POLITECHNIKA RZESZOWSKA im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI POLITECHNIKI RZESZOWSKIEJ

14.03.2023

Sztuczna Inteligencja

Sprawozdanie nr.1

Temat: Synteza układu wnioskującego

Prowadzący:

dr hab. inż. Roman Zajdel, prof. PRz

Wykonali:

2EF-DI, L8

Daniel Kleczyński

Spis treści

1. 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.

2. 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

3. 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

3.1. Część I: Wykonanie przykładowego systemu ekspertowego typu Mamdaniego

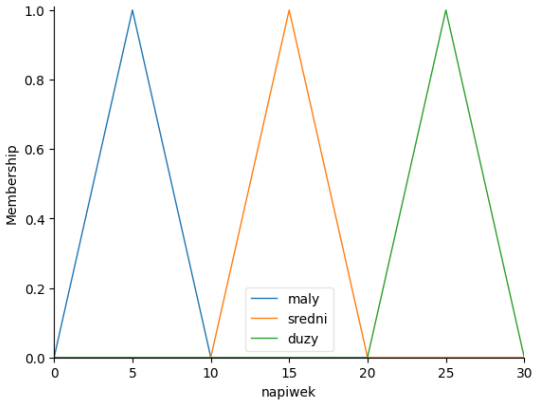
W oparciu o wytyczne oraz kod z inst. Do laboratorium można przystąpić do wykonywania sytemu

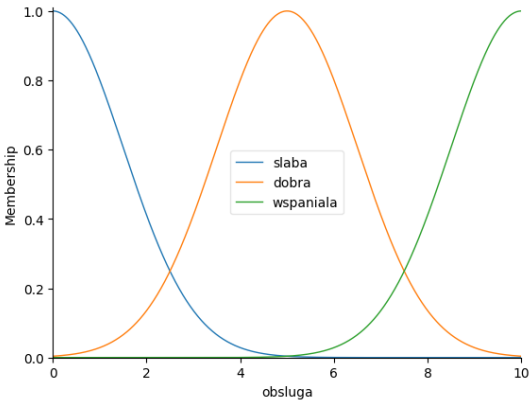
- Import potrzebnych bibliotek, definiowanie funkcji przynależności dla wejść oraz dodanie reguł do systemu rozmytego:

```
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()
```

Kod 1 Kod źródłowy

otrzymane wyniki:





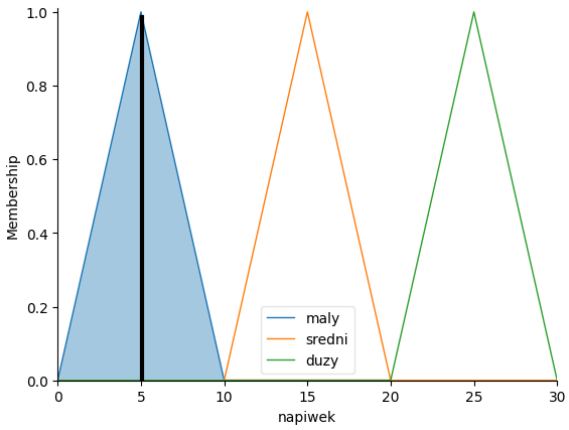
- Sprawdzenie działania systemu dla wartości obsługi równej 0

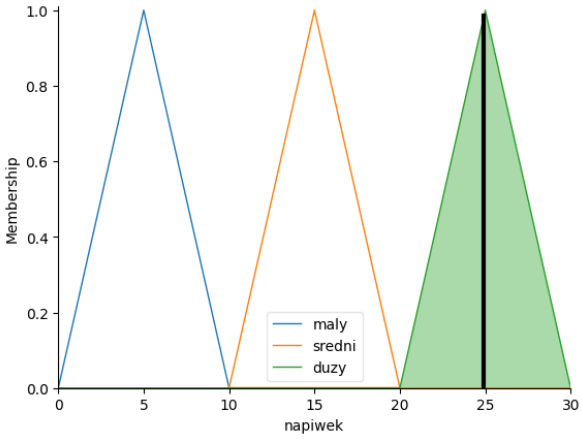
```
napiwek_sym.input['obsluga'] = 0

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)
```

Kod 2 Kod źródłowy

otrzymane wyniki:

```
Wynik 5.076578013816947
Wynik 24.923421986182834
```



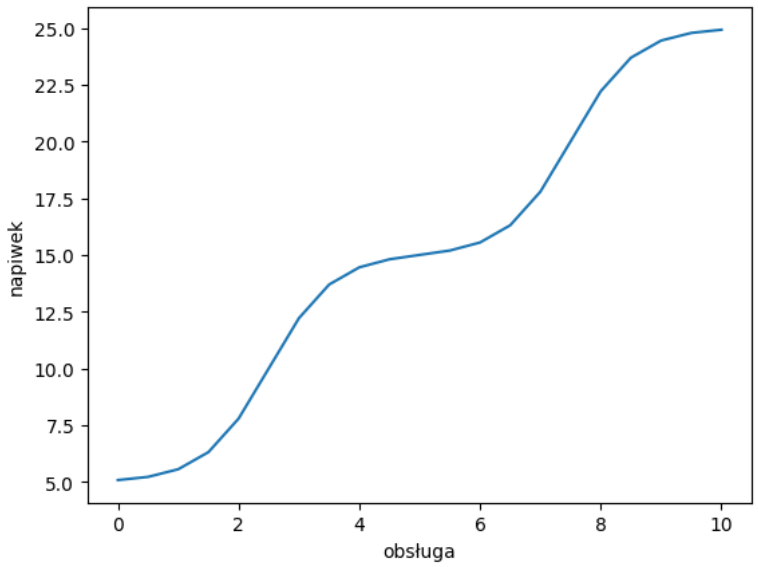


- Sprawdzenie działania systemu dla wartości obsługi od 0 do 10 (Rys. 4 - wykres funkcji napiwek=f(obsluga))

```
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()
```

Kod 3 Kod źródłowy

otrzymane wyniki:

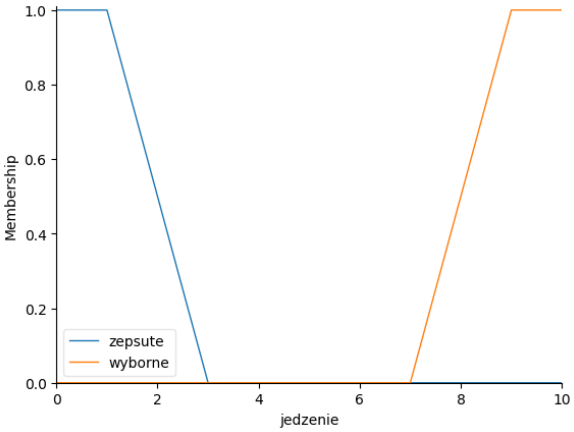


- Dodanie drugiej wejściowej zmiennej stanu „jedzenie” o trapezoidalnych (trapmf) zbiorach rozmytych „zepsute” oraz „wyborne”. Patrz punkt 3. Parametry zbiorów trapezoidalnych to: [-2, 0, 1, 3] oraz [7, 9, 10, 12] oraz dodanie reguł 4 i 5 będzie odbywało się analogicznie do punktu 5 z trzeciej strony 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()
```

Kod 4 Kod źródłowy

Otrzymane wyniki:

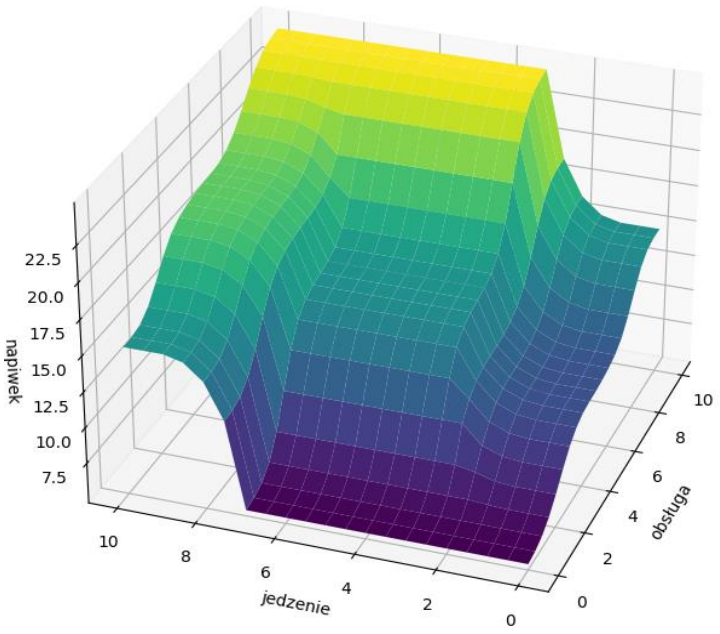


- Sprawdzenie działania systemu dla wartości obsługi i jedzenia od 0 do 10 (Rys. 5 - wykres funkcji napiwek=f(obsluga, jedzenie))

```
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)
```

Kod 5 Kod źródłowy

otrzymane wyniki:



- Wnioski:

Został stworzony system do wystawiania napiwku można już nawet na prostym przykładzie jak praktyczne mogą być systemy oparte na zbiorach rozmytych. Rozwiązuje nam ona nieliniowe problemy takie jak wtedy kiedy np. ocena jakości obsługi użytkownik nie musiał być zdecydowany czy obsługa jest wyśmienita czy dobra mógł wystawić dowolną ocenę z przedziału 1 do 10 7,9996666666… a system potrafi sobie z tym poradzić czego nie potrafią „system boolowskich" gdzie dla danego wyjścia muszą zostać podane na wejścia konkretne wartości które wcześniej zostały zdefiniowane w przypadku logiki rozmytej podjemy tylko krańcowe wartości a wszystko co jest pomiędzy robi za nas system Magnaniego. Co warto zauważyć logika rozmyta jest zgodna z logiką boolowską w punktach gdzie ona występuje a ponad to aproksymuje nieskończenie wiele wartości pomiędzy punktami końcowymi.

3.2. 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.

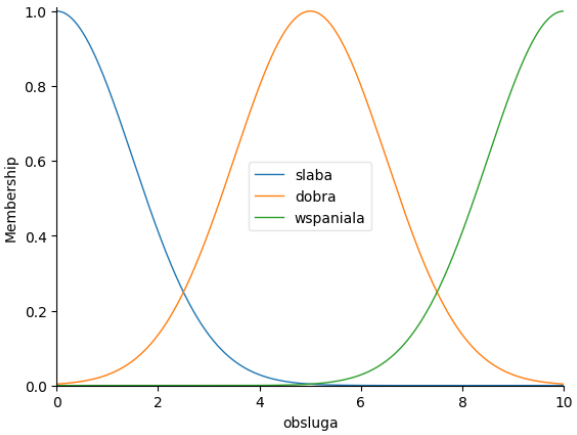
Dokonane zmienny w kodzie z części pierwszej:

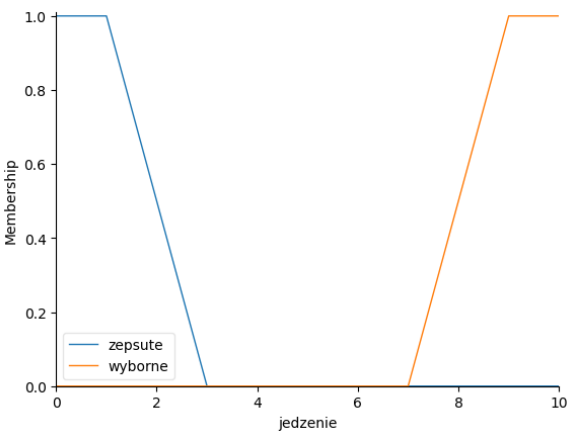
```
napiwek = ctrl.Consequent(np.arange(-10,40.01,0.01), 'napiwek')

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])
```

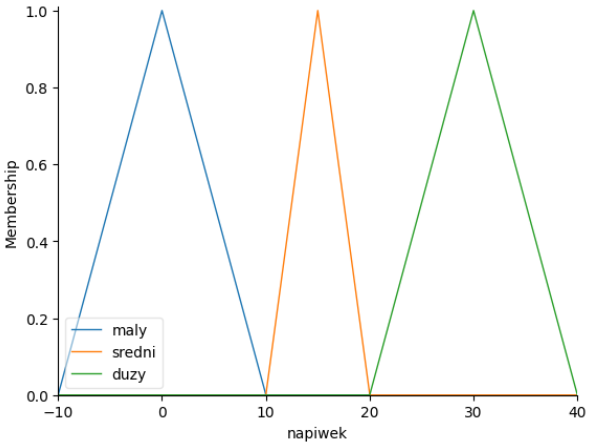
Kod 6 Kod źródłowy

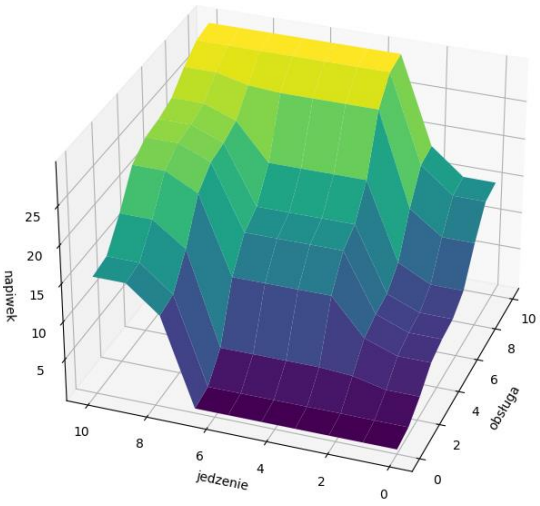
otrzymane wyniki:

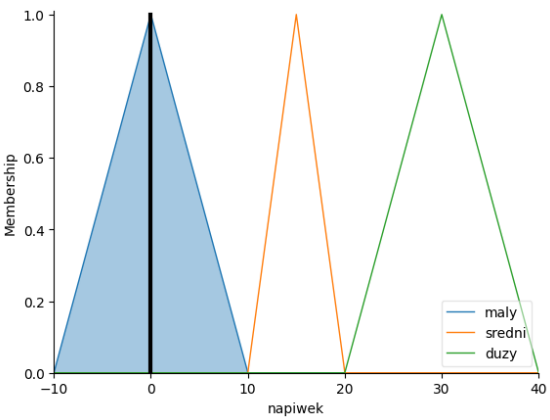
```
Niejmniejszy możliwy otrzymany napiwek:  0.057654269459063504
Największy możliwy otrzymany napiwek:  29.942345730540943
```

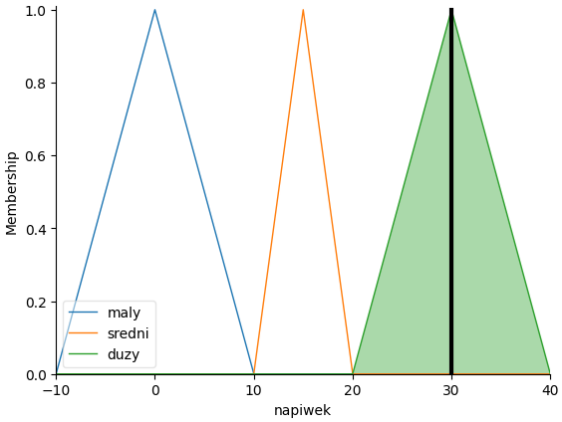

Membership
obsluga
slaba
dobra
wspaniala


Membership
jedzenie
zepsute
wyborne


Membership
napiwek
maly
sredni
duzy


napiwek
jedzenie
obsluga

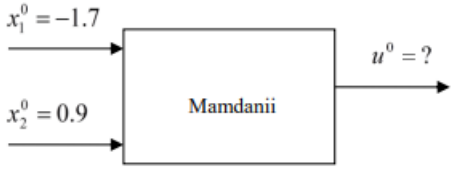

Membership
napiwek
maly
sredni
duzy


Membership
napiwek
maly
sredni
duzy

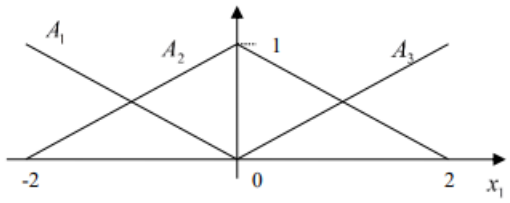
- Wnioski:

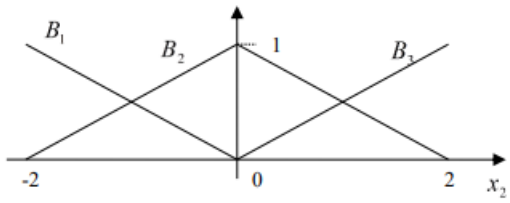
Modyfikacja kodu nie była bardzo skomplikowana a ponad został uzyskany satysfakcjonujący wynik w postaci skrajnych odpowiedzi systemy ~0 oraz ~30 przy czym nie został stracony kształt powierzchni wyjścia. Niestety z logicznego punktu patrzenia na ten przykład jest możliwe zostawienie ujemnego napiwku lub napiwku wykraczającego po za nasz zakres (np. -10, 40) lecz jeśli ale nie jest to możliwe ponieważ wartości wejściowe [0 – 10] nie mogą spowodować tego aby na wyjściu pojawiły się wartości <0 lub >30 więc praktyczność systemu została zachowana.

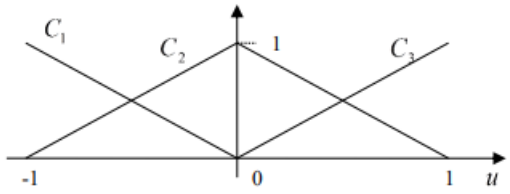
3.3. Część III: Przykładowe zadanie zaliczeniowe



Zbiory rozmyte







Tablica reguł

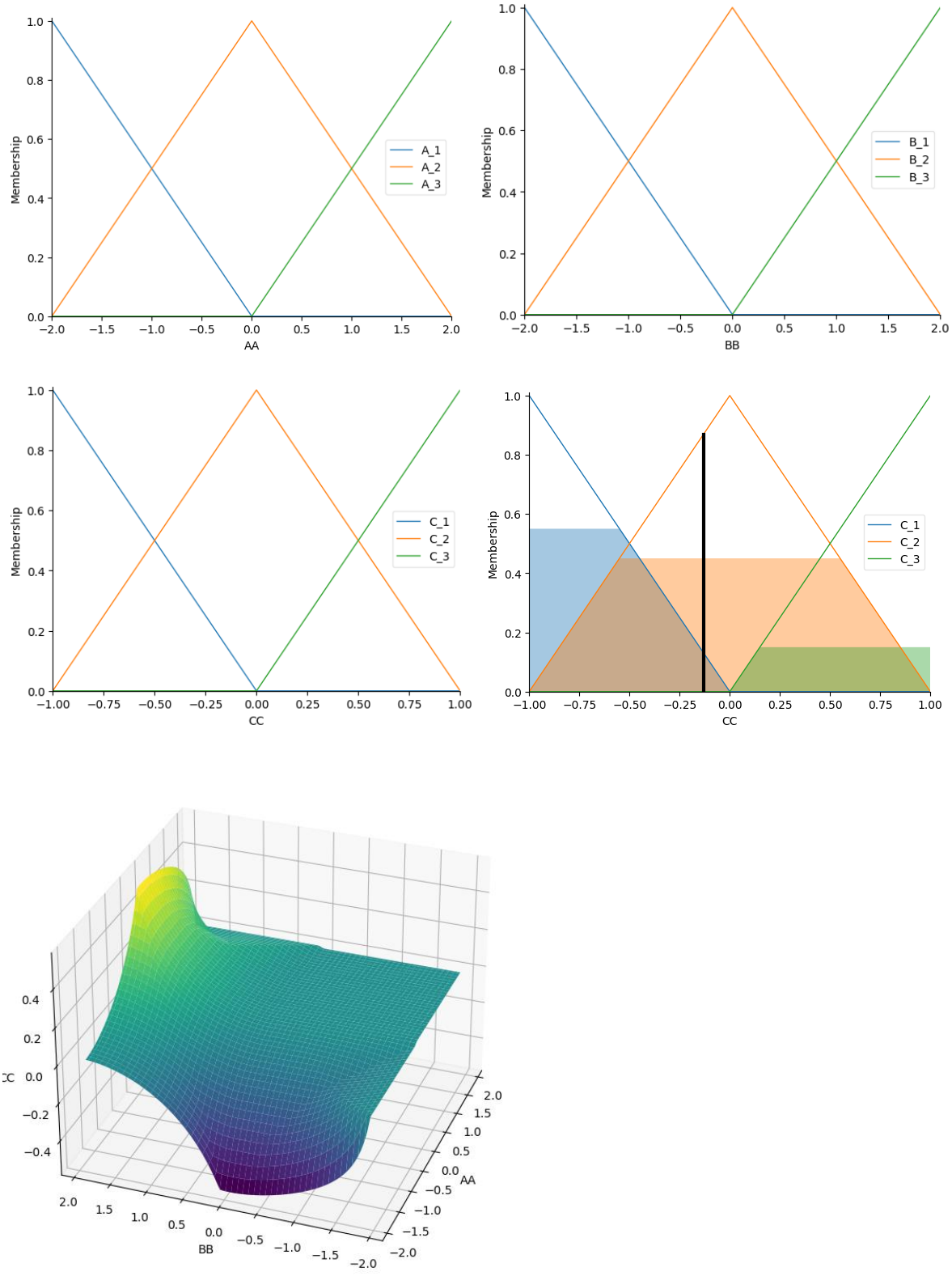
$x_1 \setminus x_2$	B_1	B_2	B_3
A_1	-	C_1	C_2
A_2	-	C_2	C_3
A_3	-	-	-

```
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)
```

Kod 7 Kod źrudłowy

Otrzymane wyniki:

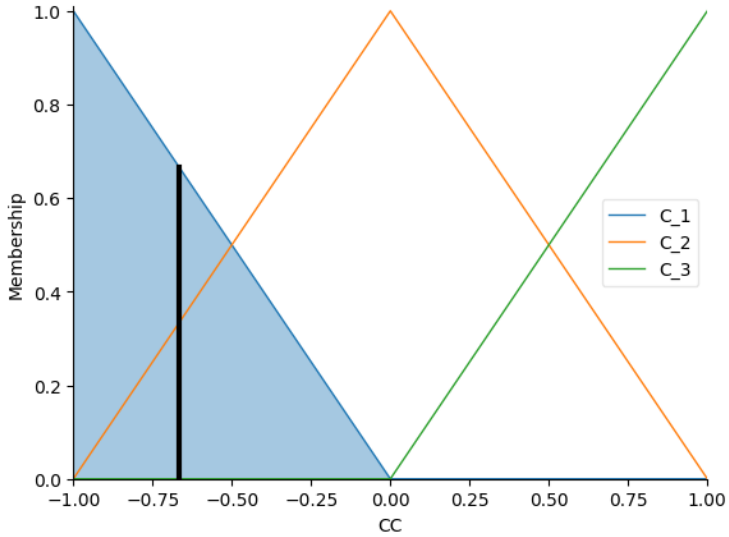
```
output:  -0.1312015503875969
```



Wnoski:

Tak jak widać na powyższym przykładzie udało się rozwiązać problem postawiony problem co udowadnia wartość wyjścia dla wejścia równego AA-1.7 BB 0.9 która powinna według instrukcji wynosić -0.131. Można również pokusić się o analizę powierzchni wyjścia gdzie min całego modelu jest wtedy kiedy mamy BB = 0 oraz AA =-2 może to utwierdzać o poprawności działania modelu ponieważ wtedy przynależności do B_2 oraz A_1 a według tabelki tylko wtedy otrzymujemy C_1 a to ono jest zbiorem gdzie mamy najmniejsze wartości :

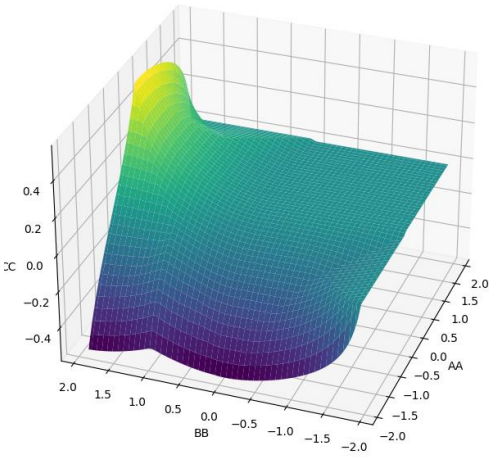
output: -0.6666666666666663



Aby aby powstało kolejne mnimum musiała by być doadna koljna regóła której następnikiem będzie C_1 przykład:

Zmioana reguły 2

```
regula2 = ctrl.Rule(AA['A_1'] & BB['B_3'], CC['C_1'] )
```



A to spowodowało powstanie drugiego minimum

4. Wykonanie ćwiczeń w innym języku (MATLAB).

4.1. Część I

KOD:

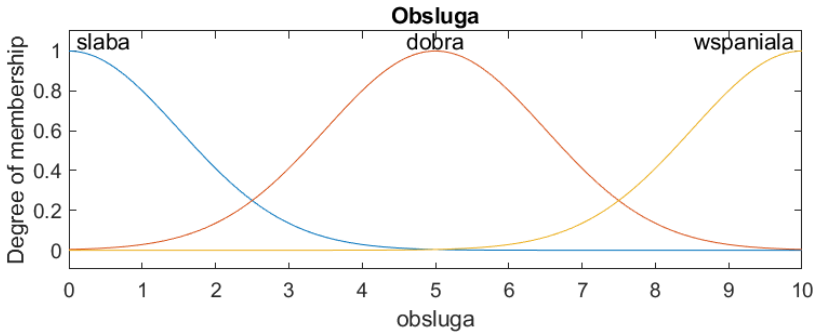
```
fis = mamfis();
fis = addInput(fis,[0 10],'Name','obsluga');
fis = addInput(fis,[0 10],'Name','jedzenie');

fis = addMF(fis,'obsluga','gaussmf',[1.5 0],'Name','slaba');
fis = addMF(fis,'obsluga','gaussmf',[1.5 5],'Name','dobra');
fis = addMF(fis,'obsluga','gaussmf',[1.5 10],'Name','wspaniala');

fis = addMF(fis,'jedzenie','trapmf',[-2 0 1 3],'Name','zepsute');
fis = addMF(fis,'jedzenie','trapmf',[7 9 10 12],'Name','wyborne');

fis = addOutput(fis,[0 30],'Name','napiwek');
fis = addMF(fis,'napiwek','trimf',[0 5 10],'Name','maly');
fis = addMF(fis,'napiwek','trimf',[10 15 20],'Name','sredni');
fis = addMF(fis,'napiwek','trimf',[20 25 30],'Name','duzy');

rule1 = "if obsluga is slaba then napiwek is maly";
rule2 = "if obsluga is dobra then napiwek is sredni";
rule3 = "if obsluga is wspaniala then napiwek is duzy";
rule4 = "if jedzenie is zepsute then napiwek is maly";
rule5 = "if jedzenie is wyborne then napiwek is duzy";
rules = [rule1; rule2; rule3; rule4; rule5];
fis = addRule(fis,rules);

figure;
subplot(2,1,1);
plotmf(fis,'input',1);
title('Obsluga');
subplot(2,1,2);
plotmf(fis,'input',2);
title('Jedzenie');
figure;
plotmf(fis,'output',1);
title('Napiwek');
figure;
gensurf(fis);
title('powierzchnia wyjścia systemu');

obsluga_val = 0;
jedzenie_val = 0;
input_vals = [obsluga_val, jedzenie_val];
napiwek_val = evalfis(input_vals, fis);
disp(['Wartość napiwku Min: ', num2str(napiwek_val)]);

obsluga_val = 10;
jedzenie_val = 10;

input_vals = [obsluga_val, jedzenie_val];
napiwek_val = evalfis(input_vals, fis);
disp(['Wartość napiwku Max: ', num2str(napiwek_val)]);
```

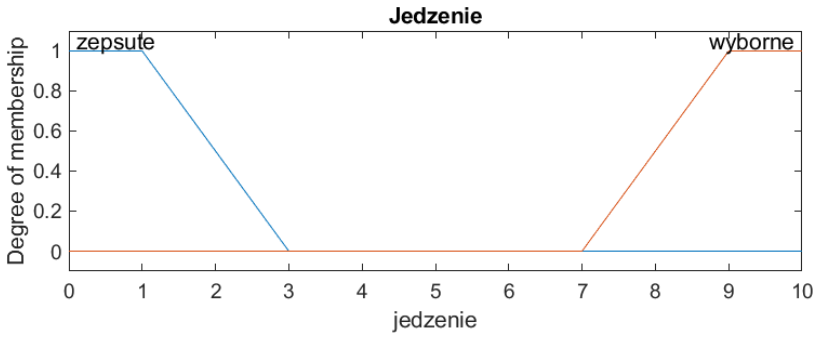
Kod 8 Kod źródłowy

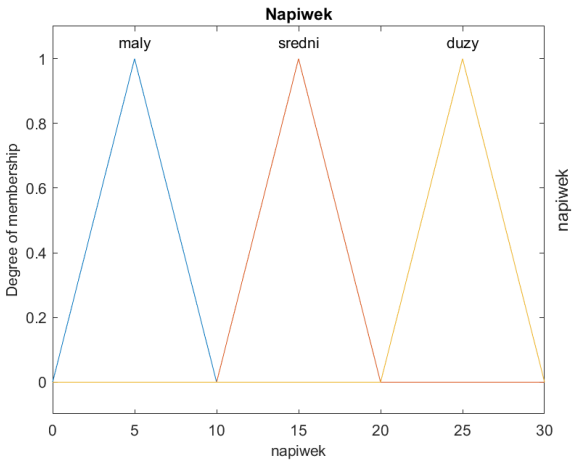
otrzymane wyniki:

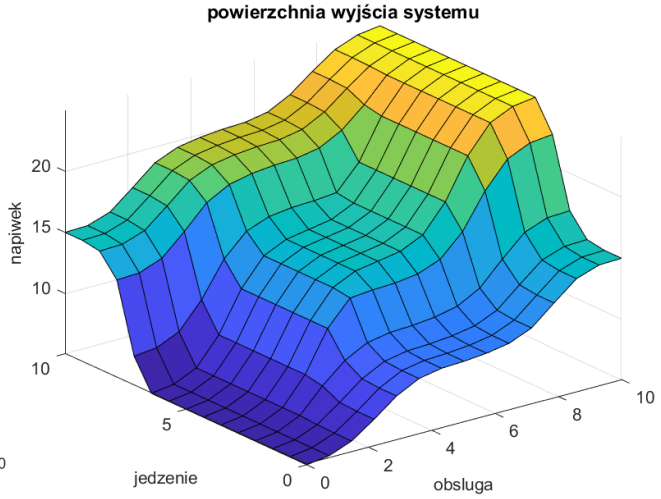
Wartość napiwku Max: 24.922

Wartość napiwku Min: 5.078









4.2. Część II

Dokonane zmiany w Kodzie :

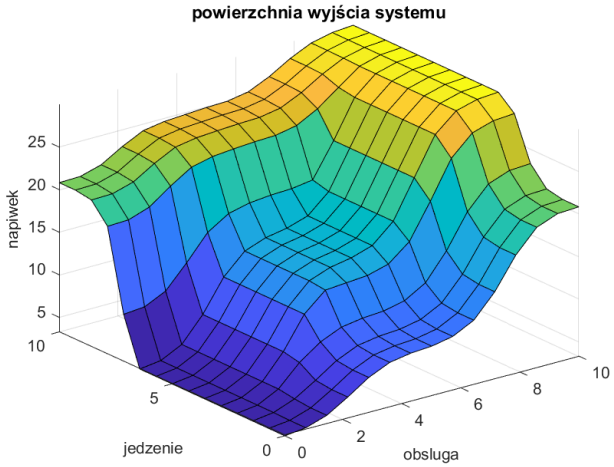
```
fis = addOutput(fis,[0 40],'Name','napiwek');
fis = addMF(fis,'napiwek','trimf',[-10 0 10],'Name','maly');
fis = addMF(fis,'napiwek','trimf',[10 15 20],'Name','sredni');
fis = addMF(fis,'napiwek','trimf',[20 30 40],'Name','duzy');
```

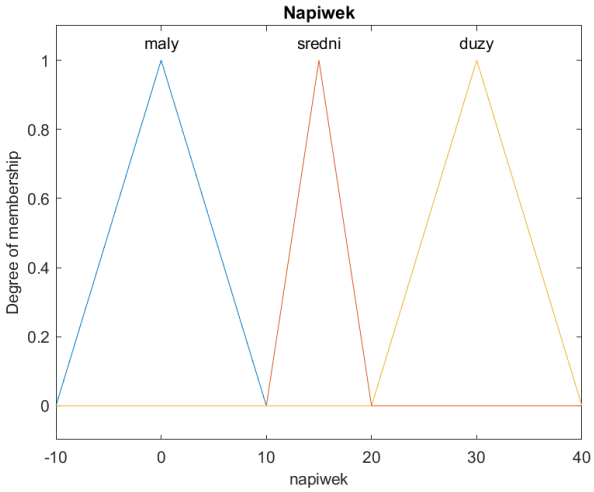
Kod 9 Kod źródłowy

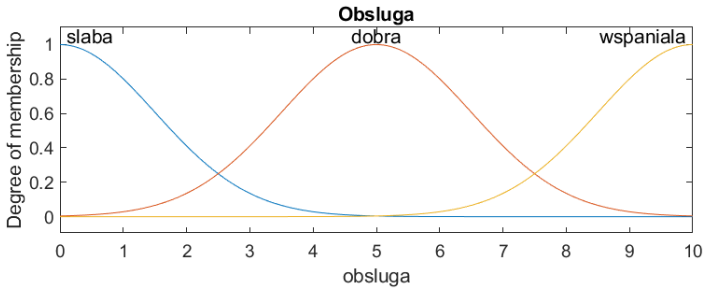
Otrzymane wyniki:

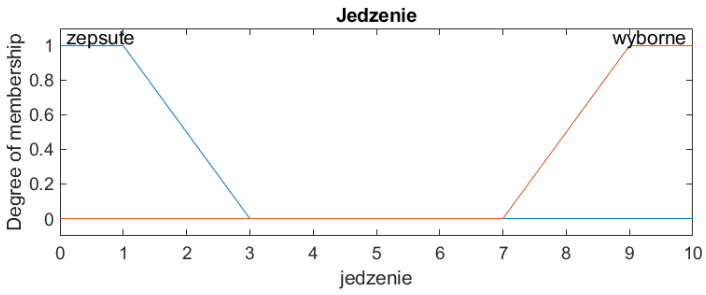
Wartość napiwku Min: 0.054888

Wartość napiwku Max: 29.9451









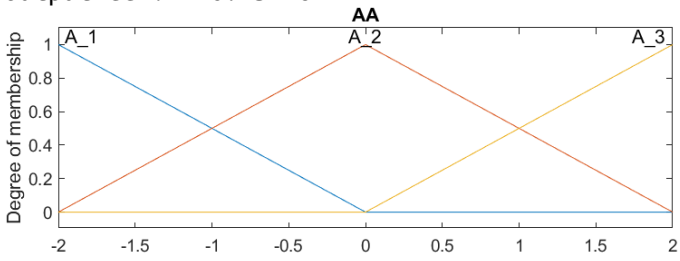
4.3. Część III

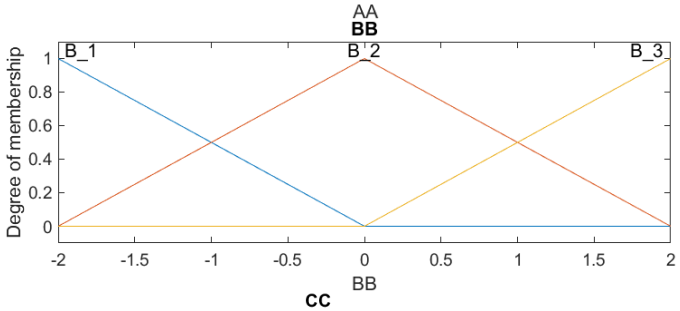
KOD:

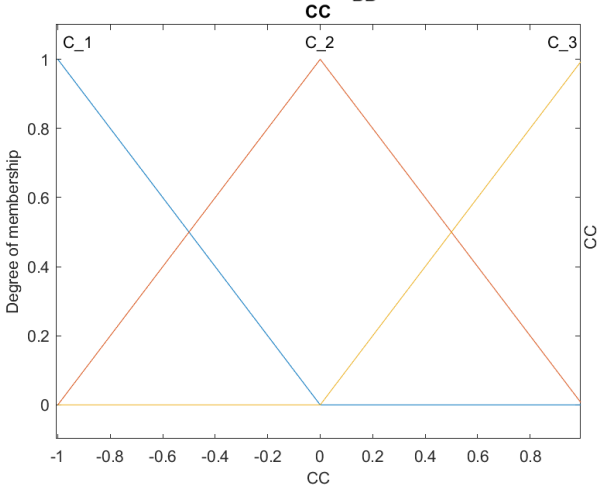
```
fis = mamfis();

fis = addInput(fis,[-2 2],'Name','AA');
fis = addInput(fis,[-2 2],'Name','BB');

fis = addMF(fis,'AA','trapmf',[-5 -4 -2 0],'Name','A_1');
fis = addMF(fis,'AA','trimf',[-2 0 2],'Name','A_2');
fis = addMF(fis,'AA','trapmf',[0 2 4 5],'Name','A_3');

fis = addMF(fis,'BB','trapmf',[-5 -4 -2 0],'Name','B_1');
fis = addMF(fis,'BB','trimf',[-2 0 2],'Name','B_2');
fis = addMF(fis,'BB','trapmf',[0 2 4 5],'Name','B_3');

fis = addOutput(fis,[-1 1],'Name','CC');
fis = addMF(fis,'CC','trapmf',[-5 -4 -1 0],'Name','C_1');
fis = addMF(fis,'CC','trimf',[-1 0 1],'Name','C_2');
fis = addMF(fis,'CC','trapmf',[0 1 4 5],'Name','C_3');

rule1 = "If AA is A_1 and BB is B_2  then CC is C_1";
rule2 = "If AA is A_1 and BB is B_3  then CC is C_2";
rule3 = "If AA is A_2 and BB is B_2  then CC is C_2";
rule4 = "If AA is A_2 and BB is B_3  then CC is C_3";
rules = [rule1; rule2; rule3; rule4]

fis = addRule(fis,rules);

figure;
subplot(2,1,1);
plotmf(fis,'input',1);
title('AA');
subplot(2,1,2);
plotmf(fis,'input',2);
title('BB');
figure;
plotmf(fis,'output',1);
title('CC');
figure;
gensurf(fis);
title('Surface of outpout');

AA = -1.7;
BB = 0.9;

input_vals = [AA, BB];
CC_vals = evalfis(input_vals, fis);
disp(['Output CC :  ', num2str(CC_vals)]);
```

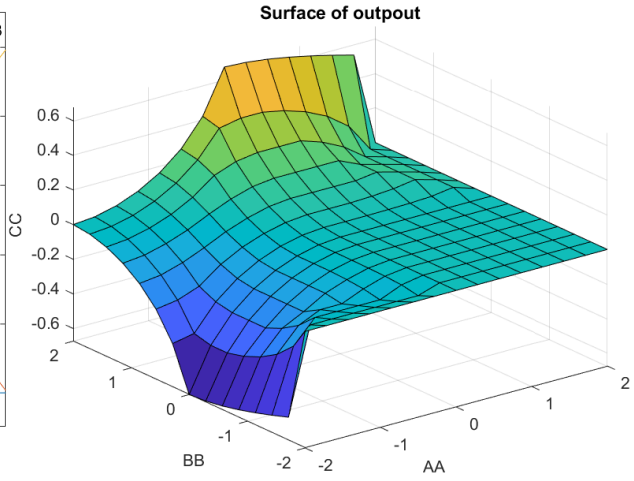
Kod 10 Kod źródłowy

```
Otrzymane Wyniki:

Output CC :  -0.13476
```









5. Wnioski Ogólne

Logika rozmyta pozwala na opisywanie problemów, które nie mają jednoznacznych i precyzyjnych rozwiązań, a które dotyczą takich dziedzin jak np. ocena ryzyka, sterowanie procesami, planowanie i podejmowanie decyzji. W porównaniu z logiką klasyczną, logika rozmyta pozwala na bardziej elastyczne i realistyczne modelowanie rzeczywistych sytuacji.

Zalety implementacji za pomocą Pythona i Matlaba obejmują ich prostotę i elastyczność, co umożliwia łatwe tworzenie i testowanie różnych modeli. Obie platformy oferują również szeroką gamę narzędzi do analizy danych, co ułatwia projektowanie i testowanie modeli opartych na logice rozmytej.

Plusy Matlaba:

- Prostota i intuicyjność - Matlab jest znany ze swojej prostoty i intuicyjności. Posiada on wiele wbudowanych funkcji i narzędzi, co ułatwia pracę z logiką rozmytą.
- Bogate biblioteki - Matlab ma wiele wbudowanych bibliotek, które ułatwiają implementację i testowanie algorytmów logiki rozmytej.
- Wysoka wydajność - Matlab jest zoptymalizowany pod kątem wydajności, co czyni go idealnym narzędziem do pracy z dużymi zbiorami danych.

Minusy Matlaba:

- Koszt - Matlab jest oprogramowaniem komercyjnym, co oznacza, że jego cena może być dość wysoka, co stanowi duże obciążenie dla mniejszych projektów i jednostek naukowych.
- Złożoność - Matlab, pomimo swojej prostoty, może być czasem złożony i wymaga od użytkowników nauki specyficznej składni i kodowania.
- Brak dostępności na wielu platformach - Matlab jest dostępny tylko na określonych platformach, co może utrudnić jego wykorzystanie w niektórych projektach.

Plusy Pythona:

- Darmowy i otwarty kod źródłowy - Python jest wolnym oprogramowaniem, co oznacza, że jego kod źródłowy jest otwarty i dostępny dla wszystkich, co stanowi znaczną przewagę nad Matlaba w kwestii dostępności dla mniejszych projektów i jednostek naukowych.
- Duża społeczność - Python ma ogromną społeczność programistów, którzy tworzą i udostępniają wiele bibliotek i narzędzi, co ułatwia pracę z logiką rozmytą.
- Uniwersalność - Python jest uniwersalnym językiem programowania, co oznacza, że można go stosować w wielu dziedzinach, nie tylko w logice rozmytej.

Minusy Pythona:

- Wydajność - Python jest językiem interpretowanym, co oznacza, że jest mniej wydajny niż Matlab i może być wolniejszy w przetwarzaniu dużych zbiorów danych.

- Brak wbudowanych bibliotek - Python nie ma tak dużego zbioru wbudowanych bibliotek, jak Matlab, co oznacza, że użytkownicy muszą pobierać i instalować dodatkowe biblioteki, aby móc pracować z logiką rozmytą.
- Trudniejsze debugowanie - Python, ze względu na swoją interpretowaną naturę, może być trudniejszy w debugowaniu.

Ciekawe zastosowania logiki rozmytej.

- Sterowanie temperaturą w procesach przemysłowych
- Sterowanie pojazdami autonomicznymi: Systemy rozmytego sterowania są szeroko stosowane w samochodach autonomicznych, robotach i dronach. Logika rozmyta pozwala na skuteczne sterowanie pojazdami.
- Diagnostyka medyczna: Logika rozmyta mogą być wykorzystane w diagnostyce medycznej do oceny stanu zdrowia pacjenta i przewidywania jego dalszego rozwoju. Systemy te pozwalają na analizę wielu parametrów medycznych jednocześnie, co zwiększa skuteczność diagnostyki i leczenia.
- Systemy kontroli ruchu w miastach: Logika rozmyta jest szeroko stosowana w systemach kontroli ruchu w miastach, takich jak sterowanie sygnalizacją świetlną czy sterowanie ruchem pieszych. Systemy te pozwalają na precyzyjne i skuteczne sterowanie ruchem w zatłoczonych miejscach, co przyczynia się do poprawy bezpieczeństwa i przepływu ruchu.
- Systemy przetwarzania języka naturalnego: Logika rozmyta i systemy rozmytego sterowania mogą być wykorzystane do przetwarzania języka naturalnego, takiego jak rozpoznawanie mowy czy tłumaczenie maszynowe.