Lab exercise 3 - Fuzzy set

Sigve Skaugvoll & Eirik Baug

# Task 1 Manual Mandani

## Step 1: Fuzzification!

The **value for distance** = 3.7 and intersects `Small` at `0.6` and `Perfect` at `0.1`. The others have value of 0

The **value for delta** = 1.2 and intersects `Stable` at `0.3` and `Growing` at `0.4`. The others have value of 0

**Fuzzy logic:**

- AND = min

- OR = max

- NOT x = 1-x

## Step 2 : Rule evaluation

Values 0.6 0.4 ==>min(0.6, 0.4) = 0.4

IF distance is Small AND delta is Growing THEN action is None

values 0.6 0.3 ==> min(0.6, 0.3) = 0.3

IF distance is Small AND delta is Stable THEN action is SlowDown

values (0.1) (0.4) ==> min(0.1,0.4) = 0.1

IF distance is Perfect AND delta is Growing THEN action is SpeedUp

values 0 (1 - 0.4) (1 - 0) ==> min(0, max(0.6, 1)) = 0

IF distance is VeryBig AND (delta is NOT Growing OR delta is NOT GrowingFast) THEN action is FloorIt

Values 0 ==> 0

IF distance is VerySmall THEN action is BrakeHard

## Step 3 : Aggregation!

If we now clip the fuzzy sets based on these rule evaluations we get:

huff.jpg

The clipped action fuzzy set are combined into just ONE new fuzzy set. (Clipped on right hand side values from step 2 (rule evaluation).

This gives us a NEW fuzzy set that looks like this: (3 images, all of same set, just shows where to cut, cut, and then another cleaner cut).

22243942_10214398011369190_1737994208_o.jpg

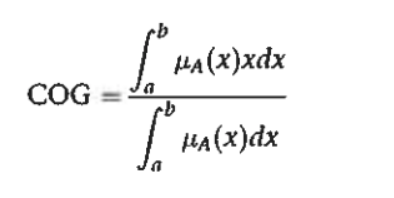
22251321_10214398011289188_1485741823_o.jpg

22251308_10214398011329189_1798030994_o (1).jpg

Now that we have created our new cut action fuzzy set. It’s time to find the goddamn solution!

## Step 4. Defuzzification

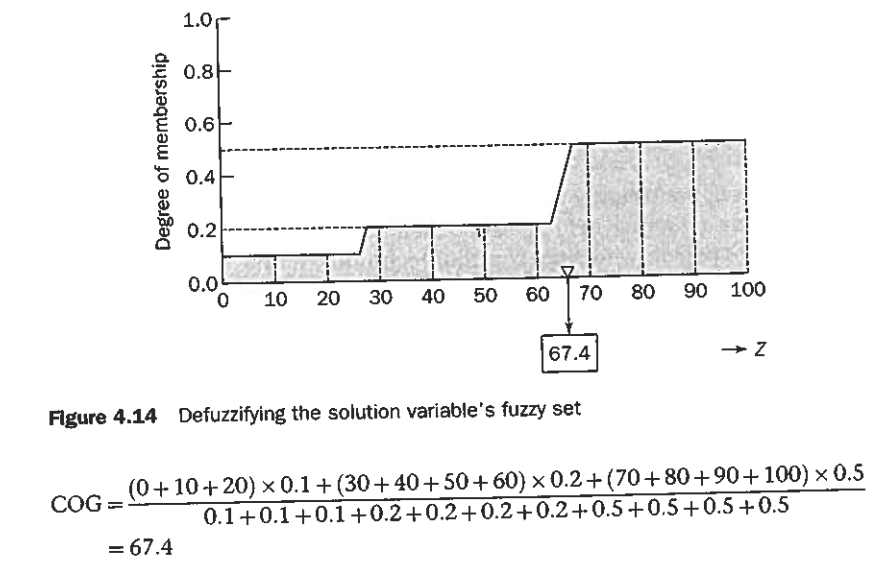
Defuzzification is just to calculate the area under / inside the new set / graph and is the actual fuzzy inference process. We do this by  **centroid technique,** integral ( ∫ ) calculation. It finds the point where a vertical line would slice the aggregate set into two equal masses. Mathematically this **centre of gravity (COG)** can be expressed as



In theory, the COG is calcualted over a continuum of points in the aggregate output membership functio, but in practice, a reasonable estimate can be obtained by calcualting it over a sample of points. Thus COG can be writtesn as



Here is an **example** of how to use the COG on a aggregated fuzzy set!



Clipped values:

0.1

0.2

0.3

22278928_10214398250415166_1743605227_o.jpg

-2 skiller SlowDown & None

2 skiller None & SpeedUp

# Task 2 Programming Mandani

**import** sys

**import** random **as** r

**class Mamdani**:

**def** \_\_init\_\_(self, disPos, delPos):

self.distancePos = disPos

self.deltaPos = delPos

**def** \_\_repr\_\_(self):

**return** "I'm Mamdani!\nWho are you?"

########

# Rules

#######

**def AND\_rule**(self, \*args):

**return** min(\*args)

**def OR\_rule**(self, \*args):

**return** max(\*args)

**def NOT\_rule**(self, value):

**return** 1 - value

########

# Sets

#######

**def distance\_set**(self):

setValues = {

"VerySmall": ['RG', [1.5, 2.75]],

"Small": ['T', [1.75, 3, 4.75]],

"Perfect": ['T', [3.75, 5, 6.75]],

"Big": ['T', [5.75, 7, 8.75]],

"VeryBig": ['G', [7.75, 9]]

}

**return** setValues

**def delta\_time\_set**(self):

setValues = {

"ShrinkingFast": ['RG', [-3.5, -2.25]],

"Shrinking": ['T', [-3.25, -1.75, -0.25]],

"Stable": ['T', [-1.25, -0.25, 1.75]],

"Growing": ['T', [0.75, 2.25, 3.75]],

"GrowingFast": ['G', [2.75, 4.25]]

}

**return** setValues

**def action\_set**(self):

setValues = {

"BrakeHard": ['RG', [-7.5, -5]],

"SlowDown": ['T', [-7, -4, -1]],

"None": ['T', [-3, 0, 3]],

"SpeedUp": ['T', [1, 4, 7]],

"FloorIt": ['G', [5, 7.5]]

}

**return** setValues

########

# Membership functions

#######

**def triangle**(self, position, x0, x1, x2, clip):

value = 0.0

**if** position >= x0 **and** position <= x1:

value = (position - x0) / (x1 - x0)

**elif** position >= x1 **and** position <= x2:

value = (x2 - position) / (x1 - x0)

**if** value > clip:

value = clip

**return** value

**def grade**(self, position, x, y, clip):

value = 0.0

**if** position >= y:

value = 1.0

**elif** position <= x:

value = 0.0

**else**:

value = (position - x) / (y - x)

**if** value > clip:

value = clip

**return** value

**def reverse\_grade**(self, position, x, y, clip):

value = 0.0

**if** position <= x:

value = 1.0

**elif** position >= y:

value = 0.0

**else**:

value = (y - position) / (y - x)

**if** value > clip:

value = clip

**return** value

########

# ACTIONS

#######

**def AND\_TRIANGLE**(self, distKey, deltKey, clip):

dix0, dix1, dix2 = self.distance\_set()[distKey][1]

dex0, dex1, dex2 = self.delta\_time\_set()[deltKey][1]

value = self.AND\_rule(self.triangle(self.distancePos, dix0, dix1, dix2, clip),

self.triangle(self.deltaPos, dex0, dex1, dex2, clip))

**return** value

**def RuleEvaluation**(self, distance, delta, clip=100):

return\_set = {}

**if** "Small" **in** distance **and** "Growing" **in** delta:

t = self.AND\_TRIANGLE("Small", "Growing", clip)

return\_set["None"] = t

**if** "Small" **in** distance **and** "Stable" **in** delta:

t = self.AND\_TRIANGLE("Small", "Stable", clip)

return\_set["SlowDown"] = t

**if** "Perfect" **in** distance **and** "Growing" **in** delta:

t = self.AND\_TRIANGLE("Perfect", "Growing", clip)

return\_set["SpeedUp"] = t

**if** "VeryBig" **in** distance **and** "Growing" **in** delta **and** "GrowingFast" **in** delta:

dix0, dix1 = self.distance\_set()["VeryBig"][1]

dex0, dex1, dex2 = self.delta\_time\_set()["Growing"][1]

dex0, dex1 = self.delta\_time\_set()["GrowingFast"][1]

a = self.grade(self.distancePos, dix0, dix1, clip)

print("\*\*\*\*\*\*\*A: " + str(a))

b = self.NOT\_rule(self.triangle(self.deltaPos, dex0, dex1, dex2, clip)) # not growing

print("\*\*\*\*\*\*\*B: " + str(b))

c = self.NOT\_rule(self.grade(self.deltaPos, dex0, dex1, clip)) # not growingFast

print("\*\*\*\*\*\*\*C: " + str(c))

t = self.AND\_rule(a, self.OR\_rule(b, c))

print("\*\*\*\*\*\*\*T: " + str(t))

return\_set["FoorIt"] = t

**if** "VerySmall" **in** distance:

dix0, dix1 = self.distance\_set()["VerySmall"][1]

t = self.reverse\_grade(self.distancePos, dix0, dix1, clip)

return\_set["BrakeHard"] = t

# return clip values set

**return** return\_set

**def getIntersection**(self, set, value):

dict = {}

**for** key **in** set:

**if** set[key][1][0] <= value **and** value <= set[key][1][-1]:

**if** set[key] == 'RG':

dict[key] = self.reverse\_grade(value, set[key][1][0], set[key][1][1], 100)

**elif** set[key] == 'T':

dict[key] = self.triangle(value, set[key][1][0], set[key][1][1], set[key][1][2], 100)

**else**:

dict[key] = self.grade(value, set[key][1][0], set[key][1][1], 100)

**return** dict

**def reasoning**(self):

# Step 1: Fuzzication (Find values for each set in fuzzyset)

intersection\_set\_dist = self.getIntersection(self.distance\_set(), self.distancePos)

intersection\_set\_delta = self.getIntersection(self.delta\_time\_set(), self.deltaPos)

print("distance intersection ", intersection\_set\_dist, "\ndelta intersection ", intersection\_set\_delta)

# step 2: Rule evaluation

action\_set = self.RuleEvaluation(intersection\_set\_dist, intersection\_set\_delta, 100)

print("action set: ", action\_set)

# Step 3: Aggregation

aggregated\_set = self.aggregate(action\_set)

print("Aggregated set ", aggregated\_set)

# step 4: Defuzzification

defuzz = self.Defuzzification(aggregated\_set)

**return** defuzz

**def Defuzzification**(self, aggregated\_set):

top\_level = 0.0

bottom\_level = 0.0

**for** key **in** aggregated\_set:

values = aggregated\_set[key][1]

val = 0

count = 0

**for** x **in** range(values[0], values[2] + 1):

val += x

count += 1

top\_level += val \* values[1]

bottom\_level += count \* values[1]

COG = top\_level / bottom\_level

**return** COG

**def aggregate**(self, action\_set):

aggregated\_set = {}

action\_s = self.action\_set()

**for** key **in** action\_set:

aggregated\_set[key] = action\_s[key]

**if** (aggregated\_set[key][0] == "RG"):

aggregated\_set[key][1][0] = action\_set[key]

**else**:

aggregated\_set[key][1][1] = action\_set[key]

**return** aggregated\_set

########

# Running this badboy

#######

**def main**(distancePos, deltaPos):

*'''*

*Assignment values: $ python Mamdani\_reasoner.py 3.7 1.7*

*'''*

distancePos = float(distancePos) # convert sys args to float, not string

deltaPos = float(deltaPos) # convert sys args to float, not string

m = Mamdani(distancePos, deltaPos)

action = m.reasoning()

print("Action crisp value is: " + str(action))

main(sys.argv[1], sys.argv[2])