**ПРИЛОЖЕНИЯ**

*Приложение 1. Код дополнения, пересечения и объединения нечётких множеств*

|  |
| --- |
| # Проинициализируем нечёткие множества  set1 = [1000, 2000, 3000, 4000]  # мощнее  set2 = [2000, 3000, 4000, 5000]  # хорош!  # Дополнение нечёткого множества  complement = [10000 - x for x in set1]  # Пересечение нечётких множеств  intersection = list(set(set1) & set(set2))  # используем функцию set для поиска пересечения  # Объединение нечётких множеств  union = sorted(set(set1) | set(set2))  # используем функцию set для объединения и сортируем результат  # Вывод результатов  print("Дополнение: ", complement)  print("Пересечение: ", intersection)  print("Объединение: ", union) |

*Приложение 2. Код, реализующий степени принадлежности с графиками термов*

|  |
| --- |
| import matplotlib.pyplot as plt  import math  def calculate\_mean(data):      return sum(data) / len(data)  def calculate\_std(data, mean):      variance = sum((x - mean) \*\* 2 for x in data) / len(data)      return math.sqrt(variance)  def gaussian(x, mean, std):      return (1 / (std \* math.sqrt(2 \* math.pi))) \* math.exp(-((x - mean) \*\* 2) / (2 \* std \*\* 2))  def plot\_separate\_membership\_functions(input\_ranges, output\_ranges):      # Постройте входные функции принадлежности.      plt.figure(figsize=(10, 5))      for index, x\_list in enumerate(input\_ranges):          x\_mean = calculate\_mean(x\_list)          x\_std = calculate\_std(x\_list, x\_mean)          y\_values = [gaussian(x, x\_mean, x\_std) for x in x\_list]          plt.plot(x\_list, y\_values, label="Вход Диапазон {}".format(index + 1))      plt.title("Функции принадлежности входных нечетких множеств (Гауссовы)")      plt.xlabel("X")      plt.ylabel("Принадлежность")      plt.legend(loc='upper left')      plt.grid(True)      plt.show()      # Построить выходные функции принадлежности на отдельном графике.      plt.figure(figsize=(10, 5))      for index, (x\_list, style) in enumerate(output\_ranges):          x\_mean = calculate\_mean(x\_list)          x\_std = calculate\_std(x\_list, x\_mean)          y\_values = [gaussian(x, x\_mean, x\_std) for x in x\_list]          plt.plot(x\_list, y\_values, style, label="Выход Диапазон {}".format(index + 1))      plt.title("Функции принадлежности выходных нечетких множеств (Гауссовы)")      plt.xlabel("X")      plt.ylabel("Принадлежность")      plt.legend(loc='upper left')      plt.grid(True)      plt.show()  # Список термов для входных данных  input\_ranges = [      [0, 1000, 2000, 3000], # мало      [2000, 3000, 4000, 5000], # достаточно      [4000, 5000, 6000, 7000] # много  ]  # Список термов для выходных данных  output\_ranges = [      ([0, 1000, 2000, 3000], 'r.-'), # добавить мощность      ([2000, 3000, 4000, 5000], 'g.-'), # не менять      ([4000, 5000, 6000, 7000], 'b.-') # снизить мощность  ]  plot\_separate\_membership\_functions(input\_ranges, output\_ranges)  def fuzzy\_inference(power):      input\_memberships = []      membership\_distributions = []      # Рассчитываем степени принадлежности      for x\_list in input\_ranges:          x\_mean = calculate\_mean(x\_list)          x\_std = calculate\_std(x\_list, x\_mean)          membership = gaussian(power, x\_mean, x\_std)          input\_memberships.append(membership)          membership\_distributions.append(membership)      # Находим индексы для диапазонов, в которые попадает power      matching\_indexes = [index for index, membership in enumerate(input\_memberships) if membership > 0]      # Распределяем принадлежность и обнуляем несоответствующий терм,      # если значение мощности не попадает точно в один из термов      if len(matching\_indexes) < len(input\_ranges):  # если power не попадает во все термы          for index, membership in enumerate(membership\_distributions):              if index not in matching\_indexes:                  membership\_distributions[index] = 0  # обнуляем непопадающий терм              else:                  # Распределяем долю непопадающего терма среди попадающих                  membership\_distributions[index] += sum(input\_memberships) - sum(membership\_distributions)      # Нормализуем значения, чтобы получить проценты      sum\_memberships = sum(membership\_distributions)      input\_memberships\_percentages = [(m / sum\_memberships) \* 100 if sum\_memberships else 0 for m in membership\_distributions]      # Подготавливаем выходные степени принадлежности      output\_memberships = []      for index, (output\_range, \_) in enumerate(output\_ranges):          output\_memberships.append(input\_memberships\_percentages[index])      # Формируем словарь результатов      output\_decision\_percentages = dict(zip(['add\_power', 'keep\_power', 'reduce\_power'], output\_memberships))      return output\_decision\_percentages  # Предположим, что все вспомогательные функции реализованы  # и мы можем проверить измененный метод нечеткого вывода  power = 2900  decision\_percentages = fuzzy\_inference(power)  print(f"\nСтепени принадлежности для мощности {power}: {decision\_percentages}") |

*Приложения 3. Код, реализующий матрицу нечётких отношений и его сжатие*

|  |
| --- |
| import math  import skfuzzy as fuzz  # список термов  ranges = [      [1000, 2000, 3000, 4000],      [3000, 4000, 5000, 6000],      [5000, 6000, 7000, 8000]  ]  # Функция для вычисления среднего значения списка  def mean(lst):      return sum(lst) / len(lst)  # Функция для вычисления стандартного отклонения списка  def stdev(lst):      mean\_value = mean(lst)      return math.sqrt(sum([(x - mean\_value) \*\* 2 for x in lst]) / (len(lst) - 1))  # Функция, которая генерирует список значений принадлежности, используя гауссовскую функцию  def gaussmf(x\_list, mean, std):      return [math.exp(-0.5 \* ((x - mean) / std) \*\* 2) for x in x\_list]  # Рассчитываем функции принадлежности  membership\_functions = [      gaussmf(range\_item, mean(range\_item), stdev(range\_item)) for range\_item in ranges  ]  # Создаем матрицу нечетких отношений для двух наборов (размер матрицы будет len(ranges) x len(ranges))  fuzzy\_relation\_matrix = [      [0 for \_ in range(len(ranges))] for \_ in range(len(ranges))  ]  for i, mfx in enumerate(membership\_functions):      for j, mfy in enumerate(membership\_functions):          # Для целей данного примера, степень будет минимальным значением принадлежности,          # которое имеют общие x и y, вы можете определить это в соответствии с вашими требованиями          degree = min(min(mfx), min(mfy))          fuzzy\_relation\_matrix[i][j] = degree  print("Матрица нечетких отношений:")  for row in fuzzy\_relation\_matrix:      print(row)  # Сжатие матрицы нечетких отношений:  # Обычно это означает сокращение до наиболее значимых отношений.  # Пример: сохранение только самой высокой степени отношения для каждой строки.  compressed\_matrix = [      [0 for \_ in range(len(ranges))] for \_ in range(len(ranges))  ]  for i, row in enumerate(fuzzy\_relation\_matrix):      max\_degree = max(row)      max\_index = row.index(max\_degree)      compressed\_matrix[i][max\_index] = max\_degree  print("\nСжатая матрица нечетких отношений:")  for row in compressed\_matrix:      print(row) |