Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

Теория систем

Лабораторная работа №2

Группа: P3324

Выполнил:

Круглов Егор Ильич

Преподаватель:

Русак Алёна Викторовна

Санкт-Петербург

2025г.

# Задание

Требуется спроектировать нечеткую систему оценки сборки компьютера по трем факторам: цена, потребности и совместимость комплектующих.

# Решение

Код был написан на python, в формате ipynb.

Графики функций принадлежности для входных переменных:

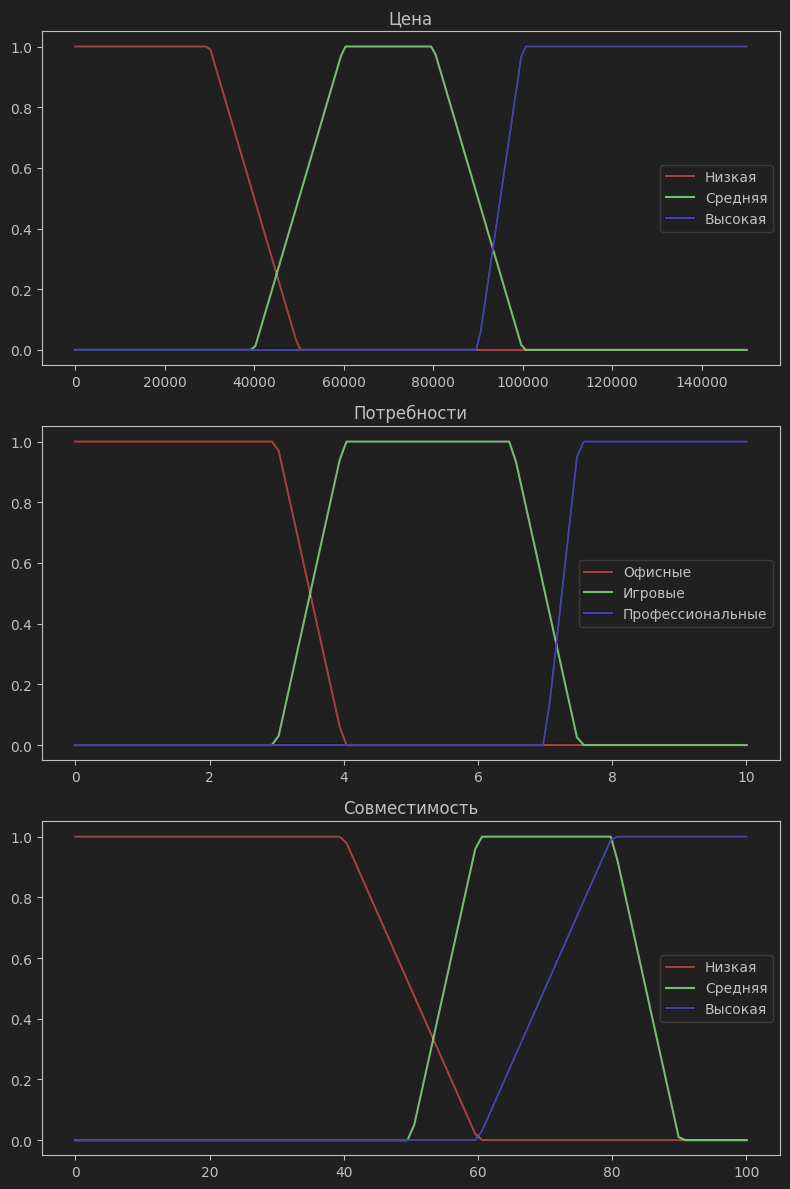
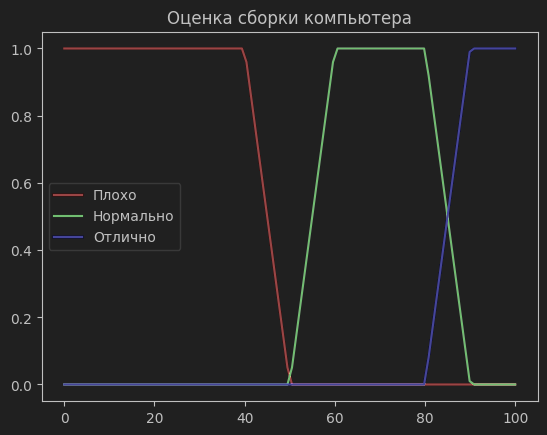


График функции принадлежности для выходной переменной (оценки):



Код представлен на GitHub - <https://github.com/KruglovEgor/System-Theory/tree/main/lab2>

# Установка необходимых библиотек (выполнить один раз)  
!pip install networkx # Для работы с графами (требуется skfuzzy)  
!pip install scikit-fuzzy # Для нечеткой логики

# Импорт библиотек  
import numpy as np  
import skfuzzy as fuzz  
from skfuzzy import control as ctrl  
import matplotlib.pyplot as plt  
  
# ---------------------------  
# 1. Определение универсумов  
# ---------------------------  
x\_price = np.linspace(0, 150000, 150)  
x\_needs = np.linspace(0, 10, 100)   
x\_comp = np.linspace(0, 100, 100)   
x\_score = np.linspace(0, 100, 100)

# ---------------------------  
# 2. Функции принадлежности  
# ---------------------------  
  
# Цена  
price\_lo = fuzz.trapmf(x\_price, [0, 0, 30000, 50000]) # Бюджетная  
price\_md = fuzz.trapmf(x\_price, [40000, 60000, 80000, 100000]) # Средняя  
price\_hi = fuzz.trapmf(x\_price, [90000, 100000, 150000, 150000]) # Дорогая  
  
# Потребности  
needs\_lo = fuzz.trapmf(x\_needs, [0, 0, 3, 4]) # Офисные  
needs\_md = fuzz.trapmf(x\_needs, [3, 4, 6.5, 7.5]) # Игровые  
needs\_hi = fuzz.trapmf(x\_needs, [7, 7.5, 10, 10]) # Профессиональные  
  
# Совместимость  
comp\_lo = fuzz.trapmf(x\_comp, [0, 0, 40, 60]) # Низкая: 0-60%  
comp\_md = fuzz.trapmf(x\_comp, [50, 60, 80, 90]) # Средняя: 40-80%  
comp\_hi = fuzz.trapmf(x\_comp, [60, 80, 100, 100]) # Высокая: 60-100%  
  
# Оценка сборки  
score\_lo = fuzz.trapmf(x\_score, [0, 0, 40, 50]) # Плохо  
score\_md = fuzz.trapmf(x\_score, [50, 60, 80, 90]) # Нормально  
score\_hi = fuzz.trapmf(x\_score, [80, 90, 100, 100]) # Отлично

# ---------------------------  
# 3. Расчет оценки через агрегирование  
# ---------------------------  
def calculate\_score(price\_val, needs\_val, comp\_val):  
 # Фаззификация входных значений  
 price\_act\_lo = fuzz.interp\_membership(x\_price, price\_lo, price\_val)  
 price\_act\_md = fuzz.interp\_membership(x\_price, price\_md, price\_val)  
 price\_act\_hi = fuzz.interp\_membership(x\_price, price\_hi, price\_val)  
   
 needs\_act\_lo = fuzz.interp\_membership(x\_needs, needs\_lo, needs\_val)  
 needs\_act\_md = fuzz.interp\_membership(x\_needs, needs\_md, needs\_val)  
 needs\_act\_hi = fuzz.interp\_membership(x\_needs, needs\_hi, needs\_val)  
   
 comp\_act\_lo = fuzz.interp\_membership(x\_comp, comp\_lo, comp\_val)  
 comp\_act\_md = fuzz.interp\_membership(x\_comp, comp\_md, comp\_val)  
 comp\_act\_hi = fuzz.interp\_membership(x\_comp, comp\_hi, comp\_val)  
  
 # Правила для "Отлично"  
 rule1 = np.fmin(price\_act\_lo, np.fmin(needs\_act\_lo, comp\_act\_hi))  
 rule2 = np.fmin(price\_act\_md, np.fmin(needs\_act\_md, comp\_act\_hi))  
 rule3 = np.fmin(price\_act\_hi, np.fmin(needs\_act\_hi, comp\_act\_hi))  
   
 # Правила для "Нормально"  
 rule4 = np.fmin(price\_act\_lo, np.fmin(needs\_act\_md, comp\_act\_hi))  
 rule5 = np.fmin(price\_act\_lo, np.fmin(needs\_act\_lo, comp\_act\_md))  
 rule6 = np.fmin(price\_act\_md, np.fmin(needs\_act\_lo, comp\_act\_hi))  
 rule7 = np.fmin(price\_act\_md, np.fmin(needs\_act\_md, comp\_act\_md))  
 rule8 = np.fmin(price\_act\_hi, np.fmin(needs\_act\_md, comp\_act\_hi))  
 rule9 = np.fmin(price\_act\_hi, np.fmin(needs\_act\_hi, comp\_act\_md))  
 rule10 = np.fmin(price\_act\_md, np.fmin(needs\_act\_hi, comp\_act\_hi))  
   
 # Активация выходных функций  
 activation\_hi = np.fmax(np.fmax(rule1, rule2), rule3) \* 1.5 # Объединяем правила для "Отлично"  
 activation\_md = np.fmax(  
 np.fmax(np.fmax(rule4, rule5), np.fmax(rule6, rule7)),  
 np.fmax(np.fmax(rule8, rule9), rule10)  
 ) \* 1.5 # Объединяем правила для "Нормально"  
  
 # Все остальные случаи → "Плохо"  
 activation\_lo = np.ones\_like(x\_score) # Базовый уровень активации для "Плохо"  
 activation\_lo = np.fmin(activation\_lo, 1 - np.fmax(activation\_hi, activation\_md)) # Инвертируем логику  
  
 # Агрегируем  
 aggregated = np.fmax(  
 np.fmin(activation\_lo, score\_lo),  
 np.fmax(  
 np.fmin(activation\_md, score\_md),  
 np.fmin(activation\_hi, score\_hi)  
 )  
 )  
  
 # Дефаззификация  
 if np.max(aggregated) == 0:  
 return 0.0 # Минимальная оценка, если все правила провалились  
 return fuzz.defuzz(x\_score, aggregated, 'centroid')

# ---------------------------  
# 4. Визуализация  
# ---------------------------  
fig, ax = plt.subplots()  
ax.plot(x\_score, score\_lo, 'r', label='Плохо')  
ax.plot(x\_score, score\_md, 'g', label='Нормально')  
ax.plot(x\_score, score\_hi, 'b', label='Отлично')  
ax.legend()  
plt.title("Оценка сборки компьютера")  
plt.show()

# Визуализация функций принадлежности для входных переменных  
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(8, 12))  
  
# Цена  
ax1.plot(x\_price, price\_lo, 'r', label='Низкая')  
ax1.plot(x\_price, price\_md, 'g', label='Средняя')  
ax1.plot(x\_price, price\_hi, 'b', label='Высокая')  
ax1.set\_title("Цена")  
ax1.legend()  
  
# Потребности  
ax2.plot(x\_needs, needs\_lo, 'r', label='Офисные')  
ax2.plot(x\_needs, needs\_md, 'g', label='Игровые')  
ax2.plot(x\_needs, needs\_hi, 'b', label='Профессиональные')  
ax2.set\_title("Потребности")  
ax2.legend()  
  
# Совместимость  
ax3.plot(x\_comp, comp\_lo, 'r', label='Низкая')  
ax3.plot(x\_comp, comp\_md, 'g', label='Средняя')  
ax3.plot(x\_comp, comp\_hi, 'b', label='Высокая')  
ax3.set\_title("Совместимость")  
ax3.legend()  
  
plt.tight\_layout()  
plt.show()

# ---------------------------  
# 5. Тестирование  
# ---------------------------  
import random  
  
for \_ in range(5):  
 price = random.randint(20000, 150000)  
 needs = random.randint(0, 10)  
 comp = random.randint(40, 100)  
 print(f"Цена: {price}₽, Потребности: {needs}, Совместимость: {comp}%")  
 print(f"→ Оценка: {calculate\_score(price, needs, comp):.1f}%")

**Пример тестирования:**

Цена: 53644₽, Потребности: 10, Совместимость: 51%

→ Оценка: 22.6%

Цена: 127201₽, Потребности: 9, Совместимость: 64%

→ Оценка: 72.5%

Цена: 51833₽, Потребности: 0, Совместимость: 95%

→ Оценка: 62.4%

Цена: 141670₽, Потребности: 3, Совместимость: 95%

→ Оценка: 24.0%

Цена: 102912₽, Потребности: 1, Совместимость: 80%

→ Оценка: 22.6%

# Вывод

В ходе выполнения лабораторной работы была разработана нечёткая система оценки качества сборки компьютера на основе трёх параметров: цены, потребностей пользователя и совместимости комплектующих. Также были:

1. Определены лингвистические переменные для входных и выходных данных:
   * Цена: низкая, средняя, высокая.
   * Потребности: офисные, игровые, профессиональные.
   * Совместимость: низкая, средняя, высокая.
   * Оценка: плохо, нормально, отлично.
2. Построены функции принадлежности для всех переменных.
3. Сформулированы правила нечёткого вывода (10 правил), охватывающие различные комбинации входных параметров.