Theorie Vragen: Fuzzy Logics

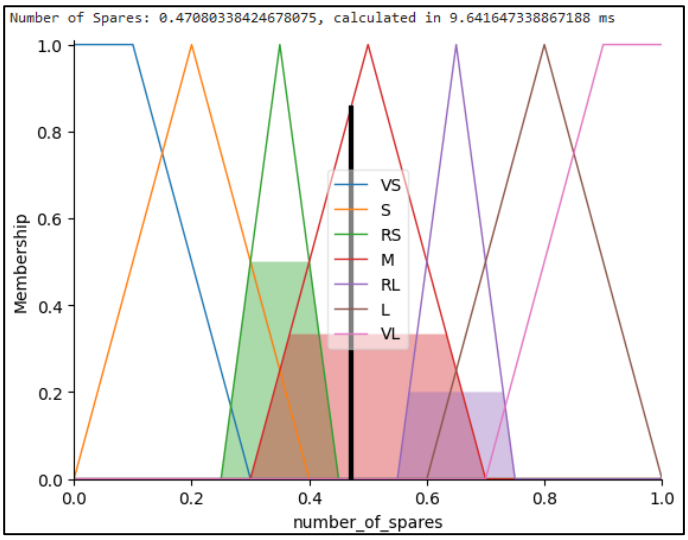
Auteur: Brian van den Berg

Demo

Als demo om te demonstreren hoe goed scikit-fuzzy in Python 3.10 werkt, heb ik het voorbeeld gemaakt uit Hoofdstuk 4.7 uit het boek 'Artificial Intelligence – A Guide to Intelligent Systems' geschreven door Michael Negnevitsky. Dit voorbeeld berekend de hoeveelheid extra onderdelen die nodig is om zo efficiënt mogelijk te kunnen werken zonder dat de hoeveelheid te veel kost. De Jupyter Notebook waarin de demo is uitgewerkt kunt u vinden in de bijlagen van dit rapport, maar in dit hoofdstuk zal ik kort langs de resultaten gaan.

Resultaten

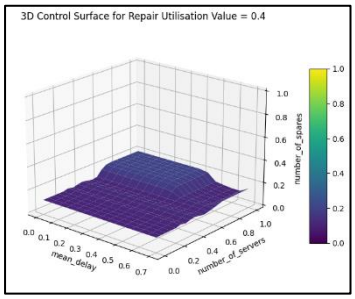
Zoals te zien is in de grafiek hiernaast kon mijn Fuzzy systeem de nodige hoeveelheid extra onderdelen berekenen in de kleine tijdspan van 9.6ms. Dat is razendsnel voor een intelligent systeem. Daarnaast bevat dit systeem ook 22 regels die te maken hebben met drie variabele oorzaken en een variabel gevolg.

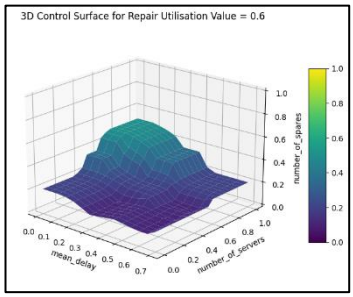


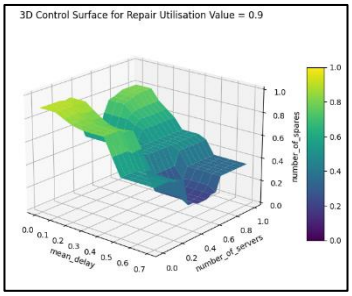
Omdat het zo snel de regels kan toepassen en met een resultaat kan komen dat op alle regels is afgestemd is het nu al mogelijk voor mij om te zeggen dat scikit-fuzzy gebuikt kan worden in robots, omdat de intelligentie realtime toepasbaar is.

3D Grafische Weergaven van het Controle Veld

De grafieken hieronder zijn gegenereerd door middel van matplotlib.pyplot, dus dit zegt nog niks over of scikit-fuzzy een eigen grafische weergaven heeft; daarover komt later in dit rapport meer.







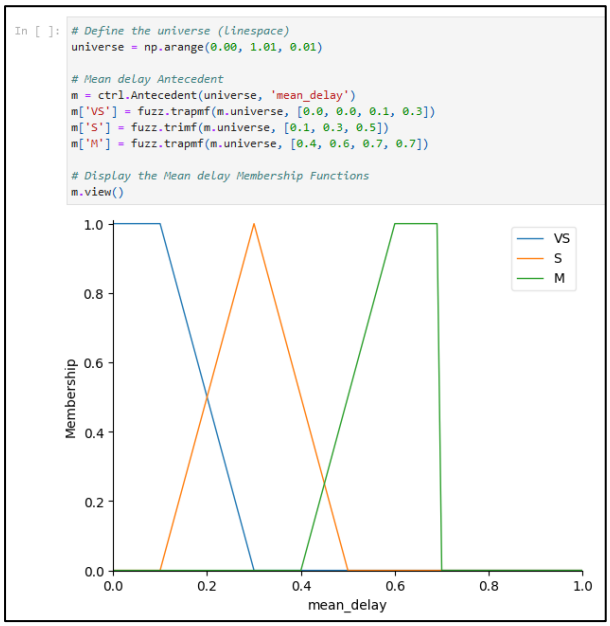
Echter is het wel een goede representatie over hoe het systeem waardes indeelt op basis van de input variabelen. Zo is er een duidelijke trend te zien dat hoe lager "mean_delay" is, hoe hoger de nodige "number_of_spares" is en hetzelfde geld voor "number_of_servers". En een hogere achterliggende "repair_utilisation" intensifieert de grafiek ook, dus deze heeft ook een impact op hoeveel extra onderdelen nodig zijn.

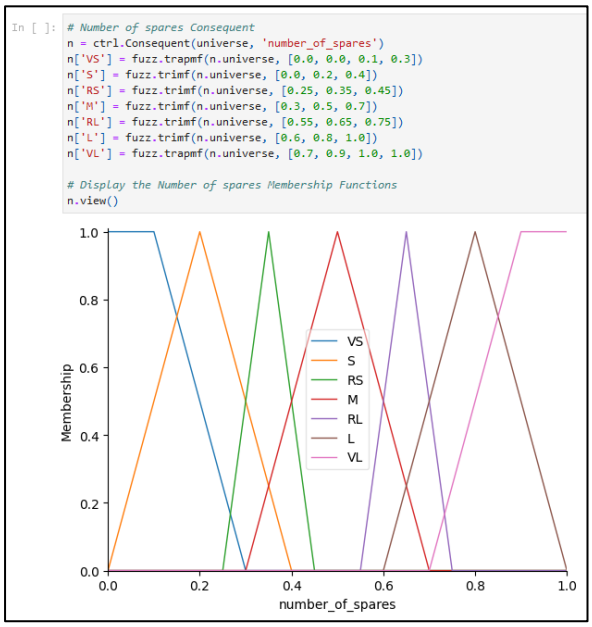
Scikit-fuzzy

In de library “skfuzzy” heb je in principe een raamwerk voor het maken van een Fuzzy system. Binnen dit raamwerk creëer je vervolgens “Fuzzy Sets” bestaande uit taalkundige variabelen. Deze variabelen kunnen van alles zijn, maar het representeert normaal de wijze van spreken die experts toepassen bij het praten over het vak. Zo kan ik bijvoorbeeld **veel** fietsen, maar voor een computer is de term “veel” niet te begrijpen. Dat wordt hiermee opgelost en op deze manier kan ik “veel” een range aan waardes geven die beschrijven hoe dicht iets bij “veel” komt. Het principe van fuzzy systemen is dat het kan omgaan met vage kennis en dus ook met waarschijnlijkheden kan werken.

Fuzzy Sets

Nu dat het principe van een Fuzzy Systeem beschreven is zal ik nu ook laten zien hoe een Fuzzy Set in elkaar zit binnen de library “scikit-fuzzy”. Dit kun je zien in de voorbeelden hieronder:





Plaatje 1 laat een “Antededent” zien waarin de taalkundige variabelen worden toegevoegt die de Member Function laten werken. Vervolgens is er door middel van de ingebouwde view() function een grafiek weergeven die de complete membership function laat zien.

Plaatje 2 ziet er een stuk complexer uit, maar in principe gebeurd hier hetzelfde. Het enige verschil is dat bij dit figuur een “Consequent” wordt laten zien. Dit kan worden beschouwd als een gevolg, waarbij de “Antecedent” of het meervoud hiervan gezien kan worden als de oorzaak.

Grafische mogelijkheden

In het resultaat op pagina 1 is een defuzzified resultaat te zien van input waardes die door het systeem heen gehaald zijn en als een “crispy” waarde teruggegeven is, daarnaast is de zwarte verticale lijn de uitgerekende “Centroid” uit de taalkundige variabelen van de “Consequent”. Dat is de eerste grafische mogelijkheid en als tweede is het ook mogelijk om de losse “Membership Functions” te zien zoals op deze pagina is weergeven. Dus er is zeker weten grafische output mogelijk van het systeem.

Door middel van het grafische overzicht van het resultaat is mijn hypothese ook dat er als defuzzification methode “COG” gebruikt, omdat het erop lijkt dat er een centroid berekent wordt uit het oppervlakte van de fuzzy set. Daarbij lijkt er gebruik te worden gemaakt van “clipping”

Fuzzy Regels

In scikit-fuzzy worden de regels opgesteld zoals verwacht in elk fuzzy logics systeem. Er wordt namelijk gewerkt met een “IF X == A, THEN Y = B” regelgeving. Interessant is dat dit in principe heel erg overeenkomt met hoe normale expertsystemen werken. De regels worden samengesteld door middel van logische operatoren. Logische operatoren zijn “AND”, “OR” en “NOT”. Deze operatoren worden toegepast op de variabele “Antecedent” die we eerder hebben gedefinieerd en de “Consequent” wordt aan de hand hiervan beïnvloed in de “THEN”. Hieronder staan de regels die in mijn demo gedefinieerd zijn. Deze zijn afgeleid uit de matrix met 45 regels in hoofdstuk 4.7 uit het boek.

```
In [ ]: # Fuzzy System Rules
system_rules = [
    ctrl.Rule(antecedent=((s['S'] | s['RS'] | s['M']) & p['L']),              consequent=n['VS']),
    ctrl.Rule(antecedent=((m['VS'] | m['S']) & (s['RL'] | s['L']) & p['L']),  consequent=n['S']),
    ctrl.Rule(antecedent=(m['M'] & (s['RL'] | s['L']) & p['L']),              consequent=n['VS']),
    ctrl.Rule(antecedent=(m['VS'] & (s['S'] | s['RS']) & p['M']),             consequent=n['S']),
    ctrl.Rule(antecedent=((m['S'] | m['M']) & (s['S'] | s['RS']) & p['M']),   consequent=n['VS']),
    ctrl.Rule(antecedent=(m['VS'] & s['M'] & p['M']),                         consequent=n['RS']),
    ctrl.Rule(antecedent=(m['S'] & s['M'] & p['M']),                          consequent=n['S']),
    ctrl.Rule(antecedent=(m['M'] & s['M'] & p['M']),                          consequent=n['VS']),
    ctrl.Rule(antecedent=(m['VS'] & (s['RL'] | s['L']) & p['M']),             consequent=n['M']),
    ctrl.Rule(antecedent=(m['S'] & (s['RL'] | s['L']) & p['M']),              consequent=n['RS']),
    ctrl.Rule(antecedent=(m['M'] & (s['RL'] | s['L']) & p['M']),              consequent=n['S']),
    ctrl.Rule(antecedent=(m['VS'] & (s['S'] | s['RS']) & p['H']),             consequent=n['VL']),
    ctrl.Rule(antecedent=(m['S'] & s['S'] & p['H']),                          consequent=n['L']),
    ctrl.Rule(antecedent=(m['M'] & s['S'] & p['H']),                          consequent=n['M']),
    ctrl.Rule(antecedent=(m['S'] & s['RS'] & p['H']),                         consequent=n['RL']),
    ctrl.Rule(antecedent=(m['M'] & s['RS'] & p['H']),                         consequent=n['RS']),
    ctrl.Rule(antecedent=(m['VS'] & s['M'] & p['H']),                         consequent=n['M']),
    ctrl.Rule(antecedent=(m['S'] & (s['M'] | s['RL'] | s['L']) & p['H']),     consequent=n['M']),
    ctrl.Rule(antecedent=(m['M'] & s['M'] & p['H']),                          consequent=n['S']),
    ctrl.Rule(antecedent=(m['VS'] & s['RL'] & p['H']),                        consequent=n['RL']),
    ctrl.Rule(antecedent=(m['M'] & (s['RL'] | s['L']) & p['H']),              consequent=n['RS']),
    ctrl.Rule(antecedent=(m['VS'] & s['L'] & p['H']),                         consequent=n['L']),
]
```

Gebruik van het Systeem

Nadat de regels gedefinieerd staan is het enorm snel en simpel om het expertsysteem te gebruiken. Zo duurt het voorbeeld gebruik uit de afbeelding hieronder maar 9.6ms om uit te voeren.

```
# Test run
sim.input['mean_delay'] = 0.03
sim.input['number_of_servers'] = 0.60
sim.input['repair_utilisation'] = 0.64
sim.compute()
output = sim.output["number_of_spares"]
```

Na het uitvoeren van deze code bevat output de waarde die uitgerekend is door het fuzzy systeem. Dit is de defuzzified waarde uit “number_of_spares”. Door dit raamwerk te gebruiken zou ik zeker zeggen dat een fuzzy systeem bouwen over het algemeen vrij simpel is gemaakt. Het gebruik is nog makkelijker en de snelheid is, voor een intelligent systeem gemaakt in een Python script, enorm goed.

Pro's en Cons

Het gebruik van sci-fuzzy is, zoals in het rapport over de demo al is beschreven, gevuld met pro's. Echter zijn er ook een paar cons die hangen aan het gebruik van dit raamwerk.

Pro's:

- Makkelijk te gebruiken en overzien.
- Ingebouwde grafische methoden voor evaluatie.
- Relatief snel vergeleken met andere intelligente systemen.

Cons

- Zover ik kon vinden, waren hedges zoals [µA[x]]^2, wat "very" voorstelt in geschreven tekst, niet echt toepasbaar in het raamwerk.
- Door het gebruik van dit raamwerk verlies je controle.
- De onderliggende methodieken zijn niet echt duidelijk bij gebruik.