ASSIGNMENT REPORT

ON

‘RESEARCH ON RBF NEURAL NETWORK MODEL REFERENCE ADAPTIVE CONTROL SYSTEM BASED ON NON-LINEAR U-MODEL’

(GROUP NO. 16, PROBLEM NO. 44)

BY

Name of the Student	ID Number
Neelabh Sinha	***2016B5A80600P***
Parth Goyal	***2016B3A80483P***
Utkarsh Kedia	***2016B5A80713P***

Prepared for Partial Fulfillment of the course

INSTR F343 - INDUSTRIAL INSTRUMENTATION AND CONTROL

AT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

July, 2020

ACKNOWLEDGEMENT

Our team wishes to express the deepest gratitude to our instructor-in-charge, Prof. Puneet Mishra for his consistent motivation and guidance throughout the duration of our Project which helped us to achieve the set target and complete the project within the stipulated time.

This would not have been possible without his guidance and support.

TABLE OF CONTENTS

1. INTRODUCTION

Although a lot of the study revolves around the linear time-invariant systems (also called as LTI systems), the actual control systems are inherently time variants and also incorporate other deviations from ideal behaviour like non-linearity, hysteresis, etc. Therefore, it is important to understand and formulate a theoretical base to understand the non-linear systems and study their behaviour, along with the development of relevant mathematics to model them.

There have been several non-linear models developed in the past and are available in the literature. Some of them include bilinear system model, NARMAX model, Hammerstein model, non-linear time series model, etc. One of these, NARMAX model is considered as versatile and has the ability to effectively represent a wide range of nonlinear systems. The U-model is one such model derived from the NARMAX model, and shall be used in this project to represent the system.

After the representation of the model comes its control, and out of several control techniques used by researchers in the past, neural networks have evolved as one of the prime techniques to design controllers lately. Since neural networks can incorporate a wide generalization with relatively simple structure, and has a fair bit of nonlinearity associated with it, it can be widely used with non-linear systems, and recently, neural networks have been developed to design control theory, and also act as mathematical models of various systems due to its general nature to accurately learn from provided data.

In this project, the aim is to design a non-linear system using U-model, and further use a Radial Basis Function Neural Network as controller to achieve optimum control performances. Section 2 consists of proper problem formulation, and Section 3 consists of theoretical formulation of the solution. Section 4 mentions the methodology followed, followed by Section 5 with results.

2. THEORETICAL BACKGROUND

Before formulation of the problem and dealing with the solution, it is important to understand the theoretical concepts involved to design the scheme and understand the approach.

The plant and reference model are designed using the U-model. So, first, it is important to understand the U-model.

2.1. U-MODEL

U-model can be represented as a polynomial function with the following equation -

$$y(t) = \sum_{j=0}^{M} \left[\prod_{l=1}^{n} \left(a_l + \sum_{i=1}^{m} y^l(t-i) \right) \right] \cdot \left[\prod_{l=1}^{n} \left(b_l + \sum_{i=2}^{m} u^l(t-i) \right) \right] u^j(t-1)$$

where u(t) is the input at the previous time step 't', y(t) is output at time step 't'. M is the degree of the model and l, j are integers.

By simplifying the above equations, we can represent the U-model in compact form as -

$$y(t) = \sum_{j=0}^{M} \alpha_j(t) u^j(t-1)$$

Where α_j

$$= \left[\prod_{l=1}^{n} \left(a_l + \sum_{i=1}^{m} y^l(t-i) \right) \right] \cdot \left[\prod_{l=1}^{n} \left(b_l + \sum_{i=2}^{m} u^l(t-i) \right) \right]$$

To establish the generality of the model, let us take an arbitrary model equation as

$y(t) = 0.3y(t-1)y(t-2) + 0.2u(t-2)u(t-1) - 0.5y(t-3)u^2(t-1)$.

This can be written from the notation above as
$y(t) = \alpha_0(t) + \alpha_1(t)u(t-1) + \alpha_2(t)u^2(t-1)$, where

$\alpha_0(t) = 0.3y(t-1)y(t-2)$
$\alpha_1(t) = 0.2u(t-2)$
$\alpha_2(t) = - 0.5y(t-3)$

In a similar fashion, all models can be converted to corresponding U-Model notation.

2.2. ADAPTIVE CONTROL

As discussed in the lectures as well, adaptive control scheme is a technique that can adjust the process parameters dynamically to compensate for errors and process characteristics. Adaptation aims to find out the best adjustment of controller parameters. It is basically a dynamic control system with the facility of online error learning. So, basically, an adaptive system adapts to the change and modifies control law dynamically.

There are two types of adaptive control systems -

- Direct Adaptive Control - In this, the controller parameters are updated directly on the basis of error between output system and the reference signal
- Indirect adaptive control - In this, an estimator is used to find the updates of the model parameters based on the input and output.

2.2.1. MODEL REFERENCE ADAPTIVE CONTROL

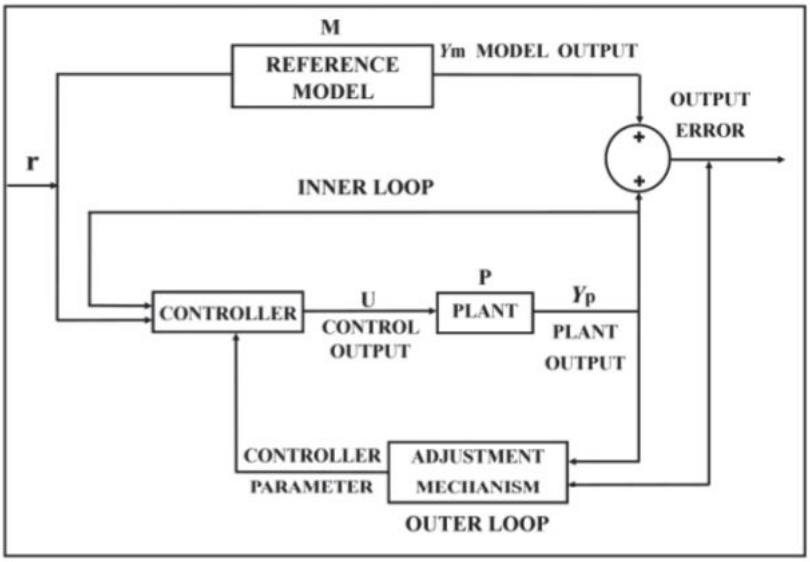


Fig 1. Model Reference Adaptive Control

Given above is the diagram of Model Reference adaptive control system. It is one of the direct control schemes. In this, there are two models, one is the plant model, and the other is a reference model assigned by the user. The aim of this scheme is that the plant parameters should be adjusted such that error between the plant model and reference model is minimum, i.e., the plant output follows the reference output.

Model reference adaptive control is used at various places, including aircraft mechanism to biomedical equipment. Typically, adjustment mechanisms are developed using Neural Networks or Fuzzy inference systems.

2.3. RADIAL BASIS FUNCTION NEURAL NETWORK

The network has 3 layers-input, hidden and output layers. Each neuron in the input layer represents a predictor variable. Every neuron in the hidden layer consists of a radial basis function such as a Gaussian function centered around a point with the same dimensions as that of the predictor variables.

Space has as many dimensions as there are predictor variables. The Euclidean distance is computed from the point being evaluated to the center of each neuron and a radial basis function (kernel function) is applied to the distance to compute the weight for each neuron. The RBF is so named because the radial distance is the argument of the function. Therefore, the closer a neuron is from being evaluated, the more the influence it has. There are 3 types of parameters to be taken care of in an RBF network:

1. The weight(w) of the connection between hidden nodes and output nodes
2. The center(c) of each node in the hidden layer
3. The unit width(r)

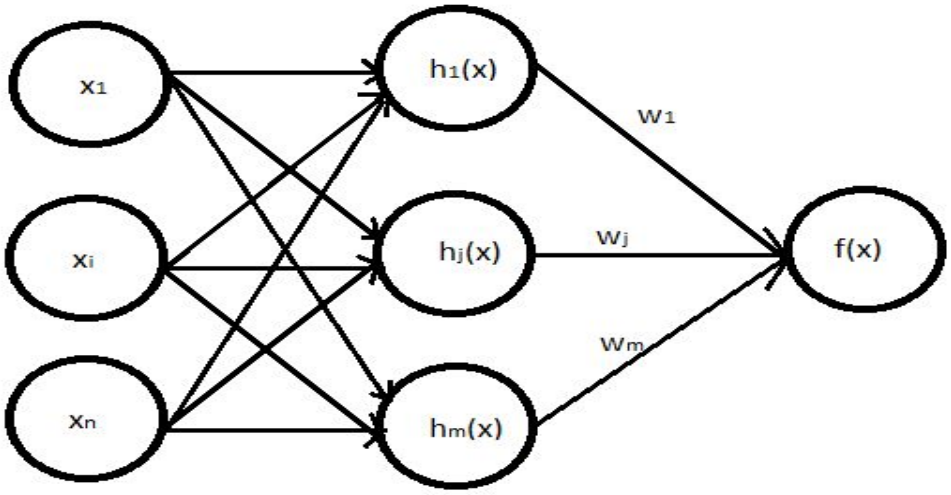


Fig 2. Radial Basis Function Network

$$h(x)=exp\{\frac{-(x-c)^2}{r^2}\}$$

$$f(x)=\sum_{j=1}^{m} w_j\ h_j(x)$$

Any clustering algorithm (such as: K-means clustering) can be used to determine the unit

centers. A set of clusters each with n-dimensional centers is determined by the number of input variables or nodes of the input layer. Cluster centers become the centers of the RBF units. Number of clusters is a design parameter and determines the number of nodes in the hidden layer

After the RBF centers have been calculated, K-nearest neighbours algorithm is used to find the width of each RBF unit. It is done by choosing a number k and then locating k nearest neighbours for each center. The final value of unit width(r) is chosen as the RMS distance between the current cluster center and its k nearest neighbours.

3. PROBLEM FORMULATION

To formulate the design problem formally, the objective is to design a model reference adaptive controller using Radial Basis Function network for a non-linear plant and derive the control law. The non-linear plant will be modelled using the U-model scheme. Furthermore, the analysis will be done on the performance of the controller when the degree of plant and reference model vary with respect to each other, and also for various individual degrees of the plant model.

RBFN controller will adjust the parameters of the plant using the compensation parameters, based on the error of output between the two models. The block diagram of the entire system can be represented as given below:

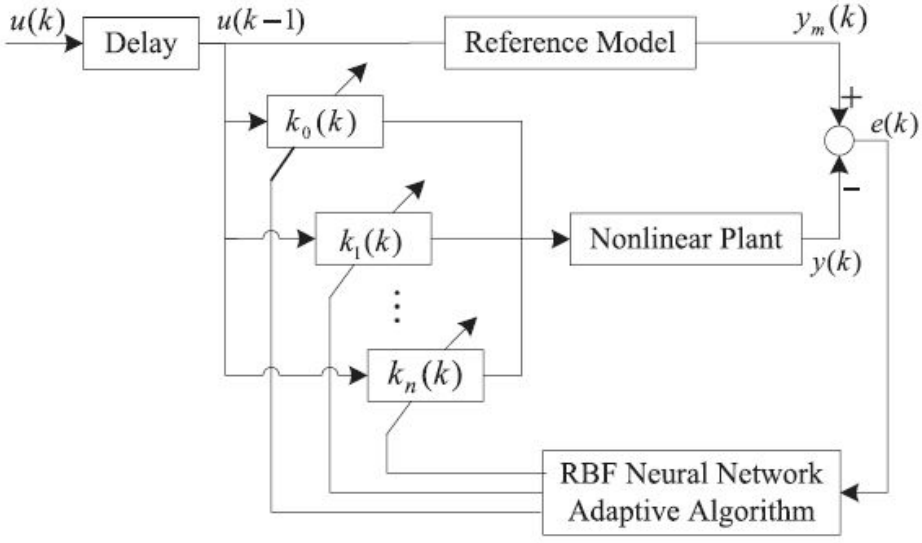


Fig 3. Block diagram of the Control Scheme

To start the mathematical formulation, let us assume that the reference model is denoted by $y_m(k)$ and the actual plant model is denoted by y(k). So,

$$y_m(k) = \sum_{j=0}^{n} \alpha_j(k) u^j(k-1)$$

, and

$$y(k) = \sum_{j=0}^{m} \bar{\alpha}_j(k) u^j(k-1)$$

Now, from this, the degree of reference model in 'n' and that of the plant model is 'm'. So, there are three possible cases that should be dealt individually -

- n > m
- n = m
- n < m

3.1. CASE I: n > m

When the degree of the reference model is greater than that of the plant model, the parameters (compensation parameters) of the plant higher than that of the reference model will be 0. This will result in an ineffective adjustment of the compensation parameter and the system will have redundancy. Thus, the performance will not be optimal.

Mathematically,

$$\begin{cases} \tilde{\alpha}_j(k) = \bar{\alpha}_j(k) & 0 \le j \le m \\ \tilde{\alpha}_j(k) = 0 & m < j \le n \end{cases}.$$

3.2. CASE II: n = m

In this case, the degree of plant and reference model are equal, which results in one-to-one correspondence of the parameters. This results in the proper training of the neural network and effective adjustment of the compensation parameters. The system would exhibit fast response and good accuracy.

3.3. CASE III: n < m

In this case, the reference model is of a lower degree than the plant model. So, the polynomial part of the plant model above the highest degree of the reference model will be incorporated in the zero-order term. Although there is no redundancy, the $k_0(k)$ term of the compensation parameter will be very complicated. Here, as we shall see, the precision will be affected.
Mathematically,

$$\begin{cases} \tilde{\alpha}_0(k) = \bar{\alpha}_0(k) + \bar{\alpha}_{n+1}(k) u^{n+1}(k-1) + \cdots \\ +\bar{\alpha}_m(k) u^m(k-1) \\ \tilde{\alpha}_j(k) = \bar{\alpha}_j(k), 1 \le j \le n \end{cases}$$

Now, to design the control system, we shall take the error between the two parameters, which comes

out to be

$$
\begin{aligned}
e(k) &= \sum_{j=0}^{n} (\alpha_j(k) - k_j(k)\tilde{\alpha}_j(k))u^j(k-1) \\
&= \alpha_0(k) - k_0(k)\tilde{\alpha}_0(k) \\
&\quad + (\alpha_1(k) - k_1(k)\tilde{\alpha}_1(k))u(k-1) \\
&\quad + \cdots + (\alpha_n(k) - k_n(k)\tilde{\alpha}_n(k))u^n(k-1)
\end{aligned}
$$

Let W be the weight of the hidden to output layers of RBFN, and H be the hidden layer value vectors, then, the compensation parameter vector K(k) will be given by **K(k) = WH** for the outermost node of RBFN.

For the hidden node, as explained earlier,

$$h_j = \exp\left(-\frac{||\boldsymbol{X} - \boldsymbol{C}_j||^2}{2b_j^2}\right)$$

Now, the training of the of the RBFN will take place as per the gradient descent algorithm, with momentum.

$$\Delta w_{lj}(k) = -\eta\frac{\partial E(k)}{\partial w_{lj}} = -\eta\frac{\partial E(k)}{\partial y(k)}\frac{\partial y(k)}{\partial K(k)}\frac{\partial K(k)}{\partial w_{lj}}$$

Where,

$$
\begin{cases}
\Delta w_{1j}(k) = -\eta\frac{\partial E(k)}{\partial w_{1j}} = \eta e(k)\tilde{\alpha}_0(k)h_j \\
\Delta w_{2j}(k) = -\eta\frac{\partial E(k)}{\partial w_{lj}} = \eta e(k)\tilde{\alpha}_1(k)u(k-1)h_j \\
\cdots \\
\Delta w_{(n+1)j}(k) = -\eta\frac{\partial E(k)}{\partial w_{(n+1)j}} \\
= \eta e(k)\tilde{\alpha}_n(k)u^n(k-1)h_j
\end{cases}
$$

$$
\begin{aligned}
w_{lj}(k) &= w_{lj}(k-1) + \Delta w_{lj}(k) \\
&\quad + \beta[w_{lj}(k-1) - w_{lj}(k-2)]
\end{aligned}
$$

To understand the stability of the model, we will analyse the error at the time infinity. Since at t=infinity, e(k) will tend to 0, it would be safe to infer that the model will be converging.

4. UNDERSTANDING THE PROPOSED SOLUTION

In order to get better compensation parameters, neural network adaptive technique is used. Precisely the controller contains a radial basis neural network to calculate the error between the ideal reference model and the controlled plant. 3 different simulations are performed to showcase the effect of relationship between reference model and plant model. We will be analyzing only one.

Reference model is chosen as a 4th degree U model, given by the equation-.

$$y_m(k) = 0.529412u(k) - 0.1u(k-3)u(k-2)u(k) + 0.529412u(k-2) - 0.823529\,y_m(k-1) - 0.294118\,y_m(k-2) + 1.058824u(k-1) + 0.12u(k-3)u(k-2)\,u^2(k-1) + 0.1\,u^3(k-1) - 0.12\,u^4(k-1)$$

For the given reference model 3 different controlled nonlinear plant models are chosen

1st degree controlled plant, given by the equation-

$$y(k) = 0.5y(k-1) + 0.2y^2(k-1) + 0.2u(k-1)$$

4th degree controlled plant, given by the equation-

$$y(k) = 0.5y(k-1) + 0.2\,y^2(k-1) + 0.2u(k-1) - 0.7y(k-1)\,u^2(k-1) + 0.5y(k-1)\,u^3(k-1) - 0.3y(k-1)\,u^4(k-1)$$

7th degree controlled plant, given by the equation-

$$y(k) = 0.5y(k-1) + 0.2\,y^2(k-1) + 0.2u(k-1) - 0.7y(k-1)\,u^2(k-1) + 0.5y(k-1)\,u^3(k-1) - 0.3y(k-1)\,u^4(k-1) + 0.1y(k-1)\,u^5(k-1) + 0.4y(k-1)\,u^6(k-1) + 0.8y(k-1)\,u^7(k-1)$$

Comparison formula for n=4 is applied for all the 3 cases considering:

$$y(k) = \tilde{\alpha}_0(k) + \tilde{\alpha}_1(k)u(k-1) + \tilde{\alpha}_2(k)u^2(k-1) + \tilde{\alpha}_3(k)u^3(k-1) + \tilde{\alpha}_4(k)u^4(k-1)$$

for 1st degree controlled plant

$$\tilde{\alpha}_0(k) = 0.5y(k-1) + 0.2y(k-1)y(k-1),$$
$$\tilde{\alpha}_1(k) = 0.2, \tilde{\alpha}_2(k) = 0, \tilde{\alpha}_3(k) = 0, \tilde{\alpha}_4(k) = 0.$$

for 4th degree controlled plant:

$$\tilde{\alpha}_0(k) = 0.5y(k-1) + 0.2y(k-1)y(k-1),$$
$$\tilde{\alpha}_1(k) = 0.2, \tilde{\alpha}_2(k) = -0.7y(k-1), \tilde{\alpha}_3(k) = 0.5y(k-1), \tilde{\alpha}_4(k) = -0.3y(k-1).$$

for 7th degree controlled plant:

$$\begin{aligned}\tilde{\alpha}_0(k) = {} & 0.5y(k-1) + 0.2y^2(k-1) \\ & + 0.1y(k-1)u^5(k-1) \\ & + 0.4y(k-1)u^6(k-1) \\ & + 0.8y(k-1)u^7(k-1), \tilde{\alpha}_1(k) = 0.2,\end{aligned}$$

$$\tilde{\alpha}_2(k) = -0.7y(k-1), \tilde{\alpha}_3(k) = 0.5y(k-1),$$
$$\tilde{\alpha}_4(k) = -0.3y(k-1).$$

The inputs of the RBF neural network are e(k), y(k),ym(k), e(k – 1) and e(k – 2).
The outputs are k0(k), k1(k), k2(k), k3(k), k4(k) which are the controller parameters
The initial values of the Gaussian parameters and the weights(w) need to be assumed. They are assumed as:

$$b = [10, 10, 10, 10, 10, 10, 10]$$

$$c = \begin{bmatrix} -10 & -5 & -2 & 0 & 2 & 5 & 10 \\ -10 & -5 & -2 & 0 & 2 & 5 & 10 \\ -10 & -5 & -2 & 0 & 2 & 5 & 10 \end{bmatrix}^T$$

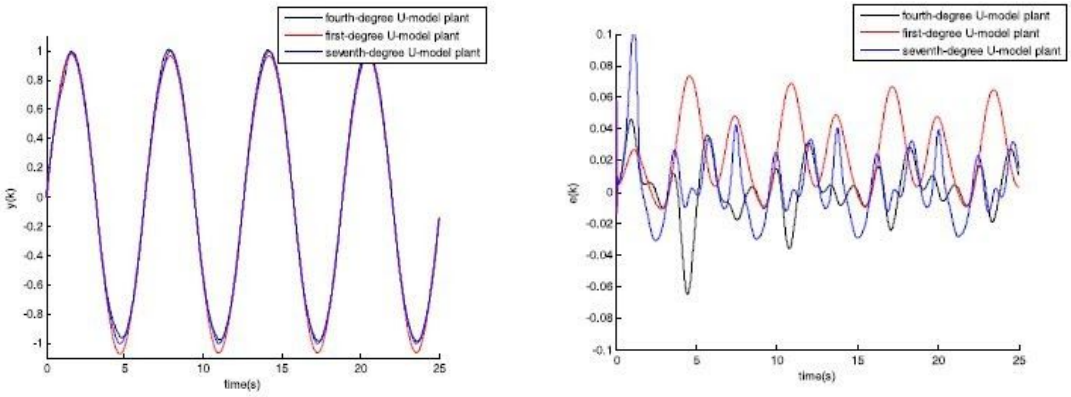
$$W = \begin{bmatrix} -0.5978 & -0.3135 & -0.3998 & 0.5745 & -0.2798 & -0.1967 & 0.2376 \\ -0.0416 & 1.6911 & 2.2545 & 2.2559 & 0.4914 & 0.6640 & -1.0074 \\ 0.2059 & -0.0059 & 0.3132 & 0.2008 & -0.3072 & 0.0390 & -0.2697 \\ 0.3078 & 0.0427 & -0.0490 & 0.1747 & -0.1722 & 0.3197 & 0.0959 \\ 0.2084 & 0.3584 & -0.2680 & 0.3834 & -0.4382 & 0.1227 & -0.1865 \end{bmatrix}$$

The learning rate(η) and momentum factor(β), both are equal to 0.01. For simulation purposes the input signals applied to the system are sine, triangular and square waves.

From the above discussions it is expected that for n=4, the fourth degree controlled plant gives the best control result followed by 7th degree controlled plant. Although the 7th degree controlled plant will be slower in error adjustment speed leading to higher error values.

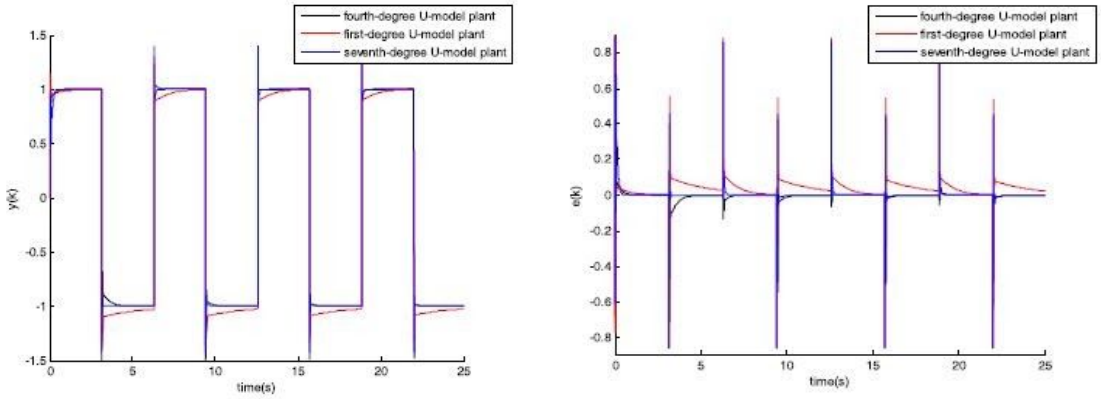
The first degree controlled plant is expected to perform the worst. It is expected to deviate from the expected output due to poor control and error adjustment speed leading to higher values of errors.

The following simulation results clearly support the argument.



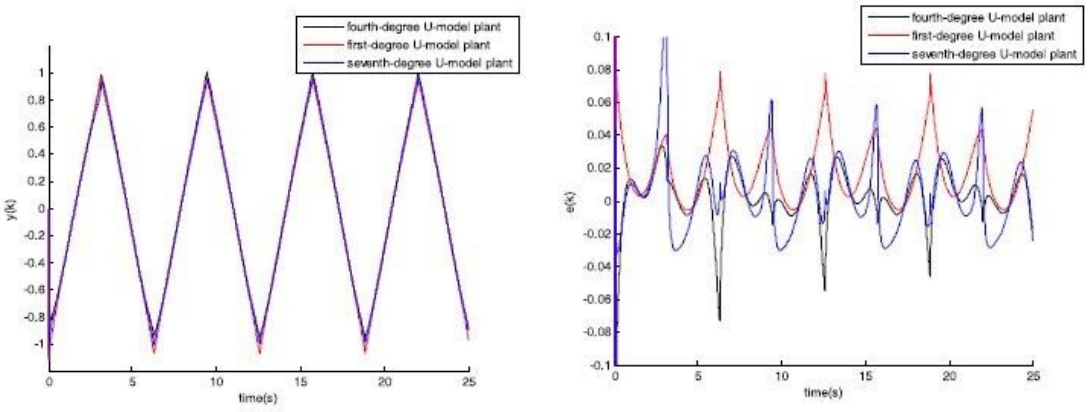
(a)Process Curve (b) Error Curve

Fig 4. Results in case of Sine Wave Input



(a)Process Curve (b) Error Curve

Fig 5. Results in case of Square Wave Input



(a)Process Curve (b) Error Curve

Fig 6. Results in case of Square Wave Input

5. OUR SIMULATIONS

The model that is implemented by us is that of a RBFN Based Predictive Controller for the non-linear plant as described by the U-Model.

The detailed description of the NN Predictive controller is given in Appendix A.

To implement the RBF neural network, the activation function of the hidden layer of the neural network of the block is replaced from tan sigmoid to radial basis, as shown below:

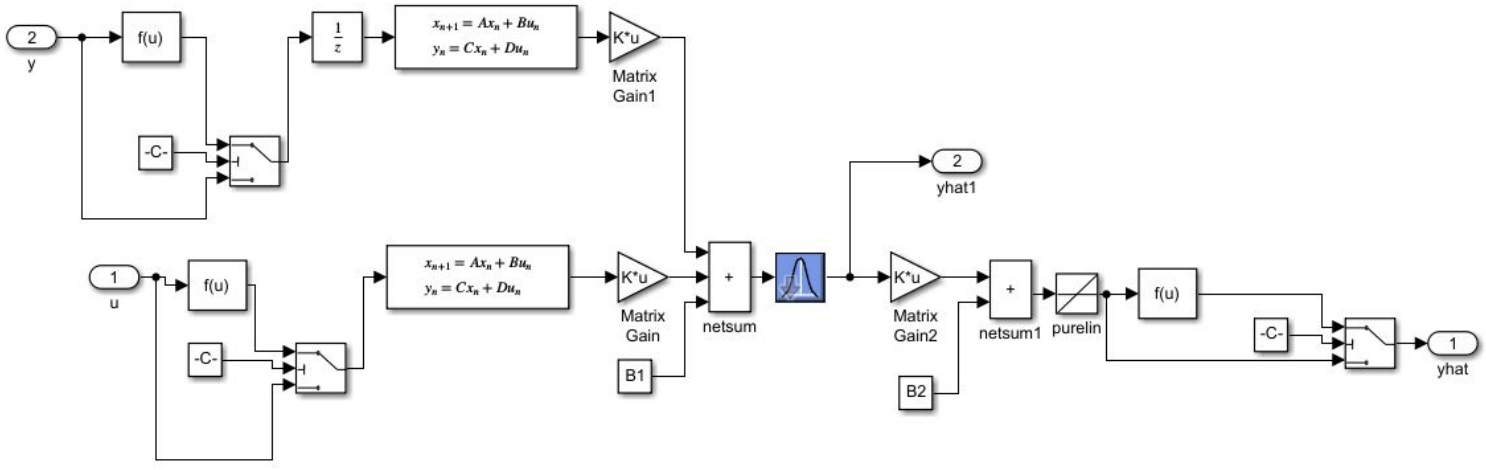


Fig 7. Modified NN architecture of the Neural Network of the Predictive NN Controller block.

This network has two inputs, the plant output (y) and the reference signal (u).

Now, for the simulation, the neural predictive controller is used to design RBFN based controllers for the plant. We have taken 3 plants into consideration:

- 1st degree U-model plant
- 4th degree U-model plant
- 7th degree U-model plant

The general steps involved in implementing the algorithm are described below:

1. The Neural Predictive controller block is used to identify the plant at first. For this, the neural network is trained by generating random data for the given plant and based on the input and output, the network is trained (in this case, on 10000 data samples).
2. After training, the plant is simulated using discrete sine wave input of sampling period similar to that used while training the model.
3. The results obtained are analysed

For our simulation, we have assumed the sine wave to be a discrete signal with sampling time of 0.2 seconds. The same parameter has been used to train the neural predictive algorithm. This finite nonzero sampling period is used to analyse the performance in a more real scenario, as in real systems, sampling is not continuous.

The details of hyperparameters used have been described individually as they vary on case by case basis. However, the schematic of simulation remains the same on the block level and is shown below:

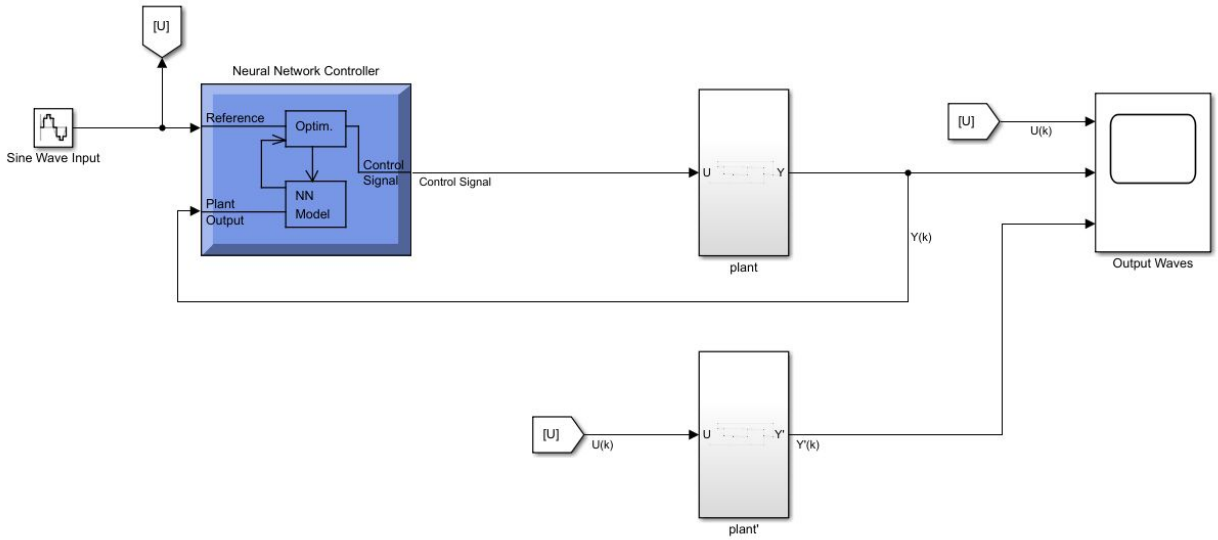


Fig 8. Simulation Setup

5.1. 1ST DEGREE PLANT SIMULATION

For the simulation, we have assumed the first degree plant given by the equation -

$y(k) = 0.5y(k-1) + 0.2y^2(k-1) + 0.2u(k-1)$

The simulink model of the plant is obtained as follows -

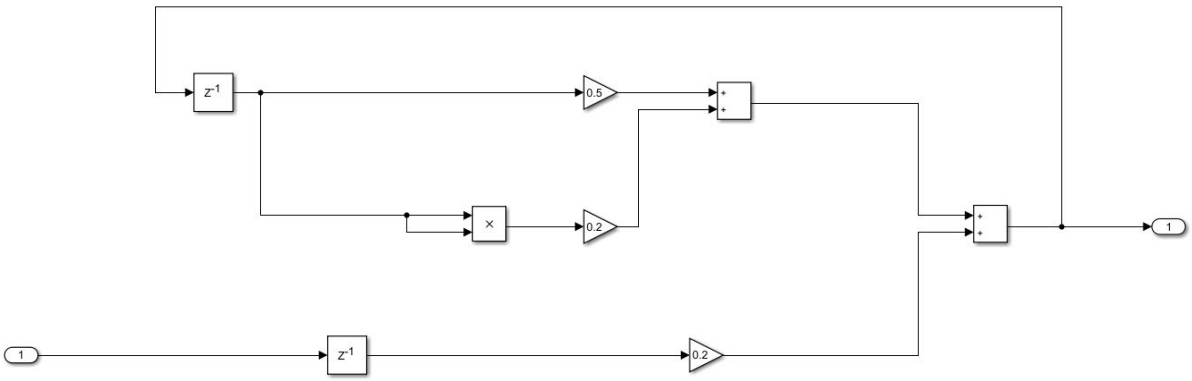
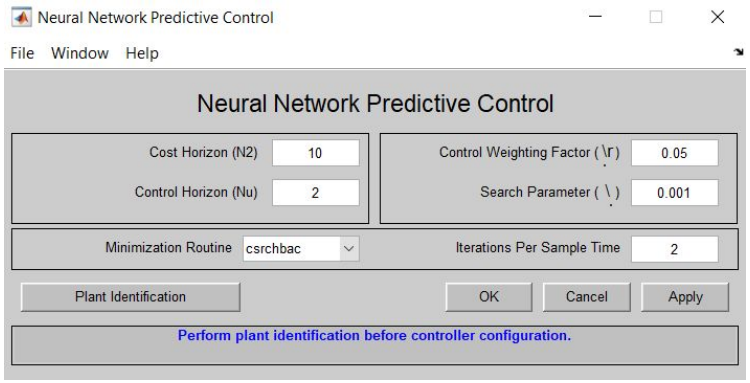
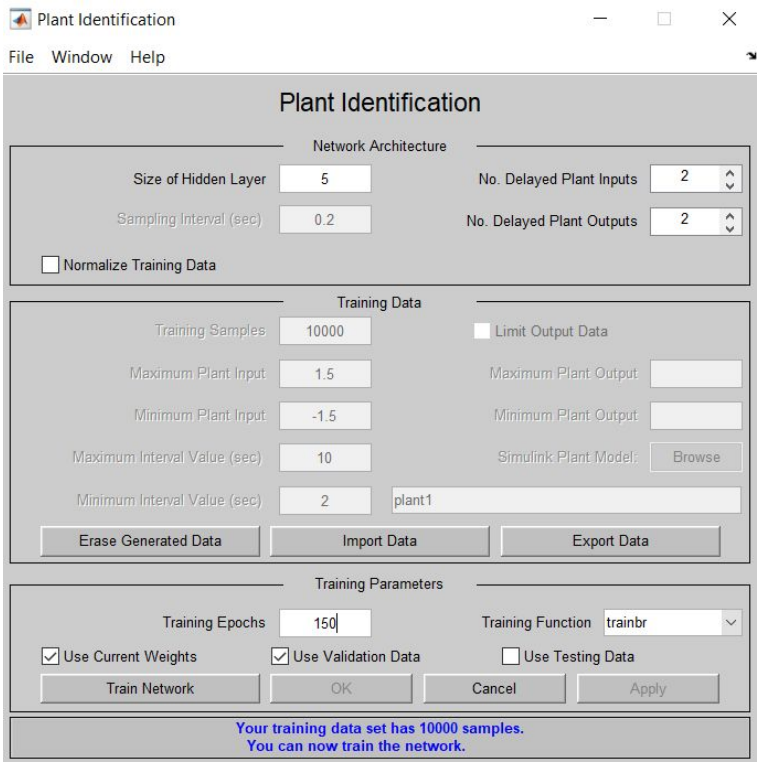


Fig 9. Plant 1 Model

Now, to train the RBFN neural predictive controller, we setup the following hyperparameters -





As it can be seen here, the sampling period is kept as 0.2 seconds.

Thereafter, random data is generated and is loaded for training. The random data can be visualized as -

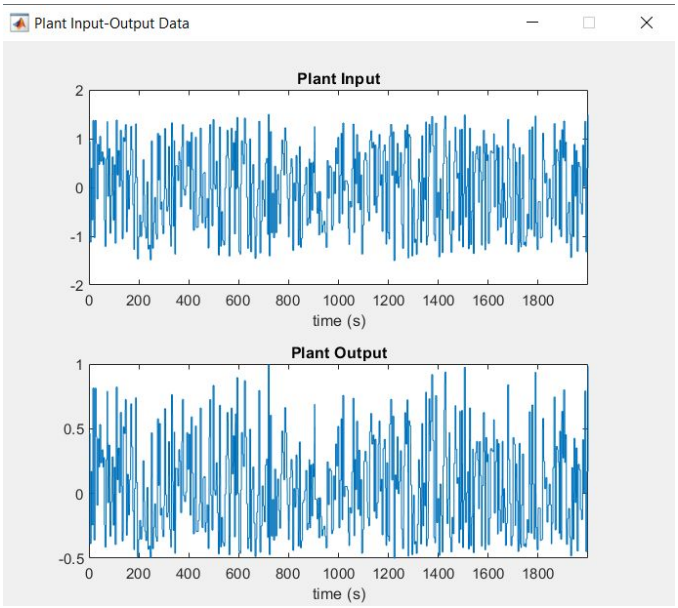


Fig 10. Random Data

After that, training is done for 150 epochs, and the final results of the training process are as follows -

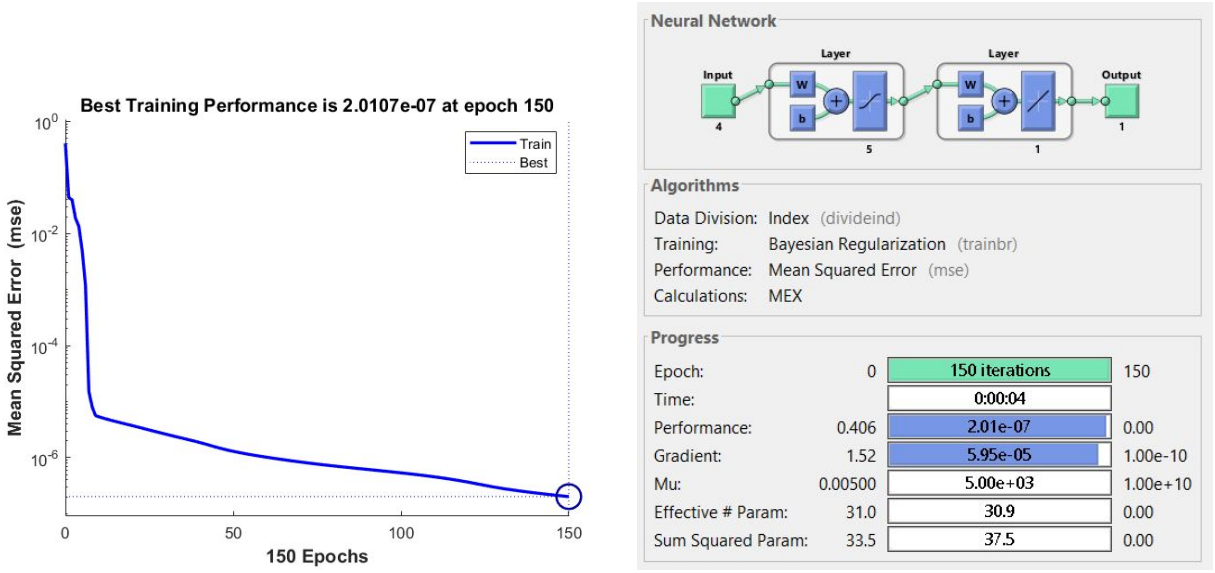
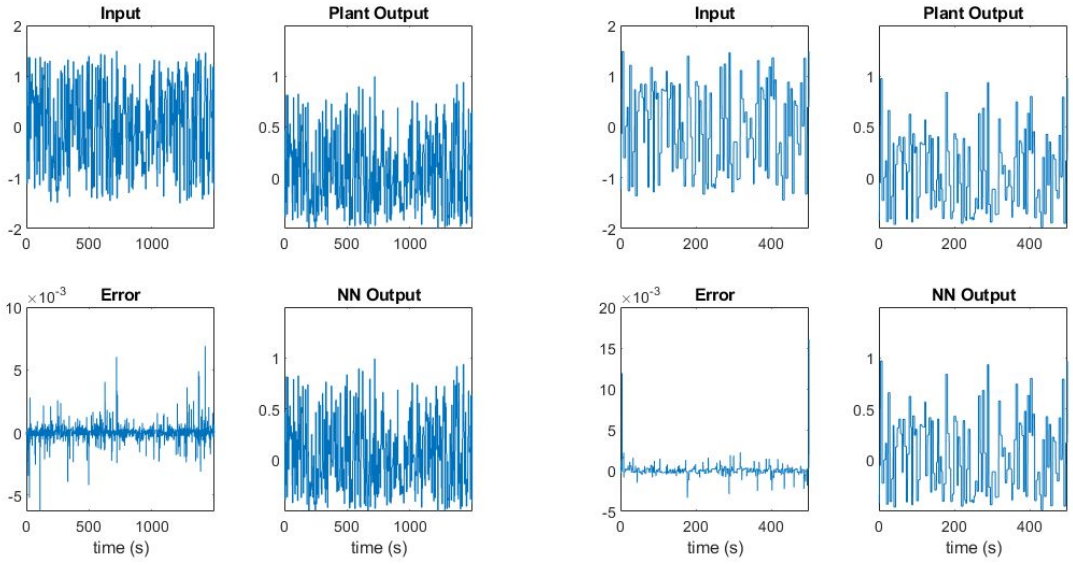


Fig 11. Training Results

It can be clearly seen that the mean squared error after 150 epochs of training is obtained to be 2.01 X 10^{-7} which is fairly good. Just to further validate the correctness of the training process, the following graphs of the input, output and error are also included.



(a) (b)

Fig 12. Evaluation of (a) Training and (b) validation process

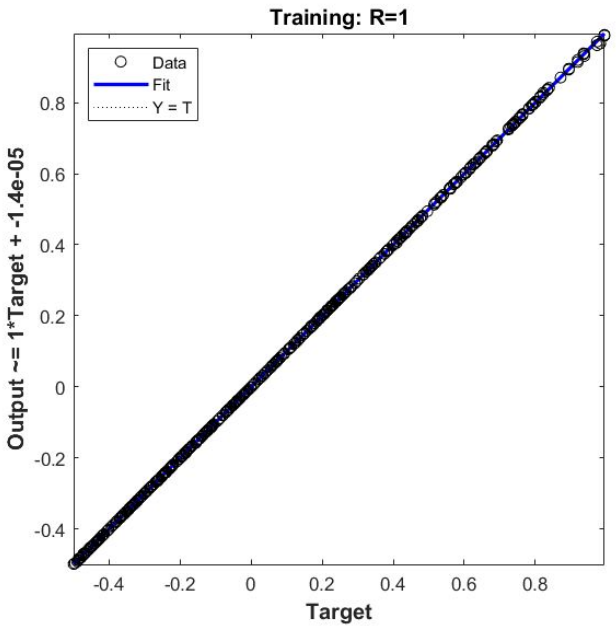


Fig 13. Regression Fit Curve

Now, to obtain final results, the entire model is simulated on a discrete sine input of sampling period 0.2 seconds and results are as follows -

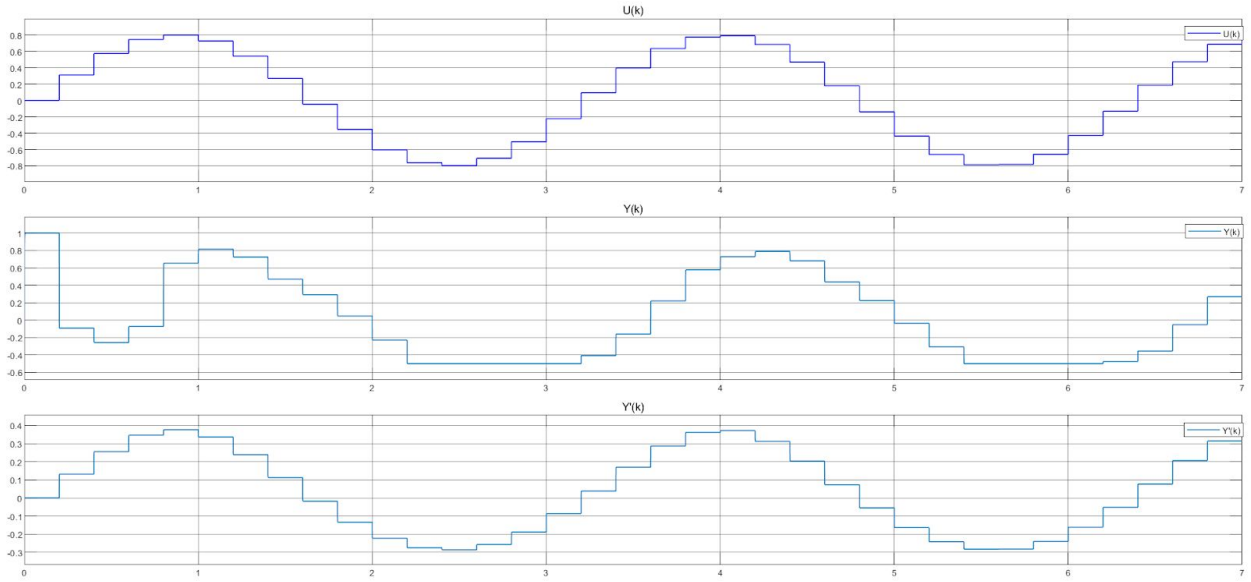


Fig 14. Simulation Results (Y is with controller and Y' is without controller)

5.2. 4TH DEGREE PLANT SIMULATION

4th degree controlled plant is given by the following equation-

$y(k) = 0.5y(k-1) + 0.2y^2(k-1) + 0.2u(k-1) - 0.7y(k-1)u^2(k-1) + 0.5y(k-1)u^3(k-1) - 0.3y(k-1)u^4(k-1)$

For simulation, the model of the plant is -

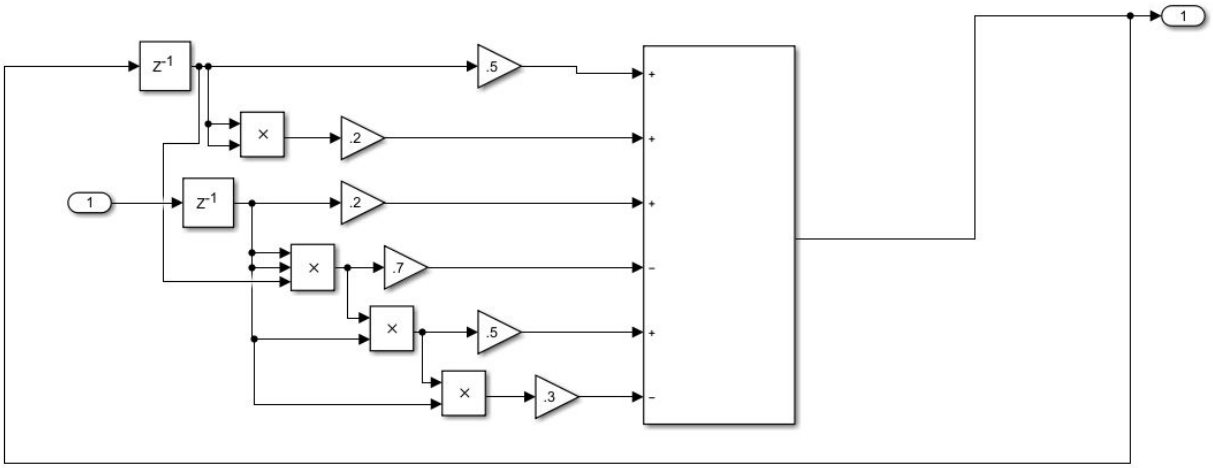
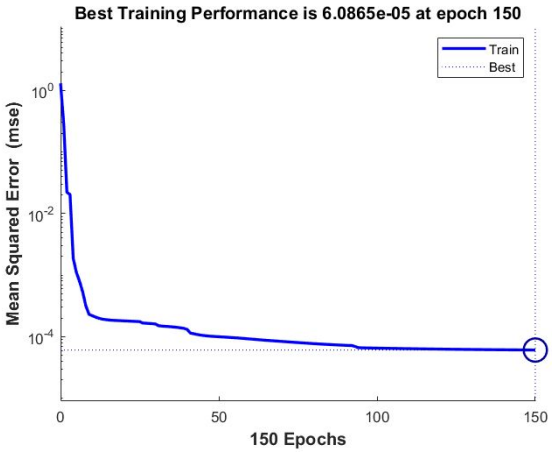


Fig 15. Plant 2 Model

For this case, 1 change has been made in the hyperparameters -

1. The Plant time range is changed to 3s-10s

The same random data generation and training process is carried out again and the training results are obtained as follows -



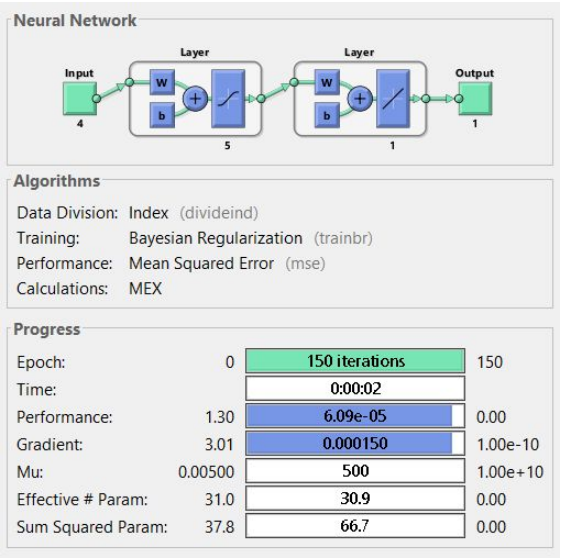
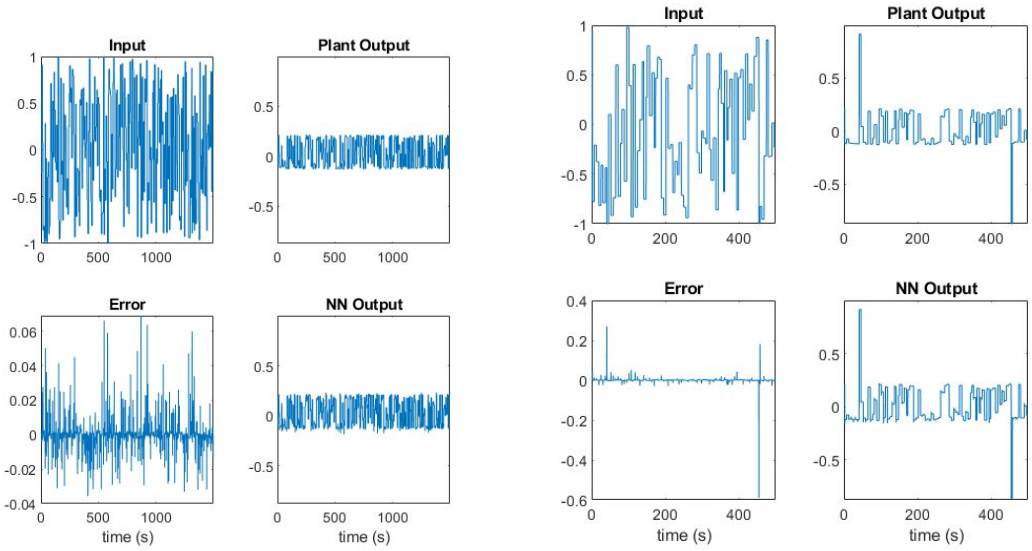


Fig 16. Training Results

It can be clearly seen that the mean squared error after 150 epochs of training is obtained to be 6.05 X 10^{-5} which is fairly good, and optimal since we can clearly see that the **gradient has saturated**. Just to further validate the correctness of the training process, the following graphs of the input, output and error are also included.



(a) (b)

Fig 17. Evaluation of (a) Training and (b) validation process

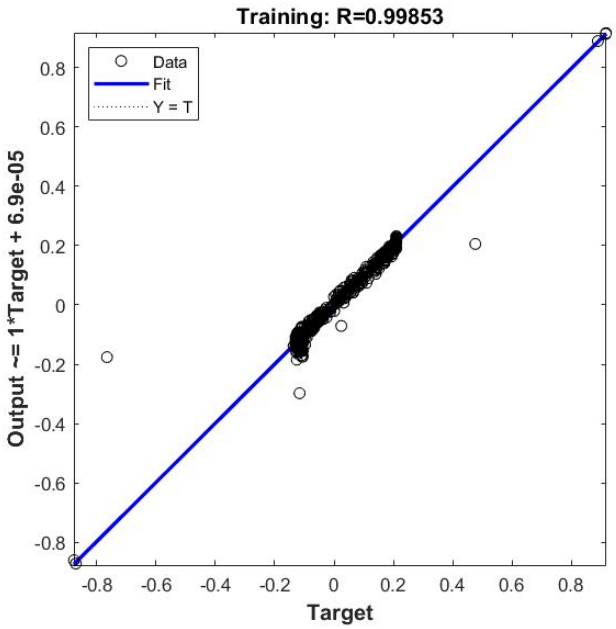


Fig 18. Regression Fit Curve

Now, to obtain final results, the entire model is simulated on a discrete sine input of sampling period 0.2 seconds and results are as follows -

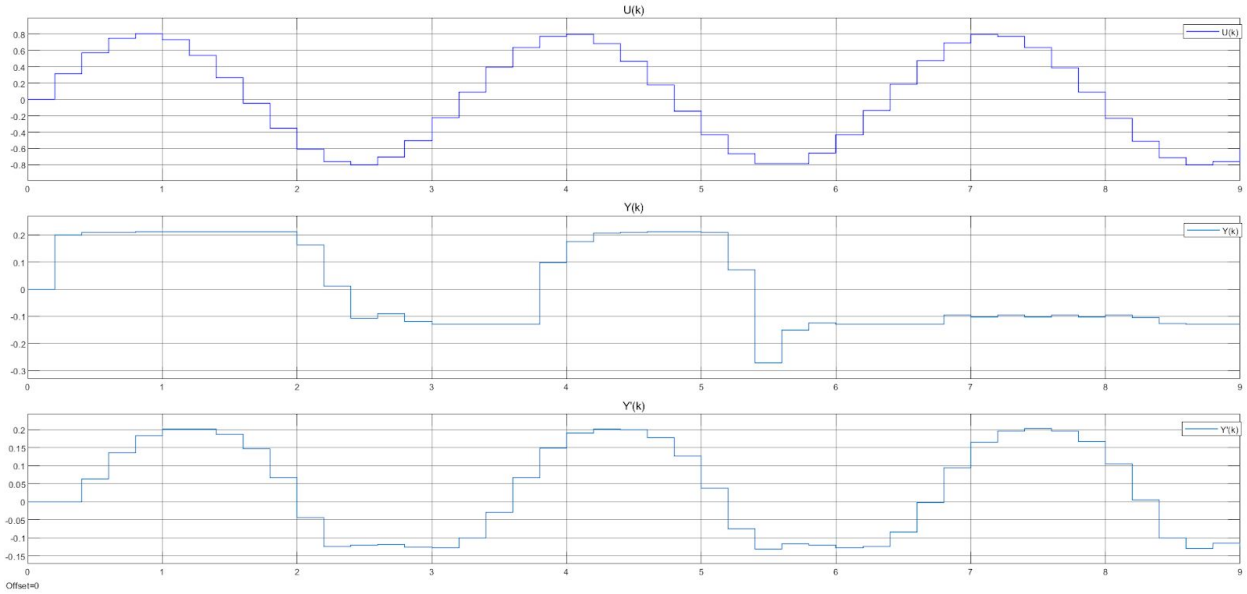


Fig 19. Simulation Results (Y is with controller and Y' is without controller)

6. CONCLUSION

This report shall be concluded in two parts, one with the conclusion of the paper and one with respect to the simulations carried out.

For the paper, It is observed that there is a significant relationship between the control plant and the reference model used, so it directly influences the accuracy of the system. It has also been proved that the accuracy of the control is best when reference as well as nonlinear control plant has the same degree. This finding can play a critical role in deciding the reference model in accordance with the nonlinear plant for which it is developed in adaptive control. A neural network algorithm could be used in such nonlinear systems with the help of U-model. By adopting U-model, NN algorithms could be used in existing classic adaptive control reference methods to design nonlinear systems.

As far as the simulations are concerned, it can be clearly seen that the performance of the controller of the 1st degree plant is far better than the 4th degree plant, and will subsequently degrade. This is why the need for additional control schemes like adaptive control, feedforward control, etc. are very important for these nonlinear systems. Since the gradient also saturated in the 4th degree plant simulation, it can safely be assumed that more optimal training is not possible for it.

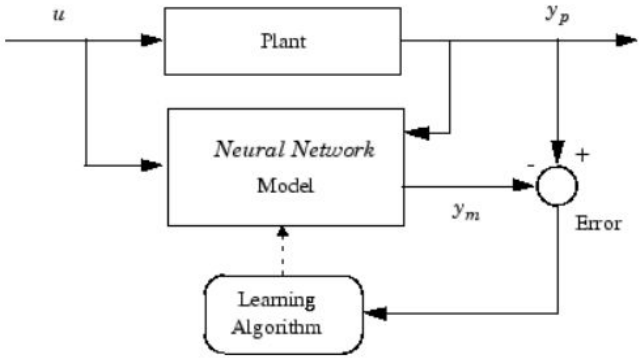
APPENDIX 'A': NEURAL NETWORK PREDICTIVE CONTROLLER

It is a model of nonlinear plants that uses deep neural networks to predict the future output of the plant. The inputs of the controller are calculated such that they give the best possible performance of the plant for a given future time.

Model Prediction:

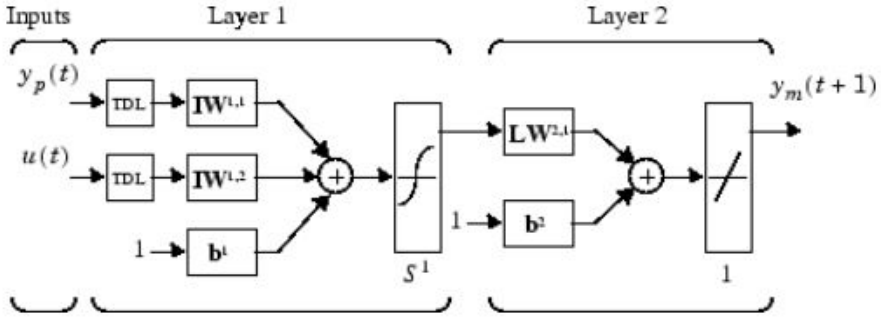
The first step is to identify the system or NN Plant model and after identifying the system, the modelled network is then used by the controller to predict the future performance of the plant Plant model identification.

In predictive control models, first we have to train the neural network model such that it has correct forward direction. The training data is the prediction error between the plant output and the designed neural network output.



Such neural network models predict the future values of the plant output using previous values of control input and the previous values of the predicted output of the plant.

The NN model structure is given below:



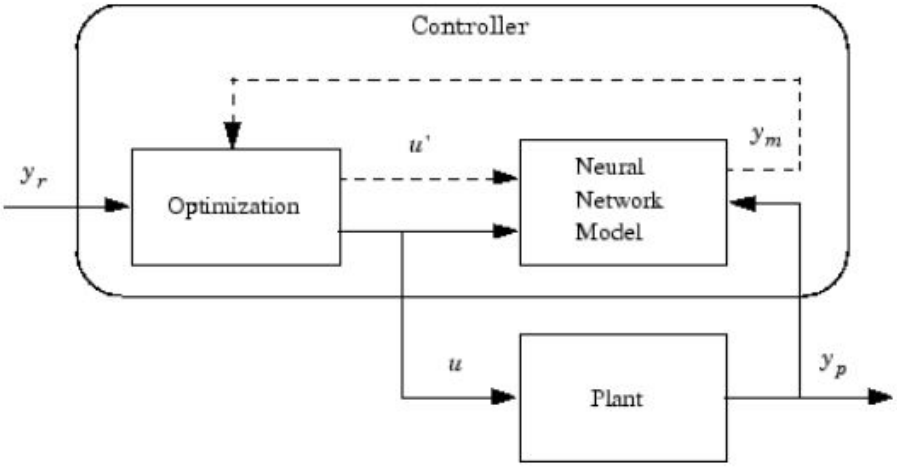
This given network is then trained using backpropagation and Multilayer Shallow neural Networks.

Since the models predict the plant output for some future time, these predictions are then used to generate control signals that minimize a specific criterion of performance in a mathematically

formulated optimization function given as:

$$J = \sum_{j=N_1}^{N_2} (y_r(t+j) - y_m(t+j))^2 + \rho \sum_{j=1}^{N_u} (u'(t+j-1) - u'(t+j-2))^2$$

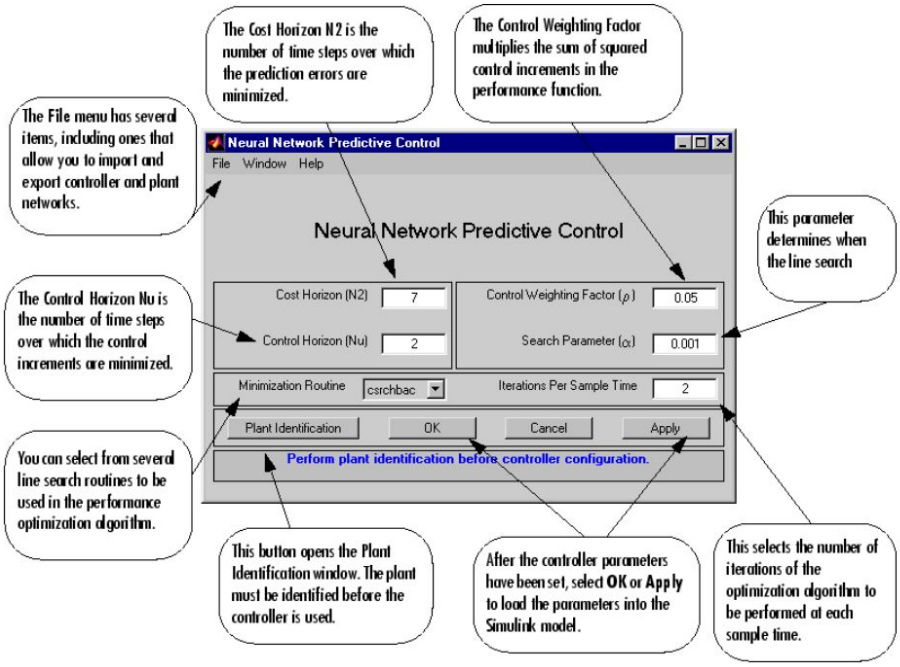
where N1, N2, and Nu are the ranges of the tracking error and the control increment. yr is the desired response,the u′ variable is the provisional control signal and ym is the model response. The value of ρ decides the amount of contribution the sum of the squares of the control increments has on the performance index.
The predictive control process is represented by the block diagram as:

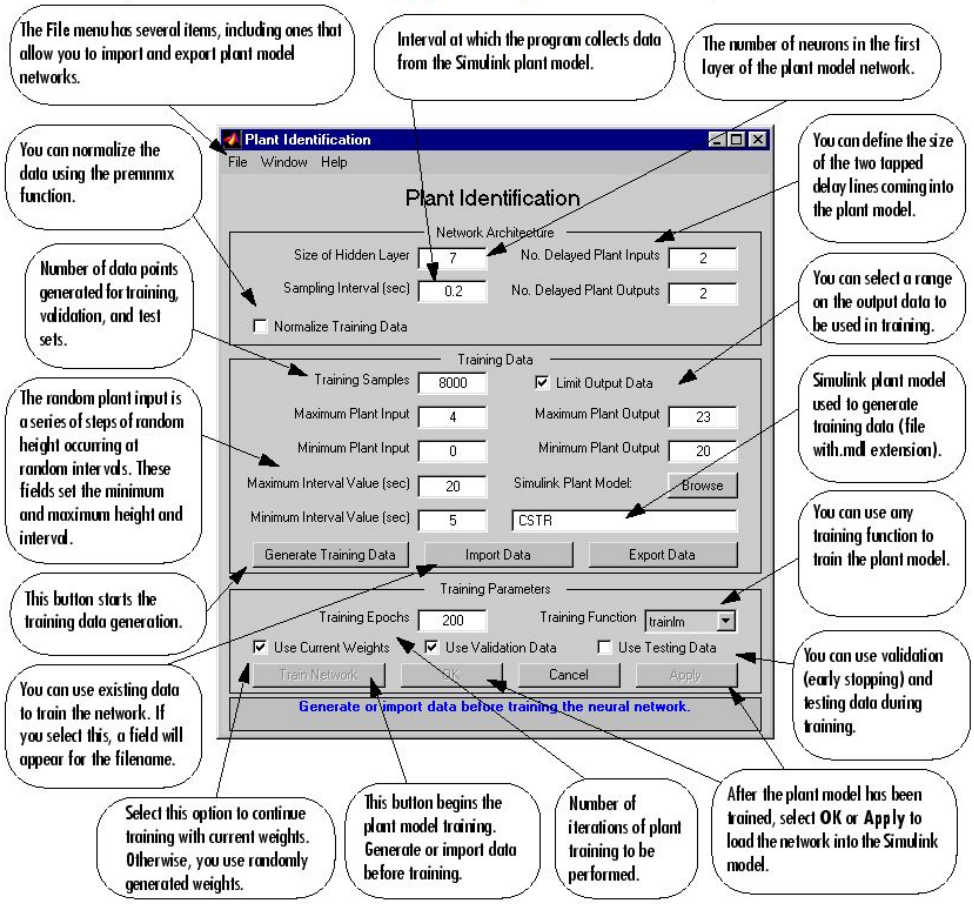


Here the optimization block and neural network model both constitute a controller. The role of the optimization block is to find the value of u' that minimizes the performance parameter J. After that, the optimal value of u is given as input to the plant which produces output y_p.

To train, the first step is to identify the plant using the data. The data can either be imported, or can be generated by loading the plant model file. After that, using the data, the neural network is trained.

The names and uses of various hyper parameters that need to be set for training the NN predictive controller are as follows:


The Cost Horizon N2 is the number of time steps over which the prediction errors are minimized.
The Control Weighting Factor multiplies the sum of squared control increments in the performance function.
The File menu has several items, including ones that allow you to import and export controller and plant networks.
Neural Network Predictive Control
File Window Help
Neural Network Predictive Control
Cost Horizon (N2) 7
Control Weighting Factor (ρ) 0.05
Control Horizon (Nu) 2
Search Parameter (α) 0.001
Minimization Routine csrchbac
Iterations Per Sample Time 2
Plant Identification
OK
Cancel
Apply
Perform plant identification before controller configuration.
This parameter determines when the line search
The Control Horizon Nu is the number of time steps over which the control increments are minimized.
You can select from several line search routines to be used in the performance optimization algorithm.
This button opens the Plant Identification window. The plant must be identified before the controller is used.
After the controller parameters have been set, select OK or Apply to load the parameters into the Simulink model.
This selects the number of iterations of the optimization algorithm to be performed at each sample time.


The File menu has several items, including ones that allow you to import and export plant model networks.
Interval at which the program collects data from the Simulink plant model.
The number of neurons in the first layer of the plant model network.
Plant Identification
File Window Help
Plant Identification
Network Architecture
Size of Hidden Layer 7
No. Delayed Plant Inputs 2
Sampling Interval (sec) 0.2
No. Delayed Plant Outputs 2
Normalize Training Data
Training Data
Training Samples 8000
Limit Output Data
Maximum Plant Input 4
Maximum Plant Output 23
Minimum Plant Input 0
Minimum Plant Output 20
Maximum Interval Value (sec) 20
Simulink Plant Model: Browse
Minimum Interval Value (sec) 5
CSTR
Generate Training Data
Import Data
Export Data
Training Parameters
Training Epochs 200
Training Function trainlm
Use Current Weights
Use Validation Data
Use Testing Data
Train Network
OK
Cancel
Apply
Generate or import data before training the neural network.
You can normalize the data using the premnmx function.
You can define the size of the two tapped delay lines coming into the plant model.
Number of data points generated for training, validation, and test sets.
You can select a range on the output data to be used in training.
The random plant input is a series of steps of random height occurring at random intervals. These fields set the minimum and maximum height and interval.
Simulink plant model used to generate training data (file with.mdl extension).
You can use any training function to train the plant model.
This button starts the training data generation.
You can use existing data to train the network. If you select this, a field will appear for the filename.
You can use validation (early stopping) and testing data during training.
Select this option to continue training with current weights. Otherwise, you use randomly generated weights.
This button begins the plant model training. Generate or import data before training.
Number of iterations of plant training to be performed.
After the plant model has been trained, select OK or Apply to load the network into the Simulink model.

APPENDIX 'B': OUR ATTEMPT TO CREATE MODEL REFERENCE ADAPTIVE CONTROLLER

Initially, we also tried creating the model reference adaptive controller, and could not just create the RBFN adaptive control algorithm, as also notified earlier. However, the attempt to do so is described as follows:

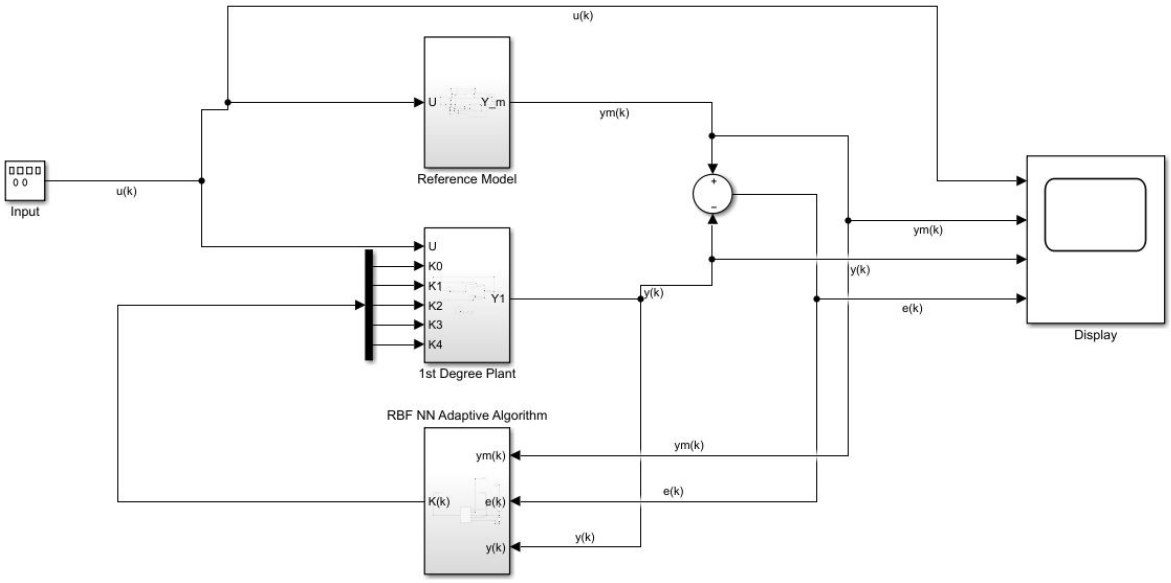


Fig 20. Overall Schematic of the Model

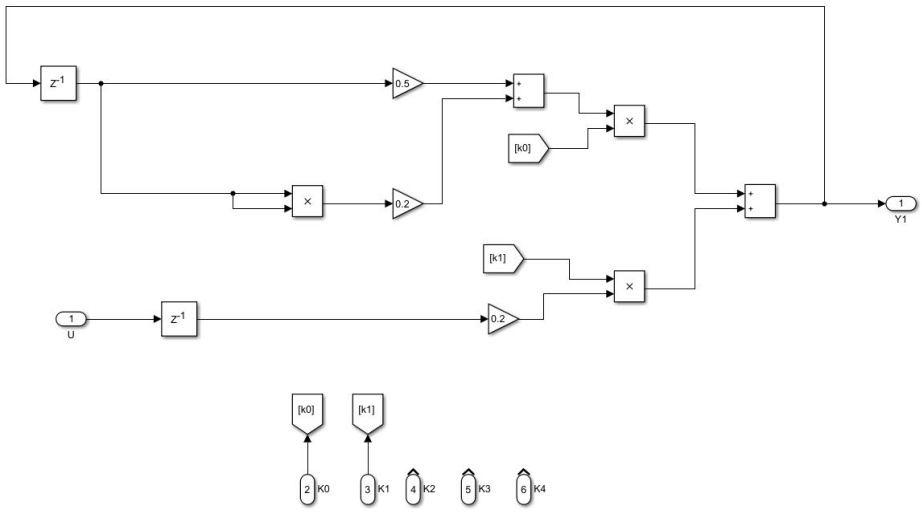


Fig 21. Plant Model 1A (1st degree) with compensation parameters

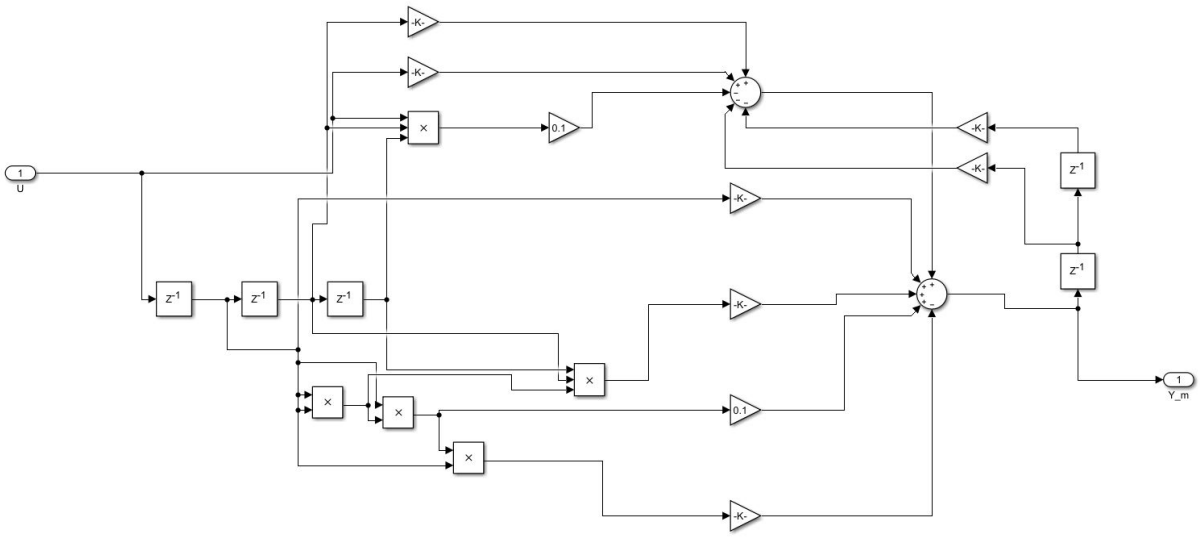


Fig 22. Reference Model 1

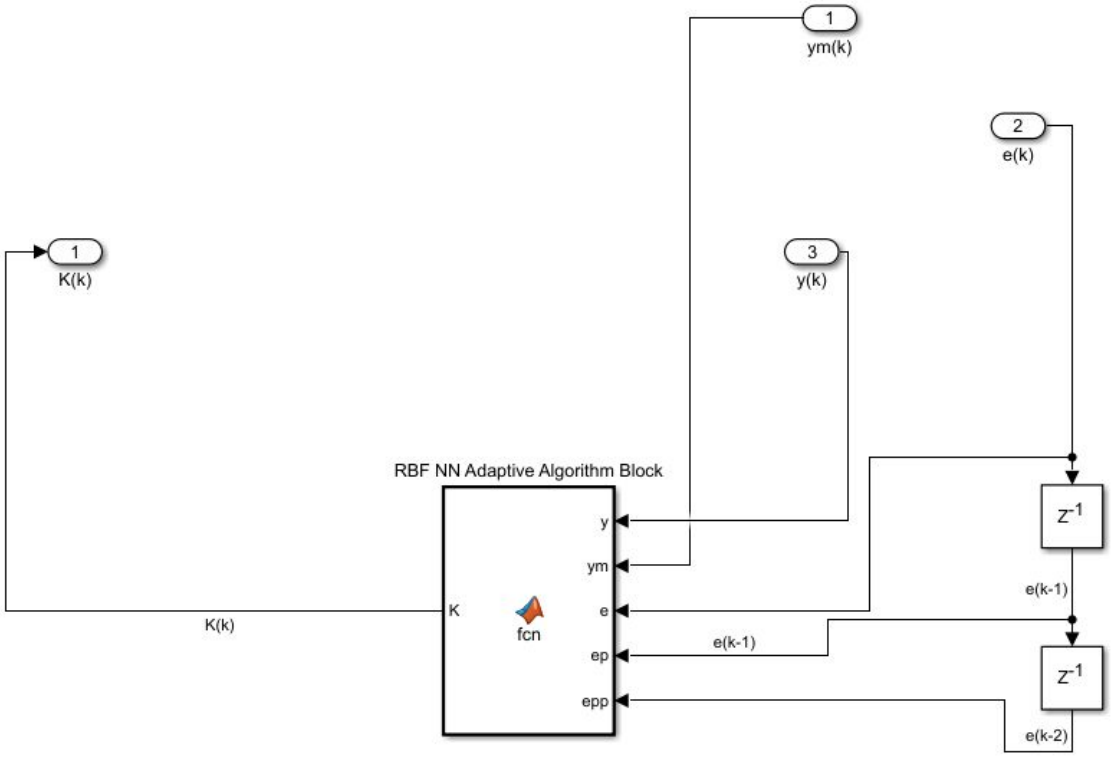


Fig 23. RBF NN Adaptive Algorithm (fcn not implemented)

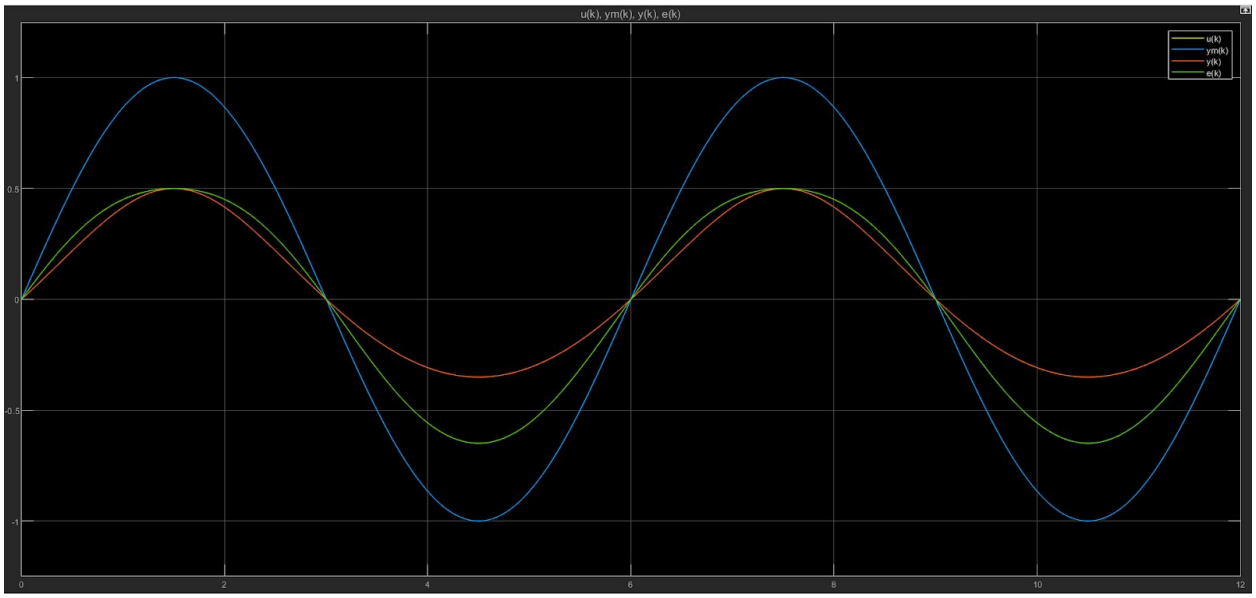


Fig 24. Output without RBFN Controller

The file is included along with this project as sim1.slx. No explanation is provided here as it is already explained above in the discussion part.