**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**(Финансовый университет)**

**Факультет**

**информационных технологий и анализа больших данных**

**Кафедра «Бизнес-информатика»**

**Расчетно-аналитическая работа**

Студент группы БИ20-5:

Руководитель:

Аксенов Дмитрий Андреевич

**Москва 2022**

**Оглавление**

[**ТРАНСПОРТНАЯ ЗАДАЧА 8**](#_Toc106203992)

[1. Физическая модель транспортной задачи 8](#_Toc106203993)

[2. Математическая модель транспортной задачи 9](#_Toc106203994)

[3. Алгоритмы решения транспортной задачи 12](#_Toc106203995)

[3.1. Метод линейного программирования 13](#_Toc106203996)

[3.1.1. Описание входных данных 13](#_Toc106203997)

[3.1.2. Описание алгоритма и архитектуры решения 14](#_Toc106203998)

[3.1.3. Описание выходных данных 17](#_Toc106203999)

[3.2. Симплекс-метод 18](#_Toc106204000)

[3.2.1. Описание входных данных 18](#_Toc106204001)

[3.2.2. Описание алгоритма и архитектуры решения 18](#_Toc106204002)

[3.2.3. Описание выходных данных 19](#_Toc106204003)

[3.3. Графический метод 20](#_Toc106204004)

[3.3.1. Описание входных данных 20](#_Toc106204005)

[3.3.2. Описание алгоритма и архитектуры решения 20](#_Toc106204006)

[3.3.3. Описание выходных данных 24](#_Toc106204007)

[4. Варианты использования системы 26](#_Toc106204008)

[4.1. ВИ 1 26](#_Toc106204009)

[5. Тестирование 28](#_Toc106204010)

[6. Заключение 29](#_Toc106204011)

[**ОПТИМИЗАЦИЯ НА ГРАФАХ: ЗАДАЧА КОММИВОЯЖЕРА 30**](#_Toc106204012)

[1. Физическая модель задачи 30](#_Toc106204013)

[2. Математическая модель задачи 31](#_Toc106204014)

[3. Алгоритмы решения задачи 34](#_Toc106204015)

[3.1. Алгоритм Дейкстры 35](#_Toc106204016)

[3.1.1. Описание входных данных 35](#_Toc106204017)

[3.1.2. Описание алгоритма решения 36](#_Toc106204018)

[3.1.3. Описание выходных данных 40](#_Toc106204019)

[3.2. Метод ветвей и границ 41](#_Toc106204020)

[3.2.1. Описание входных данных 41](#_Toc106204021)

[3.2.2. Описание алгоритма решения 43](#_Toc106204022)

[3.2.3. Описание выходных данных 45](#_Toc106204023)

[3.3. Генетический алгоритм 46](#_Toc106204024)

[3.3.1. Описание входных данных 47](#_Toc106204025)

[3.3.2. Описание алгоритма решения 47](#_Toc106204026)

[3.3.3. Описание выходных данных 49](#_Toc106204027)

[4. Варианты использования системы 52](#_Toc106204028)

[4.1. ВИ\_1\_средствами Python. 52](#_Toc106204029)

[4.2. ВИ\_2\_средствами MS Excel. 54](#_Toc106204030)

[5. Архитектура решения 55](#_Toc106204031)

[6. Тестирование 68](#_Toc106204032)

[7. Заключение 70](#_Toc106204033)

[**ТЕОРИЯ ИГР 71**](#_Toc106204034)

[1. Физическая модель задачи 71](#_Toc106204035)

[1.1. Условие антагонистической задачи 71](#_Toc106204036)

[1.2. Биматричная игра 71](#_Toc106204037)

[1.3. Игра с природой в условиях риска 72](#_Toc106204038)

[1.4. Игра с природой в условиях неопределенности 72](#_Toc106204039)

[2. Математическая модель задачи 74](#_Toc106204040)

[2.1. Антагонистическая игра. 74](#_Toc106204041)

[2.2. Биматричная игра 78](#_Toc106204042)

[2.2. Задача о принятии оптимального решения в условиях риска 80](#_Toc106204043)

[2.1.1.Критерий Байеса. 81](#_Toc106204044)

[2.1.1.Критерий Лапласа. 82](#_Toc106204045)

[2.1.2.Критерий Гермейра. 82](#_Toc106204046)

[2.4 Задача о принятии оптимального решения в условиях неопределенности. 83](#_Toc106204047)

[2.1.3.Критерий Вальда. 83](#_Toc106204048)

[2.1.4.Критерий Сэвиджа. 84](#_Toc106204049)

[2.1.5.Критерий Гурвица. 85](#_Toc106204050)

[3. Алгоритмы решения задачи 86](#_Toc106204051)

[3.1. Антагонистическая игра (игры с нулевой суммой) 87](#_Toc106204052)

[3.1.1. Описание входных данных 87](#_Toc106204053)

[3.1.2. Описание алгоритма решения 88](#_Toc106204054)

[3.1.3. Описание выходных данных 91](#_Toc106204055)

[3.2. Биматричные игры 93](#_Toc106204056)

[3.2.1. Описание входных данных 93](#_Toc106204057)

[3.2.2. Описание алгоритма решения 94](#_Toc106204058)

[3.2.3. Описание выходных данных 98](#_Toc106204059)

[3.2.3. Описание выходных данных для чистых стратегий 98](#_Toc106204060)

[3.2.3. Описание выходных данных для смешанных стратегий 98](#_Toc106204061)

[3.3. Задача о принятии оптимального решения в условиях риска 100](#_Toc106204062)

[3.3.1. Описание входных данных 100](#_Toc106204063)

[3.3.2. Описание алгоритма решения 101](#_Toc106204064)

[3.3.2.1. Критерий Лапласа. 101](#_Toc106204065)

[3.3.2.2. Решение с помощью критерия Байеса. 102](#_Toc106204066)

[3.3.2.3. Критерий Гермейера 102](#_Toc106204067)

[3.3.3. Описание выходных данных 106](#_Toc106204068)

[3.3.3.1. Критерий Лапласа 106](#_Toc106204069)

[3.3.3.2. Критерий Байеса 106](#_Toc106204070)

[3.3.3.3. Критерий Гермейра 106](#_Toc106204071)

[3.4. Задача о принятии решений в условиях неопределенности. 108](#_Toc106204072)

[3.4.1. Описание входных данных 108](#_Toc106204073)

[3.4.2. Описание алгоритма решения 108](#_Toc106204074)

[3.4.2.1. Критерий оптимизма 108](#_Toc106204075)

[3.4.2.2. Критерий пессимизма 108](#_Toc106204076)

[3.4.2.3. Критерий Вальда 109](#_Toc106204077)

[3.4.2.4. Критерий Сэвиджа 109](#_Toc106204078)

[3.4.2.5. Критерий Гурвица 110](#_Toc106204079)

[3.4.3. Описание выходных данных 111](#_Toc106204080)

[3.4.3.1. Критерий оптимизма 111](#_Toc106204081)

[3.4.3.2. Критерий Вальда 111](#_Toc106204082)

[3.4.3.3. Критерий Сэвиджа 112](#_Toc106204083)

[3.4.3.4. Критерий Гурвица 112](#_Toc106204084)

[4. Варианты использования системы 113](#_Toc106204085)

[4.1. ВИ\_1\_средствами Python. 113](#_Toc106204086)

[4.2. ВИ\_2\_средствами MS Excel. 115](#_Toc106204087)

[5. Архитектура решения 117](#_Toc106204088)

[5.1. Функции считывания информации 117](#_Toc106204089)

[5.2. Функции обработки информации 119](#_Toc106204090)

[5.3. Функции вывода информации 122](#_Toc106204091)

[6. Тестирование 125](#_Toc106204092)

[7. Заключение 129](#_Toc106204093)

[**ПРОГНОЗИРОВАНИЕ 130**](#_Toc106204094)

[1. Физическая модель задачи 130](#_Toc106204095)

[2. Математическая модель задачи 131](#_Toc106204096)

[3.1.1. Описание входных данных 134](#_Toc106204097)

[3.1.2. Описание алгоритма решения 134](#_Toc106204098)

[3.1.3. Описание выходных данных 135](#_Toc106204099)

[3.2. Приемо-сдаточное тестирование. 136](#_Toc106204100)

[3.2.1. Описание входных данных 136](#_Toc106204101)

[3.2.2. Описание алгоритма решения 136](#_Toc106204102)

[3.2.3. Описание выходных данных 136](#_Toc106204103)

[4. Варианты использования системы 138](#_Toc106204104)

[5. Архитектура решения 140](#_Toc106204105)

[5.1. Функции считывания информации 140](#_Toc106204106)

[5.2. Функции обработки информации 140](#_Toc106204107)

[5.3. Функции вывода информации 141](#_Toc106204108)

[6. Тестирование 142](#_Toc106204109)

[6.1. Тестирование линейной и квадратичной аппроксимаций. 142](#_Toc106204110)

[6.2. Тестирование задач по удалению выбросов. 147](#_Toc106204111)

[6.2.1. Задача 6. 147](#_Toc106204112)

[6.2.2. Задача 7. 147](#_Toc106204113)

[6.2.3. Задача 8. 148](#_Toc106204114)

[6.2.4. Задача 9. 150](#_Toc106204115)

[6.2.5. Задача 10. 150](#_Toc106204116)

[7. Заключение 152](#_Toc106204117)

[**МНОГОКРИТЕРИАЛЬНАЯ ОПТИМИЗАЦИЯ 153**](#_Toc106204118)

[1. Физическая модель задачи 153](#_Toc106204119)

[2. Математическая модель задачи 154](#_Toc106204120)

[2.1. Поиск оптимального решения по Паретто 154](#_Toc106204121)

[2.2. Линейная свертка критериев 155](#_Toc106204122)

[2.3. Метод идеальной точки 155](#_Toc106204123)

[2.4. Метод контрольных показателей 156](#_Toc106204124)

[3. Алгоритмы решения задачи 157](#_Toc106204125)

[3.1. Поиск Паретооптимального решения; 157](#_Toc106204126)

[3.1.1. Описание входных данных 157](#_Toc106204127)

[3.1.2. Описание алгоритма решения 157](#_Toc106204128)

[3.1.3. Описание выходных данных 157](#_Toc106204129)

[3.2. Линейная свертка критериев 158](#_Toc106204130)

[3.2.1. Описание входных данных 158](#_Toc106204131)

[3.2.2. Описание алгоритма решения 158](#_Toc106204132)

[3.2.3. Описание выходных данных 159](#_Toc106204133)

[3.3. Метод идеальной точки 160](#_Toc106204134)

[3.3.1. Описание входных данных 160](#_Toc106204135)

[3.3.2. Описание алгоритма решения 160](#_Toc106204136)

[3.3.3. Описание выходных данных 160](#_Toc106204137)

[3.4. Метод контрольных показателей 161](#_Toc106204138)

[3.4.1. Описание входных данных 161](#_Toc106204139)

[3.4.2. Описание алгоритма решения 161](#_Toc106204140)

[3.4.3. Описание выходных данных 161](#_Toc106204141)

[4. Варианты использования системы 162](#_Toc106204142)

[5. Архитектура решения 164](#_Toc106204143)

[5.2. Функции обработки информации 164](#_Toc106204144)

[5.3. Функции вывода информации 165](#_Toc106204145)

[6. Тестирование 166](#_Toc106204146)

[7. Заключение 171](#_Toc106204147)

[**ЭКСПЕРТНЫЕ ОЦЕНКИ 172**](#_Toc106204148)

[1. Физическая модель задачи 172](#_Toc106204149)

[1.1. Обработка экспертных оценок. 172](#_Toc106204150)

[1.2. Обработка ранговых оценок. 173](#_Toc106204151)

[1.3. Обработка бинарных отношений. 174](#_Toc106204152)

[2. Математическая модель задачи 175](#_Toc106204153)

[2.1. Метод средних арифметических рангов 175](#_Toc106204154)

[2.2. Метод медианы рангов 176](#_Toc106204155)

[2.3. Метод бинарных отношений 177](#_Toc106204156)

[3. Алгоритмы решения задачи 178](#_Toc106204157)

[3.1. Математическая обработка экспертных оценок; 178](#_Toc106204158)

[3.1.1. Описание входных данных 178](#_Toc106204159)

[3.1.3. Описание выходных данных 178](#_Toc106204160)

[3.2. Математическая обработка ранговых оценок 179](#_Toc106204161)

[3.2.1. Описание входных данных 179](#_Toc106204162)

[3.2.3. Описание выходных данных 180](#_Toc106204163)

[3.3. Математическая обработка бинарных отношений 181](#_Toc106204164)

[3.3.1. Описание входных данных 181](#_Toc106204165)

[3.3.2. Описание алгоритма решения 181](#_Toc106204166)

[3.3.3. Описание выходных данных 181](#_Toc106204167)

[4. Варианты использования системы 182](#_Toc106204168)

[4.1. ВИ 1\_Решение задач методом средних баллов и методом медианы рангов. 182](#_Toc106204169)

[4.2. ВИ 2\_Решение методом бинарных отношений 183](#_Toc106204170)

[5. Архитектура решения 184](#_Toc106204171)

[5.1. Функции считывания информации 184](#_Toc106204172)

[5.2. Функции обработки информации 184](#_Toc106204173)

[5.3. Функции вывода информации 184](#_Toc106204174)

[6. Тестирование 186](#_Toc106204175)

[7. Заключение 197](#_Toc106204176)

[**СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ 198**](#_Toc106204177)

[1. Физическая модель задачи 198](#_Toc106204178)

[2. Математическая модель задачи 199](#_Toc106204179)

[3. Алгоритмы решения задачи 202](#_Toc106204180)

[3.1. Описание входных данных 202](#_Toc106204181)

[3.3. Описание выходных данных 203](#_Toc106204182)

[4. Варианты использования системы 204](#_Toc106204183)

[4.1. ВИ 1 204](#_Toc106204184)

[4.2. ВИ 2\_Визуализация\_решения 205](#_Toc106204185)

[5. Архитектура решения 207](#_Toc106204186)

[5.1. Функции считывания информации 207](#_Toc106204187)

[5.2. Функции обработки информации 207](#_Toc106204188)

[5.3. Функции вывода информации 207](#_Toc106204189)

[6. Тестирование 209](#_Toc106204190)

[7. Заключение 212](#_Toc106204191)

# **ТРАНСПОРТНАЯ ЗАДАЧА**

# **1. Физическая модель транспортной задачи**

Компания имеет 3 мукомольных предприятия и 3 центра их распределения. Фабрики располагаются в Твери, Екатеринбурге и Москве с производственными возможностями 100, 40 и 36 единиц продукции ежедневно, соответственно. Центры распределения располагаются в Омске, Туле, Владимире с потребностями в 20, 45 и 150 единиц продукции ежедневно, соответственно. Стоимость перевозки единицы продукции с предприятий в пункты распределения приведены в Таблице 1.

Таблица 1. Исходные данные транспортной задачи

|  |  |  |  |
| --- | --- | --- | --- |
| **Потребление**  **Запасы** | 20 | 45 | 150 |
| 100 | 7 | 3 | 6 |
| 60 | 4 | 8 | 2 |
| 55 | 1 | 5 | 9 |

# **2. Математическая модель транспортной задачи**

Определение плана перевозок некоторого однородного товара (груза).

Груз находится у компаний-поставщиков A1, A2, Ai в объемах a1, a2, ai, его необходимо доставить компаниям-потребителям B1, B2, Bi в объемах b1, b2, bj. Известны Cij – стоимость перевозки товара от каждого i-го поставщика для каждого j-го потребителя.

Необходимо найти оптимальное решение: составить план перевозок, где суммарные затраты на перевозку всех грузов минимальны.

Данные, которые даны на входе: вектор запасов поставщиков, вектор потребления, матрица весов – цены перевозок.

Таблица 2. Входные данные транспортной задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Потребление**  **Запасы**  **поставщиков** | b1 | b2 | b3 | … |
| a1 | с11 | с12 | с13 |  |
| a2 | с21 | с22 | с23 |  |
| a3 | с31 | с32 | с33 |  |
| … |  |  |  |  |

Кроме того, необходимо составить необходимые ограничения:

Первое, переменные Xij – целые положительные числа.

В случае закрытой задачи, где сумма запасов поставщиков и сумма потребностей потребителей равны, балансовые уравнения будут иметь вид:

Для открытой задачи с разными значениями сумм спроса и потребления существует два условия.

При условии, что :

Поэтому мы будем вводить фиктивного (n+1)-го потребителя с запросами

равными разности суммарных запасов поставщиков и запросов потребителей, и нулевыми стоимостями перевозок единиц груза.

И при условии :

При таком случае будет вводиться фиктивного (m+1)-го поставщика с запасами равные разности суммарных запросов потребителей и запасов поставщиков, и нулевыми стоимостями перевозок единиц груза.

Переменные предоставлены в виде матрицы перевозок:

Целевая функция в формульном виде выражает требования обеспечить минимум суммарных затрат на перевозку всех грузов. Математическая модель в общем случае имеет вид:

Оптимальное значение целевой функции задачи равно сумме произведений всех соответствующих элементов матриц С и Х. Данная функция, определяющая суммарные затраты на все перевозки, должна достигать минимального значения.

# **3. Алгоритмы решения транспортной задачи**

Для решения транспортной задачи рассмотрим три способа:

1. метод линейного программирования;
2. симплекс-метод;
3. графический метод.

Первые два метода позволять решить транспортную задачу закрытого типа для матрицы любой размерности. Для графического метода рассмотрим теоретические основы и разберем решение подобной задачи данным методом.

## **3.1. Метод линейного программирования**

Метод линейного программирования — метод решения экстремальных задач на множествах n-мерного векторного пространства, задаваемых системами линейных уравнений и неравенств. Линейное программирование (ЛП) является частным случаем выпуклого программирования, которое в свою очередь является частным случаем математического программирования.

### **3.1.1. Описание входных данных**

Входными данными для метода линейного программирования являются:

* количество строк;
* количество столбцов;
* элементы вектора запасов (производства);
* элементы вектора потребностей;
* элементы матрицы весов.

Все данные необходимо вводить через Enter.

Введите количество строк: **2**

Введите количество столбцов: **3**

Введите элементы вектора запасов через Enter

Element: **100**

Element: **40**

[**'100' '40'**]

Введите элементы вектора потребности через Enter

Element: **60**

Element: **30**

Element: **50**

[**'60' '30' '50'**]

Введите элементы матрицы весов через Enter

Element: **7**

Element: **3**

Element: **6**

Element: **4**

Element: **8**

Element: **2**

[[**7**, **3**, **6**], [**4**, **8**, **2**]]

Кроме того, в коде предусмотрен ряд параметров: для выбора метода решения необходимо ввести цифру «1» или «2», так же у пользователя есть возможность посмотреть пример графика для решения задачи, введя необходимый параметр. Ниже приведен пользовательский интерфейс для выбора метода решения.

Соответственно, в данном разделе нужно ввести «2» при выборе метода решения.

Выберите метод решения

Введите '1' для решения симплекс-методом, '2' для решения методом линейного программирования:**2**

Необходимость графика

Введите '1' для просмотра примера графика, '0' - если график не нужен:**0**

### **3.1.2. Описание алгоритма и архитектуры решения**

Подключаем необходимые библиотеки:

**import** **numpy** **as** **np**

**from** **scipy.optimize** **import** linprog

**from** **quantecon.optimize** **import** linprog\_simplex

**from** **scipy.stats** **import** binom, betabinom

**from** **numpy** **import** \*

**from** **datetime** **import** datetime

Далее осуществляем ввод размерности матрицы (количество строк и столбцов), значения вектора запасов и потребления, а также элементы матрицы:

# Ввод размерности матрицы

# Кол-во строк

m = int(input("Введите количество строк: "))

# Кол-во столбцов

n = int(input("Введите количество столбцов: "))

# Ввод емкости (m-мерный вектор)

**print**('Введите элементы вектора запасов через Enter')

p = array([])

**for** i **in** range(m):

v = input("Element: ")

p = append(p, v)

**print**(p)

# Ввод требования (n-мерный вектор)

**print**('Введите элементы вектора потребности через Enter')

q = array([])

**for** i **in** range(n):

v = input("Element: ")

q = append(q, v)

**print**(q)

# Ввод элементов матрицы (m x n)

**print**('Введите элементы матрицы весов через Enter')

C = []

**for** i **in** range(m):

C.append([])

**for** j **in** range(n):

C[i] += [int(input("Element: "))]

**print**(C)

C = np.array(C)

Далее необходимо задать метод решения и указать необходимость графика.

#Ввод параметров для выбора решения

**print**('Выберите метод решения')

method = int(input("Введите '1' для решения симплекс-методом, '2' для решения методом линейного программирования:"))

**print**('Необходимость графика')

graphic = int(input("Введите '1' для просмотра примера графика, '0' - если график не нужен:"))

Создаем функцию, которая принимает входные данные и решает задачу методом линейного программирования:

#Функция решения лп методом

**def** **interior\_point**(C, m,n):

start\_time = datetime.now()

Производим векторизацию с помощью функции reshape():

# Векторизация матрицы C

C\_vec = C.reshape((m\*n, **1**), order='F')

Строим матрицу А по произведению Кронекера и образуем вектор целевой функции:

# Построение матрицы A по произведению Кронкера

A1 = np.kron(np.ones((**1**, n)), np.identity(m))

A2 = np.kron(np.identity(n), np.ones((**1**, m)))

A = np.vstack([A1, A2])

# Построение вектора b

b = np.hstack([p, q])

С помощью функции из пакета SciPy linprog() решаем задачу минимизацией и не допускаем ограничений-неравенств со знаком больше или равно.

# Решение основной проблемы

res = linprog(C\_vec, A\_eq=A, b\_eq=b, method='interior-point')

# Вывод результатов

**print**("message:", res.message)

**print**("nit:", res.nit)

**print**("fun:", res.fun)

**print**("z:", res.x)

**print**("X:", res.x.reshape((m, n), order='F'))

**print**("Время решения:", datetime.now() - start\_time)

Далее в коде предусмотрены разветвления в зависимости от выбранного метода:

#Основной процесс

**if** method == **1**:

simplex(C, m,n)

**if** graphic == **1**:

**print**("graphic")

**elif** graphic == **0**:

**print**("no graphic")

**else**:

**print**("Данные введены некорректно, введите '1' или '0'")

elif method == **2**:

interior\_point(C, m,n)

**if** graphic == **1**:

**print**("graphic")

**elif** graphic == **0**:

**print**("no graphic")

**else**:

**print**("Данные введены некорректно, введите '1' или '0'")

**else**:

**print**("Данные введены некорректно, введите '1' или '2'")

### **3.1.3. Описание выходных данных**

Выходными данными является минимальное значение целевой функции, а также матрица закупок X:

nit: **5**

fun: **649.9999991426155**

z: [**5.99999999e+01 3.31077017e-08 3.00000000e+01 3.57997345e-09**

**1.00000000e+01 3.99999999e+01**]

X: [[**5.99999999e+01 3.00000000e+01 1.00000000e+01**]

[**3.31077017e-08 3.57997345e-09 3.99999999e+01**]]

Время решения: **0:00:00.017397**

## **3.2. Симплекс-метод**

Математическая модель в общем случае имеет тот же вид, что было рассмотрено выше (см. формула 7), однако необходимо обозначить некоторые изменения. При нахождении целевой функции мы преобразуем матрицу X в вектор путем укладки всех его столбцов в вектор столбцов с помощьювекторизации vec(X). Аналогично преобразуем матрицу C в mn-мерный вектор vec(C). Поэтому целевая функция может быть выражена как произведение между vec(C) и vec(X):

min vec(C)′ ⋅ vec(X) (8)

При этом мы используем произведение Кронекера.

Оптимальное значение целевой функции задачи равно сумме произведений всех соответствующих элементов матриц С и Х. Данная функция, определяющая суммарные затраты на все перевозки, должна достигать минимального значения.

### **3.2.1. Описание входных данных**

Входные данные для симплекс-метода те же, что и в методе линейного программирования (см. пункт 3.1.1.), изменению подлежит лишь параметр выбора метода решения: при запросе метода решения необходимо ввести «1».

### **3.2.2. Описание алгоритма и архитектуры решения**

Подробный алгоритм решения можно увидеть в пункте 3.1.2. отличие заключается в том, что теперь необходимо создать другую функцию для решения задачи симплекс-методом. Таким образом, создается следующая функция:

#Функция решения симплекс методом

**def** **simplex**(C, m,n):

start\_time = datetime.now()

# Векторизация матрицы C

C\_vec = C.reshape((m\*n, **1**), order='F')

# Построение матрицы A по произведению Кронкера

A1 = np.kron(np.ones((**1**, n)), np.identity(m))

A2 = np.kron(np.identity(n), np.ones((**1**, m)))

A = np.vstack([A1, A2])

# Построение вектора b

b = np.hstack([p, q])

# Решение основной проблемы

res = linprog(C\_vec, A\_eq=A, b\_eq=b, method='Revised simplex')

# Вывод результатов

**print**("message:", res.message)

**print**("nit:", res.nit)

**print**("fun:", res.fun)

**print**("z:", res.x)

**print**("X:", res.x.reshape((m, n), order='F'))

**print**("Время решения:", datetime.now() - start\_time)

### **3.2.3. Описание выходных данных**

Выходными данными является минимальное значение целевой функции, а также матрица закупок X. Форматы вывода данных см. пункт 3.1.3.

## **3.3. Графический метод**

Графический метод основан на геометрической интерпретации задачи линейного программирования (ЗЛП) и применяется в основном при решении задач двумерного пространства и только некоторых задач трехмерного пространства. Задачу пространства размерности больше трех изобразить графически вообще невозможно.

### **3.3.1. Описание входных данных**

Для того, чтобы просмотреть данный график как пример графического метода решения, необходимо ввести «1» в поле ввода входных данных (см. пункт 3.1.1.).

Для описания решения графического метода возьмем следующую систему уравнений с поставленной задачей для ЗЛП:

Цель задачи: максимизация следующей целевой функции:

### **3.3.2. Описание алгоритма и архитектуры решения**

Для удобства решения и нанесения на координатную плоскость графиков функции сделаем замену: ; Далее составим уравнения графиков функций, которые нам нужно будет нанести на координатную плоскость:

Далее, изобразив графики, мы снова обращаемся к системе уравнений и приходим к выводу, что нам удовлетворяют решения, находящиеся под графиком функции. Общей областью решений станет то, где все три области пересекаются, ограничим ее красной линией на графике, для наглядности продемонстрируем это на рисунке (1):

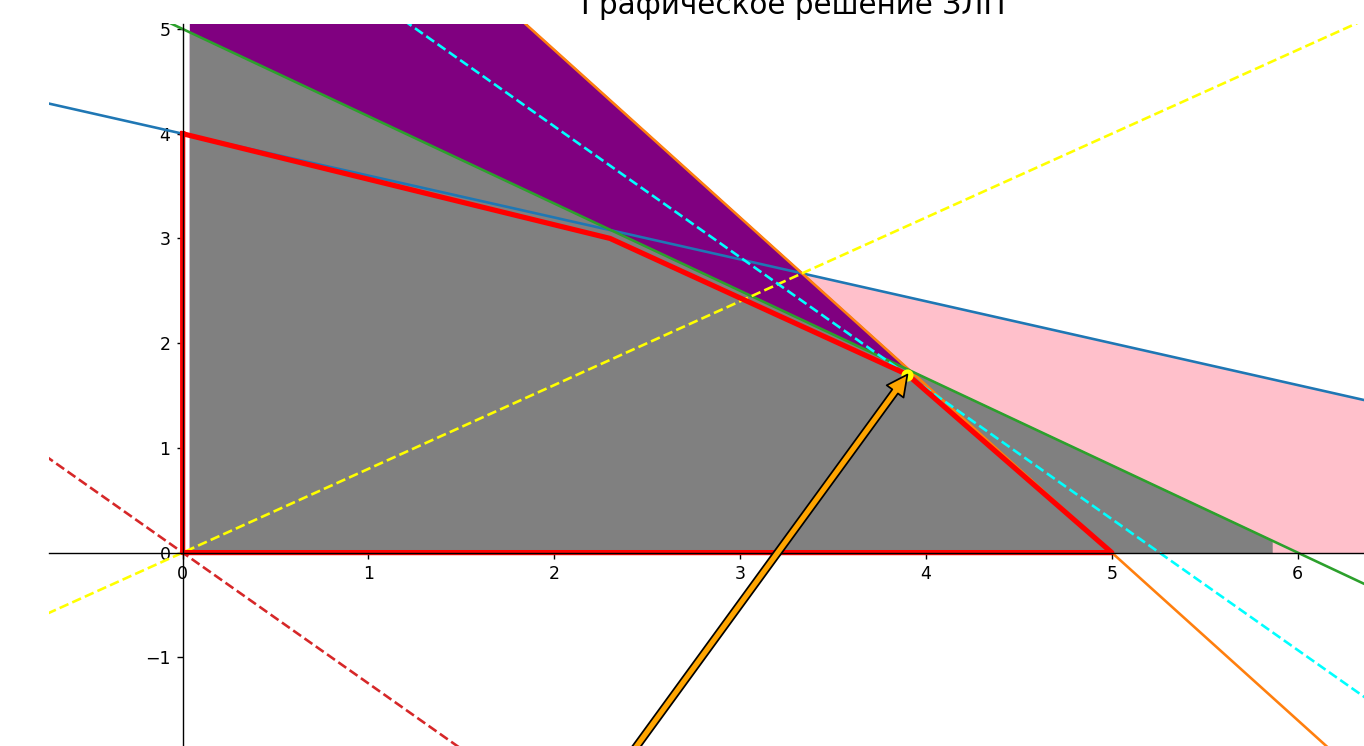


Рисунок 1. Область решений для исходной системы уравнений

Далее нам необходимо построить вектор градиент от целевой функции, на рисунке 1. он изображен пунктирной желтой линией:

Для решения данной задачи графическим методом нам необходимо построить перпендикуляр к вектору градиенту, для этого нам нужно найти уравнение графика функции этого перпендикуляра. По свойству перпендикулярности прямых:

Следовательно, , поэтому график функции перпендикулярной прямой к вектору градиенту будет равен (он изображен на рисунке 1. красной пунктирной линией):

Далее мы двигаем этот перпендикуляр до того момента пока он не станет опорным для области решений, это достигается в точке, изображенной на рисунке 1. желтой. Проводим через эту точку перпендикулярную прямую к вектору градиенту. В данной точке пересекаются два графика функции: и . Для того, чтобы найти точку пересечения нужно решить следующую систему уравнений:

Решение системы уравнений:

Следовательно, точка пересечения графиков (3.9; 1.7). Подставляем данное значение в целевую функцию:

Для визуализации используются библиотеки numpy и matplotlib.pyplot.

# импортируем модули

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

fig = plt.subplots()

# создаём область, в которой будет

# - отображаться график

x = np.linspace(-**4**, **12**, **100**)

# создаем оси координат

ax = plt.gca()

ax.spines['top'].set\_color('none')

ax.spines['bottom'].set\_position('ноль')

ax.spines['left'].set\_position('ноль')

ax.spines['right'].set\_color('ноль')

# задаем графики функций для исходной системы уравнений

l1 = **lambda** x: **4** - **2** / **5** \* x

l2 = **lambda** x: **8** - **8** / **5** \* x

l3 = **lambda** x: **5** - **5** / **6** \* x

# строим вектор градиент (значения взяты из целевой функции)

l4 = **lambda** x: **4/5** \* x

# строим перпендикуляр к вектору градиенту

l5 = **lambda** x: -5 / **4** \* x

l6 = **lambda** x: -**5** / **4** \* x + **6.57**

# данное значение (в l6) подобрано так, чтобы перпендикуляр

# к вектору градиенту проходил через искомую точку

# координата по оси х

s = **3.9**

# координата по оси у

w = **1.7**

# вспомогательные точки для выделение области

f = [**0**, **2.3**, **3.9**, **5**, **0**, **0**]

лк = [**4**, **3**, **1.7**, **0**, **0**, **4**]

# задаем название графика

plt.title('Графическое решение ЗЛП', fontsize=**17**)

# наносим функции на график

plt.plot(x, l1(x))

plt.plot(x, l2(x))

plt.plot(x, l3(x))

plt.plot(x, l4(x), '--', color='yellow')

plt.plot(x, l5(x), '--', )

плт.plot(x, l6(x), '--', color='cyan')

#выделяем область, удовлетворяющую условиям

plt.plot(f, lk, '-', color='red', linewidth=**3**)

# выделяем точку, которая является решением задачи

plt.plot(s, w, 'o', color='yellow')

# для графиков функции наносим на график области решения

ax.fill\_between(x, **4-2** / **5** \* x, где=(**10** >> x) & (x >=> **0**), color='pink')

ax.fill\_between(x, **8-8** / **5** \* x, где=(**5** >> x) & (x >=> **0**), color='purple')

ax.fill\_between(x, **5-5** / **6** \* x, где=(**6** >> x) & (x >=> **0**), color='0.5')

# подписываем графики функций

plt.text(**6.6**, **8.3**, '50x1+40x2=0(вектор градиент)', fontsize=**8**)

plt.текст(**10.6**, -3.2, '5x1+6x2 <= 30', fontsize=**8**)

plt.text(**7.6**, -**6.9**, '8x1+5x2 <= 40', fontsize=**8**)

plt.text(-**3.8**, **6.1**, '2х1+5х2 <= 20', fontsize=**8**)

# стрелкой показываем максимум функции

plt.annotate('max', xy=(**3.9**, **1.7**), xycoords='data',

xytext=(**1**, -5), textcoords='data',

arrowprops=dict(facecolor='orange'))

# показываем график

plt.show()

### **3.3.3. Описание выходных данных**

В результате получаем график, изображенный на рисунке 2:

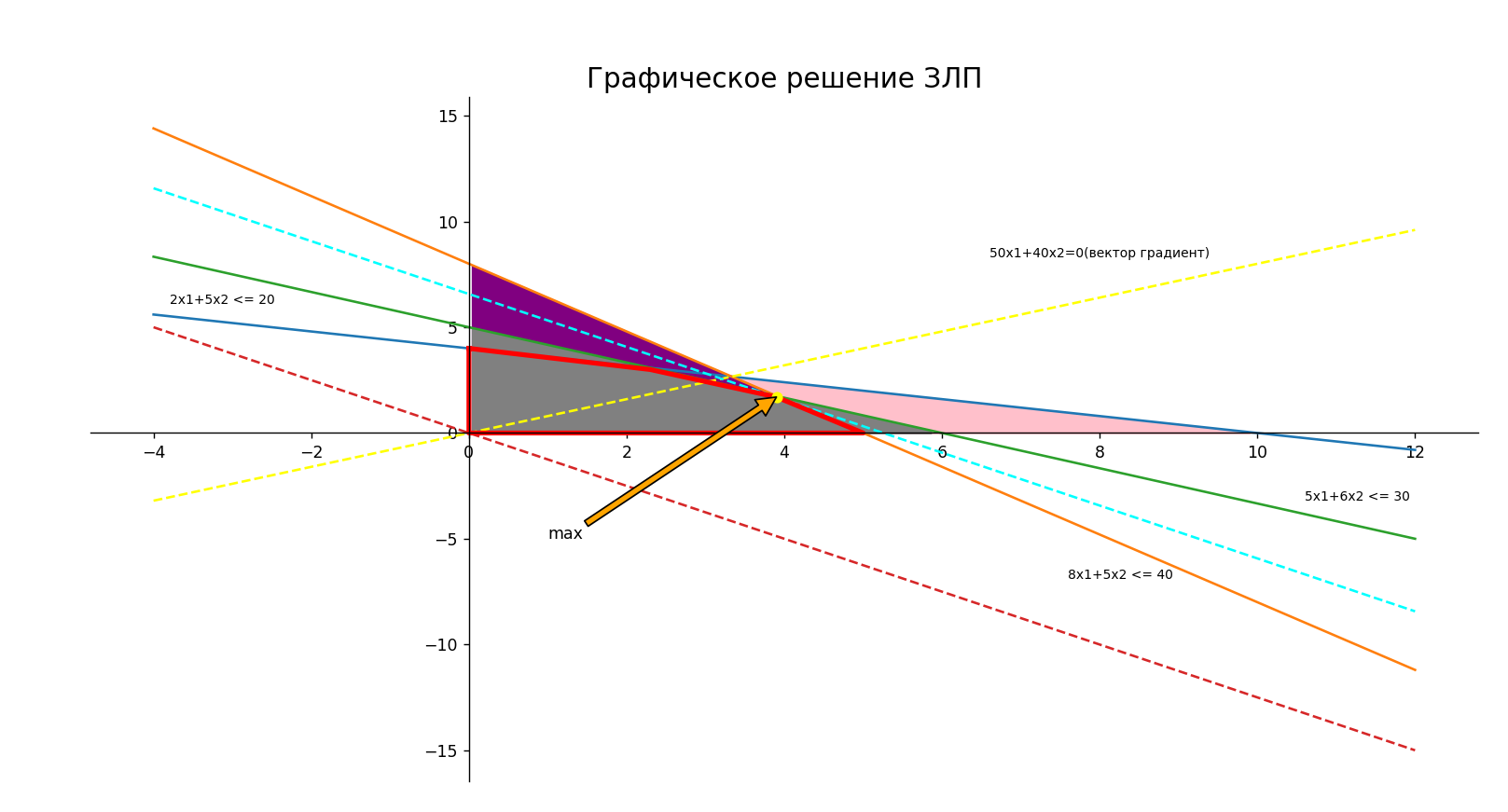


Рисунок 2. Визуализация графического решения ЗЛП относительно поставленной задачи

# **4. Варианты использования системы**

Для пользователя предусмотрены два варианта использования системы: средствами Python и средствами MS Excel. Пользователь может выбрать любой удобный для него вариант.

## **4.1. ВИ 1**

1. Пользователь должен запустить среду разработки для Python.
2. Пользователь должен запустить код для решения задачи.
   1. Пользователь запускает код для метода №1 (см. раздел 3 метод 1).
      1. Пользователю необходимо ввести следующие входные данные:
         1. Ввести количество строк: одно целое число.
         2. Ввести количество столбцов: одно целое число.
         3. Ввести элементы вектора запасов:

* через Enter
* каждый элемент в новой строке
  + - 1. Ввести элементы вектора потребности:
* через Enter
* каждый элемент в новой строке
  + - 1. Ввести элементы матрицы весов:
* через Enter
* каждый элемент в новой строке
  + - 1. Выбрать один из методов решения, нажав введя либо цифру 1, либо 2:
* '1' - для решения симплекс-методом,
* '2' - для решения методом линейного программирования
  + - 1. Выбрать вариант необходимости графика:
* '1' - для просмотра примера графика
* '0' - если график не нужен

1. Программа должна считать данные.
2. После обработки скрипта пользователю должен выводиться результат.

# **5. Тестирование**

Далее приведена таблица результатов решения задач разными методами.

*Таблица 3. Результаты тестирования программы*

|  |  |  |  |
| --- | --- | --- | --- |
| **Входные данные** | **Метод линейного программирования** | **Симплекс-метод** | **MS EXCEL** |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 2 | 7 | 8 | 3 | **120** | | 5 | 1 | 6 | 9 | 3 | **90** | | 7 | 6 | 2 | 4 | 5 | **80** | | **70** | **40** | **90** | **30** | **60** |  | | 959.999 | 960 | 960 |
| |  |  |  |  | | --- | --- | --- | --- | | 7 | 2 | 9 | **30** | | 8 | 4 | 1 | **25** | | 3 | 8 | 5 | **20** | | 6 | 1 | 2 | **35** | | **40** | **20** | **50** |  | | 304.999 | 305 | 305 |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 4 | 8 | 0 | 5 | 7 | 2 | **130** | | 1 | 6 | 8 | 2 | 9 | 6 | **150** | | **20** | **70** | **90** | **30** | **60** | **10** |  | | 999.999 | 1000 | 1000 |

*Таблица 4. Результаты замеров времени работы программы*

|  |  |  |
| --- | --- | --- |
| **Размер матрицы** | **Время решения методом линейного программирования** | **Время решения симплекс-методом** |
| 5x5 | 0:00:00.037284 | 0:00:00.011968 |
| 5x10 | 0:00:00.017619 | 0:00:00.013963 |
| 10x5 | 0:00:00.014305 | 0:00:00.014967 |
| 10x10 | 0:00:00.030586 | 0:00:00.019955 |

Из таблицы 4 мы видим, что решение задач подобного типа симплекс-методом является более быстрым и эффективным.

# **6. Заключение**

В данной проектной документации была построена математическая модель транспортной задачи методом линейного программирования и симплекс методом, были разработаны программы средствами Python для нахождения оптимального результата минимизации целевой функции, были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат, также была представлена визуализация графика для разработанной математической модели методом линейного программирования.

С помощью представленных в этой документации программ можно получить ответ на классические транспортные задачи открытого (методом линейного программирования и симплекс методом) и закрытого типа (методом линейного программирования), а также для другого ряда задач по оптимизации поиска минимального показателя – найти минимум издержек, себестоимости, трудоемкости, капиталовложений и т.п. Например, задача о смесях (диетах, питании, рационе), где необходимо отыскать наиболее дешевый набор из определенного ряда исходных материалов, которые бы обеспечивали получение смеси с заданными свойствами.

Таким образом, транспортная задача была решена простым методом решения задач линейного программирования и симплекс-методом. Последний является более эффективным, так как позволяет избежать простого перебора всех возможных угловых точек пространства. В практике решения задач линейного программирования практически всегда применяется именно он ввиду своей оптимальности. На его принципе базируется ряд иных методов численной оптимизации (метод Гомори, М-Метод).

# **ОПТИМИЗАЦИЯ НА ГРАФАХ: ЗАДАЧА КОММИВОЯЖЕРА**

# **1. Физическая модель задачи**

Имеется 5 городов, между которыми известно время переездов. Путешественнику необходимо пройти все 5 городов по одному разу, не возвращаясь в начальный. Требуется найти такой маршрут путешествия, на который будет затрачено минимальное количество времени.

Время переездов между городами представлено в следующем графе:

**2**

**7**

**9**

**10**

**8**

**4**

**3**

**5**

**11**

Рисунок 1. Неполный неориентированный граф

Приведем матрицу смежности (таблица 1), описывающую граф, изображенный на рисунке 1

Таблица 1 - Матрица смежности

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 0 | 9 | 7 | 2 |
| 2 | 0 | 0 | 4 | 8 | 10 |
| 3 | 9 | 4 | 0 | 3 | 11 |
| 4 | 7 | 8 | 3 | 0 | 5 |
| 5 | 2 | 10 | 11 | 5 | 0 |

# **2. Математическая модель задачи**

Коммивояжер должен объехать N городов. Известны затраты (временные, стоимостные, расстояния) на переезд между i – м и j – м городом, которые заданы в виде матрицы:

Коммивояжер, выехав из исходного города, должен объехать все города, посетив каждый один раз. Требуется определить в каком порядке следует объезжать города, чтобы суммарные затраты были минимальными.

В качестве переменных выбираются элементы матрицы переездов:

Задача состоит из целых чисел. Пусть , когда торговец переходит из города в город. И если , следовательно перехода между городами нет. Введем город , расположенный в том же городе, где начинает свое путешествие торговец. Теперь из первого города можно только выйти, а в город можно только зайти.

Дополнительное целочисленное значение равно количеству способов доступа к городу , . Поэтому, определяя дополнительные ограничения, связывающие переменные и количество всех городов на пути коммивояжера, который имеет вид массива: . Матрица состоит из затраченного времени между городами, где . Когда стоимость проезда или вес ребра на графе между двумя городами не зависит от направления движения: .

Целевая функция задачи имеет вид:

Где множество допустимых альтернатив – – представляется такими ограничениями:

* Искомый путь проходит через каждую вершину графа ровно по одному разу:
* принимает только значения 0 или 1:
* принимает только значения 0 или 1:
* должны принимать вещественные значения:
* Искомый путь должен представлять из себя единый цикл, то есть не должен распадаться на отдельные циклы:

Существует довольно много разновидностей и обобщений задачи коммивояжера. В классической закрытой задаче коммивояжеру необходимо вернуться в начальный пункт отправления, в открытой задаче коммивояжера возврат в начальный пункт не предполагается: коммивояжер начинает свой путь в некотором городе и посещает каждый оставшийся город в точности один раз. При этом конечные пункты маршрута могут быть произвольными либо фиксированными. Путь в графе, проходящий через каждую вершину ровно один раз, называется гамильтоновым путем. Таким образом, открытая задача коммивояжера – это задача поиска гамильтонова пути, имеющего минимальную длину. Любую открытую задачу коммивояжера можно свести к обычной (закрытой) или наоборот.

Любое допустимое решение открытой задачи коммивояжера можно отождествить с перестановкой чисел 1,…,n. Длина маршрута z равна . Количество открытых маршрутов коммивояжера составляет, очевидно, n!, если начальный и конечный пункты произвольны, (n −1)! или (n − 2)!, если фиксированы один или оба концевых пункта соответственно.

# **3. Алгоритмы решения задачи**

Для решения задачи коммивояжера рассмотрим три способа:

1. Метод Дейкстры;
2. Метод ветвей и границ;
3. Генетический метод.

## **3.1. Алгоритм Дейкстры**

Алгоритм Дейкстры для задачи коммивояжера — это алгоритм поиска кратчайших путей между узлами графа, которые могут представлять, например, дорожные сети, при этом прохождение одной вершины графа может быть только один раз.

### **3.1.1. Описание входных данных**

Входными данными является весовая матрица, основанная на матрице смежности, в которой отражено количество вершин и расстояние между вершинами графа (обратим внимание на то, что задача является открытой, т.е нам не нужно возвращаться в исходную точку):

Изображение выглядит как текст, кроссворд, легкий

Автоматически созданное описание

Рисунок 1, Матрица смежности в качестве входных данных для решения задачи алгоритмом Дейкстры

### **3.1.2. Описание алгоритма решения**

Для решения мы используем визуализацию, т.е рисуем данный граф по матрице смежности:

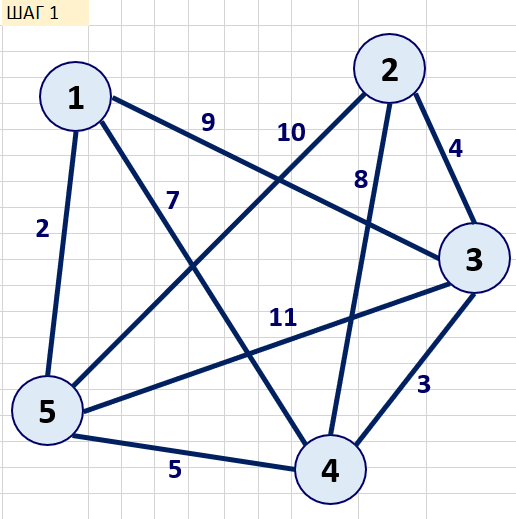


Рисунок 2, построенный граф по матрице смежности

Далее нам необходимо выбрать вершину, от которой мы будем двигаться и искать кратчайший путь, в данном случае выберем вершину 1:

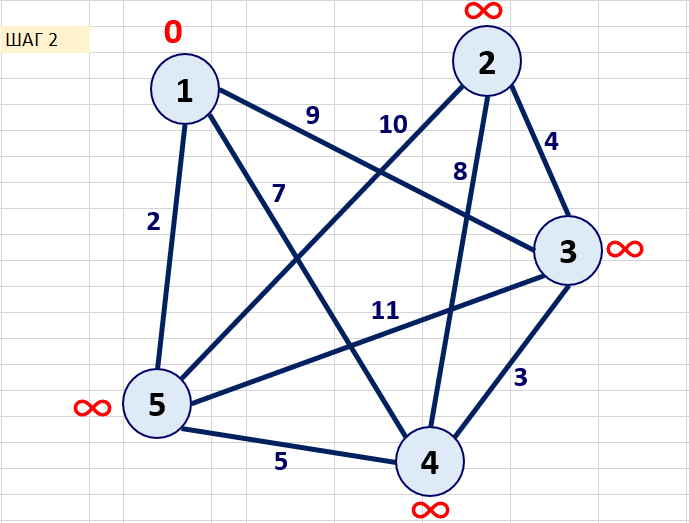


Рисунок 3, выбор вершины для начала перебора

После этого мы начинаем перебирать пути от вершины 1, при этом данный путь должен быть самым коротким:

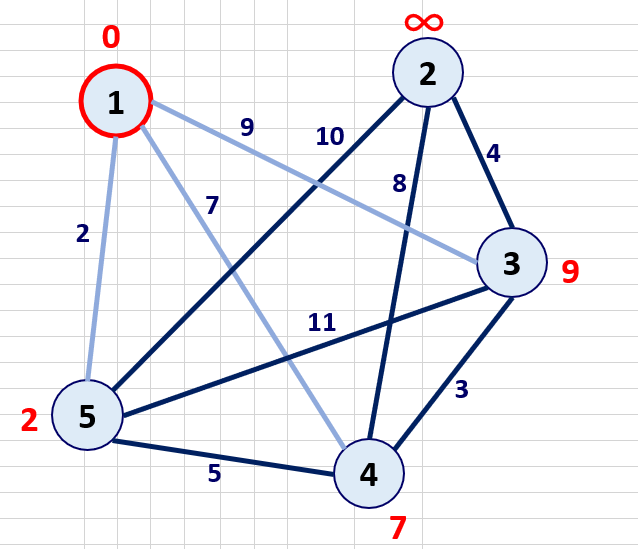


Рисунок 4, перебор кратчайшего расстояния от вершины 1 до другой вершины

На рисунке 4. мы видим, что кратчайший путь у нас будет из вершины 1 в вершину 2 и этот путь будет занимать 2 часа, далее по аналогии мы осуществляем данные действия для других вершин графа, пока мы не пройдем через каждую вершину:

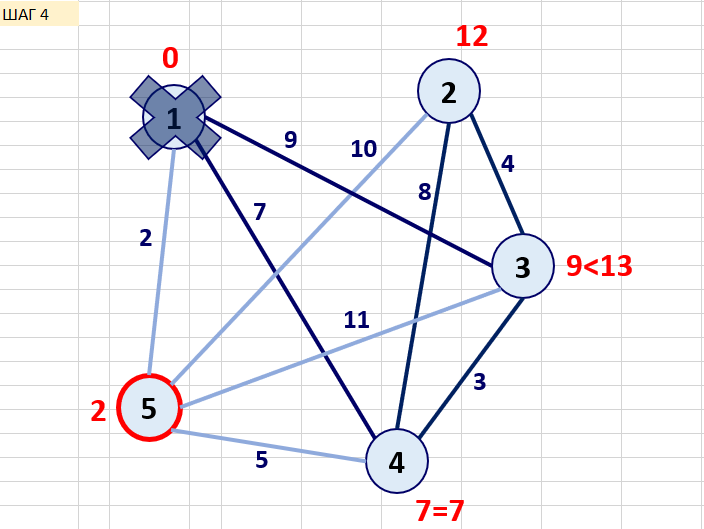


Рисунок 5, процесс нахождения кратчайшего пути для следующей вершины графа, удовлетворяющей условию

Как уже было сказано выше, то перебор осуществляется до того момента, пока мы не найдем путь, при котором он будет самым коротким, при этом обходя все вершины графа.

Задачу также можно решить с помощью поиска решений. Для этого мы создаем следующую таблицу:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 6, решение задачи через функцию "Поиск решений"

Искомыми переменными в данном случае являются вершины, т.е тот порядок прохождения вершин, при котором мы сможем получить минимальное значение. В столбце «Время переезда (часы)» будет использоваться следующая функция Excel:

Целевая функция будет являться суммой ячеек из столбца «Время переезда (часы)». Запускаем функцию «Поиск решений»:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7, параметры функции "Поиск решений" для решения исходной задачи

Ячейка целевой функции уже объявлена в данной таблице (сумма времени переезда). Мы ищем минимум целевой функции, так как нам необходимо затратить как можно меньше времени. Ячейки переменных представлены на рисунке 6 и выделены голубым цветом (вершины графа). Далее разберем ограничения, которые мы вводим для «Поиска решений»:

1. В столбце параметров все значения должны быть разными, т.к вершины графа не должны повторяться между собой
2. Все числа в столбце параметров должны быть целыми
3. Время должно быть не отрицательным, поэтому выставляем больше или равно единице

Метод решения выбираем: «Эволюционный поиск решения», т.к переменные у нас находятся в индексах матрицы.

### **3.1.3. Описание выходных данных**

Итог: 14 часов мы затратим на то, чтобы объехать все города, при этом данное количество заданного времени является минимальным, это можно увидеть на визуализированном графе, а также с помощью поиска решений:

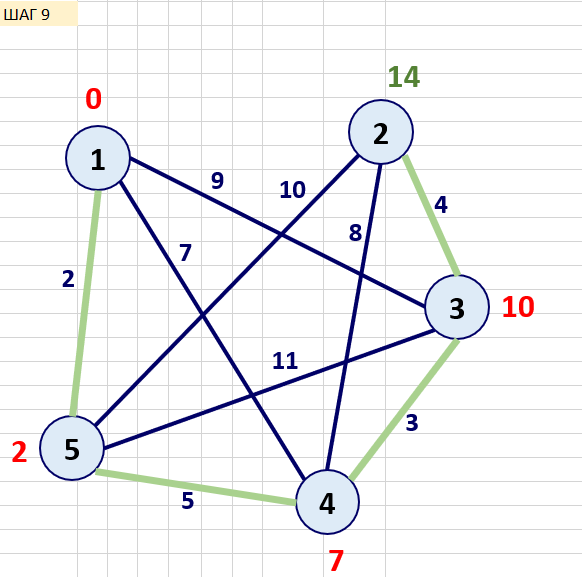


Рисунок 8, готовое решение задачи алгоритмом Дейкстры с использованием визуализации исходного графа

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 9, готовое решение задачи через «Поиск решений»

## **3.2. Метод ветвей и границ**

Подход получил название метод ветвей и границ (англ. branch and bound) — как общий алгоритмический метод оптимизации. Метод ветвей и границ связывают с деревом поиска оптимального (Торt) решения, которое строится в процессе обработки исходных данных задачи. Отсюда названия корень, которому в дереве приписывают все возможные в задаче, решения-ветви, соединяющие узлы дерева, Использование понятия границ и их расчет стимулирует или тормозит рост ветвей в таком дереве.

Важную роль играет процедура разбиения на узлы области допустимых решений (ОДР) исходной задачи, т.е. на меньшие непересекающиеся подмножества и их оценивание.

Другая процедура, названная процедурой ветвления, реализует разбиение на множества допустимых значений переменной х на подобласти меньших размеров.

Еще один важный элемент МВГ - процедура вычисления оценок, которая состоит в поиске значений границ ЦФ для решения задачи. Вычисление нижней границы ЦФ (НГЦФ) является важнейшим, ключевым элементом предложенной схемы.

Таким образом, в основе метода ветвей и границ лежит идея последовательного разбиения множества допустимых решений на подмножества (стратегия “разделяй и властвуй”) и оценивания получаемых при разбиении частей. Каждый шаг алгоритма разбиения сопровождается проверкой условия того, содержит ли конкретное подмножество оптимальное решение или нет.

### **3.2.1. Описание входных данных**

# Определяемся с вводом данных

way = input('Введите способ ввода данных (ручной, рандом\_несим, рандом\_сим, файл): ')

Входные данные могут быть представлены в следующем виде (с примером реализации в коде):

1. Ручного ввода

**if** way == 'ручной':

# Вводим количество городов и матрицу от руки

n = int(input('Введите количество городов: '))

**print**('Введите матрицу')

**for** i **in** range(n): matrix.append(list(map(float, input().split())))

**for** i **in** range(n): matrix[i][i] = float('inf')

1. Рандомная генерация матрицы

Для этого мы изначально импортируем библиотеку для ввода случайным образом (**import** **random** # импортируем библиотеку для ввода случайным образом)

Матрица может быть как симметричной, так и не симметричной, поэтому для

ввода мы добавляем опции ввода: рандом\_несим и рандом\_сим:

**elif** way == 'рандом\_несим' **or** way == 'рандом\_сим':

# Генерируем входные данные рандомом

n = random.randint(**3**, **7**)

matrix = [[random.randint(**0**, **50**) **for** i **in** range(n)] **for** j **in** range(n)]

**for** i **in** range(n): matrix[i][i] = float('inf')

**if** way == 'рандом\_сим':

**for** i **in** range(n):

**for** j **in** range(i + **1**, n):

matrix[j][i] = matrix[i][j]

**print**()

**print**('Сгенерированное количество городов: ', n)

**print**()

**print**('Сгенерированная матрица: ')

**for** row **in** matrix: **print**(\*row)

**print**()

**print**('Сгенерированное количество городов: ', n)

**print**()

**print**('Сгенерированная матрица: ')

**для** строки **в** матрице: **print**(\*строка)

1. Матрица, записанная в файле csv.

**elif** way == 'файл':

# Считываем исходные данные из файла

name = input('Сохраните ваш файл в папку с этой программой и введите его имя: ')

file = open(name, "r")

matrix = []

**for** line **in** file: matrix.append([float(x) **for** x **in** line.split()])

n = len(matrix[**0**])

**for** i **in** range(n): matrix[i][i] = float('inf')

**print**('Считанная матрица: ')

**for** row **in** matrix: **print**(\*row)

file.close()

# --------------------------------------

### **3.2.2. Описание алгоритма решения**

* + 1. Задаем данные для нашей матрицы (из пункта выше видно, что это можно сделать несколькими способами)
    2. Процесс инициализации массива для сохранения индексов и сохранение изначальной матрицы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10, инициализация массивов и сохранение изначальной матрицы

* + 1. Редуцирование матрицы, что предполагает вычитание элементов из столбцов и строк, для последующей оценки нулевых клеток и поиска ненулевой с максимальной оценкой

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11, редуцирование матрицы

* + 1. Находим нужный путь и записываем его в переменную, при этом удаляя ненужные пути
    2. Формируем путь путем выставления параметров, далее высчитывая его длину
    3. Выводим длину пути на экран, а также через какие города проходит (т.е. сам путь)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12, инициализация массивов и сохранение изначальной матрицы

### **3.2.3. Описание выходных данных**

На выходе мы получаем путь, который у нас был построен, а также стоимость:

Путь:

**3** --> **2** --> **4** --> **1** --> **3**

Стоимость:

**103**

## **3.3. Генетический алгоритм**

Генетический алгоритм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. В данном случае генетический алгоритм нам помогает решить задачу коммивояжера. Для того, чтобы приступить к описанию работы метода, нужно сделать несколько основополагающих теоретических вставок.

Начнем с нескольких определений, перефразированных в контексте TSP:

* Ген (Gene): город (в координатах (x, y));
* Особь (Individual): единственный путь, удовлетворяющий указанным выше условиям;
* Популяция (Population): набор возможных маршрутов (т. е. совокупность особей);
* Родители (Parents): два маршрута, которые объединяются для создания нового маршрута;
* Пул спаривания (Mating pool): набор родителей, которые используются для создания нашей следующей популяции (тем самым создавая следующее поколение маршрутов);
* Функция приспособленности (Fitness): функция, которая сообщает нам, насколько хорош каждый маршрут (в нашем случае, насколько короткое расстояние);
* Мутация (Mutation): способ внести разнообразие в нашу популяцию путем случайной смены местами двух городов на маршруте;
* Элитарность (Elitism): способ передать лучших особей следующему поколению.

Созданный GA работает следующим образом:

1. Создаем популяцию

2. Определяем функцию приспособленности.

3. Выбираем пул для спаривания.

4. Получаем потомство

5. Производим мутации

6. Повторяем.

### **3.3.1. Описание входных данных**

В качестве входных данных, также как и для задачи ветвей и границ, используется ввод пользователей следующих значений:

* Количество вершин (т.е городов);
* Матрица смежности, обрабатывая которую алгоритм формирует список городов с координатами (x, y) для последующей работы с ней.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13, входные данные, для решения задачи коммивояжера генетическим алгоритмом

Все данные вводятся с клавиатуры, при нажатии кнопки Enter.

### **3.3.2. Описание алгоритма решения**

1. Создаем начальную популяцию, случайным образом выбирая n различных возможных путей в графе.

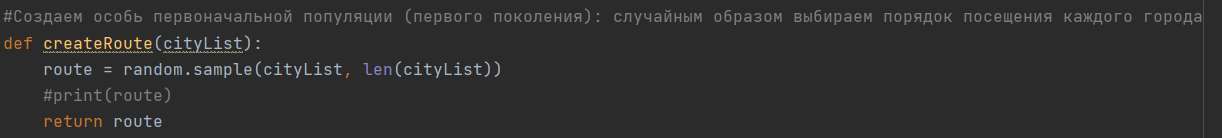


Рисунок 14, создание начальной популяции

2. Оцениваем m элитных предков, путем определения минимальных расстояний среди изначальной популяции. Для этого мы вводим понятие элитарность и сравниваем предков с последующим добавлением в пул спаривания

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15, ввод понятия эгалитарности

3. Для всех остальных (не элитных) предков проводим операцию скрещивания по следующим правилам:

3.1. Выбираем 2 случайных предка из списка

3.2. Для первого из пары выбираем  список генов, которые будут унаследованы потомком.

3.3. Для второго предка вычеркиваются гены, имеющиеся у первого предка

3.4. Оставшиеся гены второго предка переносятся в потомка с сохранением их порядка для 2 предка.

4. Операция повторяется пока количество потомков не станет таким, чтобы популяция не изменила численность.

5. Далее в полученной популяции потомков осуществляется мутация генов по следующему алгоритму:

5.1 Для любого потомка разыгрывается возможность мутации путем сравнения случайного числа с константой алгоритма.

5.2 Если для заданного потомка состоялась мутация- то случайным образом выбирается соседняя пара генов, которая меняется местами.

5.3 Если мутация не состоялась - то порядок генов потомка не меняется

6. Полученная в результате пунктов 2–5 популяция принимается за начальную, для следующего шага алгоритма, т.е. мы создаем новую популяцию, а далее оцениваем ее.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15, создание новой популяции

8. В итоге оценивается самый короткий путь, полученный на каждой итерации, и выводится результат.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15, получение самого короткого пути "FINAL DISTANCE"

### **3.3.3. Описание выходных данных**

В качестве выходных данных мы получаем сформированную нами матрицу, данные матрицы, преобразованные в список городов с координатами. Конечно, мы также хотим увидеть лучший маршрут и то, насколько мы улучшились, поэтому мы фиксируем и выводим начальное расстояние, конечное расстояние, а также лучший маршрут.

Далее выходными данными является матрица, которая получается с ручного ввода, первоначальный список городов, далее с помощью функции print поэтапно показано действие генетического кода, изменения, произошедшие в популяциях, также выводит первоначальную и финальную дистанцию и полученный путь

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 16, выходные данные после запуска генетического алгоритма

Завершает список выходных данных график, который нарисован был в ходе запуска нашей последней функции.



Рисунок кратчайшего расстояния от каждого поколения

# **4. Варианты использования системы**

Для пользователя предусмотрены два варианта использования системы: средствами Python и средствами MS Excel. Пользователь может выбрать любой удобный для него вариант.

## **4.1. ВИ\_1\_средствами Python.**

1. Пользователь для решения задачи коммивояжера обращается к разработанным методам ветвей и границ и генетическому методу, представленным в данной документации.
2. Пользователь выбирает метод решения задачи коммивояжера из представленных.
   1. Пользователь выбирает только один метод.
   2. Пользователь решает использовать все методы.
3. Приведенный в методе код пользователь использует в имеющейся к среде разработки для Python.
4. Пользователь запускает код.
   1. Для метода ветвей и границ запускается консоль, куда пользователь должен ввести входные данные, приведенные в пункте 5 настоящего ВИ.
   2. Для генетического метода ввод должен осуществляться через параметры функции в коде:

geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations)

1. Пользователю должен ввести следующие входные данные.
   1. Для метода ветвей и границ:
      1. Ввести способ ввода данных:

* «ручной»
* «рандом»
* «файл»
  + 1. Ввести количество городов:
* одно целое число.
  + 1. Матрица затрат:
       1. Ввести значения первой строки весовой матрицы через пробел:
* через пробел
* в одну строку
  + - 1. Ввести значения второй строки весовой матрицы через пробел:
* через пробел
* в одну строку
  + - 1. Ввести значения третьей строки весовой матрицы через пробел:
* через пробел
* в одну строку
  + - 1. Ввести значения четвертой строки весовой матрицы:
* через пробел
* в одну строку
  + - 1. Ввести значения пятой строки весовой матрицы:
* через пробел
* в одну строку
  1. Для генетического метода:
     1. Ввести количество городов: одно целое число
     2. Значения весовой матрицы:
* ввести каждое значение отдельно через Enter
  + 1. Количество особей в каждом поколении: целое число
    2. Количество элитных особей: целое число.
    3. Частота мутаций для данного гена: целое число.
    4. Количество поколений: целое число.

1. Программа считывает данные.
2. Пользователю должен выводиться результат в следующем виде:

* строка с номерами вершин в формате: «3 --> 2 --> 4 --> 1 --> 3»
* стоимость пути: действительное число

## **4.2. ВИ\_2\_средствами MS Excel.**

1. После обработки скрипта пользователю выводится результат.
2. Пользователь для решения задачи коммивояжера обращается к методу Дейкстры, представленному в данной документации.
3. Пользователь должен открыть приложение MS Excel.
4. Пользователь вручную следует разработанному алгоритму.
5. Пользователь должен получить ответ .

# **5. Архитектура решения**

**МЕТОД ВЕТВЕЙ И ГРАНИЦ**

Подключаем необходимые библиотеки для решения:

**import** **random** # импортируем библиотеку для ввода случайным образом

Объявляем вспомогательные функции, необходимые для решения:

**def** **minmin**(lst, myindex):

'''

Функция нахождения минимального элемента, исключая текущий элемент

'''

**return** min(x **for** idx, x **in** enumerate(lst) **if** idx != myindex)

**def** **deldel**(matrix, index1, index2):

'''

функция удаления нужной строки и столбцах

'''

**del** matrix[index1]

**for** i **in** matrix:

**del** i[index2]

return matrix

# --------------------------------------

# Обозначение переменных

matrix = []

H = **0**

PathLenght = **0**

Str = []

Stb = []

res = []

result = []

StartMatrix = []

# --------------------------------------

# Определяемся с вводом данных

way = input('Введите способ ввода данных (ручной, рандом\_несим, рандом\_сим, файл): ')

**if** way == 'ручной':

# Вводим количество городов и матрицу от руки

n = int(input('Введите количество городов: '))

**print**('Введите матрицу')

**for** i **in** range(n): matrix.append(list(map(float, input().split())))

**for** i **in** range(n): matrix[i][i] = float('inf')

**elif** way == 'рандом\_несим' **or** way == 'рандом\_сим':

# Генерируем входные данные рандомом

n = random.randint(**3**, **7**)

matrix = [[random.randint(**0**, **50**) **for** i **in** range(n)] **for** j **in** range(n)]

**for** i **in** range(n): matrix[i][i] = float('inf')

**if** way == 'рандом\_сим':

**for** i **in** range(n):

**for** j **in** range(i + **1**, n):

matrix[j][i] = matrix[i][j]

**print**()

**print**('Сгенерированное количество городов: ', n)

**print**()

**print**('Сгенерированная матрица: ')

**for** row **in** matrix: **print**(\*row)

**elif** way == 'файл':

# Считываем исходные данные из файла

name = input('Сохраните ваш файл в папку с этой программой и введите его имя: ')

file = open(name, "r")

matrix = []

**for** line **in** file: matrix.append([float(x) **for** x **in** line.split()])

n = len(matrix[**0**])

**for** i **in** range(n): matrix[i][i] = float('inf')

**print**('Считанная матрица: ')

**for** row **in** matrix: **print**(\*row)

file.close()

**print**()

**print**('-------------------------------')

**print**()

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 21, Вывод от функции "print", после ввода матрицы

# --------------------------------------

# Инициализируем массивы для сохранения индексов

**for** i **in** range(n):

Str.append(i)

Stb.append(i)

# --------------------------------------

# Сохраняем изначальную матрицу

**for** i **in** range(n):

StartMatrix.append(matrix[i].copy())

# --------------------------------------

# Редуцируем

**while** True:

# Вычитаем минимальный элемент в строках

**for** i **in** range(len(matrix)):

temp = min(matrix[i])

H += temp

**for** j **in** range(len(matrix)):

matrix[i][j] -= temp

# Вычитаем минимальный элемент в столбцах

**for** i **in** range(len(matrix)):

temp = min(row[i] **for** row **in** matrix)

H += temp

**for** j **in** range(len(matrix)):

matrix[j][i] -= temp

# Оцениваем нулевые клетки и ищем нулевую клетку с максимальной оценкой

NullMax = **0**

index1 = **0**

index2 = **0**

tmp = **0**

**for** i **in** range(len(matrix)):

**for** j **in** range(len(matrix)):

**if** matrix[i][j] == **0**:

tmp = minmin(matrix[i], j) + minmin((row[j] **for** row **in** matrix), i)

**if** tmp >=> NullMax:

NullMax = tmp

index1 = i

index2 = j

# Находим нужный нам путь, записываем его в res и удаляем все ненужное

res.append(Str[index1] + **1**)

res.append(Stb[index2] + **1**)

oldIndex1 = Str[index1]

oldIndex2 = Stb[index2]

**if** oldIndex2 **in** Str **and** oldIndex1 **in** Stb:

NewIndex1 = Str.index(oldIndex2)

NewIndex2 = Stb.index(oldIndex1)

matrix[NewIndex1][NewIndex2] = float('inf')

**del** Str[index1]

**del** Stb[index2]

matrix = deldel(matrix, index1, index2)

**if** len(matrix) == **1**: **break**

# --------------------------------------

# Формируем порядок пути

**for** i **in** range(**0**, len(res) - **1**, **2**):

**if** res.count(res[i]) < **2**:

result.append(res[i])

result.append(res[i + **1**])

**for** i **in** range(**0**, len(res) -1 , **2**):

**for** j **in** range(**0**, len(res) -1 , **2**):

**if** result[len(result) -1 ] == res[j]:

result.append(res[j])

result.append(res[j + **1**])

# --------------------------------------

# Считаем длину пути

**for** i **in** range(**0**, len(result) - **1**, **2**):

**if** i == len(result) -2 :

PathLenght += StartMatrix[result[i] -1 1][result[i + **1**] -1 ]

PathLenght += StartMatrix[result[i + **1**] -1 1][result[**0**] -1 ]

**else**:

PathLenght += StartMatrix[result[i] -1 1][result[i + **1**] -1 ]

Изображение выглядит как текст, электроника, клавиатура

Автоматически созданное описание

Рисунок 22, редуцирование матрицы

# --------------------------------------

# Выводим путь

**print**()

**print**("Путь:")

index = **1**

**while** index < len(result):

**if** result[index] **in** result[: index]:

result.pop(index)

**else**:

index += **1**

result.append(result[**0**])

**print**(\*result, sep=" --> ")

# --------------------------------------

# Выводим длину пути

**print**()

**print**("Стоимость:")

**print**(PathLenght)

**ГЕНЕТИЧЕСКИЙ АЛГОРИТМ**

**import** **numpy** **as** **np**, **random**, **operator**, **matplotlib.pyplot** **as** **plt**

**import** **pandas** **as** **pd**

Сначала мы создаем класс City, который позволит нам обеспечить формирование и обработку городов по координатам (x, y). Определяем величину расстояния и создаем удобный способ вывода городов в виде координат с помощью \_\_repr\_\_.

#Создаем класс City, позволяющий нам создавать и обрабатывать наши города

**class** **City**:

**def** **\_\_init\_\_**(self, x, y):

self.x = x

self.y = y

#добавляем расчет расстояния (используя теорему Пифагора)

**def** **distance**(self, city):

xDis = abs(self.x - city.x)

yDis = abs(self.y - city.y)

distance = np.sqrt((xDis \*\* **2**) + (yDis \*\* **2**))

return distance

#добавляем способ вывода городов в виде координат

**def** **\_\_repr\_\_**(self):

**return** "(" + str(self.x) + "," + str(self.y) + ")"

Далее формируем класс Fitness. В нашем случае мы будем рассматривать пригодность (fitness) как обратную величину к расстоянию маршрута. Мы хотим минимизировать расстояние маршрута, исходя из этого выбираем наиболее высокий показатель пригодности. Рассматриваемый нами метод реализуется для закрытого типа задачи. Следовательно, начальная точка отправления должна быть и завершающей, это достигается за счет дополнительного расчета в строке 13 расчета расстояния.

#Создаем класс fitness - пригодность как обратную величину к расстоянию маршрута

**class** **Fitness**:

**def** **\_\_init\_\_**(self, route):

self.route = route

self.distance = **0**

self.fitness= **0.0**

**def** **routeDistance**(self):

**if** self.distance ==**0**:

pathDistance = **0**

**for** i **in** range(**0**, len(self.route)):

fromCity = self.route[i]

toCity = None

**if** i + **1** < len(self.route):

toCity = self.route[i + **1**]

**else**:

toCity = self.route[**0**]

pathDistance += fromCity.distance(toCity)

self.distance = pathDistance

**return** self.distance

**def** **routeFitness**(self):

**if** self.fitness == **0**:

self.fitness = **1** / float(self.routeDistance())

**return** self.fitness

Теперь мы можем создать нашу первоначальную популяцию (также известную как первое поколение). Для этого нам нужен способ создания функции, которая создает маршруты, удовлетворяющие нашим условиям (примечание: мы создадим наш список городов, когда фактически запустим GA в конце статьи). Чтобы создать особь, мы случайным образом выбираем порядок посещения каждого города:

#Создаем особь первоначальной популяции (первого поколения): случайным образом выбираем порядок посещения каждого города

**def** **createRoute**(cityList):

route = random.sample(cityList, len(cityList))

**print**(route)

**return** route

В результате получается одна особь, но нам необходимо обеспечить полную популяцию. Этот шаг реализуется в следующей функции. Исходя из заданного значения переменной «popSize» (по умолчанию оно равно 100) алгоритм осуществляет вызовы функции createRoute, пока не достигнет заданного количества маршрутов для нашей популяции.

#Создаем первую популяцию (список особей)

**def** **initialPopulation**(popSize, cityList):

population = []

**for** i **in** range(**0**, popSize):

population.append(createRoute(cityList))

**print**(population)

**return** population

#Эти функции используются только для создания начальной популяции. Последующие поколения будут производиться путем размножения и мутаций

Чтобы смоделировать «выживание наиболее приспособленных», мы можем использовать Fitness для ранжирования каждой особи в популяции. Выходными данными будет являться упорядоченный список с идентификаторами маршрутов и связанной оценкой пригодности.

#Задаем пригодность

**def** **rankRoutes**(population):

fitnessResults = {}

**for** i **in** range(**0**,len(population)):

fitnessResults[i] = Fitness(population[i]).routeFitness()

**return** sorted(fitnessResults.items(), key = operator.itemgetter(**1**), reverse = True)#Выходные данные - упорядоченный список с идентификаторами маршрутов и связанной оценкой пригодности

Для большей ясности мы создадим пул спаривания в два этапа. Первым шагом воспользуемся выходными данными rankRoutes, чтобы определить, какие маршруты выбрать в нашей функции selection, настраиваем колесо рулетки, вычисляя относительный вес физической формы для каждой особи, случайно выпавшее число с этими весами, чтобы выбрать наш пул спаривания. Мы также хотим придерживаться наших лучших маршрутов, поэтому мы вводим параметр элитарности (по умолчанию равен 20). В итоге функция выбора возвращает список идентификаторов маршрутов, который мы можем использовать для создания пула спаривания в функции matingPool.

#Создаем функцию выбора, которая будет использоваться для составления списка родительских маршрутов

**def** **selection**(popRanked, eliteSize):

selectionResults = []

df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"]) #Вычисляем относительный вес физической формы для каждой особи

df['cum\_sum'] = df.Fitness.cumsum()

df['cum\_perc'] = **100**\*df.cum\_sum/df.Fitness.sum()

**for** i **in** range(**0**, eliteSize):

selectionResults.append(popRanked[i][**0**]) #Вводим элитарность

**for** i **in** range(**0**, len(popRanked) - eliteSize):

pick = **100**\*random.random() #Сравниваем случайно выпавшее число с этими весами, чтобы выбрать наш пул спаривания

**for** i **in** range(**0**, len(popRanked)):

**if** pick <= df.iat[i,**3**]:

selectionResults.append(popRanked[i][**0**])

**break**

**return** selectionResults

Теперь, когда у нас есть идентификаторы маршрутов, мы можем создать пул спаривания. Этот шаг выполняется с помощью извлечения выбранных особей из нашей популяции.

#Создание пула спаривания

**def** **matingPool**(population, selectionResults):

matingpool = []

**for** i **in** range(**0**, len(selectionResults)):

index = selectionResults[i]

matingpool.append(population[index])

**print**(matingpool)

**return** matingpool

Создав пул спаривания, мы можем создать следующее поколение в процессе, называемом crossover. Нужно включить все местоположения ровно один раз. Чтобы соблюдать это правило, мы можем использовать специальную функцию breeding, называемую упорядоченным кроссовером. В упорядоченном кроссовере мы случайным образом выбираем подмножество первой родительской, а затем заполняем оставшуюся часть маршрута генами от второго родителя в том порядке, в котором они появляются, без дублирования каких-либо генов в выбранном подмножестве от первого родителя.

#Создание перекрестной функции для двух родителей для создания одного дочернего элемента

**def** **breed**(parent1, parent2):

child = []

childP1 = []

childP2 = []

geneA = int(random.random() \* len(parent1))

geneB = int(random.random() \* len(parent1))

startGene = min(geneA, geneB)

endGene = max(geneA, geneB)

**for** i **in** range(startGene, endGene):

childP1.append(parent1[i])

childP2 = [item **for** item **in** parent2 **if** item **not** **in** childP1]

child = childP1 + childP2

**print**(child)

**return** child

Затем мы обобщим это и создадим нашу дочернюю популяцию. В строке 5 мы используем элитарность, чтобы сохранить лучшие маршруты от текущей популяции. Затем мы используем функцию breed, чтобы заполнить оставшуюся часть следующего поколения.

**def** **breedPopulation**(matingpool, eliteSize):

children = []

length = len(matingpool) - eliteSize

pool = random.sample(matingpool, len(matingpool))

**for** i **in** range(**0**,eliteSize):

children.append(matingpool[i])

**for** i **in** range(**0**, length):

child = breed(pool[i], pool[len(matingpool)-i-**1**])

children.append(child)

**print**(children)

**return** children

Мутация выполняет важную функцию в GA, поскольку помогает избежать локальной конвергенции путем введения новых маршрутов, которые позволят нам исследовать другие части пространства решений, воспользуемся перестановкой. Это означает, что с указанной низкой вероятностью два города поменяются местами на нашем маршруте. Мы сделаем это для одной особи в нашей функции mutate:

#Создание функции для изменения одного маршрута

**def** **mutate**(individual, mutationRate):

**for** swapped **in** range(len(individual)):

**if**(random.random() < mutationRate):

swapWith = int(random.random() \* len(individual))

city1 = individual[swapped]

city2 = individual[swapWith]

individual[swapped] = city2

individual[swapWith] = city1

**return** individual

#Создание функции для запуска мутации по всей популяции

**def** **mutatePopulation**(population, mutationRate):

mutatedPop = []

**for** ind **in** range(**0**, len(population)):

mutatedInd = mutate(population[ind], mutationRate)

mutatedPop.append(mutatedInd)

**return** mutatedPop

Соберем все эти части вместе, чтобы создать функцию, которая порождает новое поколение. Сначала мы ранжируем маршруты в текущем поколении с помощью rankRoutes. Затем мы определяем наших потенциальных родителей, запустив функцию selection, которая позволяет нам создать пул спаривания с помощью функции matingPool. Наконец, мы затем создаем наше новое поколение, используя функцию blendPopulation, а затем применяем мутацию с помощью функции mutatePopulation.

#Соберем все шаги вместе, чтобы создать следующее поколение

**def** **nextGeneration**(currentGen, eliteSize, mutationRate):

popRanked = rankRoutes(currentGen) #Ранжируем маршруты в текущем поколении

selectionResults = selection(popRanked, eliteSize) #Определяем наших потенциальных родителей

matingpool = matingPool(currentGen, selectionResults)

children = breedPopulation(matingpool, eliteSize) #Создаем наше новое поколение

nextGeneration = mutatePopulation(children, mutationRate) #Применяем мутацию

**return** nextGeneration

#Заключительный шаг: создание генетического алгоритма

**def** **geneticAlgorithm**(population, popSize, eliteSize, mutationRate, generations):

pop = initialPopulation(popSize, population)

**print**("Initial distance: " + str(**1** / rankRoutes(pop)[**0**][**1**]))

**for** i **in** range(**0**, generations):

pop = nextGeneration(pop, eliteSize, mutationRate)

**print**("Final distance: " + str(**1** / rankRoutes(pop)[**0**][**1**]))

bestRouteIndex = rankRoutes(pop)[**0**][**0**]

bestRoute = pop[bestRouteIndex]

**print**('Полученный путь:',bestRoute)

**return** bestRoute

#Создание списка городов

cityList = []

matrix = []

numcity = int(input('Введите количество вершин(городов):'))

**print**('Введите элементы матрицы весов через Enter')

**for** i **in** range(numcity):

matrix.append([])

**for** j **in** range(numcity):

matrix[i] += [input("Element: ")]

#Запускаем генетический алгоритм

geneticAlgorithm(population=cityList, popSize=**5**, eliteSize=**1**, mutationRate=**0.01**, generations=**25**)

**def** **geneticAlgorithmPlot**(population, popSize, eliteSize, mutationRate, generations):

pop = initialPopulation(popSize, population)

progress = []

progress.append(**1** / rankRoutes(pop)[**0**][**1**])

**for** i **in** range(**0**, generations):

pop = nextGeneration(pop, eliteSize, mutationRate)

progress.append(**1** / rankRoutes(pop)[**0**][**1**])

plt.plot(progress)

plt.ylabel('Distance')

plt.xlabel('Generation')

plt.show()

geneticAlgorithmPlot(population=cityList, popSize=**5**, eliteSize=**1**, mutationRate=**0.01**, generations=**25**)

# **6. Тестирование**

Далее приведена таблица результатов решения задач разными методами.

*Таблица 2. Результаты тестирования программы*

|  |  |  |  |
| --- | --- | --- | --- |
| **Входные данные** | **Метод Дейкстры** | **Метод ветвей и границ** | **Метод GA** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0 | 0 | 9 | 7 | 2 | | 0 | 0 | 4 | 8 | 10 | | 9 | 4 | 0 | 3 | 11 | | 7 | 8 | 3 | 0 | 5 | | 2 | 10 | 11 | 5 | 0 | | Порядок обхода вершин: 1-5-4-3-2  Время:14 | Путь:  1 --> 5 --> 4 --> 3 --> 2 --> 1  Стоимость:  14.0 | Время: 14.0  Полученный путь: [(96,94), (123,167), (123,125), (173,26), (38,14)] |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 0 | 6 | 4 | 8 | 7 | 14 | | 6 | 0 | 7 | 11 | 7 | 10 | | 4 | 7 | 0 | 4 | 3 | 10 | | 8 | 11 | 4 | 0 | 5 | 11 | | 7 | 7 | 3 | 5 | 0 | 7 | | 14 | 10 | 10 | 11 | 7 | 0 | | Порядок обхода вершин: 1-3-2-5-6-4  Время:52 | Путь:  6 --> 5 --> 4 --> 3 --> 1 --> 2 --> 6  Стоимость:  36.0 | Время: 36.0  Полученный путь: [(157,196), (196,141), (24,7), (66,104), (88,107), (96,113)] |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 18 | 40 | 27 | 15 | 4 | 13 | 38 | 15 | | 18 | 0 | 33 | 9 | 19 | 26 | 18 | 8 | 35 | | 38 | 33 | 0 | 17 | 22 | 14 | 26 | 22 | 11 | | 25 | 10 | 15 | 0 | 33 | 22 | 6 | 20 | 5 | | 15 | 21 | 21 | 31 | 0 | 10 | 26 | 33 | 27 | | 6 | 27 | 16 | 24 | 10 | 0 | 22 | 25 | 32 | | 12 | 19 | 26 | 5 | 25 | 21 | 0 | 28 | 20 | | 36 | 7 | 24 | 21 | 31 | 27 | 26 | 0 | 13 | | 15 | 33 | 10 | 5 | 27 | 32 | 19 | 12 | 0 | | Путь:  3 - 9 -8 - 2 -4 - 7 -1 - 6 -5 -3  Время:  92 | Путь:  3 --> 9 --> 8 --> 2 --> 4 --> 7 --> 1 --> 6 --> 5 --> 3  Стоимость:  92.0 | Время:  92.0  Полученный путь: [(24,123), (80,117), (93,197), (159,157), (160,91), (199,72), (164,5), (136,23), (14,50)] |

На основе данной таблицы можно сделать вывод, что каждый метод дает верный результат решения поставленной задачи. Но стоит отметить, что метод Дейкстры возможно применять лишь в случае с небольшим количеством городов, так как требуется непосредственное построение графов, и поэтому возможность их переборов увеличивается. Метод ветвей и границ можно применять всегда, он эвристичен для поиска начального решения и отбрасывания заведомо плохих множеств. Так же разработанный универсальный код на основе алгоритма ветвей и границ сможет решить любую задачу коммивояжера.

*Таблица 3. Результаты замеров времени работы программы*

|  |  |  |
| --- | --- | --- |
| **Количество городов** | **Время решения методом ветвей и границ** | **Время решения генетическим методом** |
| 5 | 0.00800085067749023 seconds | 35.4323308467865 seconds |
| 6 | 0.00400233268737793 seconds | 38.17642664909363 seconds |
| 9 | 0.00099754333496093 seconds | 38.47505831718445 seconds |
| 25 | 0.00800633430480957 seconds | 72.86701703071594 seconds |
| 50 | 0.37584638595581055 seconds | 95.14553689956665 seconds |
| 75 | 0.7597666740417480 seconds | 100.28262233734131 seconds |
| 100 | 1.9708476066589355 seconds | 136.7348427772522 seconds |
| 500 | 207.5471920967102 seconds | 136.1406126022339 seconds |

Как можно заметить, по таблице 3, время решения программы, основанной на методе ветвей и границ растет экспоненциально, а время программы, которая строится на алгоритме генетического метода, константное. При большем количестве городов, время программы метода ветвей и границ становится больше в то время, как генетический метод обрабатывает задачу намного быстрее.

# **7. Заключение**

В данной проектной документации решалась задача коммивояжера несколькими методами: методом Дейкстры, методом ветвей и границ и генетическим методом. Сначала была построена общая математическая модель, на основании которой затем были разработаны программы средствами Python для нахождения результата целевой функции с учетом алгоритмов решения различных методов, далее были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат. С помощью представленных в этой документации программ можно получить ответ на классические задачи коммивояжера открытого и закрытого типа.

Проведённый анализ наиболее рациональных методов решения задачи коммивояжера показал и определил области их эффективного действия: для малого числа вершин можно использовать точный метод Дейкстры; для большого числа вершин рациональнее применять метод ветвей и границ или метод генетический.

Хочется отметить наиболее эффективный код программы, которым, по нашему мнению, будет являться алгоритм, основанный на методе ветвей и границ. Именно он в большинстве случаев значительно выигрывает во времени (при не большом количестве городов) и в точности предоставляемого решения в то время, как в генетическом методе принятия решения присутствует некоторая случайность.

# **ТЕОРИЯ ИГР**

# **1. Физическая модель задачи**

# **1.1. Условие антагонистической задачи**

Рынок производства головных уборов полностью поделен между двумя конкурирующими компаниями. Компания А производит головные уборы и аксессуары, которые так же производит компания В. Прибыль одной компании соответствует убытку другой. Необходимо найти чистую оптимальную стратегию для первой компании и выгоду, которую она получит в случае выбора стратегии. Прибыль (в млн. руб.) первой компании при разных стратегических ситуациях представлена в следующей таблице:

Таблица 1- Матрица выигрышей компании А и проигрышей компании В.

|  |  |  |  |
| --- | --- | --- | --- |
| Компания В  Компания А | Шапки | Кепки | 3онты |
| Шапки | 25 | 30 | 50 |
| Кепки | 65 | 25 | 40 |
| Зонты | 70 | 80 | 40 |

# **1.2. Биматричная игра**

Каждая компания намеревается получить наибольшую прибыль по производимой продукции. Необходимо найти равновесную ситуацию, отклонение от которой одной из компаний уменьшало бы ее прибыль. Возможные выигрыши фирм представлены в следующей таблице:

Таблица 2- Матрица выигрышей компании А и компании В.

|  |  |  |  |
| --- | --- | --- | --- |
| Компания В  Компания А | Шапки | Кепки | 3онты |
| Шапки | 25, 70 | 30, 25 | 50, 60 |
| Кепки | 65, 35 | 25, 50 | 40, 55 |
| Зонты | 70, 40 | 80, 85 | 40, 45 |

# **1.3. Игра с природой в условиях риска**

Компания А решила провести исследование и изучить оптимальный спрос покупателей на их продукцию, который бы зависел от осенней погоды (теплая осень, холодная осень и дождливая осень). При различных погодных условиях компания А, обладающая определенной возможной продуктовой линейкой, может получить различный доход, который задан матрицей выигрышей в тыс. руб.:

Таблица 3- Матрица выигрышей компании А при определенном состоянии П.

|  |  |  |  |
| --- | --- | --- | --- |
| Компания В  Компания А | Теплая осень | Холодная осень | Дождливая осень |
| Шапки | 20 | 80 | 50 |
| Кепки | 75 | 10 | 25 |
| Зонты | 8 | 15 | 65 |
| Вероятность | 0,4 | 0,3 | 0,3 |

Проанализировав статистику предыдущих лет, была выведена следующая вероятность погоды:

* П1- теплая осень с вероятностью 0,4,
* П2 - холодная осень с вероятностью 0,3
* П3- дождливая осень с вероятностью 0,3

В связи с возможными изменениями погоды определить стратегию фирмы в выпуске продукции, обеспечивающую ей максимальный доход.

# **1.4. Игра с природой в условиях неопределенности**

Компания А по-прежнему решила провести исследование оптимального получения прибыли от продукции при различных погодных условиях в период осени, однако теперь компания не имеет данных о возможных вероятностях погодных условий. Однако экспертами был просчитан показатель наилучшего варианта наступления погоды, которая была бы наиболее выгодна для компании: показатель оптимизма равен 0,4.

Таблица 4 - Платежная матрица рисков.

|  |  |  |  |
| --- | --- | --- | --- |
| Компания В  Компания А | Теплая осень | Холодная осень | Дождливая осень |
| Шапки | 20 | 80 | 50 |
| Кепки | 75 | 10 | 25 |
| Зонты | 8 | 15 | 65 |

# **2. Математическая модель задачи**

Составим идеализированную математическую модель коллективного поведения: несколько индивидуумов (участников, игроков) влияют на ситуацию (исход игры), причем их интересы (выигрыши при различных возможных ситуациях) различны.

В игре участвуют игроки. Множество игроков обозначим буквой N. Таким образом, элементами множества N можно считать числа – номера игроков. В игре могут участвуют N={1, 2, n} игроков.

Каждый игрок имеет множество действий (альтернатив), которые он может осуществлять. Если каждый из игроков осуществил какое-то действие, то реализовался исход игры. Исходы имеют для игроков разную ценность. Рациональный игрок стремиться к достижению как можно более благоприятного для себя исхода. Однако никакой игрок не в состоянии обеспечить наилучший для себя исход только за счет собственных действий. Принимая решение о выборе действия, игрок может учитывать интересы и возможные действия других игроков, влияющие на исход игры. В этом отметим отличие теоретико-игровой постановки задачи принятия решений от задачи оптимизации.

По виду функций выигрыша игры делятся на: матричные, биматричные, непрерывные, выпуклые и др. Поэтому стоит поделить описание математической модели на соответствующий тип игры.

# **2.1. Антагонистическая игра.**

У первого игрока А есть набор из n возможных действий, а у игрока В – набор из m возможных действий. И игроков имеются следующие стратегии:

* стратегии игрока А: A1, A2, А3, …, An;
* стратегии игрока В: В1,В2,В3, …, Вm.

Они составляют SA и SB множества стратегий игроков:

SA = {A1,A2, …, An,} - множество стратегий игрока А;

SB = {B1,B2, …, Bm} - множество стратегий игрока В.

В зависимости от выбора своих стратегий в игре возможно возникновение nxm исходов. Все множество исходов - декартово произведение множеств чистых стратегий, которое представляет собой перечисление пар возможных комбинаций стратегий:

SA × SB = {(A1, B1),(A1, B2) … (A1,Bm), (A2,B1), …, (An, Bm)}.

При обозначении исходов на первой позиции указываем стратегию первого игрока, а на второй позиции – стратегию второго.

Оба игрока могут по-разному оценивать свою удовлетворенность от складывающихся исходов в результате разыгрывания игры. Однако для всех исходов должна быть определена функция - FA и FB, которые позволят сравнить эти исходы и которые также называются функциями выигрыша. Выигрыш игроков в исходах может измеряться в различных величинах (денежные единицы, тонны или килограммы, кол-во товара

Пусть игрок А выбирает одну из своих стратегий Аi, а игрок В выбирает одну из своих стратегий Вj, где і - номер стратегии первого игрока, a j - номер стратегии второго игрока. В результате игры возникнет ситуация (Аi, Bj), которая будет характеризоваться выигрышами:

* FA (Ai, Bj) = ai,j - выигрыш первого игрока в ситуации, когда первый игрок выбрал стратегию Аi, а второй игрок Вj;
* FB (Ai, Bj) = bi,j выигрыш второго игрока в ситуации, когда первый игрок выбрал стратегию Аi, а второй игрок Вj, где под аi,j, и bi,j, обозначаем конкретные выигрыши игроков в сложившихся ситуациях.

Выигрыши игроков, в зависимости от выбираемых стратегий, приведем в виде матрицы.

Таблица 5-Выигрыши А Таблица 6-Выигрыши В

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B1 | B2 | … | Bm |  |  |  |  | B1 | B2 | … | Bm |
|  | A1 | a1,1 | a1,2 | … | a1,m |  |  |  | A1 | b1,1 | b1,2 | … | b1,m |
| A = | A2 | a2,1 | a2,2 | … | a2,m |  |  | B = | A2 | b2,1 | b2,2 | … | b2,m |
|  | … | … | … | … | … |  |  |  | … | … | … | … | … |
|  | An | an,1 | an,2 | … | an,m |  |  |  | An | bn,1 | bn,2 | … | bn,m |

Для случая антагонистических игр, когда выигрыш одного игрока происходит за счет проигрыша другого игрока, функции выигрышей FA и FB противоположны для всех исходов игры:

Или

В этом случае матрица выигрышей В может быть получена из матрицы A:

В антагонистических играх, которые также называются играми с нулевой или постоянной суммой, достаточно указывать только одну матрицу первого игрока, потому что матрицу выигрышей второго игрока всегда можно получить из матрицы первого. В результате элементы одной матрицы можно интерпретировать как выигрыши первого игрока и проигрыши второго игрока как в таблице 1, где аi,j-выигрыш первого игрока и проигрыш второго.

Основной задачей антагонистических игр является поиск седловых точек и оптимальных стратегий, приводящих к ним.

1. Для каждой стратегии игрока А ищется гарантированный выигрыш (показатель эффективности стратеги): ai = minj ai,j как минимальный элемент в строке.

2. Среди гарантированных выигрышей ищут наибольший – нижнюю цену игры: a = maxi ai = maxi minj ai,j, а стратегию, приводящую к максимальному гарантированному выигрышу – максиминная стратегия.

3. Для каждой стратегии игрока В ищется наибольший возможный проигрыш (неэффективность стратегии): βj = maxi ai,j как максимальный элемент в столбце.

4. Среди наибольших возможных проигрышей ищется наименьший (верхняя цена игры): β = minj βj= minj maxi ai,j. Стратегия, приводящая к минимальному возможному проигрышу - минимаксная.

5. Если нижняя и верхняя цена игры совпадают, то в игре есть седловые точки, которые располагаются на пересечении максиминных и минимаксных стратегий. Значение выигрыша игрока А и проигрыша игрока В в седловой точке также называют просто ценой игры: V = a = β. В игре может быть как несколько седловых точек, так может и не быть седловых точек.

Помимо чистых стратегий, где каждый игрок выбирал свою чистую стратегию, выбранную гарантированно и с полной реализацией, существуют смешанные стратегии. Смешанные стратегии представим в виде векторов:

X = (x1; x2; …,xn), (4)

Y = (y1; y2; …, ym (5)

где x ≥ 0 и у ≥ 0 - вероятности или доли, с которыми выбираются чистые стратегии A; и Bj, причем сумма всех элементов каждой смешанной стратегии равна единице,, .

В антагонистических играх условием седловой точки являлось следующее двойное неравенство, где \* - выигрыш:

ai,j\* ≤ ai\*,j\*≤ ai\*,j , i,j. (6)

То есть каждому игроку не выгодно отклоняться от своих чистых стратегий, а в смешанных стратегиях по-прежнему ни одному игроку не выгодно отклоняться от своих стратегий, но уже от смешанных:

Функция выигрыша будет являться смесью всех элементов матрицы, когда каждый элементы берется в пропорциях пропорционально вероятности выбора чистой стратегии.

F(X,Y) = x1 a1,1 y1 + x1 a1,2 y2 + … + x1 a1,m ym + x2 a2,1 y1 + x2 a2,2 y2 + … + x2 a2,m ym +………….+xn an,1 y1 + xn an,2 y2 + … + xn an,m ym =

Так как чистые стратегии - частный случай смешанных стратегий, можно рассчитывать функцию выигрыша от одной чистой стратегии и другой смешанной. Так как в чистых стратегиях единица стоит лишь на одной позиции, а на всех остальных нули, то функция выигрыша содержит только элементы одной этой стратегии:

В итоге функцию выигрыша от обеих смешанных стратегий записывают как функцию выигрыша от чистых стратегий и одной смешанной.

# **2.2. Биматричная игра**

В отличие от антагонистических игр в биматричных играх выигрыши игроков не противоположны. В некоторых случаях у игроков могут быть даже одинаковые интересы. Выигрыши игроков представляются двумя разными функциями FA и FB (см. таблицы 1 и 2).

Решение биматричных игр заключается в поиске ситуаций равновесия и в поиске равновесных стратегий. Стоит сразу заметить, что ситуации равновесия уже не будут называться седловыми точками, как в антагонистических играх.

Ситуации равновесия по-прежнему будут являться устойчивыми ситуациями, но из-за того, что рассматриваются выигрыши обоих игроков, ситуация равновесия является точкой максимума по обеим переменным, а не точкой максимума по одной и минимума по другой переменной, как в антагонистических играх.

Ситуации равновесия — это такие ситуации, в которых ни одному игроку не выгодно отклоняться от своих выбранных стратегий при условии, что другой игрок придерживается своей выбранной стратегии. Ситуация равновесия в чистых стратегиях имеет максимальный выигрыш игрока А в столбце и максимальный выигрыш игрока В в строке.

Заметим, что в антагонистических играх условием седловой точки являлось двойное неравенство (формула 7), то в биматричных играх критерием ситуации будет система неравенств:

Вместо правой части неравенства, говорившей о том, что проигрыш второго игрока может только увеличиться при отклонении от оптимальной стратегии, в биматричных играх имеем одновременное выполнение условий, говорящих о том, что выигрыш каждого игрока может уменьшиться при отклонении от ситуации равновесия.

Можем записать простой алгоритм для поиска ситуаций равновесия биматричных игр в чистых стратегиях.

1. В каждом столбце отметим максимальные выигрыши ai,j игрока А.

2. В каждой строке отметим максимальные выигрыши bi,j игрока В.

3. Среди всех ситуаций ситуациями равновесия являются те, которые отмечены одновременно максимальные.

В отличие от антагонистических игр в биматричных не выполняются теоремы о равнозначности и взаимозаменяемости седловых точек.

В чистых стратегиях может не быть ситуации равновесия, а в смешанных - обязательно есть хотя бы одна ситуация равновесия. Наличие ситуации равновесия гарантируется по теореме Джона Нэша, которая является более общим случаем для произвольных матричных игр. Для биматричных игр ситуации равновесия ищутся отличным от поиска седловых точек в антагонистических играх способом.

Смешанную стратегию игрока А и смешанную стратегию игрока В в виде векторов (см. формулу 1 и 2).

Согласно теореме о дополняющей нежесткости: если стратегии X\* и Y\* приводят к ситуации равновесия, и чистая стратегия Аk активна, т.е. входит в равновесную X\* с ненулевой вероятностью xk > 0, то:

FA(Ak, Y\*) = FA (X\*, Y\*). (11)

Аналогично, если чистая стратегия ВL, активна, т.е. входит в равновесную Y\* с ненулевой вероятностью уL > 0, то:

FB(X\*, Br) = FB (X\*,Y\*). (12)

Эту теорему доказывается тем, что функцию выигрыша можно рассчитать через функцию выигрыша от чистых стратегий FA (X\*, Y\*) = . Тогда для неактивных стратегий элементы суммы становятся равны нулю, и суммирование происходит только по активным стратегиям. Далее стоит заметить, FA (Ai, Y\*) не может быть больше FA (X\*, Y\*), иначе Аi была бы более предпочтительной, чем X\*, что означало бы, что стратегия X\* не равновесная. Тогда, если предположить, что для одной из активных стратегий FA (A;,Y\*) меньше FA (X\*, Y\*), то функция выигрыша FA (X\*, Y\*) = < FA (X\*, Y\*), что не возможно. Следовательно, для всех активных стратегий функция FA(Ai, Y\*) совпадает с FA (X\*, Y\*).

Эта теорема дает алгоритм нахождения ситуаций равновесия, схожий с алгоритмом в антагонистических играх.

# **2.2. Задача о принятии оптимального решения в условиях риска**

Для игры с природой так же необходима матрица игры, в которой в качестве стратегий второго игрока задаются возможные состояния природы. Пусть всего возможно m состояний природы. Обозначим состояния природы как Пj, где индекс j обозначает номер состояния природы j = 1, …, m. Элементы матрицы, в отличие от антагонистических игр, обозначают выигрыши только первого игрока, выигрыши природы не задаются:

Таблица 7 – Платежная матрица игры с природой

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | П1 | П2 | … | Пm |  |  |
|  | A1 | a1,1 | a1,2 | … | a1,m |  |  |
| A = | A2 | a2,1 | a2,2 | … | a2,m |  |  |
|  | … | … | … | … | … |  |  |
|  | An | an,1 | an,2 | … | an,m |  |  |

В зависимости от того, какая информация о возможных состояниях природы известна игроку, можно выделить некоторые критерии выбора оптимальных стратегий.

# **2.1.1.Критерий Байеса.**

В критерии предполагается, что игроку на основе имеющейся информации известны вероятности возникновения состояний природы q1, q2, …, qm. Тогда игрок может определить математическое ожидание выигрыша а, каждой своей стратегии Ai:

(13)

Полученный ожидаемый выигрыш , также еще называют показателем эффективности стратегии Ai. Оптимальной стратегией Ai (по критерию Байеса) считается стратегия, обладающая максимальным ожидаемым выигрышем:

Придерживаясь оптимальной стратегии, игрок в среднем будет выигрывать больше, чем при любой другой стратегии. Стоит заметить, что оптимальных стратегий может быть несколько в случае, если у некоторых стратегий ожидаемый выигрыш будет совпадать.

В этом случае игрок может принимать любую свою оптимальную стратегию, каждая из которых дает ему одинаковый ожидаемый выигрыш.

# **Критерий Лапласа.**

В тех случаях, когда игрок не знает значений вероятности возникновения состояний природы, но при этом считает, что эти состояния могут возникнуть одинаково правдоподобно. В этом случае можно предположить, что вероятности состояний природы равны между собой:

В этом случае критерий Лапласа является частным случаем критерия Байеса с равномерным распределением вероятности состояний. Причем расчет показателя эффективности стратегий упрощается:

(ai,1 + ai,2 + … + ai,m) = (16)

В итоге для определения оптимальной стратегии по критерию Лапласа достаточно определить стратегию с максимальной суммой выигрыша в строке:

# **Критерий Гермейра.**

Ориентирован на величину потерь, т.е. на отрицательные значения всех аij. При этом

maxi(air) = maxi(minj(aij)qj). (18)

В задачах преимущественно имеют дело с ценами и затратами, поэтому условие aij<0 обычно выполняется. В случае же, когда среди величин aij есть положительные значения, можно перейти к строго отрицательным значениям с помощью преобразования аij - a при подходящем образом подобранном a>0. При этом оптимальный вариант решения зависит от а.

Условия его применимости таковы:

* вероятности появления состояния Пj неизвестны;
* с появлением тех или иных состояний, отдельно или в комплексе, необходимо считаться;
* допускается некоторый риск;
* решение может реализоваться один или несколько раз.

## **Задача о принятии оптимального решения в условиях неопределенности.**

В отличие от игр с природой в условиях риска игры с природой в условиях неопределенности характерны тем, что нет никакой информации о возможных вероятностях состояний природы.

Матрица весов выглядит аналогично матрице весов для игры с природой в условиях риска (см. таблица 5).

Рассмотрим критерии, которыми может воспользоваться игрок в условиях неопределенности.

# **Критерий Вальда.**

Критерий Вальда является критерием крайнего пессимизма. Игрок действует крайне осторожно, стараясь, в случае не правильного выбора, потерять как можно меньше. Показателем эффективности Wi стратегии A; по критерию Вальда будет минимальный возможный выигрыш при использовании заданной стратегии (гарантированный выигрыш). Показатель эффективности определяется как наименьший элемент в строке:

Wi = ai,j. (19)

Оптимальной стратегией по критерию Вальда будет стратегия Ai\*, у которой показатель эффективности максимален:

Критерий Вальда совпадает с тем, как ищется нижняя цена игры в антагонистических играх в чистых стратегиях (максимин).

Максимаксный критерий: критерий q крайнего оптимизма и противоположный критерию Вальда. Игрок предполагает, что ему обязательно повезет, и он хочет выиграть как можно больше. Поэтому игрок выбирает такую стратегию, у которой возможен наибольший выигрыш. Показателем эффективности Мi стратегии Ai является наибольший выигрыш при заданной стратеги и определяется как наибольший элемент в строке:

Оптимальной стратегией по максимаксному критерию будет стратегия Аi\*, у которой показатель эффективности максимален.

# **Критерий Сэвиджа.**

Это критерий пессимизма, который ориентируется на максимальный риск и выбирает ту стратегию, у которой минимален максимальный риск.

Показатель неэффективности:

Si = maxi(aij) – aij (23)

Оптимальная стратегия с минимальным показателем неэффективности:

air = maxi(aij) = maxj(maxi(aij) - aij) (24)

Величину aij можно трактовать как максимальный дополнительный выигрыш, который достигается, если в состоянии Пj вместо варианта Аi выбирать другой, оптимальный для этого внешнего состояния вариант. Величину Si можно интерпретировать и как потери (штрафы), возникающие в состоянии Пj при замене оптимального для него варианта на вариант Аi. В последнем случае air представляет собой максимально возможные (по всем внешним состояниям Пj , j = {1,n}) потери в случае выбора варианта Аi.

Соответствующее критерию Сэвиджа правило выбора теперь трактуется так: каждый элемент матрицы решений |аij| вычитается из наибольшего результата max(аij) соответствующего столбца.

Разности Si образуют матрицу остатков |аij|. Эта матрица пополняется столбцом наибольших разностей air. Выбирают те варианты, в строках которых стоит наименьшее для этого столбца значение.

Требования, предъявляемые к ситуации, в которой принимается решение, совпадают с требованием к минимаксному критерию.

# **Критерий Гурвица.**

Критерий предполагает наиболее уравновешенную позицию, где оценочная функция, которая находится где-то между точкой зрения крайнего оптимизма и крайнего пессимизма, выглядит:

,

где С — весовой множитель.

Правило выбора согласно критерию Гурвица, формируется следующим образом: матрица решений |aij| дополняется столбцом, содержащим среднее взвешенное наименьшего и наибольшего результатов для каждой строки. Выбираются только те варианты, в строках которых стоят наибольшие элементы air этого столбца.

При С=1 критерий Гурвица превращается в минимаксный критерий. При С = 0 он превращается в критерий «азартного игрока»:

maxi (аir) = maxi(maxj(аij)), (26)

т.е. мы становимся на точку зрения азартного игрока, делающего ставку на то, что «выпадет» наивыгоднейший случай.

В технических приложениях сложно выбрать весовой множитель С, т.к. трудно найти количественную характеристику для тех долей оптимизма и пессимизма, которые присутствуют при принятии решения. Поэтому чаще всего С=1/2.

Критерий Гурвица применяется в случае, когда:

* о вероятностях появления состояния Пj ничего не известно;
* с появлением состояния Пj необходимо считаться;
* реализуется только малое количество решений;
* допускается некоторый риск.

# **3. Алгоритмы решения задачи**

Физическая модель решалась для 4 разновидностей «игровых» задач:

* Антагонистическая задача
* Биматричная игра
* Игра с природой в условиях риска
* Игра с природой в условиях неопределенности

## **3.1. Антагонистическая игра (игры с нулевой суммой)**

Антагонистическая игра или игра с нулевой суммой — термин теории игр. Антагонистической игрой называется некооперативная игра, в которой участвуют два или более игроков, выигрыши которых противоположны. Это значит, что победа одного игрока оборачивается проигрышем другого игрока. Решение задачи может быть выполнено в рамках чистых и смешанных стратегий. Поэтому целесообразно будет разделить это на два блока.

### **3.1.1. Описание входных данных**

#### **3.1.1.1 Входные данные для решения задачи с чистыми стратегиями (реализовано на Python)**

Входными данными являются:

* Количество стратегий для игрока А и для игрока Б:

Введите количество стратегий игрока А:

Введите количество стратегий игрока В:

* Название этих стратегий для двух игроков (вводится через пробел)

Введите названия стратегий для А через пробел в строку:

Введите названия стратегий для В через пробел в строку:

* Платежная матрица

Введите матрицу (строки - А, столбцы - В)

#### **3.1.1.2 Входные данные для решения задачи со смешанными стратегиями (реализовано на Excel)**

Входными данными для решения задачи со смешанными стратегиями будет являться платежная матрица, которая является физической моделью.

Таблица 8- Матрица выигрышей компании А и проигрышей компании В.

|  |  |  |  |
| --- | --- | --- | --- |
| Компания В  Компания А | Шапки | Кепки | 3онты |
| Шапки | 25 | 30 | 50 |
| Кепки | 65 | 25 | 40 |
| Зонты | 70 | 80 | 40 |

### **3.1.2. Описание алгоритма решения**

#### **3.1.2.1. Описание алгоритма решения для чистых стратегий**

Алгоритм решения можно представить следующим образом:

1. Пользователь выбирает несколько способов ввода данных. В коде представлено три варианта ввода: ручной, рандомная генерация, загрузка данных из файла.

'Введите способ ввода данных (ручной, рандом, файл): '

Если данные будут введены неверно, то будет выведена на экран пользователя ошибка.

**if** len(strat\_B) != m:

**print**('Данные введены неверно!')

exit()

1. Далее начинается само решение. Сначала ищутся минимумы по строкам (максимин)
2. Потом матрица транспонируется, чтобы найти максимумы по столбцам (после транспонирования которые становятся строками) (минимакс)

#### **3.1.2.2. Описание алгоритма решения для смешанных стратегий**

Алгоритм решения задачи для смешанных стратегий выглядит следующим образом:

1. В Excel переносится платежная матрица, которая необходима для решения задания.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1, платежная матрица

1. Применяем МИНИМАКС и МАКСИМИН для поиска максимумов по строкам и минимумов по столбцам.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2, МИНИМАКС\МАКСИМИН

1. Транспонируем матрицу и начинаем поиск решения

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3, транспонирование матрицы

1. Вводим новые переменные и прописываем с их учетом ограничения (изначально переменным присваиваем значения 1, 2 и 3 для того, чтобы они не были пустыми)



Рисунок 4, ввод новых переменных

1. Ограничения выглядят следующим образом:

(27)

Соответственно, ввиду того что строк 3, то и ограничений будет 3, а то значение, которое мы находим должно быть больше или равно единице.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 5, ввод ограничений

1. Вводим целевую функцию, которая представляет собой сумму переменных.
2. Применяем функцию «Поиск решений»

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6, "Поиск решений" для антагонистических задач

Параметрами для поиска решений являются:

* Ячейка со значением целевой функции
* Ячейки переменных
* Ограничения
* Метод решения (в данном случае применяется метод «Поиск решения лин. задач симплекс-методом», так как данная задача является задачей линейного программирования.

1. Прописываем цену игры, которая является обратной величиной целевой функции.

### **3.1.3. Описание выходных данных**

#### **3.1.3.1. Описание выходных данных для чистых стратегий**

Выходными данными являются:

1. Выигрыш обоих игроков при чистых стратегиях

Выигрыш А при чистой оптимальной стратегии: **40.0**

Выигрыш В при чистой оптимальной стратегии: **40.0**

1. Название стратегии, при которой игроки выигрывают:

Стратегия для А: Зонты

Стратегия для В: Шапки

#### **3.1.3.2. Описание выходных данных для смешанных стратегий**

Выходными данными в данном случае является смесь стратегий и вывод, который можно сделать по ней (смесь стратегий представляет собой обратную замену переменных, которые были введены в начале решения, умноженные на цену игры (т.е обратную величину целевой функции)

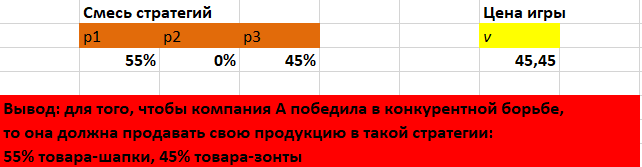


Рисунок 7, определение стратегий для задач со смешанными стратегиями

## **3.2. Биматричные игры**

В отличие от антагонистических игр в биматричных играх выигрыши игроков не противоположны. В некоторых случаях у игроков могут быть даже одинаковые интересы. Выигрыши игроков представляются двумя разными.

Решение биматричных игр заключается в поиске ситуаций равновесия и в поиске равновесных стратегий. Стоит сразу заметить, что ситуации равновесия уже не будут называться седловыми точками, как в антагонистических играх. При этом если равновесия не достигается, то задача решается смешанной стратегией. Аналогично антагонистической, биматричная игра также реализована на Python (чистые стратегии) и Excel (смешанные стратегии)

### **3.2.1. Описание входных данных**

#### **3.2.1.1. Описание входных данных для чистых стратегий**

Входными данными являются:

* Количество стратегий для игрока А и для игрока Б:

Введите количество стратегий игрока А:

Введите количество стратегий игрока В:

* Название этих стратегий для двух игроков (вводится через пробел)

Введите названия стратегий для А через пробел в строку:

Введите названия стратегий для В через пробел в строку:

* Платежные матрицы

Введите первую матрицу (строки - А, столбцы - В)

Введите вторую матрицу (строки - А, столбцы - В)

#### **3.2.1.2. Описание входных данных для смешанных стратегий**

Входными данными для решения задачи со смешанными стратегиями будут являться платежные матрицы, которые являются физической моделью.

Таблица 9- Матрица выигрышей компании А и компании В.

|  |  |  |  |
| --- | --- | --- | --- |
| Компания В  Компания А | Шапки | Кепки | 3онты |
| Шапки | 25, 70 | 30, 25 | 50, 60 |
| Кепки | 65, 35 | 25, 50 | 40, 55 |
| Зонты | 70, 40 | 80, 85 | 40, 45 |

### **3.2.2. Описание алгоритма решения**

#### **3.2.2.1. Описание алгоритма решения для чистых стратегий**

Алгоритм решения можно представить следующим образом:

1. Пользователь выбирает несколько способов ввода данных. В коде представлено три варианта ввода: ручной, рандомная генерация, загрузка данных из файла.

'Введите способ ввода данных (ручной, рандом, файл): '

Если данные будут введены неверно, то будет выведена на экран пользователя ошибка.

**if** len(strat\_B) != m:

**print**('Данные введены неверно!')

exit()

1. Происходит поиск равновесия по Нэшу. Если равновесие найдено, то программа выдает сразу ответ, если равновесий нет, то программа предупреждает о том, что стратегии нужно смешивать, что ведет нас к решению биматричной задачи со смешанными стратегиями

#### **3.2.2.2. Описание алгоритма решения для смешанных стратегий**

Смешанные стратегии применяются тогда, когда не было найдено равновесия по Нэшу

Алгоритм решения выглядит следующим образом:

1. Переносим платежные матрицы и задаем смесь стратегий (как было сделано в конце антагонистической задачи, но в данном случае мы их вводим сразу, они будут изменены после применения «Поиска решений», также мы их оставляем ненулевыми и сразу задаем 100%)

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 8, платежные матрицы и введение смеси стратегий

1. Прописываем цену игры, для этого мы составляем две матрицы с размерностями (в данном случае) 3х3, где элементы имеют следующий вид:

* Для матрицы А:

(28)

* Для матрицы Б:

(29)

Мы получаем те же самые значение (на данном этапе), что и в обеих матрицах. Цена игры будет равняться сумме всех элементов получившихся матриц

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 9, выявление цены игры для игрока А

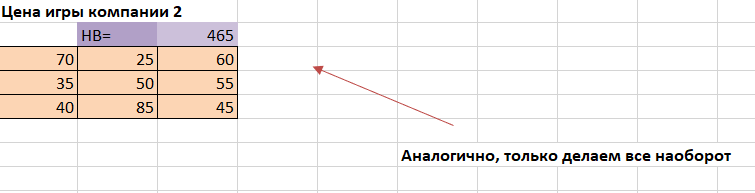


Рисунок 10, выявление цены игры для игрока B

1. Далее считаются суммы по строкам и по столбцам в матрицах, элементы в них будут равны:

(30)

(для матрицы Б результат транспонируется для удобства дальнейшего исчисления)

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 11, расчет суммы по строкам для элементов матрицы А

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 12, расчет суммы по строкам для элементов матрицы B

1. Вводим целевую функцию, она равна сумме цен игры для А и Б



Рисунок 13, расчет целевой функции

1. Вводим ограничения. Их можно разделить на три разных блока:

* Блок 1. Ограничения по матрице, полученной двумя шагами раннее. Её элементы не должны превышать цену игры по матрице А
* Блок 2. Аналогично А, только для матрицы Б
* Ограничение на смеси стратегий. Их сумма не должна превышать 100%

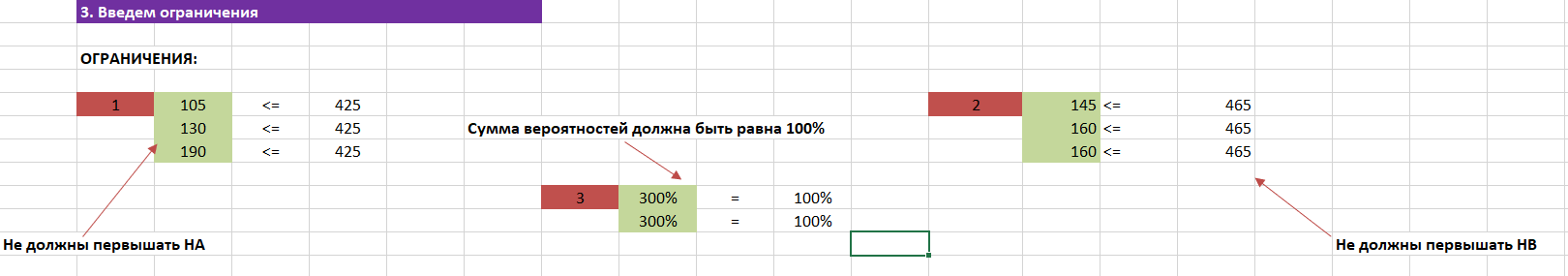


Рисунок 14, введение ограничений

1. Применяем «Поиск решения». Все аналогично антагонистической задаче: вводим целевую функцию, ее мы максимизируем, ячейками переменных являются смеси стратегий, все ограничения вводятся, метод решения будет «Поиск решения нелинейных задач методом ОПГ», так как эта задача не является задачей линейного программирования

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15, "Поиск решение" для решения биматричных задач со смешанными стратегиями

## **3.2.3. Описание выходных данных**

## **3.2.3. Описание выходных данных для чистых стратегий**

Выходными данными являются:

1. Количество равновесий по Нэшу (и предупреждение, если равновесий не найдено)

Количество равновесий по Нэшу: **0**

Стратегии нужно смешивать!

1. Выигрыш обоих игроков при чистых стратегиях

Выигрыш А при чистой стратегии: **80.0**

Выигрыш В при чистой стратегии: **85.0**

1. Название стратегии, при которой игроки выигрывают:

Стратегия для А: Шапки

Стратегия для В: Зонты

1. Суммарный выигрыш

Суммарный выигрыш: **165.0**

### **3.2.3. Описание выходных данных для смешанных стратегий**

Выходными данными будет являться полностью измененные матрицы, так как при поиске решения мы находим смеси стратегий, которые при изменении меняют полностью все значения. На основании этих данных пользователь уже может наглядно выявить, какую из стратегий для каждой компании нужно выбрать

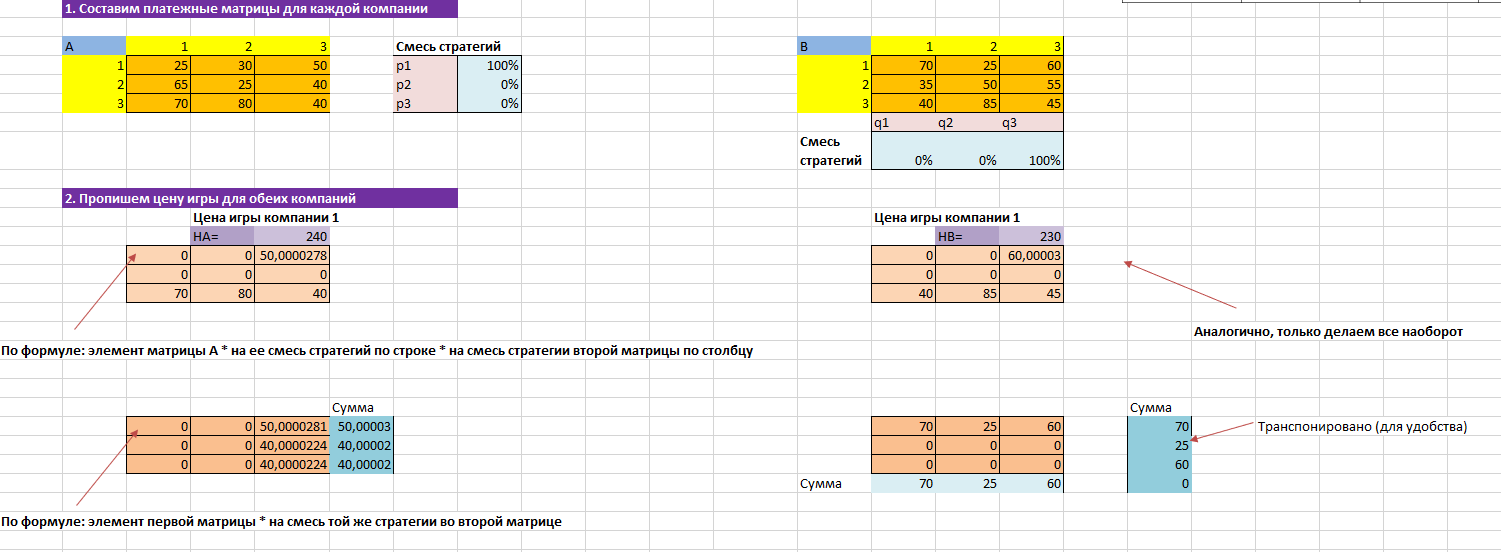


Рисунок 16, выходные данные по биматричным задачам со смешанными стратегиями



Рисунок 17, выходные данные по биматричным задачам со смешанными стратегиями

## **3.3. Задача о принятии оптимального решения в условиях риска**

Принятие решений в условиях риска, когда исход операции зависит не только от принятого решения, но и от случайных факторов (воздействий), статистические свойства которых известны или в принципе могут быть получены к нужному сроку. Такие задачи называют стохастическими (вероятностными) или задачами принятия решений в условиях риска. В стохастических ЗПР критерий оптимальности зависит, кроме вариантов решений и детерминированных факторов, также от фиксированных случайных факторов, законы распределения которых известны оперирующей стороне.

### **3.3.1. Описание входных данных**

Входными данными являются 3 возможных состояния природы и заданные вероятности их наступления, 3 стратегии компании А и матрица выигрышей размерностью 3х3.

Реализация на Python:

matrica = [[**20**, **80**, **50**], [**78**, **10**, **25**], [**8**, **15**, **65**]]

priroda = [**0.4**, **0.3**, **0.3**]

strat\_A = [**1**, **2**, **3**]

Реализация на Excel:

Стратегии первого игрока:

1. Шапки
2. Кепки
3. Зонты

Стратегии второго игрока – погодные условия осени:

1. Теплая осень
2. Холодная осень
3. Дождливая осень

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 18, стратегии игроков

Платежная матрица будет выглядеть следующим образом:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 19, матрица выигрышей

### **3.3.2. Описание алгоритма решения**

Для принятия решения, у нас есть несколько критериев: критерий Лапласа, критерий Баеса и критерий Гермейреа.

### **3.3.2.1. Критерий Лапласа.**

Для начала необходимо рассчитать вероятность наступления каждой стратегии природы, которая согласна критерию должна быть равновероятной (cм. формула 15).

matrica\_laplas = []

**for** i **in** range(len(matrica)):

k = -**1**

**for** j **in** range(len(matrica[i])):

k = k + **1**

matrica\_laplas.append(matrica[i][j] \* (**1** / m))

Далее, согласно формуле 17, получаем средневзвешенное значение:

s\_laplas = []

**for** i **in** range(m - **1**, len(matrica\_laplas) + **1**, m):

s\_laplas.append (sum (matrica\_laplas[i - (m - **1**) : i + **1**]))

По полученным средневзвешенным значениям определяем максимальное.

#Выигрыш А по критерию Лапласа:

max(s\_laplas))

#Стратегия для А:

strat\_A[s\_laplas.index(max(s\_laplas))])

### **3.3.2.2. Решение с помощью критерия Байеса.**

По критерию Байеса предлагается придать равные вероятности всем рассматриваемым стратегиям, после чего принять ту из них, при которой ожидаемый выигрыш окажется наибольшим.

Нам известны вероятности наступления каждой стратегии природы, которые представлены вектором R={0,4; 0,3; 0,3}.

Для начала необходимо найти вероятность достижения по каждой стратегии (по каждой строке) найти средневзвешенный выигрыш (см. формула 13), то есть:

matrica\_baes = []

**for** i **in** range(len(matrica)):

k = -**1**

**for** j **in** range(len(matrica[i])):

k = k + **1**

matrica\_baes.append(matrica[i][j] \* float(priroda[k]))

s\_baes = []

**for** i **in** range(m - **1**, len(matrica\_baes) + **1**, m):

s\_baes.append (sum (matrica\_baes[i - (m - **1**) : i + **1**]))

Оптимальной стратегией для компании A будет стратегия, обладающая максимальным ожидаемым выигрышем.

#Выигрыш А по критерию Баеса:

max(s\_baes))

#Стратегия для А:

strat\_A[s\_baes.index(max(s\_baes))])

### **Критерий Гермейера**

Чистые стратегии (реализовано на Python):

Правило выбора согласно критерию Гермейера формулируется следующим образом: матрица решений |аij| дополняется еще одним столбцом содержащим в каждой строке наименьшее произведение имеющегося в ней результата на вероятность соответствующего состояния Пj. Выбираются те варианты в строках которых находится наибольшее значениеe аij этого столбца.

Для начала необходимо получить матрицу Геймейеру, где условно каждую ячейку нужно умножить на вероятность наступления состояния природы.

#Гермейер

priroda\_gey = [priroda] \* m

matrica\_gey = []

**for** i **in** range(len(matrica)):

**for** j **in** range(len(matrica[i])):

matrica\_gey.append (matrica[i][j] \* priroda\_gey[i][j])

matrica\_gey\_1 = []

gey\_mn = []

**for** i **in** range(**0**, len(matrica\_gey), m):

matrica\_gey\_1.append (matrica\_gey[i : i + m])

**for** i **in** matrica\_gey\_1:

gey\_mn.append (min(i))

Теперь, согласно формуле 18, мы находим минимальное значение по каждой стратегии компании А, а затем выбираем максимальное из минимальных.

#Выигрыш А при оптимальной чистой стратегии по критерию Гермейера:

min(matrica[gey\_mn.index(max(gey\_mn))])

#Стратегия для А:

strat\_A[gey\_mn.index(max(gey\_mn))]

Смешанная стратегия (реализовано на Excel):

Для смешанной стратегии необходимо вычислить ограничения: каждое состояние природы умножить на переменные. Ограничение будет состоять в том, что полученные значения должны быть меньше или равны единице.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 28, ограничения и поиск переменных

При нахождении целевой функции, которая представляет собой сумму полученных переменных х1, х2 и х3 , будем ползоваться Поиском решения, чтобы минимизировать (обратная величина) функцию по избранным ограничениям.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 29, "Поиск решения"

Находим цену игры, которая измеряется в млн. руб \* вероятность, как число обратное от целевой функции:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 30, поиск цены игры

Теперь мы можем найти процентное соотношение комбинации стратегий, как произведение найденных переменных и цены игры v:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 31, смешанная стратегия

Для удобства составим дополнительную таблицу, где верхняя строка -СУММПРОИЗВ каждого значения стратегии компании при определенной стратегии природы и процентного распределения наступления стратегий природы:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 32, расчет процентного распределения и наступления стратегий природы

А для нижней строки найдем произведение ячейки верхней строки и вероятности состояния природы:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 33, поиск промежуточных значений

Таким образом, просуммировав нижнюю строку найденной таблицы, получим выигрыш в смешанной стратегии для компании А.

### **3.3.3. Описание выходных данных**

### **3.3.3.1. Критерий Лапласа**

Оптимальная стратегия по критерию Лапласа для компании А будет А1 – производить шапки, чтобы получить максимальную цену игры - прибыль в 50 млн руб.

Выигрыш А по критерию Баеса: 50

Стратегия для А: 1

### **3.3.3.2. Критерий Байеса**

Оптимальной стратегией для компании A видим, что считается стратегия, обладающая максимальным ожидаемым выигрышем, то есть А1 (Шапки) с прибылью в 47 млн руб.

Выигрыш А по критерию Баеса: 47

Стратегия для А: 1

### **3.3.3.3. Критерий Гермейра**

Чистые стратегии (реализовано на Python):

По критерию Гермейера на выходе получаем оптимальную чистую стратегию игрока А – Стратегия А1 (производство шапок) с ценой игры равной 20 млн руб.

Выигрыш А при оптимальной чистой стратегии по критерию Гермейера: 20

Стратегия для А: 1

Смешанная стратегия (реализовано на Excel):

А для смешанной стратегии получаем таблицу оптимальных смешанных стратегий для компании А П1-50%, П2-29% и П3 – 21% и ее прибыль при смешанной стратегии в 41,27 млн руб. :

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 37, вывод ответа

## 

## **3.4. Задача о принятии решений в условиях неопределенности.**

Принятие решений в условиях неопределенности, когда статистические свойства случайных факторов либо неизвестны, либо не могут быть определены. В результате влияния неопределенных факторов каждая стратегия ЛПР оказывается связанной с множеством возможных исходов, вероятности которых либо неизвестны, либо вовсе не имеют смысла.

### **3.4.1. Описание входных данных**

Входными данными являются 3 возможных состояния природы, 3 стратегии компании А и матрица выигрышей размерностью 3х3.

matrica = [[**20**, **80**, **50**], [**78**, **10**, **25**], [**8**, **15**, **65**]]

priroda = [**a**, **b**, **c**]

strat\_A = [**1**, **2**, **3**]

### **3.4.2. Описание алгоритма решения**

Для принятия решения, у нас есть несколько критериев: критерий Вальда, критерий Сэвиджа и критерий Гурвица.

### **3.4.2.1. Критерий оптимизма**

Алгоритм сводится к тому, что необходимо узнать максимальный выигрыш по стратегиям погодных условий для каждой стратегии компании А. Затем мы находим максимальную прибыль из максимальных, чтобы узнать оптимальную из стратегий компании А.

optimizm = []

**for** i **in** matrica:

optimizm.append (max (i))

#Выигрыш А по критерию оптимизма:

max(optimizm))

#Стратегия для А:

strat\_A[optimizm.index(max(optimizm))])

### **3.4.2.2. Критерий пессимизма**

Алгоритм сводится к тому, что необходимо узнать минимальный выигрыш по стратегиям погодных условий для каждой стратегии компании А. Затем мы находим минимальную прибыль из минимальных.

pessimizm = []

**for** i **in** matrica:

pessimizm.append (min (i))

#Выигрыш А по критерию пессимизма:

min(pessimizm))

#Стратегия для А:

strat\_A[pessimizm.index(min(pessimizm))])

### **3.4.2.3. Критерий Вальда**

Алгоритм сводится к тому, что необходимо узнать минимальный выигрыш по стратегиям погодных условий для каждой стратегии компании А.

pessimizm = []

**for** i **in** matrica:

pessimizm.append (min (i))

Затем мы находим максимальную прибыль из минимальных, чтобы узнать оптимальную стратегию для компании А.

#Выигрыш А по критерию Вальда:

max(pessimizm))

#Стратегия для А:

strat\_A[pessimizm.index(max(pessimizm))])

### **3.4.2.4. Критерий Сэвиджа**

Критерий Сэвиджа стремится смягчить консерватизм минимаксного (максиминного) критерия путем замены матрицы платежей (выигрышей или погрешностей) v(ai , sj) матрице потерь r(ai , sj).

Определим матрицу рисков(потерь), согласно формуле 23, из элементов столбцов от первого до третьего соответственно.

matrica = [[matrica[j][i] **for** j **in** range(len(matrica))] **for** i **in** range (len(matrica[**0**]))]

savidj = []

savidj\_1 = []

res\_savidj = []

**for** i **in** matrica:

**for** j **in** i:

savidj.append (abs(j - max(i)))

Теперь по формуле 24 выбираем из максимальных значений 55, 70, 65 по строкам минимальное - 55.

**for** i **in** range(**0**, len(savidj), m):

savidj\_1.append (savidj[i : i + m])

savidj\_1 = [[savidj\_1[j][i] **for** j **in** range(len(savidj\_1))] **for** i **in** range (len(savidj\_1[**0**]))]

**for** i **in** savidj\_1:

res\_savidj.append (max(i))

#Величина минимальной недополученной прибыли по критерию Сэвиджа:

min(res\_savidj))

#Стратегия для А:

strat\_A[res\_savidj.index(min(res\_savidj))])

### **3.4.2.5. Критерий Гурвица**

mnmn = []

mxmx = []

svsv = []

alpha\_vector = []

alpha\_vector\_1 = []

vector\_otvetov = []

Критерий Гурвица охватывает ряд различных подходов к принятию решений – от наиболее оптимистичного до наиболее пессимистичного.

**for** i **in** matrica:

mnmn.append (min(i))

mxmx.append (max(i))

При параметре оптимизма равному 0 критерий Гурвица становится консервативным, так как его применение эквивалентно применению обычного минимаксного критерия. При параметре оптимизма равному 1 критерий Гурвица становится оптимистичным, т.к. рассчитывает на наилучшие из наилучших условий. Степень оптимизма (или пессимизма) определяется выбором величины параметра.

**for** i **in** range (**0**, **11**):

**for** j **in** matrica:

alpha\_vector.append (round(**0.1** \* i \* float(max(j)) + (**1** - **0.1** \* i) \* float(min(j)), **2**))

**for** i **in** range(**0**, len(alpha\_vector), m):

alpha\_vector\_1.append (alpha\_vector[i : i + m])

**for** i **in** alpha\_vector\_1:

vector\_otvetov.append (max(i))

Для визуализации склонности к риску по критерию Гурвица строим таблицу линейной свертки.

alpha\_vector\_1 = [[alpha\_vector\_1[j][i] **for** j **in** range(len(alpha\_vector\_1))] **for** i **in** range (len(alpha\_vector\_1[**0**]))]

#Линейная свертка склонности к риску по критерию Гурвица:

**for** i **in** alpha\_vector\_1:

**print** (\*i)

**print** (\*vector\_otvetov)

Критерий Гурвица (реализовано на Excel):

Для визуализации склонности к риску по критерию Гурвица строим таблицу линейной свертки и гистограмму, основанную на данной таблице:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 47, линейная свертка склонности к риску по Гурвицу

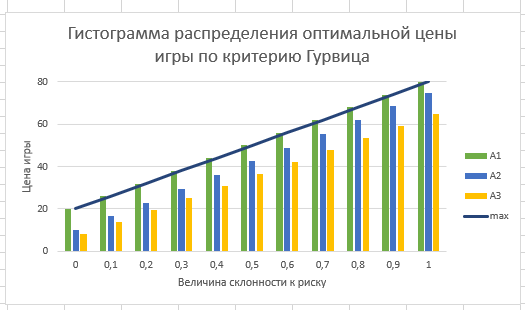


Рисунок 48, гистограмма линейной свертки

### **3.4.3. Описание выходных данных**

### **3.4.3.1. Критерий оптимизма**

Получаем оптимальную стратегию для компании А – производство шапок, благодаря которой компания получит прибыль в размере 80 млн. руб.

Выигрыш А по критерию оптимизма: 80

Стратегия для А: 1

### **3.4.3.2. Критерий Вальда**

Оптимальная стратегия для компании А – А1(Шапки) с выигрышем в размере 20 млн. руб.

Выигрыш А по критерию Вальда: 1

Стратегия для А: 20

### 

### **3.4.3.3. Критерий Сэвиджа**

Получаем оптимальную стратегию для компании А – производство шапок , благодаря которой компания получит прибыль в размере 55 млн. руб.

Величина минимальной недополученной прибыли по критерию Сэвиджа: 55

Стратегия для А: 1

### **3.4.3.4. Критерий Гурвица**

Оптимальная гарантированная стратегия для компании А – производство товаров «Шапки», при которой компания сможет заработать 12 млн руб.

Линейная свертка склонности к риску по критерию Гурвица:

20 26 32 38 44 50 56 62 68 74 80

10 16,5 23 29,5 36 42,5 49 55,5 62 68,5 75

8 13,7 19,4 25,1 30,8 36,5 42,2 47,9 53,6 59,3 65

vector\_otvetov:

20 26 32 38 44 50 56 62 68 74 80

# **4. Варианты использования системы**

Для пользователя предусмотрены два варианта использования системы: средствами Python и средствами MS Excel. Пользователь может выбрать любой удобный для него вариант.

## **4.1. ВИ\_1\_средствами Python.**

1. Пользователь для решения задачи должен обратиться к методу №1 (см. раздел 3 метод 1) и методу №2 (см. раздел 3 метод 2), представленным в данной документации.
2. Приведенный в методе код пользователь должен использовать в имеющейся к среде разработки для Python.
3. Пользователь должен запустить код для решения задачи.
   1. Пользователь запускает код для метода №1 (см. раздел 3 метод 1).
      1. Пользователю необходимо ввести следующие входные данные:
         1. Ввести способ ввода данных:

* «ручной»
* «рандом»
* «файл»
  + - 1. Ввести количество стратегий игрока А: одно целое число
      2. Ввести названия стратегий для А:
* через пробел
* в строку
  + - 1. Ввести количество стратегий игрока В: одно целое число
      2. Ввести названия стратегий для В через пробел в строку:
* через пробел
* в строку
  + - 1. Ввести матрицу (строки - А, столбцы - В):
* действительные числа через пробел
* каждую строку через Enter
  1. Пользователь запускает код метода №2 (см. раздел 3 метод 2).
     1. Пользователю необходимо ввести следующие входные данные.
        1. Ввести способ ввода данных:
* «ручной»
* «рандом»
* «файл»
  + - 1. Ввести количество стратегий игрока А: одно целое число
      2. Ввести названия стратегий для А:
* через пробел
* в строку
  + - 1. Ввести количество стратегий игрока В: одно целое число
      2. Ввести названия стратегий для В:
* через пробел
* в строку
  + - 1. Ввести первую матрицу (строки - А, столбцы - В)
* действительные числа через пробел
* каждую строку через Enter
  + - 1. Ввести вторую матрицу (строки - А, столбцы - В)
* действительные числа через пробел
* каждую строку через Enter

1. Программа должна считать данные.
2. После обработки скрипта пользователю должен выводится результат в виде:
   1. Для метода №1 (см. раздел 3 метод 1). программа должна вывести на экран:

* Выигрыш А при чистой оптимальной стратегии: «число»
* Стратегия для А: «название стратегии»
* Выигрыш B при чистой оптимальной стратегии: «число»
* Стратегия для B: «название стратегии»
  1. Для метода №2 (см. раздел 3 метод 2). программа должна вывести на экран:
* Количество равновесий по Нэшу: «число»
* Выигрыш А при чистой стратегии: «число»
* Стратегия для А: «название стратегии»
* Выигрыш B при чистой стратегии: «число»
* Стратегия для B: «название стратегии»
* Суммарный выигрыш: «число»

## **4.2. ВИ\_2\_средствами MS Excel.**

1. Пользователь для решения задачи обращается к разработанным методам №3 (см. раздел 3 метод 3) и методу №4 (см. раздел 3 метод 4), представленным в данной документации.
2. Пользователь открывает приложение MS Excel.
3. Пользователь вручную следует разработанному алгоритму.
   1. Пользователь решает задачу на принятие решения в условиях риска.
      1. Пользователь следует инструкциям, описанным в методе №3 (см. раздел 3 метод 3) данной документации.
   2. Пользователь решает задачу на принятие решения в условиях неопределенности.
      1. Пользователь следует инструкциям, описанным в методе №4 (см. раздел 3 метод 4) данной документации.
4. Пользователь должен получить ответ.

# **5. Архитектура решения**

## **5.1. Функции считывания информации**

Для Антагонистической и биматричной задач функции считывания информации одинаковы и заключаются в считывания информации, вводимой пользователем с клавиатуры, либо считывание файла

**if** way == 'ручной':

Осуществляется ввод с клавиатуры пользователем:

n = int(input('Введите количество стратегий игрока А: '))

**print** ()

strat\_A = input('Введите названия стратегий для А через пробел в строку: ').split()

**print** ()

Код проверяет совпадает ли число элементов, введенные пользователем, от числа стратегий, которые он задает изначально:

**if** len(strat\_A) != n:

**print** ('Данные введены неверно!')

exit ()

m = int(input('Введите количество стратегий игрока B: '))

**print** ()

strat\_B = input('Введите названия стратегий для B через пробел в строку: ').split()

**print** ()

Аналогично для игрока B:

**if** len(strat\_B) != m:

**print** ('Данные введены неверно!')

exit ()

**print** ('Введите первую матрицу (строки – А, столбцы – В)')

matrica1 = [[float(j) **for** j **in** input().split()] **for** i **in** range(n)]

**print** ()

**print** ('Введите вторую матрицу (строки – А, столбцы – В)')

matrica2 = [[float(j) **for** j **in** input().split()] **for** i **in** range(n)]

**print** ()

**elif** way == 'файл':

Считываем исходные данные из файла:

strat\_A = []

strat\_B = []

matrica1 = []

matrica2 = []

name = input('Сохраните ваш файл в папку с этой программой и введите его имя без кавычек: ')

file = open(name, “r”)

**print** ()

ff = [line.strip() **for** line **in** file]

razdel = ff.index(‘’)

matrica1 = ff[**0** : razdel]

matrica2 = ff[razdel + **1** : len(ff)]

Составляем матрицу:

**for** i **in** range(len(matrica1)):

matrica1[i] = matrica1[i].split()

**for** j **in** range(len(matrica1[i])):

matrica1[i][j] = float (matrica1[i][j])

**for** i **in** range(len(matrica2)):

matrica2[i] = matrica2[i].split()

**for** j **in** range(len(matrica2[i])):

matrica2[i][j] = float (matrica2[i][j])

**for** line **in** file:

matrica.append([float(x) **for** x **in** line.split()])

m = len(matrica1[**0**])

n = len(matrica1)

m\_test = len(matrica2[**0**])

n\_test = len(matrica2)

Если размер матриц не сходится, то код выводит предупреждение:

**if** m != m\_test **or** n != n\_test:

**print** ('В файле содержатся матрицы разного размера!')

exit ()

Для задач в условиях риска и неопределенности функции считывания информации одинаковы и заключаются в считывания информации, введенной изначально в код.

matrica = [[**20**, **80**, **50**], [**78**, **10**, **25**], [**8**, **15**, **65**]]

priroda = [**a**, **b**, **c**]

strat\_A = [**1**, **2**, **3**]

Для задачи в условиях неопределенности также вводится ряд списков, которые впоследствии будут пополняться полученными данными.

mnmn = []

mxmx = []

svsv = []

alpha\_vector = []

alpha\_vector\_1 = []

vector\_otvetov = []

## **5.2. Функции обработки информации**

Обработка информации осуществляется через циклы в обоих случаях:

1. Антагонистические:

Для решения данной задачи нам нужно применить метод МИНИМАКС и МАКСИМИН, для этого мы выполняем поиск минимума по строкам.

**for** i **in** matrica:

**if** len(i) != m:

**print** ('Данные введены неверно!')

exit ()

**else**:

vig\_A.append (min(i))

Для дальнейшего решения нам нужно транспонировать матрицу и уже применить метод МАКСИМИНА (т.е найти максимальное по строкам, которые до транспонирования были столбцами)

matrica = [[matrica[j][i] **for** j **in** range(len(matrica))] **for** i **in** range (len(matrica[**0**]))]

**for** i **in** matrica:

vig\_B.append (max(i))

1. Биматричные

Осуществляется поиск максимальных значений и минимальных значений в матрицах.

**for** i **in** matrica1:

vig\_A.append (max(i))

**for** i **in** matrica2:

vig\_B.append (max(i))

Выполняется поиск равновесий по Нэшу:

k = **0**

**for** i **in** range (len(vig\_A)):

**if** vig\_A[i] == min(vig\_A) **and** vig\_B[i] == min(vig\_B):

k = k + **1**

Через функцию print код выводит количество равновесий по Нэшу

**print** ('Количество равновесий по Нэшу:', k)

**for** i **in** range(len(matrica1)):

matrica1[i] = matrica1[i].split()

**for** j **in** range(len(matrica1[i])):

matrica1[i][j] = float (matrica1[i][j])

**for** i **in** range(len(matrica2)):

matrica2[i] = matrica2[i].split()

**for** j **in** range(len(matrica2[i])):

matrica2[i][j] = float (matrica2[i][j])

**for** line **in** file:

matrica.append([float(x) **for** x **in** line.split()])

1. Задачи в условиях риска.

Обработка информации осуществляется в данной задаче также осуществляется через циклы.

**for** i **in** range(len(matrica)):

k = -**1**

**for** j **in** range(len(matrica[i])):

k = k + **1**

matrica\_baes.append(matrica[i][j] \* float(priroda[k]))

matrica\_laplas.append(matrica[i][j] \* (**1** / m))

Осуществляется расчет равновероятной вероятность стратегий природы:

**for** i **in** range(len(matrica)):

k = -**1**

**for** j **in** range(len(matrica[i])):

k = k + **1**

matrica\_baes.append(matrica[i][j] \* float(priroda[k]))

matrica\_laplas.append(matrica[i][j] \* (**1** / m))

Вычисление средневзвешенного значения:

**for** i **in** range(m - **1**, len(matrica\_baes) + **1**, m):

s\_baes.append (sum (matrica\_baes[i - (m - **1**) : i + **1**]))

**for** i **in** range(m - **1**, len(matrica\_laplas) + **1**, m):

s\_laplas.append (sum (matrica\_laplas[i - (m - **1**) : i + **1**]))

Для решения методом Гермейера матрица решений дополняется столбцом, содержащим в каждой строке наименьшее произведение имеющегося в ней результата на вероятность соответствующего состояния П.

Получение матрицу Геймейера, умножение на вероятность наступления состояния природы:

#Гермейер

priroda\_gey = [priroda] \* m

matrica\_gey = []

**for** i **in** range(len(matrica)):

**for** j **in** range(len(matrica[i])):

matrica\_gey.append (matrica[i][j] \* priroda\_gey[i][j])

**for** i **in** range(**0**, len(matrica\_gey), m):

matrica\_gey\_1.append (matrica\_gey[i : i + m])

Осуществляется выбор вариантов с наибольшим значением в строках аij этого столбца.

1. Задача в условиях неопределенности.

Обработка информации осуществляется в данной задаче осуществляется через циклы.

Осуществляется расчет минимальных/максимальных значений в строках:

**for** i **in** matrica:

pessimizm.append (min (i))

optimizm.append (max (i))

Для определения оптимистичного/пессимистичного решения, а также и построения линейной свертки используется следующие циклы:

**for** i **in** range (**0**, **11**):

**for** j **in** matrica:

alpha\_vector.append (round(**0.1** \* i \* float(max(j)) + (**1** - **0.1** \* i) \* float(min(j)), **2**))

**for** i **in** range(**0**, len(alpha\_vector), m):

alpha\_vector\_1.append (alpha\_vector[i : i + m])

**for** i **in** alpha\_vector\_1:

vector\_otvetov.append (max(i))

В решении по критерию Сэвиджа используются циклы для определения матрицы рисков из элементов столбцов от первого до третьего соответственно:

matrica = [[matrica[j][i] **for** j **in** range(len(matrica))] **for** i **in** range (len(matrica[**0**]))]

**for** i **in** matrica:

**for** j **in** i:

savidj.append (abs(j - max(i)))

Выбирается из максимальных значений по строкам минимальное:

**for** i **in** range(**0**, len(savidj), m):

savidj\_1.append (savidj[i : i + m])

savidj\_1 = [[savidj\_1[j][i] **for** j **in** range(len(savidj\_1))] **for** i **in** range (len(savidj\_1[**0**]))]

**for** i **in** savidj\_1:

res\_savidj.append (max(i))

## **5.3. Функции вывода информации**

Аналогично функциям ввода, функции вывода также одинаковы и осуществляются через функцию print:

1. Антагонистические

**print** ('Выигрыш А при чистой оптимальной стратегии:', max(vig\_A)) #максимин

Здесь выводится максимальный элемент в матрице первого игрока

**print** ('Стратегия для А:', strat\_A[vig\_A.index(max(vig\_A))])

Далее выводится название стратегии, которая соответствует значению, которое выведено было ранее

**print** ()

Аналогичные действия происходят и для игрока Б:

**print** ('Выигрыш B при чистой оптимальной стратегии:', min(vig\_B)) #минимакс

**print** ('Стратегия для B:', strat\_B[vig\_B.index(min(vig\_B))])

1. Биматричные

Аналогичным образом, как и для антагонистических игр идет вывод результатов через print:

**print** ('Выигрыш А при чистой опмтимальной стратегии:', max(vig\_A))

**print** (‘Стратегия для А:’, strat\_B[vig\_A.index(max(vig\_A))])

**print** ()

**print** ('Выигрыш B при чистой оптимальной стратегии:', max(vig\_B))

**print** (‘Стратегия для B:’, strat\_A[vig\_B.index(max(vig\_B))])

**print** ()

Находим суммарный выигрыш:

**print** (‘Суммарный выигрыш:’, max(vig\_A) + max(vig\_B))

**print** ()

В случае, если равновесий по Нэшу не было найдено, то нужно смешивать стратегии и код выводит предупреждение.

**else**:

**print** ('Стратегии нужно смешивать!')

3. Задачи с условиях риска

**print** ('Выигрыш А по критерию Лапласа:', max(s\_laplas))

**print** ('Стратегия для А:', strat\_A[s\_laplas.index(max(s\_laplas))])

**print** ()

**print** ('Выигрыш А по критерию Баеса:', max(s\_baes))

**print** ('Стратегия для А:', strat\_A[s\_baes.index(max(s\_baes))])

**print** ()

**print** ('Выигрыш А при оптимальной чистой стратегии по критерию Гермейера:', min(matrica[gey\_mn.index(max(gey\_mn))]))

**print** ('Стратегия для А:', strat\_A[gey\_mn.index(max(gey\_mn))])

4. Задачи с условиях неопределенности

**print** ('Выигрыш А по критерию пессимизма:', min(pessimizm))

**print** ('Стратегия для А:', strat\_A[pessimizm.index(min(pessimizm))])

**print** ()

**print** ('Выигрыш А по критерию оптимизма:', max(optimizm))

**print** ('Стратегия для А:', strat\_A[optimizm.index(max(optimizm))])

**print** ()

**print** ('Выигрыш А по критерию Вальда:', max(pessimizm))

**print** ('Стратегия для А:', strat\_A[pessimizm.index(max(pessimizm))])

**print** ()

**print** ('Линейная свертка склонности к риску по критерию Гурвица')

**for** i **in** alpha\_vector\_1:

**print** (\*i)

**print** ()

**print** (\*vector\_otvetov)

**print** ()

**print** ('Величина минимальной недополученной прибыли по критерию Сэвиджа:', min(res\_savidj))

**print** ('Стратегия для А:', strat\_A[res\_savidj.index(min(res\_savidj))])

**print** ()

# **6. Тестирование**

Входные данные для задач:

1. Физическая модель
2. Платежная матрица для игрока А:

|  |  |  |  |
| --- | --- | --- | --- |
| 70 | 12 | 13 | 20 |
| 25 | 80 | 41 | 55 |
| 17 | 90 | 14 | 66 |

Для игры с природой:

Критерий доверия: 0,4

Вероятности={0,25; 0,25; 0,25;0,25}

Платежная матрица для биматричной игры игрока B:

|  |  |  |  |
| --- | --- | --- | --- |
| 90 | 88 | 10 | 5 |
| 80 | 41 | 69 | 20 |
| 11 | 32 | 70 | 50 |

Критерий доверия: 0,4

1. Платежная матрица для игрока А:

|  |  |  |  |
| --- | --- | --- | --- |
| 70 | 12 | 13 | 20 |
| 25 | 80 | 41 | 55 |
| 17 | 90 | 14 | 66 |
| 60 | 55 | 75 | 90 |

Для игры с природой:

Критерий доверия: 0,4

Вероятности={0,25; 0,25; 0,25;0,25}

Платежная матрица для биматричной игры игрока B:

|  |  |  |  |
| --- | --- | --- | --- |
| 90 | 88 | 10 | 5 |
| 80 | 41 | 69 | 20 |
| 11 | 32 | 70 | 50 |
| 14 | 58 | 32 | 10 |

1. Платежная матрица для игрока А:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 859 | 629 | 337 | 417 | 265 |
| 819 | 946 | 762 | 37 | 712 |
| 690 | 604 | 263 | 324 | 925 |
| 396 | 192 | 903 | 164 | 859 |
| 650 | 748 | 325 | 972 | 375 |
| 135 | 788 | 782 | 363 | 536 |
| 192 | 667 | 566 | 327 | 300 |

Для игры с природой:

Критерий доверия: 0,4

Вероятности={0,25; 0,25; 0,25;0,25}

Платежная матрица для биматричной игры игрока B:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100 | 55 | 845 | 142 | 152 |
| 95 | 123 | 781 | 100 | 74 |
| 95 | 562 | 123 | 451 | 214 |
| 145 | 256 | 321 | 152 | 324 |
| 145 | 256 | 325 | 654 | 12 |
| 698 | 384 | 286 | 176 | 279 |
| 439 | 268 | 259 | 357 | 159 |

1. Платежная матрица для игрока А:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 14 | 5 | 9 | 1 |
| 5 | 7 | 9 | 14 | 4 |
| 14 | 11 | 78 | 9 | 46 |
| 14 | 6 | 8 | 2 | 3 |
| 5 | 6 | 1 | 9 | 8 |
| 2 | 7 | 8 | 7 | 9 |
| 2 | 1 | 9 | 4 | 3 |
| 7 | 9 | 4 | 5 | 6 |

Для игры с природой:

Критерий доверия: 0,4

Вероятности={0,25; 0,25; 0,25;0,25}

Платежная матрица для биматричной игры игрока B:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 5 | 9 | 8 | 6 |
| 14 | 77 | 6 | 1 | 2 |
| 1 | 85 | 11 | 12 | 12 |
| 7 | 8 | 6 | 15 | 10 |
| 10 | 20 | 45 | 12 | 1 |
| 74 | 5 | 6 | 1 | 2 |
| 7 | 9 | 14 | 32 | 5 |
| 1 | 7 | 5 | 8 | 9 |

*Таблица 10. Тестирование*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Антагонистическая игра** | **Биматричная игра** | **Игра с природой в условиях риска** | | | **Игра с природой в условиях неопределенности** | | |
| **Лаплас** | **Байес** | **Гермейер** | **Вальд** | **Сэвидж** | **Гурвиц** |
| **1** | **Оптимальная стратегия** | **А3** | **А2**  **В3** | **А1** | **A1** | **A1** | **A1** | **A1** | **A1** |
| **Цена игры** | **40** | **80**  **85** | **50** | **47** | **20** | **20** | **55** | **12** |
| **2** | **Оптимальная стратегия** | **А2** | **А2**  **В1** | **A2** | **A2** | **A2** | **A2** | **A2** | **A3** |
| **Цена игры** | **25** | **90**  **90** | **50.25** | **50.25** | **100** | **25** | **45** | **59.6** |
| **3** | **Оптимальная стратегия** | **А4** | **А2**  **В1** | **А4** | **А4** | **А4** | **А4** | **А4** | **А4** |
| **Цена игры** | **55** | **90**  **90** | **70** | **70** | **220** | **76** | **35** | **81.49** |
| **4** | **Оптимальная стратегия** | **А5** | **А4**  **В1** | **А2** | **А2** | **А5** | **А5** | **А5** | **А5** |
| **Цена игры** | **325** | **972**  **845** | **655.2** | **819** | **1300** | **325** | **578** | **713.2** |
| **5** | **Оптимальная стратегия** | **А3** | **А3**  **В3** | **А3** | **А3** | **А3** | **А3** | **А3** | **А3** |
| **Цена игры** | **9** | **78**  **85** | **31.6** | **31.6** | **45** | **9** | **5** | **50.4** |

# **7. Заключение**

В данной проектной документации решалась «игровая» задача. В зависимости от изменения вида игры (антагонистическая игра, биматричная, игра с природой в условиях риска или в условиях неопределенности) видоизменялась физическая модель. Была разработана математическая модель для всех типов задач, на основании которой затем были разработаны программы средствами Python для нахождения результата целевой функции с учетом алгоритмов решения антагонистической и биматричной задач, далее были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат. Задачи на игры с природой были решены по приведенной физической и математическим моделям средствами MS Excel.

Проведённая работа с различными типами задач показала и определила области их эффективного действия: для игр и задач с двумя игроками, у которых цели противоположны – метод для антагонистических задач, для игроков с не противоположными целями – метод для биматричных игр. В случае, когда один из игроков «разумный», а другой действует совершенно хаотично, то – методы для игр с природой: когда известны вероятности возможных состояний – метод для игр в условиях риска, иначе – метод для игр в условиях неопределенности.

# **ПРОГНОЗИРОВАНИЕ**

# **1. Физическая модель задачи**

За последние 7 месяцев сеть продуктовых магазинов «Ласточка» получила прибыль в размере 74 млн. руб. Необходимо составить зависимость прибыли по годам деятельности компании. Прибыль (в млн. руб.) по годам представлена в следующей таблице:

Таблица 1. Исходные данные

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Год | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| *Прибыль* | *10* | *11,5* | *8* | *13* | *10,5* | *6* | *15* |

# **2. Математическая модель задачи**

Существует некоторая функциональная зависимость между величинами, которые даны в виде набора точек или таблицы:

Таблица 2. Таблица исходных данных

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | *…* | *…* |  |
|  |  |  | *…* | *…* |  |

Для определения функциональной зависимости f(x) будет использоваться такой критерий точности, как критерий наименьших квадратов. Величина R, которая равна сумме квадратов разности между значениями многочлена y от заданной функции f(x) для всех точек х0, х1, х2,…, хn:

Наилучшие параметры a0, a1, a2, …, am находятся из условия минимума функции:

Если рассматривать в качестве f(x) следующий многочлен:

Обращаемая в минимум функция примет вид:

(4)

Минимум функции будет находиться путем приравнивания к нулю частных производных переменных – параметров a0, a1, a2, …, am. То есть условием минимума будет являться:

А если обозначим две составляющие этого выражения следующим образом:

Тогда Матрица Грамма для определения искомых параметров будет иметь вид:

(8)

* 1. Линейная аппроксимация.

Это частный случай аппроксимации функции, где m=1. Тогда получаем:

Сама система уравнений примет следующий вид:

(11)

Искомая функция:

* 1. Квадратичная аппроксимация.

Это частный случай аппроксимации функции, где m=2 и многочлен примет вид:

Далее так же рассчитываем по формулам 10 b0, b1, ci, где i=0, 1, 2. Помимо этого необходимо рассчитать:

Матрица системы уравнений, с помощью которой найдем искомые коэффициенты a0, a1, a2:

(15)

**3.1. Метод наименьших квадратов**

Метод наименьших квадратов (МНК) — математический метод, применяемый для решения различных задач, основанный на минимизации суммы квадратов отклонений некоторых функций от искомых переменных. Он может использоваться для «решения» переопределенных систем уравнений (когда количество уравнений превышает количество неизвестных), для поиска решения в случае обычных (не переопределенных) нелинейных систем уравнений, для аппроксимации точечных значений некоторой функции. МНК является одним из базовых методов регрессионного анализа для оценки неизвестных параметров регрессионных моделей по выборочным данным.

### **3.1.1. Описание входных данных**

Входными данными для решения являются массив точек х и у, также возможно проверить работу кода уже с готовыми данными. Для удаления выбросов нужно ввести процент удаления точек

### **3.1.2. Описание алгоритма решения**

Решение задачи непосредственно зависит от выбора задачи для решения (линейная, квадратичная, удаление выбросов).

**Линейная аппроксимация:**

1. Cоставляем массивы из заданных точек.
2. Находим значение аппроксимирующей функции в точках х и у.
3. Выполнить необходимые вычисления для дальнейшего решения: средних значений по х и у, среднее произведение х и у, среднее по х^2
4. Выполнить поиск коэффициентов для функции линейной аппроксимации (а и б)
5. Составляем функцию по формуле: у=а`х+b
6. Вычисляем дисперсию

**Квадратичная аппроксимация:**

1. Пункты 1-2 из линейной аппроксимации
2. К вычислениям из п.3 добавить: х в 3 и 4 степенях, произведения х и у, произведения х^2\*y.
3. Согласно формулам для квадратичной аппроксимации, составляем систему уравнений
4. Решение системы уравнений матричным способом
5. Нахождение коэффициентов а, b, с
6. Составление функции по формуле: y= a`x^2+b`x+c
7. Вычисляем дисперсию

**Удаление выбросов:**

1. Пункты 1-2 линейной аппроксимации
2. Вычисление дисперсии до удаления выбросов
3. Удаление выбросов
4. Вычисление дисперсии после удаление выбросов и падения дисперсии
5. Вывести удаленные точки

### **3.1.3. Описание выходных данных**

**Линейная аппроксимация:**

А) массив точек;

Б) вид функции;

В) дисперсия.

**Квадратичная аппроксимация:**

А) массив точек;

Б) вид функции;

В) дисперсия

**Удаление выбросов:**

А) массив точек после исключения выбросов;

Б) падение дисперсии;

В) удаленные точки.

## **3.2. Приемо-сдаточное тестирование.**

В данном случае тестирование проводится путем визуализации графиков функций полученных в 3.1 с дополнительными построениями

### **3.2.1. Описание входных данных**

Входными данными для обоих графиков (линейной и квадратичной аппроксимации) будут данные, полученные в 3.1

### **3.2.2. Описание алгоритма решения**

Алгоритм решения одинаков для обоих графиков. На плоскости наносим точки, задаем линию тренда (функция, полученная при решении задания непосредственно), строим доверительные интервалы (68%, 95%, 99%), притом как выше, так и ниже линии тренда (симметричные друг другу)

### **3.2.3. Описание выходных данных**

Выходными данными будут являться графики:

1) Линейная аппроксимация

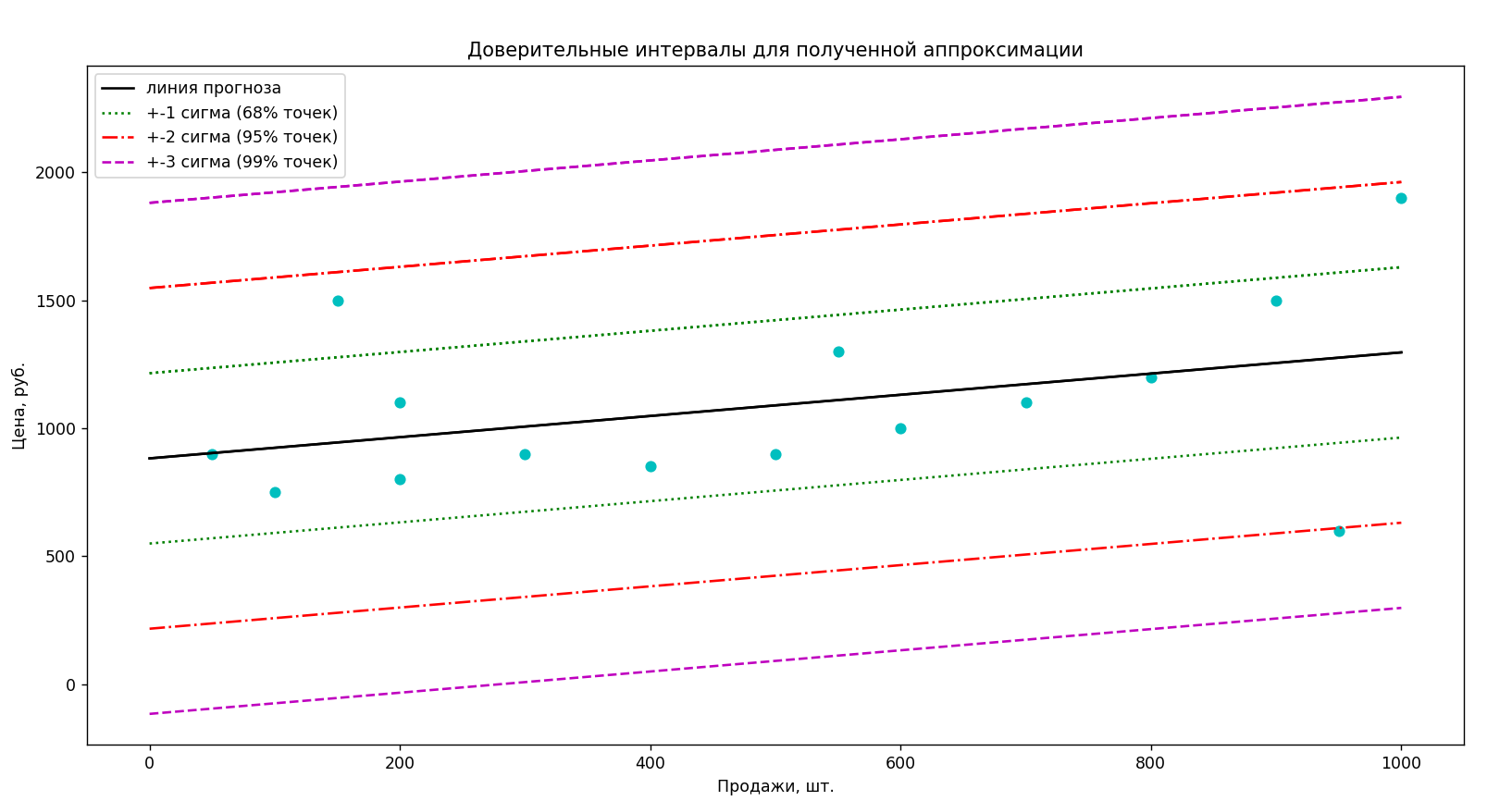


Рисунок 1. Доверительные интервалы для линейной аппроксимации

2) Квадратичная аппроксимация

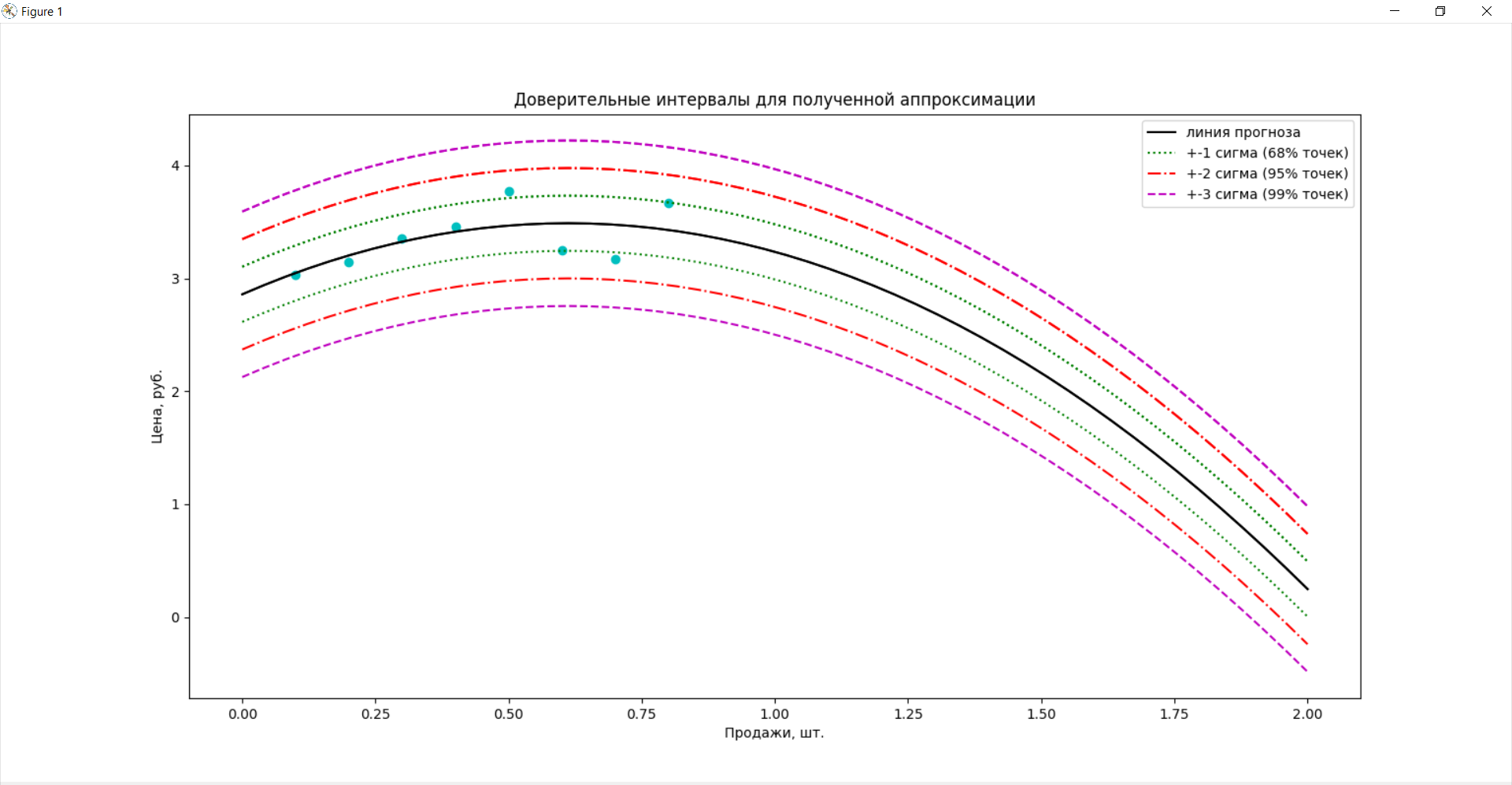


Рисунок 2. Доверительные интервалы для квадратичной аппроксимации

3) Полиномиальная регрессия

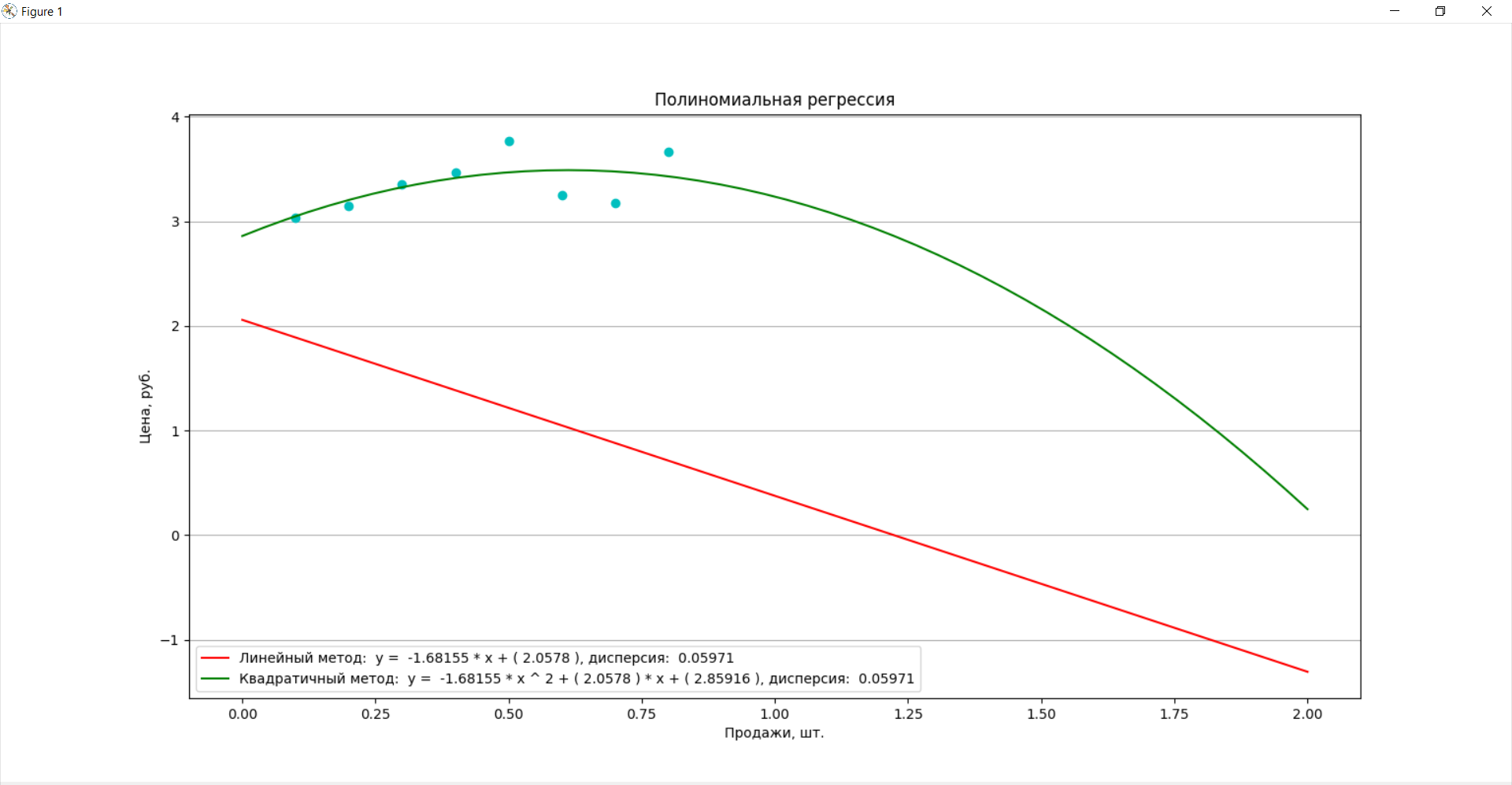


Рисунок 3. График полиномиальной регрессии

# **4. Варианты использования системы**

Для пользователя предусмотрен следующий вариант использования системы с использованием средств Python.

1. Пользователь должен запустить среду разработки для Python.
2. Пользователь должен запустить код для решения задачи.
   1. Пользователю необходимо ввести следующие входные данные:
      1. Ввести формат ввода:

* «ручной»
* «готовые данные».
  + - 1. Если пользователь выбрал «ручной», необходимо ввести значения X:
* после каждого значения пробел
* вводить в строку.
  + - 1. Ввести значения Y:
* после каждого значения пробел
* вводить в строку.
  + 1. Ввести задачу для решения:
       - * «линейная»
         * «квадратичная»
         * «выбросы».
       1. Если пользователь выбрал «выбросы», необходимо ввести процент точек:

число без знака %.

1. Программа выполняет алгоритм.
2. Пользователю должен выводиться результат в следующем виде:
   1. Для задач «линейная» и «квадратичная»:

* массив точек в формате [xi, yi, fi]
* строка с аппроксимирующей функцией
* величина дисперсии
  1. Для задачи «выбросы»:
     + - массив точек с исключенными выбросами
       - величина дисперсии падения дисперсии выборки
       - список удаленных точек и величина изменения дисперсии– массив в формате [xi,yi,Di].

# **5. Архитектура решения**

Библиотеки, необходимые для решения поставленных задач: numpy-для работы с матрицами и matplotlib.pyplot-для прорисовки графиков

**from** **statistics** **import** mean

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

## **5.1. Функции считывания информации**

Функции считывания информации в коде реализованы через ввод пользователем необходимых данных для решения конкретной задачи (выбор методов ввода, решения задачи, ввода процента точек (при выбросах)

## **5.2. Функции обработки информации**

**def** **disp**(y):

Данная функция вычисляет дисперсию, необходимую для решения задач

**def** **linaprox**(x, y):

Функция для линейной аппроксимации. Вычисляет коэффициенты линейной регрессии и составляет функцию, которая понадобиться для построения графика.

**def** **kvadaprox**(x, y):

Функция для квадратичной аппроксимации. Вычисляет все необходимые параметры для решения. Составляет матрицу из системы уравнений и составляет обратную матрицу (с использованием библиотеки numpy). Составляет функцию для дальнейшего построения графика.

**def** **chistka**(x, y, proc):

Функция для выполнения чистки от выбросов. Вычисляет дисперсию, далее выполняет поиск точек (выбросов) и удаляет их из массива, пересчитывает дисперсию и находит ее падение. Выводит точки, которые были исключены из массива точек.

**def** **plot2**(a, b):

Функция для построения графика линейной аппроксимации. Строит доверительные интервалы, относительно линии тренда, полученной при ЛА, наносит точки на график.

**def** **plot**(a, b):

Функция для построения графика квадратичной аппроксимации. На одном графике показывает функцию линейного и квадратичного метода аппроксимации.

## **5.3. Функции вывода информации**

Вывод производится во всех случаях через print

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4. Функции вывода для кода

# **6. Тестирование**

# **6.1. Тестирование линейной и квадратичной аппроксимаций.**

Далее приведена таблица результатов решения задач методами линейной и квадратичной аппроксимациями.

Входные данные для задач:

*Таблица 3. Результаты тестирования программы*

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Линейная** | **Квадратичная** |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 10 | 11,5 | 8 | 13 | 10,5 | 6 | 15 | | а) Массив:  [1.0, 10.0, 9.875];[2.0, 11.5, 10.10714];[3.0, 8.0, 10.33929];[4.0, 13.0, 10.57143];[5.0, 10.5, 10.80357];[6.0, 6.0, 11.03571];[7.0, 15.0, 11.26786]  б) Функция:  y = 0.23214 \* x + ( 9.64286 )  в) Дисперсия:  7.7449 | а) Массив:  [1.0, 10.0, 10.91667];[2.0, 11.5, 10.10714];[3.0, 8.0, 9.71429];[4.0, 13.0, 9.7381];[5.0, 10.5, 10.17857];[6.0, 6.0, 11.03571];[7.0, 15.0, 12.30952]  б) Функция:  y = 0.20833 \* x ^ 2 + ( -1.43452 ) \* x + ( 12.14286 )  в) Дисперсия:  7.7449 |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 12 | 28 | 30 | 49 | 56 | 65 | 73 | | 20 | 15 | 18 | 43 | 95 | 64 | 15 | | а) Массив:  [12.0, 20.0, 18.32643];[28.0, 15.0, 28.22791];[30.0, 18.0, 29.4656];[49.0, 43.0, 41.22361];[56.0, 95.0, 45.55551];[65.0, 64.0, 51.1251];[73.0, 15.0, 56.07584]  б) Функция:  y = 0.61884 \* x + ( 10.90031 )  в) Дисперсия:  818.53061 | а) Массив:  [12.0, 20.0, 1.93388];[28.0, 15.0, 34.26418];[30.0, 18.0, 37.20556];[49.0, 43.0, 52.95787];[56.0, 95.0, 53.20067];[65.0, 64.0, 49.11319];[73.0, 15.0, 41.32464]  б) Функция:  y = -0.03055 \* x ^ 2 + ( 3.24277 ) \* x + ( -32.57971 )  в) Дисперсия:  818.53061 |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 123 | 288 | 350 | 469 | 156 | 465 | | 209 | 135 | 218 | 343 | 495 | 764 | | а) Массив:  [123.0, 209.0, 248.35019];[288.0, 135.0, 348.25433];[350.0, 218.0, 385.79407];[469.0, 343.0, 457.84615];[156.0, 495.0, 268.33102];[465.0, 764.0, 455.42423]  б) Функция:  y = 0.60548 \* x + ( 173.87619 )  в) Дисперсия:  45886.22222 | а) Массив:  [123.0, 209.0, 359.75511];[288.0, 135.0, 189.9907];[350.0, 218.0, 248.06899];[469.0, 343.0, 546.05076];[156.0, 495.0, 288.08307];[465.0, 764.0, 532.05137]  б) Функция:  y = 0.00866 \* x ^ 2 + ( -4.58778 ) \* x + ( 793.04768 )  в) Дисперсия:  45886.22222 |
| Х{14.5;8.5;9.1;7.2;6.7;8.7;5.2;6;7.7}  У{7.8;9.1;3;5.7;5;7.9;8.8;8.3;4} | а) Массив:  [14.5, 7.8, 6.62053];[8.5, 9.1, 6.62214];[9.1, 3.0, 6.62198];[7.2, 5.7, 6.62248];[6.7, 5.0, 6.62262];[8.7, 7.9, 6.62208];[5.2, 8.8, 6.62302];[6.0, 8.3, 6.6228];[7.7, 4.0, 6.62235]  б) Функция:  y = -0.00027 \* x + ( 6.62441 )  в) Дисперсия:  4.46617 | а) Массив:  [14.5, 7.8, 7.88077];[8.5, 9.1, 5.70835];[9.1, 3.0, 5.53282];[7.2, 5.7, 6.38808];[6.7, 5.0, 6.75861];[8.7, 7.9, 5.64014];[5.2, 8.8, 8.23389];[6.0, 8.3, 7.37919];[7.7, 4.0, 6.07815]  б) Функция:  y = 0.12122 \* x ^ 2 + ( -2.42609 ) \* x + ( 17.57167 )  в) Дисперсия:  4.46617 |
| Х{1;5;7;8;9;5;4;6;8;4;5;8;5;6;2;4;5}  У{7;8;9;6;7;4;5;6;5;4;2;4;1;2;3;7;8;9} | а) Массив:  [15789.0, 7896.0, 8858.15688];[5468.0, 7456.0, 4546.15126];[458.0, 542.0, 2453.02578];[562.0, 4123.0, 2496.47589];[456.0, 789.0, 2452.1902]  б) Функция:  y = 0.41779 \* x + ( 2261.67818 )  в) Дисперсия:  9855247.76 | а) Массив:  [15789.0, 7896.0, 7888.78217];[5468.0, 7456.0, 7524.54473];[458.0, 542.0, 1746.18358];[562.0, 4123.0, 1903.34372];[456.0, 789.0, 1743.14581]  б) Функция:  y = -7e-05 \* x ^ 2 + ( 1.58554 ) \* x + ( 1035.30286 )  в) Дисперсия:  9855247.76 |

На основе данной таблицы можно сделать вывод, что результаты линейной и квадратичной аппроксимации являются приближенно равными, в том числе значение аппроксимирующей функции в точках.

# 

# **6.2. Тестирование задач по удалению выбросов.**

# **6.2.1. Задача 6.**

Входные данные для задачи №6:

* Х{400;500;600;700;800;900;1000;150;50;100;200;300;550;200;950;300;550;400;450;950;100;3000;150;200;750;300;350;450;700;1100}
* У{850;900;1000;1100;1200;1500;1900;1500;900;750;800;900;1300;1100;600;1800;3000;700;500;2700;1600;1700;1150;850;1250;6000;2400;3600;4800;1750}
* Процент точек для выбросов: 10%

Выходные данные для задачи №6:

а) Массив после исключения:

[400.0, 850.0];[500.0, 900.0];[600.0, 1000.0];[700.0, 1100.0];[800.0, 1200.0];[900.0, 1500.0];[1000.0, 1900.0];[150.0, 1500.0];[50.0, 900.0];[100.0, 750.0];[200.0, 800.0];[300.0, 900.0];[550.0, 1300.0];[200.0, 1100.0];[300.0, 1800.0];[550.0, 3000.0];[950.0, 2700.0];[100.0, 1600.0];[3000.0, 1700.0];[150.0, 1150.0];[200.0, 850.0];[750.0, 1250.0];[300.0, 6000.0];[350.0, 2400.0];[450.0, 3600.0];[700.0, 4800.0];[1100.0, 1750.0]

б) Падение дисперсии:

27758.02469

в) Список удалённых точек и дисперсий:

[950.0, 2700.0, 3879.19144];[300.0, 1800.0, 9127.95142];[950.0, 2700.0, 14750.88183]

# **6.2.2. Задача 7.**

Входные данные для задачи №7:

* Х{1988;1989;1990;1991;1992;1993;1994;1995;1996;1997;1998;1999;2000;2001;2002;2003;2004;2005;2006;2007;2008;2009;2010;2011;2012;2013;2014;2015;2016;2017;2018;2019;2020;2021;2022;2023;2024;2025;2026;2027;2028;2029;2030;2031;2032;2033;2034;2035;2036;2037;2038;2039;2040;2041;2042;2043;2044;2045;2046}
* У{1975;2417;2311;2227;2139;2125;2072;1998;1975;1972;2007;2053;2023;2000;1993;1969;1967;1938;1902;1888;1895;1951;2047;2083;2057;2111;2194;4130;2007;1892;1732;1547;1315;1385;1337;4638;1212;1232;1157;1195;1223;1286;1319;1344;1294;1305;1416;1502;1542;1567;1582;1691;1707;2739;1777;2776;1621;1579;1504}
* Процент точек для выбросов: 10%

Выходные данные для задачи №7:

а) Массив после исключения:

[1988.0, 1975.0];[1989.0, 2417.0];[1990.0, 2311.0];[1991.0, 2227.0];[1992.0, 2139.0];[1993.0, 2125.0];[1994.0, 2072.0];[1995.0, 1998.0];[1996.0, 1975.0];[1997.0, 1972.0];[1998.0, 2007.0];[1999.0, 2053.0];[2000.0, 2023.0];[2001.0, 2000.0];[2002.0, 1993.0];[2003.0, 1969.0];[2004.0, 1967.0];[2005.0, 1938.0];[2006.0, 1902.0];[2007.0, 1888.0];[2008.0, 1895.0];[2009.0, 1951.0];[2010.0, 2047.0];[2011.0, 2083.0];[2012.0, 2057.0];[2013.0, 2111.0];[2014.0, 2194.0];[2015.0, 4130.0];[2016.0, 2007.0];[2017.0, 1892.0];[2018.0, 1732.0];[2019.0, 1547.0];[2020.0, 1315.0];[2021.0, 1385.0];[2022.0, 1337.0];[2023.0, 4638.0];[2029.0, 1286.0];[2030.0, 1319.0];[2031.0, 1344.0];[2032.0, 1294.0];[2033.0, 1305.0];[2034.0, 1416.0];[2035.0, 1502.0];[2036.0, 1542.0];[2037.0, 1567.0];[2038.0, 1582.0];[2039.0, 1691.0];[2040.0, 1707.0];[2041.0, 2739.0];[2042.0, 1777.0];[2043.0, 2776.0];[2044.0, 1621.0];[2045.0, 1579.0];[2046.0, 1504.0]

б) Падение дисперсии:

12650.63939

в) Список удалённых точек и дисперсий:

[2027.0, 1195.0, 2898.1613];[2028.0, 1223.0, 2355.75928];[2025.0, 1232.0, 2321.87503];[2029.0, 1286.0, 2443.59509];[2029.0, 1286.0, 2631.24869]

# **6.2.3. Задача 8.**

Входные данные для задачи №8:

* Х{1696;1733;1773;1825;44774;2812;1757;1734;1737;1717;1714;1698;44743;1739;2995;3104;1826;1874;1947;1896;1826;1698;1531;1351;1195;1234;1193;4164;1094;1109;1045;1089;1124;1189;1223;1253;1207;44197;1294;1372;1415;1439;1442;1541;1551;1588;1678;3242;3173;3104;3035;2966;2898}
* У{1174;3195;3098;3026;2928;2974;2898;2746;2627;2535;2588;2656;2413;2704;2764;2779;2773;2734;2497;2504;2562;2758;33270;2988;2936;3003;3162;3057;2304;4471;2447;2219;1913;1884;1813;1705;1603;1643;1534;1554;1564;1633;1666;1654;1576;1601;1798;1912;1941;1983;2056;2215;2264}
* Процент точек для выбросов: 20% .

Выходные данные для задачи №8:

а) Массив после исключения:

[1733.0, 3195.0];[1773.0, 3098.0];[1825.0, 3026.0];[44774.0, 2928.0];[2812.0, 2974.0];[1757.0, 2898.0];[1734.0, 2746.0];[1737.0, 2627.0];[1717.0, 2535.0];[1714.0, 2588.0];[1698.0, 2656.0];[44743.0, 2413.0];[1739.0, 2704.0];[2995.0, 2764.0];[3104.0, 2779.0];[1826.0, 2773.0];[1874.0, 2734.0];[1947.0, 2497.0];[1896.0, 2504.0];[1826.0, 2562.0];[1698.0, 2758.0];[1531.0, 33270.0];[1351.0, 2988.0];[1195.0, 2936.0];[1234.0, 3003.0];[1193.0, 3162.0];[4164.0, 3057.0];[1094.0, 2304.0];[1109.0, 4471.0];[1045.0, 2447.0];[1089.0, 2219.0];[1124.0, 1913.0];[1189.0, 1884.0];[1223.0, 1813.0];[1253.0, 1705.0];[1442.0, 1666.0];[1678.0, 1798.0];[3242.0, 1912.0];[3173.0, 1941.0];[3104.0, 1983.0];[3035.0, 2056.0];[2966.0, 2215.0];[2898.0, 2264.0]

б) Падение дисперсии:

3637546.17014

в) Список удалённых точек и дисперсий:

[1733.0, 3195.0, 285530.7285];[1372.0, 1554.0, 317710.43774];[1415.0, 1564.0, 329903.29398];[1439.0, 1633.0, 342180.62156];[1588.0, 1601.0, 355272.43983];[1678.0, 1798.0, 369969.31943];[44197.0, 1643.0, 384016.01699];[1442.0, 1666.0, 400829.62441];[1442.0, 1666.0, 417289.98654];[1678.0, 1798.0, 434843.70116]

# **6.2.4. Задача 9.**

Входные данные для задачи №9:

* Х{3.162;3.057;2.63;2.6;2.447;2.219;1.913;1.884;1.813;1.705;1.603;1.643;1.534;1.554;1.564;1.633;1.666;1.654;1.576;1.601;1.798;1.912;1.941;1.983;2.056;2.215;2.264;2.318 2.111;2.06;1.923;1.87;1.754}
* У{2.13;2.007;1.892;1.732;1.547;1.36;1.385;1.337;1.27;1.212;1.232;1.157;1.195;1.223;1.286;1.319;1.344;1.294;1.305;1.416;1.502;1.542;1.567;1.582;1.691;1.707;1.75;1.777;1.76;1.621;1.579;1.504;1.504}
* Процент точек для выбросов: 30%

Выходные данные для задачи №9:

а) Массив после исключения:

[2.447, 1.547];[2.219, 1.36];[1.913, 1.385];[1.884, 1.337];[1.813, 1.27];[1.705, 1.212];[1.603, 1.232];[1.643, 1.157];[1.534, 1.195];[1.554, 1.223];[1.564, 1.286];[1.633, 1.319];[1.666, 1.344];[1.654, 1.294];[1.576, 1.305];[1.601, 1.416];[1.798, 1.502];[1.912, 1.542];[1.941, 1.567];[1.983, 1.582];[2.06, 1.621];[1.923, 1.579];[1.87, 1.504];[1.754, 1.504]

б) Падение дисперсии:

0.03867

в) Список удалённых точек и дисперсий:

[3.057, 2.007, 0.01068];[2.63, 1.892, 0.00745];[2.6, 1.732, 0.00477];[2.111, 1.76, 0.00244];[2.06, 1.621, 0.00247];[2.06, 1.621, 0.0027];[2.447, 1.547, 0.00275];[2.06, 1.621, 0.00264];[2.06, 1.621, 0.00277]

# **6.2.5. Задача 10.**

Входные данные для задачи №10:

* Х{2.54;2.417;2.311;2.227;2.139;2.125;2.072;1.998;1.975;1.972;2.007;2.053;2.023;2;1.993;1.969;1.967;1.938;1.902;1.888;1.895;1.951;2.047;2.083;2.057;2.111;2.194;2.13;2.007;1.892;1.732;1.547;1.36;1.385;1.337;1.27;1.212;1.232;1.157;1.195}
* У{3.32;3.195;3.098;3.026;2.928;2.974;2.898;2.746;2.627;2.535;2.588;2.656;2.66;2.704;2.764;2.779;2.773;2.734;2.497;2.504;2.562;2.758;2.91;2.988;2.936;3.003;3.162;3.057;2.63;2.6;2.447;2.219;1.913;1.884;1.813;1.705;1.603;1.643;1.534;1.554}
* Процент точек для выбросов: 50%

Выходные данные для задачи №10:

а) Массив после исключения:

[1.975, 2.627];[1.972, 2.535];[2.007, 2.588];[2.053, 2.656];[2.023, 2.66];[1.902, 2.497];[1.888, 2.504];[1.895, 2.562];[2.007, 2.63];[1.892, 2.6];[1.732, 2.447];[1.547, 2.219];[1.36, 1.913];[1.385, 1.884];[1.337, 1.813];[1.27, 1.705];[1.212, 1.603];[1.232, 1.643];[1.157, 1.534];[1.195, 1.554]

б) Падение дисперсии:

0.05492

в) Список удалённых точек и дисперсий:

[2.417, 3.195, 0.00854];[2.311, 3.098, 0.00502];[2.13, 3.057, 0.00474];[2.227, 3.026, 0.0034];[2.007, 2.63, 0.00276];[2.139, 2.928, 0.00244];[2.007, 2.63, 0.00234];[2.057, 2.936, 0.00255];[2.072, 2.898, 0.0028];[2.007, 2.63, 0.00226];[2.072, 2.898, 0.00275];[2.007, 2.63, 0.00297];[1.998, 2.746, 0.00347];[1.967, 2.773, 6e-05];[1.938, 2.734, 0.00034];[1.938, 2.734, 0.0007];[2.007, 2.63, 0.00131];[1.975, 2.627, 0.0018];[1.902, 2.497, 0.00244];[1.902, 2.497, 0.00235]

# **7. Заключение**

В данной проектной документации решалась задача на прогнозирование путем аппроксимации функции с помощью метода наименьших квадратов. Метод наименьших квадратов применим при обработке результатов эксперимента для приближения данных аналитической формулой. Была разработана математическая модель для решения типовых задач, на основании которой затем были разработаны программы средствами Python для нахождения результата функции с учетом алгоритмов решения линейной и квадратичной аппроксимации, далее были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат.

Проведённая работа с различными способами аппроксимации одной и той же функции или одних и тех же наборов значений задач показала и определила области их применения и действия: линейная - простейшая и часто используемая, а квадратичная – в случае, если линейным полиномом не удается точно аппроксимировать экспериментальные данные, применяют. Квадратичная аппроксимация, пожалуй, позволила выделить именно ее график аппроксимирующей функции за счет того, что на рассчитываемом участке при квадратичной аппроксимации с достаточно высокой точностью повторяет график исходной функции.

# **МНОГОКРИТЕРИАЛЬНАЯ ОПТИМИЗАЦИЯ**

**1. Физическая модель задачи**

Производственная компания должна принять решение о том, в какой из 10 новых проектов ей стоит вложить средства. Специалисты провели сравнительный анализ всех проектов по нескольким показателям и получили следующие данные в виде таблицы:

*Таблица 1- Сравнительный анализ проектов*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер проекта | Срок окупаемости (мес.) | Сумма вложений  (млн. руб.) | Планируемая прибыль за год (млн. руб.) | Рентабельность инвестиций (%) | Объем производства  (нат. ед.) | Коэффициент эффективности | Количество объектов модернизации (шт.) | Коэффициент дисконтирования |
| 1 | 6 | 10 | 12 | 150 | 680 | 50 | 100 | 7 |
| 2 | 5 | 6 | 8 | 80 | 770 | 40 | 250 | 9 |
| 3 | 6 | 11 | 15 | 65 | 526 | 65 | 150 | 10 |
| 4 | 7 | 5 | 8 | 163 | 458 | 30 | 85 | 11 |
| 5 | 4 | 8 | 10 | 70 | 745 | 20 | 150 | 7 |
| 6 | 8 | 8 | 12 | 110 | 652 | 25 | 65 | 12 |
| 7 | 7 | 11 | 10 | 95 | 552 | 63 | 200 | 6 |
| 8 | 6 | 9 | 9 | 123 | 410 | 45 | 180 | 5 |
| 9 | 9 | 10 | 15 | 84 | 520 | 35 | 170 | 10 |
| 10 | 9 | 12 | 12 | 130 | 630 | 55 | 130 | 12 |

**2. Математическая модель задачи**

Пусть Х – множество допустимых решений в задаче, тогда допустимым решением будет . Качество решения оценивается по нескольким критериям.

Каждое допустимое решение оцениваем по заданному количеству критериев – n, где критериев больше одного, то есть n ≥ 2.

Получаем несколько целевых функций – fi(x), которые необходимо максимизировать или минимизировать. Общая формула имеет вид:

Более подробно вектор fi(x) описывается набором допустимых решений по всем критериям:

Необходимо искать компромиссное решение, которое будет учитывать ценность каждой целевой функции. Предпочтительность допустимого решения увеличивается с ростом вектора fi(x). Таким образом, чем больше значение fi(x), тем лучше получаем решение х по критерию i (i=1, n).

Поиск решения также сопровождается рассмотрением некоторых ограничений (объем ресурсов, денежных средств и т.п.), которые запишем в следующем виде:

**2.1. Поиск оптимального решения по Паретто**

Это наиболее общий метод решения, при котором решение хотя бы по одному критерию нельзя улучшить, не ухудшив его по остальным критериям. Суть метода состоит в том, что оптимальное решение следует искать только среди элементов множества.

Таким образом, решение будет Пареттооптимальным, если существует такое решение y, которое:

**2.2. Линейная свертка критериев**

Это один из методов сведения МКО к однокритериальной задаче.

Имеются положительные весовые коэффициенты , которые обозначают степень ценности критериев. Затем максимизируются целевые функции:

Где

Когда один из коэффициентов равен нулю, а все остальные коэффициенты i≠j, получается однокритериальная задача максимизации целевой функции j.

**2.3. Метод идеальной точки**

Множество достижимости определим множеством всех значений целевых функций, получаемых при всех допустимых значениях х:

Состоящий из максимальных значений всех целевых функций вектор назовем идеальной точкой:

Оптимальной будет являться точка, в которой максимизация происходит по всем критериям.

Для решения МКО необходима точка, в которой вектор значений целевых функций по норме минимально отличается от идеальной:

Возможно решение через различные виды нормы, например, через Чебышевскую норму, когда отклонение максимально по модулю:

Но стандартным решением является норма по Евклиду:

**2.4. Метод контрольных показателей**

Данный метод находит решение, при котором все целевые функции fi(x) расположены дальше всего от своих нижних границ fi\*.

Контрольным показателем каждой функции fi(x) будет являться ее нижняя граница:

Необходимо максимизировать минимальное расстояние от fi(x) до контрольного показателя fi\*:

**3. Алгоритмы решения задачи**

**3.1. Поиск Паретооптимального решения;**

Выбор множества Пареттооптимальных решений (множества Паретто) представляет собой отбор перспективных альтернатив, из которых затем отбирается одна (лучшая) альтернатива.

Множество Паретто представляет собой множество альтернатив, обладающих следующим свойством: любая из альтернатив, входящих во множество Парето, хотя бы по одному критерию лучше любой другой альтернативы, входящей в это множество.

**3.1.1. Описание входных данных**

а) массив характеристик объектов

б) список критериев, по которым осуществляется оптимизация;

в) направление оптимизации для каждого из указанных в п.б) критериев (к максимуму или к минимуму);

**3.1.2. Описание алгоритма решения**

1. Происходит выбор, какие столбцы векторов максимизируются, какие минимизируются для поиска Пареттооптимального значения.

2. Далее программа начинает сравнивать элементы каждого столбца, для того чтобы выявить эти «не улучшаемые» значения (при этом в программе прописаны условия, которые не удовлетворяют таким значениям).

3. После обработки всех элементов по векторам выводится результат.

**3.1.3. Описание выходных данных**

а) набор точек поверхности Паретто;

б) Диаграмма значений (если количество параметров <=3);

в) Лепестковая диаграмма с указанием Пареттооптимального множества решений (если количество параметров >=3).

**3.2. Линейная свертка критериев**

Линейная свертка критериев-метод, при котором осуществляется переход от нескольких критериев к одному, путем задания весовых коэффициентов характеристик (важности) каждого из критериев, по которому выполняется выборка

**3.2.1. Описание входных данных**

а) массив характеристик объектов;

б) список критериев, по которым осуществляется оптимизация;

в) направление оптимизации для каждого из указанных в п.б) критериев (к максимуму или к минимуму);

г) весовые коэффициенты характеристик (важность);

**3.2.2. Описание алгоритма решения**

1. Определяем, какой из критериев будет максимизироваться, а какой минимизироваться.

Это необходимо для того, чтобы правильно осуществить линейную свертку, так как при разноименных критериях (один критерий к максимуму, а другой к минимуму) при поиске решения минимизируемое выражение будет идти со знаком минус

2. Осуществляем поиск точки с учетом весовых коэффициентов характеристик.

3. Определяем точку, при которой достигается искомое значение.

4. Выводим значение точки.

5. Определение набора точек поверхности Паретто:

1. Создаем два вектора весовых коэффициентов характеристик от 0 до 1 с шагом 0,1.
2. Осуществляем поиск максимальных значениям коэффициентов.
3. Выявляем количество отличных точек.

**3.2.3. Описание выходных данных**

а) Оптимальное значение;

б) Диаграмма линейной свертки критериев;

в) набор точек поверхности Паретто.

**3.3. Метод идеальной точки**

Используется для поиска сбалансированного решения, наиболее близкого к идеальному.

**3.3.1. Описание входных данных**

а) массив характеристик объектов;

б) список критериев, по которым осуществляется оптимизация;

в) направление оптимизации для каждого из указанных в п.б) критериев (к максимуму или к минимуму);

**3.3.2. Описание алгоритма решения**

1. Задаем вектор идеальных значений исходя из массива характеристик объектов (в коде: для поиска такой точки каждый из максимальных показателей умножаем на 1,)
2. Вычисляем норму по Евклиду (формула 10) для всех показателей
3. Суммируем по строке все значения и берем из них корень (чтобы найти расстояние)
4. Выбираем минимальное значение, так как расстояние от идеальной точки до другой (удовлетворяющей) должны быть минимальным

**3.3.3. Описание выходных данных**

а) Оптимальное значение;

б) Лепестковая диаграмма с указанием Пареттооптимального множества решений;

д) набор точек поверхности Паретто (идеальная точка).

**3.4. Метод контрольных показателей**

Используется для поиска наиболее сбалансированного решения, которое наиболее удалено от минимальной границы по всем показателям

3.4.1. **Описание входных данных**

а) массив характеристик объектов;

б) список критериев, по которым осуществляется оптимизация;

в) направление оптимизации для каждого из указанных в п.б) критериев (только к максимуму);

3.4.2. **Описание алгоритма решения**

1. Задаем нижнюю границу
2. Вычисляем (по формуле 12) глобальный критерий (расстояние точки до границы)
3. Расстояния меньше 1 являются выбросами, следовательно, мы их удаляем
4. Значения, которые удовлетворяют условиям складываем в новую матрицу и находим из них находим минимальное значение по всем критериям (т.е те критерии, которые больше всего приближены к нижней границе)
5. Находим из этих минимумов максимальное значение (оно максимально удалено от границы)
6. Определяем какому из показателей соответствует эта характеристика и выводим координату оптимальной точки

3.4.3. **Описание выходных данных**

Выходными данными будет являться координата оптимальной точки.

**4. Варианты использования системы**

Для пользователя предусмотрен следующий вариант использования системы с использованием средств Python.

1. Пользователь должен запустить среду разработки для Python.
2. Пользователь должен запустить код для решения задачи.
3. Пользователю необходимо объявить следующие входные данные:
   1. Для Пареттооптимального метода:

* матрица 2 столбцов
* вектор направлений( max, min или minmax).
  1. Для метода линейной свёртки:
* матрица любого количества столбцов
* вектор направлений
* вектор коэффициентов
  1. Для линейной свёртки (Паретто) для двух параметров (макс/макс):
* матрица из двух столбцов
* вектор направления - только максимизация
  1. Для метода идеальной точки:
     + - * матрица из любого количества столбцов
         * вектор направления - только максимизация
  2. Для метода контрольных показателей:
     + - * матрица из любого количества столбцов
         * вектор направления - только максимизация

1. Программа выполняет алгоритм.
2. Пользователю должен выводиться результат в следующем виде:
   1. Для Пареттооптимального метода:

* массив оптимальных точек Паретто
  1. Для линейной свёртки (Паретто) для двух параметров (макс/макс):
     + - список списков точек для поверхности Паретто
  2. Для метода линейной свёртки, метода идеальной точки и метода контрольных показателей:
     + - оптимальная точка (ее координаты)

**5. Архитектура решения**

**5.2. Функции обработки информации**

**def** **paretto**(matrica123, vector\_napr):

Функция пареттооптимального решения. Сравнивает значения и находит «не улучшаемые», которые и являются решением.

**def** **lin\_svertka**(matrica123, vector\_napr, vector\_kf):

Функция линейной свёртки. Поиск решения сводится к одному критерию при задании весовых коэффициентов характеристик (важности). Отражает исследуемый набор точек по выбранным критериям, выделяет точку оптимального значения.

**def** **lin\_svertka\_paretto**(matrica123):

Функция линейной свёртки (Паретто) для двух параметров (макс/макс). Функция необходима для поиска набора точек поверхности Паретто.

**def** **ideal\_tochka**(matrica):

Функция идеальной точки. Ищет минимальное расстояние от идеальной точки до значений через формулу нормы Эвклида.

**def** **control\_pokazat**(matrica):

Функция контрольных показателей. Задает нижнюю границу нормы и выполняет поиск такой точки, минимальный элемент характеристики был максимально удален от этой границы по сравнению со всеми остальными минимальными точками, удовлетворяющим задаче.

**def** **plot\_paretto**(matrica):

Функция для построения лепестковой диаграммы с указанием Пареттооптимального множества решений. Отражает исследуемый набор точек, выделяет набор точек поверхности Паретто.

**5.3. Функции вывода информации**

Вывод информации осуществляет с помощью функции print

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 1. Функции вывода информации*

1. **Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Входные данные** | **Поиск Пареттооптимального решения** | **Линейная свертка критериев** | **Метод идеальной точки** | **Метод контрольных показателей** |
| [[12, 12, 4.6, 80, 7, 24, 10.2],  [9, 12, 5.4, 81, 1, 16, 15],  [12, 5, 3.9, 75, 3, 19, 8.8],  [1, 10, 4.9, 63, 21, 36, 10],  [10, 2, 4.3, 66, 20, 35, 9.9],  [7, 7, 4.5, 86, 18, 35, 11.5],  [6, 13, 5.5, 79, 5, 22, 8.6],  [14, 0, 5.5, 61, 9, 26, 11.4],  [0, 12, 3.9, 63, 5, 22, 11.8],  [13, 10, 5.5, 65, 21, 36, 12.2],  [5, 4, 3.3, 86, 17, 31, 11.1],  [12, 6, 2.6, 84, 17, 33, 10.8],  [2, 15, 3.5, 59, 15, 30, 7.1],  [2, 5, 4.2, 69, 6, 23, 6.8],  [9, 12, 5.1, 83, 7, 24, 7.8],  [1, 14, 4.5, 65, 5, 19, 10.5]] | [[0, 12], [2, 15], [1, 14]] | Точка №9  Набор точек поверхности Паретто после линейной свёртки:  [[12, 12, 4.6, 80, 7, 24, 10.2],  [13, 10, 5.5, 65, 21, 36, 12.2],  [2, 15, 3.5, 59, 15, 30, 7.1],  [14, 0, 5.5, 61, 9, 26, 11.4]] | [13, 10, 5.5, 65, 21, 36, 12.2] | [12, 12, 4.6, 80, 7, 24, 10.2] |
| [[6, 10, 12, 150, 680, 50, 100, 7],  [5, 6, 8, 80, 770, 40, 250, 9],  [6, 11, 15, 65, 526, 65, 150, 10],  [7, 5, 8, 163, 458, 30, 85, 11],  [4, 8, 10, 70, 745, 20, 150, 7],  [8, 8, 12, 110, 652, 25, 65, 12],  [7, 11, 10, 95, 552, 63, 200, 6],  [6, 9, 11, 123, 410, 45, 180, 5],  [9, 10, 15, 84, 520, 35, 170, 10],  [9, 12, 12, 130, 630, 55, 130, 12]] | [[5, 8], [8, 12], [10, 15]] | Точка №9  Набор точек поверхности Паретто после линейной свёртки:  [[9, 10, 15, 84, 520, 35, 170, 10],  [9, 12, 12, 130, 630, 55, 130, 12]] | [9, 12, 12, 130, 630, 55, 130, 12] | [6, 11, 15, 65, 526, 65, 150, 10] |
| [[22.8,38,45.6,570, 2584, 190, 380, 26.6, 31.5],  [19.7, 22.8,30.4,304,2926,152,950, 34.2, 40.5],  [22.8, 41.8, 57, 247, 1998, 247, 570, 38, 45],  [26.6, 19, 30.4,619, 1740, 114,323, 41.8, 49.5],  [15.2, 30.4,38, 266, 2831, 76, 570, 26.6, 31.5],  [30.4, 30.4, 45.6, 418, 2477, 95, 247, 45.6, 54],  [26.6, 41.8, 38, 361, 2097, 239, 760, 22.8, 27],  [22.8, 34.2, 41.8, 467,1558,171, 684, 19, 22.5],  [34.2, 38, 57, 319, 1976, 133, 646, 38, 45],  [34.2, 45.6,45.6,494, 2394, 209, 494, 45.6, 54]] | [[26.6, 19], [30.4, 30.4], [34.2, 38]] | Точка №4  Набор точек поверхности Паретто после линейной свёртки:  [[34.2, 38, 57, 319, 1976, 133, 646, 38, 45],  [34.2, 45.6, 45.6, 494, 2394, 209, 494, 45.6, 54]] | [34.2, 45.6, 45.6, 494, 2394, 209, 494, 45.6, 54] | [26.6, 19, 30.4, 619, 1740, 114, 323, 41.8, 49.5] |
| [[79.8, 228, 114, 3990, 5168, 95, 456, 74.48, 59.85, 196.08, 1425],  [66.5, 136.8, 76, 2128, 5852, 76, 1140, 95.76, 76.95, 130.72, 760],  [79.8, 250.8, 142.5, 1729, 3997.6, 123.5, 684, 106.4, 85.5, 245.1, 617.5],  [79.8, 205.2, 104.5, 3271, 3116, 85.5, 820.8, 53.2, 42.75, 179.74, 1168.5],  [119.7, 228, 142.5, 2234, 3952, 66.5, 775.2, 106.4, 85.5, 245.1, 798],  [53.2, 182.4, 95, 1862, 5662, 38, 684, 74.48, 59.85, 163.4, 665],  [106.4, 142.4, 114, 2926, 4955.2, 47.5, 296.4, 127.68, 102.6, 196.08, 1045],  [93.1, 114, 76, 4335, 3480.8, 57, 387.6, 117.04, 94.05, 130.72, 1548.5],  [93.1, 250.8, 95, 2527, 4195.2, 119.7, 912, 63.84, 51.3, 163.4, 902.5],  [109.7, 273.6, 114, 3458, 4788, 104.5, 592.8, 127.68, 102.6, 196.08, 1235]] | [[196.08, 1425],  [245.1, 798],  [130.72, 1548.5]] | Точка №1  Набор точек поверхности Паретто после линейной свёртки:  [[109.7, 273.6, 114, 3458, 4788, 104.5, 592.8, 127.68, 102.6, 196.08, 1235],  [119.7, 228, 142.5, 2234, 3952, 66.5, 775.2, 106.4, 85.5, 245.1, 798]] | [109.7, 273.6, 114, 3458, 4788, 104.5, 592.8, 127.68, 102.6, 196.08, 1235] | [79.8, 205.2, 104.5, 3271, 3116, 85.5, 820.8, 53.2, 42.75, 179.74, 1168.5] |
| [[119.7, 51.3, 163.4, 902.5, 195.2, 63.84, 95, 912, 2527, 93.1, 250.8],  [85.5, 42.75, 179.74, 1168.5, 3116, 53.2, 104, 820, 3271, 79.8, 205.2],  [76, 76.95, 130.72, 760, 5852, 95.76, 76, 1140, 2128, 66.5, 136.8],  [104.5, 102.6, 196.08, 1235, 4788, 127.68, 114, 592, 3458, 119.7, 273.6],  [38, 59.85, 163.4, 665, 5662, 74.48, 95, 684, 1862, 53.2, 182.4],  [47.5, 102.6, 196.08, 1045, 4955.2, 127.68, 114, 296, 2926, 106.4, 182.4],  [66.5, 85.5, 245.1, 798, 3952, 106.4, 142, 775, 2234, 119.7, 228],  [123.5, 85.5, 245.1, 617.5, 3997.6, 106.4, 142, 684, 1729, 79.8, 250.8],  [57, 94.05, 130.72, 1548.5, 3480.8, 117.04, 76, 387, 4335, 93.1, 114],  [95, 59.85, 196.08, 1425, 5168, 74.48, 114, 456, 3990, 79.8, 228]] | [[38, 59.85],  [47.5, 102.6]] | Точка №6  Набор точек поверхности Паретто после линейной свёртки:  [[104.5, 102.6, 196.08, 1235, 4788, 127.68, 114, 592, 3458, 119.7, 273.6],  [123.5, 85.5, 245.1, 617.5, 3997.6, 106.4, 142, 684, 1729, 79.8, 250.8]] | [104.5, 102.6, 196.08, 1235, 4788, 127.68, 114, 592, 3458, 119.7, 273.6] | [85.5, 42.75, 179.74, 1168.5, 3116, 53.2, 104, 820, 3271, 79.8, 205.2] |

Случаи, когда идеальное решение задачи со многими критериями существует, крайне редки. Практически во всех случаях задача МКО сводится к решению однокритериальной задачи. Однако ни одному из существующих способов построения такого критерия невозможно заранее отдать предпочтение, в каждом из отдельных случаев выбор метода и решения осуществляется лицом, принимающим решение.

1. **Заключение**

В данной проектной документации решалась многокритериальная оптимизационная задача путем применения различных методов таких, как метод оптимизации по Паретто, метод линейной свертки, метод идеальной точки, а также метод контрольных показателей. Была разработана математическая модель для решения типовых задач, на основании которой затем были разработана программа средствами Python для нахождения результата с учетом алгоритмов решения приведенных в документации методов, были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат.

Тестирование различных методов решения задач многокритериальной оптимизации, в свою очередь, показало, что все методы дают разные наборы точек, так как их алгоритмы имеют разные подходы к решению задач. Следовательно, за верное решение может быть принят любой из наборов точек, исходя из преследуемой цели.

# **ЭКСПЕРТНЫЕ ОЦЕНКИ**

# **Физическая модель задачи**

Был создан опрос на тему: «Проблема безработицы молодых специалистов». В данном опросе выясняется мнение экспертов о причинах данного социально-экономического явления. Экспертам необходимо оценить их, используя цифры, которые будут указывать на согласие\несогласие с выбранными утверждениями. Далее приведены ссылки и таблицы с утверждениями для 3 технологий оценивания.

* 1. **Обработка экспертных оценок**.

Ссылка на опрос: [https://docs.google.com/forms/d/e/1FA](https://docs.google.com/forms/d/e/1FAIpQLScYhYiwRIitNW4ndW_ubP4QMwY3ODy9cPtvXHzAoUMk8tMJow/viewform)

Экспертам необходимо произвести ранжирование: 1-совсем не согласен, 5–50\50, 10-полностью согласен.

Таблица 1 – Список утверждений для оценивания

|  |  |
| --- | --- |
| 1 | Проблема безработицы молодых специалистов связана с их нежеланием начинать путь по карьерной лестнице с должностей с небольшим заработком |
| 2 | Проблема безработицы молодых специалистов связана с малым количеством стажировок в стране |
| 3 | Проблема безработицы молодых специалистов связана с завышенными требованиями работодателей |
| 4 | Проблема безработицы молодых специалистов связана с недостатком свободных рабочих мест |
| 5 | Проблема безработицы молодых специалистов связана со спадом в экономике |
| 6 | Проблема безработицы молодых специалистов связана со спецификой психологического склада молодежи (самонадеянность, инфантильность, упрямость и др.) |
| 7 | Проблема безработицы молодых специалистов связана с недостатком знаний, необходимых для получения рабочего места |
| 8 | Проблема безработицы молодых специалистов связана с несоответствием программ подготовки в высших и средне-профессиональных учебных заведений с реальными профессиональными требованиями, предъявляемые работодателями на рынке труда |
| 9 | Проблема безработицы молодых специалистов связана с общим ростом безработицы в стране, при этом никаких других причин нет. |
| 10 | Проблема безработицы молодых специалистов связана с ошибочным выбором сферы деятельности, который был сделан при поступлении в высшее или средне-профессиональное учебное заведение |

* 1. **Обработка ранговых оценок**.

Ссылка на опрос:

<https://vk.com/away.php?to=https%3A%2F%2Fforms.gle%2FADVE9jij4G3A2fey6&cc_key=>

Экспертам необходимо проранжировать последствия безработицы молодых специалистов: от самых незначительных, до самых фатальных для экономики страны и для домохозяйств (семей).

Таблица 2 – Список утверждений для оценивания

|  |  |
| --- | --- |
| 1 | Рост затрат общества на производство товаров и услуг |
| 2 | Снижение уровня и качества жизни населения из-за потерь личных доходов отдельных граждан и их семей |
| 3 | Препятствие полноценной социализации молодого поколения, из-за невозможности освоения социальной роли работника |
| 4 | Создание условий к формированию девиации среди молодежи, в том числе криминальное поведение |
| 5 | Снижение валового внутреннего продукта в силу недоиспользования в общественном воспроизводстве трудового потенциала |
| 6 | Деквалификация рабочей силы, потеря профессиональных навыков |
| 7 | Увеличение психологических расстройств у людей |
| 8 | Усиление предпосылок для социальной нестабильности, влияющей на политическую ситуацию в стране (протесты, митинги) |
| 9 | Безработица приводит к обесцениванию образования, потере авторитетности его приобретения |
| 10 | Увеличение уровень социальной дифференциации, т.е разрыв между "богатыми" и "бедными" растет. |
| 11 | Налоговые поступления сокращаются, тем самым затрудняется развитие тех сфер экономики, которые наиболее зависят от государственного субсидирования. |
| 12 | Увеличение социальной ценность и значимость труда, страх потери рабочего места |
| 13 | Появление личного время на переосмысление собственных приоритетов. |
| 14 | Увеличение резерва рабочей силы, который может быть использован для расширения производства при переходе в фазу экономического цикла "подъем" |
| 15 | Сокращение инвестиционных процессов в виду высокого уровня безработицы |

* 1. **Обработка бинарных отношений**.

Ссылка на опрос:

<https://docs.google.com/forms/d/e/1FAIpQLSf3vFLFiALXxDAcIUb6VWkj_0CN0VR1x1IslYixHDJr5zJRWw/viewform?fbzx=-4840749816067495713>

Экспертам предлагается сравнить между собой эффективность нескольких проектов по борьбе с безработицей.

Таблица 3 – Список мер по борьбе с безработицей

|  |  |  |
| --- | --- | --- |
| 1 | Программа по стимулированию занятости и роста числа рабочих мест | Данная программа предполагает стимулирование роста занятости происходит за счет льготных преференций для определенного вида профессий и расширение рабочих мест происходит на государственных предприятиях |
| 2 | Программа повышения квалификации и профессиональной подготовки кадров | Данная программа также ориентирована на государство: создаются центры подготовки и переподготовки кадров на специальности, востребованные на данный момент на рынке труда |
| 3 | Программа развития малого бизнеса | Данная программа предполагает стимулирование создания малых бизнесов путем введения льготного налогообложения, субсидирования и преференций |
| 4 | Программа развития информирования молодых специалистов о наличии вакансий | Данная программа предполагает расширение деятельности центров занятости, создания государственных и негосударственных сайтов, предполагающих распространение информации о востребованных профессиях на рынке труда в конкретный момент времени |

# **2. Математическая модель задачи**

После проведения опроса экспертов имеются результаты, по которым происходит их последующая обработка. Решение проблемы формируется на основе полученных результатов обработки.

Пусть n – количество экспертов, m – количество оцениваемых признаков, а поставленная оценка признака будет xij. В матричном представлении имеется следующая таблица размерностью mхn:

Таблица 4 – Матрица экспертных оценок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | … | n |
| 1 | x11 | x12 | x13 | … | x1n |
| 2 | x21 | x22 | x23 | … | x2n |
| 3 | x31 | x32 | x33 | … | x3n |
| … | … | … | … | … | … |
| m | xm1 | xm2 | xm3 | … | xmn |

Цель – осуществить ранжировку показателей (упорядочивание), то есть итоговое мнение экспертов.

**2.1. Метод средних арифметических рангов**

Метод средних баллов состоит в том, что для каждого признака рассчитывается среднее значение (матожидание):

Итоговая ранжировка строится по средним рангам. В результате можно будет сделать вывод, согласно которому: чем меньше средний ранг, тем лучше показатель.

Чтобы понять степень рассогласованность мнений экспертов по поводу признака, оценивается дисперсия выборки:

Также для понимания того, в каком диапазоне будет пребывать дисперсия (рассогласованность экспертов), рассчитывается доверительный интервал:

**2.2. Метод медианы рангов**

Применяется для измеренных в порядковой шкале экспертных оценок, то есть для признаков, оцененных экспертами по рангам.

Необходимо получить медианы каждого признака, поиск которых зависит от четности количества экспертов:

1. Для нечетного количества: медиана будет соответствовать центральному значению ряда. Формула:
2. Для четного количества: медиана - среднее значение из двух центральных значений:

Для того, чтобы учитывать мнение более компетентных экспертов с большим весом, находятся *средневзвешенные ранги*. Компетентность каждого эксперта пропорциональна дисперсии его оценок:

Таким образом, можно получить значимость мнения каждого эксперта, то есть коэффициент компетентности, рассчитываемого в процентах:

Тогда средневзвешенное мнение всех экспертов с учетом коэффициента их компетентности:

**2.3. Метод бинарных отношений**

Каждый эксперт сравнивает каждую пару вариантов признаков друг с другом. Оценки выставляются в матрицу бинарных отношений, где имеются две оценки: признак лучше или равен другому – 1, в противном случае – 0:

Таблица 5 – Пример матрицы бинарных отношений

|  |  |  |  |
| --- | --- | --- | --- |
|  | m1 | m2 | m3 |
| m1 | 1 | 0 | 1 |
| m2 | 1 | 1 | 1 |
| m3 | 0 | 0 | 1 |

Необходимо найти количество несовпадающих элементов матриц экспертов – расстояние Кемени, то есть между мнениями экспертов 1 и эксперта 2 несовпадающие элементы находятся путем вычитания матриц их бинарных отношений, взятие модуля и сложение элементов получившейся матрицы:

Чтобы найти усредненное мнение экспертов, необходимо найти медиану Кемени - матрицу (A), от которой будет минимально суммарное расстояние Кемени до остальных матриц экспертов:

Полученный результат можно привести в ранговый вид, просуммировав по признакам матрицу А

# **3. Алгоритмы решения задачи**

## **3.1. Математическая обработка экспертных оценок;**

Данный метод предусматривает выставление баллов экспертами каждому из исследуемых мнений. Этот метод обусловлен при возможности сравнения исследуемых мнений и количественном подтверждении разницы между ними. Метод реализован по средствам python.

### **3.1.1. Описание входных данных**

а) массив экспертных оценок;

**3.1.2. Описание алгоритма решения**

1. Вычисление среднего заданного списка чисел (массива экспертных оценок). Алгоритм на данном шаге возвращает среднее значение набора данных, переданного в качестве параметров.

2. Расчет дисперсии по выборке данных (выборка — это подмножество заполненных данных).

2. Далее алгоритм производит расчет доверительных интервалов - интервалов, которые покрывают неизвестный параметр с заданной надёжностью. Это происходит благодаря установке уровня значимости α - вероятности, с которой значение параметра не попадает в доверительный интервал, и уровня доверия β = 1 − α - вероятности того, что доверительный интервал накрывает значение параметра.

3. После обработки всех элементов выборки выводится результат, строится гистограмма вариантов.

### **3.1.3. Описание выходных данных**

а) Средний балл (матожидание);

б) Дисперсия;

в) Среднеквадратичное отклонение;

г) Доверительный интервал;

д) Гистограмма вариантов.

## **3.2. Математическая обработка ранговых оценок**

Математическая обработка ранговых оценок - метод, при котором осуществляется ранжирование (упорядочение) объектов экспертизы, т.е. их расположение в порядке возрастания (или, точнее, неубывания) интенсивности интересующей организаторов экспертизы характеристики. При этом методе получаемые от экспертов мнения выражены в порядковой шкале, т.е. эксперт может сказать и обосновать, что один тип продукции/показатель качества продукции будет более привлекателен и важен для потребителей, чем другой. Но он не в состоянии сказать, во сколько раз или на сколько более важен. Метод реализован по средствам python.

### **3.2.1. Описание входных данных**

а) массив ранговых оценок

**3.2.2. Описание алгоритма решения**

1. Вычисление медианы данного набора данных – массива ранговых оценок, которая позволяет избавиться от мнения диссидентов. Этот метод также сортирует данные в порядке возрастания перед вычислением медианы.

2. Вычисление среднего заданного списка чисел (массива экспертных оценок). Алгоритм на данном шаге возвращает среднее значение набора данных, переданного в качестве параметров.

3. Расчет степени отклонения ответа эксперта от полученного среднего значения, т.е. дисперсия мнений каждого эксперта.

4. Для нахождения коэффициента компетентности каждого эксперта осуществляем вычисление обратного значения от полученной дисперсии.

5. Определение средневзвешенного значения для процентного соотношения коэффициентов экспертов.

6. После обработки всех элементов выборки выводится результат, строится гистограмма вариантов.

### **3.2.3. Описание выходных данных**

а) Медиана рангов;

б) Массив коэффициентов компетентности экспертов;

в) средневзвешенные ранги с учетом коэффициентов компетентности;

г) Гистограмма вариантов.

## **3.3. Математическая обработка бинарных отношений**

Метод сравнения исследуемых объектов друг с другом попарно. Применяется в случаях, когда объекты настолько комплексные и многомерные, что нет возможности отранжировать их по одному признаку, но есть возможность попарно сравнить их между собой. Метод реализован по средствам excel.

### **3.3.1. Описание входных данных**

а) массив ранговых оценок;

### **3.3.2. Описание алгоритма решения**

1. Строим матрицу на основе массива ранговых оценок, сравниваем попарно ее объекты для каждого из экспертов:

* если объект а лучше или равен объекту b, метрика приобретает значение 1, в противном случае – значение 0.
* Результатом попарных сравнений будет являться матрица бинарных отношений.

2. Производим расчет расстояния Кемени между бинарными отношениями - число, характеризующее количество несовпадающих элементов матриц.

3. Вводим медиану Кемени – матрицу, которая представляет собой усредненное значение, используя инструмент Поиск решения и вводя целевую функцию и ограничения по бинарности. Результатом данного шага будет являться матрица, суммарное расстояние Кемени от которой до всех остальных матриц является минимальным.

4. Интерпретируем полученный результат в ранговый вид, расставив предприятия в порядке выгодности путем суммирования элементов медианы Кемени по столбцам.

### **3.3.3. Описание выходных данных**

а) Медиана Кемени.

# **4. Варианты использования системы**

Для пользователя предусмотрены два варианта использования системы с использованием средств Python и средствами MS Excel. Пользователь может выбрать любой удобный для него вариант.

## **4.1. ВИ 1\_Решение задач методом средних баллов и методом медианы рангов.**

1. Пользователь должен запустить среду разработки для Python.
2. Пользователь должен запустить код для решения задачи.
   1. Пользователю необходимо ввести следующие входные данные:
      1. Ввести задачу для решения:
         * + «экспертных»;
           + «ранговых».
      2. Ввести способ ввода данных:

* «ручной»
* «файл».
  + 1. Ввести количество объектов исследования:
       - * одно целое число.
    2. Ввести матрицу экспертных оценок:
       - * строки – объекты;
         * столбцы – эксперты;
         * целые числа через пробел.;

1. Программа выполняет алгоритм.
2. Пользователю должен выводиться результат в следующем виде:
   1. Для метода средних баллов:

* вектор медиан рангов;
* вектор коэффициентов компетентности;
* вектор средневзвешенных рангов.
  1. Для метода медианы рангов:
     + - вектор матожиданий;
       - вектор дисперсий;
       - вектор среднеквадратичных отклонений;
       - вектор доверительных интервалов.

## **4.2. ВИ 2\_Решение методом бинарных отношений**

1. Пользователь открывает приложение MS Excel.
2. Пользователь вручную следует разработанному алгоритму (см. глава 3.3.)
3. Пользователь получает ответ.

# **5. Архитектура решения**

## **5.1. Функции считывания информации**

**def** **manual\_input**():

Функция ручного ввода матрицы. Осуществляется ручной ввод количества объектов исследования, а также ввод исходной матрицы (числа через пробел, строки - объекты, столбцы - эксперты).

**def** **file\_input**():

Функция считывания матрицы из файла. Осуществляется перенос файла в папку с алгоритмом и ввод имени данного файла.

Также после запуска алгоритма необходимо выбрать задачу для решения (экспертных, ранговых). После чего выбрать способ ввода данных (ручной, файл).

## **5.2. Функции обработки информации**

**def** **srballs**(matrix):

Функция математической обработки экспертных оценок. Производит вычисление среднего заданного списка чисел, дисперсии по выборке данных, а также доверительных интервалов.

**def** **rangi**(matrix):

Функция математической обработки ранговых оценок. Функция включает в себя вычисление медианы исходного набора данных, среднего заданного списка чисел (массива экспертных оценок). Функция производит расчет дисперсии мнений каждого эксперта и коэффициент компетентности каждого эксперта.

**def** **plot**(matrix):

Функция построения гистограммы вариантов для исходной матрицы.

## **5.3. Функции вывода информации**

Вывод информации осуществляется с помощью функции print :

**print** ('Вектор мат. ожиданий: ', srballs (matrix)[**0**])

**print** ('Вектор дисперсий: ', srballs (matrix)[**1**])

**print** ('Вектор среднеквадратичных отклонений: ', srballs (matrix)[**2**])

**print** ('Вектор доверительных интервалов: ', srballs (matrix)[**3**])

**print** ('Вектор медиан рангов: ', rangi (matrix)[**0**])

**print** ('Вектор коэф. компетентности: ', rangi (matrix)[**1**])

**print** ('Вектор средневз. рангов: ', rangi (matrix)[**2**])

# **6. Тестирование**

По каждому типу задач было проведено тестирование. Ниже приведены таблицы результатов решения задач разными методами.

*Таблица 6 - Результаты тестирования программы средних баллов*

|  |  |  |
| --- | --- | --- |
| Входные данные | Результат | Гистограмма |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Э1 | Э2 | Э3 | Э4 | Э5 | | В1 | 6 | 7 | 5 | 6 | 7 | | В2 | 8 | 5 | 9 | 5 | 5 | | В3 | 6 | 9 | 10 | 8 | 8 | | В4 | 7 | 10 | 7 | 5 | 9 | | В5 | 8 | 10 | 9 | 4 | 5 | | В6 | 9 | 7 | 4 | 3 | 10 | | В7 | 9 | 9 | 3 | 8 | 6 | | В8 | 10 | 10 | 7 | 4 | 7 | | В9 | 1 | 3 | 7 | 2 | 6 | | В10 | 8 | 8 | 4 | 7 | 5 | | **Вектор мат. ожиданий:**  [6.2, 6.4, 8.2, 7.6, 7.2, 6.6, 7.0, 7.6, 3.8, 6.4]  **Вектор дисперсий:**  [0.7, 3.8, 2.2, 3.8, 6.7, 9.3, 6.5, 6.3, 6.7, 3.3]  **Вектор среднеквадратичных отклонений:** [0.8366600265340756, 1.9493588689617927, 1.4832396974191326, 1.9493588689617927, 2.588435821108957, 3.0495901363953815, 2.5495097567963922, 2.5099800796022267, 2.588435821108957, 1.816590212458495]  **Вектор доверительных интервалов:** [[4.526679946931849, 7.873320053068151], [2.501282262076415, 10.298717737923585], [5.2335206051617345, 11.166479394838264], [3.7012822620764143, 11.498717737923585], [2.0231283577820864, 12.376871642217914], [0.5008197272092367, 12.699180272790763], [1.9009804864072155, 12.099019513592784], [2.5800398407955463, 12.619960159204453], [-1.376871642217914, 8.976871642217914], [2.7668195750830105, 10.03318042491699]] |  |

*Таблица 7 - Результаты тестирования программы медианы рангов*

|  |  |  |
| --- | --- | --- |
| Входные данные | Результат | Гистограмма |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Э1 | Э2 | Э3 | Э4 | Э5 | | В1 | 14 | 7 | 7 | 11 | 7 | | В2 | 5 | 3 | 4 | 10 | 10 | | В3 | 12 | 9 | 8 | 9 | 6 | | В4 | 15 | 10 | 3 | 15 | 9 | | В5 | 7 | 15 | 6 | 8 | 4 | | В6 | 1 | 14 | 9 | 7 | 15 | | В7 | 6 | 12 | 10 | 1 | 14 | | В8 | 4 | 6 | 1 | 2 | 1 | | В9 | 3 | 8 | 14 | 3 | 8 | | В10 | 2 | 11 | 15 | 6 | 11 | | В11 | 8 | 13 | 13 | 14 | 5 | | В12 | 9 | 1 | 11 | 5 | 13 | | В13 | 13 | 2 | 12 | 4 | 2 | | В14 | 10 | 4 | 5 | 12 | 12 | | В15 | 11 | 5 | 2 | 13 | 3 | | **Вектор медиан рангов:**  [7.0, 5.0, 9.0, 10.0, 7.0, 9.0, 10.0, 2.0, 8.0, 11.0, 13.0, 9.0, 4.0, 10.0, 5.0]  **Вектор коэф. компетентности:** [0.16879937514245805, 0.2129361568294439, 0.1897366090689446, 0.2225655937289272, 0.2059622652302263]  **Вектор средневз. рангов:** [9.071858000912917, 6.5270303720679355, 8.698774720667751, 10.422706315644087, 8.118431443604855, 9.504928186931059, 8.571433530449616, 2.7936445033035207, 7.181594810056742, 9.126924091349021, 10.730870596174826, 7.609570649507534, 6.199290404714339, 8.63075573159692, 6.8121866430188796] |  |

*Таблица 8 - Результаты тестирования программы бинарных отношений*

|  |  |
| --- | --- |
| Входные данные | Результат |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 0 | | 0 | 1 | 0 | 1 |  | 0 | 1 | 1 | 0 | | 1 | 1 | 1 | 0 |  | 1 | 0 | 1 | 1 | | 0 | 0 | 1 | 1 |  | 1 | 1 | 0 | 1 | |  |  |  |  |  |  |  |  |  | | 1 | 0 | 0 | 1 |  | 1 | 1 | 0 | 0 | | 1 | 1 | 1 | 0 |  | 0 | 1 | 1 | 0 | | 1 | 0 | 1 | 1 |  | 1 | 0 | 1 | 0 | | 0 | 1 | 0 | 1 |  | 1 | 1 | 1 | 1 | |  |  |  |  |  |  |  |  |  | | 1 | 1 | 0 | 1 |  |  |  |  |  | | 0 | 1 | 0 | 1 |  |  |  |  |  | | 1 | 1 | 1 | 0 |  |  |  |  |  | | 0 | 0 | 1 | 1 |  |  |  |  |  | | **Медиана Кемени:**   |  |  |  |  | | --- | --- | --- | --- | | 1 | 1 | 0 | 1 | | 0 | 1 | 1 | 0 | | 1 | 0 | 1 | 0 | | 0 | 1 | 1 | 1 |   **Расстояния:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 4 | 4 | 4 | 2 | 4 |   **Целевая функция:**  18  **Сумма (ранги):**   |  |  |  |  | | --- | --- | --- | --- | | 2 | 3 | 3 | 2 | |

Также был проведен опрос среди экспертов данной области, результаты были собраны с помощью анкетирования в Google Forms. По итогам опроса 30 экспертов результаты были собраны, обработаны и ниже приведены таблицы результатов решения задач разными методами.

*Таблица 9 – Таблица для тестирования программы средних оценок*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Э1 | Э2 | Э3 | Э4 | Э5 | Э6 | Э7 | Э8 | Э9 | Э10 | Э11 | Э12 | Э13 | Э14 | Э15 | Э16 | Э17 | Э18 | Э19 | Э20 | Э21 | Э22 | Э23 | Э24 | Э25 | Э27 | Э28 | Э29 | Э30 |
| В1 | 6 | 7 | 5 | 6 | 7 | 7 | 9 | 8 | 7 | 9 | 6 | 9 | 5 | 7 | 8 | 10 | 10 | 6 | 1 | 10 | 7 | 2 | 3 | 5 | 10 | 10 | 7 | 6 | 5 |
| В2 | 8 | 5 | 9 | 5 | 8 | 5 | 8 | 4 | 9 | 6 | 5 | 8 | 7 | 4 | 3 | 7 | 7 | 8 | 10 | 10 | 6 | 4 | 4 | 7 | 10 | 8 | 2 | 6 | 4 |
| В3 | 6 | 9 | 10 | 8 | 8 | 8 | 2 | 6 | 6 | 7 | 4 | 8 | 9 | 6 | 8 | 6 | 6 | 5 | 10 | 10 | 4 | 7 | 6 | 9 | 9 | 9 | 4 | 8 | 9 |
| В4 | 7 | 10 | 7 | 5 | 5 | 9 | 6 | 9 | 5 | 2 | 8 | 7 | 8 | 6 | 3 | 8 | 8 | 6 | 5 | 5 | 3 | 3 | 8 | 8 | 5 | 7 | 8 | 7 | 8 |
| В5 | 8 | 10 | 9 | 4 | 7 | 5 | 5 | 7 | 8 | 4 | 8 | 8 | 9 | 7 | 6 | 5 | 5 | 9 | 10 | 10 | 5 | 6 | 6 | 8 | 10 | 3 | 8 | 8 | 9 |
| В6 | 9 | 7 | 4 | 3 | 8 | 10 | 8 | 3 | 8 | 3 | 5 | 6 | 4 | 2 | 6 | 4 | 4 | 5 | 5 | 1 | 4 | 5 | 5 | 8 | 6 | 2 | 10 | 9 | 9 |
| В7 | 9 | 9 | 3 | 8 | 3 | 6 | 6 | 2 | 2 | 5 | 4 | 5 | 7 | 7 | 10 | 9 | 9 | 6 | 5 | 5 | 4 | 5 | 6 | 2 | 8 | 6 | 6 | 7 | 10 |
| В8 | 10 | 10 | 7 | 4 | 6 | 7 | 7 | 7 | 3 | 10 | 8 | 7 | 9 | 9 | 7 | 7 | 7 | 7 | 10 | 1 | 5 | 8 | 5 | 3 | 9 | 9 | 5 | 2 | 8 |
| В9 | 1 | 3 | 7 | 2 | 1 | 6 | 1 | 4 | 4 | 1 | 3 | 6 | 3 | 1 | 5 | 5 | 5 | 5 | 1 | 1 | 2 | 2 | 10 | 4 | 5 | 7 | 8 | 4 | 8 |
| В10 | 8 | 8 | 4 | 7 | 5 | 5 | 10 | 9 | 9 | 8 | 10 | 9 | 9 | 5 | 8 | 10 | 10 | 7 | 5 | 10 | 7 | 6 | 10 | 9 | 10 | 3 | 8 | 3 | 5 |

*Таблица 10 – Результаты тестирования программы средних оценок*

|  |  |
| --- | --- |
| Результат | Гистограмма |
| **Вектор мат. ожиданий:**  [6.933333333333334, 6.5, 7.1, 6.366666666666666, 7.166666666666667, 5.633333333333334, 5.866666666666666, 6.766666666666667, 4.0, 7.5]  **Вектор дисперсий:**  [5.719540229885058, 4.810344827586207, 4.162068965517242, 4.033333333333333, 4.005747126436781, 6.240229885057471, 5.9816091954022985, 5.840229885057471, 6.137931034482759, 4.948275862068965]  **Вектор среднеквадратичных отклонений:**  [2.3915560269174247, 2.1932498324600886, 2.0401149392907354, 2.008316044185609, 2.0014362658942657, 2.4980452127728734, 2.4457328544635244, 2.4166567578076683, 2.4774848202325597, 2.2244720411974086]  **Вектор доверительных интервалов:**  [[2.150221279498484, 11.716445387168182], [2.1135003350798227, 10.886499664920176], [3.019770121418529, 11.180229878581471], [2.350034578295448, 10.383298755037885], [3.1637941348781355, 11.169539198455197], [0.637242907787587, 10.629423758879081], [0.9752009577396175, 10.758132375593714], [1.93335315105133, 11.599980182282003], [-0.9549696404651193, 8.95496964046512], [3.051055917605183, 11.948944082394817]] |  |

*Таблица 11 – Таблица для тестирования программы медианы рангов*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Э1 | Э2 | Э3 | Э4 | Э5 | Э6 | Э7 | Э8 | Э9 | Э10 | Э11 | Э12 | Э13 | Э14 | Э15 | Э16 | Э17 | Э18 | Э19 | Э20 | Э21 | Э22 | Э23 | Э24 | Э25 | Э26 | Э27 | Э28 | Э29 | Э30 |
| В1 | 14 | 7 | 10 | 7 | 8 | 2 | 11 | 3 | 14 | 7 | 5 | 2 | 8 | 6 | 2 | 9 | 7 | 9 | 6 | 4 | 9 | 2 | 7 | 8 | 7 | 6 | 5 | 5 | 7 | 9 |
| В2 | 5 | 3 | 13 | 4 | 7 | 15 | 10 | 14 | 7 | 10 | 10 | 15 | 7 | 13 | 1 | 11 | 5 | 6 | 12 | 6 | 8 | 1 | 5 | 7 | 8 | 3 | 4 | 7 | 3 | 11 |
| В3 | 12 | 9 | 14 | 8 | 6 | 14 | 9 | 5 | 6 | 6 | 9 | 14 | 9 | 12 | 3 | 10 | 8 | 8 | 14 | 3 | 10 | 3 | 9 | 9 | 10 | 9 | 3 | 9 | 9 | 13 |
| В4 | 15 | 10 | 12 | 3 | 5 | 13 | 15 | 8 | 8 | 9 | 8 | 7 | 10 | 14 | 15 | 7 | 3 | 13 | 13 | 14 | 11 | 13 | 14 | 10 | 9 | 12 | 15 | 11 | 11 | 7 |
| В5 | 7 | 15 | 15 | 6 | 9 | 12 | 8 | 6 | 9 | 4 | 13 | 12 | 11 | 7 | 13 | 12 | 10 | 11 | 10 | 13 | 12 | 15 | 15 | 14 | 12 | 15 | 13 | 13 | 15 | 12 |
| В6 | 1 | 14 | 11 | 9 | 4 | 11 | 7 | 12 | 4 | 15 | 12 | 11 | 13 | 8 | 14 | 8 | 11 | 10 | 11 | 15 | 7 | 14 | 11 | 13 | 11 | 14 | 14 | 10 | 14 | 8 |
| В7 | 6 | 12 | 9 | 10 | 3 | 10 | 1 | 10 | 5 | 14 | 4 | 8 | 4 | 15 | 4 | 15 | 12 | 15 | 15 | 2 | 15 | 4 | 12 | 11 | 15 | 10 | 1 | 4 | 12 | 15 |
| В8 | 4 | 6 | 7 | 1 | 2 | 9 | 2 | 13 | 10 | 1 | 11 | 9 | 3 | 11 | 12 | 14 | 2 | 3 | 9 | 12 | 14 | 10 | 6 | 3 | 14 | 7 | 11 | 15 | 6 | 14 |
| В9 | 3 | 8 | 5 | 14 | 1 | 7 | 3 | 15 | 11 | 8 | 7 | 13 | 6 | 4 | 5 | 13 | 1 | 7 | 7 | 5 | 13 | 5 | 8 | 6 | 13 | 8 | 2 | 6 | 8 | 10 |
| В10 | 2 | 11 | 8 | 15 | 10 | 8 | 6 | 7 | 12 | 11 | 3 | 10 | 14 | 5 | 11 | 6 | 4 | 14 | 8 | 11 | 6 | 11 | 10 | 4 | 6 | 11 | 12 | 14 | 10 | 3 |
| В11 | 8 | 13 | 4 | 13 | 11 | 6 | 14 | 2 | 13 | 5 | 14 | 6 | 5 | 10 | 6 | 5 | 14 | 5 | 2 | 1 | 3 | 6 | 13 | 5 | 5 | 13 | 6 | 8 | 13 | 5 |
| В12 | 9 | 1 | 3 | 11 | 12 | 5 | 5 | 11 | 3 | 13 | 1 | 3 | 12 | 3 | 7 | 4 | 15 | 12 | 5 | 7 | 4 | 7 | 2 | 12 | 4 | 1 | 7 | 12 | 1 | 4 |
| В13 | 13 | 2 | 1 | 12 | 13 | 1 | 4 | 9 | 1 | 2 | 2 | 1 | 1 | 1 | 10 | 3 | 6 | 1 | 3 | 10 | 5 | 12 | 1 | 1 | 3 | 2 | 10 | 1 | 5 | 6 |
| В14 | 10 | 4 | 2 | 5 | 15 | 4 | 12 | 1 | 2 | 12 | 6 | 4 | 2 | 2 | 8 | 2 | 9 | 2 | 4 | 8 | 1 | 8 | 4 | 2 | 2 | 4 | 8 | 2 | 4 | 2 |
| В15 | 11 | 5 | 6 | 2 | 14 | 3 | 13 | 4 | 15 | 3 | 15 | 5 | 15 | 9 | 9 | 1 | 13 | 4 | 1 | 9 | 2 | 9 | 3 | 15 | 1 | 5 | 9 | 3 | 2 | 1 |

*Таблица 12 –Результаты тестирования программы медианы рангов*

|  |  |
| --- | --- |
| Результат | Гистограмма |
| **Вектор медиан рангов:**  [7.0, 7.0, 9.0, 11.0, 12.0, 11.0, 10.0, 9.0, 7.0, 10.0, 6.0, 5.0, 3.0, 4.0, 5.0]  **Вектор коэф. компетентности:**  [0.016813002664555737, 0.050326030347310285, 0.0488372603044316, 0.01908238971905128, 0.01475738431642983, 0.045616025278780926, 0.021165898615482594, 0.02368866145914088, 0.023854284262117983, 0.023705120173243815, 0.029498465263856452, 0.03655744677232142, 0.0317002296148365, 0.03575295734576628, 0.029203887409207638, 0.03296626487348905, 0.020117553258696477, 0.039751993548335, 0.041076085099874766, 0.028028321314783784, 0.03431460850020827, 0.02800531480360379, 0.054275611871694864, 0.034923431807121263, 0.03893543069347793, 0.05345361200524218, 0.027989998100735037, 0.03798185096964731, 0.047020025138061965, 0.030600854468495055]  **Вектор средневз. рангов:**  [6.675518608161541, 7.748893409591036, 9.206729610861162, 10.870054387891681, 11.964867781498906, 11.027172229754017, 9.72346792101582, 8.300418988683502, 7.614599451565431, 8.933695602864757, 7.796325204643786, 5.744113664213629, 3.87868560477552, 4.369135641177958, 6.146321893301255] |  |

*Таблица 13 – Таблица для тестирования метода бинарных сравнений*

|  |
| --- |
| Входные данные |
|  |

|  |
| --- |
| Результат |
| **Медиана Кемени:**   |  |  |  |  | | --- | --- | --- | --- | | 1 | 0 | 1 | 0 | | 1 | 1 | 1 | 1 | | 0 | 0 | 1 | 0 | | 1 | 0 | 0 | 1 |   **Расстояния:**   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 6 | 6 | 6 | 6 | 2 | 6 | 2 | 2 | 2 | | 6 | 2 | 6 | 2 | 2 | 6 | 6 | 6 | 6 | 6 | | 6 | 2 | 6 | 6 | 6 | 6 | 2 | 2 | 6 | 2 |   **Целевая функция:**  132  **Сумма (ранги):**   |  |  |  |  | | --- | --- | --- | --- | | 3 | 1 | 3 | 3 | |

На основании этого тестирования мы можем сделать следующие выводы:

**ДЛЯ ТЕСТИРОВАНИЯ НА ПЯТИ СЛУЧАЙНЫХ ЗНАЧЕНИЯХ**

1. Методом оценок было выявлено топ-3 самых важных причин безработицы молодых специалистов:
2. Проблема безработицы молодых специалистов связана со спецификой психологического склада молодежи (самонадеянность, инфантильность, упрямость и др.)
3. Проблема безработицы молодых специалистов связана с несоответствием программ подготовки в высших и средне-профессиональных учебных заведений с реальными профессиональными требованиями, предъявляемые работодателями на рынке труда
4. Разделили сразу две проблемы: Проблема безработицы молодых специалистов связана со спадом в экономике; Проблема безработицы молодых специалистов связана с недостатком знаний, необходимых для получения места

2. Методом ранговых оценок было установлено, что самое «болезненное» последствие для экономики будет иметь: «налоговые поступления сокращаются, тем самым затрудняется развитие тех сфер экономики, которые наиболее зависят от государственного субсидирования.», а самое незначительное: «Усиление предпосылок для социальной нестабильности, влияющей на политическую ситуацию в стране (протесты, митинги)»

3. Методом бинарных сравнений было установлено, что программа повышения квалификации и профессиональной подготовки кадров и программа развития малого бизнеса являются более эффективными в борьбе с безработицей молодых специалистов.

**ДЛЯ ТЕСТИРОВАНИЯ НА РЕЗУЛЬТАТАХ ОПРОСА 30 ЭКСПЕРТОВ ДАННОЙ ОБЛАСТИ**

1. Методом оценок было выявлено топ-3 самых важных причин безработицы молодых специалистов:

1. Проблема безработицы молодых специалистов связана с ошибочным выбором сферы деятельности, который был сделан при поступлении в высшее или средне-профессиональное учебное заведение
2. Проблема безработицы молодых специалистов связана с их нежеланием начинать путь по карьерной лестнице с должностей с небольшим заработком
3. Разделили сразу три проблемы: Проблема безработицы молодых специалистов связана с завышенными требованиями работодателей; Проблема безработицы молодых специалистов связана со спадом в экономике; Проблема безработицы молодых специалистов связана с несоответствием программ подготовки в высших и средне-профессиональных учебных заведений с реальными профессиональными требованиями, предъявляемые работодателями на рынке труда

2. Методом ранговых оценок было установлено, что самое «болезненное» последствие для экономики будет иметь: «Снижение валового внутреннего продукта в силу недоиспользования в общественном воспроизводстве трудового потенциала», а самое незначительное: «Налоговые поступления сокращаются, тем самым затрудняется развитие тех сфер экономики, которые наиболее зависят от государственного субсидирования.»

3. Методом бинарных сравнений было установлено, что только одна программа не является эффективной: программа повышения

На основе сравнения двух тестирований можно сделать вывод, что, увеличивая количество респондентов, которые будут участвовать в вопросе, результаты изменятся. Также нужно брать во внимание, что значения в первом случае заполнены были случайно, поэтому результаты могут быть самыми не предсказуемыми, так как эксперты в области в некоторых случаях отвечали по-разному. Логическим объяснением к результату бинарных сравнений является то, что все эти меры являются эффективными, но применяются при разных экономических ситуациях в стране, когда как программа повышения квалификации и профессиональной подготовки кадров не является эффективной для данной категории граждан, так как молодые специалисты, в силу своих психологических особенностей и социального положения, не будут готовы к данной мере.

# **7. Заключение**

В данной проектной документации решалась задача экспертного принятия решения путем применения различных технологий таких, как метод средних арифметических баллов, метод медианы рангов и метод бинарных отношений. Была разработана математическая модель для решения типовых задач, на основании которой затем была разработана программа средствами Python для метода средних рангов и метода медианных рангов, а также решение задачи методом бинарных отношений средствами MS Excel с учетом алгоритмов решения приведенных в документации технологий, были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат.

# **СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ**

# **1. Физическая модель задачи**

В кондитерском магазине имеются 4 кассы. В среднем для обслуживания одного клиента на кассе требуется 2 минуты. Одновременно в очередь на кассы могут вставать не больше 7 человек. Определить показатели СМО кондитерского магазина согласно следующим наблюдениям магазина:

• в утренние часы с 9 до 12 часов в магазине побывало 120 покупателей,

• в обеденные часы с 12 до 15 часов – 250 покупателей,

• днем с 15 до 18 часов - 150 покупателей

• в вечернее с 18 до 21 часа - 300 покупателей.

# **2. Математическая модель задачи**

Имеется некоторая система массового обслуживания, для которой определен входящий поток нуждающихся в обслуживании требований, одно или несколько обслуживающих устройств и время обслуживания требования каждым прибором.

Для анализа поступающих заявок вводится понятие интенсивности входного потока заявок - скорость поступления заявок в систему – за (лямбда). Где количество заявок, пришедшее за определенный период времени t будет равняться N, а - средний интервал времени между поступлением заявок:

Для анализа обслуженных заявок обозначается интенсивность выходного потока от одного устройства – скорость обработки обслуженных заявок. Где количество заявок - N, которое обработали все n устройств за определенный период времени t, а - среднее время на обработку одним прибором одной заявки:

Для определения состояний системы, которые отражают количество заявок на обслуживании и в очереди, вводится понятие вероятности – показатель нагруженности системы. Будут рассматриваться следующие вероятности нахождения системы в каждом из состояний:

* Вероятность того, что система бездействует:
* Вероятность обслуживания:
* Вероятность нахождения системы в состоянии очереди:

Где m – максимальная длина очереди

Анализ показателя нагруженности:

* – недогруженная система, выгодная без очередей для клиента системы и невыгодная - для ее владельца.
* – сбалансированная система для клиента с приемлемыми очередями.
* – сбалансированная система для владельца с практические полностью занятыми обслуживающими устройствами.
* – перегруженная система с невыгодным большим количеством очередей для клиента системы и с выгодными полностью занятыми обслуживающими устройствами – для владельца.

Показатели СМО для клиента.

Существуют косвенные характеристики системы, которые оцениваются в пользу клиента: вероятность отказа, вероятность встать в очередь, средняя длина очереди и среднее время ожидания в очереди.

* Вероятность получить отказ в обслуживании клиент может в случае, если вся очередь заполнена и все устройства заняты:
* Вероятность заявки клиента встать в очередь, когда все устройства заняты, но есть место в очереди (сумма вероятностей очередей):
* Можно определить среднее количество заявок, которые находятся в очереди – среднюю длину очереди:
* Также по формуле Литтла можно найти среднее время ожидания клиента в очереди:

Характеристики СМО для владельца системы:

* Чтобы определить скорость обслуживания заявок за единицу времени, рассчитывается абсолютная пропускная способность:
* Для поиска процента обслуженных заявок находится относительная пропускная способность (обрабатываемые и отказываемые заявки):
* Чтобы обозначить число занятых обслуживанием устройств, рассчитывается среднее количество занятых устройств:
* Кроме того для нахождения процента времени простоя устройств обслуживания вычисляется коэффициент простоя:

# **3. Алгоритмы решения задачи**

Алгоритм решения разработан в двух средах Python и Excel.

### **3.1. Описание входных данных**

а) количество касс (int);

б) время обработки (float);

в) максимальная длина очереди (int);

г) количество интервалов (int);

д) длина интервала (float);

е) массив данных о количестве клиентов в каждом интервале.

**3.2. Описание алгоритма решения**

1. Вычисление интенсивности входного и выходного потока. Алгоритм на данном шаге производит расчет значения путем деления количества клиентов в каждом интервале на длину интервала.

2. Далее алгоритм высчитывает показатель нагруженности путем деления полученного значения интенсивности входного интервала на показатель интенсивности выходного интервала, рассчитанного как деление единицы на время обработки и умножения полученного значения на количество временных интервалов.

3. Далее алгоритм производит расчет вероятности нахождения системы в каждом из состояний для каждого временного интервала работы системы. Для нахождения значений обслуживания алгоритм находит значение за счет деления показателя нагруженности, возведенного в степень соответствующего количественного индекса, на факториал данного индекса. Для нахождения значений в очереди алгоритм производит деление показателя нагруженности, возведенного в степень соответствующего количественного индекса k , на количество касс (n) в степени k-n, умноженное на факториал количества касс n. При использовании алгоритма в Excel после нахождения матрицы выполняется построение гистограмм.

4. Расчет вероятностей отказа в обслуживании (равна значению последнего элемента в матрице состояний) и вставания в очередь (равна сумме значений в очереди без последнего элемента);

5. Далее производится расчет средней длины очереди и среднего времени ожидания в очереди по формуле Литтла.

6. Отражение характеристик для владельца. Расчет абсолютной пропускной способности – скорости обслуживания заявок (количество обработанных заявок в единицу времени), относительной пропускной способности – процента обслуженных заявок), а также среднее количество занятых касс, получаемое из деления показателя абсолютной пропускной способности на значение выходного потока, и коэффициента простоя – процента времени простоя обслуживающих касс, находящееся путем вычисления из единица полученного показателя занятых касс, деленного на их количество.

### **3.3. Описание выходных данных**

а) Интенсивность входного (лямбда) и выходного (мю) потока заявок;

б) Показатель нагруженности;

в) Матрица вероятностей состояния системы, ее гистограмма (визуализация выполнена по средствам Excel);

г) Вероятность отказа в обслуживании;

д) Вероятность встать в очередь;

е) Средняя длина очереди;

ж) Среднее время ожидания в очереди;

з) Показатель абсолютной пропускной способности;

и) Показатель относительной пропускной способности;

к) Среднее количество занятых приборов;

л) Коэффициент простоя.

# **4. Варианты использования системы**

Для пользователя предусмотрены два варианта использования системы с использованием средств Python и средствами MS Excel. Пользователь может выбрать любой удобный для него вариант.

## **4.1. ВИ 1**

1. Пользователь должен запустить среду разработки для Python.
2. Пользователь должен запустить код для решения задачи.
   1. Пользователю необходимо ввести следующие входные данные:
      1. Ввести количество касс: целое число
      2. Ввести время обработки: целое или дробное число.
      3. Ввести максимальную длину очереди: целое число.
      4. Ввести количество интервалов: целое число
      5. Ввести длину интервала: целое или дробное число.
      6. Ввести количество клиентов в каждом интервале:

* целые числа;
* через пробел;

1. Программа выполняет алгоритм.
2. Пользователю должен выводиться результат в следующем виде:

* Лямбда (интенсивность входного потока заявок);
* Мю (интенсивность выходного потока заявок);
* Показатель нагруженности;
* Матрица вероятностей состояния системы;
* Вероятности отказа в обслуживании;
* Вероятности встать в очередь;
* Средняя длина очереди;
* Среднее время ожидания в очереди;
* Абсолютная пропускная способность;
* Относительная пропускная способность;
* Среднее количество занятых приборов;
* Коэффициент простоя.

## **4.2. ВИ 2\_Визуализация\_решения**

1. Пользователь открывает приложение MS Excel.
2. Пользователь вручную следует разработанному алгоритму (см. глава 3.3.)
3. Пользователь получает:

* Лямбда (интенсивность входного потока заявок);
* Мю (интенсивность выходного потока заявок);
* Показатель нагруженности;
* Матрица вероятностей состояния системы;
* Вероятности отказа в обслуживании;
* Вероятности встать в очередь;
* Средняя длина очереди;
* Среднее время ожидания в очереди;
* Абсолютная пропускная способность;
* Относительная пропускная способность;
* Среднее количество занятых приборов;
* Коэффициент простоя;
* График, отражающий значения полученной матрицы вероятностей состояния системы:

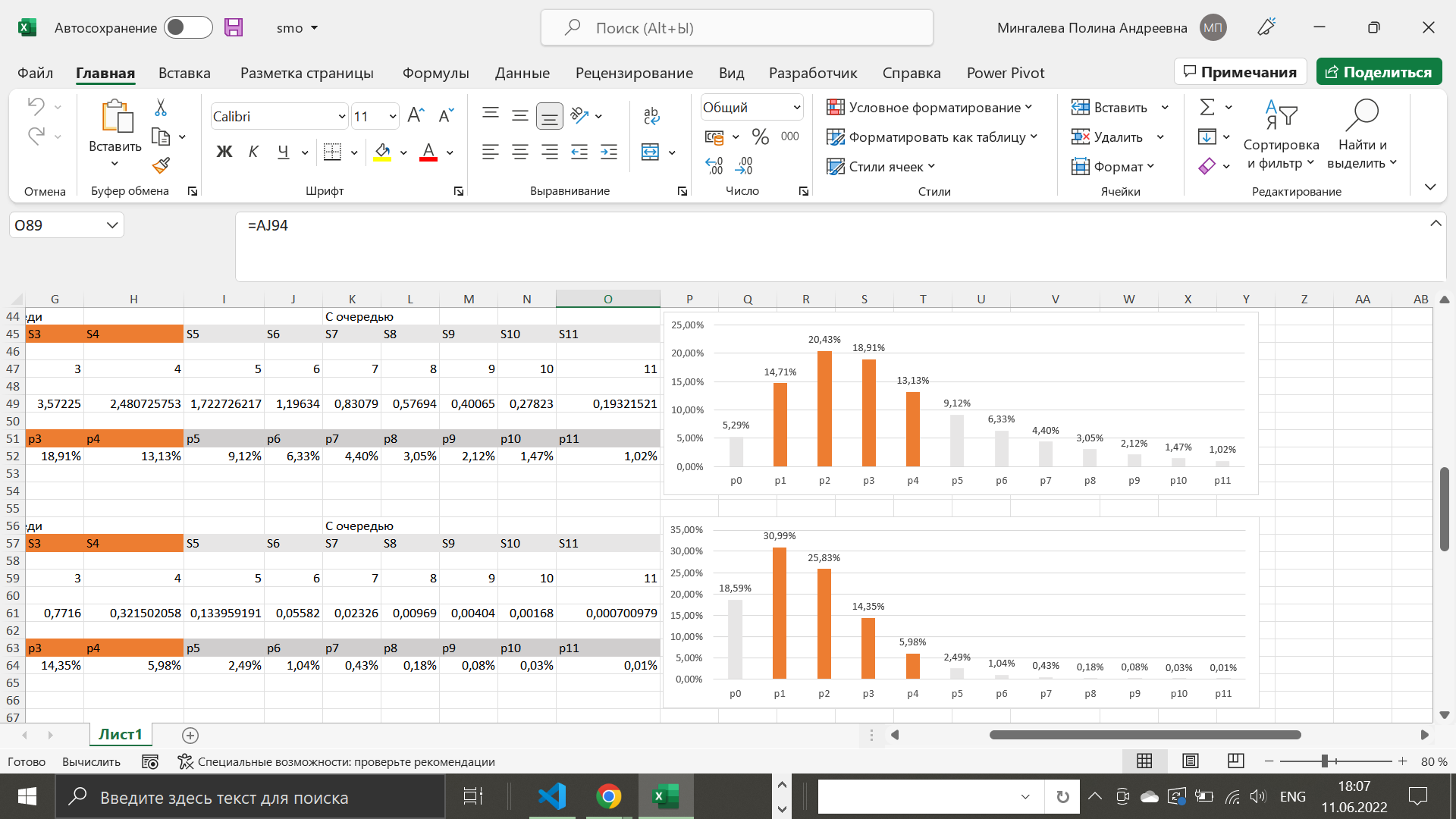


Рисунок 1. Пример гистограмм по матрицам вероятности состояний системы.

# **5. Архитектура решения**

## **5.1. Функции считывания информации**

**def** **manual\_input**():

Функция ручного ввода матрицы. Осуществляется ручной ввод количества касс, времени обработки, максимальной длины очереди, количество интервалов и их длины, а также массива данных о количестве клиентов в каждом интервале.

## **5.2. Функции обработки информации**

Функция обработки информации включает в себя нахождения интенсивности входного и выходного потока, расчет вероятностей нахождения системы в каждом из состояний для каждого временного интервала работы системы, вычисления параметров системы, необходимых клиенту: вероятностей отказа в обслуживании и вставания в очередь, средней длины очереди и среднего времени ожидания, вычисление параметров, необходимых владельцу: абсолютной и относительной пропускной способности, среднего числа занятых касс и коэффициента простоя.

## **5.3. Функции вывода информации**

Вывод информации осуществляется с помощью функции print :

**print** ('Лямбда:', \*lambd)

**print** ('Мю:', \*mu)

**print** ('Показатель нагруженности: ', \*ro)

**print** ('--------------------------')

**print** ('Матрица вероятностей состояния системы:')

**for** i **in** p\_obsl\_1:

**print** (\*i)

**print** ('Вероятности отказа в обслуживании: ', \*p\_obsl\_1[n + m])

**print** ('Вероятности встать в очередь: ', \*sum\_ocher)

**print** ('Средняя длина очереди: ', \*l\_och)

**print** ('Среднее время ожидания в очереди: ', \*t\_och)

**print** ('Абсолютная пропускная способность: ', \*A)

**print** ('Относительная пропускная способность: ', \*Q)

**print** ('Среднее количество занятых приборов: ', \*n\_zan)

**print** ('Коэффициент простоя: ', \*K\_pr)

**print** ('Вектор средневз. рангов: ', rangi (matrix)[**2**])

# 

# **6. Тестирование**

|  |  |
| --- | --- |
| **Входные данные** | **Результаты** |
| Введите количество касс: 3  Введите время обработки: 1.5  Введите максимальную длину очереди: 9  Введите количество интервалов: 4  Введите длину интервала: 180  Введите количество клиентов в каждом интервале через пробел: 50 300 150 500 | Лямбда: 0.28 1.67 0.83 2.78  Мю: 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666  Показатель нагруженности: 0.42000000000000004 2.505 1.245 4.17  --------------------------  Матрица вероятностей состояния системы:  0.66 0.28 0.06 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0.06 0.14 0.18 0.15 0.13 0.11 0.09 0.07 0.06 0.05 0.04 0.04 0.03  0.31 0.38 0.24 0.1 0.04 0.02 0.01 0.0 0.0 0.0 0.0 0.0 0.0  0.0 0.01 0.01 0.02 0.02 0.03 0.04 0.06 0.08 0.11 0.15 0.21 0.29  --------------------------  Вероятности отказа в обслуживании: 0.0 0.03 0.0 0.29  Вероятности встать в очередь: 0.0 0.5900000000000001 0.06999999999999999 0.7  Средняя длина очереди: 0.0 2.37 0.12 6.81  Среднее время ожидания в очереди: 0.01 1.46 0.14 3.45  --------------------------  Абсолютная пропускная способность: 0.28 1.62 0.83 1.97  Относительная пропускная способность: 1.0 0.97 1.0 0.71  Среднее количество занятых приборов: 0.42 2.43 1.25 2.96  Коэффициент простоя: 0.86 0.19 0.58 0.01 |
| Введите количество касс: 5  Введите время обработки: 1.5  Введите максимальную длину очереди: 9  Введите количество интервалов: 4  Введите длину интервала: 180  Введите количество клиентов в каждом интервале через пробел: 50 300 150 500 | Лямбда: 0.28 1.67 0.83 2.78  Мю: 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666  Показатель нагруженности: 0.42000000000000004 2.505 1.245 4.17  --------------------------  Матрица вероятностей состояния системы:  0.66 0.28 0.06 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0.09 0.21 0.27 0.22 0.14 0.07 0.04 0.02 0.01 0.0 0.0 0.0 0.0 0.0 0.0  0.29 0.36 0.22 0.09 0.03 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0.01 0.05 0.11 0.15 0.16 0.13 0.11 0.09 0.08 0.06 0.05 0.04 0.04 0.03 0.03  --------------------------  Вероятности отказа в обслуживании: 0.0 0.0 0.0 0.03  Вероятности встать в очередь: 0.0 0.06999999999999999 0.0 0.5  Средняя длина очереди: 0.0 0.14 0.0 2.01  Среднее время ожидания в очереди: 0.0 0.08 0.0 0.75  --------------------------  Абсолютная пропускная способность: 0.28 1.67 0.83 2.7  Относительная пропускная способность: 1.0 1.0 1.0 0.97  Среднее количество занятых приборов: 0.42 2.5 1.25 4.04  Коэффициент простоя: 0.92 0.5 0.75 0.19 |
| Введите количество касс: 3  Введите время обработки: 1  Введите максимальную длину очереди: 9  Введите количество интервалов: 4  Введите длину интервала: 180  Введите количество клиентов в каждом интервале через пробел: 50 300 150 500 | Лямбда: 0.28 1.67 0.83 2.78  Мю: 1.0 1.0 1.0 1.0  Показатель нагруженности: 0.28 1.67 0.83 2.78  --------------------------  Матрица вероятностей состояния системы:  0.76 0.21 0.03 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0.2 0.33 0.28 0.15 0.09 0.05 0.03 0.01 0.01 0.0 0.0 0.0 0.0  0.45 0.38 0.16 0.04 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0.03 0.09 0.13 0.12 0.11 0.1 0.09 0.09 0.08 0.08 0.07 0.06 0.06  --------------------------  Вероятности отказа в обслуживании: 0.0 0.0 0.0 0.06  Вероятности встать в очередь: 0.0 0.19000000000000003 0.01 0.6800000000000002  Средняя длина очереди: 0.0 0.43 0.02 3.35  Среднее время ожидания в очереди: 0.0 0.25 0.03 1.28  --------------------------  Абсолютная пропускная способность: 0.28 1.67 0.83 2.61  Относительная пропускная способность: 1.0 1.0 1.0 0.94  Среднее количество занятых приборов: 0.28 1.67 0.83 2.61  Коэффициент простоя: 0.91 0.44 0.72 0.13 |
| Введите количество касс: 2  Введите время обработки: 1.5  Введите максимальную длину очереди: 9  Введите количество интервалов: 4  Введите длину интервала: 180  Введите количество клиентов в каждом интервале через пробел: 50 300 150 500 | Лямбда: 0.28 1.67 0.83 2.78  Мю: 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666  Показатель нагруженности: 0.42000000000000004 2.505 1.245 4.17  --------------------------  Матрица вероятностей состояния системы:  0.69 0.29 0.06 0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.09 0.11 0.14 0.18 0.22  0.29 0.36 0.22 0.14 0.09 0.05 0.03 0.02 0.01 0.01 0.0 0.0  0.0 0.0 0.0 0.0 0.0 0.01 0.01 0.03 0.06 0.12 0.25 0.52  --------------------------  Вероятности отказа в обслуживании: 0.0 0.22 0.0 0.52  Вероятности встать в очередь: 0.01 0.74 0.3500000000000001 0.48  Средняя длина очереди: 0.02 6.19 0.91 8.09  Среднее время ожидания в очереди: 0.07 4.76 1.09 6.06  --------------------------  Абсолютная пропускная способность: 0.28 1.3 0.83 1.33  Относительная пропускная способность: 1.0 0.78 1.0 0.48  Среднее количество занятых приборов: 0.42 1.95 1.25 2.0  Коэффициент простоя: 0.79 0.02 0.38 -0.0 |
| Введите количество касс: 3  Введите время обработки: 1.5  Введите максимальную длину очереди: 5  Введите количество интервалов: 4  Введите длину интервала: 180  Введите количество клиентов в каждом интервале через пробел: 50 300 150 500 | Лямбда: 0.28 1.67 0.83 2.78  Мю: 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666  Показатель нагруженности: 0.42000000000000004 2.505 1.245 4.17  --------------------------  Матрица вероятностей состояния системы:  0.66 0.28 0.06 0.01 0.0 0.0 0.0 0.0 0.0  0.07 0.17 0.22 0.18 0.15 0.13 0.11 0.09 0.07  0.31 0.38 0.24 0.1 0.04 0.02 0.01 0.0 0.0  0.01 0.02 0.04 0.06 0.09 0.12 0.17 0.23 0.32  --------------------------  Вероятности отказа в обслуживании: 0.0 0.07 0.0 0.32  Вероятности встать в очередь: 0.0 0.48 0.06999999999999999 0.61  Средняя длина очереди: 0.0 1.43 0.11 3.37  Среднее время ожидания в очереди: 0.01 0.92 0.14 1.78  --------------------------  Абсолютная пропускная способность: 0.28 1.55 0.83 1.89  Относительная пропускная способность: 1.0 0.93 1.0 0.68  Среднее количество занятых приборов: 0.42 2.33 1.25 2.84  Коэффициент простоя: 0.86 0.22 0.58 0.05 |

# **7. Заключение**

В данной проектной документации решалась задача оптимизирования системы массового обслуживания. Была разработана математическая модель для решения типовых задач, на основании которой затем была разработана программа средствами Python, а также была представления визуализация задачи средствами MS Excel с учетом алгоритмов решения приведенных в документации технологий, были приведены решения задач в конкретном и общем виде с возможностью менять условия и получать новый результат.

Системы массовых обслуживаний применимы во многих областях практической деятельности человека, где необходимо многоразовое выполнение однотипных задач. Главная цель анализа таких систем заключается в выработке рекомендаций по оптимизации и рационализации их выстраивания, а также эффективном регулировании поступающих заявок и оперативной организации работы по их обслуживанию. В настоящее время область применения СМО только расширяется: оно выходит за рамки стандартных задач обслуживания.