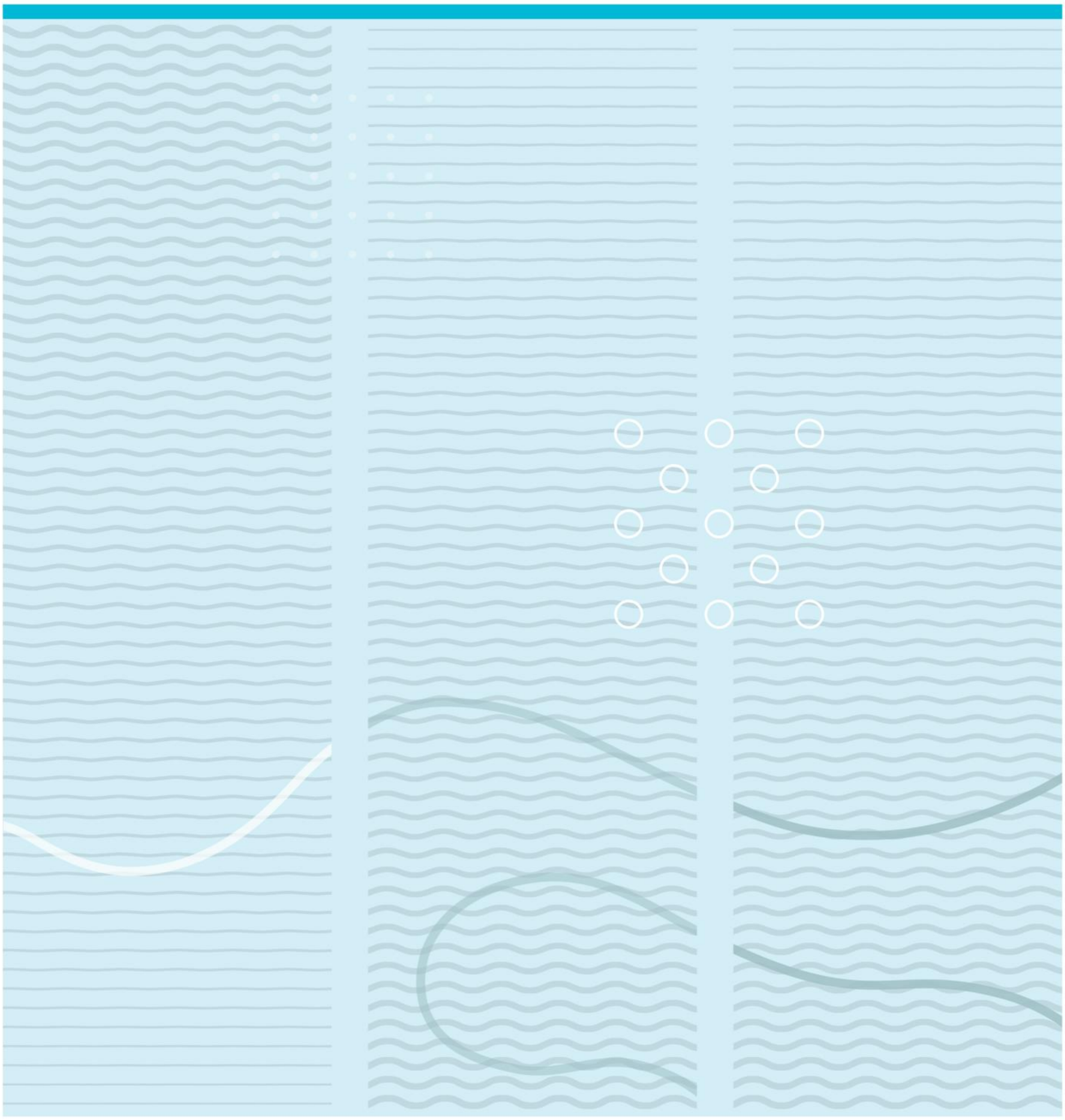
University College of Southeast Norway
Faculty of Electrical, IT and Cybernetics
-
Master's Thesis
Study program: System and Control Engineering
Spring 2016

Minh Hoang

Tuning of viscosity and density of non-Newtonian fluids through mixing process using multimodal sensors, sensor fusion and models



University College of Southeast Norway
Faculty of Electrical, IT and Cybernetics
Kjølnes Ring 56, 3918 Porsgrunn, Norway

http://www.usn.no

This thesis is worth 30 study points

Abstract

The main work of this thesis was to develop some empirical models that could be used to estimate the viscosity in drilling operations. The reason for this is due to the fact that oil companies spends a lot of money to make sure that the drilling fluid has the right value of density and viscosity. It is important to control the density because its maintains the downhole pressure and wellbore stability. Likewise, it is crucial to maintain the viscosity of the drilling fluid at a desired level for transportation of drilling cuttings and hole cleaning.

In drilling operations, the viscosity in the drilling fluid changes for each circulation so the viscosity value need to be updated back to the reference value. This can be done by adding additives to the drilling fluid. To decide the amount of additives that is needed, viscosity blending mechanisms is used. The blending methods that can be used to mix drilling fluids will be discussed in Chapter 2. In this thesis, there has been developed some classifiers that will sort the viscosity into three regions; low viscous, medium viscous and high viscous. By using classifiers, it would be easier for the mud engineers to know which additive that needs to be added.

The different models that were used to estimate the viscosity in this thesis were Fuzzy Logic model, Feedforward Artificial Neural Network model (ANN), Feedback Artificial Neural Network model and Support Vector Regression (SVR). The performance analysis of these models were done using simulation study and experimental study. Based on the simulation study, Sugeno type-1 Fuzzy Logic model, feedforward ANN model and SVR gives very good estimations compared to the feedback ANN models. For the experimental study, the experiments were done in the Venturi-rig in University College of Southeast Norway, Porsgrunn. The experimental results were very similar to the simulation results, where the three models; Sugeno type-1 Fuzzy Logic, feedforward ANN and SVR had comparable predictions with some accuracy. Based on the analysis from simulation and experimental study, it seems that the empirical models that were developed is capable of estimating the viscosity of non-Newtonian drilling fluids.

In addition to the task description, I and my supervisors have also developed a Matlab toolbox "Dynamic Artificial Neural Network for Time Series Analysis and Prediction". This toolbox was accepted as a paper for "The 9th Eurosim Congress on Modelling and Simulation-2016" in Finland. Apart from this, I as a co-

author manage to get accepted a paper on flowrate measurement of non-Newtonian fluids in the same conference.

Preface

This master thesis indicates that my two years at HSN University College of Southeast Norway is coming close to an end. It has been a great journey where I have learnt a lot and got the chance to make many good friends.

For this thesis I want to thanks all my supervisors for giving me advices and motivating me during the whole semester. It has been many tough days with confusions and little sleep. So I am without a doubt, sure that without you guys I would never be able to finish this thesis.

First of all, I want to thank my main supervisor Saba Mylvaganam. You have been very supportive and motivating from day 1. You always make sure that I make some progress every week.

My second thank goes to Khim Chhantyal. You have been the supervisor that I've been cooperating most with. It must have days where you got tired of all my questions, but I'm really thankful that you always kept your cool and never gave up on me. You are a good supervisor and a great person.

My third thank goes to Håkon Viumdal. Even though we haven't had so many meetings together, I feel that in the few meetings we had, you really put efforts and time to help me.

Finally, I want to thank my external partner, Geir Elseth. Your expertise in the field is something that cannot be learnt through books. It has been a nice experience for me cooperating with you.

Abbreviation

HSN	Høgskolen i Sørøst-Norge
SVM	Support Vector Machine
SVR	Support Vector Regression
ANN	Artificial Neural Network
DANN	Dynamic Artificial Neural Network
NN	Neural Network
MSE	Mean Squared Error
MAPE	Mean Average Percentage Error
GUI	Graphical User Interface
RNN	Recurrent Neural Network

Symbol

Re	Reynolds number
μ	Dynamic viscosity
v	Velocity
d	Diameter
ρ	Density
τ	Shear stress
γ	Shear rate
f	Friction factor
a	Empirical parameter
β	Empirical parameter
K	Flow consistency
n'	Flow behavior index
$\dot{\gamma}_w$	Wall shear rate
τ_w	Wall shear stress
η	Kinematic viscosity
x_m	Mole fraction
x_v	Volume fraction
x_w	Weight fraction
ϵ	Empirical parameter
M	Molecular weight

L	Length
dp	Differential pressure

Table of Contents

List of figures

1 Introduction:

1.1 General background

The oil companies in these days are facing challenges related to cost and efficiency in drilling operations. Oil companies are always looking for possibilities to improve their method in order to pump more efficiency and at the same time operates in a safe manner. Some of the factors that is vital for the efficiency, is to monitor and control the density and viscosity in the drilling fluid. This challenge may be solved by developing control algorithms which can ensure that the drilling fluid that circulates in the drilling loop, have a value which is acceptable for both viscosity and density.

In drilling operations, the drilling fluid is circulated in a closed loop starting from the mud tank into the wellbore and back to the mud tank. The mud can be water-based, oil-based or gas-based and is circulated during the drilling operation, until the desired depth is reached. During circulation, the properties of drilling fluid have significant importance for the safe and efficient drilling operation. The viscosity, density, and flow rate or circulating drilling fluid play a vital role, in all drilling operations. [1]

The goal of this thesis is to get a deeper understanding of how the drilling operation works, the importance of the drilling fluids, how to monitor and control density and viscosity, and develop empirical models which can be used to estimate the viscosity of non-Newtonian fluids.

1.2 Structure of the thesis

This thesis is divided into several chapters where Chapter 2 covers the literate survey of drilling functions, non-Newtonian rheology, viscosity measurements and blending mechanisms. Chapter 3 will describe the Venturi-rig and the available sensors which will be used for the experimental part. Chapter 4 will cover the methods that will be used to develop models. Chapter 5 will be focusing on simulations results based on the methods described in chapter 4, and finally Chapter 6 that will discuss the experimental results.

2 Literature survey on drilling and drilling fluids

This chapter will give a brief understanding of the fundamentals in drilling operations. It includes topic like circulation of drilling fluids, the functions of drilling fluids, rheology of non-Newtonian fluids, different methods to measure viscosity for both laminar and turbulent flow, and it will also include the methods for viscosity blending mechanism.

2.1 Circulation of drilling fluid

In today's drilling operations, companies are trying to make their operations more efficiency so that they can have a bigger profit. One of the main factor for achieving this goal, is to monitor and process the drilling fluid. The drilling fluid that is used today is a lot more advanced than the drilling fluid that was used back in the old days. In 1901, the drilling fluid was simply made of just water mixed with clay cuttings to make the fluid more viscous. Compared to the past, today's drilling fluid is more complex to make and include substances like bentonite, polymers, thinners and barite. Since the goals is to achieve optimal performance, the mud engineers are developing methods to reduce the waste of drilling additives, to control the extractions of cuttings better and limit the emissions of toxic elements. By implementing the methods together with a computer system to monitor the drilling fluid's properties, the drilling expenses can be reduced by 70 percent. [1]

The process of a drilling fluid loop can be demonstrated as shown in Figure 1. The pump is pumping out drilling fluid from the tank and transporting it down to the drill string. The drilling fluid will then carry the cuttings from the bottom of the pipe, and into a Solids control system to separate the cuttings and small particles from the drilling fluid. The drilling fluid will then go back to the mud tank where it is possible to add some additives to maintain the mud properties at a desired level. [2]

The continuous monitoring of drilling fluid properties like density and viscosity can lead to safe and efficient drilling. The density of the drilling fluids is responsible for wellbore stability and viscosity is responsible for transportation of drill cuttings. There are two main problems in drilling operations regarding wellbore stability; circulation loss and kick. A similar situation occurs frequently in geothermal drilling. In geothermal drilling, one of the costly problems is lost circulation that occurs when drilling fluid is lost to the formation rather than returning to the surface, preferable intact. The management of lost circulation is important and requires the accurate measurement of drilling fluid flow rate both into and out the well. This thesis is more concern with viscosity measurement of non-Newtonian fluids and the detail on flow measurement is not included in this thesis. However, we managed to get acceptance of the paper on flow measurement in "The 9th Eurosim Congress on Modelling and Simulation-2016", Finland, titled as: "*Flowrate Estimation using Ultrasonic Level Sensors using Dynamic Artificial Neural Networks with Real Time Recurrent Learning – A Comparative Study of Models and Practical Implementation*". The paper is attached in the appendix B.

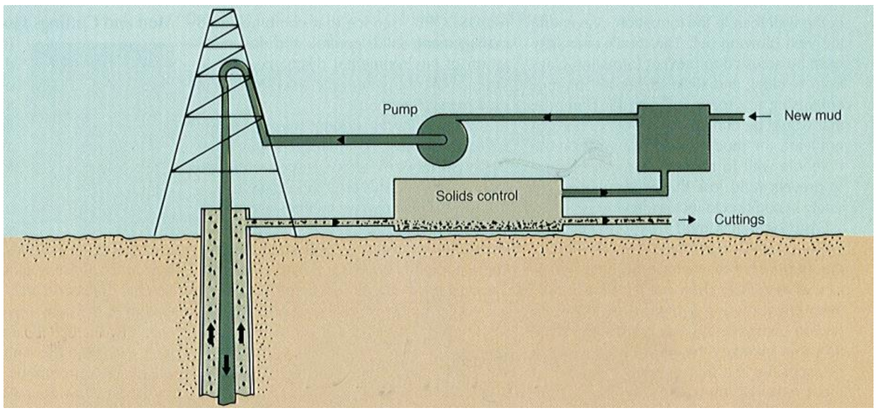


Figure 1: Circulation of drilling fluid in a drilling operation, which carries the cuttings out from the pipe. [2]

2.2 Functions of the drilling fluids

The drilling fluids that is used in the drilling operations, are designed to handle many important functions like transportation of cuttings, management of formation pressures, cooling and lubricating of the drilling bit, and ensure stability in the wellbore. Drilling fluids are created differently based on the requirements from each wellbore. The mud Engineers that design the drilling fluid need to take rig capabilities and environmental concerns into consideration when developing the fluid. The drilling fluid should be able to control subsurface pressure, reduce the formation damage as much as possible, minimize the loss of drilling fluid, and optimize hole cleaning. [3]

2.2.1 Transportation of cuttings

During a drilling process, when the drill bit is moving downwards in the pipe, a lot of cuttings will occur. These cuttings will eventually stop the drilling if they are not removed. This is due to the fact that the drill requires more power to proceed when more and more particles become obstructions. The drilling fluid is mixed in a way that when the fluid goes down to the pipe, it is very thin, but when the speed is reduced because of the cuttings, the fluid automatically becomes thicker. It is because of the drilling fluid’s ability to change viscosity that it can transport the cuttings from bottom of the pipe. The cuttings will be carried to a Solid control system where the cuttings will be removed from the drilling fluid. [2]

The Solid control system is mainly divided into three steps. When the cuttings arrive to the Solid control system, it will go into a shale shaker where the mesh screen will catch the big cuttings and smaller particles will continue to the next step. The remaining cuttings will be filtered by a mud cleaner before it reaches to the last step where all the fine solids are eliminated. The picture below shows how a Solid Control system works. [2]

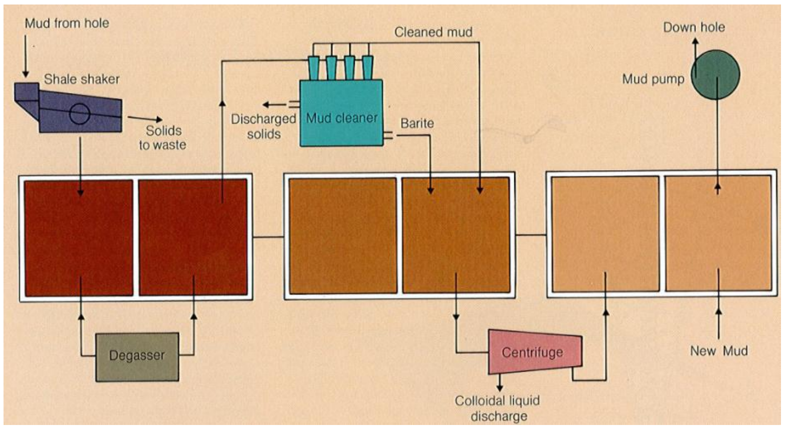


Figure 2: Solid control system which removes all the cuttings from the drilling fluid before it goes back to the mud tank. [2]

2.2.2 Lubrication and cooling

As the drilling bit is working, a lot of thermal energy will accumulate due to the frictions that is caused by the contact between the drilling bit and the cuttings. The temperature in the drilling bit needs to be cool down or else the drill might stop working as expected. This is where the drilling fluid comes into the picture. The drilling fluid that are being sent down to the wellbore, is transferring the thermal energy from the drilling bit and up to the surface. [3]

2.2.3 Management of formation pressure

The drilling fluid plays an important role when it comes to controlling a well. To prevent loss of well control, the drilling fluid that is being sent down through the drilling bit will increase the offset in the formation pressure. In this way, it is possible to avoid the formation fluids from getting into the borehole. It is however, very important to have in mind that the pressure from the drilling fluid must not be higher than the fracture pressure, or else the drilling fluid will be lost in the formation. [3]

2.2.4 Maintenance of wellbore stability

To maintain the stability in a wellbore there are some factors that needs to be fulfilled. The density in the drilling fluid should always be regulated to control the formation pressure. This is done by processing the mud column in such a way that it weighs more than the formation pressure. Furthermore, this will also prevent dangerous situation like the wellbore blowing up. [3]

2.3 Non-Newtonian rheology of drilling fluid

In the drilling process, the drilling fluids that are used is mostly non-Newtonian fluids. Non-Newtonian fluids are liquids that doesn't follow the law of Sir Isaac Newton. Newton's law says that the viscosity of fluids is dependent on only temperature or pressure. There is however, fluids that operates in a different way. These fluids viscosity can change based on other factors like pressure and shear rate and is therefore called Non-Newtonian fluids. [4]

Non-Newtonian fluids reacts differently when a force is applied to them compared to Newtonian fluids. While Newtonian fluids shows little reaction when receiving stress, Non-Newtonian fluids reacts immediately by changing form. Non-Newtonian fluids can either become more viscous or less viscous depending on which substances the fluid was made of. A Non-Newtonian fluid that becomes less viscous when receiving stress, is called Shear thinning fluid. On the other hand, fluids that change to a more viscous form after collecting stress, is called Shear thickening fluid. Despite the fact that Non-Newtonian fluid behaves differently when receiving stress, it will go back to their initial state when the force is removed. The picture below demonstrates how the viscosity changes for different types of Non-Newtonian fluids when a force is applied over time. [4]

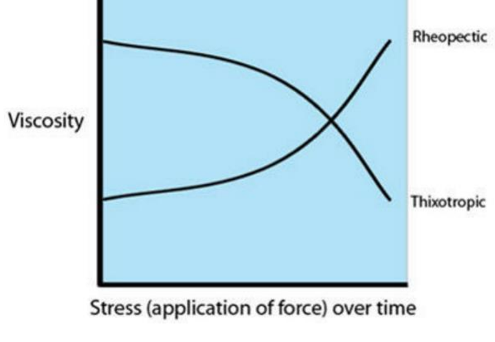


Figure 3: Shows how the viscosity changes over time for Rheopectic and Thixotropic fluids. [4]

2.3.1 Time-independent fluid

This subchapter will describe fluids that are time-independent. Fluids that are time independent are not dependent on the duration of flow. For time-independent fluid there exist two types; shear-thinning and shear-thickening fluid. [5]

2.3.1.1 Shear-thinning fluid

These fluids will have their viscosity decreased when the shear rate increases. An example of this can be Ketchup. If you want to get out the remaining Ketchup from the bottle, you need to shake the bottle a few times so it is possible to squeeze out the ketchup. The purpose behind the shaking, is to apply stress to the sauce, so that the viscosity decreases. [4]

Viscosity can be found using this formula:

$$\eta = \frac{\tau}{\gamma} \tag{1}$$

Where η is viscosity, τ is shear stress and γ is shear rate.

Figure 4: Ketchup is a shear thinning fluid. [4]

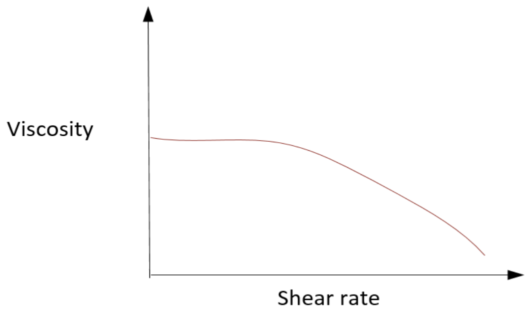


Figure 5: For shear thinning fluid, the viscosity decreases when the shear rate increase. [4]

2.3.1.2 Shear-thickening fluid

Fluids that increase in viscosity when the shear rate increase, is called Shear-thickening fluids. An example of a fluid with this function is oobleck. Before stress is applied, the oobleck is in liquid form, but as soon at someone grab it, the form will start to get thicker. [4]

Figure 6: Oobleck is a shear thickening fluid. [4]

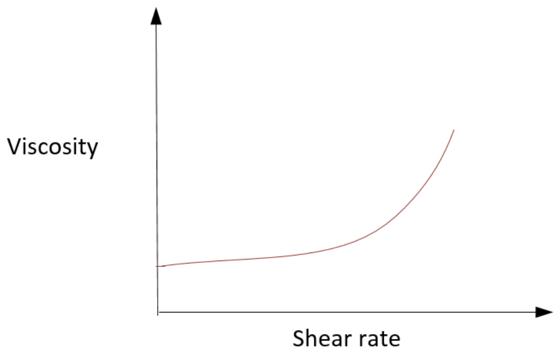


Figure 7: The viscosity for shear thickening fluids increase when the shear rate increase. [4]

2.3.2 Time-dependent fluid

This subchapter will describe fluids that are time-dependent. Fluids that are time-dependent are dependent on the duration of flow. For time-dependent fluid there exist two types; thixotropic and rheopectic fluid. [6]

2.3.2.1 Thixotropic fluid

These fluids will unlike the fluids mentioned above, have their viscosity changed over time and not only through stress. For thixotropic fluids, the viscosity will decrease as time goes by. An example of this type of fluid is honey. If you have a cup with honey and stirs the honey, you will see that the honey start changing form from solid to liquid. [4]

Figure 8: Honey is a thixotropic fluid. [4]

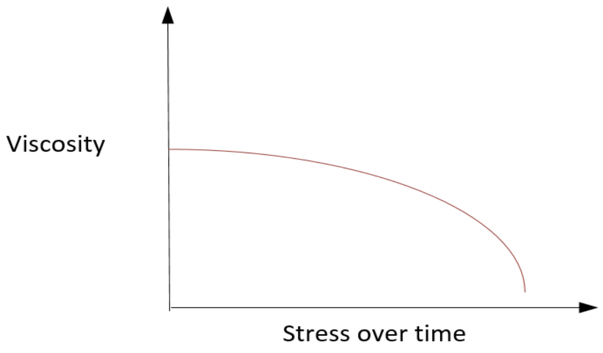


Figure 9: For thixotropic fluids, the viscosity decrease when it receives stress over time. [4]

2.3.2.2 Rheopectic fluid

Fluids that have their viscosity increased over time when stress is applied, is called Rheopectic fluids. To illustrate how these type of fluids behaves, an example with cream will be used. When the cream is getting whipped over some time, the viscosity increase and the cream gets thicker. [4]

Figure 10: Cream is a rheopectic fluid. [4]

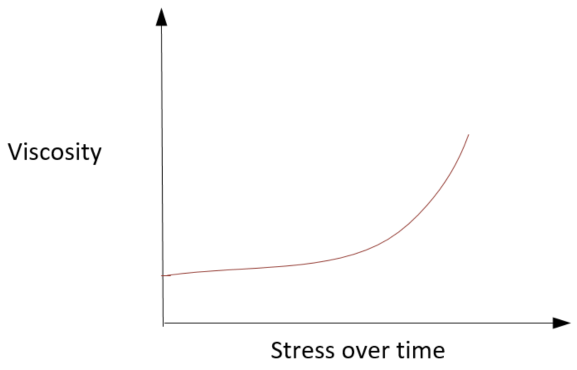


Figure 11: Shows how rheopectic fluids viscosity increase over time when stress is applied. [4]

2.4 Viscosity measurement of drilling fluid in different flow conditions

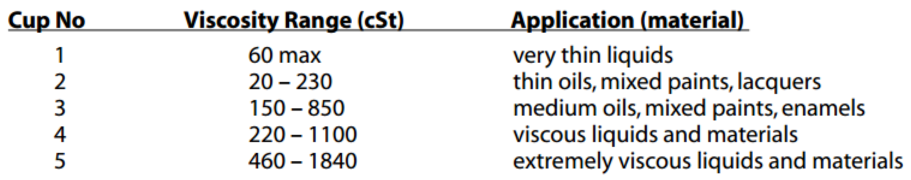
When measuring viscosity of drilling fluids, there are different measurements methods for different fluids types like Non-Newtonian and Newtonian. The choice of method is also dependent on if the flow is turbulent or laminar. This subchapter will describe the methods than can be used to measure viscosity for different flow conditions.

2.4.1 Lab scale viscosity measurement on fluid samples

This subchapter will describe some lab instruments that can be used to measure the viscosity. Note that these instruments don’t measure in real time. For real time viscosity measurements, see under laminar and turbulent flow.

3.4.1.1 Zahn Cup

A Zahn cup can measure the viscosity by taking the time a fluid uses to flow through the orifice which is positioned at the bottom of the cup. The Zahn cup comes in different sizes which support different viscosity ranges. There is therefore important to select the right cup when measuring viscosity. Zahn cups that is produced by an engineering company called Brookfield have this table that will help the user to choose the right cup [7]:



Cup No	Viscosity Range (cSt)	Application (material)
1	60 max	very thin liquids
2	20 – 230	thin oils, mixed paints, lacquers
3	150 – 850	medium oils, mixed paints, enamels
4	220 – 1100	viscous liquids and materials
5	460 – 1840	extremely viscous liquids and materials

After the time is taken, the user can use the time and the size of the Zahn cup to find the viscosity in a conversion table.

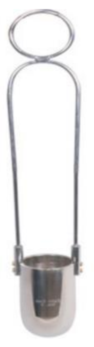


Figure 12: Zahn Cup, a lab instrument to measure viscosity.

3.4.1.2 March Funnel

This instrument is often used to measure the viscosity of drilling mud. The behavior of this device almost works like a Zahn Cup, but with this device there is no need for a conversion table. The time (seconds) it takes to transfer the fluid from a funnel to fill up the Marsh cup is equals to the amount of viscosity (1 second = 1 viscosity). [8]

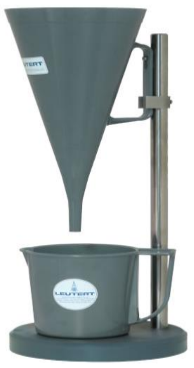


Figure 13: Marsh Funnel instrument

3.4.1.3 Capillary viscometer

This instrument is very common used to measure viscosity due to its simplicity and low cost [9]. The viscosity is found by counting the time it will take for the fluid to flow through the capillary. Since the flow time is proportional to the fluid's kinematic viscosity, it is possible to calculate the viscosity value by using a conversion factor. The conversion factor can differ from instruments to instruments, so remember to check the conversion number in the instrument's specification [10]. The picture below illustrated how to use a capillary viscometer.

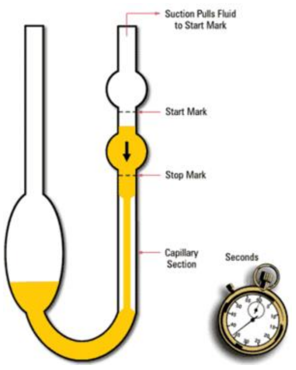


Figure 14: Capillary viscometer

3.4.1.4 Rotational viscometer

A viscometer that can find the viscosity for both Newtonian and non-Newtonian fluids. The device is calculating the viscosity based on the resistance from an object that is rotating in the fluid. The object can for an instance be a spindle that rotates inside the fluid. Viscometers are also common used it other Vendors like the Food industry that use rotational viscometer to quality check their food to have a viscosity value within an acceptable range. [11]

Figure 15: Rotational viscometer

2.4.2 Viscosity measurement in laminar flow

To determine the viscosity for fluids that moves smoothly or in proper paths, there are some instruments that can be used to find the viscosity for both Newtonian and non-Newtonian fluids. These instruments can for instance be Ultrasonic Doppler Velocimetry, or Laser Doppler Velocimetry (LDV). By using these instruments, it is possible to find the viscosity by dividing shear stress (pressure drop in the measurement) with shear rate (the velocity profile from the instrument). [12]

2.4.2.1 Ultrasonic Doppler Velocimetry

Ultrasonic Doppler was originally developed for medical purposes, but the technology has later been used in other fields such as in fluid dynamics. This sensor can be used to determine the speed in a laminar flow by emitting ultrasonic waves which will be reflected and picked up by a receiver. [13]

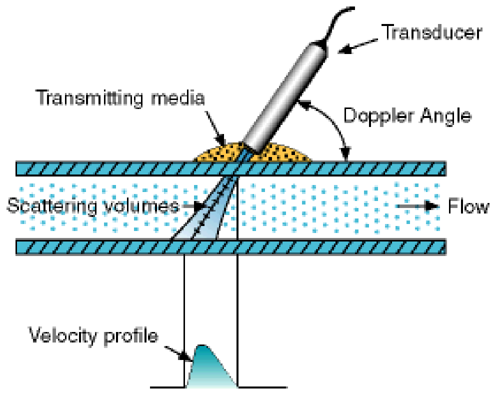


Figure 16: Ultrasonic Doppler Velocimetry can be used to determine the laminar flow.

2.4.2.2 Laser Doppler Velocimetry

This instrument can be used to measure the velocity of a flow with very high accuracy with no need for calibration. There is however a downside with this sensor, and that is the cost. The Laser Doppler Velocimetry works in a such way that it sends out a laser toward a target, and the reflected radiation from the target will determine the velocity. [14]

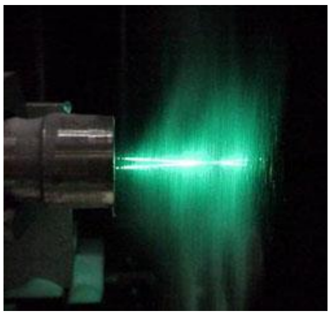


Figure 17: Laser Doppler Velocimetry, an alternative to measure laminar flow.

2.4.3 Viscosity measurement in turbulent flow

When it comes to measure the viscosity for turbulent flow, there exists different methods for Newtonian and non-Newtonian fluids. For Newtonian fluids, the viscosity can be determined by using a moody chart. Complementary to this, a model developed from Trinh can be used to find the viscosity for non-Newtonian fluids. [12]

2.4.3.1 For Newtonian fluids

To find the viscosity for Newtonian fluids in turbulent flow, a Moody Chart can be used. One possible way of doing this, is illustrated in the steps below [15]:

1. Find the number for the Pipe Roughness. This can be found in the pipe specification.
2. Use the Pipe Roughness value in a Moody chart to find the friction factor.
3. When the friction factor is known, find the Reynolds number from this formula:

$$Re = \frac{16}{f} \tag{2}$$

where, Re is the Reynolds number and f is the friction factor

4. Use Reynolds number to find the viscosity:

$$\mu = \frac{dv\rho}{Re} \tag{2}$$

where,
$\mu = viscosity\ of\ fluid$
$v = velocity\ of\ the\ fluid\ in\ the\ pipe$
$d = diameter\ of\ the\ pipe$
$\rho = density\ of\ the\ fluid$

2.4.3.2 For Non-Newtonian fluids

To find the viscosity for non-Newtonian fluids in turbulent flow, an analytical model developed by Trinh can be used. Note that this model can only be used for Power Law fluids. One possible way of doing this, is illustrated in steps below [16]:

1. Find the wall shear rate $\dot{\gamma}_w$ from this equation:

$$\dot{\gamma}_w = \left[\frac{\alpha v^{2-2\beta+n'\beta}\rho^{1-\beta}K^{\beta-1}8^{\beta(1-n')}\left(\frac{3n'+1}{4n'}\right)^{\beta n'}}{2D^{\beta n'}}\right]^{\frac{1}{n'}} \tag{3}$$

where,
$a = emperical\ parameter$
$v = flow\ velocity$
$\beta = emperical\ parameter$
$\rho = fluid\ density$
$K = flow\ consistency$
$D = pipe\ diamater$
$n' = flow\ behavior\ index$

2. Find the wall shear stress τ_w from this equation for pseudoplastic fluid:

$$\tau_w = K\dot{\gamma}_w^{n'} \tag{4}$$

where,

$\dot{\gamma}_w = wall\ shear\ rate$
$n' = flow\ behavior\ index$

3. Calculate the viscosity μ:

$$\mu = \frac{shear\ stress}{shear\ rate} = \frac{\tau_w}{\dot{\gamma}_w} \quad (5)$$

2.4.4 “Flow-viz” a new non-invasive viscosity measurement for non-Newtonian fluids

It seems there has been developed a new non-invasive instrument to measure viscosity for non-Newtonian fluids for all kinds flow conditions. The product name is “Flow-viz” and was created by Johan Wiklund with cooperation with “SP Technical Research Institute of Sweden”. Their product can do measurement in real time non-invasively by using an ultrasound based system. The instrument can deliver accurate measurements even through stainless steel pipes. The developers claim that by using their product, it is possible to; [17]

- Measure transient flows with coarse particles.
- Do consistency measurement without exposing the product.
- Increase the efficiency and flexibility without sacrificing safety.
- Monitor the changes in the system in real time.
- Increase productivity and reduce energy consumption.
- The time with inaccurate, time-consuming sampling and off-line sample analysis are now a history.

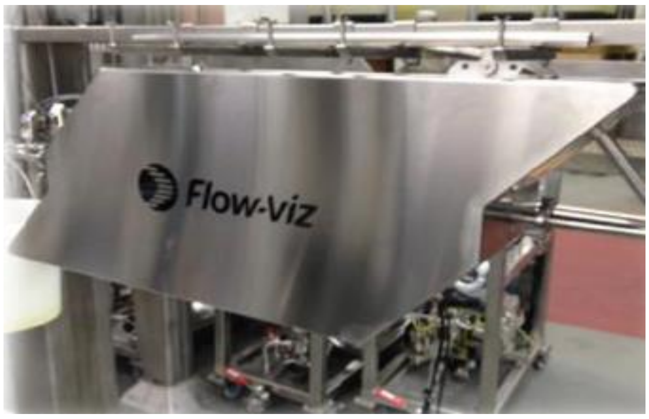


Figure 18: "Flow-viz" - an instrument to measure viscosity for non-Newtonian fluids in real time [17]

2.5 Viscosity blending mechanism

This subchapter will describe how to mix drilling fluids by using different methods that has been developed over the years since 1905. For each method, there will be some information about if the blending methods is for binary or multiply fluids, and also the steps for using the methods. The timeline below illustrates some blending mechanism methods that has been evolved from 1905 to 2000.

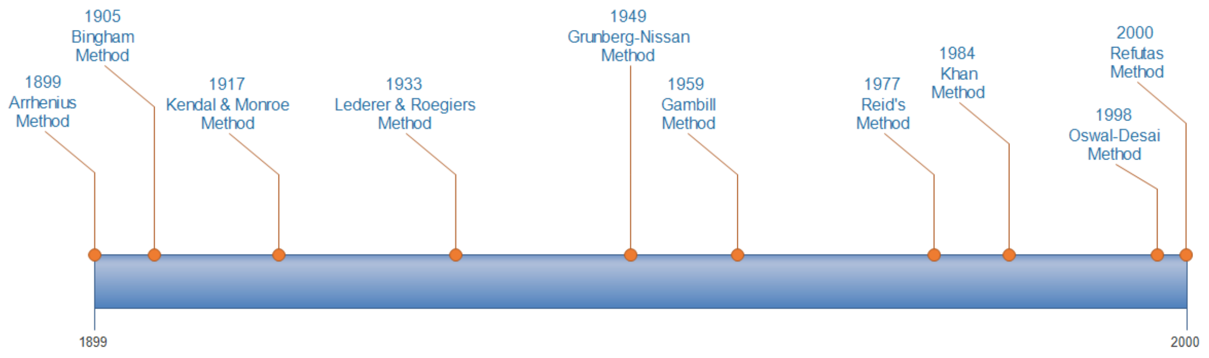


Figure 19: Timeline of blending methods from 1899-2000

2.5.1 Arrhenius method

This method is used for binary blending and is known as the ideal binary mixing, because the mixing between the fluids doesn't affect the volume. [18]

Steps:

1. Calculate the blend viscosity μ_{12} from this formula:

$$ln\mu_{12} = x_{m1} ln\mu_1 + x_{m2} ln\mu_2 \tag{6}$$

where, x_{mi} $(i = 1,2)$ is the mole fraction and μ_i $(i = 1,2)$ is the dynamic viscosity

2.5.2 Bingham's method

This model is a binary blending method that is generally not very accurate for viscosities predictions in petroleum oil blends, because it was designed for "ideal" solutions. [19]

Steps:

1. Calculate the blend viscosity μ_{12} from this formula:

$$\frac{1}{\mu_{12}} = \frac{x_{v1}}{\mu_1} + \frac{x_{v2}}{\mu_2} \tag{7}$$

where, x_{vi} $(i = 1{,}2)$ is the volume fraction and μ_i $(i = 1{,}2)$ is the dynamic viscosity.

2.5.3 Kendal & Monroe's method

This is a binary blending method which use cubic-root average of each fluid to find the blend viscosity η_{12}. The downside of Kendal & Monroe's method, is that their equation doesn't give satisfactory accuracy. [18]

Steps:

1. Calculate the blend viscosity η_{12} from this formula:

$$\eta_{12}^{1/3} = x_{w1}\eta_1^{1/3} + x_{w2}\eta_2^{1/3} \tag{8}$$

where, x_{wi} $(i = 1{,}2)$ is the weight fraction and η_i $(i = 1{,}2)$ is the kinematic viscosity.

2.5.4 Lederer & Roegiers method

This is a binary blending method which was independently created by Lederer & Roegiers. Researches shows that this method is one of the most accurate equations that use one-parameter. [18]

Steps:

1. Calculate the blend viscosity μ_{12} from this formula:

$$ln\mu_{12} = ln\mu_1 + \frac{\alpha x_2}{x_1 + \alpha x_2}\left(ln\mu_2 - ln\mu_1\right) \tag{9}$$

$$= \frac{x_1}{x_{m1} + \alpha x_2} ln\mu_1 + \frac{\alpha x_2}{x_{m1} + \alpha x_2} ln\mu_2 \tag{10}$$

where, x_{mi} $(i = 1{,}2)$ is the mole fraction and μ_i $(i = 1{,}2)$ is the dynamic viscosity, and α is the empirical parameter for the difference cohesion energy between the mixing components.

2.5.5 Grunberg & Nissan's method

This method is an extension of the Arrhenius method which include an additional term to describe non-ideality of a system. Grunberg & Nissan's method is a binary blending method. [18]

Steps:

1. Calculate the blend viscosity μ_{12} from this formula:

$$ln\mu_{12} = x_{m1} ln\mu_1 + x_{m2} ln\mu_2 + a x_1 x_2 \tag{11}$$

where, x_{mi} $(i = 1,2)$ is the mole fraction, μ_i $(i = 1,2)$ is the dynamic viscosity, and a is the empirical parameter.

2.5.6 Gambill's method

Gambill developed a model to estimate the kinematic viscosity of a binary blending. [20]

Steps:

1. Calculate the kinematic viscosity μ_k from this equation:

$$\eta_k^{\frac{1}{3}} = x_{ma} \eta_{ka}^{\frac{1}{3}} + x_{mb} \eta_{kb}^{\frac{1}{3}} \tag{12}$$

where,
$\eta_k = kinematic\ viscosity$
$x_m = mass\ fraction$
$a = component\ 1$
$b = component\ 2$

2.5.7 Reid's method

This method is for binary blending and defines the kinematic viscosity of a mixture consisting of two components. [21]

Steps:

1. Calculate the kinematic viscosity η_k from this formula:

$$ln\eta_k = x_{mA}^3 ln\eta_{k_A} + 3x_{mA}^2 X_B lnv_{AB} + 3x_{mA}x_{mB} lnv_{AB} + x_{mB}^3 ln\eta_{k_B} + R^0 \tag{13}$$

where,

$$R^0 = x_{mB}^3 ln\frac{M_B}{M_A} + 3x_{mA}x_{mB}^2 ln\frac{1+\frac{2M_B}{M_A}}{3} + 3x_{mA}^2 X_B ln\frac{2+\frac{M_B}{M_A}}{3} - lnx_{mA} + x_B\frac{M_B}{M_A} \tag{14}$$

$x_m = mole\ fraction,$
$M = molecular\ weight$
$v_{AB}, v_{BA} = constanst\ that\ will\ be\ determined\ by\ least\ squares\ method$
$A = component\ 1$
$B = component\ 2$

2.5.8 Khan's method

Khan developed two empirical models with double logarithm to predict viscosity; one linear and one non-linear model. [21]

Linear:

$$lnln(\mu) = C_1 lnT + C_2 \tag{15}$$

Non-linear:

$$lnln(\mu) = \{1.0 + b_1T + b_2(b_1T)^2\}e^{b_1T} \tag{16}$$

where,
$\mu = dynamic\ viscosity\ [mPas]$
$T = temperature[K]$
$C = parameter\ determined\ by\ least\ squares\ technique$
$b = parameter\ determined\ by\ least\ squares\ technique$

2.5.9 Oswal-Desai's method

This is a binary blending method that is built from the Grunberg-Nissan's method. Compared to Grunberg-Nissan's method, this method adds two additional terms that will improve the accuracy, but in return make a more complex model due to the extra parameters. [18]

Steps:

1. Calculate the blend viscosity η_{12} from this formula:

$$ln\eta_{12} = x_{m1}ln\eta_1 + x_{m2}ln\eta_2 + \epsilon x_1 x_2 + K_1 x_{m1} x_{m2}(x_1 - x_2) + K_2 x_{m1} x_{m2}(x_{m1} - x_{m2})^2 \tag{17}$$

where, x_i $(i = 1,2)$ is the mole fraction, η_i $(i = 1,2)$ is the dynamic viscosity, ϵ is the parameter for empirical interaction, and $K_i(i = 1,2)$ is the extra parameters.

2.5.10 Refutas method

This is a binary blending method which are very common used in the petroleum industry. This method is known for their double-logarithmic in their equation. [18]

Steps:

1. Find the blending index A_i for each components with this formula:

$$A_i = 14.534ln[ln(\mu_i + 0.8)] + 10.975 \quad (i = 1,2) \tag{18}$$

where, μ_i is the kinematic viscosity

2. Find the average viscosity blending index A_{12}:

$$A_{12} = x_1 A_1 + x_2 A_2 \tag{19}$$

where, x_i $(i = 1,2)$ is the weight fraction

3. Calculate the blend viscosity η_{12}:

$$\eta_{12} = exp\left[exp\left(\frac{A_{12} - 10.975}{14.534}\right)\right] - 0.8 \tag{20}$$

3 System description for Venturi-rig

This chapter will describe all the setups and the equipment that can be used for experiments. By reading this chapter, the reader will get an overview of how the experimental data explained later in chapter 6 was acquired. The Venturi-rig is a test rig that was created in 2013 by a project group consisting of machine and automation students at HSN. The project was a cooperation between students from HSN and a company named Statoil. The purpose behind this project was to create a rig which could be used to test different measurements methods based on fluid circulation in the rig. [22]

3.1 Overview of the system with P&ID

The Figure 20 below gives an P&ID of the sensors, actuators connected to the Venturi-rig and tanks. [23]

Description of a normal circulation process in the rig:

1. The fluid in Tank 1 will be pumped out through Valve 1, and on the way to the feeding tank, it will pass through sensors that will give information to the operators.
2. The feeding tank is the step where the tank is filled up before the fluid continues to flow through the Venturi section which has three level sensors installed.
3. The fluid flows back to tank 2 and goes back to tank 1 through valve 3.

A more detailed description of how to use the Venturi-rig, is provided in the Appendix C.

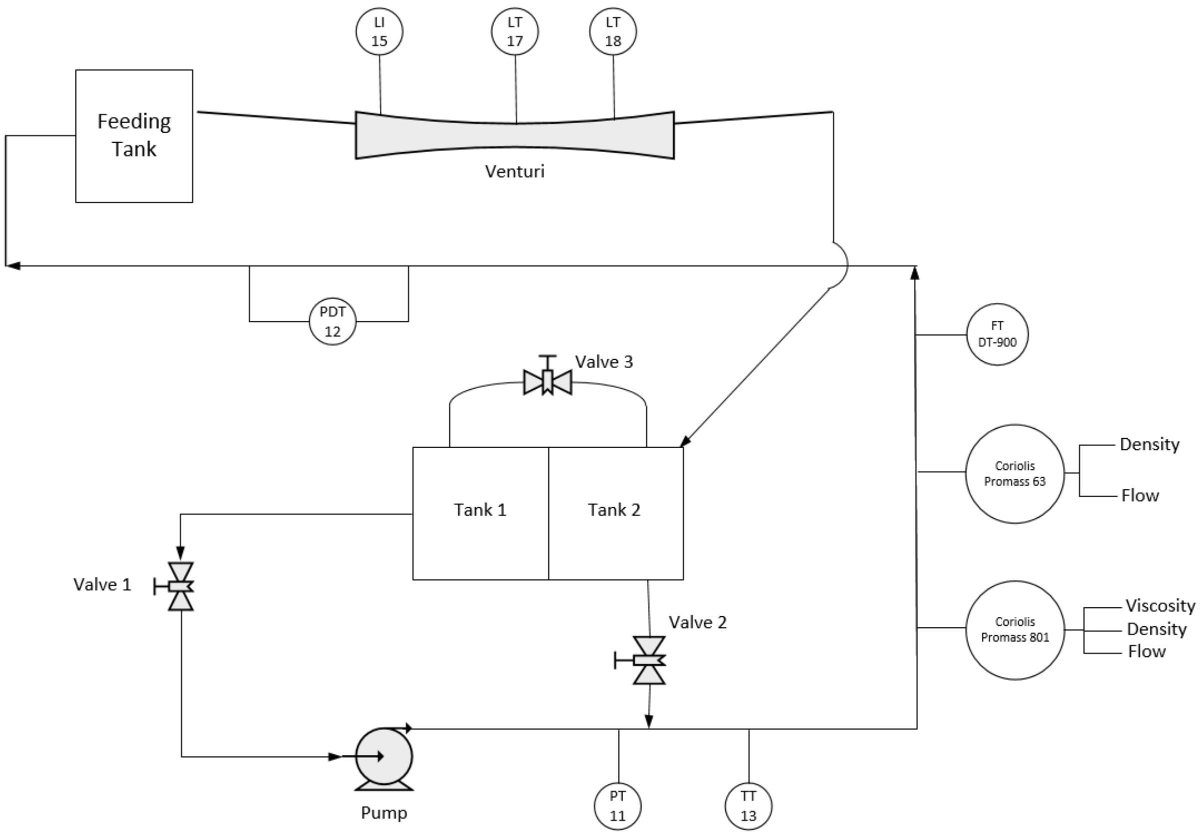


Figure 20: P&ID diagram of the system.

Where,

Sensor	Description
Coriolis promass 801	Three outputs: • Viscosity • Density • Flow
Coriolis promass 63	Two outputs: • Density and flow
TT	Temperature Transmitter
FT	Flow Transmitter
PDT	Pressure Differential transmitter
LI	Level Indicator
DT900	Gamma sensor

3.2 Functions of the Venturi section

This 3D picture below was created in AutoCAD where the specifications was taken from a bachelor thesis in 2015 at HSN. [23]

As seen in the picture, there are three level sensors, LT-18, LT-18 and LT-15. It is also possible to adjust the position of these sensors. The fluid will in this picture flow from right to left before it goes back to the tank. There is also a throat section below level sensor LT-18. This smaller section will cause a significant jump which will be registered by sensor LT-18. This information can together with the other level sensors, be used to calculate the flow rate. [24]

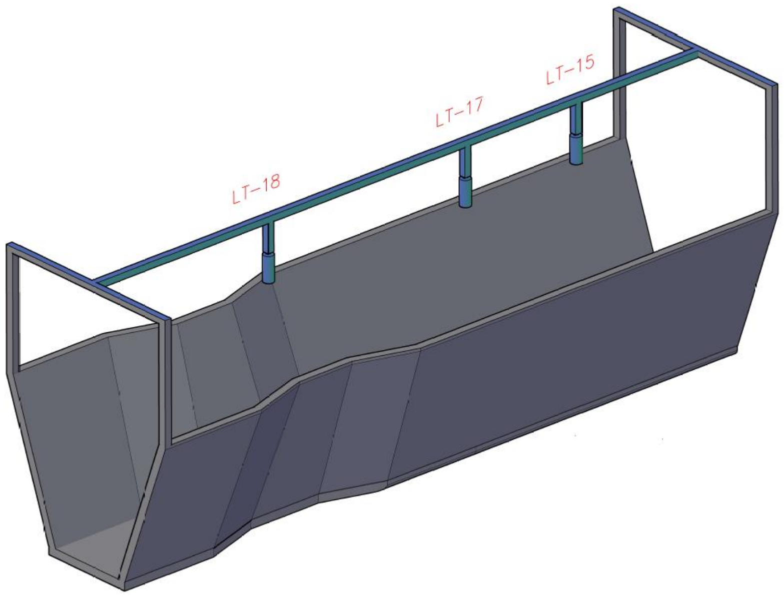


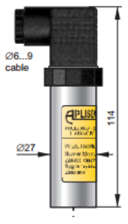
Figure 21: Venturi section with three level sensors

3.3 Sensors used in the Venturi-rig

The sensors that are available in the Venturi-rig will be specified in this subchapter. The subchapter will mainly focus on the range and the accuracy of the sensors.

3.3.1 Pressure transmitter

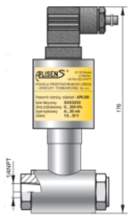
This sensor can be used to measure the pressure of gases, vapours and liquids. [23, 25]

Specification	Picture
	

Vendor:	Aplisens
Type:	PCE-28
Range:	0-7 bar
Accuracy:	$\pm 0{,}1\%$

3.3.2 Pressure differential transmitter

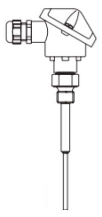
This sensor can be used to measure the differential pressure of gases, vapours and liquids. [23, 26]

Specification	Picture
	

Vendor:	Aplisens
Type:	APRE-2000
Range:	0-250 mbar
Accuracy:	$\pm 0{,}1\%$

3.3.3 Temperature transmitter

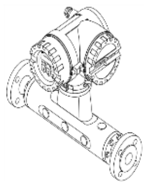
This sensor can be used to measure the temperature of gases and liquids. [23, 27]

Specification	Picture
	

Vendor:	Aplisens
Type	TST41N
Range	0-100 °C
Accuracy	$\pm 0{,}19\%$

3.3.4 Coriolis flow meter (Promass 63)

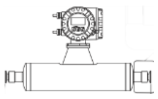
This sensor can be used to measure the mass and volume flow of fluids. It has two analog output which is mass flow and density [24].

Specification	Picture
	

Vendor:	Endress & Hausser
Type:	Promass 63
Range	Massflow: 0-1000 l/min Density: 900-1600 kg/m^3
Accuracy:	Liquid: $\pm 0.10\%$ Gas: $\pm 0.50\%$

3.3.5 Coriolis flow meter (Promass 801)

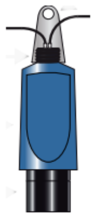
This sensor can be used to measure density, viscosity and flowrate [27]

Specification	Picture
	

Vendor:	Endress & Hausser
Type:	Promass 801
Range	Massflow: 0-1000 l/min Density: 900-1600 kg/m^3 Viscosity: 0-200 cP
Accuracy:	Liquid: $\pm 0.10\%$ Gas: $\pm 0.50\%$

3.3.6 Ultrasonic level sensor

This sensor can be used to measure the level of the fluid in the system. [23, 28]

Specification	Picture
	

Vendor:	Rosemount
Type	3107 Level
Range:	0.3-12 m
Power supply:	12…40 V DC

3.3.7 Gamma sensor

This sensor can be used to measure the density of liquids with high speed. [23, 29]

Specification	Picture

Vendor:	S-TEC
Type:	DT-9300
Range:	1-1.7 gm/cc
Accuracy:	$\pm 0.2\%$

3.4 Fluids used in the Venturi-rig

The fluids that will be used for the experimental part in chapter 6, will be described in this subchapter. The two fluids that will be used consists of water, potassium carbonate and xanthan Gum. The table below will show the properties of the fluids. [1]

Table 1: The properties of the fluids used in the experiment

	Fluid 1	**Fluid 2**
Density	1160 kg/m^3	1428 kg/m^3
pH	11.91	13.68
Characteristic	• Low density • High viscosity	• High density • High viscosity
Recipe	Water mixed with Potassium Carbonate and Xanthan gum	Water mixed with Potassium Carbonate and Xanthan gum

3.5 Empirical model setup

To determine viscosity, the general equation need shear stress and shear rate. In the Venturi-rig, it is not possible to measure shear rate based on the available sensor systems. However, it is possible to measure shear stress of incompressible non-Newtonian fluid at any flow regime using the Equation 21, where D, L, dp and τ are diameter of pipe, length of pipe, pressure drop in the length L, and shear stress respectively. The differential pressure measurement can be determined using differential pressure sensor in the Venturi-rig. To develop empirical models, 30 different fluid samples with different density and viscosity are used. Therefore, the empirical model consists of density of the fluid and shear stress as inputs and viscosity as output.

$$\tau = \frac{D}{4} \times \frac{dp}{L} \tag{21}$$

4 Basics of empirical methods used

In this thesis, different data models are studied. This chapter will describe the approaches that is used to develop data models to estimate the viscosity for non-Newtonian drilling fluid. The methods include Fuzzy Logic approach, Artificial Neural Network approach and Support Vector Machine.

4.1 Fuzzy logic

Fuzzy logic is a logical tool that can be used to represent arguments that lies between true and false. An example of a how fuzzy logic can be used, is to think about a glass of water. Where the traditional Boolean logic (0 or 1) can only describe the glass as either empty (0) or full (1), fuzzy logic can in addition to the Boolean logic, also determine other possible outcomes like half full (0.5) and almost full. A fuzzy logic tool can be thought of as a function that receives inputs and give out an output based on the rules and the membership functions that has been specified in the setup of fuzzy logic. [30]

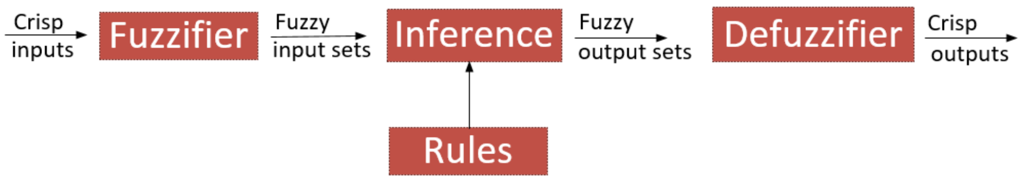


Figure 22: A block diagram of a type-1 fuzzy logic system with a complete overview of how it works. [30]

Figure 22 shows the block diagram of fuzzy logic implementation. It includes fuzzification, inference mechanism, rules and defuzzification. Fuzzification is a process of converting crisp inputs into fuzzy input sets using suitable membership functions. In Fuzzy Logic, a selection of suitable inference mechanism plays a vital role in the output of fuzzy logic model. In general, there are two types of inference mechanism; Mamdani inference mechanism and Sugeno inference mechanism. The inference mechanism is connected to the rules of fuzzy logic. The rules will govern the implementation of fuzzy logics. Finally, defuzzification converts the fuzzy set values to crisp outputs. [30]

In this thesis, fuzzy logic with Mamdani and Sugeno inference mechanism are studied for viscosity estimation. For this study, Matlab fuzzy logic toolbox is used. In addition, type-2 fuzzy logic with Sugeno inference mechanism is analyzed for viscosity modeling. In this analysis, the toolbox developed by Kumbasar, [31] is used. Figure 23, shows the block diagram of type-2 fuzzy logic system. The only difference with type-1 fuzzy logic system is the “type-reducer” block in type-2 fuzzy logic system. In type-2 fuzzy logic system, the membership functions are interval type and need to be reduced during defuzzification process using type-reducer [31]. There are several types of type-reducer incorporated in the toolbox developed by Kumbasar, [31].

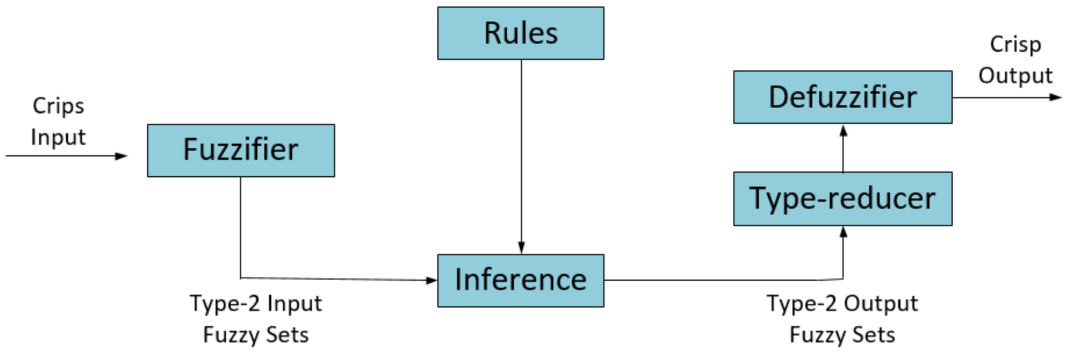


Figure 23: A block diagram of a type-2 fuzzy logic system with a complete overview of how it works. [31]

4.2 Artificial Neural Network

Artificial Neural Network (ANN) is a pattern recognition method that was inspired by the way human brain interacts. A neuron is like a living cell in a human's brain that receives and processes inputs before it generates an output. By creating a lot of neurons, we have a network that can train computers to think more like a human. The reason for why it is desired to make the computers operates more like a human, is due to the fact that there are tasks that are very simple for humans, but not for computers and opposite. A human can for instance easily distinguish between a cat and a dog, while it would be more difficult for a computer. [30, 32]

Before implementing a ANN model, the model has to be trained and validated. Without proper training, the model will have inaccuracy in the outputs. There are different types of training algorithm and the general process for training the model can be briefly summarized as follows [30];

1. The inputs to the model are connected to neurons in the hidden layer, neurons are connected to each other in multiple hidden layers and with output layer. Each connection is assigned with a synaptic weight, which are the model parameters. A bias can be added to each of the neurons in the network. The weights are updated as the model is trained.
2. The neurons will combine the weight and the inputs together before it moves on to the activation function where the output of the model will be determined. There exist many

different types of activation functions, so the selection of an activation functions depends on the specification of the application that the model will be used for. Some common activation functions are linear function, step function, ramp function and tansigmoid function.

3. The output from the model are compared with target values and the weights of the network are updated based on the error between model output and target value. For each updated weights, the error in the model becomes smaller and smaller. This step will be repeated until the error is within the threshold that has been specified in the training algorithm.
4. The trained model is further validated and tested with validation and testing datasets before implementing.

Neural network can be broadly divided into two types; feedforward (static) and feedback (dynamic) [30]. These types will be described in the subchapter below.

4.2.1 Feedforward Artificial Neural Network

A feedforward neural network uses current inputs and outputs to develop the model [30]. It is used for estimation and classification of static applications. The architecture of the network will always move in one direction and never backwards as shown in Figure 24.

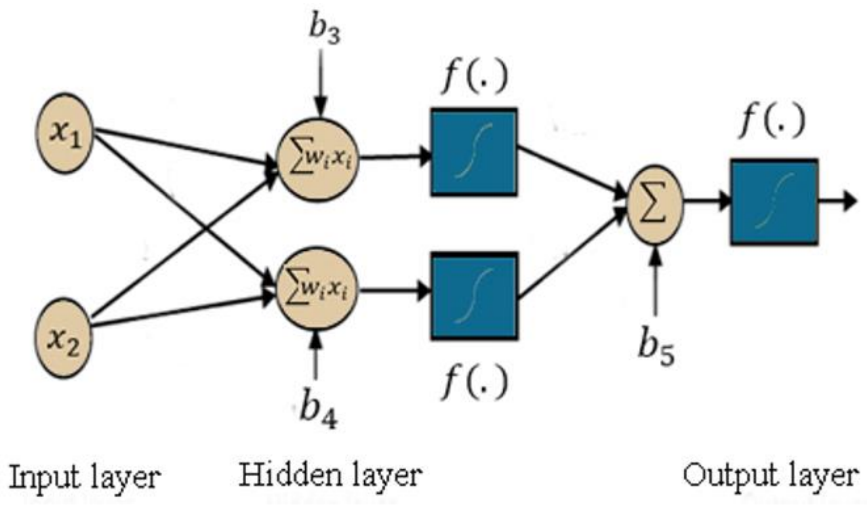


Figure 24: Architecture of Feedforward Neural Network with two inputs, two neurons and one output. [33]

In order to implement Feedforward ANN, Matlab Neural Network Toolbox is used in this thesis. In Matlab NN toolbox, there are three different types of learning algorithms;

a) **Levenberg-Marquardt learning algorithm**: in this learning algorithm, training stops as the validation error increases as compared to the training error. This algorithm is faster in learning, but takes more memory. [34]
b) **Bayesian Regularization learning algorithm**: in this learning algorithm, training stops as per the minimization of adaptive weights. This algorithm is slow, but can give good result for difficult, small and noisy datasets. [34]
c) **Scaled Conjugate Gradient learning algorithm**: this learning algorithm works similar to Levenberg-Marquardt learning algorithm, but takes less memory. [34]

The simulation study and experimental study using Feedforward ANN is discussed in Chapter 5 and Chapter 6 respectively.

4.2.2 Feedback Artificial Neural Network

A Feedback ANN uses previous inputs and previous outputs to develop the model. It is used to perform time-series predictions. Based on the architecture of a network, feedback ANN can either be Partially Connected Recurrent Neural Network or a fully connected Neural Network. The difference between these two architecture lies in the feedback loops. Partially Neural Network do not have self-feedback loops, while Fully Connected Neural Network has it. [35]

4.2.2.1 Partially Connected Recurrent Neural Network

To implement Partially Connected Recurrent NN, Matlab NN toolbox is used in this thesis. The architecture of Partially Connected Recurrent NN with feedback loops is shown in Figure 25. The toolbox has a possibility to use three different kinds of networks.

a) **Non-linear Input- Output network**: in this partially connected recurrent NN, different number of previous inputs are used to estimate current output.
b) **Non-linear Autoregressive**: in this partially connected recurrent NN, different number of previous outputs are used to estimate current output.
c) **Non-linear Autoregressive with External Input**: in this partially connected recurrent NN, different numbers of both previous input and output are used to estimate current output.

The learning algorithms for all these partially connected recurrent NN are same as that are used in feedforward ANN. In this thesis, Non-linear Autoregressive with External Input network is used for the simulation and experimental analysis of partially connected NN. The simulation study and experimental study are discussed in Chapter 5 and Chapter 6 respectively.

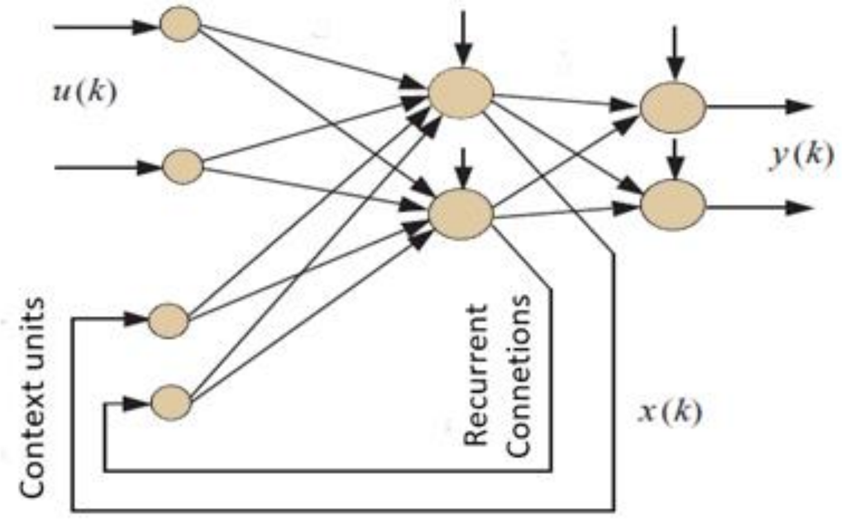


Figure 25: A simple architecture of Partially Connected Recurrent NN with feedback from the hidden neurons. [30]

4.2.2.2 Fully Connected Recurrent Neural Network

To make a usage of Fully Connect Recurrent Neural Network, a Matlab toolbox was developed under this thesis work. This toolbox was developed as the existing Matlab NN toolbox doesn't have a possibility to use fully connected recurrent NN. A detailed description on how to use this toolbox is presented in [35] and is attached in the appendix C. Figure 26 shows the architecture of Fully Connected Recurrent NN that is used in the developed toolbox. The simulation study and experimental study using Feedforward ANN is discussed in Chapter 5 and Chapter 6 respectively. The developed toolbox includes three different types of learning algorithms:

a) Back Propagation Through Time (BPTT): it is extension of classical gradient-based backpropagation algorithm. In this learning algorithm, the feedforward ANN architecture is unfolded into feedback ANN with different number of folds. This is an offline learning algorithm. It converges faster and can be complex if the number of fold increases. [33, 35]
b) Real Time Recurrent Learning (RTRL): it is the most accepted online learning algorithm. It is simple but converges very slow. [33, 35]
c) Extended Kalman Filter Learning (EKF): it is an online learning algorithm. It is the fastest converging learning algorithm and is complex compared to other learning algorithms. [33, 35]

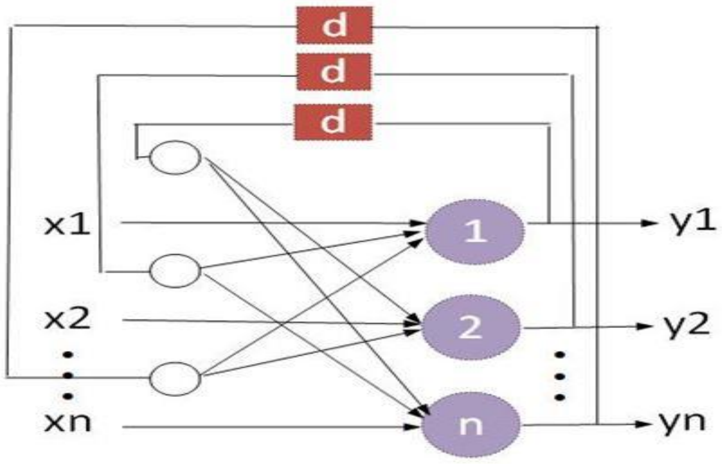


Figure 26: General architecture of Fully Connected Recurrent Neural Network with feedback loops. [35]

4.3 Support Vector Machine

Support Vector Machine (SVM) is used in different applications like pattern recognition, classification problems, time-series prediction and regression analysis [33]. The basic idea of SVM is to develop a mapping between input and output space by transferring the original input space into higher dimensional feature space using kernel functions [36]. In this thesis, SVM is used in regression mode as Support Vector Regression (SVR) to solve the regression problem. A general architecture of SVR is shown in Figure 27. The simulation study and experimental study using Feedforward ANN is discussed in Chapter 5 and Chapter 6 respectively.

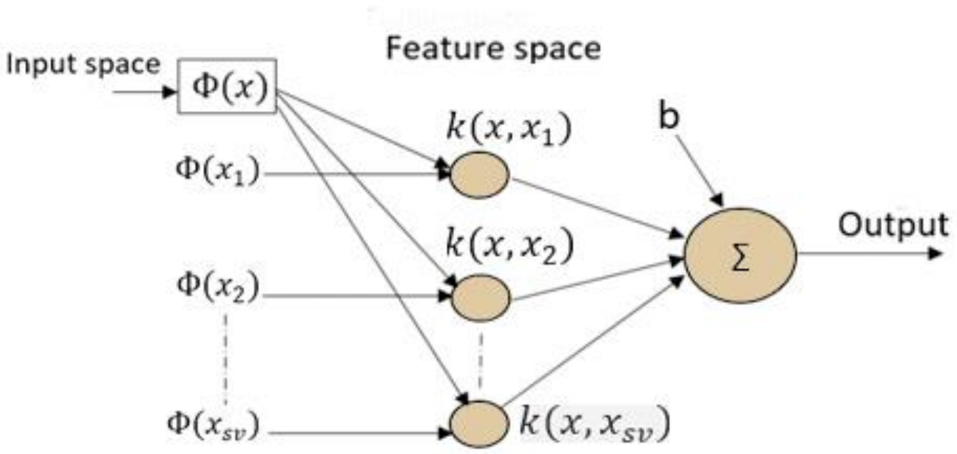


Figure 27: General architecture of Support Vector Regression method showing input space, feature space and output space.

5 Viscosities of sample fluids in empirical models

In this Chapter, all the simulation results from different methods discussed in Chapter 4 are presented. Simulations are performed in Matlab and LabVIEW software. For the development of different data models, previously measured dataset is used. The dataset is generated using 30 different drilling fluids samples with laboratory viscosity measuring device in Statoil [12]. The dataset consists of shear stress, shear rate and viscosity of 30 different samples. Chhantyal, [12] has used shear stress and density as inputs and viscosity as outputs for his empirical models. In this work, different empirical models are tested for the estimation of viscosity. The evaluation of the developed models is done based on the Mean Absolute Percentage Error (MAPE) criterion.

5.1 Fuzzy logic simulation with fluid samples

Two different types of fuzzy logic approaches are investigated for viscosity estimation. The classical type-1 fuzzy logic with Mamdani and Sugeno inference and type-2 fuzzy logic with Sugeno inference mechanism are tested.

5.1.1 Mamdani Inference Mechanism with Type-1 Fuzzy Logic

Figure 28 is a block representation of Fuzzy logic model with Mamdani inference mechanism with shear stress and density as two inputs, and viscosity as output of the model. Figure 29, Figure 30, and Figure 31 show the membership plot of each variables in the fuzzy logic model. In this modeling, triangular membership function is selected due to its simplicity. Each variables is represented by the linguistic variables. The linguistic variables are represented as LL is low-low, L is low, M1 is close to medium, M2 is medium, H is high, and HH is high-high. In order to compare between the variables, the values of each variable are normalized to [0,1]. Table 2 shows 20 different rules designed for the estimation of viscosity based on the shear stress and density measurements in column and rows respectively. The rules are generated based on the basic principle of non-Newtonian drilling fluids. Based on these rules, the predictions are plotted against the target values as shown in Figure 32. The model prediction using the Fuzzy model is capable of identifying the behavior and basic principle of the non-Newtonian fluid. However, the accuracy of the model is not acceptable. It is also true that the model can be improved a lot by formulating more rules that can cover different cases. Nevertheless, as rules increases the complexity of the model increases and might not be applicable to implement in online applications. In fuzzy logic modeling, the better understanding of system, behavior or dynamics of process can lead to better model. It is therefore necessary to spend a lot of time and tune the model to map the inputs and output for better model representation. The optimal number, type of membership functions and the implementation of suitable rules are biggest challenges while developing fuzzy logic model with Mamdani inference mechanism.

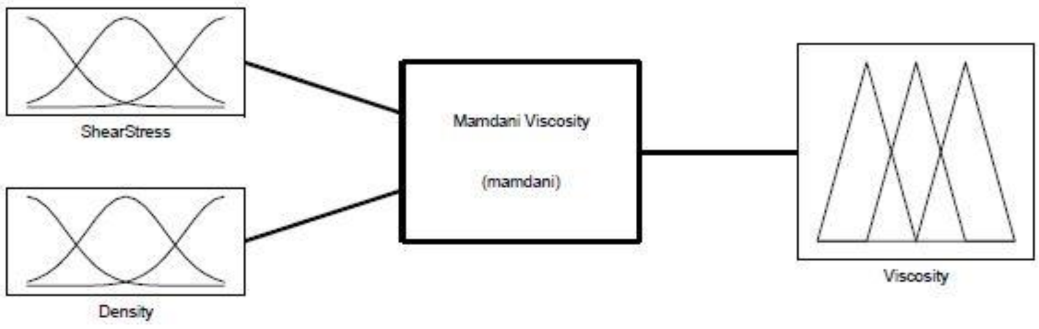


Figure 28: A block representation of Fuzzy Logic approach with Mamdani inference mechanism. The model consists of two inputs (Shear stress and Density) and one output (Viscosity).

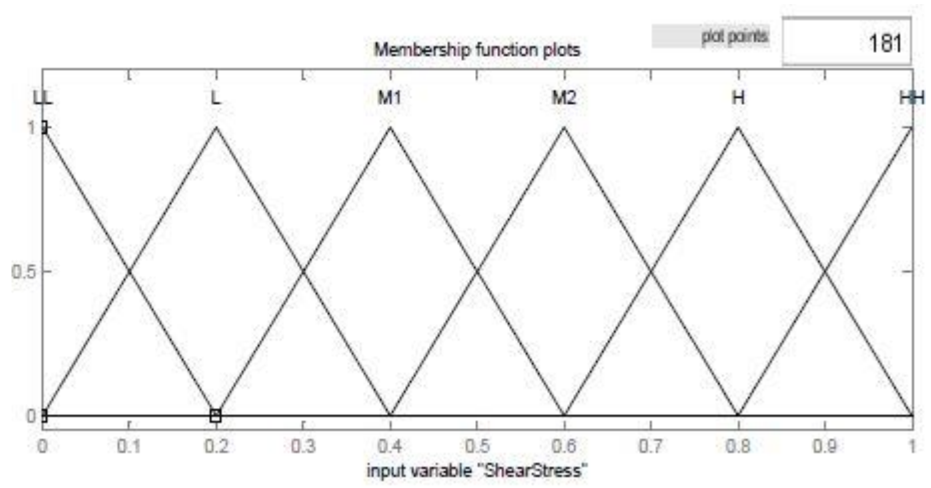


Figure 29: The membership function plot in Mamdani type-1 fuzzy logic for shear stress with 6 different linguistic values within the range of [0,1].

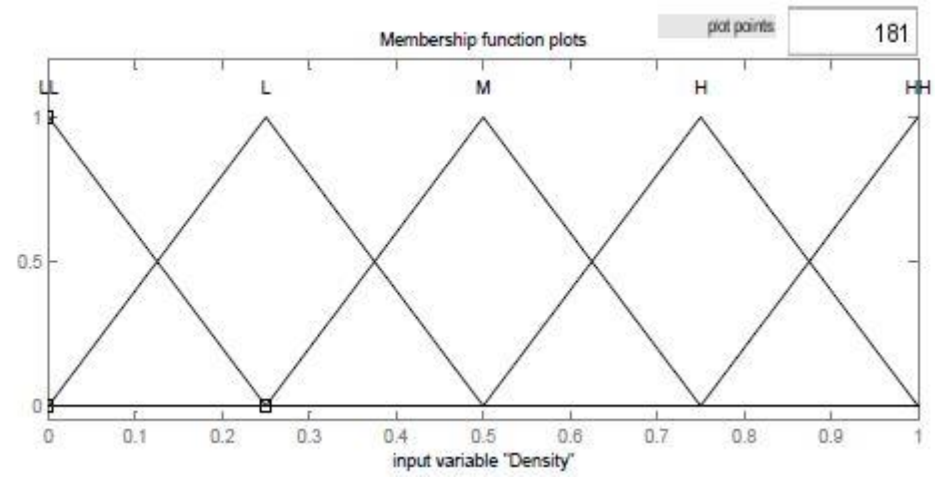


Figure 30: The membership function plot in Mamdani type-1 fuzzy logic for density with 5 different linguistic values within the range of [0,1].

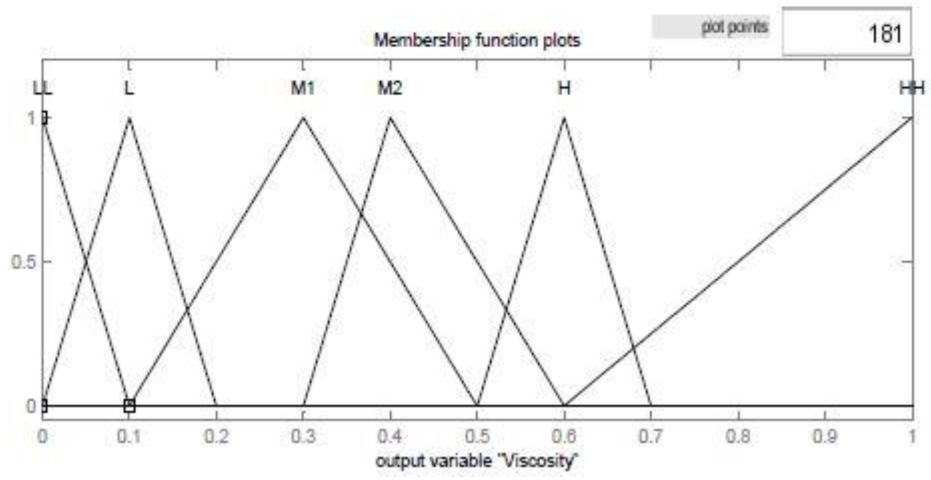


Figure 31: The membership function plot in Mamdani type-1 fuzzy logic for viscosity with 6 different linguistic values within the range of [0,1].

Table 2: If-then rules of Mamdani type Fuzzy model for viscosity estimations based on shear stress and density measurements.

$\tau\backslash\rho$	**LL**	**L**	**M**	**H**	**HH**
LL	M1	L	M2	H	HH
L	L	L	L	M1	H
M1	LL	LL	LL	L	M2
M2			LL	LL	
H				LL	L
HH					LL

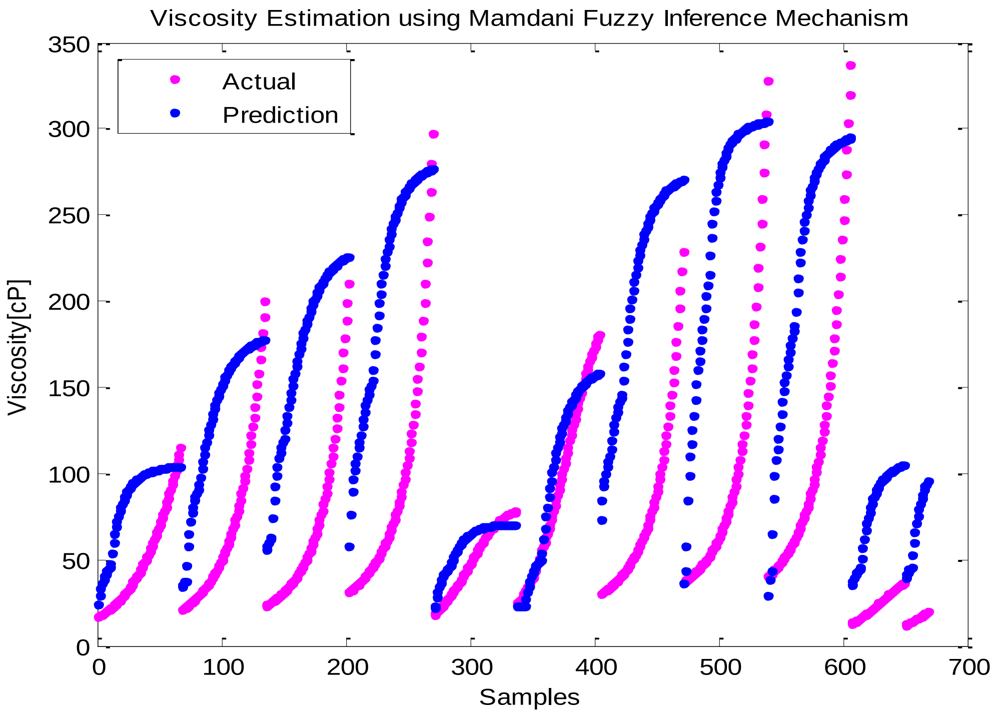


Figure 32: The target vs. prediction plot using the Mamdani type-1 Fuzzy logic model.

5.1.2 Mamdani Inference Mechanism with Type-2 Fuzzy Logic

Type-2 Fuzzy logic uses interval type membership function that has upper limits and lower limits. The performance gets better with type-2 fuzzy logic method if the data consist of noise [31]. In this thesis, type-2 fuzzy logic with Sugeno inference mechanism is used with an intension to improve the fuzzy logic predictions. The open source toolbox provided by Kumbasar, [31] is used to implement type-2 fuzzy logic. Figure 33 and Figure 34 show the membership function plot for shear stress and density under type-2 fuzzy logic system respectively. The new linguistic values are added constructing an interval for each fuzzy sets. The same rules used in type-1 fuzzy logic model were considered in this model. However, there was no significant improvement in the model predictions using interval type fuzzy logic system. It is due to the fact that, type-2 fuzzy logic system can create a significant impact on fuzzy logic modeling with noisy data. In our case, all the calibration dataset are generated using highly accurate laboratory measurement system with almost no noise. Similar to type-1 fuzzy logic with Mamdani inference mechanism, the toolbox developed by Kumbasar, [31] has same limitations. It is difficult to find the optimal number of membership functions, type of membership functions, optimal rules and need a lot of experience and time to tune the model.

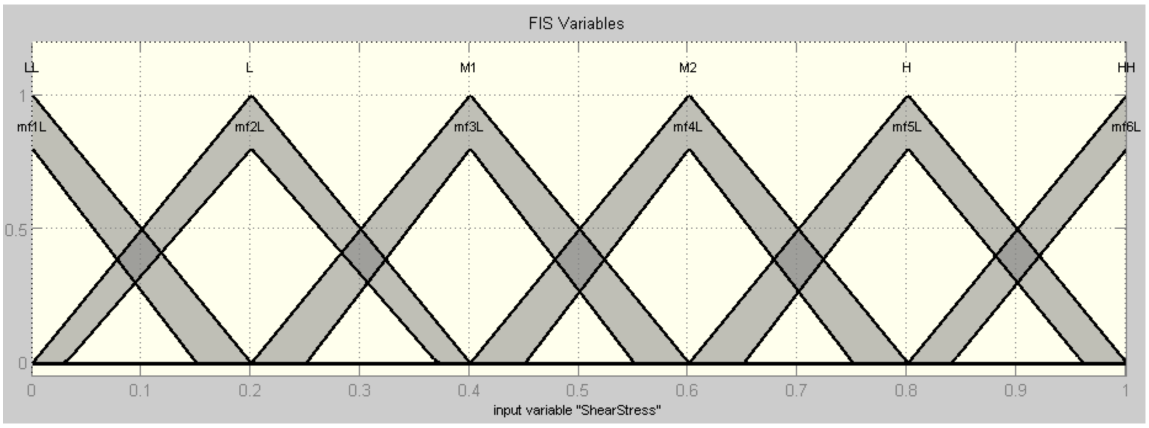


Figure 33: The membership function plot in Mamdani type-2 fuzzy logic for shear stress with 6 different interval type linguistic values within the range of [0,1].

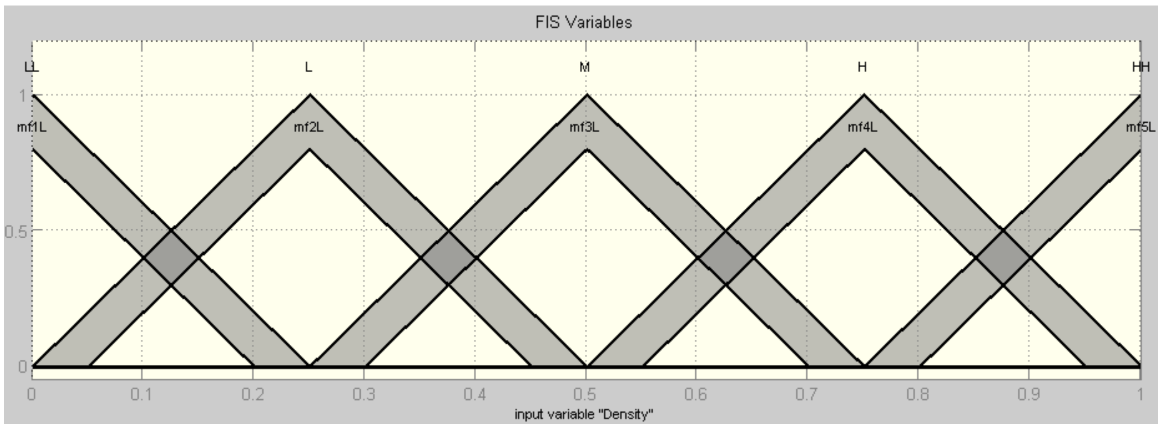


Figure 34: The membership function plot in Mamdani type-2 fuzzy logic for density with 5 different interval type linguistic values within the range of [0,1].

5.1.3 Sugeno Inference Mechanism with Type-1 Fuzzy Logic

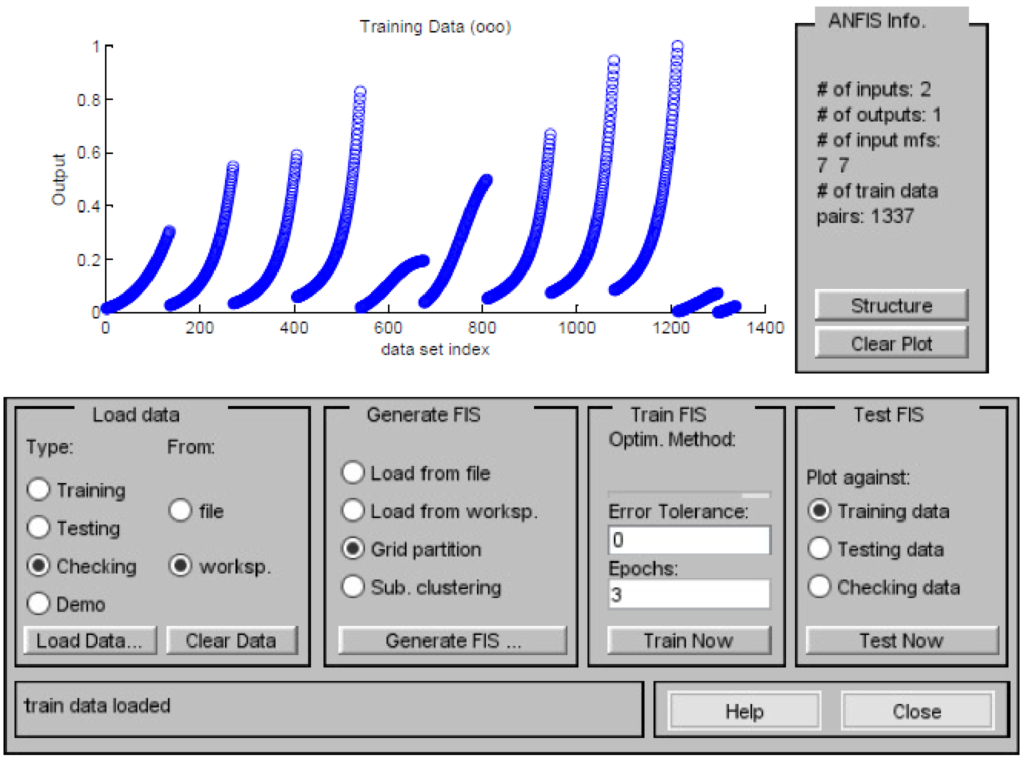
The calibration data are also used with type-1 fuzzy logic method with Sugeno inference mechanism. In Sugeno type-1 fuzzy logic, the output linguistic variables are not fuzzy sets. It is either linear or constant. Sugeno type-1 fuzzy logic has an additional feature called Adaptive Neuro-Fuzzy Inference System (ANFIS) GUI incorporated in Matlab Fuzzy Logic Toolbox as shown in Figure 35. In ANFIS, it is possible to import training data set, checking data set, and testing data set. As data sets are imported, it is possible to generate membership functions for each input variables using grid pattern or clustering approach. This solves one of the biggest challenges with type-1 Mamdani fuzzy logic. Though a user should define the number and type of membership function, the tuning of each membership function is automatically done by ANFIS. Most importantly, it will generate the fuzzy if-then rules automatically as the training is completed. This shows that type-1 Sugeno fuzzy logic is much easier to tune an optimal parameters and rules as compared to type-1 Mamdani fuzzy logic. However, the limitation with Sugeno type is the requirement of datasets. In the cases where dataset is not possible, Mamdani type is better as the output of Sugeno type is very difficult to tune manually. Figure 36 shows the ANFIS model structure with 7 membership functions for each inputs and 49 automatically generated if-then rules with respective outputs. Figure 37 and Figure 38 show the membership function plot for density and shear stress with the parameters of membership functions automatically generated using ANFIS GUI.

As the model is developed, it is tested using testing set. Figure 39 shows the comparison between model predictions and target values. It shows that the developed Sugeno type-1 fuzzy logic model is far better than Mamdani type-1 fuzzy logic model and is very accurate. Figure 40 shows the calibration results for 5 different samples out of 30 samples. The solid lines with different colors represents the test data and the characters with same color represents the predictions from the Sugeno type-1 fuzzy logic model. The calibration results shows the predictions are highly accurate with a MAPE of 2.45%.

Based on the simulation study on viscosity prediction using Fuzzy Logic approach, it can be seen that Sugeno type-1 fuzzy logic model has the best estimates. Hence, this method is selected to be implemented in the Venturi-rig for an experimental analysis, which is discussed under Chapter 6.

Table 3: If-then rules of Sugeno type Fuzzy model for viscosity estimations based on shear stress and density measurements.

$\tau\backslash\rho$	**Mf1**	**Mf2**	**Mf3**	**Mf4**	**Mf5**	**Mf6**	**Mf7**
Mf1	Mf1	Mf2	Mf3	Mf4	Mf5	Mf6	Mf7
Mf2	Mf8	Mf9	Mf10	Mf11	Mf12	Mf13	Mf14
Mf3	Mf15	Mf16	Mf17	Mf18	Mf19	Mf20	Mf21
Mf4	Mf22	Mf23	Mf24	Mf25	Mf26	Mf27	Mf28
Mf5	Mf29	Mf30	Mf31	Mf32	Mf33	Mf34	Mf35
Mf6	Mf36	Mf37	Mf38	Mf39	Mf40	Mf41	Mf42
Fm7	Mf43	Mf44	Mf45	Mf46	Mf47	Mf48	Mf49



ANFIS Info.

# of inputs: 2
# of outputs: 1
# of input mfs:
7 7
# of train data
pairs: 1337

Load data
Type: Training Testing [✓ Checking] Demo
From: file [✓ worksp.]

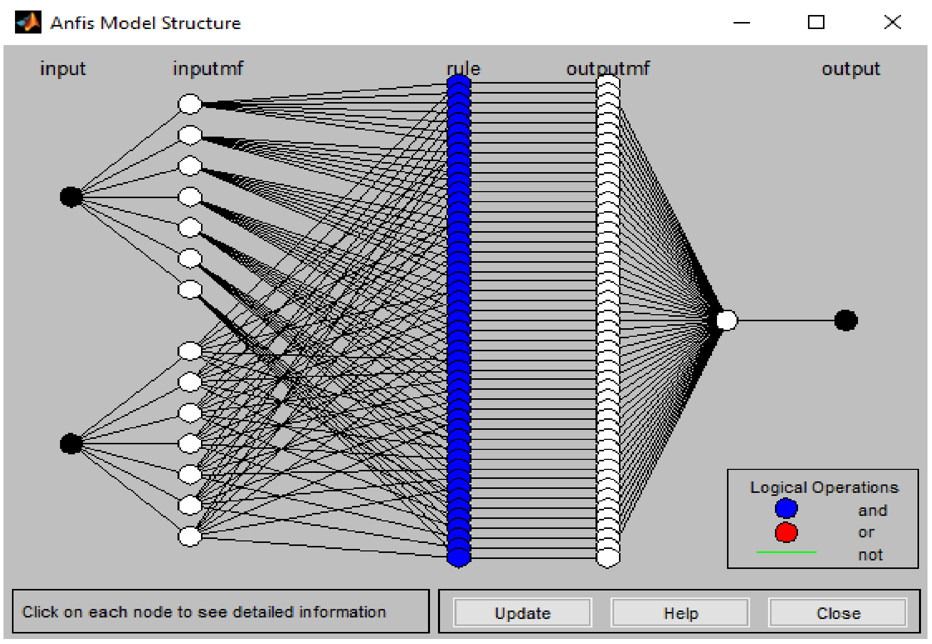
Generate FIS
Load from file
Load from worksp.
[✓ Grid partition]
Sub. clustering

Train FIS
Optim. Method:
Error Tolerance: 0
Epochs: 3

Test FIS
Plot against:
[✓ Training data]
Testing data
Checking data

train data loaded

Figure 35: ANFIS GUI of type-1 Sugeno fuzzy logic with a possibility to import different datasets for automatic tuning of parameters of membership functions and automatic generation of if-then fuzzy rules.



Click on each node to see detailed information

Figure 36: The Sugeno ANFIS model structure with two inputs having seven membership functions each and with 49 if-then rules and an output.

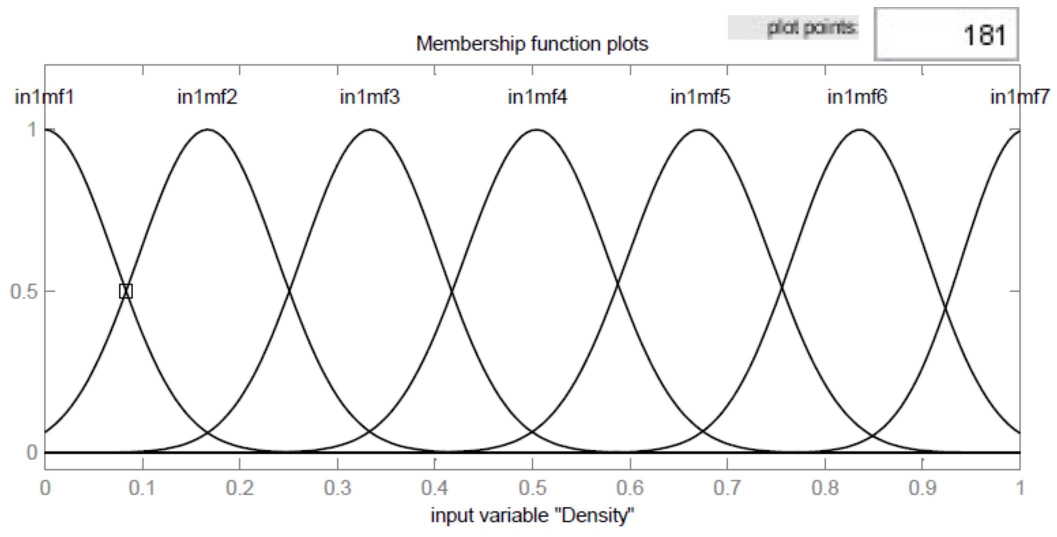


Figure 37: The membership function plot in Sugeno type-1 fuzzy logic for density with 7 different interval type linguistic values within the range of [0,1].

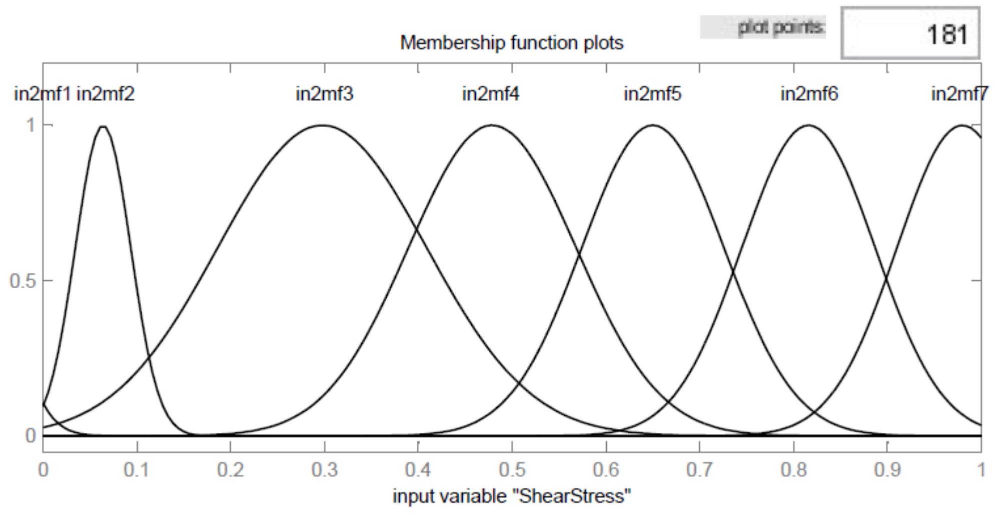


Figure 38: The membership function plot in Sugeno type-1 fuzzy logic for shear stress with 7 different interval type linguistic values within the range of [0,1].

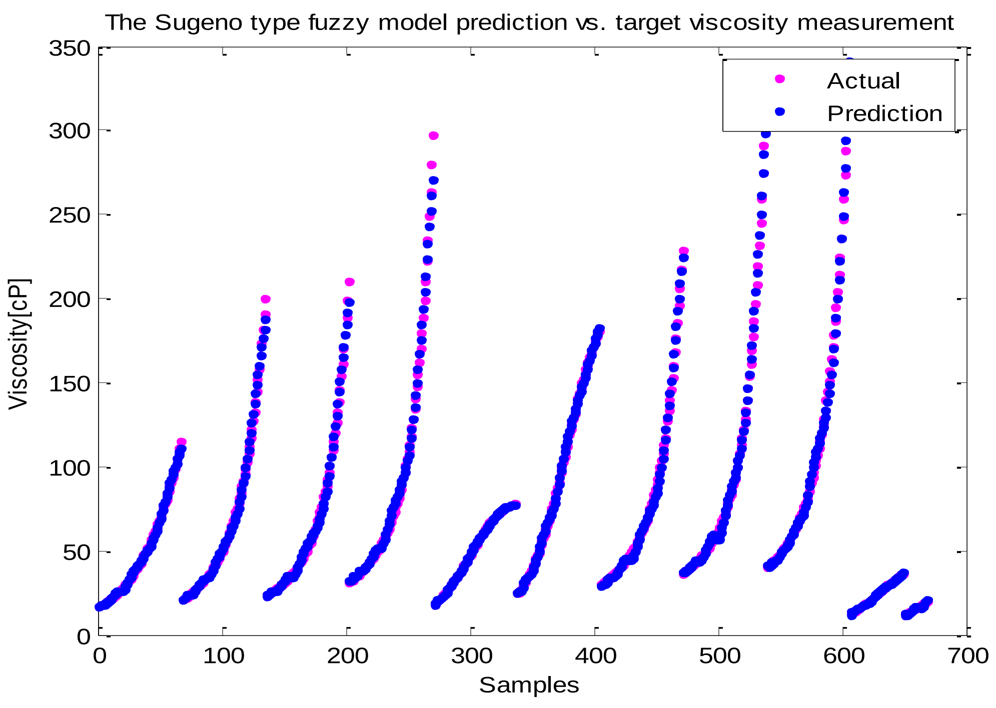


Figure 39: The target vs. prediction plot using the Sugeno type-1 fuzzy logic model developed using Matlab Fuzzy Logic toolbox with ANFIS.

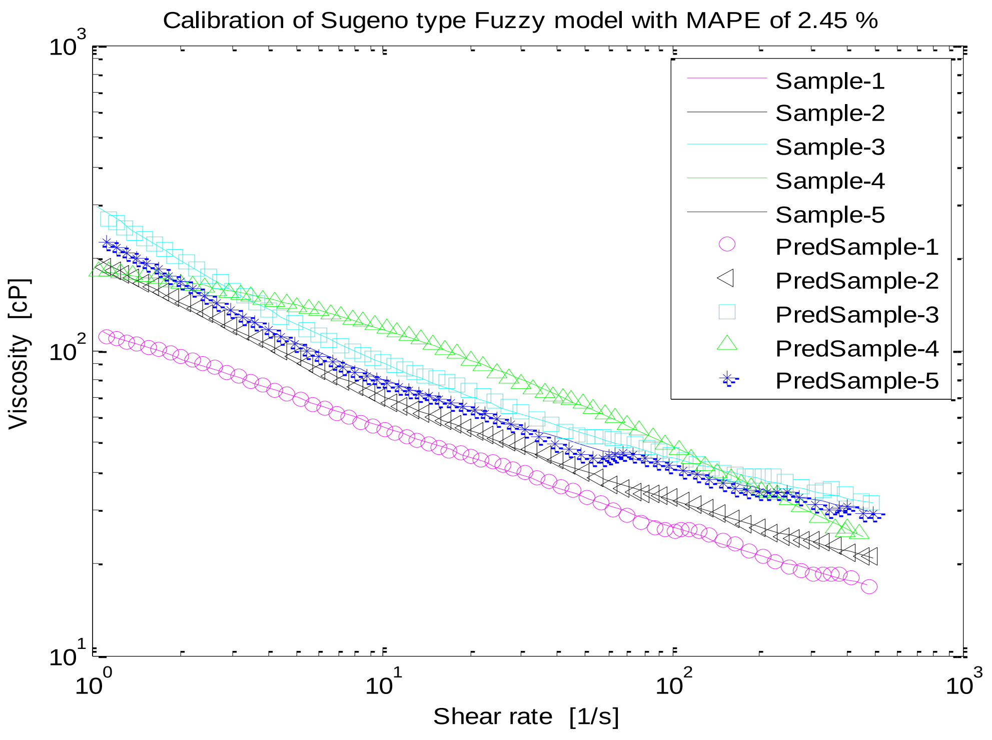


Figure 40: The calibration results of 5 different test samples using Sugeno type-1 fuzzy logic model with a MAPE of 2.45%.

5.2 ANN simulations with fluid samples

In this section, the simulation results using different types of ANN are discussed. Finally, different model based on the simulation results are implemented in the Venturi-rig for a practical implementation.

5.2.1 Feedforward ANN for simulations

Feedforward ANN in MATLAB Neural Network toolbox is used to estimate the viscosity of non-Newtonian fluid. For this, data is uploaded into the toolbox and is divided into training set (70%), test set (15%) and testing set (15%). After several simulations, the optimal number of neurons is selected as 10 and Bayesian Regularization learning algorithm had best simulation results. Figure 41 shows the architecture of finally selected feedforward ANN with 10 hidden neurons, tan-sigmoid activation function in hidden layer and linear activation function in output layer.

Figure 42 shows the performance plot using feedforward ANN. The training stops at epoch 737 with a Mean Squared Error (MSE) of 74.95 based on the minimization of adaptive weights while learning. Figure 43 and Figure 44 show the regression plot for feedforward ANN with a correlation (R) between target and output of different datasets. The correlation values for different datasets are very close to 1, meaning that target and output from the model are highly correlated to each other. Finally, an optimal model with highest possible correlation is developed from optimal tuning of neurons and learning algorithms. Thus obtained model is tested with new testing dataset. Figure 45 shows the target vs. prediction plot for the new testing data set using the optimal model. The figure shows that the model predictions are close to the required target values. Figure 46 presents the calibration results for five different samples out of 30 selected samples. The solid lines in the plot represents the target viscosity values for five different samples and the characters represents different predictions for respective samples. It can be seen that the developed static feedforward model is capable of determining the shear thinning behavior of different visco-plastic fluid samples with MAPE of 8.12%.

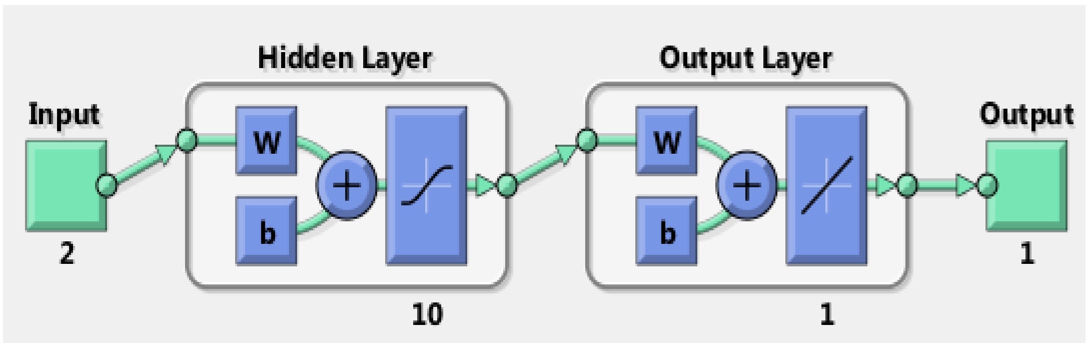


Figure 41: Feedforward setup that shows the number of inputs, hidden layers, output layer.

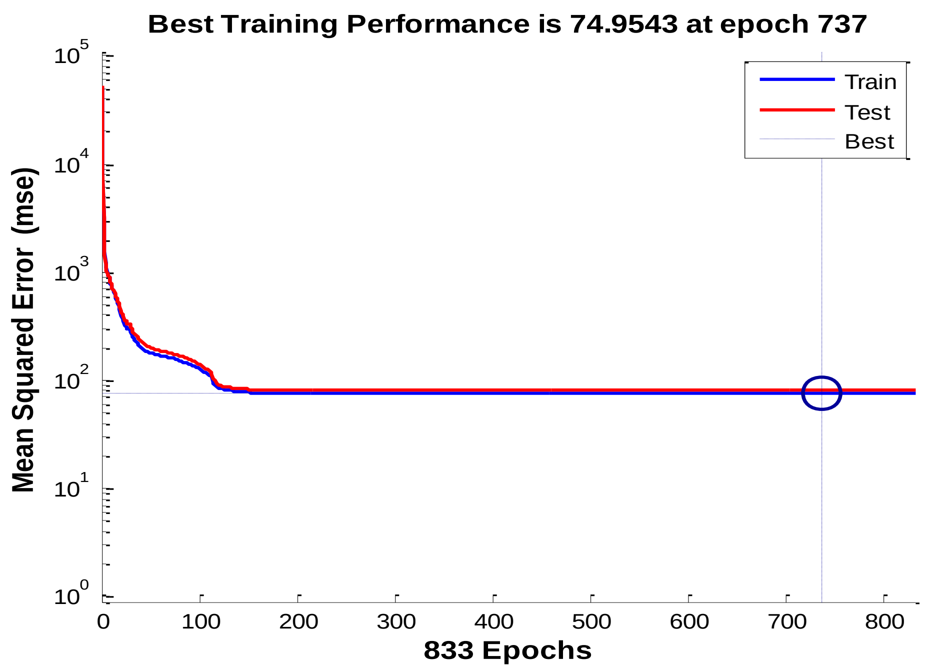


Figure 42: Performance plot of feedforward ANN, where training stops at epoch 737 with best training performance of 74.95 due to validation error check.

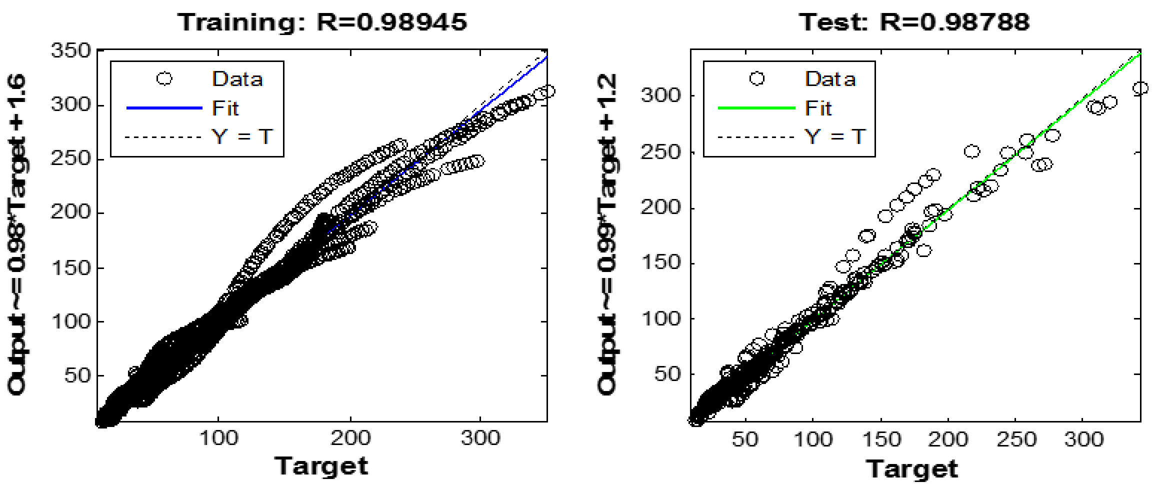


Figure 43: Regression plot for feedforward ANN with the correlation between target and output for training (R=0.98) and testing datasets (R=0.98).

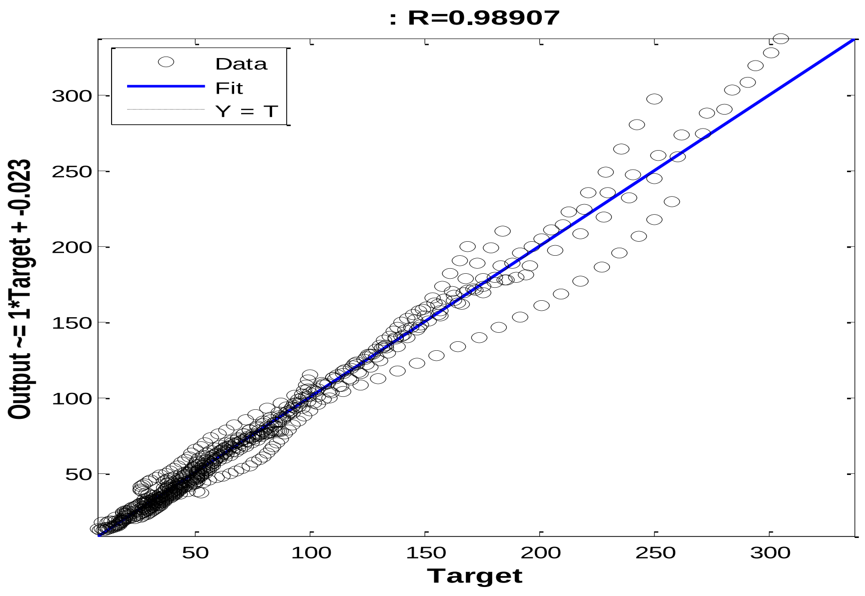


Figure 44: Regression plot for feedforward ANN with the correlation (R=0.98) between target and output for new test data.

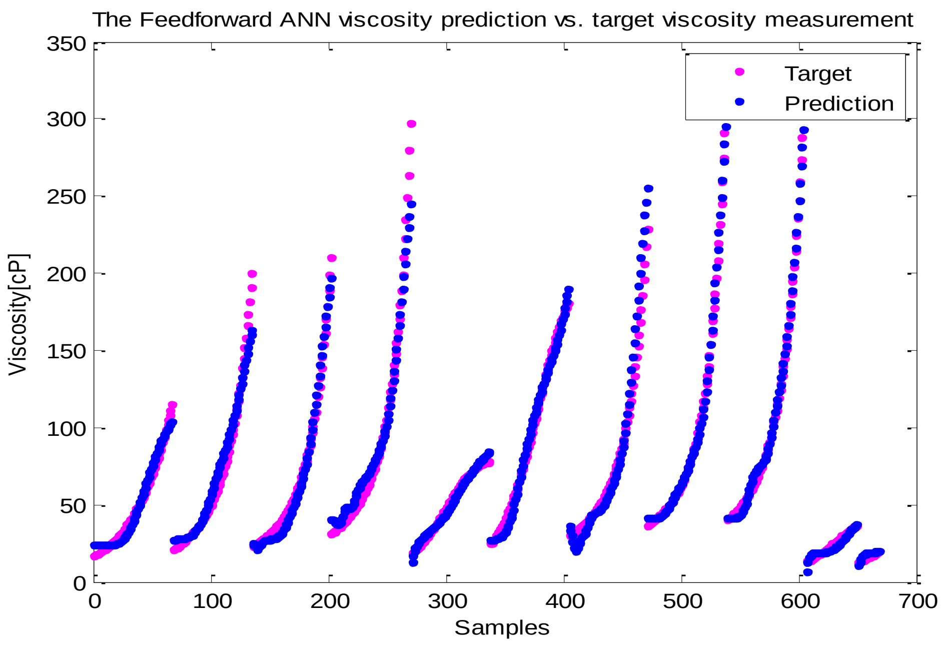


Figure 45: The target vs. prediction plot using the feedforward ANN model developed using Matlab NN toolbox.

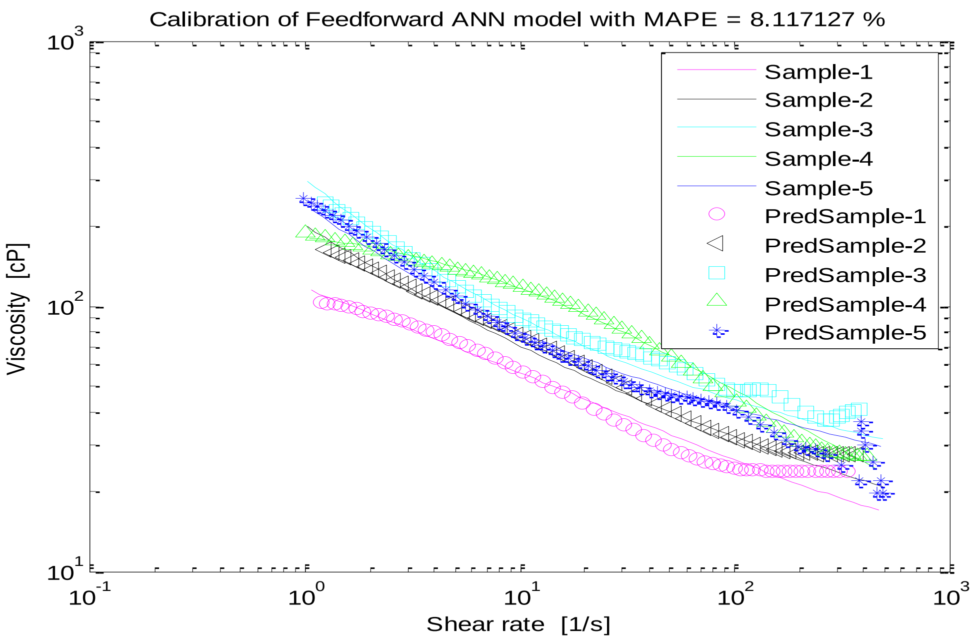


Figure 46: The calibration results of 5 different test samples using feedforward ANN with a MAPE of 8.12%.

5.2.2 Feedback ANN for simulations

The simulation study is performed to estimate viscosity of non-Newtonian drilling fluid using feedback ANN.

5.2.2.1 Partially Connected RNN for simulations

For the simulation study of partially connected RNN, dynamic time series tool in Matlab NN toolbox is used. Among three different types of partially connected RNN, Nonlinear Autoregressive with External input method is selected for the viscosity estimation. Figure 47 shows the block representation of partially connected RNN with two main inputs with delays and external input with delays. After several simulations, the optimal number of neurons is selected to be 8 and the delay in both input and output is optimally selected to be 2. The simulation results showed best results with Scaled Conjugate Gradient learning algorithm.

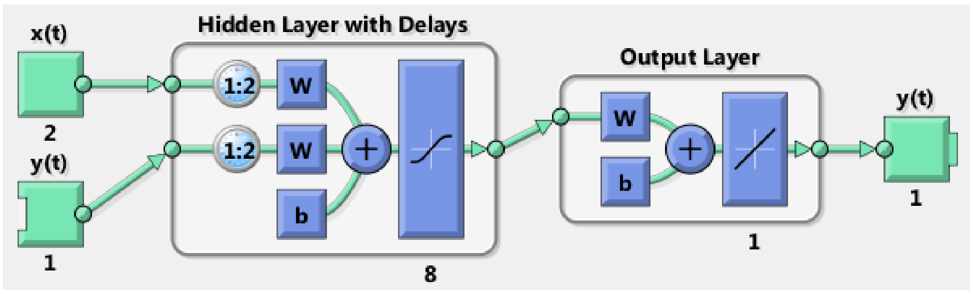


Figure 47: Partially connected RNN setup, which shows the number of inputs, hidden layers, output layer and feedback loops with delays.

Figure 48 shows the performance plot of partially connected RNN with the best validation performance of 41.775 at epoch 28. In this learning algorithm, the validation error is compared with the training error. Both validation error and training error keeps on decreasing as learning increases. In case of validation error increases though the training error decreases, algorithm will count 6 consecutive increments before it stops the training. In the Figure 48, the training is performed for 34 epochs. However, the best validation performance is taken from epoch 28 based on the validation check criterion in this learning algorithm. The validation check criterion is performed to avoid over-fitting of the model.

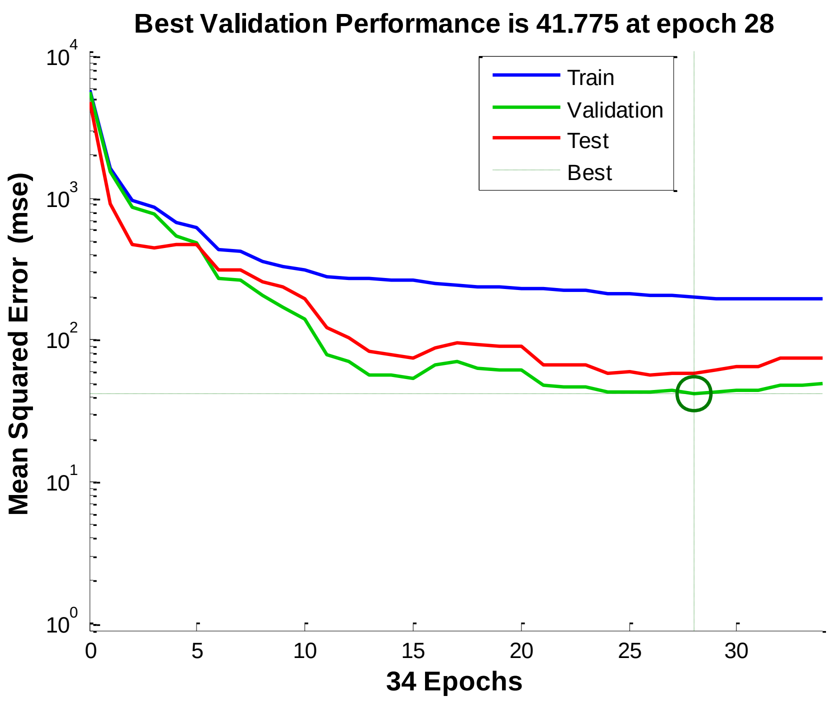


Figure 48: Performance plot of partially connected RNN, where training stops at epoch 28 with best training performance of 41.775 due to validation error check.

Figure 49 shows the regression plot of partially connected RNN for the simulation study of viscosity estimation. The regression plot shows that the target values and model predictions are highly correlated to each other. Figure 50 shows the autocorrelation error plot for partially connected RNN. It is a plot that determines the tuning of a number of neurons. For a specific number of neuron to be optimal, the correlation error bars at different lag other than zero lag, should be inside the confidence limit as indicated by red dotted lines. In zero lag, the correlation error bar must be maximum for a selected neuron to be optimal for that model.

Figure 51 shows the comparison between target values and model predictions for training set, validation set and test set. It also shows the error between target and model predictions in the error plot. In the error plot, the difference between the target and model is up to 300 units. This shows that the model is not that accurate to be reliable for viscosity estimation. This might be because the density input is constant for varying shear rates for typical fluid samples. However, the partially connected RNN is a dynamic method that needs a time varying input and output variables.

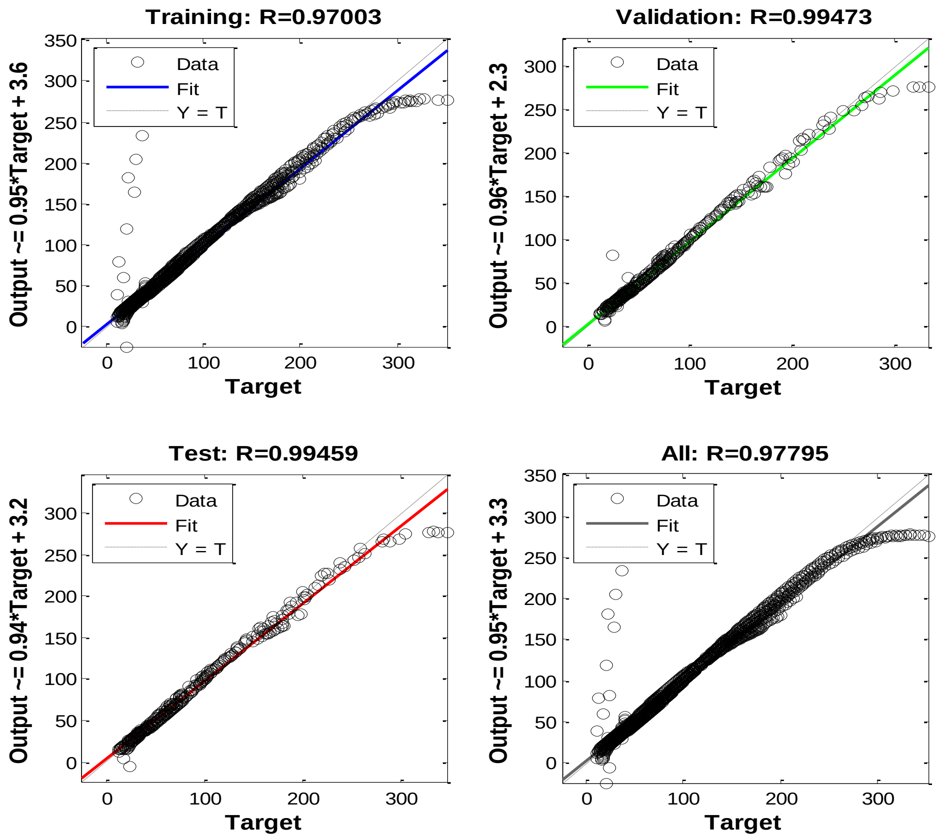


Figure 49: Regression plot for feedback RNN with the correlation between target and output for training (R=0.97), validation (R=0.99) and testing datasets (R=0.99).

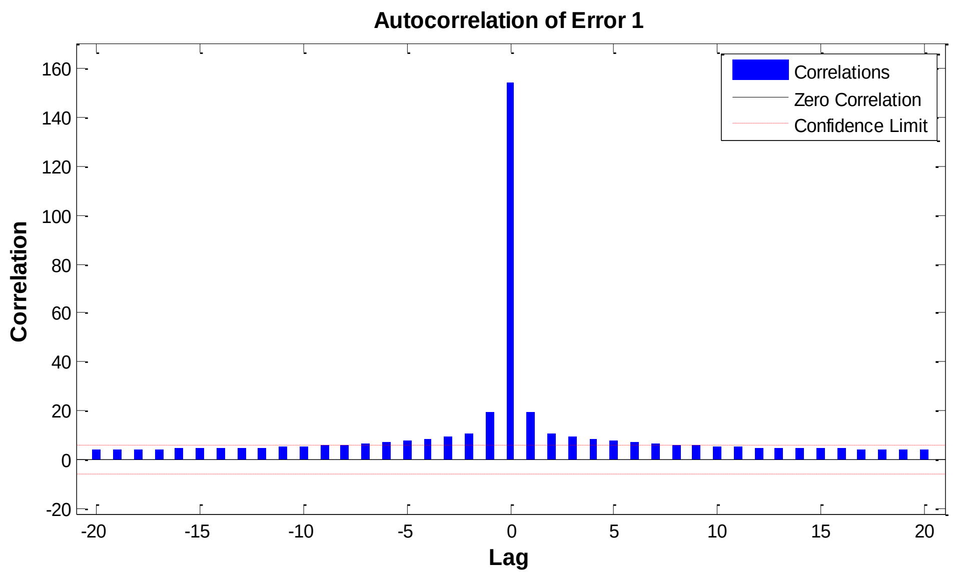


Figure 50: Autocorrelation of error with confidence limit for partially connected RNN.

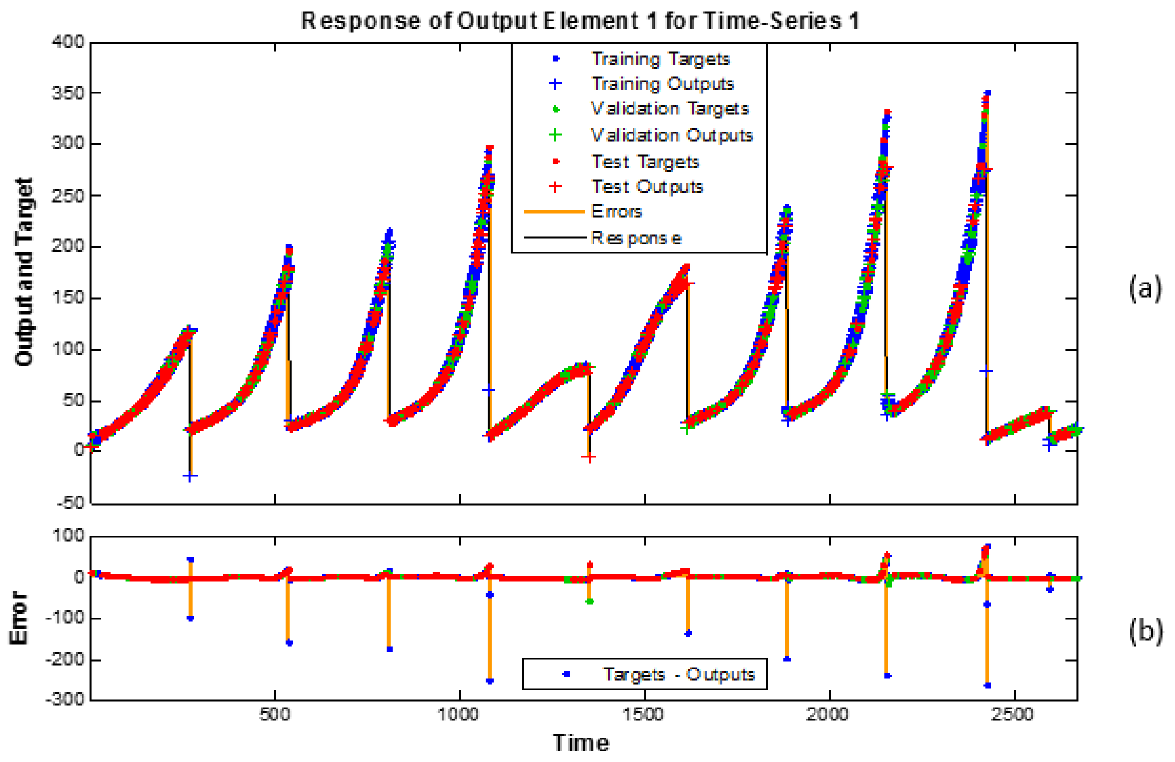


Figure 51: a) The target vs. output plot for training set, validation set and test sets in partially connected ANN. b) The error plot showing error between target and output at each samples.

5.2.2.2 Fully Connected RNN for simulations

To estimate the viscosity of non-Newtonian drilling fluid, fully connected RNN with RTRL learning algorithm in the developed toolbox [35] is used. Based on the grid search method available in the developed toolbox, the optimal parameters for viscosity estimation are found as in Table 4. Figure 52 shows the performance plot of fully connected RNN with RTRL learning algorithm. The Mean Squared Error (MSE) for the training algorithm decreases significantly to very low value. The state plot in Figure 53 shows that the state of randomly chosen five different weights of the network. The state plot shows that the weight are almost at steady state after 3000 iterations. The regression plot in Figure 54 shows that the fully connected RNN with RTRL as a learning algorithm is able to predict the target output with the correlation of 84%. The prediction plot in Figure 55 shows that the model is able to track the dynamics of non-Newtonian fluid regardless of very high MAPE. This large error in prediction can be seen in the error plot as shown in Figure 56. With fully connected RNN the highest error unit has been reduced to 80 as compared to the partially connected RNN. However, the error in the estimation is still very large and the reason might be the implementation of constant density measurement in the training dataset.

Table 4: Optimal number of parameters based on the grid search method available in the developed Matlab DANN toolbox.

Parameter	**Optimal number**
Number of epochs	3000
Learning rate	0.1
Number of previous input	1
Number of previous output	2
Number of neurons	9

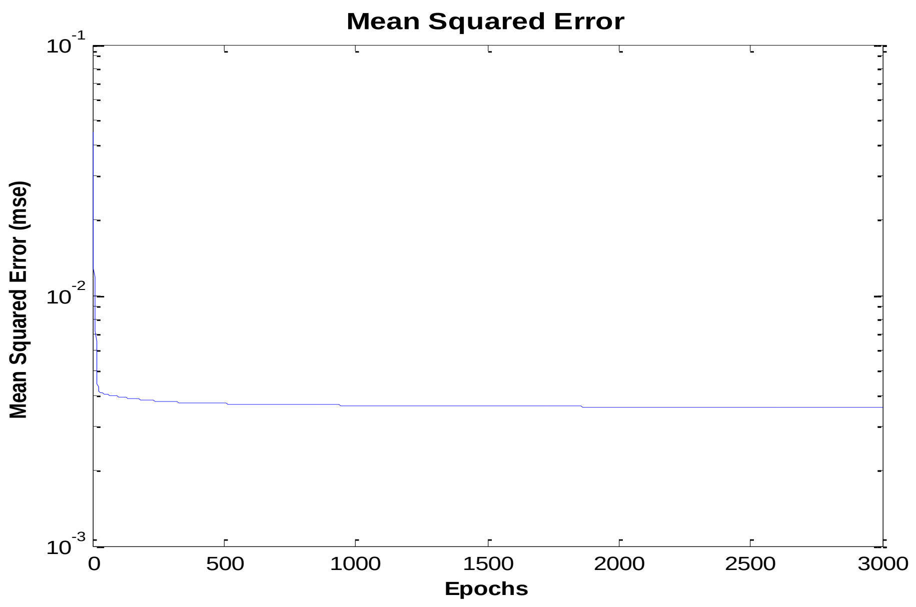


Figure 52: The performance plot for viscosity estimation using fully connected RNN with RTRL learning algorithm.

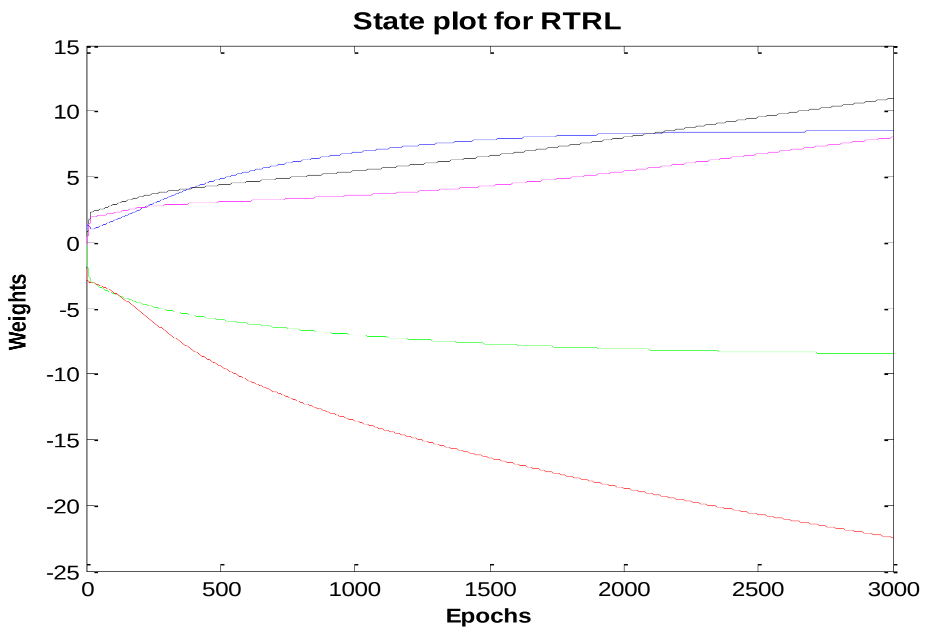


Figure 53: The state plot for viscosity estimation using fully connected RNN with RTRL learning algorithm.

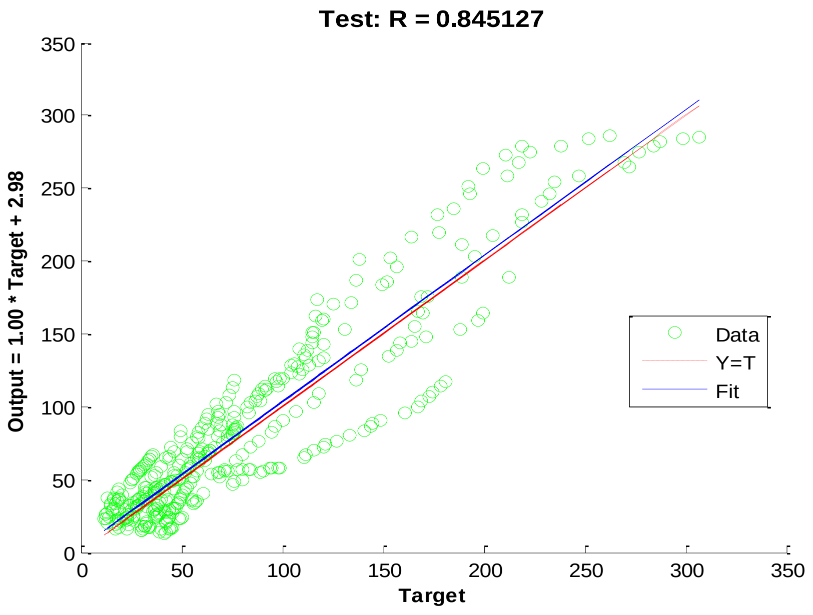


Figure 54: The regression plot for viscosity estimation using fully connected RNN with RTRL learning algorithm with 84% correlation between target values and model predictions.

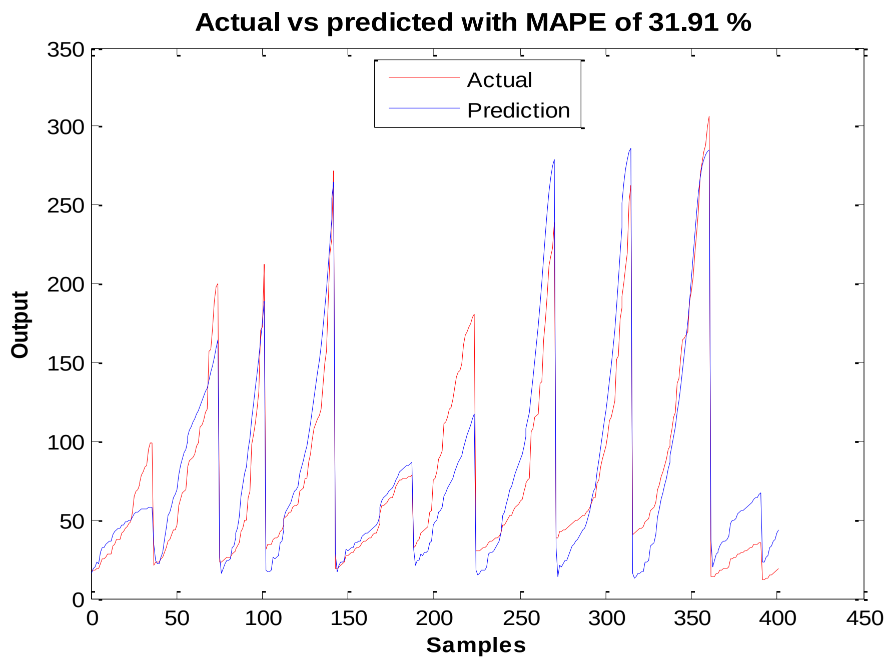


Figure 55: The prediction plot for viscosity estimation using fully connected RNN with RTRL learning algorithm with MAPE of 31.91%.

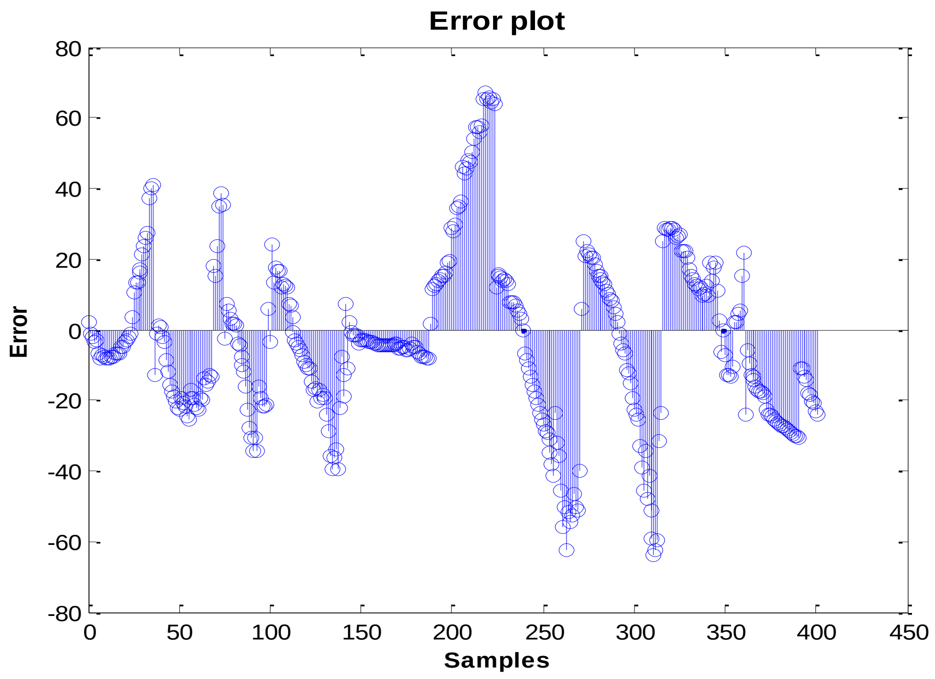


Figure 56: The error plot for viscosity estimation using fully connected RNN with RTRL learning algorithm with 80 units of highest error in the test samples.

5.3 SVM simulations with fluid samples

Support Vector Machine in regression mode as SVR is used to estimate the viscosity of non-Newtonian drilling fluid. For this estimation model, radial basis kernel function with sigma=0.02, punishment factor of C=0.53 and error toleration of ϵ=0.001 is used. The optimal tuning of these parameters are done based on the method described in [33]. The prediction plot in Figure 57 shows the comparison between the target values and the SVR model predictions. The comparison shows that the SVR model is capable of predicting the viscosity of non-Newtonian model with high accuracy. The calibration plot in Figure 58 shows the calibration results of the SVR model for 5 different test samples with MAPE of 2.70%. Based on this simulation result, the optimal SVR model is implemented in the Venturi rig for the experiment as discussed in Chapter 6.

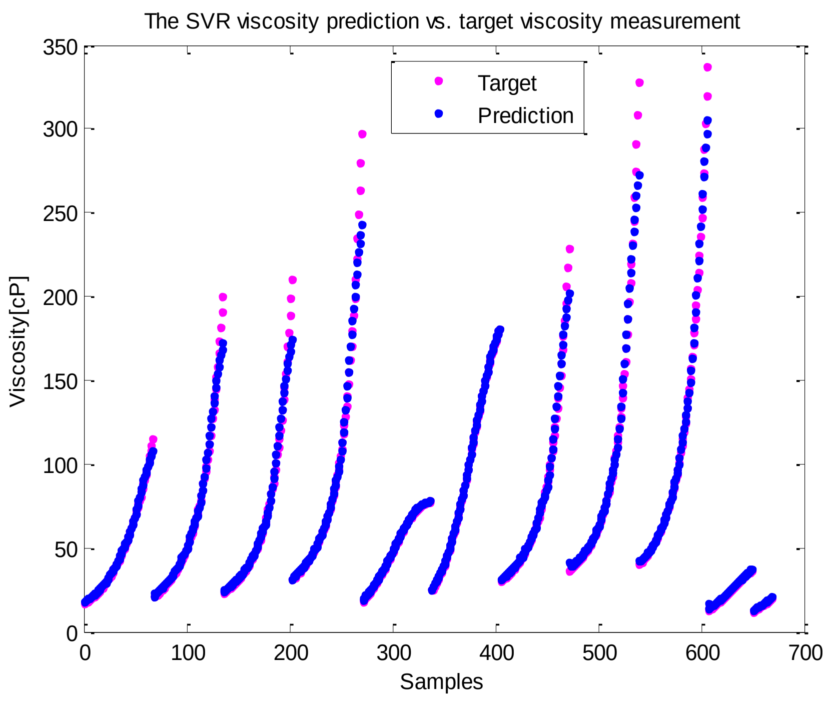


Figure 57: The target vs. prediction plot using the Support Vector Regression model.

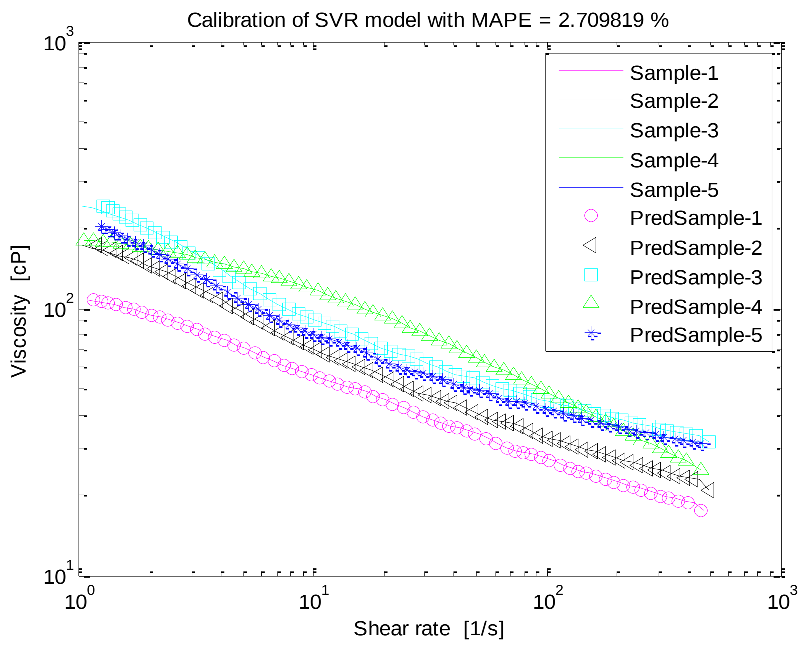


Figure 58: The calibration results of 5 different test samples using Support Vector Regression model with a MAPE of 2.70%.

5.4 Viscosities of fluid samples - classification into groups

The viscosity of drilling fluid changes while drilling and it can affect the rock cutting transportation capacity of the drilling fluid. Therefore, it is necessary to measure the viscosity accurately and classify it for the further process of additive control. In this section, different methods are used to classify the viscosity of non-Newtonian drilling fluids. For the classification, three different classes are defined as LessViscos, Viscos and HighViscos. Based on the classification, respective control action can be considered while adjusting the viscosity in additive control section near the mud tank. For example, if the current viscosity measurement falls under Viscos class, then there is no need to add any additives. Whereas, if the current viscosity measurement falls under LessViscos or HighViscos class then the mud engineer must add additives to increase and decrease the viscosity respectively.

Figure 59, Figure 60 and Figure 61 show the classification of different test samples using Sugeno type-1 fuzzy logic classifier, feedforward ANN classifier and SVM classifier respectively.

Table 5: Different classifier with the classification error.

Classifier	**Classification error**
Sugeno type-I fuzzy logic classifier	0.59% (4 out of 668)
Feedforward ANN classifier	0%
Support Vector Machine classifier	0.74% (5 out of 668)

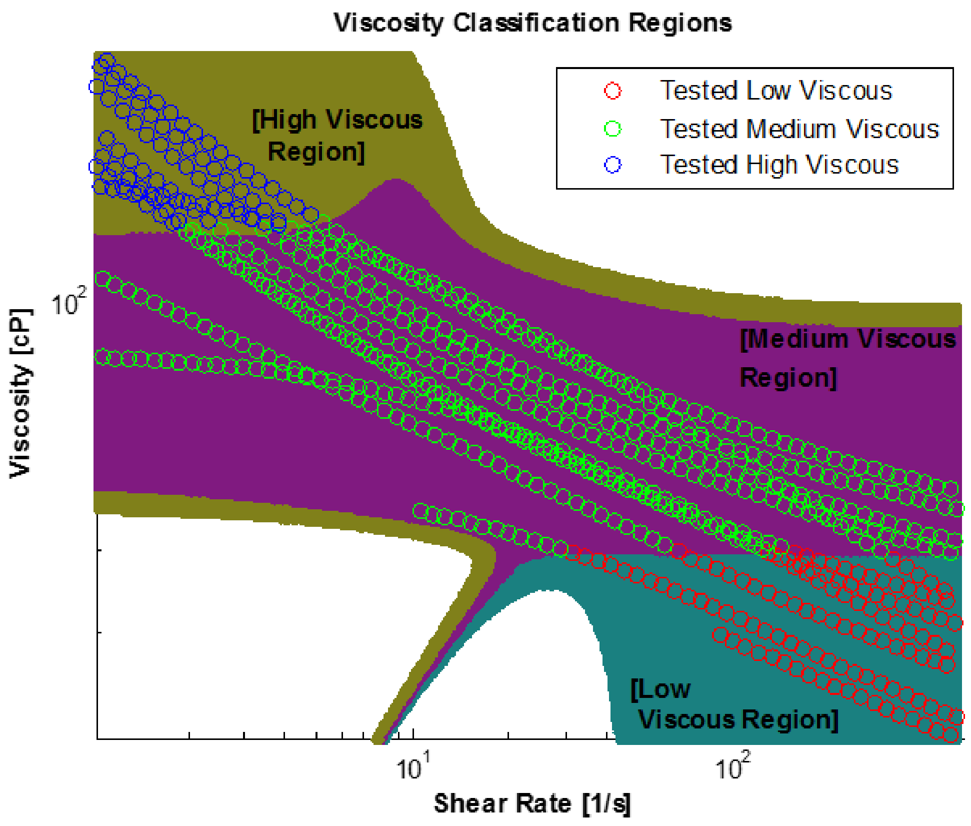


Figure 59: The classification of viscosity measurement using Sugeno type-1 fuzzy logic classifier with a misclassification percentage of 0.59%, i.e. 4 samples are misclassified out of 668 test samples.

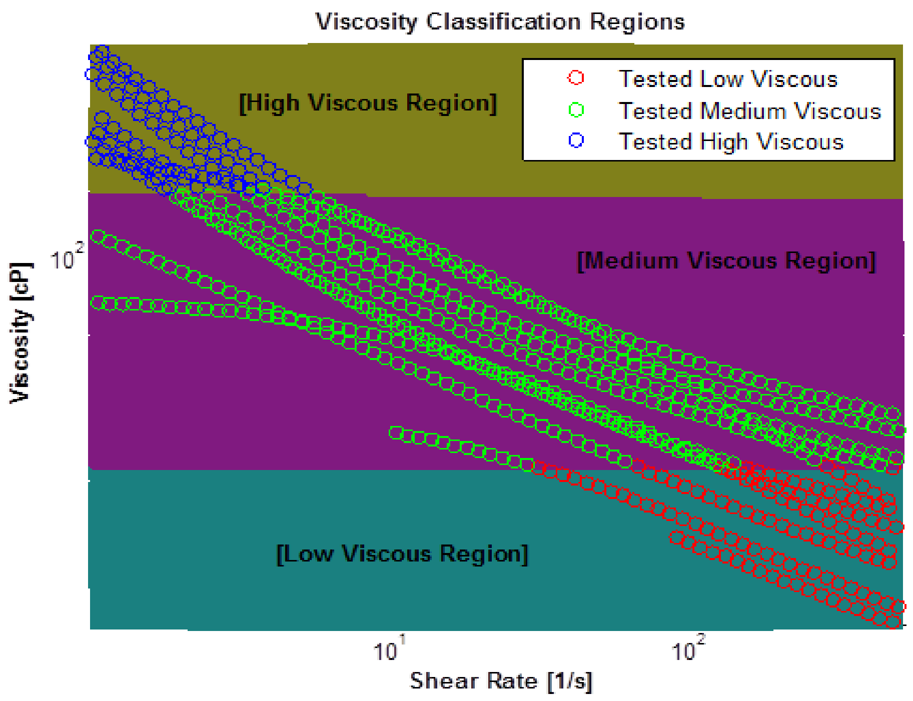


Figure 60: The classification of viscosity measurement using feedforward ANN classifier with a misclassification percentage of 0%, i.e. no samples are misclassified out of 668 test samples.

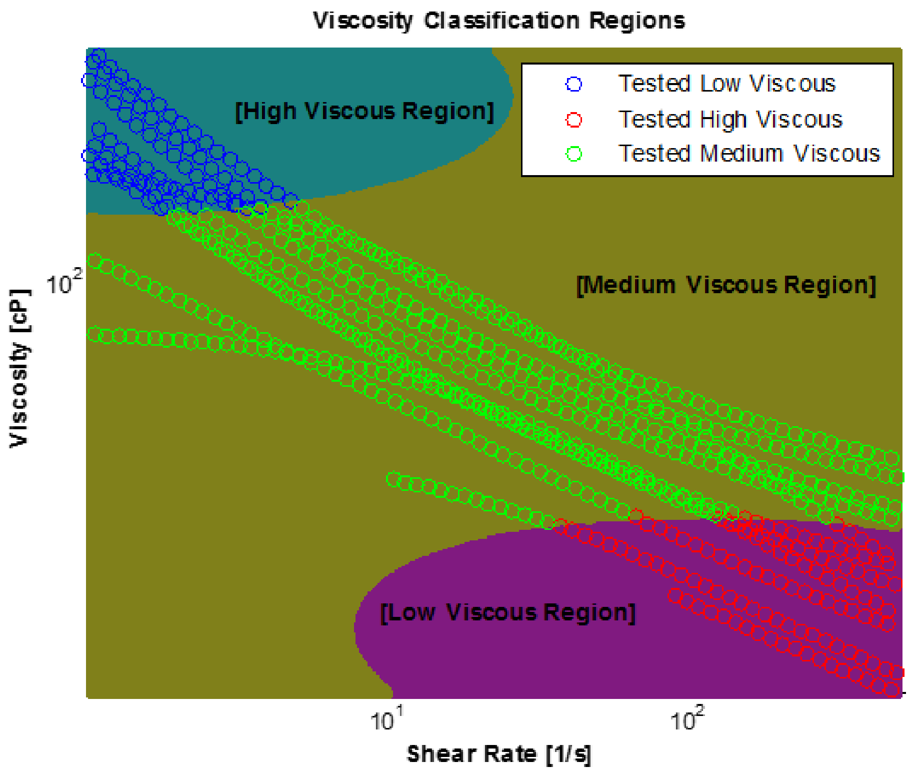


Figure 61: The classification of viscosity measurement using Support Vector Machine classifier with a misclassification percentage of 0.74%, i.e. 5 samples are misclassified out of 668 test samples.

6 Viscosity estimates using different fluids in Venturi-rig

In this chapter, different data models given in Chapter 5 are implemented in conjunction with experiments on the Venturi-rig. For the implementation, two non-Newtonian model-drilling fluids as discussed in section 3.4 are used in the experiment. All the data models used in this work has density and shear stress as inputs and viscosity as output. In Venturi-rig, it is possible to measure density using Coriolis mass flowmeter or Gamma sensor. However, the dataset used for the calibration of model assumes constant density for each fluid samples (i.e. incompressible fluid). Therefore, density is kept constant with changing flowrate for estimating viscosity of non-Newtonian model-drilling fluids. Shear stress is calculated based on the shear stress equation for incompressible non-Newtonian fluids as discussed in section 3.5. The differential pressure drop measurements required for shear stress calculation is done using differential pressure transmitter in the Venturi-rig. Figure 62 and Figure 63 shows the differential pressure measurements with different flowrates for Drilling Fluid-1 and Drilling Fluid-2 respectively. The differential pressure measurement is very unstable and fluctuates randomly. The blue dot shows the averaged differential pressure drop at respective flowrates. The error bar at each flowrate shows the standard deviation of the differential pressure drop at that flowrate. It can be seen that the standard deviations are almost same throughout the flowrate region for Drilling Fluid-1. However, the standard deviations for Drilling Fluid-2 are low at the beginning and increases as flow rate increases. The main reason for the fluctuation of differential pressure measurement is the presence of bubbles in the drilling fluids. Physically, there exist a lot of bubbles in Drilling Fluid-2 so the differential pressure drop measurement fluctuates a lot for this fluid. In the case of Drilling Fluid-1, bubbles are not a main problem for fluctuations. Apart from bubbles, the physical placement of differential pressure sensor and the vibration in the main flow pipeline affects the differential pressure measurement. The differential pressure sensor in the Venturi-rig is not placed correctly. There is mechanical bend near to the impulse line that creates a uniform disturbance to the measurement. In the flow loop, the vibration of main flowline increases as flowrate increases. This vibration partially affects the impulse line of a differential pressure sensor and thus affects the differential pressure measurement. These fluctuations in differential pressure drop measurements will eventually generates fluctuations in viscosity estimations.

Figure 64 and Figure 65 show the viscosity estimations for Drilling Fluid-1 and Drilling Fluid-2 using five different data models. In both figures, it can be seen that all the data models are able to predict the actual behavior of non-Newtonian shear thinning fluids, i.e. the viscosity estimations are decreasing as the flow rate increases. Further, it can be seen that Sugeno type-1 Fuzzy Logic model, Support Vector Regression model and feedforward Artificial Neural Network models have similar predictions. It was clear from the simulation study that these data models had very small MAPE in the predictions. Therefore, it can be concluded that these models are predicting viscosity with some accuracy. The range of viscosity predictions for Drilling Fluid-1 and Drilling Fluid-2 are [10, 50] and [30, 100] centipoise respectively. Partially connected RNN has low predictions for both fluids and the fully connected RNN has unpredictable predictions. In simulation study, these feedback models had very large MAPE and were expected to perform worse compared to other three models.

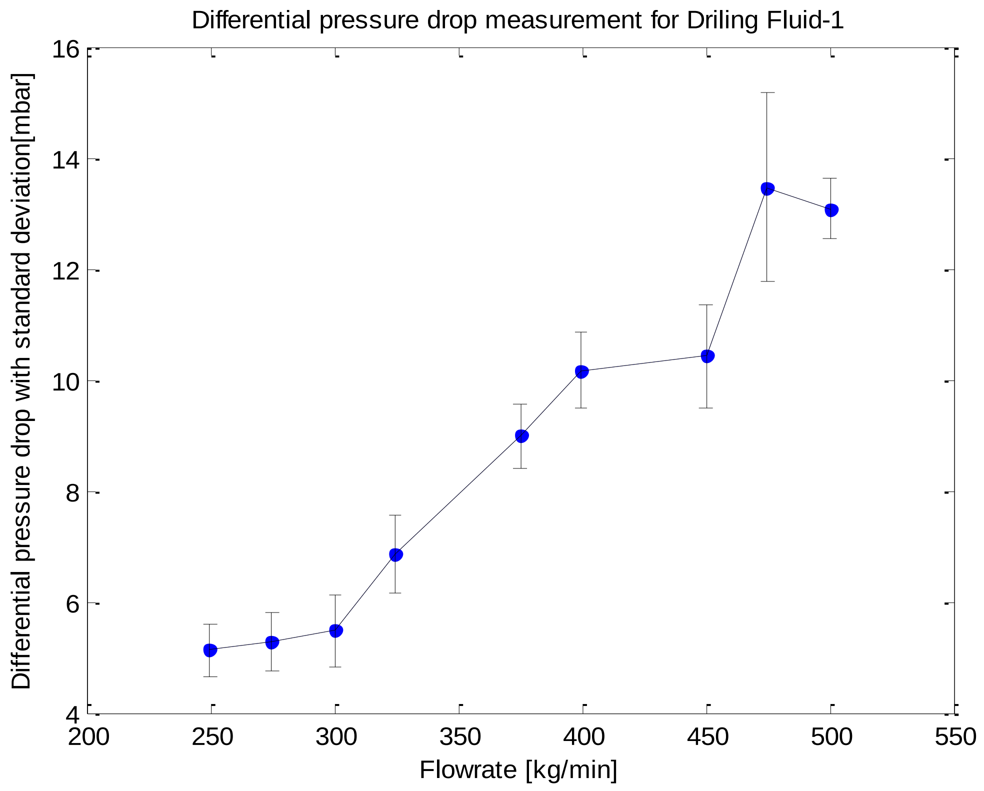


Figure 62: Averaged differential pressure drop measurements for Drilling Fluid-1 with standard deviation at each flowrates.

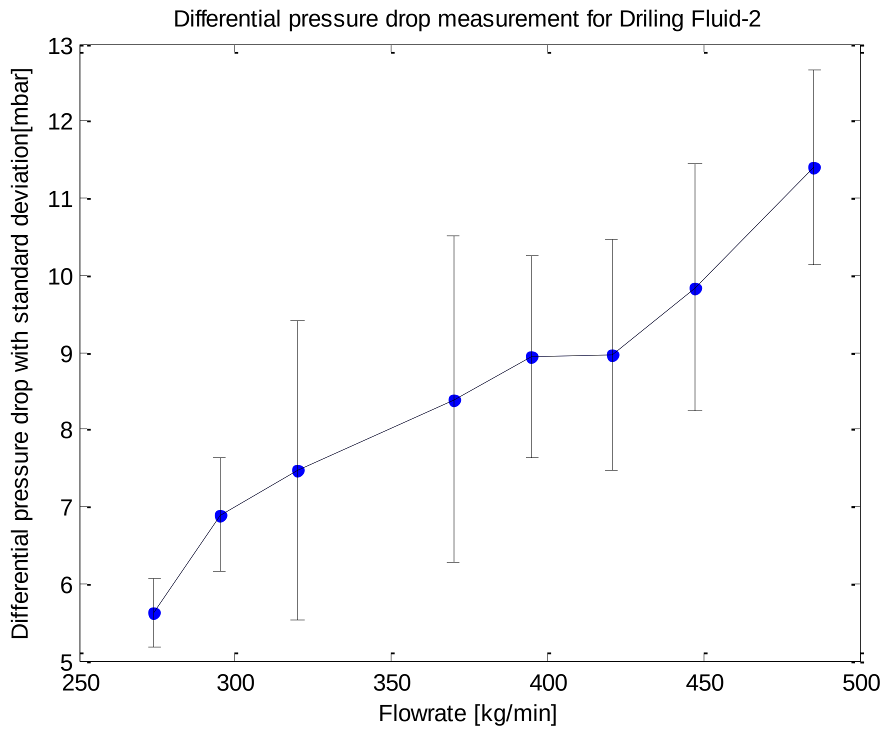


Figure 63: Averaged differential pressure drop measurements for Drilling Fluid-2 with standard deviation at each flowrates.

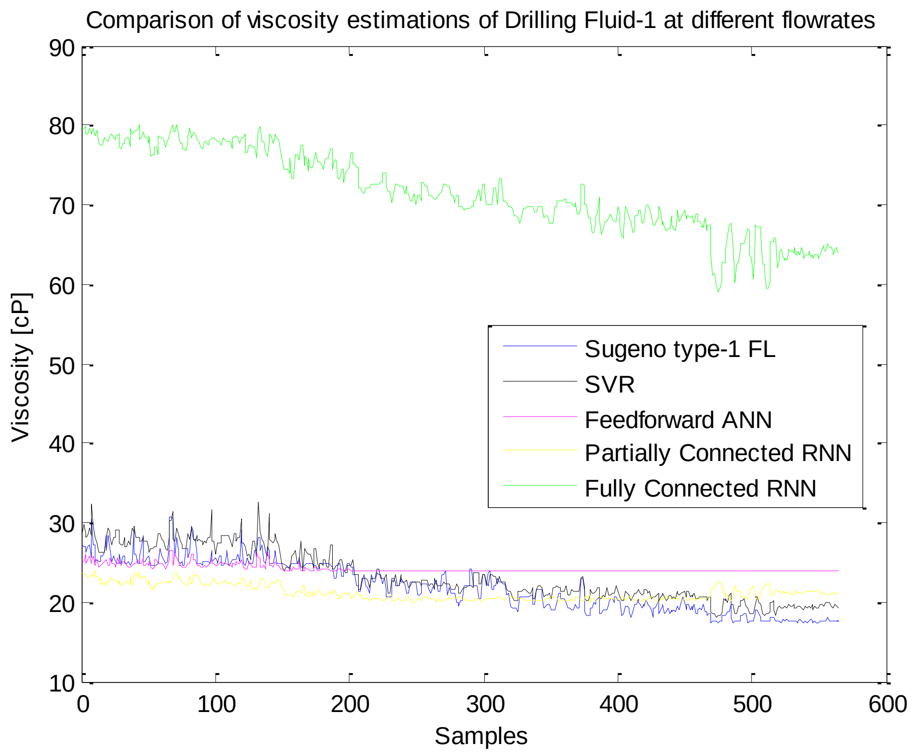


Figure 64: Comparison of viscosity estimations of Drilling Fluid-1 using different data models at different flowrates.

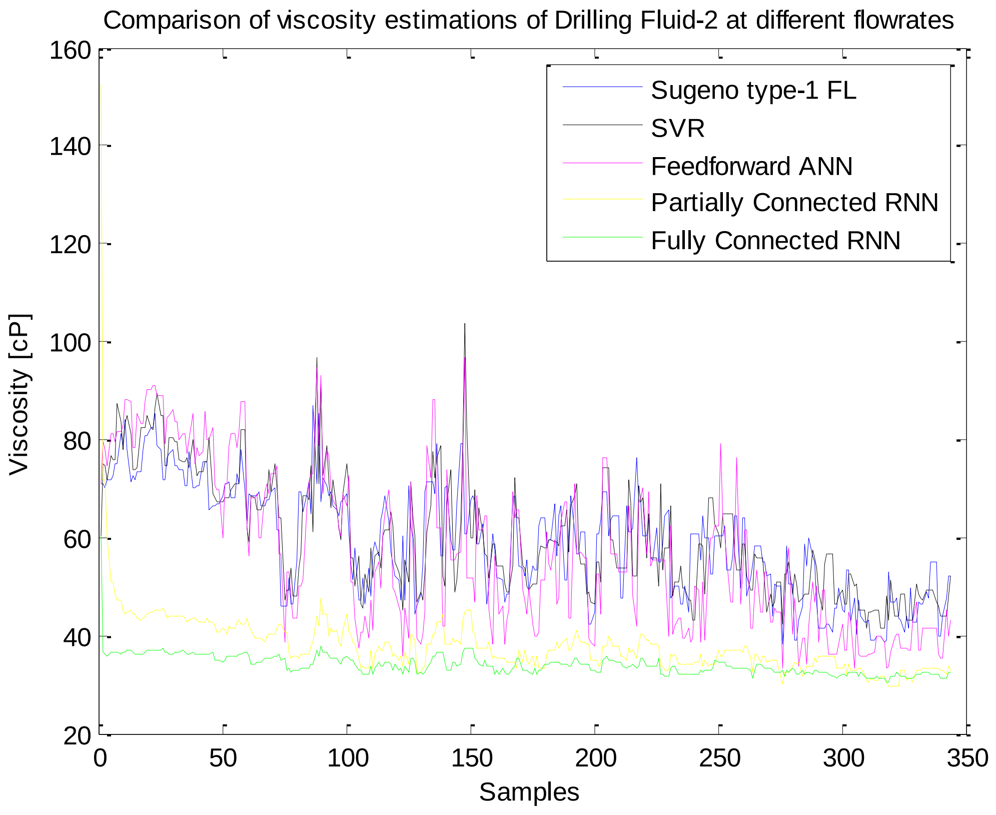


Figure 65: Comparison of viscosity estimations of Drilling Fluid-2 using different data models at different flowrates.

7 Conclusions

In a drilling operation, drilling fluid is circulated continuously in a close loop while drilling. The main functions of drilling fluid are the transportation of drilling cuttings, controlling downhole pressure, maintain wellbore stability, lubrication and cooling of the drilling bit. During circulation, the continuous monitoring of the fluid properties is very important for safe and efficient drilling operations. The two most important properties are density and viscosity of the fluid. The density is responsible for maintaining downhole pressure and wellbore stability, whereas viscosity of the fluid plays a vital role for the transportation of drilling cuttings and hole cleaning.

In general, drilling fluids are non-Newtonian in nature. In most of the drilling operation, shear thinning drilling fluids (i.e. viscosity decreases with increase in shear rate) are used. It is because; the viscosity of the fluid should be low when it is pumped down to the borehole with high flowrate and the viscosity of the same fluid should be high enough to lift the rock cuttings while flowing upward towards the ground level.

In drilling operation, the viscosity measurement is carried out using laboratory devices in continuous interval. In the field, mud engineers use Zahn Cup, March Funnel, Capillary viscometers and Rotational viscometers for viscosity measurement. In this thesis, different online viscometers found in literature for measuring viscosity of non-Newtonian fluids are discussed in Chapter 2. To point out some of them, non-invasive techniques like: Ultrasonic Doppler Velocimetry and Laser Doppler Velocimetry, Flow-viz, analytical models like: modified Power Law model for turbulent flow are discussed. Apart from viscosity measurement, literature on viscosity blending mechanism is also performed in Chapter 2. The viscosity of the fluid changes at each circulation and need to be updated to the reference value. The update is based on the current viscosity measurement and the required amount of additives. The require amount of additive is decided using viscosity blending mechanism.

In this thesis, my task is to make different empirical data models to estimate the viscosity of non-Newtonian fluids at different flowrates. The experiments are performed in the Venturi-rig, available in University College of Southeast Norway, Porsgrunn. Mainly, the Venturi-rig consists of sensors like: Coriolis mass flowmeter, Gamma sensor, pressure transmitter, pressure differential transmitter, temperature transmitter, and ultrasonic level sensors. In the empirical data, the continuous density and differential pressure measurements are used as inputs to estimate the viscosity of the model-drilling fluids. Two model-drilling fluids with different density and viscosity is circulated in a close loop and different developed empirical models are used for viscosity estimation. Different models used in this thesis are Fuzzy Logic model, Feedforward Artificial Neural Network model (ANN), Feedback Artificial Neural Network model and Support Vector Regression (SVR) model as discussed in Chapter 4.

The performance analysis of different developed models are done using simulation study and experimental study. These studies show that all the models are capable of predicting the shear thinning behavior of non-Newtonian drilling fluids. In simulation study, Sugeno type-1 Fuzzy Logic model, feedforward ANN and SVR model show very good estimation of viscosity with low value of Mean Absolute Percentage Error (MAPE) as compared to feedback ANN models. Further, Sugeno type-1 Fuzzy Logic model, feedforward ANN and Support Vector Machine models as classifiers are developed for the classification of viscosity in three different regions. The three regions of viscosity are low viscous region, medium viscous region and high viscous region. The developed classifiers are used to classify the current viscosity measurement in the correct region and help mud engineers to figure out the type of additives to be added during blending mechanism.

All the models are implemented in Venturi-rig for the estimation of viscosity of two different model-drilling fluids circulated at different flowrates. The online viscosity estimation shows that the same three models; Sugeno type-1 Fuzzy Logic, feedforward ANN and SVR models have similar predictions with some accuracy. Whereas, the two feedback ANN models have different and non-uniform viscosity predictions for two fluids.

Based on the simulation and experimental study, it can be seen that the developed empirical models are capable of estimating the viscosity of non-Newtonian drilling fluids. During simulation study, a Matlab Neural Network toolbox that can be used to simulate fully connected recurrent Neural Network is developed and able to publish an article regarding this toolbox.

Future work

The work that has been done in this thesis is only a part of viscosity estimation of non-Newtonian fluid. Due to the lack of time, there are some topics that I wished I could cover;

1. Do survey on different empirical approaches and compare them with the approaches in this thesis. Evolutionary Computing seems like an interesting empirical approach that has algorithms based on Charles Darwin evolution study. [30]
2. The more detail study on blending mechanisms, including blending techniques that are used in other industries such paint and cement.
3. Implement a blending mechanism system in the Venturi-rig so it is possible to control the viscosity and density automatically. This can be done by having one additional tank with Xanthan gum. The control system will then add Xanthan gum if the viscosity is too low and add water if the viscosity is too high.
4. If possible, get an online viscometer to measure the viscosity accurately and compare it with the empirical models. In this way, the empirical models will be more accurate and more trustworthy.
5. Improve the DANN toolbox further; make the GUI better, and implement a faster algorithm to find the optimal tuning parameter for learning algorithms used in the DANN toolbox. The current optimal method is “grid search” which goes through all possible combination and is therefore very slow.

References

1. Caenn, R., H.C.H. Darley, and G.R. Gray, *Composition and Properties of Drilling and Completion Fluids*. 6 ed. 2011, ISBN: 978-0-12-383858-2
2. Geehan, T. and A. McKee. *Drilling Mud: Monitoring and Managing It*. 2015; Available from: http://www.slb.com/~/media/Files/resources/oilfield_review/ors89/jul89/4_drilling_mud.pdf.
3. Schlumberger. *Oilfield Review Spring*. 2013; Available from: http://www.slb.com/resources/oilfield_review/~/media/Files/resources/oilfield_review/ors13/spr13/defining_fluids.ashx.
4. Sciencelearning. *Non-Newtonian fluids*. 2010; Available from: http://sciencelearn.org.nz/Science-Stories/Strange-Liquids/Non-Newtonian-fluids.
5. Collyer, A.A., *Time independent fluids*. 2016, IOPscience.
6. Collyer, A.A., *Time dependent fluids*. 2016, IOPscience.
7. Brookfield. *Dip Viscosity cups, Zahn Type*. 2016; Available from: http://www.viscometers.org/PDF/Manuals/laboratory/Zahn_Cup_M09-407.pdf.
8. Glossary, O. *Marsh Funnel*. 2016; Available from: http://www.glossary.oilfield.slb.com/Terms/m/marsh_funnel.aspx.
9. Chhabra, R.P. and J.F. Richardson, *NON-NEWTONIAN FLOW AND APPLIED RHEOLOGY*. 2 ed. 2008, ISBN: 978-0-7506-8532-0
10. Robinson, G. *What is a Capillary Viscometer*. 2016; Available from: http://www.wisegeek.com/what-is-a-capillary-viscometer.htm.
11. Elcometer. *Rotational Viscometer*. 2016; Available from: http://www.tecmos.com/carga/empresas/archivos/7dd1730c32c4bae7b973aaeeff7e9279.pdf.
12. Chhantyal, K., et al., *Estimating Viscosity of non-Newtonian Fluids using Support Vector Regression Method.* 2015.
13. Signal-processing. *Background of Ultrasonic Doppler Velocimetry*. 2016; Available from: http://www.signal-processing.com/intro_udv.html.
14. Velocimetry. *LDV - Laser Doppler Velocimetry*. 2016; Available from: http://velocimetry.net/ldv_principles.htm.
15. Carlsen, L.A. and G. Nygaard, *Utilizing Instrumented Stand Pipe for Monitoring Drilling Fluid Dynamics for Improving Automated Drilling Operations.* 2012.
16. Trinh, K.T., *The wall shear rate in non-Newtonian turbulent pipe flow.* 2010. Available from: https://arxiv.org/ftp/arxiv/papers/1009/1009.3299.pdf.
17. Wiklund, J., *Flow-Viz, A new non-invasive, in-line fluid characterization system for non-Newtonian industrial fluids*. 2016, SP Technical Research Institute of Sweden.
18. Zhmud, B., *Viscosity Blending Equations.* Lube-Tech *2014;* Available from: http://www.lube-media.com/documents/contribute/Lube-Tech093-ViscosityBlendingEquations.pdf.
19. Nelson, M.H.R.W.L., *Viscosity Blending Relationships of Heavy Petroleum Oils.* 1948.
20. Neutrium, *Estimating the viscosity of mixtures.* 2016. Available from: https://neutrium.net/fluid_flow/estimating-the-viscosity-of-mixtures/.
21. Al-Besharah, J.M., et al., *Prediction of the viscosity of lubricating oil blends.* 1989.
22. Gundersen, E., et al., *Open channel mud flow. 2013*.
23. Glittum, S., et al., *Expansion of test facility for flow measurement on drilling fluid. 2015*.
24. Berg, C., et al. *Model-based drilling fluid flow rate estimation using Venturi flume*. 2015.
25. Aplisens. *SMART PRESSURE TRANSMITTER PCE-28.SMART*. 2015; Available from: http://www.aplisens.com/dodatkowe_aplikacje_advertnet/pdf/produkty/PC-28Smart.pdf.
26. Aplisens. *SMART DIFFERENTIAL PRESSURE TRANSMITTER APRE-2000*. 2015; Available from: http://www.aplisens.com/dodatkowe_aplikacje_advertnet/pdf/produkty/APR-2000.pdf.
27. Hauser, E. *RTD Thermometer omnigrad TST41N*. 2015; Available from: https://portal.endress.com/wa001/dla/5000000/5566/000/00/TI232ten_1198.pdf.

28. Rosemount. *Rosemount Ultrasonic 3107 Level and 3108 Flow Transmitters*. 2015; Available from: http://www2.emersonprocess.com/siteadmincenter/pm%20rosemount%20documents/00825-0200-4840.pdf.
29. S-TEC. *Installation, Operation & Maintenance Manual Density Transmitter DT-9300*. 2016.
30. Adeli, N.S.H., *COMPUTATIONAL INTELLIGENE, Synergies of fuzzy logic, neural networks and evolutionary computing*. 2013, ISBN: 9781118337844
31. Kumbasar, A.T.T., *An Open Source Matlab/Simulink Toolbox for Interval Type-2 Fuzzy Logic Systems.* 2015.
32. Shiffman, D. *Neural Networks*. 2016; Available from: http://natureofcode.com/book/chapter-10-neural-networks/.
33. Chhantyal, K., et al., *Ultrasonic Level Sensors for Flowmetering of non-Newtonian Fluids in Open Venturi Channels.* 2016.
34. Demuth, H. and B. Beale. *Neural Network Toolbox User's Guide Copyright*. 2004; Available from: http://www.image.ece.ntua.gr/courses_static/nn/matlab/nnet.pdf.
35. Chhantyal, K., et al., *Dynamic Artificial Neural Network (DANN) MATLAB Toolbox for Time Series Analysis and Prediction.* 2016.
36. Haykin, S., *Neural Networks and Learning Machines.* 2009.

Appendix A: Project Abstract and Project Description

University College of Southeast Norway

MASTER'S THESIS, COURSE CODE FMH606

Student:	**Minh Hoang**		
Thesis title:	**Tuning of viscosity and density of non-Newtonian fluids through mixing process using multimodal sensors, sensor fusion and models**		
Signature:	. . . .		
Number of pages:	156		
Keywords:	Non-Newtonian Drilling fluid, Viscosity blending mechanism, Venturi-rig, Time-series measurements, Fuzzy Logic, Neural Network, Support Vector Machine		
Supervisor:	Saba Mylvaganam	Sign.:	. . . .
2nd supervisor:	Khim Chhantyal	Sign.:	. . . .
3nd supervisor:	Håkon Viumdal	Sign.:	. . . .
Censor:		Sign.:	. . . .
External partner:	Geir Elseth	Sign.:	. . . .
Availability:	Open		

Archive approval (supervisor signature)**:** Sign: . . . . **Date:** . . . .

Abstract:

The different models that were used to estimate the viscosity in this thesis were Fuzzy Logic model, Feedforward Artificial Neural Network model (ANN), Feedback Artificial Neural Network model and Support Vector Regression (SVR). The performance analysis of these models were done using simulation study and experimental study. Based on the simulation study, Sugeno type-1 Fuzzy Logic model, feedforward ANN model and SVR gives very good estimations compared to the feedback ANN models. For the experimental study, the experiments were done in the Venturi-rig in University College of Southeast Norway, Porsgrunn. The results were very similar to the simulation results, where the three models; Sugeno type-1 Fuzzy Logic, feedforward ANN and SVR had comparable predictions with some accuracy. Based on the analysis from simulation and experimental study, it seems that the empirical models that were developed is capable of estimating the viscosity of non-Newtonian drilling fluids.

HSN University College of Southeast Norway

Campus Porsgrunn/Faculty of Technology
Department of Electrical Engineering, IT and Cybernetics

FMH606 Master's Thesis

Title: Tuning of viscosity and density of non-Newtonian fluids through mixing process using multimodal sensors, sensor fusion and models

TUC supervisor: Saba Mylvaganam (main-supervisor), Khim Chhantyal (co-supervisor) and Håkon Viumdal (co-supervisor)

External partner: Geir Elseth, Statoil

Task background:

Oil and gas are natural fuels that are obtained through a series of drilling operations. Oil companies are finding it challenging to handle the current increasing drilling costs. The challenge of maintaining the correct downhole pressure during drilling operations is a major cost driving parameter. At the same time, the proper hole cleaning is essential for efficient drilling operations and for optimizing oil and gas production. Development of new sensor systems and control algorithms can reduce drilling costs and increase production.
Density and viscosity of drilling fluid (mud) have major impact on the operational efficiency with respect to cost and production. Density is measured and adjusted for maintaining the downhole pressure, whereas viscosity is primarily measured and adjusted for hole cleaning.
This study is involved in the monitoring and tuning of density and viscosity of a fluid to be used in laboratories emulating the characteristics of drilling fluid used in the oil and gas industries.

Task description:

(1) Brief literature survey on drilling operations, with emphasis on circulation of drilling fluid.
(2) Brief literature survey on the non-Newtonian rheology of the drilling fluid (mud).
(3) Literature survey on viscosity measurement of drilling fluid (both Newtonian and non-Newtonian) in different flow conditions (laminar or turbulent).
(4) Brief literature survey on viscosity blending mechanism.
(5) Identify all the measurands in the Venturi-rig and characterize all the meters with respective specifications.
(6) Perform time-series measurements of flow rate, density and viscosity for different drilling fluids in the Venturi-rig.

(7) Design empirical models to estimate the viscosity of non-Newtonian drilling fluids based on the sampled data from the experiments. Different approaches as fuzzy logic, neural network and support vector regression can be used.
(8) Analyse and compare the results of the empirical models.
(9) Design a classifier based on neural network (or support vector machine) to classify the viscosity of the non-Newtonian fluid at different flow rates.
(10) Submit a report using the template of UCSEN with systematically archived data sets and software.

Student category:

SCE students, preferably the candidate from the Master Project group SIV-13-15.

Practical arrangements:

University College of South East Norway has an open Venturi-rig as a part of mud flow loop in the process hall. There are three different types of water-based drilling fluids for circulation. For laboratory measurement, Anton Paar DMA 4500 ME density measuring module (in UCSEN) and Anton Paar MCR rheometer (in Statoil) can be used. Some models describing the density and viscosity variations of drilling fluids are available for testing.

Signatures:
Student (date and signature): [illegible] 01.02.2016
Supervisor (date and signature): [illegible] 01.02.2016

Appendix B: Publication of Flowrate Estimating Using Ultrasonic Level

Flowrate Estimation using Ultrasonic Level Sensors using Dynamic Artificial Neural Networks with Real Time Recurrent Learning – A Comparative Study of Models and Practical Implementation

Khim Chhantyal, Håkon Viumdal, Minh Hoang, Saba Mylvaganam,
University College of Southeast Norway
Faculty of Technology
Kjølnes Ring 56, 3918 Porsgrunn, Norway

Geir Elseth
Statoil
Hydrovegen 55, Porsgrunn, Norway

***Abstract* – Accurate estimation of flow in drilling operations at inflow and outflow positions can help to increase safety, to optimize production and help to save money and man-hours, as unnecessary troubleshooting costs at the drilling site can be avoided. In this paper, Dynamic Artificial Neural Network (DANN) is used to estimate the flow rate of non-Newtonian drilling fluids in an open channel Venturi-rig that can be used for outflow measurements while determining delta flow, i.e. the difference between the flow rates into and out of the well. This paper presents a simple flow estimation method using three appropriately positioned transducers above the Venturi channel normally available on drilling platforms. The paper addresses simulation and experimental studies. Simulation study looks into fully connected Recurrent Neural Network (RNN) with three different learning algorithms: Back Propagation Through Time (BPTT), Real-Time Recurrent Learning (RTRL) and Extended Kalman Filter (EKF). The simulation results show that BPTT and EKF algorithms converge very quickly as compared to RTRL. However, RTRL gives results that are more accurate, is less complex and computationally fastest among these three algorithms. Hence, in the experimental study RTRL is chosen as the learning algorithm for implementing Dynamic Artificial Neural Network (DANN) for usage in the Venturi-rig based data fusion. In the Venturi-rig, DANN with RTRL learning algorithm is compared with previously developed Support Vector Regression (SVR) and static ANN models to assess their performance in estimating flow rates. The comparisons show that the proposed DANN is a most accurate model among three models as it uses previous inputs and outputs for the estimation of current output.**

Keywords— Drilling operations, open channel Venturi flume, non-Newtonian fluid, flow rate estimation, ultrasonic level measurements, Recurrent Neural Network, Real-Time Recurrent Learning

List of symbols and abbreviations

Symbol	Quantity
ANN	Artificial Neural Network
BPTT	Back Propagation Through Time
CFD	Computational Fluid Dynamics
DANN	Dynamic Artificial Neural Network
EKF	Extended Kalman Filter
n	Number of folds
LT	Level Transmitter
MAPE	Mean Absolute Percentage Error
MSE	Mean Squared Error
N	Number of neurons
O	Order
P_b	Bottom hole pressure
P_f	Formation pressure
P_{ff}	Formation fracture pressure
RNN	Recurrent Neural Network
RTRL	Real Time Recurrent Learning
SVR	Support Vector Regression
t	time

I. Introduction

In drilling operations, the drilling mud is circulated in a closed loop starting from the mud tank into the wellbore and back to the mud tank. The mud can be water-based, oil-based or gas-based and is circulated during the drilling operation, until the desired depth is reached. During circulation, the properties of drilling mud have significant importance for the safe and efficient drilling operation. The viscosity, density, and flow rate of circulating mud play a vital role, in all the drilling operations, [1].

In general, drilling muds are non-Newtonian in nature, and the viscosity of the mud along with other rheological properties govern the transport of rock cuttings while drilling, [1].

The density or mud weight is mainly responsible for maintaining the pressure in the wellbore. Depending on the types of the drilling operation and the reservoir, the wellbore pressure or bottom-hole pressure (P_b) is limited within the pressure window given by formation pressure (P_f) and formation fracture pressure (P_{ff}). If the wellbore pressure is less than the formation pressure ($P_b < P_f$), the formation gasses and fluids will flow into the drilling mud, and is called "*kick*". The occurrences of kick should be detected as early as possible during drilling operations. If the early kick detection is ignored or is not detected, it can lead to problems in maintaining the density of the mud and in the extreme case, it can result in blow-out of hydrocarbons on the rig, e.g. the Deepwater Horizon explosion, [2]. In the case of ($P_b > P_f$), the high pressure circulating fluids may enter the formation pores, causing fluid losses. If the wellbore pressure is further increased, beyond the formation fracture pressure ($P_b > P_{ff}$), the circulation fluid can fracture the formation and cause an increased fluid loss, often called *"lost circulation"*. The fluid loss will decrease the volume of the mud in the circulation loop and in the mud tank, and will affect the production, [1].

A similar situation occurs frequently in geothermal drilling. In geothermal drilling, one of the costly problems is lost circulation. that occurs when drilling fluid is lost to the formation rather than returning to the surface, preferably intact. The management of lost circulation is important and requires the accurate measurement of drilling fluid flow rate both into and out of the well.

Reliable detection of unusual conditions can allow the use of low weight mud, efficient drilling, less formation damage, and lead to lower drilling costs. Delta flow method, i.e. the difference between flows at inflow and outflow points of the circulation mud, is one of the best methods to detect kick and fluid loss, which uses the flow measurements before and after the wellbore, [3-8]. The difference in outflow and inflow measurements can be used as an indication of unusual conditions while drilling. If the flow rate before wellbore is less than the flow rate in the return line, then it can be considered as an indication of early kick detection. Whereas, if the inflow is greater than the outflow, it is an early indication of fluid loss. In addition, the flow rate of circulating fluid will determine the transportation of rock cuttings. The flow velocity of the circulation mud is often maintained higher than the settling velocity of the rock cuttings for efficient transportation of cuttings. In addition to the delta flow method, other methods of early kick detection are discussed in [8-12].

In literature [3-8], there are different systems for measuring delta flow. For inflow measurement, conventional pump stroke counter, rotatory pump speed counter, magnetic flow meter, Doppler ultrasonic flow meter or Coriolis mass flowmeter can be used. For outflow measurement, magnetic flowmeter, Doppler ultrasonic-based flowmeter, standard paddle meter**,** ultrasonic level meter, a prototype rolling float meter or open channel Venturi flowmeter can be used. The scenario of inflow and outflow measurement is completely different. For example, the inflow measurement can be carried out using Coriolis mass flow meter, more accurate but an expensive flowmeter. However, Coriolis mass flow meter is not suitable for outflow measurements as the returning mud contains solid rock cuttings, other formation particles, formation fluids and gases. An overview of different flowmeters based on reliability and accuracy is given in [7]. Based on the analysis in [7], magnetic flowmeter or Doppler ultrasonic flowmeter are suitable for inflow measurements and prototype rolling float meters for outflow measurement. Speers and Gehrig, [4] presents the implementation of delta flow method by using magnetic flowmeters at inflow and outflow locations. The magnetic flowmeter is limited in applications to conductive fluids or to only water-based muds. In addition, magnetic flowmeters need some additional U-tube design in the return section. For lower flow velocity of circulating fluids, the rock cuttings will settle at the bottom of this U-tube. These problems are avoided in open channel return line, in which efficient rock cutting transportation and their easier separation from mud, [5-6].

This paper presents the outflow measurement based on open channel flow with a Venturi section. In an open channel flow, the upstream pressure relative to a reference level in the 'control section' of the loop structure can be used to estimate the flow rate, [13]. The control section used in the flow loop is the Venturi flume. The flow measurement is based on an extension of the application of the well-known Venturi principle, to flow of fluid in an open channel, [14]. The constriction in the Venturi section results in the transition of flow from subcritical to supercritical flow in the vicinity of the throat, [15]. For sufficiently long throat, the critical condition occurs in the throat, giving the critical depth [16]. The level of the fluid in upstream is measured as the critical depth is identified. The level can be measured using ultrasonic or RADAR level sensors and flow rate can be calculated as a function of measured level.

To study the possibility of using Venturi flume in estimating flow rate, a flow loop (i.e. Venturi rig) is available in University College of Southeast Norway (USN), Porsgrunn, Norway. For this Venturi rig, the CFD simulation study of open channel flow measurement is investigated in [17]. The numerical algorithm using Saint Venant equation is presented in [18-19]. However, the developed numerical model is not applicable for real-time monitoring and controlling purpose due to the high computational cost. The study presented in [20] shows the successful implementation of static Artificial Neural Network (ANN) and Support Vector Regression (SVR) techniques for flow measurement in the test loop. The present study is a continuation, where, Dynamic Artificial Neural Networks (DANN) are investigated and implemented in the software used in running, monitoring and controlling the flow loop.

In the following sections, the simulation study of fully connected Recurrent Neural Network (RNN) with three different learning algorithms for estimating the flow rate of the non-Newtonian liquids is presented. Finally, the experimental results of flow rate estimation using RNN, ANN and SVR are discussed.

II. Dynamic Artificial Neural Network

ANN can be of the static or dynamic type. Static ANN or feedforward ANN type uses current inputs and current outputs whereas, DANN uses current and previous inputs and outputs for modeling purpose. Further, DANN can be partially connected RNN or fully connected RNN based on the feedback loops. Fully connected RNNs have self-feedback loops, and partially connected RNNs does not have self-feedback loops, [21].

The delta flow measurement discussed in Section I is a dynamic problem, where the previous information about the kick detection and fluid loss is important for the current measurement. Therefore, fully connected RNN is used for modeling, the estimation of the flow rate being based on the level measurements in the open channel Venturi flow loop. For the estimation of the flow rate, three different learning algorithms are used. These algorithms are presented here briefly.

A. Back Propagation Through Time (BPTT)

BPTT is an extension of gradient-based back propagation algorithm that is used in static ANN. The idea in BPTT is to unfold the RNN architecture into feedforward ANN architecture in an arbitrary number of time steps or folds. These folds make the error to propagate even further in time, so it is called back propagation through time. However, the number of folds are usually low to avoid deep network and this approach is called is often called truncated BPTT. In general, recurrent weights are simply duplicated over the folds while unfolding, [22]. The basic BPTT architecture is shown in Fig. . The computational complexity of BPTT is of order O(N^2) and the storage requirement is of order O(nN^2), where N being number of neurons and n is the arbitrary number of folds. The drawbacks of BPTT are; it is an offline learning algorithm and requires large memory to store state information at different folds, [23].

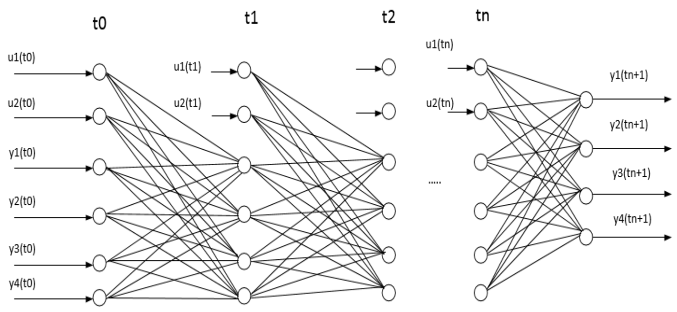


Fig. 1: A general architecture for Back Propagation Through Time (BPTT) learning algorithm with 'N' number of neurons and 'n' numbers of foldings.

B. Real Time Recurrent Learning (RTRL)

RTRL is one of the most used real-time learning algorithms for RNN. In RTRL, the gradients at time 't' are computed based on the gradients at previous time steps. The gradient information is propagated in time, [24-26]. The basic RTRL architecture is shown in Fig. . The connections with blue color are the additional self-feedback and feedback connections, which is not included in static ANN. These additional connections make the network get previous input values and output values and consider them as additional internal inputs in the current time. By doing this, a network can work dynamically. However, RTRL algorithm suffers from slow convergence, which is typical for all gradient-based algorithms. Mandic and Chambers, [25] has presented an RTRL-based learning algorithm with an adaptive learning rate that can improve the convergence performance. RTRL further suffers from the large computational complexity of the order of O(N^4) and even critically with a storage requirement of the order of O(N^3), [23].

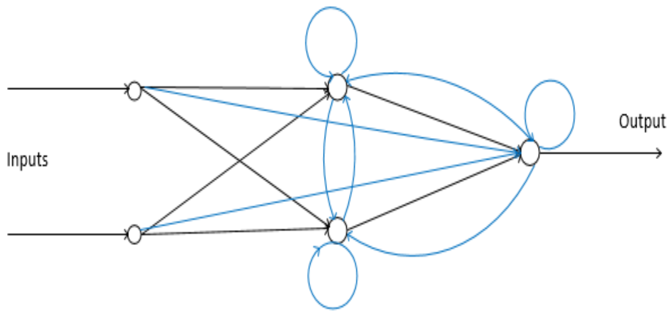


Fig. 2: A general architecture for Recurrent Neural Network (RNN) with self-feedback and feedback loops from neurons.

C. Extended Kalman Filter Learning (EKF)

EKF is a recursive algorithm that computes state estimations based on the previous state information at the current time, [27]. EKF can be used as a supervised on-line learning algorithm to determine the weights of an RNN. In EKF learning algorithm, the state vector consists of weights and the locally induced outputs of each neuron in the network. Regarding convergence to a solution, EKF is very fast compared to BPTT and RTRL. The order of computational complexity for EKF is same as RTRL, O(N^4), and the storage requirement increases to the order of O(N^4) for EKF. The RTRL algorithm is identical to the simplified EKF algorithm, and the architecture is the same, [23].

III. Experimental Set-up

To develop RNN models, model-drilling fluid is circulated in the flow loop. The circulated fluid is visco-plastic in nature with the fluid properties of density at 1136 kg/m^3 and a viscosity ranging from 23–180 [centipoise] for the 500–1 [s^{-1}] shear rate. Recent study shows that the level measurements at the throat (LT-18), the level of the downstream (LT-17) and the level of the upstream (LT-15) are highly correlated to flow rate, [20]. Therefore, these variables are considered for modeling and are given in Table. 1 and some concurrent measurements from these three ultrasonic sensors are shown in Fig. 3, along with simultaneous measurements of flow from a Coriolis meter. Fig. 4 shows the open channel section of flow rig with a Venturi constriction and three ultrasonic level sensors. All the three different levels measured using the ultrasonic sensors are used in the three models discussed above. The mass flow measurement is performed using Coriolis mass flow meter and is considered as a reference for RNN models.

For the mass flow rate range of 250-500 [kg/min], 1800 data samples for each variable are measured. The data samples are normalized in the range of (0-1). Out of 1800 normalized data samples, 70%, 15% and 15% of data are selected as training, validation, and test sets respectively.

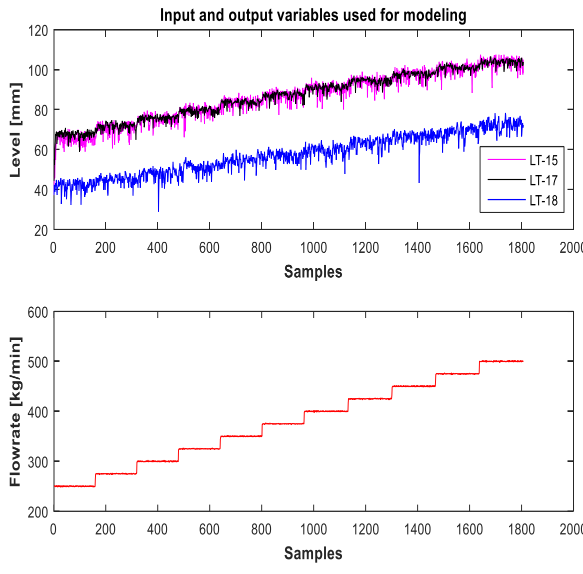


Fig. 3: Input and output variables used for developing RNN models. First plot shows three level measurements with LT-15, LT-17 and LT-18. Second plot shows flowrate measurement using Coriolis mass flowmeter

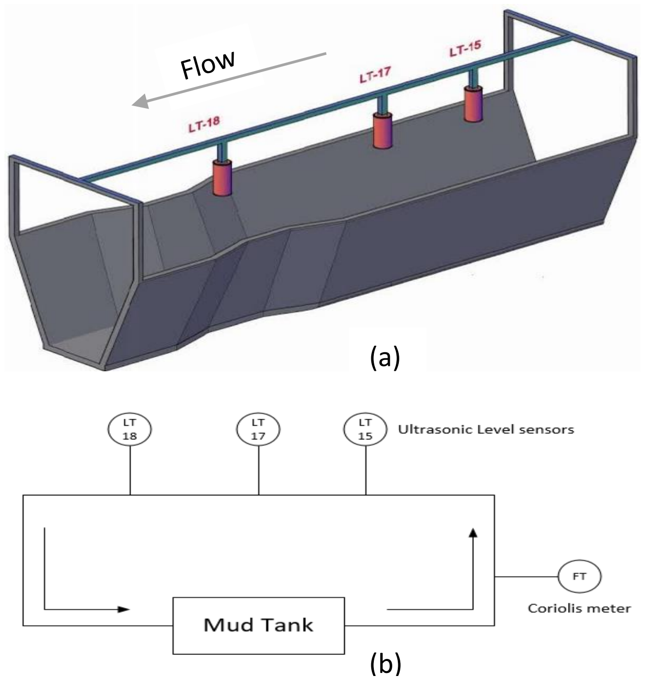


Fig. 4: a) An open channel with Venturi section and three level sensors, LT-15, LT-17 and LT-18, with an arrow showing a flow direction. (b) Extremely simplified P&ID for the Venturi-rig flow loop with the measurands used in this study, viz. ultrasonic level sensors and FT-Coriolis mass flowmeter.

TABLE. 1: INPUT AND OUTPUT VARIABLES USED FOR DEVELOPING RNN MODELS WITH THE RANGE AND VARIABLE TYPE.

Variables	Range	Units	Type
Upstream level measurement	31.2 - 107.5	mm	Input
Level measurement at the throat	28.9 - 78.3	mm	Input
Downstream level measurement	44.3 – 106.6	mm	Input
Mass flow rate	250 - 500	kg/min	Output

IV. Results

This paper presents results from both simulations based on the three models and practical implementation of RNN for flow rate measurement in an open channel flow loop.

A. Simulation study

RNN is implemented using all the three learning algorithms discussed in Section II. Table. 2 shows the optimal parameters used in the simulations. These optimal parameters are determined using grid search method and the optimization is done using Mean Absolute Percentage Error (MAPE). Apart from these parameters, number of neurons selected is 7, learning rate is 0.1 and number of folds for BPTT is 7.

Fig. 5 shows the comparison of RNN with different algorithms. As discussed in Section II, EKF learning algorithm can quickly converge to a solution. From Fig. 5 showing the MSE, it can be seen that EKF converges well before 20 epochs, BPTT converges around 100 epochs, and RTRL takes around 300 epochs to converge. The converging efficiency of these algorithms can be observed using the state parameters, which are weights of the neural network. Fig. 6 shows the states of some of the weights while training a network. The state representation shows that the states in EKF and BPTT algorithms go to steady state very quickly. However, RTRL needs numerous training epochs for achieving steady states.

Fig. 7 shows the estimations of different learning algorithms with reference to flow measurements from Coriolis mass flowmeter. The simulation results show that all the models using different learning algorithms are capable of describing the dynamics of the reference flow measurements well. RTRL has minimum MAPE out of the three models used, as shown in Table. 2.

TABLE. 2: OPTIMAL PARAMETERS FOR DIFFERENT LEARNING ALGORITHMS

Learning algorithms	Epochs	Number of previous inputs	Number of previous outputs	MAPE [%]
BPTT	200	1	3	2.97
RTRL	500	4	4	2.55
EKF	20	4	4	3.70

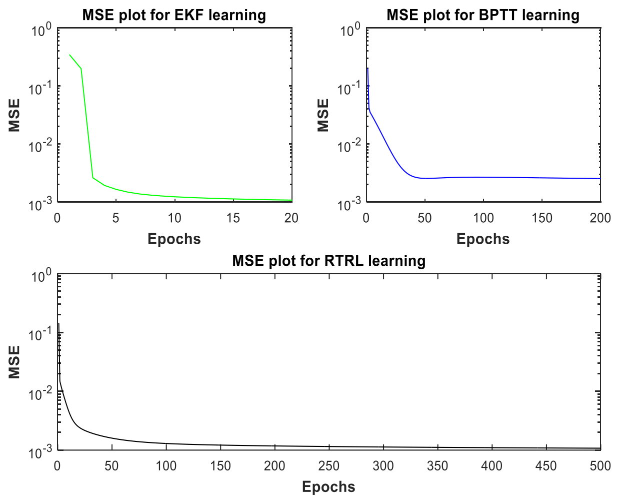


Fig. 5: Mean Squared Error (MSE) plot for three different learning algorithms in RNN. Simulation results.

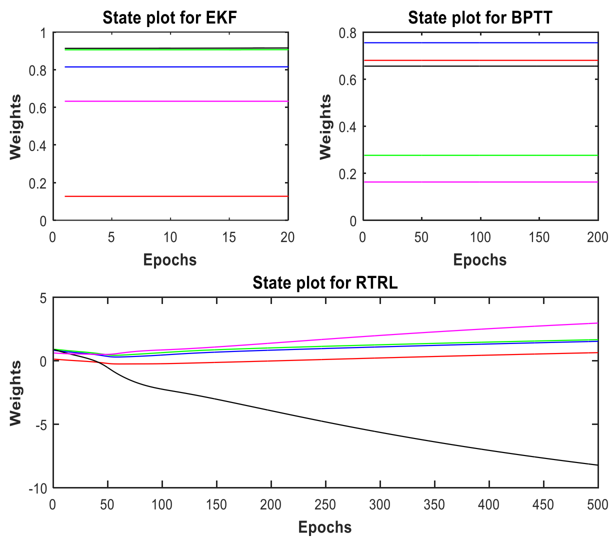


Fig. 6 Different weights of the network in a state plot illustrating the convergence of the learning algorithms. Simulation results.

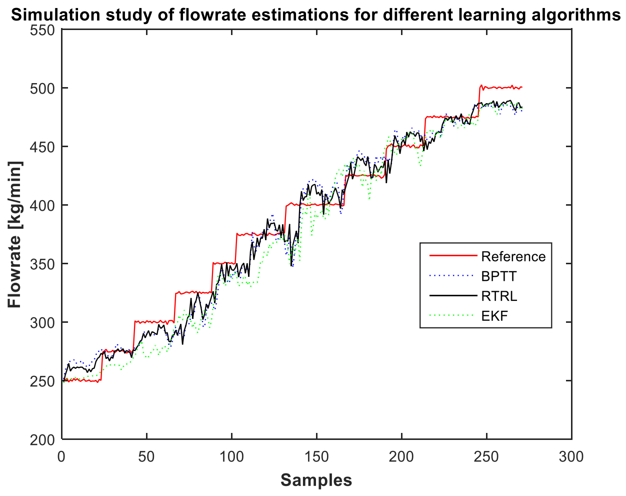


Fig. 7: Comparison of flow rate estimation of Recurrent Neural Network (RNN) with three different learning algorithms with respect to Coriolis mass flow measurement as a reference measurement. Simulation results.

B. Experimental study

The experimental study involves the implementation of simulation study in the Venturi rig. Despite slow convergence, RNN with RTRL learning algorithm is selected for its accuracy, less complexity, and faster computation. The algorithms for both BPTT and EKF have complex architectures and they are computationally demanding. This makes RTRL a suitable choice for implementing in the Venturi rig for the flow estimation. Fig. 8 shows the experimental results obtained using model-drilling fluid in the test Venturi rig. The flow rate estimation using RNN is compared with the estimation previously made using static ANN and SVR. The comparison shows that RNN has better performance than other empirical models. The MAPE for RNN, ANN and SVR are 5.6%, 8.5%, and 7.7% respectively.

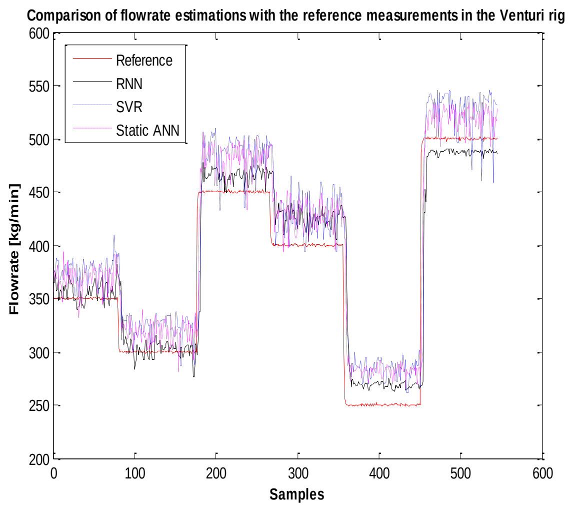


Fig. 8: Comparison of flow rate estimations of a Dynamic Artificial Neural Network with Real Time Recurrent Learning algorithm (MAPE of 5.6%), a static Artificial Neural Network model (MAPE of 8.5%) and a Support Vector Regression model (MAPE of 7.7%) with respect to the Coriolis mass flow measurement as a reference flow measurement. Based on experiments using the Venturi-rig.

For the future work, we will try to improve the sensor measurements using suitable signal processing. As shown in Fig. 3, the output mass flowrate using Coriolis mass flowmeter is less noisy as compared to the three input level measurements. Since the model completely depends on the data, we will work on online signal processing of level sensor measurements to reduce the noise in the measurements. In Fig. 7 and Fig. 8, we can see discontinuous peaks in the predictions of all the empirical models. By implementing these three models as an integral part of the processing algorithms (signal and control), we believe that our model can be trained and operated with less noisy data resulting in improved predictions.

V. Conclusion

One way of having safe and efficient drilling operation is by continuously monitoring the properties of drilling mud. Any unwanted change in fluid properties can lead to two main problems; the influx of formation fluid and circulation fluid loss. The delta flow measurement while drilling is one of the best methods to detect the early influx or early fluid loss. In this paper, we introduced dynamic Artificial Neural Network to estimate the flow rate of non-Newtonian drilling fluids in an open channel venturi flume, which can be used for outflow measurement while determining delta flow. With Recurrent Neural Network, we simulated three different learning algorithms; Back Propagation Through Time, Real-Time Recurrent Learning and Extended Kalman Filter algorithm. The simulation results show that BPTT and EKF converge very quickly as compared to RTRL algorithms. Whereas, RTRL algorithm is more accurate, less complex and computationally faster than other two algorithms. So, based on this simulation analysis, RNN with RTRL algorithm is selected for the practical implementation. In the Venturi rig, RNN model with RTRL is implemented along with static ANN and Support Vector Regression (SVR) models. The experimental estimations of flow rates with respect to reference flow rate using Coriolis mass flowmeter show that the estimates based on RNN model has higher accuracy compared to ANN and SVR models. This improved performnce is due to the fact that RNN contains previous inputs and outputs as additional inputs for the current time, which are not considered in static ANN and SVR models.

Acknowledgment

The Ministry of Education and Research of the Norwegian Government is funding Khim Chhantyal's PhD studies at University College of Southeast Norway (USN). The authors at USN appreciate the collaboration with and support from STATOIL for the assembly and commissioning of the open channel Venturi-rig with the various sensors and control system dedicated to the studies related to non-Newtonian fluids. In addition, we acknowledge the practical work done by various groups of bachelor and master students of USN in conjunction with this project.

References

[1] R. Caenn, H. C. H. Darley, G. R. Gray, "Composition and properties of drilling and completion fluids," 6th ed., ISBN: 978-0-12-383858-2, Waltham, USA: Gulf professional publishing, 2012, p.7-16.

[2] S. Hauge, K. Øien, "Deepwater Horizon: Lessons learned for the Norwegian Petroleum Industry with focus on Technical Aspects," Chemical Engineering transactions, vol. 26, 2012, pp. 621–626.

[3] L. D. Maus, J. D. Tannich, W. T. Ilfrey, "Instrumentation requirements for kick detection in deep water," in Offshore technology conference, Houston, Texas, 1978.

[4] J. M. Speers, G. F.Gehrig, "Delta flow: an accurate, reliable system for detecting kicks and loss of circulation during drilling," in SPE Drilling Engineering, 1987.

[5] J.J. Orban, K.J. Zanner, A.E. Orban, Anadrill/ Schlumberger, "New flowmeters for kick and loss detection druing drilling," in 62nd Annual Technical Conference and Exhibition of the Society of Petroleum Engineers, Dallas, 1987.

[6] J.J. Orban, K.J. Zanner, Anadrill/ Schlumberger, "Accurate flow-out measurements for kick detection, actual response to controlled gas influxes," in IADC/SPE Drilling Conference, Dallas, Texas, 1988.

[7] D. M. Schafer, G. E. Loeppke, D. A. Glowka, D. D. Scott, E. K. Wright, "An evaluation of flowmeters for the detection of kicks and lost circulation during drilling," in SPE/IADC drilling conference, New Orleans, Louisiana, 1992.

[8] M. Kamyab, S. R. Shadizadeh, H. Jazayeri-rad, N. Dinarvand, "Early kick detection using real time data analysis with dynamic neural network: a case study in Iranian oil fields," in Nigeria annual international conference and exhibition, Tinapa-Calabar, Nigeria, 2010.

[9] I. Mills, D. Reitsma, Z. Tarique, "Simulator and the first field test results of an utomated early kick detection system that uses standpipe pressure and annular discharge pressure," in SPEC/IADC managed pressure drilling and underbalanced operations conference and exhibition, Milan, Italy, 2012.

[10] T. H. Ali et al., "Automated alarms for smart flowback fingerprinting and early kick detection," in SPE/IADC drilling conference, Amsterdam, Netherland, 2013.

[11] B. Patel, T. Cooper, W. Billings, "The application of advanced gas extraction and analysis system complements early kick detection & control capabilities of managed pressure drilling system with added HSE value," in SPE/IADC drilling conference, Amsterdam, Netherland, 2013.

[12] A. K. Vajargah, S. Z. Miska, M. Yu. M. E. Ozbayoglu, R. Majidi, "Feasibility study of applying intelligent drill pipe in early detection of gas influx during conventional drilling," in SPE/IADC drilling conference, Amsterdam, Netherland, 2013.

[13] F. M. White, "Fluid mechanics," WCB McGraw-Hill, 2002, p. 659–708.

[14] M. Skorpik, "Flow measurement options for pipeline and open channel flow," in 2013 Workshop, Montana association of dam and canal systems conference (MADCS).

[15] F. Frenzel et al., "Industrial flow measurement basics and practice," ABB automation products Gmbh, 2011.

[16] G. Gerätebau "Equipment for engineering education, intruction manual HM 162.51 venturi flume, " Germany, 2003.

[17] C. Berg, M. Anjana, C. E. Agu, C. Khim, F. Mohammadi, “Simulatioin of open channel flow for mass flow measurement,” University of South East Norway, Norway, 2013.

[18] C. E. Agu, B. Lie, “Numerical solution of the Saint Venant equation for non-Newtonian fluid,” in Proceedings from the 55th conference on simulation and modelling (SIMS 55), Aalborg, Denmark, 2014.

[19] C. E. Agu, B. Lie, “Smart sensors for measuring fluid flow using a venturi channel,” in Proceedings from the 55th conference on simulation and modelling (SIMS 55), Aalborg, Denmark, 2014.

[20] K. Chhantyal, H. Viumdal, S. Mylvaganam, G. Elseth, “Ultrasonic level sensors for flowmetering of non-Newtonian fluids in open venturi channels,” IEEE Sensors Applications Symposium (SAS), Catania, Italy, 2016.

[21] E. O. Dijk, “Analysis of Recurrent Neural Networks with application to speaker independent phoneme recognition,” University Twente, Enschede, The Netherlands, 1999.

[22] M. Boden, “A guide to Recurrent Neural Network and backpropagation,” Halmstad University, Sweden, 2001.

[23] R. J. Williams, “Some observations on the use of the Extended Kalman Filter as a Recurrent Network Learning algorithm,” College of Computer Science, Northeastern University, Boston, 1992.

[24] M. W. Mak, K. W. Ku, Y. L. Lu, “On the improvement of the real time recurrent learning algorithm for Recurrent Neural Networks,” in Neurocomputing, vol. 24, Elsevier, 1999, p.13-36.

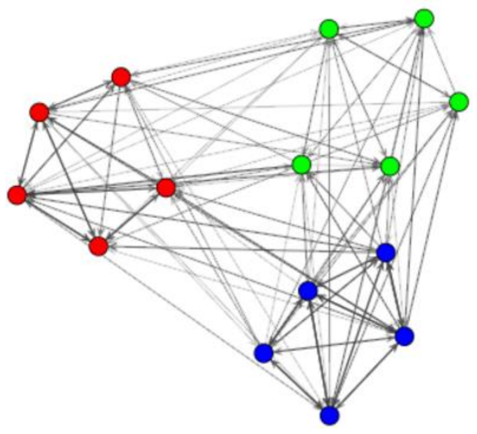
[25] D. P. Mandic, J. A. Chambers, “A normalised real time recurrent learning algorithm,” in Signal Processing, vol. 80, Elsevier, 2000, p.1909-1916.

[26] D. B. Budik, “A resouce efficient localized Recurrent Neural Network architecture and learning algorithm,” University of Tennessee, USA, 2006.

[27] P. Kim, “Kalman filter for beginners,” ISBN-13:978-1463648350, CreateSpace Independent Publishing Platform, 2011.

Appendix C: Dynamic Neural Network Toolbox - Tutorial

Dynamic Artificial Neural Network (DANN) MATLAB Toolbox – Tutorial



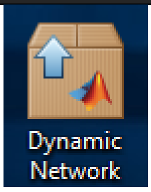
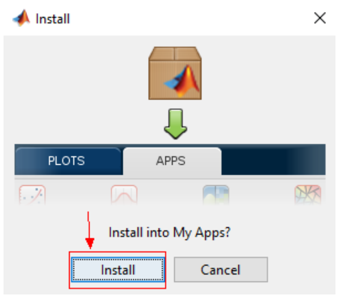
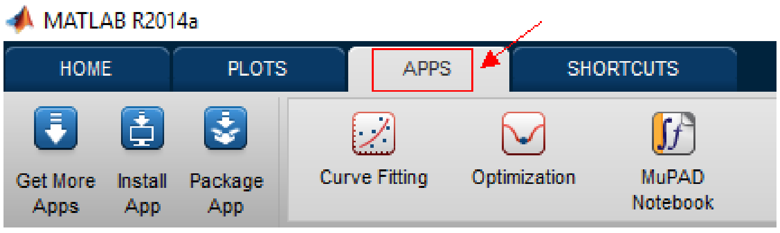
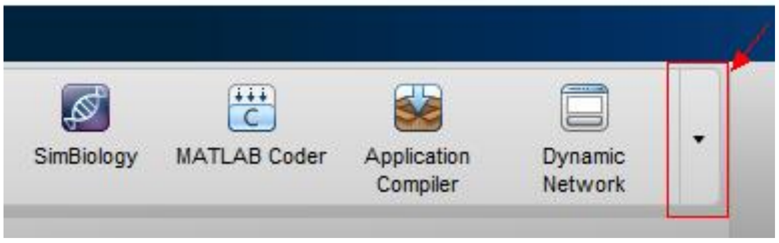
Khim Chhantyal

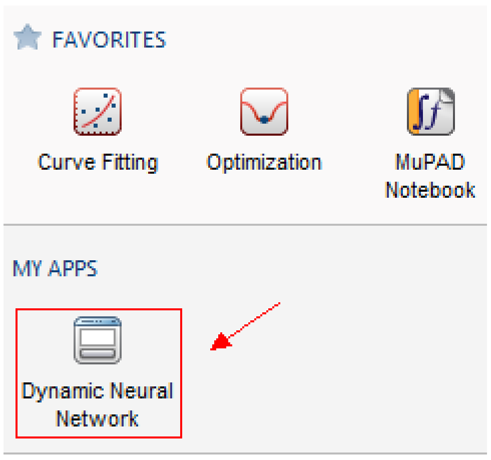
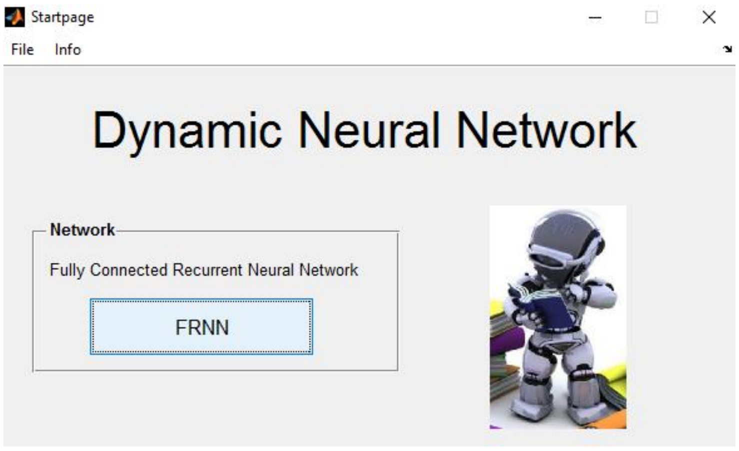
Minh Hoang

2016

C.1 How to install the toolbox

The first step in installing the toolbox is to download the installation file that can be found in this link. After downloaded, follow the steps below to complete the installation.

Steps	Picture
Double click on the MATLAB App Installer "Dynamic Network"	
Click on "Install"	 Install PLOTS APPS Install into My Apps? Install Cancel
In Matlab, go to the tab "APPS"	 MATLAB R2014a HOME PLOTS APPS SHORTCUTS Get More Apps Install App Package App Curve Fitting Optimization MuPAD Notebook
Click on the arrow "↓" to see more options	 SimBiology MATLAB Coder Application Compiler Dynamic Network

Click on "Dynamic Network"	 FAVORITES Curve Fitting Optimization MuPAD Notebook MY APPS Dynamic Neural Network
The toolbox is now ready to use.	 Startpage File Info Dynamic Neural Network Network Fully Connected Recurrent Neural Network FRNN

C.2 General information

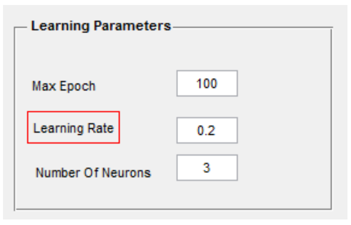
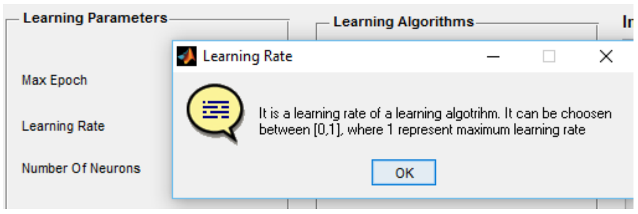
For first time users, it is recommended to take a look on this section.

C.2.1 Help

This section describes how you can get more information if you have some questions or problems related to the toolbox.

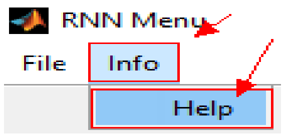
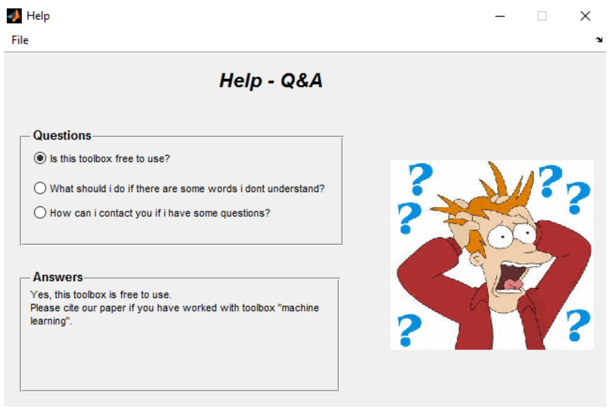
C.2.2 Right click:

If there are some words or expressions in the toolbox that you are not sure about, you can right-click on the word to get a pop-up description message.

Description	Picture
Let say you want to know more about what "Learning Rate" is.	 Learning Parameters Max Epoch 100 Learning Rate 0.2 Number Of Neurons 3
Hover your mouse over the text and right click. A textbox with explanation will appear. Do this for all the expression you want to know more about.	 Learning Rate It is a learning rate of a learning algotrihm. It can be choosen between [0,1], where 1 represent maximum learning rate OK

C.2.3 Q&A section:

For common questions and answers related to the toolbox. You're are welcome to check out the "Help windows". The instructions below will show how you get to the "Help Window".

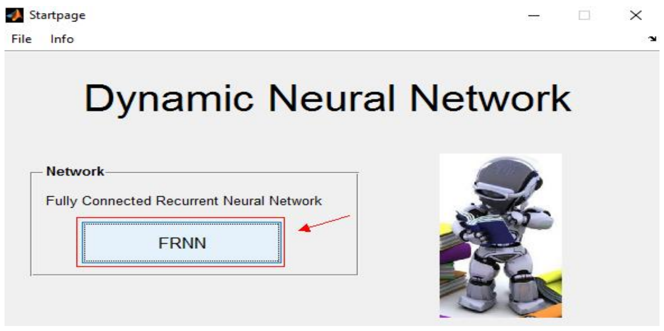
Descriptions	Picture
At the top left corner, click on "Info" and then on "Help"	
The "Help" window will appear and you can click on the question that you are interested in. If you can't find your question, feel free to send us an email. Click on the question "**How can I contact you if I have some questions**" to get more information.	 *Help - Q&A* **Questions** [✓ Is this toolbox free to use?] What should i do if there are some words i dont understand? How can i contact you if i have some questions? **Answers** Yes, this toolbox is free to use. Please cite our paper if you have worked with toolbox "machine learning".

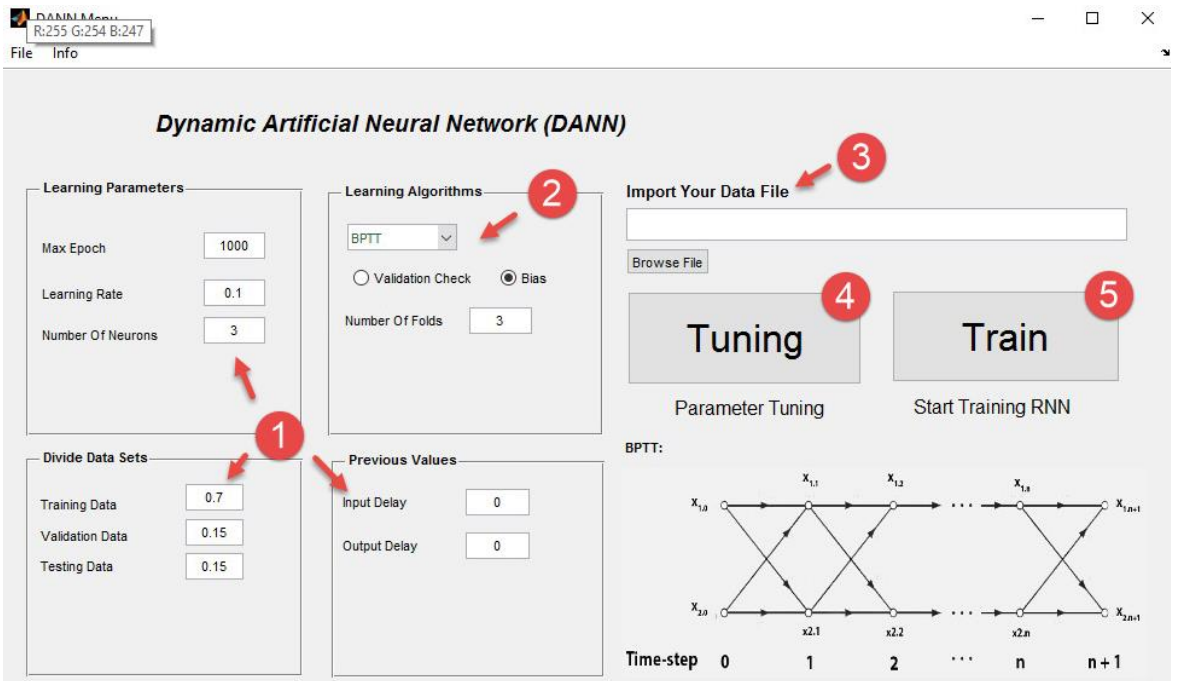
C.3 Functions in DANN toolbox

All the functions in the toolbox will be specified below. There will be given examples of how to use and interpret the results from each functions.

C.3.1. How to use DANN

Description	Picture

On the Startpage, click on "Go to FRNN".	 Startpage File Info Dynamic Neural Network Network Fully Connected Recurrent Neural Network FRNN



Max Epoch 1000
Learning Rate 0.1
Number Of Neurons 3
BPTT
Validation Check [✓ Bias]
Number Of Folds 3
Training Data 0.7
Validation Data 0.15
Testing Data 0.15
Input Delay 0
Output Delay 0

A new window "RNN" will appear. Follow these procedures.

1. You can choose to keep the default parameters or change it.

2. Select your learning algorithm, and choose if you want to have validation check and bias on or not.

3. Import the data file. Remember that the imported file must be a Matlab file with the variable name "Data".

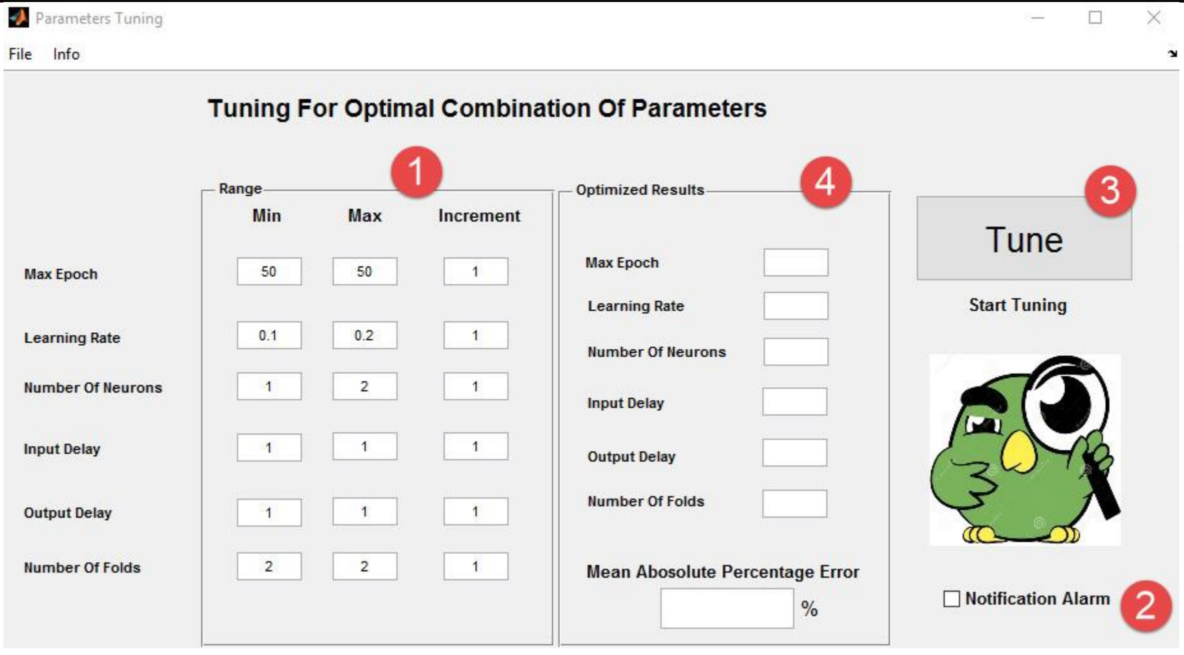
4. (Optional) Hit the "Tuning" button to find the optimal parameters based on grid search method.

5. Hit the "Train" button to start training the model.

C.3.1.1 Tuning (Optional)

In any implementation of Artificial Neural Network (ANN), tuning of parameters is one of the biggest challenges. This DANN MATLAB toolbox provides a facility to tune the parameters optimally.

Description


Parameters Tuning
File
Info
Tuning For Optimal Combination Of Parameters
1
2
3
4
Tune
Start Tuning
Notification Alarm
Mean Abosolute Percentage Error
%

	Range Min	Max	Increment
Max Epoch	50	50	1
Learning Rate	0.1	0.2	1
Number Of Neurons	1	2	1
Input Delay	1	1	1
Output Delay	1	1	1
Number Of Folds	2	2	1

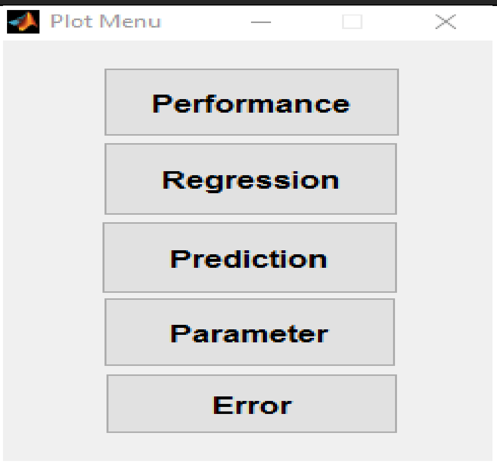
Optimized Results	
Max Epoch	
Learning Rate	
Number Of Neurons	
Input Delay	
Output Delay	
Number Of Folds	

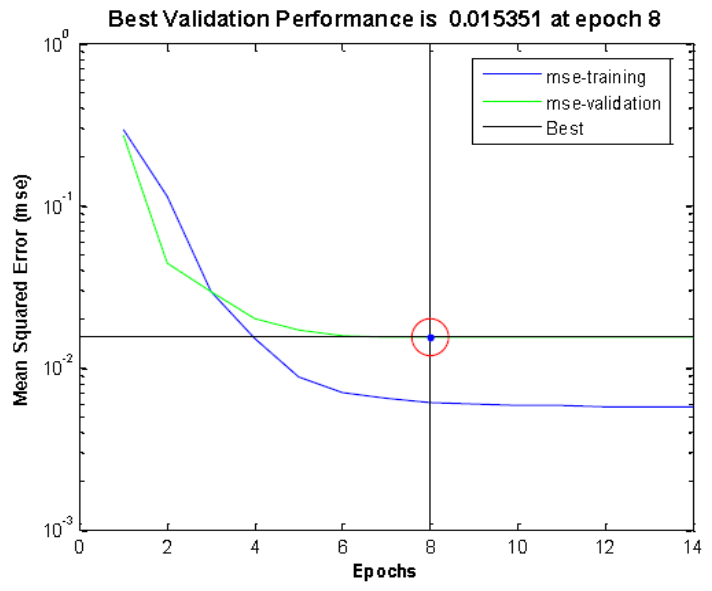
1. Assign lower limit, higher limit and an increment to each parameter

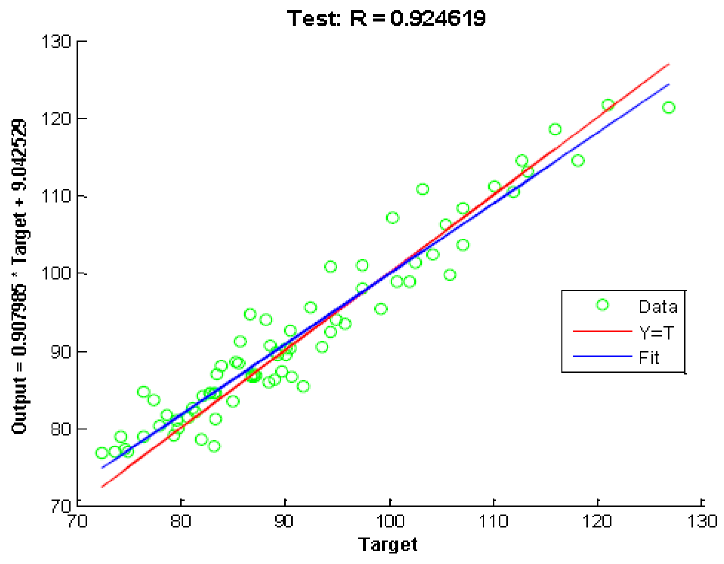
2. (Optional), check on “Notification Alarm” if you want the program to play a sound when the tuning is finished. 3. Hit the “Tune” button to start tuning with the parameters specified. 4. When the tuning is finished, the optimal parameters will be displayed in this panel.

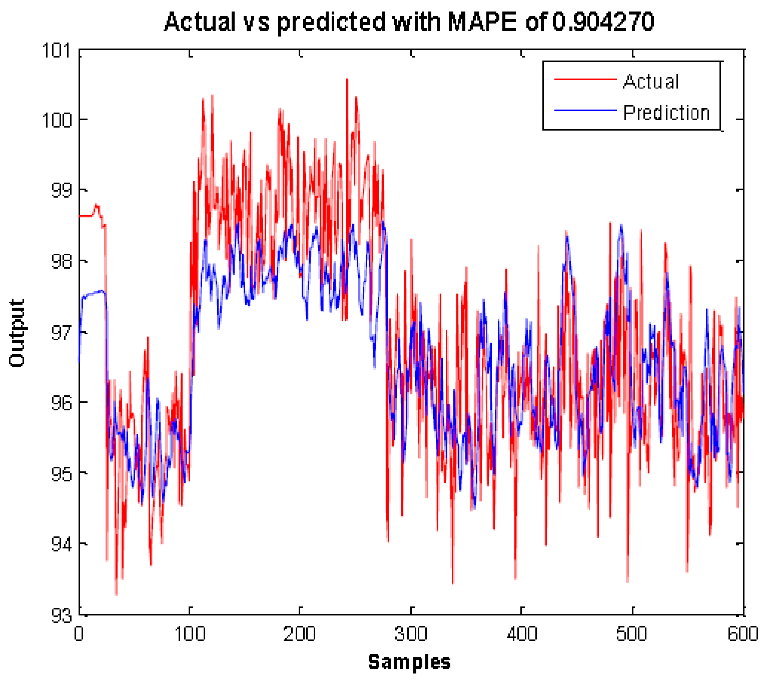
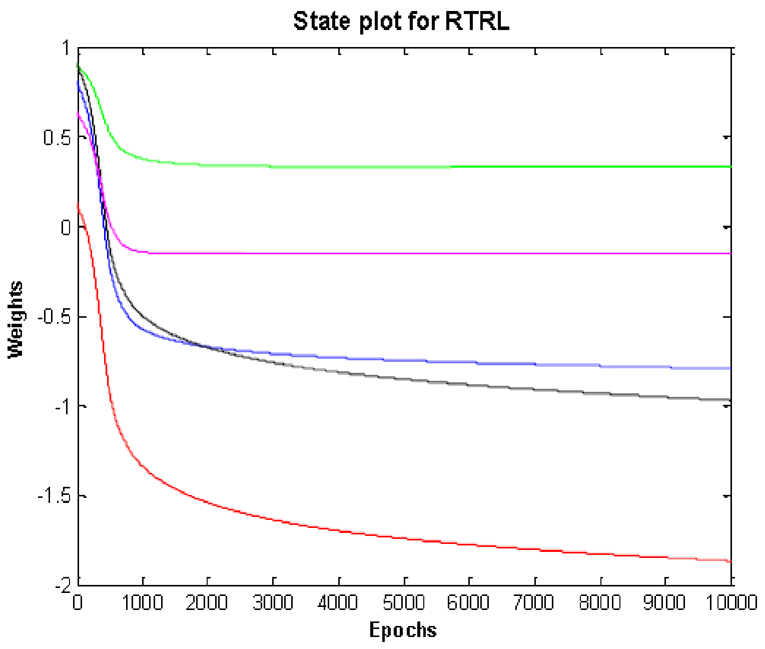
C.3.1.2 Train

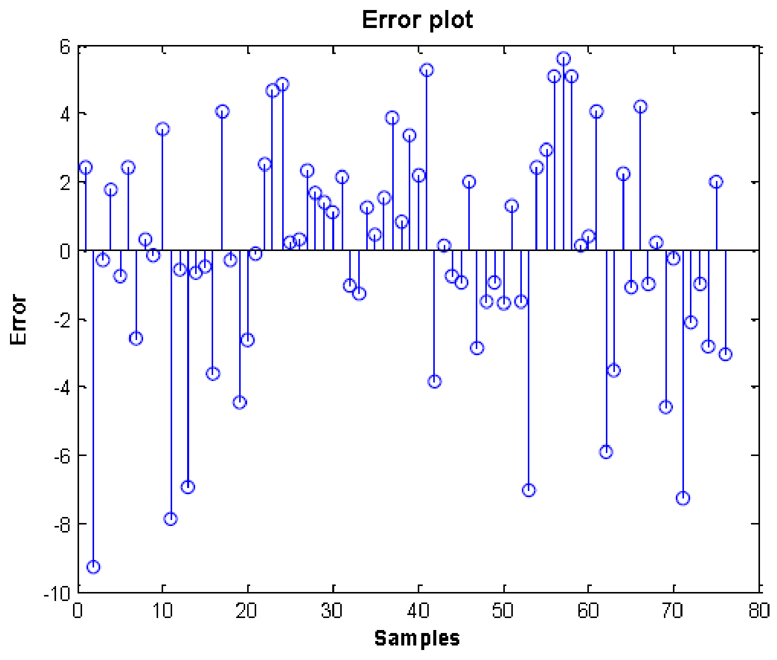
After you finished training your model, the program will show a Plot Menu where you can choose the option you want to have a closer look at. The options will be described below.

Description	Picture
Plot Menu The GUI that let the user select what to see after the training is finished.	 Plot Menu Performance Regression Prediction Parameter Error

Performance It shows the MSE for training data set and validation error for each epoch.	

Regression It compares the target output and model prediction in terms of squared correlation coefficient such that '0' meaning not related at all and '1' meaning highly correlated to each other.	

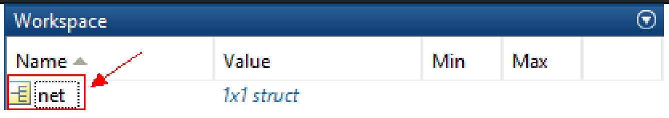
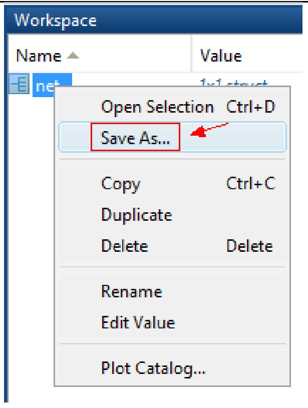
Prediction It shows the test data and model prediction with MAPE between them.	
Parameter: It shows the states of five different randomly chosen weights at different epochs. The analysis using parameter plot is very efficient if you are working with some system identification problems. In that case, one can visualize how the weights change with epochs. The steady state values of the weights after some epochs are the model parameters in typical system identification problems.	

Error It shows the error between the target value and the model prediction for each test samples.	

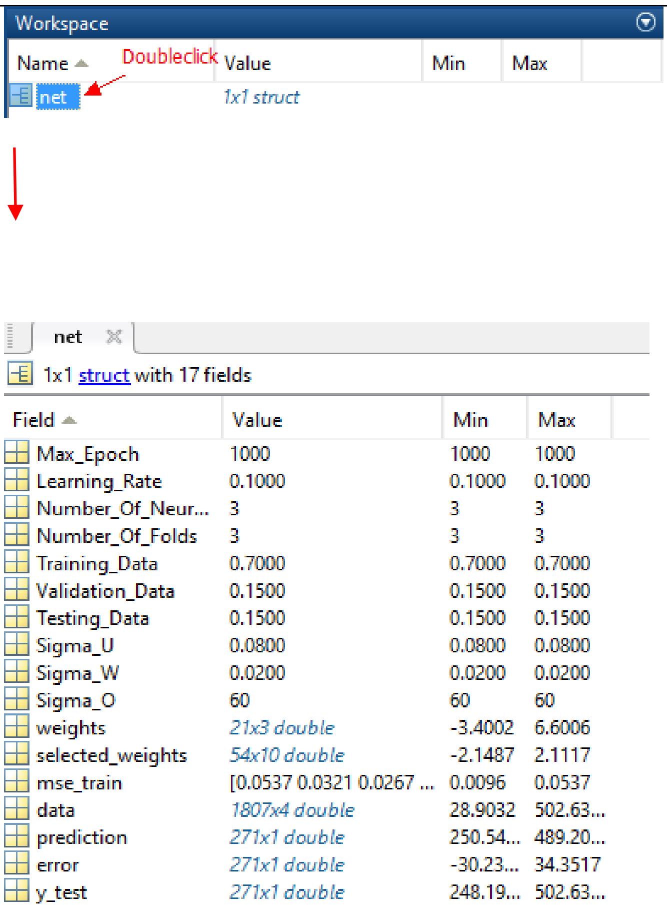
C.3.2 Where is the data saved?

For each time you train your model, the data is saved in Matlab as a struct variable name "net".

If you want to see the data or save it, then follow the steps below.

Steps	Picture
After your model is trained, a struct named "net" is created in the workspace.	Workspace Name / Value / Min / Max net / 1x1 struct
If you want to save this data so you can have a look at this later. Right-click on "net" and click on "Save As…"	 Workspace Name Value net 1x1 struct Open Selection Ctrl+D Save As... Copy Ctrl+C Duplicate Delete Delete Rename Edit Value Plot Catalog...

If you only want to have a look at the data, then double click on "net" and a new window with all the data will appear.



Name	Value	Min	Max
net	1x1 struct		

1x1 struct with 17 fields

Field	Value	Min	Max
Max_Epoch	1000	1000	1000
Learning_Rate	0.1000	0.1000	0.1000
Number_Of_Neur...	3	3	3
Number_Of_Folds	3	3	3
Training_Data	0.7000	0.7000	0.7000
Validation_Data	0.1500	0.1500	0.1500
Testing_Data	0.1500	0.1500	0.1500
Sigma_U	0.0800	0.0800	0.0800
Sigma_W	0.0200	0.0200	0.0200
Sigma_O	60	60	60
weights	21x3 double	-3.4002	6.6006
selected_weights	54x10 double	-2.1487	2.1117
mse_train	[0.0537 0.0321 0.0267 ...	0.0096	0.0537
data	1807x4 double	28.9032	502.63...
prediction	271x1 double	250.54...	489.20...
error	271x1 double	-30.23...	34.3517
y_test	271x1 double	248.19...	502.63...

C.4 Publication on DANN toolbox

This paper on DANN toolbox is accepted for EUROSIM 2016, 12-16 September, Oulu, Finland, "The 9th Eurosim Congress on Modelling and Simulation".

Dynamic Artificial Neural Network (DANN) MATLAB Toolbox for Time Series Analysis and Prediction

Khim Chhantyal, Minh Hoang, Håkon Viumdal, Saba Mylvaganam
University College of Southeast Norway
Faculty of Technology
Kjølnes Ring 56, 3918 Porsgrunn, Norway

Abstract **– MATLAB® Neural Network (NN) Toolbox can handle both static and dynamic neural networks. To use this MATLAB® NN Toolbox, in cases where recurrent neural networks occur is not straight forward. We present a Dynamic Artificial Neural Network (DANN) MATLAB toolbox capable of handling fully connected neural networks for time-series analysis and predictions. Three different learning algorithms are incorporated in the MATLAB DANN toolbox: Back Propagation Through Time (BPTT) an offline learning algorithm and two online learning algorithms; Real Time Recurrent Learning (RTRL) and Extended Kalman Filter (EKF). In contrast to existing MATLAB® NN Toolbox, the presented MATLAB DANN toolbox has a possibility to perform the optimal tuning of network parameters using grid search method.**

Three different cases are used for testing three different learning algorithms. The simulation studies confirm that the developed MATLAB DANN toolbox can be easily used in time-series prediction applications successfully. Some of the essential features of the learning algorithms are seen in the graphical user interfaces discussed in the paper. In addition, installation guide for the MATLAB DANN toolbox is also given.

Keywords— Dynamic Artificial Neural Network (DANN), Back Propagation Through Time (BPTT), Real-Time Recurrent Learning (RTRL), Extended Kalman Filter (EKF), time series.

I. Introduction

Artificial Neural Networks (ANN) are computational models consisting of many neurons in different layers with varying degrees of interconnections between them. The interconnection have weights assigned to them so that the ANNs can be tuned thus enabling them to learn and adapt. Feedforward or feedback networks are two broad classifications of ANNs. Feedforward ANNs use current inputs and current outputs, whereas, feedback ANNs use current and previous inputs and outputs. Feedback ANN performs time-series predictions and is a dynamic network. This type of network constitutes recurrent neural networks (RNN) either partially or fully connected depending on the extent of the feedback loops available in the network. Fully connected RNNs have interconnected feedback loops including self-feedback loops, whereas partially connected RNNs do not have self-feedback loops, [1-2].

In an existing MATLAB® Neural Network Toolbox, there is a possibility to use feedforward ANN for static estimations and partially connected RNN for time-series predictions. This paper presents a MATLAB toolbox that can perform the empirical modeling using fully connected RNNs with three different learning algorithms. The following sections present the overview of the developed toolbox and the usage of the toolbox in three different practical applications.

II. Overview of Toolbox

The developed Dynamic Artificial Neural Network (DANN) toolbox consists of three main user interfaces, which are DANN Menu, Parameter Tuning, and Plot Menu. Each window consists of different elements as given below.

A. DANN Main

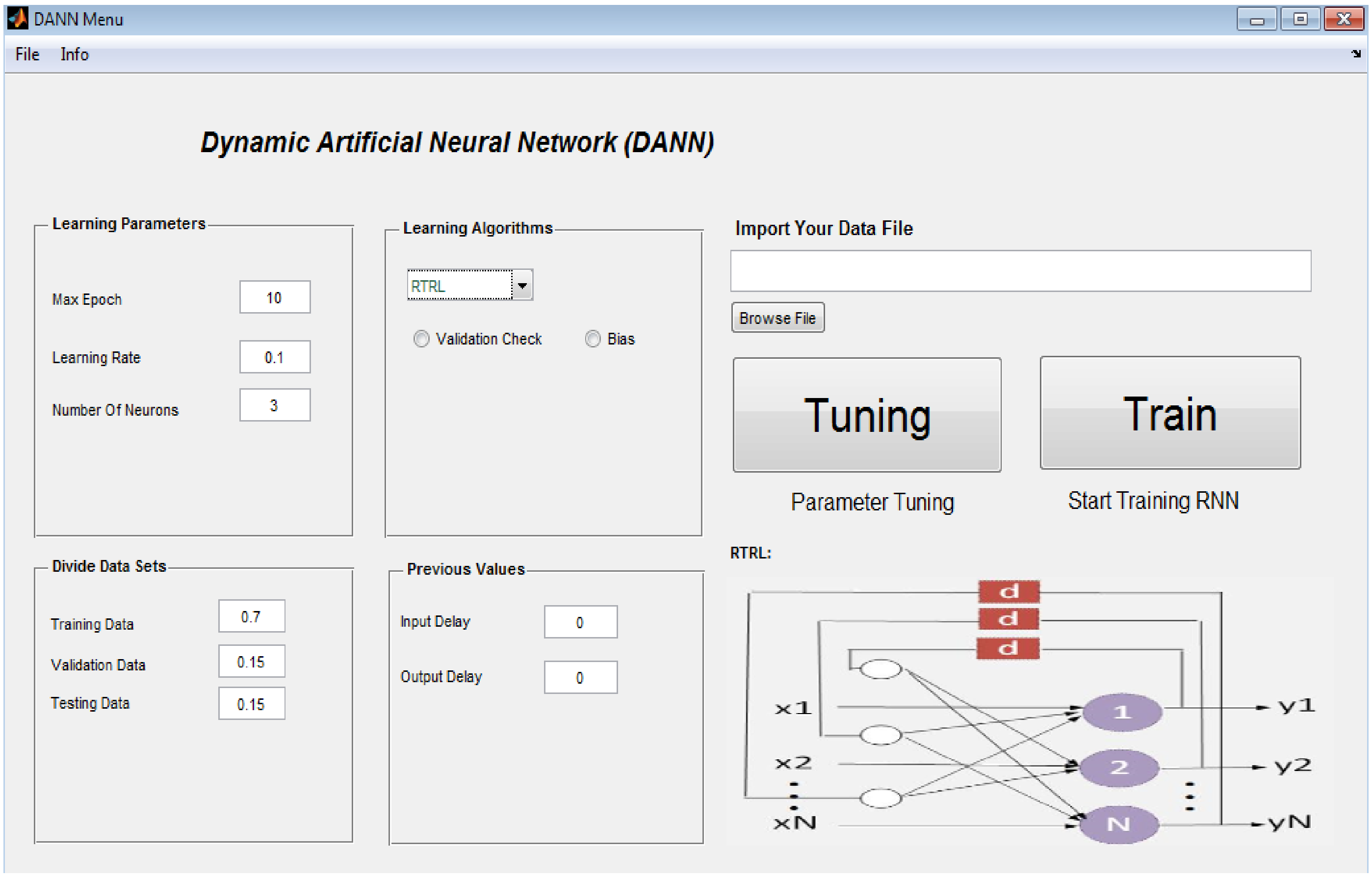
DANN Main is the main window of MATLAB DANN toolbox as shown in Fig. 1. In this window, the user can upload the data set, divide data sets, select validation check, include bias, select learning algorithm, define learning parameters, select the number of previous inputs and outputs, and finally train the model.

1) *Uploading data set*: A user needs to upload his/her data set to train the model using MATLAB DANN toolbox. The format of the data set should be in '.mat' and each column should

represent the variables in the model, where the last column is an output variable.

2) *Division of data set*: The uploaded data set should be divided into a training set, validation set, and testing set. A user can choose the percentage of data for training, validation and testing. Experience shows that 70% for the training set, 15% for both validation and testing works fine for any learning algorithms.

3) *Validation check*: A validation check is an option that prevents the over-fitting of the network. Over-fitting and under-fitting are most common problems encountered while dealing with data models. Under-fitting can be improved by either tuning the learning rate or increasing the number of neurons in the network. The concept of over-fitting in MATLAB DANN toolbox is similar to the concept used in NN Toolbox in MATLAB®, [2]. The main idea is to terminate the RNN before the network gets over-fitted. For early stop, Mean Squared Error (MSE) for both training and validation is continuously monitored. While training a network, learning algorithm builds a certain hypothetical model for the network at each epoch.


DANN Menu

File Info

Dynamic Artificial Neural Network (DANN)

Learning Parameters

Max Epoch	10
Learning Rate	0.1
Number Of Neurons	3

Learning Algorithms

RTRL

Validation Check Bias

Import Your Data File

Divide Data Sets

Training Data	0.7
Validation Data	0.15
Testing Data	0.15

Previous Values

Input Delay	0
Output Delay	0

Fig. 9: DANN Menu of MATLAB DANN toolbox with the possibility to upload data, data division, selecting learning algorithms, tuning learning parameters, and training the algorithm.

The validation data are validated using the hypothetical model at that particular epoch. While learning, the value of MSE of training and validation data keep on reducing and the training of the network gets better with increasing epoch. However, the validation error can increase though the error for training decreases, which occurs in cases of over-fitting. The increase in validation error can be due to some randomness in the training process. Therefore, the MATLAB DANN toolbox will count six consecutive increments in the validation error before it stops the learning algorithm.

When the model is over-fitted, the trained model seems to have good performance with training data, but it can have a large error while testing with the new data set. In other words, the trained model is not a generalized model when it is over-fitted. The implementation of validation check is presented in Case II in Section III. In case, if the validation check is not selected, validation data will be part of the training data set.

4) *Bias*: It is an offset value added to the output of the neurons. It is often important to include bias in each neuron while constructing a neural network model. MATLAB DANN toolbox facilitates a choice to include or exclude bias terms in the network.

5) *Learning algorithm*: In this toolbox, there are three learning algorithms. The user can select any one of these algorithms based on the requirements regarding complexity, accuracy and application.

a) *Back Propagation Through Time (BPTT):* BPTT is an extension of gradient-based backpropagation algorithm that is used in feedforward ANN. The idea in BPTT is to unfold the RNN architecture into feedforward ANN architecture using an arbitrary number of folds. The BPTT architecture for the neural network with two neurons is shown in Fig. 2. The network with BPTT algorithm is less complex compared to other learning algorithms. However, the complexity and the memory requirement increase when the number of folds increases. [3]

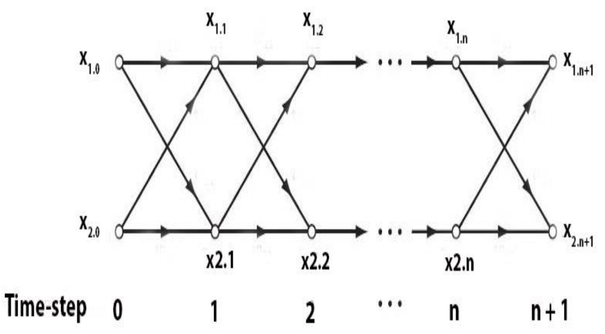


Fig. 2: Architecture for Back Propagation Through Time (BPTT) learning algorithm with two neurons and 'n' numbers of folding. [4]

b) *Real Time Recurrent Learning (RTRL):* RTRL is one of most accepted real-time learning algorithms for RNN. In RTRL, the gradients at time 't' are computed using the propagation of gradients at previous time steps, [5-7]. The underlying RTRL architecture is shown in Fig. 3, where x, y, N and d are inputs, outputs, number of neurons and unit time delay respectively. Based on the complexity, RTRL is the simplest online learning algorithm. However, the algorithm converges slowly and requires large memory for storage. [3]

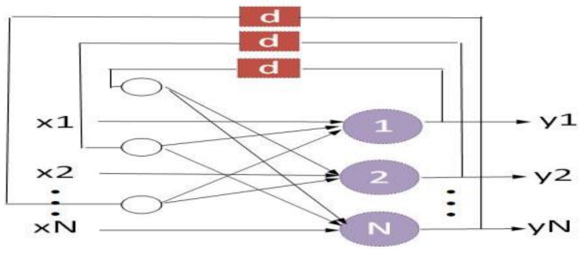


Fig. 3: A general architecture for Real Time Recurrent Learning (RTRL) and Extended Kalman Filter (EKF) learning algorithms showing self-feedback and feedback loops within the neurons.

c) *Extended Kalman Filter Learning (EKF):* EKF can be used as a supervised on-line learning algorithm to tune the weights of RNN. In EKF, the state vector consists of weights and the locally induced outputs of each neuron in the network. Regarding convergence, EKF is the fastest algorithm among the algorithms presented in the MATLAB DANN toolbox. The order of computational complexity for EKF is the same as for RTRL, and the storage requirement is larger for EKF. The general architecture for EKF learning is shown in Fig. 3. [3]

6) *Learning parameters*: The parameters of learning algorithms such as the number of neurons, learning rate, the maximum number of epochs and number of folds are discussed in this section.

a) *Number of neurons*: The neurons and the connections between the neurons are essential features of a neural network. The number of neurons plays a vital role in the performance of the neural network. Too few neurons may not completely describe the dynamics of the system, and too many neurons can increase the complexity of the network. Therefore, an optimal selection of a number of a neuron is one of the most important aspects of neural network modeling. In MATLAB DANN toolbox, each neuron is associated with the sigmoid function with the range [0, 1].

b) *Learning rate:* The learning rate determines the rate of learning of gradient-based learning algorithms like BPTT and RTRL. The range of learning rate is [0, 1] and determines the converging efficiency while learning. The very small value of learning rate will slow down the learning algorithm and may require a large number of epochs to converge to a solution. Whereas the high value of learning rate can converge quickly, but it might have large variations and fluctuations in MSE of a training data.

c) *Maximum number of the epoch*: In MATLAB DANN toolbox, there are two stopping criterions. One of them is validation check, which is already discussed. Another way of stopping the training is the maximum number of epochs. A user can select a maximum number of epochs for the training using DANN Menu.

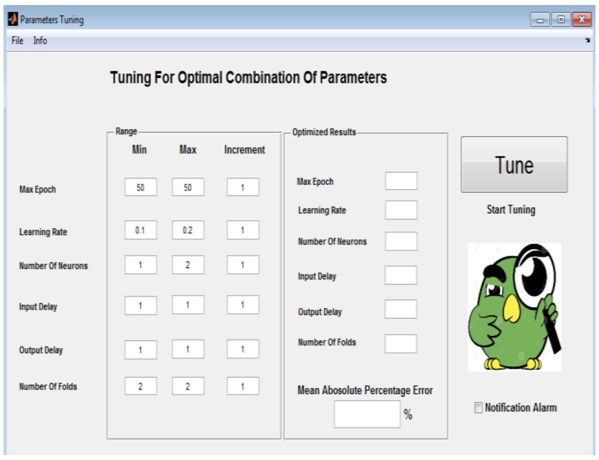
d) *Number of folds*: The number of folds is a parameter for BPTT learning. The default selection is '3', which is the minimum possible value that can be selected for a given number of folds.

7) *Past inputs and outputs*: In applications involving prediction of time-series, the current output depends on the previous inputs and outputs. MATLAB DANN toolbox allows a user to select a number of previous inputs and previous outputs as additional inputs to find the output at the current time. By default, if the values are selected as '0' for both input and output, MATLAB DANN toolbox will use one previous output from each neuron.

8) *Additional parameters*: In EKF learning algorithm, a user must assign three more parameters for learning, which are Sigma_U, Sigma_W, and Sigma_O in MATLAB DANN toolbox. These parameters are tuning parameters for the output of each neuron as a state, weights as a state and output of the network respectively. These parameters are responsible for Kalman gain calculation for the states (i.e. output of neuron and weight) and determine the update of the output of each neuron and the weight connections between the neurons. As the simulation stops, the parameters, weights and other information regarding the simulation are saved in the workspace in MATLAB®.

B. Parameter Tuning

In any implementation of ANN, tuning of parameters is one of the biggest challenges. The optimal selection of network parameters can only lead to a good model. Contrary to existing MATLAB® Neural Network Toolbox, MATLAB DANN toolbox has a facility to tune the parameters optimally. In DANN Main, if you click on Tuning button, Parameter Tuning window will open as shown in Fig. 4. The optimal tuning is based on the grid search method, and optimality is evaluated using Mean Absolute Percentage Error (MAPE). In the left panel of the window, a user can assign lower limit, higher limit and an increment to each parameter and start tuning. At the end of the tuning, optimal values of the parameters are displayed in the right panel of the window with minimum MAPE. Usually, parameter tuning takes a longer time before it completes, so MATLAB DANN toolbox provides an option to get notification alarm. It is to be noted that a user must upload data, select learning algorithm, decide to or not to include bias and validation check before starting the tuning process. Thus, obtained optimal parameters can be used for training the model.



Range	Min	Max	Increment
Max Epoch	50	50	1
Learning Rate	0.1	0.2	1
Number Of Neurons	1	2	1
Input Delay	1	1	1
Output Delay	1	1	1
Number Of Folds	2	2	1

Fig. 4: Parameter Tuning window of MATLAB DANN toolbox that allows a user to tune the optimal parameters based on grid search method.

C. Plot Menu

Plot Menu window pops-up when the simulation is completed as shown in Fig. 5. It consists of five different types of plots, which are performance plot, regression plot, prediction plot, parameter plot, and error plot.

1) *Performance plot*: It shows the MSE for training data set and validation error for each epoch.

2) *Regression plot*: It compares the target output and model prediction in terms of squared correlation coefficient such that '0' meaning not related at all and '1' meaning highly correlated to each other.

3) *Prediction plot*: It shows the test data and model prediction with MAPE between them.

4) *Parameter plot*: It shows the states of five different randomly chosen weights at different epochs. The analysis using parameter plot is very efficient if you are working with some system identification problems. In that case, one can visualize how the weights change with epochs. The steady state values of the weights after some epochs are the model parameters in typical system identification problems.

5) *Error plot*: It shows the error between the target value and the model prediction for each test samples.

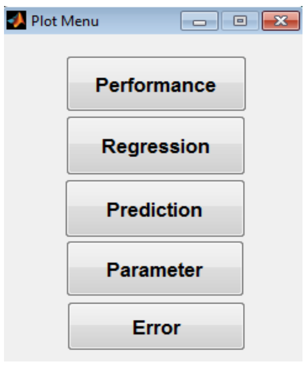


Fig. 5: Plot Menu of MATLAB DANN toolbox with different plots for the analysis of the model.

D. *Additional information*

The MATLAB DANN toolbox has additional help options for the users. A user can get general information in Q&A section inside the Help Window. With a right-click in any parameter name, action buttons or selection options, a prompt help window related to that expression will pop up.

E. *Installing the MATLAB DANN toolbox*

The first step in installing MATLAB DANN toolbox is to download installation file found in this link. Double-click in the downloaded installation file will direct to the installation process in the MATLAB®. It is recommended to have a MATLAB® version 2014 or later.

III. Case Studies

In this section, the usage of the MATLAB DANN toolbox in three different practical applications is discussed. These three different cases use the data set from an experimental flow rig, and example data sets from MATLAB® Neural Network Toolbox. To give a better understanding in analyzing the simulation results, different sets of plots are investigated under these cases.

A. *Case I: BPTT learning algorithm for flow measurement*

In drilling operations, the flow rates of drilling mud at inflow and outflow positions can be used to detect kick and fluid loss. An open channel flow loop is available at University College of Southeast Norway (USN) for the study of outflow measurement. The data set with three level measurements as inputs and a flow measurement as the single output are taken from the flow loop for the analysis of BPTT learning algorithm in MATLAB DANN toolbox. Fig. 6 and Fig. 7 show the regression plot and prediction plot for flow estimation using BPTT learning algorithm in the toolbox. The simulation results show that the BPTT learning algorithm provided by MATLAB DANN toolbox is capable of mapping the inputs and outputs with high accuracy.

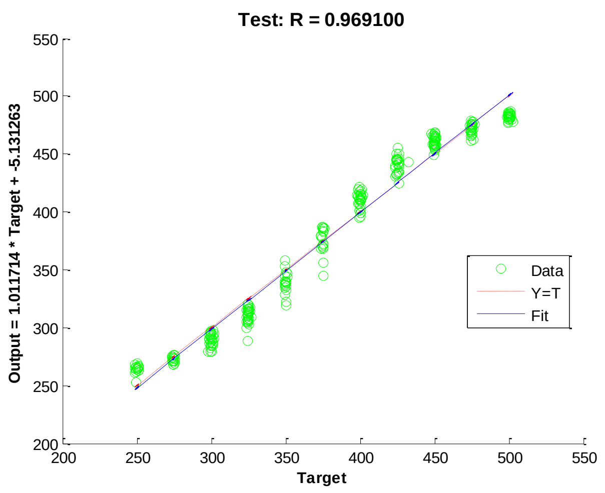


Fig. 6: The regression plot for flow measurement using BPTT learning algorithm, with a correlation of 96% between the target values and the model prediction values. Data set from an experimental flow rig at USN.

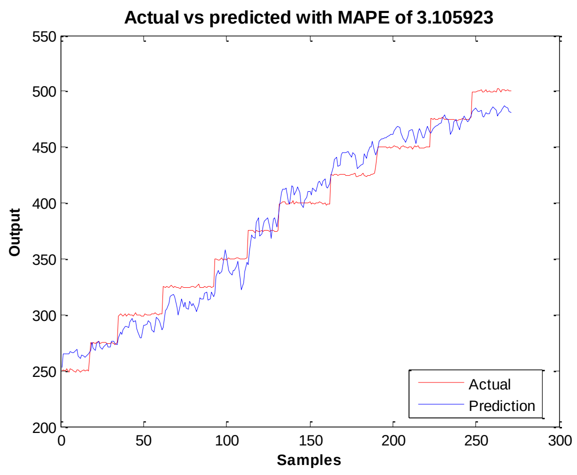


Fig. 7: The prediction plot for flow measurement using BPTT learning algorithm with a MAPE of 3.1 %. Data set from an experimental flow rig at USN.

B. Case II: EKF learning algorithm for temperature measurement

To analyze the performance of EKF learning algorithm, an example data set provided by MATLAB® Neural Network Toolbox is used. The data set of a liquid-saturated steam heat exchanger consists of time-series liquid flow rate and liquid outlet temperature, used as input and output to the ANN feedback network respectively. Fig. 8 and Fig. 9 show the performance plot and prediction plot for fully connected RNN with EKF learning algorithm. The learning algorithm has an early stop at 8 epochs due to the validation check with MSE of 0.015. The low value of MAPE in prediction plot shows that the EKF learning algorithm with validation check is able to generalize the model and avoid over-fitting.

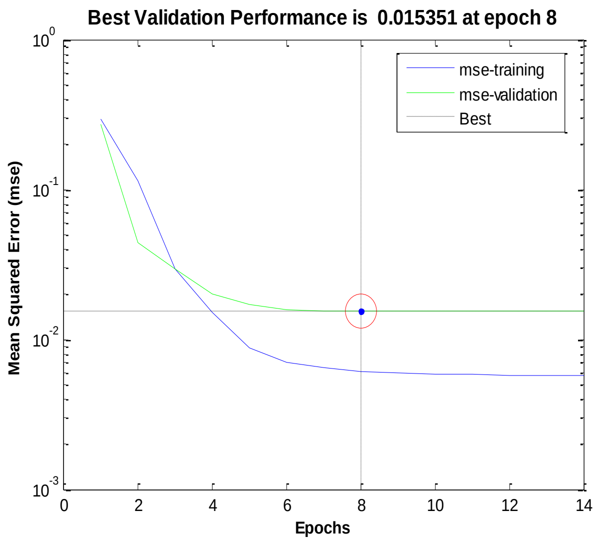


Fig. 8: The performance plot for temperature measurement using EKF learning algorithm. The best validation performance is 0.015351 at epoch 8. Data set from MATLAB® Neural Network Toolbox.

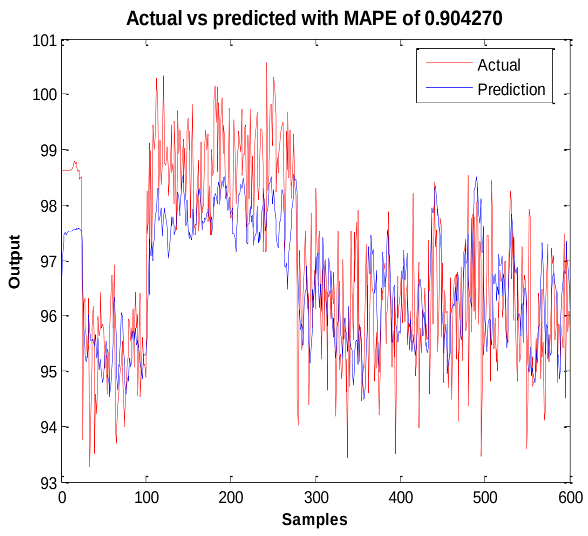


Fig. 9: The prediction plot for temperature measurement using EKF learning algorithm with a MAPE of 0.9 %. Data set from MATLAB® Neural Network Toolbox.

C. Case III: RTRL learning algorithm for mortality prediction

Another example data set provided by MATLAB® Neural Network Toolbox is used to investigate the performance of RTRL learning algorithm. The data set is a Pollution mortality data set that consists of eight input variables (Temperature, Relative humidity, Carbon monoxide, Sulfur dioxide, Nitrogen dioxide, Hydrocarbons, Ozone, and Particulates) and total mortality as an output variable. Fig. 10 to Fig. 13 shows the simulation results for mortality prediction using RTRL learning algorithm using MATLAB DANN toolbox.

The parameter plot as shown in Fig. 10 shows the states of randomly chosen weights of the network. As discussed in Section II, it takes longer time for the weights to converge to a steady state when using RTRL.

The regression plot as in Fig. 11 illustrates that the predictions using RTRL are highly correlated with the target values with a correlation of 92%. The MAPE between the predicted values and target values are 2.93% as shown in Fig. 12. The error in each sample is shown in the error plot in Fig. 13.

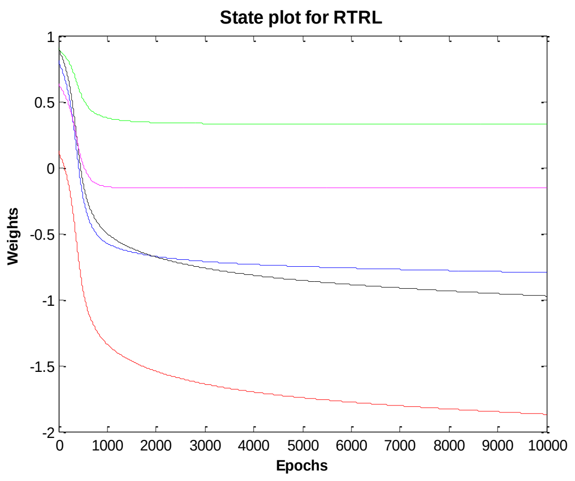


Fig. 10: The state plot for mortality time-series prediction using RTRL learning algorithm. Data set from MATLAB® Neural Network Toolbox.

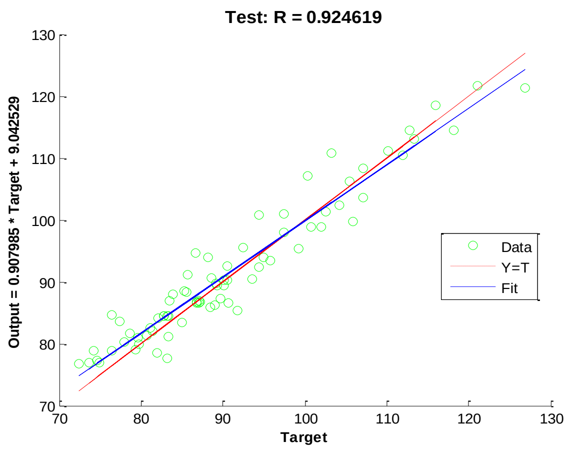


Fig. 11: The regression plot for mortality time-series prediction using RTRL learning algorithm with 92% correlation between target values and model predictions. Data set from MATLAB® Neural Network Toolbox.

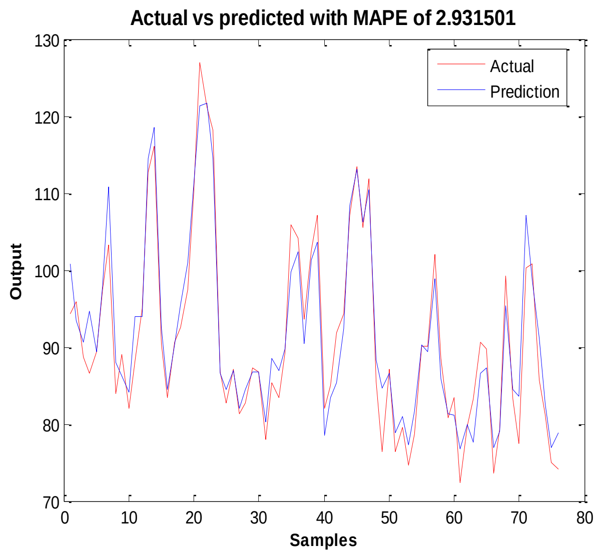


Fig. 12: The prediction plot for mortality time-series prediction using RTRL learning algorithm with MAPE of 2.93%. Data set from MATLAB® Neural Network Toolbox.

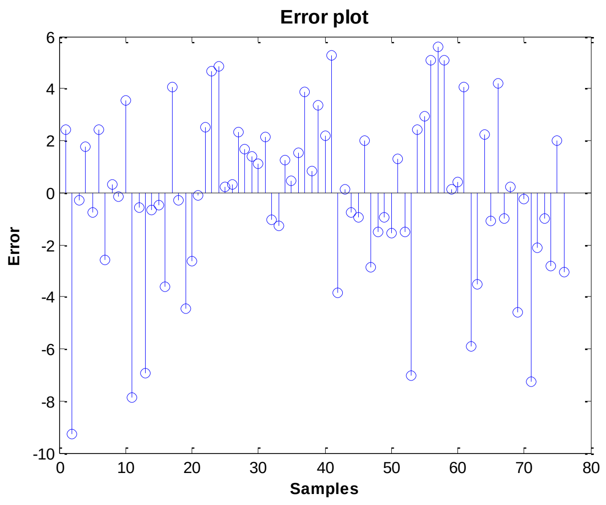


Fig. 13: The error plot for mortality time-series prediction using RTRL learning algorithm with 9 units as the highest error in the test samples. Data set from MATLAB® Neural Network Toolbox.

IV. Conclusion

The existing MATLAB® Neural Network Toolbox has a possibility to use both static and dynamic neural networks. However, it is not possible directly use the toolbox to fully connected recurrent neural networks. For this reason, this study presents the Dynamic Artificial Neural Network MATLAB toolbox that gives an opportunity to use the fully connected neural network for time-series predictions. The toolbox consists of three different learning algorithms, where Back Propagation Through Time (BPTT) is an offline learning algorithm, Real Time Recurrent Learning (RTRL) and Extended Kalman Filter (EKF) learning algorithm are online learning algorithms. Main details and guides for installing and using the developed toolbox are presented in this paper.

To demonstrate the features of the MATLAB DANN toolbox, three different practical problems are considered using three different learning algorithms. The simulation studies presented in this paper show that the developed toolbox can be used in applications involving time-series predictions. In addition, the developed toolbox has a dedicated option for the optimal tuning of parameters.

This work is meant for academic use with particular focus on the students using the existing MATLAB® Neural Network Toolbox. In future, other different learning algorithms can be included in the developed toolbox with some programming efforts.

Acknowledgment

The Ministry of Education and Research of the Norwegian Government is funding Khim Chhantyal's PhD studies at University College of Southeast Norway (USN). The authors at USN appreciate the collaboration with and support from STATOIL for the rig used in the current studies for generation of time series of flow rates.

V. References

[1] E. O. Dijk, "Analysis of Recurrent Neural Networks with application to speaker independent phoneme recognition," University Twente, Enschede, The Netherlands, 1999.

[2] D. Howard, B. Mark, "Neural Network Toolbox User's Guide © Copyright," The MathWorks, Inc., 2004.

[3] R. J. Williams, "Some observations on the use of the Extended Kalman Filter as a Recurrent Network Learning algorithm," College of Computer Science, Northeastern University, Boston, 1992.

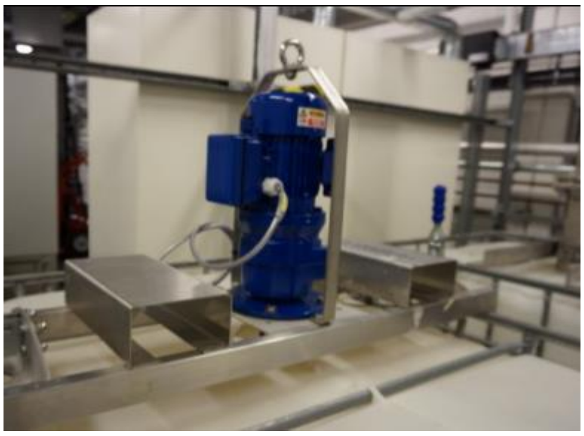
[4] S. Haykin, "Neural Networks and Learning Machines," 3rd ed., ISBN: 978-0-13-147139-9, Upper Saddle River, New Jersey 07458,: Pearson Education, Inc., 2009, p.808.

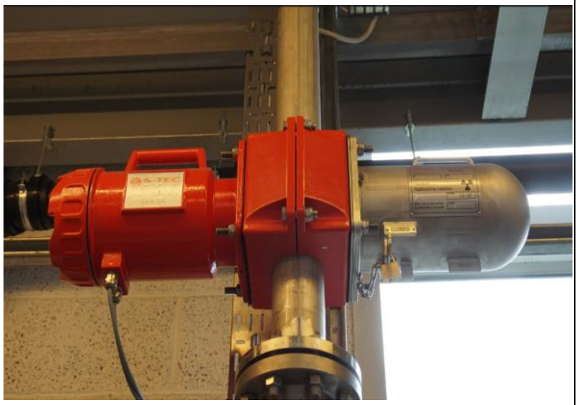
[5] M. W. Mak, K. W. Ku, Y. L. Lu, "On the improvement of the real time recurrent learning algorithm for Recurrent Neural Networks," in Neurocomputing, vol. 24, Elsevier, 1999, p.13-36.

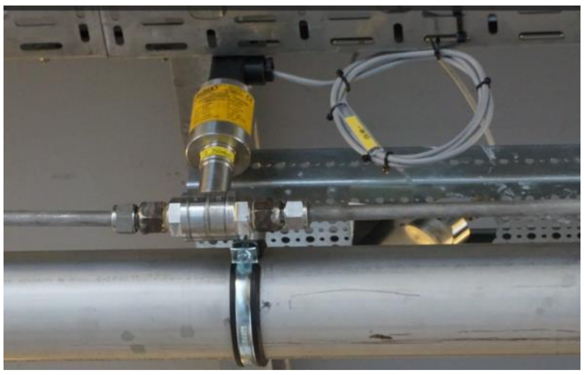
[6] D. P. Mandic, J. A. Chambers, "A normalised real-time recurrent learning algorithm," in Signal Processing, vol. 80, Elsevier, 2000, p.1909-1916.

[7] D. B. Budik, "A resource efficient localized Recurrent Neural Network architecture and learning algorithm," University of Tennessee, USA, 2006.

Appendix D: Picture of the Venturi-rig

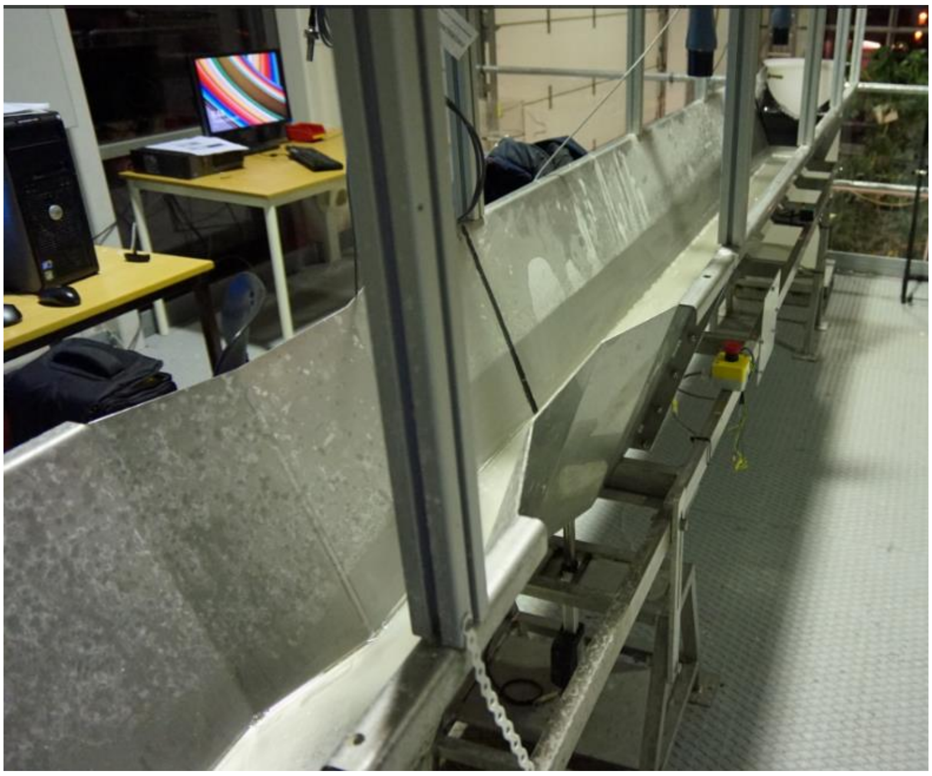
Venturi section	Exit of the Venturi section
Overview of the tank 1#	**Overview of the tank #2**
Mixture to create the fluids	**Water pipe to clean the tank**
	

Tanks for fluid 1 and 2 respectively	**Pressure transmitter**
Temperature transmitter	**Coriolis flow meter (Promass 801)**
Coriolis flow meter (Promass 63)	**Gamma transmitter**
	

Pressure differential transmitter	Ultrasonic level transmitter
	

Appendix E: Venturi-rig Tutorial

Venturi-rig Tutorial



Khim Chhantyal

Minh Hoang

2016

E.1 Introduction

The document is created as a guideline for those who want to run the Venturi-rig for the first time. The document covers almost everything that is needed to handle the rig. The Health, Safety and Environment (HSE) part of this document is partially taken from a Bachelor's Thesis group, Spring 2015, University College of Southeast Norway.

E.2 Precautions (HSE)

Before doing anything on the rig, please read through this HSE section to get information about the potentially dangerous situations that may occur, and precautions to protect yourself.

E.2.1 Drilling fluid

The fluid that is used is a mixture of potassium carbonate, xanthan gum, and water. Under normal condition, there should not be any danger related to this fluid.

E.2.2 Health concerns

Be careful not to swallow in or get the fluid in your eyes during the experiments. If the accident has already happened that you've swallowed some of it, your stomach may be upset for a while. If in case, your eyes get in contact with the fluid, irritation may appear, and it is highly recommended that you contact a doctor if this doesn't go away over some time.

E.2.3 Environmental concerns

Watch out for leakages when you are working. Spilled fluid should be gathered into a deposition if possible. The fluid is not dangerous, and the remains of the fluid should be properly cleaned with water. When cleaning, have in mind that the spilled fluid can cause the floor to be slippery.

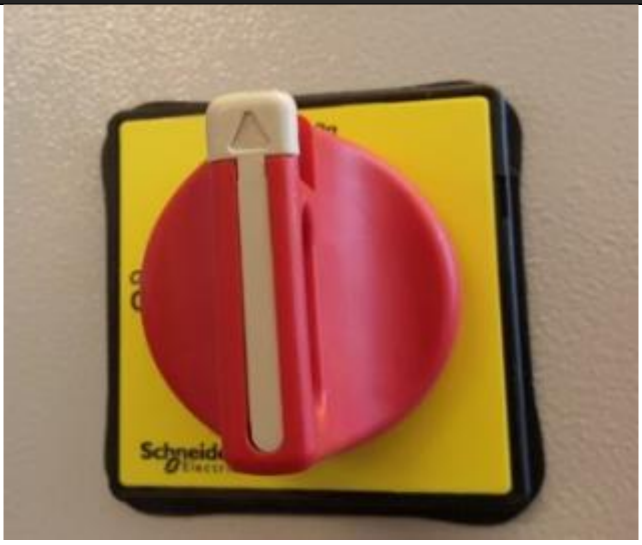
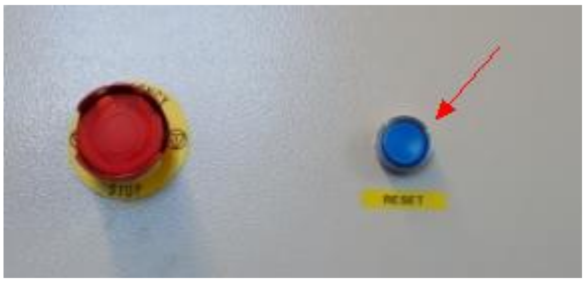
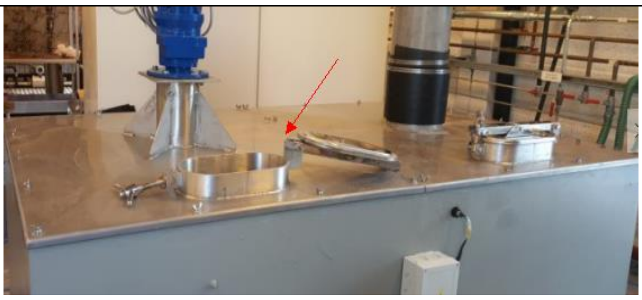
E.2.4 Safety concerns

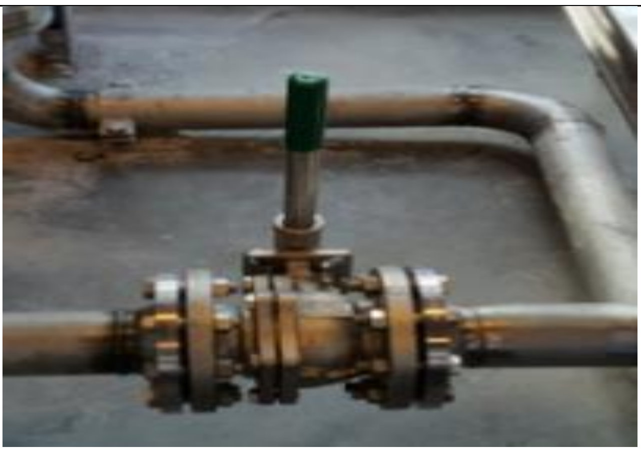
Whenever the rig is running and if there is a possibility of direct contact with the fluid, wear safety glasses. If you are going to do some handling with the fluid, for example: changing the fluid, making new fluid, cleaning the tank, etc., remember to use gloves and have proper clothes.

E.3 Running the rig

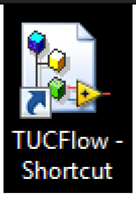
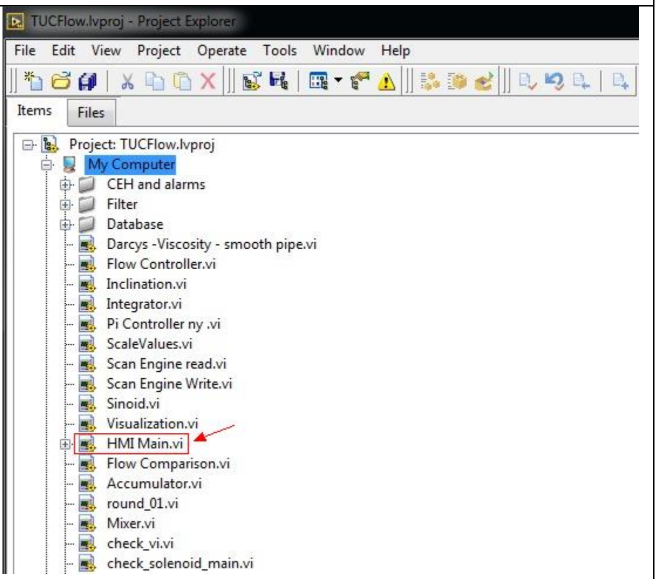
This section will explain what kind of preparations that need to be done before you start the rig and how to use the software for control purpose.

E.3.1 Preparation

Steps	Descriptions	Pictures
1	Turn on the power This switch is on the left side of the security cabinet (on the first floor). Make sure the pointer is upwards (see picture) Then click on the blue reset button (see picture)	 
2	Open one of the cover See picture *Else, there is a possiblity of damaging the tank while running the Main-Pump.*	

3	Close the valve When the valve is closed, the switch should be in 90 degrees with the pipe (see picture) *We close this valve because we don't want the flow to go back to the tank when running it*	

E.3.2 Run the program

Steps	Descriptions	Pictures
1	Open the LabVIEW project file *Double-click on "TUCFlow - Shortcut" on the desktop*	 TUCFlow - Shortcut
2	Open the Main file *Click on "HMI Main.vi"*	

TUCFlow.lvproj - Project Explorer

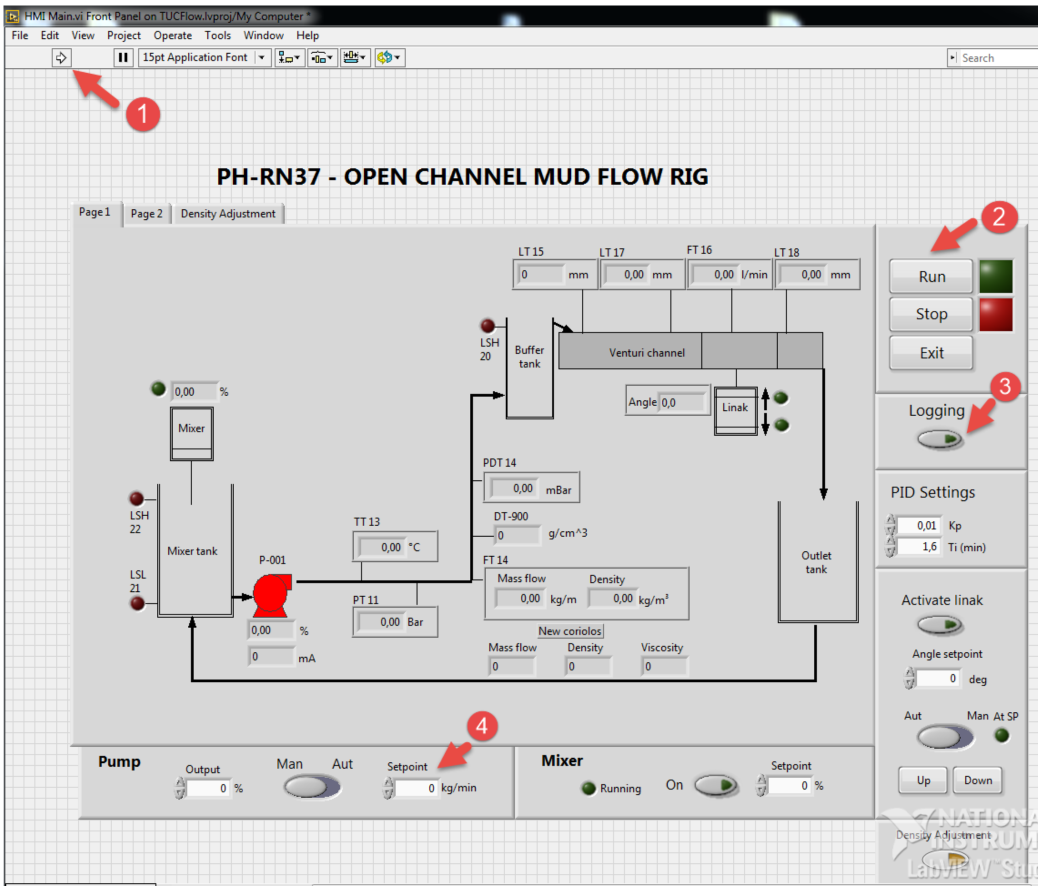
File Edit View Project Operate Tools Window Help

Items Files

- Project: TUCFlow.lvproj
  - My Computer
    - CEH and alarms
    - Filter
    - Database
    - Darcys -Viscosity - smooth pipe.vi
    - Flow Controller.vi
    - Inclination.vi
    - Integrator.vi
    - Pi Controller ny .vi
    - ScaleValues.vi
    - Scan Engine read.vi
    - Scan Engine Write.vi
    - Sinoid.vi
    - Visualization.vi
    - HMI Main.vi
    - Flow Comparison.vi
    - Accumulator.vi
    - round_01.vi
    - Mixer.vi
    - check_vi.vi
    - check_solenoid_main.vi

3

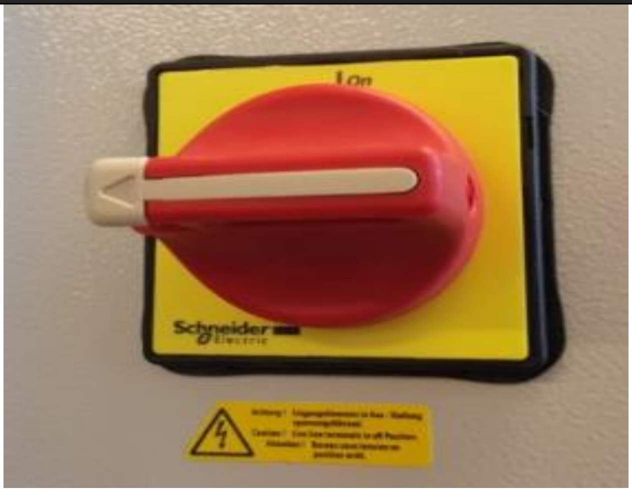
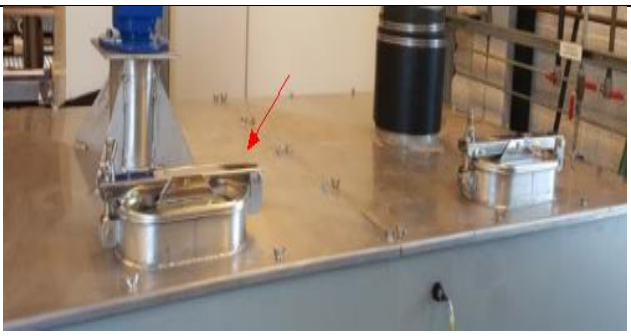
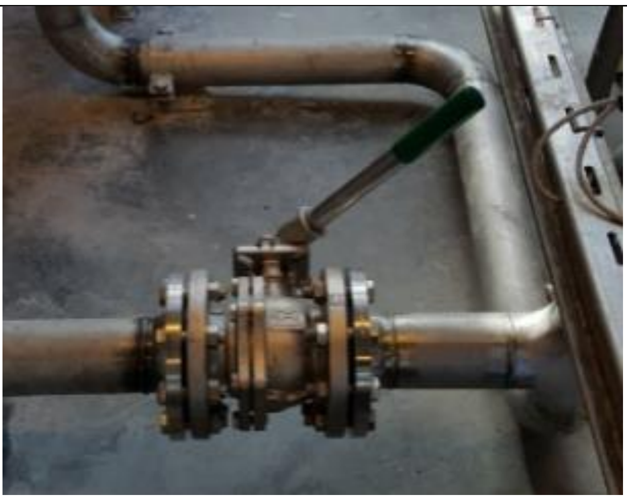
Run the program



1. Click on the arrow to activate the program.
2. Click on the button “Run” to start the experiment.
3. Click on logging button when you want to save data to the database.
4. Here you can change the set-point of the flowrate.

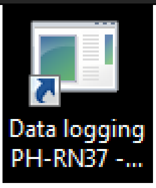
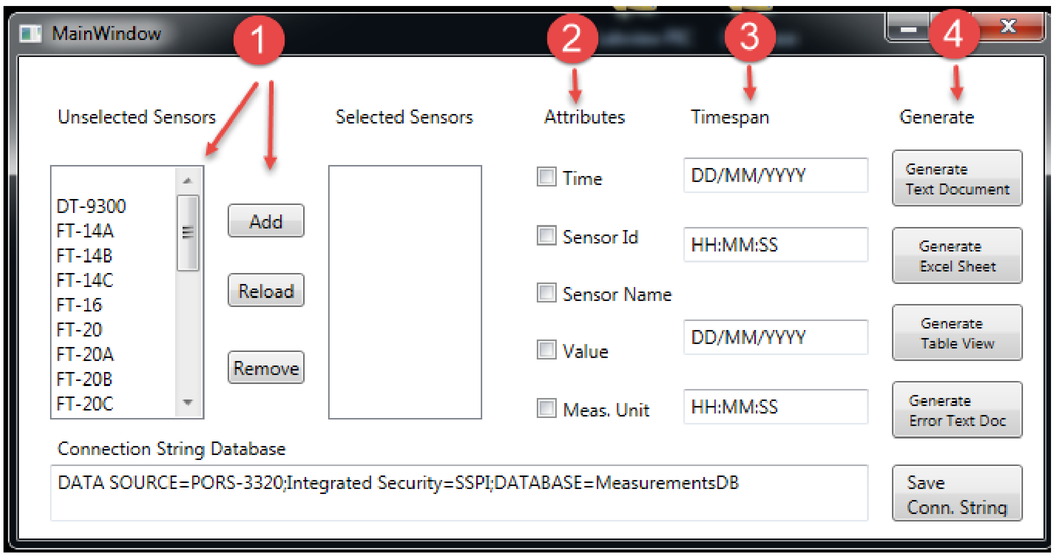
E.3.3 Finished with the experiments?

Do the following steps

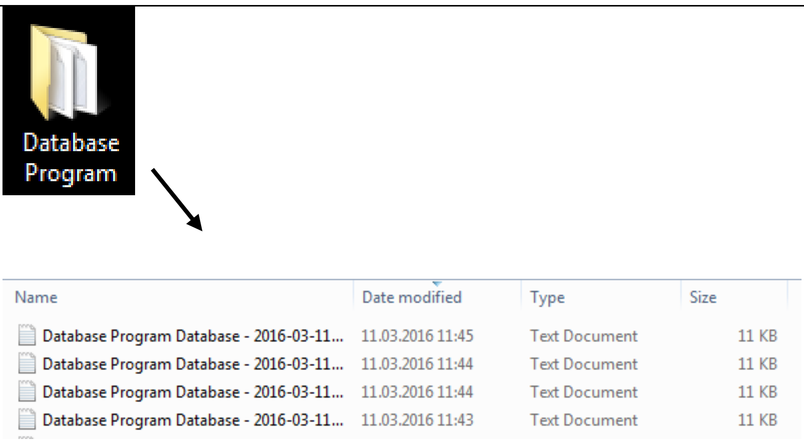
Steps	Descriptions	Pictures
1	Turn of the power This switch is on the left side of the security cabinet (on the first floor). Make sure the pointer is pointing left (see picture)	
2	Close the cover See picture *We "must" close it.*	
3	Open the valve When the valve is opened, the switch should be something like in the picture. *We open this valve because we want the fluids to go back to the tank.*	

E.4 Get data from database

To download data from the database, perform these steps

Steps	Descriptions	Picture
1	Open the database program *Double-click on the icon "Data logging PH-RN37 – Shortcut" on the desktop*	
2	Select your data and generate file  1. Select the sensors you want to include in the data file and then click on "Add". 2. Select the attributes you want to include in the data file. 3. Select the timespan of your data. 4. Generate the data file, here you can choose between 4 options.	

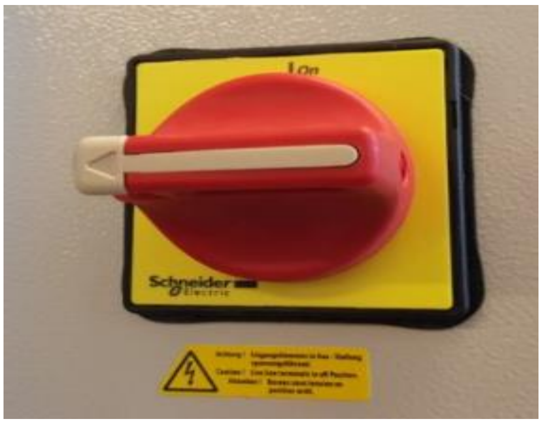
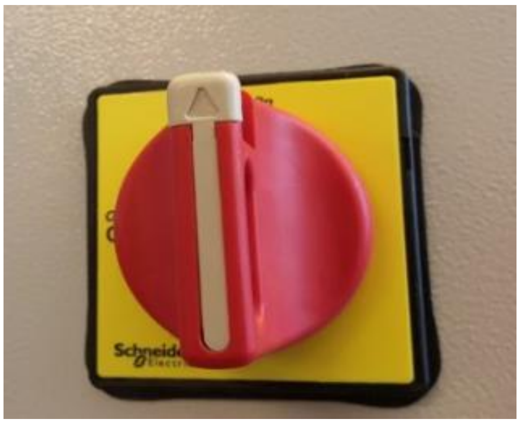
DATA SOURCE=PORS-3320;Integrated Security=SSPI;DATABASE=MeasurementsDB

3	Get the generated data file The generated file will be stored in a folder called "Database Program". Double click on the folder and pick the newest file.	 Name | Date modified | Type | Size Database Program Database - 2016-03-11... | 11.03.2016 11:45 | Text Document | 11 KB Database Program Database - 2016-03-11... | 11.03.2016 11:44 | Text Document | 11 KB Database Program Database - 2016-03-11... | 11.03.2016 11:44 | Text Document | 11 KB Database Program Database - 2016-03-11... | 11.03.2016 11:43 | Text Document | 11 KB

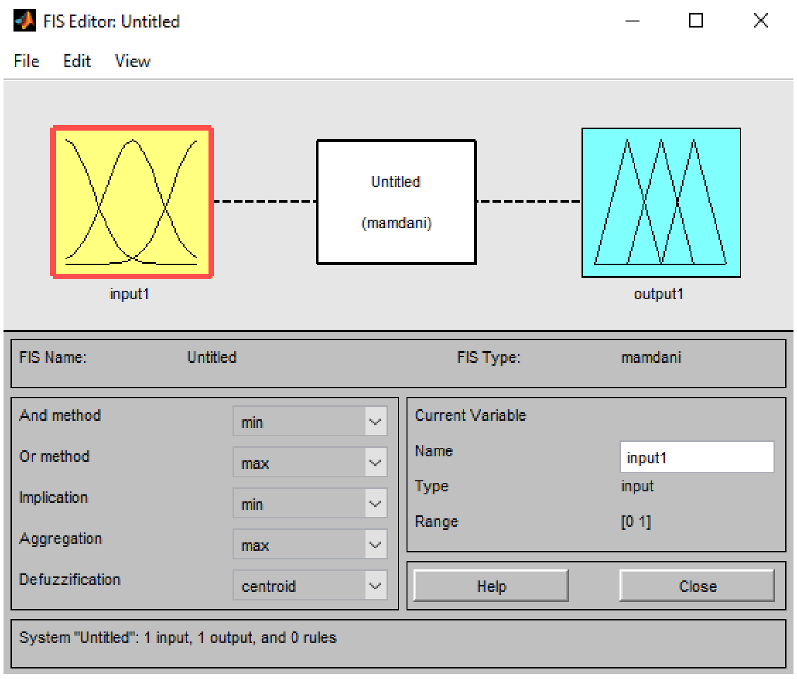
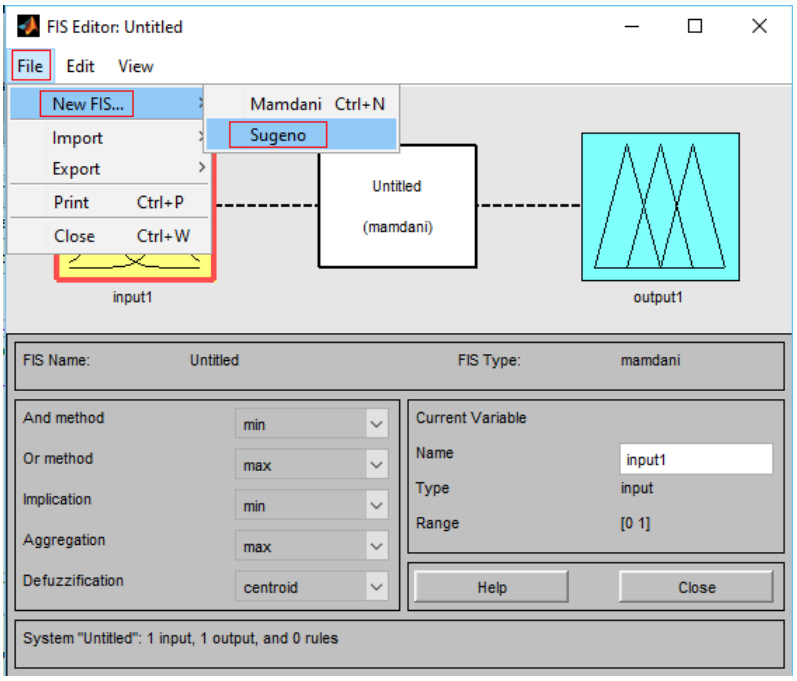
E.5 Troubleshooting

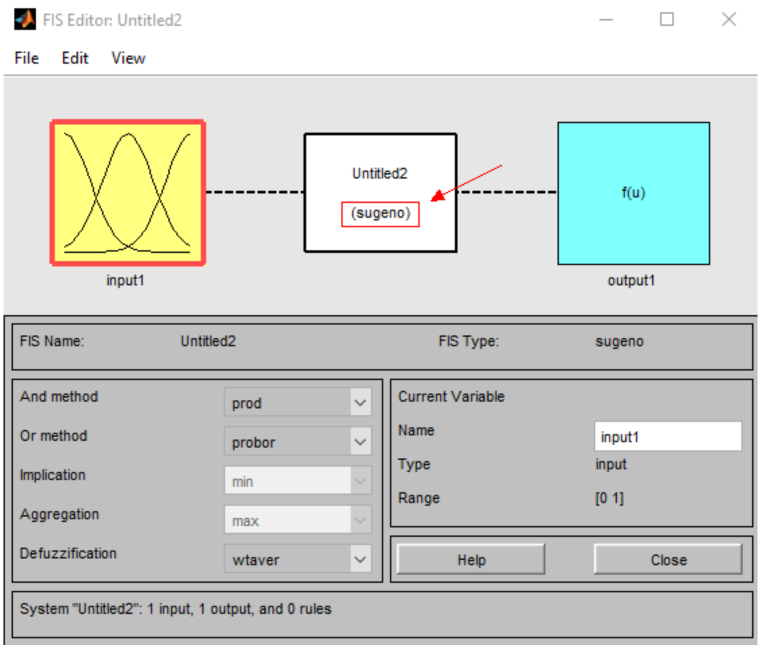
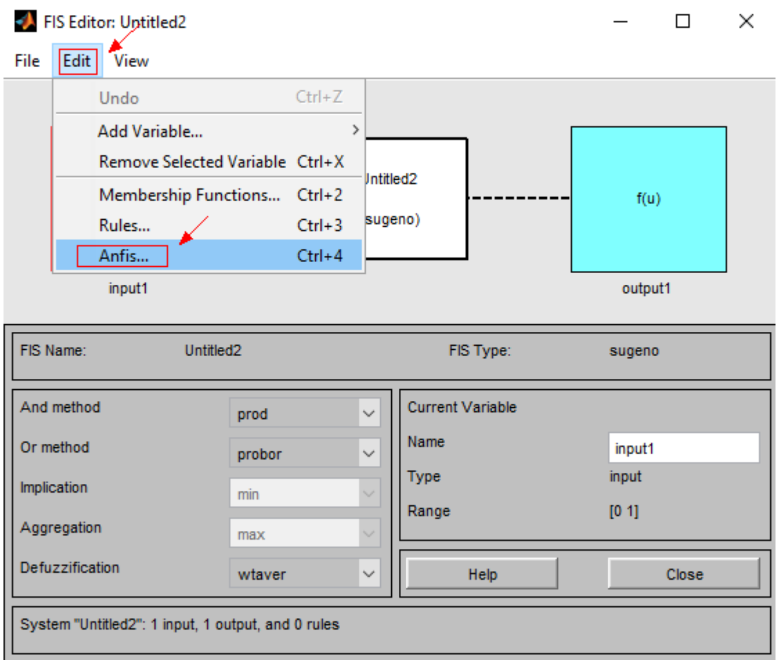
Sometimes there can be problems with running the LabVIEW program. The button will not respond or that the rig will not respond to the commands from LabVIEW. If this happens, then do the following steps.

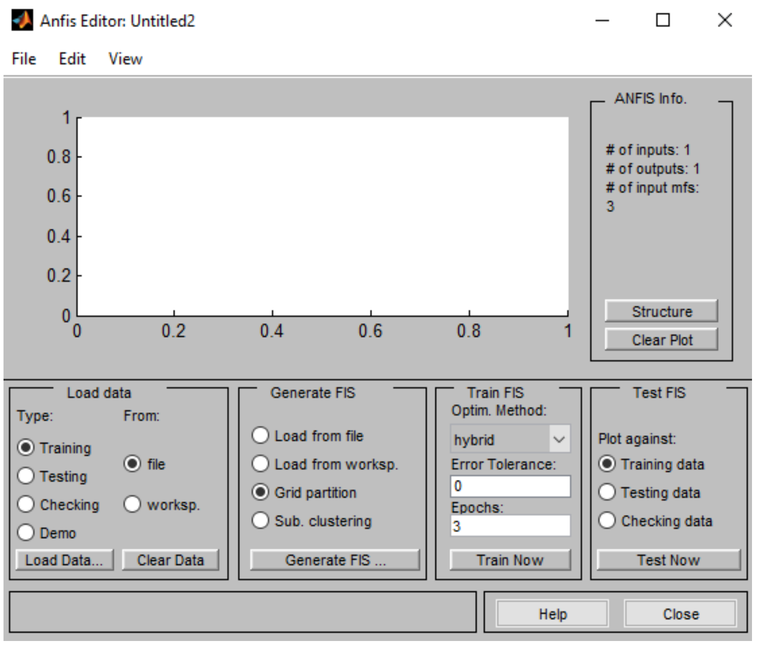
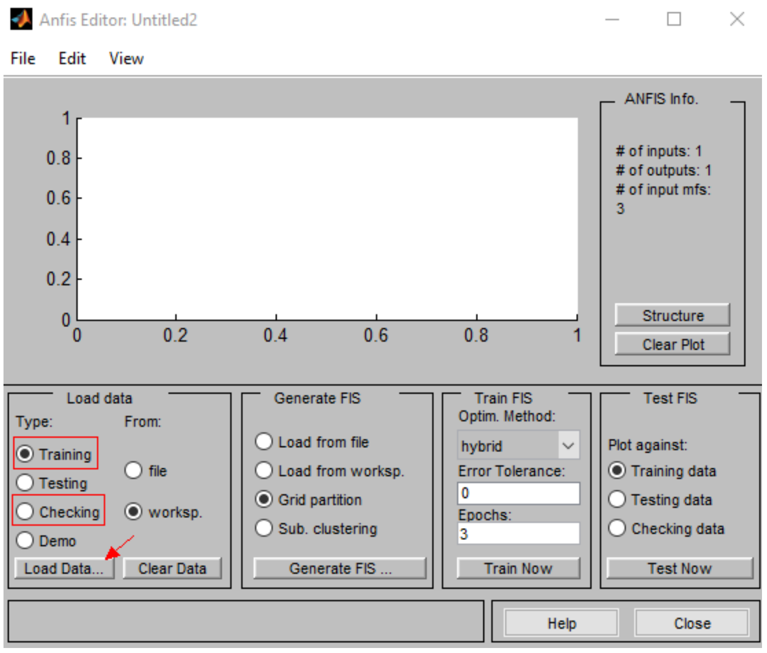
1. Close the LabVIEW program.
2. Restart the Venturi rig by turning the power off and then on.

Power Off	**Power On**
	

Appendix F: How to use Sugeno Anfis

Steps	Screenshots
Open fuzzy toolbox: Write “fuzzy” in the command window	 FIS Editor: Untitled File Edit View input1 Untitled (mamdani) output1 Help Close FIS Name: Untitled FIS Type: mamdani And method: min Or method: max Implication: min Aggregation: max Defuzzification: centroid Current Variable Name: input1 Type: input Range: [0 1] System "Untitled": 1 input, 1 output, and 0 rules
Open new FIS file “Sugeno”: Click on “File” -> “New FIS…” -> “Sugeno”	 FIS Editor: Untitled File Edit View New FIS... Import Export Print Ctrl+P Close Ctrl+W Mamdani Ctrl+N Sugeno input1 Untitled (mamdani) output1 Help Close FIS Name: Untitled FIS Type: mamdani And method: min Or method: max Implication: min Aggregation: max Defuzzification: centroid Current Variable Name: input1 Type: input Range: [0 1] System "Untitled": 1 input, 1 output, and 0 rules

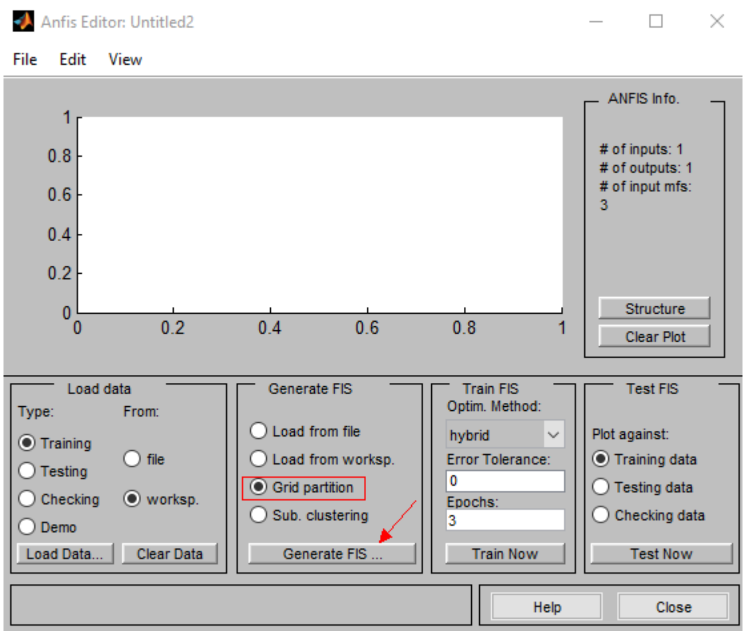
A new FIS editor with “Sugeno” will appear. Remember to have at least two inputs to make the Anfis work. This can be done by clicking on “Edit” –> “Add variable” -> “Input”.	 FIS Editor: Untitled2 File Edit View Untitled2 (sugeno) f(u) input1 output1 Help Close FIS Name: Untitled2 FIS Type: sugeno And method: prod Or method: probor Implication: min Aggregation: max Defuzzification: wtaver Current Variable Name: input1 Type: input Range: [0 1] System "Untitled2": 1 input, 1 output, and 0 rules
Open ANFIS: Click on “Edit” -> “Anfis…”	 FIS Editor: Untitled2 File Edit View Undo Ctrl+Z Add Variable... Remove Selected Variable Ctrl+X Membership Functions... Ctrl+2 Rules... Ctrl+3 Anfis... Ctrl+4 Untitled2 sugeno) f(u) input1 output1 Help Close FIS Name: Untitled2 FIS Type: sugeno And method: prod Or method: probor Implication: min Aggregation: max Defuzzification: wtaver Current Variable Name: input1 Type: input Range: [0 1] System "Untitled2": 1 input, 1 output, and 0 rules

An Anfis Editor will appear	 Anfis Editor: Untitled2 File Edit View ANFIS Info. # of inputs: 1 # of outputs: 1 # of input mfs: 3 Structure Clear Plot Load data Type: [✓ Training] Testing Checking Demo From: [✓ file] worksp. Load Data... Clear Data Generate FIS Load from file Load from worksp. [✓ Grid partition] Sub. clustering Generate FIS ... Train FIS Optim. Method: hybrid Error Tolerance: 0 Epochs: 3 Train Now Test FIS Plot against: [✓ Training data] Testing data Checking data Test Now Help Close
Load training and checking data: Check on "training" and then click on "Load data". Repeat this step again, but this time check on "checking" and then click on "Load Data..."	 Anfis Editor: Untitled2 File Edit View ANFIS Info. # of inputs: 1 # of outputs: 1 # of input mfs: 3 Structure Clear Plot Load data Type: [✓ Training] Testing Checking Demo From: file [✓ worksp.] Load Data... Clear Data Generate FIS Load from file Load from worksp. [✓ Grid partition] Sub. clustering Generate FIS ... Train FIS Optim. Method: hybrid Error Tolerance: 0 Epochs: 3 Train Now Test FIS Plot against: [✓ Training data] Testing data Checking data Test Now Help Close

Generate FIS:

Use “Grid partition” and click on “Generate FIS…”

(We use grid partition because this is better than Sub. Clustering. More details will come soon)



Anfis Editor: Untitled2

ANFIS Info.

# of inputs: 1
# of outputs: 1
# of input mfs:
3

Load data — Type: [✓ Training] Testing Checking Demo; From: file [✓ worksp.]

Generate FIS: Load from file, Load from worksp., [✓ Grid partition], Sub. clustering

Train FIS — Optim. Method: hybrid; Error Tolerance: 0; Epochs: 3

Test FIS — Plot against: [✓ Training data] Testing data Checking data

Train FIS:

<u>**Choose**</u>:

Optim. Method: “hybrid”

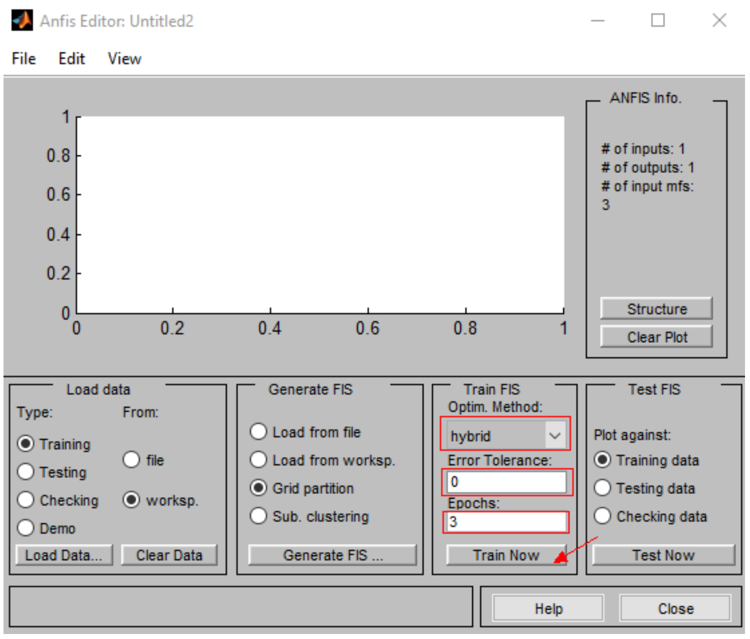
(hybrid = backpropagation & least squre)

Error Tolerance: 0

(We want to have the error small as possible)

Epochs: use the number that makes the epoch error stable.

Click on “Train Now”



Anfis Editor: Untitled2

ANFIS Info.

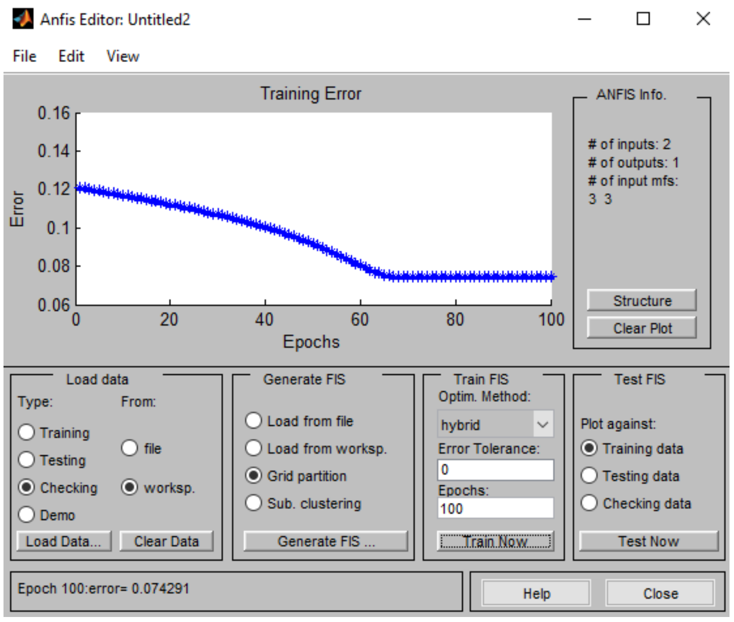
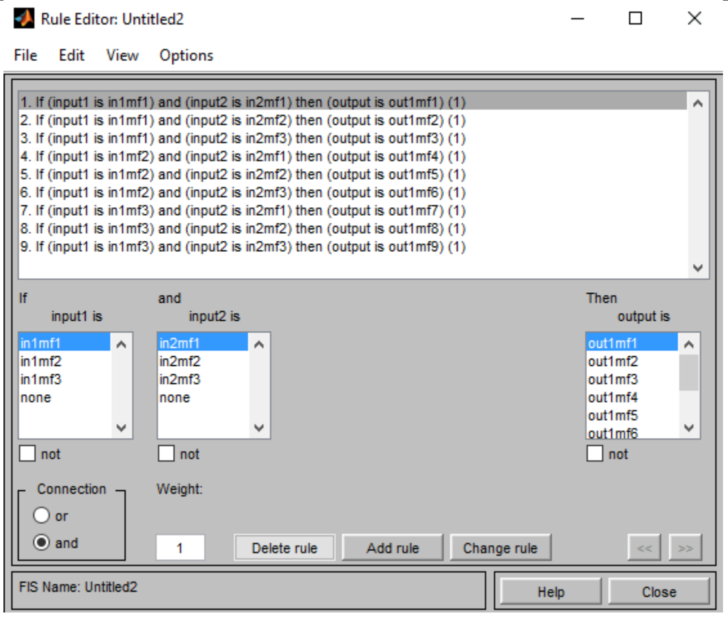
# of inputs: 1
# of outputs: 1
# of input mfs:
3

Load data — Type: [✓ Training] Testing Checking Demo; From: file [✓ worksp.]

Generate FIS: Load from file, Load from worksp., [✓ Grid partition], Sub. clustering

Train FIS — Optim. Method: hybrid; Error Tolerance: 0; Epochs: 3

Test FIS — Plot against: [✓ Training data] Testing data Checking data

Training error: When clicking on “Train now”, an error value will show how much the estimated value deviate from the real value.	
Rules are automatically created After you have trained the FIS, the rules will be created automatically.	

Anfis Editor: Untitled2

ANFIS Info.
# of inputs: 2
# of outputs: 1
# of input mfs:
3 3

Load data — Type: ( ) Training ( ) Testing (●) Checking ( ) Demo — From: ( ) file (●) worksp.

Generate FIS: ( ) Load from file ( ) Load from worksp. (●) Grid partition ( ) Sub. clustering

Train FIS — Optim. Method: hybrid — Error Tolerance: 0 — Epochs: 100

Test FIS — Plot against: (●) Training data ( ) Testing data ( ) Checking data

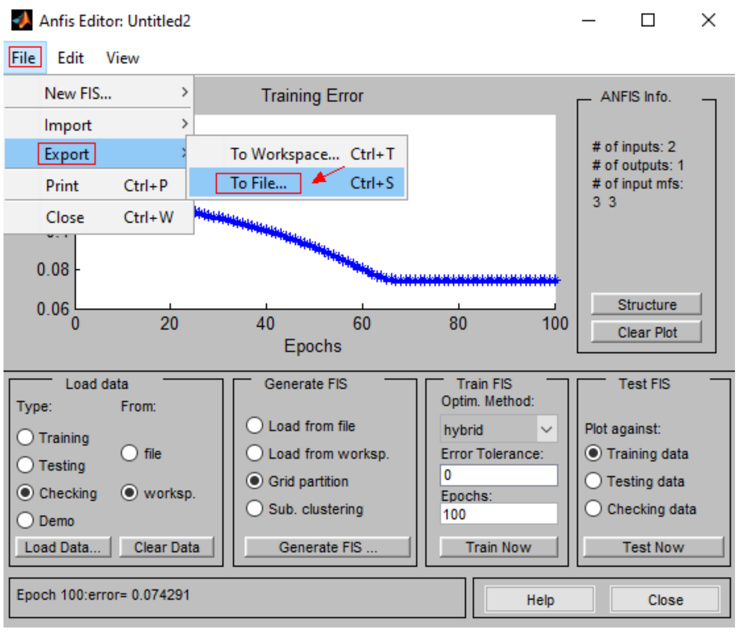
```
Epoch 100:error= 0.074291
```

Rule Editor: Untitled2

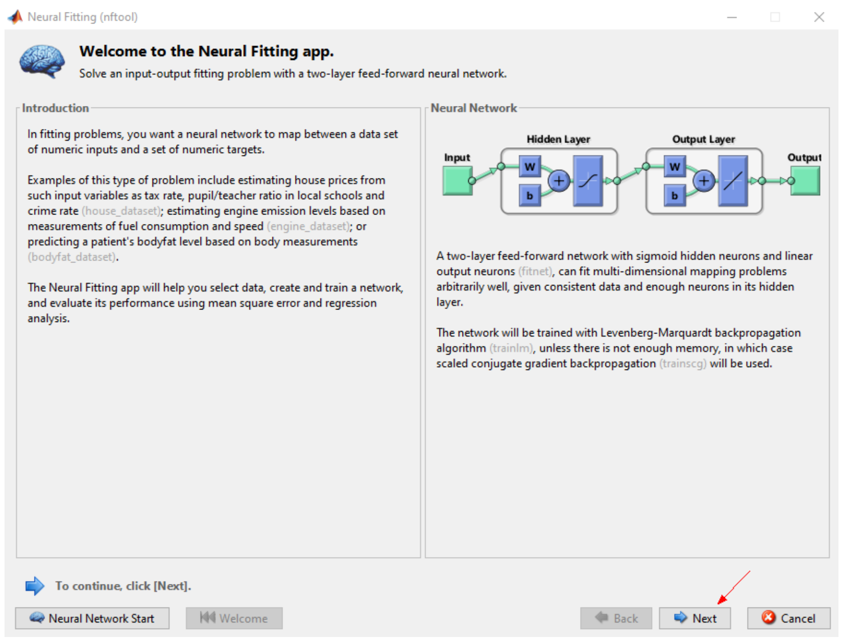
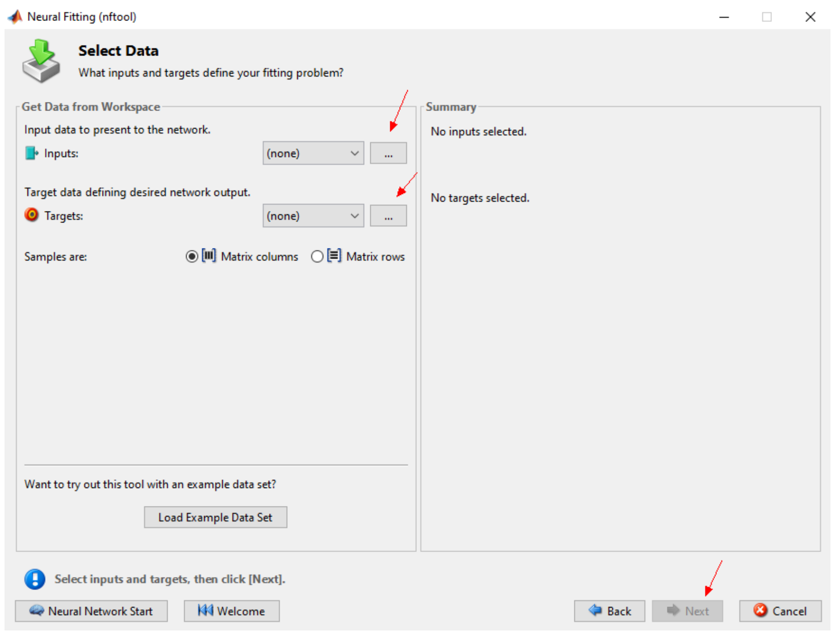
```
1. If (input1 is in1mf1) and (input2 is in2mf1) then (output is out1mf1) (1)
2. If (input1 is in1mf1) and (input2 is in2mf2) then (output is out1mf2) (1)
3. If (input1 is in1mf1) and (input2 is in2mf3) then (output is out1mf3) (1)
4. If (input1 is in1mf2) and (input2 is in2mf1) then (output is out1mf4) (1)
5. If (input1 is in1mf2) and (input2 is in2mf2) then (output is out1mf5) (1)
6. If (input1 is in1mf2) and (input2 is in2mf3) then (output is out1mf6) (1)
7. If (input1 is in1mf3) and (input2 is in2mf1) then (output is out1mf7) (1)
8. If (input1 is in1mf3) and (input2 is in2mf2) then (output is out1mf8) (1)
9. If (input1 is in1mf3) and (input2 is in2mf3) then (output is out1mf9) (1)
```

If input1 is: in1mf1, in1mf2, in1mf3, none — [ ] not
and input2 is: in2mf1, in2mf2, in2mf3, none — [ ] not
Then output is: out1mf1, out1mf2, out1mf3, out1mf4, out1mf5, out1mf6 — [ ] not
Connection: ( ) or (●) and
Weight: 1

FIS Name: Untitled2

Export FIS file The model is now trained and can be exported for usage. Click on “File” -> “Export” -> “To File…”	 Anfis Editor: Untitled2 File Edit View New FIS... Import Export Print Ctrl+P Close Ctrl+W To Workspace... Ctrl+T To File... Ctrl+S Training Error 0.08 0.06 0 20 40 60 80 100 Epochs ANFIS Info. Structure Clear Plot Load data Load Data... Clear Data Generate FIS Generate FIS ... Train FIS Train Now Test FIS Test Now Help Close # of inputs: 2 # of outputs: 1 # of input mfs: 3 3 Load data — Type: Training, Testing, [✓ Checking], Demo; From: file, [✓ worksp.] Generate FIS: Load from file, Load from worksp., [✓ Grid partition], Sub. clustering Optim. Method: hybrid Error Tolerance: 0 Epochs: 100 Test FIS — Plot against: [✓ Training data], Testing data, Checking data Epoch 100:error= 0.074291

Appendix G: How to use Neural Network Toolbox in Matlab

Description	Screenshot
Open neural network toolbox: Write “nnstart” in the command window Click on Next	
Select the input and target data (output) and click on next. (You can also load example data if you don’t have any data available)	

Neural Fitting (nftool)

Welcome to the Neural Fitting app.

Solve an input-output fitting problem with a two-layer feed-forward neural network.

Introduction

In fitting problems, you want a neural network to map between a data set of numeric inputs and a set of numeric targets.

Examples of this type of problem include estimating house prices from such input variables as tax rate, pupil/teacher ratio in local schools and crime rate (house_dataset); estimating engine emission levels based on measurements of fuel consumption and speed (engine_dataset); or predicting a patient's bodyfat level based on body measurements (bodyfat_dataset).

The Neural Fitting app will help you select data, create and train a network, and evaluate its performance using mean square error and regression analysis.

Neural Network

A two-layer feed-forward network with sigmoid hidden neurons and linear output neurons (fitnet), can fit multi-dimensional mapping problems arbitrarily well, given consistent data and enough neurons in its hidden layer.

The network will be trained with Levenberg-Marquardt backpropagation algorithm (trainlm), unless there is not enough memory, in which case scaled conjugate gradient backpropagation (trainscg) will be used.

To continue, click [Next].

Neural Fitting (nftool)

Select Data

What inputs and targets define your fitting problem?

Get Data from Workspace

Input data to present to the network.

Inputs: (none) ...

Target data defining desired network output.

Targets: (none) ...

Samples are: [✓ Matrix columns] Matrix rows

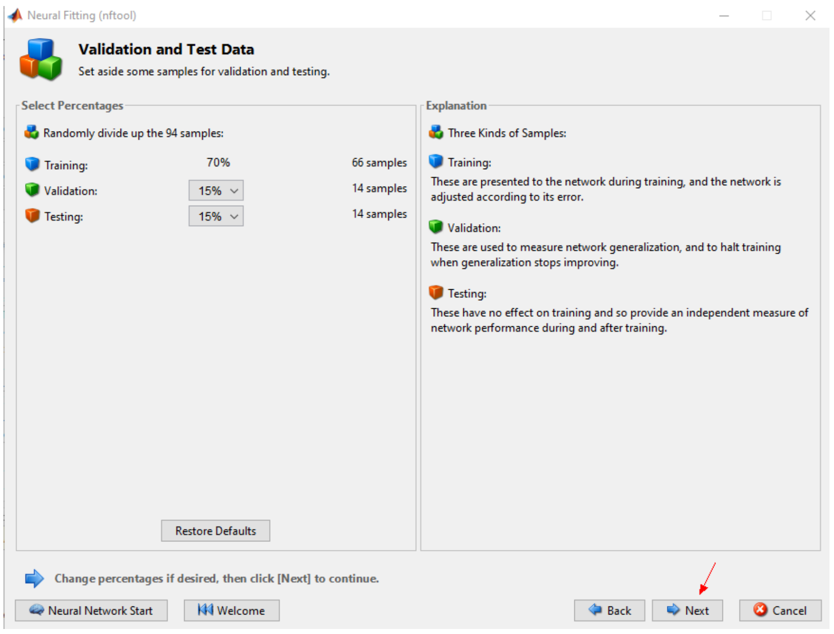
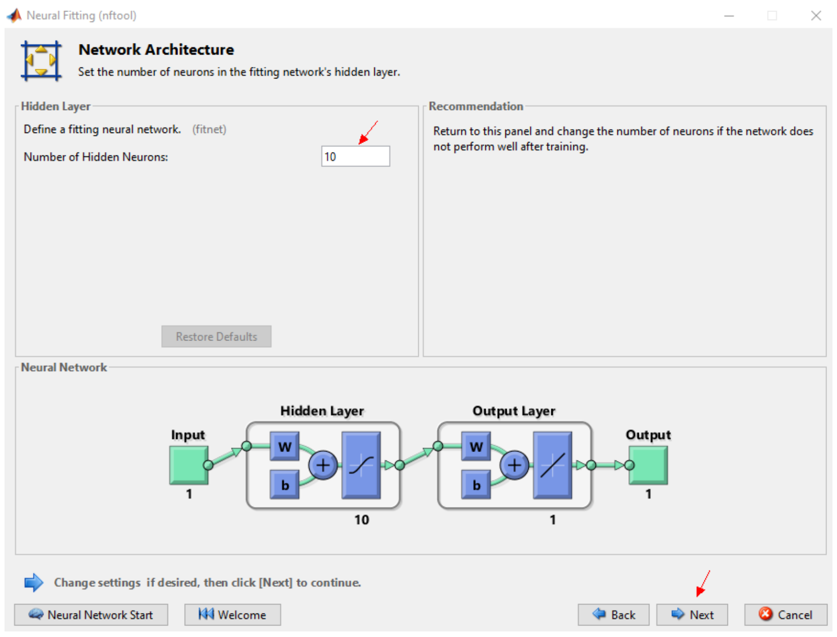
Want to try out this tool with an example data set?

Summary

No inputs selected.

No targets selected.

Select inputs and targets, then click [Next].

Click on next. (Here you can choose how big the Validation and Testing percentages should be)	
Click on next. (Here you can also specify the number of hidden Neurons)	

Neural Fitting (nftool)

Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 94 samples:

Training:	70%	66 samples
Validation:	15%	14 samples
Testing:	15%	14 samples

Explanation

Three Kinds of Samples:

Training:
These are presented to the network during training, and the network is adjusted according to its error.

Validation:
These are used to measure network generalization, and to halt training when generalization stops improving.

Testing:
These have no effect on training and so provide an independent measure of network performance during and after training.

Change percentages if desired, then click [Next] to continue.

Neural Fitting (nftool)

Network Architecture

Set the number of neurons in the fitting network's hidden layer.

Hidden Layer

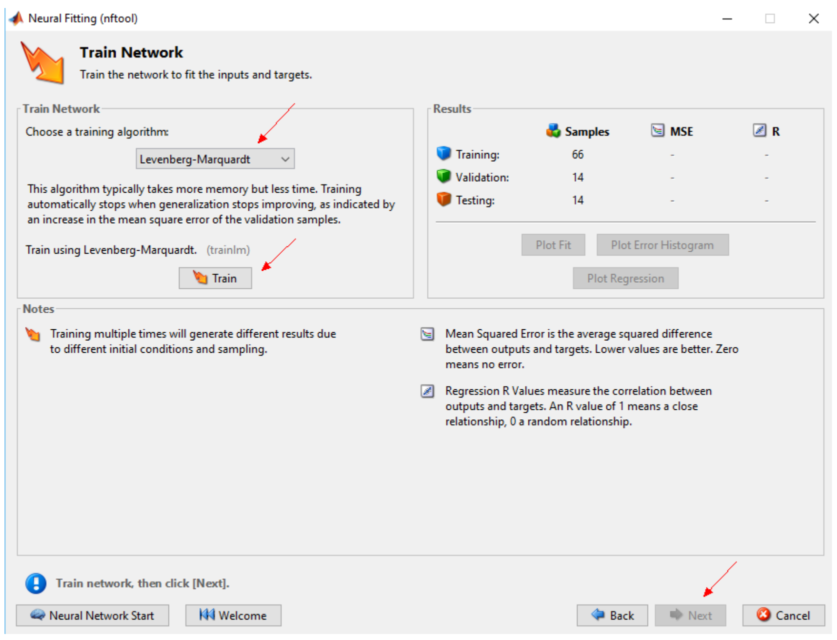
Define a fitting neural network. (fitnet)

Number of Hidden Neurons: 10

Recommendation

Return to this panel and change the number of neurons if the network does not perform well after training.

Change settings if desired, then click [Next] to continue.

Choose training algorithm “Levenberg-Marquardt” and click on train. When finished training, click on next.	

Neural Fitting (nftool)

Train Network

Train the network to fit the inputs and targets.

Train Network

Choose a training algorithm:

Levenberg-Marquardt

This algorithm typically takes more memory but less time. Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

Train using Levenberg-Marquardt. (trainlm)

Train

Results

	Samples	MSE	R
Training:	66	-	-
Validation:	14	-	-
Testing:	14	-	-

Plot Fit | Plot Error Histogram | Plot Regression

Notes

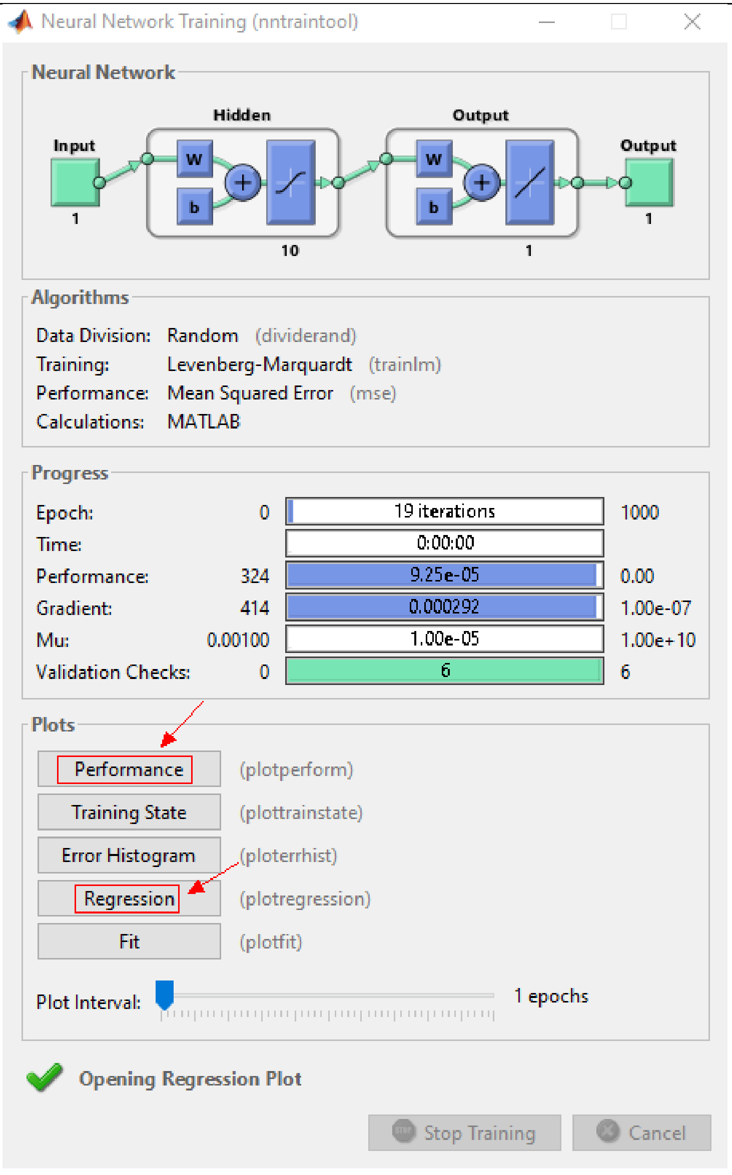
Training multiple times will generate different results due to different initial conditions and sampling.

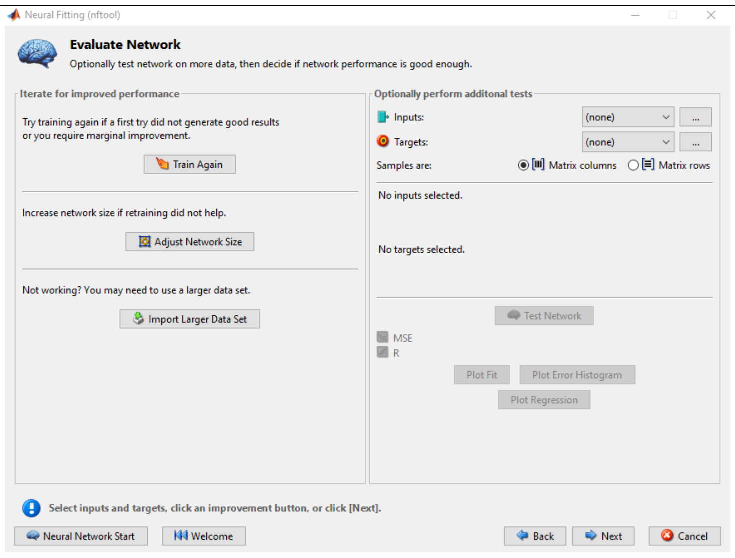
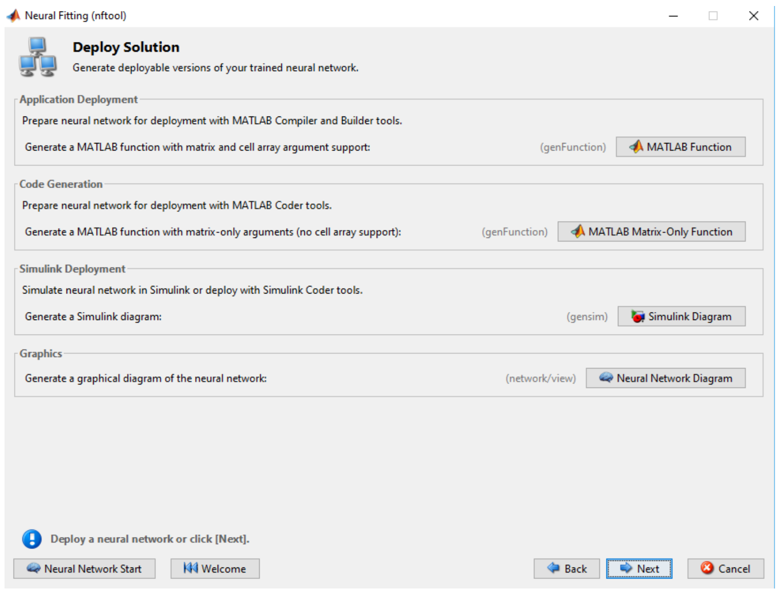
Mean Squared Error is the average squared difference between outputs and targets. Lower values are better. Zero means no error.

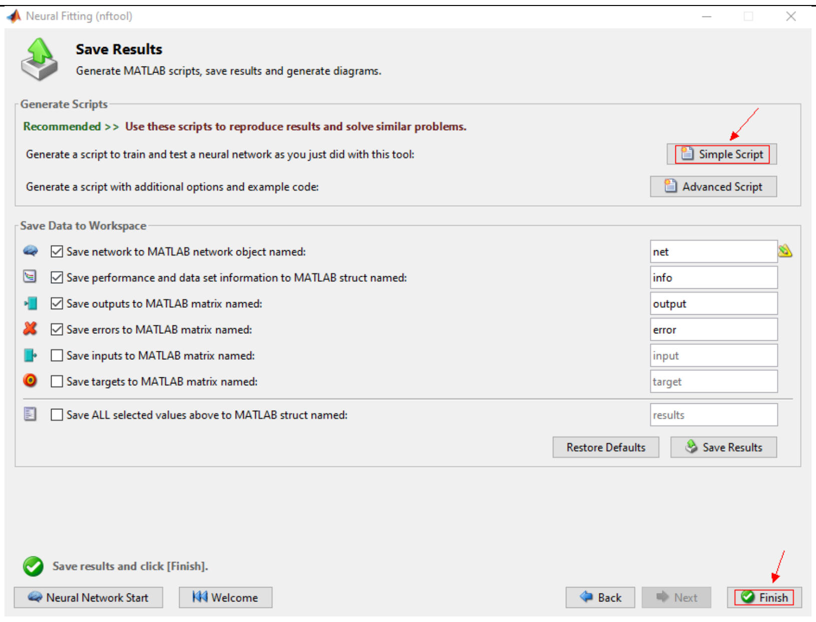
Regression R Values measure the correlation between outputs and targets. An R value of 1 means a close relationship, 0 a random relationship.

Train network, then click [Next].

Neural Network Start | Welcome | Back | Next | Cancel

Here you can see an overview of the training. You can click on "Performance" and "Regression" to see additional information about your model. Close this windows when finished.	 Neural Network Training (nntraintool) **Neural Network** Input 1 Hidden W b 10 Output W b 1 Output 1 **Algorithms** Data Division: Random (dividerand) Training: Levenberg-Marquardt (trainlm) Performance: Mean Squared Error (mse) Calculations: MATLAB **Progress** Epoch: 0 — 19 iterations — 1000 Time: 0:00:00 Performance: 324 — 9.25e-05 — 0.00 Gradient: 414 — 0.000292 — 1.00e-07 Mu: 0.00100 — 1.00e-05 — 1.00e+10 Validation Checks: 0 — 6 — 6 **Plots** Performance (plotperform) Training State (plottrainstate) Error Histogram (ploterrhist) Regression (plotregression) Fit (plotfit) Plot Interval: 1 epochs ✓ Opening Regression Plot
Click on next. (Here you can do some more modification if you model)	

	 Neural Fitting (nftool) **Evaluate Network** Optionally test network on more data, then decide if network performance is good enough. Iterate for improved performance Try training again if a first try did not generate good results or you require marginal improvement. Train Again Increase network size if retraining did not help. Adjust Network Size Not working? You may need to use a larger data set. Import Larger Data Set Optionally perform additonal tests Inputs: (none) ... Targets: (none) ... Samples are: [✓ Matrix columns] Matrix rows No inputs selected. No targets selected. Test Network MSE R Plot Fit · Plot Error Histogram Plot Regression Select inputs and targets, click an improvement button, or click [Next]. Neural Network Start · Welcome · Back · Next · Cancel
Click on next.	 Neural Fitting (nftool) **Deploy Solution** Generate deployable versions of your trained neural network. Application Deployment Prepare neural network for deployment with MATLAB Compiler and Builder tools. Generate a MATLAB function with matrix and cell array argument support: (genFunction) MATLAB Function Code Generation Prepare neural network for deployment with MATLAB Coder tools. Generate a MATLAB function with matrix-only arguments (no cell array support): (genFunction) MATLAB Matrix-Only Function Simulink Deployment Simulate neural network in Simulink or deploy with Simulink Coder tools. Generate a Simulink diagram: (gensim) Simulink Diagram Graphics Generate a graphical diagram of the neural network: (network/view) Neural Network Diagram Deploy a neural network or click [Next]. Neural Network Start · Welcome · Back · Next · Cancel
Click on “Simple Script” then on “Finish”.	



Neural Fitting (nftool)

Save Results

Generate MATLAB scripts, save results and generate diagrams.

Generate Scripts

Recommended >> Use these scripts to reproduce results and solve similar problems.

Generate a script to train and test a neural network as you just did with this tool: Simple Script

Generate a script with additional options and example code: Advanced Script

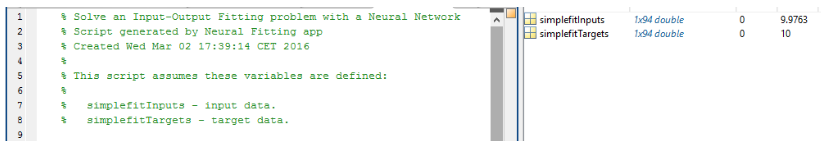
Save Data to Workspace

☑	Save network to MATLAB network object named:	net
☑	Save performance and data set information to MATLAB struct named:	info
☑	Save outputs to MATLAB matrix named:	output
☑	Save errors to MATLAB matrix named:	error
☐	Save inputs to MATLAB matrix named:	input
☐	Save targets to MATLAB matrix named:	target
☐	Save ALL selected values above to MATLAB struct named:	results

Restore Defaults Save Results

Save results and click [Finish].

Run the script that you generated. Hit "F5"



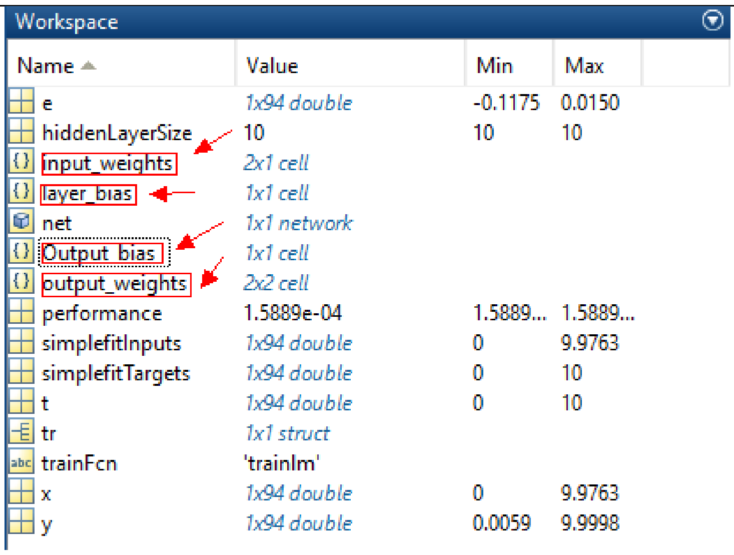
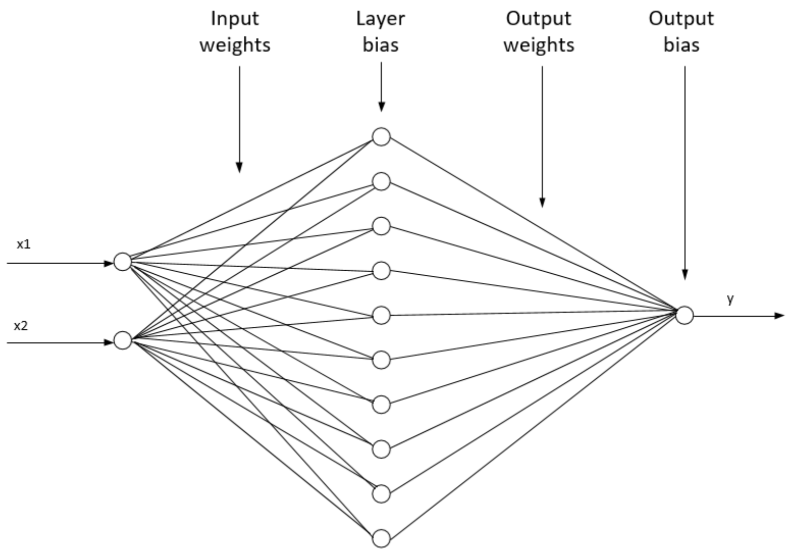
```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created Wed Mar 02 17:39:14 CET 2016
%
% This script assumes these variables are defined:
%
%   simplefitInputs - input data.
%   simplefitTargets - target data.
```

```
simplefitInputs    1x94 double    0    9.9763
simplefitTargets   1x94 double    0    10
```

Paste and run this code to get out the weights.

```
%% Input weights

input_weights=net.iw

%% Layer bias

input_bias=net.b(1)

%% Output weights

output_weights=net.lw

%% Output bias

Output_bias=net.b(2)
```

The weights data will be shown in the Workspace. (See picture)

	
NN picture	

Workspace

Name	Value	Min	Max
e	1x94 double	-0.1175	0.0150
hiddenLayerSize	10	10	10
input_weights	2x1 cell		
layer_bias	1x1 cell		
net	1x1 network		
Output_bias	1x1 cell		
output_weights	2x2 cell		
performance	1.5889e-04	1.5889...	1.5889...
simplefitInputs	1x94 double	0	9.9763
simplefitTargets	1x94 double	0	10
t	1x94 double	0	10
tr	1x1 struct		
trainFcn	'trainlm'		
x	1x94 double	0	9.9763
y	1x94 double	0.0059	9.9998

Appendix H: Simulation Code

This appendix chapter will show all the code that have been used for the simulations in the thesis. The simulations were either done with Matlab or LabVIEW.

H.1 Type -1 Fuzzy logic with Mamdani Inference Mechanism

Main

```
%% Mamdani model for estimating viscosity of fluid.
clear all; clc; close all;
data=xlsread('corrected_data');
% Density
rho1= data(:,1);%[kg/m^3]
% Viscosity
mu1= data(:,2); % [PaS]
% Shear rate
chi1= data(:,3); % [1/s]
% Shear stress
tou1= data(:,4);% [Pascal]
N=length(rho1);
%% Normalize data
[rho,rho_min,rho_max]=normalize(rho1);
[mu,mu_min,mu_max]=normalize(mu1);
[chi,chi_min,chi_max]=normalize(chi1);
[tou,tou_min,tou_max]=normalize(tou1);

%% Input and output
input=[tou,rho];%input
output=[mu,chi];

%%
[x_train,y_train,x_val,y_val,x_test,y_test]=data_division(input,output);
training_set=[x_train,y_train(:,1)];
checking_set=[x_val,y_val(:,1)];
test_set=[x_test,y_test(:,1)];
fis=readfis('MamdaniViscosity_different_range');
y_prediction1=evalfis(x_test,fis);

%% set back to respective value from normalized value
mu_test=y_test(:,1);
chi_test=y_test(:,2);
y_prediction=((y_prediction1))*(mu_max-mu_min)+mu_min;%DE-Normalizing test
set
mu_test=(mu_test)*(mu_max-mu_min)+mu_min;%DE-Normalizing test set
chi_test=(chi_test)*(chi_max-chi_min)+chi_min;%DE-Normalizing test set
e_test=(mu_test-y_prediction)./mu_test;%error with validation set
test_mse=(sum((abs(e_test)))/length(e_test))*100;
fprintf('The Mean Absolute Percentage Error in test set is = %f %%
\n',test_mse);
figure(1)
plot (mu_test,'m.');
hold on
plot(y_prediction,'b.')
hold off
legend('Actual','Prediction')
ylabel('Viscosity[mPaS]')
```

Data_division

```
function [x_train,y_train,x_val,y_val,x_test,y_test]=data_division(x1,y1)
%% 2) Division by odd and even row number
% lets take last data into training set. It is because, if we perform odd
% and even division and if we consider odd data for training then, we will
% not cover last data under training data. It means this data will be
either
% validation set or in test set. Now, when we will perform FUZZY Logic then
% we will have a error message that the 'some input is not in range'... it
% is better to take that data into training while dividing.
x_end=x1(end,:);y_end=y1(end,:);
x1=x1(1:end-1,:);y1=y1(1:end-1,:);
odd_input=x1(1:2:end, 1:1:end);
even_input=x1(2:2:end, 1:1:end);
odd_output=y1(1:2:end, 1:1:end);
even_output=y1(2:2:end, 1:1:end);
% Initialze training, validation and test sets
x_train=[];y_train=[];x_test=[];y_test=[]; y_val=[];x_val=[];
% Training set
x_train=[odd_input;x_end]; y_train=[odd_output;y_end];
% For validation and test set, divide the even data set further into even
% and odd
odd_set_input=even_input(1:2:end, 1:1:end);
even_set_input=even_input(2:2:end, 1:1:end);


odd_set_output=even_output(1:2:end, 1:1:end);
even_set_output=even_output(2:2:end, 1:1:end);
% Additional training set to make the training set 75% of whole data
x_val=odd_set_input; y_val=odd_set_output;
x_test=even_set_input; y_test=even_set_output;
end
```

Normalize

```
function[x_normalized,x_min,x_max]=normalize(x)
x_min=min(x);
x_max=max(x);
x_normalized=(x-x_min)/(x_max-x_min);
end
```

H.2 Type-2 Fuzzy logic with Mamdani Inference Mechanism

```
%% Type-2 Fuzzy logic model for estimating viscosity of fluid.
clear all; clc; close all;
data=xlsread('corrected_data_02_26_2016');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2)./1000; % [PaS]
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
N=length(rho);
%% Normalize data
[rho,rho_min,rho_max]=normalize(rho);
[mu,mu_min,mu_max]=normalize(mu);
```

```
[chi,chi_min,chi_max]=normalize(chi);
[tou,tou_min,tou_max]=normalize(tou);

%% Input and output
input=[rho,tou];%input
output=[mu,chi];

%%
[x_train,y_train,x_val,y_val,x_test,y_test]=data_division(input,output);
training_set=[x_train,y_train(:,1)];
checking_set=[x_val,y_val(:,1)];
test_set=[x_test,y_test(:,1)];

%% FIS model
fis=readt2fis('latest.t2_1fis');
for i=1:50
y_prediction(i)=evalt2(fis,x_test(i,:),1,false);
end



%% set back to respective value from normalized value
mu_test=y_test(:,1);
chi_test=y_test(:,2);
y_prediction=(y_prediction)*(mu_max-mu_min)+mu_min;%Normalizing test set
mu_test=mu_test(1:50);
chi_test=chi_test(1:50);
mu_test=(mu_test)*(mu_max-mu_min)+mu_min;%Normalizing test set
chi_test=(chi_test)*(chi_max-chi_min)+chi_min;%Normalizing test set
e_test=(mu_test-y_prediction')./mu_test;%error with validation set
test_mse=(sum((abs(e_test)))/length(e_test))*100;
fprintf('The Mean Absolute Percentage Error in test set is = %f %%
\n',test_mse);
figure(1)
clf(1)
plot (mu_test.*1000,'m.');
hold on
plot(y_prediction.*1000,'b.')
hold off
legend('Actual','Prediction')
ylabel('Viscosity[mPaS]')
```

H.3 Type-1 Fuzzy logic with Sugeno Inference Mechanism

Main

```
%% Sugeno ANFIS model for estimating viscosity of fluid.
clear all; clc; close all;
data=xlsread('corrected_data_02_26_2016');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2)./1000; % [PaS]
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
```

```
N=length(rho);
%% Normalize data
[rho,rho_min,rho_max]=normalize(rho);
[mu,mu_min,mu_max]=normalize(mu);
[chi,chi_min,chi_max]=normalize(chi);
[tou,tou_min,tou_max]=normalize(tou);

%% Input and output
input=[rho,tou];%input
output=[mu,chi];

%%
[x_train,y_train,x_val,y_val,x_test,y_test]=data_division(input,output);
training_set=[x_train,y_train(:,1)];
checking_set=[x_val,y_val(:,1)];
test_set=[x_test,y_test(:,1)];
%% ANFIS model
fis=readfis('khim');
y_prediction1=evalfis([x_test],fis);

%% set back to respective value from normalized value
mu_test=y_test(:,1);
chi_test=y_test(:,2);
y_prediction=(y_prediction1)*(mu_max-mu_min)+mu_min;%Normalizing test set
mu_test=(mu_test)*(mu_max-mu_min)+mu_min;%Normalizing test set
chi_test=(chi_test)*(chi_max-chi_min)+chi_min;%Normalizing test set
e_test=(mu_test-y_prediction)./mu_test;%error with validation set
test_mse=(sum((abs(e_test)))/length(e_test))*100;
fprintf('The Mean Absolute Percentage Error in test set is = %f %%
\n',test_mse);
figure(1)
plot (mu_test.*1000,'m.');
hold on
plot(y_prediction.*1000,'b.')
hold off
legend('Actual','Prediction')
xlabel('Samples')
ylabel('Viscosity[cP]')
title('The Sugeno type fuzzy model prediction vs. target viscosity
measurement')
figure(2)
tou_test=x_test(:,2);
tou_test=(tou_test)*(tou_max-tou_min)+tou_min;%Normalizing test set
chi_prediction=tou_test./y_prediction;
loglog(chi_test,mu_test.*1000,'m');
hold on
loglog(chi_prediction,y_prediction.*1000,':b')
hold off
legend('Actual','Prediction')
ylabel('Viscosity [mPaS]')
xlabel('Shear rate [1/s]')

%%
figure(5)
clf(5)
y_prediction=y_prediction.*1000;mu_test=mu_test.*1000;
loglog(chi_test(1:67),mu_test(1:67),'m')
hold on
loglog(chi_test(68:135),mu_test(68:135),'k')
loglog(chi_test(203:270),mu_test(203:270),'c')
loglog(chi_test(338:404),mu_test(338:404),'g')
```

```
 loglog(chi_test(405:471),mu_test(405:471),'b')

% predicted
loglog(chi_prediction(1:67),y_prediction(1:67),'om')
loglog(chi_prediction(68:135),y_prediction(68:135),'<k')
loglog(chi_prediction(203:270),y_prediction(203:270),'sc')
loglog(chi_prediction(338:404),y_prediction(338:404),'^g')
 loglog(chi_prediction(405:471),y_prediction(405:471),'*b')
hold off
legend('Sample-1','Sample-2','Sample-3','Sample-4','Sample-5','PredSample-
1',...
    'PredSample-2','PredSample-3','PredSample-4','PredSample-5')
xlabel('Shear rate  [1/s]');
ylabel('Viscosity  [cP]');
str=sprintf('Calibration of Sugeno type Fuzzy model with MAPE of %.2f %% ',
test_mse);
title(str)
```

Data_division

```
function [x_train,y_train,x_val,y_val,x_test,y_test]=data_division(x1,y1)
%% 2) Division by odd and even row number
% lets take last data into training set. It is because, if we perform odd
% and even division and if we consider odd data for training then, we will
% not cover last data under training data. It means this data will be
either
% validation set or in test set. Now, when we will perform FUZZY Logic then
% we will have a error message that the 'some input is not in range'... it
% is better to take that data into training while dividing.
x_end=x1(end,:);y_end=y1(end,:);
x1=x1(1:end-1,:);y1=y1(1:end-1,:);
odd_input=x1(1:2:end, 1:1:end);
even_input=x1(2:2:end, 1:1:end);
odd_output=y1(1:2:end, 1:1:end);
even_output=y1(2:2:end, 1:1:end);
% Initialze training, validation and test sets
x_train=[];y_train=[];x_test=[];y_test=[]; y_val=[];x_val=[];
% Training set
x_train=[odd_input;x_end]; y_train=[odd_output;y_end];
% For validation and test set, divide the even data set further into even
% and odd
odd_set_input=even_input(1:2:end, 1:1:end);
even_set_input=even_input(2:2:end, 1:1:end);


odd_set_output=even_output(1:2:end, 1:1:end);
even_set_output=even_output(2:2:end, 1:1:end);
% Additional training set to make the training set 75% of whole data
x_val=odd_set_input; y_val=odd_set_output;
x_test=even_set_input; y_test=even_set_output;
end
```

Normalize

```
function[x_normalized,x_min,x_max]=normalize(x)
x_min=min(x);
x_max=max(x);
x_normalized=(x-x_min)/(x_max-x_min);
end
```

H.4 Feedforward Artificial Neural Network

Main

```
%% Feedforward model for estimating viscosity of fluid.
clear all; clc; close all;
data=xlsread('corrected_data_02_26_2016');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2); % [cP]
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
N=length(rho);

%% Input and output
input=[rho,tou];%input
output=[mu,chi];
output1=output(:,1);
%%
[x_train,y_train,x_val,y_val,x_test,y_test]=data_division(input,output);
training_set=[x_train,y_train(:,1)];
checking_set=[x_val,y_val(:,1)];
test_set=[x_test,y_test(:,1)];
%% ANN model
y_prediction = StaticNN_Viscosity(x_test');
%% set back to respective value from normalized value
mu_test=y_test(:,1);
chi_test=y_test(:,2);
e_test=(mu_test-y_prediction')./mu_test;%error with validation set
test_mse=(sum((abs(e_test)))/length(e_test))*100;
fprintf('The Mean Absolute Percentage Error in test set is = %f %%
\n',test_mse);
figure(1)
plot (mu_test,'m.');
hold on
plot(y_prediction','b.')
hold off
legend('Actual','Prediction')
ylabel('Viscosity[mPaS]')

figure(2)
tou_test=x_test(:,2);
chi_prediction=tou_test./(y_prediction./1000)';
loglog(chi_test,mu_test,'m');
hold on
loglog(chi_prediction,y_prediction,':b')
hold off
legend('Actual','Prediction')
ylabel('Viscosity [mPaS]')
xlabel('Shear rate [1/s]')

%%
figure(5)
clf(5)
```

```
y_prediction=y_prediction;mu_test=mu_test;
loglog(chi_test(1:67),mu_test(1:67),'m')
hold on
loglog(chi_test(68:135),mu_test(68:135),'k')
loglog(chi_test(203:270),mu_test(203:270),'c')
loglog(chi_test(338:404),mu_test(338:404),'g')
loglog(chi_test(405:471),mu_test(405:471),'b')

% predicted
loglog(chi_prediction(1:67),y_prediction(1:67),'om')
loglog(chi_prediction(68:135),y_prediction(68:135),'<k')
loglog(chi_prediction(203:270),y_prediction(203:270),'sc')
loglog(chi_prediction(338:404),y_prediction(338:404),'^g')
loglog(chi_prediction(405:471),y_prediction(405:471),'*b')
hold off
legend('Sample-1','Sample-2','Sample-3','Sample-4','Sample-5','PredSample-
1',...
    'PredSample-2','PredSample-3','PredSample-4','PredSample-5')
xlabel('Shear rate  [1/s]');
ylabel('Viscosity  [mPaS]');
title('Calibration of ANFIS model with MAPE of 2.18%')% TITLE
str=sprintf('Calibration of Feedforward model with MAPE = %f %%',
test_mse);
title(str,'fontsize',12)
```

Data_division

```
function [x_train,y_train,x_val,y_val,x_test,y_test]=data_division(x1,y1)
%% 2) Division by odd and even row number
% lets take last data into training set. It is because, if we perform odd
% and even division and if we consider odd data for training then, we will
% not cover last data under training data. It means this data will be
either
% validation set or in test set. Now, when we will perform FUZZY Logic then
% we will have a error message that the 'some input is not in range'... it
% is better to take that data into training while dividing.
x_end=x1(end,:);y_end=y1(end,:);
x1=x1(1:end-1,:);y1=y1(1:end-1,:);
odd_input=x1(1:2:end, 1:1:end);
even_input=x1(2:2:end, 1:1:end);
odd_output=y1(1:2:end, 1:1:end);
even_output=y1(2:2:end, 1:1:end);
% Initialze training, validation and test sets
x_train=[];y_train=[];x_test=[];y_test=[]; y_val=[];x_val=[];
% Training set
x_train=[odd_input;x_end]; y_train=[odd_output;y_end];
% For validation and test set, divide the even data set further into even
% and odd
odd_set_input=even_input(1:2:end, 1:1:end);
even_set_input=even_input(2:2:end, 1:1:end);
odd_set_output=even_output(1:2:end, 1:1:end);
even_set_output=even_output(2:2:end, 1:1:end);
% Additional training set to make the training set 75% of whole data
x_val=odd_set_input; y_val=odd_set_output;
x_test=even_set_input; y_test=even_set_output;
end
```

Normalize

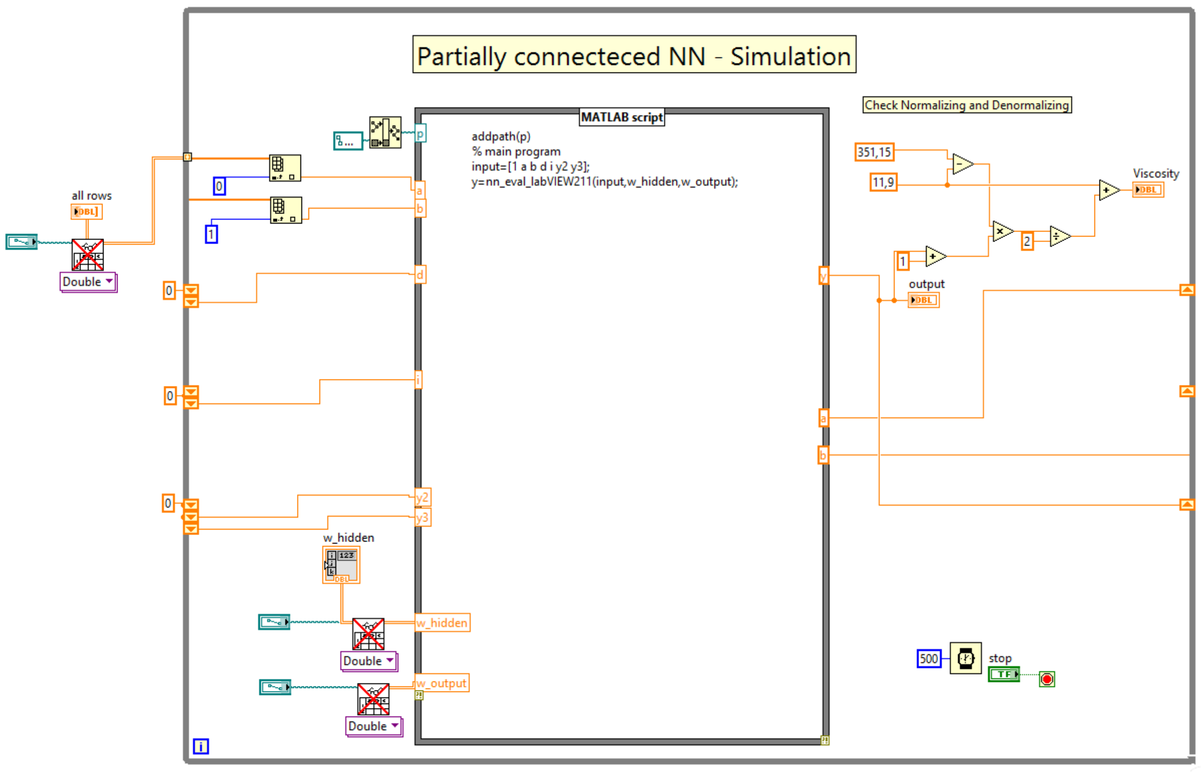
```
function[x_normalized,x_min,x_max]=normalize(x)
x_min=min(x);
x_max=max(x);
x_normalized=(x-x_min)/(x_max-x_min);
end
```

StaticNN_Viscosity

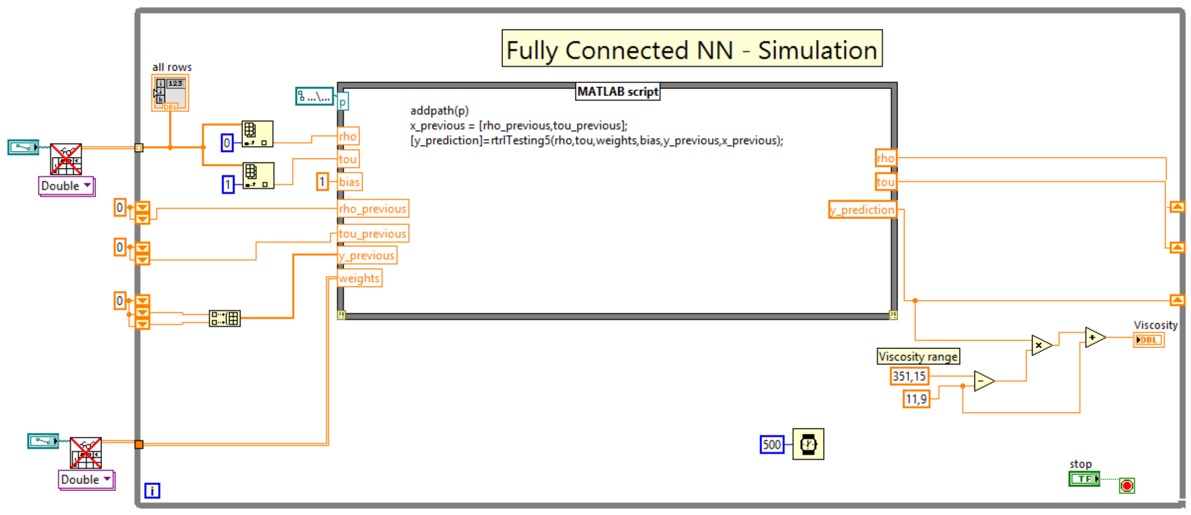
```
function [y1] = StaticNN_Viscosity(x1)
%MYNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 15-May-2016
23:40:40.
%
% [y1] = myNeuralNetworkFunction(x1) takes these arguments:
%   x = 2xQ matrix, input #1
% and returns:
%   y = 1xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

  % ===== NEURAL NETWORK CONSTANTS =====

  % Input 1
  x1_step1_xoffset = [1140;0.0798];
  x1_step1_gain = [0.00429590170976888;0.0994025904315066];
  x1_step1_ymin = -1;

  % Layer 1
  b1 = [-
9.3172800880287507;0.51230188417128175;10.418138949584778;15.787278733247952;8.2412698288411192;-14.774319186376818;-
9.7844253717626586;10.104098938503467;-
1.5301441784267593;10.409438118848641];
  IW1_1 = [12.692744793354024 2.8383760835224972;9.382923808932647 -
1.2970031450577029;-14.759008126685442 -3.757890968990957;-
0.87137483944994176 15.12014106899902;20.709544166224042 -
10.736128717644821;0.79188703610239031 -13.485107158858796;-
24.406144649494923 11.931854952779378;-13.850890811419378 -
2.9466183315784034;-7.7710365727086792 -4.5164486470010283;-
14.999334033742272 -4.2539485593116293];

  % Layer 2
  b2 = 15.111376483085019;
  LW2_1 = [-3.6690861102969077 0.33191560418552407 9.7789241577555153
6.9253600414492524 8.1302774598643328 22.914742461211702 8.1452127499401072
-7.9574518914382821 0.30569935609040971 -5.5308070925461799];

  % Output 1
  y1_step1_ymin = -1;
  y1_step1_gain = 0.0058953747837134 4;
  y1_step1_xoffset = 11.901;

  % ===== SIMULATION ========

  % Dimensions
  Q = size(x1,2); % samples
  % Input 1
  xp1 = mapminmax_apply(x1,x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);

  % Layer 1
  a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

  % Layer 2
  a2 = repmat(b2,1,Q) + LW2_1*a1;

  % Output 1
  y1 = mapminmax_reverse(a2,y1_step1_gain,y1_step1_xoffset,y1_step1_ymin);
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y =
mapminmax_apply(x,settings_gain,settings_xoffset,settings_ymin)
  y = bsxfun(@minus,x,settings_xoffset);
  y = bsxfun(@times,y,settings_gain);
  y = bsxfun(@plus,y,settings_ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n)
  a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x =
mapminmax_reverse(y,settings_gain,settings_xoffset,settings_ymin)
  x = bsxfun(@minus,y,settings_ymin);
  x = bsxfun(@rdivide,x,settings_gain);
  x = bsxfun(@plus,x,settings_xoffset);
end
```

H.5 Feedback Artificial Neural Network

H.5.1 Partially Connected Recurrent Neural Network


```
addpath(p)
% main program
input=[1 a b d i y2 y3];
y=nn_eval_labVIEW211(input,w_hidden,w_output);
```

H.5.1 Fully Connected Recurrent Neural Network


```
addpath(p)
x_previous = [rho_previous,tou_previous];
[y_prediction]=rtrlTesting5(rho,tou,weights,bias,y_previous,x_previous);
```

H.6 Support Vector Machine

```
%% Support Vector Regression model for estimating viscosity of fluid.
clear all; clc; close all;
%% Parameters of SVR model.
```

```
epsilon=0.01;% Tips: Keep epsilon small if you are confident about the
accuracy of your trianing data
sigma_range=2;% Tips: This is a parameter with Radial Basis function. It
varies with application.
C_range=500;% Tips: Keep C big if you are confident about the accuracy of
your dat
data=xlsread('corrected_data_02_26_2016');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2)./1000; % [PaS]
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
N=length(rho);
%% Normalize data
[rho,rho_min,rho_max]=normalize(rho);
[mu,mu_min,mu_max]=normalize(mu);
[chi,chi_min,chi_max]=normalize(chi);
[tou,tou_min,tou_max]=normalize(tou);

%% Input and output
x=[rho,tou];%input
y=[mu,chi]; %output is only shear rate... viscosity is considered in order
to track the index of samples

%% 2) Division by odd and even row number
odd_input=x(1:2:end, 1:1:end);
even_input=x(2:2:end, 1:1:end);
odd_output=y(1:2:end, 1:1:end);
even_output=y(2:2:end, 1:1:end);
% Initialze training, validation and test sets
x_train=[];y_train=[];x_val=[];y_val=[];x_test=[];y_test=[];
% Training set
x_train=odd_input; y_train=odd_output;
% For validation and test set, divide the even data set further into even
% and odd
odd_set_input=even_input(1:2:end, 1:1:end);
even_set_input=even_input(2:2:end, 1:1:end);

odd_set_output=even_output(1:2:end, 1:1:end);
even_set_output=even_output(2:2:end, 1:1:end);
% Validation set
x_val=odd_set_input; y_val=odd_set_output;
% Lets include this validation set into training set
x_train=[x_train;x_val];y_train=[y_train;y_val];y_train=y_train(:,1);
% Test set
x_test=even_set_input;y_test=even_set_output;
N1=length(x_train);
%% Find the SVR model parameters
C_Yun=3*std(y_train)
%load('kkk_viscosity');
k=3;
%noise_std=sqrt((k)/((k-1)*N1)*sum(kkk_viscosity.^2));% =0.435
noise_std=0.01
tou=3;
epsilon_Yun=tou*noise_std*sqrt(log(N1)/N1)
sigma_Yun=0.026;% 0.03;0.026 for dataset2

%% Optimal solutions
```

```
[svr,n_sv]=my_svr(x_train,y_train,epsilon_Yun,sigma_Yun,C_Yun);%Radial

y_prediction=svr.predict(x_test);% svr prediction for input validation set

%% set back to respective value from normalized value
mu_test=y_test(:,1);chi_test=y_test(:,2);
e_test=mu_test-y_prediction;%error with validation set
mse=sum(((e_test).^2))/length(e_test);
mean_error=mean(abs(e_test));%mse of validation
relative_error=(mean_error/(max(mu_test)-min(mu_test)))*100;
fprintf('The relative error in prediction is = %f %% \n',relative_error);
fprintf('The mse in prediction is = %f\n',mse);


y_prediction=(y_prediction)*(mu_max-mu_min)+mu_min;%Normalizing test set
mu_test=(mu_test)*(mu_max-mu_min)+mu_min;%Normalizing test set
chi_test=(chi_test)*(chi_max-chi_min)+chi_min;%Normalizing test set

e_test=(mu_test-y_prediction)./mu_test;%error with validation set
test_mse=(sum((abs(e_test)))/length(e_test))*100;
fprintf('The Mean Absolute Percentage Error in test set is = %f %%
\n',test_mse);

figure(1)
plot (mu_test.*1000,'m.');
hold on
plot(y_prediction.*1000,'b.')
hold off
legend('Target','Prediction')
ylabel('Viscosity[cP]')
xlabel('Samples')
title('The SVR viscosity prediction vs. target viscosity measurement')
figure(2)
tou_test=x_test(:,2);
tou_test=(tou_test)*(tou_max-tou_min)+tou_min;%Normalizing test set
chi_prediction=tou_test./y_prediction;
loglog(chi_test,mu_test.*1000,'m');
hold on
loglog(chi_prediction,y_prediction.*1000,':b')
hold off
legend('Target','Prediction')
ylabel('Viscosity [cP]')
xlabel('Shear rate [1/s]')

%%
figure(5)
clf(5)
y_prediction=y_prediction.*1000;
loglog(chi_test(1:67),y_prediction(1:67),'m')
hold on
loglog(chi_test(68:135),y_prediction(68:135),'k')
loglog(chi_test(203:270),y_prediction(203:270),'c')
loglog(chi_prediction(338:404),y_prediction(338:404),'g')
loglog(chi_prediction(405:471),y_prediction(405:471),'b')

% predicted
loglog(chi_prediction(1:67),y_prediction(1:67),'om')
loglog(chi_prediction(68:135),y_prediction(68:135),'<k')
loglog(chi_prediction(203:270),y_prediction(203:270),'sc')
loglog(chi_prediction(338:404),y_prediction(338:404),'^g')
loglog(chi_prediction(405:471),y_prediction(405:471),'*b')
hold off
legend('Sample-1','Sample-2','Sample-3','Sample-4','Sample-5','PredSample-
1',...
    'PredSample-2','PredSample-3','PredSample-4','PredSample-5')
xlabel('Shear rate  [1/s]');
ylabel('Viscosity  [cP]');
str=sprintf('Calibration of SVR model with MAPE = %f %%', test_mse);
title(str)
```

H.7 Classification

ANN as a classifier

```
%% ANN model for estimating viscosity of fluid.
clear all; clc; close all;
data=xlsread('corrected_data');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2); % [PaS]
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
N=length(rho);
X1=[(chi) (mu)];
load index
Y1=index';
%% Examine a scatter plot of the data.
figure(1)
gscatter(X1(:,1),X1(:,2),Y1);
title('{\bf Scatter Diagram of Viscosity Measurements}');
set(gca,'xscale','log')
set(gca,'yscale','log')
xlabel('{\bf Shear Rate [1/s]}');
ylabel('{\bf Viscosity [cP]}');
legend('Location','Northeast');
%% Taking log of input data
input=log(X1);
output=[Y1];
%%
[x_train1,y_train1,x_val,y_val,x_test,y_test]=data_division(input,output);
x_train=[x_train1;x_val];
y_train=[y_train1;y_val];
training_set=[x_train,y_train];
test_set=[x_test,y_test];

%% ANN model
d = 0.01;
[x1Grid,x2Grid] = meshgrid(min(x_train(:,1)):d:max(x_train(:,1)),...
    min(x_train(:,2)):d:max(x_train(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
y_prediction=ANN_Classifier(xGrid');
y_prediction_round=round(y_prediction);
%%
figure(2)
% anti-log
xGrid=exp(xGrid);
```

```
h(1:3) = gscatter(xGrid(:,1),xGrid(:,2),y_prediction_round,...
    [0.1 0.5 0.5; 0.5 0.1 0.5; 0.5 0.5 0.1]);
hold on
% gscatter(X1(:,1),X1(:,2),Y1);
title('{\bf Viscosity Classification Regions}');
xlabel('{\bf Shear Rate [1/s]}');
ylabel('{\bf Viscosity [cP]}');
set(gca,'xscale','log')
set(gca,'yscale','log')
axis tight

%% Testing a model
y_prediction1=ANN_Classifier(x_test');
y_prediction_test=round(y_prediction1);
% anti-log
x_test=exp(x_test);
%
h(4:6) = gscatter(x_test(:,1),x_test(:,2),y_prediction_test,...
    'rgb','ooo',6,[0.1 0.5 0.5; 0.5 0.1 0.5; 0.5 0.5 0.1]);
 legend(h(4:6),{%'LessViscos region','Viscos region','HighViscos
region',...
%    'Observed LessViscos','Observed Viscos', 'Observed HighViscos',...
    'Tested LessViscos','Tested Viscos','Tested HighViscos'},...
    'Location','NorthEast');

txt1 = '{\bf [LessViscos Region]}';
text(5,18,txt1)
txt2 = '{\bf [Viscos Region]}';
text(100,90,txt2)
txt3 = '{\bf [HighViscos Region]}';
text(5,250,txt3)
hold off
%% Classification Error
% Compare the prediction and test data
count=0;
for i_comp=1:length(y_prediction_test)
    if (y_prediction_test(i_comp)~=y_test(i_comp))
        count=count+1;
    else
    end
end
per=count*100/length(y_prediction_test);
fprintf('The mis-classification percentage is %f %% \n',per);
```

Sugeno type-1 Fuzzy Logic as a classifier

```
%% ANFIS model for estimating viscosity of fluid.
clear all; clc; close all;
data=xlsread('corrected_data');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2); % [PaS]
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
N=length(rho);
X1=[(chi) (mu)];
load index
```

```
Y1=index';
X=X1;Y=Y1;
%% Input and output
input=[X];%input
output=[Y];

% Examine a scatter plot of the data.
figure(1)
gscatter(X(:,1),X(:,2),Y);
title('{\bf Scatter Diagram of Viscosity Measurements}');
set(gca,'xscale','log')
set(gca,'yscale','log')
xlabel('{\bf Shear Rate [1/s]}');
ylabel('{\bf Viscosity [cP]}');
legend('Location','Northeast');
%% Taking log of input data
X=log(X);
input=X;
%%
[x_train,y_train,x_val,y_val,x_test,y_test]=data_division(input,output);
%%
training_set=[x_train,y_train];
checking_set=[x_val,y_val];
test_set=[x_test,y_test];
 training_set1=[input,output];
d = 0.01;
[x1Grid,x2Grid] = meshgrid(min(X(:,1)):d:max(X(:,1)),...
    min(X(:,2)):d:max(X(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
fis=readfis('Sugeno_Classifier');
y_prediction=evalfis([xGrid],fis);
% Take only those samples that belongs to classes out of three
for i_index=1:length(y_prediction)
    if y_prediction(i_index)>0.5 &&y_prediction(i_index)<3.5
        indexxx(i_index)=i_index;
        y_output(i_index)=y_prediction(i_index);
    end
end

%%
k = find(indexxx)
xGrid=xGrid(k,:);
yy=y_output(k)
y_prediction_round=round(yy);

%%
figure(2)
%% anti-log
xGrid=exp(xGrid);
X=exp(X);
 gscatter(xGrid(:,1),xGrid(:,2),y_prediction_round,...
    [0.1 0.5 0.5; 0.5 0.1 0.5; 0.5 0.5 0.1]);
hold on
gscatter(X(:,1),X(:,2),Y);
title('{\bf Viscosity Classification Regions}');
xlabel('{\bf Shear Rate [1/s]}');
ylabel('{\bf Viscosity [cP]}');
set(gca,'xscale','log')
set(gca,'yscale','log')
legend('HighViscos region','LessViscos region','Viscos region',...
   'Observed LessViscos','Observed Viscos', 'Observed HighViscos',...
```

```
    'Location','NorthEast');
axis tight
hold off
```

SVM as a classifier

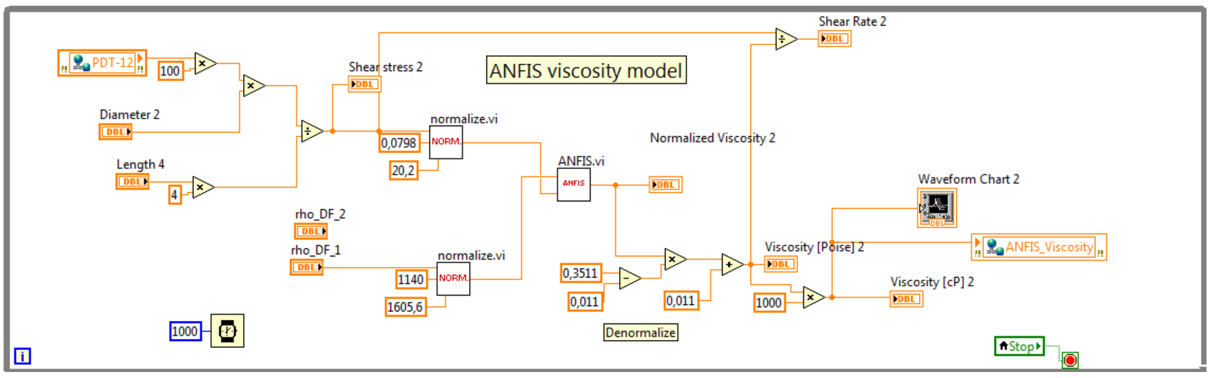
```
clear all;
clc;
data=xlsread('corrected_data');
% Density
rho= data(:,1);%[kg/m^3]
% Viscosity
mu= data(:,2); % [m PaS] or cP
% Shear rate
chi= data(:,3); % [1/s]
% Shear stress
tou= data(:,4);% [Pascal]
N=length(rho);
input=[(chi) (mu)];
load className
output=className';
%% Examine a scatter plot of the data.
figure(1)
gscatter(input(:,1),input(:,2),output);
title('{\bf Scatter Diagram of Viscosity Measurements}');
set(gca,'xscale','log')
set(gca,'yscale','log')
xlabel('{\bf Shear Rate [1/s]}');
ylabel('{\bf Viscosity [cP]}');
legend('Location','Northeast');
%%
[x_train1,y_train1,x_val,y_val,x_test,y_test]=data_division(input,output);
 x_train=[x_train1;x_val];
 y_train=[y_train1;y_val];
%% Define a class
SVMModels = cell(3,1);
classes = unique(y_train);
rng(1); % For reproducibility
%% Taking log of input data
X=log(x_train);
%%
for j = 1:numel(classes);
    indx = strcmp(y_train,classes(j)); % Create binary classes for each
classifier
    SVMModels{j} = fitcsvm(X,indx,'ClassNames',[false
true],'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);
end
d = 0.01;
[x1Grid,x2Grid] = meshgrid(min(X(:,1)):d:max(X(:,1)),...
    min(X(:,2)):d:max(X(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
N = size(xGrid,1);
Scores = zeros(N,numel(classes));

for j = 1:numel(classes);
    [~,score] = predict(SVMModels{j},xGrid);
    Scores(:,j) = score(:,2); % Second column contains positive-class
scores
```

```
end
[~,maxScore] = max(Scores,[],2);
%% Classification plot
figure(2)
%% anti-log
xGrid=exp(xGrid);
X=exp(X);
%%
h(1:3) = gscatter(xGrid(:,1),xGrid(:,2),maxScore,...
    [0.1 0.5 0.5; 0.5 0.1 0.5; 0.5 0.5 0.1]);
hold on
title('{\bf Viscosity Classification Regions}');
xlabel('{\bf Shear Rate [1/s]}');
ylabel('{\bf Viscosity [cP]}');
set(gca,'xscale','log')
set(gca,'yscale','log')
axis tight

%% Testing a model
x_test=log(x_test);
for k = 1:numel(classes);
    [~,score1] = predict(SVMModels{k},x_test);
    Scores1(:,k) = score1(:,2); % Second column contains positive-class
scores
end
[~,maxScore1] = max(Scores1,[],2);

%% anti-log
x_test=exp(x_test);
%%
h(4:6) = gscatter(x_test(:,1),x_test(:,2),maxScore1,...
    'brg','ooo',6,[0.1 0.5 0.5; 0.5 0.1 0.5; 0.5 0.5 0.1]);
legend(h(4:6),{%'HighViscos region','LessViscos region','Viscos region',...
   %'Observed LessViscos','Observed Viscos', 'Observed HighViscos',...
    'Tested LessViscos','Tested HighViscos','Tested Viscos'},...
    'Location','NorthEast');
txt1 = '{\bf [LessViscos Region]}';
text(9,18,txt1)
txt2 = '{\bf [Viscos Region]}';
text(100,90,txt2)
txt3 = '{\bf [HighViscos Region]}';
text(3,250,txt3)
hold off
%% Classification Error
y_prediction=cell(length(maxScore1),1);
for i_test=1:length(maxScore1)
    if maxScore1(i_test)==1
        y_prediction(i_test)={'HighViscos'};
    elseif  maxScore1(i_test)==2
        y_prediction(i_test)={'LessViscos'};
    elseif  maxScore1(i_test)==3
        y_prediction(i_test)={'Viscos'};
    end
end
% Compare the prediction and test data
count=0;
for i_comp=1:length(y_prediction)
    c=strcmp(y_prediction(i_comp),y_test(i_comp));
    if c==0
        count=count+1;
    else
    end
end
per=count*100/length(y_prediction);
fprintf('The mis-classification percentage is %f %% \n',per);
```

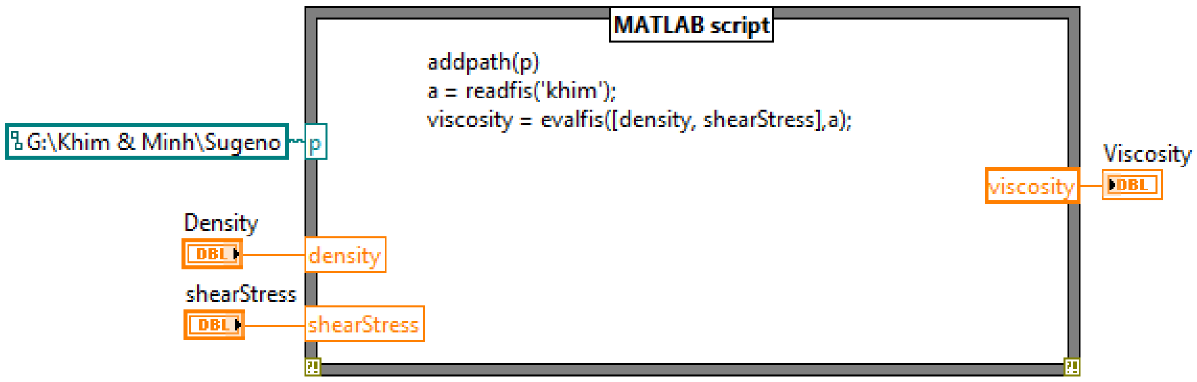
Appendix I: Experimental Code

This appendix chapter will show all the code that have been used for the experiments in the thesis. The experiments were either done with Matlab or LabVIEW.

I.1 Type-1 Fuzzy logic with Sugeno inference system

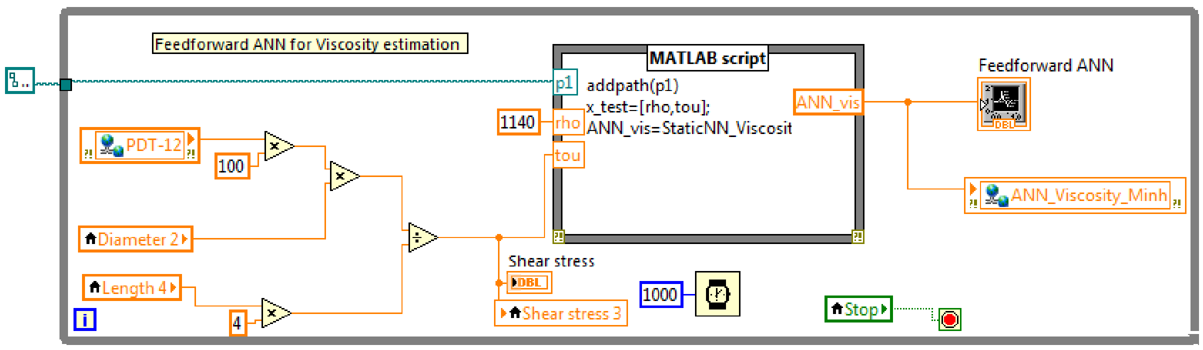


ANFIS VI



```
addpath(p)
a = readfis('khim');
viscosity = evalfis([density, shearStress],a);
```

I.2 Feedforward Artificial Neural Network



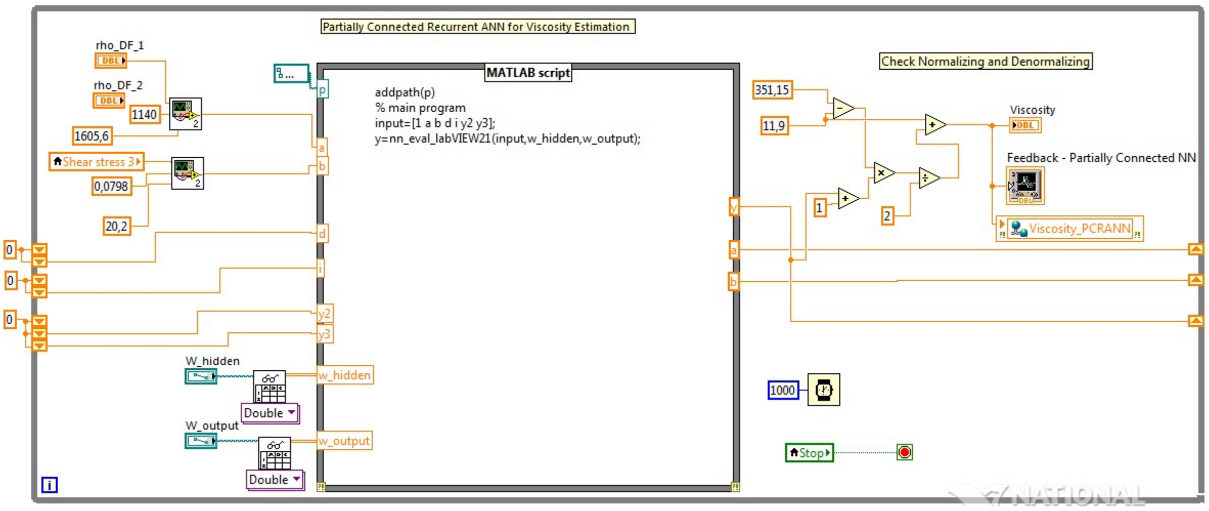
```
addpath(p1)
x_test=[rho,tou];
ANN_vis=StaticNN_Viscosit
```

StaticNN_viscosity.m

```
function [y1] = StaticNN_Viscosity(x1)
%MYNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 15-May-2016
23:40:40.
%
% [y1] = myNeuralNetworkFunction(x1) takes these arguments:
%   x = 2xQ matrix, input #1
% and returns:
%   y = 1xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

  % ===== NEURAL NETWORK CONSTANTS =====

  % Input 1
  x1_step1_xoffset = [1140;0.0798];
  x1_step1_gain = [0.00429590170976888;0.0994025904315066];
  x1_step1_ymin = -1;

  % Layer 1
  b1 = [-
9.3172800880287507;0.51230188417128175;10.418138949584778;15.78727873324795
2;8.2412698288411192;-14.774319186376818;-
9.7844253717626586;10.104098938503467;-
1.5301441784267593;10.409438118848641];
  IW1_1 = [12.692744793354024 2.8383760835224972;9.382923808932647 -
1.2970031450577029;-14.759008126685442 -3.757890968990957;-
0.87137483944994176 15.120141068999902;20.709544166224042 -
10.736128717644821;0.79188703610239031 -13.485107158858796;-
24.406144649494923 11.931854952779378;-13.850890811419378 -
2.9466183315784034;-7.7710365727086792 -4.5164486470010283;-
14.999334033742272 -4.2539485593116293];

  % Layer 2
  b2 = 15.111376483085019;
  LW2_1 = [-3.6690861102969077 0.33191560418552407 9.7789241577555153
6.9253600414492524 8.1302774598643328 22.914742461211702 8.1452127499401072
-7.9574518914382821 0.30569935609040971 -5.5308070925461799];

  % Output 1
  y1_step1_ymin = -1;
  y1_step1_gain = 0.00589537478371344;
  y1_step1_xoffset = 11.901;

  % ===== SIMULATION ========

  % Dimensions
  Q = size(x1,2); % samples

  % Input 1
  xp1 = mapminmax_apply(x1,x1_step1_gain,x1_step1_xoffset,x1_step1_ymin);

  % Layer 1
  a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

  % Layer 2
  a2 = repmat(b2,1,Q) + LW2_1*a1;
  % Output 1
  y1 = mapminmax_reverse(a2,y1_step1_gain,y1_step1_xoffset,y1_step1_ymin);
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y =
mapminmax_apply(x,settings_gain,settings_xoffset,settings_ymin)
  y = bsxfun(@minus,x,settings_xoffset);
  y = bsxfun(@times,y,settings_gain);
  y = bsxfun(@plus,y,settings_ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n)
  a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x =
mapminmax_reverse(y,settings_gain,settings_xoffset,settings_ymin)
  x = bsxfun(@minus,y,settings_ymin);
  x = bsxfun(@rdivide,x,settings_gain);
  x = bsxfun(@plus,x,settings_xoffset);
end
```

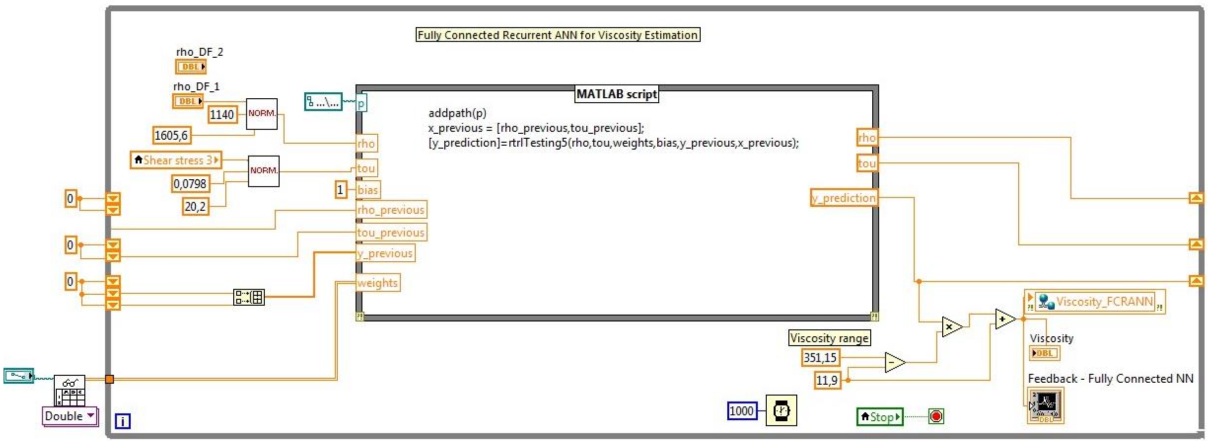
I.3 Feedback Artificial Neural Network

I.3.1 Partially Connected Recurrent Neural Network



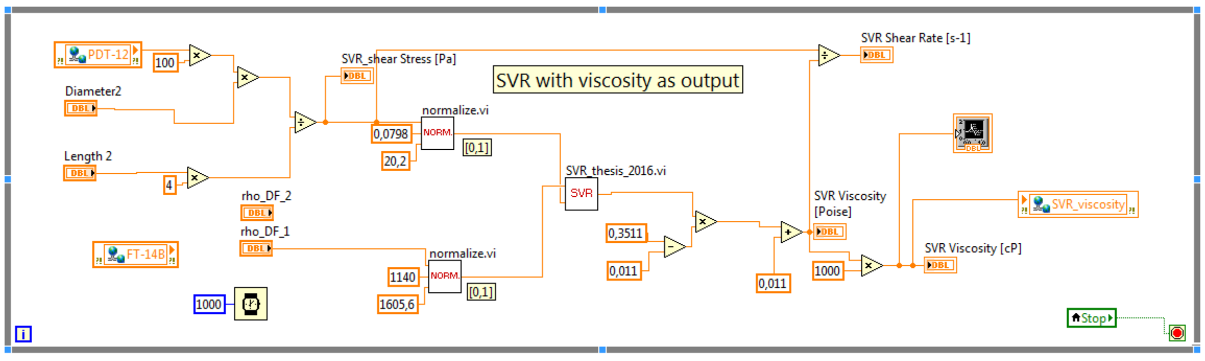
```
addpath(p)
% main program
input=[1 a b d i y2 y3];
y=nn_eval_labVIEW21(input,w_hidden,w_output);
```

I.3.2 Fully Connected Recurrent Neural Network

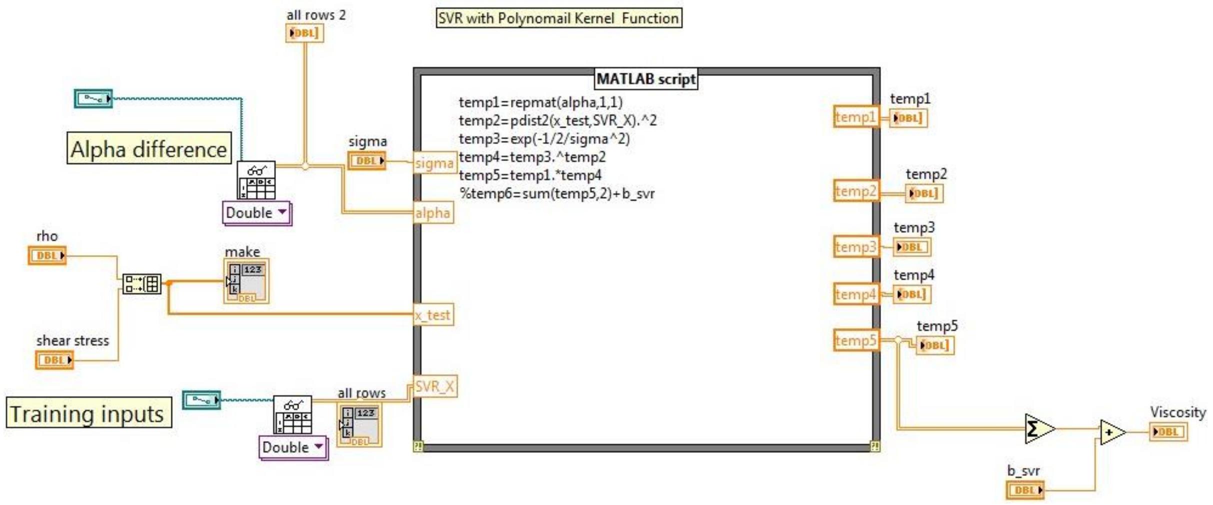


```
addpath(p)
x_previous = [rho_previous,tou_previous];
[y_prediction]=rtrlTesting5(rho,tou,weights,bias,y_previous,x_previous);
```

I.4 Support Vector Machine



SVR VI



```
temp1=repmat(alpha,1,1)
temp2=pdist2(x_test,SVR_X).^2
temp3=exp(-1/2/sigma^2)
temp4=temp3.^temp2
temp5=temp1.*temp4
%temp6=sum(temp5,2)+b_svr
```