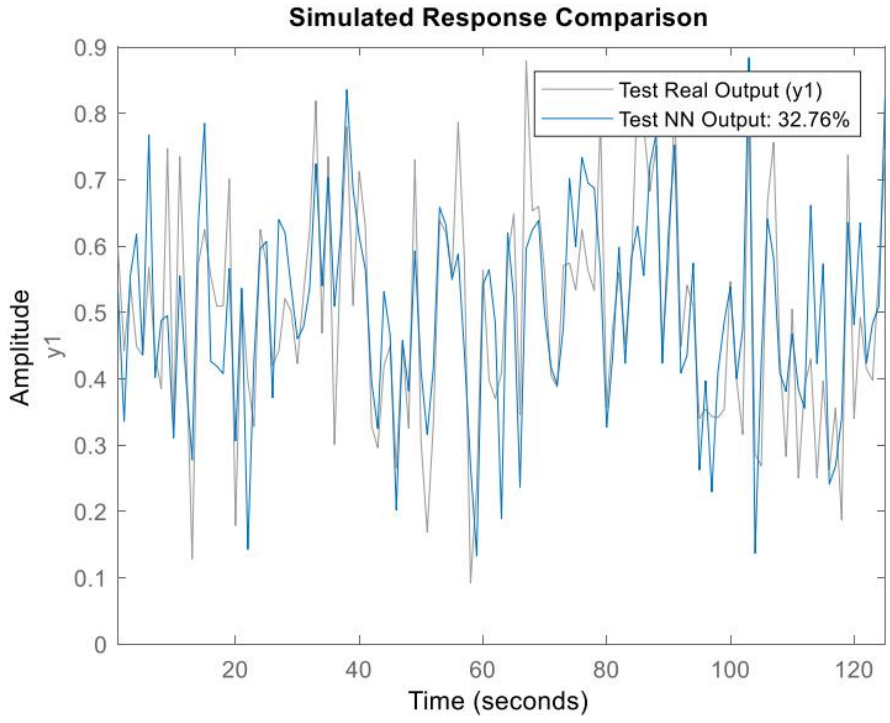


*Figure 145 the actual and estimated system output and fitness percentage-**high noise-test data***

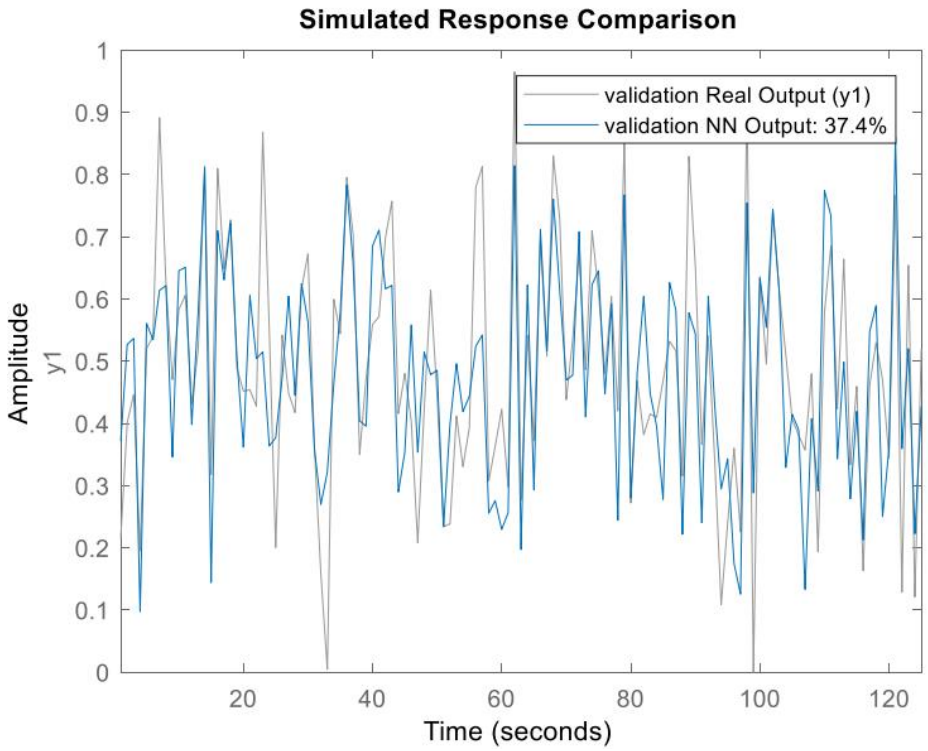


*Figure 146 the actual and estimated system output and fitness percentage-**high noise-validation data***

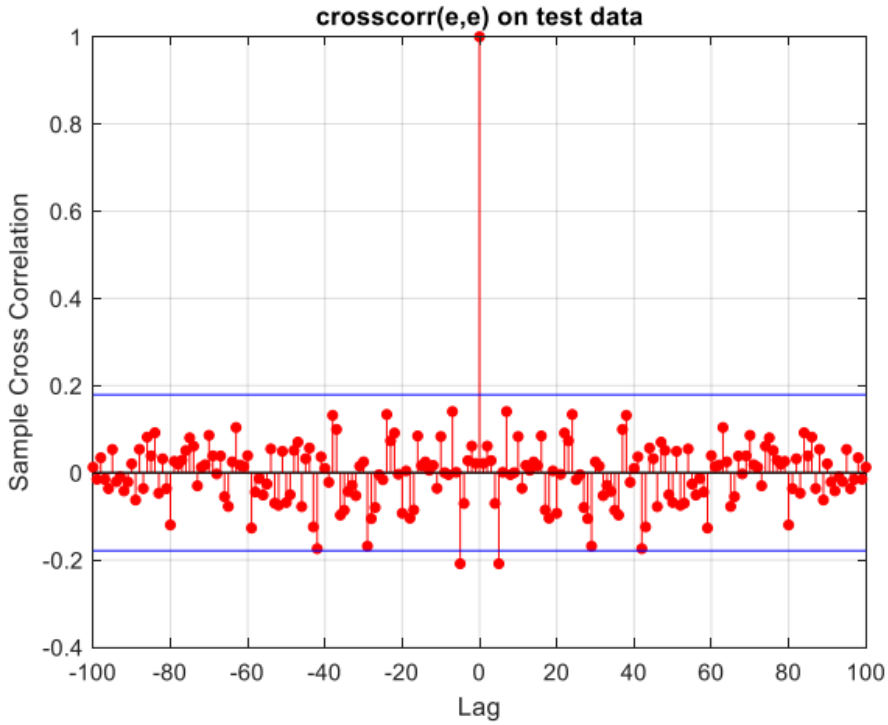


*Figure 147 Error correlation plot-**high noise test data***

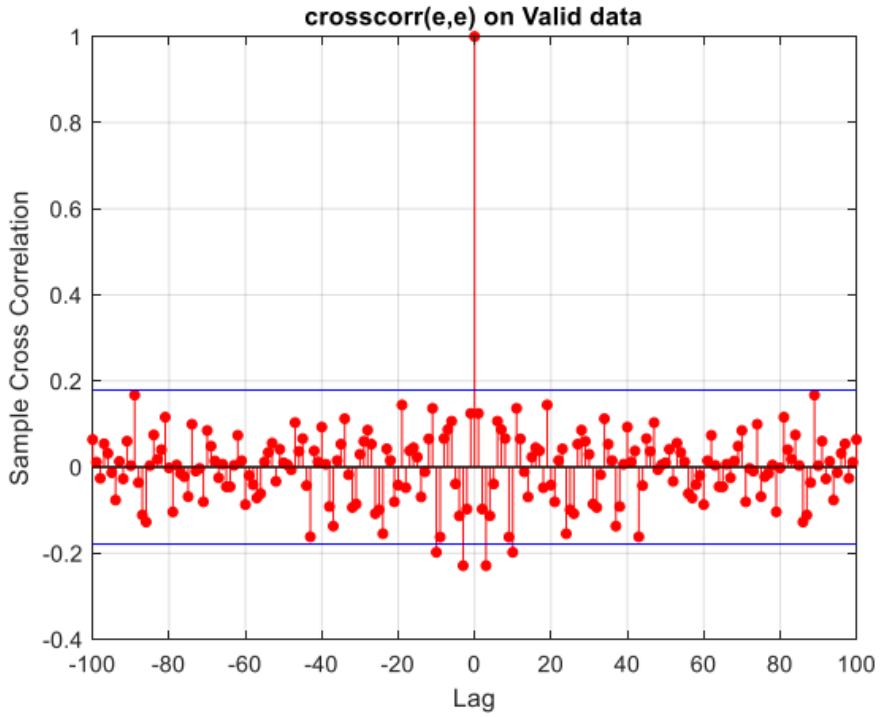


*Figure 148 Error correlation plot-**high noise validation data***

Fitness percentages are much better but still low. And error whitening has happened. It can be deduced that ANFIS is sensitive to noise.

3-6)

ANFIS vs LoLiMoT when it comes to noise

For robustness against noise we use RNN criterion, which is defined as below:

$$Ration\ of\ no\ noise\ to\ with\ noise(RNN) = \frac{Best\ SSE\ Without\ Noise}{Best\ SSE\ with\ Noise}$$

As the value of RNN approaches 1, it means that the network is more resistant to noise, or in other words, the network is more robust against noise. Specifically, when the input has noise, the change in the Sum of Squared Errors (SSE) is less. We calculate the values on the validation data.

ANFIS	Without Noise	Low Noise	Moderate Noise	High Noise
RNN	-	0.5882	0.1101	0.0791

LoLiMoT	Without Noise	Low Noise	Moderate Noise	High Noise
RNN	-	0.0.9695	0.4256	0.1673

In all three cases and for each of the three types of noise, LOLIMOT is more robust against noise compared to ANFIS.

Comparison of All Models

Comparisons were mentioned in each section. Here, it is noted that the gradient descent method did not provide satisfactory accuracy for our model, and we used the Levenberg-Marquardt method for training the network weights. In noise-free data, the two-layer MLP network performed better than the single-layer one, and the single-layer network, like NRBF, had reasonable accuracy. Then, ANFIS and LOLIMOT are in the next ranks. With an increase in noise, it is observed that the single-layer MLP network has slightly better performance than the two-layer one. The NRBF network has provided better fitting percentages compared to RBF. ANFIS showed one of the best performances up to moderate noise but was sensitive to strong noise, resulting in lower fitting percentages. Overall, on the entire validation set, ANFIS has outperformed LOLIMOT, which is comparable to the MLP fitting percentages. However, these results are not definitive, and they are based on the parameters extracted and the model obtained for this particular dataset. In general, all networks had similar performances.

Additionally, it can be noted that training with the Levenberg–Marquardt method requires fewer neurons. The two-layer network also requires more epochs compared to the single-layer (in the gradient descent method), and overall, with the Levenberg–Marquardt method, fewer neurons and epochs lead to better results. RBF networks required more epochs for training, and the NRBF network required fewer epochs compared to RBF. For suitable modeling with ANFIS, a large number of epochs were used for both low-noise and noiseless data. Moreover, LLM neurons were more than the other networks in the Levenberg–Marquardt method.

Question 2

In this question, we want to identify the Wind Turbine block available in Simulink MATLAB. To identify it, we need to first familiarize ourselves a bit more with the system. You can refer to the MATLAB Help and examine the inputs, outputs, and governing equations. Firstly, this model is designed for the steady-state of a wind turbine, and secondly, the formula for the mechanical power output is as follows:

$$P_m = \frac{c_p(\lambda, \beta)\rho A}{2} v_{wind}^3$$

The normalized formula:

$$P_{m_{pu}} = k_p c_{p_{pu}} v_{wind_{pu}}^3$$

As observed, the mechanical power output is a function of the performance coefficient C_p, wind speed, and power coefficient K_p. C_p is also a function of the generator speed and pitch angle, as shown in the figure below:

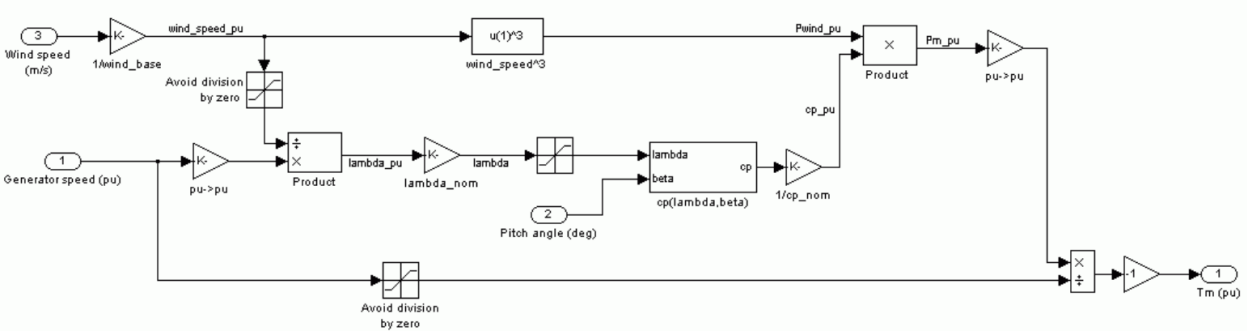


Figure 149 wind turbine block in matlab

An important and notable point regarding the system output is that it has been normalized for the nominal condition (PU), meaning the output is divided by the nominal value. Therefore, the output in the nominal condition is approximately 1 PU. Hence, if appropriate inputs are provided (inputs within acceptable power system limits), the output should be around 1 PU. Now, let's proceed to adjust the inputs.

1. The Pitch angle for the turbine is vertically oriented and zero degrees in the optimal condition. For better modeling, if necessary, different operating points should be considered. For simplicity, we model the system with a constant pitch angle, and we try to increase the range of variations to improve the model. Since the pitch angle should be positive, we choose a constant value greater than zero to ensure that the input pitch angle, accompanied by the Chirp signal, does not become negative.
2. The wind speed is a crucial parameter that is itself a function of environmental conditions, such as height, etc.

In the figure below we have:

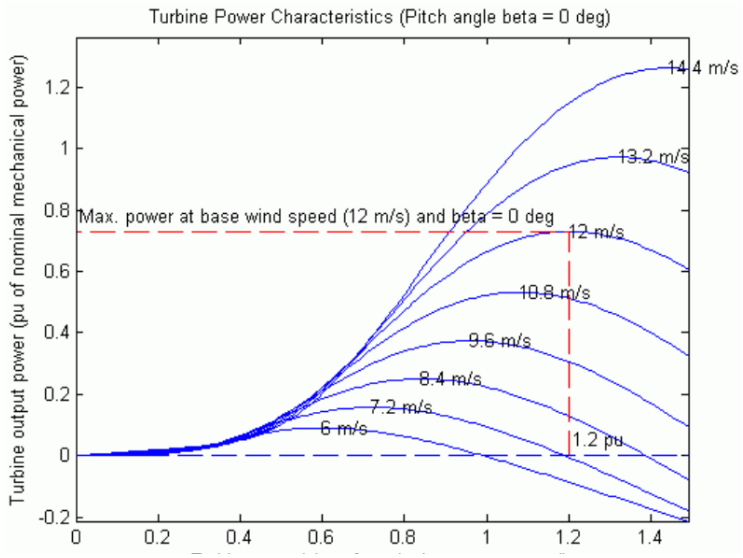


Figure 150 Turbine speed (pu of nominal generator speed)

Thank you for providing more information. It seems you've explained the inputs and outputs of the wind turbine system, including the mechanical power output and the factors influencing it.

3. If you have any specific questions or if there's anything particular you'd like assistance with regarding the simulation or identification of the wind turbine system in Simulink MATLAB, please let me know, and I'll do my best to help.

Another point to consider is that the inputs should be rich. Therefore, we sum the inputs within the nominal range with a Chirp signal with a suitable amplitude and frequency to make the inputs diverse. We try to consider the Chirp signal's amplitude and frequency range as much as possible so that the identified model can follow the model for a larger range of inputs. We also set the sampling frequency to 0.1.

Continuing with the above explanations, we apply the indicated input signals to the system and observe the results.

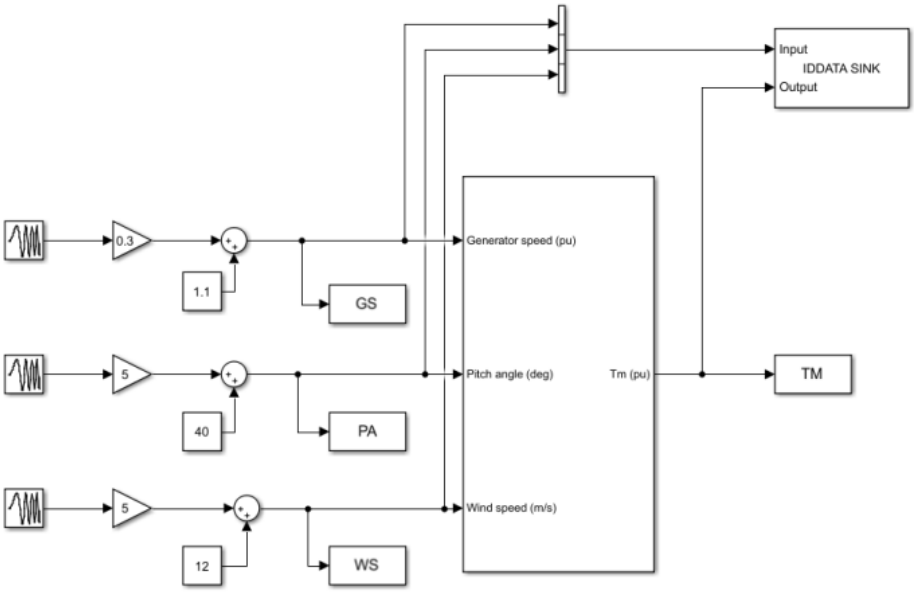


Figure 151 Inputs of the turbine

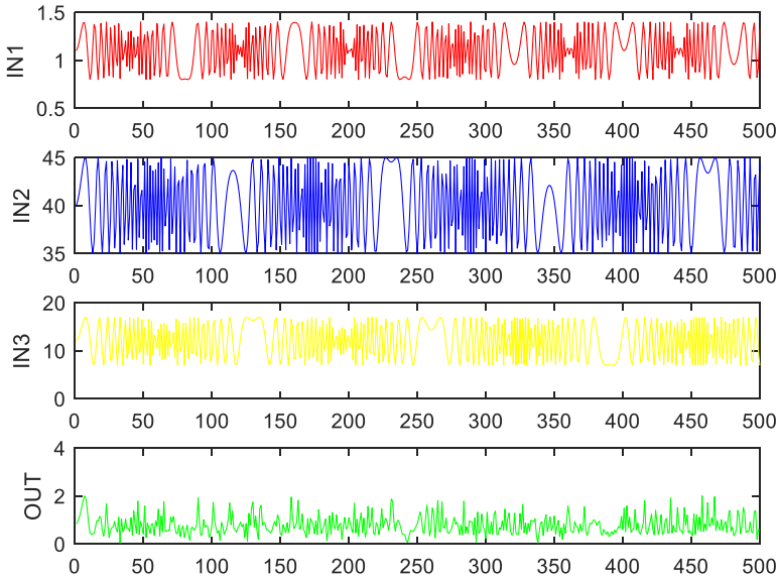


Figure 152 Input and output of the turbine

At first, we plot the correlation between inputs and outputs, and it is observed that we only have a peak at zero.

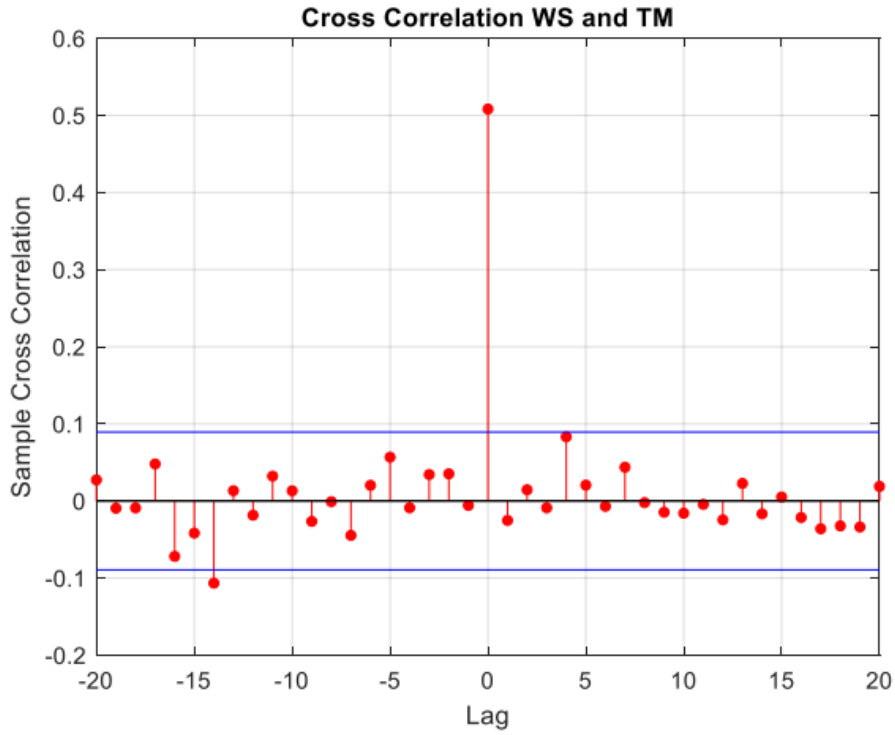


Figure 153 Cross-correlation of first input and the output

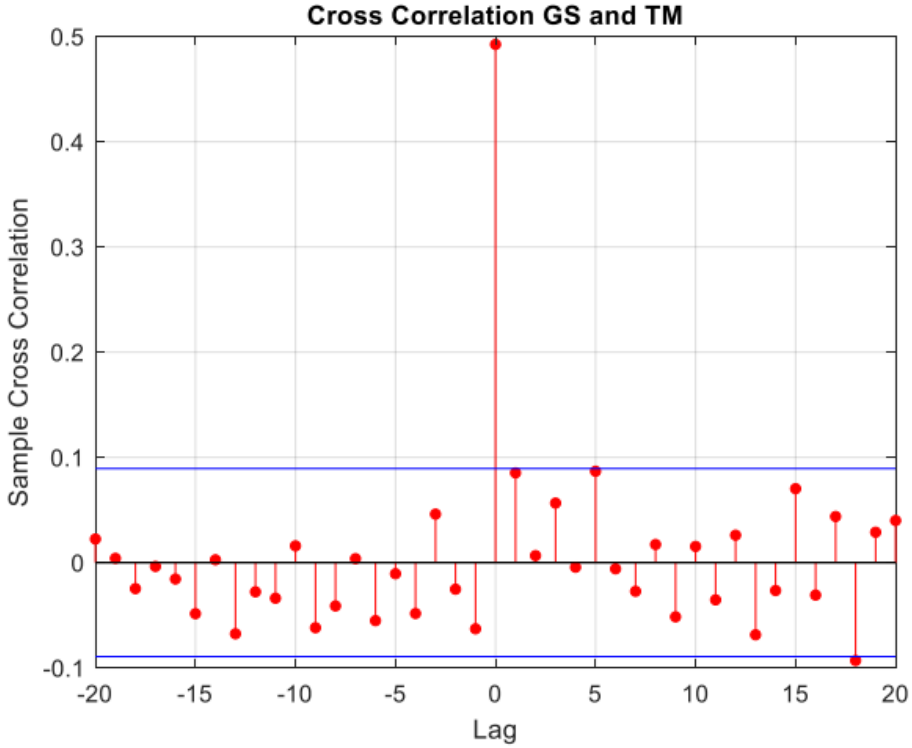


Figure 154 Cross-correlation of second input and the output

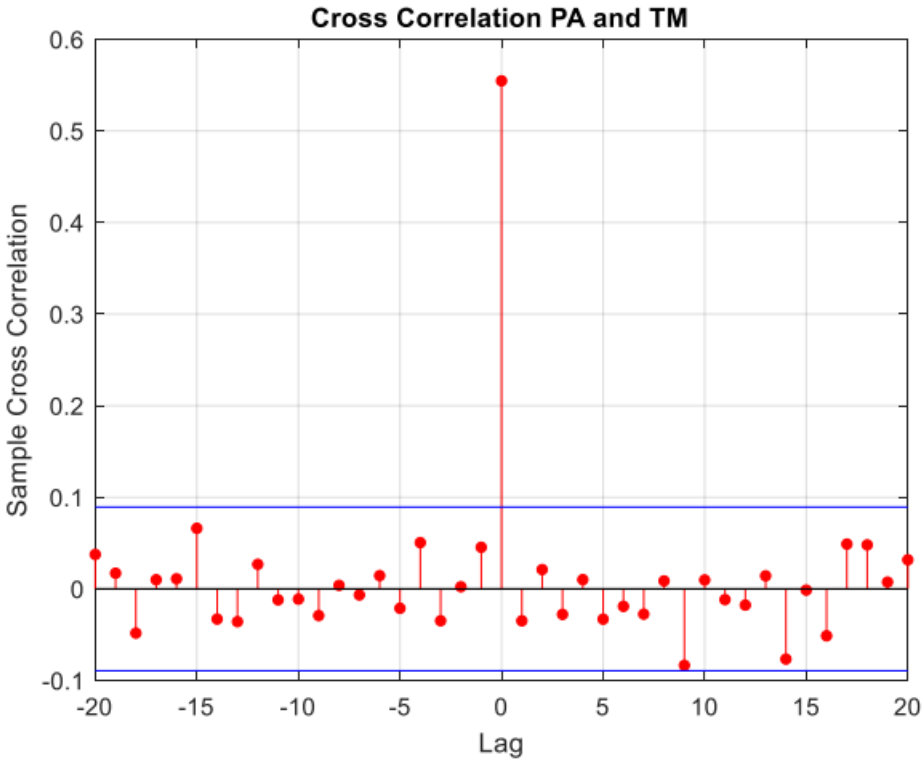


Figure 155 Cross-correlation of third input and the output

As can be seen, the system does not have a delay. According to the provided formula at the beginning, the identification should be nonlinear. Also, considering the correlation plots, it is observed that nonlinear static identifiers can be used. We use a neural network with one hidden layer and a structure similar to the previous question.

```
net = newff(total_P,total_T,num_neuron,{'tansig','purelin'},'traingd','learngd','mse');
```

We take the data partitioning as follows

```
net.divideParam.trainRatio = 0.4636;
net.divideParam.valRatio = 0.2682;
net.divideParam.testRatio = 0.2682;
```

For varying the number of neurons from 1 to 30, we calculate the MSE values.

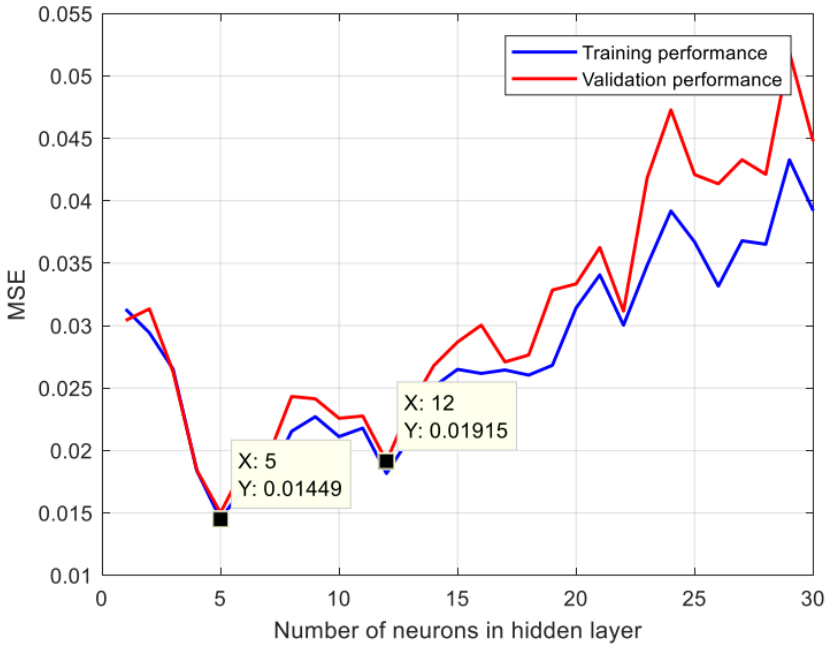


Figure 156 Error-neuron plot

Therefore, with 5 neurons, we have an optimal structure for the middle layer. Beyond 12 neurons, the over-parameterized phenomenon occurs.

Now, with these 5 neurons, we investigate the optimal epoch from 1 to 400.

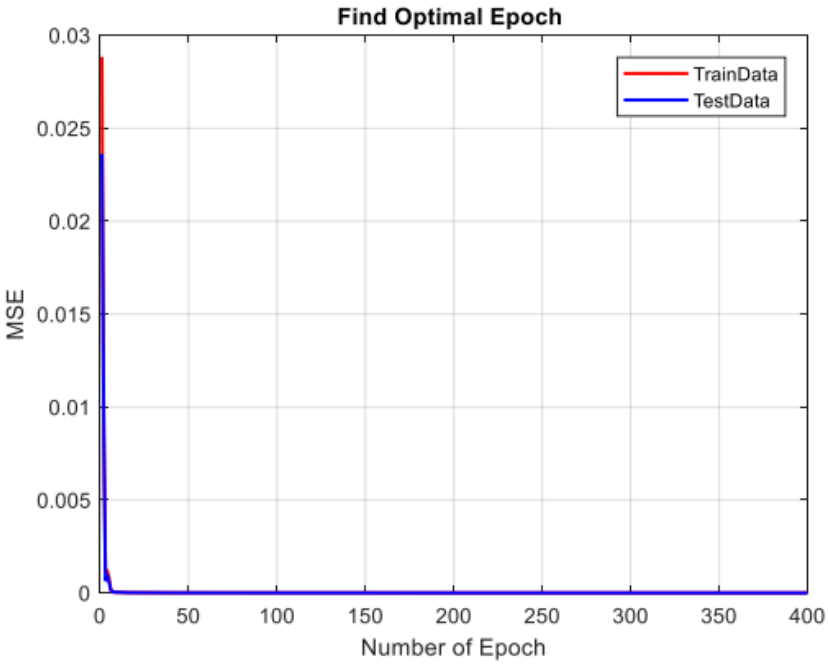


Figure 157 Error-epoch plot

If we zoom in:

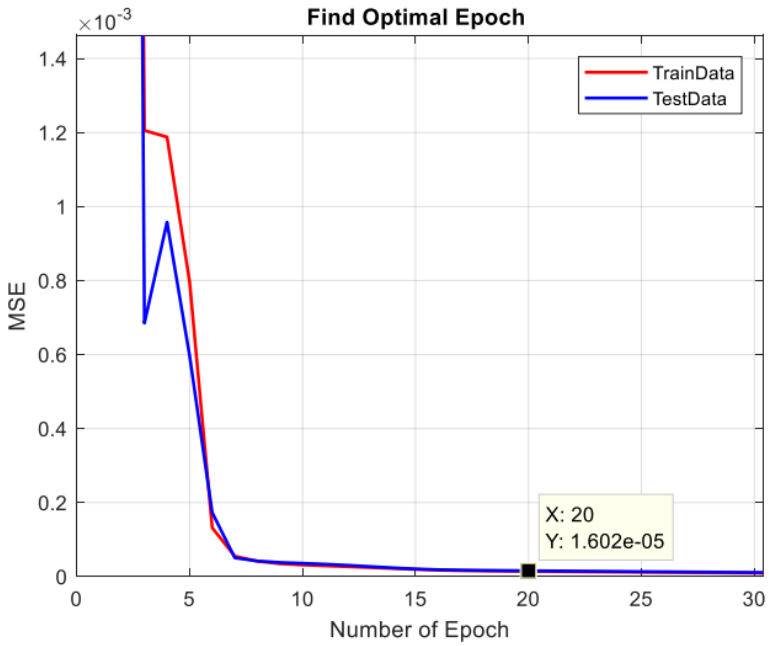


Figure 158 Error-epoch plot

Approximately 20 epochs seem sufficient and optimal. The result of selecting the network with this structure, 20 epochs, and 5 neurons in the hidden layer, on the training, testing, and validation data is as follows.

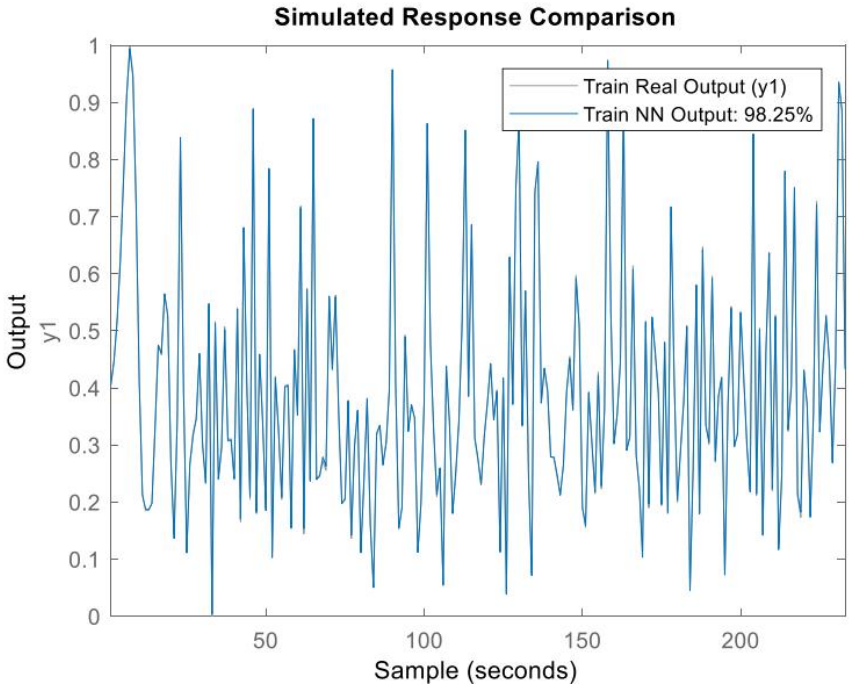


Figure 159 The actual and estimated output

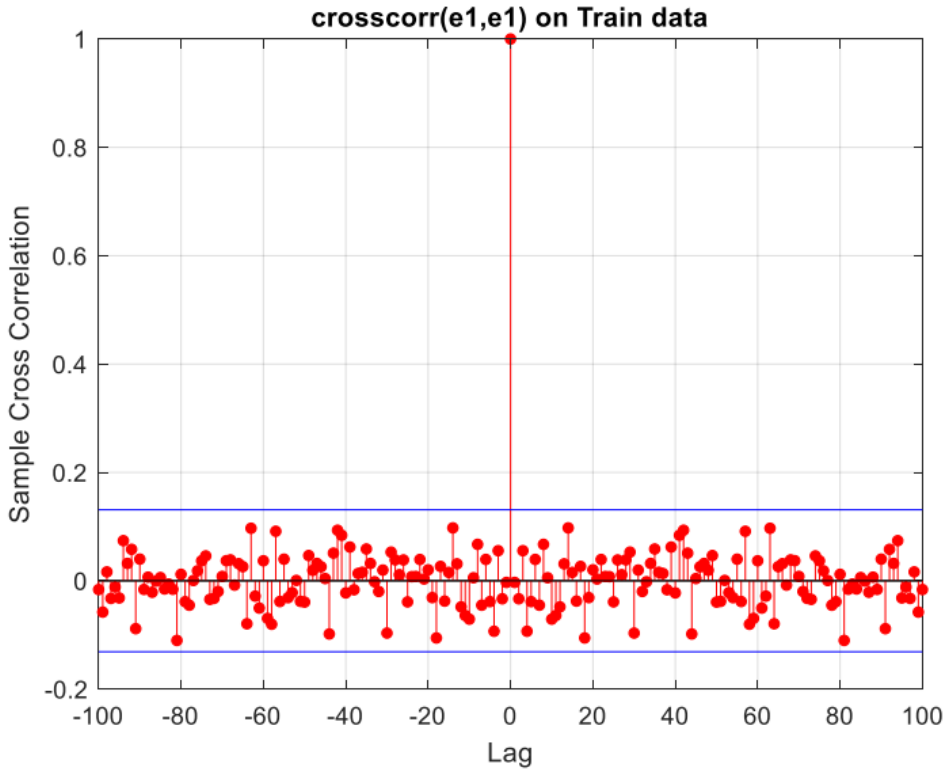


Figure 160 The error cross-correlation

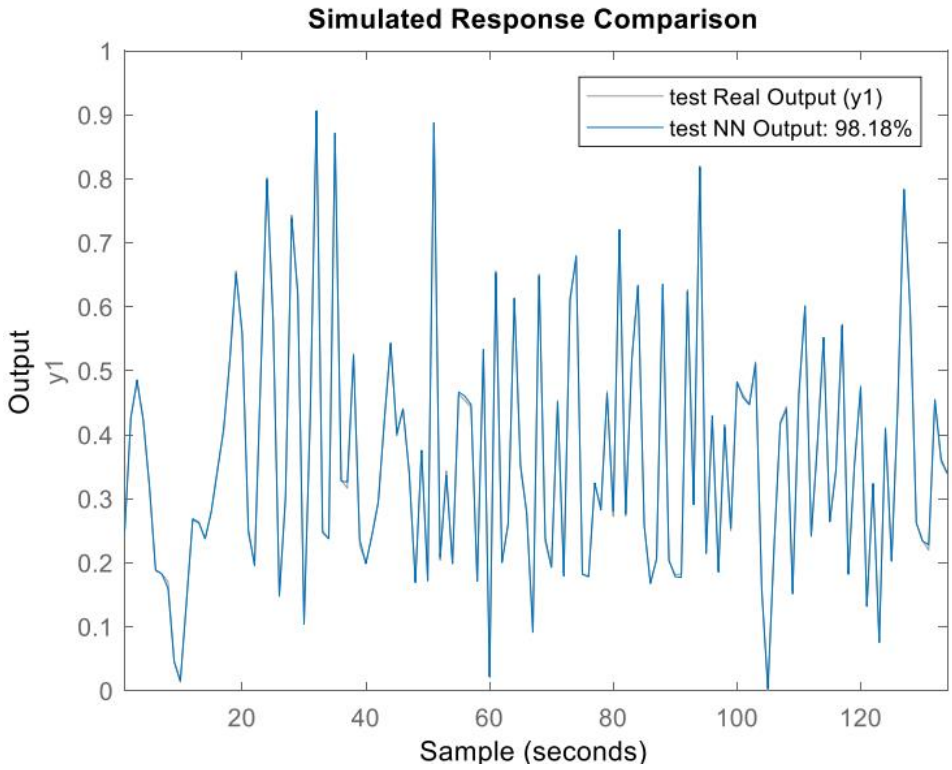


*Figure 161 The actual and estimated output-**test data***

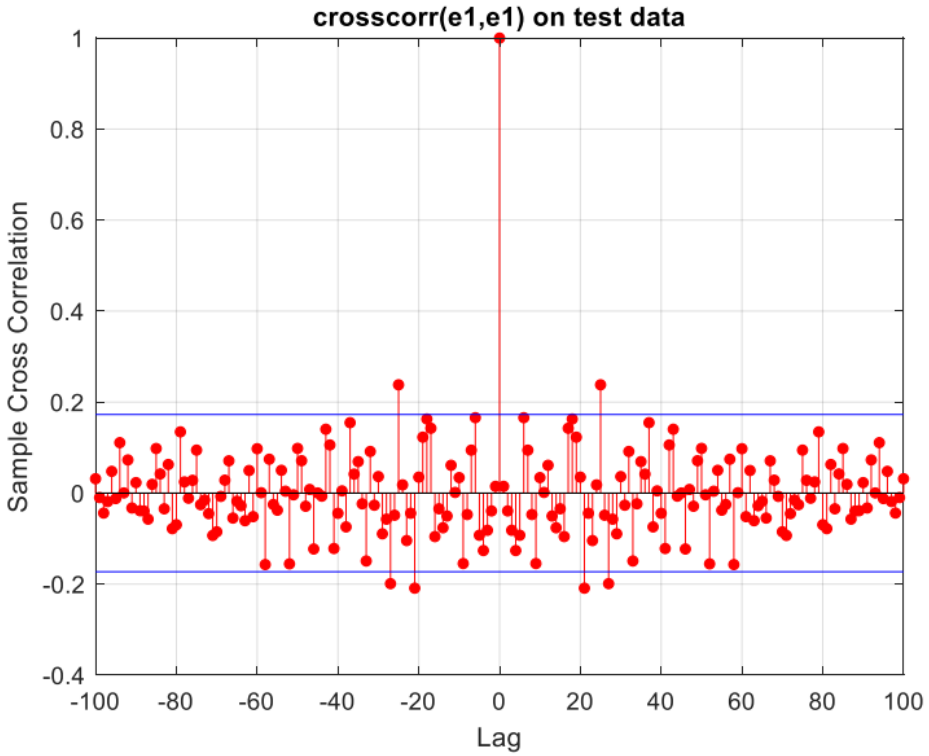


*Figure 162 error cross-correlation-**test data***

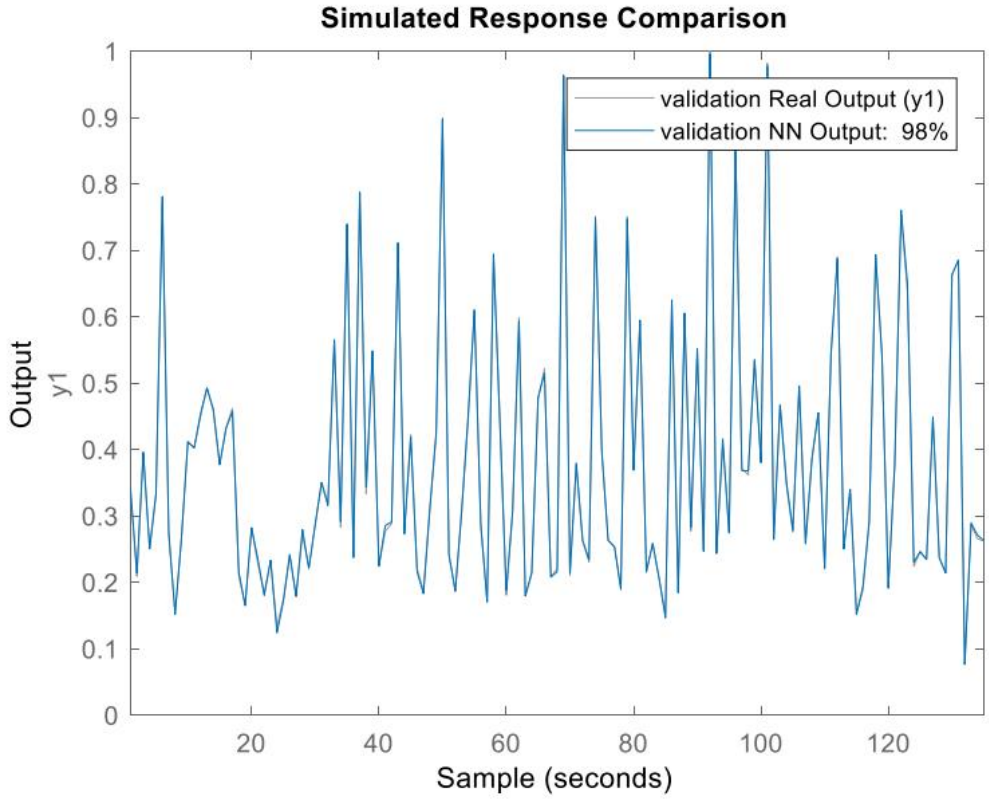


*Figure 163 The actual and estimated output-**validation data***

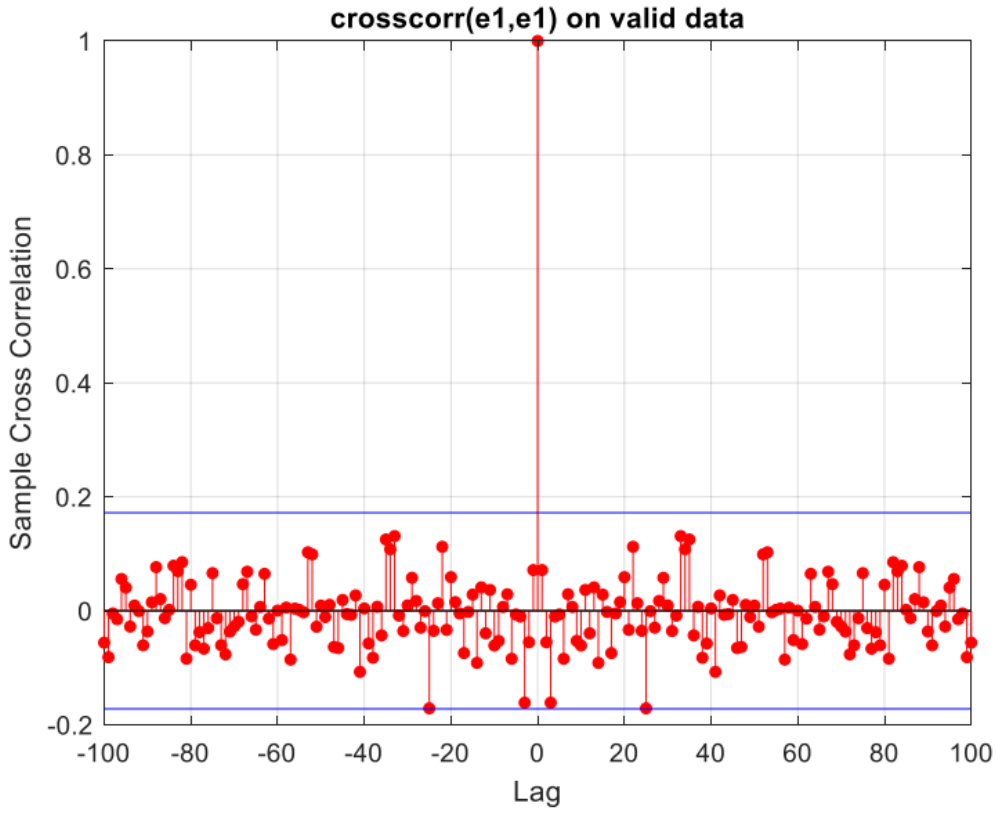


*Figure 164 error cross-correlation-**validation data***

It can be observed that on all three sets of data, the fitting percentage is above 98%, and error whitening is observed. Therefore, the selected network is suitable and we have chosen an appropriate optimal structure.