DATA: 31/03/2022 EXP No: 06	**Title of the Lab** Implementations of Fuzzy Logic and Dempster Shafer theory for representing Uncertainty in Knowledge	**Name:Avinash reddy Vasipalli** **Registration Number: RA1911027010007** **Section: N1 Lab Batch: 1** **Day Order: 2**

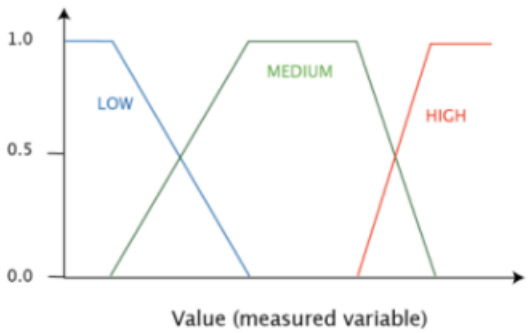
AIM: Implementations of Fuzzy Logic and Dempster Shafer theory for representing Uncertainty in Knowledge

Algorithm:

1. Define Non-Fuzzy Inputs with Fuzzy Sets. The non-fuzzy inputs are numbers from a certain range, and find how to represent those non-fuzzy values with fuzzy sets.
2. Locate the input, output, and state variables of the plane under consideration.
3. Split the complete universe of discourse spanned by each variable into a number of fuzzy subsets, assigning each with a linguistic label. The subsets include all the elements in the universe.
4. Obtain the membership function for each fuzzy subset.
5. Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and output of fuzzy subsets on the other side, thereby forming the rule base.
6. Choose appropriate scaling factors for the input and output variables for normalizing the variables between [0, 1] and [-1, I] interval.
7. Carry out the fuzzification process.
8. Identify the output contributed from each rule using fuzzy approximate reasoning.
9. Combine the fuzzy outputs obtained from each rule.
10. Finally, apply defuzzification to form a crisp output.

Optimization Technique:

Various numerical optimization techniques can be used such as dynamic programming, Lagrangian relaxation method, mixed integer programming, and branch-and-bound method. The dynamic programming method is simple but the calculation time required to converge to the optimal solution is quite long. Regarding the branch-and-bound method, it adopts a linear function to represent the fuel and start-up costs during a time horizon. The mixed integer programming uses linear programming to attain optimal solutions. Nevertheless, this method was applied to small problems of unit commitment and they required major assumptions that limit the margin of solutions. For the Lagrangian relaxation method, we note that the convergence time is an advantage, but the obtained solution is not ideal because of the complexity of the problem especially when the optimization problem contains a great number of production units.



Source Code

Code 1

```
import math

import cv2
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import display, Markdown
from glob2 import glob

PATH = '/content/enishtein.png'
def G(x, mean, std):
   return np.exp(-0.5*np.square((x-mean)/std))

def ExtremelyDark(x, M):
   return G(x, -50, M/6)

def VeryDark(x, M):
   return G(x, 0, M/6)

def Dark(x, M):
   return G(x, M/2, M/6)

def SlightlyDark(x, M):
   return G(x, 5*M/6, M/6)

def SlightlyBright(x, M):
   return G(x, M+(255-M)/6, (255-M)/6)

def Bright(x, M):
   return G(x, M+(255-M)/2, (255-M)/6)

def VeryBright(x, M):
   return G(x, 255, (255-M)/6)

def ExtremelyBright(x, M):
   return G(x, 305, (255-M)/6)
for M in (128, 64, 192):
   x = np.arange(-50, 306)

   ED = ExtremelyDark(x, M)
   VD = VeryDark(x, M)
   Da = Dark(x, M)
   SD = SlightlyDark(x, M)
   SB = SlightlyBright(x, M)
   Br = Bright(x, M)
   VB = VeryBright(x, M)
   EB = ExtremelyBright(x, M)

   plt.figure(figsize=(20,5))
   plt.plot(x, ED, 'k-.',label='ED', linewidth=1)
   plt.plot(x, VD, 'k-',label='VD', linewidth=2)
   plt.plot(x, Da, 'g-',label='Da', linewidth=2)
   plt.plot(x, SD, 'b-',label='SD', linewidth=2)
   plt.plot(x, SB, 'r-',label='SB', linewidth=2)
```

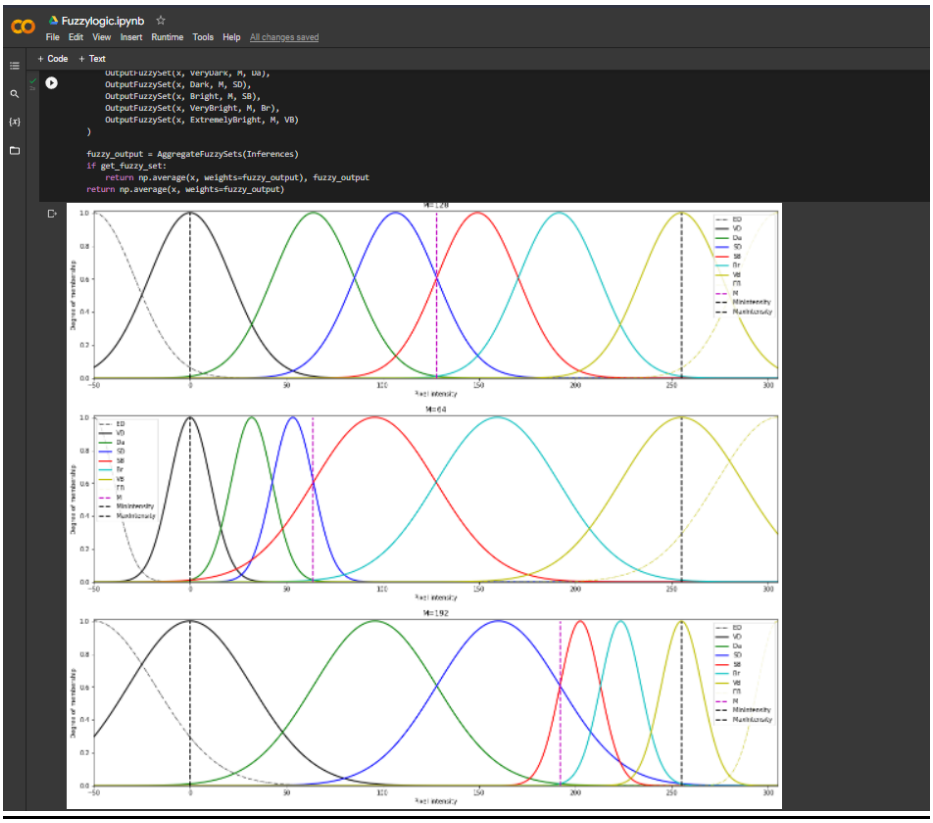
```
    plt.plot(x, Br, 'c-',label='Br', linewidth=2)
    plt.plot(x, VB, 'y-',label='VB', linewidth=2)
    plt.plot(x, EB, 'y-.',label='EB', linewidth=1)
    plt.plot((M, M), (0, 1), 'm--', label='M', linewidth=2)
    plt.plot((0, 0), (0, 1), 'k--', label='MinIntensity', linewidth=2)
    plt.plot((255, 255), (0, 1), 'k--', label='MaxIntensity', linewidth=2)
    plt.legend()
    plt.xlim(-50, 305)
    plt.ylim(0.0, 1.01)
    plt.xlabel('Pixel intensity')
    plt.ylabel('Degree of membership')
    plt.title(f'M={M}')
    plt.show()
def OutputFuzzySet(x, f, M, thres):
    x = np.array(x)
    result = f(x, M)
    result[result > thres] = thres
    return result

def AggregateFuzzySets(fuzzy_sets):
    return np.max(np.stack(fuzzy_sets), axis=0)

def Infer(i, M, get_fuzzy_set=False):
    VD = VeryDark(i, M)
    Da = Dark(i, M)
    SD = SlightlyDark(i, M)
    SB = SlightlyBright(i, M)
    Br = Bright(i, M)
    VB = VeryBright(i, M)

    x = np.arange(-50, 306)
    Inferences = (
        OutputFuzzySet(x, ExtremelyDark, M, VD),
        OutputFuzzySet(x, VeryDark, M, Da),
        OutputFuzzySet(x, Dark, M, SD),
        OutputFuzzySet(x, Bright, M, SB),
        OutputFuzzySet(x, VeryBright, M, Br),
        OutputFuzzySet(x, ExtremelyBright, M, VB)
    )

    fuzzy_output = AggregateFuzzySets(Inferences)
    if get_fuzzy_set:
        return np.average(x, weights=fuzzy_output), fuzzy_output
    return np.average(x, weights=fuzzy_output)
```

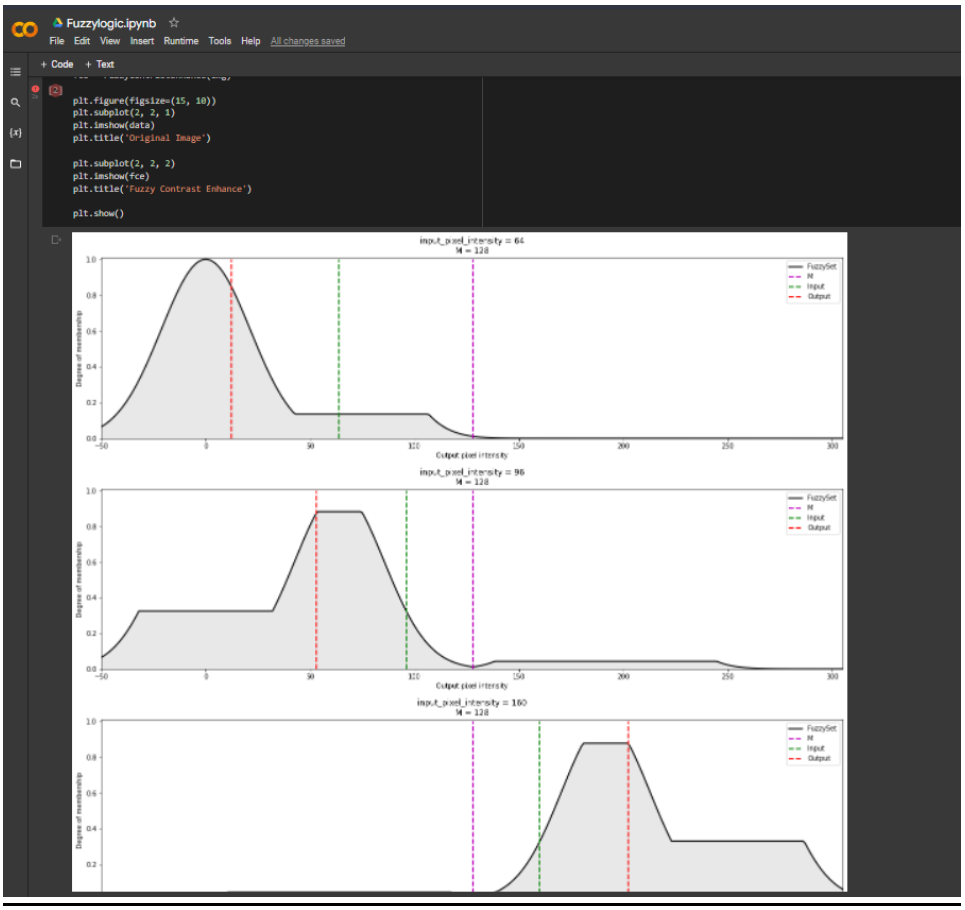
Code 2

```
for pixel in (64, 96, 160, 192):
    M = 128
    x = np.arange(-50, 306)
    centroid, output_fuzzy_set = Infer(np.array([pixel]), M, get_fuzzy_set=True)
    plt.figure(figsize=(20,5))
    plt.plot(x, output_fuzzy_set, 'k-',label='FuzzySet', linewidth=2)
    plt.plot((M, M), (0, 1), 'm--', label='M', linewidth=2)
    plt.plot((pixel, pixel), (0, 1), 'g--', label='Input', linewidth=2)
```

```
    plt.plot((centroid, centroid), (0, 1), 'r--', label='Output', linewidth=2)
    plt.fill_between(x, np.zeros(356), output_fuzzy_set, color=(.9, .9, .9, .9))
    plt.legend()
    plt.xlim(-50, 305)
    plt.ylim(0.0, 1.01)
    plt.xlabel('Output pixel intensity')
    plt.ylabel('Degree of membership')
    plt.title(f'input_pixel_intensity = {pixel}\nM = {M}')
    plt.show()
means = (64, 96, 128, 160, 192)
plt.figure(figsize=(25,5))
for i in range(len(means)):
    M = means[i]
    x = np.arange(256)
    %time y = np.array([Infer(np.array([i]), M) for i in x])
    plt.subplot(1, len(means), i+1)
    plt.plot(x, y, 'r-', label='IO mapping')
    plt.xlim(0, 256)
    plt.ylim(-50, 355)
    plt.xlabel('Input Intensity')
    plt.ylabel('Output Intensity')
    plt.title(f'M = {M}')
plt.show()
def FuzzyContrastEnhance(rgb):
    lab = cv2.cvtColor(rgb, cv2.COLOR_RGB2LAB)
    l = lab[:, :, 0]
    M = np.mean(l)
    if M < 128:
        M = 127 - (127 - M)/2
    else:
        M = 128 + M/2
    x = list(range(-50,306))
    FuzzyTransform = dict(zip(x,[Infer(np.array([i]), M) for i in x]))
    u, inv = np.unique(l, return_inverse = True)
    l = np.array([FuzzyTransform[i] for i in u])[inv].reshape(l.shape)
    Min = np.min(l)
    Max = np.max(l)
    lab[:, :, 0] = (l - Min)/(Max - Min) * 255
    return cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
data = cv2.cvtColor(cv2.imread(PATH), cv2.COLOR_BGR2RGB)
data.shape
img = data
fce = FuzzyContrastEnhance(img)

plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
plt.imshow(data)
plt.title('Original Image')

plt.subplot(2, 2, 2)
plt.imshow(fce)
plt.title('Fuzzy Contrast Enhance')

plt.show()
```

Output





Result:

Successfully implemented Fuzzy logic and Uncertain method for an application for Dempster Shafer Theory