Computer Assignment 2

CPE 261456 (Introduction to Computational Intelligence)

โดย

นายธนาคม หัสแดง

รหัสนักศึกษา 590610624

เสนอ

ผศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่

Sous Vide Fuzzy Logic Simulator

ลักษณะการทำงาน :

เริ่มต้นจากการกำหนด ข้อกำหนด ทั้ง 4 แบบ คือ

- Size โดยจะมี 3 ระดับ คือ
  - SMALL
  - MEDIUM
  - BIG
- Time โดยจะมี 3 ระดับ คือ
  - SHORT
  - MEDIUM
  - LONG
- Temperature โดยจะมี 4 ระดับ คือ
  - Moderate
  - WARM
  - Very Warm
  - Hot
  - Scorching
- Done โดยจะมี 4 ระดับ คือ
  - Rare
  - Medium Rare
  - Medium
  - Medium Well
  - Well Done

โดยเมื่อกำหนดข้อกำหนดแล้ว ก็จะนำ กฎที่ตั้งเข้าสู่ Fuzzy และ เริ่มคำนวณจาก Input โดยการที่จะเลือกตัวแปรที่มีค่า Input ตรงกับ กฎ นั้น ก็จะนำค่าสมาชิกที่ได้ มา และ หาค่าน้อยทีสุดของ แต่ละ กฎ และเมื่อ

ได้ครบทุกกฎแล้ว จะนำค่าของ ทุกกฎมาหา ค่ามากที่สุด จากทั้งหมด และเมื่อได้ค่ามากที่สุดแล้ว จึงไปทำการ defuzzification โดยการหา Centroid ก็จะได้ผลลัพธ์ออกมาอยู่ในช่วงของ คำตอบที่ต้องการ

Rule ที่ใช้:

กฎข้อที่	**If (ขนาด, Size)**	**If (เวลา, Time)**	**If (อุณหภูมิ, Temp)**	**Else (ความสุก, Done)**
1	SMALL	SHORT	Moderate	Rare
2	SMALL	SHORT	Warm	Medium Rare
3	SMALL	SHORT	Very Warm	Medium Rare
4	SMALL	SHORT	Hot	Medium
5	SMALL	SHORT	Scorching	Medium
6	SMALL	MEDUIM	Moderate	Medium Rare
7	SMALL	MEDUIM	Warm	Medium
8	SMALL	MEDUIM	Very Warm	Medium
9	SMALL	MEDUIM	Hot	Medium Well
10	SMALL	MEDUIM	Scorching	Medium Well
11	SMALL	LONG	Moderate	Medium
12	SMALL	LONG	Warm	Medium Well
13	SMALL	LONG	Very Warm	Medium Well
14	SMALL	LONG	Hot	Well Done
15	SMALL	LONG	Scorching	Well Done
16	MEDIUM	SHORT	Moderate	Rare
17	MEDIUM	SHORT	Warm	Rare
18	MEDIUM	SHORT	Very Warm	Medium Rare
19	MEDIUM	SHORT	Hot	Medium Rare
20	MEDIUM	SHORT	Scorching	Medium
21	MEDIUM	MEDUIM	Moderate	Medium Rare

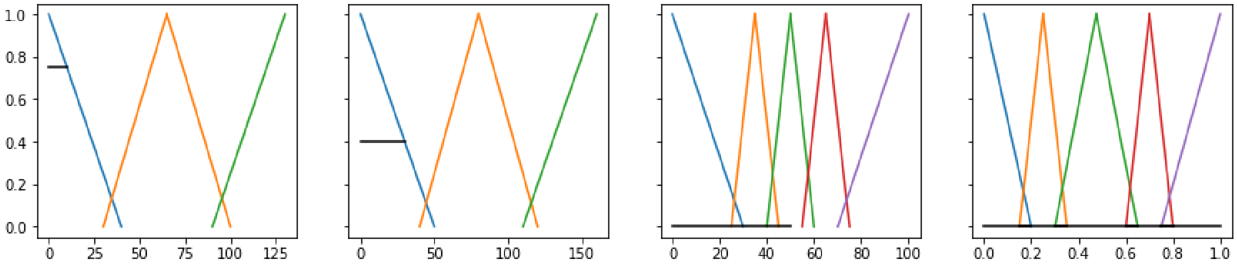
กฎข้อที่	If (ขนาด, Size)	If (เวลา, Time)	If (อุณหภูมิ, Temp)	Else (ความสุก, Done)
22	MEDIUM	MEDUIM	Warm	Medium Rare
23	MEDIUM	MEDUIM	Very Warm	Medium
24	MEDIUM	MEDUIM	Hot	Medium
25	MEDIUM	MEDUIM	Scorching	Medium Well
26	MEDIUM	LONG	Moderate	Medium
27	MEDIUM	LONG	Warm	Medium
28	MEDIUM	LONG	Very Warm	Medium Well
29	MEDIUM	LONG	Hot	Medium Well
30	MEDIUM	LONG	Scorching	Well Done
31	BIG	SHORT	Moderate	Rare
32	BIG	SHORT	Warm	Rare
33	BIG	SHORT	Very Warm	Rare
34	BIG	SHORT	Hot	Medium Rare
35	BIG	SHORT	Scorching	Medium Rare
36	BIG	MEDUIM	Moderate	Medium Rare
37	BIG	MEDUIM	Warm	Medium Rare
38	BIG	MEDUIM	Very Warm	Medium Rare
39	BIG	MEDUIM	Hot	Medium
40	BIG	MEDUIM	Scorching	Medium
41	BIG	LONG	Moderate	Medium
42	BIG	LONG	Warm	Medium
43	BIG	LONG	Very Warm	Medium
44	BIG	LONG	Hot	Medium Well
45	BIG	LONG	Scorching	Medium Well

Simulator

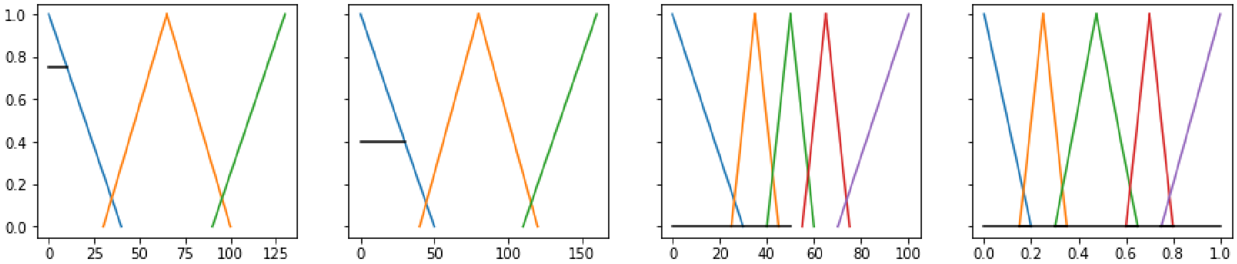
กำหนด Input คือ

- Size = 10
- Time = 30
- Temp = 50

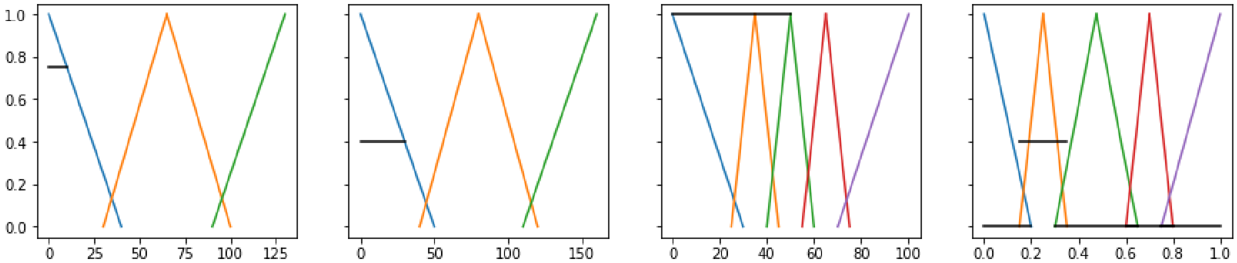
กฎข้อที่ 1 :



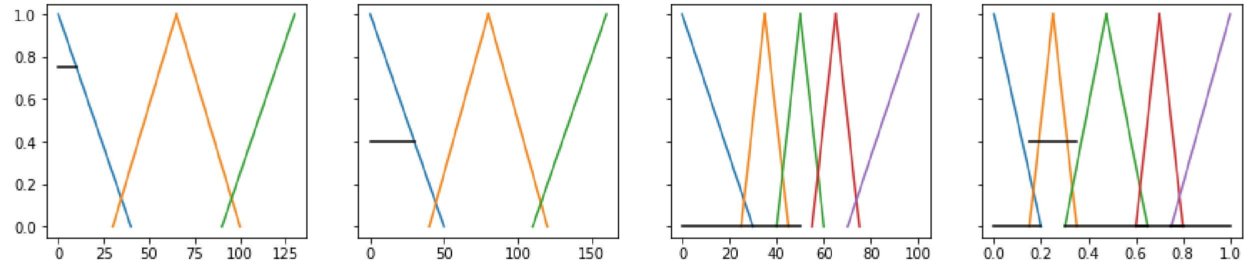
กฎข้อที่ 2 :



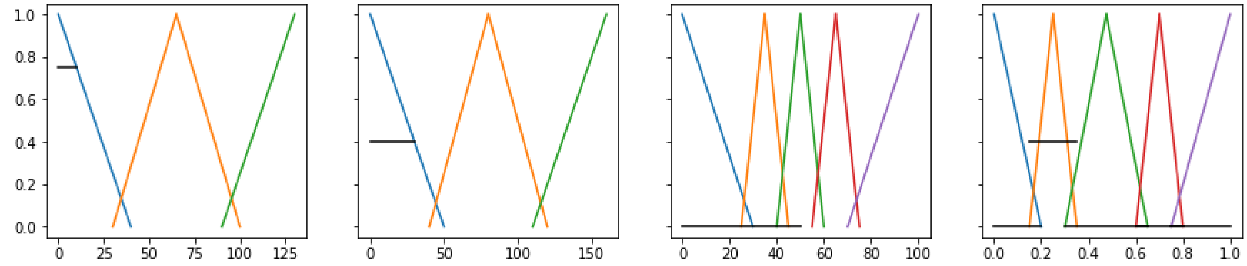
กฎข้อที่ 3 :



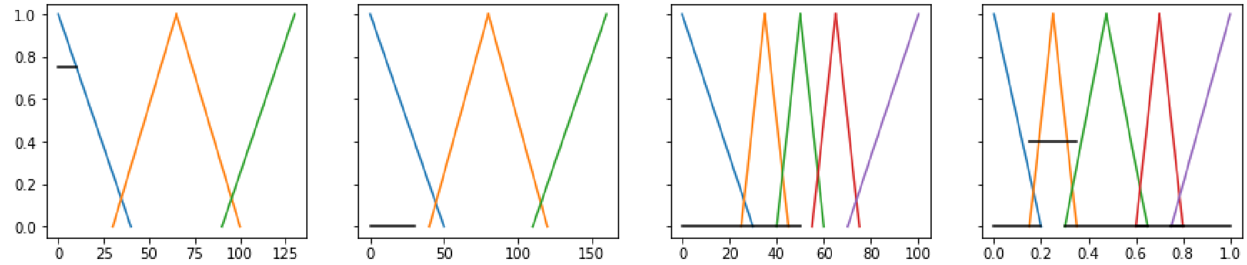
กฎข้อที่ 4 :



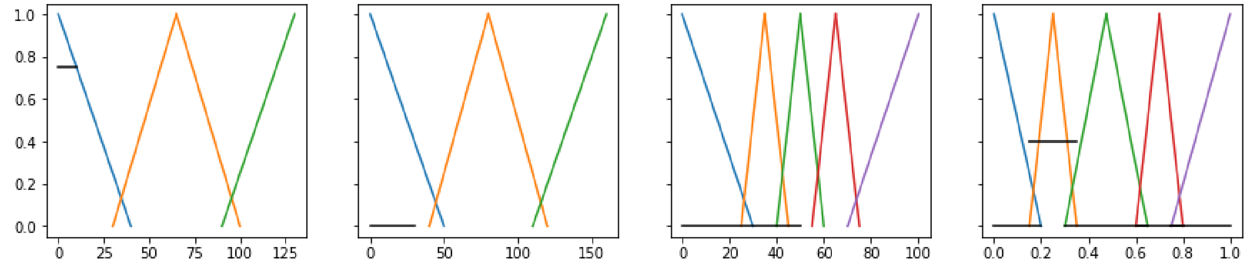
กฎข้อที่ 5 :



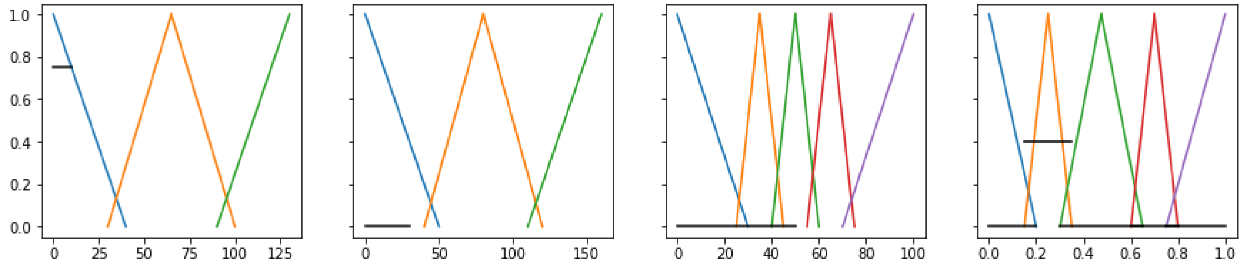
กฎข้อที่ 6 :



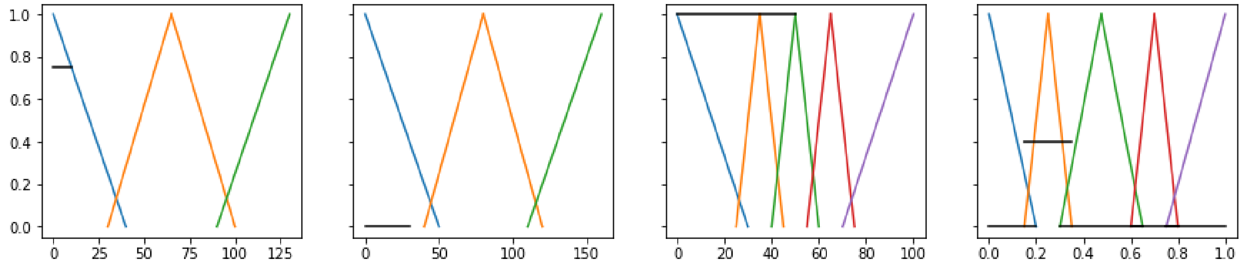
กฎข้อที่ 7 :



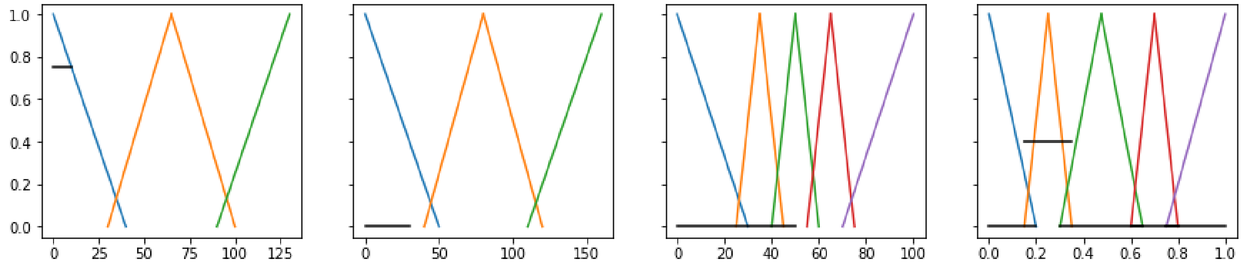
กฎข้อที่ 8 :



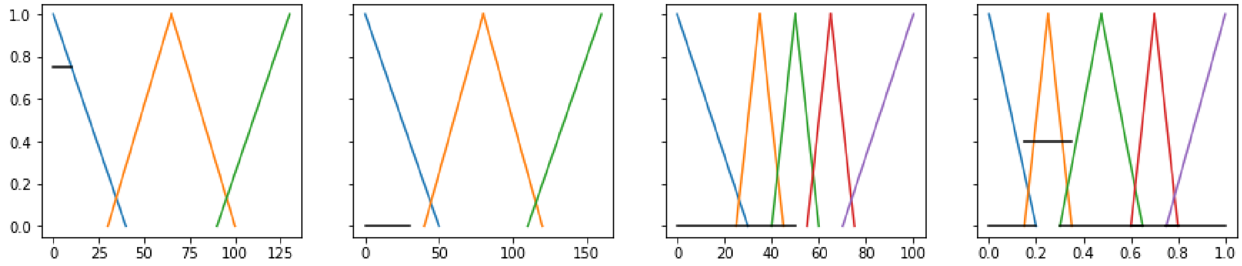
กฎข้อที่ 9 :



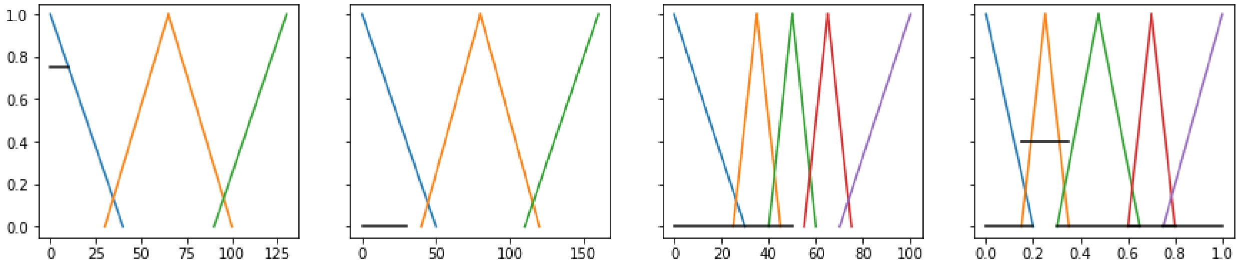
กฎข้อที่ 10 :



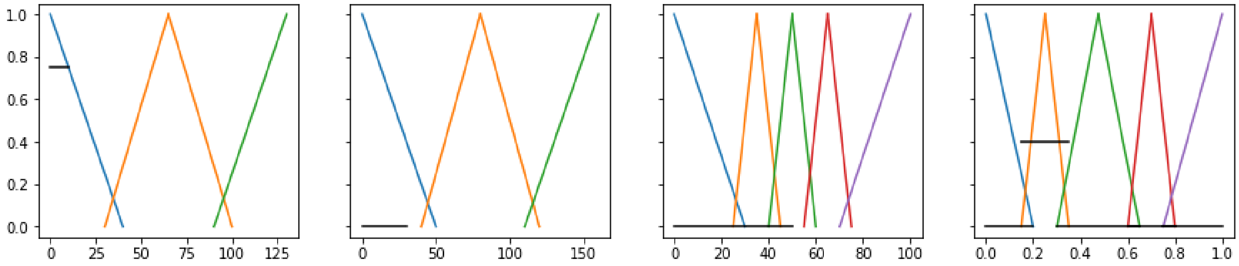
กฎข้อที่ 11 :



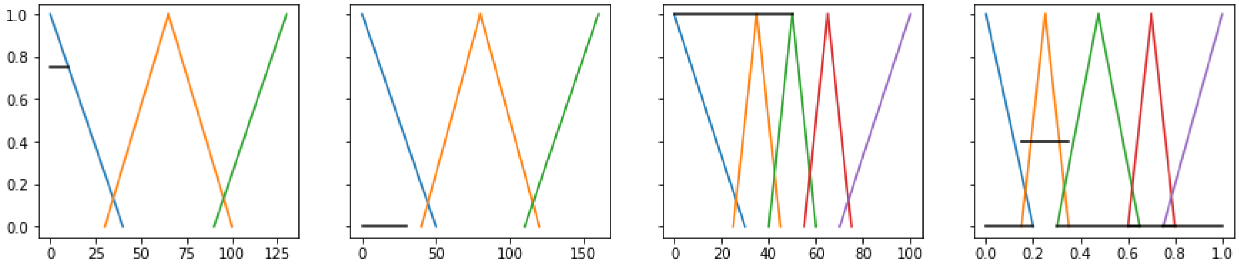
กฎข้อที่ 12 :



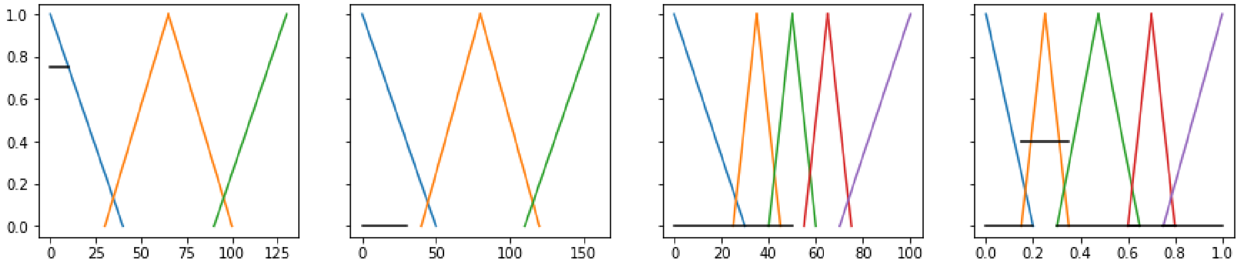
กฎข้อที่ 13 :



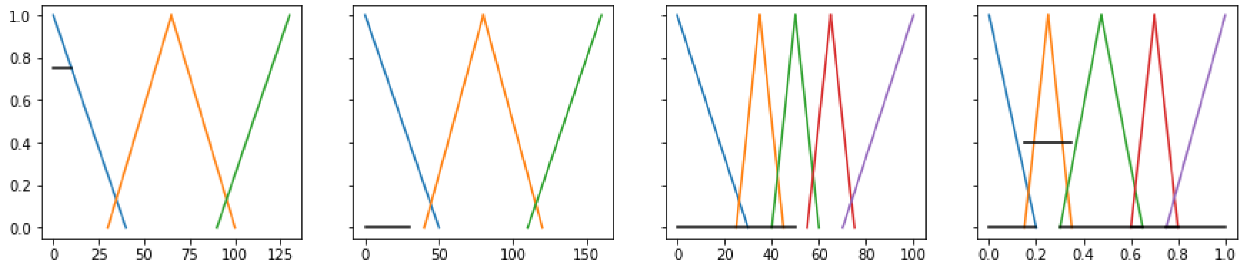
กฎข้อที่ 14 :



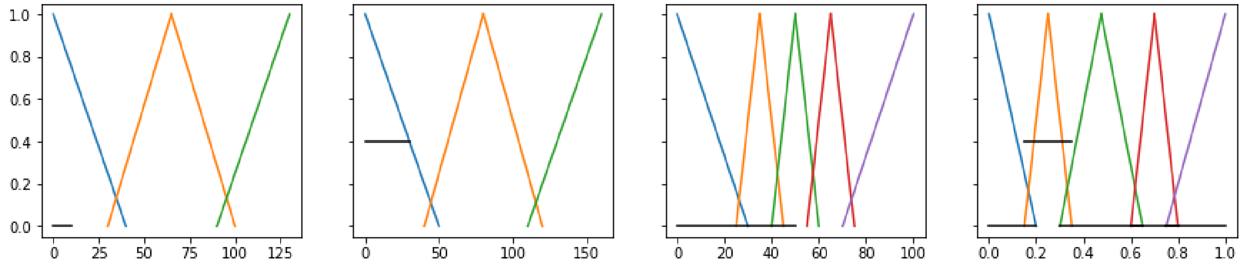
กฎข้อที่ 15 :



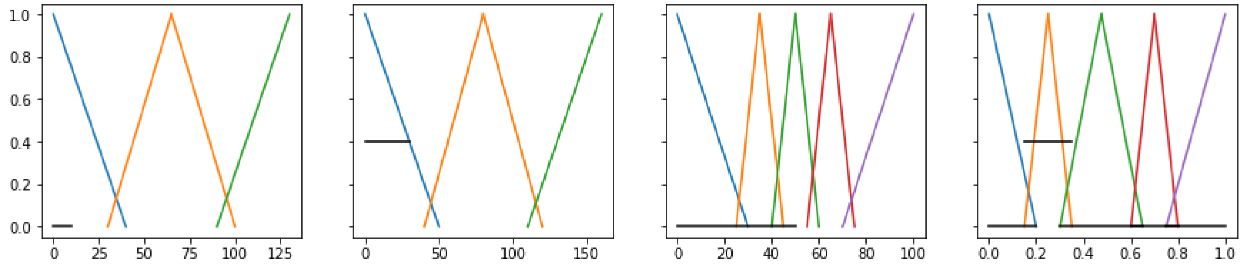
กฎข้อที่ 16 :



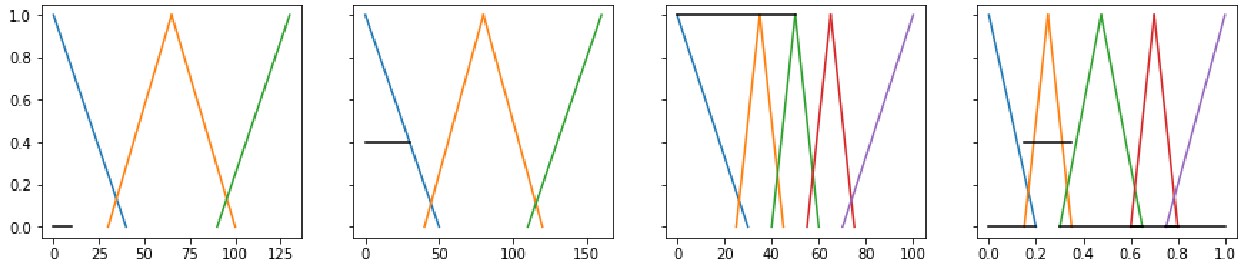
กฎข้อที่ 17 :



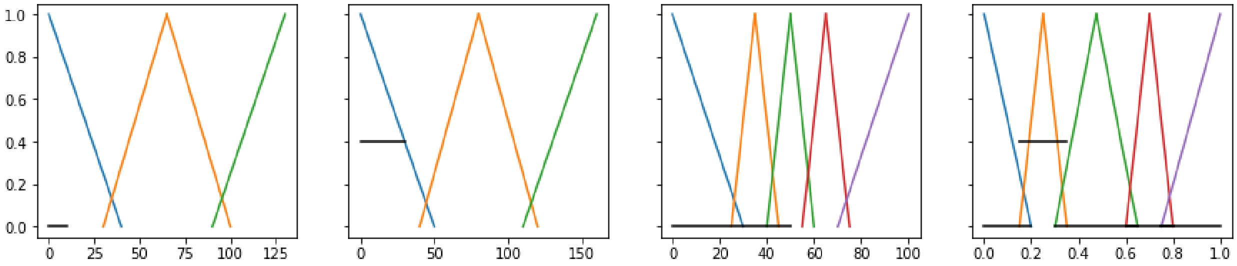
กฎข้อที่ 18 :



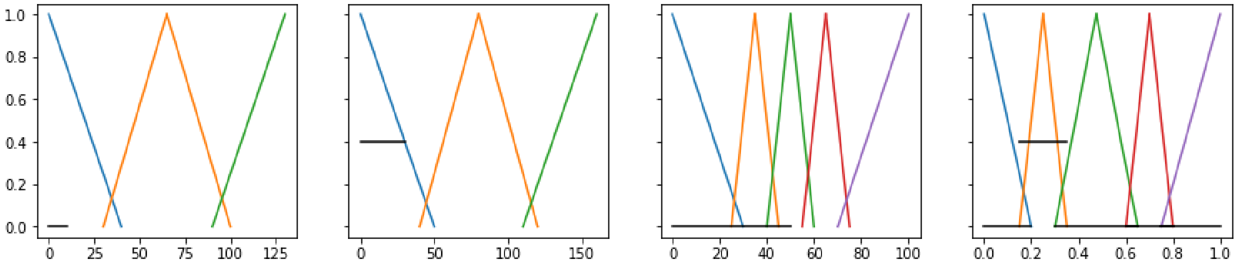
กฎข้อที่ 19 :



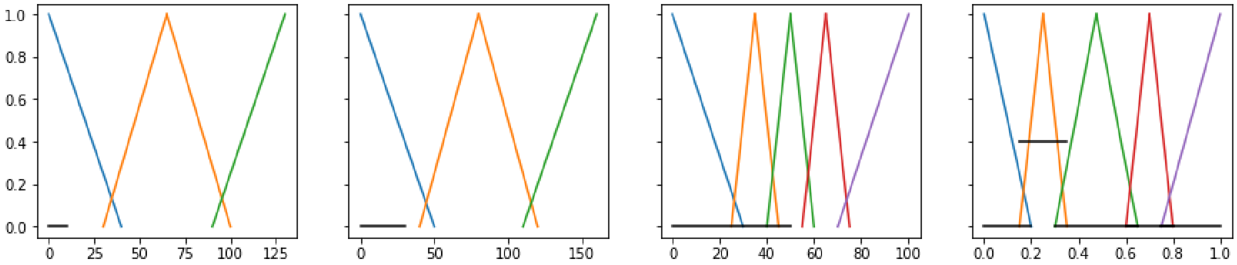
กฎข้อที่ 20 :



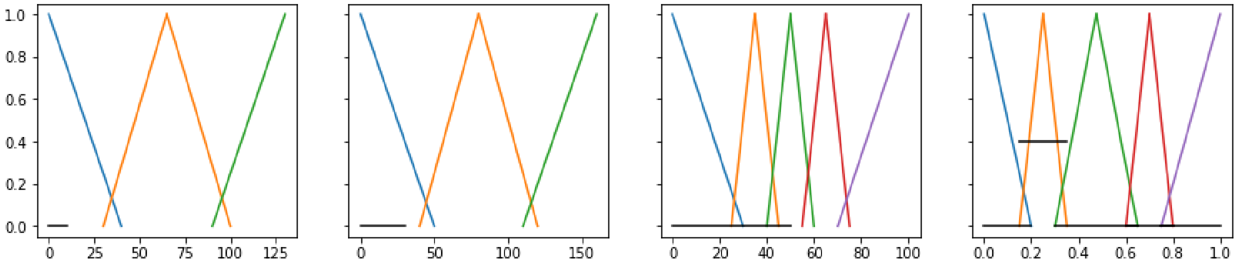
กฎข้อที่ 21 :



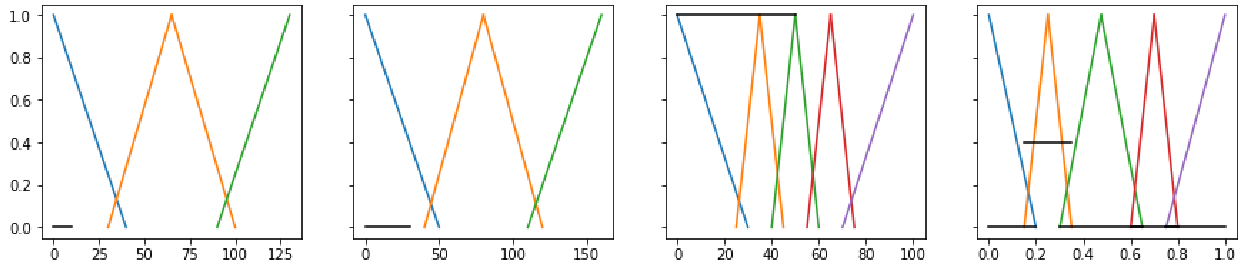
กฎข้อที่ 22 :



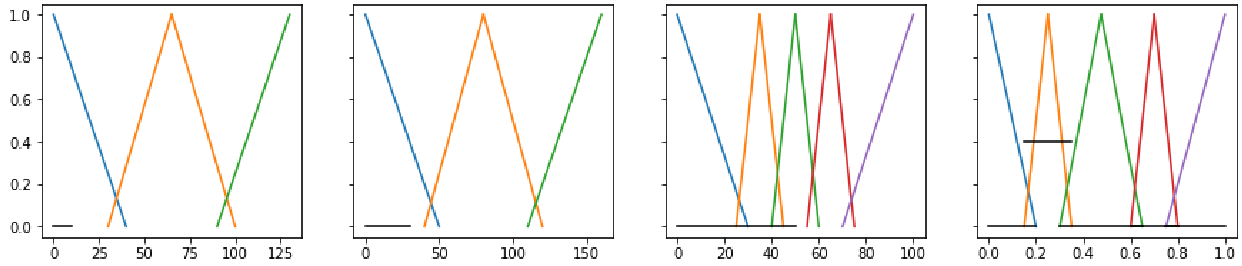
กฎข้อที่ 23 :



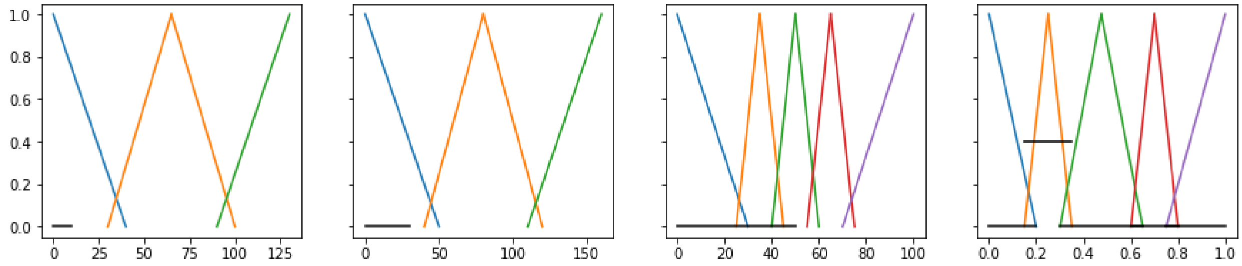
กฎข้อที่ 24 :



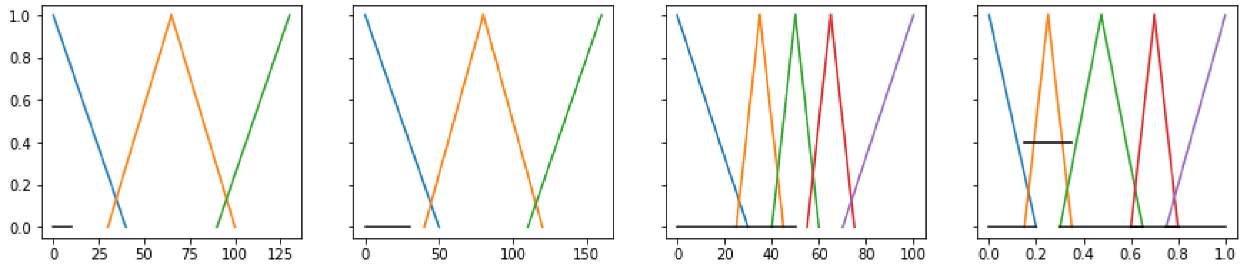
กฎข้อที่ 25 :



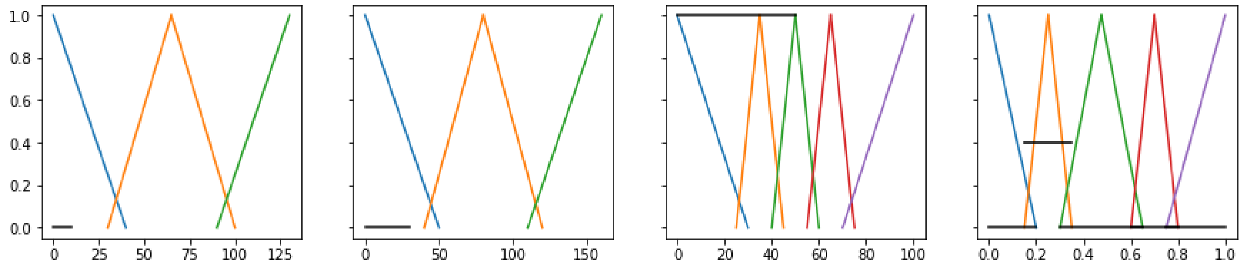
กฎข้อที่ 26 :



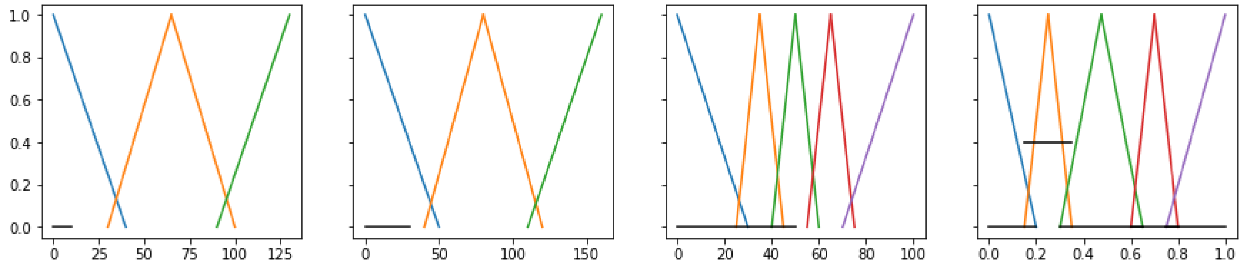
กฎข้อที่ 27 :



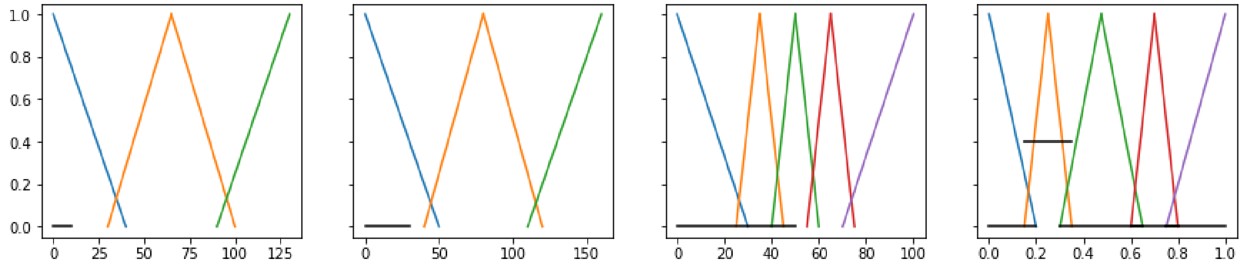
กฎข้อที่ 28 :



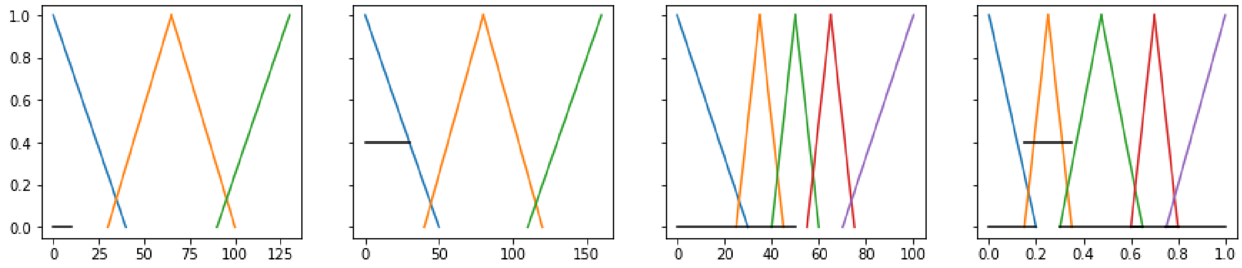
กฎข้อที่ 29 :



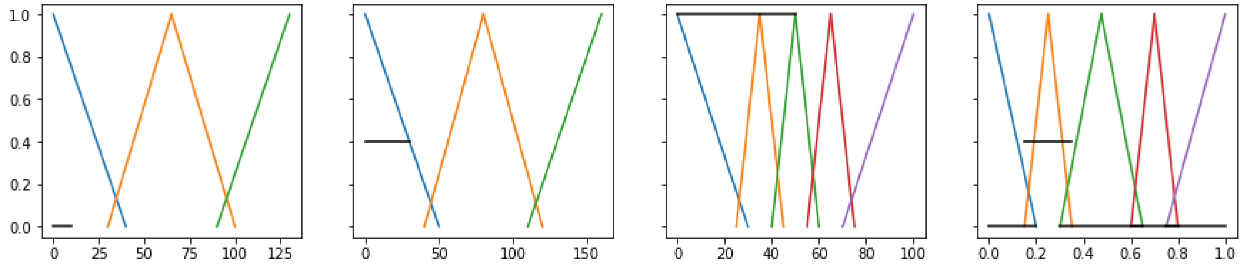
กฎข้อที่ 30 :



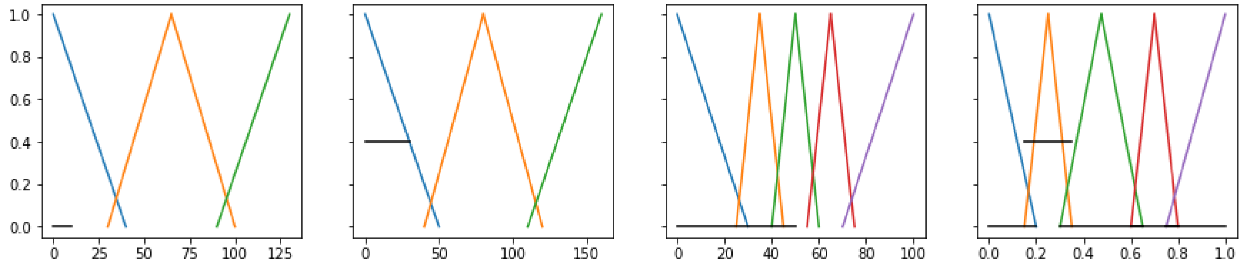
กฎข้อที่ 31 :



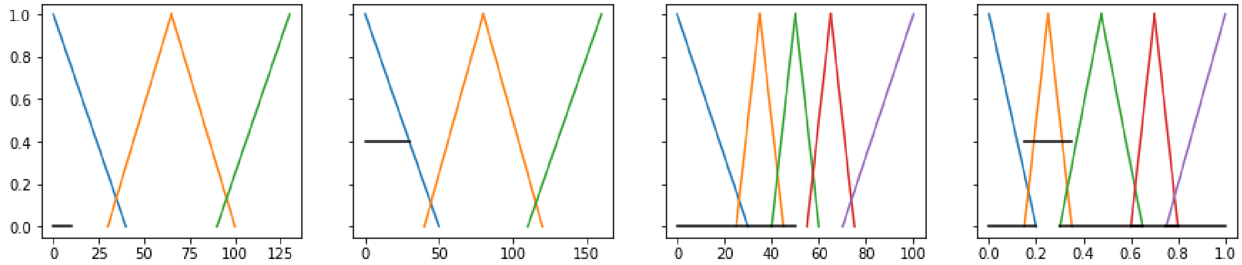
กฎข้อที่ 32 :



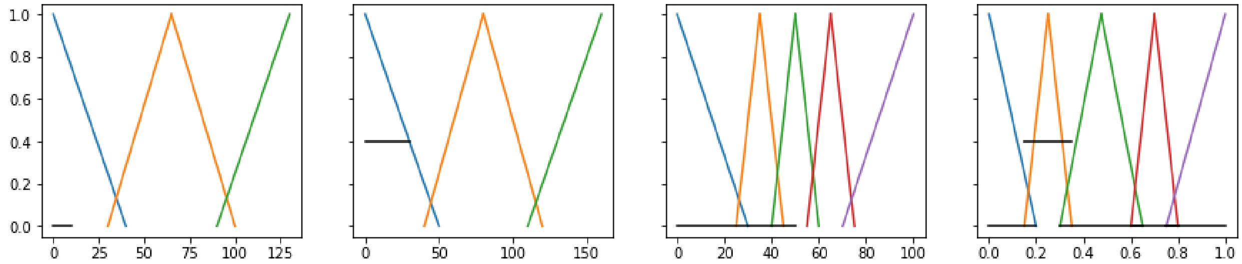
กฎข้อที่ 33 :



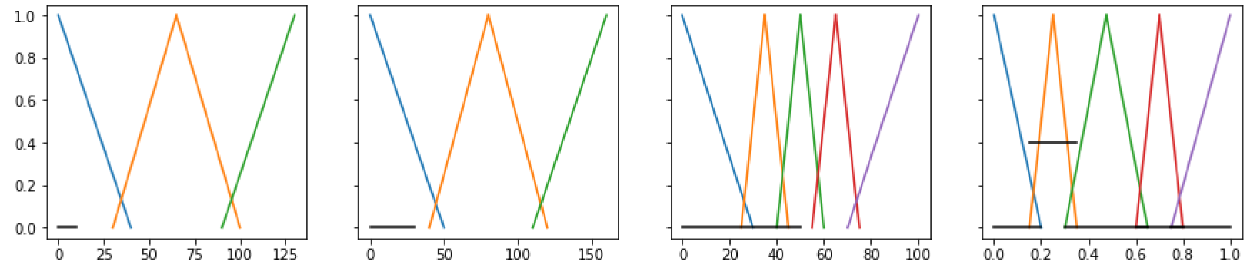
กฎข้อที่ 34 :



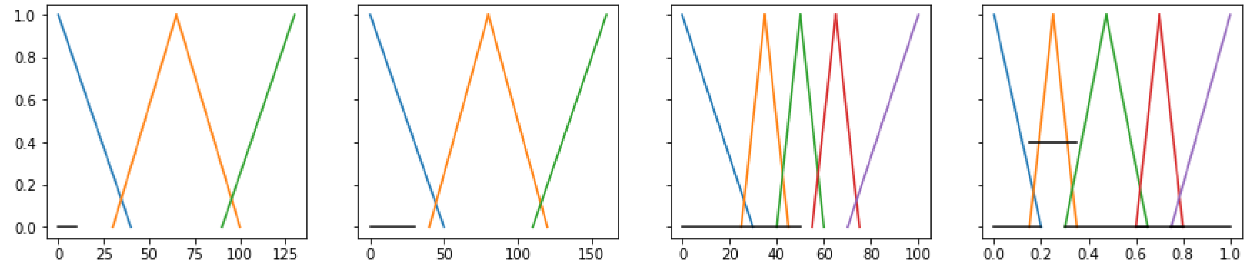
กฎข้อที่ 35 :



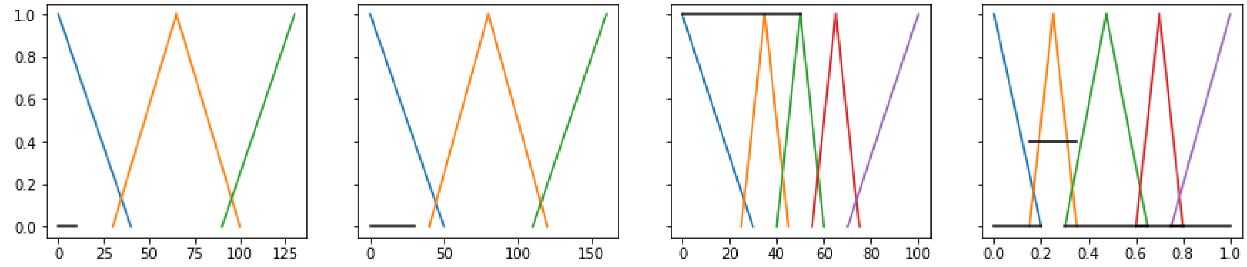
กฎข้อที่ 36 :



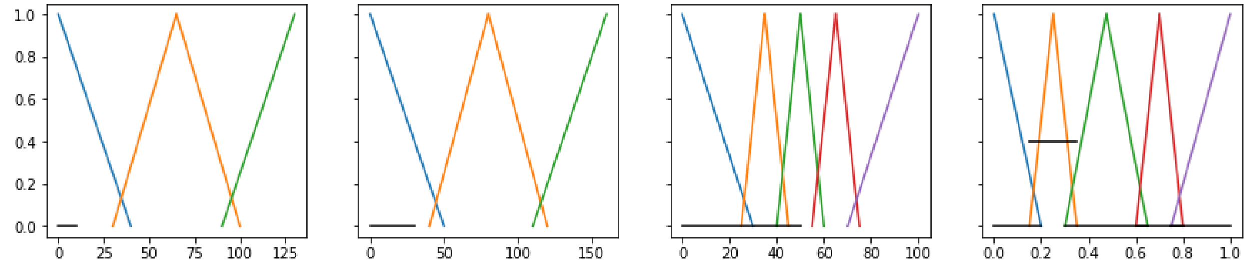
กฎข้อที่ 37 :



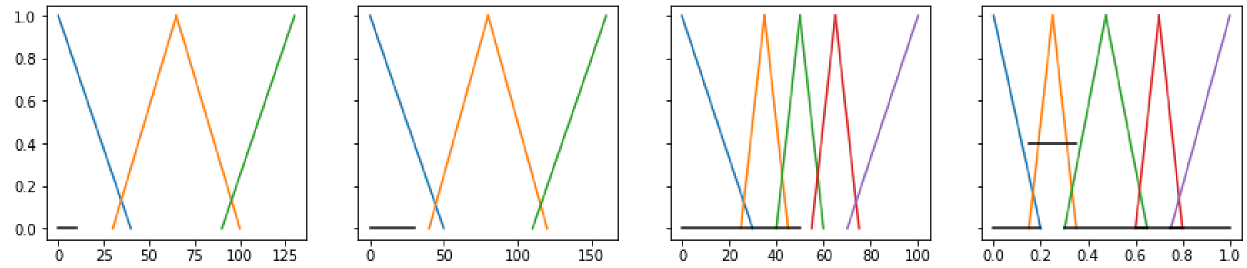
กฎข้อที่ 38 :



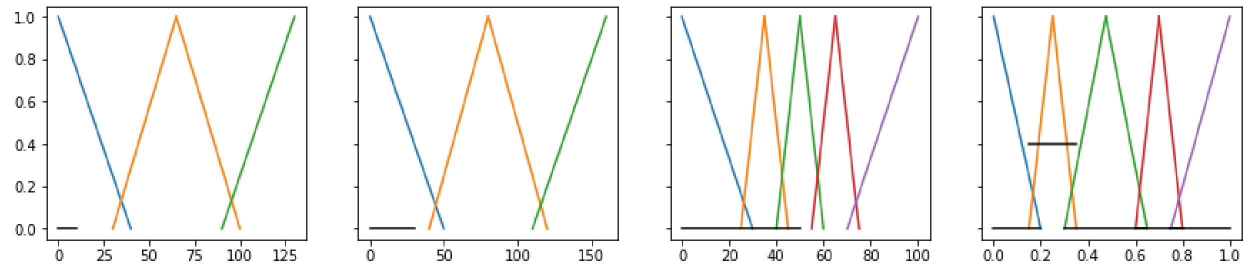
กฎข้อที่ 39 :



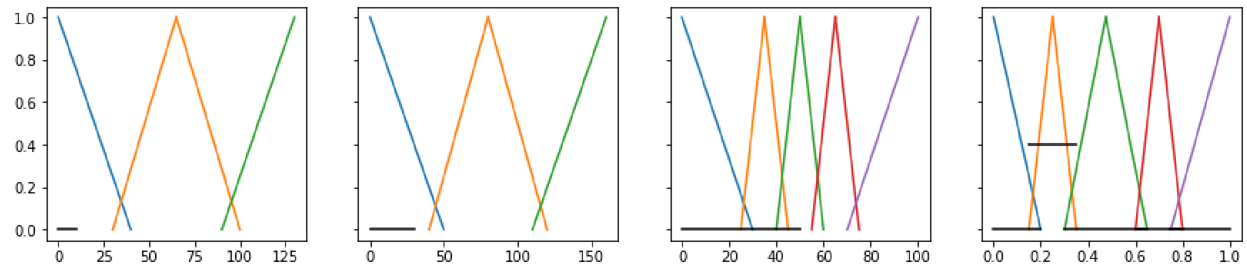
กฎข้อที่ 40 :



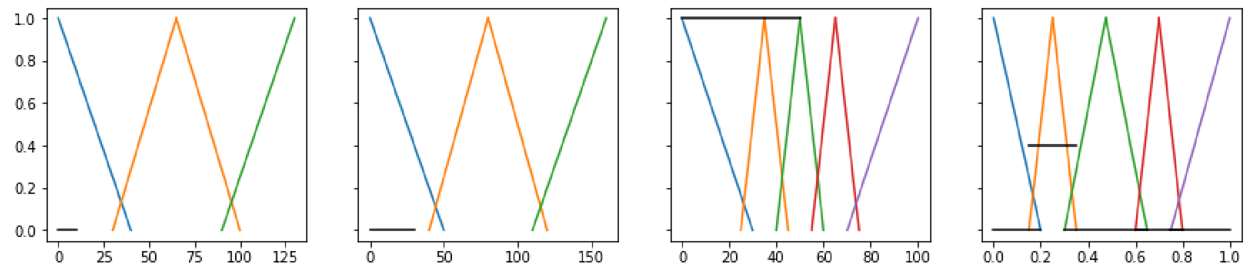
กฎข้อที่ 41 :



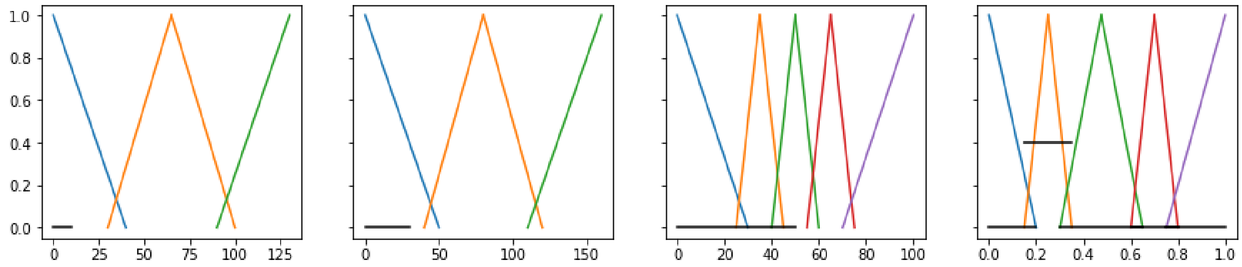
กฎข้อที่ 42 :



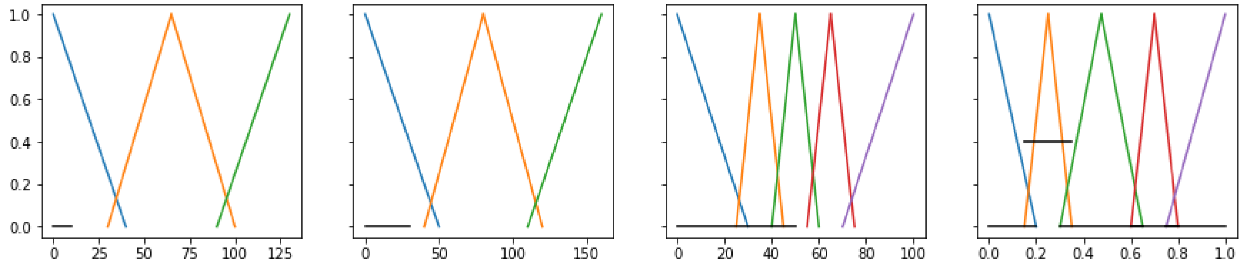
กฎข้อที่ 43 :



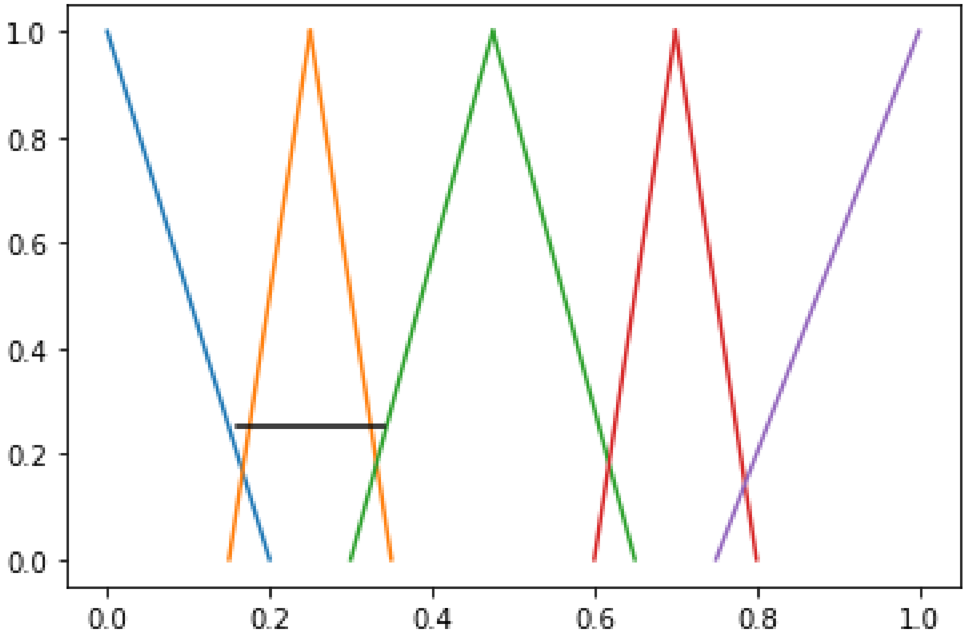
กฎข้อที่ 44 :



กฎข้อที่ 45 :



Output:



โดยเมื่อได้ Output แล้วจะนำไป Defuzzification โดยจะได้ค่าระดับความสุข = 0.25

ผลการทดลอง

INPUT			OUTPUT
If (ขนาด, Size)	**If (เวลา, Time)**	**If (อุณหภูมิ, Temp)**	**Else (ความสุก, Done)**
10	10	10	0.16342960288807
34	67	25	0.25000000000000
34	120	65	0.44273615941347
57	150	74	0.42808481532147
68	20	100	0.38489098408956
120	87	78	0.35111531190926
120	46	36	0.23071428571428
130	39	49	0.20533834586466

วิเคราะห์ผลการทดลอง

จะเห็นว่าเมื่อนำค่า Input ทั้งหมด 3 ค่า (Size , Time , Temp) มาใส่เข้าไปยัง Fuzzy Simulator จะได้ผลลัพธ์ที่ออกมาอยู่ในช่วงที่ตรงกับกฎที่ตั้งไว้ โดยเมื่อได้ผลลัพธ์ที่ตรงแม้จะเปลี่ยนแปลงค่า Input ทั้งหมด จึงสรุปได้ว่า Fuzzy Simulator นั้นทำงานได้อยากถูกต้อง แต่ข้อจำกัดของ Simulator นี้ก็คือ เมื่อเราใส่ Input ค่าอยู่เกิน หรือ อยู่นอกช่วงของขอบเขต ที่เรากำหนดไว้ จะทำให้เกิด Error ขึ้น

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from enum import Enum, auto
import matplotlib.pyplot as plt

class Fuzzy(Enum):
    SIZE = auto()
    TIME = auto()
    TEMPURETURE = auto()
    DONENESS = auto()

class SIZELVL(Enum):
    SMALL = auto()
    MEDIUM = auto()
    BIG = auto()

class TIMELVL(Enum):
    SHORT = auto()
    MEDIUM = auto()
    LONG = auto()

class TEMPLVL(Enum):
    MODERATE = auto()
    WARM = auto()
    VERY_WARM = auto()
    HOT = auto()
    SCORCHING = auto()

class DONELVL(Enum):
    RARE = auto()
    MEDIUM_RARE = auto()
    MEDIUM = auto()
    MEDIUM_WELL = auto()
    WELL_DONE = auto()

class range_step :
    def __init__(self,start,end,step):
        self.start = start
        self.end = end
        self.step =step

class Rule :
    def __init__(self,ifRule,thenRule):
        self.ifRule = ifRule
        self.thenRule = thenRule

class RuleData :
    def __init__(self,fuzzy,level) :
        self.fuzzy = fuzzy
        self.level = level

class Data :
    def __init__(self,name,RangeDat):
        self.name = name
        self.RangeDat = RangeDat

class Graph :

```

```python
    def __init__(self,start,end) :
        self.x_start = start
        self.x_end = end

    def getFuzzyleft(self,x) :
        m = 1/(self.x_start-self.x_end)
        c = -m*self.x_end
        y = m*x+c
        if(y > 1) :
            return 1
        elif(y < 0):
            return 0
        else:
            return y

    def getFuzzymid(self,x):
        center = self.x_start + (self.x_end - self.x_start)/2
        if(x < center) :
            x_end = center
            m = -1/(self.x_start-x_end)
            c = -m*self.x_start
            y = m*x+c
            if(y > 1) :
                return 1
            elif(y < 0):
                return 0
            else:
                return y
        else :
            x_start = center
            m = 1/(x_start-self.x_end)
            c = -m*self.x_end
            y = m*x+c
            if(y > 1) :
                return 1
            elif(y < 0):
                return 0
            else:
                return y

    def getFuzzyright(self,x) :
                 m = -1/(self.x_start-self.x_end)
                 c = -m*self.x_start
                 y = m*x+c
                 if(y > 1) :
                     return 1
                 elif(y < 0):
                     return 0
                 else:
                     return y
         def newRule(sizelvl,timelvl,templvl,donenesslvl):
             ifRule = []

             ifRule.append(RuleData(Fuzzy.SIZE,sizelvl))
             ifRule.append(RuleData(Fuzzy.TIME,timelvl))
             ifRule.append(RuleData(Fuzzy.TEMPURETURE,templvl))

             thenRule = RuleData(Fuzzy.DONENESS,donenesslvl)

             return Rule(ifRule,thenRule)

         c = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22", "#17becf"]

         def showSimulator(s,t,m,plot,maxinrule):
             fig, axs = plt.subplots(1, 4, figsize=(15, 3), sharey=True)
             axs[0].plot([0,40], [1,0],color=c[0])
             axs[0].plot([30,65], [0,1],color=c[1])
             axs[0].plot([65,100], [1,0],color=c[1])
             axs[0].plot([90,130], [0,1],color=c[2])
             axs[0].plot([0,s],[plot[0],plot[0]],color='black')

             axs[1].plot([0,50], [1,0],color=c[0])
             axs[1].plot([40,80], [0,1],color=c[1])
             axs[1].plot([80,120], [1,0],color=c[1])
             axs[1].plot([110,160], [0,1],color=c[2])
             axs[1].plot([0,t],[plot[1],plot[1]],color='black')

             axs[2].plot([0,30], [1,0],color=c[0])
             axs[2].plot([25,35], [0,1],color=c[1])
             axs[2].plot([35,45], [1,0],color=c[1])
             axs[2].plot([40,50], [0,1],color=c[2])
             axs[2].plot([50,60], [1,0],color=c[2])
             axs[2].plot([55,65], [0,1],color=c[3])
             axs[2].plot([65,75], [1,0],color=c[3])
             axs[2].plot([70,100], [0,1],color=c[4])
             axs[2].plot([0,m],[plot[2],plot[2]],color='black')

             levelofoutput_split = [[0,0.2],[0.15,0.25],[0.25,0.35],[0.3,0.475],[0.475,0.65],[0.6,0.7],[0.7,0.8],[0.75,1]]
             levelofoutput= [[0,0.2],[0.15,0.35],[0.3,0.65],[0.6,0.8],[0.75,1]]
             axs[3].plot(levelofoutput_split[0], [1,0],color=c[0])
             axs[3].plot(levelofoutput_split[1], [0,1],color=c[1])
             axs[3].plot(levelofoutput_split[2], [1,0],color=c[1])
             axs[3].plot(levelofoutput_split[3], [0,1],color=c[2])
             axs[3].plot(levelofoutput_split[4], [1,0],color=c[2])
             axs[3].plot(levelofoutput_split[5], [0,1],color=c[3])
             axs[3].plot(levelofoutput_split[6], [1,0],color=c[3])
             axs[3].plot(levelofoutput_split[7], [0,1],color=c[4])
             for idx,val in enumerate(maxinrule):
                 axs[3].plot(levelofoutput[idx],[val,val],color='black')

             plt.show()

         def showSimulator_result(use_x,defuz):

             plt.plot([0,0.2], [1,0],color=c[0])
             plt.plot([0.15,0.25], [0,1],color=c[1])
             plt.plot([0.25,0.35], [1,0],color=c[1])
             plt.plot([0.3,0.475], [0,1],color=c[2])
             plt.plot([0.475,0.65], [1,0],color=c[2])
             plt.plot([0.6,0.7], [0,1],color=c[3])
             plt.plot([0.7,0.8], [1,0],color=c[3])
             plt.plot([0.75,1], [0,1],color=c[4])
             plt.plot([use_x[0],use_x[-1]],[defuz,defuz],color='black')
             plt.show()

         Size = {SIZELVL.SMALL : Graph(0,40),
                    SIZELVL.MEDIUM : Graph(30,100),
                    SIZELVL.BIG : Graph(90,130)}

         Time = {TIMELVL.SHORT : Graph(0,50),
                    TIMELVL.MEDIUM : Graph(40,120),
                    TIMELVL.LONG : Graph(110,160)}

         Tempureture = {TEMPLVL.MODERATE : Graph(0,30),
                                 TEMPLVL.WARM : Graph(25,45),
                                 TEMPLVL.VERY_WARM : Graph(40,60),
                                 TEMPLVL.HOT : Graph(55,75),
                                 TEMPLVL.SCORCHING : Graph(70,100)}

         Doneness = {DONELVL.RARE : Graph(0,0.2),
                             DONELVL.MEDIUM_RARE : Graph(0.15,0.35),
                             DONELVL.MEDIUM : Graph(0.3,0.65),
                             DONELVL.MEDIUM_WELL : Graph(0.6,0.8),
                             DONELVL.WELL_DONE : Graph(0.75,1)}

         Input = {Fuzzy.SIZE : Data(Size,range_step(0,130,1)),
                     Fuzzy.TIME : Data(Time,range_step(0,160,1)),
                     Fuzzy.TEMPURETURE : Data(Tempureture,range_step(0,100,1))}

         Output = {Fuzzy.DONENESS : Data(Doneness,range_step(0,1,0.01))}

         Rules = []

         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.SHORT,TEMPLVL.MODERATE,DONELVL.RARE))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.SHORT,TEMPLVL.WARM,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.SHORT,TEMPLVL.VERY_WARM,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.SHORT,TEMPLVL.HOT,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.SHORT,TEMPLVL.SCORCHING,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.MEDIUM,TEMPLVL.MODERATE,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.MEDIUM,TEMPLVL.WARM,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.MEDIUM,TEMPLVL.VERY_WARM,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.MEDIUM,TEMPLVL.HOT,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.MEDIUM,TEMPLVL.SCORCHING,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.LONG,TEMPLVL.MODERATE,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.LONG,TEMPLVL.WARM,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.LONG,TEMPLVL.VERY_WARM,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.LONG,TEMPLVL.HOT,DONELVL.WELL_DONE))
         Rules.append(newRule(SIZELVL.SMALL,TIMELVL.LONG,TEMPLVL.SCORCHING,DONELVL.WELL_DONE))

         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.SHORT,TEMPLVL.MODERATE,DONELVL.RARE))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.SHORT,TEMPLVL.WARM,DONELVL.RARE))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.SHORT,TEMPLVL.VERY_WARM,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.SHORT,TEMPLVL.HOT,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.SHORT,TEMPLVL.SCORCHING,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.MEDIUM,TEMPLVL.MODERATE,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.MEDIUM,TEMPLVL.WARM,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.MEDIUM,TEMPLVL.VERY_WARM,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.MEDIUM,TEMPLVL.HOT,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.MEDIUM,TEMPLVL.SCORCHING,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.LONG,TEMPLVL.MODERATE,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.LONG,TEMPLVL.WARM,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.LONG,TEMPLVL.VERY_WARM,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.LONG,TEMPLVL.HOT,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.MEDIUM,TIMELVL.LONG,TEMPLVL.SCORCHING,DONELVL.WELL_DONE))

         Rules.append(newRule(SIZELVL.BIG,TIMELVL.SHORT,TEMPLVL.MODERATE,DONELVL.RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.SHORT,TEMPLVL.WARM,DONELVL.RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.SHORT,TEMPLVL.VERY_WARM,DONELVL.RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.SHORT,TEMPLVL.HOT,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.SHORT,TEMPLVL.SCORCHING,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.MEDIUM,TEMPLVL.MODERATE,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.MEDIUM,TEMPLVL.WARM,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.MEDIUM,TEMPLVL.VERY_WARM,DONELVL.MEDIUM_RARE))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.MEDIUM,TEMPLVL.HOT,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.MEDIUM,TEMPLVL.SCORCHING,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.LONG,TEMPLVL.MODERATE,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.LONG,TEMPLVL.WARM,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.LONG,TEMPLVL.VERY_WARM,DONELVL.MEDIUM))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.LONG,TEMPLVL.HOT,DONELVL.MEDIUM_WELL))
         Rules.append(newRule(SIZELVL.BIG,TIMELVL.LONG,TEMPLVL.SCORCHING,DONELVL.MEDIUM_WELL))

         class SousVideFuzzyLogic :
             def __init__(self,input_data,output_data,rules):
                 self.input_data = input_data
                 self.output_data = output_data
                 self.rules = rules

             def defuzzifier(self,size,time,temp):

                 maxInRule = np.zeros(5)
                 plot = np.zeros(3)

                 for rule in self.rules:
                     minInRule = 1
                     for ruleData in rule.ifRule :
                         done = 0
                         if(ruleData.fuzzy == Fuzzy.SIZE):
                             if(ruleData.level == SIZELVL.SMALL):
                                 done = (self.input_data[Fuzzy.SIZE]).name[SIZELVL.SMALL].getFuzzyleft(size)
                             if(ruleData.level == SIZELVL.MEDIUM):
                                 done = (self.input_data[Fuzzy.SIZE]).name[SIZELVL.MEDIUM].getFuzzymid(size)
                             if(ruleData.level == SIZELVL.BIG):
                                 done = (self.input_data[Fuzzy.SIZE]).name[SIZELVL.BIG].getFuzzyright(size)
                             plot[0] = done

                         elif(ruleData.fuzzy == Fuzzy.TIME):
                             if(ruleData.level == TIMELVL.SHORT):
                                 done = (self.input_data[Fuzzy.TIME]).name[TIMELVL.SHORT].getFuzzyleft(time)
                             if(ruleData.level == TIMELVL.MEDIUM):
                                 done = (self.input_data[Fuzzy.TIME]).name[TIMELVL.MEDIUM].getFuzzymid(time)
                             if(ruleData.level == TIMELVL.LONG):
                                 done = (self.input_data[Fuzzy.TIME]).name[TIMELVL.LONG].getFuzzyright(time)
                             plot[1] = done

                         elif(ruleData.fuzzy == Fuzzy.TEMPURETURE):
                             if(ruleData.level == TEMPLVL.MODERATE):
                                 done = (self.input_data[Fuzzy.TEMPURETURE]).name[TEMPLVL.MODERATE].getFuzzyleft(temp)
                             if(ruleData.level == TEMPLVL.WARM):
                                 done = (self.input_data[Fuzzy.TEMPURETURE]).name[TEMPLVL.WARM].getFuzzymid(temp)
                             if(ruleData.level == TEMPLVL.VERY_WARM):
                                 done = (self.input_data[Fuzzy.TEMPURETURE]).name[TEMPLVL.VERY_WARM].getFuzzymid(temp)
                             if(ruleData.level == TEMPLVL.HOT):
                                 done = (self.input_data[Fuzzy.TEMPURETURE]).name[TEMPLVL.HOT].getFuzzymid(temp)
                             if(ruleData.level == TEMPLVL.SCORCHING):
                                 done = (self.input_data[Fuzzy.TEMPURETURE]).name[TEMPLVL.SCORCHING].getFuzzyright(temp)
                             plot[2] = done

                         if(minInRule > done and done >= 0):
                             minInRule = done

                     if(rule.thenRule.level == DONELVL.RARE):
                         if(maxInRule[0] < minInRule):
                             maxInRule[0] = minInRule
                     elif(rule.thenRule.level == DONELVL.MEDIUM_RARE):
                         if(maxInRule[1] < minInRule):
                             maxInRule[1] = minInRule
                     elif(rule.thenRule.level == DONELVL.MEDIUM):
                         if(maxInRule[2] < minInRule):
                             maxInRule[2] = minInRule
                     elif(rule.thenRule.level == DONELVL.MEDIUM_WELL):
                         if(maxInRule[3] < minInRule):
                             maxInRule[3] = minInRule
                     elif(rule.thenRule.level == DONELVL.WELL_DONE):
                         if(maxInRule[4] < minInRule):
                             maxInRule[4] = minInRule

         #              showSimulator(size,time,temp,plot,maxInRule)

                 start_out = self.output_data[Fuzzy.DONENESS].RangeDat.start
                 end_out = self.output_data[Fuzzy.DONENESS].RangeDat.end
                 step_out = self.output_data[Fuzzy.DONENESS].RangeDat.step

                 start = []
                 end =[]
                 for i in DONELVL :
                     start.append(self.output_data[Fuzzy.DONENESS].name[i].x_start)
                     end.append(self.output_data[Fuzzy.DONENESS].name[i].x_end)

                 defuzzi = np.zeros(2)
                 use_x = []
                 for x in range(start_out*100,end_out*100,int(step_out*100)):

                     x = x/100
                     y = np.zeros(5)

                     if(x <= end[0]):
                         y[0] = self.output_data[Fuzzy.DONENESS].name[DONELVL.RARE].getFuzzyleft(x)
                         if(y[0] > maxInRule[0]):
                             y[0] = maxInRule[0]
                     if(x >= start[1] and x <= end[1]):
                         y[0] = self.output_data[Fuzzy.DONENESS].name[DONELVL.MEDIUM_RARE].getFuzzymid(x)
                         if(y[1] > maxInRule[1]):
                             y[1] = maxInRule[1]
                     if(x >= start[2] and x <= end[2]):
                         y[2] = self.output_data[Fuzzy.DONENESS].name[DONELVL.MEDIUM].getFuzzymid(x)
                         if(y[2] > maxInRule[2]):
                             y[2] = maxInRule[2]
                     if(x >= start[3] and x <= end[3]):
                         y[3] = self.output_data[Fuzzy.DONENESS].name[DONELVL.MEDIUM_WELL].getFuzzymid(x)
                         if(y[3] > maxInRule[3]):
                             y[3] = maxInRule[3]
                     if(x >= start[4]):
                         y[4] = self.output_data[Fuzzy.DONENESS].name[DONELVL.WELL_DONE].getFuzzyright(x)
                         if(y[4] > maxInRule[4]):
                             y[4] = maxInRule[4]

                     maximum = np.max(y)
                     if(maximum > 0):
                         use_x.append([x,maximum])

                     defuzzi[0] += maximum*x
                     defuzzi[1] += maximum

                 if(defuzzi[1] == 0):
                     defuzzi[1] = 1
         #         print(use_x)
         #         showSimulator_result(use_x,defuzzi[0]/defuzzi[1])

                 return defuzzi[0]/defuzzi[1]

         souvidefuzzy = SousVideFuzzyLogic(Input,Output,Rules)
         souvidefuzzy.defuzzifier(130,39,49)
```