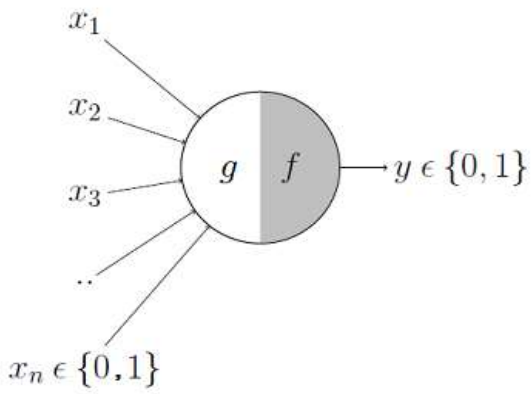
Experiment No.01

Theory: The first computational model of a neuron was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943.



It may be divided into 2 parts. The first part, ***g*** takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, ***f*** makes a decision. Lets suppose that I want to predict my own decision, whether to watch a random football game or not on TV. The inputs are all boolean i.e., {0,1} and my output variable is also boolean {0: Will watch it, 1: Won't watch it}.

- So, ***x_1*** could be *isPremierLeagueOn* (I like Premier League more)
- ***x_2*** could be *isItAFriendlyGame* (I tend to care less about the friendlies)
- ***x_3*** could be *isNotHome* (Can't watch it when I'm running errands. Can I?)
- ***x_4*** could be *isManUnitedPlaying* (I am a big Man United fan. GGMU!) and so on.

These inputs can either be *excitatory* or *inhibitory*. Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs i.e., if ***x_3*** is 1 (not home) then my output will always be 0 i.e., the neuron will never fire, so ***x_3*** is an inhibitory input. Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together. Formally, this is what is going on:

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

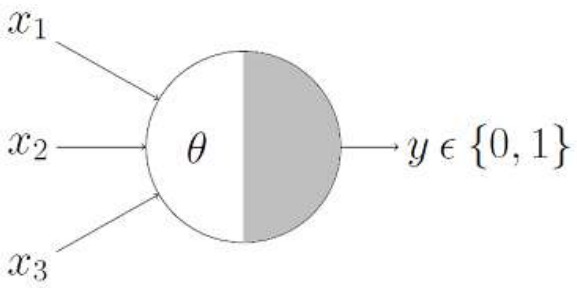
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

We can see that ***g*(x)** is just doing a sum of the inputs — a simple aggregation. And ***theta*** here is called thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the ***theta*** is 2 here. This is called the Thresholding Logic.

Boolean Functions Using M-P Neuron

So far we have seen how the M-P neuron works. Now lets look at how this very neuron can be used to represent a few boolean functions. Mind you that our inputs are all boolean and the output is also boolean so essentially, the neuron is just trying to learn a boolean function. A lot of boolean decision problems can be cast into this, based on appropriate input variables— like whether to continue reading this post, whether to watch Friends after reading this post etc. can be represented by the M-P neuron.

M-P Neuron: A Concise Representation



This representation just denotes that, for the boolean inputs ***x_1***, ***x_2*** and ***x_3*** if the ***g*(x)** i.e., **sum ≥ theta**, the neuron will fire otherwise, it won't.

Program:

```
clear;
clc;
disp('Enter the Weights: ');
w1 = input('Weight 1 = ');
w2 = input('Weight 2 = ');
disp('Enter threshold value: ');
theta = input('Theta = ');
y = [0,0,0,0];
x1 = [1,1,0,0];
x2 = [1,0,1,0];
z = [1,0,0,0];
con = 1;
while con
```

```
zin = x1*w1 + x2*w2;
for i = 1:4
if zin(i) >= theta
        y(i) = 1;
    else
        y(i) = 0;
    end;
end;
disp('Output of net: ');
disp(y);
if y == z
    con = 0;
else
   disp('Network is not leaning!');
w1 = input('Weight w1 = ');
w2 = input('Weight w2 = ');
theta = input('Theta = ');
end;
end;
disp('Mucculloch Pits Model Function: ')
disp('Weight of Nuron ');
disp(w1);
disp(w2);
disp('Threshold Value: ')
disp(theta);
```

Output:

Enter the Weights:

Weight 1 = 1

Weight 2 = 1

Enter threshold value:

Theta = 2

Output of net:

1 0 0 0

Mucculloch Pits Model Function:

Weight of Nuron

1

1

Threshold Value:

2

Coclusion: The MP Neuron model is implemented successfully.

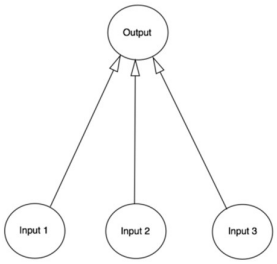
Experiment No.02

Linear Regression

It is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

what is a neural network?

The human brain consists of 100 billion cells called neurons, connected together by synapses. If sufficient synaptic inputs to a neuron fire, that neuron will also fire. We call this process "thinking".



We can model this process by creating a neural network on a computer. It's not necessary to model the biological complexity of the human brain at a molecular level, just its higher-level rules. We use a mathematical technique called matrices, which are grids of numbers. To make it really simple, we will just model a single neuron, with three inputs and one output.

Linear Regression using simple Neural Network

Hypothesis and cost function

Hypothesis is a sort of function that expect to represent the pattern of given data. It is usually expressed with some linear equation, but in this post, it will cover the simple linear regression, so the hypothesis may be simple linear equation like this,

$$H(x) = W x + bH(x)=Wx+b$$

Here, xx is the given data, and WW is weight vector, and bb is bias. So the output H(x)H(x) is the form of product weight vector and given data, adding bias. The purpose of hypothesis is to clearly represent the actual output(yy) with this formula. To do this, hypothesis should be same as actual output.

Maybe the simple hypothesis can be drawn as line on 2D-space. But as you know that, it is hard to clearly represent all the data with given hypothesis. There're must be the error between the output of hypothesis and actual output. Through this process, we should find the way to minimize the error between them.

Cost is the error I mentioned. Usually, lots of data is given, so the cost can be expressed with the average of error. And as you notice that, hypothesis is related on weight vector(WW) and bias(bb), so the cost can be described as a function:

$$cost(W,b)=m1i=1∑m(H(xi)−yi)2$$

And there are lots of cost function in the world. The expression above is the mean squared error(MSE for short), cause we measure the error with Euclidean distance ((x-y)^2(x−y)2). Maybe we can use the cost function with mean absolute error (MAE for short). Anyway, we'll use cost function with MSE.

So, we need to find the best WW and bb for minimizing the cost function. The process we try to find this is called learning or training.

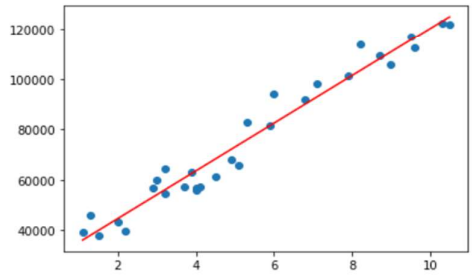
Expt-2 : Linear regression using single neural model

Using sklearn

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
```

In [2]:
```
data = pd.read_csv('data.csv')
X = data.iloc[:, 0].values.reshape(-1, 1)
Y = data.iloc[:, 1].values.reshape(-1, 1)
lin_reg = LinearRegression()
lin_reg.fit(X, Y)
Y_pred = lin_reg.predict(X)
```

In [3]:
```
plt.scatter(X, Y)
plt.plot(X, Y_pred, color='red')
plt.show()
```



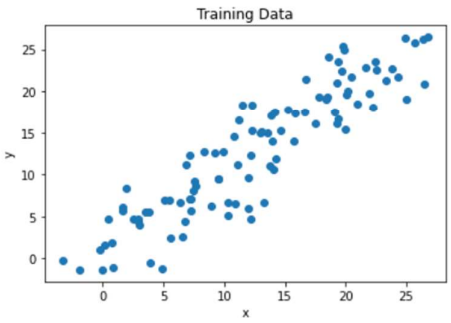
Using Single Neuron Model

In [20]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

In [21]:
```
np.random.seed(101)
tf.random.set_seed(101)
```

In [22]:
```
x = np.linspace(0, 25, 100)
y = np.linspace(0, 25, 100)
# Adding noise to the random linear data
x += np.random.uniform(-4, 4, 100)
y += np.random.uniform(-4, 4, 100)
n = len(x) # Number of data points
```

In [23]:
```
# Plot of Training Data
plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title("Training Data")
plt.show()
```



In [24]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

In [25]:
```
X = tf.placeholder("float")
Y = tf.placeholder("float")
```

In [26]:
```
W = tf.Variable(np.random.randn(), name = "W")
b = tf.Variable(np.random.randn(), name = "b")
```

In [27]:
```
learning_rate = 0.01
training_epochs = 1000
```

In [28]:
```
# Hypothesis
y_pred = tf.add(tf.multiply(X, W), b)

# Mean Squared Error Cost Function
cost = tf.reduce_sum(tf.pow(y_pred-Y, 2)) / (2 * n)

# Gradient Descent Optimizer
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Global Variables Initializer
init = tf.global_variables_initializer()
```

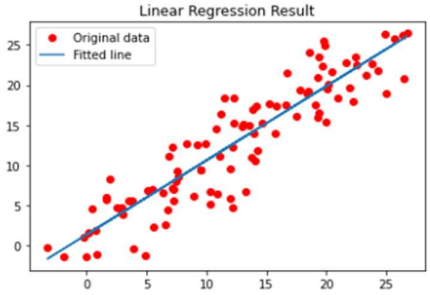
In [29]:
```
# Starting the Tensorflow Session
with tf.Session() as sess:

    # Initializing the Variables
    sess.run(init)

    # Iterating through all the epochs
    for epoch in range(training_epochs):

        # Feeding each data point into the optimizer using Feed Dictionary
        for (_x, _y) in zip(x, y):
            sess.run(optimizer, feed_dict = {X : _x, Y : _y})

        # Displaying the result after every 50 epochs
        if (epoch + 1) % 50 == 0:
            # Calculating the cost a every epoch
            c = sess.run(cost, feed_dict = {X : x, Y : y})
            print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(W), "b =", sess.run(b))

    # Storing necessary values to be used outside the Session
    training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
    weight = sess.run(W)
    bias = sess.run(b)
```

```
Epoch 50 : cost = 5.0503244 W = 0.9835112 b = 0.14373533
Epoch 100 : cost = 5.0009503 W = 0.9753103 b = 0.31200755
Epoch 150 : cost = 4.964513 W = 0.9681324 b = 0.4592994
Epoch 200 : cost = 4.937812 W = 0.961849 b = 0.58822864
Epoch 250 : cost = 4.91842 W = 0.95634913 b = 0.7010838
Epoch 300 : cost = 4.9044952 W = 0.951535 b = 0.7998663
Epoch 350 : cost = 4.8946433 W = 0.9473212 b = 0.8863343
Epoch 400 : cost = 4.8878093 W = 0.94363254 b = 0.9620219
Epoch 450 : cost = 4.8832 W = 0.94040394 b = 1.0282723
Epoch 500 : cost = 4.880215 W = 0.9375775 b = 1.086269
Epoch 550 : cost = 4.878408 W = 0.93510383 b = 1.1370277
Epoch 600 : cost = 4.877444 W = 0.93293846 b = 1.18146
Epoch 650 : cost = 4.877072 W = 0.9310429 b = 1.220356
Epoch 700 : cost = 4.8771086 W = 0.9293839 b = 1.254396
Epoch 750 : cost = 4.8774176 W = 0.92793167 b = 1.2841942
Epoch 800 : cost = 4.877902 W = 0.9266605 b = 1.3102798
Epoch 850 : cost = 4.8784885 W = 0.9255478 b = 1.3331128
Epoch 900 : cost = 4.8791265 W = 0.9245738 b = 1.3530993
Epoch 950 : cost = 4.8797803 W = 0.92372143 b = 1.3705887
Epoch 1000 : cost = 4.8804255 W = 0.9229752 b = 1.3859004
```

In [30]:
```
# Calculating the predictions
predictions = weight * x + bias
print("Training cost =", training_cost, "Weight =", weight, "bias =", bias, '\n')
```

```
Training cost = 4.8804255 Weight = 0.9229752 bias = 1.3859004
```

In [31]:
```
# Plotting the Results
plt.plot(x, y, 'ro', label ='Original data')
plt.plot(x, predictions, label ='Fitted line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
```



Conclusion: Thus, Linear Regression using single neural network/ model is implemented successfully.

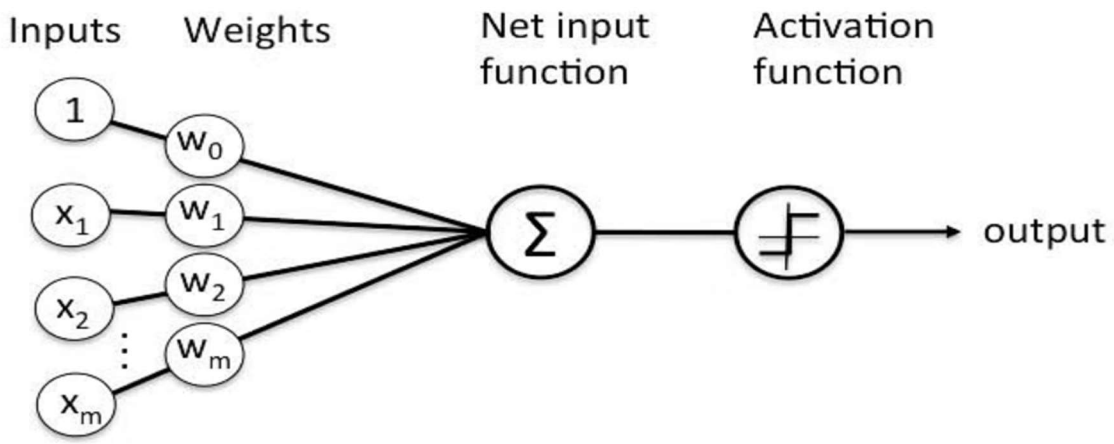
Experiment No.03

Theory :-

Linear Perceptron Model -

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

A perceptron model is also classified as one of the best and most specific types of Artificial Neural networks. Being a supervised learning algorithm, it is a single-layer neural network with four main parameters: input values, weights and Bias, net sum, and an activation function.

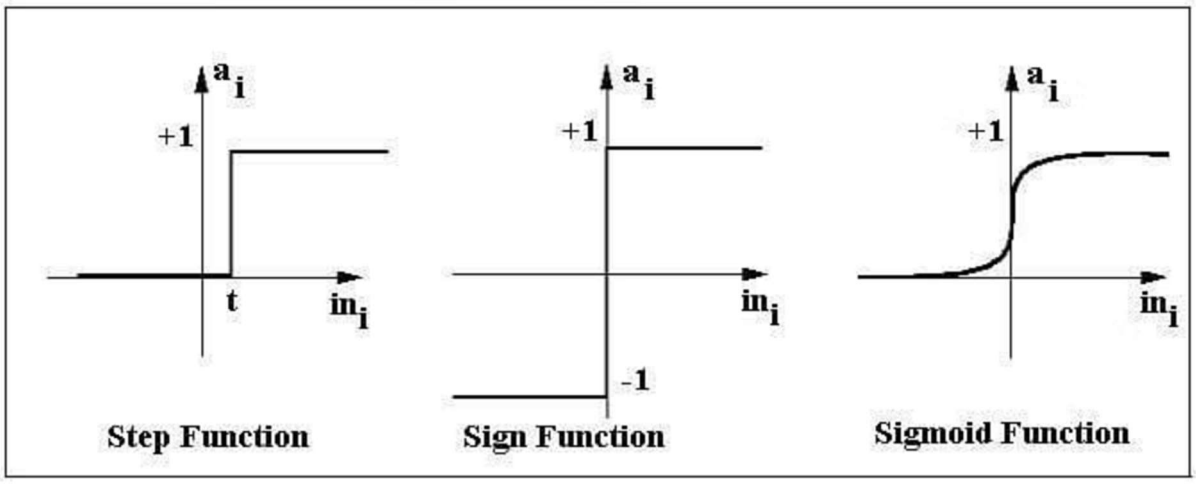


The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output.

Boolean output is based on inputs and has only two values: Yes and No (True and False). The summation function "∑" multiplies all inputs of "x" by weights "w" and then adds them up as follows:

$$w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.



Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

Characteristics of Perceptron

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

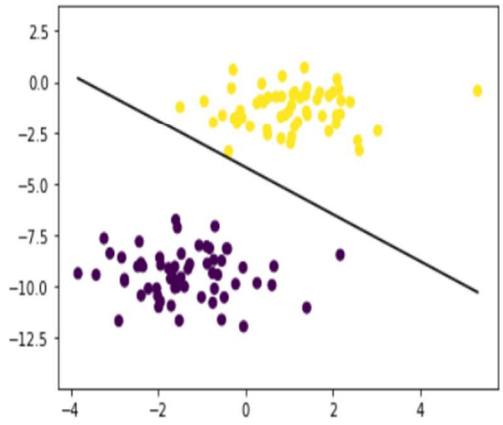
Program –

In [1]:
```
import numpy as np
```

In [2]:
```
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation_func = self._unit_step_func
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        y_ = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.n_iters):

            for idx, x_i in enumerate(X):

                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_func(linear_output)

                # Perceptron update rule
                update = self.lr * (y_[idx] - y_predicted)

                self.weights += update * x_i
                self.bias += update

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_func(linear_output)
        return y_predicted

    def _unit_step_func(self, x):
        return np.where(x >= 0, 1, 0)

if __name__ == "__main__":

    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn import datasets

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    X, y = datasets.make_blobs(
        n_samples=150, n_features=2, centers=2, cluster_std=1.05, random_state=2
    )
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=123
    )

    p = Perceptron(learning_rate=0.01, n_iters=1000)
    p.fit(X_train, y_train)
    predictions = p.predict(X_test)

    print("Perceptron classification accuracy", accuracy(y_test, predictions))

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    plt.scatter(X_train[:, 0], X_train[:, 1], marker="o", c=y_train)
```

```
x0_1 = np.amin(X_train[:, 0])
x0_2 = np.amax(X_train[:, 0])

x1_1 = (-p.weights[0] * x0_1 - p.bias) / p.weights[1]
x1_2 = (-p.weights[0] * x0_2 - p.bias) / p.weights[1]

ax.plot([x0_1, x0_2], [x1_1, x1_2], "k")

ymin = np.amin(X_train[:, 1])
ymax = np.amax(X_train[:, 1])
ax.set_ylim([ymin - 3, ymax + 3])

plt.show()
```

```
Perceptron classification accuracy 1.0
```



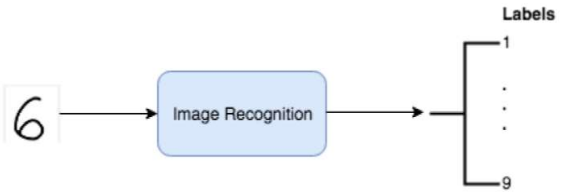
Conclusion: Thus, Linear Perceptron model is implemented successfully.

Experiment No. 04

Theory:-

Handwritten Character Recognition

Handwritten character recognition is a field of research in artificial intelligence, computer vision, and pattern recognition. A computer performing handwriting recognition is said to be able to acquire and detect characters in paper documents, pictures, touch-screen devices and other sources and convert them into machine-encoded form. Its application is found in optical character recognition, transcription of handwritten documents into digital documents and more advanced intelligent character recognition systems.
Handwritten character recognition can be thought of as a subset of the image recognition problem.



The general flow of an image recognition algorithm.

Basically, the algorithm takes an image (image of a handwritten digit) as an input and outputs the likelihood that the image belongs to different classes (the machine-encoded digits, 1–9).
The goal is to take an image of a handwritten digit and determine what that digit is. The digits range from one (1) through nine (9).
We will look into the Support Vector Machines (SVMs) and Nearest Neighbor (NN) techniques to solve the problem. The tasks involved are the following:

1. Download the MNIST dataset
2. Preprocess the MNIST dataset
3. Train a classifier that can categorize the handwritten digits
4. Apply the model on the test set and report its accuracy

The dataset for this problem will be downloaded from kaggle, which was taken from the famous MNIST (Modified National Institute of Standards and Technology) dataset.

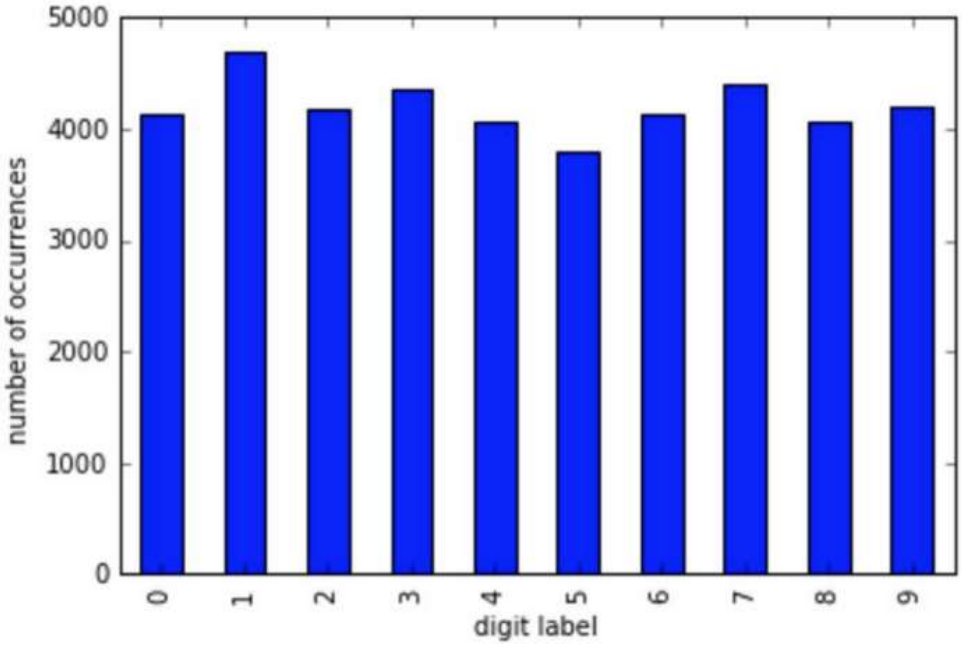
Metrics

We will be using the **accuracy score** to quantify the performance of our model. The accuracy will tell us what percentage of our test data was classified correctly. The accuracy is a good metric choice because it will be easy to compare our model's performance to that of the benchmark as it uses the same metric. Also, our dataset is balanced (equal number of training examples for each label) which makes the accuracy appropriate for this problem.

Some examples of the dataset.

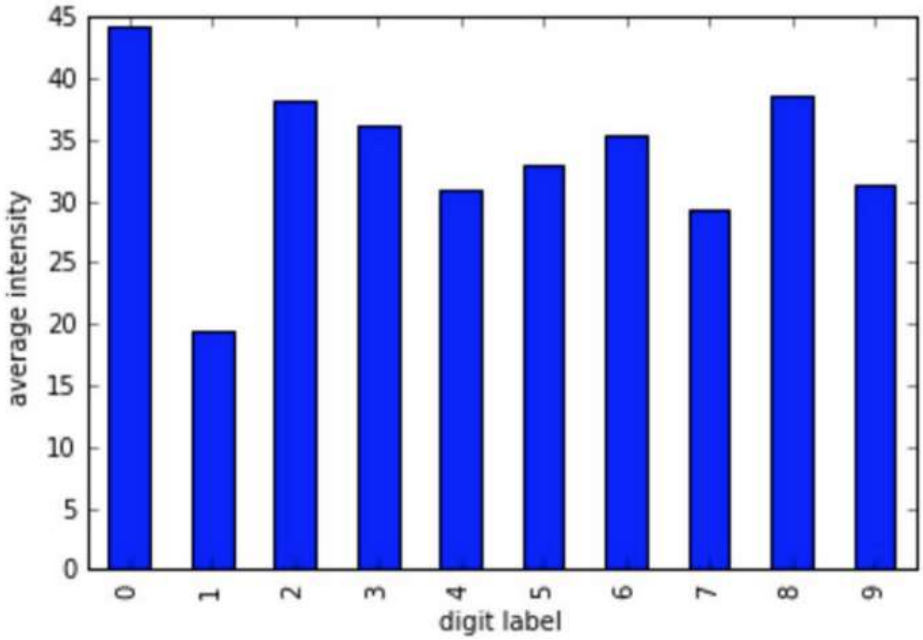
Exploratory Visualization

We have counted the number of occurrences of each label in the training set. The figure below illustrates the distribution of these labels. It is obvious from the figure that the distribution is uniform meaning our dataset is balanced.



The number of occurrences of each label in the dataset.

We'd also like to know more about average intensity, that is the average value of a pixel in an image for the different digits. Intuition tells me that the digit "1" will on average have less intensity than say an "8".



The average intensity of each label in the dataset.

As we can see, there are differences in intensities and our intuition was correct. "8" has a higher intensity than a "1". Also, "0" has the highest intensity, even higher than "8" which is surprising. This could be attributed to the fact that different people write their digits differently. Calculating the standard deviation of intensities gives a value of **11.08** which shows that there exists some variation in the way the digits are written.

Program/Output:-

```
In [1]: import tensorflow as tf
        from tensorflow import keras
        import matplotlib.pyplot as plt

        import numpy as np
```

```
In [3]: (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

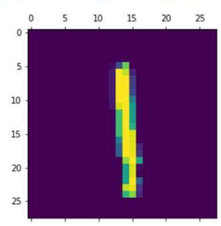
```
In [4]: len(x_train)
Out[4]: 60000
```

```
In [5]: len(x_test)
Out[5]: 10000
```

```
In [6]: x_train[0].shape
Out[6]: (28, 28)
```

```
In [7]: x_train[0]
Out[7]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
                 18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
                253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
```

```
In [8]: plt.matshow(x_train[8])
Out[8]: <matplotlib.image.AxesImage at 0x1f0c422ee20>
```




```
In [9]: y_train[2]
Out[9]: 4
```

```
In [10]: y_train[:5]
Out[10]: array([5, 0, 4, 1, 9], dtype=uint8)
```

```
In [11]: x_train = x_train/255
         x_test = x_test/255
```

Flatten the array

```
In [12]: x_train_flattened = x_train.reshape(len(x_train), 28*28)
         x_train_flattened.shape
Out[12]: (60000, 784)
```

```
In [13]: x_test_flattened =  x_test.reshape(len(x_test), 28*28)#reshap()
         x_test_flattened.shape
Out[13]: (10000, 784)
```

```
In [14]: arr = np.around(x_train[0])
```

```
In [15]: x_train_flattened[0]
Out[15]: array([0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
In [16]: model = keras.Sequential([
             keras.layers.Dense(10, input_shape=(784,), activation ="sigmoid" )

         ])

         model.compile(optimizer = "adam", loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

         model.fit(x_train_flattened, y_train, epochs=5)

         Epoch 1/5
         1875/1875 [==============================] - 7s 3ms/step - loss: 0.4700 - accuracy: 0.8784
         Epoch 2/5
         1875/1875 [==============================] - 6s 3ms/step - loss: 0.3040 - accuracy: 0.9150
         Epoch 3/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.2836 - accuracy: 0.9204
         Epoch 4/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.2735 - accuracy: 0.9240
```

Model Evaluation on Testing

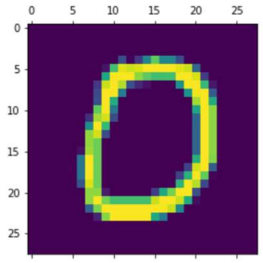
In [17]:
```
model.evaluate(x_test_flattened, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2714 - accuracy: 0.9235
```

Out[17]:
```
[0.27143678069114685, 0.9235000014305115]
```

In [18]:
```
plt.matshow(x_test[10])
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x1f0c4193a90>
```



In [19]:
```
y_pred = model.predict(x_test_flattened)
y_pred[0]
```

```
313/313 [==============================] - 1s 2ms/step
```

Out[19]:
```
array([3.1486277e-02, 5.6377291e-07, 5.5533860e-02, 9.6941030e-01,
       3.7192116e-03, 1.8132681e-01, 3.2584201e-06, 9.9983460e-01,
       1.0339971e-01, 6.7108470e-01], dtype=float32)
```

In [20]:
```
y_pred_labels = [np.argmax(i) for i in y_pred]
y_pred_labels[:5]
```

Out[20]:
```
[7, 2, 1, 0, 4]
```

In [21]:
```
y_test[:5]
```

Out[21]:
```
array([7, 2, 1, 0, 4], dtype=uint8)
```

In [22]:
```
import numpy as np
```

In [23]:
```
np.argmax(y_pred[10])
```

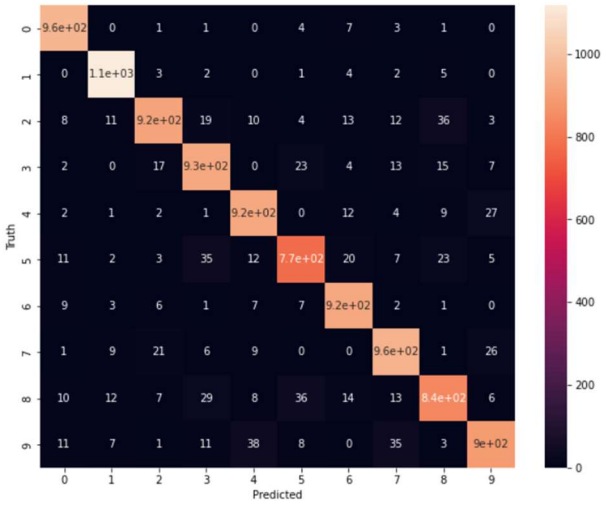
Out[23]:
```
0
```

In [24]:
```
conf_mat = tf.math.confusion_matrix(labels= y_test, predictions=y_pred_labels)
conf_mat
```

Out[24]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 963,    0,    1,    1,    0,    4,    7,    3,    1,    0],
       [   0, 1118,    3,    2,    0,    1,    4,    2,    5,    0],
       [   8,   11,  916,   19,   10,    4,   13,   12,   36,    3],
       [   2,    0,   17,  929,    0,   23,    4,   13,   15,    7],
       [   2,    1,    2,    1,  924,    0,   12,    4,    9,   27],
       [  11,    2,    3,   35,   12,  774,   20,    7,   23,    5],
       [   9,    3,    6,    1,    7,    7,  922,    2,    1,    0],
       [   1,    9,   21,    6,    9,    0,    0,  955,    1,   26],
       [  10,   12,    7,   29,    8,   36,   14,   13,  839,    6],
       [  11,    7,    1,   11,   38,    8,    0,   35,    3,  895]])>
```

In [26]:
```
import seaborn as sns
plt.figure(figsize =(10,8))
sns.heatmap(conf_mat, annot=True)
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

Out[26]:
```
Text(69.0, 0.5, 'Truth')
```



Adding Dense Layers

In [27]:
```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation ="relu" ),
    keras.layers.Dense(10, activation ="sigmoid" )
])

model.compile(optimizer = "adam", loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

model.fit(x_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2707 - accuracy: 0.9219
Epoch 2/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1225 - accuracy: 0.9645
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0849 - accuracy: 0.9743
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0642 - accuracy: 0.9804
Epoch 5/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0497 - accuracy: 0.9845
```

Out[27]: <keras.callbacks.History at 0x1f0eeb0cc10>

Model Evaluation

In [28]:
```
model.evaluate(x_test_flattened, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0882 - accuracy: 0.9732
```

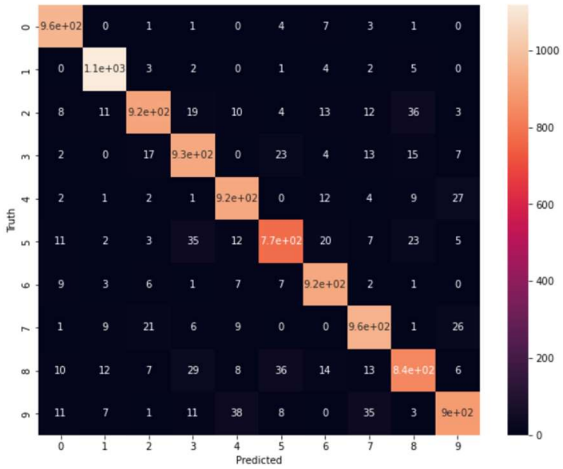
Out[28]: [0.08823973685503006, 0.9732000231742859]

In [29]:
```
conf_mat = tf.math.confusion_matrix(labels= y_test, predictions=y_pred_labels)
conf_mat
```

Out[29]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 963,    0,    1,    1,    0,    4,    7,    3,    1,    0],
       [   0, 1118,    3,    2,    0,    1,    4,    2,    5,    0],
       [   8,   11,  916,   19,   10,    4,   13,   12,   36,    3],
       [   2,    0,   17,  929,    0,   23,    4,   13,   15,    7],
       [   2,    1,    2,    1,  924,    0,   12,    4,    9,   27],
       [   2,    0,   17,  929,    0,   23,    4,   13,   15,    7],
       [   2,    1,    2,    1,  924,    0,   12,    4,    9,   27],
       [  11,    2,    3,   35,   12,  774,   20,    7,   23,    5],
       [   9,    3,    6,    1,    7,    7,  922,    2,    1,    0],
       [   1,    9,   21,    6,    9,    0,    0,  955,    1,   26],
       [  10,   12,    7,   29,    8,   36,   14,   13,  839,    6],
       [  11,    7,    1,   11,   38,    8,    0,   35,    3,  895]])>
```

In [30]:
```
import seaborn as sns
plt.figure(figsize =(10,8))
sns.heatmap(conf_mat, annot=True)
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

Out[30]: Text(69.0, 0.5, 'Truth')



Using keras to flatten the layers

In [31]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(100, input_shape=(784,), activation ="relu" ),
    keras.layers.Dense(10, activation ="sigmoid" )
])

model.compile(optimizer = "adam", loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2791 - accuracy: 0.9203
Epoch 2/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1265 - accuracy: 0.9626
Epoch 3/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0857 - accuracy: 0.9745
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0653 - accuracy: 0.9803
Epoch 5/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0518 - accuracy: 0.9845
```

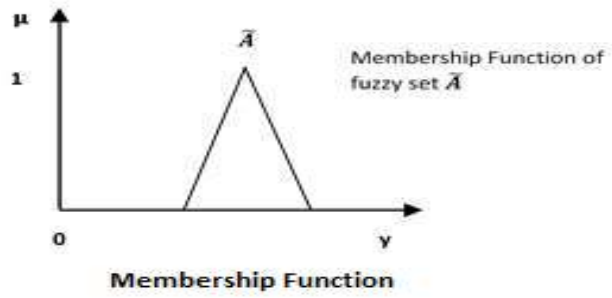
Out[31]: <keras.callbacks.History at 0x1f0dff1c970>

Experiment No.05

Theory:

Membership function: -

Membership function represents the degree of truth in fuzzy logic.

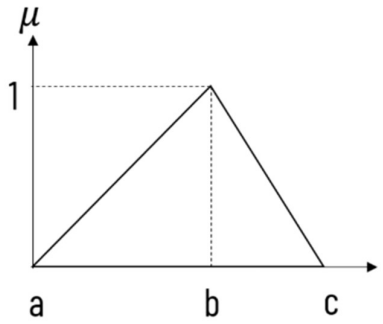


Membership Function

1. Membership functions were first introduced in 1965 by Lofti A. Zadeh in his first research paper "fuzzy sets".
2. Membership functions characterize fuzziness (i.e., all the information in fuzzy set), whether the elements in fuzzy sets are discrete or continuous.
3. Membership functions can be defined as a technique to solve practical problems by experience rather than knowledge.
4. Membership functions are represented by graphical forms.
5. Rules for defining fuzziness are fuzzy too.

1. Triangular membership function:

This is one of the most widely accepted and used membership function (MF) in fuzzy controller design. The triangle which fuzzifiers the input can be defined by three parameters a, b and c, where and c defines the base and b defines the height of the triangle.



Here, in the diagram, X axis represents the input from the process (such as air conditioner, washing machine, etc.) and Y axis represents corresponding fuzzy value.

If input x = b, then it is having full membership in the given set. So,

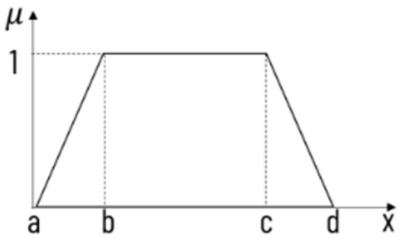
$\mu(x) = 1$, if $x = b$

And if input is less than a or greater then b, then it does belong to fuzzy set at all, and its membership value will be 0

$\mu(x)=0$, $x<a$ or $x>c$

2. Trapezoidal membership function:

Trapezoidal membership function is defined by four parameters: a, b, c and d. Span b to c represents the highest membership value that element can take. And if x is between (a, b) or (c, d), then it will have membership value between 0 and 1.



We can apply the triangle MF if elements is in between a to b or c to d.

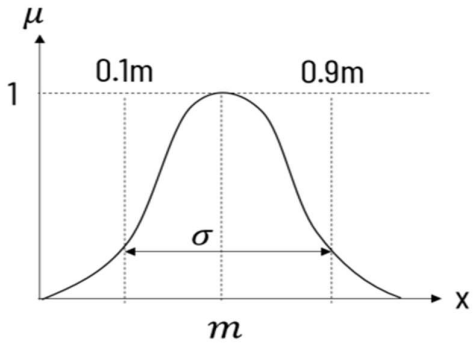
It is quite obvious to combine all together as,

$$\mu_{trapezoidal}(x; a, b, c, d) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ \dfrac{d-x}{d-c}, & c \le x \le d \\ 0, & d \le x \end{cases}$$

$$= \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right)$$

3. Gaussian membership function:

A Gaussian MF is specified by two parameters {m, σ} and can be defined as follows.



$$\mu_{gaussian}(x; m, \sigma) = e^{-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2}$$

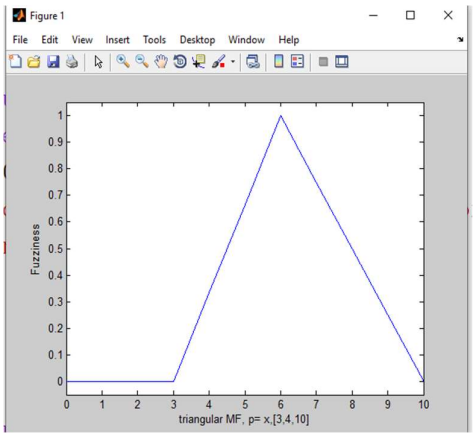
Gaussian membership function

In this function, m represents the mean / centre of the gaussian curve and σ represents the spread of the curve. This is more natural way of representing the data distribution, but due to mathematical complexity it is not much used for fuzzification.

Program and Output :-

% 1. Triangular MF

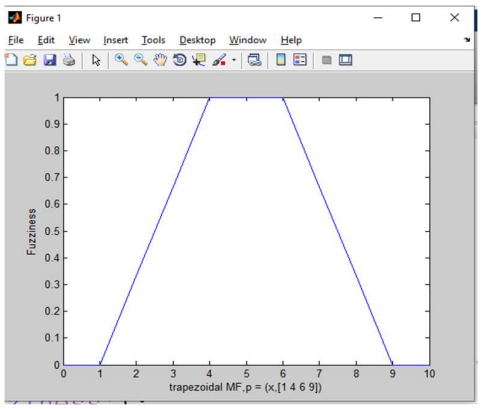
```
x = 0:10;
y = trimf(x,[3 6 10]);
plot(x, y);
xlabel('triangular MF, p= x,[3,4,10]')
ylabel('Fuzziness')
ylim([-0.05 1.05]);
```



Triangular Membership Function

% 2. Trapezoidal MF

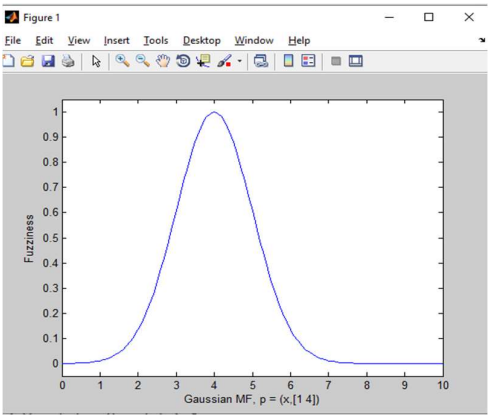
```
x = 0:10;
y = trapmf(x,[1 4 6 9]);
plot(x,y);
xlabel('trapezoidal MF,p = (x,[1 4 6
9])');
ylabel('Fuzziness');
ylim([-0.05,1,05]);
```



Trapezoidal Membership Function

% 3. Gaussian MF

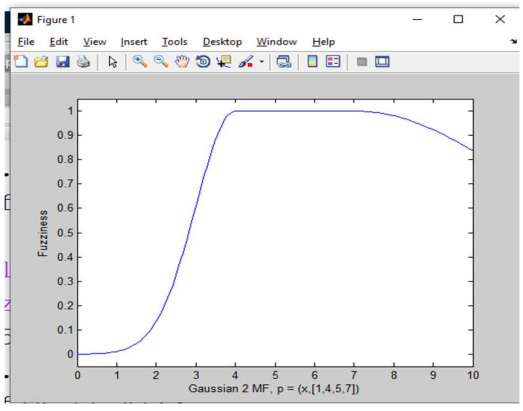
```
x = 0:0.1:10;
y = gaussmf(x,[1 4]);
plot(x,y);
xlabel('Gaussian MF, p = (x,[1 4])');
ylabel('Fuzziness');
ylim([-0.05 1.05]);
```



Gaussian Membership Function

% 4. Gaussian 2 MF

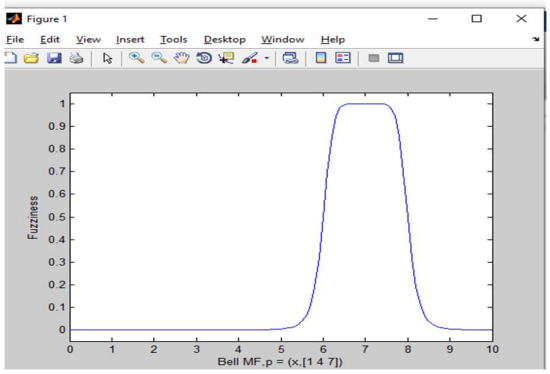
```
x = 0:0.1:10;
y = gauss2mf(x, [1,4,5,7]);
plot(x,y);
xlabel('Gaussian 2 MF, p =
(x,[1,4,5,7])');
ylabel('Fuzziness');
ylim([-0.05 1.05]);
```



Gaussian 2 Membership Function

% 5. Bell MF

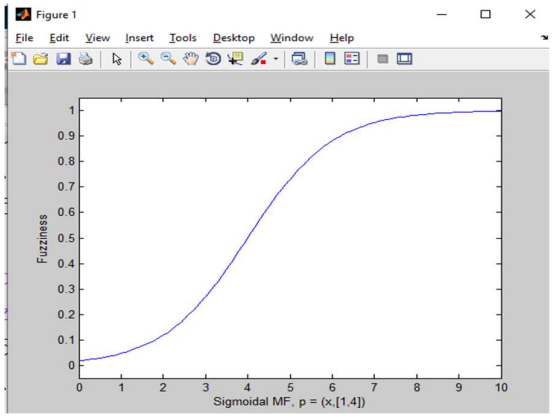
```
x = 0:0.1:10;
y = gbellmf(x,[1 4 7]);
plot(x,y);
xlabel('Bell MF,p = (x,[1 4 7])');
ylabel('Fuzziness');
ylim([-0.05 1.05]);
```



Bell Membership Function

% 6. sigmoidal MF

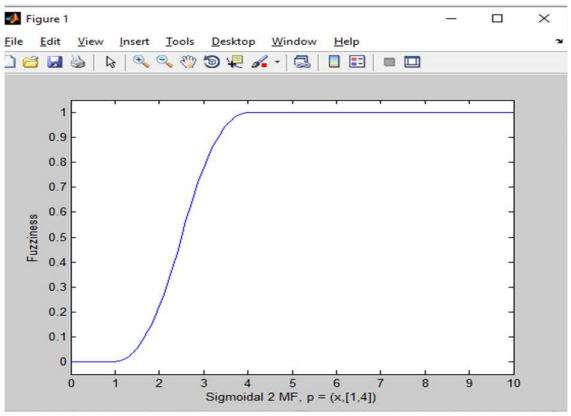
```
x = 0:0.1:10;
y = sigmf(x,[1,4]);
plot(x,y);
xlabel('Sigmoidal MF, p = (x,[1,4])');
ylabel('Fuzziness');
ylim([-0.05 1.05]);
```



Sigmoidal Membership Function

% 7. Sigmoidal 2 MF

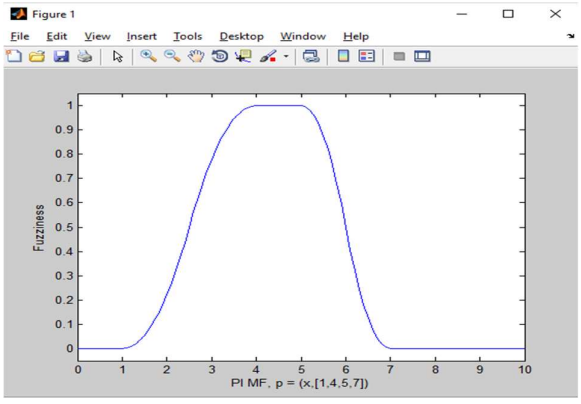
```
x = 0:0.1:10;
y = smf(x,[1,4]);
plot(x,y);
xlabel('Sigmoidal MF, p = (x,[1,4])');
ylabel('Fuzziness');
ylim([-0.05 1.05])
```



Sigmoidal 2 Membership Function

% 8. PI MF

```
x = 0:0.1:10;
y = pimf(x,[1,4, 5,7]);
plot(x,y);
xlabel('PI MF, p = (x,[1,4,5,7])');
ylabel('Fuzziness');
```



PI Membership Function

Conclusion: Thus, Linear Perceptron model is implemented successfully.

Experiment No.06

Theory :-

Fuzzy Inference System is the key unit of a fuzzy logic system having decision making as its primary work. It uses the "IF…THEN" rules along with connectors "OR" or "AND" for drawing essential decision rules.

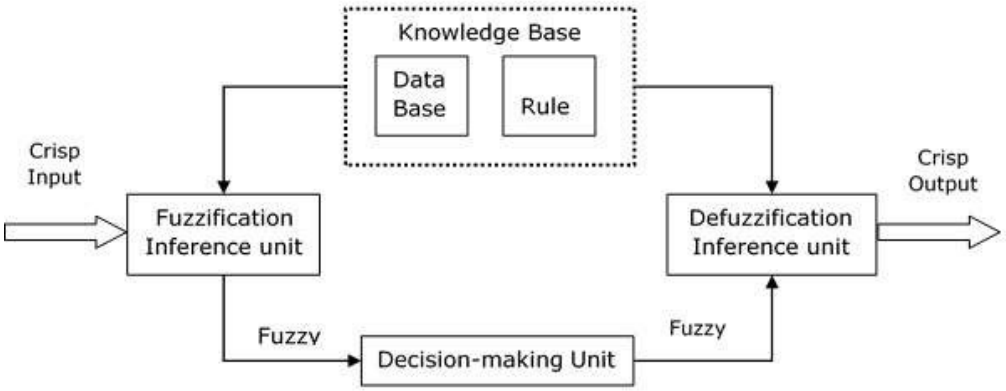
Characteristics of Fuzzy Inference System

- The output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp.
- It is necessary to have fuzzy output when it is used as a controller.
- A defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables.

Functional Blocks of FIS

The following five functional blocks will help you understand the construction of FIS −

- **Rule Base** − It contains fuzzy IF-THEN rules.
- **Database** − It defines the membership functions of fuzzy sets used in fuzzy rules.
- **Decision-making Unit** − It performs operation on rules.
- **Fuzzification Interface Unit** − It converts the crisp quantities into fuzzy quantities.
- **Defuzzification Interface Unit** − It converts the fuzzy quantities into crisp quantities. Following is a block diagram of fuzzy interference system.



Working of FIS

The working of the FIS consists of the following steps −

- A fuzzification unit supports the application of numerous fuzzification methods, and converts the crisp input into fuzzy input.
- A knowledge base - collection of rule base and database is formed upon the conversion of crisp input into fuzzy input.
- The defuzzification unit fuzzy input is finally converted into crisp output.

Methods of FIS

Let us now discuss the different methods of FIS. Following are the two important methods of FIS, having different consequent of fuzzy rules −

- Mamdani Fuzzy Inference System
- Takagi-Sugeno Fuzzy Model (TS Method)

Mamdani Fuzzy Inference System

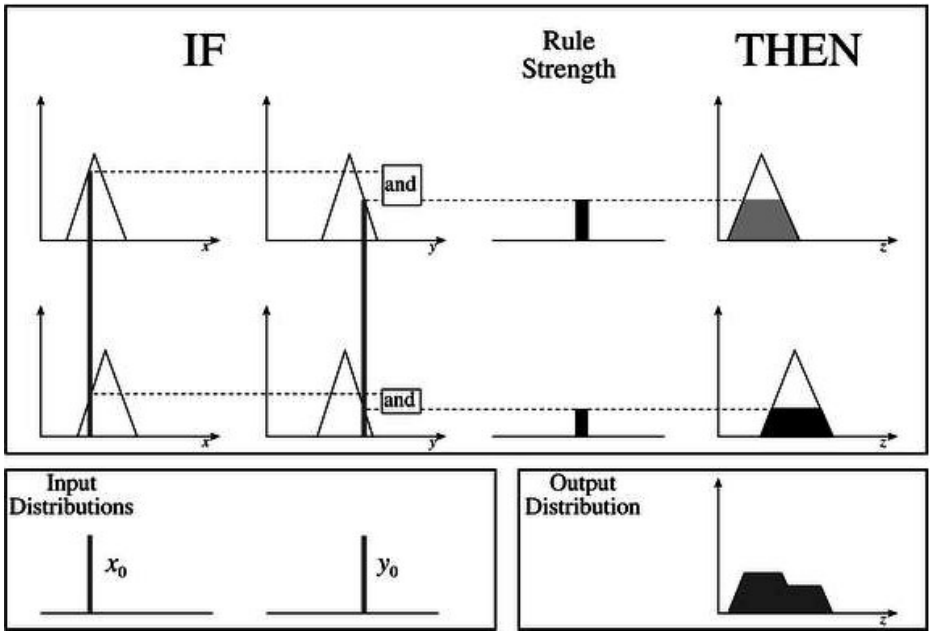
This system was proposed in 1975 by Ebhasim Mamdani. Basically, it was anticipated to control a steam engine and boiler combination by synthesizing a set of fuzzy rules obtained from people working on the system.

Steps for Computing the Output

Following steps need to be followed to compute the output from this FIS −

- **Step 1** − Set of fuzzy rules need to be determined in this step.
- **Step 2** − In this step, by using input membership function, the input would be made fuzzy.
- **Step 3** − Now establish the rule strength by combining the fuzzified inputs according to fuzzy rules.
- **Step 4** − In this step, determine the consequent of rule by combining the rule strength and the output membership function.
- **Step 5** − For getting output distribution combine all the consequents.
- **Step 6** − Finally, a defuzzified output distribution is obtained.

Block diagram of Mamdani Fuzzy Interface System



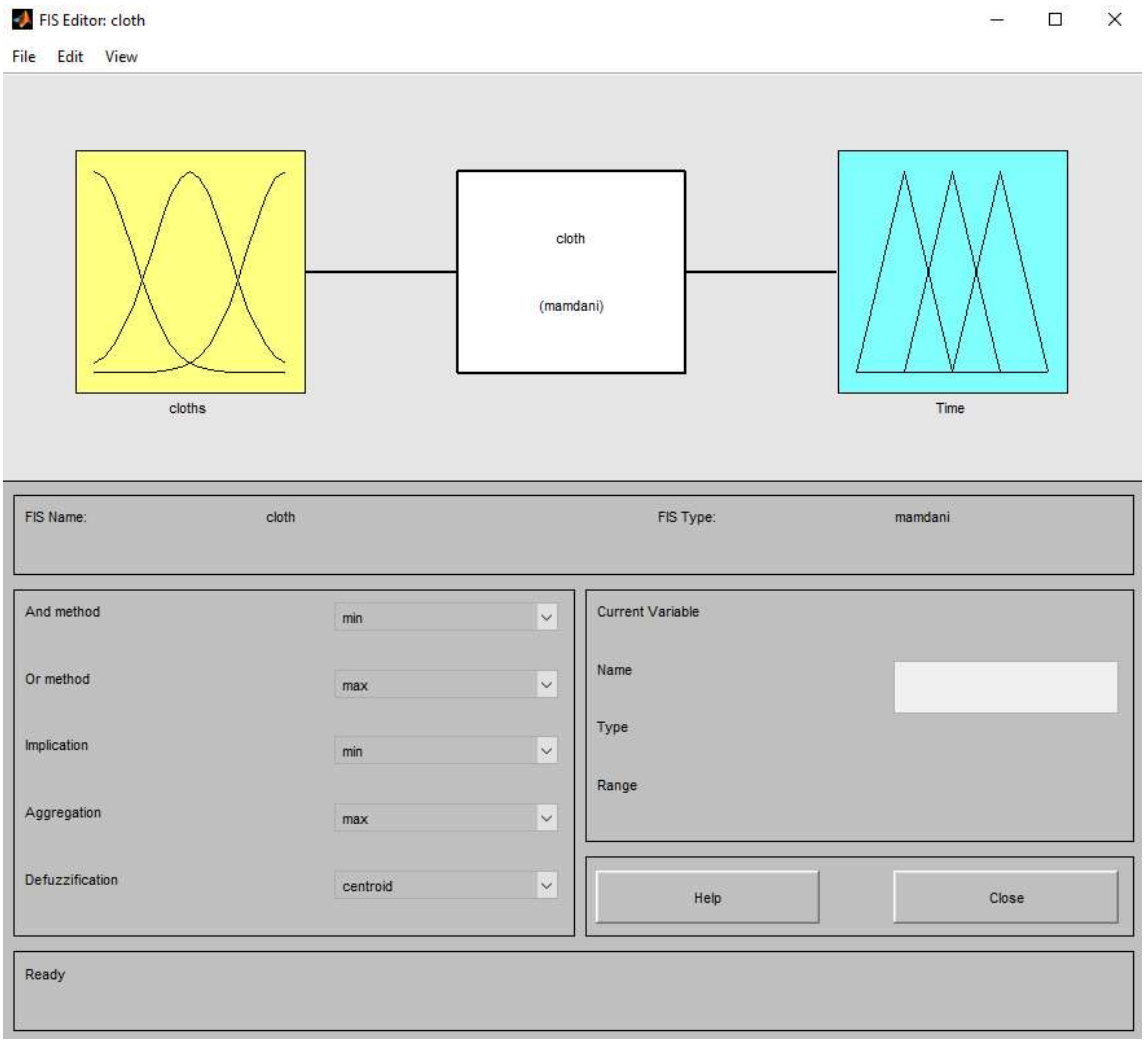

FIS Editor: cloth
File Edit View
cloth
(mamdani)
cloths
Time
FIS Name: cloth
FIS Type: mamdani
And method min
Or method max
Implication min
Aggregation max
Defuzzification centroid
Current Variable
Name
Type
Range
Help
Close
Ready

Fig. 1

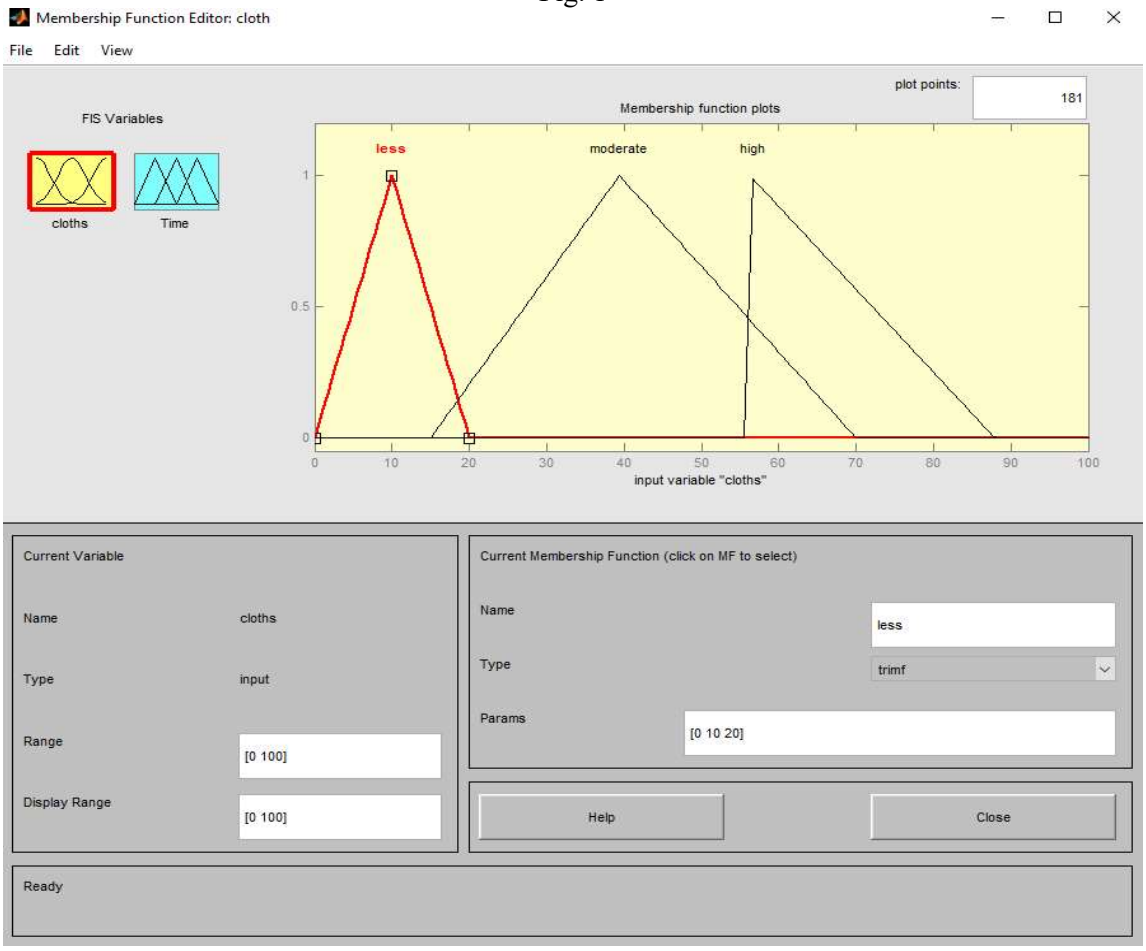

Membership Function Editor: cloth
File Edit View
plot points: 181
FIS Variables
cloths
Time
Membership function plots
less
moderate
high
input variable "cloths"
Current Variable
Name cloths
Type input
Range [0 100]
Display Range [0 100]
Current Membership Function (click on MF to select)
Name less
Type trimf
Params [0 10 20]
Help
Close
Ready

Fig. 2

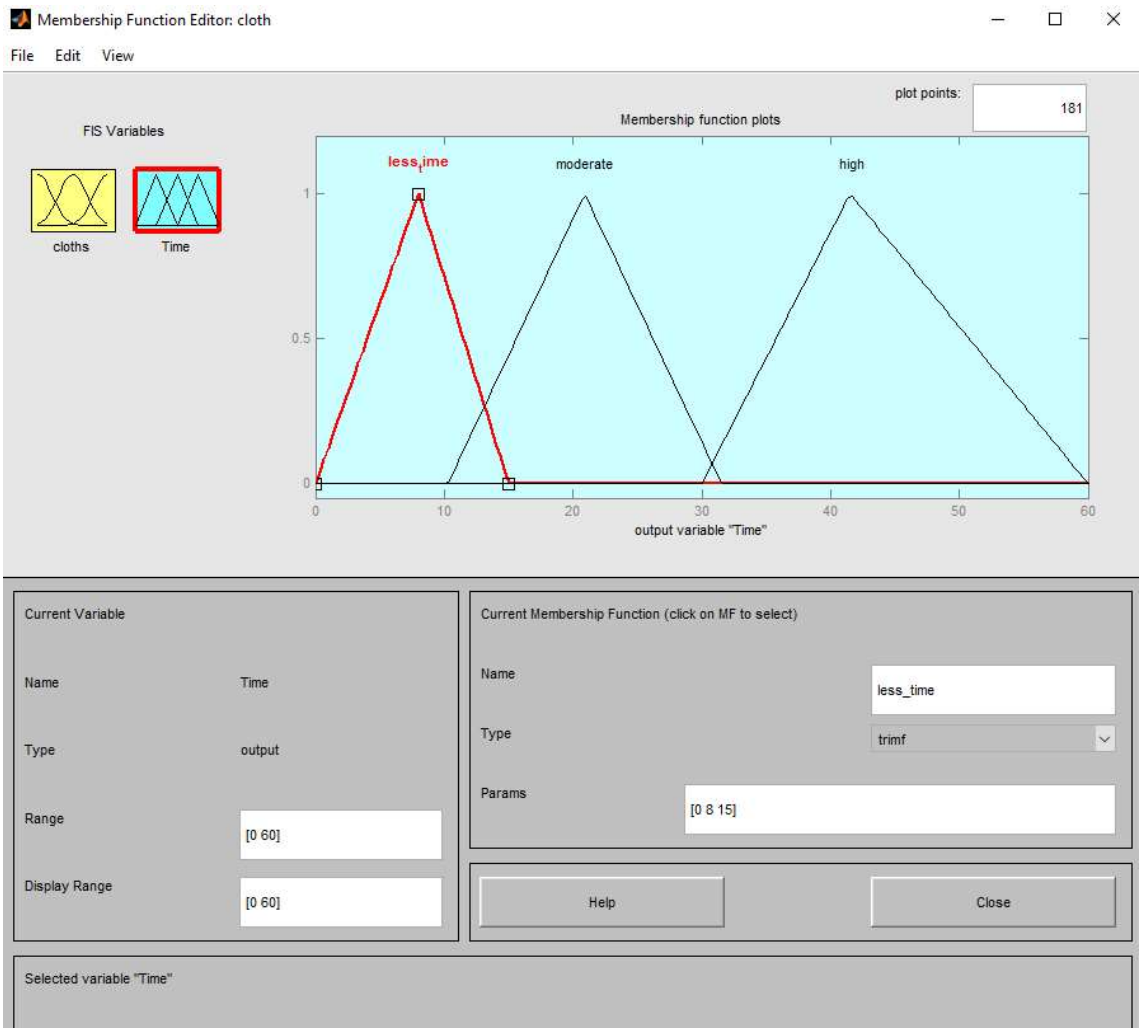

Membership Function Editor: cloth
File Edit View
plot points:
181
FIS Variables
cloths
Time
Membership function plots
less_time
moderate
high
1
0.5
0
0 10 20 30 40 50 60
output variable "Time"
Current Variable
Name
Time
Type
output
Range
[0 60]
Display Range
[0 60]
Current Membership Function (click on MF to select)
Name
less_time
Type
trimf
Params
[0 8 15]
Help
Close
Selected variable "Time"

Fig. 3

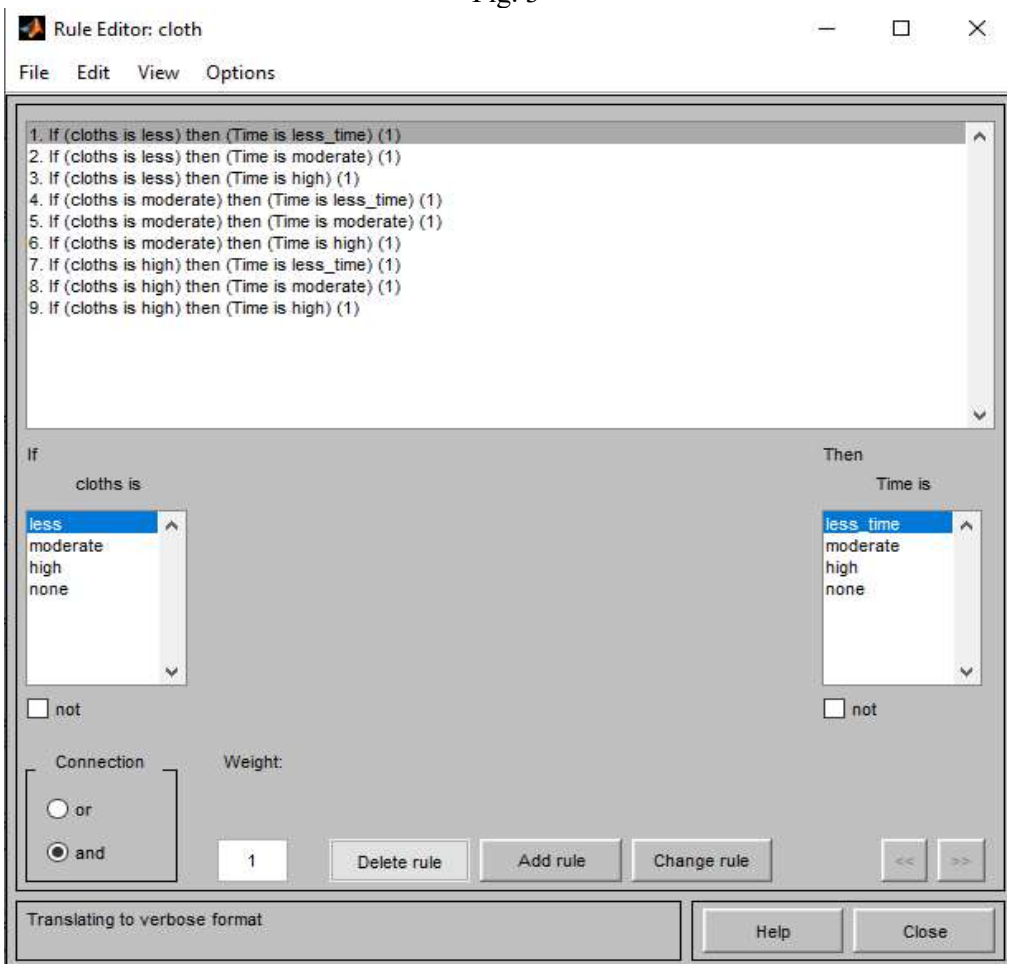

Rule Editor: cloth
File Edit View Options
1. If (cloths is less) then (Time is less_time) (1)
2. If (cloths is less) then (Time is moderate) (1)
3. If (cloths is less) then (Time is high) (1)
4. If (cloths is moderate) then (Time is less_time) (1)
5. If (cloths is moderate) then (Time is moderate) (1)
6. If (cloths is moderate) then (Time is high) (1)
7. If (cloths is high) then (Time is less_time) (1)
8. If (cloths is high) then (Time is moderate) (1)
9. If (cloths is high) then (Time is high) (1)
If
cloths is
less
moderate
high
none
not
Then
Time is
less_time
moderate
high
none
not
Connection
or
and
Weight:
1
Delete rule
Add rule
Change rule
<<
>>
Translating to verbose format
Help
Close

Fig. 4

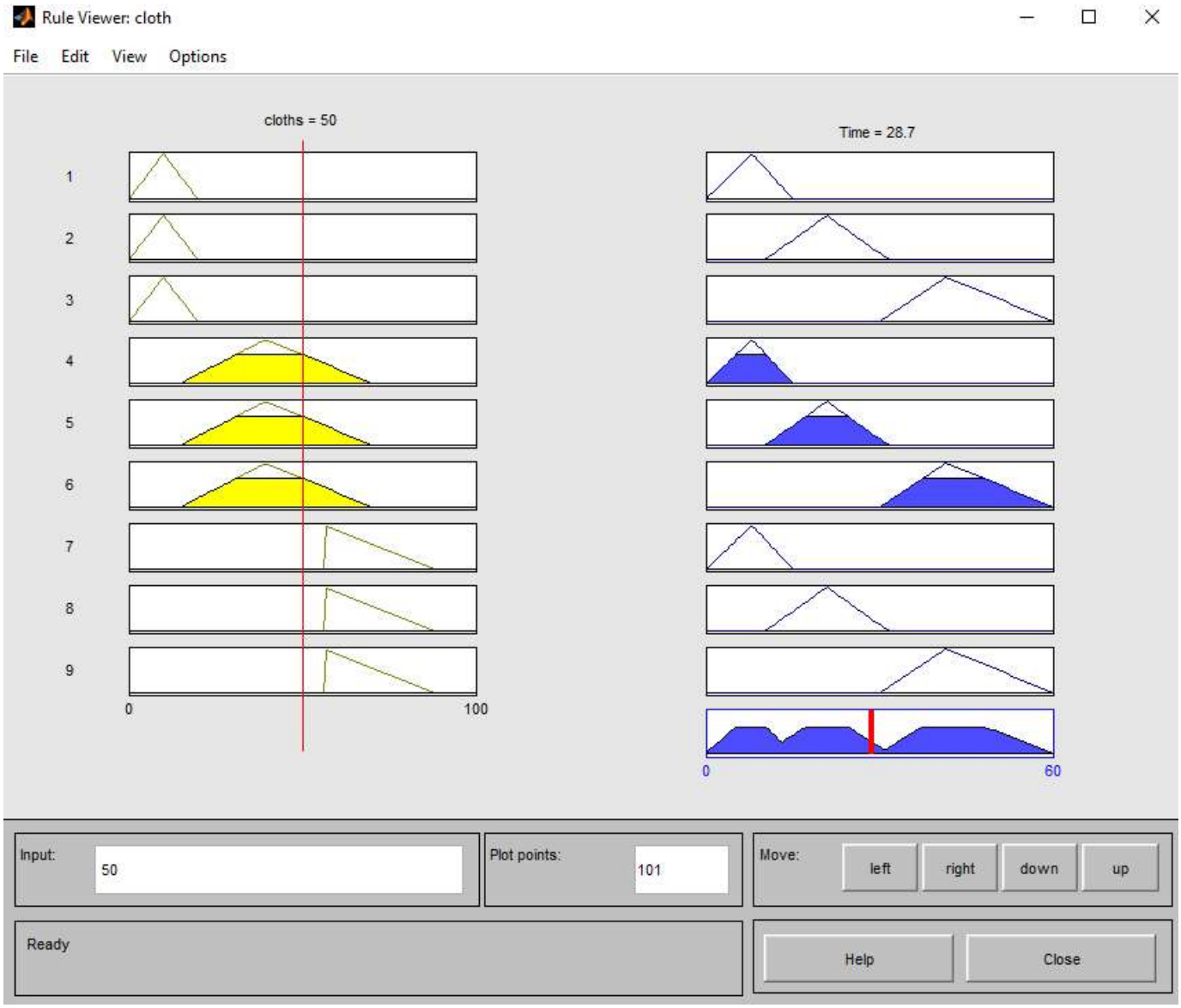


Fig. 5

Conclusion :-

Thus, FIS with Mamdani Inferencing mechanism is implemented successfully.

Experiment No. 07

Theory:

Defuzzification:

Defuzzification is the process of conversion of a fuzzy quantity into a precise quantity. The output of a fuzzy set process may be union of two or more fuzzy membership functions defined on the universe of discourse of the output variable.

The inverse of fuzzification. The former one was used to convert the crisp results into fuzzy results but here the mapping is done to convert the fuzzy results into crisp results. This process is capable of generating a non-fuzzy control action which illustrates the possibility distribution of an inferred fuzzy control action. Defuzzification process can also be treated as the rounding off process, where fuzzy set having a group of membership values on the unit interval reduced to a single scalar quantity.

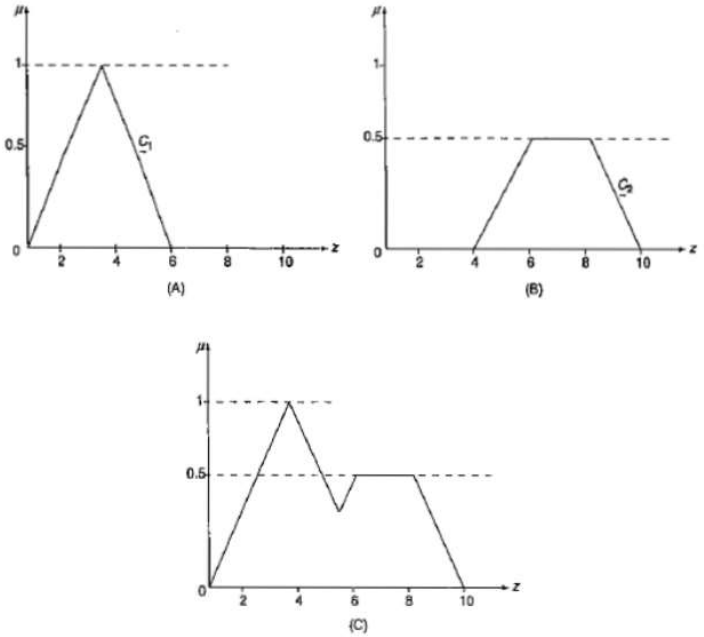


Figure 4.7: (A) First part of fuzzy output, (B) second part of fuzzy output (C) union of parts (A) and (B)

Defuzzification methods include the following:

1. Max membership principle
2. Centroid method
3. Weighted average method
4. Mean-max membership
5. Center of sums
6. Center of largest area
7. First of maxima, last of maxima

Max – Membership principle:

This method is limited to peak output functions and also known as height method. Mathematically it can be represented as follows

$$\mu_{\tilde{A}}(x^*) > \mu_{\tilde{A}}(x) \ \ for\ all\ x \in X$$

Here, x^* is the defuzzified output.

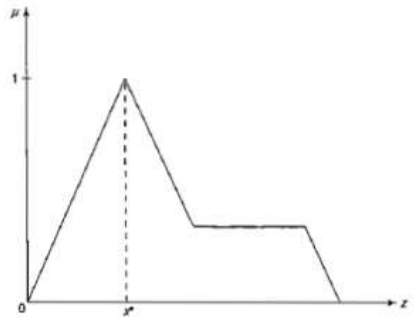


Figure 4.8: Max-membership defuzzification method

Centroid Method

This method is also known as the center of area or the center of gravity method. Mathematically, the defuzzified output $x*$ will be represented as

$$x^* = \frac{\int \mu_{\tilde{A}}(x) \,.\, x dx}{\int \mu_{\tilde{A}}(x) \,.\, dx}$$

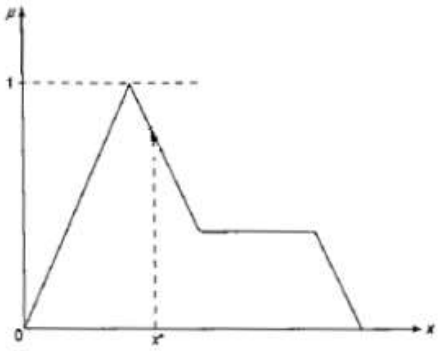


Figure 4.9: Centroid defuzzification method

Weighted Average Method

In this method, each membership function is weighted by its maximum membership value. Mathematically, the defuzzified output $x*$ will be represented as

$$x^* = \frac{\sum \mu_{\tilde{A}}(\overline{x_i}) \cdot \overline{x_i}}{\sum \mu_{\tilde{A}}(\overline{x_i})}$$

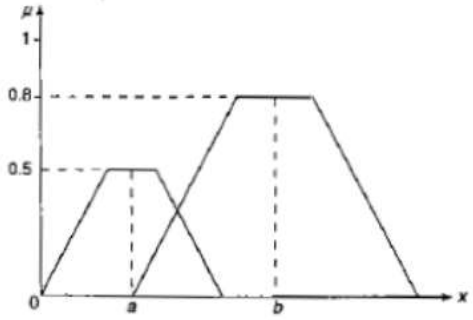
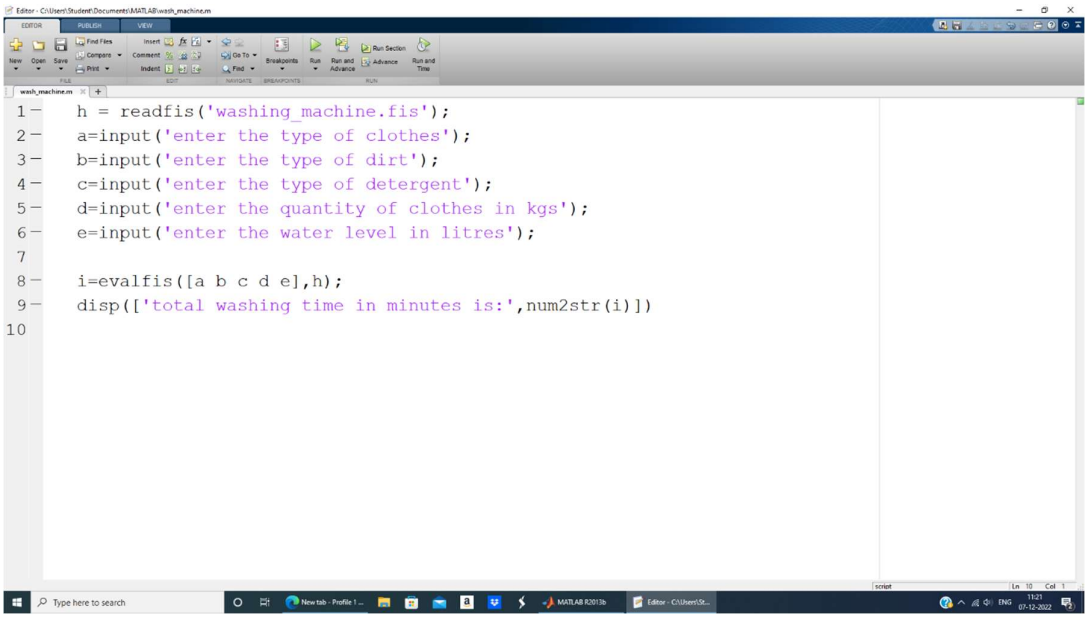
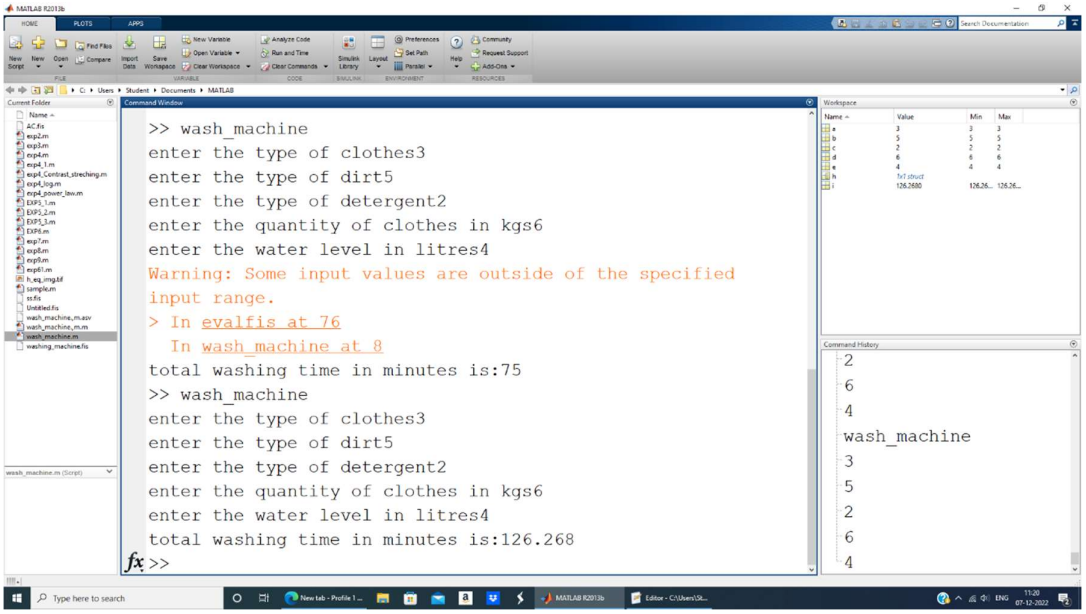


Figure 4.10: Weighted average defuzzification method (two symmetrical functions)

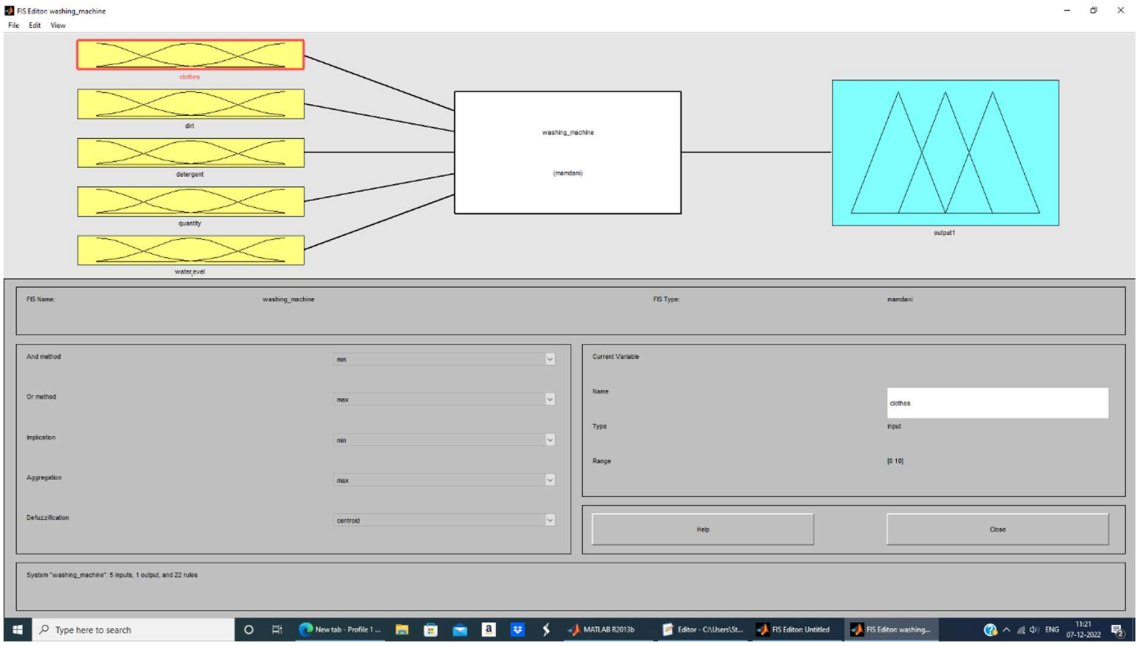
Implementation :-

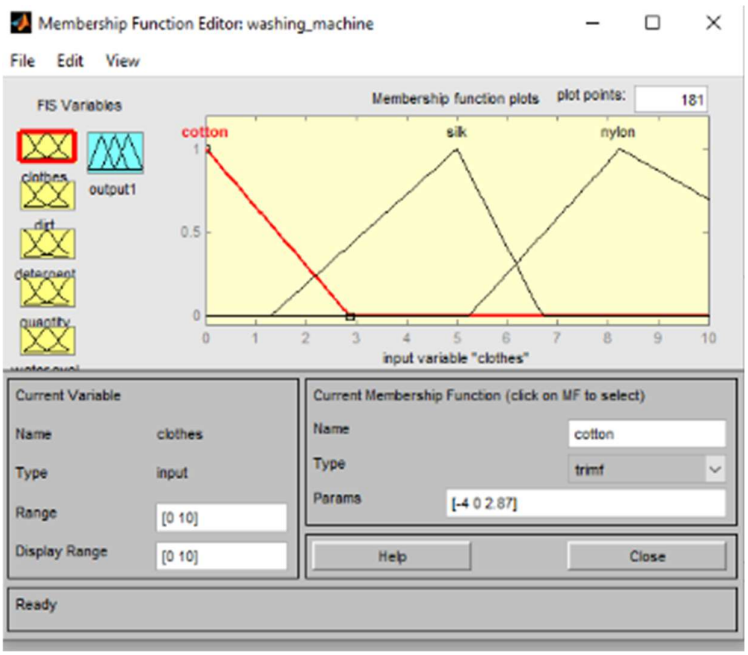


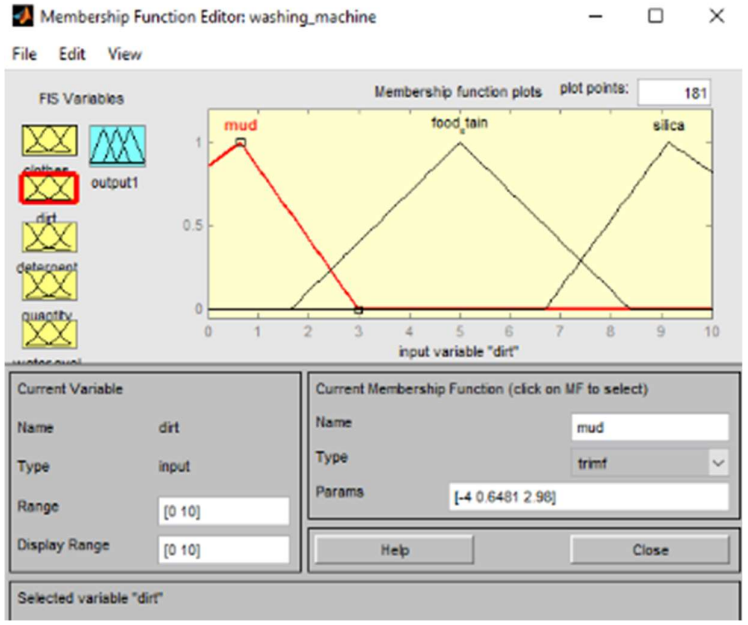
```
h = readfis('washing_machine.fis');
a=input('enter the type of clothes');
b=input('enter the type of dirt');
c=input('enter the type of detergent');
d=input('enter the quantity of clothes in kgs');
e=input('enter the water level in litres');

i=evalfis([a b c d e],h);
disp(['total washing time in minutes is:',num2str(i)])
```

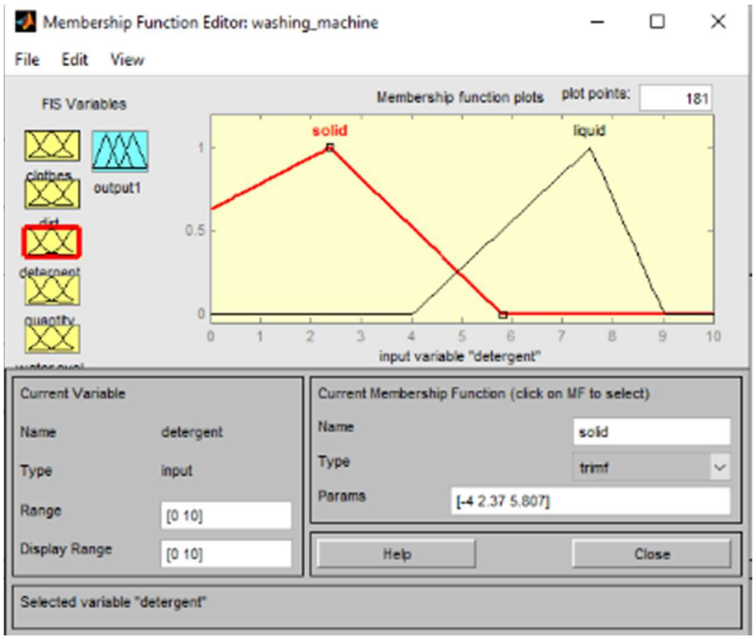


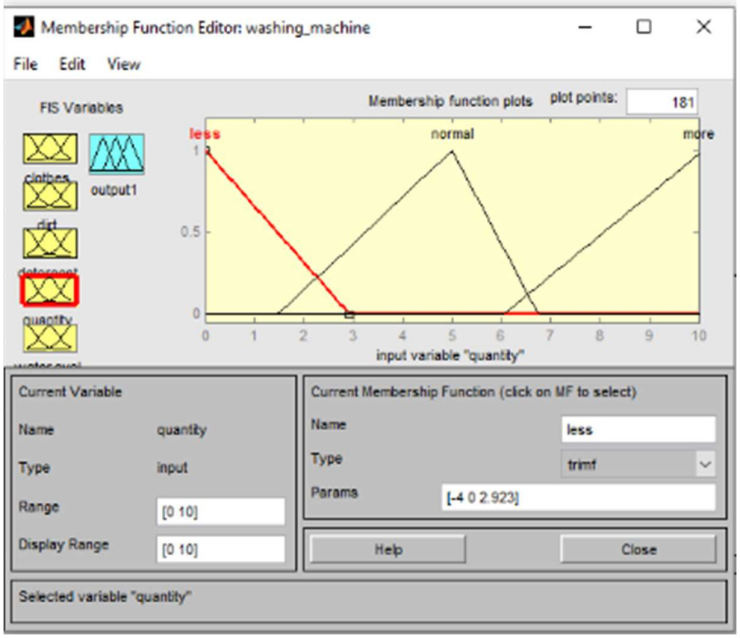
```
>> wash_machine
enter the type of clothes3
enter the type of dirt5
enter the type of detergent2
enter the quantity of clothes in kgs6
enter the water level in litres4
Warning: Some input values are outside of the specified
input range.
> In evalfis at 76
  In wash_machine at 8
total washing time in minutes is:75
>> wash_machine
enter the type of clothes3
enter the type of dirt5
enter the type of detergent2
enter the quantity of clothes in kgs6
enter the water level in litres4
total washing time in minutes is:126.268
fx >>
```

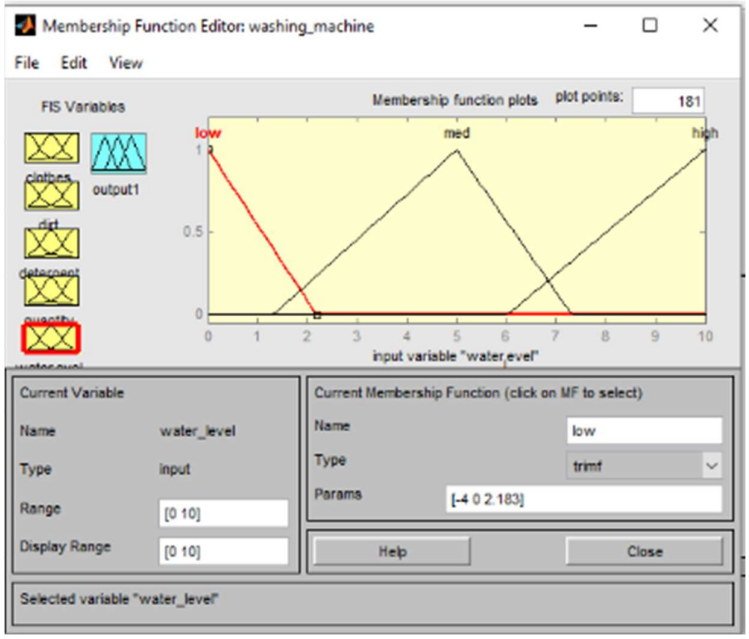

FIS Editor: washing_machine
File Edit View
clothes
dirt
detergent
quantity
water_evel
washing_machine
(mamdani)
output1
FIS Name:
washing_machine
FIS Type:
mamdani
And method
min
Or method
max
Implication
min
Aggregation
max
Defuzzification
centroid
Current Variable
Name
clothes
Type
input
Range
[0 10]
Help
Close
System "washing_machine": 5 inputs, 1 output, and 22 rules
Type here to search

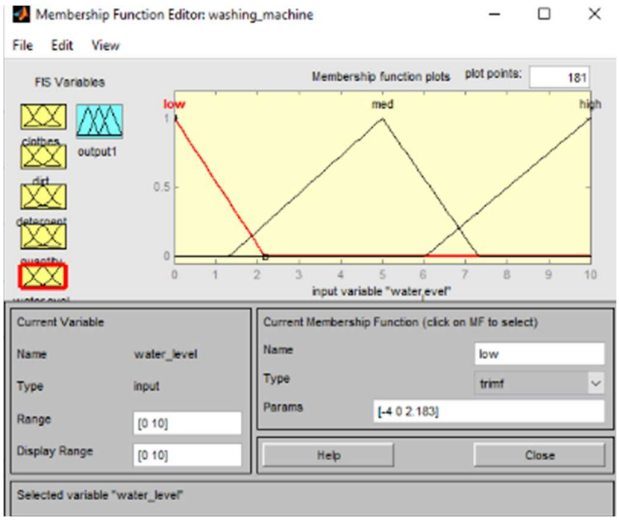

Membership Function Editor: washing_machine
File Edit View
FIS Variables
clothes
output1
dirt
detergent
quantity
Membership function plots
plot points:
181
cotton
silk
nylon
input variable "clothes"
Current Variable
Name
clothes
Type
input
Range
[0 10]
Display Range
[0 10]
Current Membership Function (click on MF to select)
Name
cotton
Type
trimf
Params
[-4 0 2.87]
Help
Close
Ready

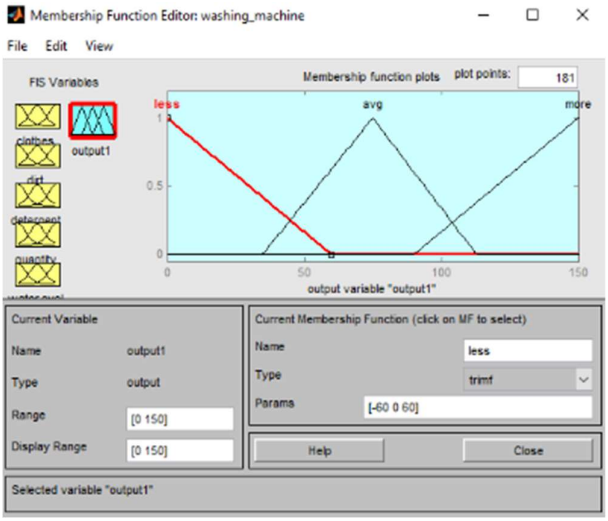

Membership Function Editor: washing_machine
File Edit View
FIS Variables
clothes
output1
dirt
detergent
quantity
Membership function plots
plot points:
181
mud
food_tain
silca
input variable "dirt"
Current Variable
Name
dirt
Type
input
Range
[0 10]
Display Range
[0 10]
Current Membership Function (click on MF to select)
Name
mud
Type
trimf
Params
[-4 0.6481 2.98]
Help
Close
Selected variable "dirt"

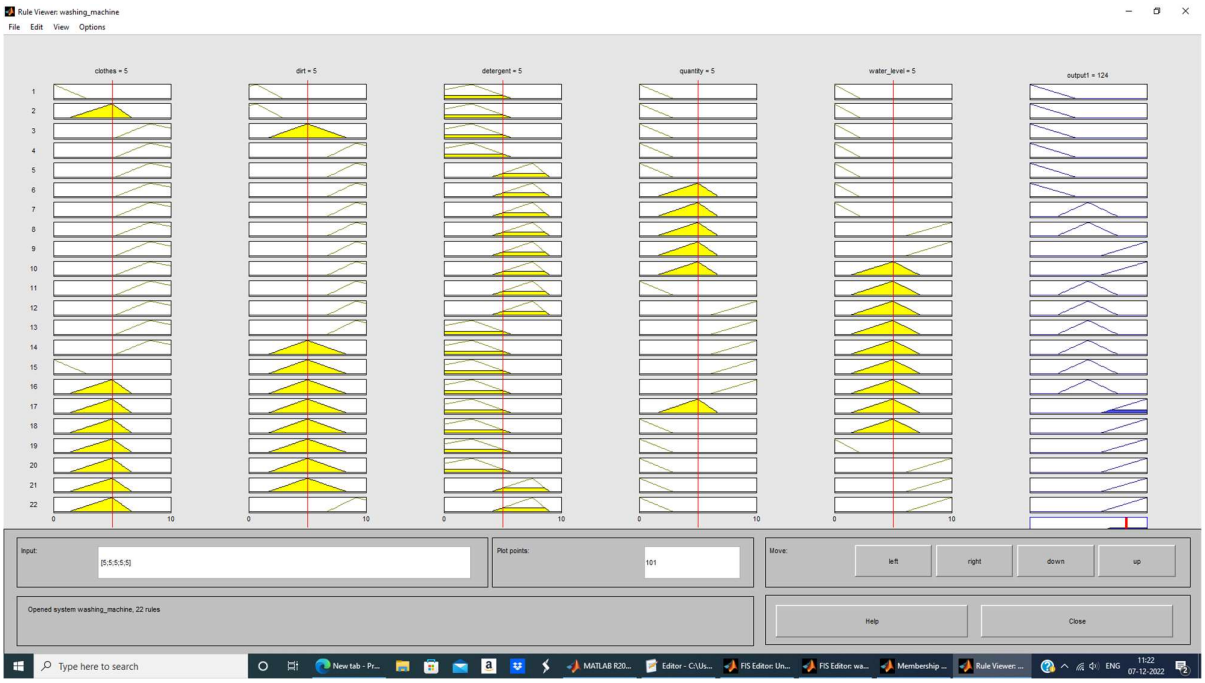

Membership Function Editor: washing_machine
File Edit View
FIS Variables
clothes
output1
dirt
detergent
quantity
Membership function plots
plot points:
181
solid
liquid
input variable "detergent"
Current Variable
Name detergent
Type input
Range [0 10]
Display Range [0 10]
Current Membership Function (click on MF to select)
Name solid
Type trimf
Params [-4 2.37 5.807]
Help
Close
Selected variable "detergent"

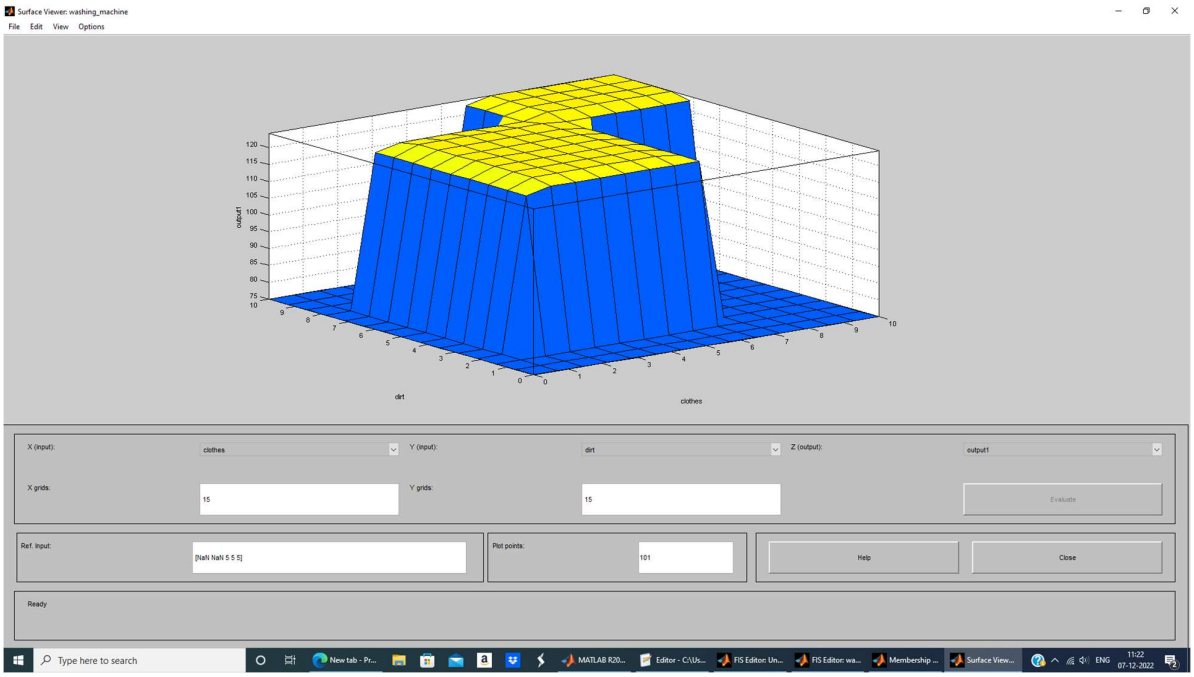

Membership Function Editor: washing_machine
File Edit View
FIS Variables
clothes
output1
dirt
detergent
quantity
Membership function plots
plot points:
181
less
normal
more
input variable "quantity"
Current Variable
Name quantity
Type input
Range [0 10]
Display Range [0 10]
Current Membership Function (click on MF to select)
Name less
Type trimf
Params [-4 0 2.923]
Help
Close
Selected variable "quantity"


Membership Function Editor: washing_machine
File Edit View
FIS Variables
clothes
output1
dirt
detergent
quantity
Membership function plots
plot points:
181
low
med
high
input variable "water_level"
Current Variable
Name water_level
Type input
Range [0 10]
Display Range [0 10]
Current Membership Function (click on MF to select)
Name low
Type trimf
Params [-4 0 2.183]
Help
Close
Selected variable "water_level"









Conclusion :- Thus , defuzzification is implemented successfully