PREDICTING FLOW RATE AND LEVEL OF CUSTOM TANK USING PARTICLE FILTER

By

HUZAIFA TAWFEIG AHMED IZZELDIN

FINAL REPORT

Submitted to the Electrical & Electronics Engineering Programme

in Partial Fulfilment of the Requirements

for the Degree

Bachelor of Engineering (Hons)

(Electrical & Electronics Engineering)

Universiti Teknologi Petronas

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

PREDICTING FLOW RATE AND LEVEL OF CUSTOM TANK USING PARTICLE FILTER

by

Huzaifa Tawfeig Ahmed Izzeldin

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

_____

Dr. Vijanth Sagayan Asirvadam
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

December 2009

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

Huzaifa Tawfeig Ahmed Izzeldin

ABSTRACT

One of the basic elements that any plant process facility consists of is control tank system. Custom tank is one type of these control tank systems. Proper controlling of this element will help the facility to work smoothly and it will increase the reliability of the whole system. This project is looking into predicting the state of variables that completely represent the dynamics the custom tank (height of fluid or output flow rate). This prediction can be used in controlling the custom tank (predictive control). The project involve MATLAB SIMULINK simulation program for the custom tank along with different prediction models. The obtained results showed that introducing Multilayer Perceptron (MLP) Neural Network architecture improve the prediction significantly where different algorithms, Recursive Kalman Filter (RKF) and Extended Kalman Filter (EKF) have been used simultaneously to estimate fluid height and output flow. It further shows that introducing cantered finite difference with EKF (particle filter) improve the performance of the network. The report consists of an introduction, problem statement, objectives, literature review and methodology used to solve the problem. It further looks into the obtained results with consistent discussion.

ACKNOWLEDGEMENTS

Firstly, my utmost gratitude to ALLAH the All-Mighty for his uncountable graces upon me and for the successful completion of this project in due course of time.

My enormous thanks to my family members for their priceless support and continuous encouragement. Special gratitude to my Mother for her continuous and unlimited support that kept me going. There is no words can fulfill her effort.

My respectful gratitude goes to my supervisor, Dr. Vijanth Sagayan Asirvadam for his full support in the completion of this project. His constant guidance, helpful comments and suggestions has helped me not only to complete but also to enhance the expected results of the project. His kindness, valuable advice, friendly approach and patience will always be appreciated.

I would like also to express my thanks for the FYP committee for their guidance and management in making all projects run smoothly. A special gratitude to Siti Hawa Tahir for her effort on monitoring and checking the reports to match the university's standards.

Lastly, great appreciation to my friends, who were a constant source of support during my work. To all UTP lecturers, students and staff and to all whose their names are not mentioned here but they provided help directly or indirectly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATION

RLS	Recursive Least Square
RKF	Recursive Kalman Filter
MLP	Multilayer Perceptron
RPE	Recursive Prediction Error
ARX	AutoRegressive eXternal input
SQE	Squared Error
SSE	Sum of Squared Error

CHAPTER 1
INTRODUCTION

1.1 Background of Study

Custom tank is one of the common vessels being used in chemical or plant process industry. The project aims to build a simulation program using MATLAB-SIMULINK to predict the flow rate and level of the fluid inside the custom tank which will enhance the controllability of the system. The system will be mathematically characterized and modelled using MATLAB. Different algorithm will be used to predict the system variables.

1.2 Problem Statement

The estimation of state variables (fluid level and flow rate) is very essential to produce a reliable controlling system for the custom tank. In order to control such a process and to keep level and flow rate at desired set point, an efficient control strategy is required. Also the suggested techniques such as Recursive Least Square, Recursive KALMAN Filter and Multilayer Neural Network need to be implemented under specific conditions. All the above will help in building a reliable system that can be used in the real application.

1.3 Objective and Scope of Study

The Objectives of the project are:

- To do System Identification model for the custom tank that describes the dynamics of the system.
- Build a MATLAB\SIMULINK model that predicts the flow rate and level of the fluid inside the tank using several techniques.

The scope of the study starts with investigating custom tank system focusing on the flow rate and level. Then design a MATLAB model for the tank. Then implement the different estimation techniques on the model.

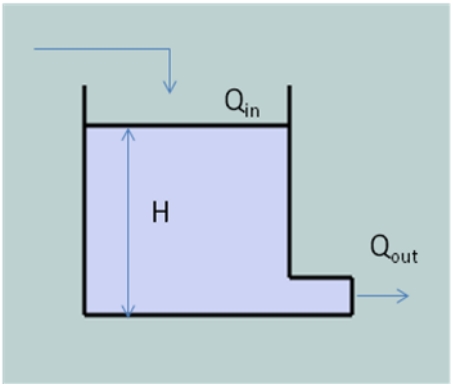
CHAPTER 2
LITERATURE REVIEW

Control tank system is an important element of any plant process facility. Different types of tank or vessels are being used in different process plants. Such tanks like: storage tanks, pressure tanks, mixing tanks and custom tanks have different structure and functionality depending on the purpose of the facility where they are being used. This project focuses only on custom tank system. However, in able to understand the custom tank system other types of control tank systems must be investigated and studied carefully. [1]

The idea behind the project is to build up a filter (program) to predict the dynamic of the custom tank (flow rate and level only). And by using different families and subfamilies of system identification techniques which can be implemented as an integral part of a controller (e.g. adaptive control). [5]

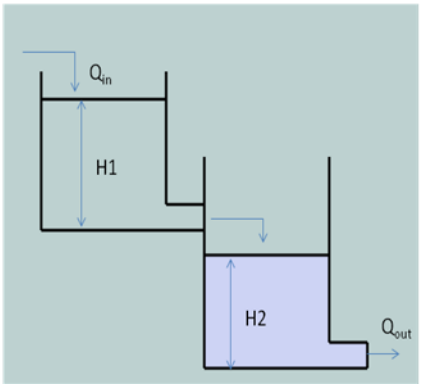
2.1 Types of Control Tank System

There are several types of tank system which varies upon the application. One Basic formation is the single tank. The single tank is used to form other types of tanks such split tank, ladder tank and custom tank system (Figure 1, 2 and 3). However this report will focus on the simulation done for single, split and custom tank formation which will be examined using different estimation/prediction techniques.



$$\frac{dH}{dt} = \frac{Qin - k\sqrt{H}}{A}$$

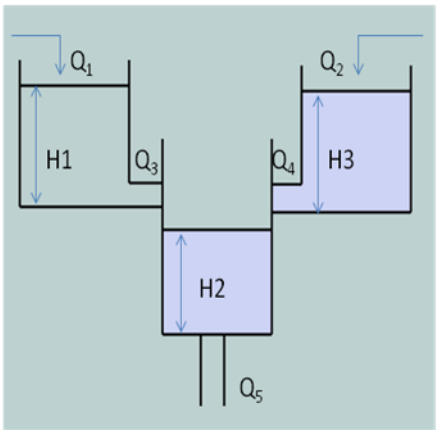
Figure 1: Single Tank with Its Mathematical Expression [1]



$$\frac{dH1}{dt} = \frac{Qin - k1\sqrt{H1}}{A1}$$

$$\frac{dH2}{dt} = \frac{k1\sqrt{H1} - k2\sqrt{H2}}{A2}$$

Figure 2: Split Tank with Its Mathematical Expression [1]



$$\frac{dH1}{dt} = \frac{Q1 - k1\sqrt{H1 - H2}}{A1} \qquad \frac{dH3}{dt} = \frac{Q2 - k3\sqrt{H3 - H2}}{A3} \qquad \frac{dH2}{dt} = \frac{-k2\sqrt{H2} + k1\sqrt{H1 - H2} + k3\sqrt{H3 - H2}}{A2}$$

Figure 3: Custom Tank with Its Mathematical Expression [1]

2.2 System Identification

A common and practically oriented approach to control system design is to use physical insights about the system in combination with practical closed-loop tests. However those simple approaches fail to work because performance demand is too strong and cannot be satisfied with simple design approaches. So more advances design methods must be considered. Generally these designs require knowledge about the system to be controlled and it should be described in terms of differential or difference equations. A mathematical description of this kind is called a *model* of the system. [5]

Basically there are two ways in which a model can be established:

- Derived in deductive manner using law of nature.
- It can be inferred (System Identification)

The first way can be simple but time consuming and it may even be considered unrealistic or impossible to obtain a sufficiently accurate model. The second method is commonly known as *System Identification*. [5]

One definition of *System Identification* is the process of constructing a model (mathematical description) of a dynamic system by means of measurement. [5] The identification has a different approach in which how we depend on the measured data in comparing to our insight about the internal structure of the system.

There are three types of mathematical model namely, *White-box, Gray-box, Black-box* modelling. However for its simplicity *Black-box modelling* has been used.

2.2.1 *White-box Modelling* [2]

Also known as phenomenological or mechanistic model. And it has the following characteristics:

- It is Based on:
  - Energy and material balances
  - Physical laws, constitutive relationships
  - Kinetic and thermodynamic models
  - Heat and mass transfer models
- Valid over wide operating range.
- Provide insight in the internal working of systems.
- Development and validation process: difficult and time consuming.

2.2.2 *Black-box Modelling* [2]

Has the following characteristic:

- Static maps (correlations)/ dynamic models (difference equations) developed directly from input-output data.
- Valid over limited operating range.
- Provide no insight into internal working of systems.
- Development and validation process: much less time consuming comparatively easy.

2.2.3 *Gray-box Modelling* [2]

Semi-Phenomenological model were part of model developed from the first principles and part developed from data. In some cases it is better choice than complete black box models.

2.2.4 *System identification procedure*

When attempting to construct a model for a dynamic system it is common to follow the procedure in figure 4.
Each step in the figure is explained in details in the following sections.

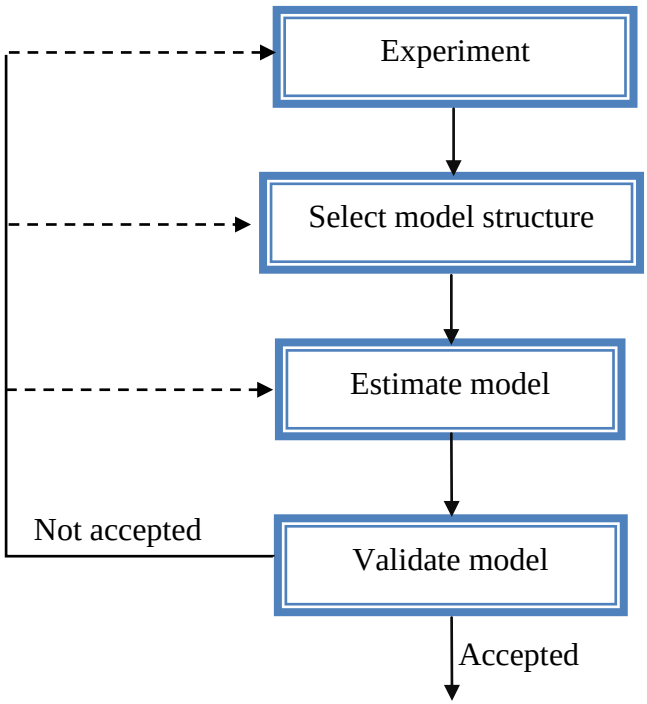


Figure 4: Basic System Identification Procedure [5]

2.2.4.1 Experiment [5]

The purpose of this part is to collect a set of data that describe how the system behaves over its entire range of operation. Basically the idea is to vary the input to the system and observe the corresponding outputs as shown in figure 5.

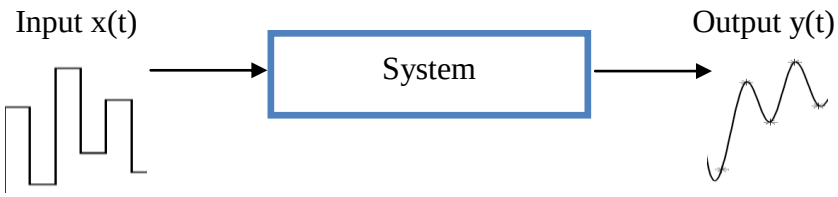


Figure 5: Input is applied to a system and output is observed

2.2.4.2 Select model structure [5]

To construct a model we must first choose from set of a family of model structure such as linear, multilayer perceptron networks or radial basis function. Then we have to choose a subfamily such as ARX (AutoRegressive eXternal input), OE (Output Error). [5] However we will be using ARX only throughout the project.

2.2.4.3 Estimate model [5]

Once a set of candidate models has been chosen, we choose one particular model form this set. The choice will be according to some type of criterion. The criterion can be formulated in many different was but it should relate to the intended use of the model. [5]

The most common way is to choose the model that provides the best one-step a head prediction which provides the smallest squared error between the observed and the estimated data. Another way is the sum of squared error between observed and estimated data which has been used in this project. [5]

2.2.4.4 Validation

After the model has been estimated it must be evaluated to investigate whether or not it meets the necessary requirement. The validation is connected to the intended use of the model. It is often the most hand-waved stage in the identification procedure because the acceptance standards are some time fussy. [5]

2.2.5 *ARX model [5]*

An AutoRegressive eXternal input or ARX model has the following form:

$y(t)= a_1y(t-1)+a_2y(t-2)+....+b_1x(t-1)+b_2y(t-2)+...$

Where

- y(t) is the output at time t, similarly y(t-1) is the output at time t-1 and so on.
- x(t) is the input at time t, similarly x(t-1) is the input at time t-1 and so on.
- a_1, a_2, b_1, b_2 called model parameter.

A term like ARX(2,2,1) signifies a time delay of one sampling period and the present output depend on two past outputs and two past inputs. [5]

2.3 Prediction Algorithms

2.3.1 *Recursive Least Square (RLS)* [2][7]

Real time or recursive identification algorithms are applied in tracking of time varying parameters, prediction and artificial neural networks. Some modifications need to be applied on the off-line algorithm so that can be implemented for the real time simulation for single tank, custom tank or other formation. The recursive least square identification is derived from the ordinary least square where

$$X_k = \begin{matrix} x_1^T \\ x_2^T \\ \dots \\ x_k^T \end{matrix} \qquad Y_k = \begin{matrix} y_1 \\ y_2 \\ \dots \\ y_k \end{matrix} \qquad \hat{\theta}_k = (X_k^T X_k)^{-1} X_k^T Y_k = \left[\sum_{i=1}^{k} x_i x_i^T\right]^{-1} \left[\sum_{i=1}^{k} x_i y_i\right]$$

The following is recursive least square algorithm that has been used for the simulation

$$\hat{\theta}_k = \hat{\theta}_{k-1} + P_k x_i \varepsilon_k \text{ and } P_k = (X_k^T X_k)^{-1} \text{ and } P_{k-1} = (X_{k-1}^T X_{k-1})^{-1}$$

$$\varepsilon_k = y_k - x_i^T \hat{\theta}_{k-1}$$

$$P_k = \frac{1}{\lambda}\left(P_{k-1} + \frac{P_{k-1} x_i x_i^T P_{k-1}}{\lambda + x_i^T P_{k-1} x_i}\right)$$

$\hat{\theta}_k$ is a vector consist of parameters to be estimated and ε_k is the prediction error and P_0 is given initial matrix usually an identity matrix. Since RLS estimation gives equal weighting to old and new input vectors, a forgetting factor λ ($0 < \lambda \leq 1$)is introduced in P_k matrix as above to suppress old input vector in order to track the parameter which is time varying in real time identification. [2][7]

2.3.2 *Recursive KALMAN Filter* [2][7]

Assume the time varying system parameter θ may be described as:

$$\theta_{k+1} = \theta_k + v_k \quad E\{v_i\} = 0 \quad and \quad E\{v_i v_j\} = R_1 \delta_{ij}$$

$$y_k = x_k^T \theta_k + e_k \quad E\{e_i\} = 0 \quad and \quad E\{e_i e_j\} = R_2 \delta_{ij}$$

$$E\{vv^T\} = R_1 \; and \; E\{ee^T\} = R_2$$

Recursive KALMAN Filter for estimation of θ and can be expressed as:

$$\hat{\theta}_k = \hat{\theta}_{k-1} + K_k \varepsilon_k$$

$$K_k = \left(\frac{P_{k-1} x_k}{R_2 + x_k^T P_{k-1} x_k} \right)$$

$$\varepsilon_k = y_k - x_i^T \hat{\theta}_{k-1}$$

$$P_k = \left(P_{k-1} - \frac{P_{k-1} x_i x_i^T P_{k-1}}{R_2 + x_i^T P_{k-1} x_i} + R_1 \right)$$

K_k is KALMAN Filter gain and the estimation parameter θ using KALMAN Filter is almost the same as Recursive Least Square (RLS) but there are two terms added (R1, R2) into the matrix P_k. The terms R1 and R2 added into P_k cause changes in the matrix property.

$$P_k \neq infinity \quad when \quad x_k = 0$$

$$P_k \neq 0 \quad when\, k \to infinity\, for \quad x_k \neq 0$$

2.3.3 *Extended KALMAN Filter* [2][7]

For Extended KALMAN Filter (EKF) we use the same formulas as in RKF however EKF is intended to be used for nonlinear estimation where the curves cannot be tracked by linear RKF. So basically EKF is an extension of RKF for nonlinear systems.

We exchange x_i and x_i^T with $\nabla y(w)$ (grad y) and $\nabla y^T(w)$ respectively where

$$\nabla y(w) = \frac{dy}{dw}\Big/_{\varepsilon}$$

AutoRegressive eXternal input Model order of [2 2 1] has been used to represent our output [5]

$$y(t) = a_1 y(t-1) + a_2 y(t-2) + b_1 x(t-1) + b_2 x(t-2)$$

Where the regressor vector $\theta = [a_1\ a_2\ b_1 b_2]$.[2][7]

2.4 Multilayer Perceptron Neural Network (MLP) [5]

For a given application it is difficult to say that one identification technique will outperform another before they have both been evaluated. It is desirable to consider one technique for all applications. [5]

Artificial neural networks have been the subject of much research in recent years in the field of nonlinear system identification due to their ability to learn complex nonlinear relationships from training examples.

They are characterised by a set of adjustable parameters which can be tuned using appropriate training procedures in order to obtain an input-output mapping that approximates the actual system. There are several forms of artificial neural network architecture, but the two that show immense practical interest and popularity are MLP and Radial Basis Function networks (Noorgard et. al. 2000). However we considered one type for the project which is MLP. Figure 4 show a typical structure of MLP.

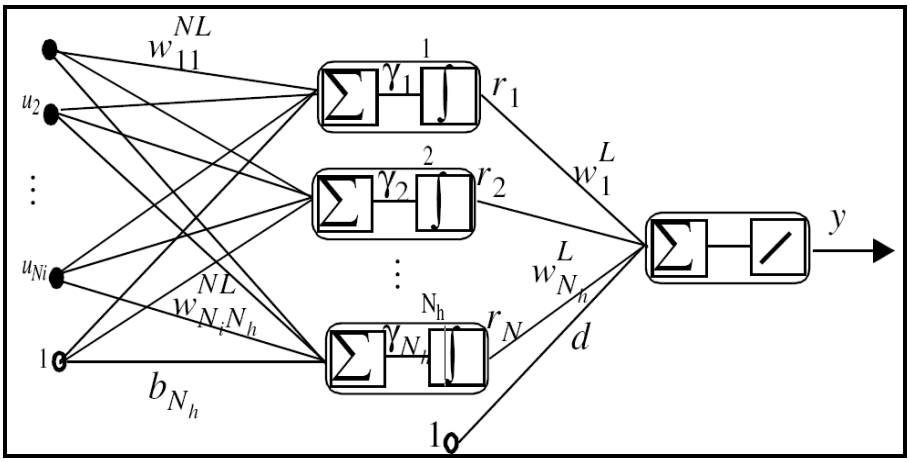


Figure 6: The structure of a MLP (Ni, Nh, 1) Network [6]

The MLP network is a simple structure consisting of layers of processing units, called neurons. These neurons are interconnected into a network by a set of weights (the arrows in figure 4) which are tuned using a chosen training algorithm.

Ni represents the input of the network.

Nh stands for the hidden neurons.

The hidden neurons can have different types of activation functions which pass their outputs to the next layer through the connected set of weights.[5][6] Example of an activation function is the nonlinear Sigmoid

$$f(v) = \frac{1}{1+e^{-v}}$$

And finally the output layer which is usually a linear summation of the inputs from the previous layer. The general output equation [6] is given as follow

$$y = \sum_{j=1}^{N_h} w_j^L f\left(\sum_{i=1}^{N_i} w_{ij}^{NL} u_i + b_j \right) + d$$

Where;

y is the network output

u_i is the i^{th} element of the network input vector u. The various weights which make up the overall weight vector w are:

W_j^L = The weight between the j^{th} neuron in the hidden layer and the linear (L) output neuron.

W_{ij}^{NL} = The weight between the i^{th} input and the j^{th} nonlinear hidden (NL) layer neuron.

b_j = The bias on the j^{th} hidden neuron.

d = The bias on the linear output neuron.[5][6]

2.5 Nonlinear Model Structure Based on Neural Networks [5]

When widening the focus to including black-box modelling identification of nonlinear dynamic system, the problem of selecting model structures becomes more difficult. In the previous section it was discussed that MLP networks is good at learning nonlinear relationships from a set of data. [5]

Therefore in choosing a family of model structures suitable for identification of nonlinear dynamic systems, it is natural to bring up MLP networks. [5] By choosing MLP the model structure selection is basically reduced to dealing with the following two issues:

- Selecting the input to the network.

- Selecting an internal network structure.

An often common approach is to reuse the input structures from the linear models while letting the internal architecture be feed-forward MLP network. This approach has several advantages:

- ✓ It is a natural extension of the well-known linear model structures.
- ✓ The internal architecture can be expanded gradually as a higher flexibility is needed to model more complex nonlinear relationships.
- ✓ The structural decision required by the user is reduced to a level that is reasonable to handle.
- ✓ Suitable for design of control systems.

Nonlinear model structure has the following general form:

$$y(t) = g[\varphi(t,\theta),\theta] + e(t)$$

Or on predictor form

$$\hat{y}(t|\theta) = g[\varphi(t,\theta),\theta]$$

Where y is the output

θ is the parameter vector which specify the relation between the output y and the input ψ.

2.5.1 *Neural Network ARX (NNARX)*

As for its linear counterpart, the predictor is always stable because there is a pure algebraic relationship between prediction and past measurements and inputs. This is important in the nonlinear case since the stability issue here is more complex than in linear system. [5] The model structure is described in figure 7.

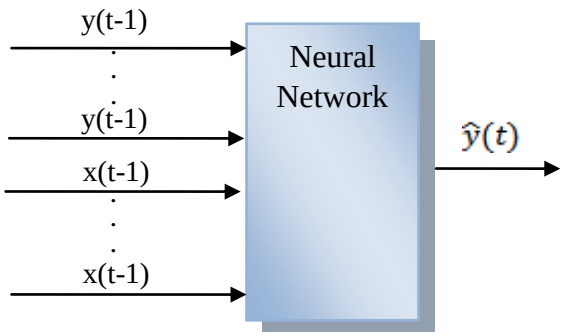


Figure 7: NNARX model structure

2.6 Particle Filter

Also known as Monte Carlo method is a nonparametric filter. The idea is that "if we cannot solve the integrals required for a Bayesian recursive filter analytically, we represent the posterior probabilities by a set of randomly chosen weighted samples" (Matthias, 2003, p.27). Hence increasing the number of sample will result in convergence to true probability density function (pdf) [3].

$$P(x_k|z_{1:k}) \approx \sum_{i=0}^{N_s} W_k^i \, \delta(x_{0:k} - x_{0:k}^i)$$

X_k is the state (sample) at time k; $z_{1:k}$ is all the measurements up to k; N_s is the number of samples; w_k is associated weights.[3]

Filtering techniques are wildly used in control system, in spacecraft navigation (KALMAN Filter), Robot Localization (Particle Filter) and in econometrics (stock markets, monetary flow) [3], [4].

For both techniques we utilize the formula after the dynamic system has been characterized by its state variables.

One way for finding the term grad y is analytically by finding the vector gradient of the MLP network cost function (squared error).

$$d(f(v)) = f(v)(1 - f(v))$$

Where $f(v) = \frac{1}{1+e^{-v}}$

Another approach is using Centered Finite Difference theorem where the derivative is calculated using

$$\frac{\partial y}{\partial w} = \frac{f(v+d) - (f(v) - d)}{2h}$$

Where h is a small value change (noise) around v. One approach to implement the particle filter concept is by using Hybrid EKF with Centered Finite Difference theorem. [8][9][10]

Where we initialize the weights and use EKF to propagate it through time t and then use Centered finite difference to add noise to it and propagate again.

CHAPTER 3
METHODOLOGY

3.1 Procedure identification flow

The following flow chart explains the methodology in executing the project:

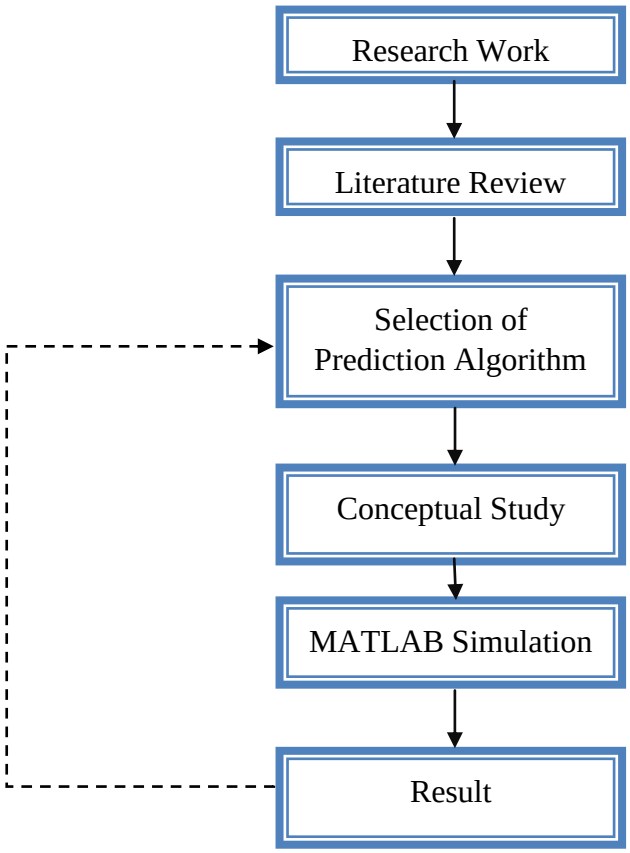


Figure 8: Methodology Flow Chart

3.2 Tools and Equipment

The main tool will be used in the project is MATLAB and MATLAB SIMULINK tool.

CHAPTER 4
RESULT AND DISCUSSION

4.1 Experiment

Below are the simulations results obtained from applying different input and observe the corresponding output using MATLAB SIMULINK software.

4.1.1 *Single Tank*

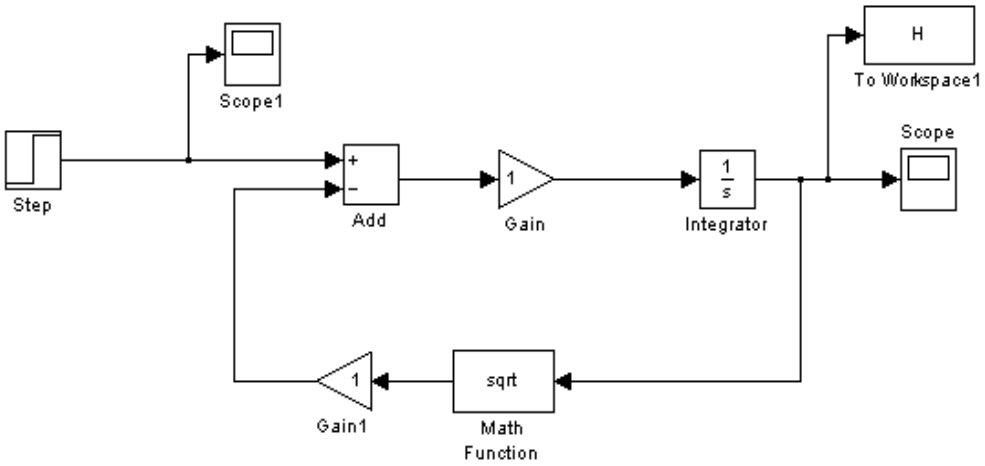


Figure 9: Single Tank SIMULINK Block Diagram

Figure 10 represent the applied input and figure 11 represent the output of the tank (height h).

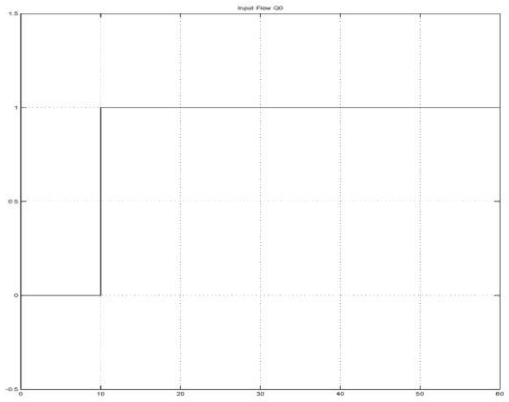


Figure 10: Step Input Pulse

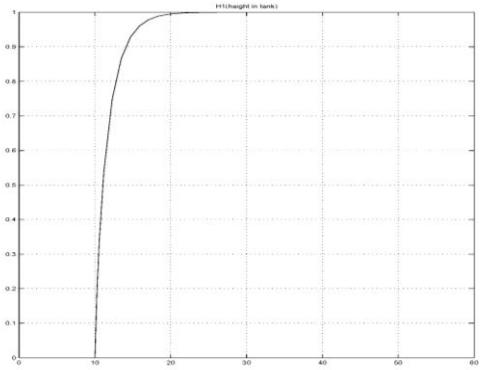


Figure 11: Single Tank Output waveform (height H)

4.1.2 *Split Tank*

For this tank the same step input used before was applied as an input. The SIMULINK block in figure 12 represent the split tank formation and the output behaviour is showed in figure 13.

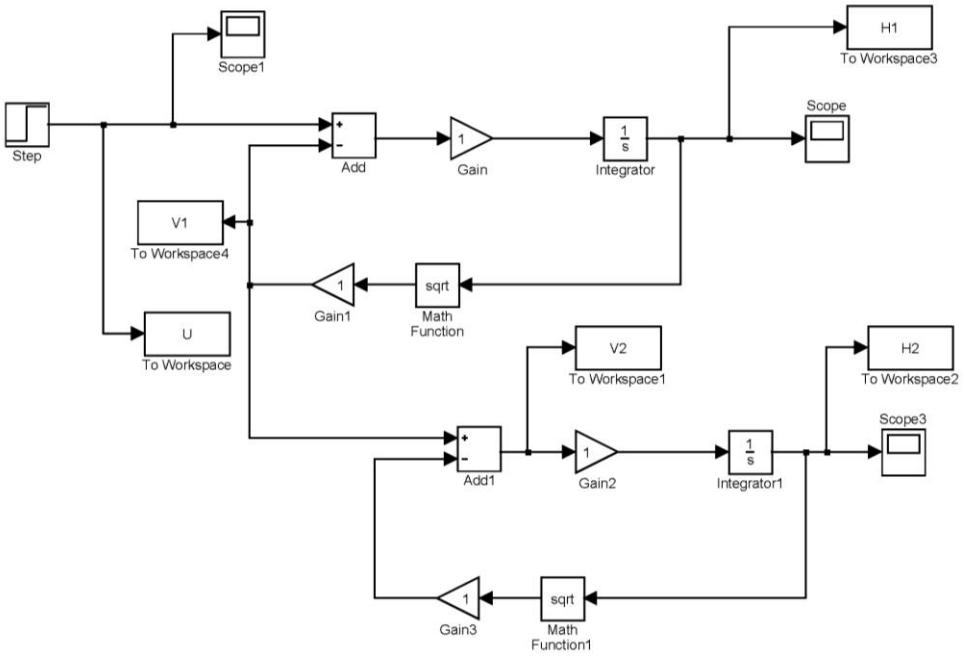


Figure 12: Split Tank SIMULINK Block Diagram

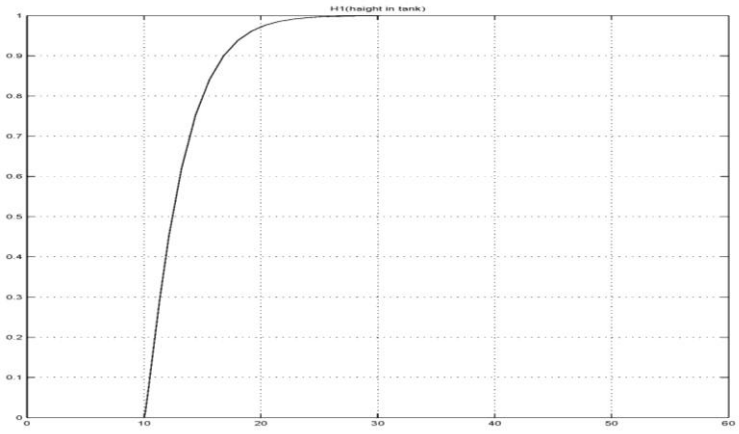


Figure 13: Split Tank Output waveform (height H2)

4.1.3 *Custom Tank*

For this tank there are two step inputs that have been used. One for each upper tank and the output flow for both is fed into the third tank as an input as shown previously in figure 3. Figure 14 illustrate the custom tank in SIMULINK and figure 15 shows the output.

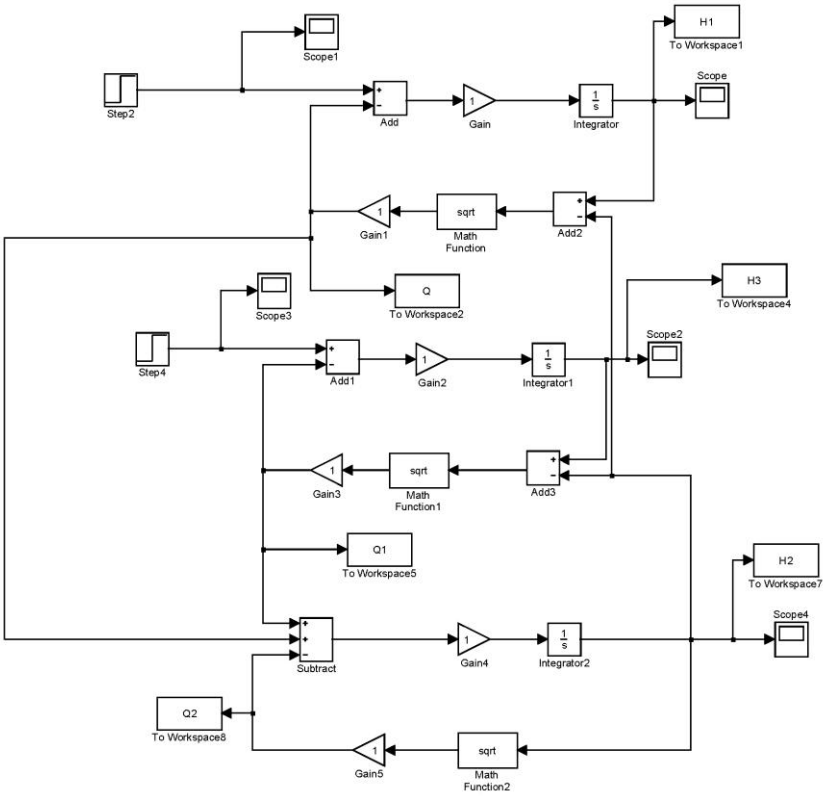


Figure 14: Custom Tank SIMULINK Block Diagram

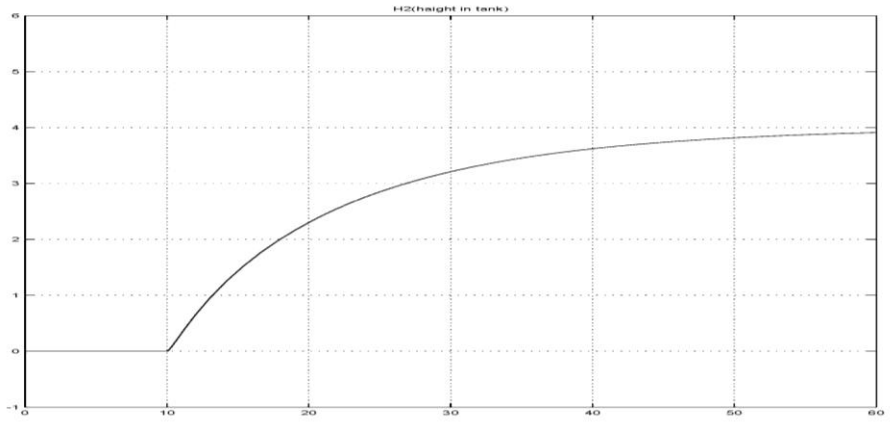


Figure 15: Custom Tank Output waveform (height H2)

4.2 Results for Various Input Signals

All the previous results were obtained by applying step input to the system. A pure sine wave and mixed of step and sine wave signals figure 16 and 17 respectively were

applied to each tank system and the output were observed.

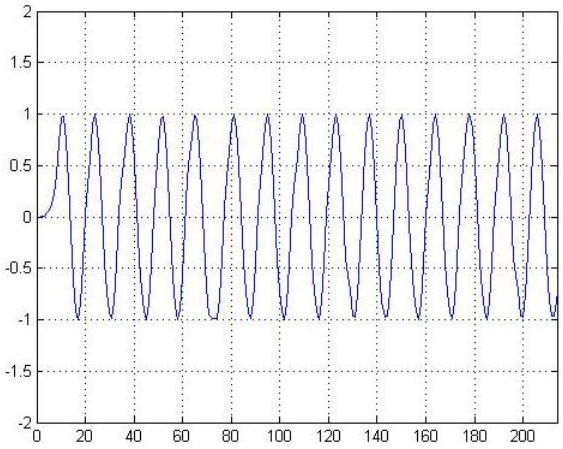


Figure 16: Sine Wave

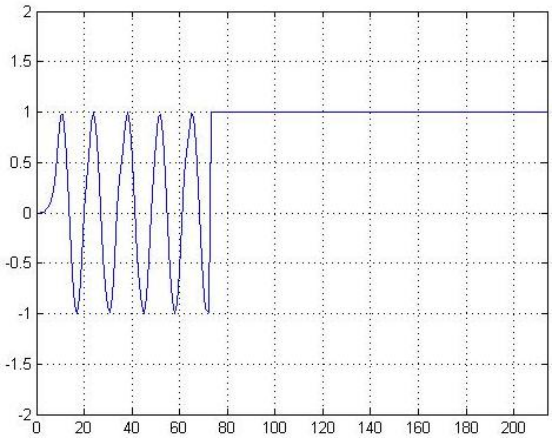


Figure 17: Mixed input (Step + Sine)

4.2.1 *Single Tank Model*

Figure 18 shows the output response for the single tank system when the input is a pure sine wave. We can notice the amplitude of the input has changed from -1 to +1 to around -0.56 to +0.56 in the output. Figure 19 shows the response for mix input signals. The simulation time was 100s with 0.5 as a max step size.

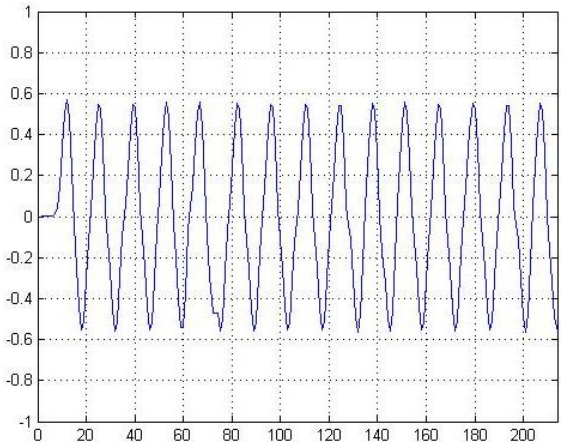


Figure 18: Single Tank Output Waveform (pure sine input)

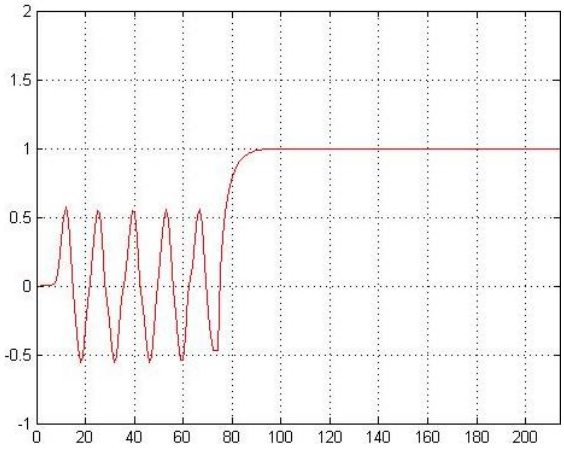


Figure 19: Single Tank Output Waveform (mixed input signal)

4.2.2 *Split Tank Model*

Figure 20 and 21 shows the output response for pure sine wave and mixed input signal respectively.

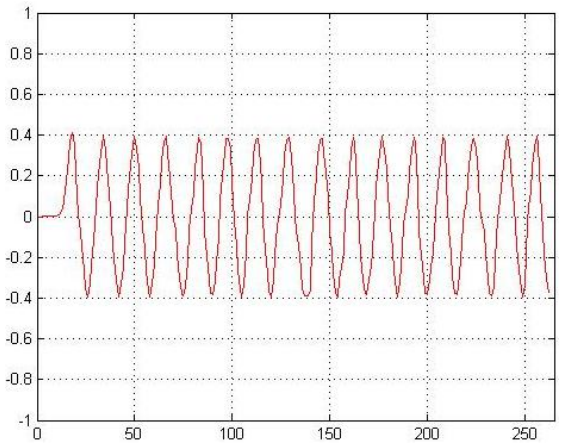


Figure 20: Split Tank Output Waveform (pure sine input)

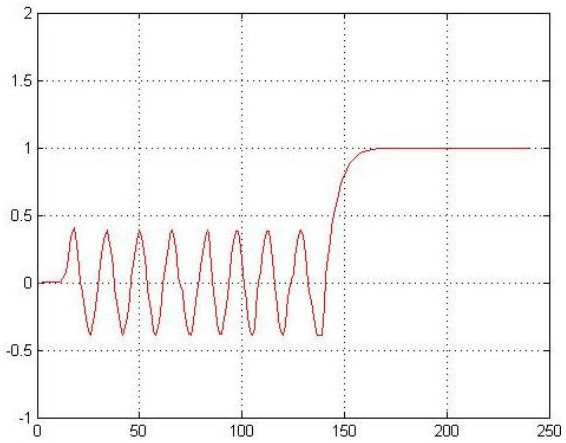


Figure 21: Split Tank Output Waveform (mixed input signal)

4.2.3 *Custom Tank Model*

Since the custom tank has two input, both types of signals pure sine and mixed wave were applied and the output response was observed for each signal as in figure 22 and 23.

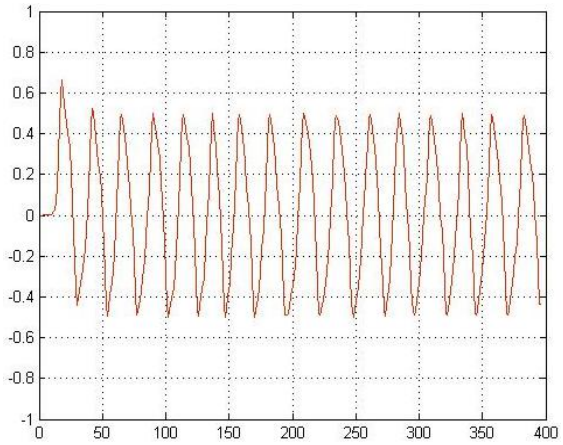


Figure 22: Custom Tank Output Waveform (pure sine input)

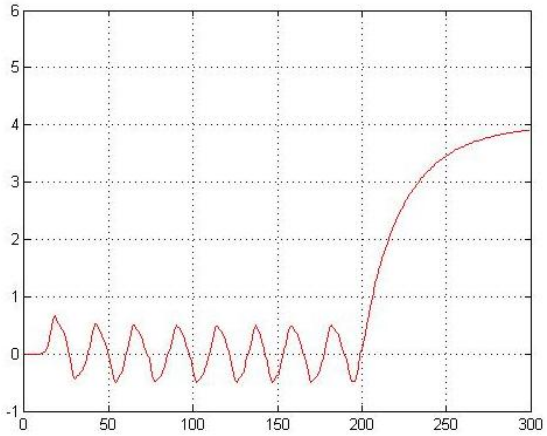


Figure 23: Custom Tank Output Waveform (mixed input signal)

4.3 Linear Prediction

In this part we have implemented different prediction algorithms on custom tank only. Two linear prediction algorithms have been used namely Recursive Least Square and Recursive KALMAN Filter. Also MLP with RPE and MLP with RKF have been tested on the custom tank as nonlinear prediction algorithms.

4.3.1 *Recursive Least Square Algorithm* [2]

An ARX (AutoRegressive eXternal input) Recursive Least Square algorithm with the following formulas has been used to predict the output (height H) of the custom tank model.

$$\hat{\theta}_k = \hat{\theta}_{k-1} + P_k x_i \varepsilon_k$$

$$\varepsilon_k = y_k - x_i^T \hat{\theta}_{k-1}$$

$$P_k = \frac{1}{\lambda}\left(P_{k-1} + \frac{P_{k-1} x_i x_i^T P_{k-1}}{\lambda + x_i^T P_{k-1} x_i}\right)$$

The algorithm has been implemented as an S-function block that has two inputs (input for the system x and the measured output y) as shown in figure 24. The block performs prediction for the height (H). With 100s simulation time and a step size of 0.5 the measured output verses the predicted output (Hp) was obtained as in figure 25. Figure 26 shows the squared error (SQE). Note that a first order ARX model ([1 1 1]) and a forgetting factor λ=0.9999 has been used to obtain the result.

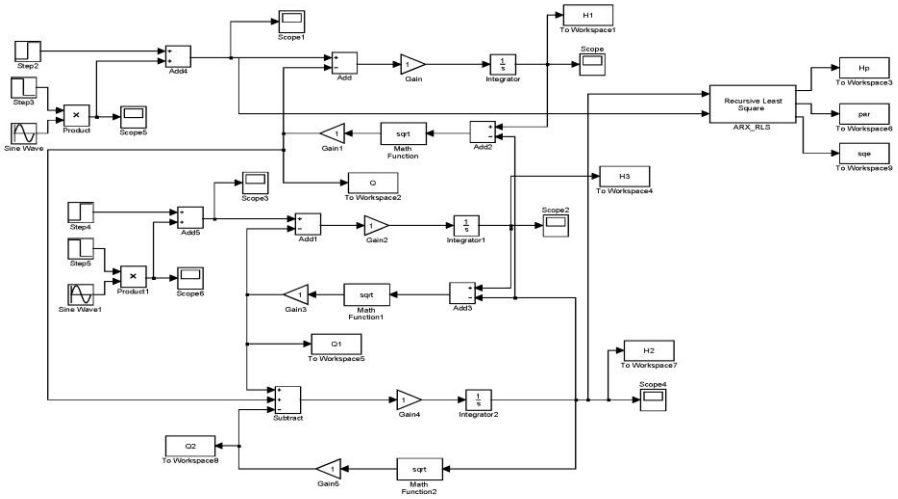


Figure 24: Custom Tank Block with RLS S-function

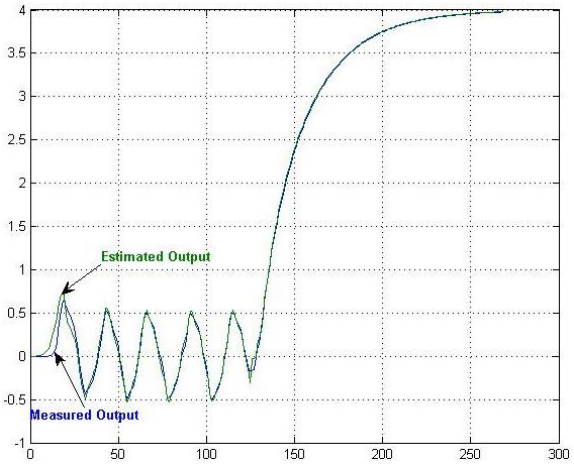


Figure 25: Measured Height Vs Predicted Height (RLS Algorithm)

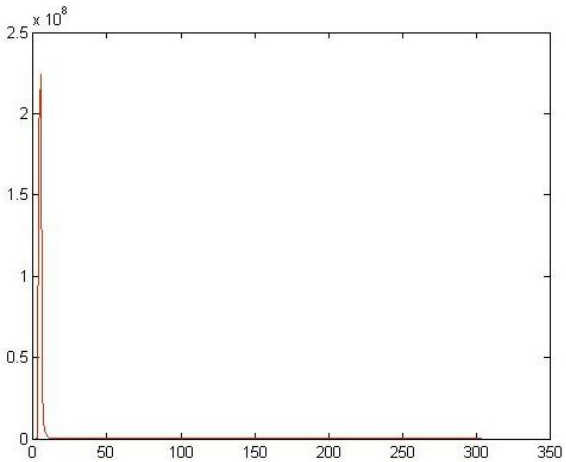


Figure 26: Custom Tank RLS Prediction Error

4.3.2 *Recursive KALMAN Filter Algorithm* [2]

An ARX Recursive KALMAN Filter Algorithm with the following formulas has been used to predict the output (height H) of the custom tank model.

$$\hat{\theta}_k = \hat{\theta}_{k-1} + K_k \varepsilon_k$$

$$K_k = \left(\frac{P_{k-1} x_k}{R_2 + x_k^T P_{k-1} x_k} \right)$$

$$\varepsilon_k = y_k - x_i^T \hat{\theta}_{k-1}$$

$$P_k = \left(P_{k-1} - \frac{P_{k-1} x_i x_i^T P_{k-1}}{R_2 + x_i^T P_{k-1} x_i} + R_1 \right)$$

The algorithm has been implemented as an S-function block that has two inputs (input for the system x and the measured output y) as shown in figure 27. The block performs prediction for the height (H). With 100s simulation time and a step size of 0.5. Note that a first order ARX model ([1 1 1]) and a forgetting factor λ=0.9999 has been used to obtain the result.

$$E\{vv^T\} = R_1 \; and \; E\{ee^T\} = R_2$$

R1 is the variance in the parameter and its being set is zero. R2 is the variance in the estimated output, it is calculated (R2= 3.1506) from off-line data and fed back again into the ARX recursive KALMAN S-function block (figure 27) and the result was obtained as in figure 28 and the squared error in figure 29.

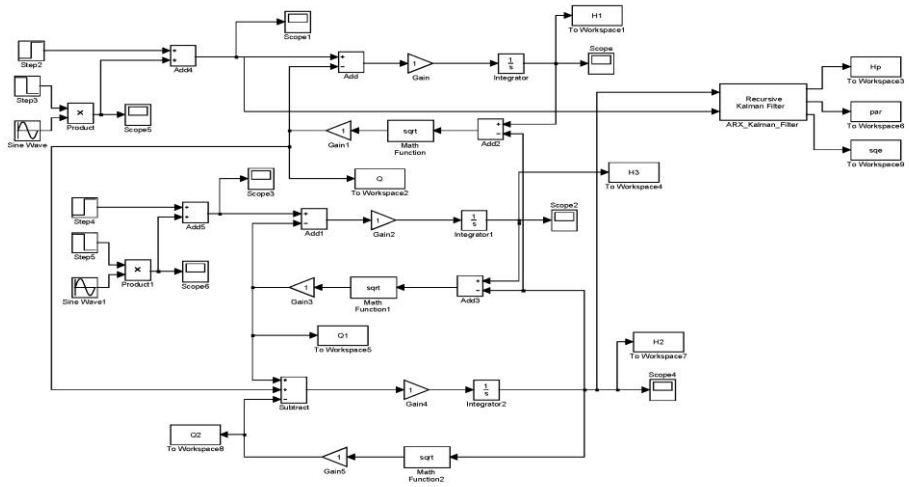


Figure 27: Custom Tank Block with RKF S-function

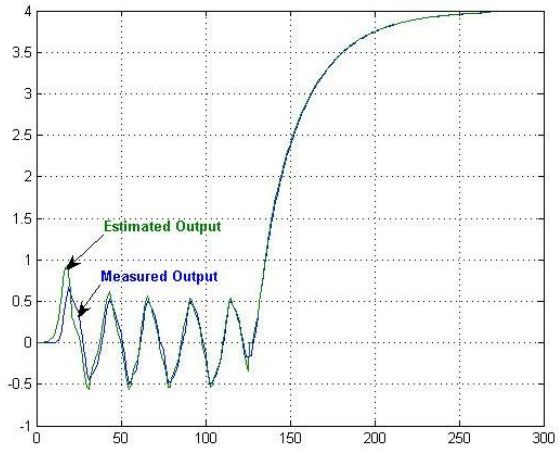


Figure 28: Measured Height Vs Predicted Height (KALMAN Algorithm)

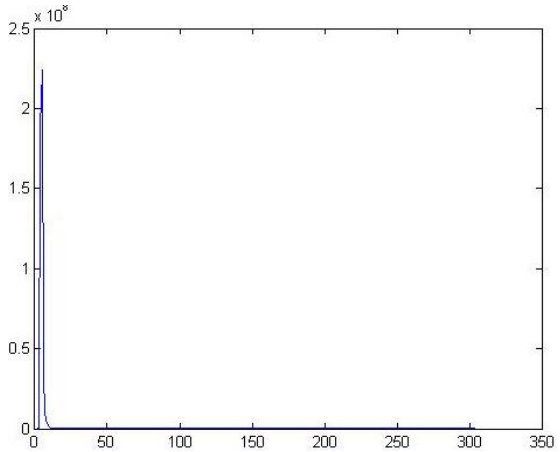


Figure 29: Custom Tank KALMAN Prediction Error

4.4 Nonlinear Prediction Based on MLP

4.4.1 *Recursive Prediction Error (RPE)*[7]

In this part we developed a MLP (4,3,1) neural network model to estimate the output height of the custom tank model in SIMULINK. Two training algorithms for the MLP have been used to develop an input-output mapping scheme, Recursive Prediction Error and RKF.

RPE is basically RLS for nonlinear system where instead of having x_i in the equations

$$\hat{\theta}_k = \hat{\theta}_{k-1} + P_k x_i \varepsilon_k$$

$$\varepsilon_k = y_k - x_i^T \hat{\theta}_{k-1}$$

$$P_k = \frac{1}{\lambda}\left(P_{k-1} + \frac{P_{k-1} x_i x_i^T P_{k-1}}{\lambda + x_i^T P_{k-1} x_i}\right)$$

We exchange x_i and x_i^T with $\nabla y(w)$ and $\nabla y^T(w)$ respectively where

$$\nabla y(w) = \frac{dy}{dw}\Big|_{\varepsilon}$$

AutoRegressive eXternal input Model order of [2 2 1] has been used to represent our output as [5]

$$y(t) = a_1 y(t-1) + a_2 y(t-2) + b_1 x(t-1) + b_2 x(t-2)$$

Where the regressor vector $\theta = [a_1\ a_2\ b_1 b_2]$. Figure 30 show the block diagram for the custom tank along with the MLP block with RPE training algorithm.

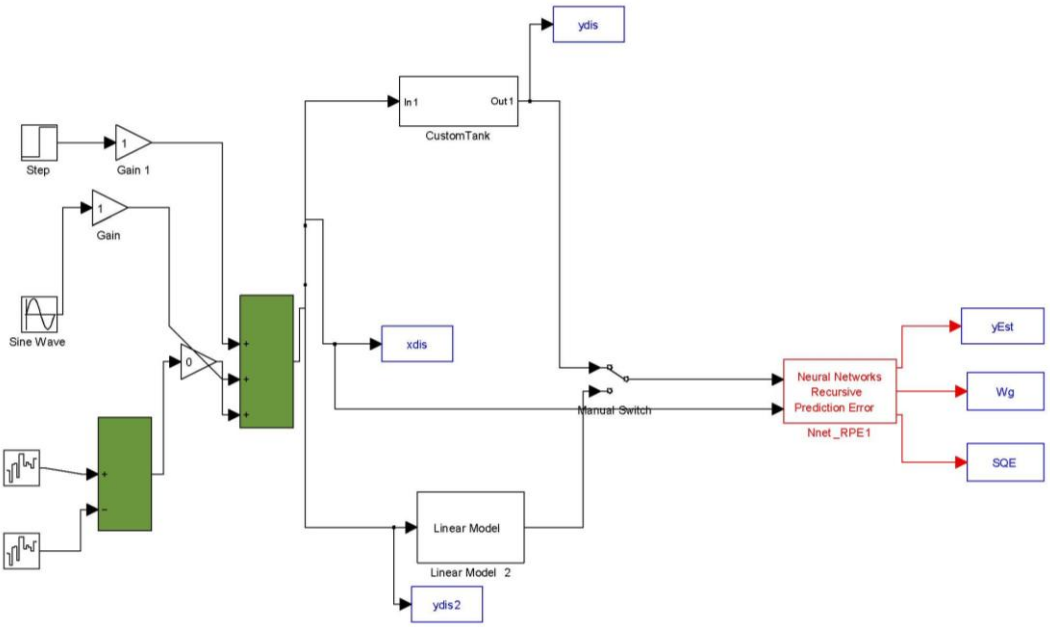


Figure 30: MLP [4 3 1] with RPE Training Algorithm

Setting the Learning Rate to 1, ARX [2 2 1], 3 hidden neurons and single output simulation result has been obtained as shown in figure 31.

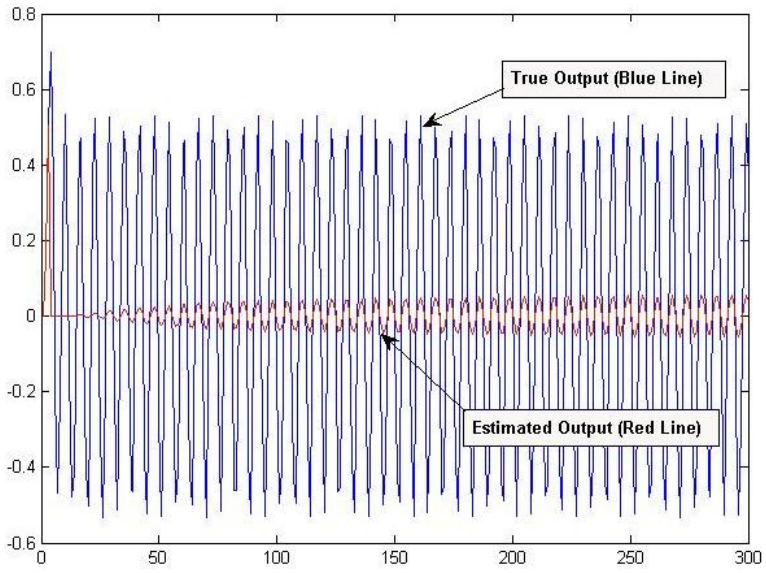


Figure 31: Custom Tank Simulation Result for RPE Training Algorithm

4.4.2 *MLP with RKF for the Linear Weights*

Referring to section 2.4 there are two sets of weights, nonlinear weights between the input and the hidden layer neuron and linear weight between the hidden neuron and the output layer neuron. In this section we used RKF to estimate the linear weights only while the nonlinear weights were fixed to the initialization value this process known as Extreme Machine Learning. The output of the activation functions act as an input to the RKF and the linear weights as the parameter vector $\widehat{\theta}$ where $\hat{Y}=input^{'}*\widehat{\theta}$ R2= 1.45 and R1= 0 figure 23 shows the block diagram of the custom tank with the MLP block. The simulation result obtained is shown in figure 33.

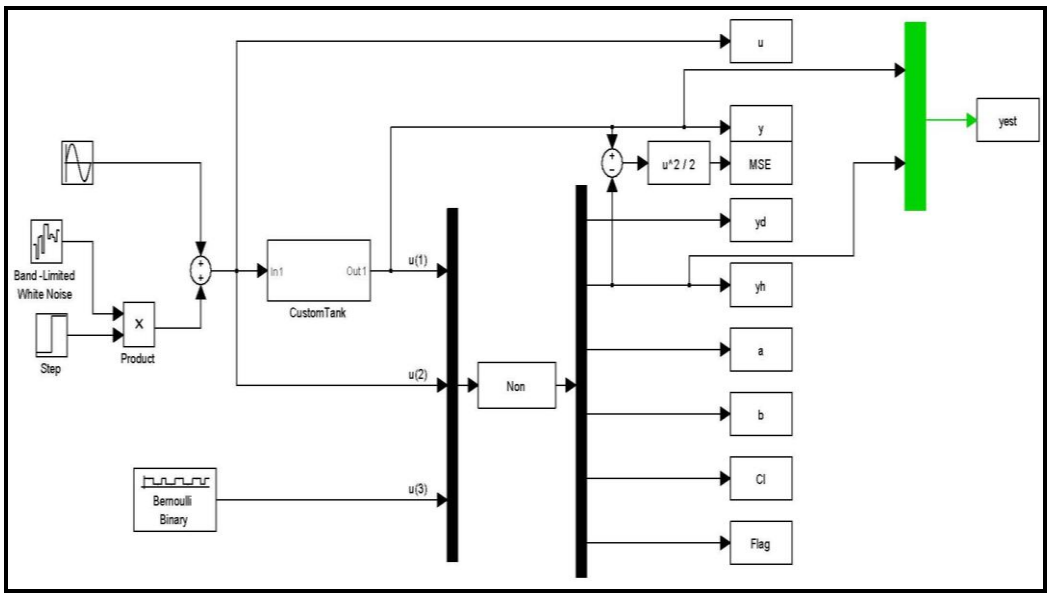


Figure 32: Custom Tank with MLP and RKF estimation Block

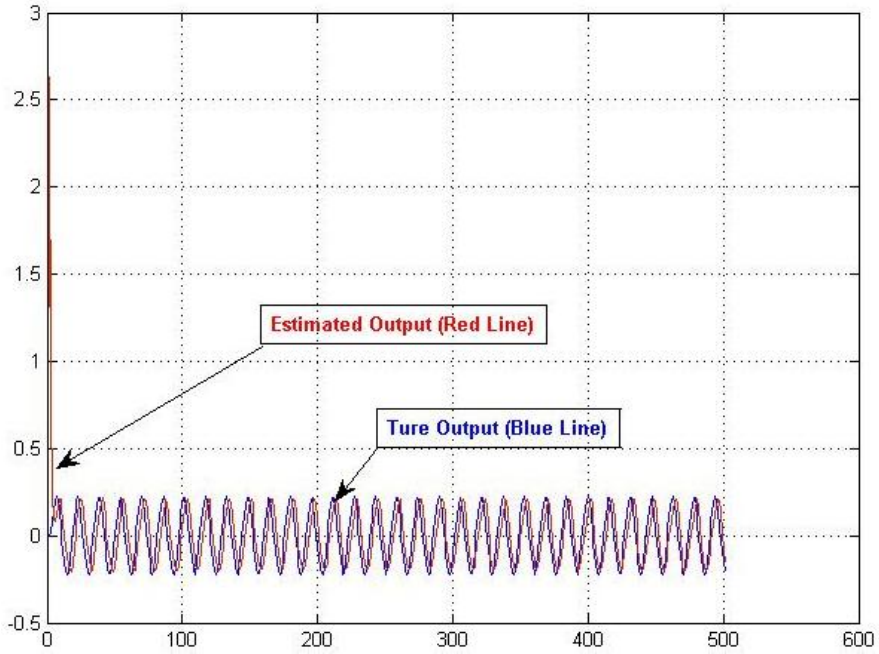


Figure 33: MLP with RKF estimation for Custom Tank

Figure 34 and 35 shows the error in the prediction for MLP with RPE and MLP with RKF training algorithm respectively.

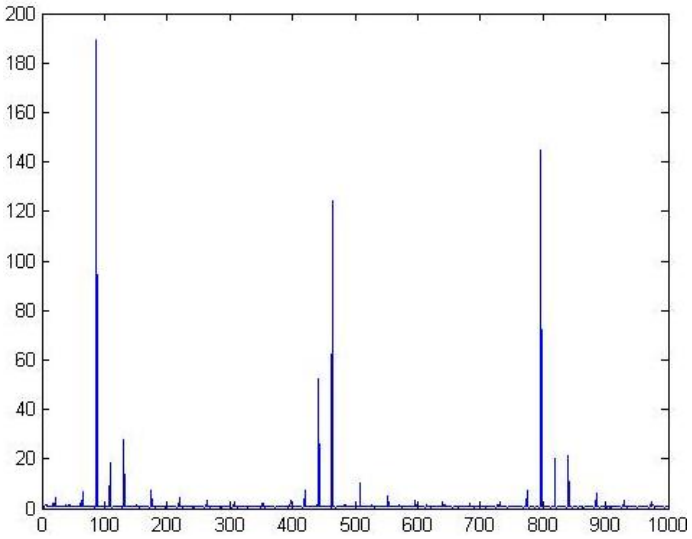


Figure 34: Error in MLP with RPE Training

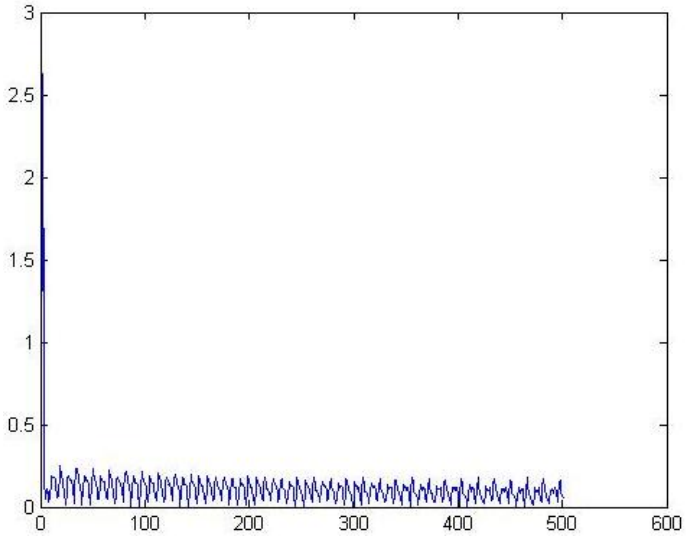


Figure 35: Error in MLP with RKF Training

4.4.3 *Hybrid Learning*

In this section EKF has been used for the nonlinear weights in the MLP architecture and RKF has been used for estimation of the linear weights. Combing both algorithms to find optimum prediction is called Hybrid learning. Figure 35 illustrate the prediction output and figure 36 for the squared error.

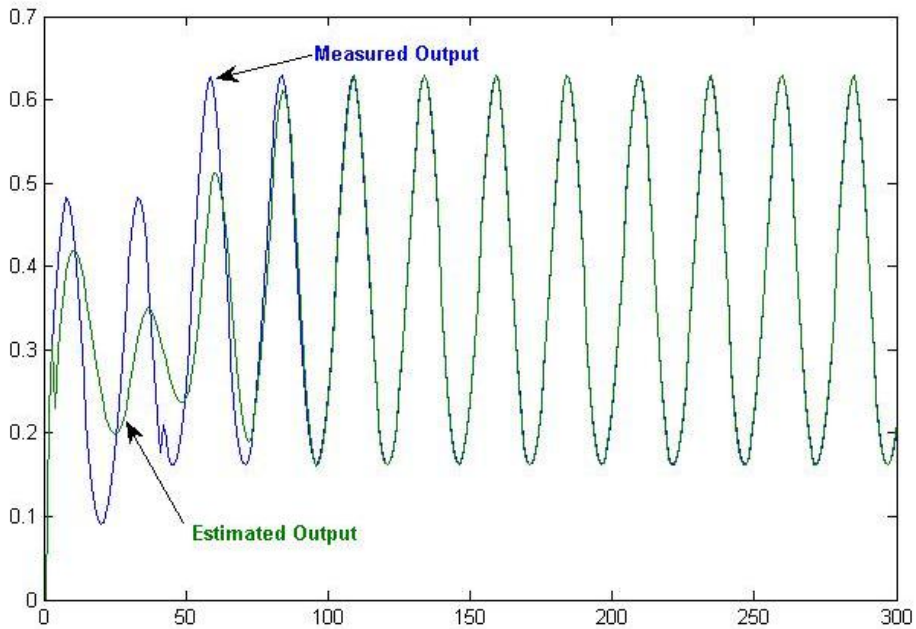


Figure 36: Estimated vs. Measured output using Hybrid Training

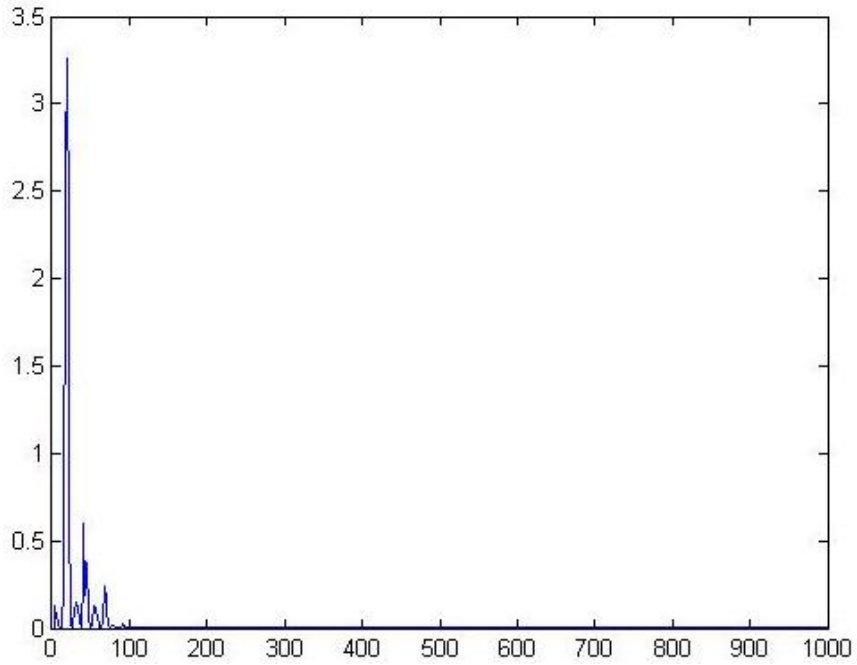


Figure 37: SQE for Hybrid Training

Figure 38 shows a comparison between MLP with hybrid training when grad y is calculated analytically and using centered finite difference. Figure 39 shows the advantage of using centered finite difference over the analytical approach where the estimation can follow up the true output around the curve points. Figure 40 shows estimation for the flow rate in the tank.

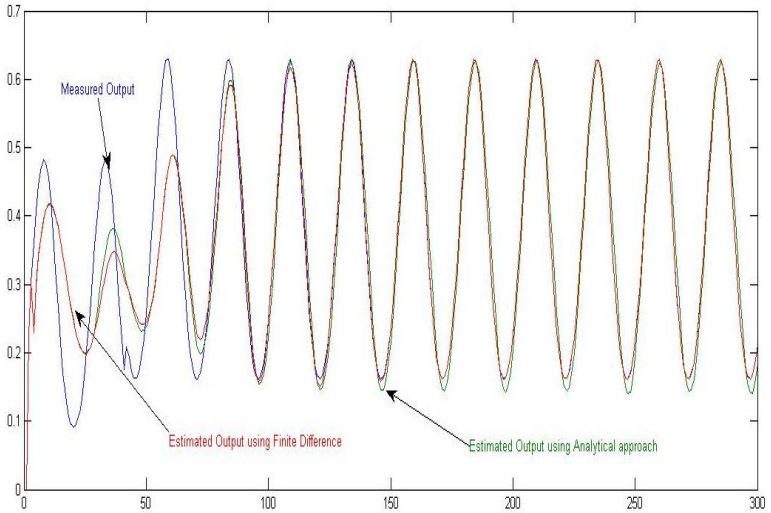


Figure 38: Comparison using Centered Finite Difference and Analytical Approach

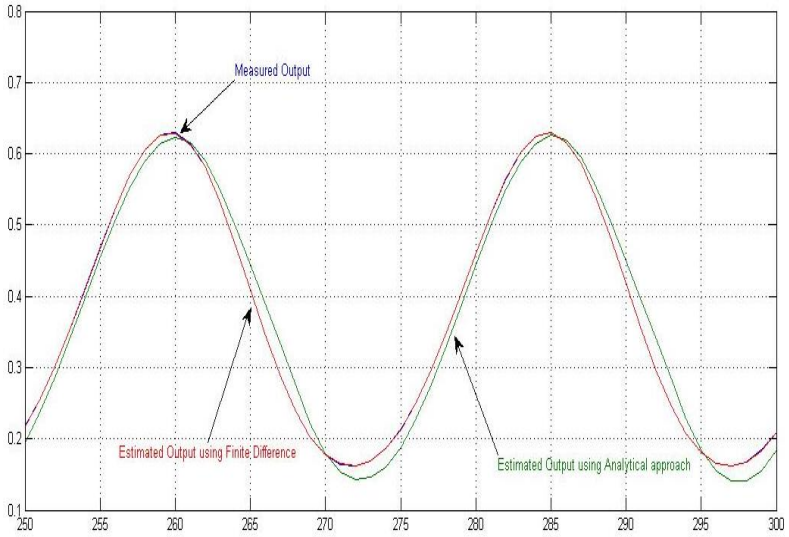


Figure 39: Improvment using Centered Finite Difference

For 5000 samples the analytical approach has 6.252×10^{-3}% mean percentage sum of squared error. Centered finite difference showed 4.518×10^{-3}% and MLP with extreme learning machine has 4.572×10^{-3}%.

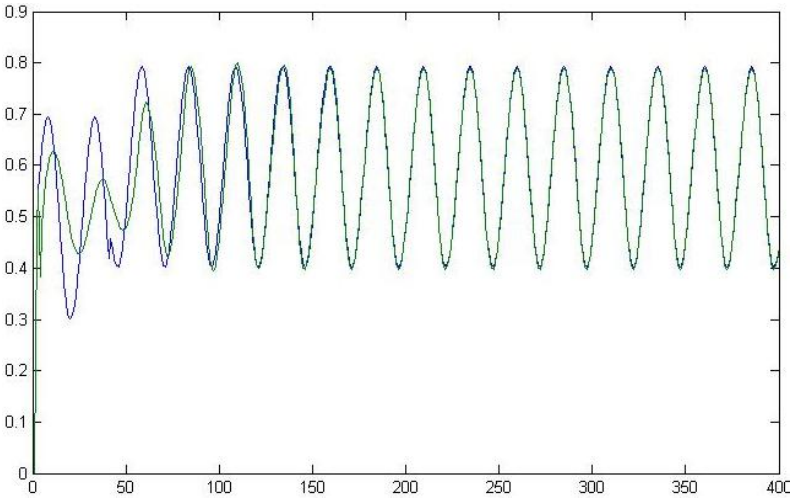


Figure 40: Flow Rate Estimation

4.5 Discussion

From the results obtained we observe the differences between RLS and Recursive KALMAN, where the overshoot in Recursive KALMAN algorithm is greater than RLS algorithm, the R2 value has an effect on the result. The value of the variance R2 has been set to 1.45 but it can also be calculated form offline data and inserted to the system. Furthermore the MLP with hybrid training has demonstrated better performance. From the graphs we notice that the MLP architecture require more a bit longer time to learn the function.

The evaluation of the different algorithms was based on the error between the estimated outputs and measured one. The flow rate has a similar wave form like the height of the fluid in the tank, so most of the results focus on the height. MLP with Hybrid training with cantered finite difference approach showed better perform in terms error and the ability to predict smooth changes (curves).

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

The dynamics of custom tank needs very good control strategy. Implementing a proper filtering or prediction technique will help in achieving optimal control for the system. Good mathematical analysis will lead at the end to a reliable simulation which can help in maximizing safety for the system.

On-Line System identification techniques are useful tools in tracking, filtering and smoothing dynamics systems nonlinear or linear ones. Different linear (RLS and RKF) and nonlinear (MLP with different training algorithms) have been tested.

Multilayer Perceptron has shown better performance especially with the new approach of cantered finite difference to find the derivative for the change in the output to the change in the network weights

In conclusion, it can be said that this project is able to be completed with the given time line frame to achieve the required objectives.

5.2 Recommendation

Through this project different linear and nonlinear techniques have been implemented for the system identification of the project. This work can be expanded to investigate on how to enhance those algorithms to produce a better estimation. Furthermore a practical implementation for the project within a university laboratory or with any host company will enhance the performance from any practical issues that might rise.

REFERENCES

[1] Michael, Alwin Prakash 2007, *system identification applied on various forms of control-tank system*, Bachelor of Electrical & Electronics Engineering dissertation, Universiti Teknologi Petronas, Malaysia.

[2] Ljung L., *System Identification: Theory for the user*, Englwood Cliff, NJ: Prentice-Hall, 1987.

[3] Dan Simon 2001, *kalman filtering*, Cleveland State University, USA.

[4] Matthias M¨uhlich 2003, *Particle filters an overview*, J.W.Goethe-University at Frankfurt, Germany.

[5] NØrgaard M., Ravn O., Poulsen N.K. and Hansen L.K., *Neural Networks for Modelling and Control of Dynamic Systems*, A Practitioner's Handbook.

[6] Asirvadam V.S., McLoone S.F., *Hybrid Recursive Training Algorithms using Fixed Size MLP-Network: A Survey*, Asian Institute of Medicine, Science and Technology (AIMST), National University of Ireland Maynooth.

[7] Haykin S., *Kalman Filtering and Neural Networks* McMaster University, Canada.

[8] Freitas JFG., Niranjan M., Gee A.H. and Doucet A., *Sequential Monte Carlo Methods for Optimization of Neural Network Models*, Cambridge University Engineering Department, UK

[9] Simandl M., Dunik J. 2009, *Derivative-free Estimation Methods: New Result and Performance Analysis*, Department of Cybernetics & Research Center Data-Algorithm-Decision Making, Faculty of Applied Sciences, University of West Bohemia, Czech Republic.

[10] Merwe R.V.D., Wan E.A., *Efficient Derivative-Free Kalman Filters for Online Learing*, Oregon Graduate Institute of Science and Technology, USA.

APPENDICES

APPENDIX A

Below are the m-files for the S-function blocks used for different estimation algorithms.

```
%*******RLS algorithm*******************
function [sys,x0,str,ts] = SfunRLS(t,x,u,flag,...
                                   samTime,Morder,cfac,Ts,erridx)

global count;
global st_y;
global st_x;
global lag;
global max_st;
global Ystore;
global Xstore;
global Inp;
global theta;
global Pmat;
global er;

switch flag,

case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(samTime,Morder);

    if (size(Morder,2) ==  3)
       st_y = Morder(1);
       st_x = Morder(2);
       lag =  Morder(3);
    else
       error('Wrong Morder row vector dimension ');
    end

    if ( (st_y < 0 | st_x <= 0) | lag <= 0)
         error('Vector element should be positive');
    end

    if ( cfac < 0.9 | cfac > 1 )
         error(' Large or incorrect forgeting factor input' );
    end

er = 0;
max_st = max(st_y,st_x);
Ystore = ones(1,(max_st+1)+(lag-1));
Xstore = ones(1,(max_st+1)+(lag-1));
Pmat = eye((st_y+st_x));
Inp = zeros((st_y+st_x),1);
theta = ones((st_y+st_x),1);

count =0;

case 2,

     sys = mdlUpdate(t,x,u,st_y,st_x,lag,max_st,cfac);

case 3,
    sys = mdlOutputs(t,x,u);
```

```
case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


function [sys,x0,str,ts]=mdlInitializeSizes(samTime,Morder)

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = Morder(1)+Morder(2)+3;
sizes.NumOutputs     = Morder(1)+Morder(2)+3;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0  = 0;
str = [];
ts  = [samTime  0]; % Sample period
function sys = mdlUpdate(t,x,u,st_y,st_x,lag,max_st,cfac);

global Xstore;
global Ystore;
global Pmat;
global Inp;
global theta;
global count;
global er;

Ystore(2:(max_st+1)+(lag-1)) = Ystore(1:(max_st)+(lag-1));
Xstore(2:(max_st+1)+(lag-1)) = Xstore(1:(max_st)+(lag-1));

if (count >= (max_st+1 + lag-1))

  Inp = [ Ystore((lag+1):(lag+1)+st_y-1)  Xstore((lag+1):(lag+1)
+st_x-1) ]';

  Pmat = (Pmat- (Pmat*Inp*Inp'*Pmat)/(cfac + Inp'*Pmat*Inp))/cfac;
  er = u(1) - Inp'*theta;
  theta = theta + Pmat*Inp*er;
  yEst = theta'*Inp;
  LSQE = win_SQE(theta,u(1),Inp);
else
   count = count +1;
   yEst = theta'*Inp;
   LSQE = 1;
end
Ystore(1) = u(1);
Xstore(1) = u(2);
sys = [theta ; u(1); yEst;LSQE];
function sys = mdlOutputs(t,x,u)
sys = x;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for Kalman filter replace the equation with:
Pmat_new = Pmat- (Pmat*Inp*Inp'*Pmat)/(var_est + Inp'*Pmat*Inp) +
var_par;
  er = u(1) - Inp'*theta;
  K = Pmat*Inp/(var_est + Inp'*Pmat*Inp);
```

```
  theta = theta + K*er;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%*******MLP with RKF for linear weights algorithm*******************

function [sys,x0,str,ts] = Non(t,x,u,flag,Ts)

%Variables declaration:
global ym;
global ym_1;
global xm;
global xm_1;
global w1;
global w2;
global yhat;
global yv;
global p;
global PmatL;
global PmatNL;
global Vphy;
global Vyd;
global prer;
global Pold;

switch flag,

case 0,
     [sys,x0,str,ts] = mdlInitializeSizes(Ts);
     ym = 0;
     ym_1 = 0;
     xm = 0;
     xm_1 = 0;
     Ni = 4; %No. of model inputs.
     Nh = 3; %No. of hidden layers.
     No = 1; %No. of model outputs.
     w1 = 2 * (1 - 0.5 * rand(Nh,Ni + 1)) * (3 / sqrt(Ni))
     w2 = (1 - 0.5 * rand(Nh + 1,No))

%*******intitialization of Kalman parameters*******************

     PmatNL = eye((Ni+1)* Nh);
     p = 10 * eye(Nh+1);
     yhat = 0;
     load Vset2nd;
     prer = zeros(Nh+1,1);

case 2,

    [sys] = mdlUpdate(t,x,u);

case 3,
    sys = mdlOutputs(t,x,u);
case 9,
    sys = [];
otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


function [sys,x0,str,ts]=mdlInitializeSizes(Ts)
sizes = simsizes;
```

```
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 6;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0  = 0;
str = [];
ts  = [Ts 0];

function [sys] = mdlUpdate(t,x,u);

%Variables declaration:
global ym;
global ym_1;
global ym_2;
global xm;
global xm_1;
global xm_2;
global phy;
global w1;
global w2;
global r;
global er;
global yhat;
global ni;
global nh;
global r;
global w1;
global w2;
global p;
global k;
global PmatNL;
global Vphy;
global Vyd;
global Ver;
global Vyh;
global prer;
global estW2;
global estY;
global er_Kalman;
global Pold;
%Regressor vector formation:
if(u(3)==1)
ym_1 = ym;
ym_2 = ym_1;
ym = u(1);
xm_1 = xm;
xm_2 = xm_1;
xm = u(2);
phy = [ym_1;ym_2;xm_1;xm_2]

[nc ni1] = size(w1);
[nc1 no] = size(w2);
ni = ni1-1;

[yhat,r]=mlpnet(w1,w2,phy);
er = ym-yhat;

fgfcL = 1;
```

```
%********Estimation of W2 using Recursive Kalman
Filter***************

[estW2 estY]=RKFforW2(w2,r,ym,t);

w2=estW2;
Fl = 0;
else
    Fl = 1
end
sys = [ym;yhat;u(1);estY;ym;Ver];

function sys = mdlOutputs(t,x,u)
sys = x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*******MLP with Hybrid EKF and RKF********************
function [sys,x0,str,ts] = SfunRPE1(t,x,u,flag,...
                                    samTime,Morder,Nh,Lr,tc)


global count;
global st_y;
global st_x;
global lag;
global max_st;
global Ystore;
global Xstore;
global Inp;
global W1;
global Tu;
global Ty;
global W2;
global Rgain;
global Pmat;
global prevdel_W;
global Weight;
global er;
global lamda;
global optionflag;
global PMSEflag;
global Pold;
global Pw2old;
global erw2;

switch flag,

case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(samTime,Morder,Nh);

    if (size(Morder,2) ==  3)
       st_y = Morder(1);
       st_x = Morder(2);
       lag =  Morder(3);
    else
       error('Wrong Morder row vector dimension ');
    end

    if ( (st_y < 0 | st_x < 0) | lag <= 0)
```

```
          error('Vector element should be positive');
    end

    if ( Nh <= 0  )
          error(' The  number of hidden neuron should be positive ' );
    end

er = 0;
erw2 = 0;
max_st = max(st_y,st_x);
Ystore = ones(1,(max_st+1)+(lag-1));
Xstore = ones(1,(max_st+1)+(lag-1));

[W1,W2,optionflag] = createMLP(st_y+st_x,Nh,1);

structMLP = [st_y+st_x Nh 1];
save structM structMLP;
Inp = zeros((st_y+st_x),1);
Rgain = [0.001 0.1];
lamda = 1; % Learning Rate
prevdel_W = zeros(1,(st_y+st_x+1)*Nh + Nh + 1)';
no_weight = (st_y+st_x+1)*Nh + Nh + 1;
%Pmat = eye(no_weight);
Pold = eye(((st_y+st_x+1)*Nh));
Pw2old = eye(Nh + 1);

[Tu,Ty] = loadDataNN(tc);
Weight = 0.001*rand(no_weight,1);

[nc ni1] = size(W1);
[nc1 no] = size(W2);
ni = ni1-1;

[W1,W2] = vect2mlp(Weight,ni,nc,no);

count =0;

case 2,

     sys = mdlUpdate(t,x,u,st_y,st_x,lag,max_st,Lr);

case 3,
    sys = mdlOutputs(t,x,u);

case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end


function [sys,x0,str,ts]=mdlInitializeSizes(samTime,Morder,Nh)

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = (Morder(1)+Morder(2)+1)*Nh + (Nh+1) +3;
sizes.NumOutputs     = (Morder(1)+Morder(2)+1)*Nh + (Nh+1) +3;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
```

```
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0  = 0;
str = [];
ts  = [samTime  0]; % Sample period

function sys = mdlUpdate(t,x,u,st_y,st_x,lag,max_st,Lr);

global Xstore;
global Ystore;
global Weight;
global Inp;
global W1;
global W2;
global Pmat;
global Tu;
global Ty;
global Rgain;
global prevdel_W;
global optionflag;
global count;
global er;
global lamda;
global PMSEflag;
global neta;
global weightflag;
global Pold;
global Pnew;
global R1;
global R2;
global er;
global K;
global KK;
global r;
global Pw2old;
global Pw2new;
global erw2;

Ystore(2:(max_st+1)+(lag-1)) = Ystore(1:(max_st)+(lag-1));
Xstore(2:(max_st+1)+(lag-1)) = Xstore(1:(max_st)+(lag-1));


if (count >= (max_st+1 + lag-1))


      Inp = [ Ystore((lag+1):(lag+1)+st_y-1)  Xstore((lag+1):(lag+1)
+st_x-1) ]';

     [nc ni1] = size(W1);
     [nc1 no] = size(W2);
     ni = ni1-1;

     grad = finiteWBgrad(W1,W2,optionflag,Inp',u(1)');%to use the
analytical approach replace finiteWBgrad with mlp_WinBgrad%%%%%%

     [dw1,dw2] = vect2mlp(grad,ni,nc,no);
     Mgrad = -grad;
```

```
      Weight = mlp2vect(W1,W2);
      prevdel_W = mlp2vect(dw1,dw2);

      [yEst,r] = mlpnet(W1,W2,optionflag,Inp');

      fgfc = 0.9;
      error = u(1) - yEst;
      gradY = dw1/(error + eps);
      gradY = gradY(:);
      no_w = size(Weight,1);

     %%%%%%%%%%%%%%%%%%%%%%%%EKF for W1%%%%%%%%%%%%%%%%%%%%%%%%%%

      R1=0;
      R2=1;

      Pnew = (1/fgfc)*(Pold - (Pold*gradY*gradY'*Pold)/(R2*fgfc +
gradY'*Pold*gradY) + R1);
      er = u(1) - yEst;
      K = Pold*gradY/(R2 + gradY'*Pold*gradY);

      W1=W1(:);
      W1 = W1 + K*er;

      Pold = Pnew;

      Weight = mlp2vect(W1,W2);

      [W1,W2] = vect2mlp(Weight,ni,nc,no);

     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      [yEst,r] = mlpnet(W1,W2,optionflag,Inp');
       er = u(1) - yEst;
     %%%%%%%%%%%%%%%%%%%%%%%%RKF for W2%%%%%%%%%%%%%%%%%%%%%%%%%%%

      r=r';
      Pw2new = (1/fgfc)*(Pw2old- (Pw2old*r*r'*Pw2old)/(R2*fgfc+
r'*Pw2old*r) + R1);

      KK = Pw2old*r/(R2 + r'*Pw2old*r);

      W2 = W2 + Lr*KK*er;
      Pw2old = Pw2new;

     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      PMSEflag =1;
      Weight = mlp2vect(W1,W2);
      [W1,W2] = vect2mlp(Weight,ni,nc,no);

      [yEst,r] = mlpnet(W1,W2,optionflag,Inp');

      SQEN = mlp_NSQE(W1,W2,optionflag,Inp',u(1));

else
   if (count == 0)
       weightflag = 0;
       neta = Lr;
   end
```

```
   SQEN = 0;
   yEst = u(1);
end

 count = count +1;
 Weight = mlp2vect(W1,W2);
 Ystore(1) = u(1);
 Xstore(1) = u(2);

 sys = [Weight ; u(1); yEst; SQEN];

function sys = mdlOutputs(t,x,u)
sys = x;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%RPE algorithm%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sys,x0,str,ts] = SfunRPE1(t,x,u,flag,...
                                    samTime,Morder,Nh,Lr,tc)

global count;
global st_y;
global st_x;
global lag;
global max_st;
global Ystore;
global Xstore;
global Inp;
global W1;
global Tu;
global Ty;
global W2;
global Rgain;
global Pmat;
global prevdel_W;
global Weight;
global er;
global lamda;
global optionflag;
global PMSEflag;

switch flag,

case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(samTime,Morder,Nh);

    if (size(Morder,2) ==  3)
       st_y = Morder(1);
       st_x = Morder(2);
       lag =  Morder(3);
    else
       error('Wrong Morder row vector dimension ');
    end

    if ( (st_y < 0 | st_x < 0) | lag <= 0)
         error('Vector element should be positive');
    end

    if ( Nh <= 0  )
         error(' The  number of hidden neuron should be positive ' );
```

```
    end
er = 0;
max_st = max(st_y,st_x);
Ystore = ones(1,(max_st+1)+(lag-1));
Xstore = ones(1,(max_st+1)+(lag-1));

[W1,W2,optionflag] = createMLP(st_y+st_x,Nh,1);

structMLP = [st_y+st_x Nh 1];
save structM structMLP;
Inp = zeros((st_y+st_x),1);
Rgain = [0.001 0.1];
lamda = 1; % Learning Rate
prevdel_W = zeros(1,(st_y+st_x+1)*Nh + Nh + 1)';
no_weight = (st_y+st_x+1)*Nh + Nh + 1;
Pmat = eye(no_weight);

[Tu,Ty] = loadDataNN(tc);
Weight = 0.001*rand(no_weight,1);

[nc ni1] = size(W1);
[nc1 no] = size(W2);
ni = ni1-1;

[W1,W2] = vect2mlp(Weight,ni,nc,no);

count =0;

case 2,

     sys = mdlUpdate(t,x,u,st_y,st_x,lag,max_st,Lr);

case 3,
    sys = mdlOutputs(t,x,u);

case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes(samTime,Morder,Nh)

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = (Morder(1)+Morder(2)+1)*Nh + (Nh+1) +3;
sizes.NumOutputs     = (Morder(1)+Morder(2)+1)*Nh + (Nh+1) +3;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0  = 0;
str = [];
ts  = [samTime  0]; % Sample period


function sys = mdlUpdate(t,x,u,st_y,st_x,lag,max_st,Lr);
```

```
global Xstore;
global Ystore;
global Weight;
global Inp;
global W1;
global W2;
global Pmat;
global Tu;
global Ty;
global Rgain;
global  prevdel_W;
global optionflag;
global count;
global er;
global lamda;
global PMSEflag;
global neta;
global weightflag;

Ystore(2:(max_st+1)+(lag-1)) = Ystore(1:(max_st)+(lag-1));
Xstore(2:(max_st+1)+(lag-1)) = Xstore(1:(max_st)+(lag-1));


if (count >= (max_st+1 + lag-1))

      Inp = [ Ystore((lag+1):(lag+1)+st_y-1)  Xstore((lag+1):(lag+1)
+st_x-1) ]';

     [nc ni1] = size(W1);
     [nc1 no] = size(W2);
     ni = ni1-1;

     grad = mlp_WinBgrad(W1,W2,optionflag,Inp',u(1)');

     Mgrad = -grad;

     [dw1, dw2] = vect2mlp(Mgrad,ni,nc,no);

     Momen = 0.0;
     neta = Lr;

     [del_w1,del_w2] = vect2mlp(prevdel_W,ni,nc,no);

     chgw1 = neta*dw1 + Momen*del_w1;
     chgw2 = neta*dw2 + Momen*del_w2;
     Weight = mlp2vect(W1,W2);
     prevdel_W = mlp2vect(chgw1,chgw2);

     yEst = mlpnet(W1,W2,optionflag,Inp');
     beta = 0.01;
     n_lamda = min(beta,1/(count^(1-beta)));
     fgfc = lamda*(1-n_lamda)/n_lamda;
     lamda = n_lamda;
     %fgfc = 0.99*lamda + (1-0.99);
     %lamda = fgfc;
     fgfc = 0.999;
     error = u(1) - yEst;
     gradY = grad/(error + eps);
     no_w = size(Weight,1);
```

```
      denom = inv(fgfc + gradY'*Pmat*gradY);
      Pmat = (Pmat - (Pmat*gradY*denom*gradY'*Pmat))/fgfc;
      Pmat = (1/trace(Pmat))*Pmat;

      Weight = Weight + Pmat*prevdel_W;
      PMSEflag =1;
      [W1,W2] = vect2mlp(Weight,ni,nc,no);

      yEst = mlpnet(W1,W2,optionflag,Inp');

      SQEN = mlp_NSQE(W1,W2,optionflag,Inp',u(1));

else
   if (count == 0)
       weightflag = 0;
       neta = Lr;
   end

   SQEN = 0;
   yEst = u(1);
end

 count = count +1;
 Weight = mlp2vect(W1,W2);
 Ystore(1) = u(1);
 Xstore(1) = u(2);
 sys = [Weight ; u(1); yEst; SQEN];

function sys = mdlOutputs(t,x,u)
sys = x;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

APPENDIX B

Below are common m-files used by all S-functions.

```
% function [w1 ,w2 ,flags]=createMLP(Ni,Nh,No,method)
%
% Initialises  MLP given dimensions
% Sigmoid nonlinearities and linear output neurons assumed
%
% See also saveMLP and loadMLP.

function [w1 ,w2 ,flags]=createMLP(Ni,Nh,No,method);

   fprintf('Creating MLP network ... ');

   if nargin <3
      error('Invalid number of input parameters');
   end

   if nargin >4
      error('Invalid number of input parameters');
   end
   if nargin ==3
    method =1;
   end
   flags =[1 1 0]; % sigmoid + linear output
   if method ==1
    w1=2*(1-2*rand(Nh,Ni+1))*(3/sqrt(Ni));
    w2=(1-2*rand(Nh+1,No));
   end
  if method ==2
    w1=2*(1-2*rand(Nh,Ni+1))/Ni;
    w2=(1-2*rand(Nh+1,No));
    sc=0.7*4*Nh^(1/Ni);
    for i =1:Nh
        w1(i,1:Ni)=w1(i,1:Ni)/sqrt(w1(i,1:Ni)*w1(i,1:Ni)')*sc;
    end
    w1(:,Ni+1)=w1(:,Ni+1)*sc;
   end
  if method >2
    w1=2*(1-2*rand(Nh,Ni+1))/Ni;
    w2=(1-2*rand(Nh+1,No));
    for i =1:Nh
        w1(i,:)=4*w1(i,:)/(Ni+1);
    end
   end
   fprintf('MLP(%d,%d,%d) initialised\n',size(w1,2)-1,
size(w1,1),size(w2,2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Out]= extractTheta(Vect)
array_size = size(Vect,1);
Out = [Vect(1:array_size-3)];
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Out]= extractSQE(Vect)
array_size = size(Vect,1);
Out =  [Vect(array_size)];
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Out]= extractY(Vect)
```

```
array_size = size(Vect,1);
Out = [Vect(array_size-2) Vect(array_size-1)];
Return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Out]= extractWeight(Vect)
array_size = size(Vect,1);
Out = [Vect(1:array_size-3)];
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function [grad] = finiteWBgrad(w1,w2,flags,u,yd)

%The function return the vector gradient using finite difference
method of the
%MLP network for SSE cost function
function [grad] = finiteWBgrad(w1,w2,flags,u,yd,block)

   [nh ni1]=size(w1);
   [nh1 no]=size(w2); %Implemented only for single output

   CenDiff_w1 = eye(nh,ni1);
   CenDiff_w2 = eye(nh1,no);
   percentV = 0.001;


        for i=1:nh
          for j=1:ni1

            w1bw = w1;
            w1fw = w1;

            Dx = percentV*w1bw(i,j);
            w1bw(i,j) = w1(i,j) - Dx;
            w1fw(i,j) = w1(i,j) + Dx;

            sqeb_w1= mlp_SQE(w1bw,w2,flags,u,yd);
            sqef_w1= mlp_SQE(w1fw,w2,flags,u,yd);

            % Using Central Difference in finite difference

            CenDiff_w1(i,j)= (sqef_w1 - sqeb_w1)/(2*Dx);

          end
         end


    grad=mlp2vect(CenDiff_w1,w2);
 return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [grad] = mlp_WinBgrad(w1,w2,flags,TSu,TSy)

% the function returns the vector gradient of the MLP network
% cost function for SSE cost function.

function [grad] = mlp_WinBgrad(w1,w2,flags,u,yd,block)

   global PMSEflag;
```

```
   [nh ni1]=size(w1);
   [nv ni]=size(u);
   [nh1 no]=size(w2);

   grad_w1=zeros(nh,ni1);
   grad_w2=zeros(nh1,no);
   uni1 = ones(nv,1);

     [y,r]=mlpnet(w1,w2,flags,u);

     err=y-yd;          % (nv *  no) matrix
    % NB: Using PMSE gradient scaling factor leads to rounding errors
    if ~isempty(PMSEflag)
      if PMSEflag ==1
         err=err/length(err(:))*100;% normalized if % Mean squared
error used;
       end
    end

     r=r';           % Transposed for use below (nh1 *nv) matrix
     grad_w2= 2*r*err;     % nh1 *no matrix

     %compute factors independent of Nv

    %For sig function dr = r(1-r)

    r=r.*(1-r);
    r=r(2:nh1,:);       % NB r is now dr and is a (nh *nv) matrix
....
    u1 = [uni1 u];      % Add one column for  for bias weights
    P=w2(2:nh1,:)*err'; % (nh * nv) matrix

    grad_w1= (P.*r)*u1;

    grad=mlp2vect(grad_w1,grad_w2);
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [lsqe] = win_SQE(the,Yv,IN)

smallvalue = eps;
yestimate = the'*IN;
NErr = (Yv - yestimate)*(Yv - yestimate)';
Derr = Yv*Yv';
lsqe = NErr/(Derr + smallvalue) * 100;

return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [v]=mlp2vect(w1,w2)

   v=[w1(:);w2(:)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  [w1,w2]=vect2mlp(vect,ni,nh,no)
%   Transforms the mlp parameters stored in vector form
%   to the standard w1,w2 form.
function [w1,w2]=vect2mlp(v,ni,nh,no)

   lv=length(v);

   if lv ~= (ni+1)*nh +(nh+1)*no;
```

```
      error('input vector dimension does not agree with MLP
dimensions');
   end

   w1=zeros(nh,ni+1);
   w2=zeros(nh+1,no);

   % convert vector to matrix form for w1 and w2;
   for i=1:(ni+1)
      w1(:,i)=v( 1+(i-1)*nh : i*nh);
   end
   v=v((ni+1)*nh+1:lv);

   for i=1:no
     w2(:,i)=v( 1+(i-1)*(nh+1) : i*(nh+1));
   end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [estW2 estY]=RKFforW2(w2,r,ym,t)

global Pold;
global Pnew;
global Inp;
global theta;
global R1;
global R2;
global er;
global k;
global yEst;
global t;

if(t==0)
    Pold = eye(size(w2,1));
    er=0;
end
%**********Initilization of Kalman Parameters***************

theta=w2;
Inp=r;
R1=0;
R2=1.45;
%*****************End of Initilization*********************

  Pnew = Pold- (Pold*Inp*Inp'*Pold)/(R2+ Inp'*Pold*Inp) + R1;
  er = ym - Inp'*theta;
  K = Pold*Inp/(R2 + Inp'*Pold*Inp);

  theta = theta + K*er;
  Pold = Pnew;
  yEst = Inp'*theta;
  %LSQE = win_SQE(theta,u(1),Inp);
estW2=theta;
estY=yEst;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```