NEURAL NETWORK MODEL PREDICTIVE CONTROL OF A ULTRA HIGH TEMPERATURE MILK TREATMENT PLANT

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Technology

at Massey University

VENKATESWARA RAO KARLA

1997

ABSTRACT

This thesis reports the development of a Model Predictive Control system for a Ultra High Temperature milk treatment pilot plant. This control system utilises an Artificial Neural Network model of the plant dynamics. The entire process was divided into two parts for modelling purposes. Separate models were trained; one for simulating the dynamics of the hot water heating loop and the second the dynamics of the heat exchanger circuit. The two sub-models, when concatenated, form a complete model of the plant referred to as a composite neural network model. The results of training and testing of the sub-models with various sets of plant data were presented.

Of all the possible combination of sub-models, the best trained and tested sub-models were concatenated to form the best composite network model, and the combination of worst sub-models to form worst composite network model. Two model predictive control (MPC) systems for the process were developed, one using the best composite network model for prediction purposes and to act as the plant, and the other using the worst composite model for prediction and best composite model as the plant.

Both the developed MPC systems were evaluated in terms of setpoint tracking and disturbance rejection. As a part of these performance tests, a PI (Proportional-Integral) control system of the UHT plant was developed in a simulated environment using the best composite neural network model to act as the plant. The responses of both the MPC control systems were studied and compared with the responses of the PI control system.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

It is a great pleasure for me in taking this opportunity to express my sincere appreciation to my supervisors, namely, Dr. H. H. C. Bakker and Dr. C. Marsh for their valuable guidance and support and Dr. I. H. Noell, formerly with Massey University. I would like to express my heartful gratitude to Dr Huub Bakker, Dr Clive Marsh and Prof. R. M. Hodgson for their help and efforts in applying for Massey University Postgraduate Scholarship for my research work.

I would also like to appreciate Massey University scholarships committee for awarding the Massey University Postgraduate Scholarship which enabled me to undertake this research with high spirit and moral confidence for this recognition.

I would like to acknowledge the support of the New Zealand Dairy Research Institute, Palmerston North for allowing me to carry this research work using their equipment and gratefully appreciate all the staff of the Department of Production Technology at Massey University.

My sincere thanks to Prof. D. J. Barnes for his efforts in involving me for a Technology for Business Growth (TBG) project which helped me with financial stability and in acquiring diversified knowledge.

Now it's time for me to express my special thanks to my chief supervisor, Dr. Huub Bakker for his help and support not only in doing the research work but also in presenting it to the people in the field in the form of a journal paper (Australian Journal for Intelligent Information Processing Systems). I am and will be grateful to him during my life time for making me to utilise my stay at Massey University and to achieve a career growth and goal in my life. My self and my family will remember the pleasant and prosperous friendship of Dr Huub Bakker and his family.

Finally, I would like to dedicate this thesis to Dr Huub Bakker and my wife, Jyothi. I wish to thank Jyothi and my family for their support and understanding, and also all my colleagues and friends at Massey for making a fruitful and memorable stay at Massey University.

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

This chapter outlines the scope of the research work reported in this thesis. It briefly describes the usage of artificial neural networks in a model based control methodology which offers a great potential to enhance the control performance in complex multi-variable systems compared to the conventional Proportional-Integral-Derivative (PID) controllers.

Ultra high temperature (UHT) milk treatment plants are commonly used in the dairy industry for sterilisation of milk which demand a highly accurate and precise control to keep up the higher product quality. The inability of the conventional controllers being used by the existing UHT plants in giving better performance and also the *fouling* problem in the heat exchangers have been the motivation for this research work. This research work presents a specific application of neural networks in a model predictive control scheme for an UHT plant.

The section 1.1 of this chapter presents the scope of this research and sections 1.2 and 1.3 briefly outlines the development of neural network UHT plant model and its usage in the model predictive control system development. The merits and demerits of the neural network model based control system design over the classical controller designs are also presented. The section 1.4 of this chapter provides an outline of all the subsequent chapters presented in this chapter.

1.1 SCOPE OF THIS THESIS

The design of control systems require both steady state and dynamic information about the process. By providing a model of the actual system using the input-output process data to be used in the design and simulation of the system, system representation, modelling and identification proved to be the most vital in process engineering.

The insufficient control performance of the conventional PID controllers in terms of accuracy and preciseness in complex non-linear systems forced the control engineering field to come up with highly accurate and intelligent controllers. This requirement lead the field to make use of computing for developing intelligent systems.

In the recent times, the use of artificial intelligence techniques for tackling many real world problems has increased significantly. These artificial intelligence techniques include neural networks, expert systems and fuzzy logic. The application of artificial neural networks in identifying highly non-linear dynamic system models and designing control strategies based on these models has been a promising advance in the control systems engineering field.

The UHT process is used for the sterilisation of milk so as to kill the bacteria by means of heating the milk to a high temperature and holding it at that temperature for a certain time and then cooling it. This sort of milk treatment process helps to keep the milk stored for longer time periods at room temperature compared to the pasteurised milk which can not last as long and needs to be kept refrigerated.

This work is based on a specific UHT plant at the New Zealand Dairy Research Institute, in Palmerston North. In this UHT process, the existing conventional control system was unable to cope with the requirement of accurate control for steady high temperature in the process for a short period of time. Apart from this, the *fouling* problem in the heat exchanger also lead to this research work for developing an intelligent and smart control system for UHT plant. The term *fouling* relates to a condition where the walls of the heat exchanger in a UHT plant became coated with a thick layer of milk fat globules over a period.

As UHT systems in general are expected to contain significant non-linearities due to the complex properties of milk, a model based control methodology seemed to be the best form of intelligent control and thus the research was carried out in developing an neural network based model predictive control system for the UHT process. The next sections briefly describes the system development and performance evaluation.

1.2 NEURAL NETWORK PLANT MODELLING

Basically, artificial neural networks were developed after an in-depth study of the human brain which has a parallel architecture of neurons. These artificial neural networks copy many of the biological features of the human brain which include their parallel processing capability.

For any process model design and development using artificial neural networks, first the process inputs to be manipulated and outputs which need to be measured and controlled have to be determined first. After designing the network with the desired number of past and present inputs and past outputs, and neural network topology the network has to be trained. The training has to be carried out using the past plant input and output data so that the neural network will learn the process.

The trained network model has to be tested for its ability to predict the output data using the input data which the model hasn't seen before. The performance of the model will be evaluated based on the error in predicting the unknown data.

After studying the UHT plant, it was decided to take the entire UHT process together for designing a single process model for neural network training. The data from various UHT plant runs was collected for training the network and in developing neural network models using different process runs. Developing a plant model which can cope with all the possible input and output constraints and can handle variations in the process lead to the idea of designing and developing the neural network models with different process runs.

1.3 MODEL BASED CONTROL USING NEURAL NETWORKS

The model based control strategy uses a dynamic model of the plant to provide efficient control of the system in a real time environment. In the past, this methodology proved to have the ability to cope with the process constraints and in handling non-linear processes. So the usage of the neural network model in developing a model predictive control scheme for the UHT process seemed to be a reasonable option in achieving the desired results which also became one of intelligent control system design.

The use of neural networks in model based control systems design offered a promising ability to enhance the performance of the control system as was proved before in quite a few process industries, especially in petro-chemical industries (Garcia *et al.*, 1989).

The performance of the developed model predictive control system which uses a neural network model of the UHT plant needs to be evaluated so as to prove its ability in comparison with the existing conventional PI control system. For this performance evaluation it was decided to evaluate the performance of the model predictive control system and the PI control system in a simulated environment. The next section presents an outline of the research work including a brief description of all the chapters in this thesis.

1.5 OUTLINE OF THIS THESIS

The six subsequent chapters in this thesis are

Chapter 2 Gives a background on the neural networks, classical PID control, Modern control strategies like model based control. Also compares the benefits of the modern control over classical control. Explains past research in successful development and implementation of model predictive control systems.

Chapter 3 Explains in detail about the single neural network model development of the UHT milk process. The development of two individual neural network sub-models, steam valve and heat exchanger models after dividing the UHT process. This chapter also explains the concatenation of the above two sub-models to form a composite neural network model to be used in the model predictive control system development.

Chapter 4 Explains the development of a model predictive control system for the UHT plant using the developed composite neural network model of the plant.

Chapter 5 Presents the training and testing results of the neural network models and the performance evaluation results of the developed model predictive control system compared to the PI control system. Also presents a comprehensive discussion of the above results.

Chapter 6 Conclusions of this research work and recommendations for any future work in this area.

Chapter 7 References and Bibliography

Chapter 8 Presents the software used in this work and the complete prediction error results of neural network models in the Appendices.

CHAPTER TWO

BACKGROUND

This chapter gives a brief outline of classical control and describes in detail the development of artificial neural networks and modern control strategies which include model based control. This chapter also presents a brief description of UHT plants.

In this chapter, section 2.1 presents a detailed description of the evolution of artificial neural networks and their development. A brief description of the conventional controllers such as PI and PID controllers is presented in section 2.2. Section 2.3 describes the development of modern and intelligent controllers using artificial intelligence, techniques, especially model based control methodologies, and the comparative merits and demerits of modern control over classical control. A general description of the Ultra High Temperature milk treatment plant, its use and advantages are presented in section 2.4 of this chapter.

2.1 NEURAL NETWORKS

This section describes the evolution of artificial neural networks which was considered to be one of the intelligent development in the field of artificial intelligence. Apart from artificial neural networks, the other areas in this field are fuzzy logic and expert systems. This section also describes the various types of neural networks in detail.

2.1.1 Human Brain and Its Structure

In the general human life, the problems encountered are of immensely parallel nature which require the processing of a pool of information to provide a solution. Due to its parallel design, the human brain is able to store the information which can be accessible and processible. The important feature of the brain is its ability to learn on its own. The approach of neural networks is to mimic the functionality of the human brain i.e. to capture the brain's method of solution of problems and apply them to computer systems. With this capability, it is possible to utilise computing systems to solve the most critical problems with greater speed and efficiency.

The structure of the human brain is extremely complex with a highly interconnected network of approximately ten thousand million basic processing units, called neurons. Each one of these neurons is connected to about ten thousand others. The neuron acts as a stand alone analogue logical processing unit. The neurons form two types of cells, one is soma and the other axon.

The body of the neuron is called soma. Long and irregularly shaped filaments attached to the soma are called as dendrites which are often less than a micron in diameter with complex shapes. All inputs to the neuron arrive through dendrites which act as the connections.

The axon, which is attached to the soma acts as an output connection to the neuron. It is electrically active and a non-linear threshold device. This will produce an action potential which lasts for about 1 millisecond at times when the internal potential within the soma rises above a certain critical threshold. The synapse connects the axon with the dendrite of another cell which is a temporary chemical contact. A single neuron will have many synaptic input and output connections.

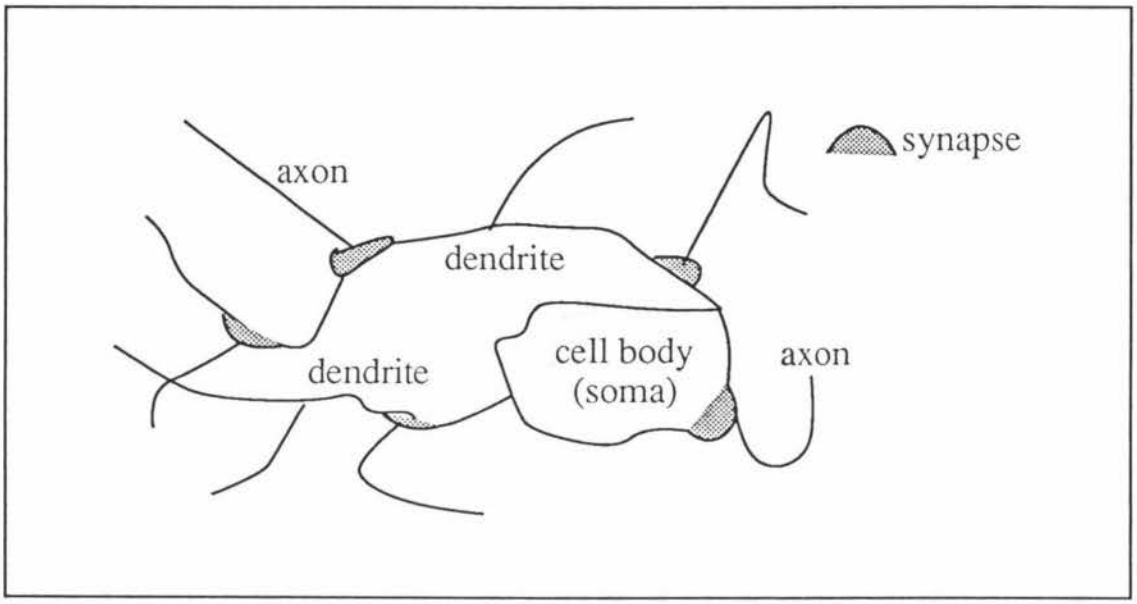

Fig 2.1.1 Biological neuron features

2.1.2 Single Neuron Modelling

The idea of neural computing is to build a model that can capture the important features of human brain so that it can exhibit a similar behaviour. The function of the biological neuron is to add all its inputs and produce an output if it is greater than a certain value called threshold. The neuron inputs come through the dendrites and connected by synapses to other neuron outputs as shown in figure 2.1.2. These synapses alter the effectiveness of the signal transmission from the neuron. Some of the synapses act as good junctions which allow

large signal while others allow small signal. The neuron receives all these inputs and emits output if the total input exceeds the threshold value.

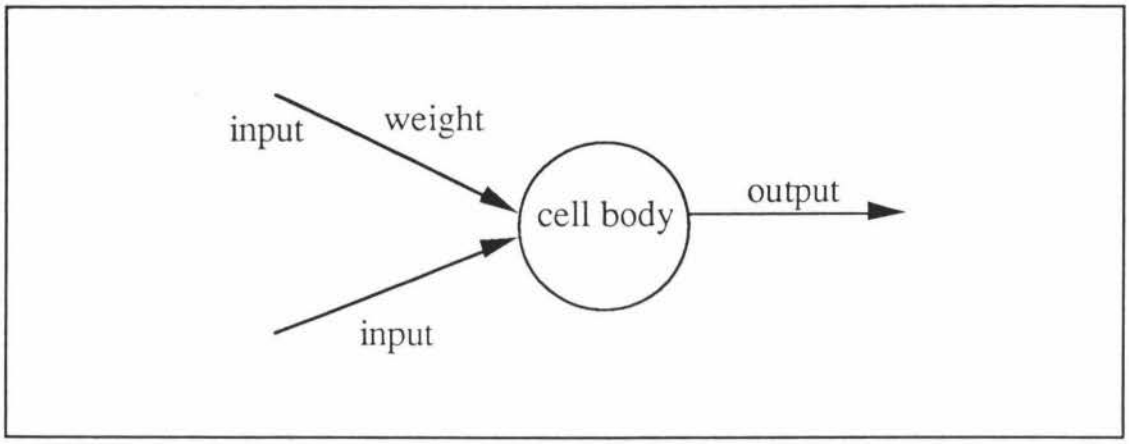

Fig 2.1.2 Outline of the basic neuron model

The neuron model needs to capture the following important features.

- The output from a neuron is either on or off
- The output depends only on the inputs, so a certain number of inputs must be on at any time in order to make the neuron fire.

The efficiency of the synapses for transmission of signal into the cell can be altered by having a multiplicative factor on each of the inputs to the neuron as shown in the above figure 2.1.2. These multiplicative factors are referred to as weights. The next section describes the type of network used in the design of artificial neural network to be used in the control system applications.

2.1.3 The Sigmoidal Feedforward Network

The basic feedforward network performs a non-linear transformation of input data in order to approximate output data. The nature of the modelling problems, the manner of input data representation and required network output form determines the number of input and output nodes.

The most widely used network architecture is the multi-layer, feed forward network as shown in figure 2.1.3.1.

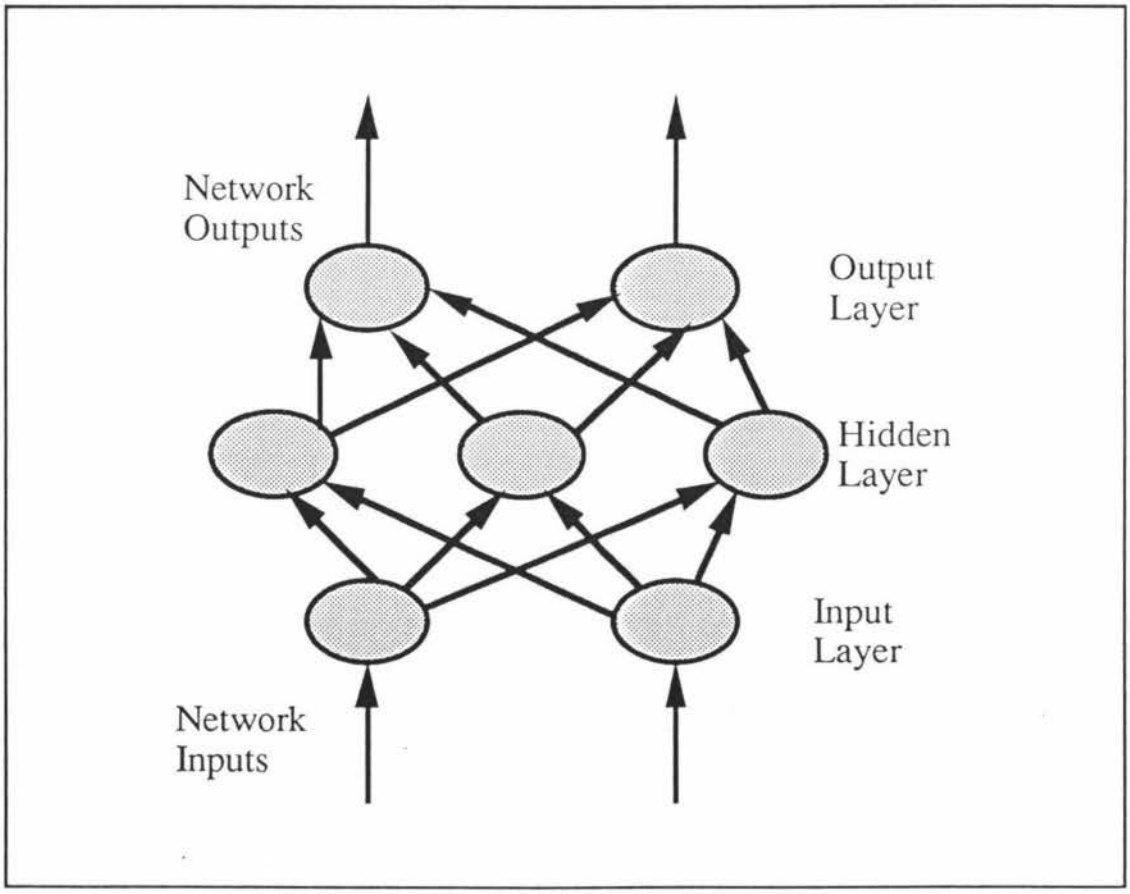

Fig 2.1.3.1 Feedforward network structure

As shown in the figure 2.1.3.1, the feedforward type of network consists of an input layer, an output layer with one or more hidden layers in between. The number of neurons in the input and output layers

depends on the respective number of inputs and outputs being considered. The number of hidden layers and the number of neurons in each layer may vary from zero to any finite number as chosen by the user. These can be changed by trial and error while training the network to minimise a sum-squared-error term.

The neurons in the input layer provide a means to introduce scaled data into the network. These signals are then fed forward via connections through the hidden layer and finally to the output layer. Each interconnection has associated with it a weight which modifies the strength of the signal flowing along that path. Therefore, the input to each neuron is a weighed sum of the outputs from neurons in the previous layer.

The structure of a single artificial neuron is shown in figure 2.1.3.2. Neurons within a network are interrelated by means of a weighed connection. The function of a neuron is to sum the weighed inputs of incoming signals and then pass this sum through a non-linear function, for example a sigmoid curve.

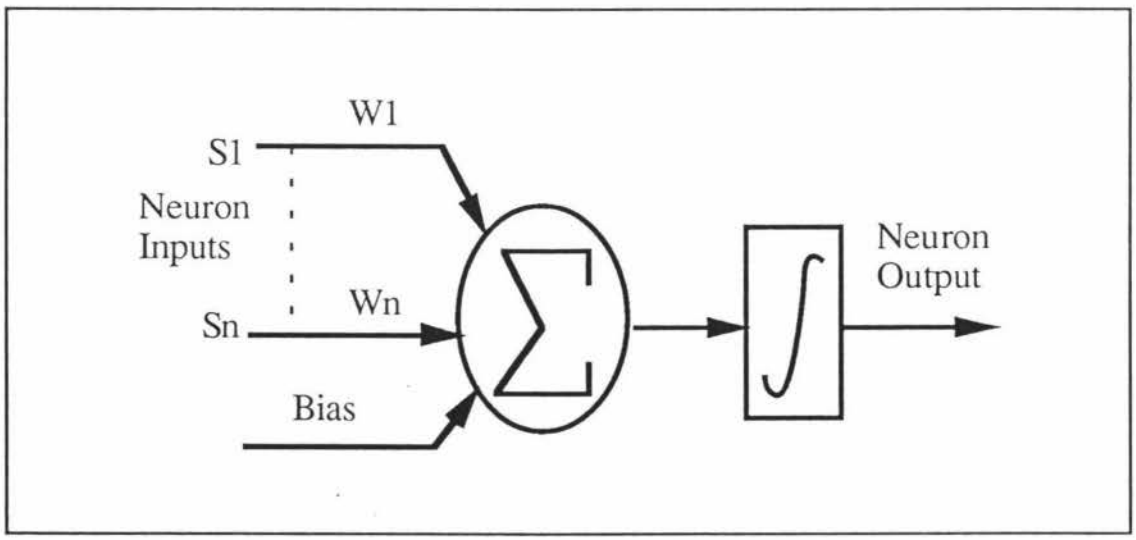

Fig 2.1.3.2 Neuron processing

For example, if the information from the i^{th} neuron in the $j\text{-}1^{th}$ layer, to the k^{th} neuron in the j^{th} layer is $I_{j-1,j}$. Then the total input to the k^{th} neuron in the j^{th} layer is

$$Sj,k = \sum_{i=1}^{nH} (bj,k + wj-1,i,kIj-1,i)$$

Where nH is the number of neurons in the hidden layer and $b_{j,k}$ is a bias term to determine the co-ordinate space of the nonlinearity. The output of each node is obtained by passing the weighed sum, $S_{j,k}$ through a nonlinear sigmoidal function in the interval (0,+1 or -1,+1) which is given by:

$$Oj,k = 1/(1 + \exp[-Sj,k])$$

2.1.4 Feed Forward Network Training

Before the neural network training, the number of hidden layers and the number of neurons in each of the input, output and hidden layer are to be specified. After specifying the network topology, a set of input-output data is used to train the network; i.e. to determine the appropriate values for the weights associated with each interconnection including the bias terms. This determination of network weights can be considered as a non-linear optimisation task. The Jacobian of the objective function is the simplest optimisation technique used to determine the search direction. The distributed gradient descent technique uses a learning rate term to influence the weight adjustment rate and forms a basis for the back propagation algorithm.

At each instance, the data are propagated forward through the network and the difference in the network output and corresponding ôutput in the data set results in the error. This error is used to update the weights, based on the gradient information, in order to the drive cost function to a minimum. This error is backpropagated again through the network until it reaches the specified goal. Thus the algorithm which uses this approach for neural network training is known as backpropagation algorithm. A momentum term is used to avoid

settling into a local minimum during training. This term makes the current change in weight to depend on the previous weight change.

With a good network approximation, the squared error between the training data outputs and network predicted outputs should be relatively small, and uncorrelated with all combinations of past inputs and outputs. The back propagation algorithm used to train the networks was first proposed by Rumelhart, Hinton and Williams, (1986). Details of the neural net training and literature on neural networks are given by Werbos (1974), Rumelhart and McClelland (1987), Hornik *et al.*, (1989) and Chessari (1992).

The next two sections present a brief outline of the different types of artificial neural networks which are also based on the feedforward type structure.

2.1.5 Autoassociative Neural Networks

These are feedforward networks whose inputs and outputs are identical, and consists of a data compression or bottle-neck inner layer apart from the input and output layers. Following convergence, the network bottle-neck provides information on significant features of the data and multivariate statistical techniques can be used for the comparison of this compressed original plant data.

Using the linear statistical methods such as principal component analysis and partial least squares, noisy and highly correlated process measurements can be treated effectively to use the measurements for process modelling to predict future quality variables. More details about the autoassociative networks and linear statistical methods is given by Morris *et al.*, (1994).

2.1.6 Dynamic Neural Networks

The approach for a dynamic mapping of input and output data to describe system dynamics is similar to the linear ARMA (Auto-Regressive Moving Average) modelling. A time series of past process inputs (u) and outputs (y) are used to predict the present process outputs. ARMA series approach is inappropriate if models are trained and used for more than one-step-ahead prediction which does not capture the process dynamics (Morris *et al.*, 1994). The auto regressive nature of this network results in the need to predict y(t+n) from the estimates of y(t+n+1). So the errors accumulate in the estimation of y as prediction horizon increases. This problem can be overcome by minimising network prediction error over time. This approach is called 'back-propagation-in-time'.

Incorporation of dynamics into the network makes the network model free from prediction problems unlike autoregressive models. This description of the above two network types does not actually explain the working aspects of them in detail.

2.2 CLASSICAL CONTROL METHODOLOGY

This section presents the details of classical control systems, their types and the working aspects of them.

For many control applications, the control strategy will be either an open loop or closed loop control structure. Basically in the open loop control, the output of the system is neither measured nor fed back for comparison with the system input i.e. the output has no effect on the input control action. This type of control can not perform the desired task in case of internal or external disturbances.

In the closed loop control, the deviation of the actual system output termed as 'controlled variable' from the desired value of the output termed as 'setpoint' will be fed back to the system. This deviation can be reduced by changing the input to the system termed as 'manipulated variable' which makes the whole system a closed loop.

A system which maintains a prescribed relationship between the output and some reference input by comparing them and using the difference as a means of control is called a feed back control system which is a closed loop control. An advantage of the closed loop control is the relative insensitiveness of the system response for external disturbances and internal deviations in system parameters.

For any successfully optimised and efficient control system, the performance evaluation of the system is the most important aspect. Of the performance indicators, the important ones are the system response features such as rise time, maximum overshoot and settling time.

The classical control systems consist of systems designed with Proportional (P), Proportional-Integral (PI), Proportional-Derivative (PD) and Proportional-Integral-Derivative (PID) controllers. These names of the controllers correspond to the type of control action they perform on the manipulated variable of the system (input variable) in order to bring the controlled variable (system output) close to the desired setpoint. All the above controllers will have a feed back of the

deviation signal which is the difference between the actual output and the setpoint. The following figure shows the working aspects of a feed back control system and the control action block shown in the figure can be any of the above controllers for a classical control system design.

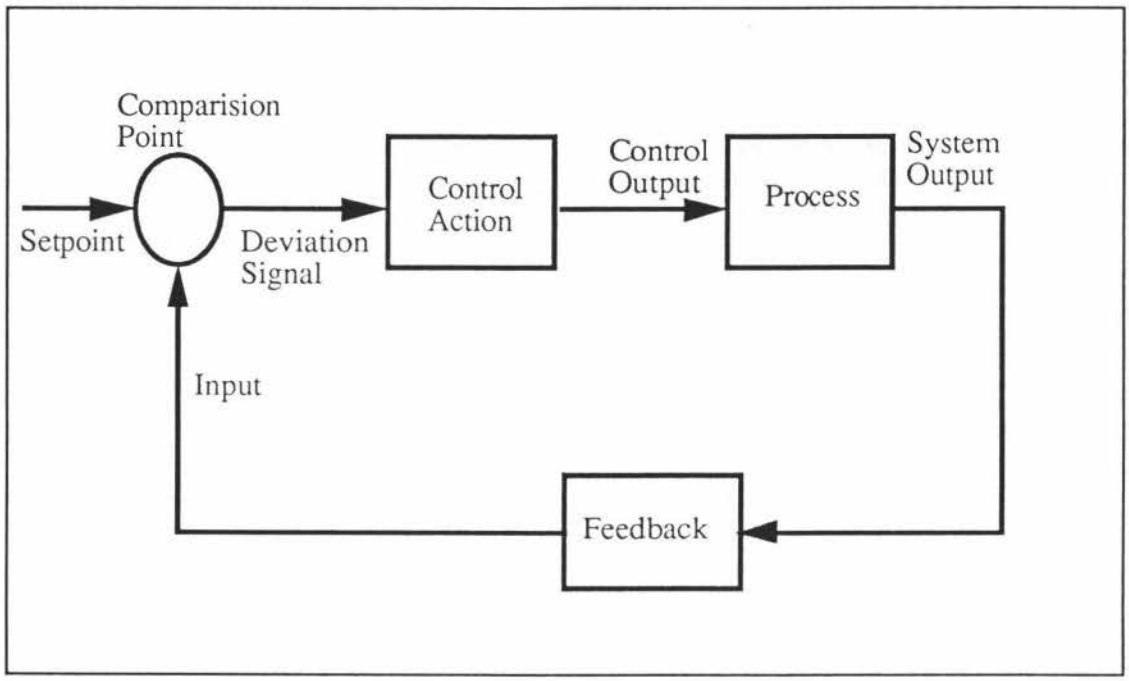

Fig 2.2 Feedback control system schematic

The proportional control action produces an output signal that is proportional to the deviation signal. This action responds to the sign and size of the deviation. For each value of the deviation, the controller responds with a specific output correlated to the manipulated output of the system. The advantage of proportional controller is their ease of design and implementation and the disadvantage is the possibility of sustained deviation of system output called as 'offset'. The offset can be reduced by means of narrowing the proportional band but this may result in an unstable operation.

The addition of integral control action to the proportional action will respond to the continued existence of the offset. Unlike the proportional action, the integral action is completely dependent upon time. By this combined action, the controller output will be changed at a steady rate

for each value of the system output deviation. The actual response of the integral action depends on the characteristics of the proportional control action and the setting of the controller's integral rate adjustment.

According to the past research (Ogata, 1992), the combined PI action proved to have eliminated the offset. The disadvantage of the integral action is the possibility of an overshoot of the system response for systems with a large deadtime. The deadtime is a period of delay between two related actions- such as the beginning of a change in the controlled variable and the beginning of a related change in the system output.

The addition of derivative action to the PI control action provides the ability to overcome the overshoot in system response. The derivative action responds to change in the speed and direction of the deviation from setpoint. A constant deviation from the setpoint initiates no response from the derivative action. If the deviation starts or stops to change, or alters its rate of change due to a change in process variable or setpoint, then the derivative action provides an early and additional change in controlled input to compensate the effect of process deadtime. The disadvantage in using only derivative with proportional action is it's inability to eliminate sustained offset in system output.

In general, both PI and PID controllers had been widely used for many real world applications in the past and even today majority of the systems use them compared to the modern control technologies.

2.3 MODERN CONTROL METHODOLOGY

This section presents the past research in the application and development of modern control. Due to the increasingly stringent requirements on the performance of control systems, increase in demand for good real-time control in performing complex tasks of the present industry, the need to develop modern control strategies became a dominant challenge for the control engineering field.

Past research (Garcia *et al.*, 1989) proved that the model based methodology was able to deliver high performance for multi-variable and highly complicated systems even with lot of process constraints. Especially in modern control theory, model based control became an important strategic development and it's applications to many complex systems proved its ability to handle unstable processes due to the use of step response models and to handle process constraints explicitly through on-line optimization. The next section presents the application criteria of model based control and some of its important aspects.

2.3.1 Model Based Control

Taken as an example the petro-chemical and chemical industries (Garcia *et al.*, 1989) Model based control schemes proved their ability to satisfy one or more of the performance criteria such as economic feasibility, safety, process equipment and product quality. In the performance criteria, the economic feasibility is associated with either the maintenance of process variables at the targets calculated by means of optimisation or dynamically minimizing an operating cost function.

In general, the operating point of any process or system which satisfies the overall economic goals of its system will lie at the intersection of system constraints (Arkun, 1978; Prett and Gillette, 1979). So in the economic aspect of control system design and development, the system must have the ability to meet the high performance criteria.

Model based control is based on the use of an explicit and separately identifiable model of the process to be controlled in order to make a long range prediction of process outputs for given control inputs. This model prediction capability is then used to select control inputs which drive the process outputs to their desired reference levels i.e. set targets.

Using the principle of model based control, various types of controllers have been developed and are categorized as part of the model based control. The different model based control designs are classified according to the process model representation, criteria for control inputs selection and the establishment of the desired reference levels. Of all the model based control systems, application of model predictive control and internal model control have been most often looked at due to their use in applications associated with the artificial neural networks. The working aspects of model predictive control were explained in the chapter 4 in detail as a part of neural network based UHT plant model predictive control system development.

The internal model control (IMC) strategy is an unconstrained control technique which utilises a dynamic inverse plant model as well as a forward model. This type of model based control design proved to have been successful in the past (Garcia and Morari, 1982). The details of the IMC strategy for linear systems were initially developed by Garcia and Morari (1982) and their work has been extended to non-linear systems (Economou *et al.*, 1986).

The utilisation of two models is one of the important feature of internal model control scheme. The IMC scheme uses one model as plant and the inverse model as controller.

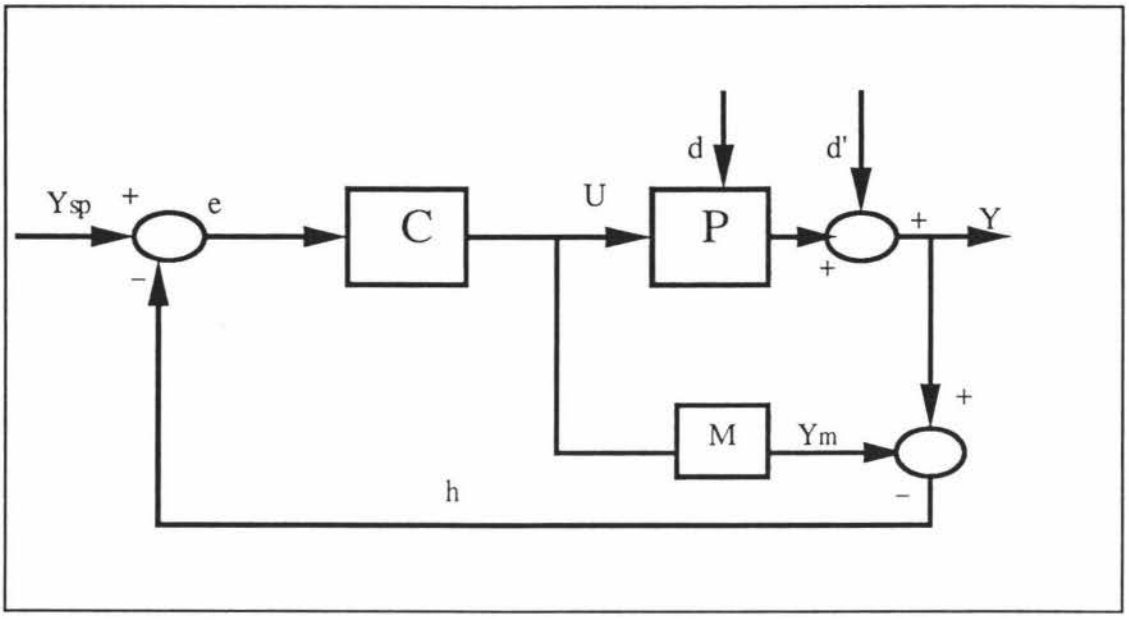

Fig 2.3.1 Internal Model Control System

Figure 2.3.1 shows the general IMC scheme. The operators C, P and M represent the controller, the plant and the plant model respectively. The figure represents the control design for a non-linear system. The disturbances to the system output and plant model are represented by d' and d respectively. According to past research, the IMC scheme is shown to have a number of desirable properties for both linear and non-linear systems (Hunt and Sbarbaro, 1991).

The following relationships can be drawn directly from the figure 2.3:

$$e = ysp - y + ym \qquad (2.1)$$

$$ym = MCe \qquad (2.2)$$

$$h = (Pd - M)Ce + d' \qquad (2.3)$$

Where P_d represents the plant under disturbance.

The most important characteristics of IMC are given below.

Dual Stability:
Assume that the plant and controller are input-output stable and that the model is a perfect representation of the plant. Then the closed-loop system is input-output stable.

Proof: When $M = P_d$ then h (refer to eqn 2.3) is not a function of the plant input and the system is effectively in open loop.

Perfect Control:
Assume that the inverse of the plant model operator exists, that this inverse is used as the controller and that the closed-loop system is input-output stable with this controller. Then the control will be perfect, i.e. $y = y_{sp}$.

Proof: Follows directly from (eqn 2.1) and (eqn 2.2).

Zero Offset:
Assume that the inverse of the steady state model operator exists, that the steady state controller operator is equal to this and that the closed loop system is stable with this controller. Then offset free control is attained for asymptotically constant inputs.

Proof: Follows directly from (eqn 2.1) and (eqn 2.2) by taking the limit as $t \longrightarrow \infty$.

From the above properties, it is clearly evident that obtaining a perfect inverse model is important in designing a best IMC system to achieve perfect control and zero offset. The advantage of the IMC system design is its ability to provide unconstrained nonlinear control without computational difficulties but the hard limitation of IMC design is the attainment of an accurate inverse model.

After all the research studies, the model based methodologies proved to be a valuable achievement in the control engineering area which can cater the highly demanding needs of the present world requirements.

2.4 UHT (ULTRA HIGH TEMPERATURE) MILK TREATMENT PLANT

In general, UHT treatment plants are used the for sterilisation of milk or milk made from the reconstituted milk powder. The sterilisation of milk kills all living micro organisms so that it will keep for many days as opposed to pasteurising which kills mainly pathogenic bacteria and lasts only a few days.

There are a number of different types of UHT plants as shown in figure 2.4.1. They can be categorised according to the heating medium used and equipment design.

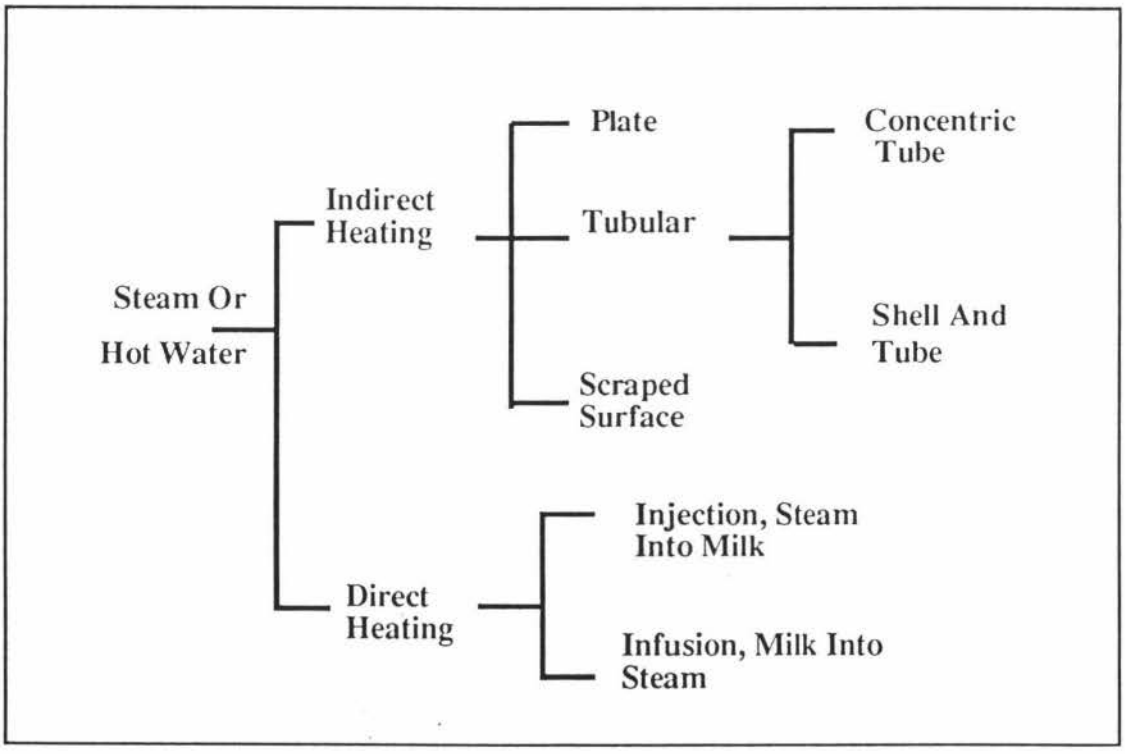

Fig 2.4.1 Types of UHT plant

Directly heated UHT plants use direct mixing of steam and milk to provide sterilisation. In the indirectly heated plants, steam/hot water is used as the heating medium, separated from the product by a heat conducting barrier such as stainless steel.

In the New Zealand Dairy Research Institute (NZDRI) UHT pilot plant, the product is heated to a temperature of 135 to 140°C and held at that temperature for a few seconds before cooling to a suitable temperature for filling. The holding time is approximately 5 seconds (Kessler *et al.*, 1986). The chemical changes in the product after heating may cause the loss of flavour, colour and nutrient constituents. To minimise chemical changes in the UHT process the product is heated to the highest possible temperature for shortest required time.

The NZDRI plant is an indirectly heated plant. In this system the untreated milk is heated to 65°C in a preheater. The product is then pumped into a secondary plate heat exchanger consisting of three stages as shown in the figure 2.4.2 (courtesy: NZDRI, Palmerston North). In the first stage, the product gets heated regeneratively to 85°C by the hot product coming out. In the second and third stages, hot water from a hot water loop heats the milk to the final temperature of 140°C. From here the product passes through the regeneration section and gets cooled by the incoming cold milk before carrying on to the filling section. The hot water loop is heated by the direct injection of steam via a steam control valve. The condensate is removed from the loop and fresh water added.

The existing control system is unable to control the sterilisation process adequately due to control fluctuations. These cause a loss of taste and odour, and cause fouling if they raise the temperature beyond 140°C while fluctuations which lower the temperature below this point will result in incomplete sterilisation which resulted in this research work of developing an alternate control strategy for the UHT process plant.

PILOT plants for UHT treatment

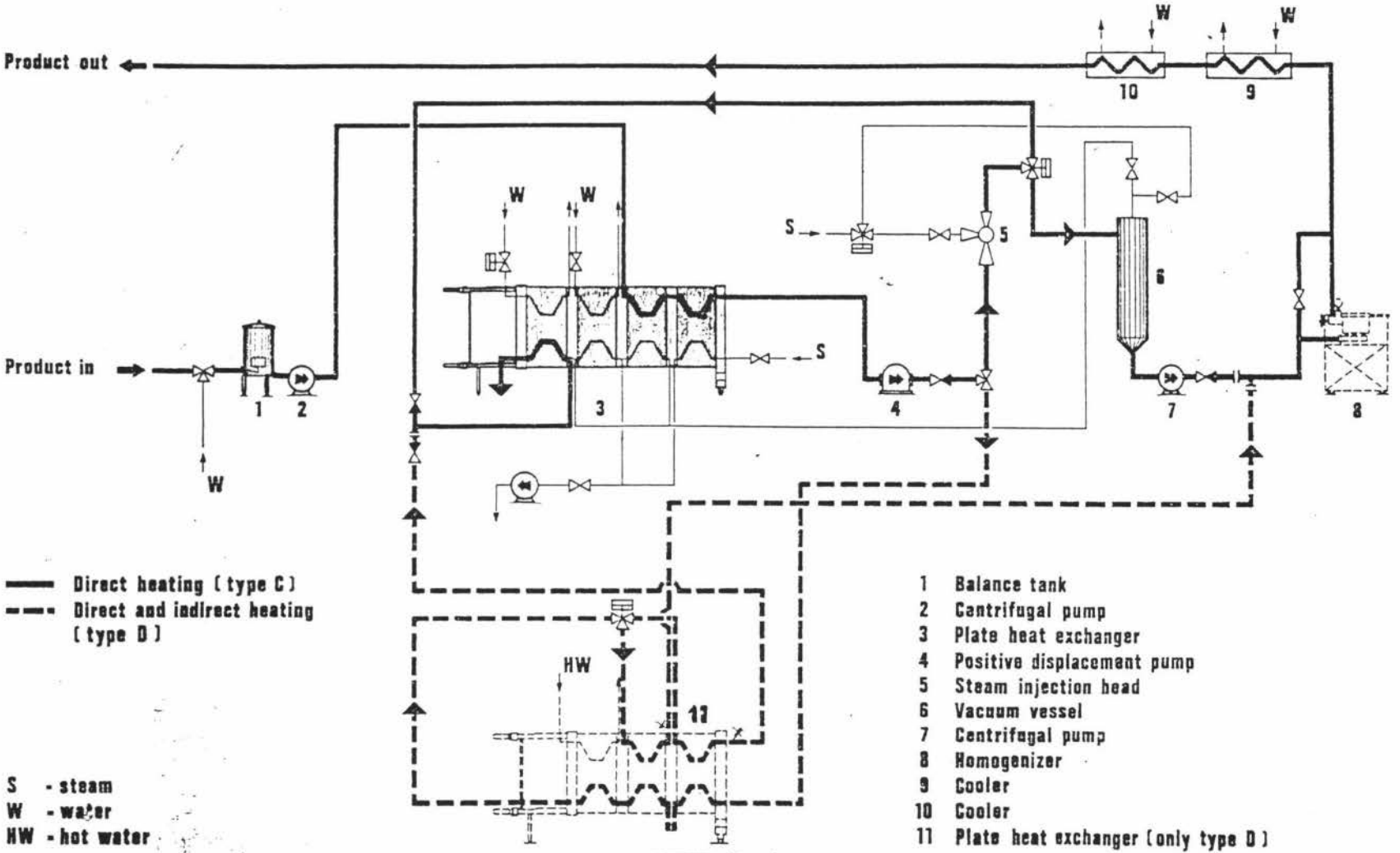

Fig 2.4.2 Schematic of UHT pilot plant

CHAPTER THREE

DEVELOPMENT OF UHT PLANT MODEL

This chapter deals with the development of dynamic neural network models of the UHT process from plant data. Two types of model are discussed; a single neural network model and one with two sub-models namely, the steam valve model and the heat exchanger model.

A broad description is given of the neural network model design and training along with the training criteria. The selection of the input and output variables for training is also discussed. The model predictive control simulation uses these developed neural net models for both the prediction model and the plant model.

3.1 NEURAL NET MODELLING OF UHT PROCESS

The following UHT pilot plant schematic gives an idea of the UHT process variables which can be used for the process modelling.

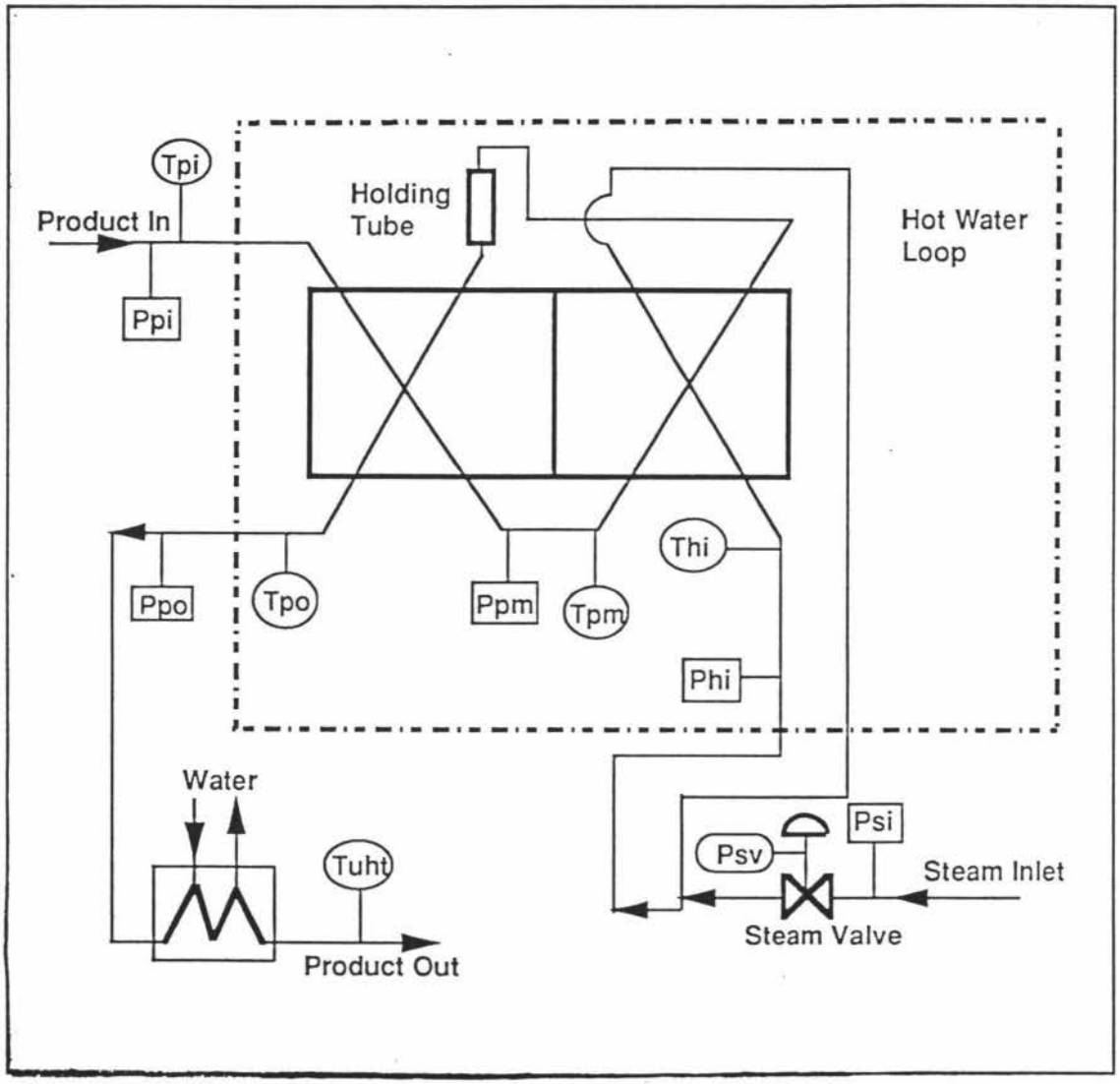

Fig 3.1.1 Schematic of UHT Plant

The simplest approach for modelling a process such as the UHT pilot plant would be to use a single neural network model. If this was done then the input and output variables might be chosen from the following list.

T_{pi}	Temperature of product entering the heat exchanger
T_{pm}	Temperature of product entering the heat exchanger's second stage
P_{pi}	Product pressure entering the heat exchanger
P_{pm}	Product pressure entering the heat exchanger's second stage
P_{si}	Steam supply pressure
P_{sv}	Steam valve opening (in %)
T_{uht}	Temperature of product after cooling

Table 3.1.1 Input variables for modelling

T_{po}	Product temperature after the heat exchanger
T_{hi}	Temperature entering the heat exchanger's hot water loop
P_{po}	Product pressure after the heat exchanger
P_{hi}	Pressure entering the hot water loop

Table 3.1.2 Output variables for modelling

A single network model which included the dynamics of both the hot water loop and the heat exchanger was developed and trained. It is discussed in the following section, 3.2.

With neural network models it is often more useful to break up the modelling task into a number of smaller sub-models. The process was therefore divided into two parts namely, the hot water heating loop and the heat exchanger. Individual neural network sub-models were developed (steam valve and heat exchanger sub-models respectively). They are discussed in sections 3.3 and 3.4.

These two trained sub-models were then combined to form a composite neural network model to represent the UHT process.

3.2 SINGLE NEURAL NETWORK MODEL

This model describes the dynamics of both the hot water loop and the heat exchanger. The boundary of this model is shown as a dotted line in the UHT plant schematic (figure 3.1.1, section 3.1).

The input variables to this model were the past and present values of the steam inlet valve position, P_{sv}, and steam supply pressure, P_{si}, (at times t = k-2, k-1 and k) and the three previous output values of the hot water temperature, T_{hi}, and product output temperature, T_{po}. The model predicted the output product temperature, T_{po}, at time t = k.

The input and output process data of the UHT pilot plant was recorded by a data collection computer every ten seconds. The data was then scaled between 0.1 and 0.9 in order to make it suitable for neural network training. The scaled data was placed into a matrix containing time delayed inputs and previous outputs (refer to section 8.1 of Appendices).

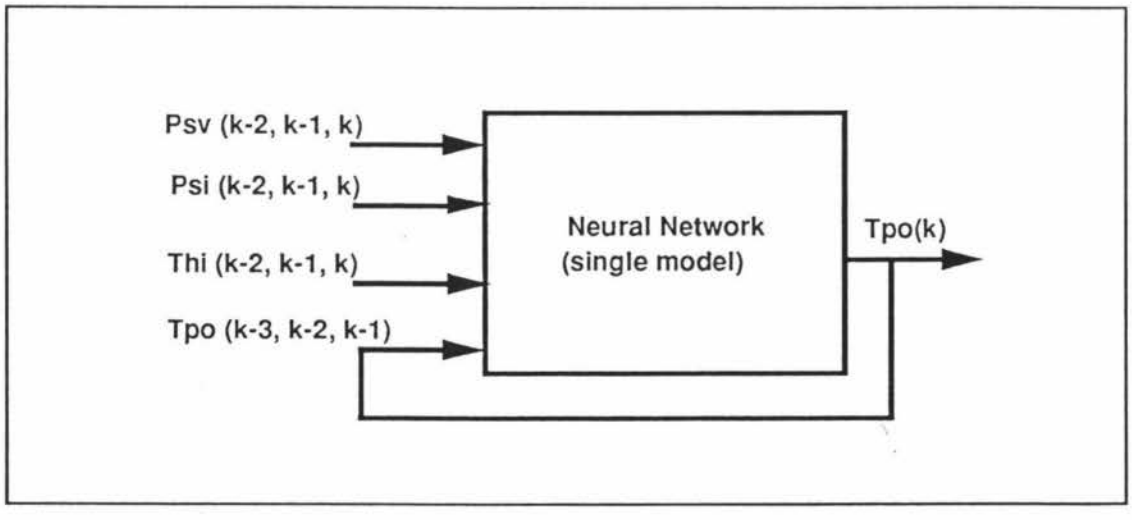

Fig 3.2.1 Single neural network model

Before training the network topology was specified. Twelve input neurons were required for the present and past values of the four input variables. One output neuron was required for the output product temperature. The hidden layer consisted of three tan-sigmoid neurons.

Five versions of the single neural net model were trained using 5 sets of patterned data, each from a different process run. The Matlab Neural Network Toolbox [1] was used to train the neural network models using the backpropagation algorithm (refer to section 8.2 of Appendices).

All the data from a given run was used for training a given version although some points, where the operator intervened, were removed. The training parameters were taken as shown in the following table.

Momentum	0.55
Learning rate	0.1
Learning increment	1.04
Learning decrement	0.8
Error ratio	1.04
Error goal	0.001

Table 3.2.1 Network training parameters

Each network was trained for a total of 9000 epochs in sessions of 3000 epochs with the weights and biases being saved after each session. After training each neural network model on one data set was tested on all the other data sets.

In this way a set of single neural network models were developed for each data set with varied network topology, i.e. by altering the number of neurons in the hidden layer, the number of past and present values of the network inputs and outputs and the type of hidden and output processing neurons as shown below.

	Hidden Neurons	Input & Output	Hidden Processing	Output Processing
First training set	3	3	Tan-Sigmoid	Linear
2nd training set	3	3	Tan-Sigmoid	Log-Sigmoid
3rd training set	2	2	Tan-Sigmoid	Linear
4th training set	4	3	Linear	Tan-Sigmoid
5th training set	5	5	Log-Sigmoid	Tan-Sigmoid
6th training set	7	5	Tan-Sigmoid	Log-Sigmoid

Table 3.2.2 Neural network training topology

In training the neural network models for each data set with varied topology, the network training parameters such as momentum and learning rate were also altered. The momentum was varied from 0.45 to 0.85 and the learning rate from 0.05 to 0.2 to see the affect of these parameters in reducing the network mean-squared-error while training.

3.3 STEAM VALVE SUB-MODEL

This model describes the dynamics of the hot water loop. The boundary of this model is indicated by a dotted line [model 1] in the following figure.

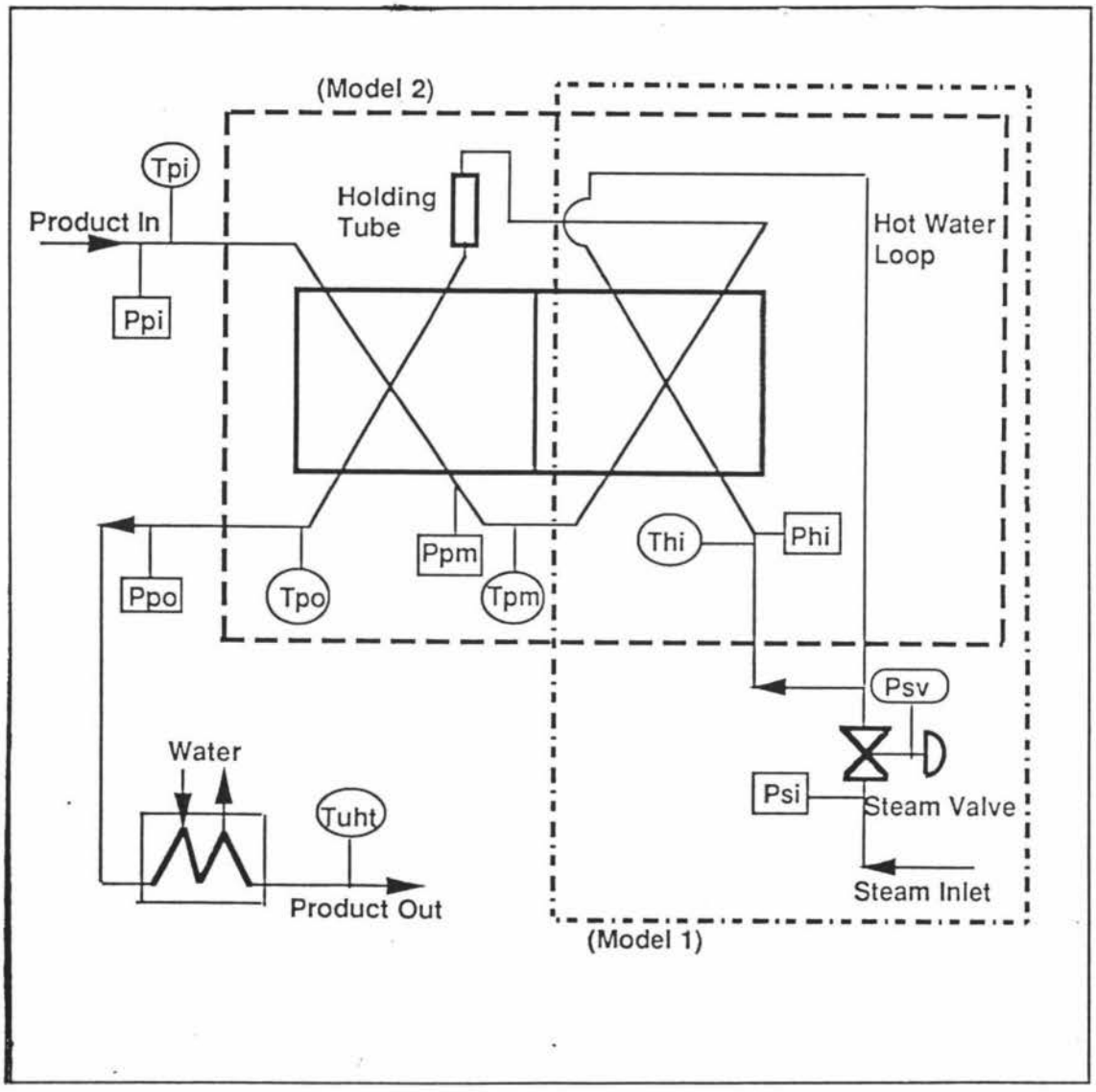

Fig 3.3.1 UHT schematic with model boundaries

The input variables to this model were the past and present values of the steam inlet valve position, P_{sv}, and steam supply pressure, P_{si}, (at times t = k-2, k-1 and k) and the three previous output values of the hot water temperature, T_{hi}. The model predicted the hot water temperature, T_{hi}, at time t = k as shown below.

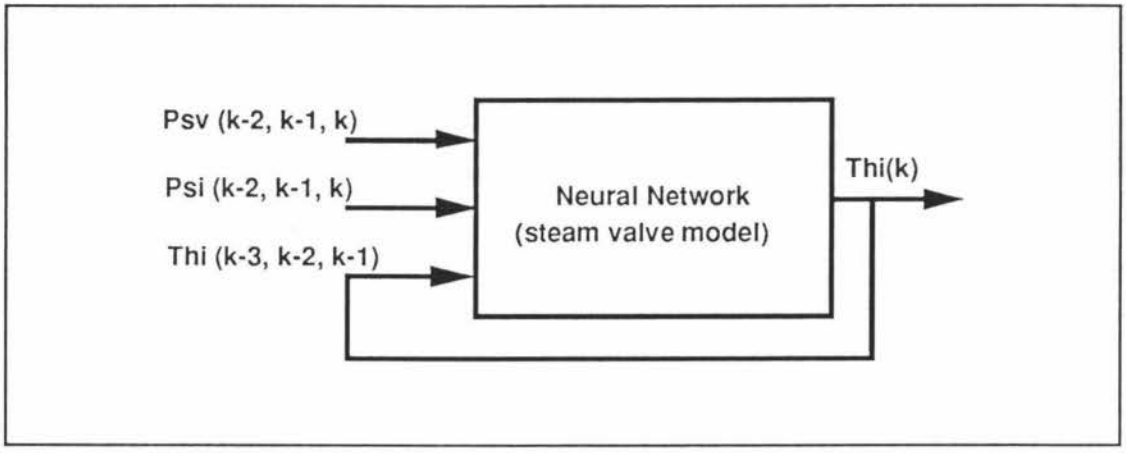

Fig 3.3.2 Neural network steam valve sub-model

Versions of the steam valve sub-model were trained using each of 5 sets of representative data (i.e., patterned data) from 5 different process runs. The Matlab Neural Network Toolbox was used to train the neural network models using the backpropagation algorithm (refer to section 8.2 of Appendices).

The collected process data was scaled between 0.1 and 0.9 in order to make it suitable for neural network training. The scaled data was placed into a matrix containing time delayed inputs and previous outputs (refer to section 8.1 of Appendices).

Before training the network topology was specified. Nine input neurons were required for the present and past values of the three input variables. One output neuron was required for the output hot water temperature. The hidden layer consisted of three tan-sigmoid neurons.

All data from a given run was used for training a given version although some points, where the operator intervened, were removed. The training parameters were taken as shown in the following table.

Momentum	0.55
Learning rate	0.1
Learning increment	1.04
Learning decrement	0.8
Error ratio	1.04
Error goal	0.001

Table 3.3.1 Network training parameters

Each network was trained for a total of 8000 epochs in sessions of 2000 epochs with the weights and biases being saved after each session. After training, the neural network model of each data set was tested on all the other data sets and the error was recorded.

In this way a set of neural network steam valve sub-models were developed for each data set with varied network topology as shown in the table 3.3.2, i.e. by altering; the number of neurons in the hidden layer, the number of past and present values of the network inputs and outputs, and the type of hidden and output processing neurons.

	Hidden Neurons	Input & Output	Hidden Processing	Output Processing
First training set	3	3	Tan-Sigmoid	Log-Sigmoid
2nd training set	3	3	Log-Sigmoid	Linear
3rd training set	2	2	Log-Sigmoid	Linear
4th training set	5	5	Linear	Tan-Sigmoid
5th training set	5	4	Log-Sigmoid	Tan-Sigmoid
6th training set	6	5	Tan-Sigmoid	Log-Sigmoid

Table 3.3.2 Neural network training topology

During the training of the different neural network topologies, the network training parameters such as momentum and learning rate were also altered. The momentum was varied from 0.4 to 0.9 and the learning rate from 0.05 to 0.15 to see the affect of these parameters in reducing the networks mean squared error while training.

3.4 HEAT EXCHANGER SUB-MODEL

This model describes the dynamics of the heat exchanger. The boundary of this model was represented with a dotted line [ii] as shown in the figure 3.3.1 (refer to section 3.3).

The input variables to this model were the past and present values of the hot water temperature, T_{hi}, (at times t = k-2, k-1 and k) and the three previous output values of product output temperature, T_{po}. The model predicted the output product temperature, T_{po}, at time t = k as shown below.

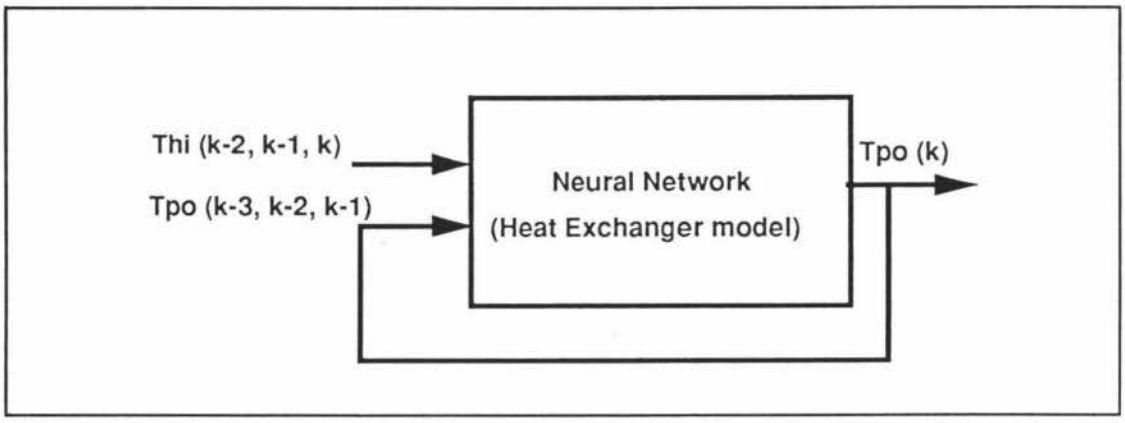

Fig 3.4.1 Neural network heat exchanger sub-model

Versions of the steam valve sub-model were trained using each of 18 sets of patterned data from different process runs. The Matlab Neural Network Toolbox was used to train the neural network models using backpropagation algorithm (refer to section 8.2 of Appendices).

The collected process data was scaled between 0.1 and 0.9 in order to make it suitable for neural network training. The scaled data was placed into a matrix containing time delayed inputs and previous outputs (refer to section 8.1 of Appendices).

Before training the network topology was specified. Six input neurons were required for the present and past values of the three input variables. One output neuron was required for the output product temperature, T_{po}. The hidden layer consisted of three tan-sigmoid neurons.

All data from a given run was used for training a given version although some points, where the operator intervened, were removed. The training parameters were taken as shown in the following table.

Momentum	0.85
Learning rate	0.1
Learning increment	1.04
Learning decrement	0.8
Error ratio	1.04
Error goal	0.001

Table 3.4.1 - Network training parameters

Each network was trained for a total of 8000 epochs in sessions of 500 epochs with the weights and biases being saved after each session. After training, the neural network model of each data set was tested on all the other data sets and the error was recorded.

In this way a set of neural network heat exchanger sub-models were developed for each data set with varied network topology as shown in table 3.4.2, i.e. by altering; the number of neurons in the hidden layer, the number of past and present values of the network inputs and outputs and the type of hidden and output processing neurons.

	Hidden Neurons	Input & Output	Hidden Processing	Output Processing
First training set	3	3	Tan-Sigmoid	Log-Sigmoid
2nd training set	3	3	Log-Sigmoid	Linear
3rd training set	3	3	Log-Sigmoid	Tan-Sigmoid
4th training set	2	2	Linear	Tan-Sigmoid
5th training set	4	3	Tan-Sigmoid	Linear
6th training set	4	4	Tan-Sigmoid	Log-Sigmoid
7th training set	5	5	Tan-Sigmoid	Log-Sigmoid
8th training set	7	5	Tan-Sigmoid	Log-Sigmoid

Table 3.4.2 - Neural network training topology

The momentum and learning rates were altered as for the previous models.

3.5 COMBINED NEURAL NETWORK MODEL

The combined model representing the process dynamics of the UHT plant was formed by the concatenation of the two neural network sub-models; the steam valve sub-model and the heat exchanger sub-model. In this combined model, the predicted output from the first model becomes one of the inputs to the second neural net model as shown in the figure below.

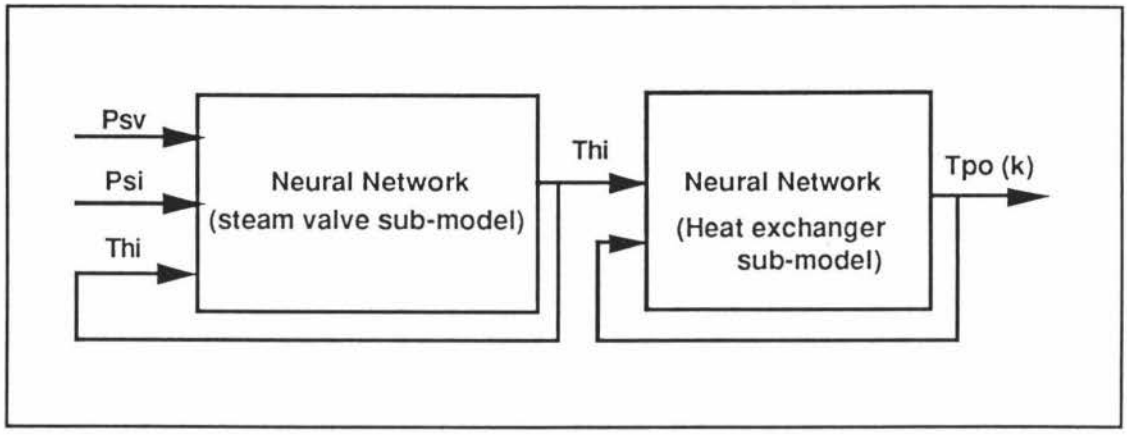

Fig 3.5.1 Combined neural network model

This combined model was used as the predictive model in the development of the UHT plant model-predictive-control system as well as the plant model. The MPC system was therefore using a perfect model to perform predictions with. Later the predictive model was systematically degraded (see section 4.4 of chapter 4).

CHAPTER FOUR

MODEL PREDICTIVE CONTROL SYSTEM AND SOFTWARE DEVELOPMENT

Model predictive control is one of the family of model based controllers which includes model algorithmic control (MAC), dynamic matrix control (DMC) and internal model control (IMC) (already discussed in chapter 2).

This chapter gives a general description of the model predictive control theory and the development of a model predictive control (MPC) strategy for the UHT plant. This MPC strategy utilises the perfect, composite neural network process model (as discussed in chapter 3) as the predictive model in the model predictive controller and also to simulate the real plant. For performance tests, the prediction model and plant model were taken from different process runs, so that the prediction and plant model were not identical.

The MPC software was developed as individual blocks and then an MPC test was designed to simulate the UHT plant MPC scheme which includes all the developed blocks. This chapter discusses the MPC software development in detail.

4.1 INTRODUCTION

The model predictive control scheme is one of the family of model based controllers. This strategy uses an explicit and separately identifiable process model as do the other types of model based controller. The ability of MPC designs to yield high performance control systems makes them capable of operating without expert intervention for long periods of time (Garcia, *et al.*, 1989).

During the past decade, a number of theoretical papers on the development of linear model predictive control techniques and their applications in the industry paved the way for the effective growth in the usage of advanced control features (Sistu *et al.*, 1991). As the linear MPC methods are based on linear systems theory, their performance is affected largely by modelling errors caused by system uncertainties and nonlinearities. As system uncertainties exist in almost all the real process plants, these methods may not perform well on highly nonlinear systems. This lead to the design of nonlinear predictive control (NLPC) method which became an effective strategy for controlling nonlinear process systems with constraints and time delays.

As a control methodology, MPC has received interest from a number of industries like the petrochemical industry which requires sophisticated control strategies for its complex and highly critical processes (Cutler and Ramaker, 1980 ; Richalet *et al.*, 1978).

The MPC methodology's control law can be described as follows. At the present timestep, k, the plant responses over a specified prediction horizon, P, are considered. The objective is to select a set of M future control moves, known as the control horizon, to minimise a function based on a desired output trajectory over a predicted horizon as shown in Figure 4.1.1.

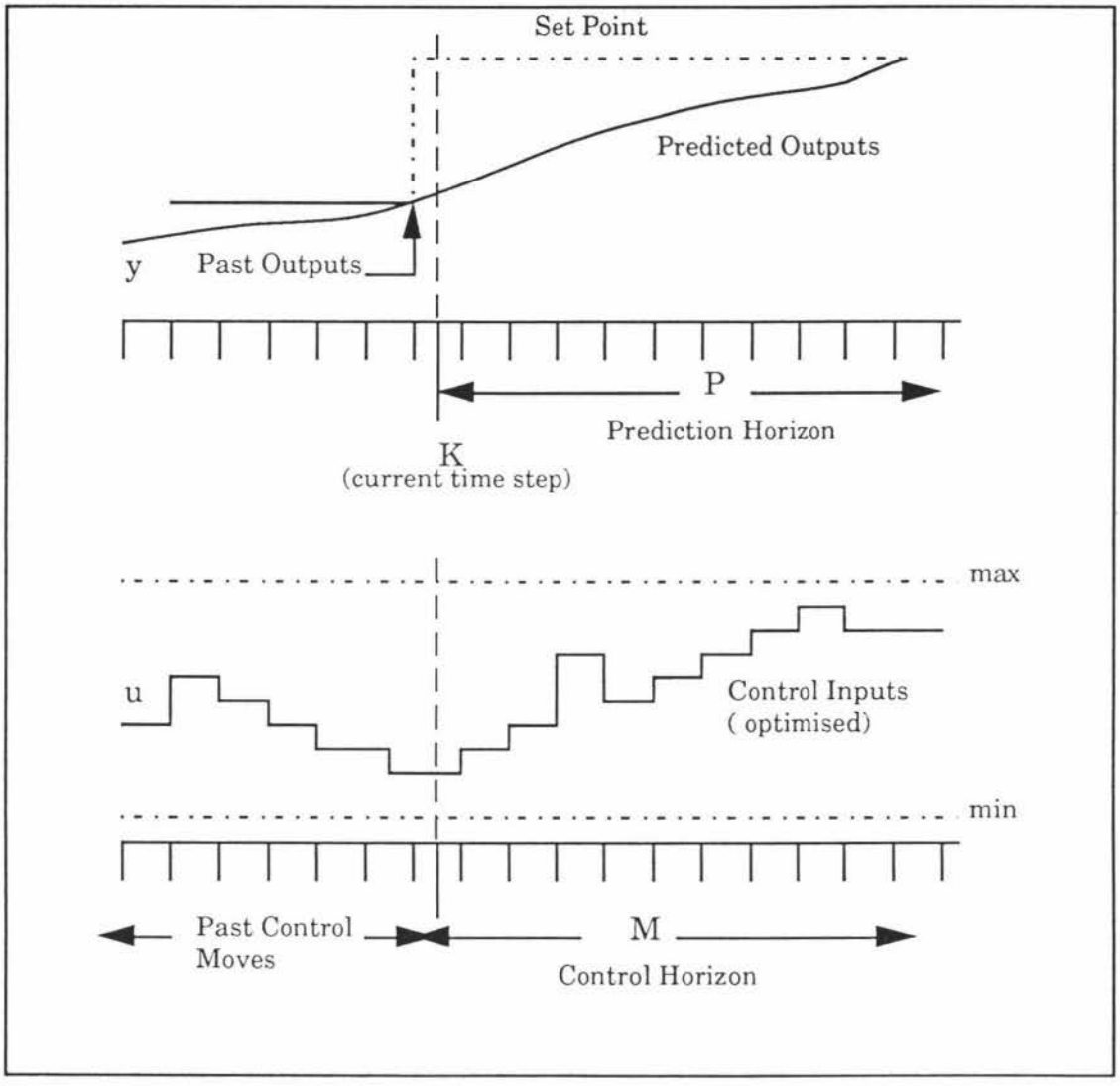

Fig 4.1.1 Predictive control approach of MPC

The optimisation variables are the control actions M time steps into the future and it is assumed that the control action after the M^{th} time step remains constant. Although M future control inputs are calculated, only the first computed change in the manipulated variable is implemented on the actual plant. Then the actual plant output measurements are obtained and at the next time step, i.e. k+1, the computation is repeated with the prediction horizon moved forward by one time interval.

We will assume a dynamic model for the plant requiring control is given by:

$$\dot{x} = f(x,u)$$

$$y_{mp} = g(x)$$

where the system output, y_m, is a function of the system states, x, and the derivatives of the states are a function of the states and the system inputs, u.

The MPC strategy uses a dynamic model of the plant for the prediction of the effect of future actions of the control inputs and these inputs are selected in a way to minimise the prediction error which is given as

$$e(p) = y_{sp}(p) - y_{mpr}(p)$$

where e(p) is the prediction error at present time step, y_{sp} is the reference setpoint or desired output and y_{mpr} is the output predicted by the model.

In this methodology, the future moves of the manipulated variables or control inputs, u, are determined by optimisation with the objective of minimising the prediction error subject to constraints. In carrying out the optimisation, the non-linear programming problem needs to be solved. The optimisation is repeated at each sampling time based on the updated actual process measurements from the plant.

The objective function to be minimised is the sum of the square of the prediction errors over the prediction horizon of P time steps which can be written as:

$$\min \Phi(u) = \int_{tk}^{tk+Tp} e^2 dt = \sum_{i=k+1}^{k+P} S[(y_{sp}(i) - y_{mpr}(i)]^2$$

for u(k),....., u(k + M - 1) where S is a positive weighting factor.

In Dynamic Matrix Control, the term to minimise the changes in the control input, u, will also be included to the above objective function which is given as

$$\min \Phi(\mathrm{u}) = \sum_{i=k+1}^{k+P} \mathrm{S}[(y_{sp}(i) - y_{mpr}(i)]^2 + \sum_{j=k+1}^{k+M} \mathrm{T}[(u(j) - u(j-1)]^2$$

where T is a positive weighting factor.

In general, the operating point of a plant which satisfies the overall economic goals of the process will lie at the intersection of constraints (Arkun, 1978; Prett and Gillette, 1979). So a good control system needs the anticipation of constraint violations and action against them in a systematic way. In the conventional control methodologies, the operating constraints on the manipulated inputs, state variables and outputs will be ignored at the system design stage and taken into consideration only during implementation and dealt in an *ad hoc* manner. The important feature of model predictive control strategy is that it allows the system designer to consider the operating constraints during the design and implementation of the controller itself.

So the basic objective function to be minimised at each sampling time needs to cope with the operating constraints as given below.

- Constraints on the control inputs:

$$u_{\min} \le u(i) \le u_{\max}$$

where u_{min} is the minimum value of the control input and u_{max} is the maximum value of the control input.

- Constraints on the control input deviation from the previous input:

$$u(i-1) - \Delta u_{\max} \le u(i) \le u(i-1) + \Delta u_{\max}$$

- Constant control action assumed for time steps beyond the control horizon, M :

$$u(i) = u(k+M-1) \text{ for all } i > (k+M-1)$$

- Constraints on the model states and outputs:

$$x_{\min} \leq x(i) \leq x_{\max}$$

$$y_{\min} \leq ymp(i) \leq y_{\max}$$

where the model output, y_{mp} is a function of state variables. However a correction needs to be applied for the plant/model mismatch to obtain a better prediction of the outputs (y_{mpr}). This correction is based on the difference between the actual value of the controlled output, y, at the current time step and the model prediction, and is assumed to remain constant over the entire prediction horizon:

$$d_i(k) = y(k) - y_{mp}(k)$$

Then the predicted output becomes:

$$y_{mpr}(k+i) = y(k) - y_{mp}(k) \quad \text{for i = 1 to P}$$

The compensation of the plant/model mismatch is performed, and the optimisation is performed again.

The general scheme is to utilise a model to predict the output into the future and try to minimise the difference between these predicted outputs and the desired trajectory.

In addition to the non-linear optimisation problem, the model predictive control strategy needs to cope with the constraints on both the manipulated inputs and outputs. The basic objective function to be minimised at each sampling time needs to be defined. This function has to deal with variable horizons of predictions, time-variant set-points and be differentiable.

The basic properties of this algorithm are:

a) ability to minimise a non-linear smooth function

b) ability to deal with any kind of non-linear equality and/or inequality constraints on the manipulated variables

c) the ability to generate only feasible iterations

The last feature is important for two reasons. one is that a neural network can only predict outputs inside the trained region; an infeasible point in this case may be physically feasible but infeasible from the neural network point of view as the point lies outside the training region. The other reason is the possibility of not reaching the optimal set of manipulated variables at each sampling time in the case of implementation on a real complex process plant, so in this case of incomplete optimisation a better set of manipulated variables is always available (Donat *et al.*, 1991).

Although the model predictive control systems involve more computation compared to the linear time invariant algorithms, the flexible constraint handling capabilities made them more attractive for practical applications. For example, a sudden failure in a process operation such as a stuck valve can be specified on the console as an additional constraint for the optimisation routine by the process operator. The control algorithm compensates this failure by means of adjusting the actions of all other manipulated variables automatically. Thus MPC system will keep the process operating safely away from all the constraints or allow the operation to be shut down in a smooth manner.

A number of tuning parameters can be used to adjust the desired speed of response of the closed-loop systems. For example, longer prediction horizons, P, and shorter control horizons, M, increase the closed-loop speed of response and add robustness for open-loop, stable, minimum-phase process systems.

4.2 UHT PLANT MODEL PREDICTIVE CONTROL SYSTEM

After the development of the neural network model of the process as described in chapter 3, the UHT plant model predictive control system was developed in the form of blocks and interconnected at the functional level as given in the figure 4.2.1. The model predictive controller comprising the optimiser and neural network process model, and the real UHT plant forms the UHT MPC system.

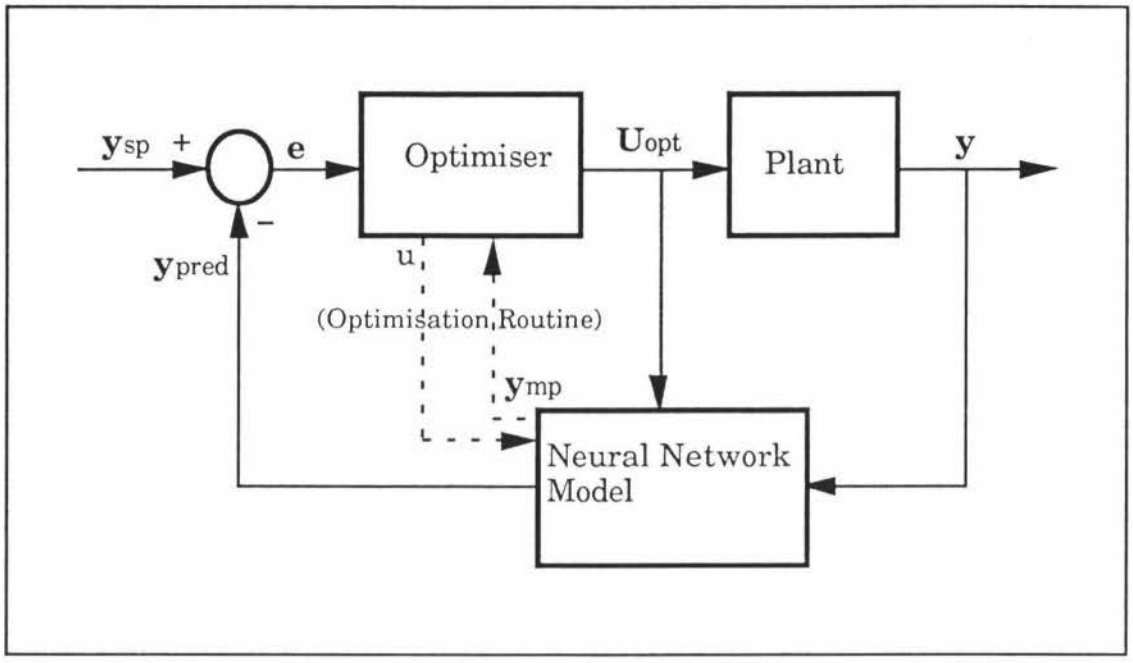

Fig 4.2.1 UHT plant Model Predictive Control System

In this control system design in the simulated environment, a perfect composite neural network model of the process acts as a real plant and will be referred to as the 'plant'. The designed model predictive controller consists of an optimiser and a neural network model representing the UHT process. This model will be used for the prediction of outputs and will be referred to as the 'prediction model'.

The inputs to the prediction model in this MPC system are the past inputs to the plant, past outputs from the plant and the optimised control input sent to the plant from the optimiser. The prediction model in turn calculates a horizon of predicted outputs and sends them to the performance evaluation function which is a part of the optimiser. The performance function also takes in the predicted future control actions to compute the performance index i.e., a factor which balances the plant output deviation from the reference point and the input control actions.

The functional aspects of the model predictive controller which includes the optimiser and the prediction model and their software development are explained in the following sections. The working methodology of the complete MPC system is also explained in detail.

4.2.1 Model Predictive Controller

The complete model predictive controller consists of an optimiser block and the prediction model. The model predictive controller computes a horizon of future control actions, i.e. optimised control inputs, over a specified control horizon, but only the first control input will be sent to the plant. The input variables to the controller are the past inputs to the plant, the past outputs from the plant, the reference output or setpoint and the constraints on the controlled inputs, state variables and outputs.

The performance evaluation function which is a part of the optimiser block takes in the present and past control inputs, the future outputs at the present moment predicted by the prediction model and the reference or desired output. This function computes the performance index, a scalar to minimise the objective function as given below.

$$J = \sum_{i=k+1}^{k+T} (y_i - y_{sp}) Q (y_i - y_{sp})^T + (u_i - u_{i-1}) R (u_i - u_{i-1})^T \quad \text{(eqn 4.2)}$$

where
- y_i = output from the prediction model at i^{th} time step
- y_{sp} = desired or reference output
- u_i = control input at i^{th} time step
- Q = positive weighting matrix to minimise change in output
- R = positive weighting matrix to minimise control action

In the process of calculating the present and future control actions, the optimiser enables a compromise between the excessive control action and the output deviation from the desired setpoint. The reason for the

computation of a horizon of control actions at each time step is to ensure the control system realises the consequences of the present control action on the future plant outputs which helps in not making any harmful and short term decisions. With the exception of the present control action, all the other future actions will not be passed on to the plant to avoid an open loop plant and to have the benefit of feedback from the plant.

4.2.2 MPC Software Development

After the functional development of the complete MPC system for the UHT plant, as explained in the section 4.2.1, the software was developed for the final MPC scheme. In this development attempts were made to maximise the generic nature of the software so that it could be used for any number of system inputs, outputs and future prediction steps by the prediction model.

The model predictive control test software was developed as blocks and then combined to simulate the complete model predictive control strategy of the UHT process. In this MPC test all the required variables for simulation, e. g. the input and output variable values, the desired constraints on the control input and output process variables, and the number of future control input predictions were initialised.

In the plant function software development, the combined neural network model function acted as a real UHT process plant. The control input from the optimiser U_{opt}, was a set of past inputs and past plant outputs and a vector of the disturbance input, i.e. inlet steam pressure, P_{si}, became the inputs to the plant model.

The past plant outputs were fedback to the neural network model acting as the real plant as past outputs for the computation which would not exist in a 'real' control situation. This function computed the plant output, T_{po}, (the output product temperature) and send it to the prediction model.

The flow chart of the MPC system software development was shown in the figure below.

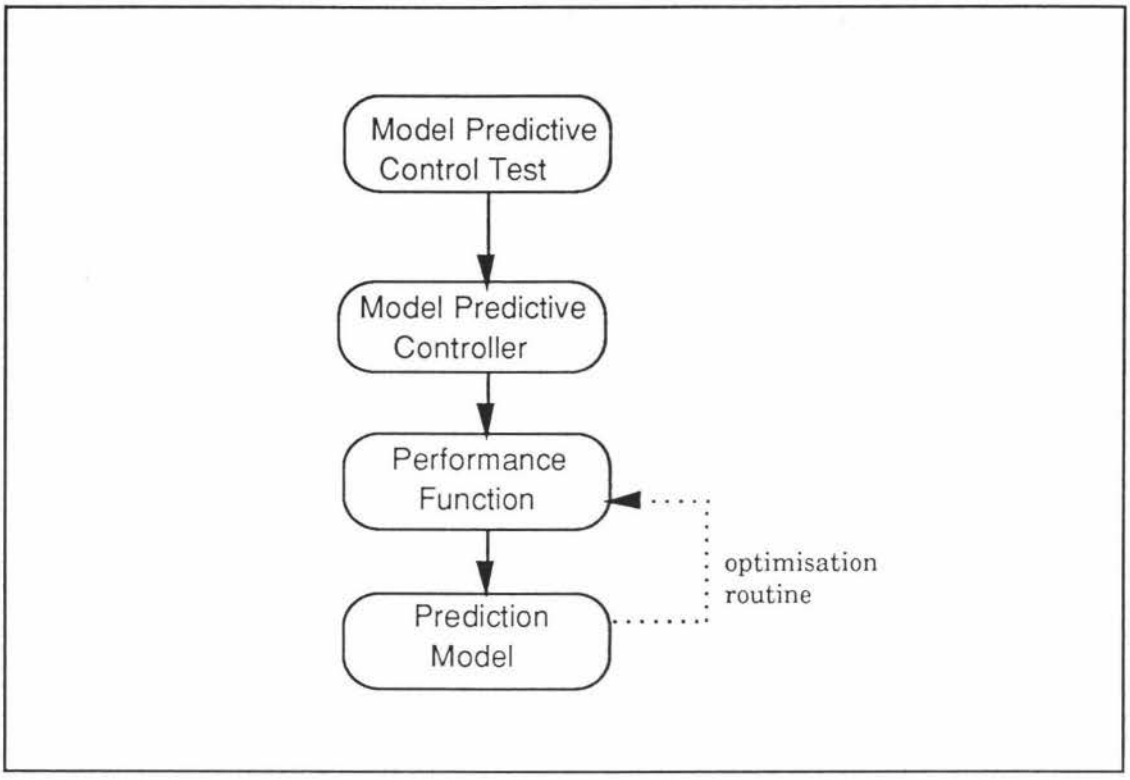

Fig 4.2.2 UHT MPC system functional flow chart

The model predictive controller software was written in such a way that it perform the computation of a horizon of future control inputs but only the first control action will be implemented, i.e. only the first control input will be allowed to go to the plant. In the model predictive controller the performance function will calculate the balance between the excessive input control action and the deviation of the output from the desired setpoint.

In the prediction model software development, the combined neural network model as explained in chapter 3 will act as the process model for the prediction purposes ('predmod' in the simulation software, refer to section 8.3 of Appendices). The prediction model function takes in the horizon of control inputs from the optimiser, actual past inputs and outputs of the plant and the disturbance input vector, i.e. the inlet steam pressure, P_{si}. This function computes a horizon of future output predictions in time steps equivalent to the number of control inputs computed by the optimiser.

The software for the UHT plant MPC test was written in such a way that it allows specification of the number of future control action steps in its functional parameters. The prediction model software was verified in a similar manner to the plant model software. This prediction model function has a major influence on the computation time of the entire MPC scheme as it is called many times by the optimiser for each set of control inputs required by the plant. So this part of the software was written in an efficient way to reduce its runtime.

The optimiser computes the horizon of control inputs and dynamically passes them to the prediction model, which in turn gives the predicted output horizon to the objective function. This entire process is iterative until the minimisation of the objective function has reached a desired level. The optimiser then sends this set of control inputs to the plant model.

If the system was run in real-time, then the maximum number of optimiser iterations would need to be specified as inputs need to be supplied to the plant at regular control intervals. If the optimal horizon of control inputs can not be obtained in time then the 'best' current set of control inputs should be sent to the plant. In general, it is desirable that the optimiser iteration process cease most of the time when the error goal is obtained and rarely cut short by the maximum number of iterations limit. If the optimiser can not obtain the error goal for the majority of the time then the plant would not suitable for the MPC scheme design.

The constrained optimisation algorithm called 'CONSTR' from the optimisation toolbox of MATLAB was chosen in the optimiser software development. Basically this algorithm uses the Sequential Quadratic Programming (SQP) optimisation technique. This algorithm was selected after looking at the various conditions of suitability such as the capability to handle the constraints on both the control inputs and the outputs.

The model predictive controller function calls this function to obtain the best optimised guesses for the control input. The optimiser uses the constrained optimisation algorithm. This inturn uses performance function (perform.m file, refer to appendices 8.3.7) and calls the neural net model for calculating the output predictions which will be used to calculate the control input guesses.

The neural network model will calculate the output predictions equal to the number of future control inputs specified in the model predictive control test and pass on these to the performance function. This process repeats until the optimiser function finds the best optimised input guesses by means of performance evaluation or the number of specified optimiser steps. The first control input from the calculated control horizon of the model predictive controller will be passed onto the plant.

In this model predictive control strategy, the developed neural network models were used for both prediction model and the plant. At the initial stages the same neural network model was used for both giving a perfect prediction in the absence of disturbances.

Evaluation of the model predictive control system performance was carried out in terms of disturbance rejection and setpoint tracking. To create a plant-model mismatch, neural network models trained on different sets of data were used to represent the plant and the prediction model.

Based on these tests a comparison between the model predictive control system performance and a Proportional-Integral controller was carried out. The results and the observations are given in the next chapter.

As the designed control system is based on a horizon of predicted future outputs from a neural network model, offsets may occur due to the mismatch between the process and model gains. In the internal model control (IMC) system design, this discrepancy can be estimated as the difference between the plant and model outputs and the estimate is used as an additional signal by the control algorithm.

CHAPTER FIVE

RESULTS AND DISCUSSION

The sum-squared and mean-squared error results of training the single and composite neural network models are presented in this chapter. The composite neural network model of the UHT process comprises both the steam valve and heat exchanger sub-models. The composite network model results also include the training and testing results of the steam valve and heat exchanger network sub-models.

This chapter also presents the testing results of the developed UHT plant model predictive system as described in chapter 4 for setpoint changes and disturbance rejection. These results include the testing of both the developed model predictive control systems; one with the best composite neural network model and the other with the worst composite model. The best composite model comprises the best steam valve and heat exchanger sub-models, and the worst composite model with the worst steam valve and heat exchanger sub-models.

A PI (Proportional-Integral) control system was also developed using the best composite neural network model as a UHT plant. This chapter presents the comparative performance results of both the model predictive control systems and the PI control system for setpoint changes and disturbance rejection.

5.1 NEURAL NETWORK PROCESS MODELS

Individual neural network sub-models were developed to represent the two parts of the UHT process namely; the steam valve sub-model and the heat exchanger sub-model, as described in chapter 3. By means of concatenating these two sub-models, a composite network model was developed to represent the complete UHT process. A single neural network model which represents the whole UHT process was also developed.

As a first performance measure, each neural network model was validated against itself by means of comparing the sum-squared-error between net output and target values as the network training progressed. Initially these errors were high but as the training proceeded the neural network learned the system and the errors decreased.

Each of the neural network models was trained for a specified number of epochs and then tested against the remaining sets of data using the trained network model weights and biases and the scaled plant data of other sets. The training of the neural network models was stopped when the errors reached a minimum. Attempts to train the network model further resulted in learning of the data set specifically and poor generalisation to other data sets. The neural network training results of the single and composite network process models were presented in the following sections.

5.1.1 Single Network Models

Each single neural network model of a specific data set was trained by varying the number of past input and output values, and neurons for hidden layer. Then the trained model was tested against all the other available data sets.

The single neural network model trained with 3 tansigmoid neurons in the hidden layer and one logsigmoid as output neuron proved to be the best for all the available data sets. This model comprised three past and present values of inputs, steam valve position P_{SV}, steam inlet pressure

P_{si} and hot water temperature entering the loop T_{hi}, and three past values of the product temperature exiting the heat exchanger T_{po}.

As an example the following graph in figure 5.1.1.1 shows the one-step-ahead prediction of a single neural network model trained on one data set (UH76) and tested on another data set (UH71) with the scaled value of the product temperature exiting the heat exchanger, T_{po}, against training pattern number.

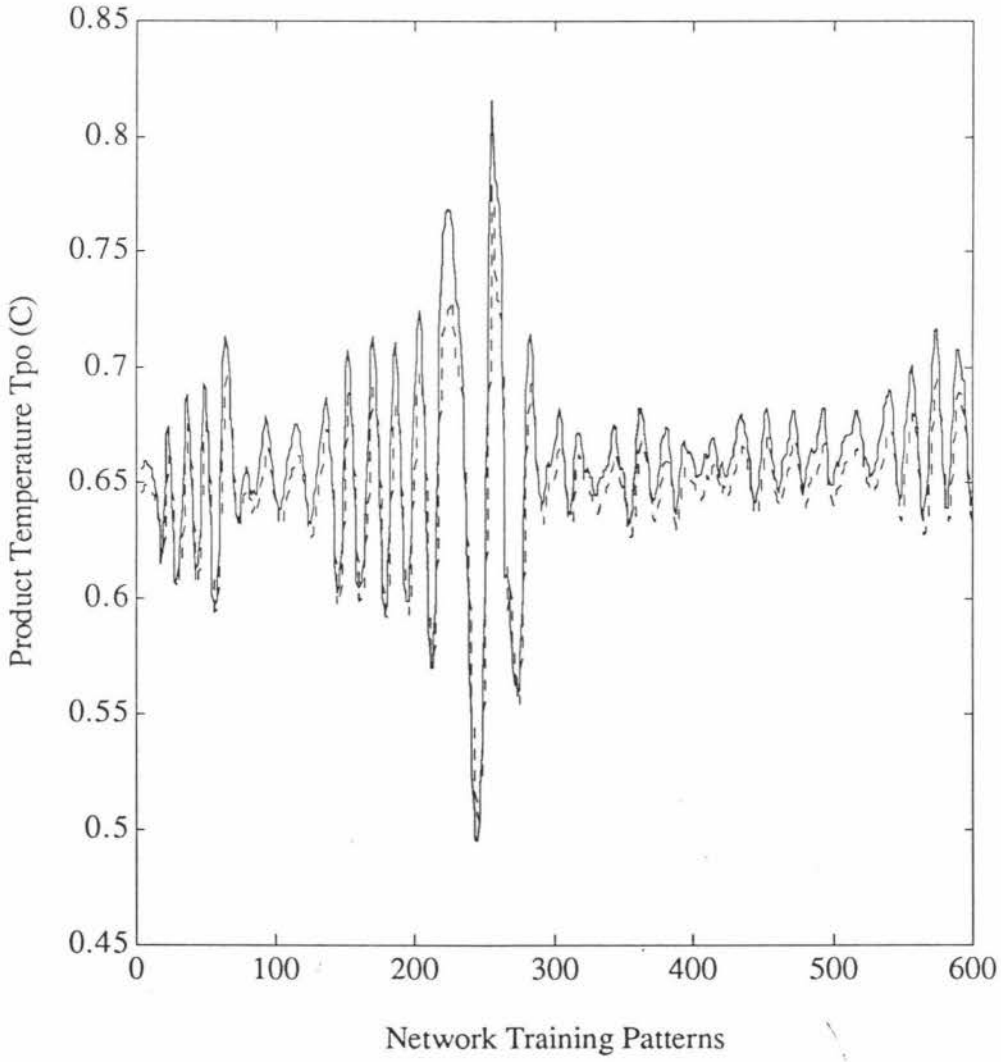

Fig 5.1.1.1 Single neural network model output prediction

The mean-squared-errors of some data sets trained and tested with the above configuration are given in Table 5.1.1. Refer to the Appendices (Section 8.4.1) for the sum-squared and mean-squared errors of each data set tested against all other sets.

ERROR	BEST	MODELS	WORST	MODELS
(MSE % FS)	UH76	UH71	UH74	UH73
TRAINING	0.0032	0.0078	0.0027	0.0028
BEST	0.0029	0.0078	0.0027	0.0028
AVERAGE	0.0112	0.0220	0.0272	0.0328
WORST	0.0418	0.0401	0.0792	0.1362

Table 5.1.1 Mean-square-errors as a percentage of full scale for single-network models

Table 5.1.1 shows the mean-square-error of the best and worst single models as a percentage of the full scale output i.e. product temperature exiting the heat exchanger, T_{po}. These errors were the result of one-step-ahead prediction of the scaled plant data.

The single models UH76 and UH71 proved to be the best while the UH74 and UH73 single network models were the worst. The 'Training' figure shows the error in predicting the plant data on which the model has been trained. The 'Best', 'Average' and 'Worst' figures show the best result, the average result and the worst result respectively of testing the model against all other plant data runs.

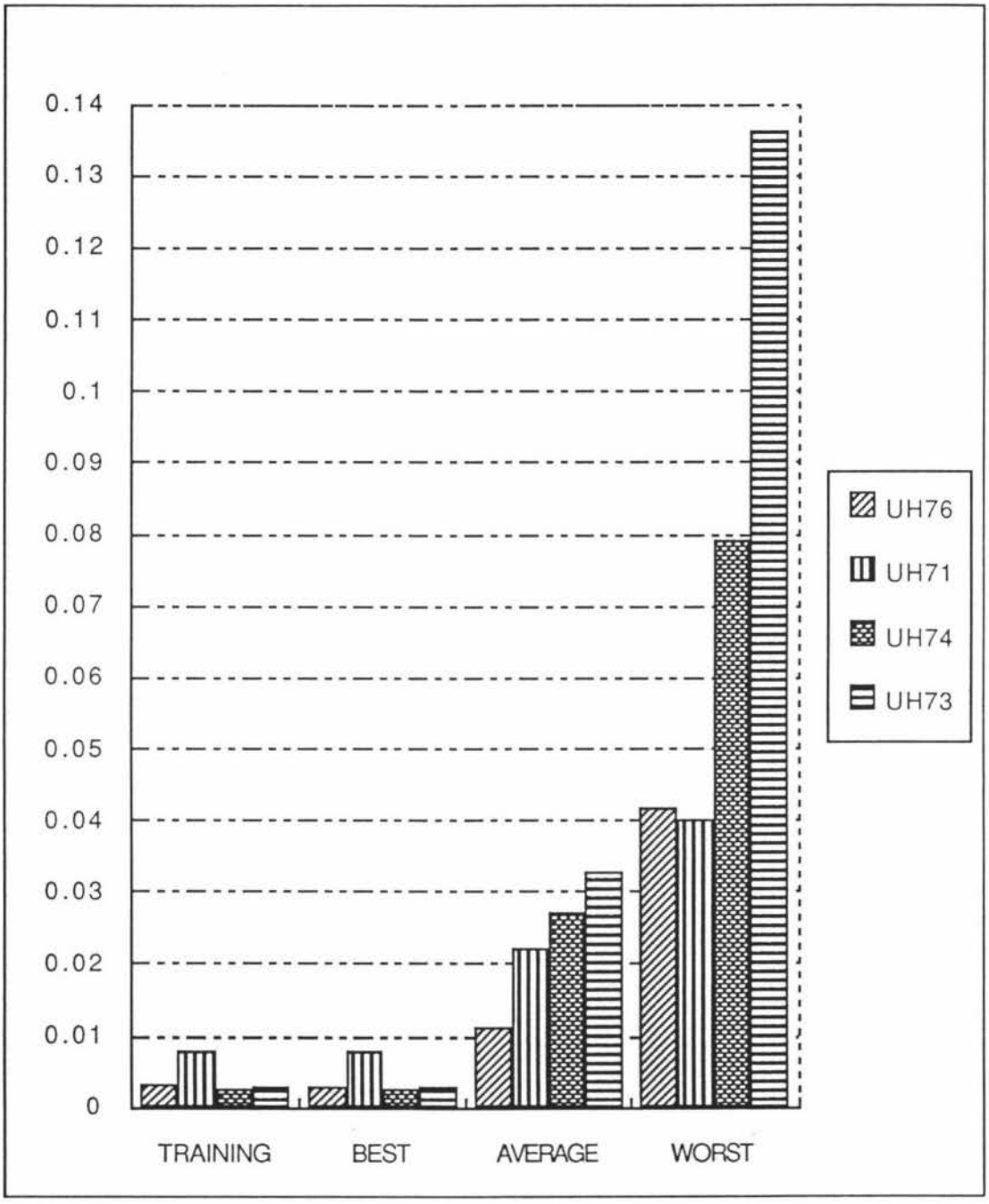

Fig 5.1.1.2 Mean-square-errors for single models

The graph in figure 5.1.1.2 shows the same error results as shown in table 5.1.1. From the graph it is evident that the single network model was able to perform respectably for all the plant data runs with most of the errors remaining below 0.05% of the full scale output except one which remains below 0.1%.

For the single network model trained with more than 3 past input values, the sum-squared-error for one data set tested on itself did not change and resulted in high error values when tested against other data sets. The same model when trained with less than 3 past input values had very high training errors and high testing errors.

The single network model trained with less than 3 hidden neurons gave high errors when tested against itself and against other plant data sets due to underfitting of the trained network model i.e. the weights and biases resulted from the trained model unable to produce outputs reasonably close to the targets. The same model when trained with more than 3 hidden neurons learned the plant data on which it was trained but failed to generalize to other plant data runs.

5.1.2 Composite Network Models

After dividing the whole UHT process into two sub models, as described in chapter 3, both the steam valve and the heat exchanger sub models of a specific data set were trained by varying the number of past input and output values, and the number of neurons for hidden layer. Then the trained models were tested against all the other data sets.

This section describes the neural network training and testing results of both the steam valve and heat exchanger submodels and also the composite model formed by the concatenation of the above two sub-models.

5.1.2.1 Steam valve sub-model

The steam valve neural network sub-model trained with 3 tansigmoid neurons in the hidden layer and one logsigmoid as output neuron proved to be the best for all the available data sets. This model

comprised three past and present values of inputs, steam valve position, P_{sv}, and steam inlet pressure, P_{si}, and three past values of the hot water temperature entering the loop, T_{hi}.

The mean-squared-error data of some data sets trained and tested with the above configuration are given in the Table 5.1.2.1. Refer to the Appendices (Section 8.4.2) for the sum-squared and mean-squared errors of each data set tested against all other sets.

ERROR	BEST	MODELS	WORST	MODELS
(MSE % FS)	UH74	UH75	UH76	UH73
TRAINING	0.0012	0.0012	0.0037	0.0025
BEST	0.0012	0.0012	0.0025	0.0025
AVERAGE	0.0087	0.0100	0.0100	0.0237
WORST	0.0337	0.0375	0.0375	0.0737

Table 5.1.2.1 Mean-square-errors as a percentage of full scale for steam valve sub-models

Table 5.1.2.1 shows the mean-square-error of the best and worst steam valve sub-models as a percentage of the full scale output i.e. hot water temperature entering the loop T_{hi}. These errors are the result of output prediction for one-step-ahead prediction against the plant data.

The steam valve network sub-models UH74 and UH75 proved to be the best while the UH76 and UH73 steam valve sub-models were the worst. The 'Training' figure shows the error in predicting the plant data on which the model has been trained. The 'Best', 'Average' and 'Worst' figures show the best result, the average result and the worst result respectively of testing the model against all other plant data runs.

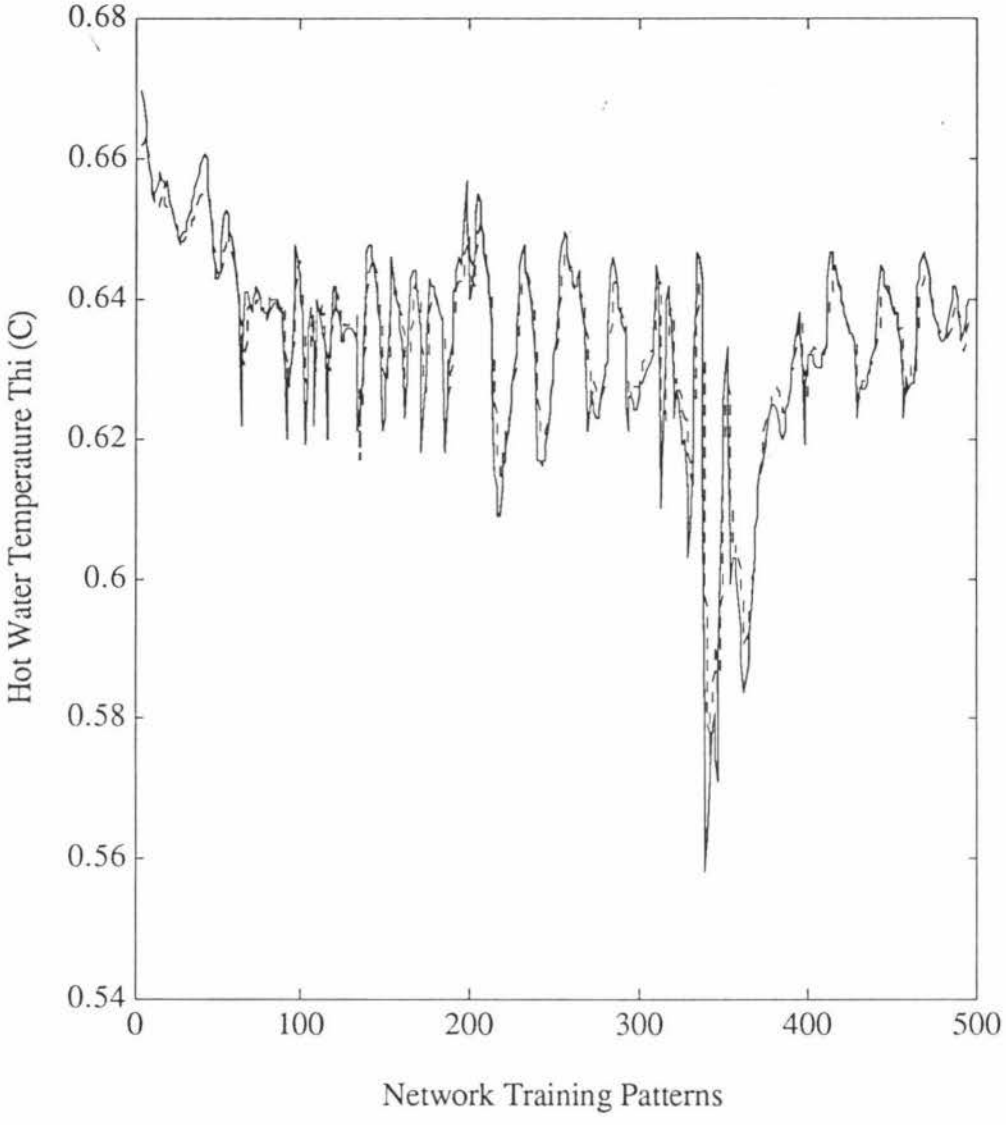

Fig 5.1.2.1.1 Output prediction by best steam valve sub-model

Figure 5.1.2.1.1 shows the best one-step-ahead prediction of steam valve sub-model trained on plant data set UH74 and tested on UH76 data set with the scaled value of the hot water temperature entering the heat exchanger loop T_{hi} along the Y-axis and the network training patterns along the X-axis. The graph shows that the network model has learned the data quite well and closely predicts the outputs of the other test data sets.

Figure 5.1.2.1.2 is the worst one-step-ahead prediction of steam valve sub-model trained on UH73 scaled plant data and tested against the UH76 plant data. The graph shows the network model's inability in predicting the scaled values of hot water temperature entering the heat exchanger loop of the UH76 data.

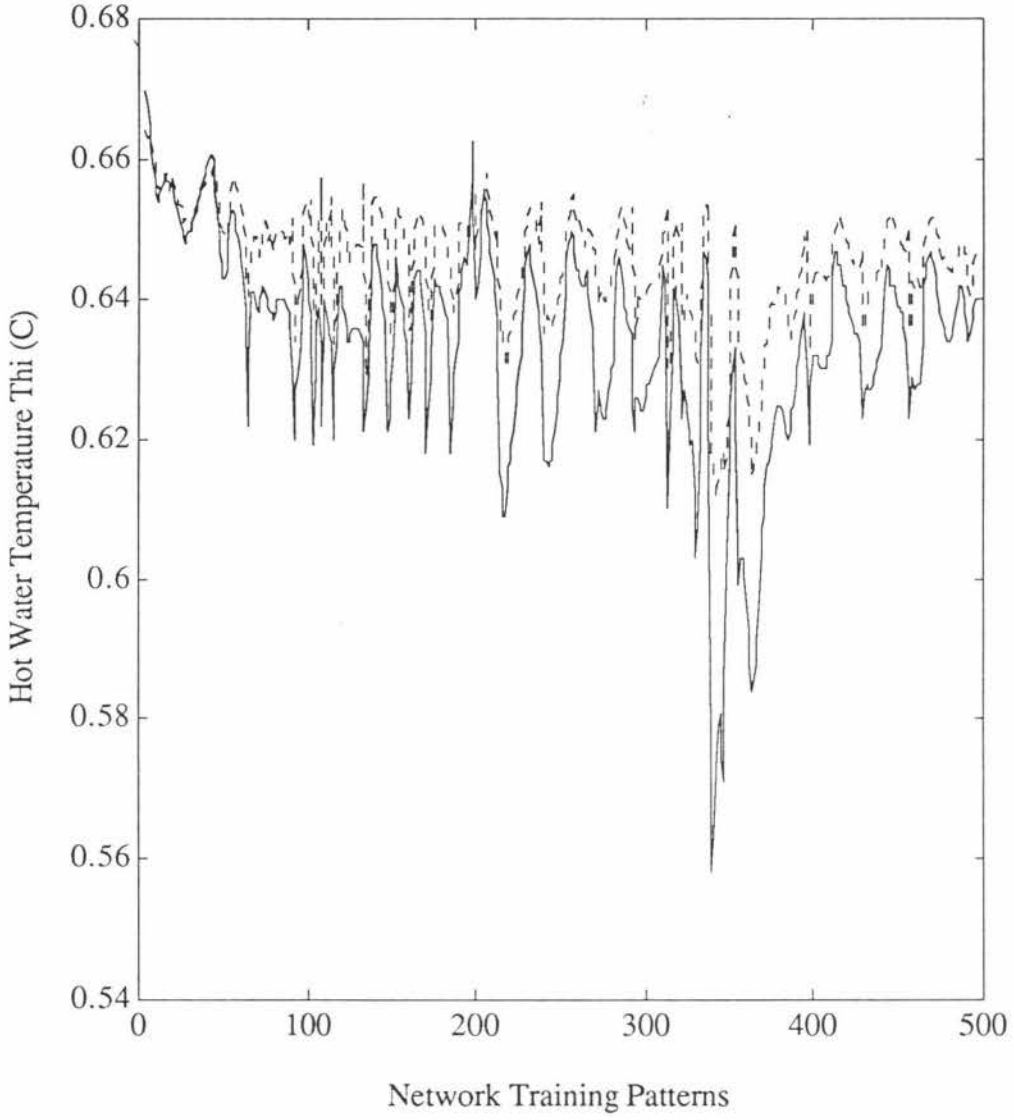

Fig 5.1.2.1.2 Output prediction by worst steam valve sub-model

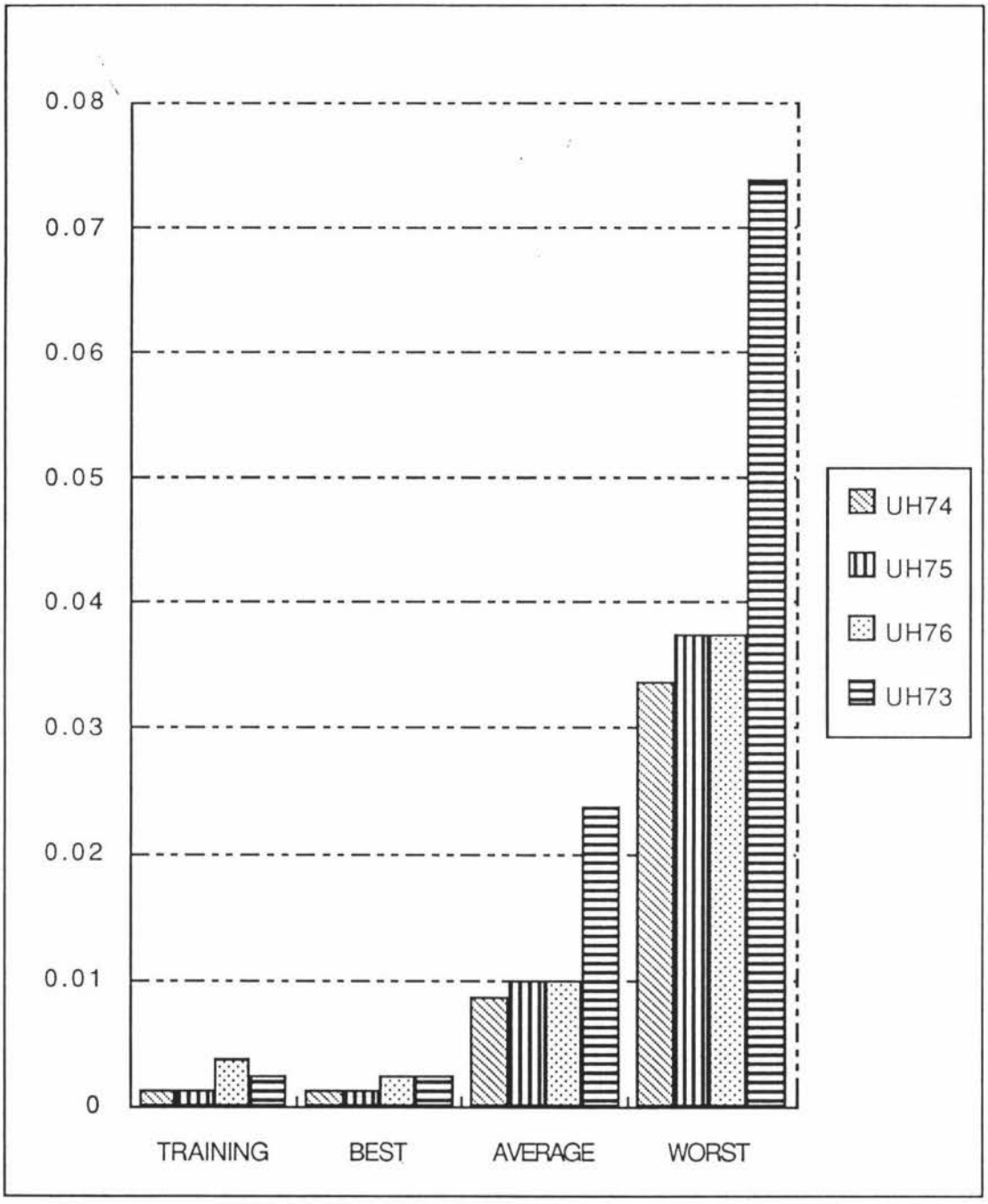

Fig 5.1.2.1.3 Mean-square-errors for steam valve sub-models

Figure 5.1.2.1.3 shows the results from the table 5.1.2.1. The steam valve sub-model shows the reasonable performance in predicting the other plant data runs.

The steam valve sub-model was trained with various values of learning rate, learning increment and learning decrement. The final values of 0.1, 1.04 and 0.8 respectively were found to work best. Similarly a momentum value of 0.95 provided the best performance during training.

The steam valve sub-model was trained with 2, 3 and 4 past input values. The results showed high training errors with less than 3 hidden neurons tested against its own input/output trained data and data of other plant runs resulted in high prediction errors due to underfitting of the trained network model i.e. the weights and biases resulted from trained model unable to produce outputs reasonably close to the targets. The same model when trained with more than 3 hidden neurons learned the plant data on which it was trained but failed to predict outputs close to the actual plant outputs when tested against other plant data runs.

After analysing the steam valve network sub-model training and testing results, the trained sub-models of UH74 and UH73 plant data were used in the development of the best and worst composite UHT network models as a part of the model predictive control and Proportional-Integral (PI) systems.

5.1.2.2 Heat exchanger sub-model

Of all the heat exchanger neural network sub-models trained with different network topologies, the sub-model trained with 3 tansigmoid neurons in the hidden layer and one logsigmoid as output neuron gave the least prediction error when tested against all the available data sets. This sub-model used as inputs for the network model training the present and two past values of hot water temperature entering the loop, T_{hi} and three past values of the product temperature exiting the heat exchanger, T_{po}.

The mean-squared-errors of selected data sets are given in the Table 5.1.2.2. For the complete set of sum-squared and mean-squared errors refer to Appendices (Section 8.4.3).

ERROR	BEST	MODELS	WORST	MODELS
(MSE % FS)	UH62	UH75	UH53	UH65
TRAINING	0.0012	0.0025	0.0025	0.0032
BEST	0.0012	0.0025	0.0025	0.0032
AVERAGE	0.0048	0.0225	0.0412	0.0562
WORST	0.0312	0.3425	0.4662	1.2262

Table 5.1.2.2 Mean-square-errors as a percentage of full scale for heat exchanger sub-models

Table 5.1.2.2 presents the mean-square-error of the best and worst heat exchanger sub-models as a percentage of the full scale output, i.e. the product temperature exiting the heat exchanger, Tpo. These errors are the result of output prediction for one-step-ahead prediction against the scaled plant data.

After training the heat exchanger sub-models using each of the plant runs and testing them against all the remaining plant runs, the sub-models UH62 and UH75 proved to be the best, and UH53 and UH65 sub-models as the worst. The 'Training' figure shows the error in predicting the plant data on which the model has been trained. The 'Best', 'Average' and 'Worst' figures show the best result, the average result and the worst result respectively of testing the model against all other plant data runs.

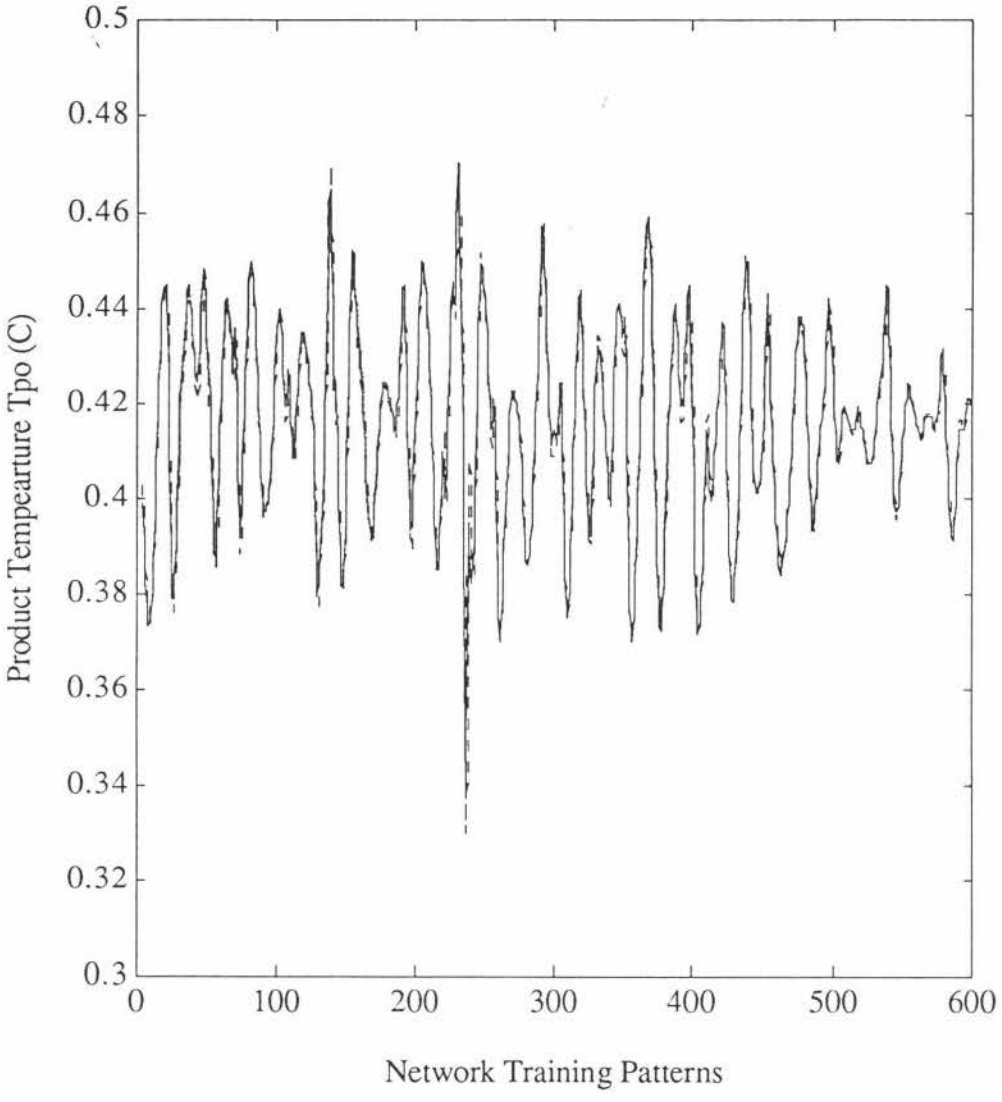

Fig 5.1.2.2.1 Output prediction by best heat exchanger sub-model

The graph in figure 5.1.2.2.1 shows the best one-step-ahead prediction of heat exchanger sub-model trained on plant data set UH62 and tested on UH53 data set with the scaled value of the product temperature exiting the heat exchanger, T_{po}, along the Y-axis and the network training patterns along X-axis. The graph shown above confirms the network model's ability in learning the data and in the close prediction of the other plant data set outputs.

Figure 5.1.2.2.2 shows the worst one-step-ahead prediction of the heat exchanger sub-model trained on UH65 scaled plant data and tested against the UH71 plant data. This graph for the worst sub-model prediction shows that the model, to certain extent, can predict the product temperature exiting the heat exchanger, T_{po}, of the UH71 data.

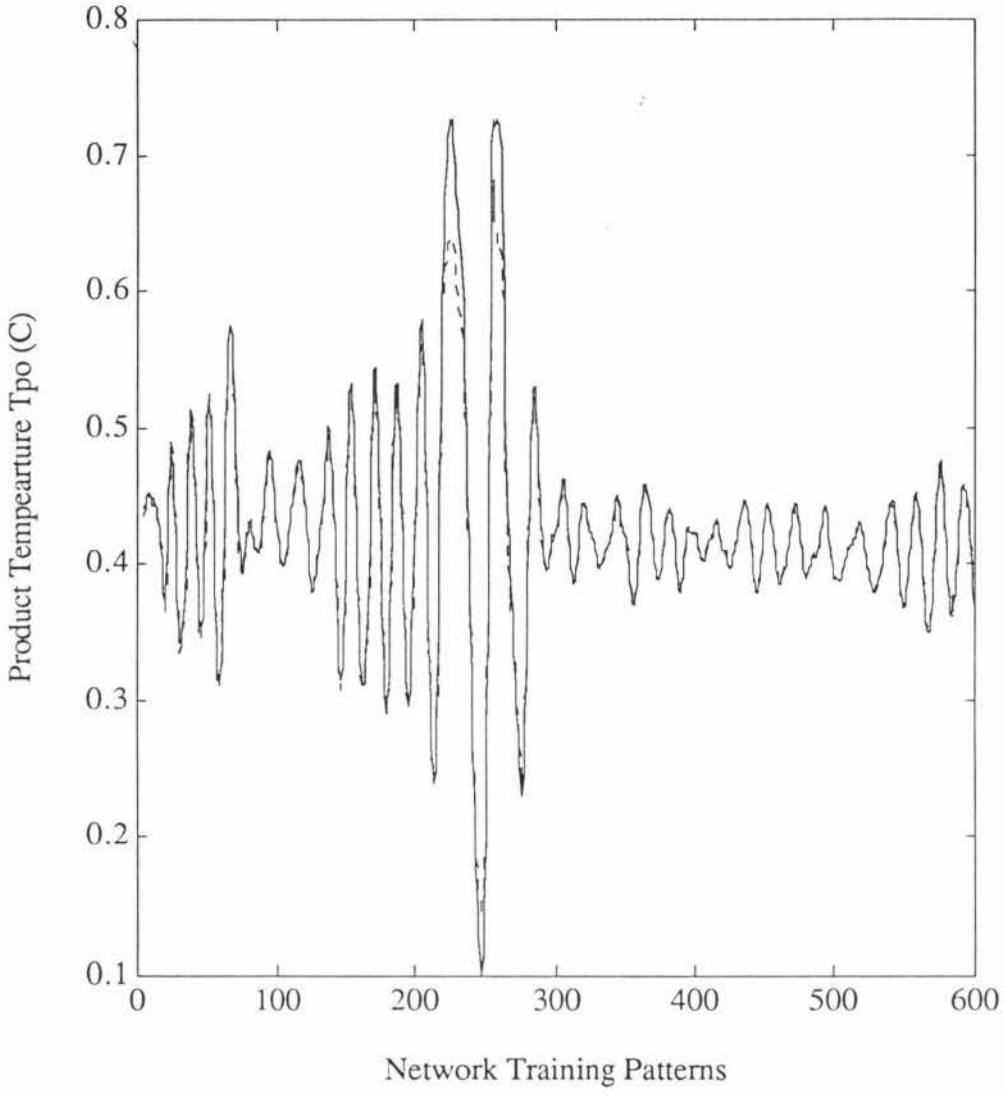

Fig 5.1.2.2.2 Output prediction by worst heat exchanger sub-model

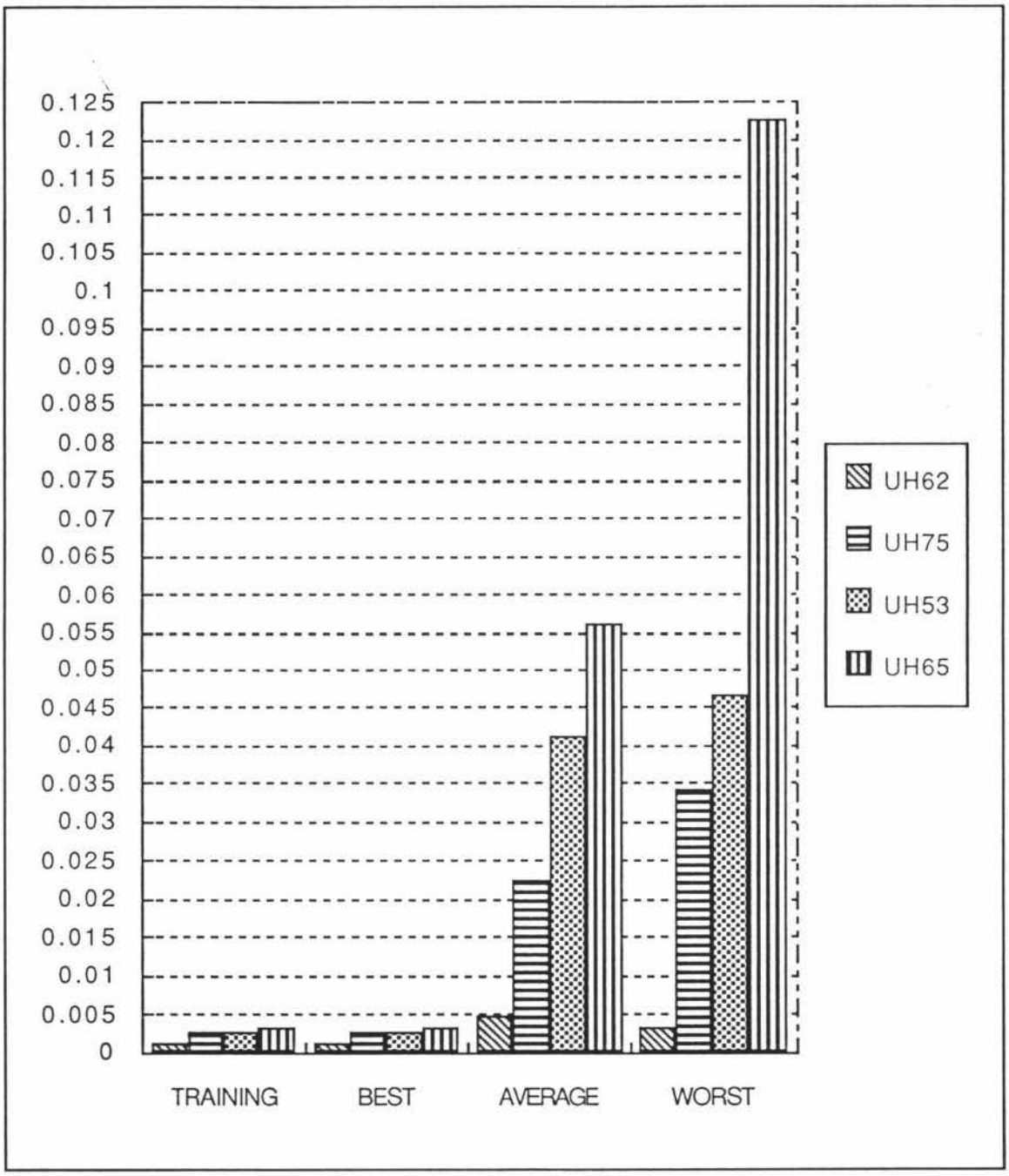

Fig 5.1.2.2.3 Mean-square-errors for heat exchanger sub-models

Figure 5.1.2.2.3 presents the mean-square-prediction error results of the heat exchanger sub-models. As seen in this figure and the best and the worst heat exchanger sub-model prediction graphs, the heat exchanger sub-model performed well in predicting the other plant data runs.

The heat exchanger sub-model was trained with various values of learning rate, learning increment and learning decrement. The final values of 0.1, 1.04 and 0.8 respectively were found to work best. Similarly a momentum value of 0.93 provided the best performance during training.

The heat exchanger neural network sub-model trained with 3 past values of all the model inputs proved to have predicted the output values of other plant runs close to the actual data. The same sub-model when tested with varied number of past inputs except three resulted in high mean-square-errors. The high errors show the deviation of predicted outputs by the sub-model from the actual outputs of the plant run data which indicates a degraded performance of the sub-model with all other number of past inputs except three.

The heat exchanger network sub-model trained with less than 3 hidden neurons tested against it's own trained data and other plant runs showed high mean errors implying the poor learning and prediction ability of the sub-model. The test results of same sub-model trained with more than 3 hidden neurons and tested against it's own trained data showing the least mean error proves the good learning capability of the sub-model and the high errors when tested against other plant data runs show the inability of the sub-model in predicting the other plant data outputs.

After analysing the heat exchanger training and testing results as explained above, the trained sub-models of UH62 and UH65 plant data were used in the development of best and worst composite UHT network models as a part of the model predictive control and Proportional-Integral (PI) systems.

5.1.2.3 Composite neural network model

As shown in previous sections the best trained and tested models were the steam valve sub-model of the UH74 data run and the heat exchanger sub-model of the UH62 data run. The best composite neural network model formed by means of concatenating these two sub-models. Similarly the worst composite model used the trained and tested steam valve and heat exchanger sub-models of the UH73 and UH65 data runs which had proved to be the worst sub-models.

Table 5.1.2.3 shows the mean-square-errors of the best and worst composite network models as a percentage of full-scale output for a one-step-ahead prediction of the actual plant data from all the runs.

ERROR (MSE % FS)	BEST MODELS UH74/UH62	WORST MODELS UH73/UH65
BEST	0.0013	0.0038
AVERAGE	0.0026	0.0051
WORST	0.0038	0.0063

Table 5.1.2.3 Mean-square-errors as a percentage of full scale for composite-network models

In the table shown above, the 'Best', 'Average' and 'Worst' errors show the best, average and worst results respectively in predicting the output data of other plant runs. The bar graph in figure 5.1.2.3 presents the mean-square-errors of the best and worst composite network models as given in table 5.1.2.3. The following section presents the results of the model predictive control and PI control systems using the composite network models.

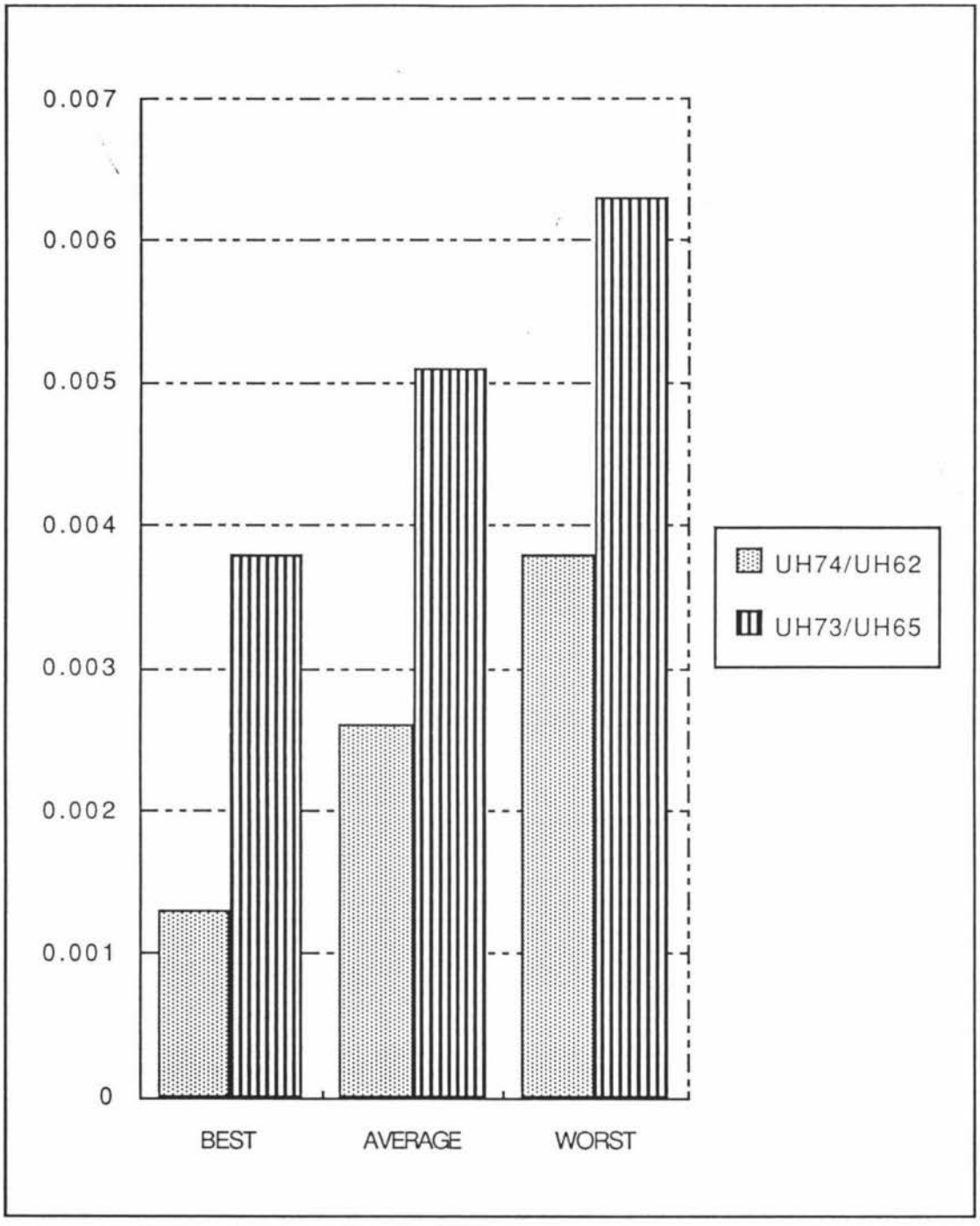

Fig 5.1.2.3 Mean-square-errors for composite-network models

5.2 MPC PERFORMANCE TESTS

The test results of both the developed model predictive control (MPC) systems in a simulated environment are presented in the following sub-section 5.2.1. The developed MPC systems were compared with the simulated system of the proportional-Integral (PI) control system which is currently being used in the UHT plant to evaluate their performance and these comparative results were presented in sections 5.2.2 and 5.2.3. The PI control and MPC systems used the best composite neural network model to simulate the actual UHT plant.

5.2.1 Model Predictive Control Simulations

This section presents the performance results of two types of the developed MPC systems as described in chapter 4. Of the two model predictive control systems, one of them used the best composite network model for both prediction and plant models which represented the ideal MPC system. The other MPC system used the worst composite network model for prediction and best composite network model as plant.

To evaluate the system's performance, both the developed MPC systems were tested for setpoint changes i.e. changes in product temperature exiting the heat exchanger, T_{po} and for disturbance rejection i.e. disturbance in the form of steam inlet pressure, P_{si}.

The figure 5.2.1.1 shows the response of both the MPC systems for a setpoint change i.e. change in temperature of the product exiting the heat exchanger. The solid line indicates the response of the MPC system using the best composite neural network models for both prediction and as plant. The dashed line indicates the MPC system using the worst composite-network model for prediction and best model as plant.

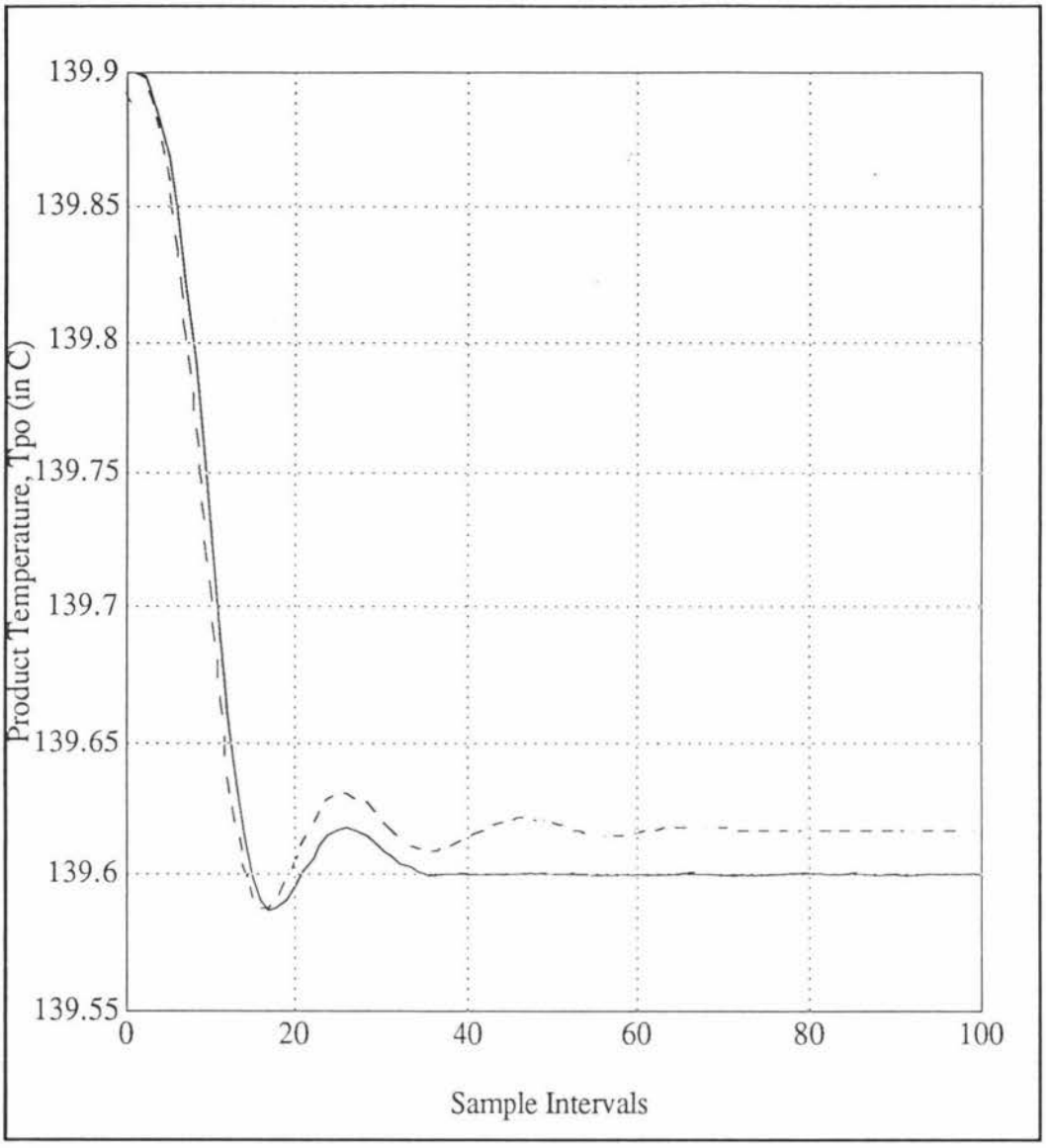

Fig 5.2.1.1 Response of both MPC systems to a setpoint change

Using the test software developed as explained in chapter 4, both the MPC systems were tested for their performance. In the MPCTEST module, provision was made to pass on the required parameters for testing the developed MPC systems.

The performance of the MPC system using the worst prediction model indicates the handling ability of plant/model mismatch. This MPC system gives a more realistic evaluation of the system performance. The response of the UHT system with the MPC controller would be expected to lie between the responses of the above two MPC systems.

At first the setpoint altered by means of a change in the product temperature coming out of the heat exchanger from 139°C. For each incremental change, the system's response was studied in terms of its settling time and its attainment of the targeted setpoint.

From the response as shown in figure 5.2.1.1, the MPC system using the perfect neural network model showed a quickly settled response compared to the MPC system using the worst prediction model. This comparison shows that the ideal model predictive control system does well due to it's perfect prediction model which seem to be a reasonably expected result.

The other important aspect to notice is the sustained offset of the worst MPC control system in attaining the changed final setpoint indicating the imperfect prediction model. Even though this offset may be small in value, due to the precise control requirements of the process itself the offset might not be an acceptable one. Looking into the past applications of model predictive control, inclusion of an integral term will be the best possible solution to overcome this problem.

After evaluating the performance of both ideal and worst MPC systems in terms of setpoint changes, both the systems were tested for a disturbance rejection to steam inlet pressure, Psi.

In evaluating the system's performance for its ability to handle various values of disturbance, the input disturbance vector was changed by means of incrementing and decrementing the steam inlet pressure, P_{si}. The performance test was carried out for each of the changed disturbance vectors and as a part of the evaluation the system's response was studied in terms of its settling time after disturbance and its attainment of the actual setpoint i.e. product temperature exiting the heat exchanger, T_{po}.

The response of both the MPC systems for a disturbance rejection to a specific value of steam inlet pressure, P_{si} was shown in figure 5.2.1.2. The solid line indicates the response of the MPC system using the best composite neural network models for both prediction and as plant. The dashed line indicates the MPC system using the worst composite-network model for prediction and best model as plant.

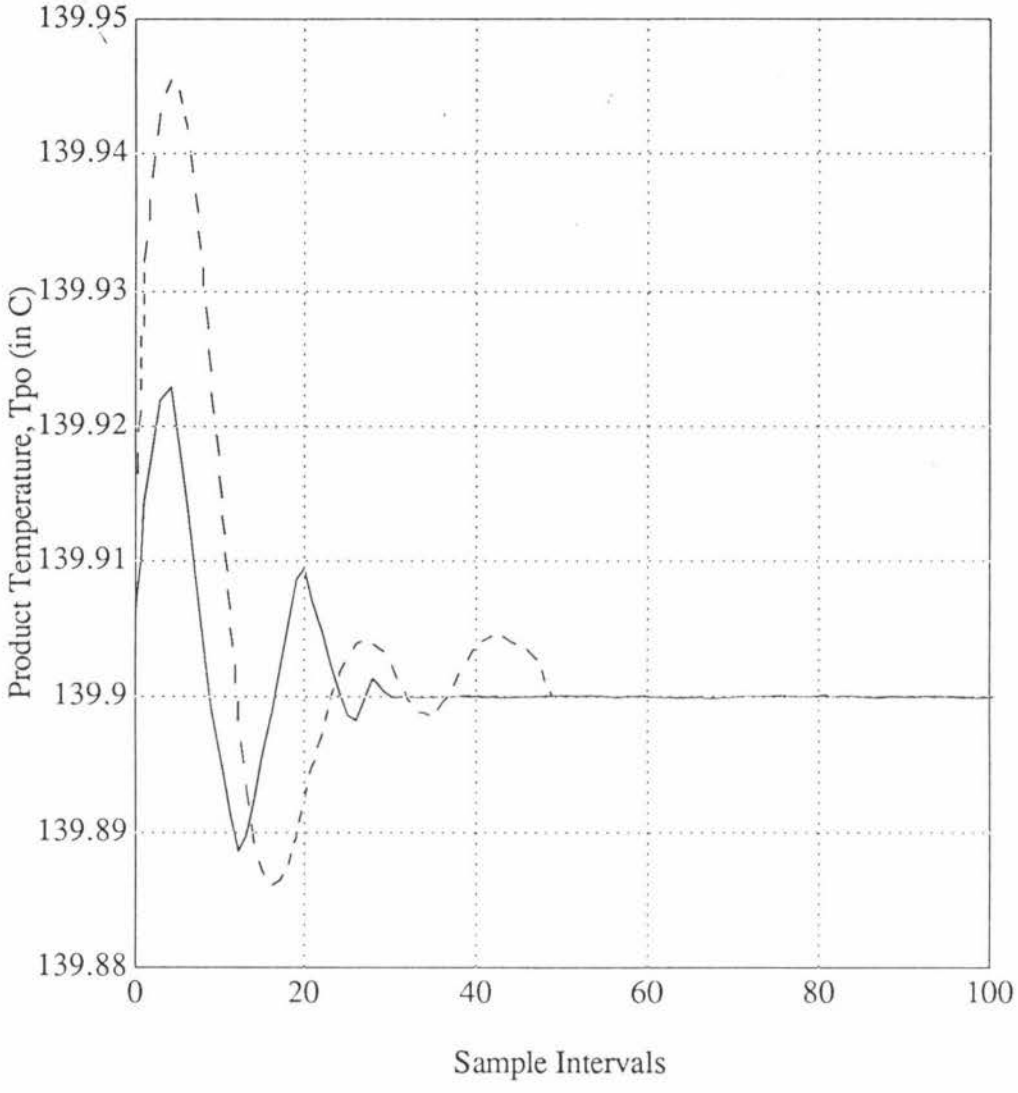

Fig 5.2.1.2 Response of both MPC systems to a inlet steam pressure disturbance

From the above graph as shown in figure 5.2.1.2, it is clearly evident that the ideal MPC system settled quickly with a small and smooth variation in output i.e. change in product temperature exiting the heat exchanger, T_{po}. The worst MPC system took more time to settle down and with a significant overshoot in output.

After studying the responses of both the MPC systems for setpoint changes and disturbance rejection, the focus was turned towards the evaluation of their performance so as to implement this new model predictive controller to the UHT system. For this a PI control system for UHT plant was developed as the existing UHT plant uses the PI controller, and then the MPC systems were compared with the PI control system. The results are discussed in the next sections.

5.2.2 PI Control System Development

A PI control system for the UHT plant was developed for use in a simulation environment as shown in figure 5.2.2. This system was developed using the SIMULINK tool box of MATLAB (a mathematical software package). This UHT plant PI control system uses the best composite neural network model as a plant.

In the PI control system shown below, the UHT system was simulated by means of sending in the plant inputs to the steam valve neural-network sub-model, written as a SIMULINK function. The inputs to this model include the past plant output values of inlet steam pressure, P_{si}, steam valve position, P_{sv} and the hot water temperature entering the loop, T_{hi}. Of these inputs, the values of T_{hi} are the outputs of the steam valve sub-model function and the values of P_{sv} were fed back from the PI controller. Apart from the inputs, the model requires values of inlet steam pressure, P_{si}, which were sent into the model after scaling.

The plant inputs to the next sub-model i.e. the heat exchanger neural network sub-model were the values of T_{hi} which were the outputs from the steam valve sub-model and the past values of product temperature exiting the heat exchanger, T_{po}, being fed back from this heat exchanger sub-model. For simulation purposes, both the sub-models were written as SIMULINK functions in the development of PI control system.

The outputs from the heat exchanger sub-model were again sent in through the PI controller, so as to calculate the required change in the value of steam valve opening, Psv, and this value is again fed back to the system. In this PI control system development, all the inputs to both the neural network sub-model functions were sent in after scaling them using their respective scaling functions so as to convert them from their original values to values in the range of 0.1 to 0.9. Again the outputs from the model functions were converted back to their original values using the respective descale functions.

While testing the performance of the PI control system for setpoint changes, the output change i.e. change in product temperature, T_{po}, was included as an input to the PI controller instead of the output from the heat exchanger sub-model. For disturbance rejection, a vector with specified disturbance values i.e. inlet steam pressure, P_{si}, was included as inputs apart from the actual inputs to the steam valve sub-model function.

The developed PI system response was studied for various changes in setpoint and the ability of the system to reject the inlet steam pressure disturbance. The next section describes the comparative aspects of both the model predictive control systems with the responses of the simulated PI control system of the UHT plant.

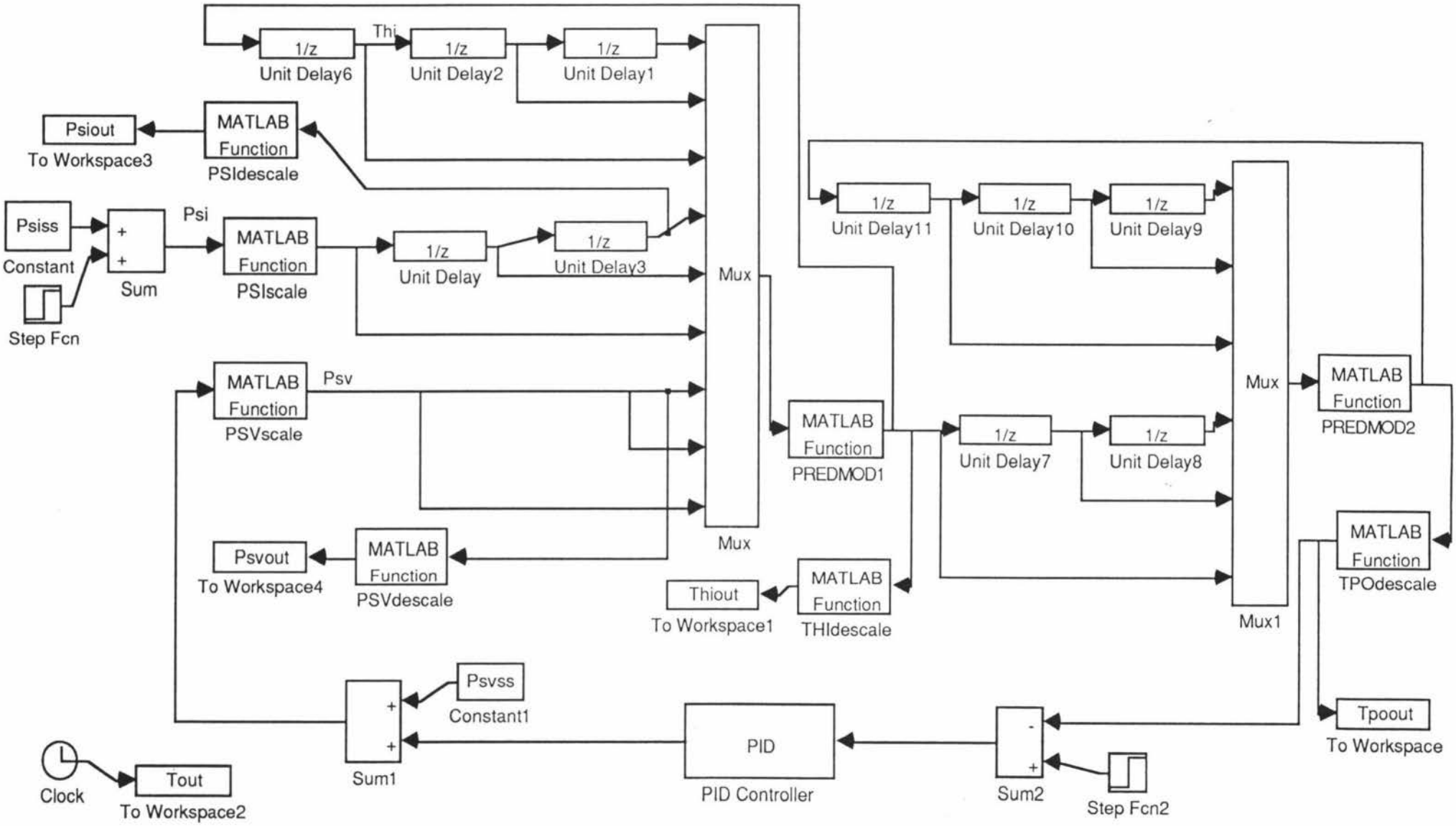

FIGURE 5.2.2 UHT PLANT PI CONTROL SYSTEM (Simulation)

5.2.3 Performance Evaluation of MPC and PI Control Systems

This section explains the performance evaluation of the developed UHT plant model predictive control systems compared to the PI (Proportional-Integral) control system in terms of setpoint tracking and disturbance rejection.

The following figure 5.2.3.1 shows the responses of the ideal and worst MPC systems, and the PI control system for a specific change in the setpoint i.e. change in the product temperature, T_{po}.

In the figure 5.2.3.1, the solid line in the graph indicates the response of the PI system using the best neural network plant model and the dashed line indicates the ideal MPC system's response using the best composite neural network models for both prediction and as plant. The dot-dashed line in the graph indicates the worst MPC system using the worst composite-network model for prediction and the best network model as plant.

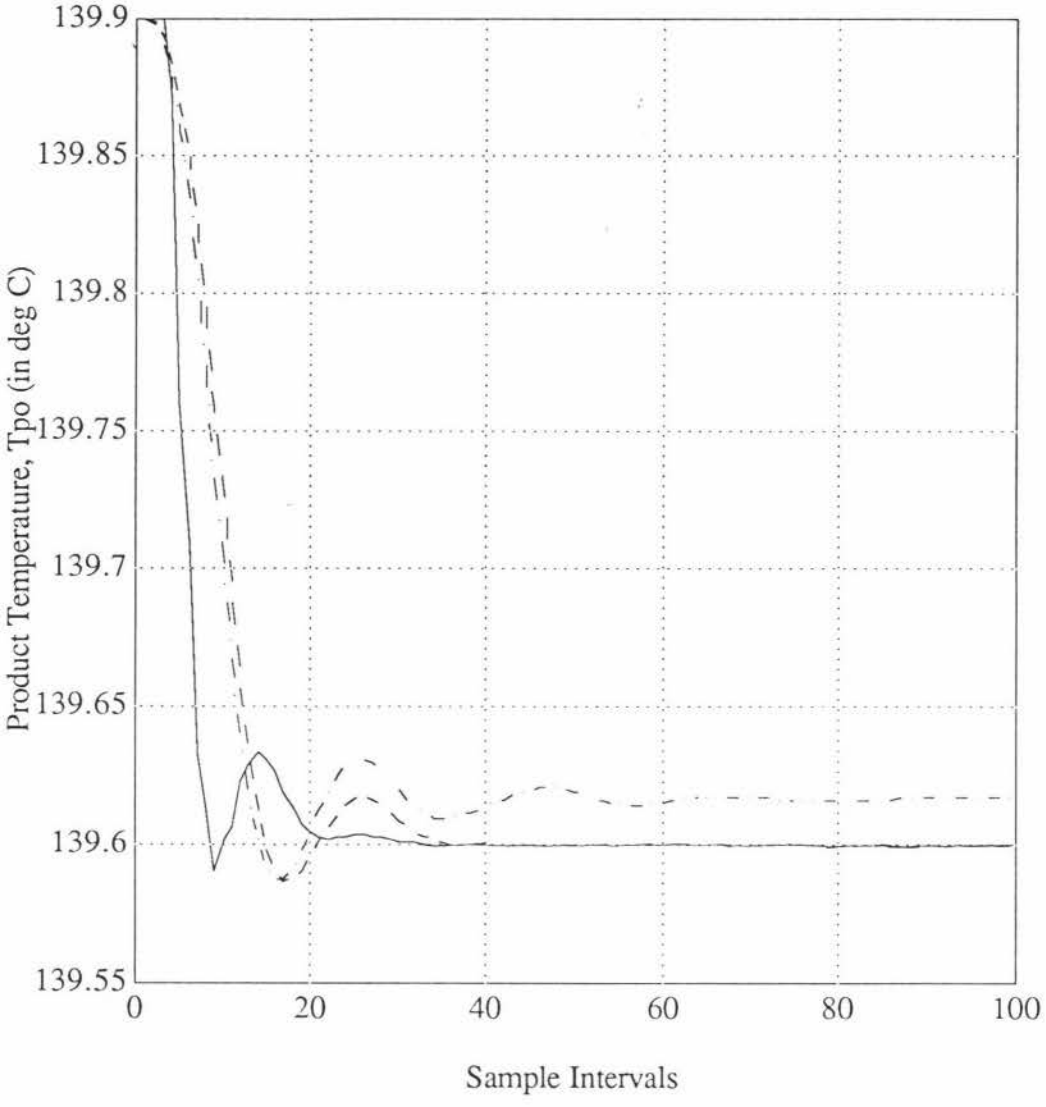

Fig 5.2.3.1 MPC and PI systems response to a setpoint change

In achieving the targeted setpoint the PI control system performed well compared to both the MPC systems as shown in the above figure. The PI control system showed a quick response in attaining the changed setpoint compared to the ideal MPC system even though the ideal MPC system attained the setpoint in a smooth manner.

When compared to the PI control system, the worst MPC system had a poor performance in terms of setpoint attainment as it's response resulted in a sustained offset from the targeted setpoint. This offset is due to the use of a imperfect prediction model in the worst MPC system.

The other reason for this sustained offset in PI system's response could be due to the optimiser. During the performance evaluation tests, the starting estimate of the control vector was taken from the unimplemented values of the previous solution. This might have lead to the optimiser being locked into a local minimum. Tests using a randomised initial control vector at each step were carried out so as to avoid the optimiser getting stuck in a local minima. This resulted in a remarkable performance improvement which suggests that the initial control vector randomisation as the possible solution for the optimiser problem. However randomising the initial control vector tripled the time required for the optimisation which seemed to be a severe penalty as the MPC controller is supposed to be running in real time.

The responses of the ideal and worst MPC systems, and the PI control system to a specified disturbance i.e. to a specific value of inlet steam pressure, P_{si} were shown in the figure 5.2.3.2.

In the figure shown below, the solid line in the graph indicates the response of the PI system using the best neural network plant model and the dashed line indicates the ideal MPC system's response. The dot-dashed line in the graph indicates the worst MPC system's response.

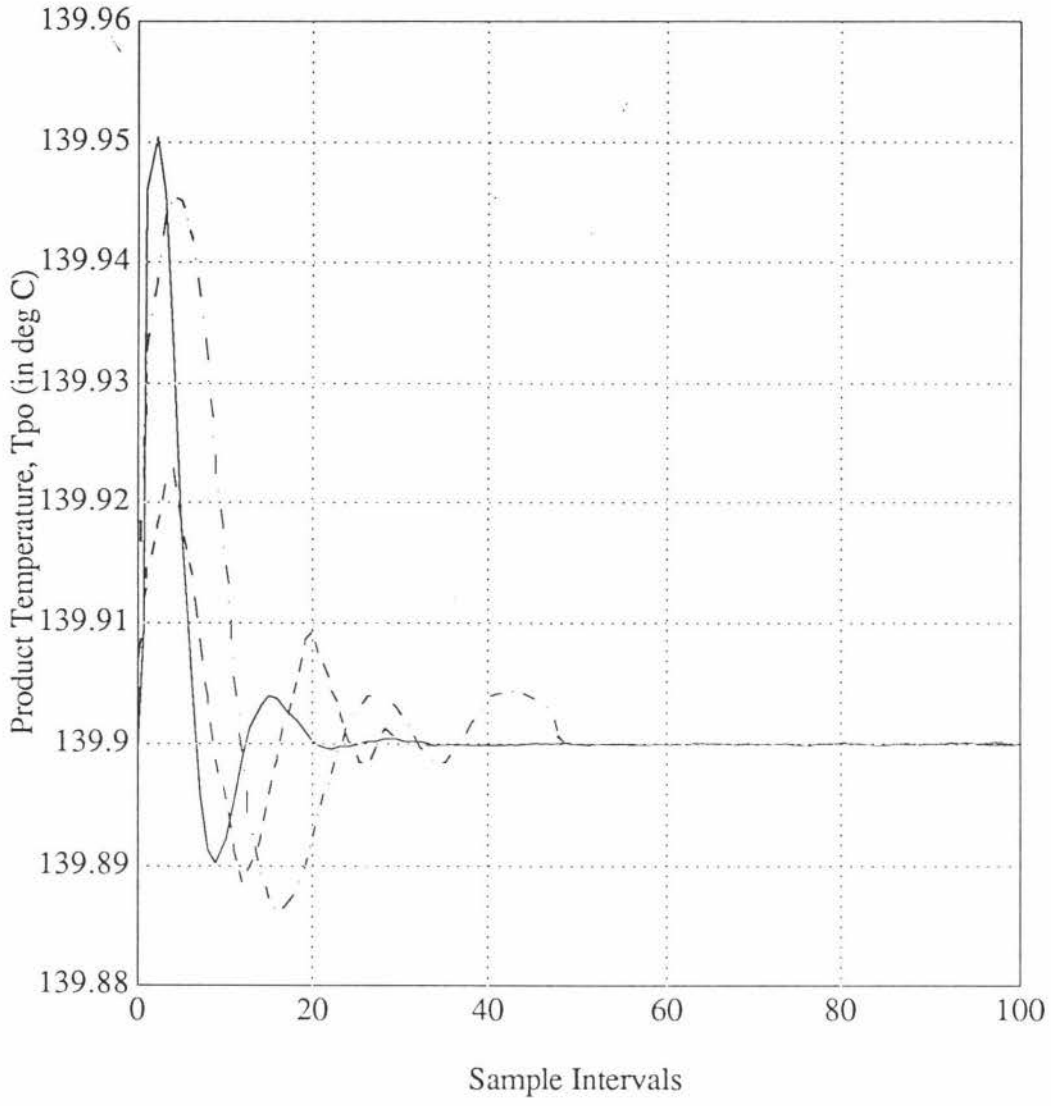

Fig 5.2.3.2 MPC and PI systems response for disturbance rejection

The PI control system performed better in comparison with both the MPC systems in terms of disturbance rejection as shown in the above figure. The PI control system settled down quickly from its initial departure from the setpoint after a disturbance to the system compared to the settling time for both the ideal and worst MPC systems. But the response of the PI control system shows a large initial departure from the setpoint compared to either of the model predictive control systems.

From the figure 5.2.3.1, it is clearly evident that the response to a setpoint change under the PI controller is not particularly smooth. This suggests that the simulation model is non-linear.

During the course of the simulations it became apparent that the long-term response of the different neural network models was significantly different despite having similar errors during training and testing. This lead to a reasonable conclusion that the one-step-ahead prediction may not be a valid method for training artificial neural network models for use in n-step-ahead predictions as part of a model predictive control scheme. Apart from this, the one-step-ahead prediction may not even be good for ranking the neural network models.

The existing trained composite neural network models of UHT system showed to be predicting three steps ahead against their own plant data sets which seemed to be reasonable. This research showed that the training and testing of the neural networks require enormous computing time and also the requirement of computing resources to proceed with the training simultaneously for various models which became the constraints for further research within the specified time.

CHAPTER SIX

CONCLUSIONS AND RECOMMENDATIONS

This chapter presents the conclusions drawn from this research work. The conclusions include the selection of the training methodology.

Section 6.1 of this section describes the conclusions and section 6.2 of this chapter presents the recommendations for any future work in this area.

6.1 CONCLUSIONS

The UHT process is commonly used for the sterilisation of milk. The UHT pilot plant which was looked at uses a PI control system. As this existing control system could not provide accurate and efficient control, it was decided to look at other intelligent control alternatives and in particular at the application of artificial neural networks.

For any process modelling, the analysis of plant data is an important aspect which provides the information regarding the relationships between input and output variables of the process. After collecting the required data from a number of UHT plant runs and data analysis, the output and input plant data required for modelling was chosen. Then each of the plant run data was scaled to lie between 0.1 and 0.9 and patterned into a form suitable for neural network training.

After deciding the network training topology, single neural network model was developed using each of the patterned plant run data to represent the complete UHT process and tested for its ability to predict the unknown plant data of all the other available plant runs. The single network model showed it's inability in predicting the other plant run data.

The UHT process was then divided into two parts for modelling purpose. Two neural network sub-models namely the steam valve and heat exchanger models were developed and trained for each of the available plant data runs. Both the sub-models were tested for their ability in predicting unknown output data of other plant runs. Both the sub-models showed positive results as explained in the earlier chapter.

A composite network model was then formed by the concatenation of the above two sub-models to represent the UHT process. The composite models developed using data from various plant runs were tested against each other for their output prediction ability. A best composite network model was formed using the best steam valve and heat exchanger sub-models and similarly worst composite model comprises the combination of both the worst sub-models.

Two model predictive control (MPC) systems were developed using the best UHT plant composite network model as plant. One of the MPC systems uses the best composite model for prediction and the other uses the worst model. Both the MPC systems were evaluated for their performance in terms of setpoint tracking and disturbance rejection. These performance results were compared with the PI control system which uses the neural network composite model as the plant in simulated environment.

The conclusions drawn from this case study are:

1. The computation effort required is large for training the single neural network model as compared to the composite network model.

2. A composite neural network model of the UHT treatment plant was more easily trained in terms of computation time and more accurate in its predictions than a single network model.

3. A model predictive controller using a perfect prediction composite network model was found to have better disturbance rejection and setpoint tracking than that using the worst prediction composite model.

4. A well-tuned PI (proportional-Integral) controller settled more quickly than either of the model predictive control systems although the initial departure was large.

5. A sustained offset in one model predictive control system simulation suggested that the optimisation algorithm was becoming trapped in a local minimum.

6.2 RECOMMENDATIONS

This section presents the recommendations for any future work in the application of neural networks in process control applications.

The recommendations are:

1. To avoid the optimisation algorithm being trapped in a local minimum during model predictive control system simulation, the random initial vectors be chosen at each time step despite the large increase in computation time this produces.

2. The use of one-step-ahead, two-step-ahead etc. prediction in neural network training was recommended.

CHAPTER SEVEN

REFERENCES AND BIBLIOGRAPHY

This chapter presents the details of all the reference material used during this project work which includes the journal and conference papers and text books.

The Section 7.1 lists all the journal and conference papers and the text books referred directly towards this work. Section 7.2 consists of all the papers and books referred to in the papers listed in section 7.1.

7.1 REFERENCES

Beale, R., Jackson, T., Neural Computing: An Introduction, Bristol, U.K, Adam Higer, 1990.

Burton, H., Ultra High Temperature Processing of Milk Products, 1990

Chessari, C. J., Barton, G. W., "Model Based Control Using Neural Network Models", IEE Int. Conf. Control 92, November 1992.

Chessari, C. J., Barton, G. W., "On The Use of Neural Networks in Process Modelling", IEE Int. Conf. Control 92, November 1992.

Chitra, S. P., "Use of Neural Networks for Problem Solving", Chemical Engineering Progress, pp 44 - 52, April 1993.

Donat, J. S., Bhat, N., and McAvoy, T. J., "Neural Net Based Model Predictive Control", Int. Control, Vol. 54, No. 6, pp 1453 - 1468, 1991.

Garcia, C. E., Prett, D. M. and Morari, M., "Model Predictive Control: Theory and Practice - A Survey", Automatica, Vol. 25, pp 335 - 348, 1989.

Gelrmino, M. S., Ricker, N. L., "Model-Predictive Control of a Combined Sewer System", Int. J. Control, Vol. 59, No. 3, pp 793 - 816, 1994.

Hunt, K. J, Sbarbaro, D., "Neural Networks for Nonlinear Internal Model Control", IEE Proceedings-D, Vol. 138, No. 5, 1991

Karla, V. R., Bakker, H. H. C., "Neural-network-based Model Predictive Control: A Case Study", Proceedings of the Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, ANNES'95, Dunedin, New Zealand, November 20-23, 1995.

Karla, V. R., Bakker, H. H. C., "Application of Artificial Neural Networks in Process Modelling and Control", Australian Journal of Intelligent Information Processing Systems AJIIPS, Vol. 3, No. 2, pp 70 -75, Winter (June) 1996.

Lee, J. H., Morari, M., and Garcia, C. E., "State-space Interpretation of Model Predictive Control", Automatica, Vol. 30, No. 4, pp 707 - 717, 1994.

Lee, M., Park, S., "A New Scheme Combining Neural Feedforward Control with Model-Predictive Control", AIChE Journal, Vol. 38, No. 2, pp 193 - 200, February 1992.

Morris, A. J., Montague, G. A., and Willis, M. J., "Artificial Neural Networks: Studies in Process Modelling and Control", Trans IChemE, Vol. 72, Part A, pp 3- 19, January 1994.

Ogata, K., Modern Control Engineering, Second Edition, Prentice-Hall, USA, 1992.

Pao, Y. A., Phillips, S. M., and Sobajic, D. J., "Neural Net Computing and the Intelligent Control Systems", Int. J. Control, Vol. 56, No. 2, pp 263 - 289, 1992.

Psichogios, D. C., Ungar. L. H., "Non Linear Internal Model Control and Model Predictive Control Using Neural Networks", IEEE Proceedings, 1990.

Richalet, J., Rault, A., Testud, J. L., and Papon, J., "Model Predictive Heuristic Control: Applications to Industrial Processes", Automatica, Vol. 14, pp 413 - 428, 1978.

Rouhani, R., Mehra, R. K., "Model Algorithmic Control (MAC); Basic Theoretical Properties", Automatica, Vol. 18, No. 4, pp 401 - 414, 1982.

Rumelhart, D., McClelland, J., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Cambridge, MA, MIT press, 1987.

Sistu, P. B., Bequette, B. W., "Nonlinear Predictive Control of Uncertain Processes: Application to a CSTR", AIChE Journal, Vol. 37, No. 11, pp 1711 - 1723, November 1991.

Sorsa, T., Koivo, H. N., "Application of Artificial Neural Networks in Process Fault Diagnosis", Automatica, Vol. 29, No. 4, pp 843 - 849, 1993.

Thompson, M. L., Kramer, M. A., "Modelling Chemical Processes Using Prior Knowledge and Neural Networks", AIChE Journal, Vol. 40, No. 8, pp 1328 - 1340, August 1994.

Widrow, B., Lehr, M. A., "30 years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation", IEEE Proceedings, Vol. 78, No. 9, September 1990.

Willis, M. J., Di massimo, C., Montague, G. A., Tham, M. T., and Prof Morris, A. J., "Artificial Neural Networks in Process Engineering", IEE Proceedings, Vol. 138, No. 3, pp 256 - 265, 1991.

Willis, M. J., Montague, g. A., Morris, A. J., "Modelling of Industrial Processing Using Artificial Neural Networks", Computing and Control Engg Journal, pp 113 - 116, May 1992.

7.2 BIBLIOGRAPHY

Bhat, N., Minderman, P., McAvoy, T., and Wang, N., "Modelling Chemical Process Systems via Neural Computation", Proc. 3rd Int. Symp. 'Control for Profit', Newcastle-upon-Tyne, 1989a

Bhat, N., Minderman, P., and McAvoy, T. J., "Use of Neural Nets for Modelling of Chemical Process Systems", Preprints IFAC Symp. Dycord+89, Maastricht, The Netherlands, pp 147 - 153, August 21-23, 1989b.

APPENDICES

The section 8.1 of these appendices contains all the software files related to pattern data preparation which includes the data scaling and patterned data printing. The neural network training software for the steam valve and heat exchanger sub-models, and single network model were written using Matlab and the associated toolboxes are presented in section 8.2.

Section 8.3 includes the software related to the testing of other plant data runs using the weights and biases of neural network trained models. This includes the testing software for the single network model, steam valve and heat exchanger sub-models, and the composite network model. All the software related to the model predictive control system development and testing using the composite network trained model is presented in section 8.4.

The section 8.5 consists of all the sum-squared and mean-square error tables for the steam valve and heat exchanger sub-models, and the single network model.

The software files 8.3.7 and 8.3.8 in the section 8.3 are the contributions from Dr Huub Bakker of Massey University.

8.1 PATTERN FILES

8.1.1 'msprintf.m'

```
function y = msprintf(forstr,x)
% Y = MSPRINTF(FORSTR,X)
%
% This function returns the matrix X in the form of row vector, Y.
% Elements in a row are concatenated into one text string.
% Separate rows are converted to separate rows in Y.
%
[rows,columns] = size(x);
width = size(sprintf(forstr,x(1,1)));
width = width(2);
y = ones(1,width*columns);
for i = 1:rows
 str = '';
 for j =1:columns
       str = [str sprintf(forstr,x(i,j))];
 end
disp(size(str))
%disp(str)
       y(i,:) = str;
end
end
```

8.1.2 'psvscale.m'

```
function y = psvscale(x,range)
% Y = PSVSCALE(X,RANGE)
%
% Scales elements of the vector X into the range 0.1 to RANGE.
% The vector X for this function consists of all the actual data of plant steam
% valve opening, Psv (in percentage %) from a specific run data of UHT plant.
%
maximum = 36.75;
% This value is the maximum Psv value of all the plant data runs.
minimum = 17.29;
% This value is the minimum Psv value of all the plant data runs.
%
y =  ((x-minimum)*(range-0.1)/(maximum-minimum))+0.1;
end
```

8.1.3 'psiscale.m'

```
function y = psiscale(x,range)
% Y = PSiSCALE(X,RANGE)
%
% Scales elements of the vector X into the range 0.1 to RANGE.
% The vector X for this function consists of all the actual data of steam inlet
% pressure, Psi ( in psi) from a specific run data of UHT plant.
%
maximum = 7.606;
% This value is the maximum Psi value of all the plant data runs.
minimum = 6.678;
% This value is the minimum Psi value of all the plant data runs.
%
y =  ((x-minimum)*(range-0.1)/(maximum-minimum))+0.1;
end
```

8.1.4 'thiscale.m'

```
function y = thiscale(x,range)
% Y = THISCALE(X,RANGE)
%
% Scales elements of the vector X into the range 0.1 to RANGE.
% The vector X for this function consists of all the actual data of hot water
% temperature entering the loop, Thi ( in C) from a specific run data of UHT
% plant.
%
maximum = 150.62;
% This value is the maximum Thi value of all the plant data runs.
minimum = 120.55;
% This value is the minimum Thi value of all the plant data runs.
%
y =  ((x-minimum)*(range-0.1)/(maximum-minimum))+0.1;
end
```

8.1.5 'tposcale.m'

```
function y = tposcale(x,range)
% Y = TPOSCALE(X,RANGE)
%
% Scales elements of the vector X into the range 0.1 to RANGE.
% The vector X for this function consists of all the actual data of product
% temperature exiting the heat exchanger, Tpo ( in C) from a specific run data
% of UHT plant.
%
maximum = 146.81;
% This value is the maximum Thi value of all the plant data runs.
minimum = 135.51;
% This value is the minimum Thi value of all the plant data runs.
%
y =  ((x-minimum)*(range-0.1)/(maximum-minimum))+0.1;
end
```

8.1.6 'svmodpat.m'

```
function svmodpat(z,ninputs,filename)
% SVMODPAT(Z, NINPUTS, FILENAME)
%
% This function Produces a pattern file for a MISO (multi-input single-
% output) steam valve neural network sub-model of UHT plant.
% The patterns are written to the FILENAME.
% The matrix Z contains the actual input and output data of one UHT plant
% run.
% NINPUTS indicates the number of previous samples to be used as inputs for
% the neural network model.
%
% This file scales the elements of one variable formed as a column vector from
% the input output matrix of Z.
% Depending upon the type of variable like steam valve position, Psv or the hot
% water temperature entering the loop, Thi, the related scaling function scales
% the data to lie between 0.1 and 0.9 before writing them out to pattern file.
%
% The column 19 of the matrix Z consists of all the actual steam inlet pressure,
% Psi data.
% The column 18 of the matrix Z consists of all the actual steam valve position,
% Psv data.
% The column 9 of the matrix Z consists of all the actual hot water temperature
% entering the loop, Thi data.
%
clear;
fstr = '%7.3f';
%
size(z); % Gives the size of the matrix Z
a =  psiscale(z(:,19));
b =  psvscale(z(:,18));
c = thiscale(z(:,9));
%
clear z; % Clears the data of Z matrix
%
[rows,columns]=size(a);
```

```
%
% Generates the pattern numbers according to the size of column vector a (b or
% c can also be used)
for row=ninputs+1:rows
  if row/50==fix(row/50), disp(row);end
    patno = sprintf('\n%4.0f',row);
%
% Writes the scaled inputs and output of the model into pattern file
   in1 = msprintf(fstr,a(row-ninputs+1:row)');
   in2 = msprintf(fstr,b(row-ninputs+1:row)');
   out = msprintf(fstr,c(row-ninputs:row-1)');
%
% Writes the targets(Thi at k th step) for the model to the file
  tar = sprintf(fstr,c(row));
   fprintf(filename, [patno out in1 in2  tar]);
end
%
% EXAMPLE with ninputs = 3
%
% input-output vector
% [Thi(k-1),Thi(k-2),Thi(k-3),Psi(k),Psi(k-1),Psi(k-2),
%  Psv(k),Psv(k-1),Psv(k-2)]
%
% target
% Thi(k)
```

8.1.7 'hemodpat.m'

```
function hemodpat(z,ninputs,filename)
% HEMODPAT(Z, NINPUTS, FILENAME)
%
% This function Produces a pattern file for a MISO (multi-input single-output)
% heat exchanger neural network sub-model of UHT plant.
% The patterns are written to the FILENAME.
% The matrix Z contains the actual input and output data of one UHT plant
% run.
% NINPUTS indicates the number of previous samples to be used as inputs for
% the neural network model.
%
% This file scales the elements of one variable formed as a column vector from
% the input output matrix of Z.
% Depending upon the type of variable like the hot water temperature entering
% the loop, Thi or the product temperature exiting the heat exchanger, Tpo, the
% related scaling function scales the data to lie between 0.1 and 0.9 before
% writing them out to the pattern file.
%
% The column 7 of the matrix Z consists of all the actual product temperature
% exiting the heat exchanger, Tpo data.
% The column 9 of the matrix Z consists of all the actual hot water temperature
% entering the loop, Thi data.
%
clear;
fstr = '%7.3f';
%
size(z); % Gives the size of the matrix Z
a =  thiscale(z(:,9));
b =  tposcale(z(:,7));
%
clear z; % Clears the data of Z matrix
[rows,columns]=size(a);
%
% Generates the pattern numbers according to the size of column vector a
% (vector b can also be used).
```

```
for row=ninputs+1:rows
 if row/50==fix(row/50), disp(row);end
   patno = sprintf('\n%4.0f',row);
%
% Writes the scaled input and outputs of the model into pattern file
   in = msprintf(fstr,a(row-ninputs+1:row)');
   out = msprintf(fstr,b(row-ninputs:row-1)');
%
% Writes the targets (Tpo at k th step) for the model to the file
   tar = sprintf(fstr,c(row));
   fprintf(filename, [patno out in tar]);
end
%
% EXAMPLE with ninputs = 3
%
% input-output vector
% [Tpo(k-1),Tpo(k-2),Tpo(k-3),Thi(k),Thi(k-1),Thi(k-2)]
%
% target
% Tpo(k)
```

8.1.8 'simodpat.m'

```
function simodpat(z,ninputs,filename)
% SIMODPAT(Z, NINPUTS, FILENAME)
%
% This function Produces a pattern file for a MIMO (multi-input multi-output)
% single neural network model of UHT plant.
% The patterns are written to the FILENAME.
% The matrix Z contains the actual input and output data of one UHT plant
% run.
% NINPUTS indicates the number of previous samples to be used as inputs for
% the neural network model.
%
% This file scales the elements of one variable formed as a column vector from
% the input output matrix of Z.
% Depending upon the type of variable like steam valve position, Psv, or the
% hot water temperature entering the loop, Thi, the related scaling function
% scales the data to lie between 0.1 and 0.9 before writing them out to the
% pattern file.
%
% The column 19 of the matrix Z consists of all the actual steam inlet pressure,
% Psi, data.
% The column 18 of the matrix Z consists of all the actual steam valve position,
% Psv data.
% The column 9 of the matrix Z consists of all the actual hot water temperature
% entering the loop, Thi data.
% The column 7 of the matrix Z consists of all the actual product temperature
% exiting the heat exchanger, Tpo data.
%
clear;
fstr = '%7.3f';
%
size(z); % Gives the size of the matrix Z
a =  psiscale(z(:,19));
b =  psvscale(z(:,18));
c = thiscale(z(:,9));
d = tposcale(z(:,7));
```

```
%
clear z; % Clears the data of Z matrix
%
[rows,columns]=size(a);
%
% Generates the pattern numbers according to the size of column vector a
% (vectors b, c or d can also be used)
for row=ninputs+1:rows
  if row/50==fix(row/50), disp(row);end
    patno = sprintf('\n%4.0f',row);
%
% Writes the scaled input and outputs of the model into pattern file
    in1 = msprintf(fstr,a(row-ninputs+1:row)');
    in2 = msprintf(fstr,b(row-ninputs+1:row)');
    in3 = msprintf(fstr,c(row-ninputs+1:row)');
    out = msprintf(fstr,d(row-ninputs:row-1)');
%
% Writes the targets(Tpo at k th step) for the model to the file
    tar = sprintf(fstr,d(row));
    fprintf(filename, [patno out in1 in2  in3 tar]);
end
%
% EXAMPLE with ninputs = 3
%
% input-output vector
% [Tpo(k-1),Tpo(k-2),Tpo(k-3),Psi(k),Psi(k-1),Psi(k-2),
%  Psv(k),Psv(k-1),Psv(k-2),Thi(k),Thi(k-1),Thi(k-2)]
%
% target
% Tpo(k)
```

8.2 NEURAL NETWORK MODEL TRAINING AND TESTING FILES

8.2.1 'svmodtrn.m'

```
% Steam valve sub-model neural network training function.
%
% This function trains a neural network, consisting of 3 past values of two
% previous inputs (steam inlet pressure, Psi and steam valve position, Psv) and 3
% past values of output (hot water temperature entering heat exchanger loop,
% Thi).
%
% The input and output values are fed to a two layer network in which layer 1
% consists of 3 tan-sigmoid neurons, while layer 2 consists of one logsig
% neuron.
% This training uses back propagation with momentum and an adaptive
% learning rate.
%
clg
% Clears any existing graph
%
hold off
axis('normal')
pausetime = 0;
clear;
%
% Matrix P defines the input and output data for network training from the
% pattern data file (i.e. DATFILE1) of the network to be trained.
load DATFILE1;
P=DATFILE1(:,2:10)';
%
% Row vector T defines the associated targets (i.e. values of Thi) of the
% network.
T=DATFILE1(:,11)';
clear DATFILE1;
```

```
% Initialising the Network
% Setting up the Input vector size R for training, the training  batch size Q, and
% the number of neurons in each training layer, S1 for layer 1 and S2 for layer
% 2 (value for S1 is taken as 3).
[R,Q] = size(P)
S1 = 3
[S2,Q] = size(T)
%
% Initialise weights and biases only in case of this model using the same pattern
% data being trained before for specific number of epochs and the weights and
% biases data stored in a file.
% Loading the past weights and biases resulted from the model being trained
% for 500 epochs.
% load svmod1.w1;
%  W10=W1;
%  B10=B1;
%  W20=W2;
%  B20=B2;
% clear svmod1.w1;
%
% Creating the weights and biases using the previous weights and biases for the
% network model training continuation for the rest of the epochs as specified
% in the training parameters.
[W10,B10] = NWTAN(S1,R);
W20 = rands(S2,S1)*0.5;
B20 = rands(S2,1)*0.5;
%
% NETWORK TRAINING
%
% Defining the Training Parameters
% Defines the display interval for all the graphs in terms of number of epochs.
 disf = 50;
% Defines the number of training epochs ( only 500 epochs taken each time
% for training upto a total of 8000 epochs for a better training check for
% network learning).
trep = 500;
% defines the maximum converging error goal
```

```
erg = 0.001;
% defines the learning rate
lr = 0.1;
% defines the learning increment
lr_in = 1.04;
% defines the learning decrement
lr_de = 0.8;
% defines the momentum
mom = 0.95;
% defines the error ratio
errat = 1.04;
%
% All the training parameters being placed in a row vector TP.
 TP = [disf trep erg lr lr_in lr_de mom erat];
%
% Passing on all the parameters which includes the previous weights and biases,
% input output matrix and target vectors formed from the pattern data of the
% model to be trained, and the training parameters vector to trainbpx function
% as input arguments.
% This function will return weights and biases as the training progress.
 [W1,B1,W2,B2,epoch,TR]= trainbpx(W10,B10,'tansig',W20,B20,'logsig',P,T,TP);
 TR(:,epoch)
%
% The oupt (output) resulting from the calculations using the returned weights
% and biases of the network model training gives the network model prediction
% of the targets.
oupt = logsig(W2*tansig(W1*P,B1),B2);
%
% Gives the number of training patterns.
tpn = [1:Q]';
%
% Gives the error calculated as the difference between the target data and
% output prediction result from model training.
err = T-oupt;
%
% sse gives the sum-squared-error.
 sse=sumsqr(err);
```

```
 fprintf('\n S.S.E is %8.4f',sse);
 fprintf('\n Mean Error is %8.4f\n',sse/Q);
%
% Gives the error result after each epoch as the training progress.
 fprintf('Error at %4.0f epoch is %8.4f\n',epoch,TR(1,epoch));
%
% Plotting four graphs one each for output and target comparision, error,
% learning rate, and the sum squared error of the model as training progress.
%
clg,
 subplot(221),plot(tpn,[oupt;T],'--'),xlabel('training patterns'),
 ylabel('output & Target'),title('pattterns Vs output & targets');
 subplot(222),plot(tpn,T-oupt),xlabel('training patterns'),
 ylabel('error'),title('patterns Vs error');
 subplot(223),plot(1:epoch,TR(2,1:epoch)),xlabel('number of epochs'),
 ylabel('learning rate'),title('epochs Vs learning rate');
 subplot(224),plot(1:epoch,TR(1,1:epoch)),xlabel('number of epochs'),ylabel
('sum squared error'),title('epochs Vs sum squared error');
%
% Saving the returned weights and biases of the model after training.
 save svmod1.w2;
 end
```

8.2.2 'hemodtrn.m'

```
% Heat exchanger sub-model neural network training function.
%
% This function trains a neural network, consisting of 3 past values of the
% previous input (hot water temperature entering heat exchanger loop, Thi)
% and 3 past values of previous output (product temperature exiting the heat
% exchanger, Tpo).
%
% The input and output values are fed to a two layer network in which layer 1
% consists of 3 tan-sigmoid neurons,while layer 2 consists of one logsig
% neuron.
% This training uses back propagation with momentum and an adaptive
% learning rate.
%
clg
% Clears any existing graph
%
hold off
axis('normal')
pausetime = 0;
clear;
%
% Matrix P defines the input and output data for network training from the
% pattern data file (i.e. DATFILE2) of the network to be trained.
load DATFILE2;
P=DATFILE2(:,2:7)';
%
% Row vector T defines the associated targets (i.e. values of Tpo) of the
% network.
T=DATFILE2(:,8)';
clear DATFILE2;
%
% Initialising the Network
% Setting up the Input vector size R for training, the training  batch size Q, and
% the number of neurons in each training layer, S1 for layer 1 and S2 for layer
% 2 (value for S1 is taken as 3).
```

```
[R,Q] = size(P)
S1 = 3
[S2,Q] = size(T)
%
% Intialize weights and biases only in case of this model using the same pattern
% data being trained before for specific number of epochs and the weights and
% biases data stored in a file.
% Loading the past weights and biases resulted from the model being trained
% for 500 epochs.
% load hemod2.w1;
% W10=W1;
% B10=B1;
% W20=W2;
% B20=B2;
% clear hemod2.w1;
%
% Creating the weights and biases using the previous weights and biases for the
% network model training continuation for the rest of the epochs as specified
% in the training parameters.
[W10,B10] = NWTAN(S1,R);
W20 = rands(S2,S1)*0.5;
B20 = rands(S2,1)*0.5;
%
% NETWORK TRAINING
%
% Defining the Training Parameters
% Defines the display interval for all the graphs in terms of number of epochs.
 disf = 50;
% Defines the number of training epochs ( only 500 epochs taken each time
% for training upto a total of 8000 epochs for a better training check for
% network learning).
trep = 500;
% defines the maximum converging error goal
erg = 0.001;
% defines the learning rate
lr = 0.1;
%
```

```
% defines the learning increment
lr_in = 1.04;
% defines the learning decrement
lr_de = 0.8;
% defines the momentum
mom = 0.95;
% defines the error ratio
errat = 1.04;
%
% All the training parameters being placed in a row vector TP.
 TP = [disf trep erg lr lr_in lr_de mom erat];
%
% Passing on all the parameters which includes the previous weights and biases,
% input output matrix and target vectors formed from the pattern data of the
% model to be trained, and the training parameters vector to trainbpx function
% as input arguments.
% This function will return weights and biases as the training progress.
 [W1,B1,W2,B2,epoch,TR]= trainbpx(W10,B10,'tansig',W20,B20,'logsig',P,T,TP);
 TR(:,epoch)
%
% The oupt (output) resulting from the calculations using the returned weights
% and biases of the network model training gives the network model prediction
% of the targets.
oupt = logsig(W2*tansig(W1*P,B1),B2);
%
% Gives the number of training patterns.
tpn = [1:Q]';
%
% Gives the error calculated as the difference between the target data and
% output prediction result from model training.
err = T-oupt;
%
% sse gives the sum-squared-error.
 sse=sumsqr(err);
 fprintf('\n S.S.E is %8.4f',sse);
 fprintf('\n Mean Error is %8.4f\n',sse/Q);
%
```

```
% Gives the error result after each epoch as the training progress.
 fprintf('Error at %4.0f epoch is %8.4f\n',epoch,TR(1,epoch));
%
% Plotting four graphs one each for output and target comparision, error,
% learning rate, and the sum squared error of the model as training progress.
%
clg,
 subplot(221),plot(tpn,[oupt;T],'--'),xlabel('training patterns'),
 ylabel('output & Target'),title('pattterns Vs output & targets');
 subplot(222),plot(tpn,T-oupt),xlabel('training patterns'),
 ylabel('error'),title('patterns Vs error');
 subplot(223),plot(1:epoch,TR(2,1:epoch)),xlabel('number of epochs'),
 ylabel('learning rate'),title('epochs Vs learning rate');
 subplot(224),plot(1:epoch,TR(1,1:epoch)),xlabel('number of epochs'),ylabel
('sum squared error'),title('epochs Vs sum squared error');
%
% Saving the returned weights and biases of the model after training.
 save hemod2.w2;
 end
```

8.2.3 'simodtrn.m'

```
% UHT plant single model neural network training function.
%
% This function trains a neural network, consisting of 3 past values of the
% previous inputs (steam valve position, Psv, steam inlet pressure, Psi, and hot
% water temperature entering heat exchanger loop, Thi) and 3 past values of
% previous output (product temperature exiting the heat exchanger, Tpo).
%
% The input and output values are fed to a two layer network in which layer 1
% consists of 3 tan-sigmoid neurons,while layer 2 consists of one logsig
% neuron.
% This training uses back propagation with momentum and an adaptive
% learning rate.
%
clg
% Clears any existing graph
%
hold off
axis('normal')
pausetime = 0;
clear;
%
% Matrix P defines the input and output data for network training from the
% pattern data file (i.e. DATFILE3) of the network to be trained.
load DATFILE3;
P=DATFILE3(:,2:13)';
%
% Row vector T defines the associated targets (i.e. values of Tpo) of the
% network.
T=DATFILE3(:,14)';
clear DATFILE3;
%
% Initialising the Network
```

```
% Setting up the Input vector size R for training, the training  batch size Q, and
% the number of neurons in each training layer, S1 for layer 1 and S2 for layer
% 2 (value for S1 is taken as 3).
[R,Q] = size(P)
S1 = 3
[S2,Q] = size(T)
%
% Intialize weights and biases only in case of this model using the same pattern
% data being trained before for specific number of epochs and the weights and
% biases data stored in a file.
% Loading the past weights and biases resulted from the model being trained
% for 500 epochs.
% load hemod2.w1;
% W10=W1;
% B10=B1;
% W20=W2;
% B20=B2;
% clear hemod2.w1;
%
% Creating the weights and biases using the previous weights and biases for the
% network model training continuation for the rest of the epochs as specified
% in the training parameters.
[W10,B10] = NWTAN(S1,R);
W20 = rands(S2,S1)*0.5;
B20 = rands(S2,1)*0.5;
%
% NETWORK TRAINING
%
% Defining the Training Parameters
% Defines the display interval for all the graphs in terms of number of epochs.
 disf = 50;
% Defines the number of training epochs ( only 500 epochs taken each time
% for training upto a total of 8000 epochs for a better training check for
% network learning).
trep = 500;
% defines the maximum converging error goal
erg = 0.001;
```

```
% defines the learning rate
lr = 0.1;
%
% defines the learning increment
lr_in = 1.04;
% defines the learning decrement
lr_de = 0.8;
% defines the momentum
mom = 0.95;
% defines the error ratio
errat = 1.04;
%
% All the training parameters being placed in a row vector TP.
 TP = [disf trep erg lr lr_in lr_de mom erat];
%
% Passing on all the parameters which includes the previous weights and biases,
% input output matrix and target vectors formed from the pattern data of the
% model to be trained, and the training parameters vector to trainbpx function
% as input arguments.
% This function will return weights and biases as the training progress.
 [W1,B1,W2,B2,epoch,TR]= trainbpx(W10,B10,'tansig',W20,B20,'logsig',P,T,TP);
 TR(:,epoch)
%
% The oupt (output) resulting from the calculations using the returned weights
% and biases of the network model training gives the network model prediction
% of the targets.
oupt = logsig(W2*tansig(W1*P,B1),B2);
%
% Gives the number of training patterns.
tpn = [1:Q]';
%
% Gives the error calculated as the difference between the target data and
% output prediction result from model training.
err = T-oupt;
%
% sse gives the sum-squared-error.
 sse=sumsqr(err);
```

```
 fprintf('\n S.S.E is %8.4f',sse);
 fprintf('\n Mean Error is %8.4f\n',sse/Q);
%
% Gives the error result after each epoch as the training progress.
 fprintf('Error at %4.0f epoch is %8.4f\n',epoch,TR(1,epoch));
%
% Plotting four graphs one each for output and target comparision, error,
% learning rate, and the sum squared error of the model as training progress.
%
clg,
 subplot(221),plot(tpn,[oupt;T],'--'),xlabel('training patterns'),
 ylabel('output & Target'),title('pattterns Vs output & targets');
 subplot(222),plot(tpn,T-oupt),xlabel('training patterns'),
 ylabel('error'),title('patterns Vs error');
 subplot(223),plot(1:epoch,TR(2,1:epoch)),xlabel('number of epochs'),
 ylabel('learning rate'),title('epochs Vs learning rate');
 subplot(224),plot(1:epoch,TR(1,1:epoch)),xlabel('number of epochs'),ylabel
('sum squared error'),title('epochs Vs sum squared error');
%
% Saving the returned weights and biases of the model after training.
 save simod.w2;
end
```

8.2.4 'svmodtes.m'

```
% Trained steam valve neural network sub-model test function
%
% This function gives the steam valve network sub-model's output prediction of
% another plant run data and the  resulting error (i.e. the difference between
% the scaled predicted output and the scaled actual output of plant data) in
% terms of sum-squared-error and mean-squared-error.
%
% This function also presents the graph of sum-squared-error against the
% number of data patterns of actual testing plant data.
%
% Clears the existing data and graphs
clear;
clg;
%
% Loads the pattern data file of the test plant run
load datfile1;
%
% P gives a matrix of input and output data of test plant run and T gives a row
% vector of target data (actual outputs of plant run).
P = datfile1(:,2:10)';
T = datfile1(:,11)';
%
% PT gives a row vector of pattern numbers
PT = datfile1(:,1)';
%
% Clearing the pattern file data
clear datfile1;
% Defines the actual number of patterns
PT = PT-3;
%
% Loading the weights and biases data of the trained steam valve model used
% for testing the plant data.
load svmod1.w1;
%
% Actual testing by means of calculating the A1 and A2 vectors using the
```

```
% input-output matrix P, the weights and biases of the trained model.
A1 = tansig(W1*P,B1);
A2 = logsig(W2*A,B2);
%
% Clearing the weights and biases data of trained model.
clear svmod1.w1;
%
% To check the size of the result i.e. number of row and columns of A2.
[m,n] = size(A2)
%
% err gives the error vector i.e. the difference between the targets from test
% plant data and the predicted outputs of the trained model.
err = T'-A2';
SSE = err'*err;
%
% Gives the sum-squared-error and mean-squared-error values in predicting
% the test plant run data.
fprintf('\n SUM SQUARED ERROR IS %8.4f',SSE);
fprintf('\n MEAN SQUARED ERROR IS %8.5f',SSE/n);
%
% Gives the plot of targets (actual outputs of test plant run) and predicted
% outputs of test plant run by the trained model.
%
plot(PT,T,PT,A2,'--'),xlabel('Test data Patterns'),ylabel('Hot Water Temperature
Thi (in deg C)');
end
```

8.2.5 'hemodtes.m'

```
% Trained heat exchanger neural network sub-model test function
%
% This function gives the heat exchanger network sub-model's output
% prediction of another plant run data and the  resulting error (i.e. the
% difference between the scaled predicted output and the scaled actual output
% of plant data) in terms of sum-squared-error and mean-squared-error.
%
% This function also presents the graph of sum-squared-error against the
% number of data patterns of actual testing plant data.
%
% Clears the existing data and graphs
clear;
clg;
%
% Loads the pattern data file of the test plant run
load datfile2;
%
% P gives a matrix of input and output data of test plant run and T gives a row
% vector of target data (actual outputs of plant run).
P = datfile2(:,2:7)';
T = datfile2(:,8)';
%
% PT gives a row vector of pattern numbers
PT = datfile2(:,1)';
%
% Clearing the pattern file data
clear datfile2;
% Defines the actual number of patterns
PT = PT-3;
%
% Loading the weights and biases data of the trained heat exchanger model
% used for testing the plant data.
load hemod1.w1;
%
% Actual testing by means of calculating the A1 and A2 vectors using the
```

```
% input-output matrix P, the weights and biases of the trained model.
A1 = tansig(W1*P,B1);
A2 = logsig(W2*A,B2);
%
% Clearing the weights and biases data of trained model.
clear hemod1.w1;
%
% To check the size of the result i.e. number of row and columns of A2.
[m,n] = size(A2)
%
% err gives the error vector i.e. the difference between the targets from test
% plant data and the predicted outputs of the trained model.
err = T'-A2';
SSE = err'*err;
%
% Gives the sum-squared-error and mean-squared-error values in predicting
% the test plant run data.
fprintf('\n SUM SQUARED ERROR IS %8.4f',SSE);
fprintf('\n MEAN SQUARED ERROR IS %8.5f',SSE/n);
%
% Gives the plot of targets (actual outputs of test plant run) and predicted
% outputs of test plant run by the trained model.
%
plot(PT,T,PT,A2,'--'),xlabel('Test data Patterns'),ylabel('Product Temperature
Tpo (in deg C)');
end
```

8.2.6 'simodtes.m'

```
% Trained single neural network model test function of UHT plant
%
% This function gives the single neural network model's output prediction of
% another plant run data and the  resulting error (i.e. the difference between
% the scaled predicted output and the scaled actual output of plant data) in
% terms of sum-squared-error and mean-squared-error.
%
% This function also presents the graph of sum-squared-error against the
% number of data patterns of actual testing plant data.
%
% Clears the existing data and graphs
clear;
clg;
%
% Loads the pattern data file of the test plant run
load datfile;
%
% P gives a matrix of input and output data of test plant run and T gives a row
% vector of target data (actual outputs of plant run).
P = datfile(:,2:13)';
T = datfile(:,14)';
%
% PT gives a row vector of pattern numbers
PT = datfile(:,1)';
%
% Clearing the pattern file data
clear datfile;
% Defines the actual number of patterns
PT = PT-3;
%
% Loading the weights and biases data of the trained single network model
% used for testing the plant data.
load simod.w1;
%
% Actual testing by means of calculating the A1 and A2 vectors using the
```

```
% input-output matrix P, the weights and biases of the trained model.
A1 = tansig(W1*P,B1);
A2 = logsig(W2*A,B2);
%
% Clearing the weights and biases data of trained model.
clear simod.w1;
%
% To check the size of the result i.e. number of row and columns of A2.
[m,n] = size(A2)
%
% err gives the error vector i.e. the difference between the targets from test
% plant data and the predicted outputs of the trained model.
err = T'-A2';
SSE = err'*err;
%
% Gives the sum-squared-error and mean-squared-error values in predicting
% the test plant run data.
fprintf('\n SUM SQUARED ERROR IS %8.4f',SSE);
fprintf('\n MEAN SQUARED ERROR IS %8.5f',SSE/n);
%
% Gives the plot of targets (actual outputs of test plant run) and predicted
% outputs of test plant run by the trained model.
%
plot(PT,T,PT,A2,'--'),xlabel('Test data Patterns'),ylabel('Product Temperature
Tpo (C)');
end
```

8.2.7 'cbmodtes.m'

```
% Composite neural network model test function
%
% The composite neural network model is a combination of steam valve and
% heat exchanger neural network sub-models.
%
% This function gives the composite neural network model output prediction
% of another plant run and the resulting error in terms of sum-squared and
% mean-squared errors.
%
% This function uses the weights and biases of the trained steam valve and heat
% exchanger network sub-models and the scaled pattern data of sub-models.
%
% Clears all the existing data in the work space before the function execution.
clear;
% Loads the weights and biases data of the trained steam valve sub-model.
load svmod1.w1;
%
% Moving the weights and biases of the steam valve sub-model for calculation
% purposes.
W11=W1;
W12=W2;
B11=B1;
B12=B2;
%
% Clearing the weights and biases data of steam valve model
clear svmod1.w1;
%
% Loads the weights and biases data of the trained heat exchanger
% sub-model.
load hemod1.w1;
%
% Moving the weights and biases of the heat exchanger sub-model for
% calculation purposes.
W21=W1;
W22=W2;
```

```
B21=B1;
B22=B2;
%
% Clearing the weights and biases data of heat exchanger model
clear hemod1.w1;
%
% Loads the steam valve network sub-model pattern data file of the test plant
% run.
load datfile1;
%
% Gives the number of rows and columns of pattern data of 'datfile1'.
[mrows,mcons]=size(z1)
%
% Loads the heat exchanger network sub-model pattern data file of the test
% plant run.
load datfile2;
%
% Gives the number of rows and columns of pattern data of 'datfile2'.
[nrows,ncons]=size(z2)
%
% Calculating the output predictions for the pattern data of both steam valve
% and heat exchanger sub-models.
%
% err1 gives the error vector of the composite model tested against another
% plant run steam valve pattern data.
% err2 gives the error vector of the composite model tested against another
% plant run heat exchanger pattern data.
err1=[ ];
err2=[ ];
%
% Calculation of output prediction by the trained model.
for x = 1:mrows
%
% P1 gives a matrix of input and output data of test plant run, T1 gives a row
% vector of target data (actual outputs of plant run) and PAT1 gives a row
% vector of pattern numbers for steam valve sub-model.
    P1=datfile1(pstep,2:10)';
```

```
        T1=datfile1(pstep,11)';
        PAT1=datfile1(pstep,1)';
%
% Actual testing by means of calculating the A1 and A2  vectors using the
% input-output matrix P1, the weights and biases of the trained steam valve
% sub-model.
        A1=tansig(W11*P1,B11);
        A2=logsig(W12*A1,B12);
%
% P2 gives a matrix of input and output data of test plant run, T2 gives a row
% vector of target data (actual outputs of plant run) and PAT2 gives a row
% vector of pattern numbers for heat exchanger sub-model.
        P2=[(pstep,2:6) A2]';
        T2=datfile2(pstep,8)';
        PAT2=datfile2(pstep,1)';
%
% Actual testing by means of calculating the A3 and A4 vectors using the
% input-output matrix P2, the weights and biases of the trained heat exchanger
% sub-model.
        A3=tansig(W21*P2,B21);
        A4=logsig(W22*A1,B22);
%
% er1 gives the error in the output prediction of composite model (i.e. Hot
% water temperature entering the loop, Thi).
        er1=T1-A2;
%
% er2 gives the error in the output prediction of the composite model
% (i.e. product temperature exiting the heat the heat exchanger, Tpo).
        er2=T2-A4;
        err1=[err1;er1];
        err2=[err2;er2];
%
% Displays the composite model prediction error for Thi
        disp(err1)
% Displays the composite model prediction error for Tpo
        disp(err2)
end
```

```
%
% Calculating the sum squared-errors for Thi and Tpo by the composite
% model.
SSE1=err1'*err1
SSE2=err2'*err2
```

8.3 NEURAL NETWORK MODEL PREDICTIVE CONTROL SYSTEM SOFTWARE

8.3.1 'svnmod1.m'

```
function yp1 = svnmod1(u1,ypast,MW11,MW12,MB11,MB12)
% YP1 = SVNMOD1(U1,Ypast,MW11,MW12,MB11,MB12)
%
% This function gives the predicted output at present step (i.e. hot water
% temperature entering the loop, Thi) by the steam valve neural network
% sub-model.
%
% The neural network prediction model of the model predictive control system
% uses this function.
%
% The input argument u1 is a vector of present and past inputs to the steam
% valve sub-model (i.e. steam valve position, Psv, and Inlet steam pressure, Psi).
% ypast is a vector of past outputs (i.e. Thi)
% uu1 is the latest input vector with past outputs (Thi), disturbance inputs for
% prediction (Psi) and controlled inputs (Psv).
% uu1 = [ypast u1];
%
% yp1 gives the predicted output (Thi) using the weights and biases of the
% steam valve sub-model.
yp1=logsig(MW12*(tansig(MW11*uu1,MB11)),MB12);
```

8.3.2 'henmod2.m'

```
function yp2 = henmod2(u2,ypast,MW21,MW22,MB21,MB22)
% YP2 = HENMOD2(U2,Ypast,MW21,MW22,MB21,MB22)
%
% This function gives the predicted output at present step (i.e. product
% temperature exiting the heat exchanger, Tpo) by the heat exchanger neural
% network sub-model.
%
% The neural network prediction model of the model predictive control
% system uses this function.
%
% The input argument u2 is a vector of present and past inputs to the heat
% exchanger sub-model (i.e. hot water temperature entering the loop, Thi).
% ypast is a vector of past outputs (i.e. Tpo)
% uu2 is the latest input vector with past outputs (Tpo) and controlled
% inputs (Thi).
% uu2 = [ypast u2];
%
% yp2 gives the predicted output (Tpo) using the weights and biases of the heat
% exchanger sub-model.
yp2 = logsig(MW22*(tansig(MW21*uu2,MB21)),MB22);
```

8.3.3 'svpmod1.m'

```
function ptyp1 = svpmod1(pu1,yppast,PW11,PW12,PB11,PB12)
% PTYP1 = SVPMOD1(PU1,Yppast,PW11,PW12,PB11,PB12)
%
% This function gives the predicted output at present step (i.e. hot water
% temperature entering the loop, Thi) by the steam valve neural network
% sub-model.
%
% The neural network plant model of the model predictive control system uses
% this function.
%
% The input argument pu1 is a vector of present and past inputs to the steam
% valve sub-model (i.e. steam valve position, Psv and Inlet steam pressure, Psi).
% yppast is a vector of past outputs (i.e. Thi)
% puu1 is the latest input vector with past outputs (Thi), disturbance
% inputs for prediction (Psi) and controlled inputs (Psv).
% puu1 = [yppast pu1];
%
% ptyp1 gives the predicted output (Thi) using the weights and biases of the
% steam valve sub-model.
ptyp1=logsig(PW12*(tansig(PW11*puu1,PB11)),PB12);
```

8.3.4 'hepmod2.m'

```
function ptyp2 = hepmod2(pu2,yppast,PW21,PW22,PB21,PB22)
% YP2 = HEPMOD2(PU2,Yppast,PW21,PW22,PB21,PB22)
%
% This function gives the predicted output at present step (i.e. product
% temperature exiting the heat exchanger, Tpo) by the heat exchanger neural
% network sub-model.
%
% The neural network plant model of the model predictive control system uses
% this function.
%
% The input argument pu2 is a vector of present and past inputs to the heat
% exchanger sub-model (i.e. hot water temperature entering the loop, Thi).
% yppast is a vector of past outputs (i.e. Tpo)
% puu2 is the latest input vector with past outputs (Tpo) and the controlled
% inputs (Thi).
% puu2 = [yppast pu2];
%
% ptyp2 gives the predicted output (Tpo) using the weights and biases of the
% heat exchanger sub-model.
ptyp2 = logsig(PW22*(tansig(PW21*puu2,PB21)),PB22);
```

8.3.5 'predmod.m'

```
function [y,g]=predmod(u,upast,ypast,diss)
% [Y,G] = PREDMOD(U,Upast,Ypast,DISS)
%
% This function gives the composite neural network prediction model output
% prediction (i.e. product temperature exiting the heat exchanger loop, Tpo).
%
% This function uses both the steam valve and heat exchanger neural network
% prediction sub-models.
%
% The input arguments to this function are present inputs, u; past inputs, upast;
% past outputs, ypast and disturbance for prediction, Psi.
%
% This function also updates the input and output values of both steam valve
% and heat exchanger prediction sub-models.
%
% mrows and mcol gives the rows and columns of past inputs matrix of
% the omposite network model.
[mrows,mcol] = size(upast);
%
% y stores the predicted values of hot water temperature entering the heat
% exchanger loop, Thi and product temperature, Tpo.
y = [ ];
%
% diss defines the disturbance vector and u1 is a matrix of past predicted
% outputs, prediction disturbance vector and present inputs.
%
% yp1 gives the output prediction of steam valve sub-model (Thi).
for nstep =1:mrows
    diss = diss(nstep:nstep+2,:);
    u1 = [ypast(:,1);diss;(ones(mrows,1)*u)];
    yp1 = svnmod1(u1,ypast,MW11,MW12,MB11,MB12);
% The input vector to the heat exchanger sub-model includes the latest
% predicted output (i.e. Thi at present step).
%
% ymrows and ymcol gives the rows and columns of past outputs matrix of
```

```
% composite network model.
      [ymrows,ymcol] =size(ypast);
%
% u2 is the input matrix of past predicted outputs and present inputs to the heat
% exchanger sub-model.
% yp2 gives the output prediction of composite network model (Tpo).
      u2 = [ypast(:,2);[ypast(2:ymrows,1);yp1]];
      yp2 = henmod2(u2,ypast,MW21,MW22,MB21,MB22);
%
% Updates the output vector with the model prediction values of thi and tpo.
      y = [y;[yp1 yp2]];
end
```

8.3.6 'plantmod.m'

```
function [y,g]=plantmod(u,upast,ypast,dipr)
% [Y,G] = PLANTMOD(U,Upast,Ypast,DIPR)
%
% This function gives the composite neural network plant model output
% prediction (i.e. product temperature exiting the heat exchanger loop, Tpo).
%
% This function uses both the steam valve and heat exchanger neural network
% plant sub-models.
%
% The input arguments to this function are present inputs, u; past inputs, upast;
% past outputs, ypast and steady inputs for neural network  prediction purpose,
% Psi (Inlet steam pressure).
%
% This function also updates the input and output values of both steam valve
% and heat exchanger plant sub-models.
%
% prows and pcol gives the rows and columns of past inputs matrix of
% composite network model.
[prows,pcol] = size(upast);
%
% y stores the predicted values of hot water temperature entering the heat
% exchanger loop, Thi and product temperature, Tpo.
y = [ ];
%
% pu1 is a matrix of past predicted outputs and present inputs.
% yp1 gives the output prediction of steam valve sub-model (Thi).
for npstep =1:prows
      pu1 = [ypast(:,1);dipr;(ones(mrows,1)*u)];
      ptyp1 = svpmod1(pu1,ypast,PW11,PW12,PB11,PB12);
%
%
% The input vector to the heat exchanger sub-model includes the latest
% predicted output (i.e. Thi at present step).
%
% yprows and ypcol gives the rows and columns of past outputs matrix of
```

```
% composite network model.
      [yprows,ypcol] =size(ypast);
%
% u2 is the input matrix of past predicted outputs and present inputs to the heat
% exchanger sub-model.
% ptyp2 gives the output prediction of composite network model (Tpo).
      pu2 = [ypast(:,2);[ypast(2:yprows,1);ptyp1]];
      ptyp2 = hepmod2(pu2,ypast,PW21,PW22,PB21,PB22);
%
% Updates the output vector with the model prediction values of thi and tpo.
      y = [y;[ptyp1 ptyp2]];
end
```

8.3.7 'perform.m'

```
function [f,g] = perform(u,model,Ysp,Q,R,u0,P1,P2,P3)
% [F,G] = PERFORM(U,MODEL,Ysp,Q,R,U0,P1,P2,P3)
%
% This function evaluates the performance index
%
% [F,G] = PERFORM(U,'model',Ysp,Q,R,U0,P1,P2,P3,P4) returns the
% performance index for the model 'model' under the given control inputs.
% 'model' is a string giving the name of the m-file containing the model. U0 is
% the input at U(0).
%
% The performance index is given by
% J = SUM(deltaY*Q*deltaY' + deltaU*R*deltaU')
%
% from k = 1 to k = rows(U)
% where deltaY is Y-Ysp and deltaU is U(k)-U(k-1)
%
% This function returns two arguments; a scalar of the function to minimise, F,
% and a matrix of constraints, G (see the CONSTR function),
% i.e. [F,G] = model(U,P1,P2,P3,P4).
%
% [F,G] = PERFORM(U,'model',YSP,Q,R,U0,P1,P2,P3,P4) allows the arguments
% P1, P2, P3 etc. to be passed directly to the function.
%
% Copyright (C) 1993 Huub Bakker
% version 1
%
% Evaluating the model
if nargin == 6, [y, g] = feval(Model, u); end;
if nargin == 7, [y, g] = feval(Model, u, P1); end;
if nargin == 8, [y, g] = feval(Model, u, P1, P2); end;
if nargin == 9, [y, g] = feval(Model, u, P1, P2, P3); end;
if nargin == 10, [y, g] = feval(Model, u, P1, P2, P3, P4); end;
%
if debug < 3,
  disp('y=')
```

```
  disp(y)
end
disp(u)
disp(P1)
disp(P2)
disp(P3)
% Nsteps and Ninputs give the number of time steps and inputs from the input
% matrix
[Nsteps, Ninputs] = size(u);
u=[u0;u];
%
% Ysp defines the setpoint for the performance index evaluation of the model
%
y=(y-ones(Nsteps,1)*Ysp);
u=(u(2:Nsteps+1,:)-u(1:Nsteps,:));
disp(u)
disp(y)
if debug < 2,
  disp('Y cost =')
   disp(diag(Y*Q*Y'))
  disp('U cost =')
   disp(diag(u*R*u'))
end
%
f=sum(diag(y*Q*y'))+sum(diag(u*R*u'));
%
if DEBUG < 4,
  disp('f=')
  disp(f)
  pause;
end
```

8.3.8 'mpc.m'

```
function[Uopt]=mpc(model,u,Ysp,Q,R,u0,Low,Up,FOpt,P1,P2,P3)
%
% MPC Model Predictive Controller
% [Uopt] = MPC('predmodel', u, Ysp, Q, R, u0, Low, Up, FOpt,upast,
% ypast, dipr)
%
% Returns optimal inputs, uopt, for rows(u) time steps ahead, as estimated from
% the predictive model, 'predmodel', using the performance index:
%
% J = SUM(deltaY*Q*deltaY' + deltaU*R*deltaU') from k = 1 to k = rows(u)
% where deltaY is Y-Ysp and deltaU is u(k)-u(k-1) and Y is the predictive
% output of the model.
% u0 contains the inputs at u(0).
%
% u is the starting point for the optimisation and u0 contains the current inputs,
% i.e. at u(0).
% Ysp is a row vector of output setpoints.
%
% Q and R are weighting matrices for the outputs and controls respectively. If
% not given these are defined as identity matrices with the size given by the
% number of outputs and inputs respectively.
%
% The number of future steps included in the optimisation is taken from the
% number of rows in u. 'predmodel' is the composite prediction model m-file
% where the composite model is a combination of steam valve and heat
% exchanger sub-models.
%
% upast and ypast are the stored input and output values to be used for output
% predictions in 'predmodel'. dipr gives the disturbance vector for the purpose
% of neural net model output prediction.
%
% The model returns two arguments; a scalar of the function to be minimised,
% F, and a matrix of constraints, G (see the CONSTR function),
% i.e. [Y,G] = nnmodel(u,upast,ypast,dipr).
% The arguments upast,ypast,dipr are passed directly to the function and If
```

```
% more than four arguments need to be passed will use the GLOBAL
% command.
%
% Low and Up are row vectors of lower and upper bounds, respectively, on
% possible control inputs. They are set to zeros and ones, respectively, if not
% specified.
% FOpt, if given, is an options vector which is passed to the CONSTR function.
%
% Copyright (C) 1993 Huub Bakker
%
% Check for minimum number of arguments
if nargin < 9,
  error('Too few arguments')
  return;
end
%
highval = 0.9; % Multiplicative value for the upper bound constraints
lowval = 0.14; % Multiplicative value for the lower bound constraints
%
% Defines the name of the performance index function
PerformFun = 'perform';
%
% steps and Ninputs defines the number of predictive steps and inputs of input
% matrix.
[steps, NInputs] = size(u);
%
% Defines the options for optimisation if not already specified
if length(FOpt) == 0,
  FOpt = foptions;      % Get the default options for the CONSTR function
  FOpt(16) = 0.005;   % min change for finite diffs in gradient calculation
  FOpt(2) = 0.001;              % Uopt tolerance0.001
  FOpt(3) = 0.05;               % Tolerance for f (cost function)
  FOpt(14) = 16*NInputs;      % Max No of optimiser iterations
disp('Shouldnt be in here!')
end
%
% Initialising lower bounds on u if not already done
```

```
if length(Low) == 0, Low = ones(NInputs,steps)*lowval; end
%
% Initialising upper bounds on u if not already done
if length(Up) == 0, Up = ones(NInputs,steps)*highval; end
%
% Low and Up defines the matrices of lower and upper bounds for CONSTR
% function.
Low = ones(steps,1)*Low;
Up = ones(steps,1)*Up;
%
% Initialising weighting matrices if not already done
if length(Q) == 0, Q = eye(length(Ysp)); end
if length(R) == 0, R = eye(NInputs); end
%
% Constructs a string to evaluate the CONSTR function
evalstr = ['constr(PerformFun,u,FOpt,Low,Up,[],model,Ysp,Q,R,u0'];
  for i=1:nargin - 9
   evalstr = [evalstr,',P',int2str(i)];
  end;
evalstr = [evalstr, ')'];
%
% Calls the CONSTR function with the required arguments
Uopt = eval(evalstr);
5
%if debug < 5,
  disp('Uopt=');
  disp(Uopt);
end
```

8.3.9 'mpctest.m'

```
y = mpctest()
% Y = MPCTEST()
%
% MPCTEST is an m-file to test the model predictive controller functions.
clear
%
% Loads the neural net weights and biases to be used by the prediction and
% plant neural network models of UHT plant for output predictions.
%
load plantwt;          % Loads the weights and biases for the plant model
%
load modelwt;          % Loads the weights and biases for the prediction model
%
% Declaring the weights and biases as global
global PW11,PW12,PB11,PB12,PW21,PW22,PB21,PB22,MW11,MW12
       MB11,MB12,MW21,MW22,MB21,MB22;
%
debug = 5;
%
% Initialising the variables of the 'plant'
%
% predictive interval, seconds
nntime = 10;
% Simulation time in seconds
simtime = 1000;
% number of future steps
steps = 3;
% number of past steps
psteps = 2;
%
% Initialising the inputs matrix
% psvss defines the steady state input value of steam valve position
u0 = [psvsca(psvss)];
%
% Initialising the outputs matrix
```

```
% 'thiss' and 'tposs' define the hot water temperature entering the loop and
product temperature exiting the heat exchanger.
y0 = [thisca(thiss) tposca(tposs)];
ypast = ones(steps,1)*y0;
%
Uopt = ones(steps,1)*u0;        % Initialising the optimiser input vector
upast = ones(steps,1)*u0;
%
% Gives the disturbance input vector to be seen by the plant and
% for network model predictions.
din = ones(2,1)*psisca(psiss);
di = [ones(psteps,1);ones((simtime/nntime),1);ones(steps,1)]*psisca(psiss);
%
% Initialising setpoints vector
% tposs is the steady state value of product temperature exiting the heat
% exchanger.
Ysp = [0 tposca(tposs)];
%
% Initialising the lower and upper bounds on inputs
% Low defines the lower bound on the value of steam valve position
Low = [psvsca(psvmx)];
% Up defines the upper bound on the value of steam valve position
Up = [psvsca(psvmn)];
%
% Initialising the weighting matrices Q and R.
% Weighting matrix Q for output Tpo in the output vector Y = [thi tpo].
Q = [0 0; 0 0.1];
% Weighting scalar R for the control input psv (steam valve position)
R = 0.1 ;
%
% Initialising the optimisation options vector
FOpt = foptions; % Get the default options for the CONSTR function
FOpt(16) = 0.005 % min change for finite diffs in gradient calculation
FOpt(2) = 0.001;        % Uopt tolerance
FOpt(3) = 0.05;         % Tolerance for f (cost function)
FOpt(14) = 16*steps; % Maximum number of optimiser iterations
%
```

```
T = nntime:nntime:simtime'; % Defines the length of time horizon
ys = ones(length(T),length(Ysp))*NaN;
us = ones(length(T),length(u0))*NaN;
dist = ones(length(T),length(dipr(steps)))*NaN;
% Initialise the graphs
clg;
subplot(221);
title('Inlet Steam Pressure');

subplot(222);
title('Steam Valve Position');

subplot(223);
title('Hot Water Temp In (Thi)');

subplot(224);
title('Product Temp Out (Tpo)');

% This is the main loop
for t = nntime/nntime:nntime/nntime:simtime/nntime,
%
% The present input disturbance (Inlet steam pressure, psi) is for model
% prediction purpose which contains five values so that the model can take 3 at
% a time in a sliding window.
diss = di(t:t+steps+1,:);
%
Uopt = mpc('nnmodel',Uopt, Ysp, Q, R, u0, Low, Up, FOpt, upast, ypast, diss);
u = Uopt(1,:);          % Controlled first input vector to the plant
%
% Step change to disturbance (put it here so the process sees it first!)
dipr = di(t:t+psteps,:);          % Disturbance vector for the plant
%
% Operation of the plant starts here
% This plantmodel acts as a plant which is also a neural net model
 y = plantmodel(u,upast,ypast,dipr);
%
[parows,pacol]=size(ypast);
```

```
%
% Updating the output vector for next step
% ypast = [ypast(2:parows,:);y];
 ypast = y;
%
% Updating the input vector for next step
 upast = ones(parows,1)*u;
 u0 = u;
 Uopt = [Uopt(2:steps,:); Uopt(steps,:)];
%
 ys(t,:) = y(1,:);
 us(t,:) = u;
%
dist(t,:) = dipr(steps,:);
disp(us)
%
% The following graphs shows the changes in the input and output variables of
% the entire combination of the model predictive controller and the plant
% simulation
clg
% Plot of inlet steam pressure, Psi
  subplot(221), plot(T,dist), title('Inlet Steam Pressure');
% Plot of steam valve position, Psv
  subplot(222), plot(T,us), title('Steam Valve Position');
% Plot of hot water temperature entering the loop, Thi
  subplot(223), plot(T,ys(:,1)), title('Hot Water Temp In');
% Plot of product temperature exiting the heat exchanger
  subplot(224), plot(T,ys(:,2)), title('Product Temp Out');
end,
```

8.4 SUM-SQUARED AND MEAN-SQUARED ERROR TABLES

8.4.1 Steam Valve Network Sub-Model Error Table

N.N.TRAINED			UH71	UH73	UH74	UH75	UH76
DATA TESTED	PATS						
PSV71TH		S S E	0.02340	0.35660	0.32880	0.17940	0.18200
	606	M S E	0.00004	0.00059	0.00054	0.00030	0.00030
PSV73TH		S S E	0.01650	0.00860	0.00940	0.00780	0.01280
	554	M S E	0.00003	0.00002	0.00002	0.00001	0.00002
PSV74TH		S S E	0.02110	0.04420	0.00690	0.01090	0.01180
	476	M S E	0.00004	0.00009	0.00001	0.00002	0.00002
PSV75TH		S S E	0.01470	0.03610	0.01170	0.00530	0.01450
	527	M S E	0.00003	0.00007	0.00002	0.00001	0.00003
PSV76TH		S S E	0.02500	0.10060	0.01650	0.01930	0.01290
	492	M S E	0.00005	0.00020	0.00003	0.00004	0.00003
N.N INPUTS	Thi	at K-3, K-2, K-1					
N.N INPUTS	Psi,Psvp	at K-2, K-1, K					
N.N TARGETS	Thi	at K					

8.4.2 Heat Exchanger Network Sub-Model Error Table

N.N.TRAINED			UH51	UH52	UH53	UH54	UH55	UH56
DATA TESTED	PATS							
		S S E	0.00890	0.00840	0.00880	0.02480	0.01900	0.01060
TH51TPO3	567	M S E	0.00002	0.00001	0.00002	0.00004	0.00003	0.00002
		S S E	0.01440	0.01060	0.01450	0.01340	0.02500	0.01890
TH52TPO3	584	M S E	0.00002	0.00002	0.00002	0.00002	0.00004	0.00003
		S S E	0.01180	0.01040	0.01070	0.01090	0.03370	0.02380
TH53TPO3	620	M S E	0.00002	0.00002	0.00002	0.00002	0.00005	0.00004
		S S E	0.01090	0.00980	0.01030	0.01040	0.02540	0.01880
TH54TPO3	575	M S E	0.00002	0.00002	0.00002	0.00002	0.00004	0.00003
		S S E	0.00860	0.00380	0.00870	0.00720	0.00750	0.00540
TH55TPO3	625	M S E	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001
		S S E	0.01120	0.00490	0.01180	0.00960	0.00980	0.00690
TH56TPO3	616	M S E	0.00002	0.00001	0.00002	0.00002	0.00002	0.00001
		S S E	0.02630	0.01510	0.02730	0.02410	0.03870	0.03250
TH57TPO3	642	M S E	0.00004	0.00002	0.00004	0.00004	0.00006	0.00005
		S S E	0.04990	0.02240	0.05630	0.04570	0.06440	0.05510
TH58TPO3	655	M S E	0.00008	0.00003	0.00009	0.00007	0.00010	0.00008
		S S E	0.00700	0.00320	0.00700	0.00590	0.00880	0.00640
TH60TPO3	543	M S E	0.00001	0.00001	0.00001	0.00001	0.00002	0.00001
		S S E	0.00690	0.00460	0.00630	0.00590	0.01290	0.00890
TH61TPO3	505	M S E	0.00001	0.00001	0.00001	0.00001	0.00002	0.00002
		S S E	0.02680	0.01210	0.02880	0.02370	0.04030	0.03140
TH62TPO3	681	M S E	0.00005	0.00002	0.00005	0.00004	0.00007	0.00006

N.N.TRAINED			UH51	UH52	UH53	UH54	UH55	UH56
DATA TESTED	PATS							
TH63TPO3		S S E	0.01270	0.00560	0.01330	0.01090	0.01310	0.00930
	756	M S E	0.00002	0.00001	0.00002	0.00001	0.00002	0.00001
TH64TPO3		S S E	0.57150	0.20350	0.65300	0.50770	0.45270	0.40420
	521	M S E	0.00110	0.00039	0.00125	0.00097	0.00087	0.00078
TH65TPO3		S S E	0.54760	0.22400	0.60440	0.48710	0.53770	0.45730
	1323	M S E	0.00041	0.00017	0.00046	0.00037	0.00041	0.00035
TH66TPO3		S S E	0.28880	0.10230	0.34000	0.25990	0.26250	0.23130
	478	M S E	0.00060	0.00021	0.00071	0.00054	0.00055	0.00048
TH67TPO3		S S E	0.03070	0.01820	0.03240	0.02820	0.05500	0.04270
	513	M S E	0.00006	0.00004	0.00006	0.00005	0.00011	0.00008
TH68TPO3		S S E	0.03140	0.01230	0.03520	0.02790	0.03660	0.03060
	487	M S E	0.00006	0.00003	0.00007	0.00006	0.00008	0.00006
TH70TPO3		S S E	3.09910	2.25870	2.25930	2.50580	2.71790	2.15740
	486	M S E	0.00511	0.00373	0.00373	0.00413	0.00448	0.00356
TH71TPO3		S S E	0.15810	0.12070	0.11480	0.12690	0.29480	0.18930
	606	M S E	0.00029	0.00022	0.00021	0.00023	0.00053	0.00034
TH73TPO3		S S E	0.01830	0.01310	0.01690	0.01650	0.01570	0.01340
	554	M S E	0.00004	0.00003	0.00004	0.00003	0.00003	0.00003
TH74TPO3		S S E	0.09570	0.04050	0.09450	0.08090	0.04580	0.04000
	476	M S E	0.00018	0.00008	0.00018	0.00015	0.00009	0.00008
TH75TPO3		S S E	0.02790	0.01170	0.02720	0.02310	0.01580	0.01260
	527	M S E	0.00006	0.00002	0.00006	0.00005	0.00003	0.00003
TH76TPO3		S S E	0.22400	0.09520	0.22310	0.19000	0.14820	0.12360
	492	M S E	0.00046	0.00019	0.00045	0.00039	0.00030	0.00025

N.N.TRAINED			UH57	UH58	UH60	UH61	UH62	UH63
DATA TESTED	PATS							
		S S E	0.01060	0.0101	0.0189	0.0147	0.0108	0.0171
TH51TPO3	567	M S E	0.00002	0.00002	0.00003	0.00003	0.00002	0.00003
		S S E	0.01200	0.01150	0.02050	0.01720	0.01100	0.01800
TH52TPO3	584	M S E	0.00002	0.00002	0.00004	0.00003	0.00002	0.00003
		S S E	0.01420	0.01410	0.02900	0.01660	0.01230	0.02330
TH53TPO3	620	M S E	0.00002	0.00002	0.00005	0.00003	0.00002	0.00004
		S S E	0.01250	0.01220	0.02180	0.01390	0.01150	0.01820
TH54TPO3	575	M S E	0.00002	0.00002	0.00004	0.00002	0.00002	0.00003
		S S E	0.00400	0.00400	0.00590	0.00600	0.00290	0.00490
TH55TPO3	625	M S E	0.00001	0.00001	0.00001	0.00001	0.00000	0.00001
		S S E	0.00520	0.00490	0.00860	0.00790	0.00330	0.00660
TH56TPO3	616	M S E	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001
		S S E	0.01150	0.01070	0.02120	0.04260	0.01240	0.02500
TH57TPO3	642	M S E	0.00002	0.00002	0.00003	0.00007	0.00002	0.00004
		S S E	0.01210	0.00950	0.02610	0.08470	0.01040	0.03750
TH58TPO3	655	M S E	0.00002	0.00001	0.00004	0.00013	0.00002	0.00006
		S S E	0.00340	0.00340	0.00600	0.00690	0.00220	0.00540
TH60TPO3	543	M S E	0.00001	0.00001	0.00001	0.00001	0.00000	0.00001
		S S E	0.00630	0.00630	0.01170	0.00580	0.00510	0.00880
TH61TPO3	505	M S E	0.00001	0.00001	0.00002	0.00001	0.00001	0.00002
		S S E	0.00930	0.00920	0.02280	0.04180	0.00590	0.02380
TH62TPO3	681	M S E	0.00002	0.00002	0.00004	0.00008	0.00001	0.00004

N.N.TRAINED			UH57	UH58	UH60	UH61	UH62	UH63
DATA TESTED	PATS							
TH63TPO3		S S E	0.00540	0.00500	0.00960	0.00720	0.00350	0.00840
	756	M S E	0.00001	0.00001	0.00001	0.00001	0.00000	0.00001
TH64TPO3		S S E	0.01920	0.01480	0.09620	0.30600	0.01120	0.09300
	521	M S E	0.00004	0.00003	0.00018	0.00059	0.00002	0.00018
TH65TPO3		S S E	0.08140	0.07020	0.18120	0.33930	0.05250	0.17030
	1323	M S E	0.00006	0.00005	0.00014	0.00026	0.00004	0.00013
TH66TPO3		S S E	0.01520	0.01310	0.05410	0.15170	0.00690	0.05040
	478	M S E	0.00003	0.00003	0.00011	0.00032	0.00001	0.00011
TH67TPO3		S S E	0.02180	0.02080	0.03680	0.03240	0.01870	0.03290
	513	M S E	0.00004	0.00004	0.00007	0.00006	0.00004	0.00006
TH68TPO3		S S E	0.00450	0.00490	0.01140	0.02170	0.00340	0.01070
	487	M S E	0.00001	0.00001	0.00002	0.00004	0.00001	0.00002
TH70TPO3		S S E	0.19480	0.24390	0.30630	0.18640	0.15260	0.28060
	486	M S E	0.00032	0.00040	0.00051	0.00031	0.00025	0.00046
TH71TPO3		S S E	0.01680	0.01700	0.01920	0.01780	0.01860	0.01820
	606	M S E	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003
TH73TPO3		S S E	0.02230	0.02570	0.03270	0.05520	0.02500	0.03190
	554	M S E	0.00005	0.00005	0.00007	0.00012	0.00005	0.00007
TH74TPO3		S S E	0.00820	0.01220	0.01180	0.01670	0.00940	0.01150
	476	M S E	0.00002	0.00002	0.00002	0.00003	0.00002	0.00002
TH75TPO3		S S E	0.04730	0.01580	0.07950	0.13470	0.04470	0.07840
	527	M S E	0.00010	0.00003	0.00016	0.00027	0.00009	0.00016
TH76TPO3		S S E	0.04730	0.01580	0.07950	0.13470	0.04470	0.07840
	492	M S E	0.00010	0.00003	0.00016	0.00027	0.00009	0.00016

N.N.TRAINED			UH64	UH65	UH66	UH67	UH68
DATA TESTED	PATS						
		S S E	0.0143	0.0095	0.0149	0.0069	0.0143
TH51TPO3	567	M S E	0.00003	0.00002	0.00003	0.00001	0.00003
		S S E	0.01390	0.01110	0.01490	0.00800	0.01470
TH52TPO3	584	M S E	0.00002	0.00002	0.00003	0.00001	0.00003
		S S E	0.01610	0.01120	0.01670	0.00760	0.01970
TH53TPO3	620	M S E	0.00003	0.00002	0.00003	0.00001	0.00003
		S S E	0.01510	0.01160	0.01470	0.00820	0.01600
TH54TPO3	575	M S E	0.00003	0.00002	0.00003	0.00001	0.00003
		S S E	0.00290	0.00230	0.00420	0.00220	0.00380
TH55TPO3	625	M S E	0.00000	0.00000	0.00001	0.00000	0.00001
		S S E	0.00380	0.00260	0.00410	0.00230	0.00510
TH56TPO3	616	M S E	0.00001	0.00000	0.00001	0.00000	0.00001
		S S E	0.01480	0.01310	0.01940	0.00910	0.01420
TH57TPO3	642	M S E	0.00002	0.00002	0.00003	0.00001	0.00002
		S S E	0.01250	0.00890	0.01400	0.00760	0.01280
TH58TPO3	655	M S E	0.00002	0.00001	0.00002	0.00001	0.00002
		S S E	0.00250	0.00160	0.00280	0.00160	0.00350
TH60TPO3	543	M S E	0.00000	0.00000	0.00001	0.00000	0.00001
TH61TPO3		S S E	0.00660	0.00380	0.00680	0.00320	0.00780
	505	M S E	0.00001	0.00001	0.00001	0.00001	0.00001
		S S E	0.00720	0.00390	0.00700	0.00400	0.01110
TH62TPO3	681	M S E	0.00001	0.00001	0.00001	0.00001	0.00002

N.N.TRAINED			UH64	UH65	UH66	UH67	UH68
DATA TESTED	PATS						
TH63TPO3		S S E	0.00400	0.00260	0.00430	0.00260	0.00570
	756	M S E	0.00001	0.00000	0.00001	0.00000	0.00001
TH64TPO3		S S E	0.00510	0.00590	0.01860	0.03380	0.01700
	521	M S E	0.00001	0.00001	0.00004	0.00006	0.00003
TH65TPO3		S S E	0.04380	0.02240	0.04680	0.05520	0.08340
	1323	M S E	0.00003	0.00002	0.00004	0.00004	0.00006
TH66TPO3		S S E	0.00790	0.00320	0.00500	0.01410	0.01870
	478	M S E	0.00002	0.00001	0.00001	0.00003	0.00004
TH67TPO3		S S E	0.01960	0.01210	0.01960	0.00950	0.02360
	513	M S E	0.00004	0.00002	0.00004	0.00002	0.00005
TH68TPO3		S S E	0.00250	0.00230	0.00400	0.00240	0.00450
	487	M S E	0.00001	0.00000	0.00001	0.00000	0.00001
TH70TPO3		S S E	2.16960	5.94230	4.77610	2.01350	2.22550
	486	M S E	0.00358	0.00981	0.00788	0.00332	0.00367
TH71TPO3		S S E	0.10350	0.14740	0.18630	0.09590	0.20820
	606	M S E	0.00019	0.00027	0.00034	0.00017	0.00038
TH73TPO3		S S E	0.01720	0.01490	0.02180	0.01170	0.01460
	554	M S E	0.00004	0.00003	0.00005	0.00002	0.00003
TH74TPO3		S S E	0.01900	0.02410	0.03630	0.01860	0.01870
	476	M S E	0.00004	0.00005	0.00007	0.00004	0.00004
TH75TPO3		S S E	0.00480	0.00680	0.01640	0.00650	0.00690
	527	M S E	0.00001	0.00001	0.00003	0.00001	0.00001
TH76TPO3		S S E	0.03070	0.03990	0.05560	0.03660	0.03850
	492	M S E	0.00006	0.00008	0.00011	0.00007	0.00008

N.N.TRAINED			UH70	UH71	UH73	UH74	UH75	UH76
DATA TESTED	PATS							
		S S E	1.58610	0.00380	0.02310	0.19810	0.01850	0.00840
TH51TPO3	567	M S E	0.00280	0.00001	0.00004	0.00035	0.00003	0.00001
		S S E	1.66750	0.00900	0.02270	0.15020	0.01870	0.00860
TH52TPO3	584	M S E	0.00286	0.00002	0.00004	0.00026	0.00003	0.00001
		S S E	1.73290	0.00630	0.02540	0.19420	0.02650	0.00870
TH53TPO3	620	M S E	0.00280	0.00001	0.00004	0.00031	0.00004	0.00001
		S S E	1.62070	0.00730	0.02010	0.16770	0.02050	0.00920
TH54TPO3	575	M S E	0.00282	0.00001	0.00003	0.00029	0.00004	0.00002
		S S E	1.75120	0.00450	0.00780	0.13070	0.00510	0.00180
TH55TPO3	625	M S E	0.00280	0.00001	0.00001	0.00021	0.00001	0.00000
		S S E	1.74340	0.00570	0.00890	0.10850	0.00700	0.00200
TH56TPO3	616	M S E	0.00283	0.00001	0.00001	0.00018	0.00001	0.00000
		S S E	1.79150	0.00550	0.05950	0.30570	0.01910	0.00910
TH57TPO3	642	M S E	0.00279	0.00001	0.00009	0.00048	0.00003	0.00001
		S S E	1.81000	0.00610	0.12000	0.47610	0.02250	0.00770
TH58TPO3	655	M S E	0.00276	0.00001	0.00018	0.00073	0.00003	0.00001
		S S E	1.52330	0.00310	0.01060	0.13350	0.00520	0.00150
TH60TPO3	543	M S E	0.00281	0.00001	0.00002	0.00025	0.00001	0.00000
TH61TPO3		S S E	1.40420	0.00330	0.00890	0.12190	0.01050	0.00330
	505	M S E	0.00259	0.00001	0.00002	0.00022	0.00002	0.00001
		S S E	1.98300	0.00510	0.06190	0.30990	0.01890	0.00400
TH62TPO3	681	M S E	0.00365	0.00001	0.00011	0.00057	0.00003	0.00001

N.N.TRAINED			UH70	UH71	UH73	UH74	UH75	UH76
DATA TESTED	PATS							
TH63TPO3		S S E	2.12400	0.00590	0.01410	0.16160	0.00840	0.00220
	756	M S E	0.00281	0.00001	0.00002	0.00021	0.00001	0.00000
TH64TPO3		S S E	1.37080	0.11430	1.03940	2.09110	0.08050	0.01310
	521	M S E	0.00263	0.00022	0.00200	0.00401	0.00015	0.00003
TH65TPO3		S S E	3.54580	0.07630	1.04820	2.43840	0.16720	0.03490
	1323	M S E	0.00268	0.00006	0.00079	0.00184	0.00013	0.00003
TH66TPO3		S S E	1.22760	0.04530	0.59870	1.45510	0.05770	0.00990
	478	M S E	0.00257	0.00009	0.00125	0.00304	0.00012	0.00002
TH67TPO3		S S E	1.41040	0.00540	0.07130	0.30240	0.03310	0.01050
	513	M S E	0.00275	0.00001	0.00014	0.00059	0.00006	0.00002
TH68TPO3		S S E	1.36000	0.00610	0.07040	0.27370	0.01070	0.00180
	487	M S E	0.00279	0.00001	0.00014	0.00056	0.00002	0.00000
TH70TPO3		S S E	0.01060	0.33480	2.62540	0.51320	1.66210	1.29750
	486	M S E	0.00002	0.00055	0.00433	0.00085	0.00274	0.00214
TH71TPO3		S S E	1.53870	0.02820	0.20160	0.32400	0.24430	0.09020
	606	M S E	0.00278	0.00005	0.00036	0.00058	0.00044	0.00016
TH73TPO3		S S E	1.53640	0.00690	0.01100	0.11250	0.01490	0.01100
	554	M S E	0.00323	0.00001	0.00002	0.00024	0.00003	0.00002
TH74TPO3		S S E	1.31650	0.01520	0.07500	0.00950	0.02070	0.01230
	476	M S E	0.00250	0.00003	0.00014	0.00002	0.00004	0.00002
TH75TPO3		S S E	1.45390	0.00870	0.02380	0.10090	0.00760	0.00380
	527	M S E	0.00296	0.00002	0.00005	0.00021	0.00002	0.00001
TH76TPO3		S S E	1.38930	0.04560	0.25240	0.16430	0.05420	0.02070
	492	M S E	0.00282	0.00009	0.00051	0.00033	0.00011	0.00004
N.N INPUTS	Tpo	at K-3, K-2, K-1						
N.N INPUTS	Thi	at K-2, K-1, K						
N.N TARGETS	Tpo	at K						

8.4.3 Single Network Model Error Table

N.N.TRAINED			UH71	UH73	UH74	UH75	UH76
DATA TESTED	PATS						
PSV71TPO		S S E	0.03780	0.66050	0.38410	0.38000	0.20280
	606	M S E	0.00006	0.00109	0.00063	0.00063	0.00033
PSV73TPO		S S E	0.07270	0.01230	0.05430	0.02110	0.01340
	554	M S E	0.00013	0.00002	0.00010	0.00004	0.00002
PSV74TPO		S S E	0.11280	0.01920	0.01010	0.05630	0.02010
	476	M S E	0.00024	0.00004	0.00002	0.00012	0.00004
PSV75TPO		S S E	0.06830	0.01370	0.04500	0.00810	0.01210
	527	M S E	0.00013	0.00003	0.00009	0.00002	0.00002
PSV76TPO		S S E	0.15790	0.06630	0.12290	0.16190	0.01240
	492	M S E	0.00032	0.00013	0.00025	0.00033	0.00003
N.N INPUTS	Tpo	at K-3, K-2, K-1					
N.N INPUTS	Psi,Psvp,Thi	at K-2,K-1,K					
N.N TARGETS	Tpo	at K					