14.03.2023

**Sztuczna Inteligencja**

Sprawozdanie nr.1

Temat: Synteza układu wnioskującego

Prowadzący:

dr hab. inż. Roman Zajdel, prof. PRz

Wykonali:

2EF-DI, L8, gr.2

Daniel Kleczyński

Michał Rzeźnikiewicz

Mikołaj Selamak

Dominik Serafin

Tomek Słapiński

Rafał Stępień

Spis treści

[1. Wstęp teoretyczny 3](#_Toc130766639)

[2. Środowisko programistyczne 3](#_Toc130766640)

[3. Przebieg ćwiczenia 4](#_Toc130766641)

[3.1. Część I: Wykonanie przykładowego systemu ekspertowego typu Mamdaniego 4](#_Toc130766642)

[3.2. Część II. Modyfikacja systemu 8](#_Toc130766643)

[3.3. Część III: Przykładowe zadanie zaliczeniowe 10](#_Toc130766644)

# Wstęp teoretyczny

Do stworzenia systemu wnioskującego można użyć systemów rozmytych. Są to modele matematyczne, które pozwalają na opisywanie sytuacji, w których obiekty lub zjawiska są trudne do opisania jednoznacznie. Zaletą systemów rozmytych jest to, że pozwalają one na modelowanie nieliniowych zależności pomiędzy zmiennymi, co czyni je szczególnie przydatnymi w przypadku zagadnień związanych z kontrolą procesów oraz sterowaniem przykładam może być system sterowania pociągów w Japonii lub sterowanie robotem khepler .

Jednym z najbardziej znanych i popularnych modeli systemów rozmytych jest model Mandianiego. Model ten opiera się na trzech podstawowych elementach: zbiorach rozmytych, regułach rozmytych oraz wnioskowaniu rozmytym. Zbiory rozmyte to zbiory, w których elementy posiadają stopień przynależności do zbioru, a nie są one jednoznacznie określone, jak w przypadku zbiorów klasycznych. Reguły rozmyte opisują zależności pomiędzy zmiennymi lingwistycznymi, np. "jeśli prędkość jest duża, to hamowanie powinno być ostre". Wnioskowanie rozmyte polega na stosowaniu reguł rozmytych do obliczenia stopnia przynależności obiektu do różnych zbiorów rozmytych.

W Pythonie możemy użyć pakietu scikit-fuzzy. Biblioteka ta oferuje wiele narzędzi i funkcji, które ułatwiają budowę systemów rozmytych oraz przeprowadzanie wnioskowania rozmytego. Scikit-fuzzy zawiera między innymi funkcje do tworzenia i operowania na zbiorach rozmytych, generowania reguł rozmytych oraz obliczania stopnia przynależności obiektów do zbiorów rozmytych.

# Środowisko programistyczne

Aby móc zacząć tworzyć system ekspertowy, konieczne będzie zainstalowanie pakietu Scikit-fuzzy (skfuzzy), a także przydatnych pakietów numpy i matplotlib, które umożliwią wykonanie kodu z instrukcji do laboratorium. Poniższe zadania zostały wykonane w edytorze kodu Visual Studio Code z zainstalowaną wtyczką Jupyter aby muc korzystać z notatnika. Python został zainstaloway w wersji 3.11.2

# Przebieg ćwiczenia

Aby zapoznać się z metodami wnioskowania rozmytego zostanie stworzony system ustalający jaki napiwek powinniśmy zostawić w restauracji po tym jak użytkownik oceni obsługę oraz jakość jedzenia. Oceny muszą być z zakresu [0-10] a pozostawiony napiwek [0-30] można zauważyć że w zakres tych liczb wchodzą wszystkie liczby rzeczywiste pomiędzy granicami zbioru

## Część I: Wykonanie przykładowego systemu ekspertowego typu Mamdaniego

W oparciu o wytyczne oraz kod z inst. Do laboratorium możemy przystąpić do wykonywania sytemu

* Import potrzebnych bibliotek, definiowanie funkcji przynależności dla wejść oraz dodanie reguł do systemu rozmytego:

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import matplotlib.pyplot as plt

obsluga = ctrl.Antecedent(np.arange(0,10.01,0.01), 'obsluga')

napiwek = ctrl.Consequent(np.arange(0,30.01,0.01), 'napiwek')

obsluga['slaba'] = fuzz.gaussmf(obsluga.universe,  0, 1.5)

obsluga['dobra'] = fuzz.gaussmf(obsluga.universe,  5, 1.5)

obsluga['wspaniala'] = fuzz.gaussmf(obsluga.universe,  10, 1.5)

napiwek['maly'] = fuzz.trimf(napiwek.universe, [0, 5, 10])

napiwek['sredni'] = fuzz.trimf(napiwek.universe, [10, 15, 20])

napiwek['duzy'] = fuzz.trimf(napiwek.universe, [20, 25,30])

regula1 = ctrl.Rule(obsluga['slaba'], napiwek['maly'])

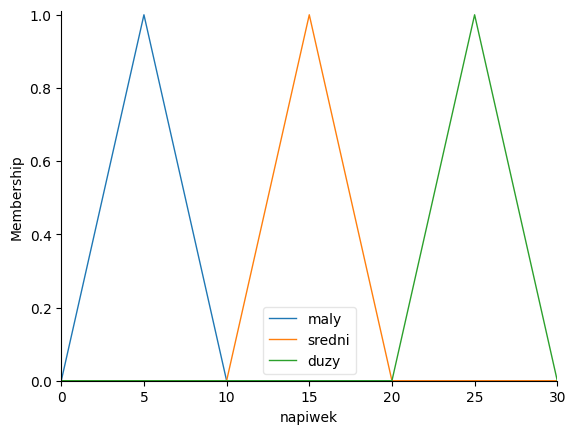
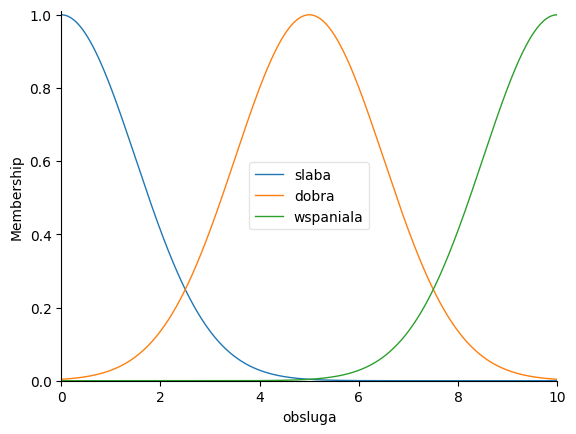
regula2 = ctrl.Rule(obsluga['dobra'], napiwek['sredni'])

regula3 = ctrl.Rule(obsluga['wspaniala'], napiwek['duzy'])

obsluga.view()

napiwek.view()

otrzymane wyniki:

* Sprawdzenie działania systemu dla wartości obsługi równej 0

napiwek\_sym.input['obsluga'] = 0

napiwek\_sym.compute()

print('Wynik',napiwek\_sym.output['napiwek'])

napiwek.view(sim=napiwek\_sym)

napiwek\_sym.input['obsluga'] = 10

napiwek\_sym.compute()

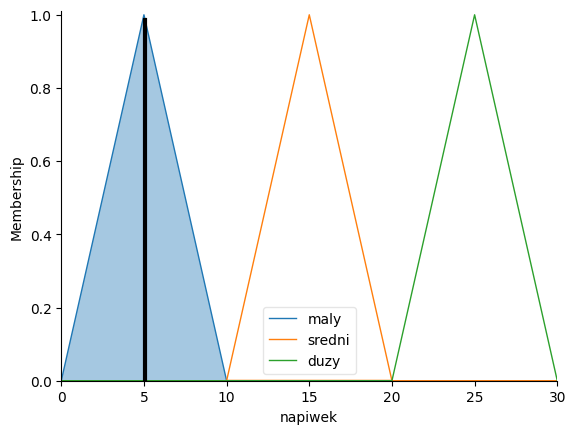
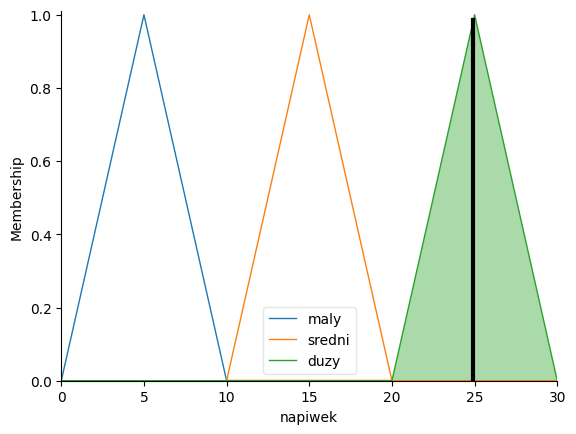
print('Wynik',napiwek\_sym.output['napiwek'])

napiwek.view(sim=napiwek\_sym)

otrzymane wyniki:

Wynik 5.076578013816947

Wynik 24.923421986182834

* Sprawdzenie działania systemu dla wartości obsługi od 0 do 10 (Rys. 4 - wykres funkcji napiwek=f(obsluga))

n\_points = 21

x = np.linspace(0, 10, n\_points)

z = np.zeros\_like(x)

for i in range(n\_points):

    napiwek\_sym.input['obsluga'] = x[i]

    napiwek\_sym.compute()

    z[i] = napiwek\_sym.output['napiwek']

fig, ax = plt.subplots()

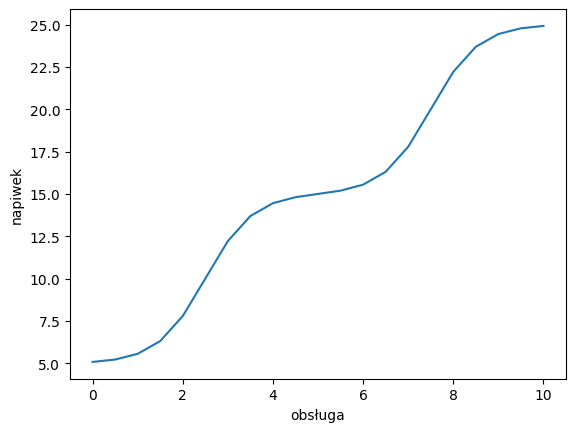
ax.set\_xlabel('obsługa')

ax.set\_ylabel('napiwek')

ax.plot(x,z)

fig.show()

otrzymane wyniki:



* Dodanie drugiej wejściowej zmiennej stanu „jedzenie” o trapezoidalnych (trapmf) zbiorach rozmytych „zepsute” oraz „wyborne”. Patrz punkt 3. Parametry zbiorów trapezoidalnych to: [-2, 0, 1, 3] oraz [7, 9, 10, 12] oraz

dodanie reguł 4 i 5 będzie odbywało się analogicznie do punktu 5 z trzeciej strony instrukcji.

jedzenie = ctrl.Antecedent(np.arange(0,10.01,0.01), 'jedzenie')

jedzenie['zepsute'] = fuzz.trapmf(jedzenie.universe,  [-2, 0, 1, 3])

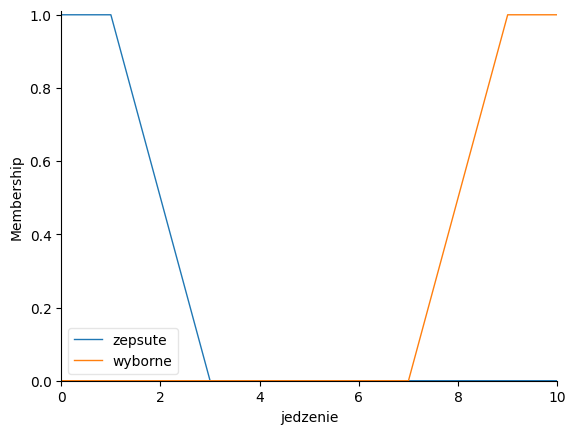
jedzenie['wyborne'] = fuzz.trapmf(jedzenie.universe, [7, 9, 10, 12])

regula4 = ctrl.Rule(jedzenie['zepsute'], napiwek['maly'])

regula5 = ctrl.Rule(jedzenie['wyborne'], napiwek['duzy'])

jedzenie.view()

otrzymane wyniki:



* Sprawdzenie działania systemu dla wartości obsługi i jedzenia od 0 do 10 (Rys. 5 - wykres funkcji napiwek=f(obsluga, jedzenie))

n\_points = 21

upsampled = np.linspace(0, 10, n\_points)

x, y = np.meshgrid(upsampled, upsampled)

z = np.zeros\_like(x)

# Loop through the system 21\*21 times to collect the control surface

for i in range(n\_points):

    for j in range(n\_points):

        napiwek\_sym.input['obsluga'] = x[i, j]

        napiwek\_sym.input['jedzenie'] = y[i, j]

        napiwek\_sym.compute()

        z[i, j] = napiwek\_sym.output['napiwek']

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(111, projection='3d')

surf = ax.plot\_surface(x, y, z, cmap='viridis')

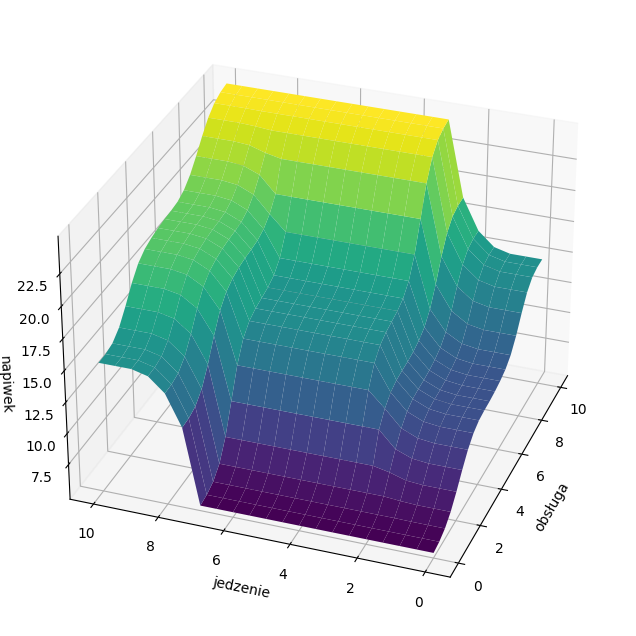
ax.set\_xlabel('obsługa')

ax.set\_ylabel('jedzenie')

ax.set\_zlabel('napiwek')

ax.view\_init(30, 200)

otrzymane wyniki:



## Część II. Modyfikacja systemu

Wadami tego sytemu jest to że nie możliwe aby dał odpowiedź mniejszą niż 5.076… oraz większą niż 24.923… dlatego zostały poczynione pewne zamiany w programie aby to umożliwić jedną z opcji jest skorzystanie z singletona rozmytego i wstwienie go w granicach przenidzłu wartości dla funkcji napiwek mały oraz duży w odpowiednio [0,0,0] oraz [30,30,30]. Poniżysz przykład pokazuje jak można osiągnąć ten sam efekt bez użycia z singletona rozmytego. Należy tak zdefiniować funkcje napiwków tak aby kiedy podamy 0 i 0(obsługa, jezdnie) na wejście to środek ciężkości był w 0 (napiwek) analogicznie dla wartości 10 i 01 to wynik 30.

Dokonane zmienny w kodzie z części pierwszej:

napiwek = ctrl.Consequent(np.arange(-10,40.01,0.01), 'napiwek')

napiwek['maly'] = fuzz.trimf(napiwek.universe, [-10, 0, 10])

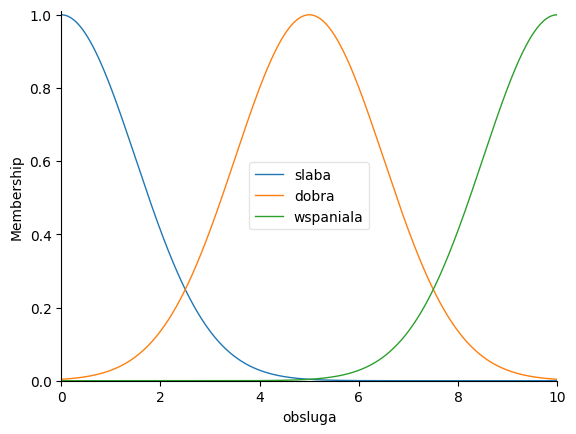
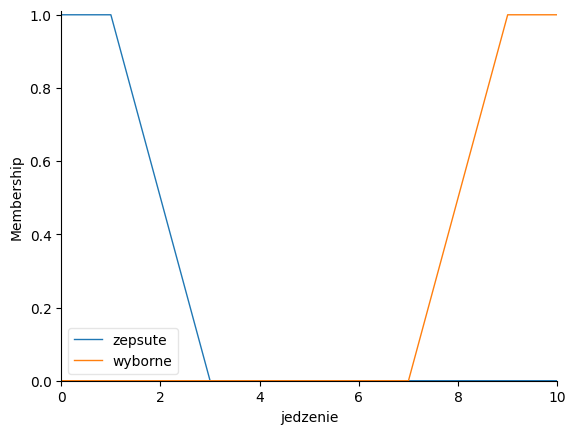
napiwek['sredni'] = fuzz.trimf(napiwek.universe, [10, 15, 20])

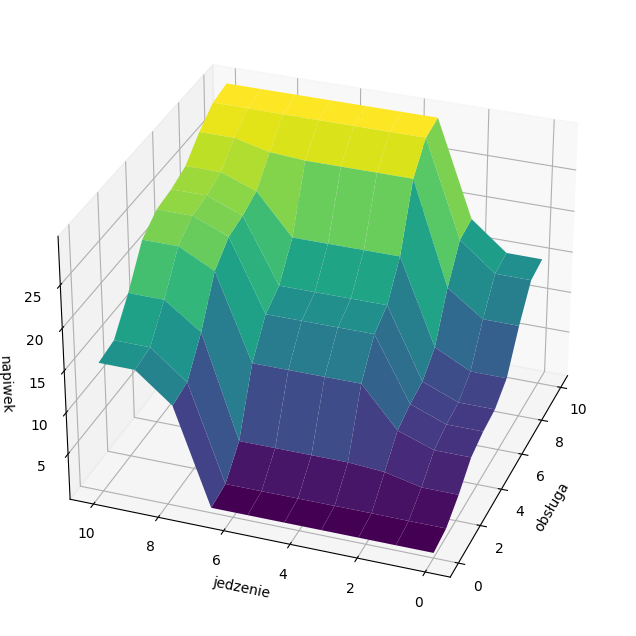
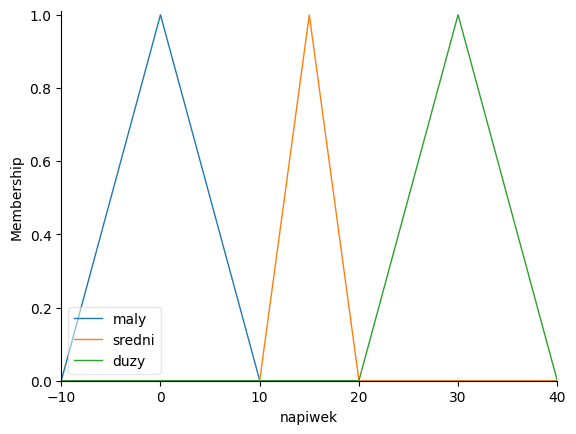
napiwek['duzy'] = fuzz.trimf(napiwek.universe, [20, 30,40])

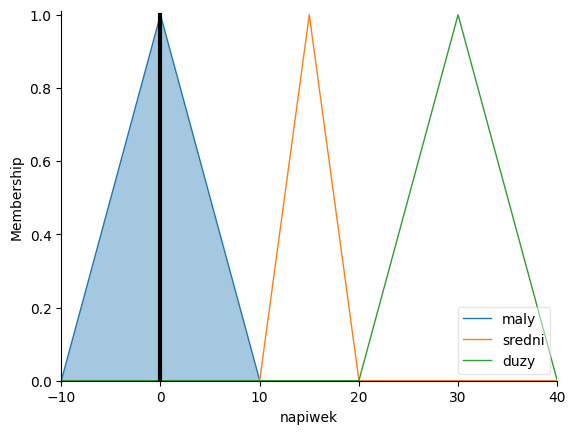
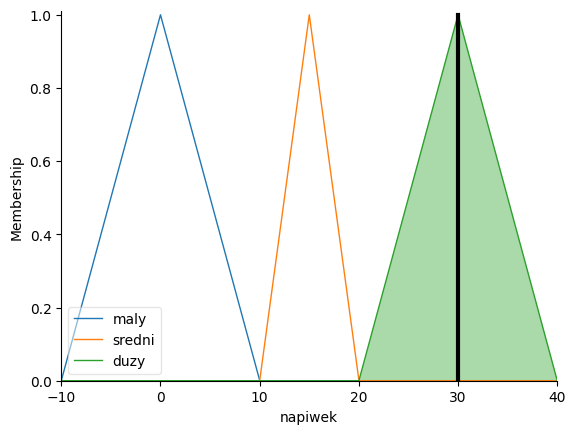
otrzymane wyniki:

Niejmniejszy możliwy otrzymany napiwek: 0.057654269459063504

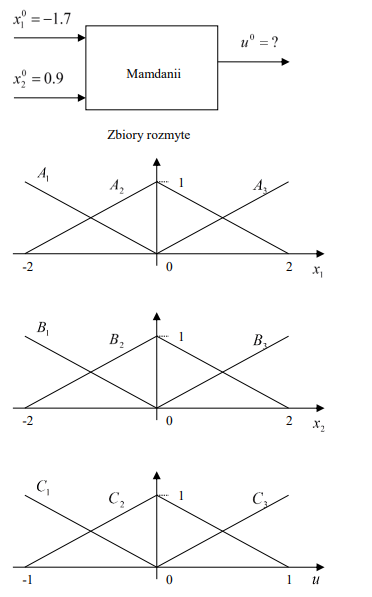
Największy możliwy otrzymany napiwek: 29.942345730540943

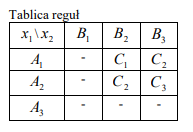
 



## Część III: Przykładowe zadanie zaliczeniowe





import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import matplotlib.pyplot as plt

AA = ctrl.Antecedent(np.arange(-2,2.01,0.01), 'AA')

BB = ctrl.Antecedent(np.arange(-2,2.01,0.01), 'BB')

CC = ctrl.Consequent(np.arange(-1,1.01,0.01), 'CC')

AA['A\_1'] = fuzz.trapmf(AA.universe, [-5,-4,-2,0])

AA['A\_2'] = fuzz.trimf(AA.universe, [-2,0,2])

AA['A\_3'] = fuzz.trapmf(AA.universe, [0,2,4,5])

BB['B\_1'] = fuzz.trapmf(BB.universe, [-5,-4,-2,0])

BB['B\_2'] = fuzz.trimf(BB.universe, [-2,0,2])

BB['B\_3'] = fuzz.trapmf(BB.universe, [0,2,4,5])

CC['C\_1'] = fuzz.trapmf(CC.universe, [-5,-4,-1,0])

CC['C\_2'] = fuzz.trimf(CC.universe, [-1,0,1])

CC['C\_3'] = fuzz.trapmf(CC.universe, [0,1,4,5])

AA.view()

BB.view()

CC.view()

regula1 = ctrl.Rule(AA['A\_1'] & BB['B\_2'], CC['C\_1'])

regula2 = ctrl.Rule(AA['A\_1'] & BB['B\_3'], CC['C\_2'] )

regula3 = ctrl.Rule(AA['A\_2'] & BB['B\_2'], CC['C\_2'] )

regula4 = ctrl.Rule(AA['A\_2'] & BB['B\_3'], CC['C\_3'] )

CC\_ctr = ctrl.ControlSystem([regula1,regula2,regula3,regula4,])

CC\_sym = ctrl.ControlSystemSimulation(CC\_ctr)

CC\_sym.input['AA']= -1.7

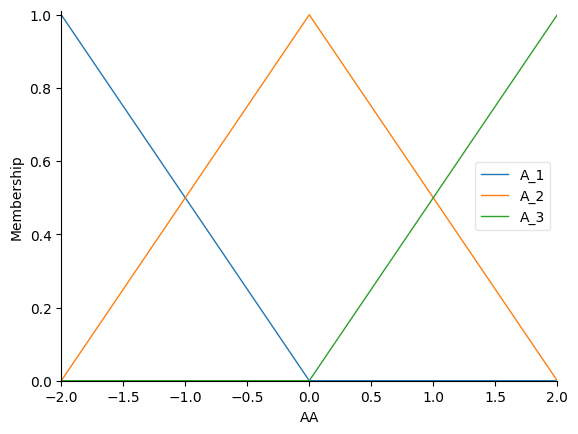
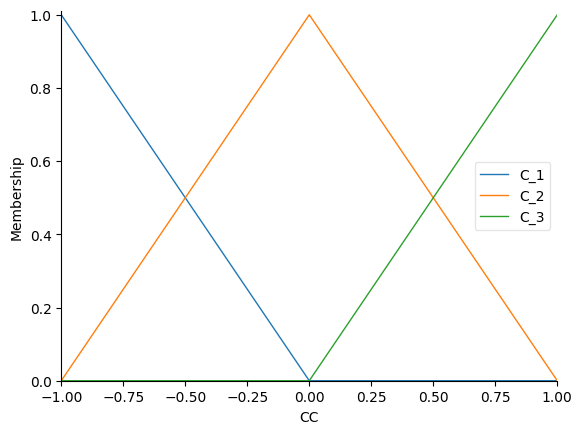
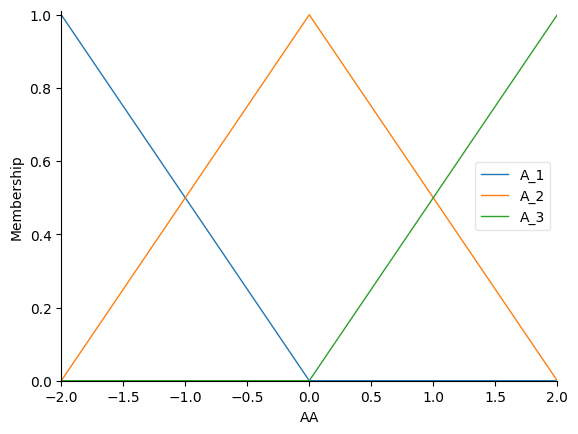
CC\_sym.input['BB']= 0.9

CC\_sym.compute()

print("output: ", CC\_sym.output['CC'])

CC.view(sim=CC\_sym)

otrzymane wyniki:

output: -0.1312015503875969

