Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота *№1*

із дисципліни ***«Методи і технології штучного інтелекту»***

Тема: ***«Дослідження способів формування нечітких множин і операцій над ними»***

Виконав:

Студент групи ІА-34

Ястремський Богдан

Перевірив:

старший викладач кафедри ІСТ

Польшакова Ольга Михайлівна

Київ — 2025

Тема: Дослідження способів формування нечітких множин і операцій над ними.

Мета: Побудувати нечіткі множин з використанням різних типів функцій приналежності. Виконати найбільш поширені логічні операції над нечіткими множинами.

Примітки: Код усіх файлів буде винесено окремо в додаток А наприкінці звіту.

Хід роботи:

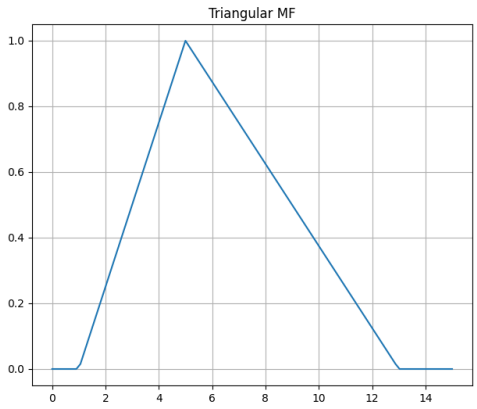
1. Побудувати трикутну і трапецієподібну функцію приналежності.

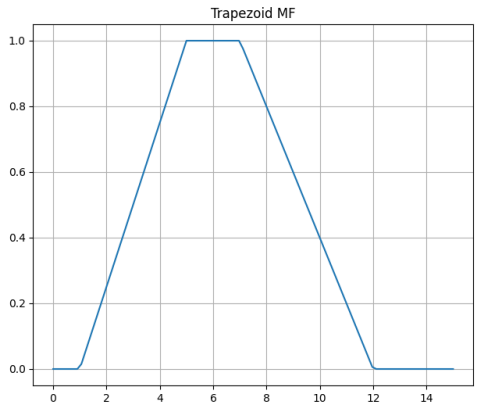
   Створюємо функції trimf/trapmf відповідно у власному пакеті myskfuzzy.

   Параметри:

   tri_mf = trimf(x, [1, 5, 13])

   trap_mf = trapmf(x, [1, 5, 7, 12])





2. Побудувати просту і двосторонню функцію приналежності Гауса, утворену за допомогою різних функцій розподілу.

   Створюємо функції gaussmf/gauss2mf відповідно у власному пакеті myskfuzzy.

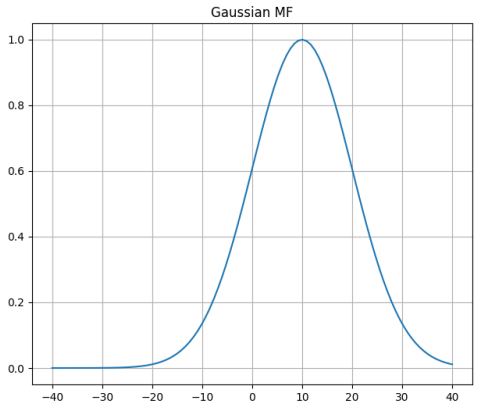
   Параметри:

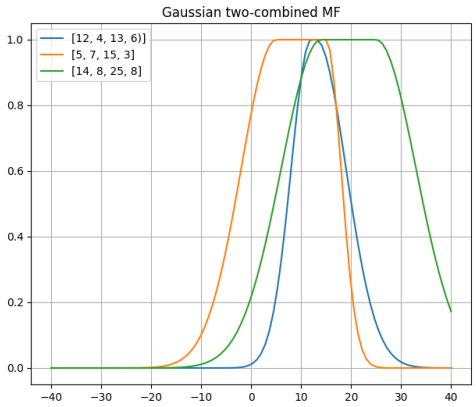
   gauss = gaussmf(x, 10, 10)

   gauss21 = gauss2mf(x, 12, 4, 13, 6)

   gauss22 = gauss2mf(x, 5, 7, 15, 3)

   gauss23 = gauss2mf(x, 14, 8, 25, 8)



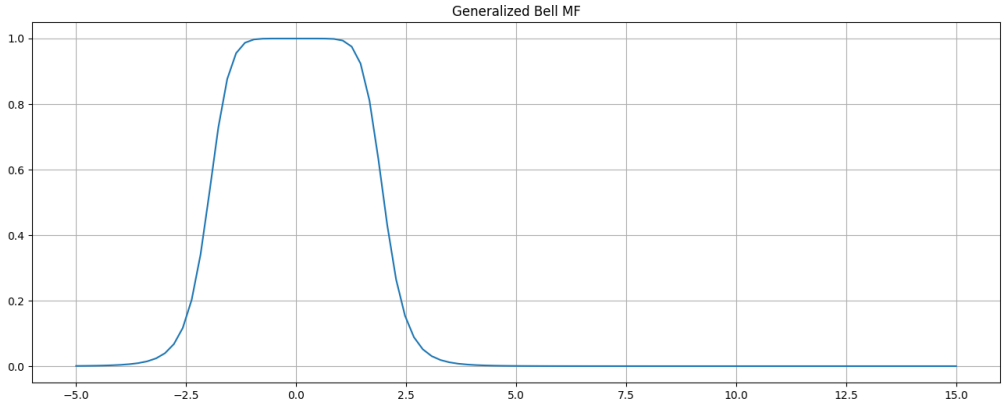


3. Побудувати функцію приналежності "узагальнений дзвін", яка дозволяє представляти нечіткі суб'єктивні переваги.

   Створюємо функції gbellmf відповідно у власному пакеті myskfuzzy.

   Параметри:

   bell = gbellmf(x, 2, 4, 0)



4. Побудувати набір сигмоїдних функцій: основну односторонню, яка відкрита зліва чи справа; додаткову двосторонню; додаткову несиметричну.

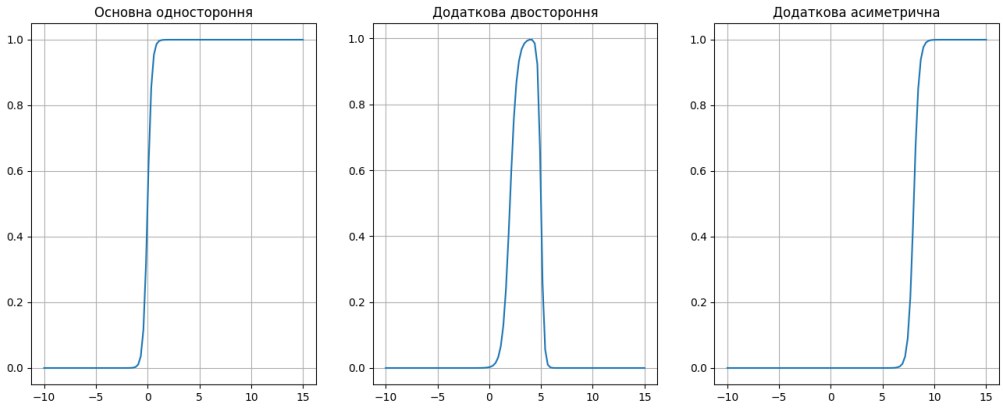
   Створюємо функції sigmf, dsigmf, psigmf відповідно у власному пакеті myskfuzzy.

   Параметри:

   one_side = sigmf(x, 0, -5)

   two_side = dsigmf(x, 2, 3, 5, 7)

   add_asym = psigmf(x, 4, 5, 8, 4)



5. Побудувати набір поліноміальних функцій приналежності (Z-, PI- і S-функцій).

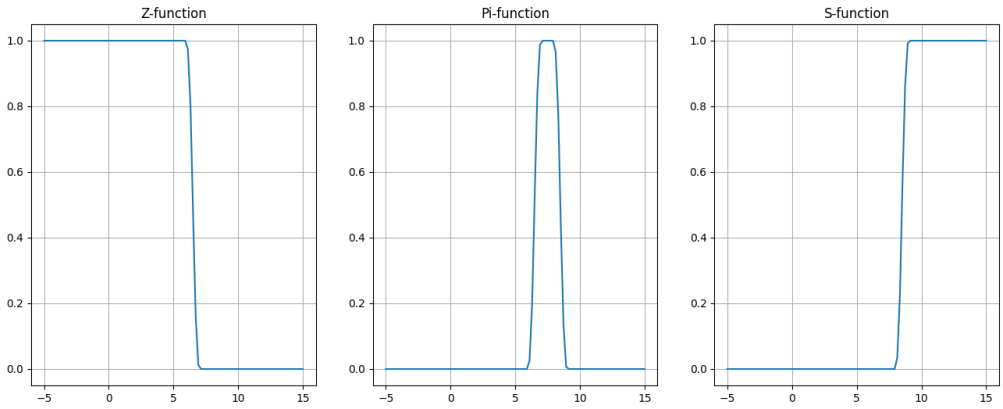
   Створюємо функції zmf, pimf, smf відповідно у власному пакеті myskfuzzy.

   Параметри:

   z = zmf(x, 6, 7)

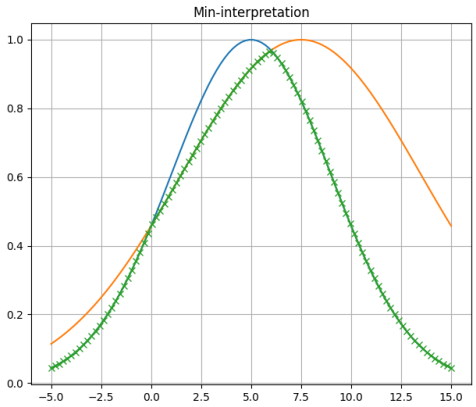
   pi = pimf(x, 6, 7, 8, 9)

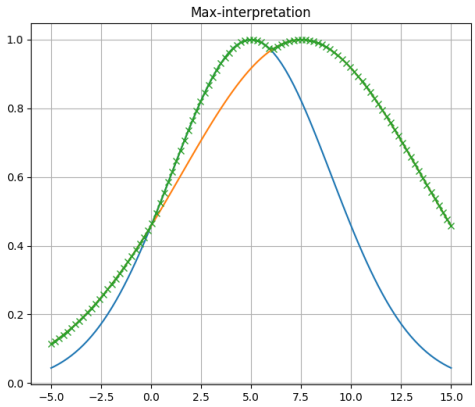
   s = smf(x, 8, 9)



6. Побудувати мінімаксну інтерпретацію логічних операторів з використанням операцій пошуку мінімуму і максимуму.

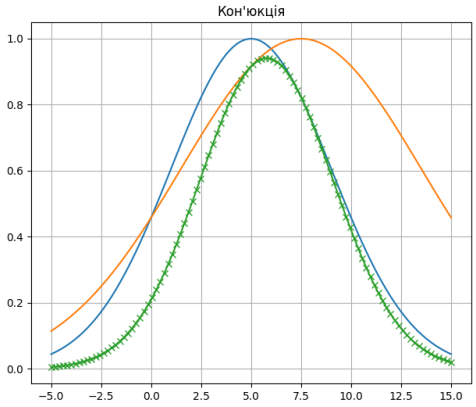
   Беремо 2 будь-які функції, створюємо масив мінімуму і максимуму з них двох. Для наглядності промаркуємо.

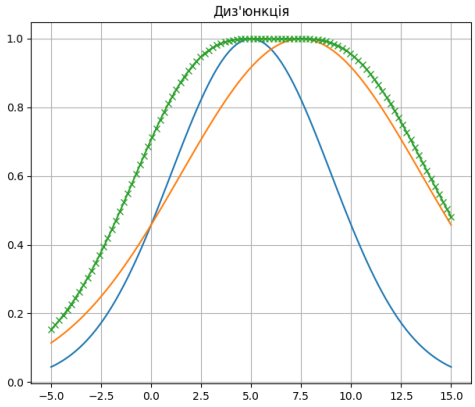




7. Побудувати вірогідну інтерпретацію кон’юнктивних і диз’юнктивних операторів.

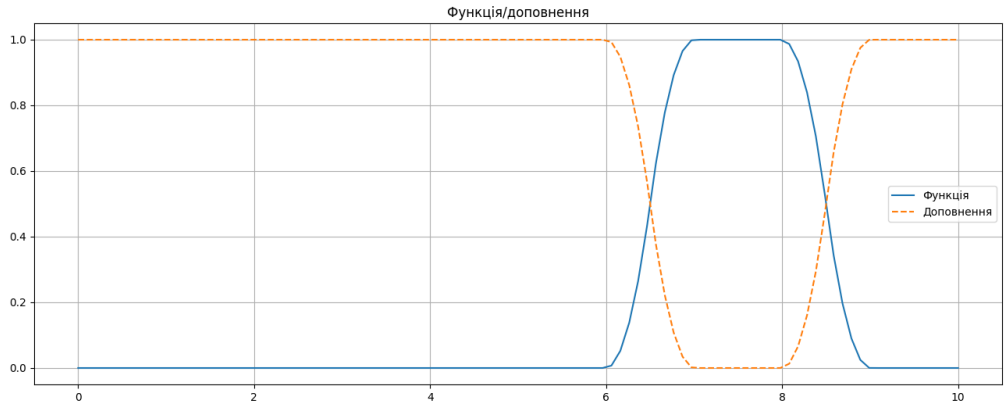
   Беремо 2 будь-які функції, створюємо масив and і or з них двох. Для наглядності промаркуємо.





8. Побудувати доповнення нечіткої множини, яке описує деяке розмите судження і представляє собою математичний опис вербального вираження, який заперечує це нечітка множина.

   Беремо 1 будь-яку функції, створюємо масив not з неї. Для наглядності змінимо тип лінії.



Висновок: на даному лабораторному занятті я познайомився з поняттям нечітких множин і функцій приналежності: які бувають їхні види, якими формулами або сплайнами вони задаються, а також для чого вони застосовуються і як проводити по ним аналіз. Також я пригадав основні булеві операції типу І, АБО, НЕ, та як вони реалізуються в контексті нечітких множин. Також я навчився використовувати мінімум і максимум в контексті нечітких множин.

Відповіді на контрольні питання:

1. Що таке нечітка множина і чим вона відрізняється від класичної (чіткої) множини?

   Нечітка множина — це сукупність елементів, які належать до неї з певним ступенем, який задається функцією приналежності. Класична множина чітко розрізняє що входить, що не входить.
2. Яке основне призначення функції приналежності в нечіткій множині?

   Присвоєння елементу його ступеня приналежності до множини.
3. У якому діапазоні приймають значення функції приналежності?

   Від 0 (не належить) до 1 (повністю належить).
4. Чим відрізняється повна належність елемента множині від часткової?

   Повна належність — найідеальніший вибір. Часткова — хороший, але не ідеальний.

5. Наведіть приклади реальних задач, де доцільно використовувати нечіткі множини.

   Керування температурою або освітленням (варіація температури або яскравості світла). Прогноз погоди (легкий вітер, невеликий дощ). Оцінки в 1 класі (добре, відмінно, і т.д.)
6. Що таке універсальна множина у теорії нечітких множин?

   Множина у яку входить абсолютно все. Вона єдина.
7. Які типові форми функцій приналежності застосовуються на практиці?

   Трикутна, трапецієподібна, сигмоїдна, «узагальнений дзвін», Гауса, Z-, Pi-, S-функції.
8. Яка різниця між трикутною та трапецієподібною функціями приналежності?

   Трикутна форма має ідеал лише на піку, трапеція — ціле плато.
9. Як інтерпретувати значення функції приналежності, наприклад $\mu(x) = 0.7$?

   Щось є чимось на 70%. Тобто наприклад, температура є ідеальною на 70%.
10. Що таке α-рівень (α-cut) нечіткої множини?

    Це чітка множина елементів, степінь приналежності яких вища або більша за число α.
11. Як визначається комплементарна (доповняльна) нечітка множина?

    Стандартною операцією НЕ. $NOT(\alpha) = 1 - \alpha$.
12. У чому полягає відмінність між операціями об’єднання та перетину в нечітких множинах?

    Перетин — мінімум серед функцій, об’єднання — максимум.
13. Що таке нечітке відношення і як воно пов’язане з функціями приналежності?

    Нечітке відношення — відношення між елементами n множин, де ступінь приналежності може бути частковою, і обчислюється добутком всіх приналежностей.

14. Як можна використовувати функції приналежності для прийняття рішень у нечіткій логіці?

    Можна ставити умови, і обробляти певні дії, базуючись на них.

15. У чому полягають основні переваги нечітких множин у порівнянні з класичною логікою?

    Ми можемо більш обширно описати те, що нам треба.

ДОДАТОК А.

modules/graphbuilder.py

```
import matplotlib.pyplot as plt


def build_plot(x, graph_data):
    """
    Builds a plot of graphs
    :param x:
    1-D array of values for x-axi of any graphs
    :param graph_data:
    a dictionary, based on:
    data = {
        [number, starting with 1]: {
            'title': [any title],
            'lines': [
                {
                'y': [array of values of function],
                ('label'): [any label],
                ('color'): [any color],
                ('linestyle'): [any linestyle],
                ('marker'): [any marker],

                }
            ]
        },
        ...
    }
    :return: the picture of plot
    """

    # Quantity of cols and rows needed for a plot
    num_cols = len(graph_data)
    num_rows = 1

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 6))

    # If axes is 1-D array, make it 2-D for easier handling
    if num_cols == 1:
        axes = [axes]

    # Getting index of ax and ax itself
    for i, ax in enumerate(axes):
        graph = graph_data[i + 1]  # Getting an info about graph
        ax.set_title(graph['title'])  # Setting a title for ax

        for line in graph['lines']:
            y = line['y']
            line_label = line.get('label', None)
            line_color = line.get('color', None)
            line_linestyle = line.get('linestyle', None)
            line_marker = line.get('marker', None)

            ax.plot(x,
                    y,
                    label=line_label,
                    color=line_color,
                    linestyle=line_linestyle,
                    marker=line_marker)
            ax.grid(True)
```

```
            if line_label is not None:  # If we have set up a label, then add
legend
                ax.legend()

    plt.show()
```

modules/tasks_functions.py

```
from .myskfuzzy import *
from .graphbuilder import *


def task1():
    x = np.linspace(0, 15, 100)

    try:
        tri_mf = trimf(x, [1, 5, 13])
        trap_mf = trapmf(x, [1, 5, 7, 12])

        data = {
            1: {
                'title': 'Triangular MF',
                'lines': [
                    {'y': tri_mf}
                ]
            },
            2: {
                'title': 'Trapezoid MF',
                'lines': [
                    {'y': trap_mf}
                ]
            }
        }

        build_plot(x, data)
    except ValueError as ve:
        print(ve)


def task2():
    x = np.linspace(-40, 40, 100)

    gauss = gaussmf(x, 10, 10)
    gauss21 = gauss2mf(x, 12, 4, 13, 6)
    gauss22 = gauss2mf(x, 5, 7, 15, 3)
    gauss23 = gauss2mf(x, 14, 8, 25, 8)

    data = {
        1: {
            'title': 'Gaussian MF',
            'lines': [
                {'y': gauss}
            ]
        },
        2: {
            'title': 'Gaussian two-combined MF',
            'lines': [
                {'y': gauss21, 'label': '[12, 4, 13, 6)]'},
                {'y': gauss22, 'label': '[5, 7, 15, 3]'},
                {'y': gauss23, 'label': '[14, 8, 25, 8]'}
            ]
        }
```

```
    }

    build_plot(x, data)


def task3():
    x = np.linspace(-5, 15, 100)
    bell = gbellmf(x, 2, 4, 0)

    data = {
        1: {
            'title': 'Generalized Bell MF',
            'lines': [
                {'y': bell}
            ]
        }
    }

    build_plot(x, data)


def task4():
    x = np.linspace(-10, 15, 100)

    one_side = sigmf(x, 0, 5)
    two_side = dsigmf(x, 2, 3, 5, 7)
    add_asym = psigmf(x, 4, 5, 8, 4)

    data = {
        1: {
            'title': 'Основна одностороння',
            'lines': [
                {'y': one_side}
            ]
        },
        2: {
            'title': 'Додаткова двостороння',
            'lines': [
                {'y': two_side}
            ]
        },
        3: {
            'title': 'Додаткова асиметрична',
            'lines': [
                {'y': add_asym}
            ]
        }
    }

    build_plot(x, data)


def task5():
    x = np.linspace(-5, 15, 100)

    z = zmf(x, 6, 7)
    pi = pimf(x, 6, 7, 8, 9)
    s = smf(x, 8, 9)

    data = {
        1: {
            'title': 'Z-function',
            'lines': [
                {'y': z}
```

```
            ]
        },
        2: {
            'title': 'Pi-function',
            'lines': [
                {'y': pi}
            ]
        },
        3: {
            'title': 'S-function',
            'lines': [
                {'y': s}
            ]
        }
    }

    build_plot(x, data)


def task6():
    x = np.linspace(-5, 15, 100)

    f1 = gaussmf(x, 5, 4)
    f2 = gaussmf(x, 7.5, 6)

    fmin = np.fmin(f1, f2)
    fmax = np.fmax(f1, f2)

    data = {
        1: {
            'title': 'Min-interpretation',
            'lines': [
                {'y': f1},
                {'y': f2},
                {'y': fmin, 'marker': 'x'}
            ]
        },
        2: {
            'title': 'Max-interpretation',
            'lines': [
                {'y': f1},
                {'y': f2},
                {'y': fmax, 'marker': 'x'}
            ]
        }
    }

    build_plot(x, data)


def task7():
    x = np.linspace(-5, 15, 100)

    f1 = gaussmf(x, 5, 4)
    f2 = gaussmf(x, 7.5, 6)

    and_f = f1 * f2
    or_f = f1 + f2 - and_f

    data = {
        1: {
            'title': 'Кон\'юкція',
            'lines': [
                {'y': f1},
```

```
                {'y': f2},
                {'y': and_f, 'marker': 'x'}
            ]
        },
        2: {
            'title': 'Диз\'юнкція',
            'lines': [
                {'y': f1},
                {'y': f2},
                {'y': or_f, 'marker': 'x'}
            ]
        }
    }

    build_plot(x, data)


def task8():
    x = np.linspace(0, 10, 100)

    f = pimf(x, 6, 7, 8, 9)
    not_f = 1 - f

    data = {
        1: {
            'title': 'Функція/доповнення',
            'lines': [
                {'y': f, 'label': 'Функція'},
                {'y': not_f, 'linestyle': '--', 'label': 'Доповнення'}
            ]
        }
    }

    build_plot(x, data)
```

modules/myskfuzzy.py

```
import numpy as np


def fraction_subtractions(num1, num2, den1, den2):
    """
    Calculates the value of [(num1 - num2) / (der1 - der2)]

    :param num1: numerator 1st term
    :param num2: numerator 2nd term
    :param den1: denominator 1st term
    :param den2: denominator 2nd term
    :return: the value of [(num1 - num2) / (der1 - der2)]
    """

    return (num1 - num2) / (den1 - den2)


def trimf(dot_set, abc):
    """
    Forms a set of dots, which represent Triangular Membership Function

    :param dot_set: A set of dots for parsing through conditions
    :param abc: A set of boundary dots
    :return: A set of dots, which unite in a triangular
    :raises ValueError: if triangular is not correct. (a < b < c)
```

```
    """

    a, b, c = abc

    if not (a < b < c):
        raise ValueError("You set a wrong points for triangle. "
                         "Each number should be sharply bigger than previous
one.")

    triangle_dots = np.zeros(len(dot_set))

    for i, x in enumerate(dot_set):
        if a <= x <= b:
            triangle_dots[i] = fraction_subtractions(x, a, b, a)
        elif b <= x <= c:
            triangle_dots[i] = fraction_subtractions(c, x, c, b)

    return triangle_dots


def trapmf(dot_set, abcd):
    """
    Function for creating Trapezoid Membership Function

    :param dot_set: A set of dots for parsing through conditions
    :param abcd: A set of boundary dots
    :return: A set of dots, which unite in a trapezoid
    :raises ValueError: if trapezoid is not correct. (a <= b <= c <= d)
    """
    trapezoid_dots = np.zeros(len(dot_set))
    a, b, c, d = abcd

    if not (a <= b <= c <= d):
        raise ValueError("You set a wrong points for trapezoid. "
                         "Each number should be bigger or equally bigger than
previous one.")

    for i, x in enumerate(dot_set):

        if a <= x < b:
            trapezoid_dots[i] = fraction_subtractions(x, a, b, a)
        elif b <= x <= c:
            trapezoid_dots[i] = 1
        elif c < x <= d:
            trapezoid_dots[i] = fraction_subtractions(d, x, d, c)

    return trapezoid_dots


def gaussmf(x, c, sigma):
    """
    Computes values using a Gaussian membership function

    :param x: set of x-axi coordinates
    :param c: center of the curve
    :param sigma: width of the curve
    :return: values using a Gaussian membership function
    """
    return np.exp(-((x - c) ** 2) / (2 * sigma ** 2))


def gauss2mf(dot_set, c1, sigma1, c2, sigma2):
    """
    Computes values using a combination of two Gaussian membership functions.
```

```
    :param dot_set: set of x-axi coordinates
    :param c1: center of the curve 1
    :param sigma1: width of the curve 1
    :param c2: center of the curve 2
    :param sigma2: width of the curve 2
    :return: values using a combination of two Gaussian membership functions
    """
    gauss2mf_dots = np.ones(len(dot_set))

    for i, x in enumerate(dot_set):
        if x <= c1:
            gauss2mf_dots[i] = gaussmf(x, c1, sigma1)
        elif x > c2:
            gauss2mf_dots[i] = gaussmf(x, c2, sigma2)

    return gauss2mf_dots


def gbellmf(x, a, b, c):
    """
    Computes values using a generalized bell-shaped membership function

    :param x: set of x-axi coordinates
    :param a: width of the membership function,
    :param b: shape of the curve on either side of plateau
    :param c: center of the membership function
    :return: values using a generalized bell-shaped membership function
    """

    return 1 / (1 + np.abs(((x - c) / a))**(2 * b))


def sigmf(x, c, a):
    """
    Computes values using a sigmoidal membership function

    :param x: set of x-axi coordinates
    :param c: center of the membership function
    :param a: width of the transition area
    :return: values using a sigmoidal membership function
    """

    return 1.0 / (1.0 + np.exp(-a * (x - c)))


def dsigmf(x, c1, a1, c2, a2):
    """
    Computes values using the difference between two sigmoidal membership
functions

    :param x: set of x-axi coordinates
    :param a1: width of the transition area 1
    :param c1: center of the membership function 1
    :param a2: width of the transition area 2
    :param c2: center of the membership function 2
    :return: values using the difference between two sigmoidal membership
functions
    """

    return sigmf(x, c1, a1) - sigmf(x, c2, a2)


def psigmf(x, c1, a1, c2, a2):
```

```
    """
    Computes values using the product of two sigmoidal membership functions

    :param x: set of x-axi coordinates
    :param a1: width of the transition area 1
    :param c1: center of the membership function 1
    :param a2: width of the transition area 2
    :param c2: center of the membership function 2
    :return: values using the product of two sigmoidal membership functions
    """
    return sigmf(x, c1, a1) * sigmf(x, c2, a2)


def zmf(dot_set, a, b):
    """
    Computes values using a spline-based Z-shaped membership function

    :param dot_set: set of x-axi coordinates
    :param a: shoulder of function
    :param b: foot of function
    :return: values using a spline-based Z-shaped membership function
    """
    zmf_dots = np.ones(len(dot_set))

    sum_mid = (a + b) / 2

    for i, x in enumerate(dot_set):
        if a <= x <= sum_mid:
            zmf_dots[i] = 1 - 2 * fraction_subtractions(x, a, b, a) ** 2
        elif sum_mid <= x <= b:
            zmf_dots[i] = 2 * fraction_subtractions(x, b, b, a) ** 2
        elif x >= b:
            zmf_dots[i] = 0

    return zmf_dots


def pimf(dot_set, a, b, c, d):
    """
    Computes values using a spline-based Pi-shaped membership function

    :param dot_set: set of x-axi coordinates
    :param a: left foot of function
    :param b: left shoulder of function
    :param c: right shoulder of function
    :param d: right foot of function
    :return: values using a spline-based Pi-shaped membership function
    """
    pimf_dots = np.zeros(len(dot_set))

    sum_ab_mid = (a + b) / 2
    sum_cd_mid = (c + d) / 2

    for i, x in enumerate(dot_set):
        if a <= x <= sum_ab_mid:
            pimf_dots[i] = 2 * fraction_subtractions(x, a, b, a) ** 2
        elif sum_ab_mid <= x <= b:
            pimf_dots[i] = 1 - 2 * fraction_subtractions(x, b, b, a) ** 2
        elif b <= x <= c:
            pimf_dots[i] = 1
        elif c <= x <= sum_cd_mid:
            pimf_dots[i] = 1 - 2 * fraction_subtractions(x, c, d, c) ** 2
        elif sum_cd_mid <= x <= d:
            pimf_dots[i] = 2 * fraction_subtractions(x, d, d, c) ** 2
```

```
    return pimf_dots


def smf(dot_set, a, b):
    """
    Computes values using a spline-based S-shaped membership function

    :param dot_set: set of x-axi coordinates
    :param a: foot of function
    :param b: shoulder of function
    :return: values using a spline-based S-shaped membership function
    """
    smf_dots = np.zeros(len(dot_set))

    sum_mid = (a + b) / 2

    for i, x in enumerate(dot_set):
        if a <= x <= sum_mid:
            smf_dots[i] = 2 * fraction_subtractions(x, a, b, a) ** 2
        elif sum_mid <= x <= b:
            smf_dots[i] = 1 - 2 * fraction_subtractions(x, b, b, a) ** 2
        elif x >= b:
            smf_dots[i] = 1

    return smf_dots
```

main.py

```
from modules.tasks_functions import *

# === 1. Побудувати трикутну і трапецієподібну ФП ===
task1()

# === 2. Побудувати просту і двосторонню ФП Гауса, утворену за допомогою
різних функцій розподілу ===
task2()

# === 3. Побудувати ФП "узагальнений дзвін", яка дозволяє представляти
нечіткі суб'єктивні переваги. ===
task3()

# === 4. Побудувати набір сигмоїдних ФП: основну односторонню; додаткову
двосторонню; додаткову несиметричну ===
task4()

# === 5. Побудувати набір поліноміальних функцій приналежності (Z-, PI- і S-
функцій). ===
task5()

# === 6. Побудувати мінімаксну інтерпретацію логічних операторів з
використанням операцій мінімуму і максимуму ===
task6()

# === 7. Побудувати вірогідну інтерпретацію кон’юнктивних і диз’юнктивних
операторів ===
task7()

# === 8. Побудувати доповнення нечіткої множини ===
task8()
```