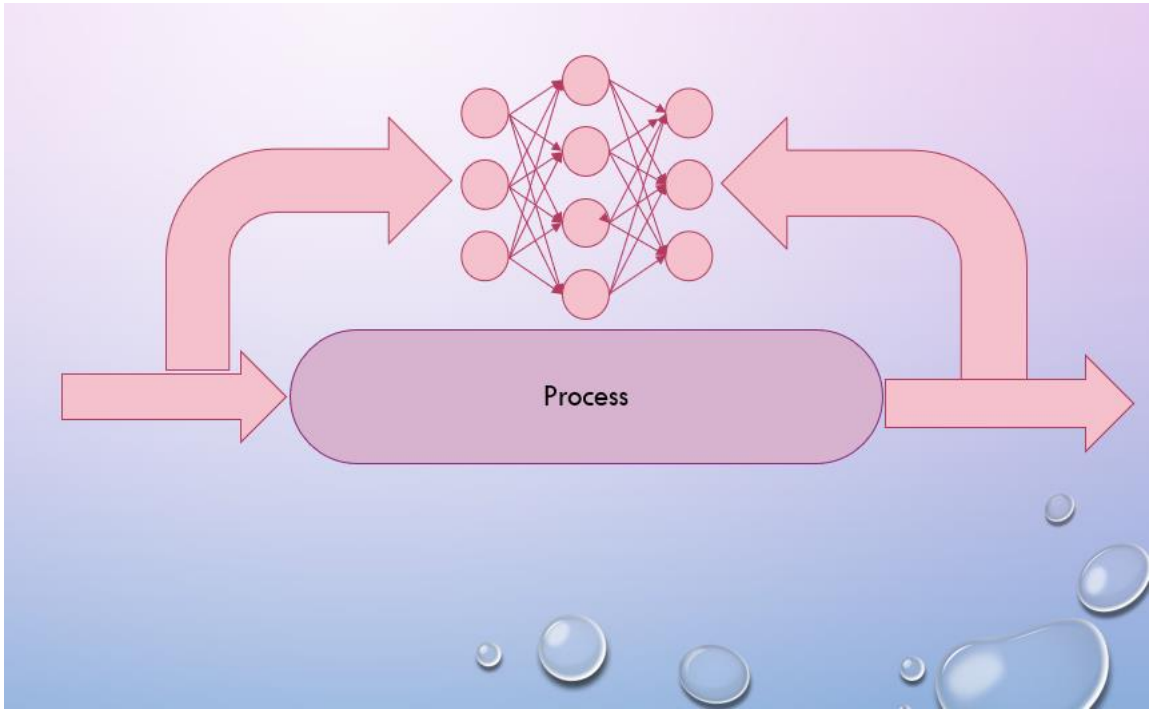
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2021

Industrial IT and Automation (IIA)

Modeling of the chemical dosing at a water resource recovery facility (WRRF)



Anas Muhamad Hashem Aldabbagh

Faculty of Technology, Natural sciences and Maritime Sciences

Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2021

Title: Modeling of the chemical dosing at a water resource recovery facility (WRRF)

Number of pages: 113

Keywords: Dynamic system, Time series NARX neural network, Artificial neural network, Machine learning ,Modelling, First order transfer function, Chemical process modeling.

Student: Anas Muhamad Hashem Aldabbagh

Supervisor: Finn Aakre Haugen

External partner: Vestfjorden Avløpsselskap v/ Jonas Pettersen

University of South-Eastern Norway

www.usn.no

Summary:

The focus here in this master thesis is developing a smart artificial neural network to model wastewater dosing. In order to use this model in the optimization process to optimize using chemical dosing materials. The thesis gives the necessary introduction to modeling dynamic systems, machine learning, and neural network. Then the dynamic system is modeled in a classical way, as first-order and second-order transfer function without and with time delay. Afterward, time delay estimation is developed and tested with the simulator.

Machine learning and artificial intelligence neural network (ANN) of the dynamic system is derived from the mathematical model of the dynamic system. The ANN is implemented by using the data from the simulator. The ANN result is compared with the result from the simulator and shows a high performance of the ANN model.

The ANN is implemented in a real process in different methods and software. The implementation has been performed as a multi-input single-output system and multi-input multi outputs system. The difference between methods is discussed.

NARX neural network model gives high performance and good accuracy in a dynamic system. It can deal with time series and handle time delays automatically.

Preface

This master thesis is performed as part of the master's program, Industrial IT and Automation at the University of South-East Norway.

The results of this thesis can be used many applications that deal with dynamic system and time series. It will be part in a significant project at VEAS for Optimizing of chemical dosing at a water resource recovery facility.

I would like to thank from my heart everyone who give me help or advice especially, Prof. Finn Aakre Haugen and Jonas Pettersen.

A lot of thanks to Norway and USN for this effort and to Industrial IT and Automation employees one by one for help and kindness.

I will never forget the source of love and kindness my mother, my wife, my siblings who support me, encourage me, push me forward.

Porsgrunn, 30.09.2021

Anas Muhamad Hashem Aldabbagh

Contents

Nomenclature

WRRF Water resource recovery facility.

FTU Formazin Turbidity Unit.

SISO Single input single output.

MISO Multi input single output.

MIMO Multi input multi output.

NARX Nonlinear autoregressive exogenous.

ANN Artificial neural network.

RNN recurrent neural network.

MPC Model predictive control.

SS Suspended solids.

PIX iron chloride sulfate

PAX poly aluminum chloride

POL polymer

1 Introduction

Veas is a company which takes care of treating a wastewater in Norway. It is located in Slemmestad. The treatment is divided into three different parts, Mechanical, biological and chemical treatment [1].

Wastewater is the byproduct of many uses in the households (showering, laundry, ..etc.), industry, commercial enterprises and other. After the water has been used, it goes to the wastewater stream and flow to the WRRF [2].

Wastewater treatment is needed to remove the wastewater pollutants to take care of the public health and protect environment. Some of materials in the wastewater can kill fish in the lake and affect public health directly or indirectly [2].

The advent of artificial neural network, machine learning, Internet of things (IOT), Cloud computing are revolution that remove many professional approaches their work. These new technologies offer exciting ways for engineers to face real world challenges. This thesis will take advantage of these new technologies to project these technologies into water dosage process to see the power of these technologies. The focus will be on modeling of chemical water dosage process by using artificial neural networks and machine learning.

This thesis will go briefly through classical modelling process and chemical materials that used in the dosage process. The focus will be on dynamic system.

1.1 Purpose of study

VEAS has a current control system to manage the chemical dosage. This system is expensive, about (60 million NOK Norwegian kroner) [1], due to the non-optimal system. Overdosage leads to overuse of chemical materials and under dosage leads to unsatisfactory results. The optimization is required to obtain satisfactory results and minimum usage of chemical materials. The operation cost of the current control system can be reduced by optimizing the use of chemical materials and choose the dosage of chemical materials optimally.

In order to get the lowest turbidity, the highest percentage of the same materials as (phosphate and alkalinity materials) in the system output, the real model (very close to real) is needed to know the exact behavior of the system regarding inputs and outputs. This model will be developed to utilize in a control system. The chemical reaction is not interested in this study, but the most important thing that to know the effect of the input on the output. Therefore, the system will be modeled as a gray or black box, and this study will be performed in terms of a control system.

This model should be smart enough to track the variation of the parameters in order to let the control system take the suitable response at a suitable time. Maybe the artificial neural network is a good choice to represent the current system.

As known, the chemical reactions take time to respond and form the final results and this time is varied according to many factors like flow rate which increase the complexity of the system. There are many variables that manipulate the system and influence the results, so the case here multivariate system. Multi-inputs and multi-outputs.

2 System description

The wastewater comes to the treatment from different sources. It is treated in several stages. The wastewater is pumped to a chemical process chamber where chemical materials are added to clean the wastewater, then the treated water goes out of the chemical chamber to the next stage as shown in Figure 2-1. [1]

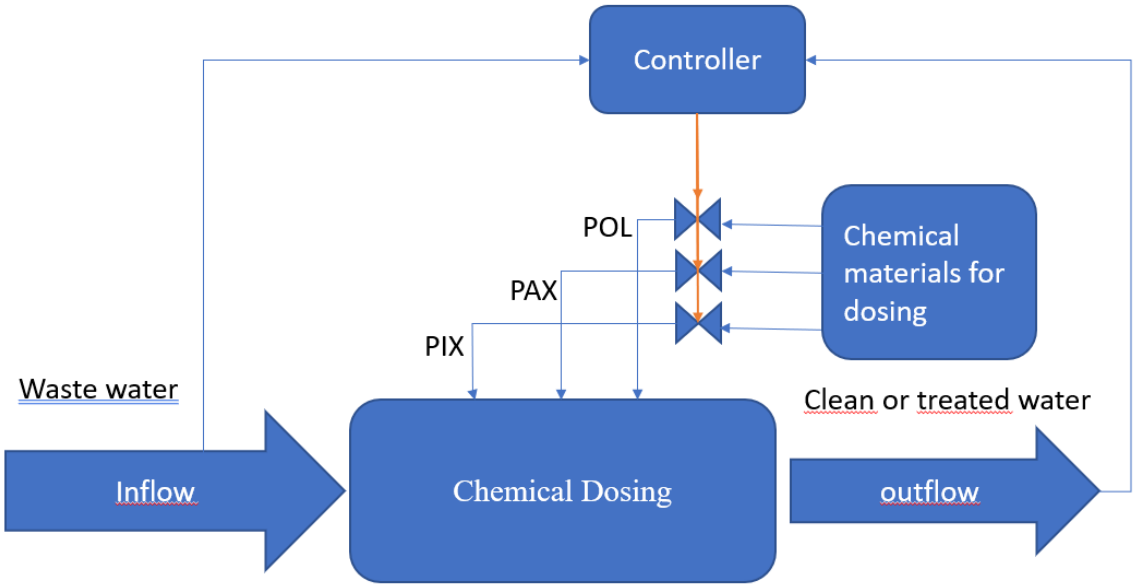


Figure 2-1 Dosing diagram

The chemical materials used for treating are coagulation chemicals and polymer. The coagulation chemicals are PIX 318 iron chloride sulfate and PAX XL61 poly aluminum chloride are used to precipitate saluted phosphate in the wastewater [1].

Polymer is important for the formation of flocs. The PIX, PAX, and POL materials react with wastewater and reduce the turbidity of the water as well as influence the alkalinity and phosphate materials. After analyzing the system inputs and outputs, it seems a dynamic system with some delay. The dynamic system can be presented as a first or a second-order transfer function with time delay. Using more than one type of material at the same time to reduce the turbidity can be led to a new reaction which is different from each material individually. The turbidity measures how much the water is clean. The turbidity sensor measures the light intensity that passes through the water [1].

2.1 System variables

Veas has an internal document that describes all important variables. This document is not constant, and it is modifying due to different situations of operation [1]. The most important variables are:

Phosphorus: these values should be maximized 0.5 mgP/l

Alkalinity: these values should be maximized. The minimum value is 1.5 mmol/l

, [3]. The Pia Ryrfors experiments said that the values can go until 0.3 mmol/l [1]

Turbidity: 6 FTU, The turbidity should be as low as possible, there are arguments about the value of turbidity some of them said 15 FTU is fine but the issue here is the cost of dosing. [1]

PIX, PAX, POL: are controllable inputs, it is used for the dosage process. PIX materials interact with alkalinity and phosphate more than PAX, so using PIX decreases the alkalinity and precipitates more phosphate in the wastewater. That means PIX can be used just when the concentration of alkalinity and phosphate are high [1].

Suspended solids (SS): refers to the number of particular solids in the water. This variable affects the water turbidity. The particular solids are taken out of the water by sedimentation.

Temperature sensor: gives information about water temperature, which could affect the chemical reactions.

The level of water's PH should not change. Alkalinity refers to the ability to maintain the level of PH in water. PH value can change after treating water with acid or base solution.

The water inflow plays a significant role in the dosage process and manipulates the dynamic system of dosage. The inflow has the most impact on system time delay. The reaction time is related much to how much water is entering the dosage process. The power of chemical mixing is affected by water flow as well [1].

3 Modelling of the System

The chemical dosing of the water process seems a dynamic slow and stable process with time delay. This kind of system can be modeled as a first or second-order transfer function. [1]. To simplify the system, each input will model as first order transfer function as single input single output SISO. After that, the transfer functions of the inputs will be combined to give the final response of the system. The inputs are PIX. POL.PAX.

The form of the transfer function H(s) for input U and output y is shown in the equation (3-1) [4] [1]

$$H(s) = \frac{k}{\tau s+1} e^{\theta S} \qquad (3\text{-}1)$$

Where k is the gain of transfer function and τ is the time constant. θ is a time delay.

Transfer function concept is applied for each of PIX, PAX, POL. The output of the transfer function for each input can be added to each other since the turbidity is affected by each input. In this case, the coupling between inputs is not considering. In other words, there is no chemical reaction between PIX, PAX, POL. Each input has an independent effect on the output. By considering the previous assumption, the block diagram of the system can be modeled as Figure 3-1.

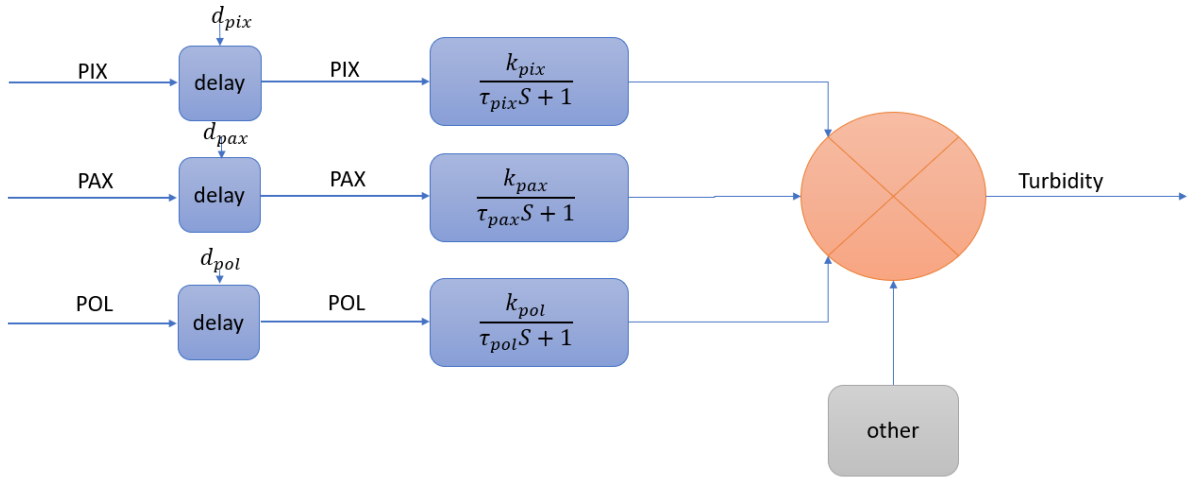


Figure 3-1 function block diagram of chemical dosing model, PIX, PAX, POL are dosing chemical materials, $d_{pix}, d_{pax}, d_{pol}$ are inputs time delay, τ is the time constant, k is the gain.

The system model in Figure 3-1 is influenced by many parameters like temperature and inflow…etc. some unknown parameters are added to the system in the other block. The time delay is not constant, and it is manipulated by other variables.

4 Artificial intelligence modelling

4.1 Artificial neural networks (ANNs)

ANNs are computational programs or systems inspired by neural networks of humans and animals. It is based on a collection of nodes called neurons that are connected to each other. Inputs and outputs in a specific way as biological brain. Each neuron receives data and transmits data after some processing. Receiving and transmitting can be to the neurons, input, or output. [5] [6]

4.1.1 Artificial Neuron Model

The neuron is the basic unit, which receives one or more inputs and processes them to produce the output. As shown in Figure 4-1

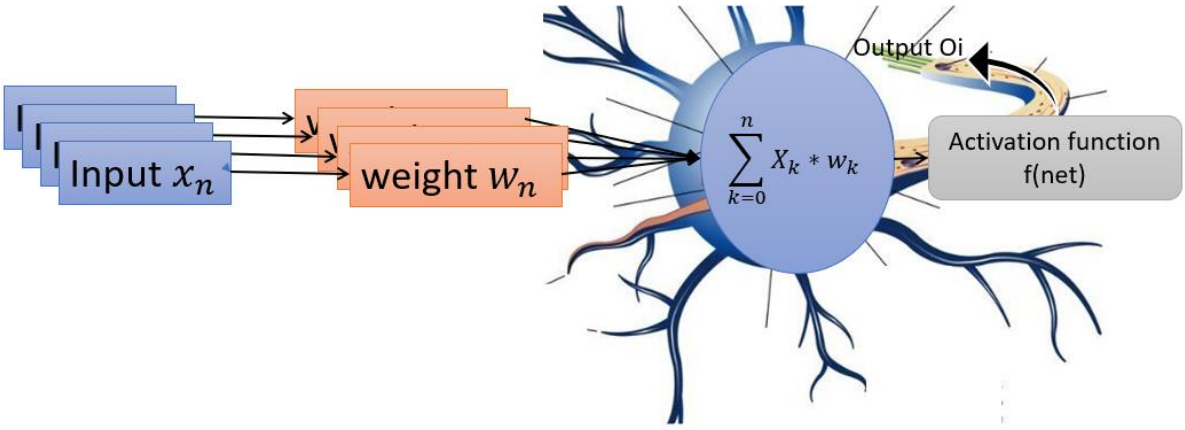


Figure 4-1 Neuron model, x1, x2---x_n are inputs, w1, w2 are weights of the inputs, o is output of the neuron (Perceptron model)

The mathematical model of the neuron represents how the neuron calculate the output. Each input will be multiplied by weight which determined how this input influence the output. the mathematical model of neuron can be described as (4-1)

$$net = w_1x_1 + w_2x_2 + \cdots + w_nx_n \qquad (4\text{-}1)$$

If the input has vector form $(x_1\ x_2 \ldots x_n)$ and weights $(w_1\ w_2 \ldots w_n\ w_2)^T$. The neuron has also bias as input which represent the shift of the output, the bias plays a significant role for some neuron, so it is important to add it separately as Figure 4-2

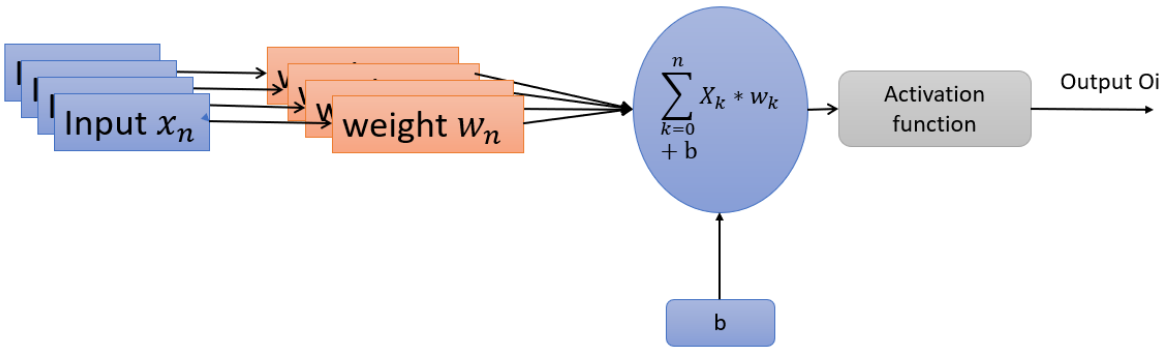


Figure 4-2 Neuron model with bias b term, (x_1 x_2 ... x_n) are input vector and (w_1 w_2 ... w_n w_2)^T are weights of the inputs, o is output of the neuron (Perceptron model)

It is necessary to include a nonlinear activation function in order to allow varying inputs conditions that determine the level of amplification. Nonlinear activation function can prevent driving the output to out of allowed limits. [6]

4.1.2 Activation functions

The common type of activation functions that are used in neural networks is a step function, Linear function, ramp function, sigmoid function, tansigmoid function. These functions are shown in Figure 4-3 the selecting of the activation functions depends on the application.

Linear function: the output of this function has a linear relationship with the input, the output is the input multiply by a constant factor as equation (2). It can be used to show the total output [6]

$$f(x) = k.x \qquad (4\text{-}2)$$

k is constant

Step function: has two output states or limits +1 or -1 according to the input value equation (4-3) . It suits classification problems

$$y = f(x) = \begin{cases} +1 \; when \; x > 0 \\ -1 \; when \; x < 0 \end{cases} \qquad (4\text{-}3)$$

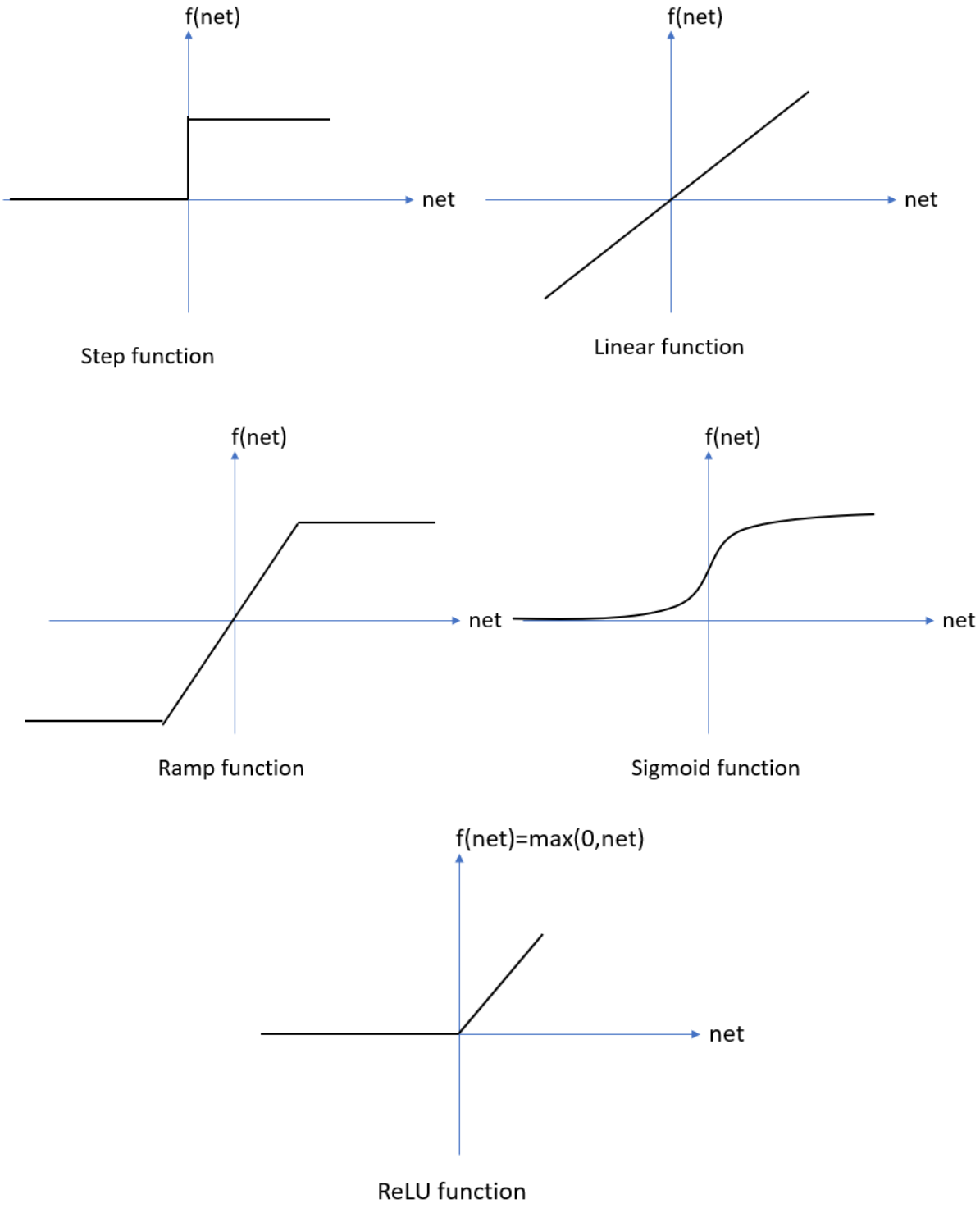


Figure 4-3 Activation functions, Unit step, Linear, Ramp, ReLU and Sigmoid.

Ramp function: The output is a combination of a linear function and a step function. It is described as (4-5) [6]

$$y = f(x) = \begin{cases} \max & when\ x > upper\ limit \\ k.x & when\ upper\ limit > x > lower\ limit \\ \min & when\ x < lower\ limit \end{cases} \quad (4\text{-}5)$$

Sigmoid and hyperbolic tangent functions: these two functions are widely used in neural networks. The sigmoid function has special properties which are nonlinearity properties and derivative properties as shown in equation (4-6) and (4-7) [6]

$$S(x) = \frac{1}{1+e^{-x}} = 1 - S(-x) \quad (4\text{-}6)$$

$$\frac{d(S(x))}{dx} = S(x)(1 - S(x)) \quad (4\text{-}7)$$

Tansigmoid function equation (4-8)

$$y(x) = \frac{1-e^{-x}}{1+e^{-x}} \quad (4\text{-}8)$$

ReLU (Rectified linear unit) function equation (4-9). it is used when the output is only positive

$$y(x) = \begin{cases} x\ if\ x \geq 0 \\ 0\ if\ x < 0 \end{cases} \quad (4\text{-}9)$$

4.2 Artificial neural network architecture

ANN architecture is formed by at least one layer. The way of connecting neurons produces two basic types of ANN architecture. They are Feedforward ANN and Recurrent or Feedback ANN [6].

4.2.1 Feedforward ANN

In this type of ANN, the connection of neurons has one direction just forward connectivity. Figure 4-4 illustrate this kind of ANN [6]

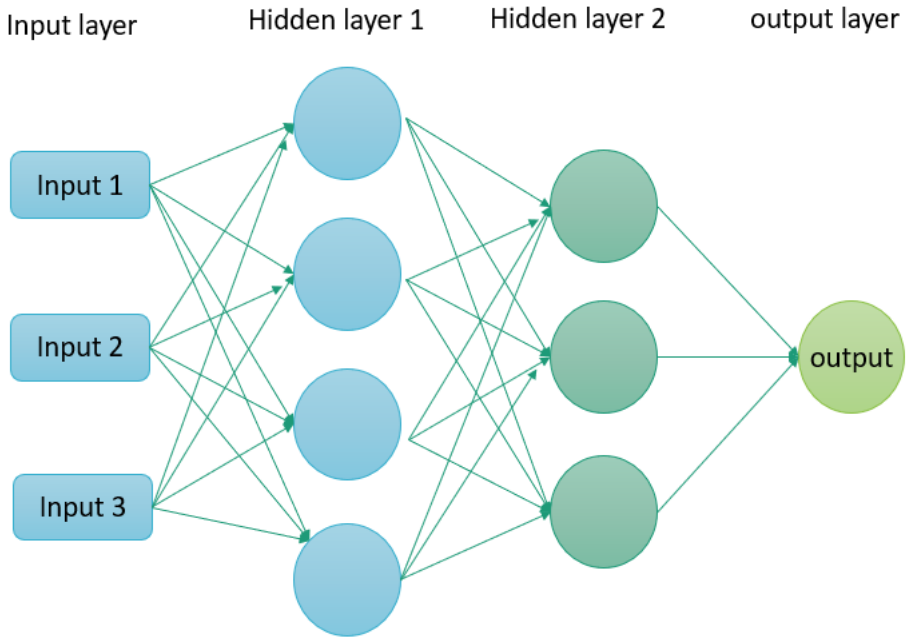


Figure 4-4 Feedforward neural network. The input layer has 4 neurons, one hidden layer which has three neurons, one neuron in the output layer.

The input vector $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ and the weights for each input formed as a matrix

$w = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix}$,the bias vector $b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

The output vector (4-10)

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = f(w.x + b) \tag{4-10}$$

The method of connection between layers and neurons in feedforward ANN is determined by the application which can classify the feedforward ANN into many types like Multilayer perceptron network, Radial basis function networks, generalized regression neural networks, Probabilistic neural networks, Belief networks, Hamming networks, and Stochastic networks. [6]

4.2.1.1 Multilayer perceptron network

MLP has several layers of neurons each layer has a weight matrix, a bias vector, and an output vector. The output of the first layer represents the input of the second layer. The activation function can be chosen for each layer.

There are no rules for determining exactly the number of neurons in the layer and the number of hidden layers. The hidden layer tries to memorize the input pattern rather than learning features of input if the number of neurons of the hidden layer exceeds the number of a training pattern.

4.2.2 Recurrent ANN or feedback ANN

The neuron has close-loop form. There is feedback from the output of neuron to the input. [6]. That represents a sequence of dynamic behavior. RNNs use their internal state as a memory for previous value (memory) to process the sequences of inputs series. [7].The RNNs are used to take care of series of inputs, so the output depends on the current value and the memory. The RNNs diagram illustrated in Figure 4-5

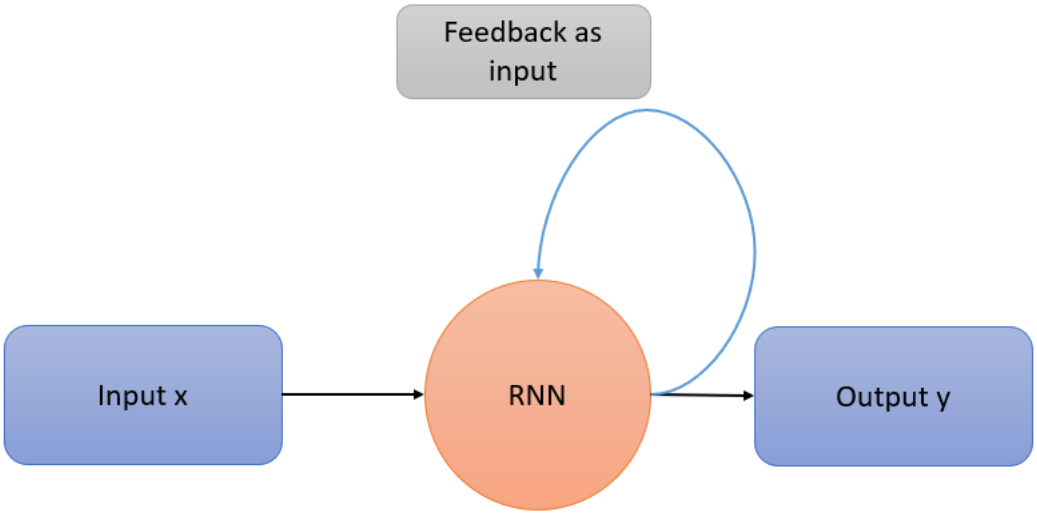


Figure 4-5 Recurrent neural network diagram

The output of RNN for a discrete time can be describe as (4-11):

$$h_{k+1} = f(w.h_k) \qquad (4\text{-}11)$$

h_{k+1} is the output of neuron k, f is the activation function of RNN, h_k is the output of the previous time step. The sequences of time series RNNs are shown in Figure 4-6

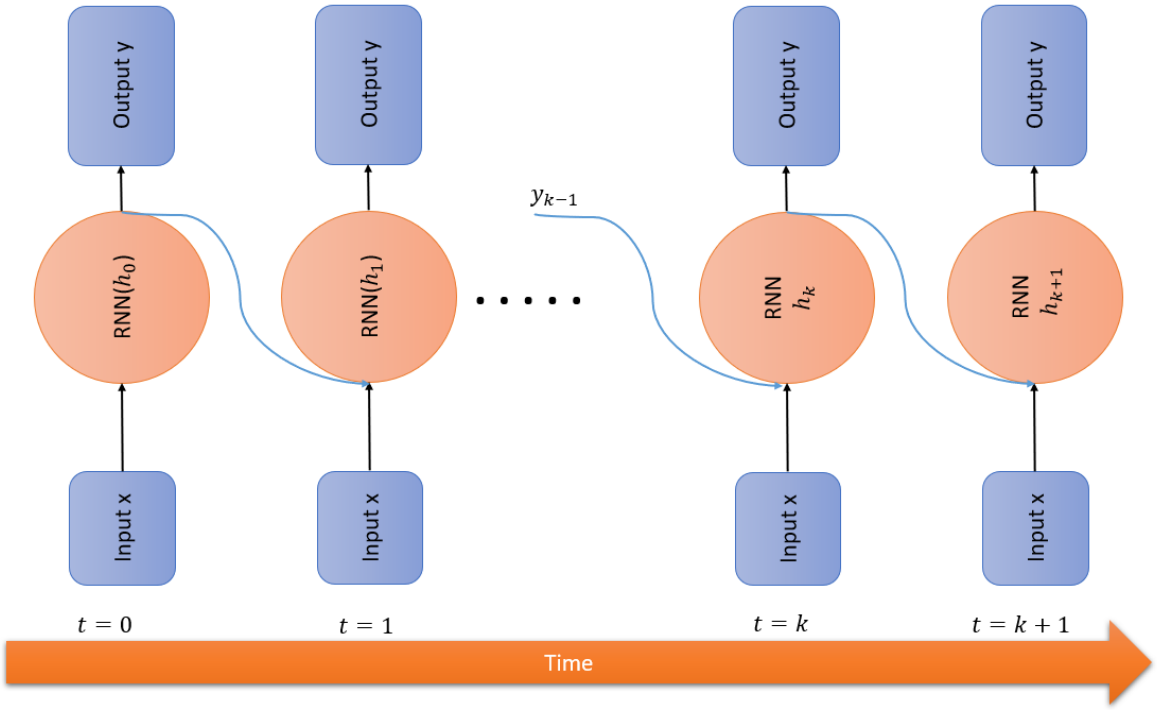


Figure 4-6 RNN model with time series

RNN is widely used and has different applications and different types like, NARX neural network, Elman network, Jordan networks.

4.3 Machine learning

Machine learning is part of artificial intelligence, which focuses on algorithms that learn automatically from experience data. As known training data. The machine learning algorithms try to fit the model based on training data in order to predict the output according to the inputs. The model uses its experience to predict the output [8].

The ML approaches are divided into three categories: Supervised learning, unsupervised learning, and Reinforcement learning, the approach depends on the nature of feedback that is available to learn the model [8].

The learning of neural networks is used to optimize the weights and biases in order to have the best fit of the model [6].

4.3.1 Supervised learning

Supervised learning is used training data inputs and desired outputs to teach the model. Learning algorithm tries to map inputs to output [8] [6].

The inputs are applied to neural network inputs and the learning algorithm tune the weights and bias in an iterative loop to minimize the difference between the neural network and the target data. [6] In this thesis, the case is Supervised learning. The learning data is available.

4.3.2 Ordinary Least squared fitting.

Ordinary Least squared fitting is a method to find the best fit of data set. the best fit can be obtained by minimizing the vertical distance from the data set to the regression line [9].

The mean value for N sample observation $(x_1, x_2, \dots x_n)$ is defined as (4-15) which gives the average value of observations [9].

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{4-15}$$

The variance value gives information about how much the data set fluctuates about the mean value which can be expressed as (4-16)

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})^2 \tag{4-16}$$

The standard deviation σ can be obtained from square root of the variance as (4-17)

$$\sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})^2} \tag{4-17}$$

Linear model of the SISO system can be presented as (4-18)

$$y = ax + b \tag{4-18}$$

The observations are $((x_1, y_1), (x_2, y_2), \dots (x_n, y_n))$. The best fit of the model gives the smallest error which mean smallest value of variance equation (4-19)

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n - (ax_n + b))^2 \tag{4-19}$$

The error that associates with the fitting can be formed as (4-20)

$$E(a, b) = \frac{1}{N}\sum_{n=1}^{N}(y_n - (ax_n + b))^2 \tag{4-20}$$

The function of error has minimum value of error the derivative equal to zero (4-21).

$$\frac{dE}{da} = 0, \frac{dE}{db} = 0 \tag{4-21}$$

The values of a,b which give best fitting obtain from solving (4-21)as equation (4-22)

$$\binom{a}{b} = \begin{pmatrix} \sum_{n=1}^{N} x_n^2 & \sum_{n=1}^{N} x_n \\ \sum_{n=1}^{N} x_n & N \end{pmatrix}^{-1} \begin{pmatrix} \sum_{n=1}^{N} x_n y_n \\ \sum_{n=1}^{N} y_n \end{pmatrix} \tag{4-22}$$

This method is used commonly for simple linear regression to find best fit line for set of statistic data points.

4.3.3 Multiple Linear Regression MLR:

MLR is suitable for the multi-variate systems that have one variable output y and a set of independent x variables as input that impact the output. The formulation of x should be full mathematical rank (x variables are linearly independent) practically uncorrelated. [10]

The purpose of MLR is to find a mathematical model that gives a connection between multi-input independent variables and dependent output variables. MLR is an extension of the OLS method, the formulation of the model is [11] (4-23)

$$\hat{y} = A_1 x_1 + A_2 x_2 + \cdots + A_p x_p + A_0 + e \quad (4\text{-}23)$$

A_p is coefficient regarding the variable x_p, e is the error, A_0 is the constant coefficient or bias.

4.3.4 Mean squired error MSE /Loss cost function.

MSE is a procedure to estimate unsupervised parameters. This method can be used to optimize the neural network parameters (weights and biases). The data set from a system that has x data as input and y data as output. the same data applies to the ANN model, the model gives the prediction of the output $\hat{y}$. The loss or cost function can be described as equation (4-24) [12].

$$c = (\hat{y} - y)^2 \quad (4\text{-}24)$$

The gradient should be calculated in terms of all weights and biases. To do that the derivative of cost function is needed in terms of all weights and biases. It is iterative process to optimize weights and biases which can be summarized as the following steps [6]:

- Initialize the step size or learning rate:lr=0.01
- Initialize the weights and biases randomly: $w_1, w_2, b = np.random.randn(3)$
- Calculate $\hat{y}$ from the ANN model.
- Calculate the loss $c = (\hat{y} - y)^2$
- Calculate the derivative $\frac{dc}{dw_1}, \frac{dc}{dw_2}, \frac{dc}{db}$
- Update weights and biases as the following

$$w_1 = w_1 - \frac{dc}{dw_1}.lr$$

$$w_2 = w_2 - \frac{dc}{dw_2}.lr$$

$$b = b - \frac{dc}{db}.lr$$

- Go to the next iteration step 3.

4.3.5 Adam optimizer

It is called Adaptive Moment estimation which is an optimization process for gradient descent. It is an efficient method for optimizing large parameters which require less memory. This method uses a combination of two algorithm gradient descent with momentum and Root Mean Square Propagation RMSP [13] [14].

Momentum:

The purpose of the Momentum algorithm is to accelerate the gradient descent by taking into account the exponentially weighted average of gradient which allows accelerating moving to a minimum. The idea is that the gradient with momentum has two jobs [13] [15]:

- Remembers the update of the aggregate of gradients m_k at each iteration. The initial value of $m_0 = 0$. As describes in the equation (4-25)
- Determines the next update and the previous update of gradient as a linear combination. equation (4-26)

$$m_k = \beta m_{k-1} + (1-\beta)\frac{dL}{dw_k} \quad (4\text{-}25)$$

$$w_{k+1} = w_k - \alpha . m_k \quad (4\text{-}26)$$

Where: m_k is the aggregate of gradients at time step k. w_k is the weight at time step k.

L is the loss function, α is learning rates. β is moving average parameter (constant value =0.9)

Root Mean Square Propagation RMSP

RMSP is an adaptive learning algorithm to calculate the weight by using the sum of squares of past gradients with an exponential moving average. Calculating the weights can be performed by using equations (4-30) and (4-31) [13]

$$\vartheta_k = \beta \vartheta_{k-1} + (1-\beta) * \left(\frac{dL}{dw_k}\right)^2 \quad (4\text{-}30)$$

$$w_{k+1} = w_k - \frac{\alpha_k}{\sqrt{\vartheta_k + \varepsilon}} * \frac{dL}{dw_k} \quad (4\text{-}31)$$

Where: α_k learning rates at time k. $\varepsilon = 10^{-8}$ small positive number to prevents division by zero. ϑ_k sum of the square of past gradients. β is moving average parameter (constant value =0.9). L is loss function. w_k is the weight at time step k.

Adam

The rate of gradient descent is controlled to minimize the oscillation when the weights reach the global minimum. The optimizer will take enough step size in order to pass the local minima. That will allow reaching the global minimum efficiently. The equations (4-30) and (4-31) can be written as (4-32) and (4-33) [13] [15].

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\frac{dL}{dw_k} \quad (4\text{-}32)$$

$$\vartheta_k = \beta_2 \vartheta_{k-1} + (1-\beta_2) * \left(\frac{dL}{dw_k}\right)^2 \quad (4\text{-}33)$$

The initial values of $m_0, \vartheta_0 = 0$ tend to be biased towards zero and the β_1, β_2 tend to be 1. The optimizer computes the bias-corrected m_k, ϑ_k each iteration and controlling the weights to reach the global minimum. That can be performed by formula (4-34) [13]

$$\widehat{m_k} = \frac{m_k}{1-\beta_1^k}, \widehat{\vartheta_k} = \frac{\vartheta_k}{1-\beta_2^k} \quad (4\text{-}34)$$

The weight can be updated by using formula (4-35)

$$w_{k+1} = w_k - \widehat{m_k}\frac{\alpha_k}{\sqrt{\widehat{\vartheta_k}+\varepsilon}} \quad (4\text{-}35)$$

4.4 Levenberg–Marquardt training algorithm

This method is designed to work with a loss function. It uses gradient vector and Jacobian matrix. The loss function takes the form of a sum of the squared errors as (4-36) [16].

$$f_{loss} = \sum_{i=1}^{n} e_i^2 \quad (4\text{-}36)$$

Where e is error, n number of training samples.

The Jacobian matrix is (4-37)

$$J_{i,j} = \frac{de_i}{dw_j} \quad (4\text{-}37)$$

W is the weight parameter. For i=1,2,…, n sample and j=1,2,…,m parameter

The Levenberg-Marquardt algorithm is (4-38):

$$w_{i+1} = w_i - \left(J_i{}^T . J_i + \lambda_i I\right)^{-1} . (cJ_i^T . e_i) \quad (4\text{-}38)$$

λ is the damping factor, I is the identity matrix. i=0,1,…n sample.

Levenberg-Marquardt algorithm diagram is shown in the Figure 4-7 [16]

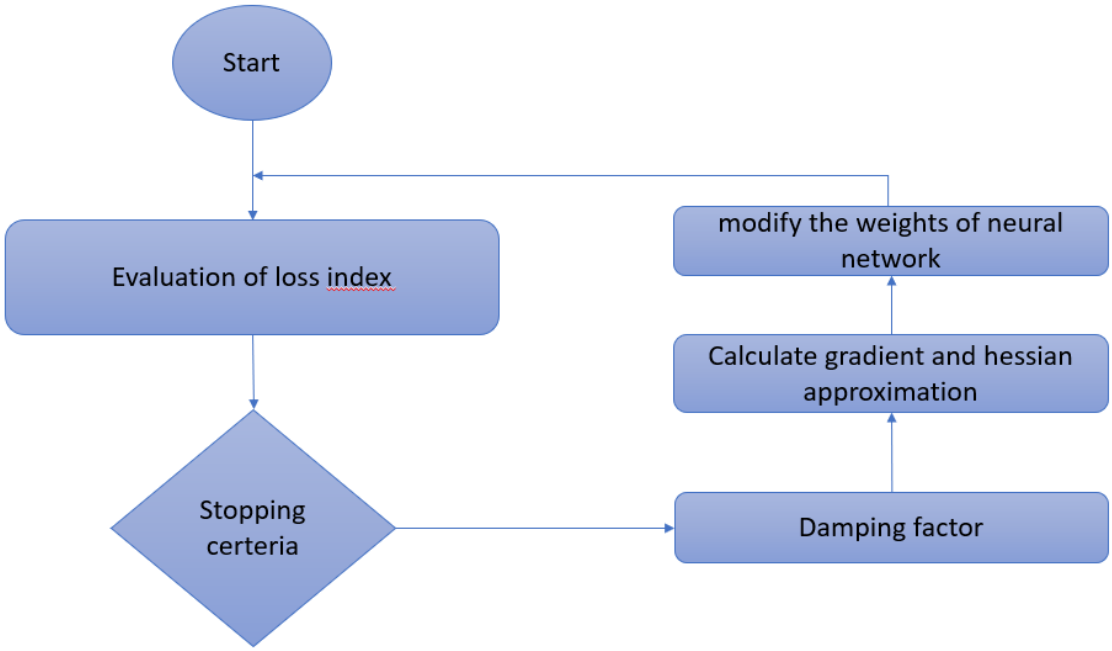


Figure 4-7 Levenberg-Marquardt algorithm diagram

4.5 Nonlinear autoregressive exogenous (NARX) model

NARX uses the current and past values of inputs and outputs series. To get exactly predict the model contain an error term related to the knowledge of other terms not available at the current value of time series. The NARX model can be written as (4-39) [17].

$$y_t = F(y_{t-1}, y_{t-2}, \dots\ y_{t-n}, u_t, u_{t-1}, \dots u_{t-n}) + \varepsilon \qquad (4\text{-}39)$$

y_t is the current value of the output and y_{t-1} is the previous value of the output, u_t is the current input and u_{t-1} is the previous input. ε is the error term. The F function is a non-linear function, it can be a neural network function or any function [17].

4.6 NARX neural network model

The architecture of the NARX network is proposed to deal with previous values. Therefore, it is classified as a recurrent neural network. This kind of neural network is suitable to work with time series, which is a powerful method to deal with dynamic systems. The NARX model has the ability to process with a long memory component, where the past event has an impact on the future [18]. The block diagram of the NARX neural network is shown in Figure 4-8.

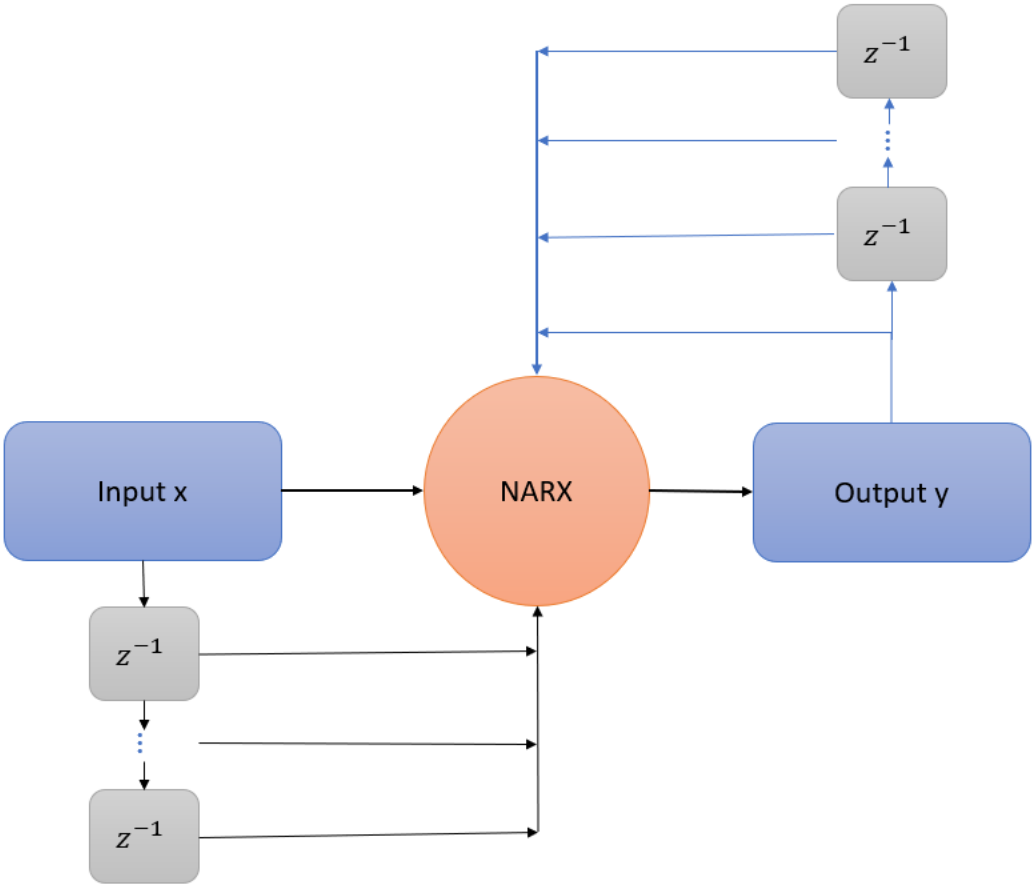


Figure 4-8 NARX neural network block diagram Z^{-1} is the previous value.

The NARX neural network can be implemented as feedforward with embedded memory. The embedded memory will be inputs and outputs which presents tapped time delay. Leaning NARX neural network is more effective than other neural networks (better gradient descent) and it converges faster than another neural network [18].

4.7 Neural network performance evaluation

After training a neural network model, it is important to check the performance of the network. Mean squared error MSE gives information about the average squared difference between data target and neural network model. Lower value refers to better performance [19].

Regression R values show the correlation between data target and neural network model. R=1 means the correlation is 100%. 0 means no correlation at all [19].

It is possible to evaluate the performance by running the neural network model in parallel with the process to see the neural network model behaves.

5 Dynamic system Modelling

5.1 Using forward Euler and backward

The response of the chemical dosing system shows a slow, stable process with a time delay of the input, the system's behavior can be presented by first-order transfer function as equation (5-1) [1].

$$H(s) = \frac{Y(s)}{U(s)} = \frac{k}{T_c+1} \tag{5-1}$$

Equation (5-1) can be written as (5-2)

$$Y(s)(T_c s + 1) = U(s)k \tag{5-2}$$

Move to time domain by inverse Laplace transform (5-3)

$$T_c \dot{y}(t) + y(t) = k.u(t) \tag{5-3}$$

The equation (5-3) is a continuous-time equation that can be solved in different ways. Forward and backward Euler methods can be good approaches to discretize (5-3). equation (5-4) is forward Euler form and equation (5-5) is backward form [4].

$$y_{k+1} = y_k + \frac{dt}{T_c}(K * u_k - y_k) \tag{5-4}$$

$$y_{k+1} = \frac{k.dt}{(T_c+dt)}u_k + \frac{T_c}{T_c+dt}y_k \tag{5-5}$$

5.2 Using neural network

Equations (5-4) and (5-5) show that it is possible to predict the next step if the current step is known. In order to design ANN to model the system, the current output will be used to predict the next step. These tricks can be performed to include time series in ANN Figure 5-1 and it can be written as (5-6), (5-7).

$$y_{k+1} = f(u_k, y_k) = y_k + \frac{dt}{T_c}(k.u_k - y_k) = \frac{dt}{T_c}k.u_k + \left(1 - \frac{dt}{T_c}\right)y_k \tag{5-6}$$

$$y_{k+1} = f(u_k, y_k) = \frac{k.dt}{(T_c+dt)}u_k + \frac{T_c}{T_c+dt}y_k \tag{5-7}$$

The T_c term in equations (5-6) and (5-7) is in the denominator, so the T_c should not be zero. This condition is approved in this system.

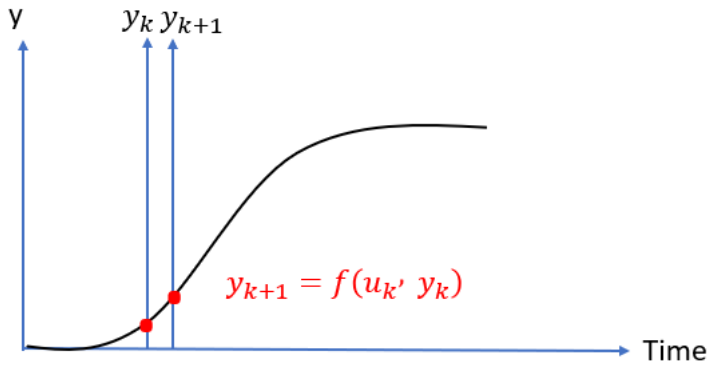


Figure 5-1 First order transfer function response.

The equations (5-6) and (5-7), have two inputs (u_k, y_k) and two constants. The constant terms are $\frac{dt}{T_c}k$ and $\left(1-\frac{dt}{T_c}\right)$ in the equation (5-6), and $\frac{k.dt}{(T_c+dt)}$ and $\frac{T_c}{T_c+dt}$, these constants can presents the weights of the input neuron in the neural network, consequently, these two equations can be written in a neuron form in a neural network as shown in the equation (5-8).

$$y_{k+1} = f(u_k = x_1, y_k = x_2) = w_1x_1 + w_2x_2 + b \text{ , b=0} \qquad (5\text{-}8)$$

Forward and backward Euler equations can be formed as a neuron equation (4-1) as illustrated in the equations (5-8). The MIMO dynamic system model can be formed as neural network equation (4-10).

5.3 Using Machine learning

The Euler equation can be converted into a linear equation. The prediction of the next step has a linear form. The inputs will be the current output and current input. It is returned to the form of the equation (5-8). The model can be fitted as a linear form.

5.4 Recurrent neural network

The recurrent neural network is used to store the current value in internal memory and use it to predict the next future value. exactly as shown in the Euler equations. The value of the predicted output relays on the current output and input, the ANN model can be illustrated as Figure 5-2

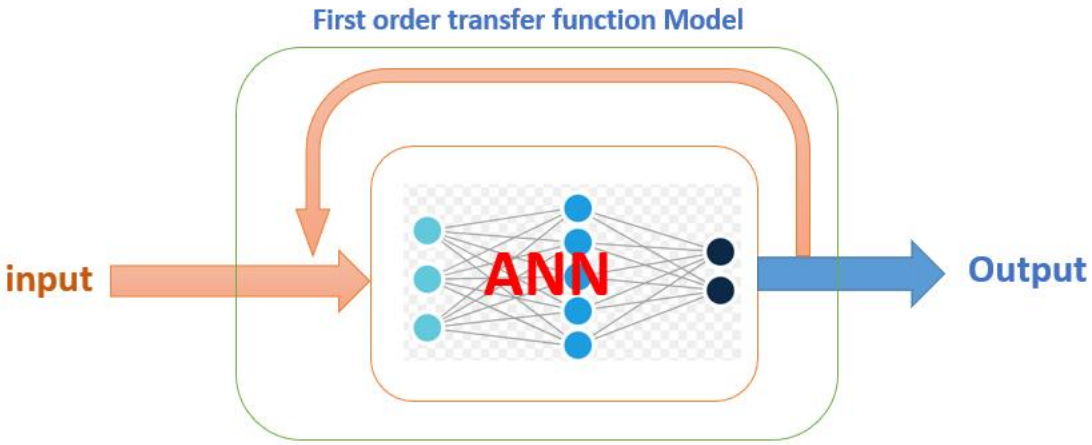


Figure 5-2 Diagram of first order transfer function by Multilayer perceptron network.

The mathematical form of ANN can be described as (5-9)

$$y_{k+1} = ANN = f(u_k, y_k) \tag{5-9}$$

$u_k$is the input at time k and y_k is output at time k.

The input layer of the network has two variables y_k, u_k. There is no specific rule to determine the number of neurons, layers, and activation function. Many layers configurations have been tested by sequential network and functional API network.

Equations (5-6) can be formed as (5-10).

$$y_{k+1} = f(u_k, y_k) = \left(1 - \frac{dt}{T_c}\right) y_k + k.\frac{dt}{T_c} u_k = w_1 u_k + w_2 y_k \tag{5-10}$$

Where $w_1 = K.\frac{dt}{T_c}$, $w_2 = \left(1 - \frac{dt}{T_c}\right)$

Equation (5-10) presents one neuron with two input and linear activation functions. The same procedure can be performed for (5-7) to get neuron equation as (5-11)

$$y_{k+1} = f(u_k, y_k) = \frac{k.dt}{(T_c+dt)} u_k + \frac{T_c}{T_c+dt} y_k = w_1 u_k + w_2 y_k \tag{5-11}$$

Where $w_1 = \frac{k.dt}{(T_c+dt)}$, $w_2 = \frac{T_c}{T_c+dt}$, $w_1\ and\ w_2$ form the weights of the inputs in a neuron equation.

The artificial neural network diagram that presents the system is shown in Figure 5-3.

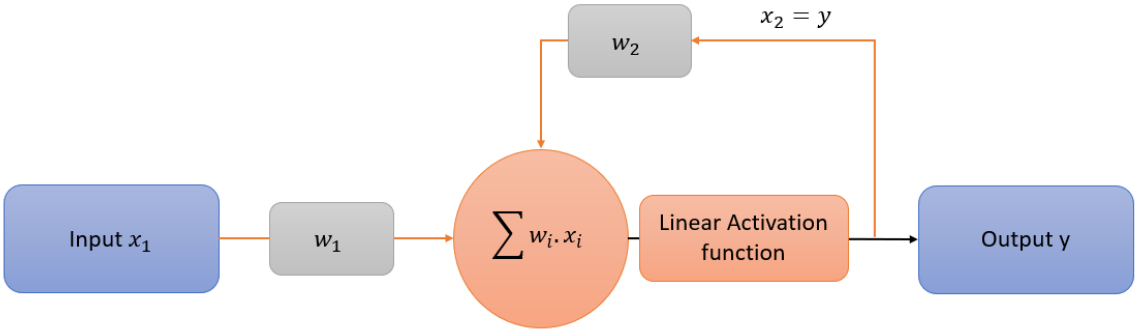


Figure 5-3 Artificial neural network diagram for first order transfer function with one input.

As an example **Modeling of the SISO system using linear activation function** section can be referred.

6 Simulator for Chemical dosing System

The simulator is important to test the model, and its behavior, Applying the control signals directly to the process is costly and sometimes, it is dangerous, but no problem for a simulator. The simulator for the water dosage process is developed by using first order transfer function. The first step is a transfer function of the MISO system without delay and the second step the time delay will be included in the process.

6.1 MISO system simulation without time delay

The simulator is also used to generate the training data, the parameter of the simulator is indicated in Table 6-1. The values of gain and time constant from [1].

Table 6-1 System parameter that used for simulator.

Materials	PIX	PAX	POL
First order			
Time constant Tc [min]	10	30	20
Gain k	-0.2	-0.3	-0.02

The simulator is developed by using the forward or backward Euler method to discretize the continuous system.

The equations that are used in the code are included in the forward Euler and backward sections.

The Python code for the transfer function is:

```
def forward_euler(ts,Kc,u_k,Tc,y_k):
    dy_dt=(Kc*u_k-y_k)/Tc
    return y_k+dy_dt*ts
```

Simulator Python code in the Appendix.

6.1.1 Simulation Results

The single input and corresponding output as SISO are shown in Figure 6-1 Figure 6-2 Figure 6-3. The multi-inputs are fed the system and correspond single output is shown in Figure 6-4.

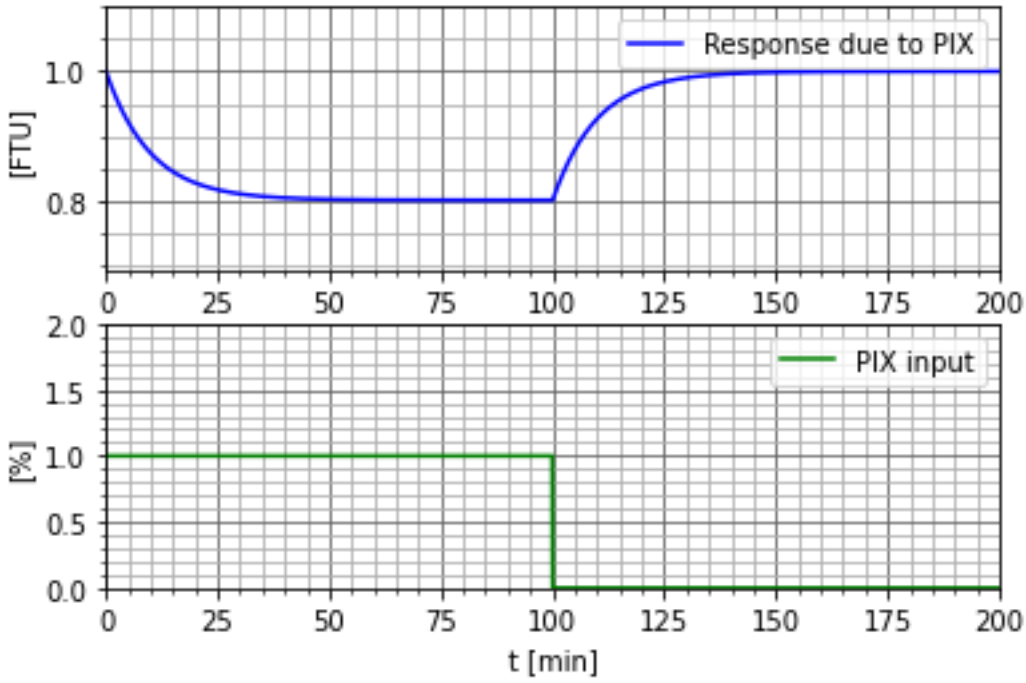


Figure 6-1 Simulator response due to PIX without time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the input of PIX with time, Y-axis is the PIX materials as percentage, x-axis is a time in minute.

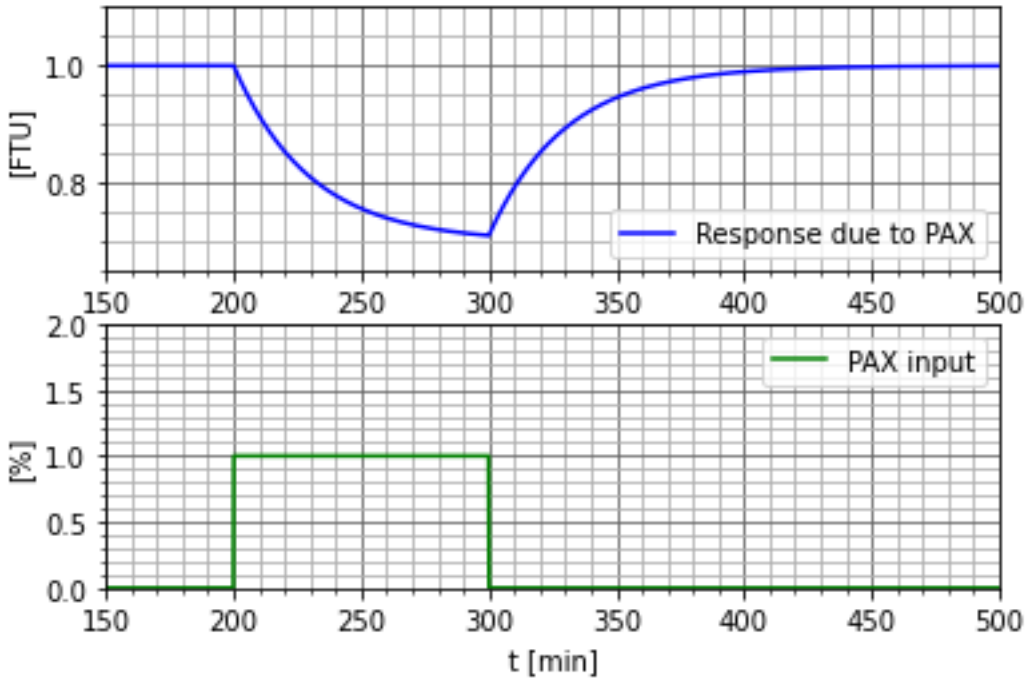


Figure 6-2 Simulator response due to PAX without time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the input of PAX with time, Y-axis is the PAX materials as percentage, x-axis is a time in minute.

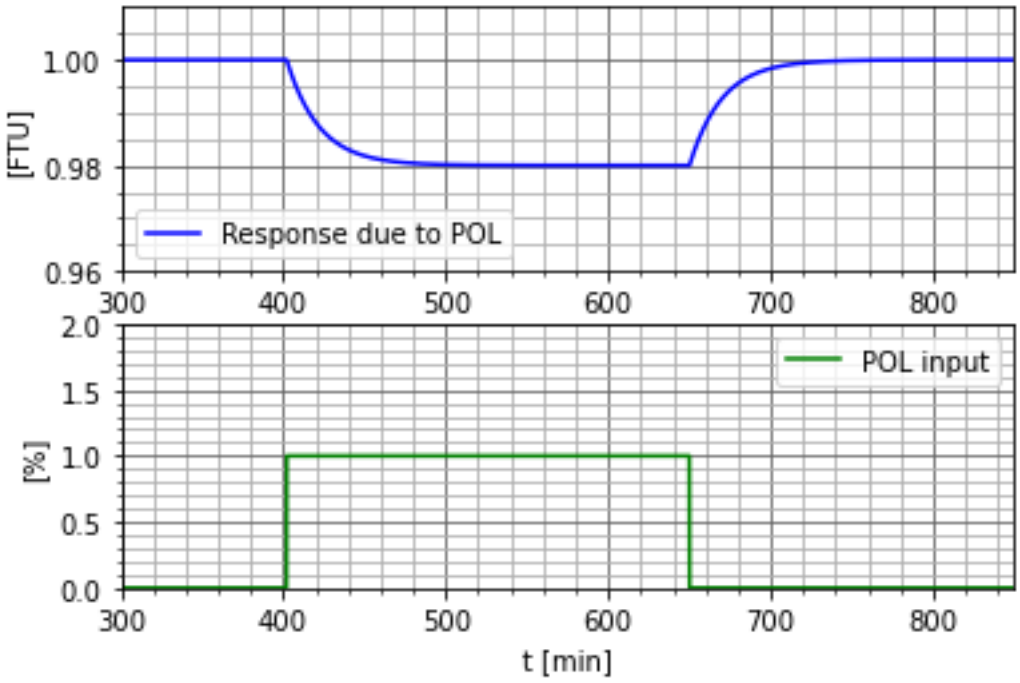


Figure 6-3 Simulator response due to POL without time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the input of POL with time, Y-axis is the POL materials as percentage, x-axis is a time in minute.

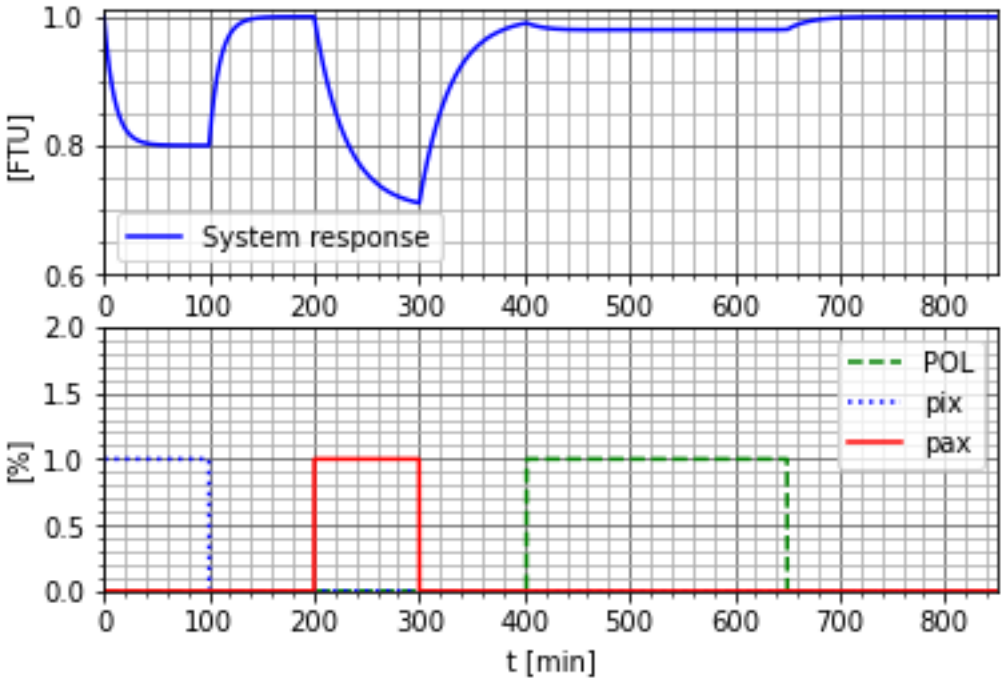


Figure 6-4 Simulator response due to POL, PIX, PAX. without time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the inputs POL, PIX, PAX with time, Y-axis is the POL, PIX, PAX materials as percentage, x-axis is a time in minute.

The step function is applied to each input PIX, PAX, POL to see how each input affects the output.

6.1.2 Simulation discussion

The SISO system response in Figure 6-1, Figure 6-2, Figure 6-3, Figure 6-4 show that when the step function is applied the output starts to respond to this input and after a time equal to approximately 5 times of time constant $t = 5 * \tau$ reach to steady state. The system is stable and there is no overshoot. The response of PIX is faster than PAX, POL due to a smaller time constant of PIX. Time constant represents how much time the system needs to reach the steady state. A higher value of time constant leads to slower response.

POL response has a smaller gain than PIX, PAX. And PAX has a higher gain than PIX and POL. Consequently, POL has the lowest impact and PAX has the highest impact. As shown in Figure 6-4.

6.2 MISO system with time delay simulation

A real process may have a time delay, so it is important to include time delay in the simulation.

The simulator is used to generate the data with time delay, the parameter of the simulator is indicated in Table 6-2 the values from [1].

Table 6-2 System parameter of the time delay simulator

Materials	PIX	PAX	POL
Time constant Tc [min]	10	30	20
Time delay[min]	10	30	60
Gain k	-0.2	-0.3	-0.02

6.2.1 Simulation method

These inputs fed the system with a time delay as indicated in Table 6-2

The simulator will handle the time delay as the following [4]:

- Initial value of the delay. It can be zero.
- Create an array for delay (delay_array) that has time delay / time sample element
- Give initial value to delay array
- System loop
- Read the input value from the last element
- Rotate the delay_array one element to the right
- Set the value of the input to the first element

The delay function diagram is shown in Figure 6-5. The value of the input stored in the first element will reach the last element after a time equal to the number of elements in the

delay array *time step. When the value reaches the last element, the model will use this value as input of the system, and the system will react according to this value [4].

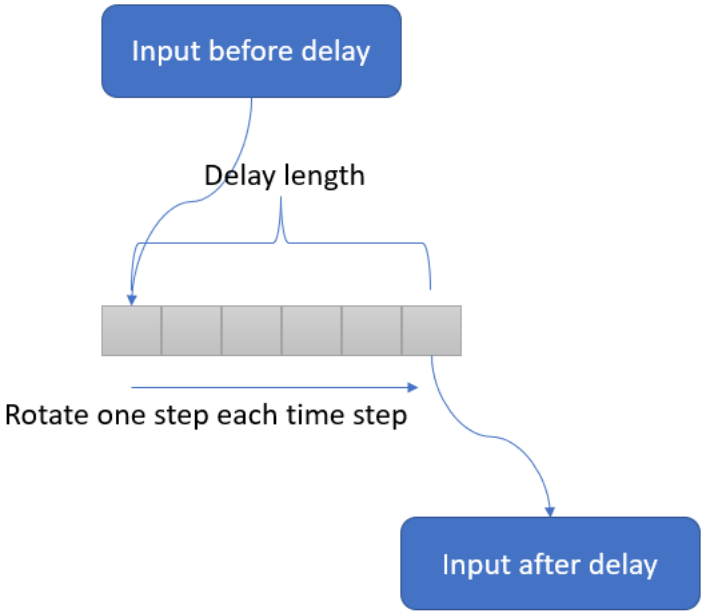


Figure 6-5 Delay function diagram, the number of the elements in the delay length equal to delay/time sample

6.2.2 Simulation results

Python language is used to implement the first order transfer function with time delay. And the software is Spyder. The system is fed as SISO to check the time delay function. The single input and corresponding output as SISO are shown in the Figure 6-6, Figure 6-7, Figure 6-8. The multi-inputs are fed the system and correspond single output is shown in the Figure 6-9.

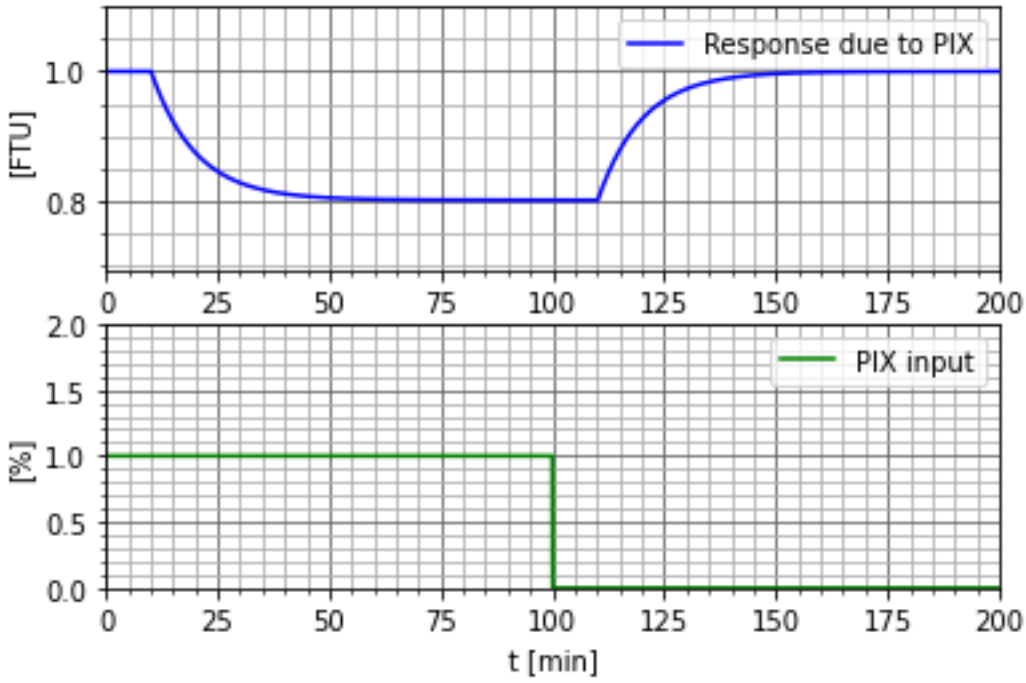


Figure 6-6 Simulator response due to PIX with time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the input of PIX with time, Y-axis is the PIX materials as percentage, x-axis is a time in minute.

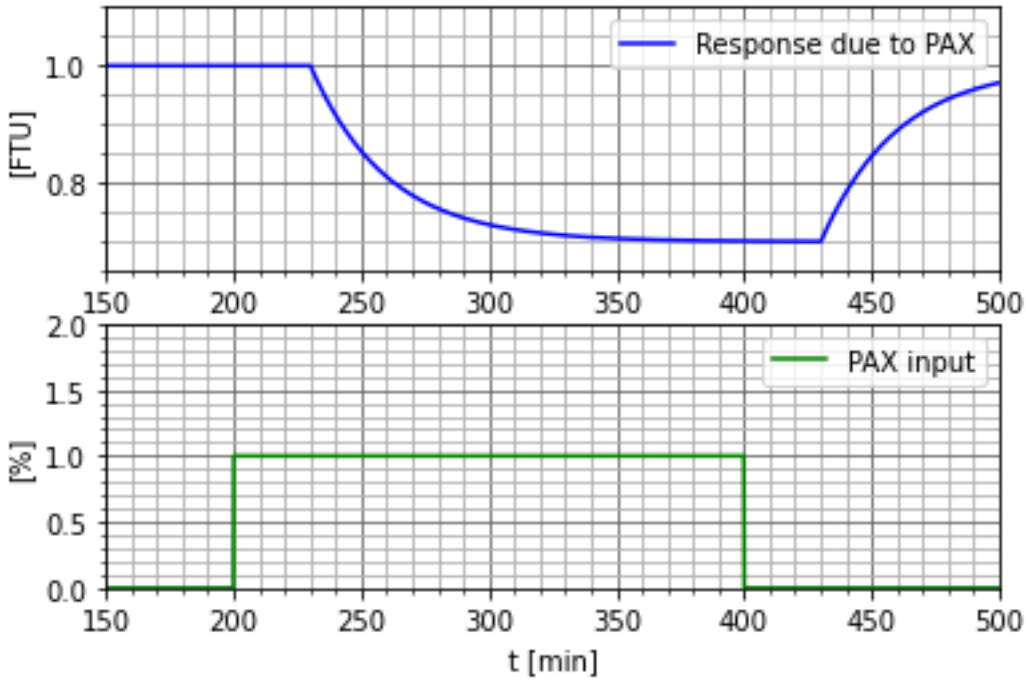


Figure 6-7 Simulator response due to PAX with time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the input of PAX with time, Y-axis is the PAX materials as percentage, x-axis is a time in minute.

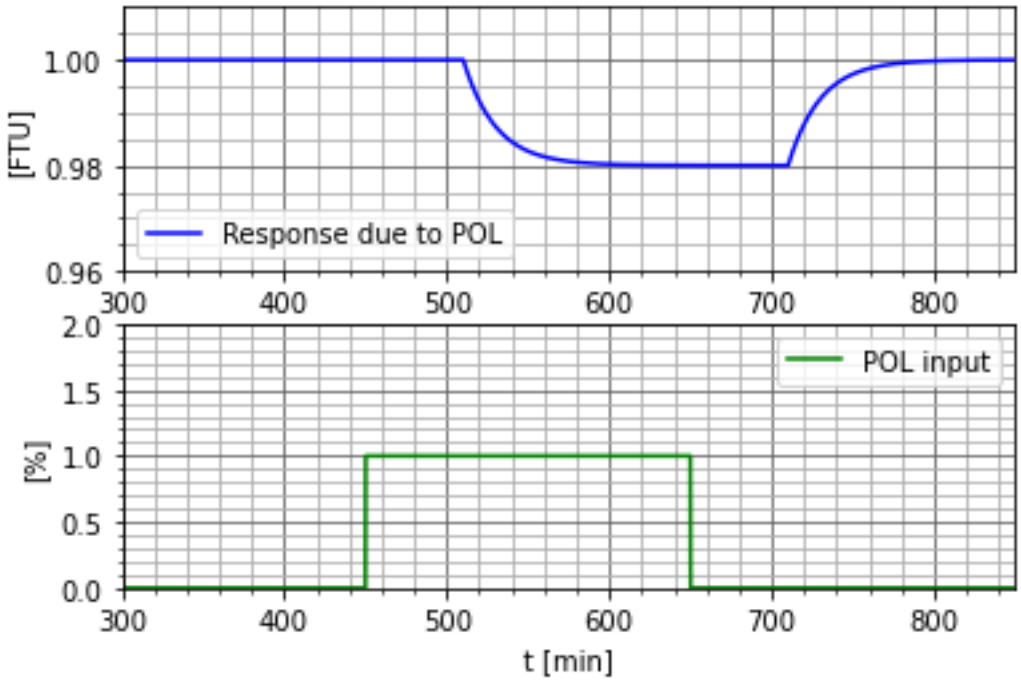


Figure 6-8 Simulator response due to POL with time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the input of POL with time, Y-axis is the POL materials as percentage, x-axis is a time in minute

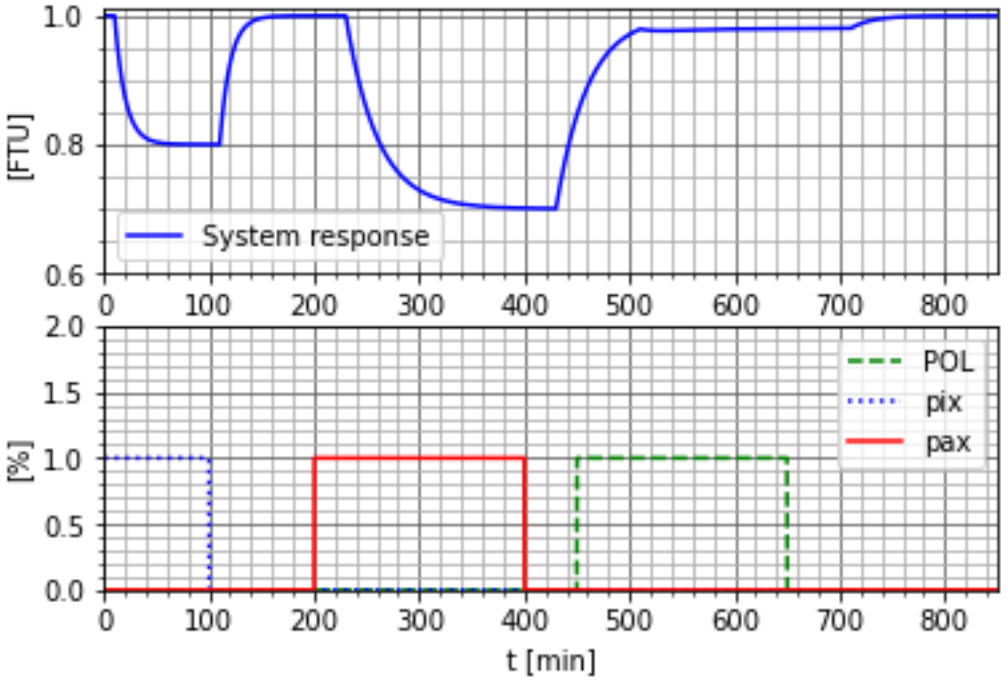


Figure 6-9 Simulator response due to POL, PIX and PAX with time delay. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the inputs POL, PIX, PAX with time, Y-axis is the POL, PIX, PAX materials as percentage, x-axis is a time in minute.

6.2.3 Simulation discussion

The Figure 6-6 Figure 6-7Figure 6-8Figure 6-9 show clearly that the response of the system is delayed. The figure gives the amount of delay as shown in Figure 6-10, the time delay of PIX graphically=10 minutes, time delay from parameters =10 minutes Table 6-2, the same procedure applied to the PAX, and POL. The results from graph equal to parameters. That mean the time delay of the simulator is working properly.

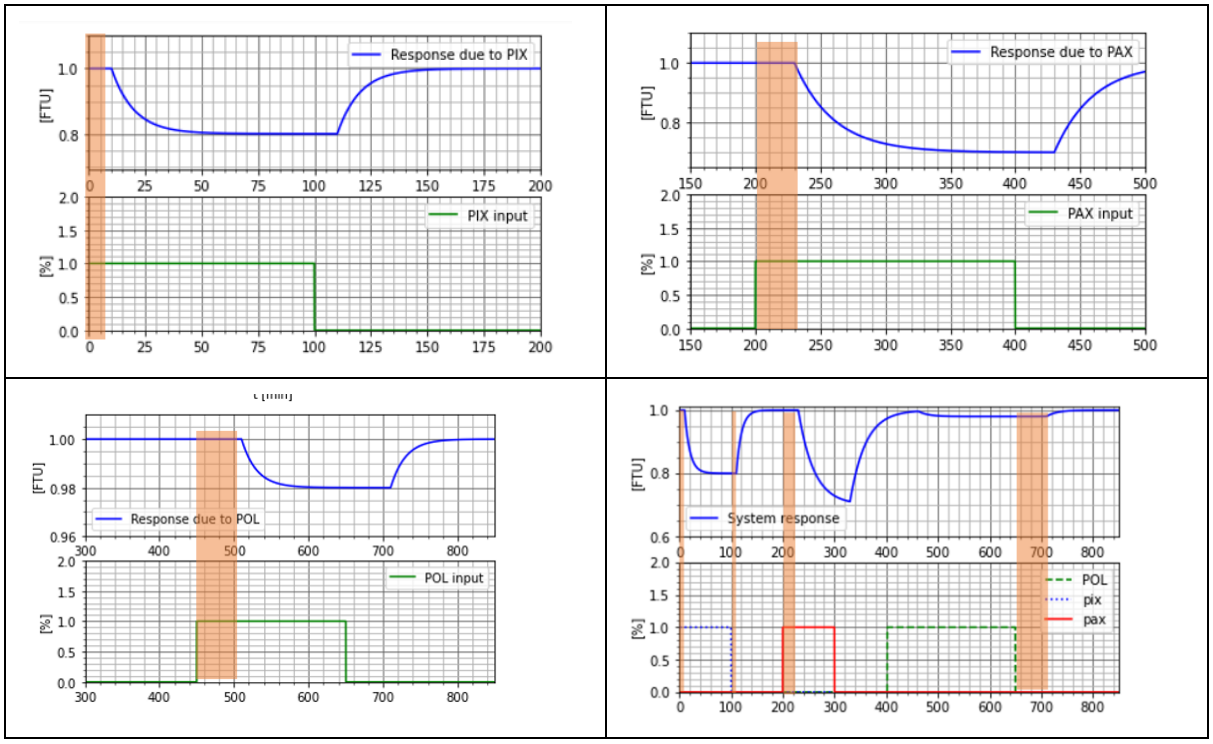


Figure 6-10 System response with time delay for SISO system and MISO system, the orange rectangle represents the amount of delay. Y-axis is the turbidity of the water as FTU, x-axis is a time in minute. The bottom figure is the inputs with time.

7 Time delay estimation algorithm

Time delay estimation is a critical issue in system identification. The time delay influences the model as much as the model order. [20] The process with time delay does not guarantee a good model. Not correct time delay estimation can lead to a mismatch between the model and real process in some systems. Due to the connection between process input-output, and model input-output.

The time delay can be estimated by observing the inputs and output. This method works only with step inputs. The estimation can be executed by the following steps:

- Set all inputs to the constant values
- Wait until the output of the system to be stable.
- Give step to the one of the inputs.
- Count the time until get response in the output to this input.
- Time delay equals to the time from input step to get change in the output.
- Verify the result.
- Repeat the same procedure to the other inputs to calculate time delay to the other signals.

To be sure the result should be verified, when the output starts to change, it will continue changing to reach the steady state. If the output changed for just one time step that mean it is just a noise, and final value of time delay is not reached yet. It is possible to continue counting the time or sent error message to repeat the estimation.

The sequence diagram of the time delay estimation is shown in Figure 7-1

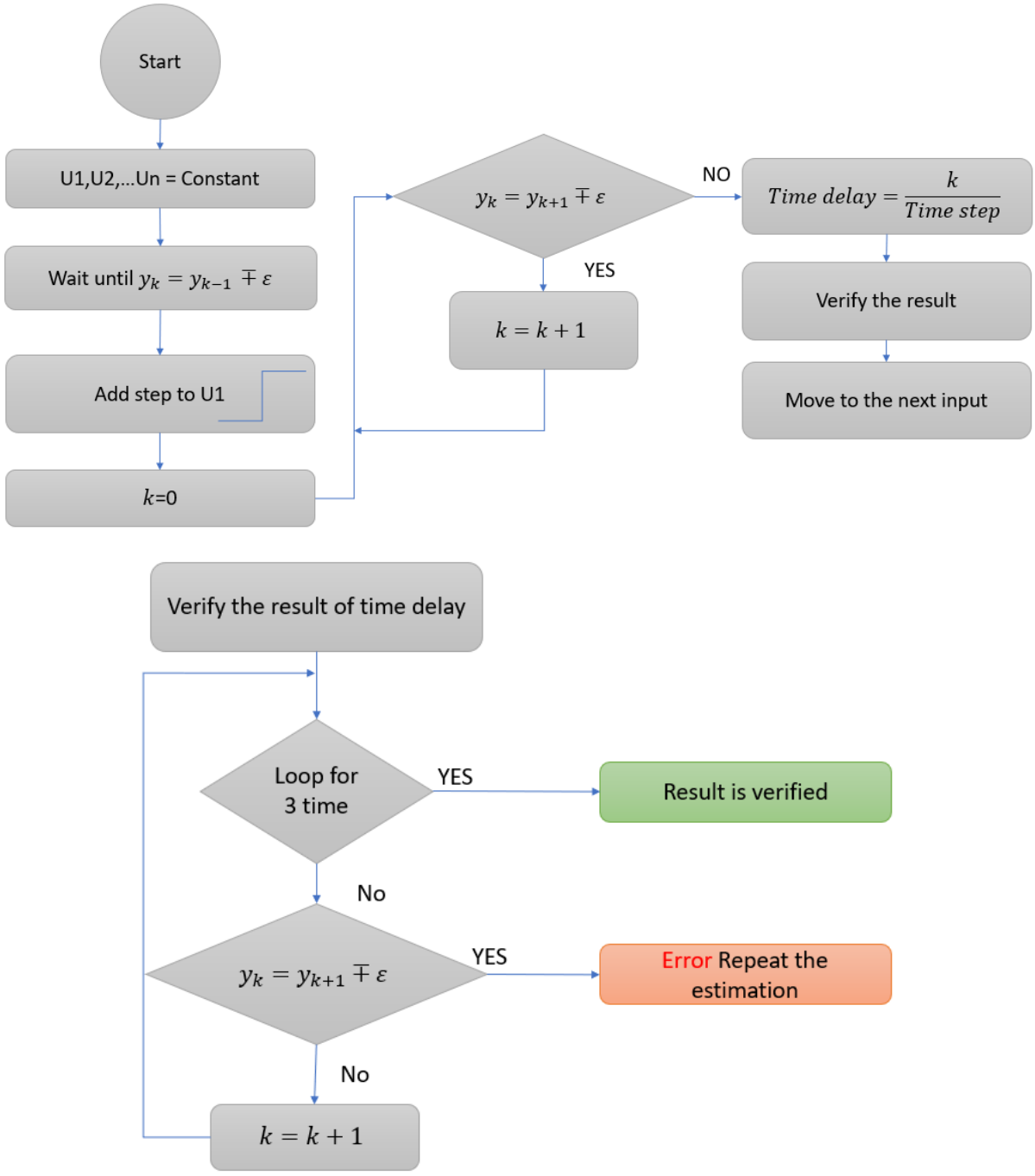


Figure 7-1 The sequence diagram of the time delay estimation algorithm for one input signal, ε is an error term

The error term is added to include the noise, the verification loop is needed to be sure the changing is continuous for the next 3 steps. when the output starts responding to the changing of inputs. The output continues changing until reach the steady state, which required many steps much larger than 3 steps. Verifying the results with 3 steps is enough to avoid the error due to spike noise. If data has no spikes, the verification part is not needed.

7.1 Method

The time delay estimation is implemented by using Python and the software is Spyder. The clean data is needed to do estimation. The data from estimation is provided from Simulator for Chemical dosing System section. The data in Figure 6-10 is exported to CSV file.

Python code for time delay estimation:

```
# load training data
train_df = pd.read_csv("xtrain_data.csv")
u1=train_df[['U_pax']].values
train_df = pd.read_csv("ytrain_data.csv")
y=train_df[['TURB']].values
delay= np.zeros(len(y))
delay_count=0
Condition_u=0
eps_u=0.1 # noise margen rang
eps_y=0.00001# noise margen rang
for k in range(0, len(y)-5):


   if (y[k+1]>=y[k]-eps_y and y[k+1]<=y[k]+eps_y):# Condition_y
      if (u1[k+1]<u1[k]-eps_u or u1[k+1]>u1[k]+eps_u):#Condition_u
         Condition_u=1


      if Condition_u==1:
         delay[k]=1
         delay_count+=1


      #print (k)
   else:
      Condition_u=0
```

```
print ("time delay=")
print (int (delay_count*ts))
```

The data that used in the time delay estimation is generated from the simulator (MISO system with time delay simulation). Inputs and output data is plotted in Figure 7-2

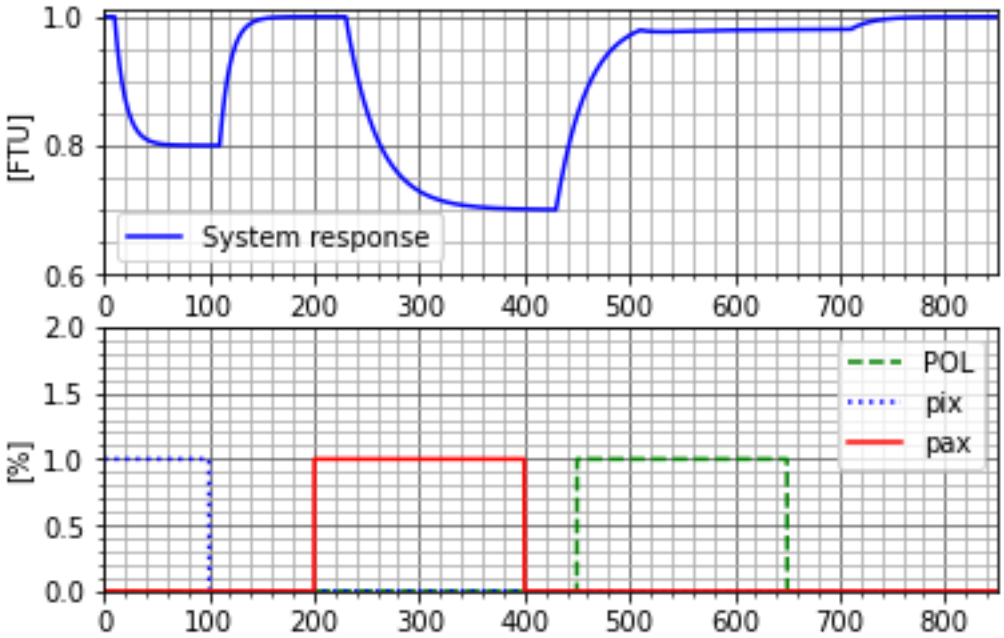


Figure 7-2 Time delay estimation data set. The input is a step function of POL, PIX, PAX, the output is a system response due to inputs. The top figure is the response with time, Y-axis is the turbidity of the water as FTU, x-axis is a time in minutes. The bottom figure is the inputs POL, PIX, PAX with time, Y-axis is the POL, PIX, PAX materials as percentage, x-axis is a time in minutes.

7.2 Simulation Results

The estimation has been performed for each input individually. the length of time is plotted with the input signals and the output response signal. The results are shown in Figure 7-3, Figure 7-4, Figure 7-5 ,the noise margin range for input equal to eps_u=0.1, the noise margin range for output equal to eps_y=0.00001.

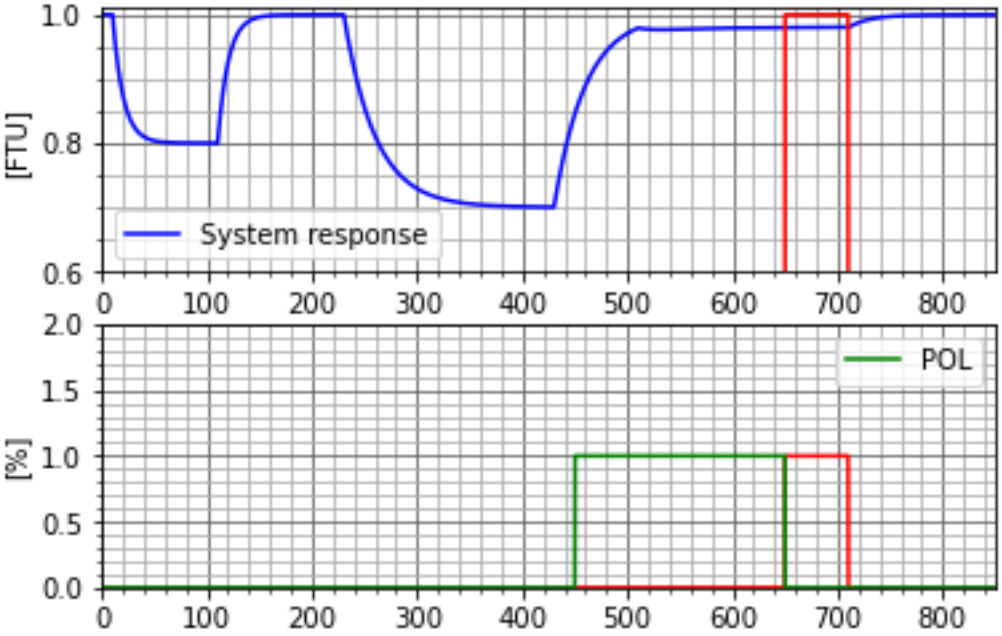


Figure 7-3 Time delay estimation for POL input. Red color represents the delay period. The time delay estimation is calculated by program, and it is equal to 60 minutes.

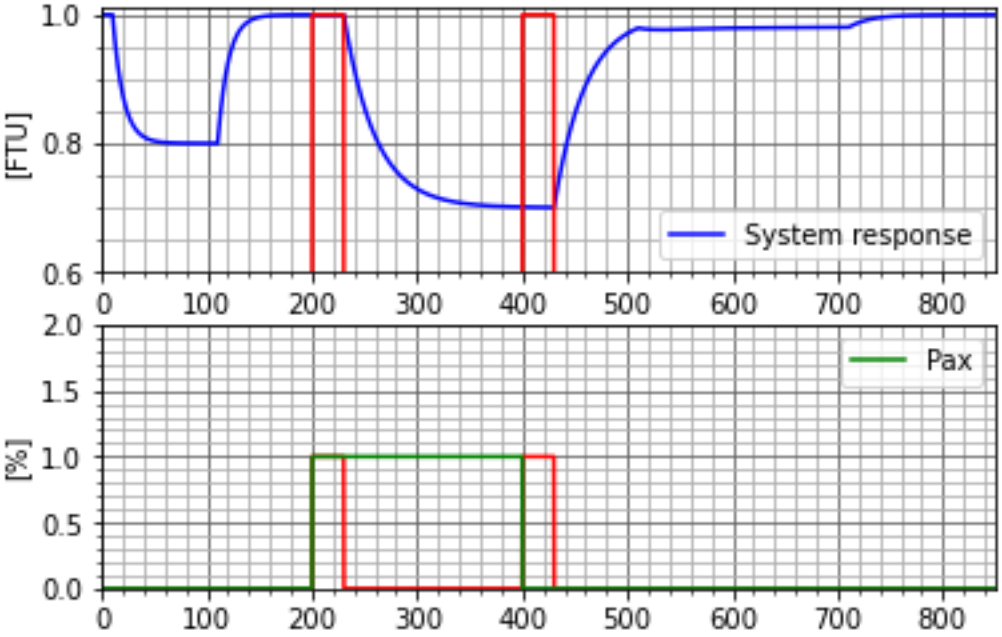


Figure 7-4 Time delay estimation for Pax input. Red color represents the delay period. The time delay estimation is calculated by program, and it is equal to 30 minutes.

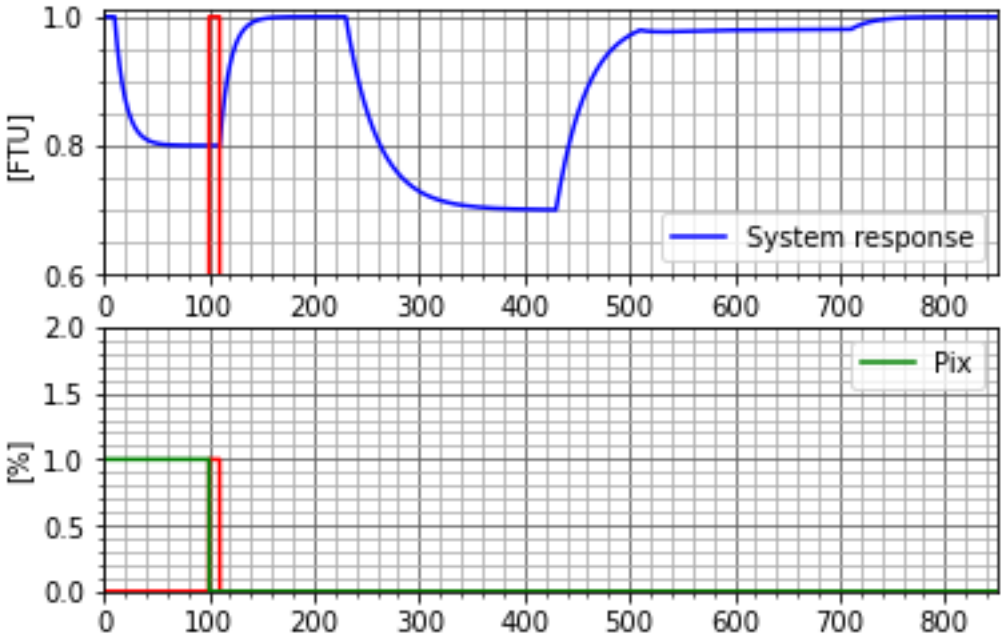


Figure 7-5 Time delay estimation for Pix input. Red color represents the delay period. The time delay estimation is calculated by program, and it is equal to 10 minutes.

7.2.1 Simulation Discussion

The results show that the estimation algorithm is working properly. The estimated time delay is approximately equal to real time delay. The comparison between results and Table 6-2. In the simulated data there is no noise taking in account. in the time delay estimation assumed that the data has noise in both input and output. To do estimation correctly the noise margin should be very small compared to the response gain (smaller than 0.3% of the gain the results from simulation). If there is influence of more than one input, the estimation will give fail result or will not success to do estimation. The advantage of this method that the estimation depends on the data from input and output, the disadvantage is very sensitive to noise and input data.

8 Simulator model development using Artificial neural network

The training and testing data used in this section is from the simulator that was built in the previous section. The reasons are to simplify the model and it is easy to control the input of the system in the simulation while it is not possible with real data. The noise is not included in the simulation that will give a possibility to judge the ANN model compared to simulated model.

8.1 Methods

The method that is used in this thesis to train the model and using the trained model is summarized as the following steps:

1. Create the training data set from the simulator and store it in CSV file.
2. Creating the model and using the training data set.
3. Save the model on the hard disk.
4. Do the simulation with new inputs different from training data.
5. Using the new inputs to feed the neural network model.
6. Compare the results from the simulation model and the neural network model. To see how the neural network model is accurate.

The block diagram of developing neural network model is shown in Figure 8-1

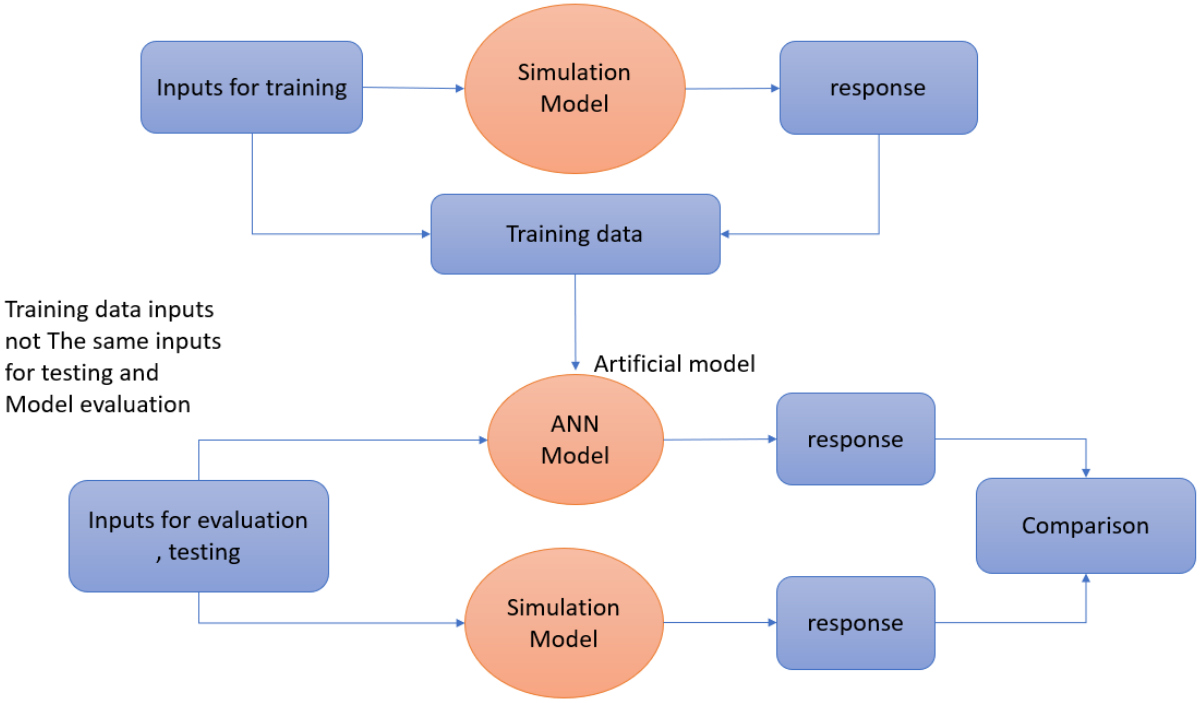


Figure 8-1 Simulator model development using ANN diagram.

Notes: The keras library is used for creating and training the neural network, so it is important to install it on your computer and import it in your code.

8.2 Modeling of the SISO system using linear activation function

The data set used for training was generated by the simulator, the input is POL chemical material, and output is the turbidity as shown in Figure 8-2. POL chemical material is selected as a single input and the turbidity is an output.

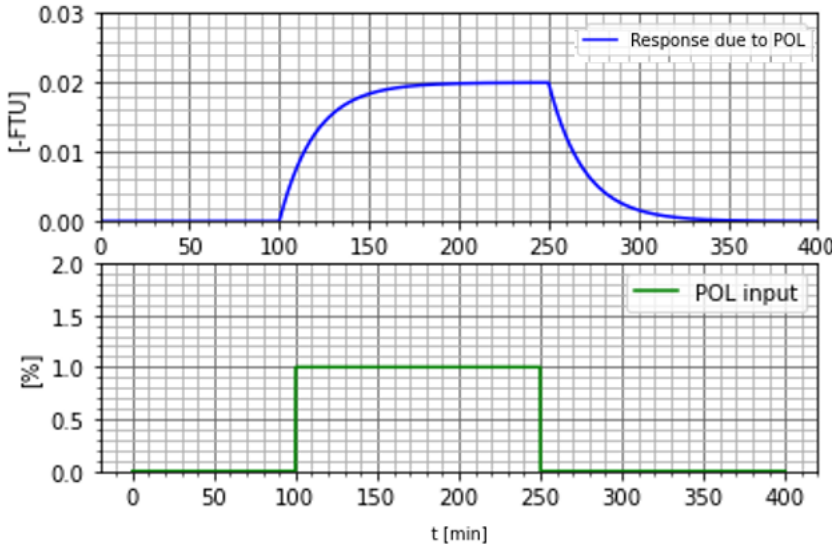


Figure 8-2 The training data set that used to train the neural network model for SISO system. The top figure is the response with time, Y-axis is the turbidity of the water as -FTU, x-axis is a time in minute. The bottom figure is the input with time, Y-axis is the POL materials as percentage, x-axis is a time in a minute.

The block diagram of the SISO system ANN is shown in Figure 8-3, the input is POL and the output is the turbidity

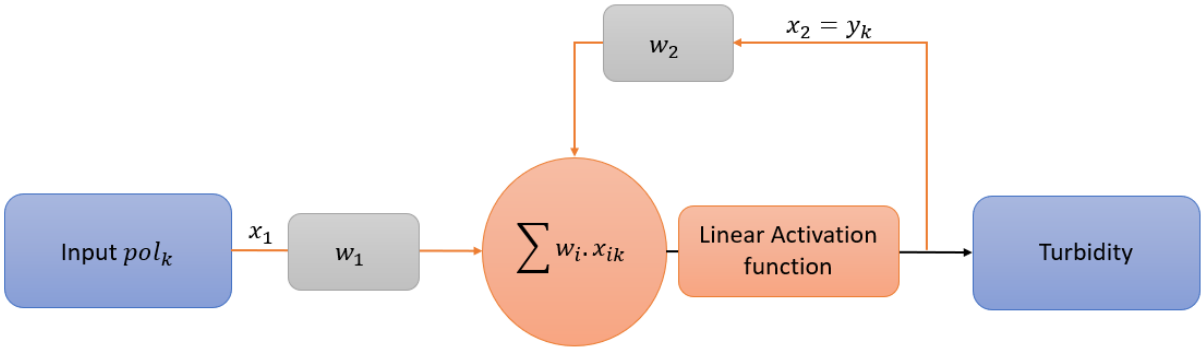


Figure 8-3 SISO system ANN block diagram, the input is POL, and the output is the turbidity.

This kind of neural network is called recurrent neural network due to feedback from the output.

8.2.1 Method:

Creating the recurrent neural network model and training the model can be perform as the following in python.

The training data is used to train the model as the following

```
from keras.models import Sequential
# create neural network model
model = Sequential()
model.add(Dense(1, input_dim=2, activation='linear'))
model.add(Dense(1, activation='linear'))
model.compile(loss="mean_squared_error", optimizer="adam")
# load training data
train_df = pd.read_csv("train_data.csv")
X1 = train_df.drop('y', axis=1).values
Y1 = train_df[['y']].values
# train the model
model.fit(X1,Y1,batch_size=32,epochs=1000,verbose=2)
# Save the model to hard drive
model.save('model.h5')
```

Using the trained model as the following:

```
model = Sequential()
model.add(Dense(1, input_dim=2, activation='linear'))
model.add(Dense(1, activation='linear'))
model.compile(loss="mean_squared_error", optimizer="adam")
model.load_weights('model.h5')
ypol_k = 0
for k in range(0, N_sim): # N_sim=(time stop-time start)/time step
    NN_input=np.vstack((ypol_k,U_pol[k])).T # reformate the input data to suit the
                                            # ANN model
```

```
yNN_plot_array[k] = model.predict(NN_input)

ypol_k = yNN_plot_array[k]
```

8.2.2 Results and discussion:

Figure 8-4 shows the results from simulation model and neural network model

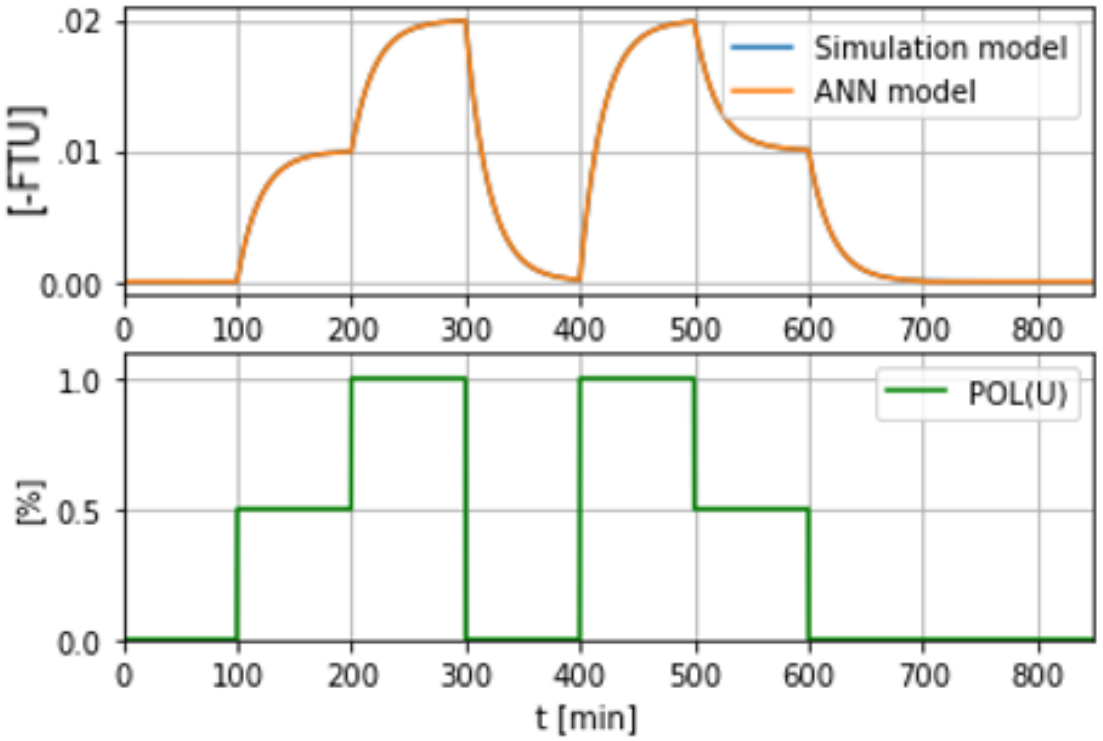


Figure 8-4 The response of the SISO system using simulator model and neural network model. The top figure is the response with time, Y-axis is the turbidity of the water as -FTU, x-axis is a time in minute. The bottom figure is the input of POL with time, Y-axis is the POL materials as percentage, x-axis is a time in minute.

The model of the ANN behaves exactly the same as the model of the simulator as shown in Figure 8-4. That means the ANN model fits well the data, and the results correspond to the mathematical studies.

8.3 MISO system modeling using linear activation function

The data set that was used for training was generated by the simulator and it contains three inputs and one output, the inputs are PIX, PAX, POL chemical materials, and the output

is the turbidity as shown in the figure. POL chemical material is selected as multi-input and the turbidity is the output as shown in Figure 8-5.

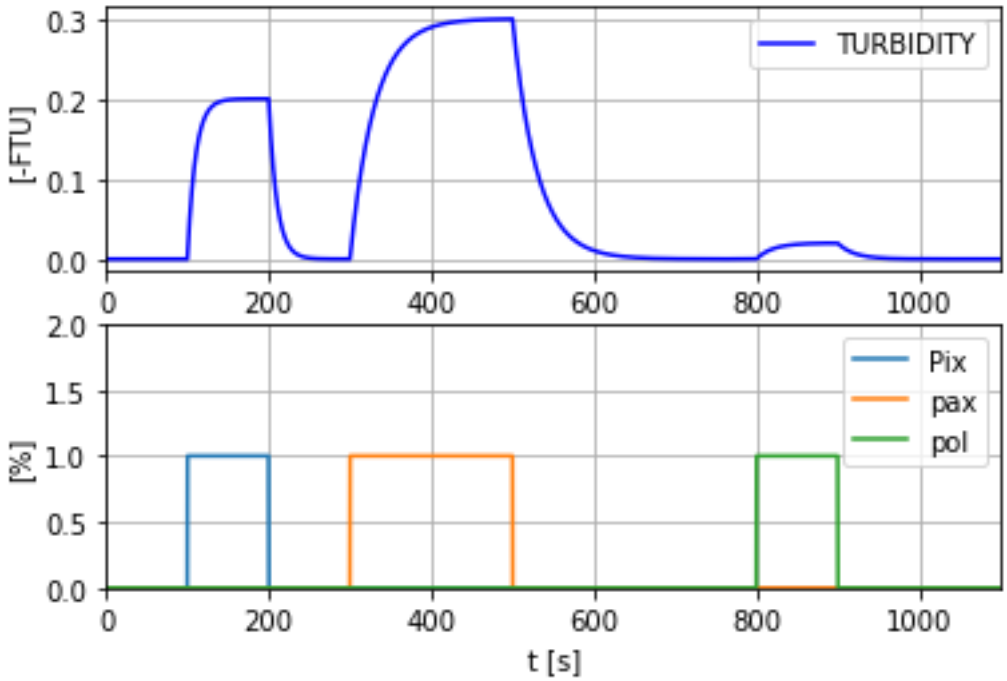


Figure 8-5 The training data set that used to train the neural network model for MISO system. The top figure is the response with time, Y-axis is the turbidity of the water as -FTU, x-axis is a time in minute. The bottom figure is the inputs with time, Y-axis is the PIX, PAX, POL materials as percentage, x-axis is a time in minute.

The training code for MISO.

```
# create neural network model
model = Sequential()
model.add(Dense(3, input_dim=4, activation='linear'))
model.add(Dense(10, activation='linear'))
model.add(Dense(1, activation='linear'))
model.compile(loss="mean_squared_error", optimizer="adam")
# train the model
model.fit(X1,Y1,batch_size=32,epochs=200,verbose=1)
```

Trained model is used in the code as the following:

```
model = Sequential()
```

```
model.add(Dense(3, input_dim=4, activation='linear'))
model.add(Dense(10, activation='linear'))
model.add(Dense(1, activation='linear'))
model.compile(loss="mean_squared_error", optimizer="adam")
model.load_weights('model.h5')
ypol_k = 0
for k in range(000,8000):
    NN_input=np.vstack((ypol_k,U_pol[k],U_pix[k],U_pax[k])).T
    yNN_plot_array[k] = model.predict(NN_input)
    ypol_k = yNN_plot_array[k]
```

8.3.1 Results and discussion

Mean Squared Error (MSE) is 4.95e-08 which refers to the difference between the model output and the target. Simulator and ANN model fed with same inputs the responses are shown in Figure 8-6

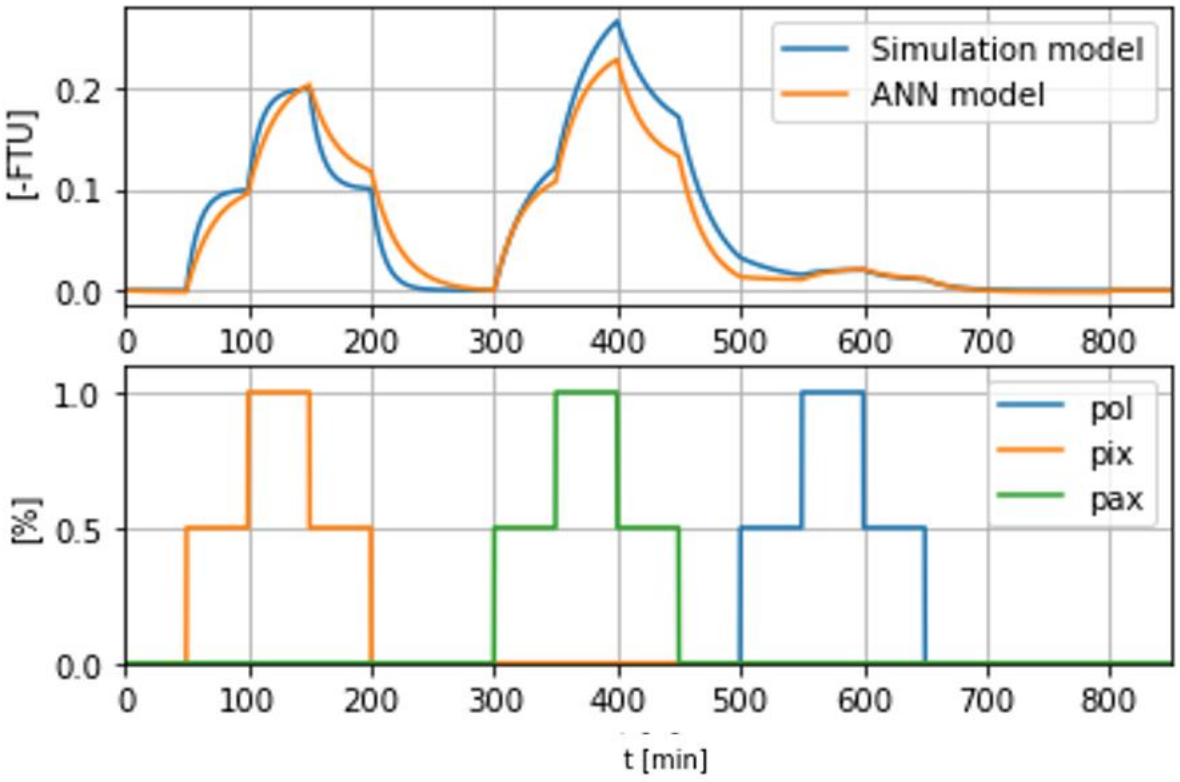


Figure 8-6 The response of the MISO system using simulator model and neural network model, ANN has 3 layers and 10 neurons in the hidden layer. The top figure is the response with time, Y-axis is the turbidity of the

water as -FTU, x-axis is a time in a minute. The bottom figure is the input of POL with time, Y-axis is the PIX, PAX, POL materials as a percentage, x-axis is a time in a minute.

The ANN model represents the Simulator model as shown in Figure 8 6. The ANN and the simulator fed with different levels of step functions so that the ANN model response with small error as illustrated in Figure 8-6. The errors occur due to underfitting.

9 NARX feedback neural network

The real dynamic process can have a time delay that will increase the complexity of the system. Time delay can create mismatch problems between a real process and a model. NARX network can overcome these problems. The neural network is based on the NARX model. The diagram of the network is shown in Figure 9-1

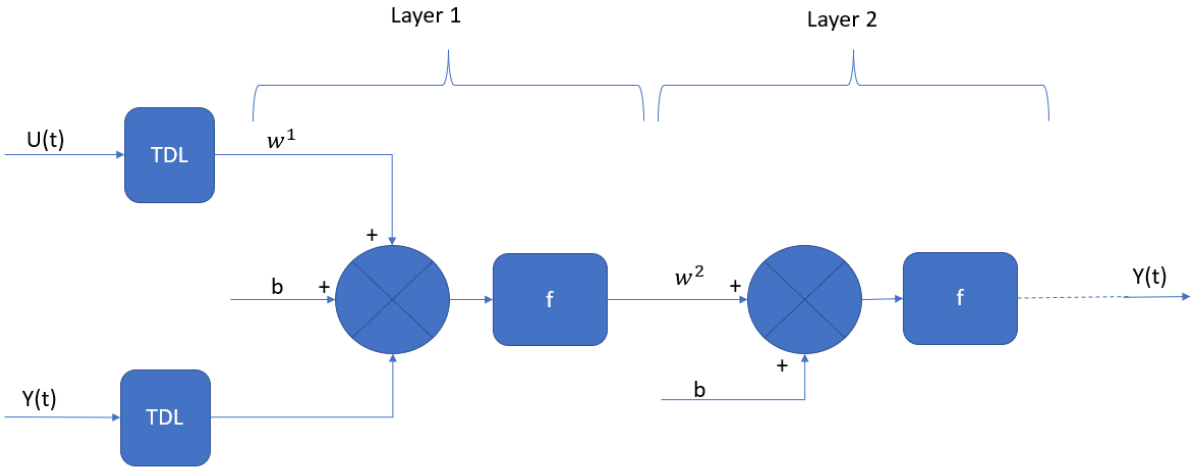


Figure 9-1NARX feedback neural network, TDL is tapped time delay, f is an activation function, w is a weight, b is a bias.

9.1 NARX Neural network model

The final model of water dosing process is developed by NARX neural network to overcome time delay problems. The mathematical NARX form that can be applied is illustrated in equation (10.3) [18]

$Y(t) = f(Y(t-1), Y(t-2), \dots, Y(t-n), U(t-1), U(t-2), \dots, U(t-n))$	(10.3)

Where n is a number of tapped time delay line, Y is the output vector $Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \end{pmatrix}$ where i is the number of outputs, U is the input vector $U = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_j \end{pmatrix}$ where j is the number of inputs

The water dosage process has a variable time delay, and it is varied due to some inputs. The inflow could have a significant influence on time delay and how many parameters influence on time delay is unknown. The time delay index should be assumed in the network. Time delay index n is the maximum number of steps in the time delay.

The form of one neuron in the NARX network will be as equation (10.4)

$$f_{net} = f\left(W_1^{input} U_t + W_2^{input} U_{t-1} + \cdots + W_n^{input} U_{t-n} + W_1^{output} Y_{t-1} + W_2^{output} Y_{t-2} + \cdots + W_n^{input} Y_{t-n} \right) + b \quad (10.4)$$

W_n^{output} is the output wight vector $W_n^{output} = \begin{pmatrix} w_1^{output} \\ w_2^{output} \\ \vdots \\ w_i^{output} \end{pmatrix}$ where i is the number of outputs, W_n^{input} is the input weight vector $W_n^{input} = \begin{pmatrix} w_1^{input} \\ w_2^{input} \\ \vdots \\ w_j^{input} \end{pmatrix}$ where j is the number of inputs. f_{net} is the neuron activation function. b is the bias.

The NARX neural network will remember n previous inputs and outputs to predict the next output. The network will find out the time delay from the training data. It will select the correct inputs by tuning the weights. In case, the weight is negative that means it will cancel the respective input which has no impact on the output. The architecture of this NARX neural network is called feedforward parallel architecture, where the output of the network is fed as input. The block diagram that illustrates the network is shown in Figure 9-2

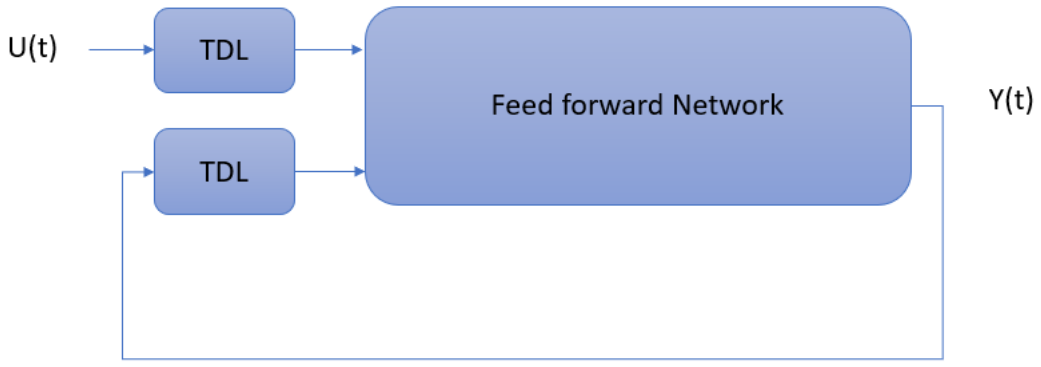


Figure 9-2 Parallel architecture NARX neural network with tapped time delay line.

The equation (10-3) has no error term which create some error in the output of the model. The predicted output goes as input for the next prediction that will propagate the error with time. In order to reduce this error, the output of the real process should be connected to the NARX model as input instead of prediction output.

This NARX neural network model architecture is called series parallel architecture. The diagram of it is shown in the Figure 9-3.

Figure 9-3 Parallel series architecture NARX neural network with tapped time delay line. Y(t) is predicted output. Ym(t) is measured output from process.

9.2 NARX neural network simulation

The data set that was used for training the NARX network was generated by the simulator, and it contains three inputs and one output. The inputs are PIX, PAX, POL chemical materials, and the output is the turbidity as shown in Figure 9. The time delay line is [1 min] for PIX, [3 min] for PAX, and [6 min] for POL.

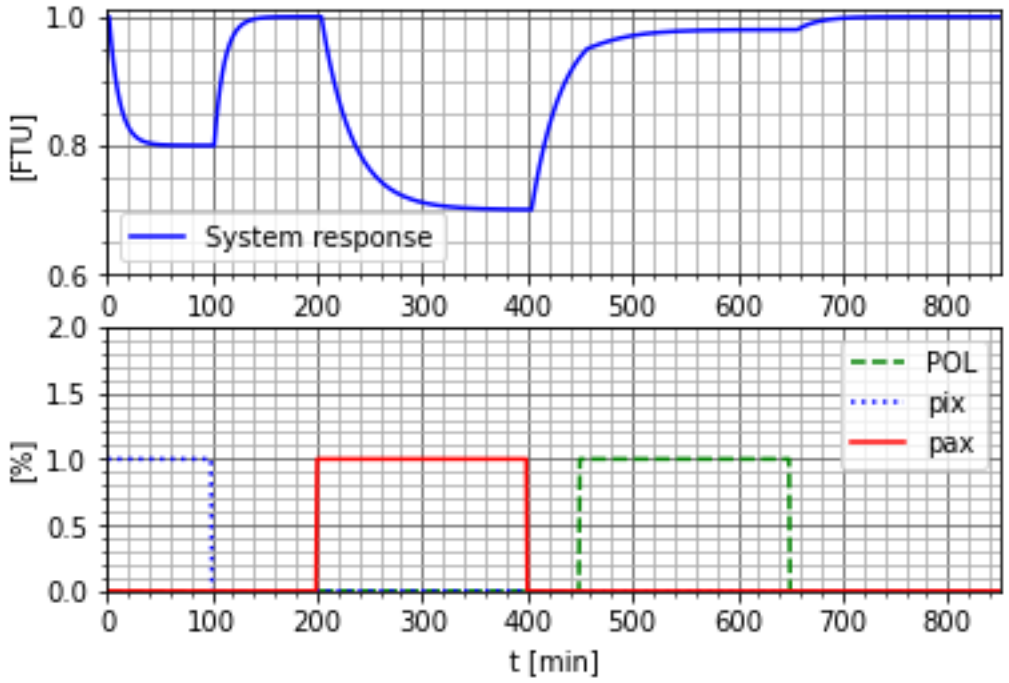


Figure 9 NARX neural network training data set from simulator model. The top figure is the response with time, Y-axis is the turbidity of the water as -FTU, x-axis is a time in a minute. The bottom figure is the inputs with time, Y-axis is the PIX, PAX, POL materials as a percentage, x-axis is a time in a minute. The time delay line is [1 min] for PIX, [3 min] for PAX, and [6 min] for POL

The NARX model is trained by MATLAB with the data that shown in Figure 9. The NARX has 10 hidden layers and 7-time delay line. The optimizer is Levenberg–Marquardt. The NARX model is trained by 850 sample from the real process. 70 % of data set for training, 15% for validation, and 15% for testing.

In order to see how the trained model behaves, the NARX model and simulator fed with different inputs which has not been used in training. The trained model is built as a black box and connected to the inputs from Simulator's data. The NARX model is used as parallel architecture and parallel series architecture as shown in Figure 9-4. The process here is the simulator.

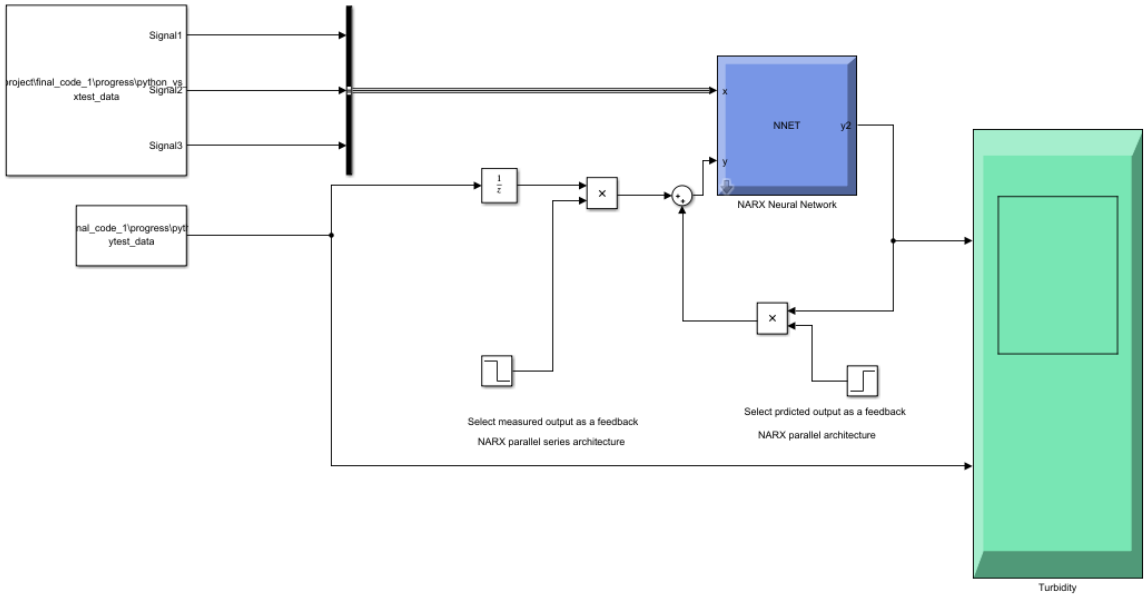


Figure 9-4 NARX ANN model in parallel with process. Both architectures are included parallel and parallel series (MATLAB Simulink).

9.3 MATLAB Simulation Results

The performance of the training is shown in the best validation is 7.6×10^{-5} at epoch 126. Mean Square Error (MSE) is the average squared difference between model output and target data. The response of the model, and target data Figure 9-6

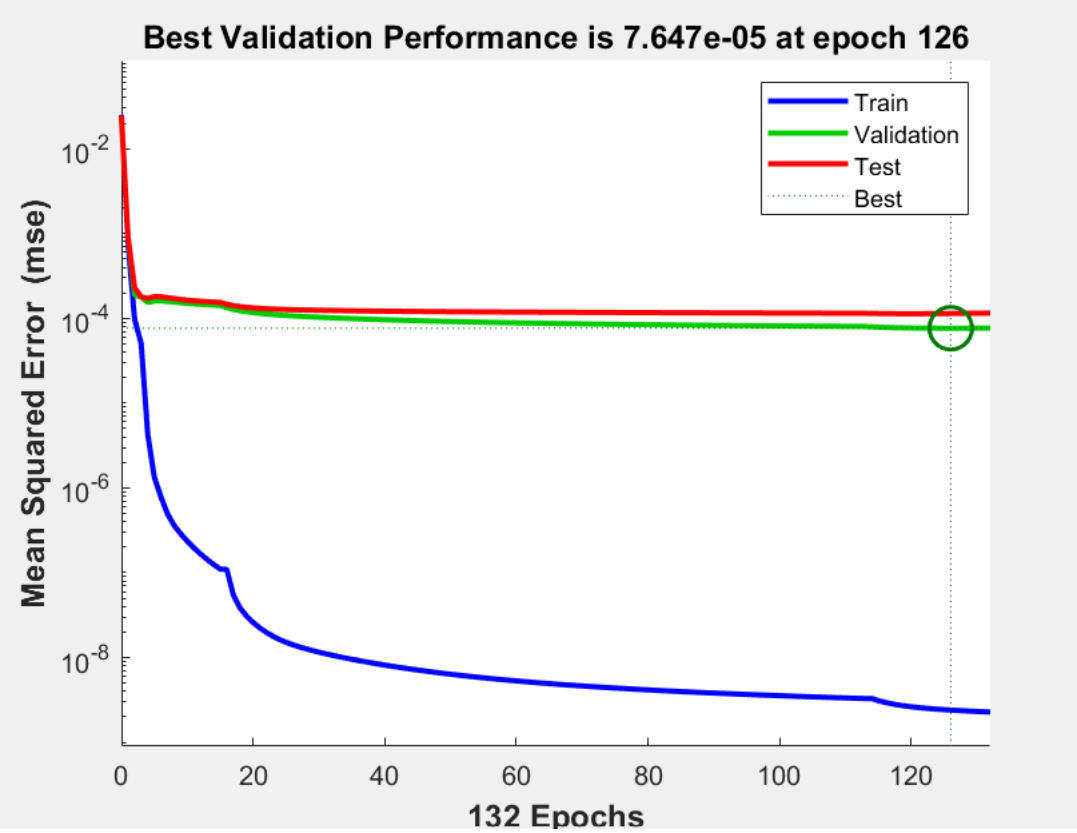


Figure 9-5 Validation performance for NARX ANN simulation, MSE of the training data with Epoches.

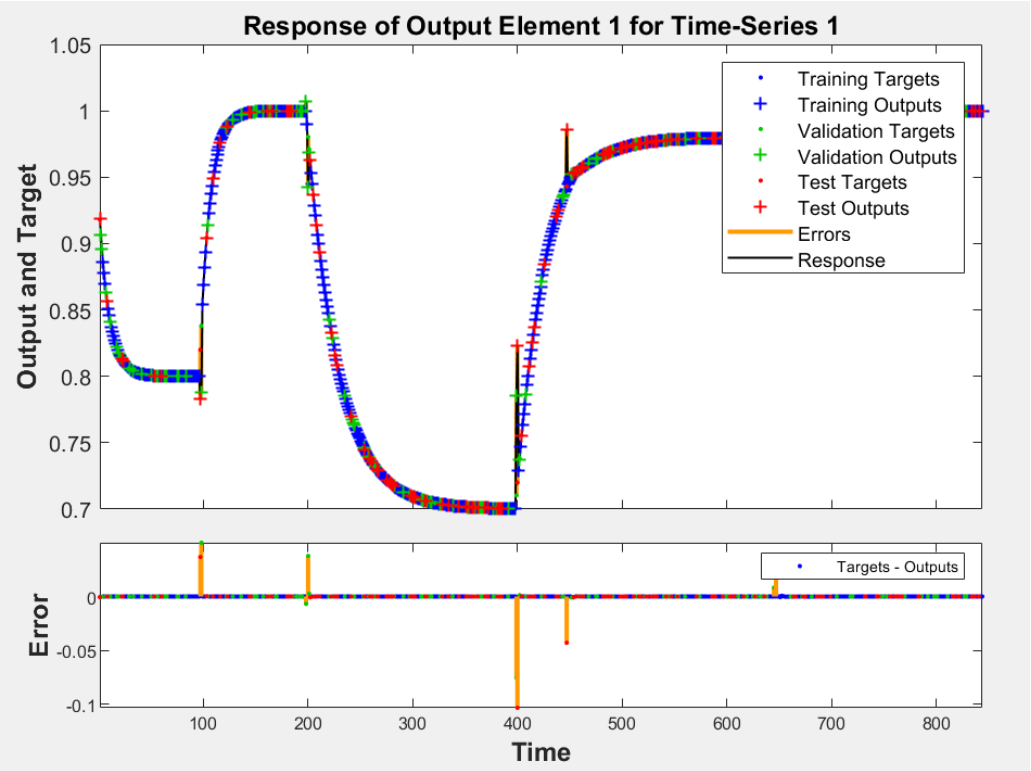


Figure 9-6 The response of the trained model and target data for NARX ANN simulation.

The MSE and Correlation for the training and validation and testing are included in Table 9-1

Table 9-1 MSE and Correlation for NARX ANN model simulation.

	Target values	MSE	R
Training	595	2.4×10^{-9}	0.99
Validation	128	7.6×10^{-5}	0.997
Testing	128	1.14×10^{-4}	0.957

The NARX ANN model feeds with new data generated from the simulator, the inputs data and correspond output from the simulator and NARX model are shown in Figure 9-7.

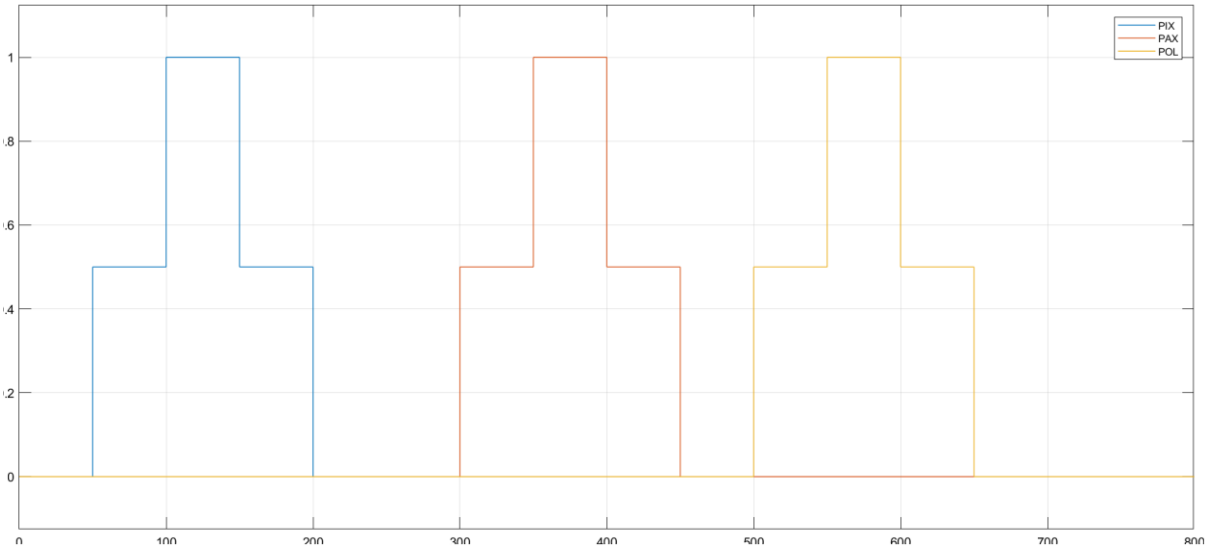


Figure 9-7 NARX ANN model and simulator inputs, y-axis is the percentage of chemical material pol, pix, pax, x-axis is the time in a minute.

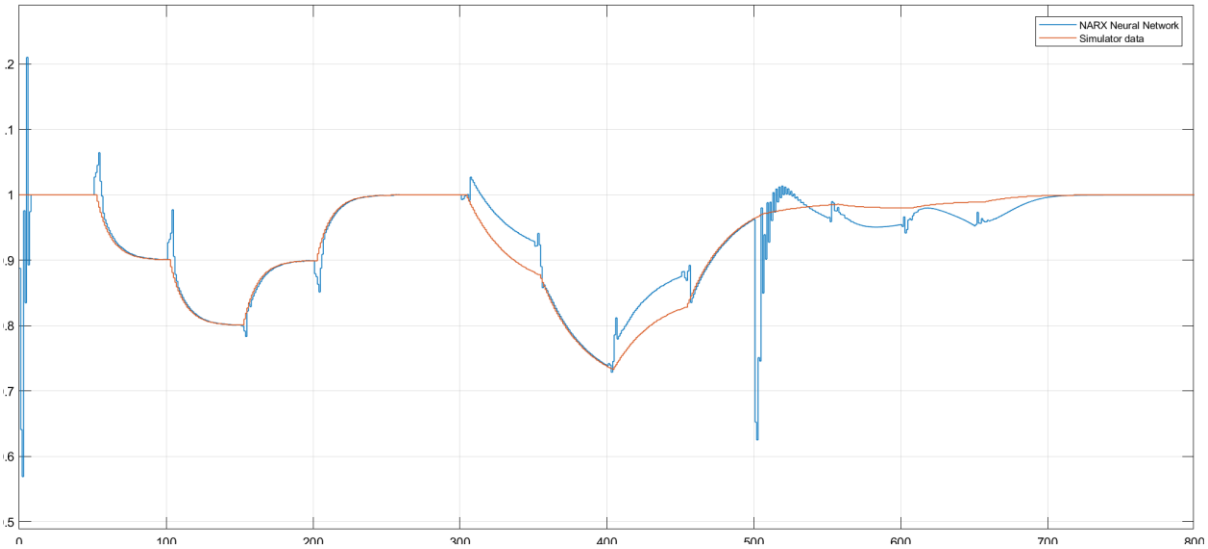


Figure 9-8 NARX model and simulator output, y-axis is turbidity [FTU], x-axis is the time in a minute, from 0 to 500 minutes the architecture is parallel series, and after 500 the architecture is changed to parallel.

9.4 Discussion

The purpose of using simulator to train and test the NARX model is to make sure that the NARX is working properly with time delay. The time delay and response of the simulator is known and can be manipulated easily. In this experiment seven tapped time delay line is used because we already know that maximum delay is 6 min.

The real process's model is unknown, therefore, the NARX simulation is important to check the model with known model. As shown in Figure 9-8 the NARX model is fit the simulator and it is working properly for both architectures.

10 Real process model development

A supervised method in Artificial neural networks and machine learning is depending on the training data set. The quality of the model depends on the data set. If the data set does not have enough information will provide a poor performance of the trained model. It is important to select the training data carefully and avoid overfitting and underfitting. Underfitting means the model is not generalized. Overfitting means the ANN model trains noise data which will impact the performance of the model [21] [22] [23]. As shown in Figure 10-1.

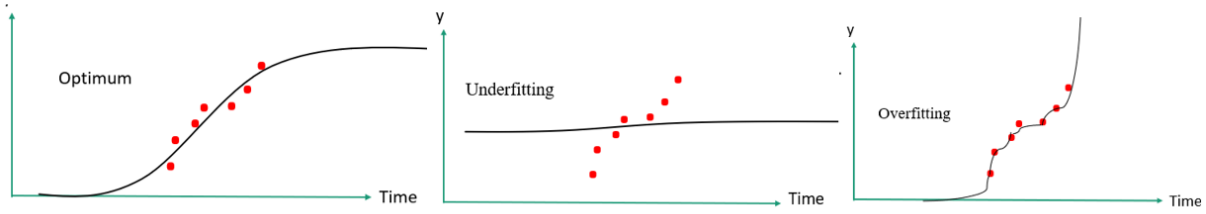


Figure 10-1 . Overfitting, underfitting, and optimum fitting of ANN model

10.1 Experimental design

Chemical dosing process is a multi-variate process, which means there are many inputs influence the output.

The purpose of the experiment.

The purpose of experimental design are [10]:

- To know how many experiments are needed to get the information.
- To develop the model of the system.
- To know individual effect of the input.
- To find out which inputs has significant impact to the output.
- The structure of the experiments.

The experiments can perform by applying specific inputs and measure the outputs. The experiment data should have both inputs and outputs as shown in Figure 10-2.

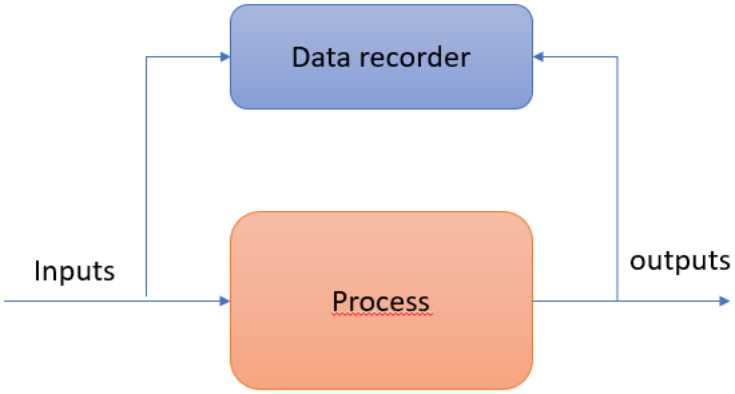


Figure 10-2 Experiment block diagram.

The suggestion of full factorial design of the experiments are indicated in Table 10-1

Table 10-1 The full factorial design experiment suggestion. Zero means no change and one mean add change. the time delay should be taken in account.

Inflow	PIX	PAX	POL	Response
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14

1	1	1	0	15
1	1	1	1	16

If there is no reaction between the inputs chemical materials PIX, PAX, POL. The full factorial design experiments table can be reduced to the Table 10-2.

Table 10-2 The reduced form of full factorial design experiment suggestion. Zero means no change and one mean add change. the time delay should be taken in account.

Inflow	PIX
0	0
0	1
1	0
1	1
Inflow	PAX
0	0
0	1
1	0
1	1
Inflow	POL
0	0
0	1
1	0
1	1

10.2Scaling (Normalization)

Scaling is a method that used to normalize the data. The purpose of that to make changing range between 0 and 1. Min-max normalization is used in this thesis. The normalization equation is (10-1) [23].

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (10\text{-}1)$$

10.3 Filtering

The noise impacts the model error. The noise can be reduced by filtering the data. The high frequency terms should be removed from the data. That can be done by passing the data into a low pass filter.

Low pass filter equation is (10-2)

$$y_{k+1} = y_k + \frac{dt}{T_c}(K * x_k - y_k) \quad (10\text{-}2)$$

dt time step. T_c time constant, x filter input, y filter output, K is the gain.

10.4 Method

MATLAB software and Python script are used to create the NARX neural network, prepare the data set, train, and test the network and model the process. MATLAB M-file is used for preparing the data, MATLAB nnstart for creating, and training NARX neural network, MATLAB Simulink for modeling. The data set was acquired from Veas.

Training, testing data is prepared by MATLAB:

Reading the data from CSV file

```
Data=readmatrix('data.csv');
```

Normalize the data:

```
function y=Norm(x)

    for k =1:length(x)
        y(k)=(x(k)-min(x))/(max(x)-min(x));

    end
end
```

Filter the data:

```
function y_f = Filter(x)
y_k=x(1);
y_f=[];
Kc=1;
Tc=40;% time constant
```

```
ts=10; % time step
    for k =2:length(x)
        dy_dt=(Kc*x(k)-y_k)/Tc;
        y_k=y_k+dy_dt*ts;
        y_f=[y_f;y_k];
    end
end
```

Save the data:

```
csvwrite('Dataset.csv',DataSet);
```

10.4.1Create, Train, Test, Evaluate the neural network.

The data is normalized, so the maximum value is 1 and minimum value is 0. To avoid large number in the neural network, the ReLU activation function can be good choice.

The network correlation and mean squared error are important to evaluate the network. If the correlation is low that the performance of the network is poor, and the neural network output has poor performance. The evaluation can be done according to correlation and mean squared error.

Training the NARX network by MATLAB can be performed by two ways: from nnstart tools or M-file.

For M-file

```
Input_tapped_delay=2;
Output_tapped_delay=2;
Hidden_layers=10;
[X,T] = dataset;
net = narxnet(1: Input_tapped_delay,1: Output_tapped_delay, Hidden_layers);
[x,xi,ai,t] = preparets(net,X,{},T);
net = train(net,x,t,xi,ai);
y = net(x,xi,ai);
view(net)
```

Training the NARX network by Python can be performed as the following:

```
import pandas as pd

from keras.models import Sequential
```

```
from keras.layers import *
# load training data
train_df = pd.read_csv("train.csv")
X = train_df.drop('y', axis=1).values
Y = train_df[['y']].values
# create neural network model
model = Sequential()
model.add(Dense(3, input_dim=6, activation='linear'))
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss="mean_squared_error", optimizer="adam")
model.fit(X,Y,batch_size=32,epochs=200,verbose=1)
```

10.4.2Trained NARX model in the real process

Figure 9-4 is used for real process as well as in simulation.

Two step functions are used to select the NARX architecture. The data set comes from real process.

Using the model in Python:

```
train_df = pd.read_csv("test.csv")
X3 = train_df.drop('y', axis=1).values
Y3 = train_df[['y']].values
t_plot_array = np.linspace(0,len(Y3),len(Y3))
yNN_plot_array = np.zeros(len(Y3))
y_k=X3[0,5]
for k in range(1,2000):
    if k<1000:
        y_k=Y3[k-1]  # parallel series architecture
    NN_input=np.vstack((X3[k,0],X3[k,1],X3[k,2],X3[k,3],X3[k,4],y_k)).T
```

```
yNN_plot_array[k] = model.predict(NN_input)

y_k = yNN_plot_array[k] # parallel architecture
```

10.5 Results

MATLAB and Python data set which used to train and test the NARX ANN model are same. Training Set for MISO is shown in Figure 10-3 and Figure 10-4. Training set for MIMO system is shown Figure 10-5, Figure 10-6 and Figure 10-7.

The level of PH data in Figure 10-3, ALKALINITY(before Gritchamber) data in Figure 10-3 and ALKALINITY(after Gritchamber) target data in Figure 10-7 have no information, it will have no impact on ANN trained model. ANN model cannot be trained by this kind of data. it leads to underfitting.

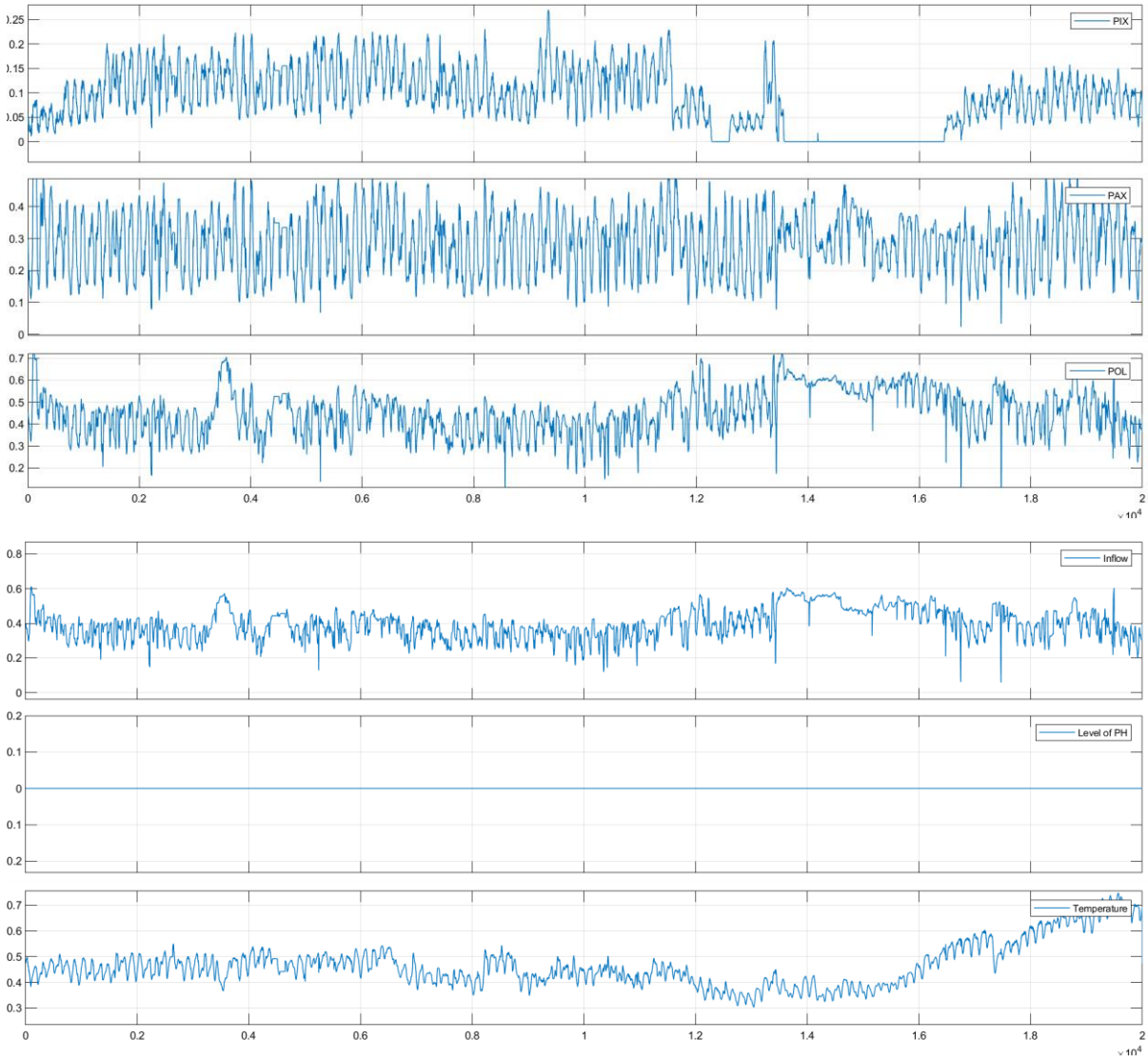


Figure 10-3 Training inputs data set for MISO system, the inputs are PIX, PAX, POL, Inflow, Level of PH, Temperature. The values are normalized where y-axis is the percentage and x-axis are the step number, the time between steps is 10 minutes.

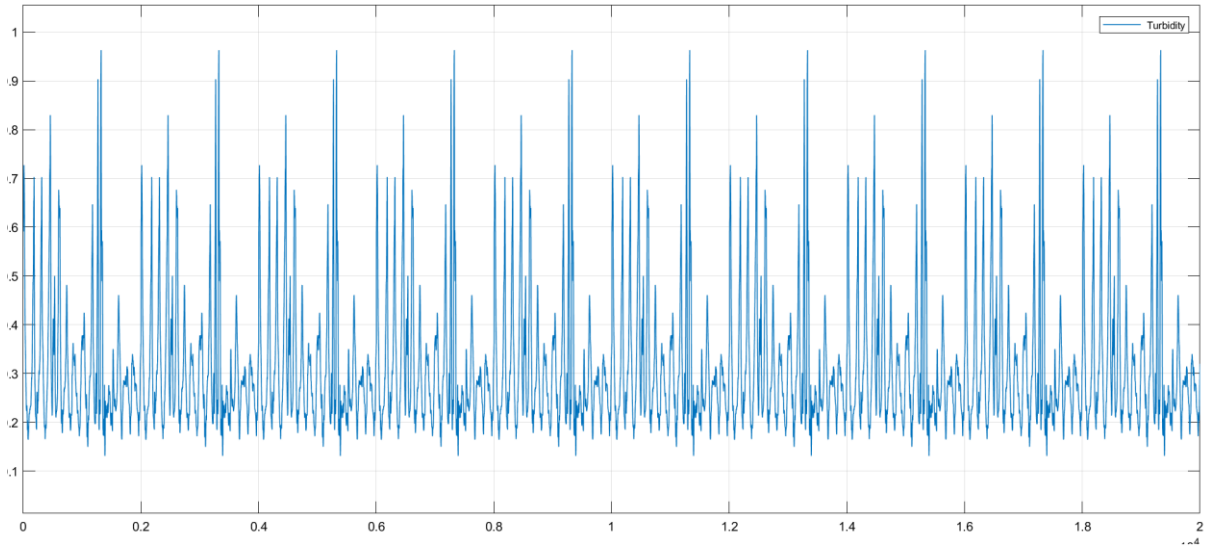


Figure 10-4 Target data set for MISO system for training. Target is turbidity. The values are normalized where y-axis is the percentage and x-axis are the step number, the time between steps is 10 minutes.

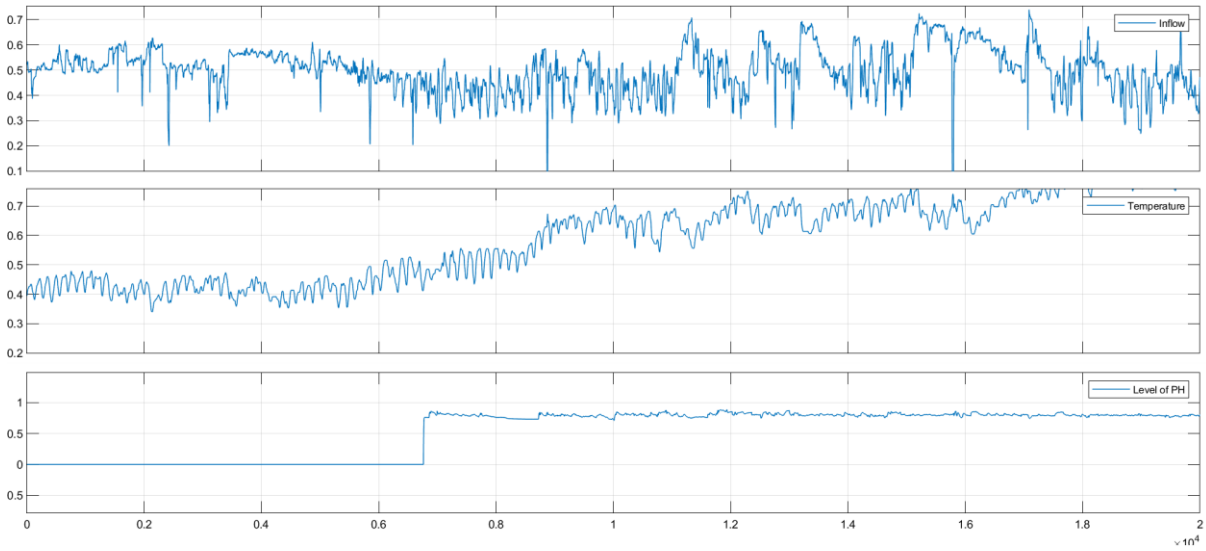


Figure 10-5 Training dataset for MIMO system, the inputs are Inflow, Level of PH, Temperature. The values are normalized where y-axis is the percentage and x-axis are the step number, the time between steps is 10 minutes.

Figure 10-6 Training data set, input data for MIMO system, the inputs are PIX,PAX, POL, SS(before Gritchamber), PHOSPHATE(before Gritchamber), ALKALINITY(before Gritchamber) The values are normalized where y-axis is the percentage and x-axis are the step number, the time between steps is 10 minutes.

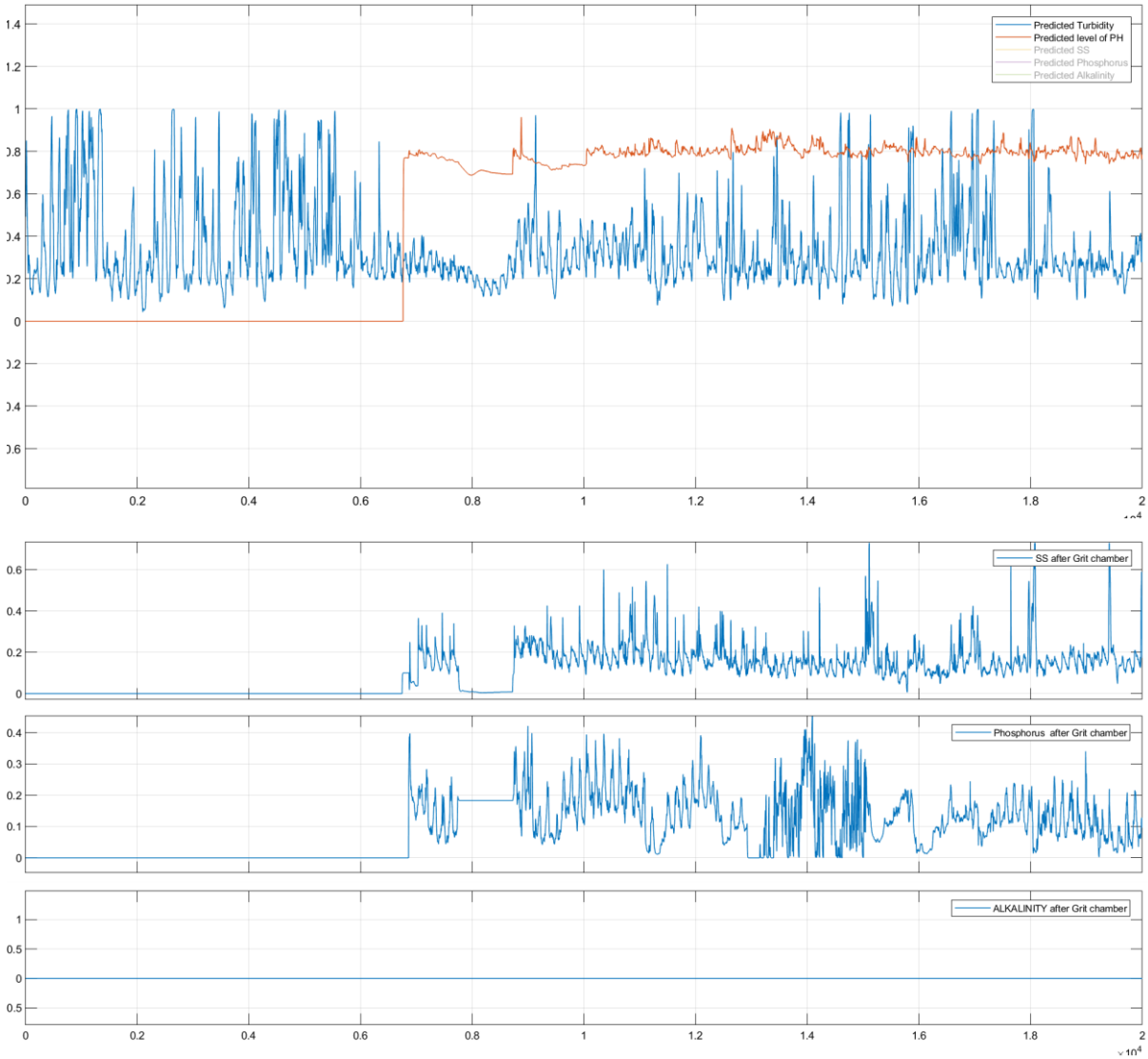


Figure 10-7 Target set data for MIMO system training, the target is turbidity, SS(after Gritchamber), PHOSPHATE(after Gritchamber), ALKALINITY(after Gritchamber)

10.5.1MATLAB Results

The NARX ANN model is trained by 20000 sample from the real process. 70 % of data set for training, 15% for validation, and 15% for testing. The optimizer is Levenberg–Marquardt optimizer. The neural network output and process output is almost similar. As shown in Figure 10-8. Ten hidden layers and one tapped time delay are used to form the network.

The response plot in Figure 10-8 shows the target data, ANN model response due to training, validation, and testing data. Error plot is the difference between target data and the model output. The histogram error plot in Figure 10-9 shows that the error near to zero at point 0.0047 most results located on this line. Figure 10-10 shows the best validation and mean squared error performance which exist in 68 epochs. The correlation plot between trained model and training, validation, testing data are shown in Figure 10-11. The correlation equal

to 0.9979 for training, 0.9979 for validation, 0.9979 for test. The correlation means how the model is fitting the data, as shown in Figure 10-11 which is more than 99%.

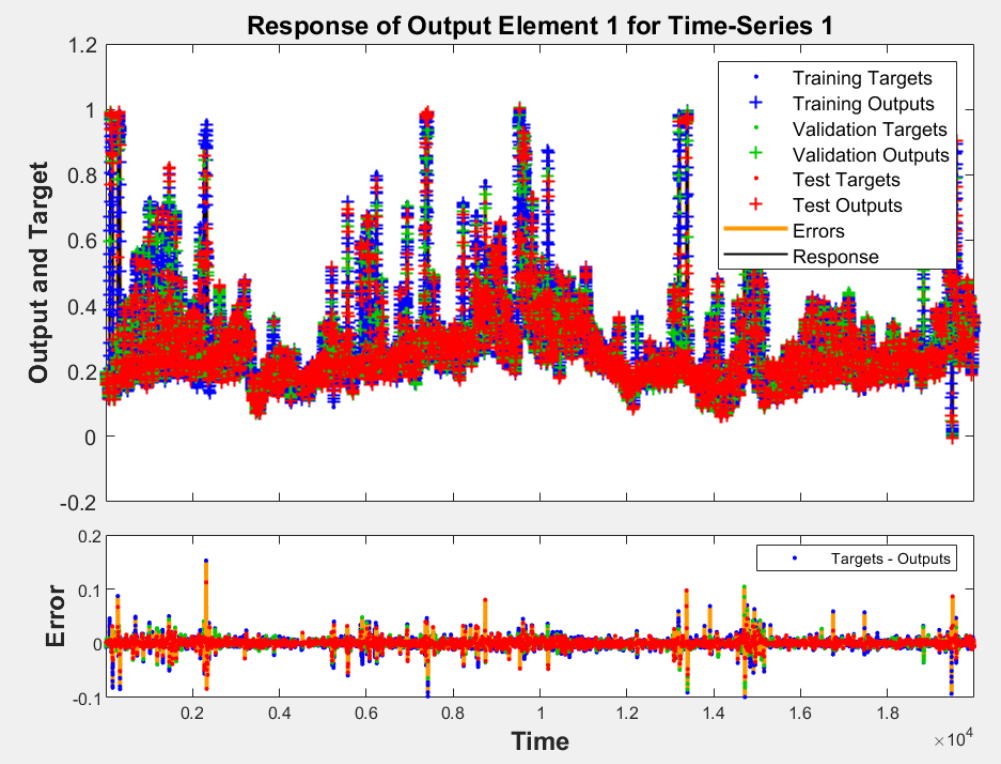


Figure 10-8 The response of the trained NARX ANN model and target data.

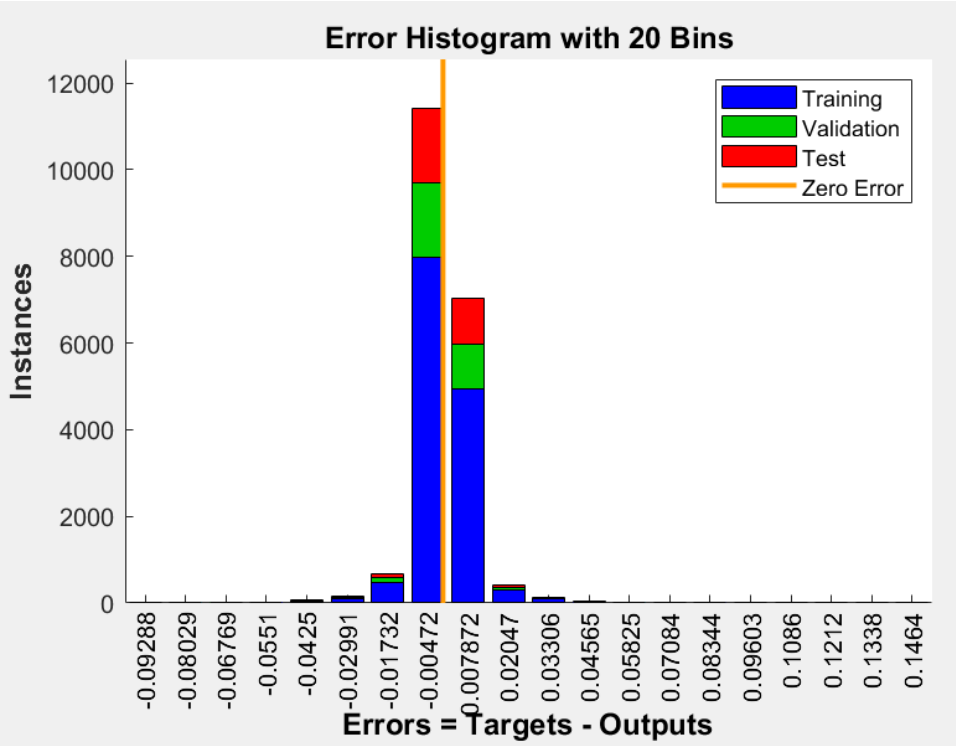


Figure 10-9 Histogram error plot for training NARX ANN for MISO system.

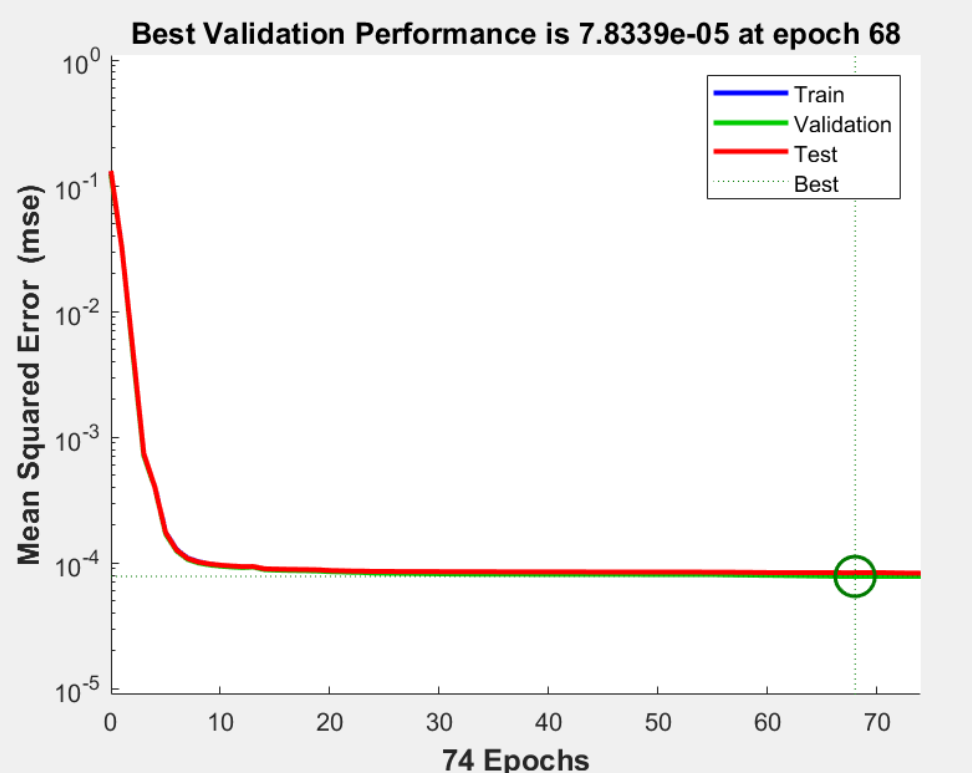


Figure 10-10 Validation performance for MISO system, MSE of the training data with Epoches.

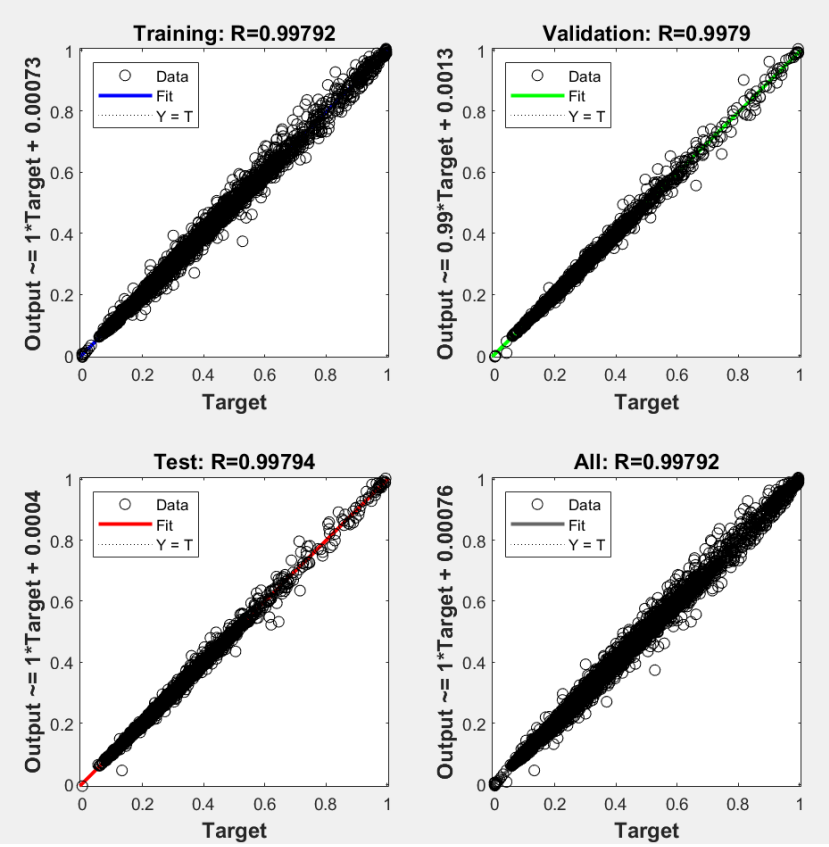


Figure 10-11 Regression correlation for MISO system.

After training process of NARX model, the model evaluation values indicated in Table 10-3

Table 10-3 MSE and Correlation for NARX ANN model for MISO system.

	Target values	MSE	R
Training	14000	8.13×10^{-5}	0.9979
Validation	3000	7.8×10^{-5}	0.99789
Testing	3000	8.3×10^{-5}	0.979

2000 sequence samples are used to see the behavior of the trained model. These 2000 samples are different from training samples. The ANN model works in parallel with the process as showing in Figure 9-4. The inputs are shown in Figure 10-12

The NARX ANN model output and target is shown Figure 10-13 and Figure 10-14. the NARX model used both architecture parallel series and parallel to compare the results between two architectures. In NARX parallel architecture, in Figure 10-14, the initial values are incorrect, therefore, the prediction is incorrect.

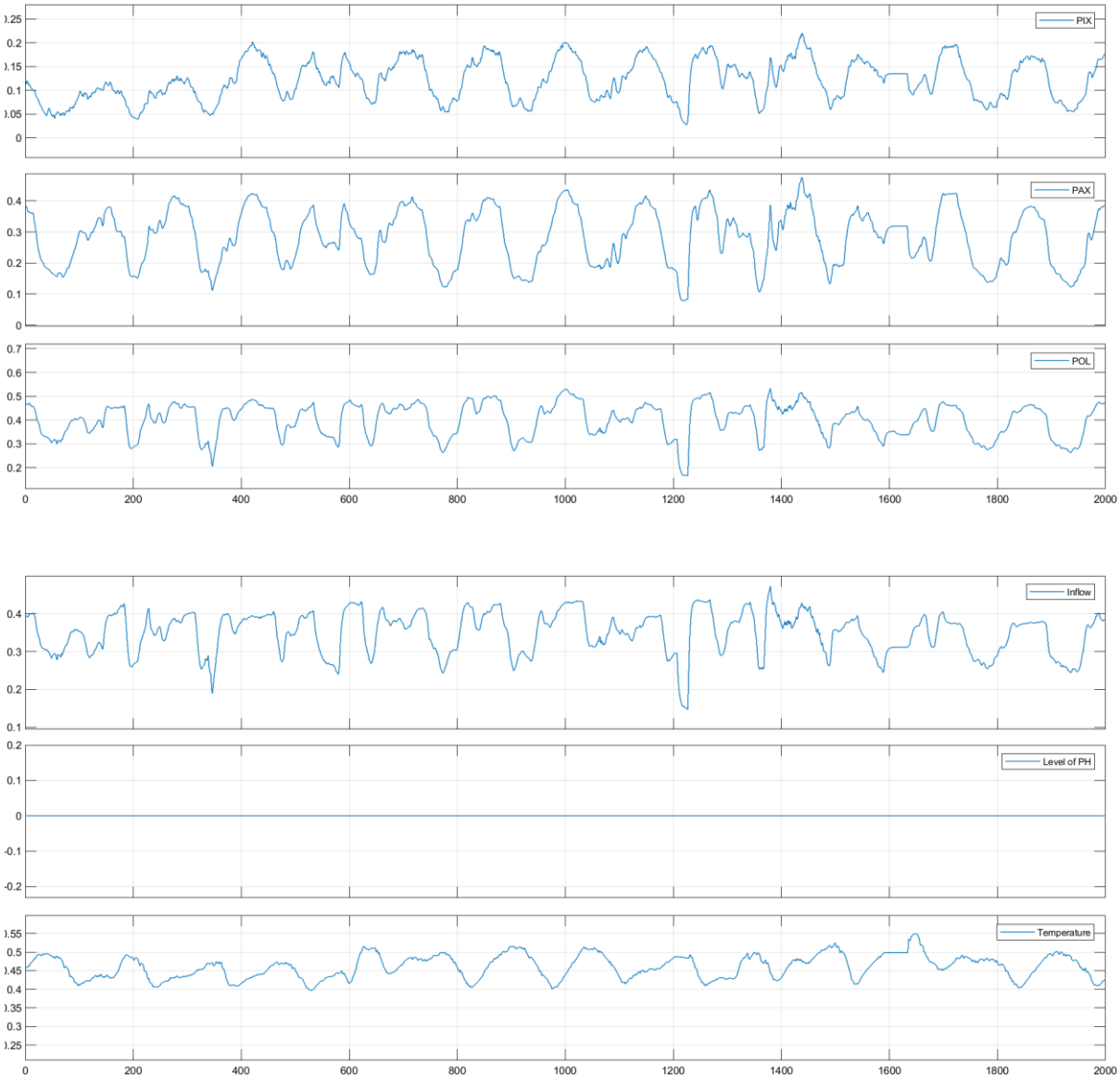


Figure 10-12 NARX ANN model inputs for MISO system. The inputs are PIX, PAX, POL, Inflow, Level of PH, Temperature. The values are normalized where y-axis is the percentage and x-axis are the step number, the time between steps is 10 minutes.

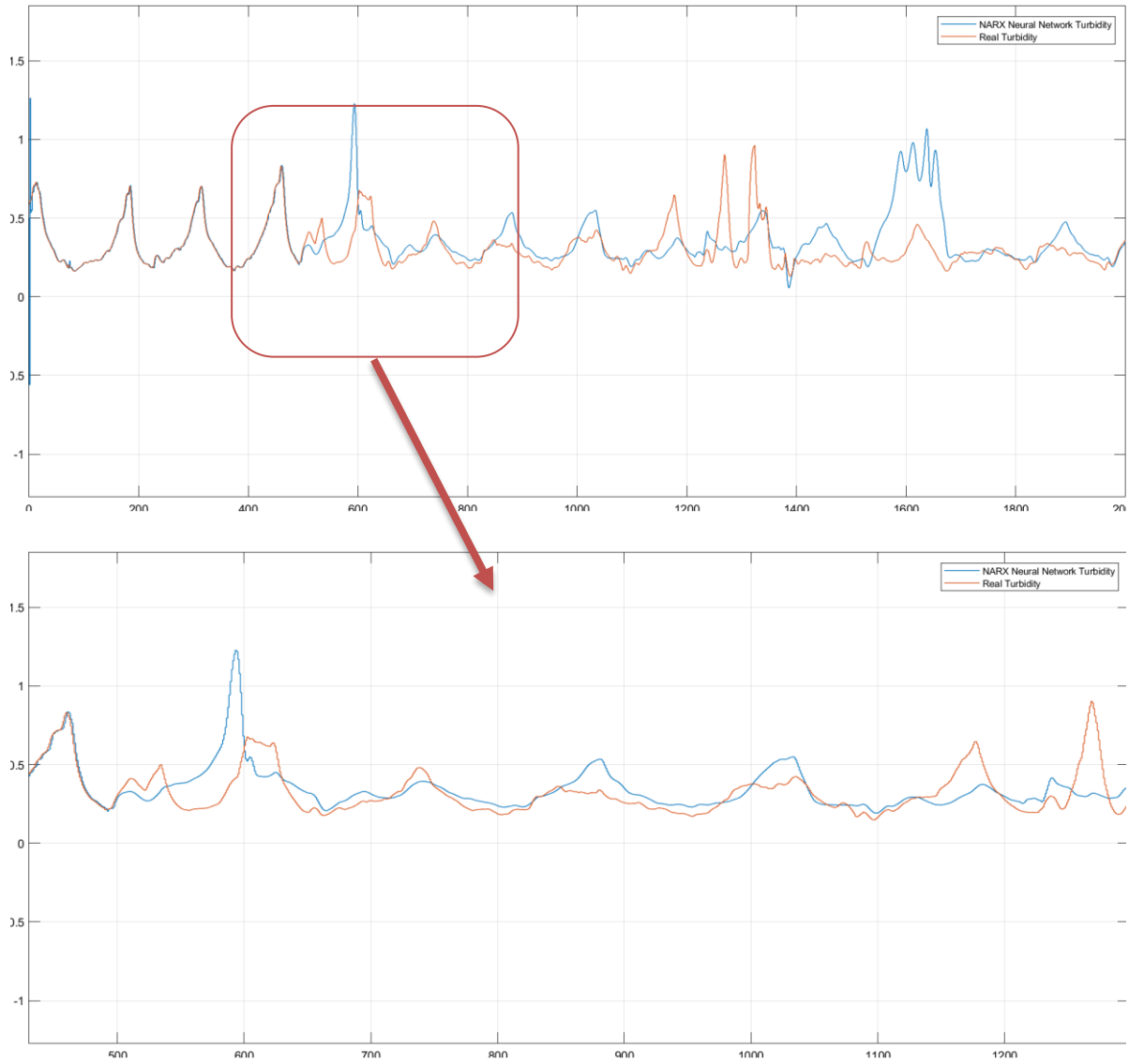


Figure 10-13 NARX ANN model response for MISO. Parallel series NARX architecture from 0 to 500, parallel NARX ANN architecture from 500 to the end, the inputs are PIX, PAX, POL, Inflow, PH, Temp. x-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a turbidity FTU. All values are normalized.

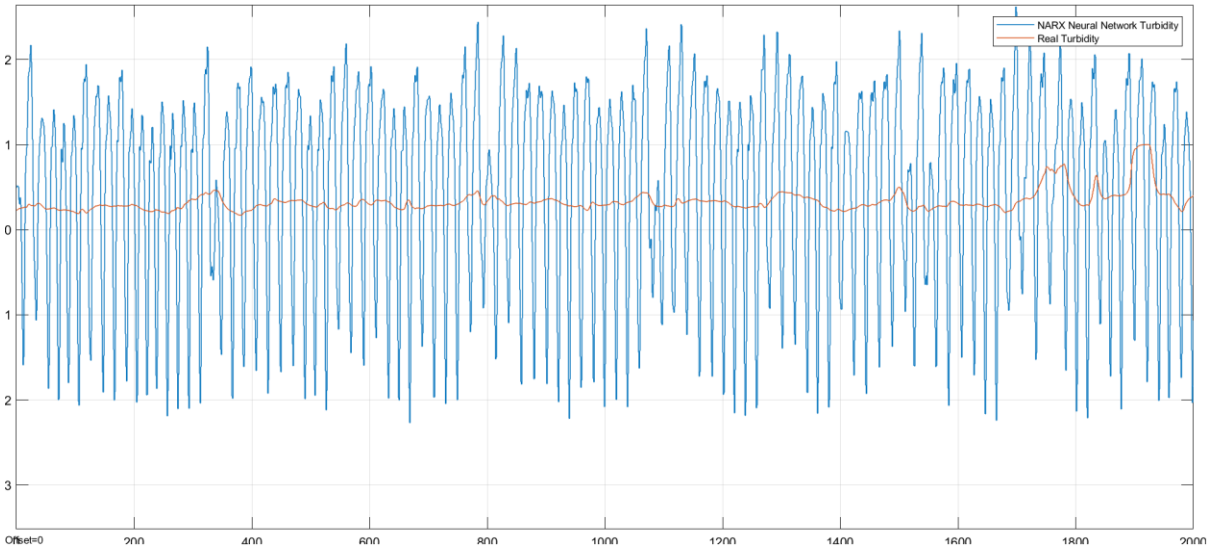


Figure 10-14 NARX ANN model response for MISO with incorrect initial inputs values. Parallel NARX architecture, the inputs are PIX, PAX, POL, Inflow, PH, Temp. x-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a turbidity FTU. The data set is normalized.

10.5.2 Python results

The model is trained by 20000 sample, the same samples that used with MATLAB from the real process. 15000 samples of data set for training, 5000 for testing. The optimizer is Adam optimizer, The mean squared error is 0.000107 at epoch 200. The network has four layers. Input layer, first hidden layer has 3 neuron and linear activation function, second hidden layer has 50 neurons and ReLU activation function, the out layer has one neuron and Sigmoid activation function. the network has seven inputs and one output. the inputs are PIX, PAX, POL, Inflow, PH, Temp, and feedback from last output.

The NARX ANN model output and target is shown in Figure 10-15, Figure 10-16, and Figure 10-17. the NARX model used both architecture parallel series and parallel to compare the results between two architectures.

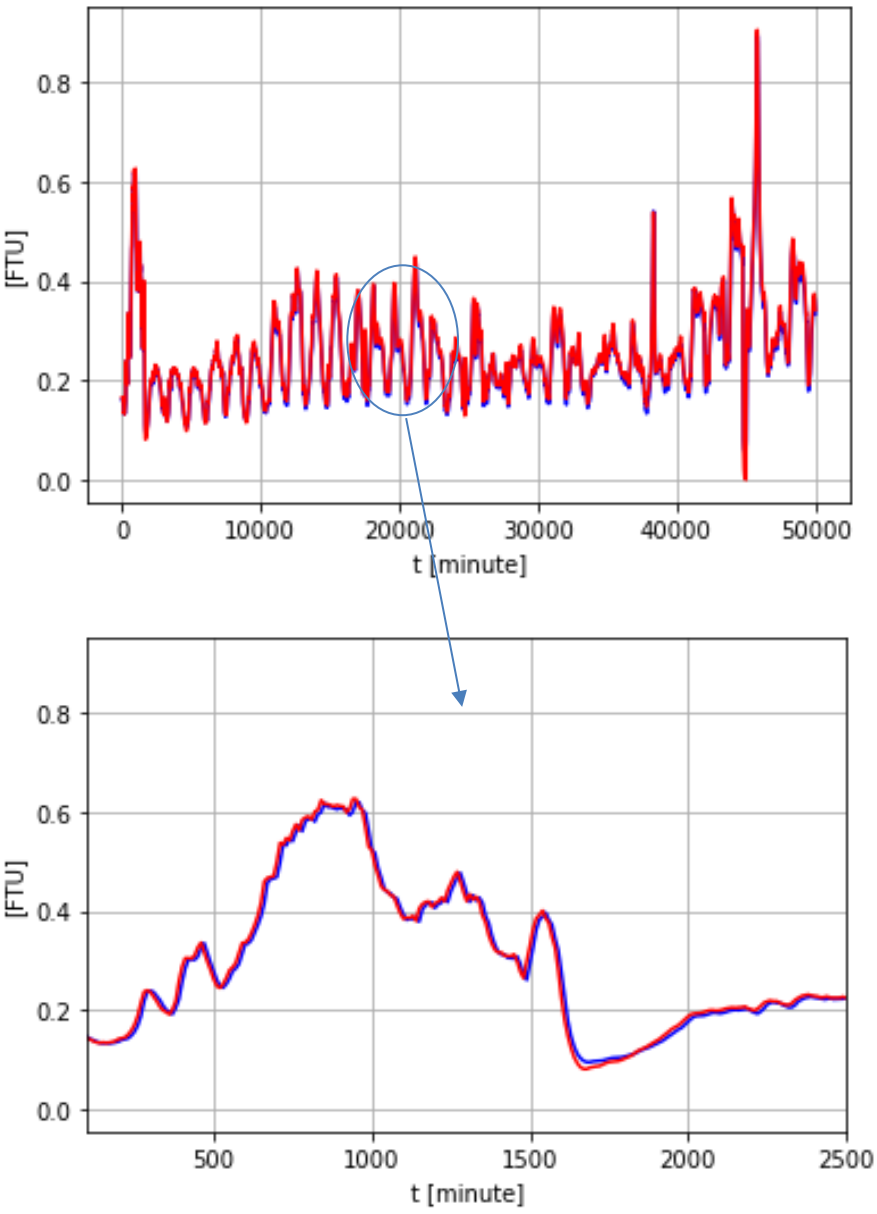


Figure 10-15 Python testing set. Red color is the target, blue color is the model output. 5000 data point is used for testing. Y-axis is a turbidity FTU. X-axis is a time in a minute.

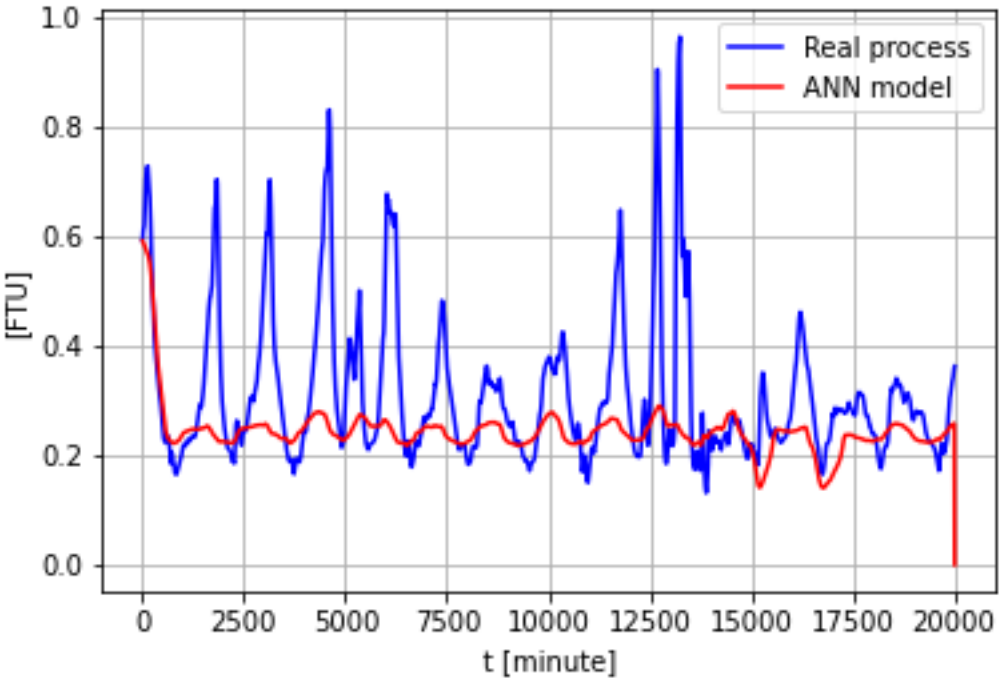


Figure 10-16 NARX ANN model response for MISO in Python. Parallel NARX ANN architecture, the inputs are PIX, PAX, POL, Inflow, PH, Temp. x-axis is a time in a minute. Y-axis is a turbidity FTU. All values are normalized.

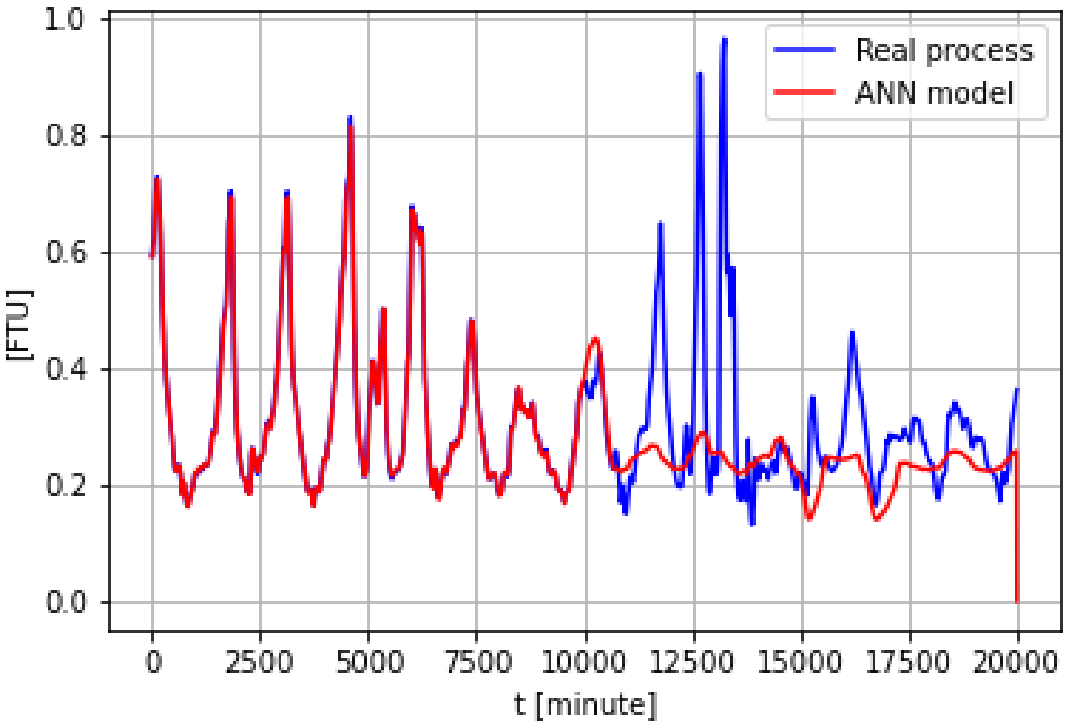


Figure 10-17 NARX ANN model response for MISO in python. Parallel series NARX architecture from 0 to 10000 minutes, parallel NARX ANN architecture from 10000 minute to the end, the inputs are PIX, PAX, POL, Inflow, PH, Temp. x-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a turbidity FTU. All values are normalized.

10.5.3MIMO system results

20000 data points are used for training validation, and testing the NARX ANN model, 70 % of data set for training, 15% for validation, and 15% for testing. The optimizer is Levenberg–Marquardt optimizer. The network is formed from 50 hidden layers and one tapped time delay. The inputs are PIX, PAX, POL, Inflow, Temperature, Level of PH, SS before Gritchamber, PHOSPHATE (before Gritchamber), ALKALINITY (before Gritchamber) shown in Figure 10-5, Figure 10-6. The target are TURBIDITY, SS(After Gritchamber), PHOSPHATE(After Gritchamber), ALKALINITY(After Gritchamber) shown in Figure 10-7.

Figure 10-20 shows the best validation and mean squared error performance which exist in 140 epochs.The correlation plot between trained model and training, validation, testing data are shown in Figure 10-21. The correlation equal to 0.999 for training, 0.999 for validation, 0.999 for test. The correlation means how the model is fit the data, as shown the model is fitting the data well more than 99%.

The histogram error plot Figure 10-19 shows that the error near to zero at 0.005544 at this point, and the most results located on this line.

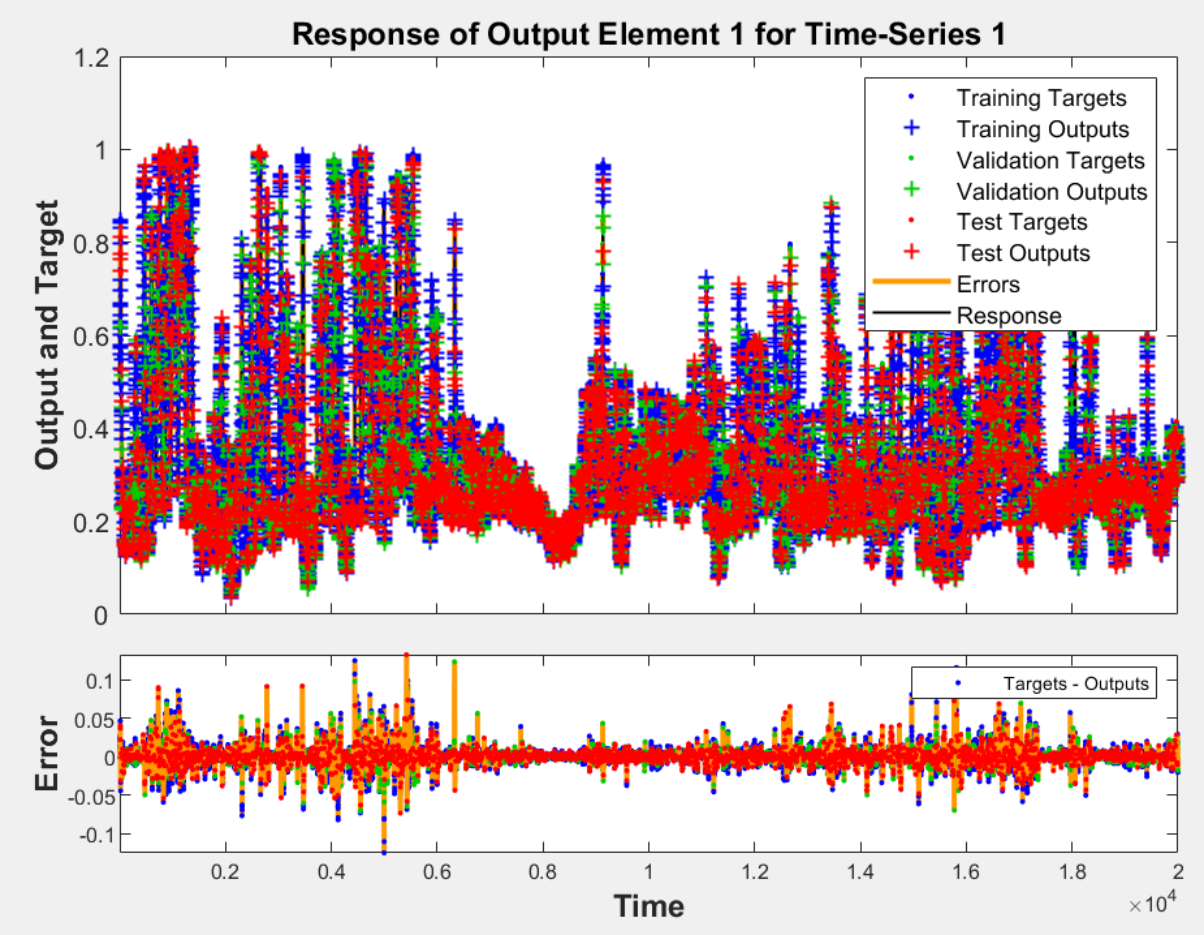


Figure 10-18 The response of the trained NARX ANN MIMO system model and target real data.

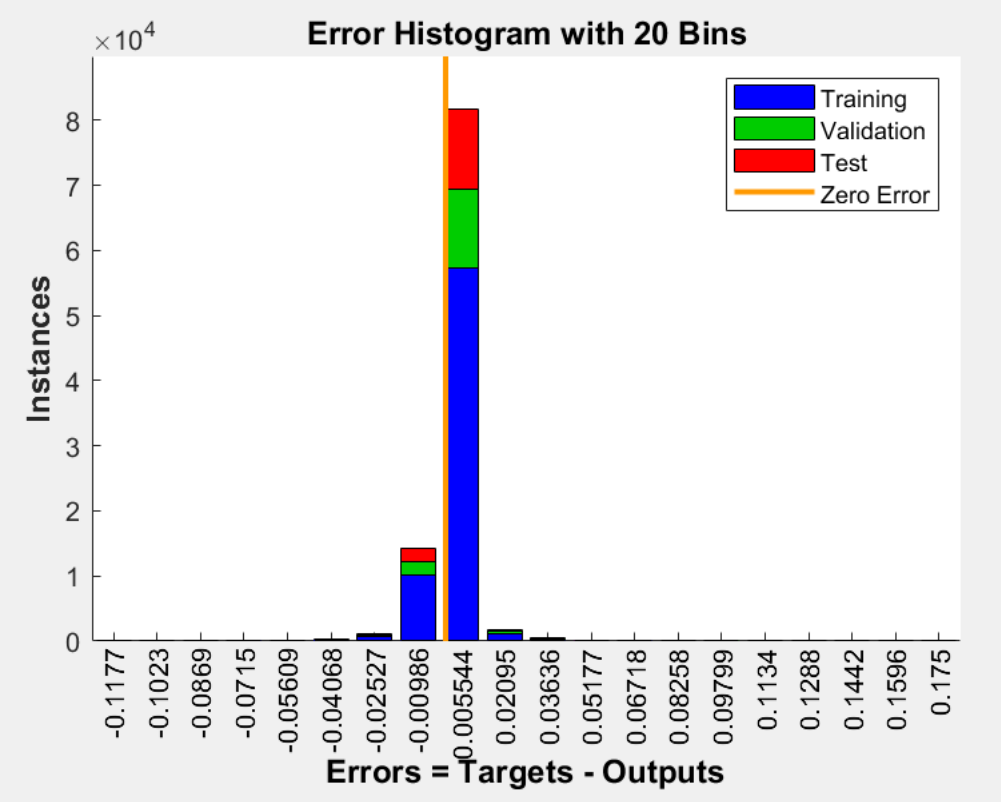


Figure 10-19 Histogram error plot for training NARX ANN for MIMO system.

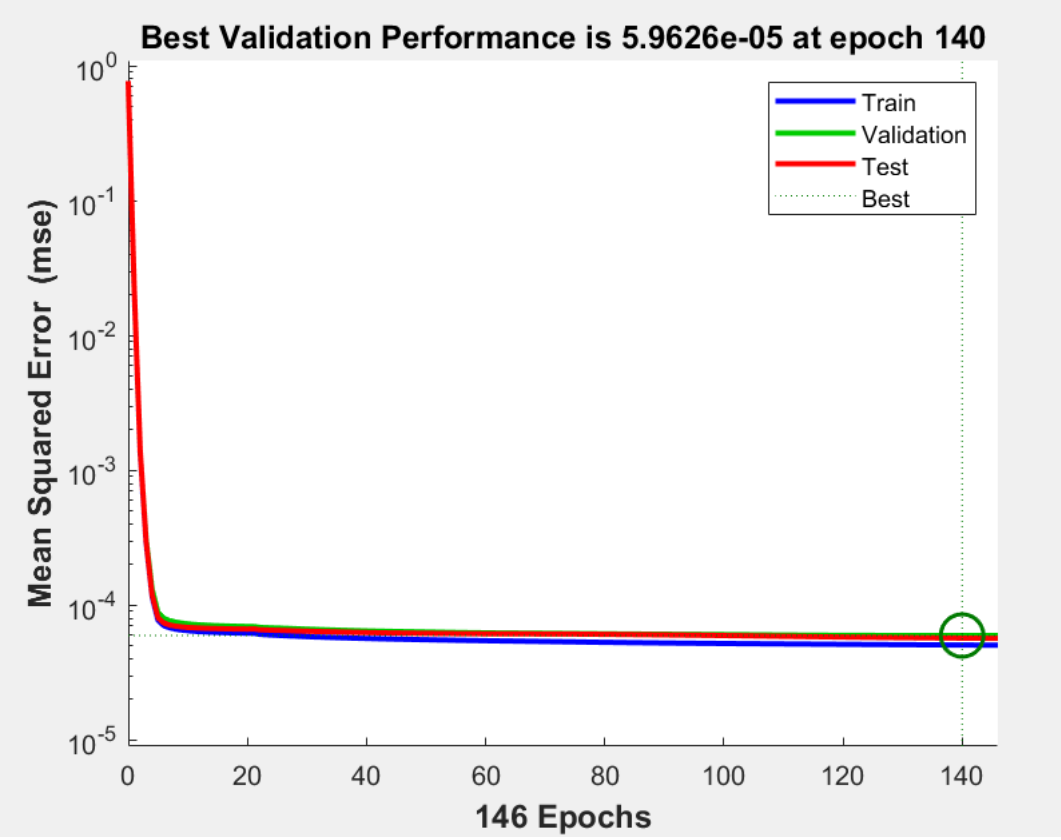


Figure 10-20 Validation performance for MIMO system, MSE of the training data with Epoches.

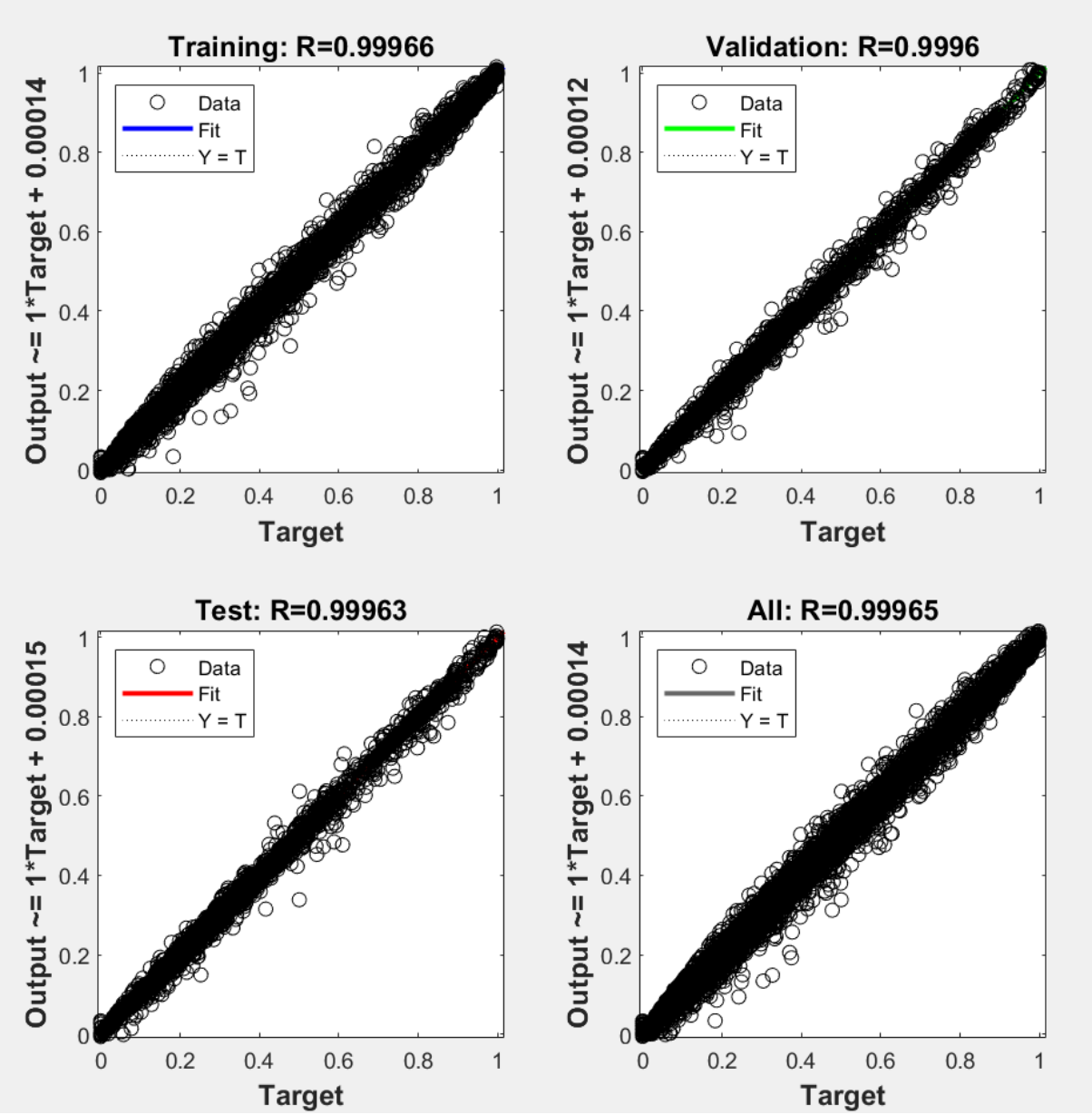


Figure 10-21 Output of the NARX ANN trained model, validation and testing with target data regression correlation for MIMO system.

After training process of NARX model, the model evaluation values indicated in Table 10-4

Table 10-4 MSE and Correlation for NARX ANN model for MIMO system.

	Target values	MSE	R
Training	70003	5.07×10^{-5}	0.9979
Validation	15001	5.96×10^{-5}	0.99789
Testing	15001	5.7×10^{-5}	0.979

2000 sequence samples are used to see the behavior of the trained model. These 2000 samples are different from training samples. Inputs, Model output and Target are shown in Figure 10-22 Figure 10-23, Figure 10-24, Figure 10-25, Figure 10-26 and Figure 10-27.

The predicted response of turbidity, SS, phosphate is close to real response. Alkalinity response has no information, training set of Alkalinity is constant and equal to zero, that means the prediction of Alkalinity will be always zero because the training set has no information.

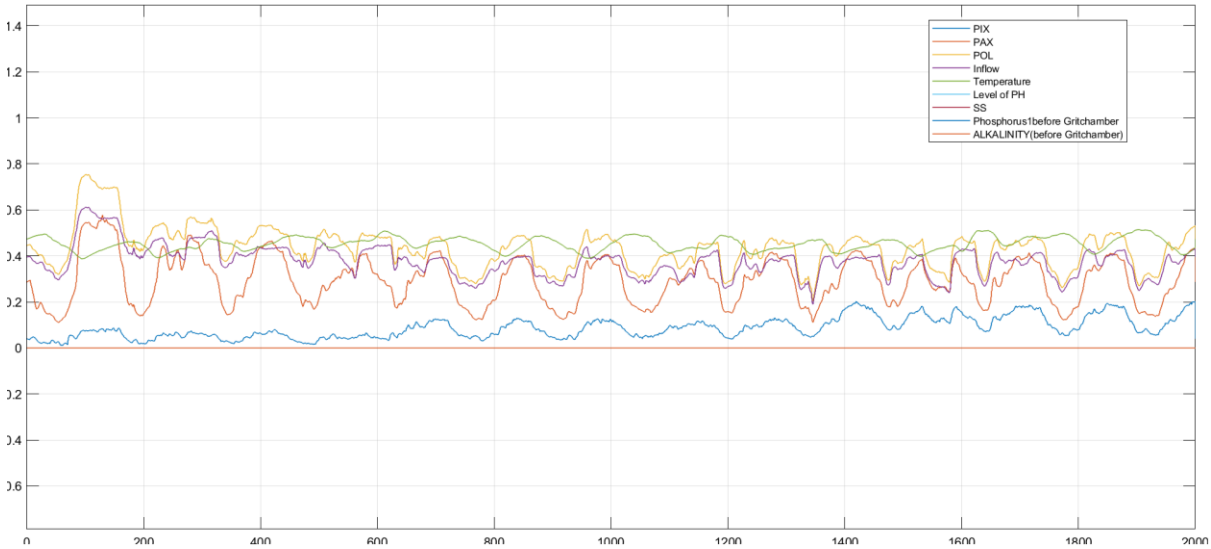


Figure 10-22 (a)

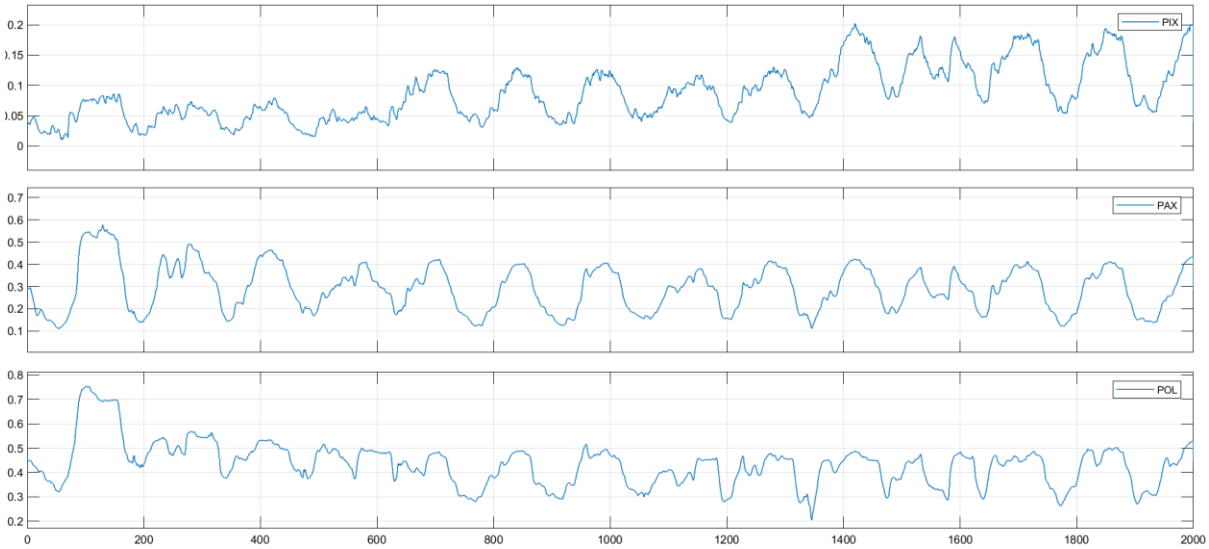


Figure 10-22 (b)

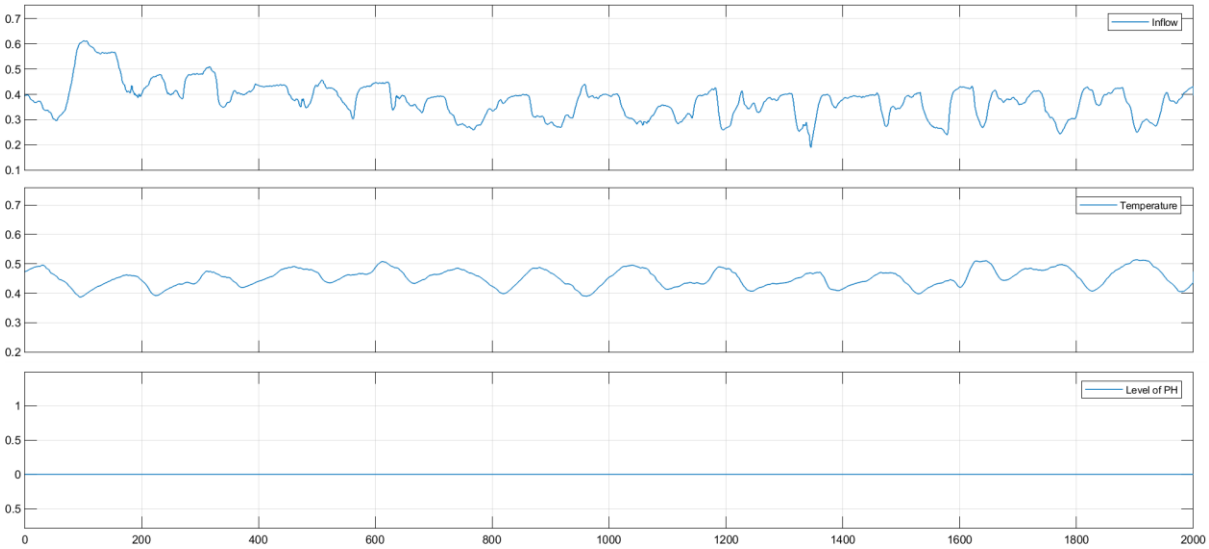


Figure 10-22 (c)

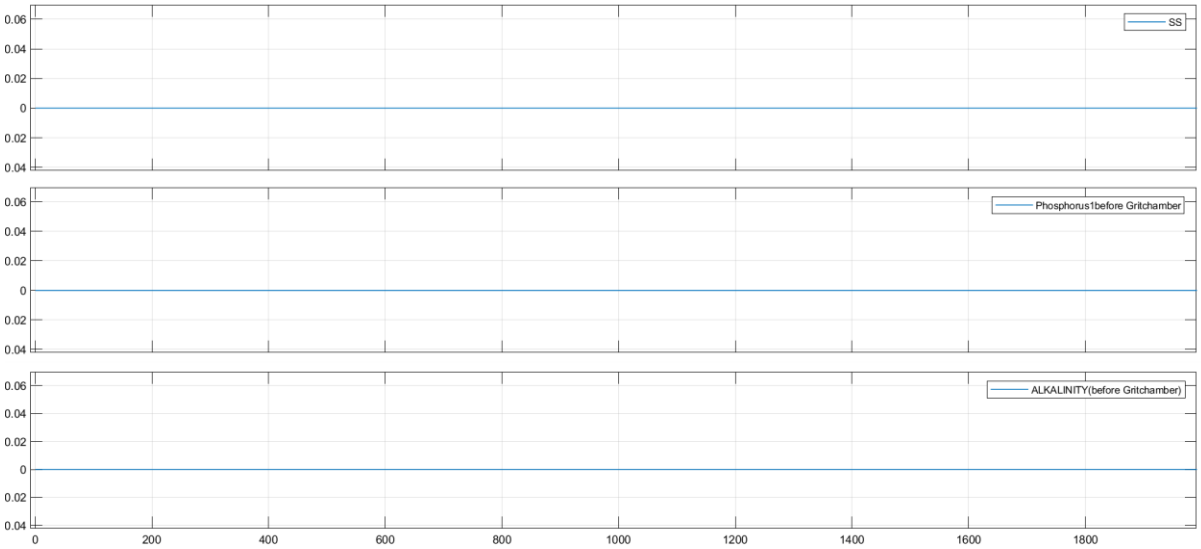


Figure 10-22 (d)

Figure 10-22 (a)Testing inputs data set for MIMO system, the inputs are (b)PIX, PAX, POL, (c)Inflow, Temperature, Level of PH, (d)SS before Gritchamber, PHOSPHATE(before Gritchamber), ALKALINITY(before Gritchamber). The values are normalized where y-axis is the percentage and x-axis are the step number, the time between steps is 10 minutes.

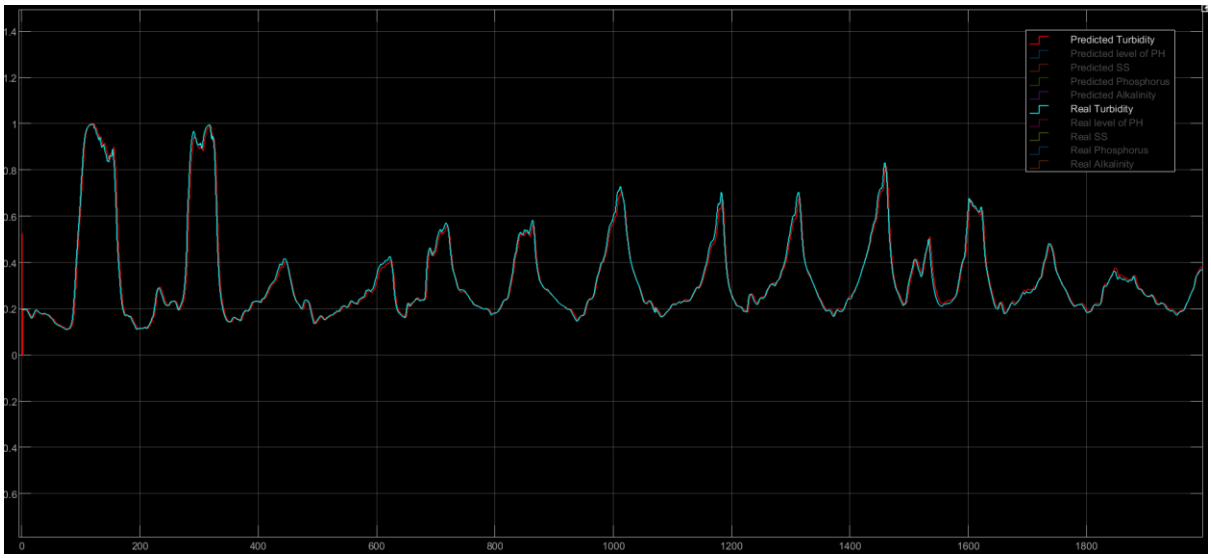


Figure 10-23 NARX ANN model response for Turbidity in MIMO system. Parallel series NARX architecture. X-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a turbidity FTU. All values are normalized.

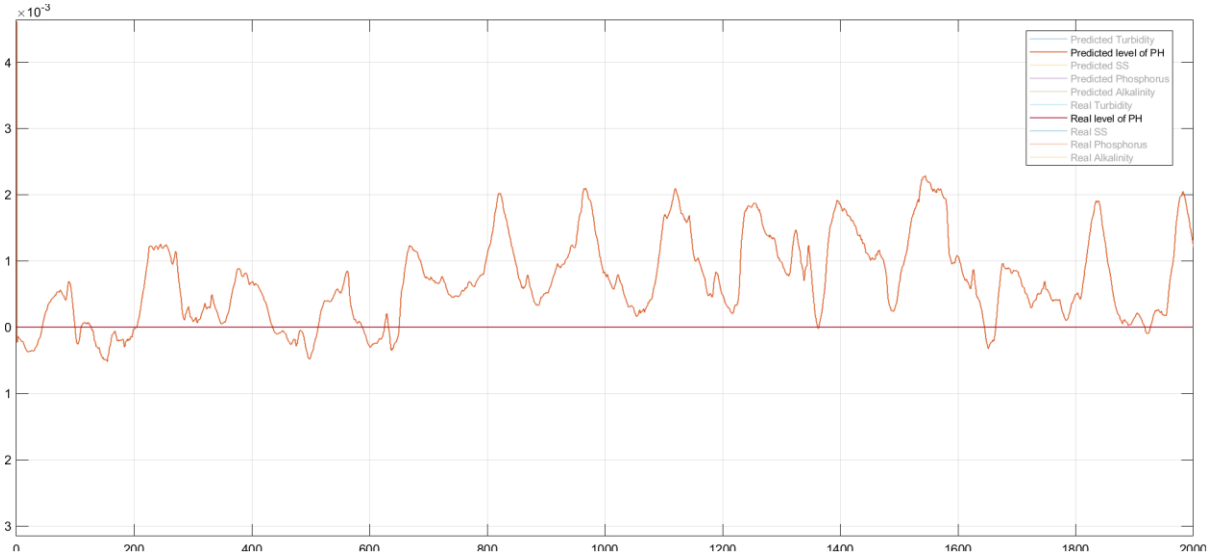


Figure 10-24 NARX ANN model response for level of PH in MIMO system. Parallel series NARX architecture. X-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a level of PH. All values are normalized.

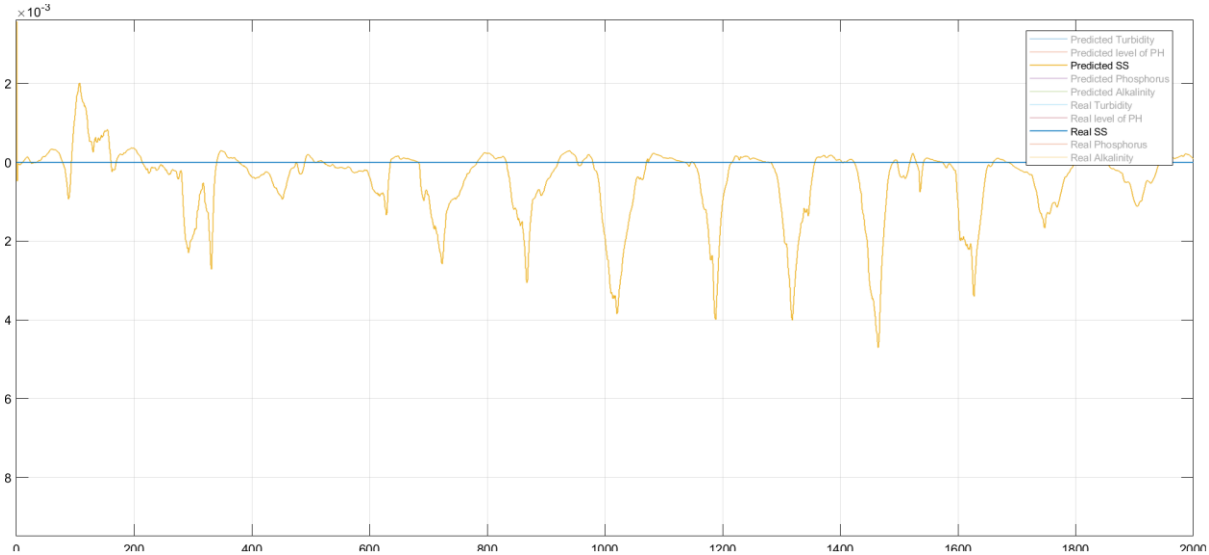


Figure 10-25 NARX ANN model response for SS in MIMO system. Parallel series NARX architecture. X-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a SS. All values are normalized.

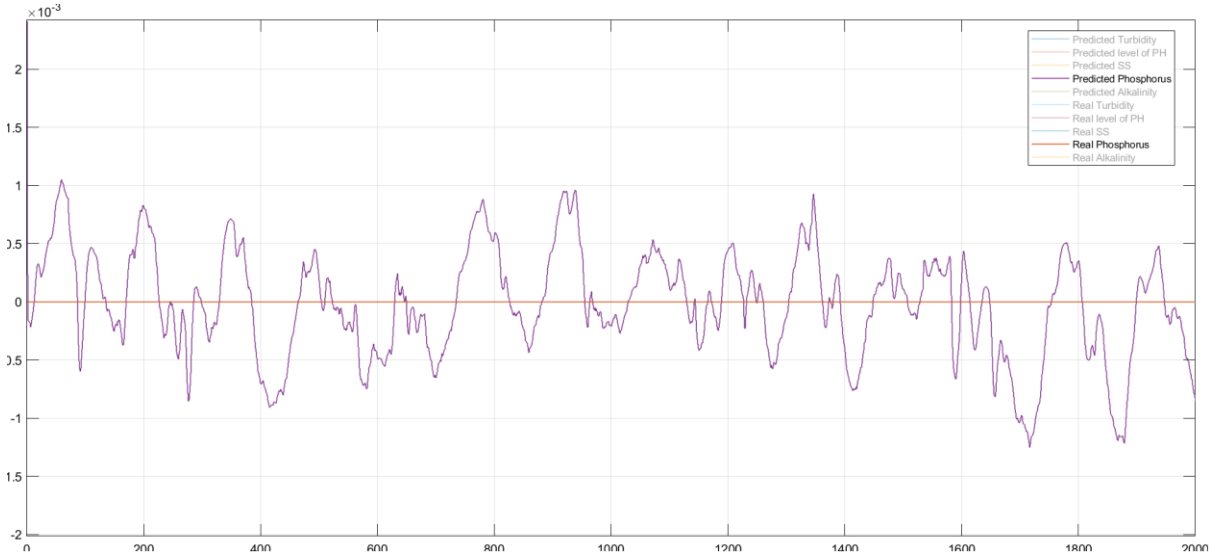


Figure 10-26 NARX ANN model response for phosphate in MIMO system. Parallel series NARX architecture. X-axis is the number of samples, the time between samples is 10 minutes. Y-axis is a phosphate. All values are normalized.

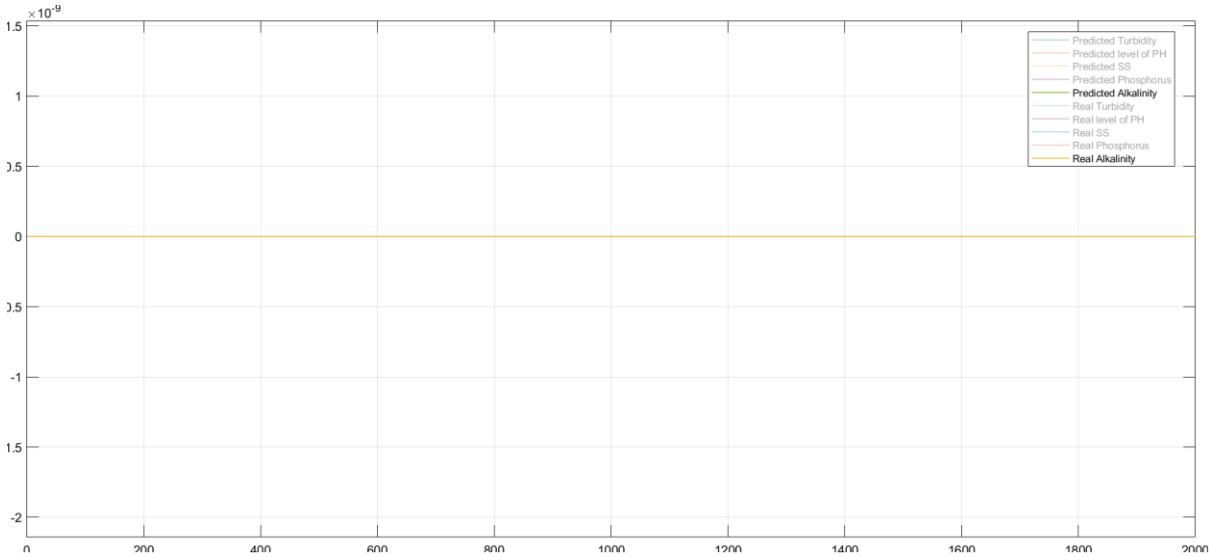


Figure 10-27 NARX ANN model response for Alkalinity in MIMO system. Parallel series NARX architecture. X-axis is the number of samples, the time between samples is 10 minutes. Y-axis is Alkalinity. All values are normalized.

10.6 Discussion

The NARX ANN model results correspond to the theoretical studies and simulation. The model is tested with a process simulator and a real process. The results from both tests show high performance. The simulator model shows the ability of NARX to handle time delay in the process. NARX ANN model can represent a time series dynamic model as a black box. The random operation data series can train the NARX ANN model, but that will not guarantee a good model, because the training data could have not enough information that will lead to underfitting. NARX ANN model does not need a time delay estimation, which is needed in classical dynamic system modelling. It is tricky to have a good time delay estimation which needs an experiment to do this estimation. NARX ANN model does not need system identification, it will adapt the process automatically from training data.

The parallel series NARX ANN model gives a high performance because the feedback of NARX comes from the real process. That will cancel the error propagation. If feedback is chosen from NARX model the performance will be worse due to error propagation.

NARX neural network with tapped time delay, shows high performance in parallel series architecture as shown in Figure 10-13, Figure 10-17, Figure 10-23, Figure 10-24, Figure 10-25, Figure 10-26 and Figure 10-27, but Figure 10-14 shows that the response in the parallel architecture has noise response. The reason is that the predicted value depends on the previous input and output, in this case, tapped time delay equal to 1, the next output depends on previous steps values. During the simulation, when the simulation starts, these initial values are equal to zero which lead to fault prediction because the initial value in the real process is not zero.

If the first prediction is not correct, then it will lead next values also being incorrect. When the system runs within parallel series architecture the initial values will have correct values, the initial values will get correct values. As shown in Figure 10-13, Figure 10-17 when the mode is changed from parallel series architecture to parallel architecture, the model continuously provide a good estimation for a period of future steps with small errors.

The number of inputs in the NARX ANN can be calculated from equation (13-1).

$$NARX\ ANN\ inputs = model\ inputs\ \times td + model\ outputs \times (td) \qquad (13\text{-}1)$$

where td is the number of tapped time delay.

If the sampling time is small and the tapped time delay is large, the NARX network will be quite large and computational work will be pretty complex and time-consuming. Therefore, the system will require large active memories to accomplish the task. If there are three inputs, one output, tapped time delay equal to 1 second and sampling time is 1 ms, the number of the tapped time delays will be 1000 as equation (13-2)

$$td = \frac{time\ delay}{sampling\ time} \qquad (13\text{-}2)$$

The number of inputs will be $3 \times 1000 + 1 \times (1000) = 4000$ by using (13-1)

The computer might not be able to complete the task. To solve this problem, the number of inputs should be reduced. That can be performed by many ways. The easiest way is increase sampling time. That will reduce the quality and accuracy of the model. If the order of the system is known or assumed, the NARX model can be modified to get the lowest variables. If a system is first order the equation (13-1) can be modified to (13-3)

$$inputs \times td + outputs \quad (13\text{-}3)$$

The model input is reduced by equation (13-3), it is very advantage to know the maximum time delay in the process that will determine the number of tapped time delays.

The parallel series architecture and parallel architecture together method is suitable to run in parallel with process, and it is an effective way to model dynamic system with/ without time delay. It has ability to give good information about the process even if the connection with output sensor. The system can predict the output from the model for many time steps with good accuracy as shown in the results.

The results from NARX that performed by MATLAB shows exactly the same result with and without time delay. That means the process data has no time delay which is assured from Python results. Data set shows the sampling time equal to 10 min. If the time delay is lower than 10 minutes, the data set will not contain this delay because the data has no information between two samples. As illustrated in the Figure 10-28The system will not notice any delay

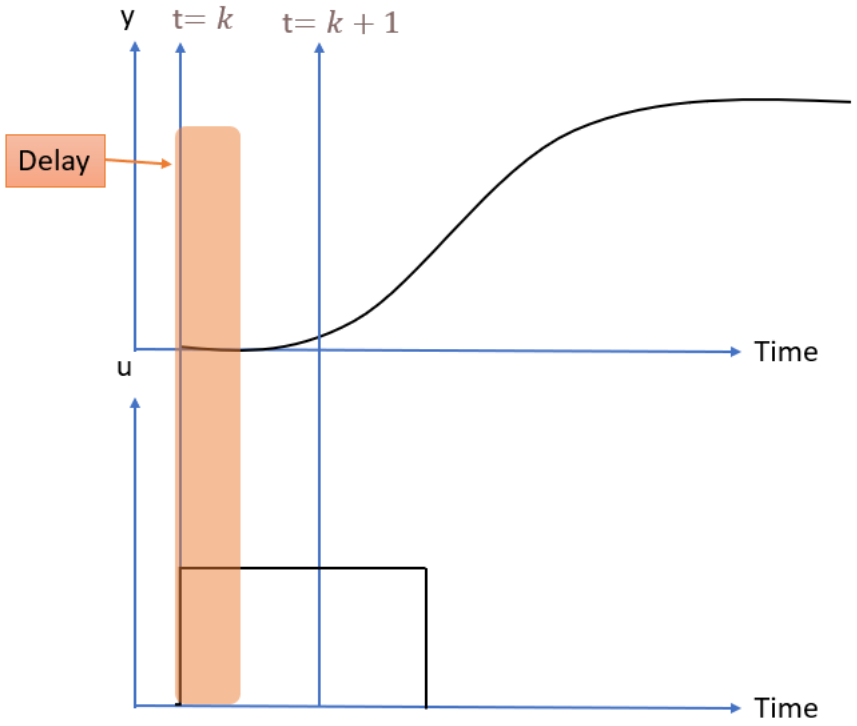


Figure 10-28 Effect of time delay in low sampling rate. System input in the lower figure and response in the higher figure, a time delay is smaller than sampling time, the recoded data will have measurement at time t=k and t=k+1 and so on.

The NARX neural network has the ability to track time delay changing, the time delay changes according to one or more input variables, as shown in the simulation results. the network has the ability to remember previous inputs and feedback output, in order to give correct prediction output. The network remembers all the time the previous steps equal to a number of a tapped time delay of all inputs and outputs. The drawback of this tracking is much time and computer memory consuming and much tough computational work.

The mean squared error is less with MATLAB compared to Python due to the optimizer algorithm. MATLAB uses Leverberg optimizer, and python code uses Adam optimizer. Leverberg gives better results ($MSE = 2 \times 10^{-5}$) at epoch 26 than Adam ($MSE = 10 \times 10^{-5}$) at epoch 200. MATLAB converges faster than Python.

The training data set should be selected and checked carefully, the variation of the input and output data is important. If the input is changing and the output is constant for a long time, that means the input has no impact on the output. If the input is constant, it is not possible to know the relation between the input and output. the data which comes from constant input cannot learn the model because it has no information. The neural network is smart to know if the data has information or not.

The PH value in the data set that was used for training and testing the MISO model has zero value that means this variable has no effect on the model. Therefore, experimental design is important to give information about the influence of each input. By having zero or a constant

value of PH without changing, it is impossible to judge the effect of PH in the neural network model. If the PH variable is removed from the model the model will still be the same without any changing.

Even the neural network model is well expressed data set, the experimental design is needed to have good data set that present the real process. Because the training data will train the model and it will give experience to the model to work properly in the future. If the trained model has not had a good experience, the model response will have a large error.

MATLAB software has a powerful tool like Simulink which is used in this process for dynamic system simulation. The ANN tools give information about errors, MSE, histogram, validation performance, correlations which are very good for network evaluation. However, the problem with MATLAB is the activation function of ANN is not controllable for time series only the number of hidden layers. It is a commercial tool, so everyone cannot afford this software. On the other hand, Python scrip is free and more flexible with ANN tools (Keras library). It is possible to use sequential, and functional methods to form ANN. Keras library does not have Leverberg-Marquardt optimizer which is better than Adam optimizer to deal with dynamic systems and time series. Python can work with Linux embedded systems; it is possible to use it in real-time processes and it is not memory consuming like MATLAB. Python needs a deep understanding of activation functions to how to select suitable activation functions and layers, while it is not needed in MATLAB.

11 Conclusion

This Project is aimed to model the chemical dosing at a water resource recovery facility, in order to use this model in the optimization process. Modeling of the real process and simulation is done, and the proper studies have been covered in this thesis.

The results of real data implementation show that NARX neural network handles time delay in the time series process in an effective way. It succeeds to model the dynamic system and shows a high performance of MIMO system modeling.

The model is data-driven if any process does not have enough information to make a model for optimization, then NARX model which is used in this thesis would be the best solution. Since this NARX model has given very good performance to predict turbidity of water and other output such as Alkalinity and phosphate.

12 Future work

This thesis can be extended by using powerful optimization process like linear quadratic optimization method and Recurrent neural network optimization for (MPC) to reduce the usage of chemical materials inputs PIX, PAX, POL.so that the process will be more cost effective.

The NARX model can be in parallel with real process as NARX parallel series architecture. The NARX can give good prediction for long period, so the NARX model can be used in model predictive control to optimize the using of PIX, PAX, POL as shown in Figure 12-1

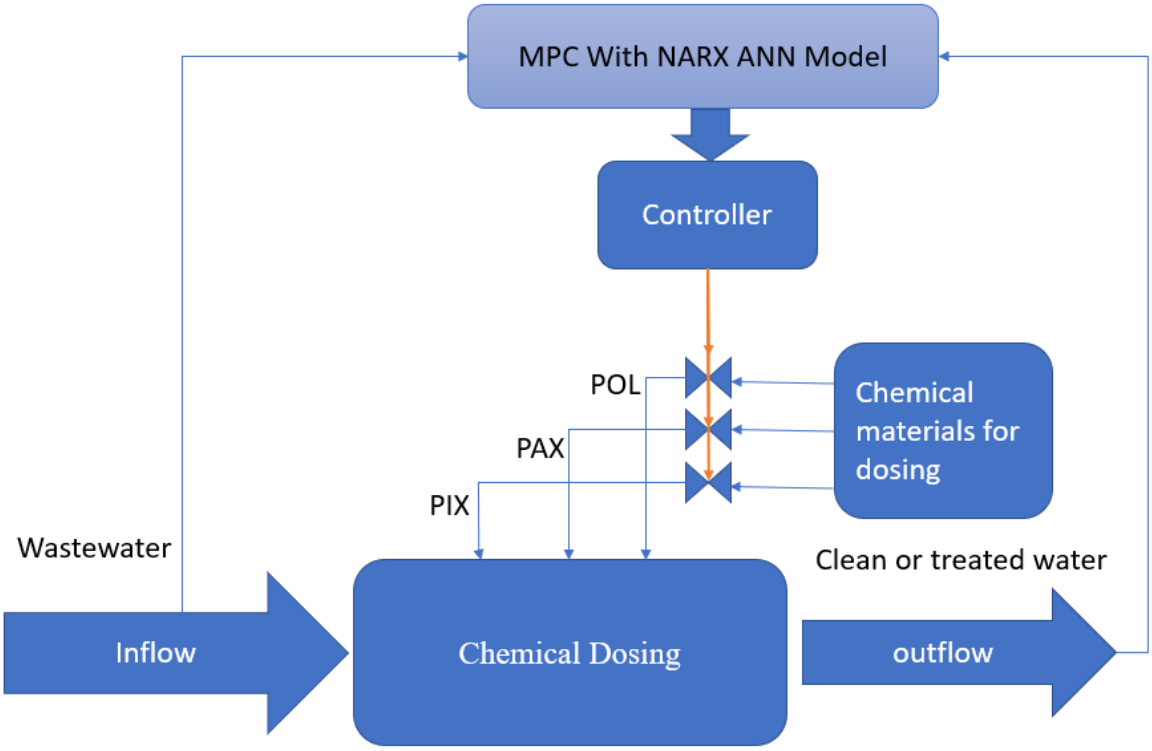


Figure 12-1 Model predictive control (MPC) with NARX ANN model in chemical dosing process

13 References

[1] J. S.-T. Pettersen, "Control of chemical dosing at a water," master thesis at USN, 2020.

[2] m. w. q. people, Water Resource Recovery Management, 2012.

[3] H.Ødegaard, Fjerning Av Næringstoffer Ved Rensing Av Avlpsvann, Tapir Forlag.

[4] F. A. Haugen, Automatic Control 6th,p97-107, lecture note at USN, 2019.

[5] "wikipedia.org," 2 May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network#cite_note-1.

[6] N. Siddique and H. Adeli, "Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing," Oxford, UK, John Wiley & Sons Ltd, 2013, p. Chapter 4.

[7] A. Tealab, "Time series forecasting using artificial neural networks methodologies: A systematic review," *Future Computing and Informatics Journal. 3 (2): 334–340. doi:10.1016/j.fcij.2018.10.003. ISSN 2314-7288.*, 2018-12-01.

[8] "Machine learning," 2020. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning.

[9] I. Wolfram Research, "LeastSquaresFitting," 1999-2021. [Online]. Available: https://mathworld.wolfram.com/LeastSquaresFitting.html.

[10] Å. U. E. Kim H. Esbensen, "Multivariate Data Analysis In Practice 5th Edition," in *An Introduction to Multivariate Data Analysis and Experimental Design*, Oslo, CAMO Software As, 1996-2002.

[11] T. P. S. University, "Multiple Linear Regression (MLR) Model & Evaluation," 2018. [Online]. Available: https://online.stat.psu.edu/stat462/node/132/.

[12] "Mean squared error," 31 July 2021. [Online]. Available: https://en.wikipedia.org/wiki/Mean_squared_error.

[13] "intuition-of-adam-optimizer," 24 Oct Oct 2020. [Online]. Available: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/.

[14] J. Brownlee, "adam-optimization-algorithm-for-deep-learning," 3 July 2017 . [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[15] "Stochastic gradient descent," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[16] L. Artificial Intelligence Techniques, "algorithms to train a neural network," 2021. [Online]. Available: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network.

[17] "wikipedia," 25 August 2021. [Online]. Available: https://en.wikipedia.org/wiki/Nonlinear_autoregressive_exogenous_model.

[18] E. DIACONESCU, "The use of NARX Neural Networks to predict Chaotic Time Series," *Electronics, Communications and Computer Science Faculty,University of Pitesti,Issue 3, Volume 3, ,* March 2008.

[19] M. Halstensen, "Multivariate Data Analysis course, lecture note," USN, 2020.

[20] J. J. Klemes, P. S. Varbanov and P. Y. Liew, 24th European Symposium on Computer Aided Process Engineering, p670, Elsevier Science & Technology, 17.07.2014.

[21] M. L. M. P. L. A. Rights, "Machine Learning Mastery," 2021. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[22] "wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Overfitting.

[23] "Feature scaling," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Feature_scaling.

[24] "overfitting-vs-underfitting," wiki, 2021. [Online]. Available: https://docs.paperspace.com/machine-learning/wiki/overfitting-vs-underfitting. [Accessed 29 sep 2021].

14 List of tables and charts

Appendices

Topic Description

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

University of South-Eastern Norway

FMH606 Master Thesis

Title: Modeling of the chemical dosing at a water resource recovery facility (WRRF)

USN supervisor: Finn Aakre Haugen, Docent

External partner: VEAS WRRF (Vestfjorden Avløpsselskap) with Jonas Pettersen as main contact person and external supervisor.

Task background:
VEAS is Norway's largest water resource recovery facility (WRRF) - historically denoted wastewater treatment plant - located at Slemmestad, south of Oslo, and is treating the inflow from the combined system (sewage and grey water) in different process steps. The wastewater treatment process is comprised of three treatment steps. One of these steps is the chemical treatment in which precipitation chemicals (coagulants) and a polymer (flocculant) are added to the wastewater so that particles can grow into sediment. The objective of the chemical treatment step is to precipitate and separate particular contamination from the water phase.

The present chemical dosing strategy in this step is based on adding chemicals in a dose proportional to the wastewater flow. The dosage is corrected by an assumed time-dependent concentration of phosphate and ammonium in the wastewater, however, the dosage is not linked or corrected by online feedback control.

The present strategy is probably suboptimal regarding economical cost of chemicals when the water flow is high and the concentration of contamination in the water is low. Also, the purification is not reaching its potential.

VEAS wants to use model-based control to control this process. The aim of this thesis is to prepare for model-based control by deriving models useful for control. However, developing and deploying model-based controllers are not an aim of the thesis.

The present chemical dosing strategy in this step is based on adding chemicals in a dose proportional to the wastewater flow. The dosage is corrected by an assumed time-dependent concentration of phosphate and ammonium in the wastewater, however, the dosage is not linked or corrected by online feedback control.

The present strategy is probably suboptimal regarding economical cost of chemicals when the water flow is high and the concentration of contamination in the water is low. Also, the purification is not reaching its potential.

VEAS wants to use model-based control to control this process. The aim of this thesis is to prepare for model-based control by deriving models useful for control. However, developing and deploying model-based controllers are not an aim of the thesis.

Task description

- Deriving and evaluating alternative process models for control and analysing purposes. These models should be data driven (black box models or grey box models). Machine learning models, as neural network models, are of particular interest. An overview over relevant data driven models and modelling techniques should be made.
- Real data from VEAS are available for model adaptation.
- Implementing a simulator for checking the applicability of the selected modeling strategies.
- Further development of the models to suit the goals of VEAS.
- Study of the features of the models.

Student:

Anas Muhamad Hashem Aldabbagh (IIA master programme).

Practical arrangements:

The workplace is mainly the home or USN. If possible, according to the visiting rules defined by USN and by VEAS, one or more visits at VEAS will be arranged.

Supervision:

As a general rule the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature): 31.1 2021 

Student (write clearly in all capitalized letters):

Student (date and signature): 10.02.2021 

Appendix A MISO system simulator

```
#
================================================================
============

import matplotlib.pyplot as plt

import numpy as np

import modelparameters

import sim_function

# %% Time settings:


ts = 1  # Time-step [min]

t_start = 0.0  # [min]

t_stop = 850.0  # [min]

N_sim = int((t_stop-t_start)/ts) + 1


# Solvermode= 'first_order' or second_order
================================================================
============

Solver_mode='first_order'

#%%% SYSTEM INPUT ARRAYS

# PIX input

U_pix = np.zeros(N_sim)

U_pix[int(0/ts):] += 1

U_pix[int(100/ts):] += -1


# PAX input

U_pax = np.zeros(N_sim)

U_pax[int(200/ts):] += 1
```

```
U_pax[int(300/ts):] += -1
# POL input
U_pol = np.zeros(N_sim)
U_pol[int(402/ts):] += 1
U_pol[int(650/ts):] -= 1
#
============================================================
============
# %% Arrays for plotting:


t_plot_array = np.zeros(N_sim)
ypix_plot_array = np.zeros(N_sim)
ypax_plot_array = np.zeros(N_sim)
ypol_plot_array = np.zeros(N_sim)
yTurb_plot_array = np.zeros(N_sim)
ypix_k=0
ypax_k=0
ypol_k=0
#
============================================================
============
# # for second order
y1pix_k=0
y1pax_k=0
y1pol_k=0


y2pix_k=0
y2pax_k=0
y2pol_k=0
```

```
for k in range(0, N_sim):
    t_k = k*ts
    #First order Euler-forward integration (Euler step):
    if Solver_mode=='first_order':
        ypix_kp1=sim_function.forward_euler(ts,modelparameters.turbParrameters['pix']['K'],
                                U_pix[k],modelparameters.turbParrameters['pix']['Tc'],
                                ypix_k)
       ypax_kp1=sim_function.forward_euler(ts,modelparameters.turbParrameters['pax']['K'],
                                U_pax[k],modelparameters.turbParrameters['pax']['Tc'],
                                ypax_k)
        ypol_kp1=sim_function.forward_euler(ts,modelparameters.turbParrameters['pol']['K'],
                                U_pol[k],modelparameters.turbParrameters['pol']['Tc'],
                                ypol_k)

    if Solver_mode=='second_order':

[y1pix_k,y2pix_k]=sim_function.second_order(ts,modelparameters.turbParrameters['pix']['
K'],
                                U_pix[k],modelparameters.turbParrameters['pix']['Tc1'],
                                modelparameters.turbParrameters['pix']['Tc2'],
                                y1pix_k,y2pix_k)
        ypix_kp1=y2pix_k
y1pax_k,y2pax_k=sim_function.second_order(ts,modelparameters.turbParrameters['pax']['
K'],
                                U_pax[k],modelparameters.turbParrameters['pax']['Tc1'],
                                modelparameters.turbParrameters['pax']['Tc2'],
                                y1pax_k,y2pax_k)
        ypax_kp1=y2pax_k
```

```
y1pol_k,y2pol_k=sim_function.second_order(ts,modelparameters.turbParrameters['pol']['K
'],
                                U_pol[k],modelparameters.turbParrameters['pol']['Tc'],
                                modelparameters.turbParrameters['pol']['Tc2'],
                                y1pol_k,y2pol_k)
        ypol_kp1=y2pol_k
    # Storage for plotting:
    t_plot_array[k] = t_k
    ypix_plot_array[k] = ypix_k
    ypax_plot_array[k] = ypax_k
    ypol_plot_array[k] = ypol_k
    yTurb_plot_array[k]=1-(ypix_k+ypax_k+ypol_k)



    # Time shift:
    ypix_k = ypix_kp1
    ypax_k = ypax_kp1
    ypol_k = ypol_kp1
   # %% Plotting:
plt.close('all')
plt.figure(1)

plt.subplot(2, 1, 1)
plt.plot(t_plot_array, 1-ypix_plot_array, 'b')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
```

```
plt.ylim(0.69, 1.1)
plt.xlim(t_start,200)
plt.xlabel('t [min]')
plt.ylabel('[FTU]')
plt.legend(('Response due to PIX',))
plt.subplot(2, 1, 2)
plt.plot(t_plot_array, U_pix, 'g')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.ylim(0, 2)
plt.xlim(t_start, 200)
plt.xlabel('t [min]')
plt.ylabel('[%]')
plt.legend(('PIX input',))
plt.show()
plt.figure(2)
plt.subplot(2, 1, 1)
plt.plot(t_plot_array, 1-ypax_plot_array, 'b')
plt.ylim(0.65, 1.1)
plt.xlim(150, 500)
plt.xlabel('t [s]')
plt.ylabel('[FTU]')
plt.legend(('Response due to PAX',))
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.subplot(2, 1, 2)
```

```
plt.plot(t_plot_array, U_pax, 'g')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.ylim(0, 2)
plt.xlim(150, 500)
plt.xlabel('t [min]')
plt.ylabel('[%]')
plt.legend(('PAX input',))
plt.figure(3)
plt.subplot(2, 1, 1)
plt.plot(t_plot_array,1- ypol_plot_array, 'b')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.ylim(0.96, 1.01)
plt.xlim(300, 850)
plt.xlabel('t [min]')
plt.ylabel('[FTU]')
plt.legend(('Response due to POL',))

plt.subplot(2, 1, 2)
plt.plot(t_plot_array, U_pol, 'g')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.ylim(0, 2)
plt.xlim(300, 850)
```

```
plt.xlabel('t [min]')
plt.ylabel('[%]')
plt.legend(('POL input',))

#----
plt.figure(4)

plt.subplot(2, 1, 1)
plt.plot(t_plot_array,yTurb_plot_array, 'b')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.ylim(0.6, 1.01)
plt.xlim(0, 850)
plt.xlabel('t [min]')
plt.ylabel('[FTU]')
plt.legend(('System response',))

plt.subplot(2, 1, 2)
plt.plot(t_plot_array, U_pol, 'g--', label='POL')
plt.plot(t_plot_array, U_pix, 'b:', label='pix')
plt.plot(t_plot_array, U_pax, 'r', label='pax')
plt.xlabel('t [sec]')
plt.ylabel('[%]')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
plt.ylim(0, 2)
```

```
plt.xlim(0, 850)
plt.xlabel('t [min]')
plt.ylabel('[%]')
plt.legend()


plt.figure(5)



fig, ax = plt.subplots()
ax.plot(t_plot_array, U_pol, 'k--', label='POL')
ax.plot(t_plot_array, U_pix, 'k:', label='pix')
ax.plot(t_plot_array, U_pax, 'k', label='pax')
plt.xlabel('t [min]')
plt.ylabel('[%]')
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', linestyle='-', alpha=1)
legend = ax.legend(loc='upper right', shadow=True, fontsize='x-large')

# Put a nicer background color on the legend.
legend.get_frame().set_facecolor('C0')


plt.show()
```

Simulation time setting:

```
# %% Time settings:
ts = 0.1  # Time-step [min]
t_start = 0.0  # [min]
```

```
t_stop = 850.0  # [min]
N_sim = int((t_stop-t_start)/ts) + 1
```

The values of the inputs PIX, PAX, POL are stored in arrays as the following

```
# PIX input
U_pix = np.zeros(N_sim)
U_pix[int(0/ts):] += 1
U_pix[int(100/ts):] += -1

# PAX input
U_pax = np.zeros(N_sim)
U_pax[int(200/ts):] += 1
U_pax[int(300/ts):] += -1

# POL input
U_pol = np.zeros(N_sim)
U_pol[int(402/ts):] += 1
U_pol[int(650/ts):] -= 1
```

The simulation loop :

```
for k in range(0, N_sim):
    t_k = k*ts
    #First order Euler-forward integration (Euler step):
    if Solver_mode=='first_order':
        ypix_kp1=sim_function.forward_euler(ts,modelparameters.turbParrameters['pix']['K'],
                                U_pix[k],modelparameters.turbParrameters['pix']['Tc'],
                                ypix_k)
ypax_kp1=sim_function.forward_euler(ts,modelparameters.turbParrameters['pax']['K'],
                                U_pax[k],modelparameters.turbParrameters['pax']['Tc'],
```

```
                                ypax_k)
        ypol_kp1=sim_function.forward_euler(ts,modelparameters.turbParrameters['pol']['K'],
                                U_pol[k],modelparameters.turbParrameters['pol']['Tc'],
                                ypol_k)
    if Solver_mode=='second_order':

[y1pix_k,y2pix_k]=sim_function.second_order(ts,modelparameters.turbParrameters['pix']['
K'],
                                U_pix[k],modelparameters.turbParrameters['pix']['Tc1'],
                                modelparameters.turbParrameters['pix']['Tc2'],
                                y1pix_k,y2pix_k)
        ypix_kp1=y2pix_k
y1pax_k,y2pax_k=sim_function.second_order(ts,modelparameters.turbParrameters['pax']['
K'],
                                U_pax[k],modelparameters.turbParrameters['pax']['Tc1'],
                                modelparameters.turbParrameters['pax']['Tc2'],
                                y1pax_k,y2pax_k)
        ypax_kp1=y2pax_k

y1pol_k,y2pol_k=sim_function.second_order(ts,modelparameters.turbParrameters['pol']['K
'],
                                U_pol[k],modelparameters.turbParrameters['pol']['Tc'],
                                modelparameters.turbParrameters['pol']['Tc2'],
                                y1pol_k,y2pol_k)
        ypol_kp1=y2pol_k
    # Storage for plotting:
    t_plot_array[k] = t_k
    ypix_plot_array[k] = ypix_k
    ypax_plot_array[k] = ypax_k
```

```
ypol_plot_array[k] = ypol_k
yTurb_plot_array[k]=1-(ypix_k+ypax_k+ypol_k)
# Time shift:
ypix_k = ypix_kp1
ypax_k = ypax_kp1
ypol_k = ypol_kp1
```

Appendix B Time delay estimation code

```
import numpy as np
import pandas as pd
# %% Time settings:
ts=1
t_start = 0.0  # [min]
t_stop = 850 # [min]
#"----------------------------------------"
# load training data
train_df = pd.read_csv("xtrain_data.csv")
u1=train_df[['U_pax']].values
train_df = pd.read_csv("ytrain_data.csv")
y=train_df[['TURB']].values
t_plot_array = np.linspace(t_start,t_stop,len(y))
delay= np.zeros(len(y))
delay_count=0
Condition_u=0
eps_u=0.1 # noise margin rang
eps_y=0.000099# noise margin rang
for k in range(0, len(y)-5):

    if (y[k+1]>=y[k]-eps_y and y[k+1]<=y[k]+eps_y):# Condition_y
        if (u1[k+1]<u1[k]-eps_u or u1[k+1]>u1[k]+eps_u):#Condition_u
            Condition_u=1

        if Condition_u==1:
```

```
            delay[k]=1
            delay_count+=1


        #print (k)
    else:
        Condition_u=0
print ("time delay=")


print ( (delay_count*ts))
```

Appendix A Training ANN model

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import *
# load training data
train_df = pd.read_csv("train_data.csv")
X1 = train_df.drop('y', axis=1).values
Y1 = train_df[['y']].values
# create neural network model
model = Sequential()
model.add(Dense(3, input_dim=4, activation='linear'))
model.add(Dense(10, activation='linear'))
model.add(Dense(1, activation='linear'))
model.compile(loss="mean_squared_error", optimizer="adam")

# train the model
model.fit(X1,Y1,batch_size=32,epochs=200,verbose=1)
#model.fit(X1,Y1,batch_size=10,epochs=2000,verbose=1)
# Save the model to hard drive
model.save('model.h5')
test_df = pd.read_csv("train_data.csv")
X2 = test_df.drop('y', axis=1).values
Y2 = test_df[['y']].values
# test the model
mse = model.evaluate(X2,Y2, verbose=1)
print('Mean Squared Error: ', mse)
```