# INTERNAL MODEL CONTROL OF LINEAR AND NON-LINEAR SYSTEMS USING NEURAL NETWORKS

*M. Saqib Sohail (260260)* *Ahmed A. Quadeer (260268)*

Electrical Engineering Dept., King Fahd University of Petroleum & Minerals (KFUPM)

**ABSTRACT**

This paper deals with the Internal model control of linear and non-linear systems using Feed-forward Neural network. Control is achieved by estimating the plant and then finding its inverse model using neural network. The back propagation algorithm is used to train the neural networks. Simulation results have been shown for linear minimum phase, linear non-minimum phase and non-linear systems.

***Index Terms—*** Internal Model Control, Feed-forward Neural Network, Back Propagation algorithm.

## 1. INTRODUCTION

The problem of controlling a plant consequently a very large variety of control techniques are available in the literature for modeling of linear and non linear plants. These include adaptive modeling techniques for linear plant modeling, adaptive inverse modeling, internal model control and non-linear Auto Regressive Moving Average (ARMA) models.

In recent years, neural Networks based techniques have also been explored. Neural Network based approaches are attractive as neural networks have an inherent ability to approximate nonlinear functions and so prove useful to model non linear plants. Internal Model Control (IMC) structure models the plant in parallel with the model of the plant with a controller placed at their input. If the model is exact, there will be no feedback to the controller. For inexact plant models and in the presence of disturbance, the input to the controller is the difference of the set point and the feedback signal. Here we have used two Feed Forward Neural Networks (FFNN), one to model the plant and one as controller. The back propagation algorithm is used to determine the parameters of the models.

In this paper we have compared the performance of IMC using neural networks for linear minimum phase, linear non-minimum phase and non-linear systems.

The outline of this paper is as follows. Section 2 describes the Internal model control strategy. Section 3 briefly explains the Feed-forward Neural network implemented in this paper. System model is described in Section 3. Simulation results are discussed in Section 4 and the conclusions are summed up in Section 5.

## 2. INTERNAL MODEL CONTROL

The Internal Model Control (IMC) philosophy relies on the Internal Model Principle, which states that control can be achieved only if the control system encapsulates, either implicitly or explicitly, some representation of the process to be controlled. In particular, if the control scheme has been developed based on an exact model of the process, then perfect control is theoretically possible. Consider, for example, the system shown in the figure below:

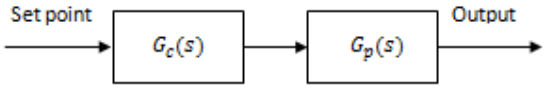


**Figure 1.** Open Loop Control Strategy

A controller, $G_c(s)$.,is used to control the process, $G_p(s)$. Suppose $\widetilde{G_p}(s)$ is a model of $G_p(s)$. By setting $G_c(s)$ to be the inverse of the model of the process,

$$G_c(s) = \widetilde{G_p}(s)^{-1} \quad (1)$$

and if $G_p(s) = \widetilde{G_p}(s)$, (the model is an exact representation of the process).

Then it is clear that the output will always be equal to the setpoint. Notice that this ideal control performance is achieved without feedback. What this tells us is that if we have complete knowledge about the process (as encapsulated in the process model) being controlled, we can achieve perfect control. It also tells us that feedback control is necessary only when knowledge about the process is inaccurate or incomplete.

In practice, however, process-model mismatch is common; the process model may not be invertible and the system is often affected by unknown disturbances. Thus the above open loop control arrangement will not be able to maintain output at setpoint. Nevertheless, it forms the basis for the development of a control strategy that has the potential to achieve perfect control. This strategy, known as Internal Model Control (IMC), has the general structure depicted in Figure 2.

In the Figure 2, *d(s)* is an unknown disturbance affecting the system. The manipulated input *U(s)* is introduced to both the process and its model. The process

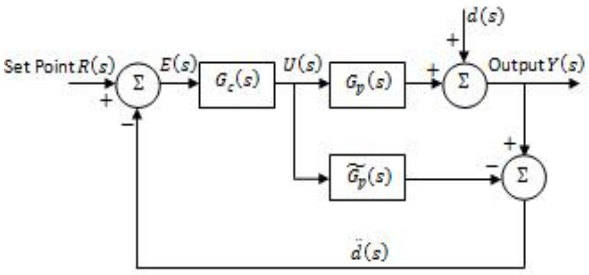


**Figure 2.** IMC Scheme

output, *Y(s)*, is compared with the output of the model, resulting in a signal $\hat{d}(s)$.That is,

$$\hat{d}(s) = \left[G_p(s) - \widetilde{G_p}(s)\right]U(s) + d(s) \qquad (2)$$

If $d(s)$ is zero for example, then $\hat{d}(s)$ is a measure of the difference in behavior between the process and its model. If $G_p(s) = \widetilde{G_p}(s)$, then $\hat{d}(s)$ is equal to the unknown disturbance. Thus $\hat{d}(s)$ may be regarded as the information that is missing in the model, $G_p(s)$ and can therefore be used to improve control. This is done by subtracting $\hat{d}(s)$ from the set point *R(s)*, which is very similar to affecting a set point trim. The resulting control signal is given by,

$$\begin{aligned} U(s) &= \left[R(s) - \hat{d}(s)\right]G_c(s) \\ &= \left\{R(s) - \left[G_p(s) - \widetilde{G_p}(s)\right]U(s) - d(s)\right\}G_c(s) \end{aligned} \qquad (3)$$

Thus,

$$U(s) = \frac{[R(s) - d(s)]G_c(s)}{1 + \left[G_p(s) - \widetilde{G_p}(s)\right]G_c(s)} \qquad (4)$$

Since

$$Y(s) = G_p(s)U(s) + d(s) \qquad (5)$$

The closed loop transfer function for the IMC scheme is therefore

$$Y(s) = \frac{[R(s) - d(s)]G_c(s)G_p(s)}{1 + \left[G_p(s) - \widetilde{G_p}(s)\right]G_c(s)} + d(s) \qquad (6)$$

Or

$$Y(s) = \frac{G_c(s)G_p(s)R(s) + \left[1 - G_c(s)\widetilde{G_p}(s)\right]d(s)}{1 + \left[G_p(s) - \widetilde{G_p}(s)\right]G_c(s)} \qquad (7)$$

From this closed loop expression, we can see that if $G_c(s) = \widetilde{G_p}(s)^{-1}$, and if $G_p(s) = \widetilde{G_p}(s)$, then perfect setpoit tracking and disturbance rejection is achieved. Notice that theoretically even if $G_p(s) \neq \widetilde{G_p}(s)$, prefect disturbance rejection can still be realized provided $G_c(s) = \widetilde{G_p}(s)^{-1}$.

Additionally, to improve robustness, the effects of process model mismatch should be minimised. Since discrepancies between process and model behaviour usually occur at the high frequency end of the system's frequency response, a low-pass filter $G_f(s)$ is usually added to attenuate the effects of process-model mismatch. Thus, the internal model controller is usually designed as the inverse of the process model in series with a low-pass filter, i.e. $G_{IMC}(s) = G_c(s)G_f(s)$. The order of the filter is usually chosen such that $G_c(s)G_f(s)$ is proper, to prevent excessive differential control action. The resulting closed loop then becomes

$$Y(s) = \frac{G_{IMC}(s)G_p(s)R(s) + \left[1 - G_{IMC}(s)\widetilde{G_p}(s)\right]d(s)}{1 + \left[G_p(s) - \widetilde{G_p}(s)\right]G_{IMC}(s)} \qquad (8)$$

Designing an internal model controller is relatively easy. Given a model of the process, $\widetilde{G_p}(s)$, first factor $\widetilde{G_p}(s)$ into invertible and non-invertible components.

$$\widetilde{G_p}(s) = \widetilde{G_p^+}(s)\widetilde{G_p^-}(s) \qquad (9)$$

The non-invertible component $\widetilde{G_p^-}(s)$, contains terms which if inverted, will lead to instability and realisability problems, e.g. terms containing positive zeros and time-delays.

Next, set $G_c(s) = \widetilde{G_p^+}(s)^{-1}$ and then $G_{IMC}(s) = G_c(s)G_f(s)$, where $G_f(s)$ is a low pass function of appropriate order.

## 3. NEURAL NETWORKS

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. We may therefore speak of the learning algorithm used in the design of the neural networks as being structured. In general, there are two different classes of network architectures:

### 3.1. Single Layer Feed-forward Network

In layered neural networks, the neurons are organized in the form of layers. In the simplest form of a layered network, we have an input layer of source nodes that projects onto an output layer of neurons, but not vice versa. In other words, the network is strictly a feedforward network, as shown in Figure 3. Such a network is known as a single layer network.

### 3.2. Multilayer Feed-forward Network

The second class of feedforward neural network distinguishes itself by the presence of one or more hidden layers whose computation nodes are correspondingly called hidden neurons. The function of the hidden neurons is to intervene between the external inputs and the network output in some useful manner. By adding one

or more hidden layer, the network is enabled to extract higher order statistics.

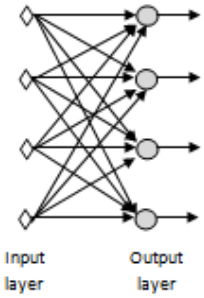


**Figure 3.** Single layer Feed-forward neural network

The source nodes in the input layer of the neural network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computable nodes) in the second layer (i.e., the first hidden layer). The output signals of the second layer are used as the inputs to the third layer, and so on for the rest of the network. Typically the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer. The architectural graph in Figure 4 shows the connection in a multilayer feedforward neural network. A multilayer neural network with m source nodes, h hidden nodes and q neurons in the output layer is known as an m-h-q network.

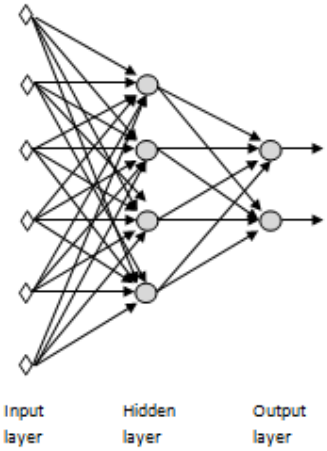


**Figure 4.** Multi-layer feed-forward neural network

### 3.3. Back propagation Algorithm

The error signal at the output of neuron j at iteration n is defined by

$$e_j(n) = d_j(n) - y_j(n) \qquad (10)$$

where $e_j(n)$ refers to the error signal at the output of neuron $j$, $d_j(n)$ is the desired response and $y_j(n)$ is the function signal appearing at the output of the $jth$ neuron. We define the instantaneous value of the error energy for neuron j as $\frac{1}{2}e_j^2(n)$. Correspondingly, the instantaneous value $\xi(n)$ of the total error energy is obtained by summing $\frac{1}{2}e_j^2(n)$ over all neurons in the output layer; these are the only visible neurons for which error signals can be calculated directly. We may thus write

$$\xi(n) = \frac{1}{2}\sum_{j=C} e_j^2(n) \qquad (11)$$

where the set $C$ includes all the neurons in the output layer of the network. Let $N$ denote the total number of patterns contained in the training set. The averaged squared error energy is obtained by summing $\xi(n)$ over all $n$ and then normalizing with respect to the set size $N$, as shown by

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n) \qquad (12)$$

The instantaneous error energy $\xi(n)$ and the average error energy $\xi_{av}$ , is a function of all the free parameters of the network. For a given set of training set, $\xi_{av}$ represents the cost function as a measure of learning performance. The objective of the learning process is to adjust the free parameters of the network to minimize $\xi_{av}$.

The sequential updating of weights is preferred method for on-line implementation of the back propagation algorithm, \for this mode of operation, the algorithm cycles through the training sample $\{(x(n), d(n))\}_{n=1}^{N}$ as follows:

3.3.1. *Initialization:* Assuming that no prior information is available, pick the synaptic weighted and thresholds from a uniform distribution whose mean is zero and whose variance is chosen to make the standard deviation of the induced local fields of the neurons lie at the transition between the linear and saturation parts of the sigmoid activation function.

3.3.2. *Forward Computation:* Let a training example in the epoch be denoted by $(x(n), d(n))$, with the input vector $x(n)$ applied to the input layer of sensory nodes and he desired response vector $d(n)$ presented to the output layer of computation nodes. Compute the induced local fields and function signals of the network by preceeding forward through the network, layer by layer. The induced local field $v_j^{(l)}(n)$ for neuron $j$ in layer $l$ is

$$v_j^{(l)} = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \qquad (13)$$

where $y_i^{(l-1)}(n)$ the output is signal of neuron i in the previous layer $l-1$ at iteration n and $w_{ji}^{(l)}(n)$ is the synaptic weight of neuron j in the layer l that is fed from neuron i in layer $l-1$. For $i=0$, we have $y_0^{(l-1)}(n) = +1$ and $w_0^{(l)}(n) = b_j^{(l)}(n)$ is the boas applied to neuron j in layer l. For a sigmoid function, the output signal of neuron j in layer l is

$$y_j^{(l)} = \varphi_j\left(v_j(n)\right) \tag{14}$$

where $\varphi_j(.)$ denots the activation function describing the input-output functional relationship of the nonlinearity associated with neuron j. If neuron j is in the first hidden layer (i.e., $l=1$), set

$$y_j^{(0)}(n) = x_j(n) \tag{15}$$

where $x_j(n)$ is the jth element of the input vector $x(n)$. If neuron j is in the output layer (i.e., $l=L$, where L is reffered to as the depth of the network), set

$$y_j^{(L)}(n) = o_j(n) \tag{16}$$

Compute the error signal as

$$e_j(n) = d_j(n) - o_j(n) \tag{17}$$

where $d_j(n)$ is the jth element of the desired response vector $d(n)$.

*3.3.3. Backward Computation:* Compute the δs (i.e., local gradients) of the network, defined by

$$\delta_j^{(l)} = \begin{cases} e_j^{(L)}(n)\varphi_j'\left(v_j^{(L)}(n)\right) \\ \qquad for\ neuron\ j\ in\ output\ layer\ L \\ \varphi_j'\left(v_j^{(l)}(n)\right)\sum_k \delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n) \\ \qquad for\ neuron\ j\ in\ hidden\ layer\ l \end{cases} \tag{18}$$

where the prime in $\varphi_j'(.)$ denotes differentiation with respect to the argument. Adjust the synaptic weights of the network layer $l$ according to the generalized delta rule:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)} + \alpha[w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n)y_j^{(l-1)} \tag{19}$$

where $\eta$ is the learning rate parameter and $\alpha$ is the momentum constant.

*3.3.4. Iteration:* Iterate the forward and backward computations under points 2 and 3 by presenting new epochs of training examples to the network until the stopping criterion is met.

## 4. SYSTEM MODEL

### 4.1 Plant Model using Feed-Forward Neural Network

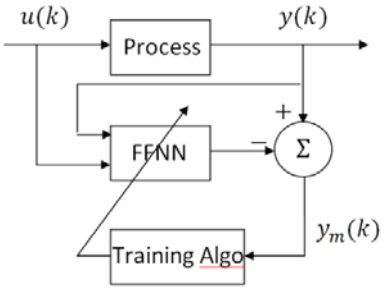


**Figure 5.** Training of the Feed Forward Neural Network

We consider single input single output systems which are described by the following discrete time equation:

$$y(k) = [y(k-1)\ \dots\ y(k-n)\ u(k-1)\ \dots\ u(k-m)] \tag{20}$$

where $y(k)$ and $u(k)$ represent, respectively, the output and the input of the system, $n$ and $m$ are the orders of $y(k)$ and $u(k)$ respectively. Using available inputs and outputs a feedforward Neural Network (FFNN) can be trained to approximate it. The structure of the FFNN, considered in this work, is shown in Figure 5. The FFNN is formed by one hidden layer with $n_1$ neurons. The neural model output is given by the following relations

$$y_m(k) = f\left(b_2 + \sum_{j=1}^{n_l}\left(w_{2j}.\varphi_j(k)\right)\right) \tag{21}$$

$$\varphi_j(k) = f(s_j(k)) \tag{22}$$

$$s_j(k) = b_1 + \sum_{i=1}^{n_1}(w_{1i}.x_i(k)) \tag{23}$$

where $x_i(k)$ is the $ith$ input to the FFNN, $s_j(k)$ is the sum of inputs to the $jth$ neuron, $\varphi_j(k)$ is the output of the $jth$ neuron, and $y_m(k)$ is the estimated networks output. Here $f(.)$ is a derivable and continues function e.g. the log sigmoid function, $w_{1i}$ and $b_1$, are respectively, the hidden layer's weights and biases. $w_{2j}$ and $b_2$ are, respectively, the output layers weights and bias. The input vector is given by:

$$x(k) = [y(k-1)\ \dots\ y(k-n)\ u(k-1)\ \dots u(k-m)]^T \tag{24}$$

The parameters of the (FFNN) Model are estimated by using the back propagation algorithm. The criterion to be minimised is given by:

$$J_1 = \frac{1}{2}\sum_{k=1}^{N_2}(y(k) - y_m(k))^2 \qquad (25)$$

where $N_2$ is the number of input output data.

### 4.2 Inverse Plant Model using Neural Network

The Inverse Neural Network Model (INNM) is determined as described in Figure 6. The criterion to be minimized is given by:

$$J_2 = \frac{1}{2}\sum_{k=1}^{N_3}(r(k+1) - y_m(k+1))^2 \qquad (26)$$

where $N_3$ is the number of set point and output data.

The FFNN for the INN is the same as that used for the plant model. The performance index $J_2$ is minimized until the control sequence which leads to the minimum of $J_2$ is found.

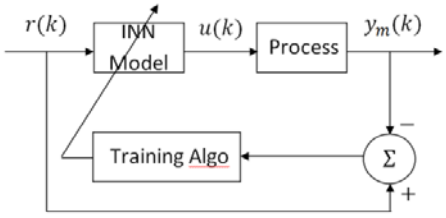


**Figure 6.** Training of the Inverse Neural Network

## 5. SIMULATION

In this paper, linear SISO plants, with both minimum phase and non-minimum phase, and Non-linear transfer functions are used. To find direct and inverse model of the plant, we have used the feed forward neural network with back propagation algorithm. The neural network used has two layers, five neurons in the first layer while one at the output.

**Example 1: Linear SISO Minimum Phase Plant**

Let us consider the following linear SISO minimum phase plant:

$$H(z) = \frac{z^{-1}(0.1065 + 0.0902z^{-1})}{(1 - z^{-1} + 0.25z^{-2})} \qquad (20)$$

Plant disturbance used is having normal distribution with variance 0.01 and a square wave is used as command input. Figure 7 shows that the plant output quickly converges to the desired output. The error between the plant output and the desired output i.e. $e(t) = y(t) - r(t)$ is shown in Figure 8 which is significantly small. Control input to the plant is bounded and is depicted in Figure 9.

**Example 2: Linear SISO Non-minimum Phase Plant**

Following linear SISO non-minimum phase plant is used in second example:

$$H(z) = \frac{z^{-1}(1 + 2z^{-1})}{(1 - z^{-1} + 0.25z^{-2})} \qquad (21)$$

Plant disturbance is kept same as in the previous example. Convergence of plant output is shown in Figure 10 which is quite good. The error between the plant output and the desired output, i.e. $e(t) = y(t) - r(t)$ is shown in Figure 11 which is little bit larger than the minimum phase example. The control input is shown in Figure 12. It can be observed that the control input is bounded even in case of non-minimum phase plant.

**Example 3: Non-linear SISO Plant**

Consider the following SISO non-linear plant given by the following input output relation:

$$y(n) = x(n) + e^{-abs(x(n))} \qquad (22)$$

Plant disturbance is kept same as in the previous examples. Convergence of plant output is shown in Figure 13 where it can be seen that the plant output quickly follows the desired output. The error between the plant output and the desired output is shown in Figure 14. The control input is bounded and is shown in Figure 15.

## 6. CONCLUSION

In this paper, Internal model control using Feed-forward neural network was implemented. Two FFNN are used, one as a plant model and other as the controller. The error in output of plant model is very small which shows the advantage of the used scheme. The high frequency components in the inverse model have been removed by using a low pass filter. The controlled output after using IMC approximately follows the input as is evident from the simulation results.


## ACKNOWLEDGEMENTS

The authors wish to acknowledge the support provided by King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

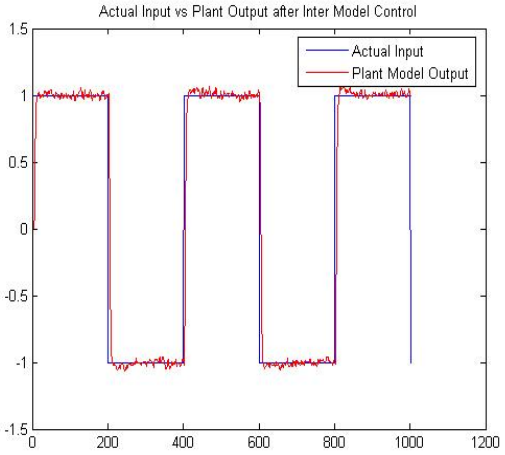


**Figure 7.** Desired Output vs Plant Output for Linear Minimum Phase Plant

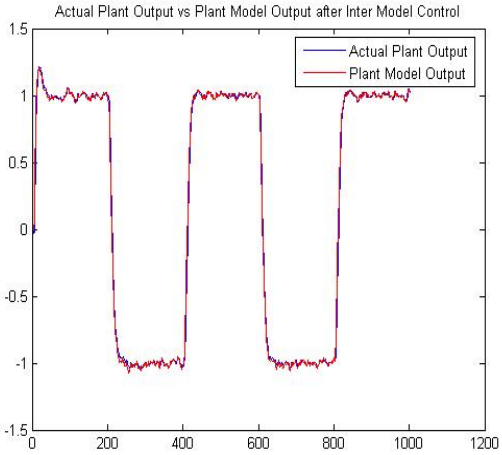


**Figure 10.** Desired Output vs Plant Output for Linear Non-minimum Phase Plant

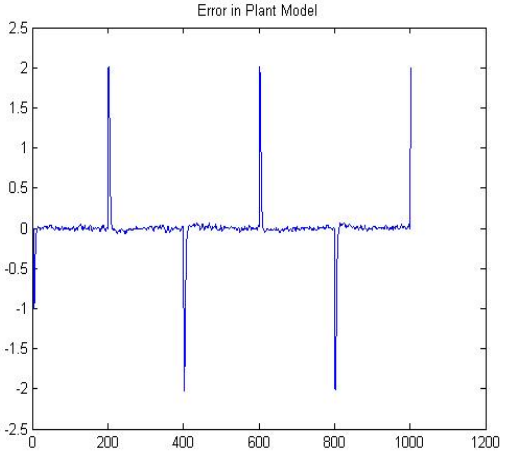


**Figure 8.** Error between Plant Output and Desired Output for Linear Minimum Phase Plant

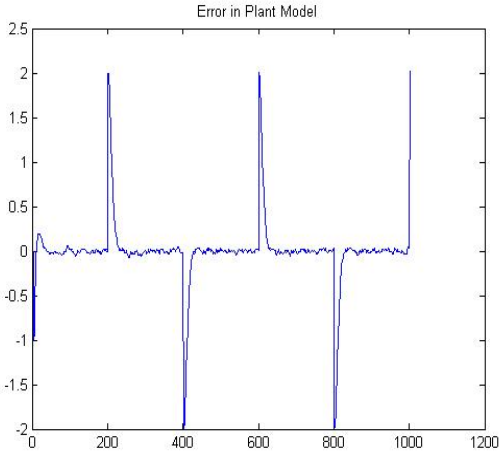


**Figure 11.** Error between Plant Output and Desired Output for Linear Non-minimum Phase Plant

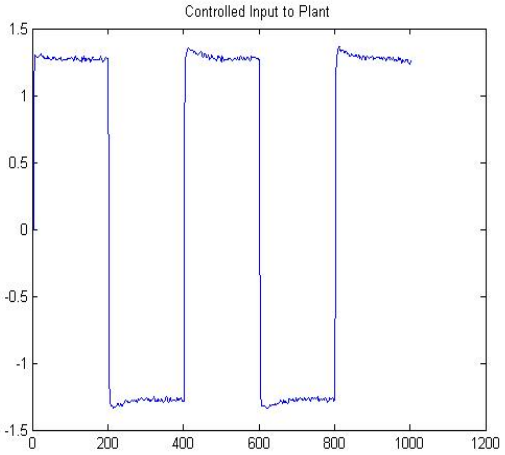


**Figure 9.** Control Input to Linear Minimum Phase Plant

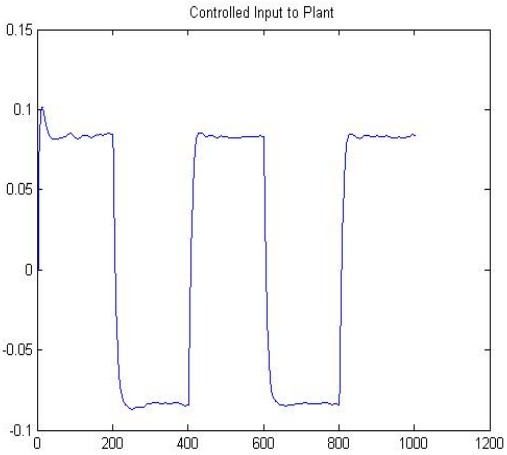


**Figure 12.** Control Input to Linear Non-minimum Phase Plant

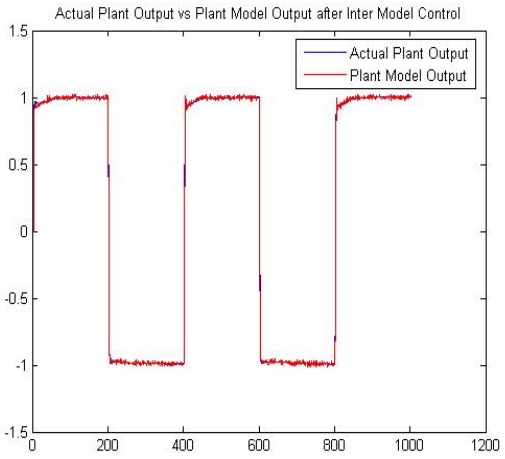


**Figure 13.** Desired Output vs Plant Output for Non-linear Plant

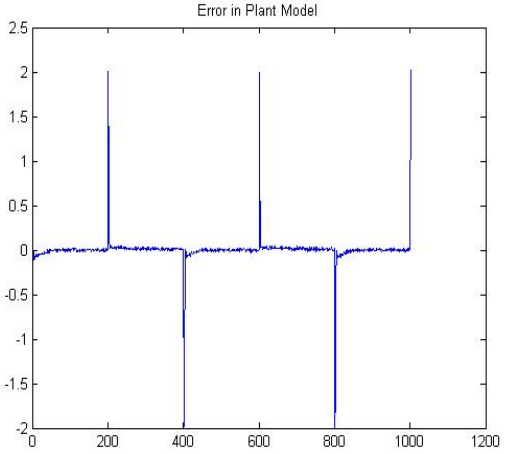


**Figure 14.** Error between Plant Output and Desired Output for Non-linear Plant

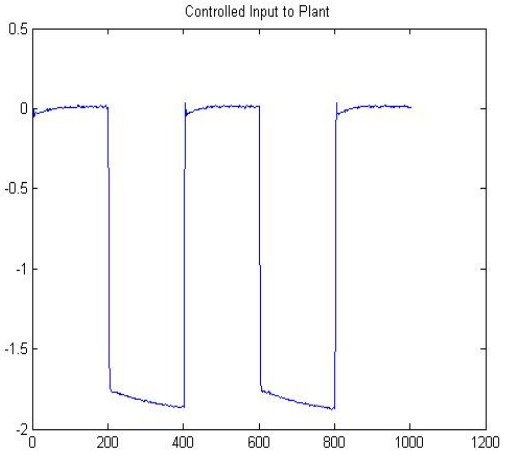


**Figure 15.** Control Input to Non-linear Plant

## REFERENCES

[1] Bouani Faouzi, Chatti Abderrazak, Gallah Tarek, **"**Internal Model Control using Neural networks", *IEEE International Conference on Industrial Technology (ICIT)*, 2004, Vol. 1 Page 1121-1126.

[2] Simon Haykin, "Neural Networks: A comprehensive foundation", Second Ed., Prentice Hall International, Inc. 1999.

[3] Fukuda T. and T. Shibata, "Theory and applications of neural networks for industrial control systems", *IEEE Trans. On Industrial Electronics, 1993.*

# Appendix

## Matlab Code

```
%%%%%%%%Estimating Plant Model using NN %%%%%%%%%%%%


clear all;

close all;

clc;


%%%%% Linear Plant %%%%%%%%%%%%%%%%%%


%%%%%H(z)=(z^-1*(0.1065+0.0902*z^-1))/(1-z^-1+0.25*z^-2)


yp(1:10)=0;  %Making initial value zero because yp(n-1) and yp(n-2) are required first.

y3(1:10)=0;

e1(1:10)=0;

uc(1:10)=0;


N1=5;

N2=5;

%alpha=.01;     %Minimum Phase plant and Non-Linear Plant

alpha=.005;    %Non- minimum phase plant

%epoch=100;


p1=1000;

p2=10000;
```

```
w1=randn(N1,4)/sqrt(N1*p1);

w11=randn(1,N1)/sqrt(N1*p1);

w2=randn(N2,4)/sqrt(N2*p2);

w22=randn(1,N2)/sqrt(N1*p2);



b1=zeros(N1,1);

b11=zeros(1,1);

b2=zeros(N2,1);

b22=zeros(1,1);

%plant=[-1 .25 .1065 .0902];       % Minimum Phase Plant

plant=[-1 .25 1 2];               % Non-minimum phase plant

%plant=[1 1 -1 0.5];        % Non linear plant



t = 0:0.01:5;

lent = length(t);

inp = square(pi*t);               % Input

d = randn(1,lent)*.01;            % Random Noise

iterations= 10;



for k=1:iterations

   waitbar(k/iterations)
```

```
for i=5:(lent)    %For Linear Plant

%%%%%Passing through INN %%%%%%%%%%%%%%%

uc(i)=purelin(w22*tansig(w2*[(inp(i-1)-e1(i-1)) (inp(i-2)-e1(i-2)) (inp(i-3)-e1(i-3)) (inp(i-4)-e1(i-4))]'+b2)+b22);
%uc(i)=y3(i-1);

%%%%%%%%%Passing through Plant %%%%%%%%%%

ph2=[-yp(i-1) -yp(i-2) uc(i-1) uc(i-2)];     %Linear Plant
%ph2=[uc(i) 1 abs(uc(i)) abs(uc(i))^2];     %Non Linear Plant

yp(i)=ph2*plant'+d(i);  %%%%Output of Plant with added noise

%%%%Plant Model using NN%%%%%%%%%%

n1=w1*ph2'+b1;
a1=tansig(n1);
n11=w11*a1+b11;
ym(i)=purelin(n11);

er(i)=yp(i)-inp(i);

e1(i) = yp(i)-ym(i);

Y11=-2*dpurelin(n11,ym(i))*e1(i);
Y1=diag(dtansig(n1,a1),0)*w11'*Y11;
```

```
w11=w11-alpha*Y11*a1';

w1=w1-alpha*Y1*ph2;


b11=b11-alpha*Y11;

b1=b1-alpha*Y1;


%%%%%%%Passing through FNN%%%%%%%%%

%y2(i)=purelin(w11*tansig(w1*[-y2(i-1) -y2(i-2) inp(i-1) inp(i-2)]'+b1)+b11);


Err(i)=inp(i)-e1(i);


%%%%% Inverse NN Model %%%%%%%

n2=w2*[Err(i-1) Err(i-2) Err(i-3) Err(i-4)]'+b1;

a2=tansig(n2);

n22=w22*a2+b22;

y3(i)=purelin(n22);

e2(i) = inp(i)-ym(i);


Y22=-2*dpurelin(n22,y3(i))*e2(i);

Y2=diag(dtansig(n2,a2),0)*w22'*Y22;


w22=w22-alpha*Y22*a2';

w2=w2-alpha*Y2*[Err(i-1) Err(i-2) Err(i-3) Err(i-4)];


b22=b22-alpha*Y22;
```

```
    b2=b2-alpha*Y2;

   %%%%%%%%%Passing through filter%%%%%%%%%%

    %[num,den]=butter(11,0.87);      %For Minimum Phase Plant
    [num,den]=butter(2,0.25);
    Err(i)=filter(num,den,Err(i));

    end

end

figure(1)
plot(ym);
hold on;
plot(yp,'r');
title('Actual Plant Output vs Plant Model Output after Inter Model Control')
legend('Actual Plant Output','Plant Model Output')


% figure
% plot(e1)
figure(2)
plot(inp);
hold on;
plot(yp,'r');
title('Actual Input vs Plant Output after Inter Model Control')
```

```
legend('Actual Input','Plant Model Output')


figure(3)

plot(er);

title('Error in Plant Model')


figure(4)

plot(uc);

title('Controlled Input to Plant')


figure(5)

plot(e2)

title('Error in Inverse Model')
```