Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №2

з дисципліни «Методи та технології штучного інтелекту»

на тему

«Моделювання функції з двох змінних засобами нечіткої математики»

Виконала:
студентка групи ІС-12
Павлова Софія

Викладач:
Шимкович В. М.

Київ – 2023

1. Постановка задачі

Мета: Промоделювати засобами нечіткої логіки функцію з двох змінних. Провести дослідження впливу форми функції приналежності на якість моделювання.

Завдання:

1. Побудувати нечітку модель функції двох змінних згідно з варіантом. Модель функції має містити 2 входи і 1 вихід.
2. Вказати діапазони вхідних і вихідних змінних, додаючи обрану кількість функцій приналежності (6 для вхідних і 9 для вихідних змінних).
3. Заповнити базу знань моделі правилами. Так як для входів у нас по 6 функцій приналежності, то комбінацій: 6*6= 36 правил.
4. Оцінити точність моделювання.
5. Дослідити вплив форми функції приналежності на якість моделювання (порівняти помилки моделювання).
6. Дослідити можливість зменшення числа правил за рахунок виключення деяких (перевірити достатність використання правил, що представляють тільки діагональ таблиці).

9.	$y = 0.2 \cdot \sin(3x) \cdot x^2$ $z = \sin\lvert x\rvert \cdot \sin(x + y)$	9.	22.	13.

Рисунок 1 – завдання за варіантом № 22

2. Виконання

Діапазони вхідних та вихідних даних:

Вкажемо діапазони вхідних та вихідних даних та візуалізуємо функції. Розглянемо на вхід значення *x* від 0 до 6.

Лістинг:

```
import pandas as pd
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl

# Функція для виведення графіків вхідних даних
def plot(df, df_1, title, label, label_1, xlabel, ylabel):
    plt.figure(figsize=(8, 6))
    if label != label_1:
        plt.plot(df_1, label=label_1)
    plt.plot(df, label=label)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.legend()
    plt.grid(True)
    plt.title(title)
    plt.show()

# Діапазон значень x
x_values = np.arange(0, 6, 0.1)
# Значення y
y_values = 0.2 * np.sin(3 * x_values) * x_values**2
# Значення z
z_values = np.sin(np.abs(x_values)) * np.sin(x_values + y_values)

# Графік функції y
plot(y_values, y_values, 'Графік функції y', 'y = 0.2 * sin(3x) * x^2', 'y = 0.2 *
sin(3x) * x^2', 'x', 'y')
# Графік функції z
plot(z_values, z_values, 'Графік функції z', 'z = sin(|x|) * sin(x + y)', 'z = sin(|x|) *
sin(x + y)', 'x', 'z')
```

Результат:

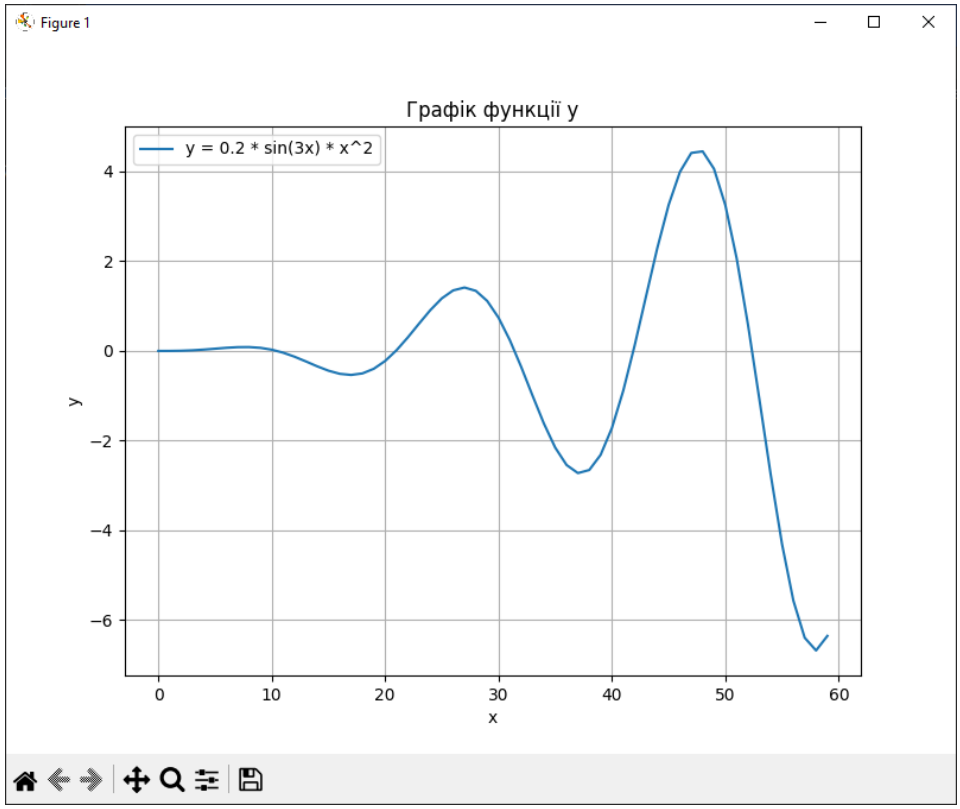

Рисунок 2 – Графік функції y

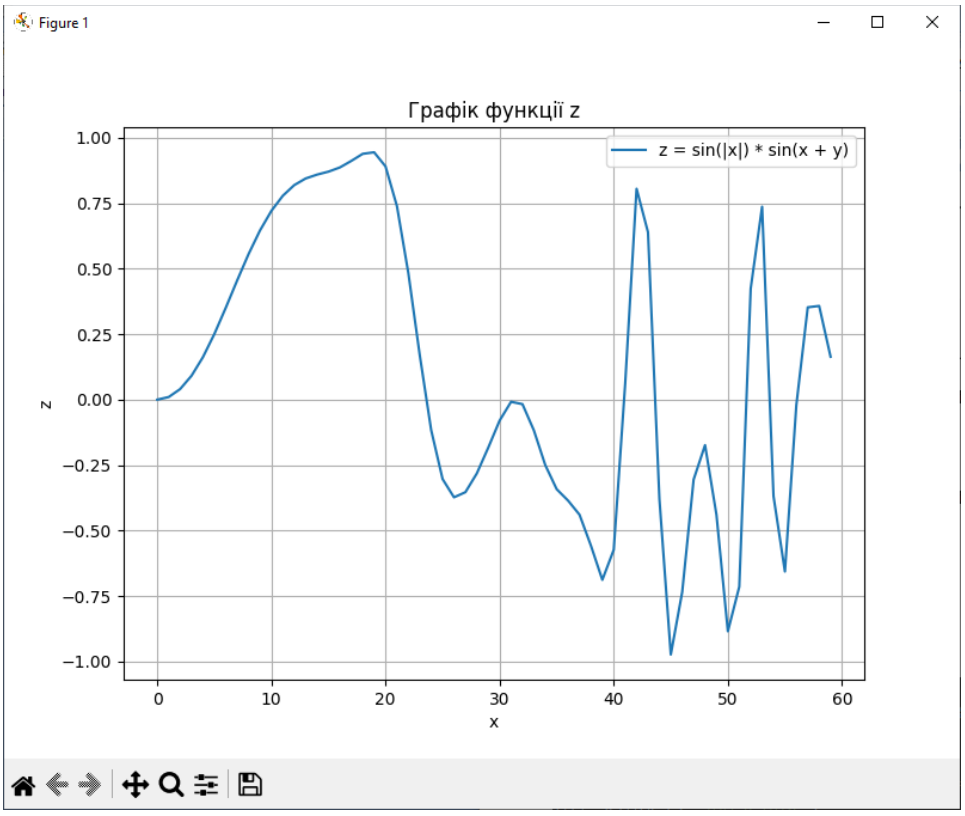

Рисунок 3 – Графік функції z

Розбиття на проміжки:

Розіб’ємо дані на проміжки відповідно кількостям функцій приналежності: x – 6 проміжків, y – 6 проміжків, z – 9 проміжків.

Лістинг:

```
# Функція розбиття змінних на інтервали
def calculate_intervals(values, n_intervals):
    start = min(values)
    end = max(values)
    intervals = []
    val = np.linspace(start, end, n_intervals + 1)
    for i in range(n_intervals):
        interval_start = val[i]
        interval_end = val[i + 1]
        ser = (interval_end + interval_start) / 2
        intervals.append([interval_start, ser, interval_end])
    diff = (intervals[0][1] - intervals[0][0]) / 2
    return intervals, diff

# Функція для виведення проміжків
def print_intervals(intervals, text):
    print('\nМежі функцій приналежності ' + str(text) + ':')
    for koord in intervals:
        print(koord)

# Створення консеквентів
x = ctrl.Antecedent(x_values, 'x')
y = ctrl.Antecedent(np.arange(min(y_values), max(y_values), 0.1), 'y')
z = ctrl.Consequent(np.arange(min(z_values), max(z_values), 0.1), 'z')

# Розбиваємо x на проміжки та зберігаємо межі
x_intervals, x_diff = calculate_intervals(x_values, 6)
print_intervals(x_intervals, 'x')
# Розбиваємо y на проміжки та зберігаємо межі
y_intervals, y_diff = calculate_intervals(y_values, 6)
print_intervals(y_intervals, 'y')
# Розбиваємо z на проміжки та зберігаємо межі
z_intervals, z_diff = calculate_intervals(z_values, 9)
print_intervals(z_intervals, 'z')
```

Результат:

```
Межі функцій приналежності x:
[0.0, 0.4916666666666667, 0.9833333333333334]
[0.9833333333333334, 1.475, 1.9666666666666668]
[1.9666666666666668, 2.4583333333333335, 2.95]
[2.95, 3.441666666666667, 3.9333333333333336]
[3.9333333333333336, 4.425000000000001, 4.916666666666667]
[4.916666666666667, 5.408333333333333, 5.9]

Межі функцій приналежності y:
[-6.678612311806419, -5.751248699627629, -4.823885087448838]
[-4.823885087448838, -3.896521475270047, -2.9691578630912567]
[-2.9691578630912567, -2.0417942509124662, -1.1144306387336753]
[-1.1144306387336753, -0.18706702655488483, 0.7402965856239057]
[0.7402965856239057, 1.6676601978026961, 2.5950238099814866]
[2.5950238099814866, 3.522387422160278, 4.449751034339069]

Межі функцій приналежності z:
[-0.9727769459962167, -0.8662844571555515, -0.7597919683148862]
[-0.7597919683148862, -0.6532994794742208, -0.5468069906335556]
[-0.5468069906335556, -0.4403145017928904, -0.33382201295222513]
[-0.33382201295222513, -0.22732952411155988, -0.12083703527089462]
[-0.12083703527089462, -0.01434454643022931, 0.092147942410436]
[0.092147942410436, 0.1986404312511012, 0.3051329200917664]
[0.3051329200917664, 0.4116254089324316, 0.5181178977730968]
[0.5181178977730968, 0.6246103866137621, 0.7311028754544274]
[0.7311028754544274, 0.8375953642950926, 0.9440878531357579]
```

Рисунок 4 – Межі інтервалів (*start*, *ser*, *end*)

Створення функцій приналежності:

Реалізуємо програму таким чином, щоб користувач міг обирати, які функції приналежності йому створити. Спершу створимо функції приналежності Гауса.

Лістинг:

```python
# Функція для виведення графіків приналежності
def plot_data(title):
    plt.title(title)
    plt.grid(True)
    plt.show()

# Функція створення функцій приналежності Гауса
def create_gauss_mf(variable, intervals, names):
    for i, (interval, name) in enumerate(zip(intervals, names)):
```

```
        universe = variable.universe
        variable[name] = fuzz.gaussmf(universe, interval[1], 0.1)

names = ['Гауса', 'Узагальнений дзвін']

# Вибір режиму створення функцій приналежності
print('Оберіть функцію приналежності:')
for i in range(len(names)):
    print(i + 1, '-', names[i])
data_mode = int(input('mode:'))

# Якщо Гаус
if data_mode == 1:
    # Створення функцій приналежності Гауса для x
    x_intervals, x_diff = calculate_intervals(x_values, 6)
    x_names = ['mx1', 'mx2', 'mx3', 'mx4', 'mx5', 'mx6']
    create_gauss_mf(x, x_intervals, x_names)
    # Створення функцій приналежності Гауса для y
    y_intervals, y_diff = calculate_intervals(y_values, 6)
    y_names = ['my1', 'my2', 'my3', 'my4', 'my5', 'my6']
    create_gauss_mf(y, y_intervals, y_names)
    # Створення функцій приналежності Гауса для z
    z_intervals, z_diff = calculate_intervals(z_values, 9)
    z_names = ['mf1', 'mf2', 'mf3', 'mf4', 'mf5', 'mf6', 'mf7', 'mf8', 'mf9']
    create_gauss_mf(z, z_intervals, z_names)

# Графік функцій приналежності x
x.view()
plot_data('Графік функцій приналежності x')
# Графік функцій приналежності y
y.view()
plot_data('Графік функцій приналежності y')
# Графік функцій приналежності z
z.view()
plot_data('Графік функцій приналежності z')
```

Результат:

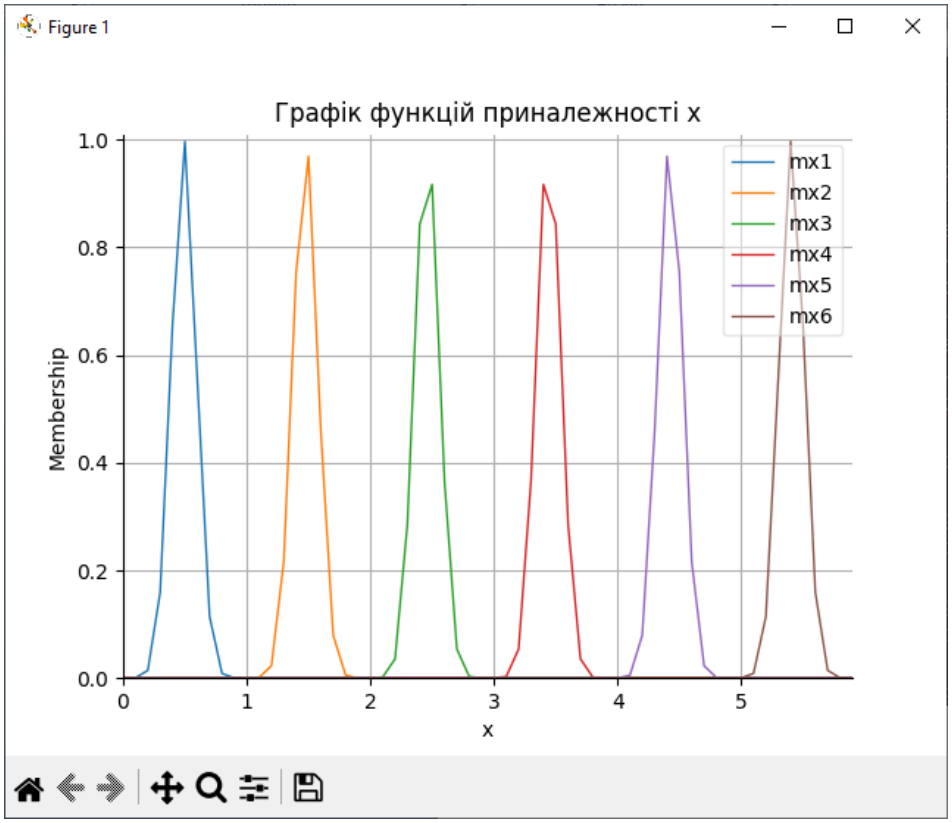

Рисунок 5 – Графік ФП *x* за Гауса

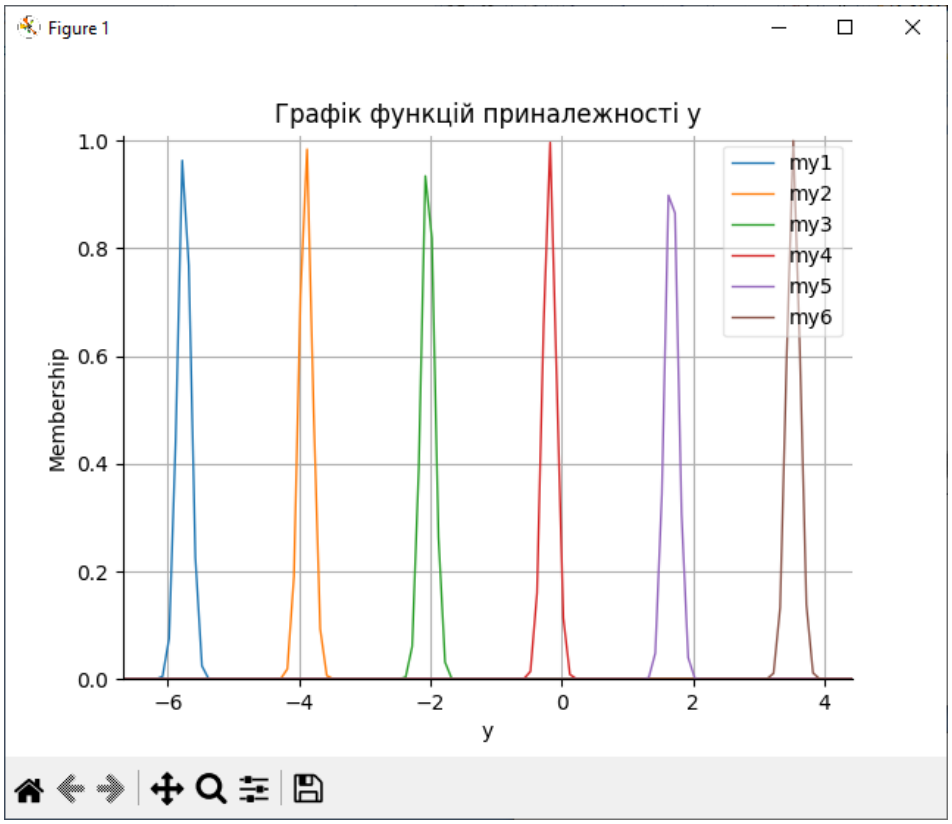

Рисунок 6 – Графік ФП *y* за Гауса

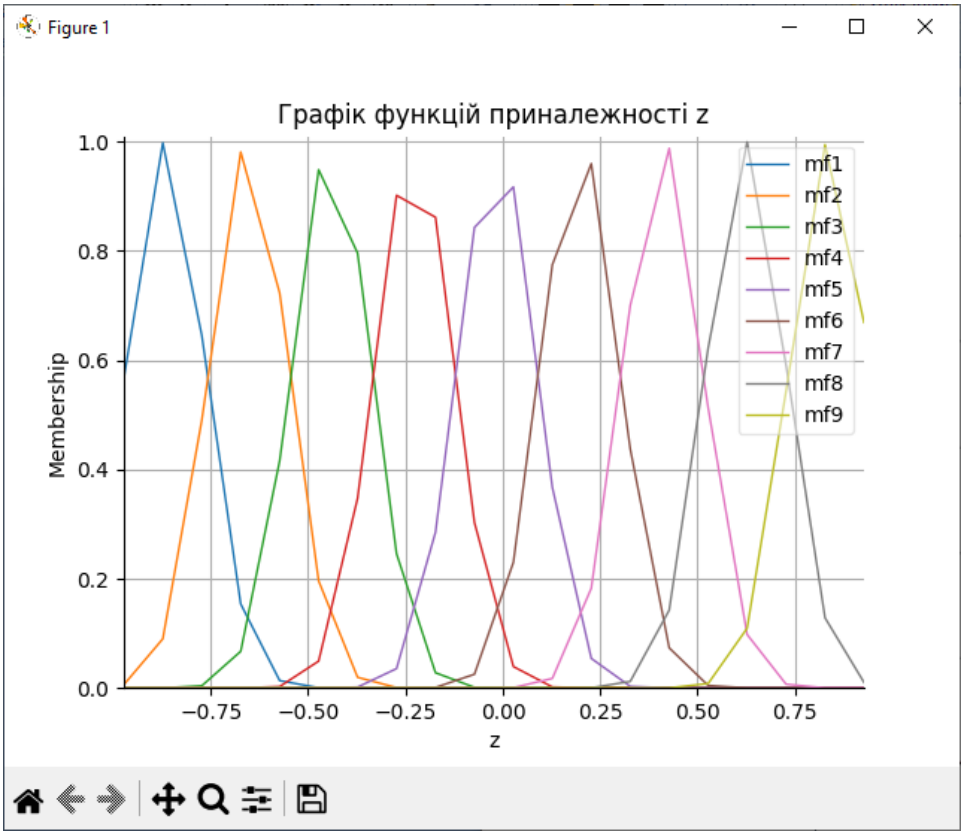

Рисунок 7 – Графік ФП z за Гауса

Таблиця приналежності:

Створимо таблицю значень функції по максимуму вхідних функцій приналежності. Потім перетворимо її в таблицю імен ФП.

Лістинг:

```
# Таблиця приналежності

# Створення масивів назв термінів
labels_x = x.terms.keys()
labels_y = y.terms.keys()
# Створення простору універсуму для оцінки приналежності
universe = np.arange(-7, 6, 0.1)
# Таблиця для значень функцій
table_values = pd.DataFrame(index=labels_y, columns=labels_x)
# Таблиця для назв функцій
table_names = pd.DataFrame(index=labels_y, columns=labels_x)

for label_x in labels_x:
    for label_y in labels_y:
        membership_x = fuzz.interp_membership(x.universe, x[label_x].mf, universe)
        membership_y = fuzz.interp_membership(y.universe, y[label_y].mf, universe)
```

```
        max_x = universe[np.argmax(membership_x)]
        max_y = universe[np.argmax(membership_y)]
        # Отримання значення функції по максимуму вхідних функцій приналежності
        value_at_maximum = z_func(max_x, max_y)
        if (value_at_maximum > z.universe.max()):
            value_at_maximum = z.universe.max()
        elif (value_at_maximum < z.universe.min()):
            value_at_maximum = z.universe.min()
        # Таблиця значень функції
        table_values.at[label_y, label_x] = value_at_maximum

        # Знаходимо, до якої функції відноситься це значення
        found_membership = False
        i = 0
        for min_range, _, max_range in z_intervals:
            i += 1
            if min_range <= value_at_maximum <= max_range:
                table_names.at[label_y, label_x] = 'mf' + str(i)
                found_membership = True
                break

# Виведення таблиць
print('\nТаблиця значень функцій по максимумам вхідних функцій приналежності:')
print(table_values)
print('\nТаблиця імен функцій:')
print(table_names)
```

Результат:

```
Таблиця значень функцій по максимумам вхідних функцій приналежності:
          mx1       mx2       mx3       mx4       mx5       mx6
my1   0.39901  0.913871  0.094406  0.172609  0.927223  0.300929
my2  0.122513 -0.673771 -0.589764  0.122513 -0.456222 -0.770829
my3 -0.478225 -0.478225  0.286923 -0.251823 -0.642772  0.197473
my4   0.14168  0.927223  0.446284  0.014917  0.829393  0.682702
my5  0.387614 -0.058228 -0.521614  0.236584  0.173346 -0.563321
my6  -0.36283 -0.956522 -0.167222 -0.147815 -0.950595 -0.387171

Таблиця імен функцій:
     mx1  mx2  mx3  mx4  mx5  mx6
my1  mf7  mf9  mf6  mf6  mf9  mf6
my2  mf6  mf2  mf2  mf6  mf3  mf1
my3  mf3  mf3  mf6  mf4  mf2  mf6
my4  mf6  mf9  mf7  mf5  mf9  mf8
my5  mf7  mf5  mf3  mf6  mf6  mf2
my6  mf3  mf1  mf4  mf4  mf1  mf3
```

Рисунок 8 – Таблиця приналежності та таблиця імен ФП Гауса

Створення правил:

Створимо правила згідно таблиці імен.

Лістинг:

```
# Створюємо список для правил
rules = []
# Ітеруємо правила через всі комбінації назв колонок та рядків таблиці
print('\nПравила:')
i = 0
for col_name in table_names.columns:
    for row_name in table_names.index:
        i += 1
        label_z = table_names.at[row_name, col_name]  # Отримуємо ім'я відповідної mf з таблиці
        antecedent = (x[col_name] & y[row_name])  # Визначаємо антецедент
        consequent = z[label_z]  # Визначаємо консеквент
        rule = ctrl.Rule(antecedent=antecedent, consequent=consequent)
        rules.append(rule)
        print(str(i) + '. if(x is ' + str(col_name) + ') and (y is ' + str(row_name) + ') then (f is ' + str(label_z) + ')')

# Додаємо всі правила до системи
system = ctrl.ControlSystem(rules)
```

Результат:

```
Правила:
1. if(x is mx1) and (y is my1) then (f is mf7)
2. if(x is mx1) and (y is my2) then (f is mf6)
3. if(x is mx1) and (y is my3) then (f is mf3)
4. if(x is mx1) and (y is my4) then (f is mf6)
5. if(x is mx1) and (y is my5) then (f is mf7)
6. if(x is mx1) and (y is my6) then (f is mf3)
7. if(x is mx2) and (y is my1) then (f is mf9)
8. if(x is mx2) and (y is my2) then (f is mf2)
9. if(x is mx2) and (y is my3) then (f is mf3)
10. if(x is mx2) and (y is my4) then (f is mf9)
11. if(x is mx2) and (y is my5) then (f is mf5)
12. if(x is mx2) and (y is my6) then (f is mf1)
13. if(x is mx3) and (y is my1) then (f is mf6)
14. if(x is mx3) and (y is my2) then (f is mf2)
15. if(x is mx3) and (y is my3) then (f is mf6)
16. if(x is mx3) and (y is my4) then (f is mf7)
17. if(x is mx3) and (y is my5) then (f is mf3)
18. if(x is mx3) and (y is my6) then (f is mf4)
19. if(x is mx4) and (y is my1) then (f is mf6)
20. if(x is mx4) and (y is my2) then (f is mf6)
21. if(x is mx4) and (y is my3) then (f is mf4)
22. if(x is mx4) and (y is my4) then (f is mf5)
```

```
23. if(x is mx4) and (y is my5) then (f is mf6)
24. if(x is mx4) and (y is my6) then (f is mf4)
25. if(x is mx5) and (y is my1) then (f is mf9)
26. if(x is mx5) and (y is my2) then (f is mf3)
27. if(x is mx5) and (y is my3) then (f is mf2)
28. if(x is mx5) and (y is my4) then (f is mf9)
29. if(x is mx5) and (y is my5) then (f is mf6)
30. if(x is mx5) and (y is my6) then (f is mf1)
31. if(x is mx6) and (y is my1) then (f is mf6)
32. if(x is mx6) and (y is my2) then (f is mf1)
33. if(x is mx6) and (y is my3) then (f is mf6)
34. if(x is mx6) and (y is my4) then (f is mf8)
35. if(x is mx6) and (y is my5) then (f is mf2)
36. if(x is mx6) and (y is my6) then (f is mf3)
```

Рисунок 9 – Правила за ФП Гауса

Симуляція

Створимо симуляцію моделі.

Лістинг:

```
# Створюємо симуляцію
simulation = ctrl.ControlSystemSimulation(system)

# Зберігаємо результати симуляції
results = []
# Виконуємо симуляцію для кожної комбінації значень x та y
for x_val, y_val in zip(x_values, y_values):
    simulation.input['x'] = x_val
    simulation.input['y'] = y_val
    simulation.compute()
    # Отримуємо вивід z для поточних значень x та y
    output = simulation.output['z']
    results.append(output)

# Графік результату симуляції z
plot(z_values, results, 'Порівняння графіків функції та симуляції', 'Графік функції z',
'Графік симуляції z', 'x', 'z')
```

Результат:

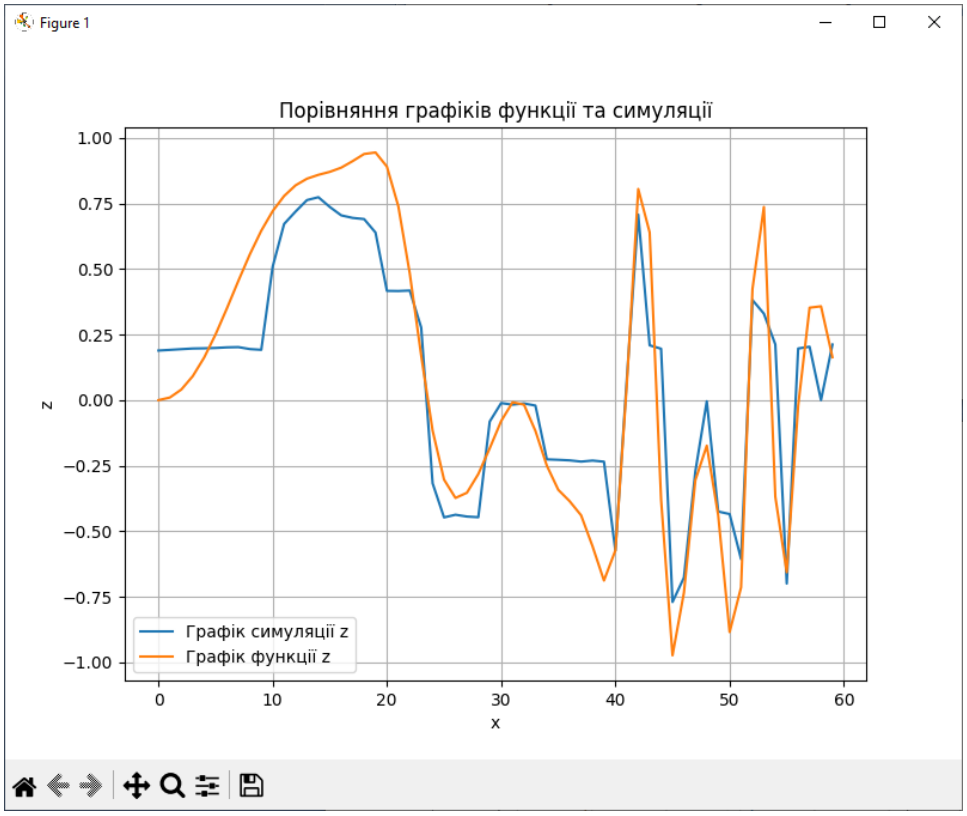

Рисунок 10 – Графіки функції та моделі симуляції за ФП Гауса

Оцінка точності симуляції

Для оцінки точності моделі симуляції використаємо середньо-абсолютну помилку (MAE) та коефіцієнт детермінації (R^2).

Лістинг:

```
# Функція для обчислення коефіцієнта детермінації R^2
def mae_r2_score(actual, predicted):
    # Розрахунок MAE
    mae = np.mean(np.abs(actual - predicted))
    # Розрахунок R^2
    mean_actual = np.mean(actual)
    sst = np.sum((actual - mean_actual) ** 2)
    ssr = np.sum((actual - predicted) ** 2)
    r2 = 1 - (ssr / sst)
    print('\nЯкість симуляції:')
    print('Середньо-абсолютна помилка =', mae)
    print('Коефіцієнт детермінації =', r2)

# Аналіз результатів
mae_r2_score(z_values, results)
```

Результат:

```
Якість симуляції:
Середньо-абсолютна помилка = 0.1800293501836771
Коефіцієнт детермінації = 0.814550326502512
```

Рисунок 11 – Оцінка точності моделі симуляції за ФП Гауса

Вплив форми функції приналежності на якість моделювання:

Створимо функції приналежності «Узагальнений дзвін».

Лістинг:

```
# Функція створення функцій приналежності Узагальненого дзвону
def create_gbell(variable, intervals, names):
    for i, (interval, name) in enumerate(zip(intervals, names)):
        universe = variable.universe
        variable[name] = fuzz.gbellmf(universe, 0.1, 5, interval[1])

# Якщо Дзвін
else:
    # Створення функцій приналежності Узагальнений дзвін для x
    x_intervals, x_diff = calculate_intervals(x_values, 6)
    x_names = ['mx1', 'mx2', 'mx3', 'mx4', 'mx5', 'mx6']
    create_gbell(x, x_intervals, x_names)
    # Створення функцій приналежності Узагальнений дзвін для y
    y_intervals, y_diff = calculate_intervals(y_values, 6)
    y_names = ['my1', 'my2', 'my3', 'my4', 'my5', 'my6']
    create_gbell(y, y_intervals, y_names)
    # Створення функцій приналежності Узагальнений дзвін для z
    z_intervals, z_diff = calculate_intervals(z_values, 9)
    z_names = ['mf1', 'mf2', 'mf3', 'mf4', 'mf5', 'mf6', 'mf7', 'mf8', 'mf9']
    create_gbell(z, z_intervals, z_names)
```

Результат:

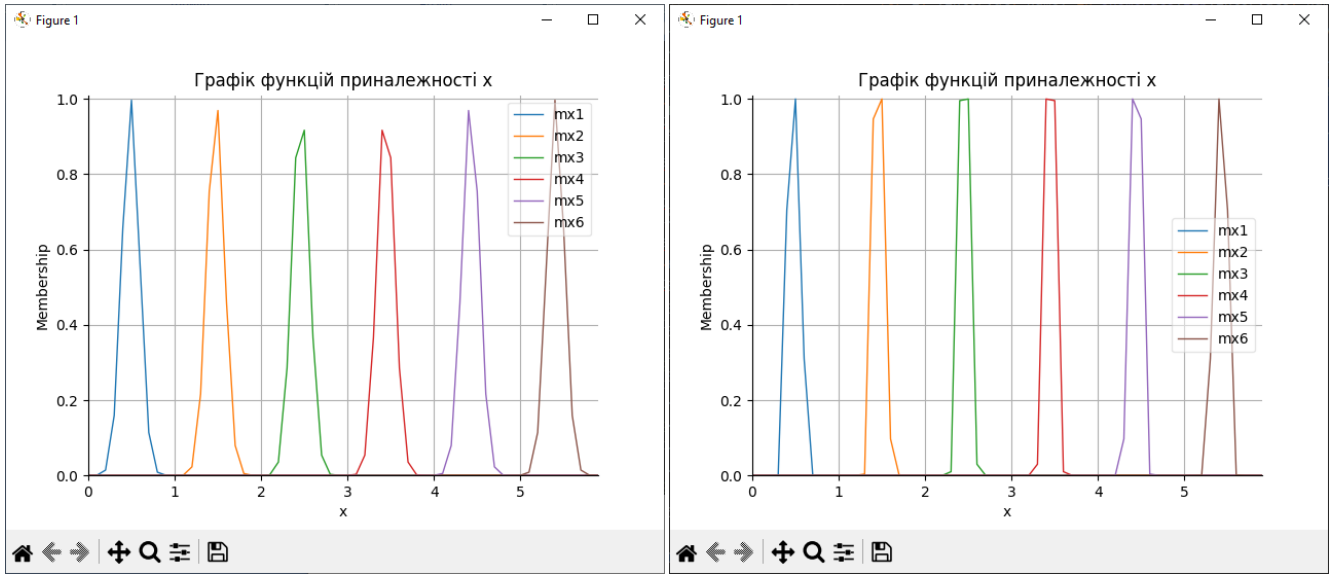

Рисунок 12 – Порівняння графіків ФП *x* за Гаусом та Дзвоном

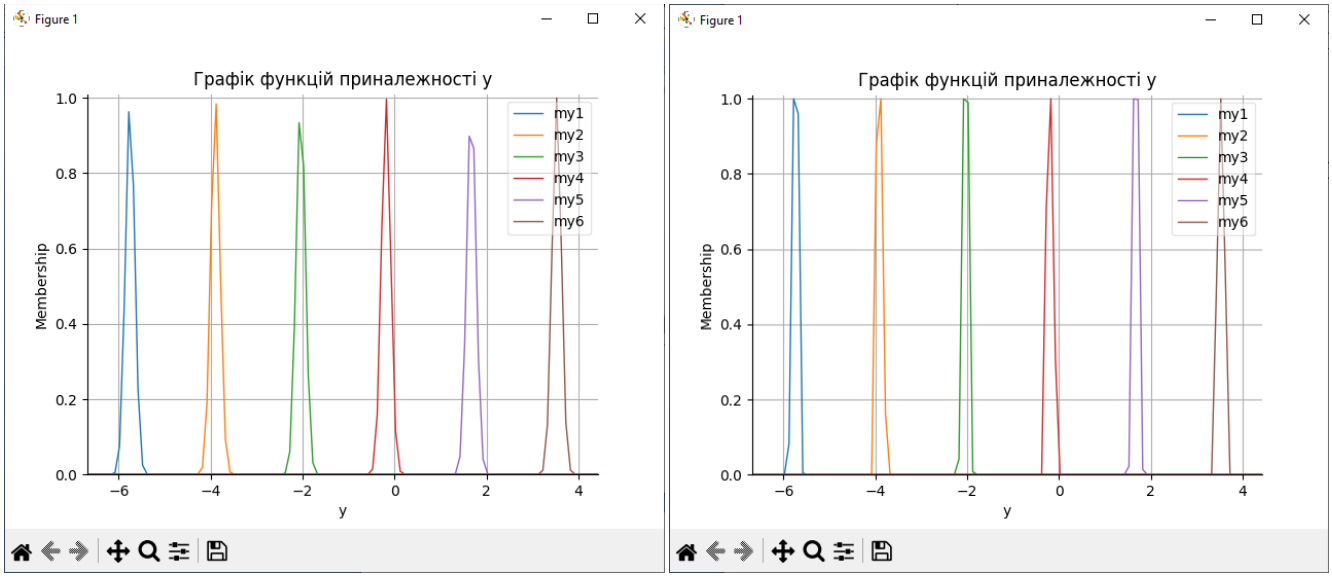

Рисунок 13 – Порівняння графіків ФП *y* за Гаусом та Дзвоном

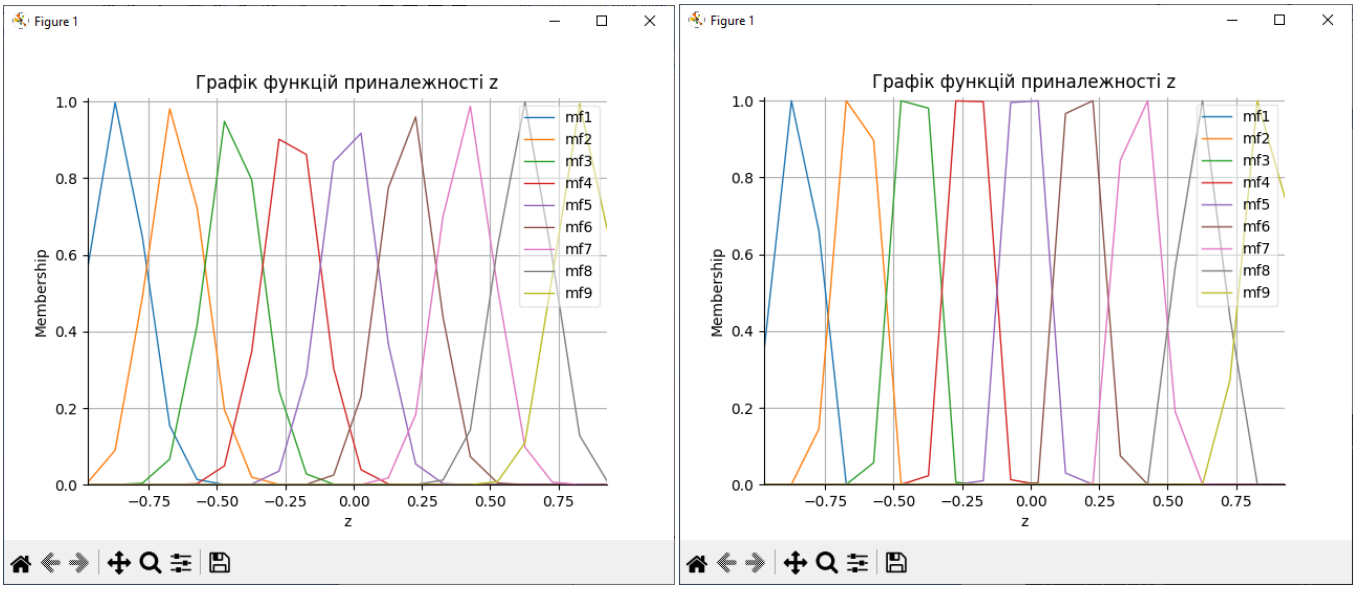

Рисунок 14 – Порівняння графіків ФП *z* за Гаусом та Дзвоном

Виконаємо ті самі кроки, що й для ФП Гауса.

Результат:

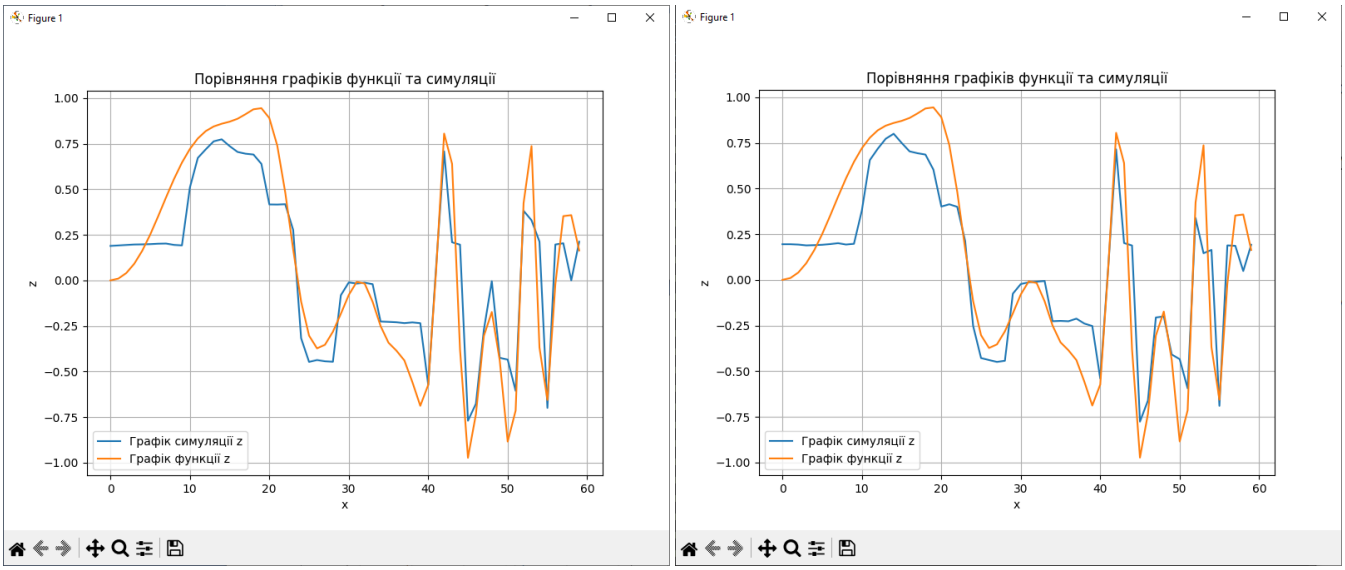

Рисунок 15 – Порівняння графіків моделі симуляції за ФП Гауса та ФП Дзвін

```
Якість симуляції:
Середньо-абсолютна помилка = 0.1800293501836771
Коефіцієнт детермінації = 0.814550326502512
```

```
Якість симуляції:
Середньо-абсолютна помилка = 0.18202507629364678
Коефіцієнт детермінації = 0.8059812633477634
```

Рисунок 11 – Порівняння точності моделей симуляції за ФП Гауса та ФП Дзвін

Бачимо, що так, як функції приналежності класів перетинались в різних точках, точність моделей симуляції несуттєво відрізняється. Репрезентативнішою виявилась функція приналежності Гауса.

Можливість зменшення числа правил

Шляхом експерименту, було виявлено, що зменшення числа правил веде до появи значень вихідної функції z, які не описуються жодним правилом. Це викликає помилку програми на етапі симуляції.

Висновок:

У даній лабораторній роботі я ознайомилась з методами моделювання функціїй двох зміних за допомогою правил нечіткої логіки та алгоритмом створення нечіткого контролеру.

Розробила нечіткий контролер на мові програмування python для моделювання власної функції двох зміних.

Провела дослідження впливу форм функцій приналежності на якість моделі і виявила, що на результат впливають точки перетину класів приналежності. Підтвердила необхідність вичерпної кількості правил для симуляції моделі.