Министерство образования Российской Федерации

МОСКВОСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э.БАУМАНА

Факультет: Информатика и системы управления (ИУ)
Кафедра: Информационная безопасность (ИУ8)

МЕТОДЫ ОПТИМИЗАЦИИ

Лабораторная работа №4 на тему:

«Решение задачи многокритериальной оптимизации»

Вариант – 1

Преподаватель:
Коннова Н.С.

Студент:
Александров А. Н.

Группа:
ИУ8-34

Москва, 2020

Цель работы: изучить постановку задачи многокритериальной оптимизации (МКО); овладеть навыками решения задач МКО с помощью различных методов, осуществить сравнительный анализ результатов, полученных при помощи различных методов.

Постановка задачи: выбрать лучшую из альтернатив решения предложенной задачи по варианту из таблицы с точки зрения указанных критериев следующими методами:

1) Заменой критериев ограничениями;
2) Формирование и сужение множества Парето;
3) Методом взвешивания и объединения критериев;
4) Методом анализа иерархий;

Задача МКО в соответствии с вариантом:

Задача, альтернативы:

Покупка автомобиля:

A. Suzuki;
B. Mitsubishi;
C. Honda;
D. Toyota.

Критерии:

1. Стоимость;
2. Расходы на обслуживание;
3. Расход бензина;
4. Комфорт.

Описание предпочтений:

Стоимость: Suzuki существенно дороже всех, Honda немного дороже Mitsubishi, Toyota существенно дешевле всех.

Расходы на обслуживание: Mitsubishi дороже всех, Toyota и Suzuki примерно равны, Honda дешевле всех.

Расход бензина: самый высокий у Suzuki, немного меньше у Honda, существенно меньше у Mitsubishi, самый низкий – у Toyota.

Комфорт: самая комфортная – Toyota, чуть менее – Mitsubishi, существенно хуже – Honda, самая некомфортная – Suzuki.

Ход решения:

Для решения задачи МКО была написана программа на языке **Python** (см. Приложение А):

В файле ***multicriteria.py*** реализован класс ***Multicriteria,*** в котором описана логика решения задачи всеми четырьмя способами.

Поля класса ***Multicriteria:***

Поле **task_name_:** хранит название задачи.

Поле **alternative_names_:** хранит названия альтернатив выбора.

Поле **criteria_names_:** хранит названия критериев.

Поле **criteria_weight_:** хранит вектор весов критериев.

Поле **json_matrix_:** хранит матрицу альтернатив, полученную из json -файла.

Поле **alternative_matrix_:** хранит матрицу оценок альтернатив.

Поле **normalized_matrix_:** хранит нормализированную матрицу оценок альтернатив.

Методы класса ***Multicriteria:***

Метод **NormingVector(self, vector):** нормирует копию переданного вектора и возвращает её.

Метод **NormalizeMatrix(self, matrix):** нормализует копию переданной матрицы и возвращает её.

Метод **OutMatrix(self, matrix):** выводит матрицу альтернатив на экран.

Метод **OutWeight(self):** выводит вектор весов критериев на экран.

Метод **MainCriteriaMethod(self):** производит решение задачи методом главного критерия.

Метод **ParetoMethod(self):** производит решение задачи методом сужения множества Парето.

Метод **NormalizeByColumns(self, current_matrix):** нормирует столбцы копии переданной марицы и возвращает её.

Метод **CriteriaEvaluation(self):** составляет матрицу экспертной оценки критериев.

Метод **WeighAndCombineMethod(self):** производит решение задачи методом взвешивания и объединения критериев.

Метод **PairCompareMatrix(self, fill_list):** заполняет матрицу попарных сравнений.

Метод **PairCompareTable(self, names, main_matrix, sum_col):** заполняет таблицу попарных сравнений для вывода на экран.

Метод **ConsensusDivision(self, main_matrix, normalize_sum_col):** находит отношение согласованности.

Метод **HierarchiesAnalysisMethod(self):** производит решение задачи методом анализа иерархий.

Выполнение программы и решение задачи:

1) Составим вектор весов критериев.

```
Ход работы:
Составляем веткор весов критериев, используя шкалу 1-10:
+-----------+-------------------------+----------------+---------+
| Стоимость | Расходы на обслуживание | Расход бензина | Комфорт |
+-----------+-------------------------+----------------+---------+
|     4     |            6            |       8        |    2    |
+-----------+-------------------------+----------------+---------+
Нормализовав, получим вектор [0.2 0.3 0.4 0.1]
```

Рисунок 1 Вектор весов критериев.

2) **Метод главного критерия (замена критериев ограничениями):**

Главным критерием выберем расход бензина f_3.

```
1) Метод замены критериев ограничениями (метод главного критерия).
Составим матрицу оценок альтернатив.
+---------------+-----------+------------------------+----------------+---------+
|  Альтернативы | Стоимость | Расходы на обслуживание | Расход бензина | Комфорт |
+---------------+-----------+------------------------+----------------+---------+
|   A. Suzuki   |    8.0    |          5.0           |      9.0       |   2.0   |
| B. Mitsubishi |    6.0    |          9.0           |      3.0       |   7.0   |
|    C. Honda   |    5.0    |          2.0           |      7.0       |   3.0   |
|   D. Toyota   |    2.0    |          5.0           |      2.0       |   8.0   |
+---------------+-----------+------------------------+----------------+---------+
Ограничения:
Стоимость не менее 0.2
Расходы на обслуживание не менее 0.3
Комфорт не менее 0.5

Проведём нормирование матрицы:
+---------------+-----------+------------------------+----------------+---------+
|  Альтернативы | Стоимость | Расходы на обслуживание | Расход бензина | Комфорт |
+---------------+-----------+------------------------+----------------+---------+
|   A. Suzuki   |    1.0    |          0.43          |      9.0       |   0.0   |
| B. Mitsubishi |    0.67   |          1.0           |      3.0       |   0.83  |
|    C. Honda   |    0.5    |          0.0           |      7.0       |   0.17  |
|   D. Toyota   |    0.0    |          0.43          |      2.0       |   1.0   |
+---------------+-----------+------------------------+----------------+---------+
При заданных ограничениях приемлимыми являются следующие решения:
D. Toyota
Итоговое решение:
D. Toyota
```

Рисунок 2 Метод главного критерия

При заданных условиях приемлимым и единственным является решение: альтернатива *A . Toyota.*

3) **Формирование и сужение множества Парето:**

Выберем два критерия: $f_2 u f_3$. С помощью библиотеки **Matplotlib** сформируем графически множество Парето:

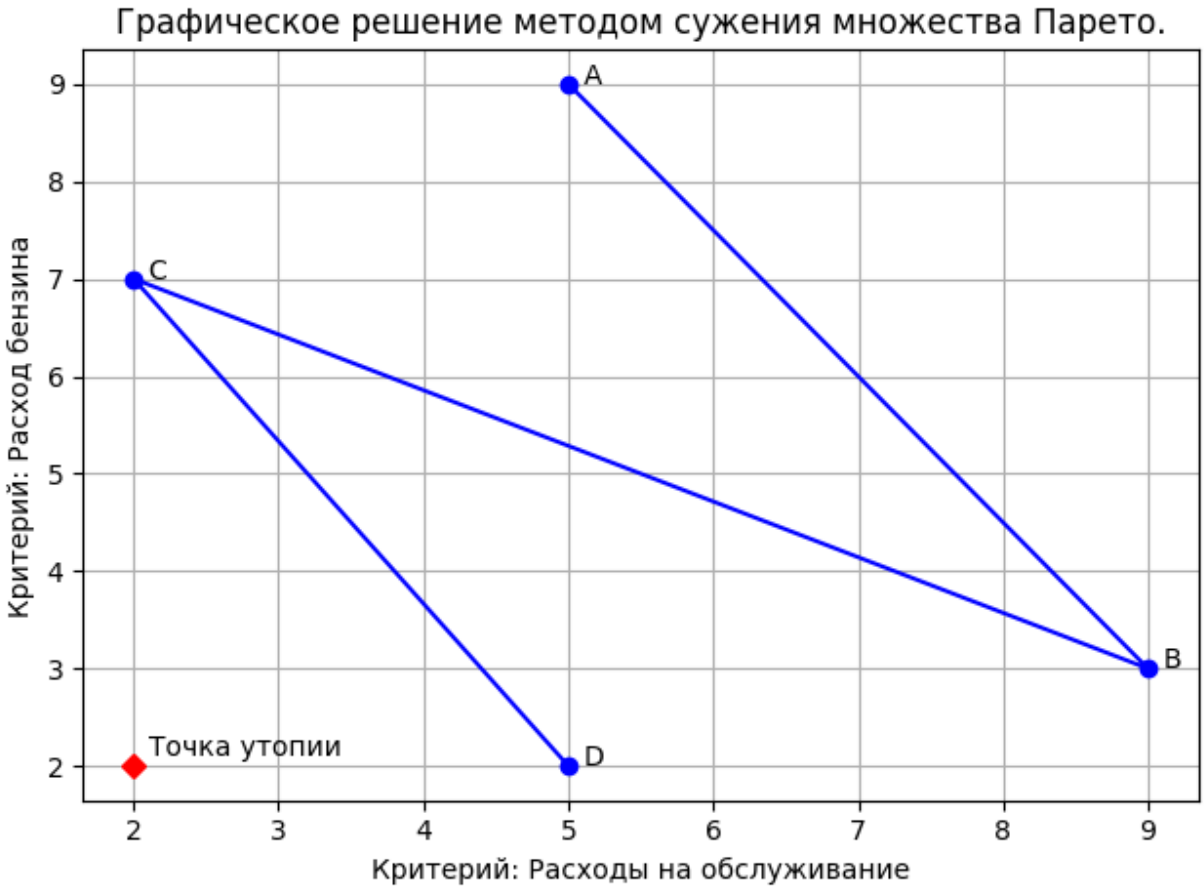


Рисунок 3 Метод сужения множества Парето

Выберем оптимальную альтернативу из множества Парето по минимуму Евклидова расстояния до точки утопии.

$$dist\left(x_i, x_j\right)=\sqrt{\sum_k \left(x_{ik}-x_{jk}\right)^2}$$

Минимальное у альтернативы *A. Toyota*, а значит она оптимальна при заданных критериях.

3) Взвешивание и объединение критериев (линейная свертка):

```
 3) Взвешивание и объединение критериев.
Составим матрицу рейтингов альтернатив по критериям, используя шкалу 1-10:

 +---------------+-----------+-------------------------+----------------+---------+
|  Альтернативы | Стоимость | Расходы на обслуживание | Расход бензина | Комфорт |
+---------------+-----------+-------------------------+----------------+---------+
|   A. Suzuki   |    8.0    |           5.0           |      9.0       |   2.0   |
| B. Mitsubishi |    6.0    |           9.0           |      3.0       |   7.0   |
|    C. Honda   |    5.0    |           2.0           |      7.0       |   3.0   |
|   D. Toyota   |    2.0    |           5.0           |      2.0       |   8.0   |
+---------------+-----------+-------------------------+----------------+---------+

 Нормализуем её:
+---------------+-----------+-------------------------+----------------+---------+
|  Альтернативы | Стоимость | Расходы на обслуживание | Расход бензина | Комфорт |
+---------------+-----------+-------------------------+----------------+---------+
|   A. Suzuki   |    0.33   |           0.21          |      0.38      |   0.08  |
| B. Mitsubishi |    0.24   |           0.36          |      0.12      |   0.28  |
|    C. Honda   |    0.29   |           0.12          |      0.41      |   0.18  |
|   D. Toyota   |    0.12   |           0.29          |      0.12      |   0.47  |
+---------------+-----------+-------------------------+----------------+---------+

Составим экспертную оценку критериев (по методу попарного сравнения):

+-------------------------+-----------+-------------------------+----------------+---------+
|                         | Стоимость | Расходы на обслуживание | Расход бензина | Комфорт |
+-------------------------+-----------+-------------------------+----------------+---------+
|        Стоимость        |     0     |           0.5           |       0        |    1    |
| Расходы на обслуживание |    0.5    |            0            |       0        |    1    |
|      Расход бензина     |     1     |            1            |       0        |    1    |
|         Комфорт         |     0     |            0            |       0        |    0    |
+-------------------------+-----------+-------------------------+----------------+---------+
alpha = [0.25 0.25 0.5  0.  ]
Умножив нормализированную матрицу на нормализированный вектор весов критериев, получаем значения объединённого критерия альтернатив:
[0.25       0.26136364 0.26136364 0.2578125 ]
Наиболее приемлемой является альтернатива:
C. Honda
```

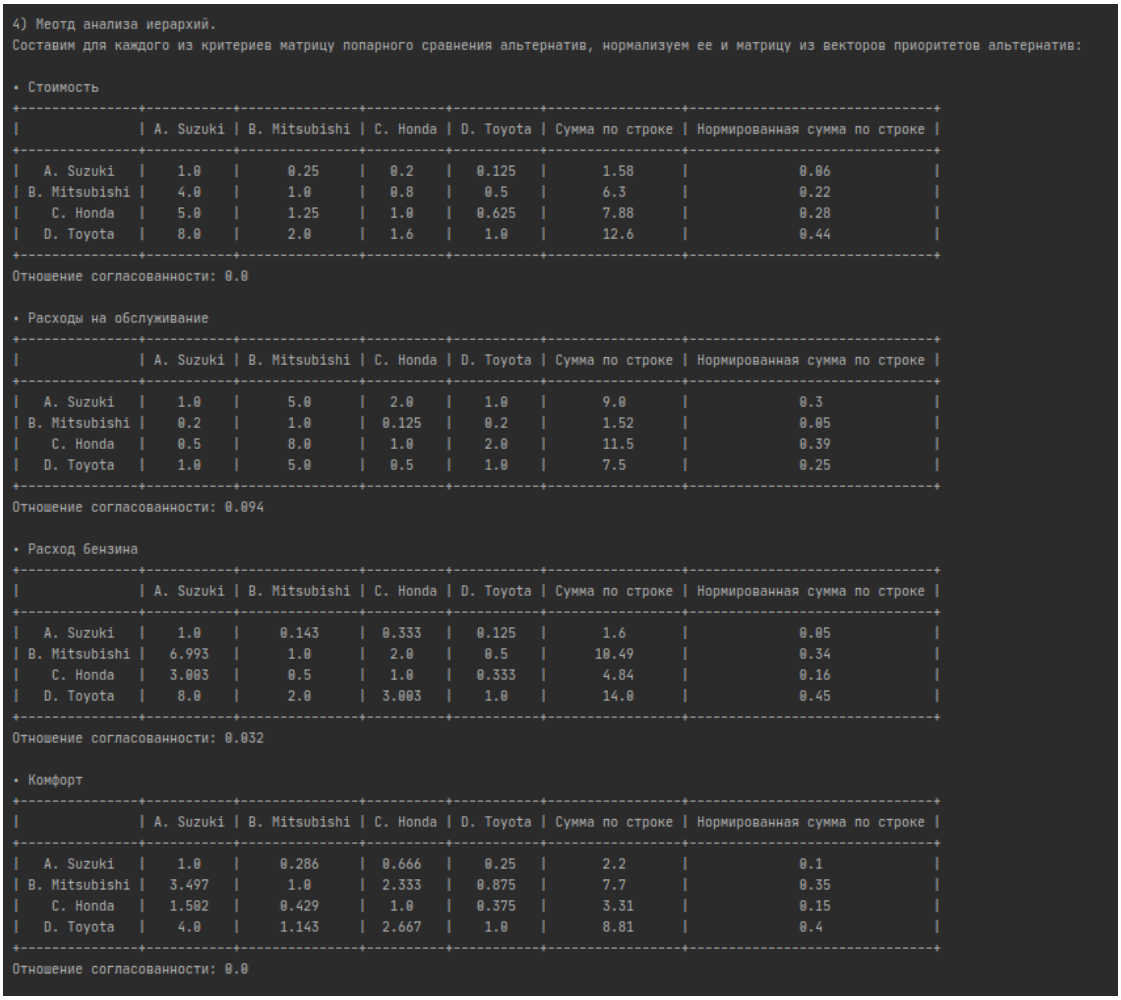
Рисунок 4 Метод взвешивания и объединения критериев

Перемножим полученные нами раннее нормализированные матрицу рейтингов альтернатив и вектор весов критериев. Получим значения объединенного критерия для всех альтернатив. В этом векторе ищем максимальное значение, оно и покажет нам альтернативу:

Здесь лучшей альтернативой стала *B. Honda.*

4) Метод анализа иерархий.

Для каждого критерия составим и нормализуем матрицу попарных альтернатив:



4) Меотд анализа иерархий.
Составим для каждого из критериев матрицу попарного сравнения альтернатив, нормализуем ее и матрицу из векторов приоритетов альтернатив:

• Стоимость

	A. Suzuki	B. Mitsubishi	C. Honda	D. Toyota	Сумма по строке	Нормированная сумма по строке
A. Suzuki	1.0	0.25	0.2	0.125	1.58	0.06
B. Mitsubishi	4.0	1.0	0.8	0.5	6.3	0.22
C. Honda	5.0	1.25	1.0	0.625	7.88	0.28
D. Toyota	8.0	2.0	1.6	1.0	12.6	0.44

Отношение согласованности: 0.0

• Расходы на обслуживание

	A. Suzuki	B. Mitsubishi	C. Honda	D. Toyota	Сумма по строке	Нормированная сумма по строке
A. Suzuki	1.0	5.0	2.0	1.0	9.0	0.3
B. Mitsubishi	0.2	1.0	0.125	0.2	1.52	0.05
C. Honda	0.5	8.0	1.0	2.0	11.5	0.39
D. Toyota	1.0	5.0	0.5	1.0	7.5	0.25

Отношение согласованности: 0.094

• Расход бензина

	A. Suzuki	B. Mitsubishi	C. Honda	D. Toyota	Сумма по строке	Нормированная сумма по строке
A. Suzuki	1.0	0.143	0.333	0.125	1.6	0.05
B. Mitsubishi	6.993	1.0	2.0	0.5	10.49	0.34
C. Honda	3.003	0.5	1.0	0.333	4.84	0.16
D. Toyota	8.0	2.0	3.003	1.0	14.0	0.45

Отношение согласованности: 0.032

• Комфорт

	A. Suzuki	B. Mitsubishi	C. Honda	D. Toyota	Сумма по строке	Нормированная сумма по строке
A. Suzuki	1.0	0.286	0.666	0.25	2.2	0.1
B. Mitsubishi	3.497	1.0	2.333	0.875	7.7	0.35
C. Honda	1.502	0.429	1.0	0.375	3.31	0.15
D. Toyota	4.0	1.143	2.667	1.0	8.81	0.4

Отношение согласованности: 0.0

Рисунок 5 Матрицы попарного сравнения альтернатив для каждого критерия

Видим, что для всех матриц отношение согласованности не превосходит 0.1.

Оценка приоритетов критериев:

```
Оценка приоритетов:
+------------------------+-----------+------------------------+----------------+---------+----------------+------------------------------+
|                        | Стоимость | Расходы на обслуживание | Расход бензина | Комфорт | Сумма по строке | Нормированная сумма по строке |
+------------------------+-----------+------------------------+----------------+---------+----------------+------------------------------+
|       Стоимость        |    1.0    |         0.666          |      0.5       |   2.0   |      4.17      |             0.19             |
| Расходы на обслуживание |   1.502   |          1.0           |     0.666      |   4.0   |      7.17      |             0.33             |
|     Расход бензина     |    2.0    |         1.502          |      1.0       |   4.0   |      8.5       |             0.39             |
|        Комфорт         |    0.5    |          0.25          |      0.25      |   1.0   |      2.0       |             0.09             |
+------------------------+-----------+------------------------+----------------+---------+----------------+------------------------------+
Отношение согласованности: 0.008

[0.13997303 0.22347313 0.25547617 0.38107767]
Умножив матрицу, составленную из норм. сумм по строкам на вектор-столбец оценки приоритетов, получим вектор:
Наиболее приемлемой является альтернатива:
D. Toyota
```

Рисунок 6 Оценка приоритетов критериев

Умножив матрицу, составленную из нормализированных сумм по строкам на вектор-столбец оценки приоритетов, получим вектор $(0.14\ 0.22\ 0.26\ 0.38)^T$. Оценив вектор, получаем оптимальный вариант *D. Toyota.*

Вывод:

В ходе работы была изучена постановка задачи многокритериальной оптимизации, были получены навыки решения задач МКО с помощью различных методов.
По результатам эксперимента можно увидеть, что не все способы решения дали одинаковое решение. Так результатом метода главного критерия, метода множества Парето, метода анализа иерархий является Toyota, а в методе взвешивания и объединения критериев наилучшая альтернатива – Honda.

В данной работе были рассмотрены методы решения задач, которые могут быть использованы при выборе различных альтернатив по ряду критериев, что может помочь в самых различных областях и ситуациях: выбор автомобиля, материала для строительства, места отдыха и так далее.

Приложение А:

Код программы:

1) input_data.json

```
{
  "task_name": "Покупка автомобиля",
  "alternative_names": ["A. Suzuki", "B. Mitsubishi", "C. Honda", "D. Toyota"],
  "criteria_names": ["Стоимость", "Расходы на обслуживание", "Расход
бензина", "Комфорт"],
  "criterias_weight": [4, 6, 8, 2],
  "criterias_direction": ["min", "min" ,"min" , "max"],
  "alternative_matrix": [
    [8, 5, 9, 2],
    [6, 9, 3, 7],
    [5, 2, 7, 3],
    [2, 5, 2, 8]
  ]
}
```

2) main.py

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

"""
Лабораторная работа № 5
Решение многокритериальной оптимизации.
Цель работы: Изучить постановку задачи МКО;
овладеть навыками решения задач МКО с помощью различных
методов,
осуществить сравнительный анализ результатов, полученных при
помощи разных методов.

Вариант 1.
"""
from multicriteria import *

if __name__ == '__main__':
    print("Ход работы:")
    multcrit = Multicriteria("input_data.json")
    print(multcrit.OutWeight())

    multcrit.MainCriteriaMethod()
    multcrit.ParetoMethod()
    multcrit.WeighAndCombineMethod()
    multcrit.HierarchiesAnalysisMethod()
```

2) multicriteria.py

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

import json

import numpy as np
from prettytable import PrettyTable
import matplotlib.pyplot as plt
from scipy.spatial import distance

CRITERIA_COUNT = 4
MAIN_CRITERIA_INDEX = 2  # Главный критерий -- расход бензина.
MAIN_CRITERIA_LIMITS = [0.2, 0.3, 0, 0.5]  # Минимально допустимые доли критериев.

MAX_WEIGHT = 10

SECOND_CRITERIA_INDEX = 1
THIRD_CRITERIA_INDEX = 2

COST_FILL_LIST = [0.25, 0.2, 0.125, 0.8, 0.5, 0.625]
EXPENSES_FILL_LIST = [5, 2, 1, 0.125, 0.2, 2]
CONSUMPTION_FILL_LIST = [0.143, 0.333, 0.125, 2, 0.5, 0.333]
COMFORT_FILL_LIST = [0.286, 0.666, 0.25, 2.333, 0.875, 0.375]
CRITERIA_FILL_LIST = [0.666, 0.5, 2, 0.666, 4, 4]


class Multicriteria:
    """Базовый класс задачи многокритериальной оптимизации"""

    def __init__(self, path_to_file):
        # Парсим JSON-файл с входными данными
        with open(path_to_file, "r") as read_file:
            json_data = json.load(read_file)
            # Задача.
            self.task_name_ = json_data["task_name"]
            # Альтернативы.
            self.alternative_names_ = list(json_data["alternative_names"])
            # Критерии.
            self.criteria_names_ = list(json_data["criteria_names"])

            # Вектор весов критериев.
            self.criteria_weight_ = np.array(json_data["criterias_weight"])
```

```
            # Нормализованный вектор весов критериев.
            self.normalized_weight_ = self.NormingVector(self.criteria_weight_)
            # Матрица A оценок для альтернатив.
            self.json_matrix_ = np.array(json_data["alternative_matrix"],
dtype=np.float64)
            self.alternative_matrix_ =
self.AlternativeMatrix(np.array(json_data["alternative_matrix"],
                                                    dtype=np.float64),
                                                list(json_data["criterias_direction"]))
            # Нормализованная матрица.
            self.normalized_matrix_ =
self.NormalizeMatrix(self.alternative_matrix_)

    def NormingVector(self, vector):
        """Нормирует вектор."""
        normalize_weight = vector.copy()
        weight_sum = np.sum(normalize_weight)

        normalize_weight = normalize_weight / weight_sum

        return normalize_weight

    def AlternativeMatrix(self, alternative_matrix, criterias_direction):
        """Своидит все криитерии к максимизации."""
        for j in range(len(criterias_direction)):
            if criterias_direction[j] == "min":
                for i in range(alternative_matrix.shape[0]):
                    alternative_matrix[i][j] = MAX_WEIGHT - alternative_matrix[i][j]
+ 1

        return alternative_matrix

    def NormalizeMatrix(self, matrix):
        """Нормализует матрицу."""
        normalized_matrix = matrix.copy()
        minimums = normalized_matrix.min(axis=0)
        maximums = normalized_matrix.max(axis=0)

        for j in range(normalized_matrix.shape[1]):
            if j != MAIN_CRITERIA_INDEX:
                for i in range(matrix.shape[0]):
                    normalized_matrix[i][j] = (normalized_matrix[i][j] - minimums[j])
/ (
                            maximums[j] - minimums[j])

        return normalized_matrix
```

```
    def OutMatrix(self, matrix):
        """Выводит матрицу альтернатив."""
        table = PrettyTable()

        table.field_names = ["Альтернативы"] + self.criteria_names_

        for i in range(len(self.alternative_names_)):
            new_row = [self.alternative_names_[i]]
            for j in range(len(self.criteria_names_)):
                new_row.append(round(matrix[i][j], 2))

            table.add_row(new_row)

        return table

    def OutWeight(self):
        """Выводит вектор весов критериев."""
        out = "Составляем веткор весов критериев, используя шкалу 1-10:\
n"
        table = PrettyTable()
        table.field_names = self.criteria_names_
        table.add_row(self.criteria_weight_)

        out += table.__str__()

        out += "\nНормализовав, получим вектор " +
self.normalized_weight_.__str__()

        return out

    def MainCriteriaMethod(self):
        """Решение методом главного критерия."""
        print("\n1) Метод замены критериев ограничениями (метод
главного критерия).\n"
              "Составим матрицу оценок альтернатив.")
        print(self.OutMatrix(self.json_matrix_))

        matrix = self.normalized_matrix_.copy()
        maximums = matrix.max(axis=0)

        print("Ограничения:")
        for j in range(len(self.criteria_names_)):
            if j != MAIN_CRITERIA_INDEX:
                print(f"{self.criteria_names_[j]} не менее
{MAIN_CRITERIA_LIMITS[j] * maximums[j]}")
```

```
        print(f"\nПроведём нормирование матрицы:\
n{self.OutMatrix(self.NormalizeMatrix(self.json_matrix_))}")

        constraints = []
        for j in range(len(self.criteria_names_)):
            if j == MAIN_CRITERIA_INDEX:
                constraints.append(None)
            else:
                constraints.append(MAIN_CRITERIA_LIMITS[j] * maximums[j])

        acceptable_rows = []

        for i in range(len(self.alternative_names_)):
            row = matrix[i]
            if (row < MAIN_CRITERIA_LIMITS).any():
                continue

            acceptable_rows.append(i)

        if len(acceptable_rows):
            print("При заданных ограничениях приемлимыми являются
следующие решения:")
            for i in acceptable_rows:
                print(self.alternative_names_[i])

            max_alternative = None
            for i in acceptable_rows:
                curr = self.normalized_matrix_[i][MAIN_CRITERIA_INDEX]
                if max_alternative is None or
self.normalized_matrix_[max_alternative][MAIN_CRITERIA_INDEX] < curr:
                    max_alternative = i

            print("Итоговое решение:")
            print(self.alternative_names_[max_alternative])

        else:
            print("При заданных ограничениях не нашлось приемлимых
решений.")

    def ParetoMethod(self):
        """Решение формированием и сужением множества Парето."""
        print(f"\n 2) Формирование и сужение множества Парето. \n"
              f"Выберем в качестве критериев для данного метода
{self.criteria_names_[SECOND_CRITERIA_INDEX]} и "
              f"{self.criteria_names_[THIRD_CRITERIA_INDEX]}.\n"
```

```
            f"{self.criteria_names_[SECOND_CRITERIA_INDEX]} - по оси X, "
            f"{self.criteria_names_[THIRD_CRITERIA_INDEX]} - по оси Y.\n"
            f"Сформируем множество Парето графическим методом. (см. график)")
        plt.title("Графическое решение методом сужения множества Парето.")
        plt.xlabel(f"Критерий: {self.criteria_names_[SECOND_CRITERIA_INDEX]}")
        plt.ylabel(f"Критерий: {self.criteria_names_[THIRD_CRITERIA_INDEX]}")

        xValues = self.json_matrix_[:, SECOND_CRITERIA_INDEX]
        yValues = self.json_matrix_[:, THIRD_CRITERIA_INDEX]
        plt.grid()
        plt.plot(xValues, yValues, "b")

        euclid_length = []
        for i in range(len(self.json_matrix_[:, SECOND_CRITERIA_INDEX])):
            x_i = self.json_matrix_[i, SECOND_CRITERIA_INDEX]
            y_i = self.json_matrix_[i, THIRD_CRITERIA_INDEX]
            plt.plot(x_i, y_i, "bo")
            plt.text(x_i + 0.1, y_i, self.alternative_names_[i][0])

            euclid_distance = distance.euclidean((x_i, y_i), (xValues.min(), yValues.min()))

            euclid_length.append(euclid_distance)

        plt.plot(xValues.min(), yValues.min(), "rD")
        plt.text(xValues.min() + 0.1, yValues.min() + 0.1, "Точка утопии")

        plt.show()
        plt.savefig("pareto.png")

        min_index = min(enumerate(euclid_length), key=lambda x: x[1])[0]

        print(f"Исходя из графика можно сказать, что Евклидово расстояние до "
              f"точки минимально для варианта:\n{self.alternative_names_[min_index]}")

    def NormalizeByColumns(self, current_matrix):
        """Нормализует колонки в матрице."""
        matrix = current_matrix.copy()
        for i in range(len(self.criteria_names_)):
            col_sum = np.sum(matrix[i])
            matrix[i] = matrix[i] / col_sum
```

```
        return matrix

    def CriteriaEvaluation(self, y12, y13, y14, y23, y24, y34):
        """Составляет """
        table = PrettyTable()
        table.field_names = [""] + self.criteria_names_
        table.add_row([self.criteria_names_[0]] + [0, y12, y13, y14])
        table.add_row([self.criteria_names_[1]] + [1 - y12, 0, y23, y24])
        table.add_row([self.criteria_names_[2]] + [1 - y13, 1 - y23, 0, y34])
        table.add_row([self.criteria_names_[3]] + [1 - y14, 1 - y24, 1 - y34, 0])

        return table

    def WeighAndCombineMethod(self):
        """Решение методом взвешивания и объединения критериев."""
        rating_matrix = self.NormalizeByColumns(self.alternative_matrix_)
        rm = self.NormalizeByColumns(self.json_matrix_)

        print("\n 3) Взвешивание и объединение критериев. \n"
              f"Составим матрицу рейтингов альтернатив по критериям,
используя шкалу 1-10: \n\n "
              f"{self.OutMatrix(self.json_matrix_)} \n\n Нормализуем её: \n"
              f"{self.OutMatrix(rm)}\n")

        print("Составим экспертную оценку критериев (по методу
попарного сравнения):\n")
        y12 = 0.5
        y13 = 0
        y14 = 1
        y23 = 0
        y24 = 1
        y34 = 1
        print(self.CriteriaEvaluation(y12, y13, y14, y23, y24, y34))

        weight_vector = np.array([y12 + y13 + y14, y12 + y14, y14 + y24 +
y34, 0])

        weight_vector = self.NormingVector(weight_vector)

        print(f"alpha = {weight_vector}")

        weight_vector.transpose()

        combine_criteria = rating_matrix.dot(weight_vector)
```

```
        print(f"Умножив нормализированную матрицу на
нормализированный вектор весов критериев, "
              f"получаем значения объединённого критерия альтернатив:\
n{combine_criteria}")

        max_index = None
        for i in range(len(combine_criteria) - 1, 0, -1):
            if max_index is None or combine_criteria[i] >
combine_criteria[max_index]:
                max_index = i

        print(f"Наиболее приемлемой является альтернатива:\
n{self.alternative_names_[max_index]}")

    def PairCompareMatrix(self, fill_list):
        """Заполянет матрицу попарных сравнений."""
        k = 0
        pc_matrix = np.ones((CRITERIA_COUNT, CRITERIA_COUNT))
        # Заполняем верхний треугольник.
        for i in range(CRITERIA_COUNT):
            for j in range(CRITERIA_COUNT):
                if i < j:
                    pc_matrix[i][j] = round(fill_list[k], 3)
                    k += 1

        k = 0
        # Заполняем нижний треугольник.
        for i in range(CRITERIA_COUNT):
            for j in range(CRITERIA_COUNT):
                if i < j:
                    pc_matrix[j][i] = round(1 / fill_list[k], 3)
                    k += 1

        return pc_matrix

    def PairCompareTable(self, names, main_matrix, sum_col,
normalize_sum_col):
        """Составляет таблицу с матрицей попарных сравнений"""
        table = PrettyTable()
        table.field_names = [""] + names + ["Сумма по строке",
"Нормированная сумма по строке"]
        for i in range(len(self.alternative_names_)):
            row = [names[i]] + list(main_matrix[i])
            row.append(round(sum_col[i], 2))
            row.append(round(normalize_sum_col[i], 2))
            table.add_row(row)
```

```
        return table

    def ConsensusDivision(self, main_matrix, normalize_sum_col):
        """Находит отношение согласованности."""
        columns_sum = np.sum(main_matrix, axis=0)
        mult_col = columns_sum * normalize_sum_col
        return (np.sum(mult_col) - CRITERIA_COUNT) / (CRITERIA_COUNT - 1)

    def HierarchiesAnalysisMethod(self):
        """Решение методом анализа иерархий."""
        print("\n4) Меотд анализа иерархий.\nСоставим для каждого из
критериев матрицу попарного сравнения альтернатив,"
              " нормализуем ее и матрицу из векторов приоритетов
альтернатив:\n")

        fill_lists = [COST_FILL_LIST, EXPENSES_FILL_LIST,
CONSUMPTION_FILL_LIST, COMFORT_FILL_LIST]

        hierarchies_matrix = None

        for i in range(len(self.criteria_names_)):
            print(f"• {self.criteria_names_[i]}")
            main_matrix = self.PairCompareMatrix(fill_lists[i])

            sum_col = np.sum(main_matrix, axis=1)

            normalize_sum_col = self.NormingVector(sum_col)
            print(self.PairCompareTable(self.alternative_names_, main_matrix,
sum_col, normalize_sum_col))

            print(f"Отношение согласованности:
{round(self.ConsensusDivision(main_matrix, normalize_sum_col), 3)}\n")

            if hierarchies_matrix is None:
                hierarchies_matrix = normalize_sum_col.transpose()
            else:
                hierarchies_matrix = np.c_[hierarchies_matrix,
normalize_sum_col.transpose()]

        print("Оценка приоритетов:")
        criteria_matrix = self.PairCompareMatrix(CRITERIA_FILL_LIST)
        sum_col = np.sum(criteria_matrix, axis=1)

        normalize_sum_col = self.NormingVector(sum_col)
        print(self.PairCompareTable(self.criteria_names_, criteria_matrix,
```

```
sum_col, normalize_sum_col))

        print(f"Отношение согласованности:
{round(self.ConsensusDivision(criteria_matrix, normalize_sum_col), 3)}\n")

        normalize_sum_col.transpose()

        resulted_vec = hierarchies_matrix.dot(normalize_sum_col.transpose())

        print(resulted_vec)

        print("Умножив матрицу, составленную из норм. сумм по строкам
на вектор-столбец оценки приоритетов, "
              "получим вектор:")

        max_index = np.argmax(resulted_vec)

        print(f"Наиболее приемлемой является альтернатива:\
n{self.alternative_names_[max_index]}")
```