**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 по темам:**

«Производственная задача линейного программирования», «Оптимизация на графах», «Теория игр»

Студентка группы БИ 20-8:

Саликова Таисия Александровна

Руководитель:

Аксенов Дмитрий Андреевич

**Москва 2022**

Оглавление

[1. «Производственная задача линейного программирования» 5](#_Toc100351945)

[1.1. Постановка задачи (физическая модель) 5](#_Toc100351946)

[1.2. Математическая модель 5](#_Toc100351947)

[1.3. Алгоритм решения 6](#_Toc100351948)

[1.3.1. Алгоритм 1. Решение с помощью Поиска решения в Excel 6](#_Toc100351949)

[1.3.1.1. Описание входных данных 6](#_Toc100351950)

[1.3.1.2. Описание алгоритма решения 7](#_Toc100351951)

[1.3.1.3. Описание выходных данных 9](#_Toc100351952)

[1.3.2. Алгоритм 2. Решение с помощью функции linprog. 9](#_Toc100351953)

[1.3.2.1. Описание входных данных 9](#_Toc100351954)

[1.3.2.2. Описание алгоритма решения 10](#_Toc100351955)

[1.3.2.3. Описание выходных данных 11](#_Toc100351956)

[1.4. Варианты использования системы 12](#_Toc100351957)

[1.4.1. Использование алгоритма 1 – поиск решения Excel 12](#_Toc100351958)

[1.4.2. Использование алгоритма 2 – Python 14](#_Toc100351959)

[1.4.2.1. Ввод данных вручную 14](#_Toc100351960)

[1.4.2.2. Импортирование данных из CSV 18](#_Toc100351961)

[1.5. Архитектура решения 20](#_Toc100351962)

[1.5.1. Fronted 20](#_Toc100351963)

[1.5.2. Backend 21](#_Toc100351964)

[1.6. Тестирование 22](#_Toc100351965)

[1.7. Заключение 24](#_Toc100351966)

[2. «Оптимизация на графах» 25](#_Toc100351967)

[2.1. Постановка задачи (физическая модель) 25](#_Toc100351968)

[2.2. Математическая модель 27](#_Toc100351969)

[2.3. Алгоритма решения 29](#_Toc100351970)

[2.3.1. Алгоритм 1. Решение с помощью Поиска решения в Excel задачи о поиске кратчайшего пути 29](#_Toc100351971)

[2.3.1.1. Описание входных данных 29](#_Toc100351972)

[2.3.1.2. Описание алгоритма решения 30](#_Toc100351973)

[2.3.1.3. Описание входных данных 33](#_Toc100351974)

[2.3.2. Алгоритм 2. Решение в Python задачи о кратчайшем пути алгоритма Дейкстры 34](#_Toc100351975)

[2.3.2.1. Описание входных данных 34](#_Toc100351976)

[2.3.2.2. Описание алгоритма решения 35](#_Toc100351977)

[2.3.2.3. Описание выходных данных 39](#_Toc100351978)

[2.3.3. Решение в Python задачи коммивояжера муравьиным алгоритмом 40](#_Toc100351979)

[2.3.3.1. Описание входных данных 40](#_Toc100351980)

[2.3.3.2. Описание алгоритма решения 41](#_Toc100351981)

[2.3.3.3. Описание выходных данных 43](#_Toc100351982)

[2.4. Варианты использования системы 43](#_Toc100351983)

[2.4.1. Использование алгоритма 1 – поиск решения Excel для задачи о поиске кратчайшего пути 43](#_Toc100351984)

[2.4.2. Использование алгоритма 2 – Python для задачи о поиске кратчайшего пути алгоритма Дейкстры 45](#_Toc100351985)

[2.4.2.1. Ввод данных вручную 45](#_Toc100351986)

[2.4.2.2. Импортирование данных из CSV 47](#_Toc100351987)

[2.4.3. Использование алгоритма – Python для задачи коммивояжёра муравьиного алгоритма 47](#_Toc100351988)

[2.4.3.1. Ввод данных вручную 47](#_Toc100351989)

[2.4.3.2. Импортирование файлов из CSV 50](#_Toc100351990)

[2.4.3.3. Случайная генерация вершин 50](#_Toc100351991)

[2.5. Архитектура решения 51](#_Toc100351992)

[2.5.1. Архитектура решения алгоритма Дейкстры 51](#_Toc100351993)

[2.5.2. Архитектуры решения муравьиного алгоритма 53](#_Toc100351994)

[2.6. Тестирование 56](#_Toc100351995)

[2.6.1. Тестирование задачи по поиску кратчайшего пути 56](#_Toc100351996)

[2.6.2. Тестирование задачи по задаче коммивояжёра 59](#_Toc100351997)

[2.7. Заключение 62](#_Toc100351998)

[3. «Теория игр» 63](#_Toc100351999)

[3.1. Постановка задачи (физическая модель) 63](#_Toc100352000)

[3.2. Математическая модель 65](#_Toc100352001)

[3.3. Алгоритм решения 68](#_Toc100352002)

[3.3.1. Алгоритм 1. Решение с помощью Поиска решения в Excel 68](#_Toc100352003)

[3.3.1.1. Решение задачи о нахождении выигрышной стратегии для антагонистической игры 68](#_Toc100352004)

[3.3.1.2. Решение задачи о нахождении выигрышной стратегии для биматричной игры 73](#_Toc100352005)

[3.3.1.3. Решение задачи о принятии оптимального решения в условиях риска 78](#_Toc100352006)

[3.3.1.4. Решение задачи о принятии оптимального решения в условиях неопределённости 85](#_Toc100352007)

[3.3.2. Алгоритм 2. Решения в Python 90](#_Toc100352008)

[3.3.2.1. Решение задачи о нахождении выигрышной стратегии для антагонистической игры 90](#_Toc100352009)

[3.3.2.2. Решение задачи о нахождение выигрышной стратегии для биматричной игры 93](#_Toc100352010)

[3.3.2.3. Решение задачи о принятии оптимального решения в условиях риска 94](#_Toc100352011)

[3.3.2.4. Решение задачи о принятии оптимального решения в условиях неопределённости 97](#_Toc100352012)

[3.4. Использование алгоритма 100](#_Toc100352013)

[3.4.1. Использование алгоритма 1 – поиск решения Excel антагонистической игры 100](#_Toc100352014)

[3.4.2. Использование алгоритма 1 – поиск решений Excel биматричной игры 102](#_Toc100352015)

[3.4.3. Использование алгоритма 1 – поиск решения Excel для задачи о принятии оптимального решения в условиях риска 104](#_Toc100352016)

[3.4.4. Использование алгоритма 1 – поиск решений Excel для задачи о принятии оптимального решения в условиях неопределённости 106](#_Toc100352017)

[3.4.5. Использование алгоритма 2 – Python для решения антагонистической игры 108](#_Toc100352018)

[3.4.6. Использование алгоритма 2 – Python для решения биматричной игры 109](#_Toc100352019)

[3.4.7. Использование алгоритма 2 – Python для решения игр с природой в условиях риска 110](#_Toc100352020)

[3.4.8. Использование алгоритма 2 – Python для решения игр с природой в условиях неопределенности 113](#_Toc100352021)

[3.5. Архитектура решения алгоритма на языке программирования Python 115](#_Toc100352022)

[3.5.2. Архитектура решения биматричной задачи 116](#_Toc100352023)

[3.5.3. Архитектура решения игр с природой в условиях риска 116](#_Toc100352024)

[3.5.4. Архитектура решения игр с природой в условиях неопределенности 117](#_Toc100352025)

[3.6. Тестирование 117](#_Toc100352026)

[3.6.1. Тестирование задачи о нахождении выигрышной стратегии для антагонистической игры 117](#_Toc100352027)

[3.6.2. Тестирование задачи о нахождении выигрышной стратегии для биматричной игры 120](#_Toc100352028)

[3.6.3. Тестирование задачи о принятии оптимального решения в условиях риска 120](#_Toc100352029)

[3.6.4. Тестирование задачи о принятии оптимального решения в условиях неопределенности 122](#_Toc100352030)

[3.7. Заключение 125](#_Toc100352031)

# **1. «Производственная задача линейного программирования»**

# **1.1. Постановка задачи (физическая модель)**

Наш заказчик — небольшая кофейня, которая решает какой план выпуска и продаж каждого из вида кофе является для нее оптимальным для получения максимальной выручки. Мы видим, что компания может производить 4 вида кофе: латте, капучино, американо и макиато. Также присутствуют ингредиенты и их количество для приготовления данных видов кофе. На складе хранится определенное количество ингредиентов, что является ограничением.

Ниже, в Таблице 1.1, представлен один из вариантов распределения ингредиентов для каждого вида напитка и их цена.

Таблица 1.1. Распределение ингредиентов, цены и остатков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Вода | Молоко | Кофе | Цена |
| Латте | 0 | 100 | 50 | 180 |
| Капучино | 0 | 75 | 75 | 200 |
| Американо | 120 | 0 | 30 | 150 |
| Макиато | 0 | 50 | 100 | 230 |
| Склад | 2000 | 4000 | 3000 |  |

# **1.2. Математическая модель**

Поставленная задача по своему типу является производственной задачей. Такая задача линейного программирования определяет план производства изделий с целью получения максимальной прибыли.

Общий вид целевой функции:

Где – цена одного вида продукции

– количество вида продукции

Стандартная форма записи:

Ограничения:

Где – элемент весовой матрицы  
 – ограничения ингредиентов (материалов) по складу

Стандартная форма записи:

# **1.3. Алгоритм решения**

## **1.3.1. Алгоритм 1. Решение с помощью Поиска решения в Excel**

### **1.3.1.1. Описание входных данных**

В качестве входных данных мы имеем:

* значения весовой матрицы, которая включает в себя количество ингредиентов, необходимых для приготовления определенных видов кофе;
* значения цены определенного вида кофе;
* вместимость склада, распределённая по определенным ингредиентам.

### **1.3.1.2. Описание алгоритма решения**

Алгоритм принимает на вход массив переменных, массив ограничений и ячейку, хранящую целевую функцию. Далее выбирается метод решения. В частности, используется симплекс метод, так как мы решаем задачу линейного программирования. Надстройка "Поиск решения" изменяет значения в ячейках переменных решения согласно пределам ячеек ограничения и выводит нужный результат в целевой ячейке.  
 Таким образом, алгоритм надстройки определяет максимальное или минимальное значение одной ячейки, изменяя другие ячейки. Например, вы можете изменить введенные количественные характеристики ингредиентов и посмотреть, как изменится планируемая сумма прибыли.

*Подробный пример реализации алгоритма*

Шаг 1: в соответствии с описанием задачи составляем весовую матрицу. Вводим соответствующие значения в матрицу. (см таблицу 1)

Шаг 2: создаём таблицу с ценой продукции, вводим соответствующие данные.

Шаг 3: создаём таблицу со вместимостью склада по определенным ингредиентам, вводим соответствующие данные.

Шаг 4: создаём пустую таблицу с искомыми значениями количества товаров и отводим пустую ячейку для значения целевой функции.

Шаг 5: согласно математической модели, реализуем целевую функцию через встроенную функцию Excel СУММПРОИЗВ(). Данное действие представлено на Рисунке 1.1:

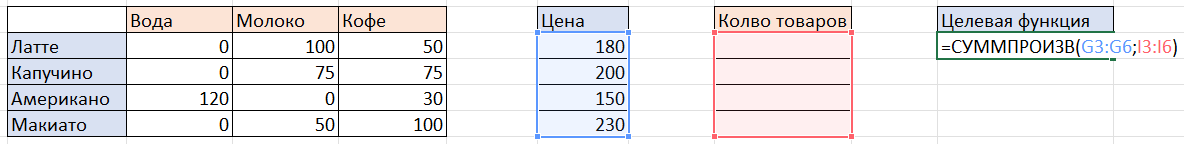


Рисунок 1.1. Ввод целевой функции

Шаг 6: устанавливаем ограничения: количество товара должно быть меньше или равно запасам на складе, данный шаг изображен на Рисунке 1.2:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.2. Установка ограничений

**Шаг 7: с помощью поиска решений применяем симплекс-метод: с учётом ограничений, ищем такие положительные значения переменных, чтобы целевая функция стала максимальной, на Рисунке 1.5 - заполненное окно «Поиск решения».**

**Ход выполнения 7 шага:**

* **В окне поиска решений вводим номер ячейки, характеризующий целевую функцию, которую мы будем оптимизировать (K3).**
* **Вводим ячейки переменных, соответствующих количеству товаров (I3:I6)**
* **Прописываем ограничение, указав, что значения в ячейках ограничения (C11:E11) должны быть меньше либо равны значениям в ячейках склада (C9:E9).**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.3. Параметры поиска решения

### **1.3.1.3. Описание выходных данных**

Сначала надстройка сообщает о наличии решения, так как не всегда оно может быть найдено.

Алгоритм выводит неизвестные переменные в определенный пользователем массив ячеек, максимальное/минимальное значение целевой функции в ячейку, в которую была введена сама функция.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.4. Результаты поиска решения

## **1.3.2. Алгоритм 2. Решение с помощью функции linprog.**

### **1.3.2.1. Описание входных данных**

Как и в Алгоритме 1 входными данными являются следующие компоненты: весовая матрица, в которой указано количество требуемых материалов для приготовления кофе, стоимость всех видов производимого кофе, а также значения количество материалов, которые присутствуют в наличии на складе.

Вышеперечисленные данные должны быть записаны в списки, чтобы функция linprog смогла принять их как аргументы и решить поставленную задачу.

### **1.3.2.2. Описание алгоритма решения**

Введенные пользователем массив цен, массив ограничений и элементы весовой матрицы при помощи циклов записываются в три списка. В последствии эти списки принимаются функцией linprog как аргументы.

В функции linprog есть три способа подсчета искомого значения целевой функции:

* Симплекс метод
* Двухфазовый симплекс метод
* Метод внутренней точки

*Симплекс метод*

Системы линейных неравенств и равенств ограничений задают многомерные выпуклые многогранники. В одномерном случае это точка, луч или отрезок, в двумерном — выпуклый многоугольник, в трехмерном — выпуклый многогранник. В общем случае, для систем из n неравенств в m-мерном пространстве — это точка, удовлетворяющая системе, для которой ровно n из этих неравенств обращаются в равенства. Таких точек всегда конечное число, хоть их и может быть очень много.

Когда у нас есть конечное число точек, мы можем найти максимум посредством перебора и подстановки. Симплекс-метод вместо перебора двигается от вершины к вершине по ребрам таким образом, чтобы значений целевой функции улучшалось. Если у вершины нет «соседей», в которых значений функции лучше, то она оптимальна.

*Двухфазовый симплекс метод*

Двухфазный симплекс-метод применяется в тех случаях, когда в задаче ЛП в канонической форме затруднительно определить начальное допустимое базисное решение с помощью элементарных преобразований.

Если в условии задачи линейного программирования не все ограничения представлены неравенствами типа «≤», то далеко не всегда нулевой вектор будет допустимым решением. Однако каждая итерация симплекс-метода является переходом от одной вершины к другой, и если неизвестно ни одной вершины, алгоритм вообще не может быть начат.

Процесс нахождения исходной вершины не сильно отличается от однофазного симплекс-метода, однако может в итоге оказаться сложнее, чем дальнейшая оптимизация.

*Метод внутренней точки*

Согласно методам внутренней точки (иначе называемым методами барьерных функции), исходную для поиска точку можно выбирать только внутри допустимой области.

Выбор начальной точки поиска осуществляется в зависимости от формулировки задачи. При отсутствии ограничений или их преобразовании к функциям штрафа с внешней точкой начальная точка выбирается произвольно. При наличии ограничений или их преобразовании к функциям штрафа с внутренней точкой начальная точка выбирается внутри допустимой области

При этом множество точек делится на допустимые и недопустимые в зависимости от ограничений. В свою очередь множество допустимых точек в зависимости от ограничений также делится на граничные и внутренние.

В независимости от выбранного метода функция linprog выводит оптимальное значение целевой функции, массив переменных оптимального решения, также дополнительно с помощью методов можно получить массив остатков на складе.

### **1.3.2.3. Описание выходных данных**

Полученные после расчета данные представлены на Рисунке 1.5.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.5. Выходные данные, полученные в Python

Заметим, что на данном рисунке представлены результаты, рассчитанные тремя различными методами. Сравнение результатов этих трех методов будет продемонстрировано позже, в пункте 6 «Тестирование».

Первой строкой выводится целевая функция:

Второй строкой выводится оптимальное значение целевой функции, в данном случае оно равно 10633,33.

В третьей строчке выводится список с искомыми переменными, таким образом, искомые переменные равны:

# **1.4. Варианты использования системы**

## **1.4.1. Использование алгоритма 1 – поиск решения Excel**

Шаг 1: Пользователь заменяет значения в основной таблице с ингредиентами и видами кофе.

Если необходимы дополнительные колонки, то пользователь должен кликнуть на столбец справа от последнего, выбрать «Вставить…», затем во всплывшем окне выбрать «столбец». После этого придется заново запустить поиск решения (см. пункт 1.3.1.2)

Аналогичные действия нужно проделать, при необходимости вставки новых строк.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.6. Результаты поиска решения

Шаг 2: Пользователь изменяет значения цены. При необходимости добавления новых строк требуется повторить аналогичные шагу 1 действия. Также запуская поиск решения заново.

Пример полученного результата:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.7. Результаты поиска решения

Согласно результатам поиска решений, чтобы получить максимальную выручку в виде 10633,33 у.е. кофейне следует продавать 36,67 товаров разновидности Латте по цене 180 у.е., не продавать капучино, продавать 16,67 товаров Американо по цене 150 у.е. и продавать 6,67 товаров Макиато по цене 230 у.е.

Разумеется, мы принимаем к сведению, что полученные результаты являются точными с точки зрения математики, но на практике полученное решение нельзя реализовать, поскольку кофейня не может произвести дробное количество напитков. Одним из вариантов решения данной проблемы может быть округление найденных переменных в меньшую сторону и нахождение нового значение целевой функции.

## **1.4.2. Использование алгоритма 2 – Python**

Данное решение визуализировано с помощью библиотеки tkinter. Пользователь может ввести данные различными путями: ввод данных вручную и с помощью импорта csv-файла.

### **1.4.2.1. Ввод данных вручную**

Рассмотрим ввод данных вручную. При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: в строке меню выбрать Производственная задача > ввод данных вручную.

Шаг 2: в новом всплывающем окне присутствует три поля для ввода исходных данных, данное окно представлено на Рисунке 1.8:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.8. Окно программного интерфейса

В них пользователь должен ввести данные, используя следующую инструкцию:

В первую ячейку необходимо записать данные о стоимости кофе через запятую без пробела, интерпретация представлена на Рисунке 1.9:

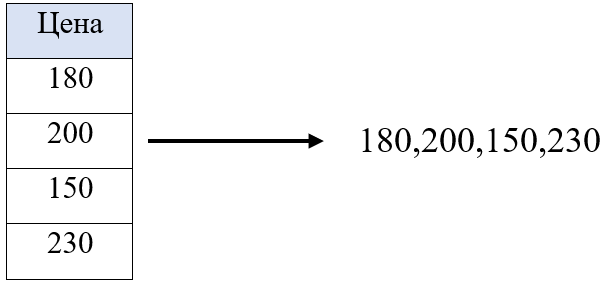


Рисунок 1.9. Ввод стоимости

Во вторую ячейку необходимо записать количество ингредиентов, присутствующих на складе через запятую без пробела, интерпретация представлена на Рисунке 1.10.

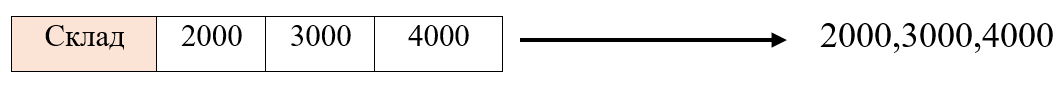


Рисунок 1.10. Ввод значений склада

В третью ячейку нужно записать весовую матрицу, разделяя количество ингредиентов по столбцу запятой, а столбцы точкой с запятой. Ниже приведен Рисунок 1.11 с интерпретацией.

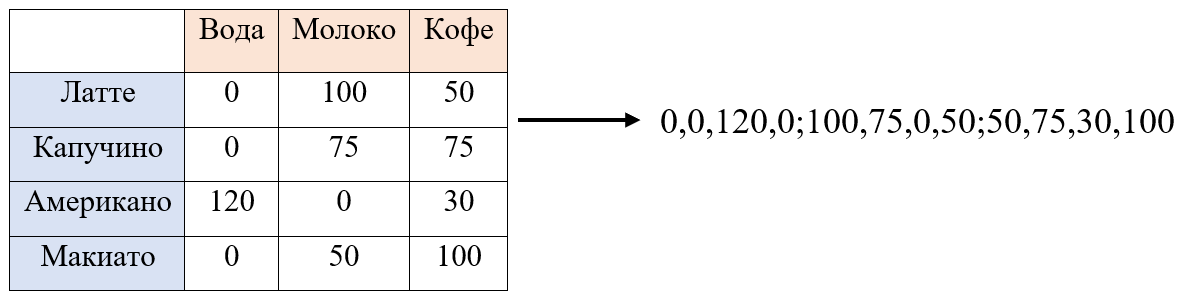


Рисунок 1.11. Ввод значений весовой матрицы

После заполнения информации окно производственной задачи должно выглядеть таким образом, как показано на Рисунке 1.12:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.12. Введенные значения

Шаг 3: после заполнения данных для получения решения пользователь должен выбрать способ подсчета оптимального значения, выбрав соответствующую кнопку.

В функции linprog есть три способа подсчета искомого значения целевой функции:

* Симплекс метод
* Двухфазовый симплекс метод
* Метод внутренней точки

В результате нажатия на все три кнопки выведется следующее окно(рис.1.13):

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.13. Найденное решение

Первой строкой выводится целевая функция:

Второй строкой выводится оптимальное значение целевой функции, в данном случае оно равно 10633,33.

В третьей строчке выводится список с искомыми переменными, таким образом, искомые переменные равны:

### **1.4.2.2. Импортирование данных из CSV**

Рассмотрим ввод данных с помощью готового csv-файла. При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: скачанный файл с кодом поместить в одну папку с csv-файлом.

Шаг 2: при выборе в выпадающем меню соответствующего способа ввода данных в открывшемся окне ввести название csv-файла с обязательным окончанием “.csv”. Пример ввода имени файла показан на Рисунке 1.14:



Рисунок 1.14. Ввод имени csv-файла

Шаг 3: нажать на кнопку “Готово”. Ниже будет выведен результат работы программы (рис. 1.15).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.15. Вывод результатов программы с csv

Структура самого CSV-файла должна выглядеть следующим образом, как показано на Рисунке 1.16. Может быть задана матрица любого размера nxm. В столбец Y записываются ограничения по складу. В последней строке цена каждого кофе соответственно. Можно заметить, что матрица отличается от первоначальной, тк имеет транспонированный вид, то есть столбцы записаны в строки.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.16. Структура csv-файла

Выходные данные так же, как и в ручном вводе выдают оптимальные значения целевой функции и переменных. Незначительно отличается вывод самой целевой функции – выводятся коэффициенты и переменные, оптимальные значения переменных также выводятся в виде списка. Вывод показан на Рисунке 1.17.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.17. Вывод результатов

# **1.5. Архитектура решения**

Считывание, обработка и вывод информации осуществляется преимущественно через графическую библиотеку tkinter, чтобы обеспечить понятный и удобный интерфейс для пользователя. Далее будут описаны функции, разделенные на два блока: Frontend – пользовательская сторона и Backend – программная часть.

## **1.5.1. Fronted**

**Пользовательский интерфейс реализован с помощью библиотеки tkinter.**

**Функция new\_win – создает окно для главного меню.**

**Функции строки меню:**

**\_close – закрытие окна;**

**about – предоставление информации о работе с программой;**

**open\_class – создание нового окна, где пользователь вводит данные вручную;**

**Метод fmen.add\_command – визуализируют функции строкового меню.**

**open\_class0 – создание нового окна, где пользователь вводит данные, импортируя csv-файл.**

**В функциях open\_class и open\_class0 находится функция widget, в которой содержаться следующие графические объекты:**

**self.lab – текстовая метка, в которой содержится инструкция для ввода, а также непосредственный вывод результатов решения;**

**self.textbox – поле ввода, в которое пользователь должен ввести данные;**

**self.btn – кнопка, реализующая запуск процесса решения.**

**Также для отображения окна используются следующие методы:**

**root.title – задает наименование окна;**

**root.geometry – задает размер выводимого окна.**

## **1.5.2. Backend**

**Последующие функции нужны для обработки информации и получения результата.**

**click\_revised – нахождения решения с помощью двухфазового симплекс-метода.**

**click\_simplex – нахождения решения с помощью симплекс-метода.**

**click\_point – нахождения решения с помощью метода внутренней точки.**

**В данных функциях импортируется библиотека scipy, а конкретно функция linprog, которая обрабатывает входные данные и выдает решение.**

***Входные данные*:**

**Основными входными данными linprog являются: вектор коэффициентов целевой функции f, матрица ограничений-неравенств A, вектор правых частей ограничений-неравенств b, матрица ограничений-равенств Aeq, вектор правых частей ограничений-равенств beq, вектор lb, ограничивающий план x снизу, вектор ub, ограничивающий план x сверху.**

**В нашем случае входными данными являются:**

**stoim – стоимость производимых товаров (поддерживает тип данных list);**

**sklad – ограничения по складу (поддерживает тип данных list);**

**matriz – весовая матрица (поддерживает тип данных list).**

**Переменная opt – поиск решения функции linprog, в который включаются входные данные и указывается метод вычисления (двухфазовый симплекс-метод – “revised simplex”, симплекс-метод – “simplex”, метод внутренней точки указан по умочанию).**

**Переменная vfunc – вывод целевой функции.**

***Выходные данные:***

**Переменная opt.fun = оптимальное значение целевой функции, найденное функцией linprog.**

**Переменная opt.x – оптимальное значение переменных, найденное функцией linprog в формате списка.**

# **1.6. Тестирование**

Для тестирования были использованы 5 датасетов, тесты производились следующими методами: Revised simplex, Simplex, Inner-point в Python, а также с помощью поиска решений в MS Excel. Результаты тестирования представлены в Таблице 1.2.

Таблица 1.2. Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные | Метод решения 1.  Revised simplex | Метод решения 2. Simplex | Метод решения 3.  Inner-point | MS EXCEL |
| Датасет 1:  Стоимость: 180,200,150,230  Склад:  2000,4000,3000  Весовая матрица:  0,100,50;  0,75,75;  120,0,30;  0,50,100 | Целевая функция:  Значение целевой функции: **10633,33**  Искомые переменные | Целевая функция:  Значение целевой функции: **10633,33**  Искомые переменные | Целевая функция:  Значение целевой функции: **10633,33**  Искомые переменные | Целевая функция:  Значение целевой функции: **10633,33**  Искомые переменные |
| Датасет 2:  Стоимость: 160,150  Склад:  349,479  Весовая матрица:  20,56;  56,89 | | Целевая функция:  Значение целевой функции: **1630,64**  Искомые переменные: | Целевая функция:  Значение целевой функции: **1630,64**  Искомые переменные: | Целевая функция:  Значение целевой функции: **1630,64**  Искомые переменные: | Целевая функция:  Значение целевой функции: **1630,64**  Искомые переменные: |
| Датасет 3:  Стоимость: 250,180,200  Склад:  3000,3000,3000  Весовая матрица:  100,75,50;  30,60,90;  75,150,30 | Целевая функция:  Значение целевой функции: 9600  Искомые переменные | Целевая функция:  Значение целевой функции: 9600  Искомые переменные | Целевая функция:  Значение целевой функции: 9600  Искомые переменные | Целевая функция:  Значение целевой функции: 9600  Искомые переменные |
| Датасет 4:  Стоимость: 200,180,240,220,150  Склад:  10000,12000,15000  Весовая матрица:  35,40,100;  60,30,70;  55,75,55;  25,80,80;  0,50,150 | Целевая функция:  Значение целевой функции:  **45439,22651933702**  Искомые переменные | Целевая функция:  Значение целевой функции:  **45439,22651933702**  Искомые переменные | Целевая функция:  Значение целевой функции:  **45439,22620871765**  Искомые переменные | Целевая функция:  Значение целевой функции:  **45439, 226519337**  Искомые переменные |
| Датасет 5:  Стоимость: 265,270  Склад:  10000,10000,10000,10000  Весовая матрица:  50,0,61,38;  61,37,60,0; | Целевая функция:  Значение целевой функции:  44868,24  Искомые переменные | Целевая функция:  Значение целевой функции:  44868,24  Искомые переменные | Целевая функция:  Значение целевой функции:  44868,24  Искомые переменные | Целевая функция:  Значение целевой функции:  44868,24  Искомые переменные |

**Проанализировав полученные результаты, хотелось бы сделать следующие выводы. Методы Revised Simplex, Simplex и MS Excel совпадают всегда вплоть до одиннадцатого знака после запятой. Метод Inner-point также практически всегда (за исключением Датасета 4, когда значение перестало совпадать после четвертого символа) выводит значение целевой функции, идентичное предыдущим трем.**

**Также стоит отметить, что вывод значений искомых переменных несколько отличается. Метод Inner-point выводит эти результаты экспоненциальной записью, тогда как другие десятичной дробью. Но данный вид вывода, может быть, в дальнейшем постобработан под конкретные требования заказчика.**

# **1.7. Заключение**

**Решение для предоставленной задачи мы реализовали с помощью двух алгоритмов, один из которых включал в себя несколько методов решения. Первый алгоритм основан на надстройки Excel – поиск решений. Его преимуществами можно считать – наглядное представление всех массивов данных, простота редактирования и ввода новых данных, автоматическое обновление формул. Из недостатков – отсутствие пользовательского интерфейса, ограничен выбор способов решения задач линейного программирования, нет наглядного представления самой целевой функции в формульном виде.**

**Второй алгоритм, реализованный в Python при помощи библиотеки tkinter имеет очевидное преимущество перед первым – пользовательский интерфейс, есть возможность выбора ввода данных, а также можно осуществить подсчет тремя различными методами функции linprog библиотеки SciPy и наглядно их сравнить. Эта программа также имеет перспективы для расширения функционала, тк можно добавить выгрузку данных в различных форматах и т.п.**

**Все используемые методы подсчета выдают одинаковый результат, разница незначительна (около 10го знака после запятой) и являются проверкой друг друга. Правильность кода была проверена на 5 случайно сформированных датасетах.**

**Не всегда на выходе мы получаем результаты, которые соотносимы с реальностью. Например, дробное количество чашек кофе. Мы описали следующий вариант решения такой проблемы: округление найденных переменных в меньшую сторону и нахождение нового значение целевой функции.**

# **2. «Оптимизация на графах»**

# **2.1. Постановка задачи (физическая модель)**

Наш заказчик – мебельная фабрика, специализирующаяся на производстве элитных кроватей. Данная компания помимо поставки продукции на российский рынок, занимается экспортом в другие страны. Ранее организация доставляла мебель посредством привлечения сторонних перевозчиков, сейчас же компания расширяется и открывает свой отдел логистики. Именно поэтому мебельная фабрика обратилась к нам с задачей поиска наиболее выгодного с точки зрения времени маршрута.

Одним из главных импортеров продукции, произведенной компанией-заказчиком, является Венгрия, поэтому маршрут будем прокладывать именно в ее столицу – Будапешт. Также учтем то, что заказчик осуществляет доставку при помощи сцепки, поэтому оптимальнее использовать только сухопутные маршруты.

Специалист отдела логистики составил сеть дорог, по которым можно добраться до Будапешта из Москвы (рис. 2.1). Заметим, что все отмеченные дороги имеют движение в обе стороны.

Изображение выглядит как карта

Автоматически созданное описание

Рисунок 2.1. Сеть дорог

Заметим, что у каждого отрезка указано количество часов, которое необходимо затратить, чтобы преодолеть данную дорогу.

Также заказчик поставил дополнительную задачу, с расчетом на расширение своих поставок в будущем: товар должен быть доставлен в каждый город, а далее дальнобойщик должен вернуться в пункт отправления. То есть, если обращаться к Рисунку 2.1, дальнобойщик должен отправиться из Москвы, посетить все города для доставки товара, а после вернуться обратно в Москву. Заказчик также поставил дополнительное требование о вводе вершин покоординатно, без самостоятельного указания расстояния между ними.

# **2.2. Математическая модель**

Поставленная задача по своему типу является задачей поиска кратчайшего пути между двумя заданными вершинами графа. Такая задача комбинаторной оптимизации определяет наиболее выгодный маршрут, который можно осуществить за кратчайший срок и с наименьшими затратами.

**Целевая функция** – минимальная суммарная длина суммарная пути:

**Ограничения:**

– маршрут должен начинаться в пункте А и заканчиваться в пункте В,

, – переходить можно только по существующим рёбрам

– вершины не должны повторяться

– искомые переменные целочисленные

– вход в вершину соответствует выходу из неё

– вход и выход в каждую вершину производится не более одного раза.

Дополнительная задача представляет собой задачу коммивояжера. Задача коммивояжера заключается в поиске самого выгодного маршрута, проходящего через указанные города хотя бы один раз с последующим возвратом или же с остановкой в последнем городе. Решение будет реализовано с помощью муравьиного алгоритма.

Математическая модель задачи коммивояжера имеет следующий вид:

Целевая функция:

Ограничение на посещение только одного города:

Общий вид математической модели **муравьиного алгоритма**:

Где множество вершин, в которые разрешен переход из вершины,

уровень феромона на дуге на шаге t,

расстояние между вершинами ,

константы.

Уровень феромона:

Где интенсивность испарения,

множество муравьёв, прошедших по дуге (),

расстояние между вершинами ,

предполагаемая идеальное значение ЦФ,

феромон, откладываемый k-ым муравьём на ребре ().

Обновление феромона:

,

Где М – количество муравьёв в колонии

# **2.3. Алгоритма решения**

## **2.3.1. Алгоритм 1. Решение с помощью Поиска решения в Excel задачи о поиске кратчайшего пути**

### **2.3.1.1. Описание входных данных**

В качестве входных данных мы имеем:

* значения весовой матрицы, которая является матрицей смежности для графа и содержит в себе «вес» (количество времени в пути), каждого из ребер
* ограничение по входам и выходам для каждой из вершин графа

### **2.3.1.2. Описание алгоритма решения**

Алгоритм принимает на вход массив переменных (бинарную матрицу), массив ограничений и ячейку, хранящую целевую функцию. Далее выбирается метод решения. В частности, используется симплекс метод. Надстройка "Поиск решения" изменяет значения в ячейках переменных решения согласно пределам ячеек ограничения и выводит нужный результат (количество затраченных часов) в ячейке, содержащей целевую функцию.  
 Таким образом, алгоритм надстройки определяет максимальное или минимальное значение одной ячейки, изменяя другие ячейки. Например, вы можете изменить введенные «веса» матрицы смежности графа, изменить количество вершин в нем, и посмотреть, как изменится целевая функция.

*Подробный пример реализации алгоритма*

Шаг 1: в соответствии с описанием задачи составляем весовую матрицу (матрицу смежности построенного графа). Вводим соответствующие «весовые» значения дуг графа в матрицу. (см рис 2.2)

Изображение выглядит как лампа

Автоматически созданное описание

Рисунок 2.2 Весовая матрица

Шаг 2: создаем пустую матрицу искомых переменных как показано на рисунке 2.3.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.3 Матрица переменных

Шаг 3: согласно математической модели, реализуем целевую функцию через встроенную функцию Excel СУММПРОИЗВ(). Данное действие представлено на Рисунке 2.4:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.4 Создание целевой функции

Шаг 4: устанавливаем ограничения: для удобства создаем столбец «выходы» и строку «входы», вносим в столбец сумму по каждой из строк матрицы, а в строку сумму по каждому столбцу. Данный шаги изображены на Рисунках 2.5 и 2.6:



Рисунок 2.5 Столбец "выходы"



Рисунок 2.6 Строка "входы"

Шаг 5: далее транспонируем столбец «выходов» и располагаем над строкой «входов» как показано на рисунке 2.7. Обратите внимание, должны быть четкие ограничения:

* количество выходов в первой строке равно единице, в последней - нулю;
* количество входов в первом столбце равно нулю, в последнем – единице;
* количество входов и выходов, заключенные между крайними значениями должны быть равны между собой.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.7 Создание ограничений

**Шаг 6: с помощью поиска решений применяем симплекс-метод: с учётом ограничений, ищем такие значения переменных, чтобы целевая функция стала минимальной, на Рисунке 2.8 - заполненное окно «Поиск решения».**

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 2.8 Заполненное окно "Поиск решения"

**Ход выполнения 6 шага:**

* **В окне поиска решений вводим номер ячейки, характеризующий целевую функцию, которую мы будем оптимизировать (С21).**
* **Вводим ячейки переменных (В13:G18)**
* **Прописываем ограничения, указав, что значения в ячейках (C24:F24) должны быть равны значениям в ячейках (C26:F26), матрица переменных должна содержать бинарные переменные, а также особые ограничения для первых и последних строк/столбцов, описанные в шаге 5.**

**2.3.1.3. Описание входных данных**

Сначала надстройка сообщает о наличии решения, так как не всегда оно может быть найдено.

Алгоритм выводит:

* значение целевой функции – время, которое будет затрачено на кратчайший путь
* бинарную таблицу переменных, в которых указано как соединяются вершины полученного пути (если в ячейке стоит 1, то такое соединение вершин входит в полученный путь)

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.9 Выходные данные

**2.3.2. Алгоритм 2. Решение в Python задачи о кратчайшем пути алгоритма Дейкстры**

**2.3.2.1. Описание входных данных**

Вид входных данных зависит от выбранного способа их введения. Для ручного ввода выходными данными являются:

* Перечисленные вершины графа, их расстояние между друг другом в следующем виде: Вершина a\_Вершина b\_расстояние между ними, Вершина a\_Вершина c\_расстояние между ними, где «\_» это пробел.

Пример показан на рисунке 2.10:

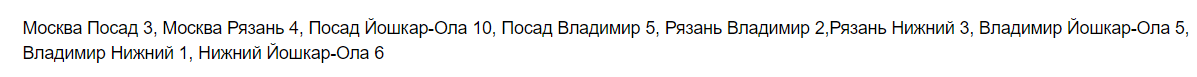


Рисунок 2.10 Образец ввода вершин и расстояний

* Точка отправления
* Точка прибытия

Перечисленные вершины записываются в словарь, ключами которого являются все вершины графа, а значениями словари, содержащие соединенные с ними вершины в качестве ключей и расстояние в качестве значений. Такая запись используется для последующей удобной записи данных в список, который передается в nx.Graph(), подробнее это описано в пункте 2.5.1.

Рассмотрим импорт входных данных из csv-файла. При таком способе пользователь должен передать:

* Матрицу смежности графа, записанную в csv файле. Пример можно видеть на рисунке 2.11

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 2.11 Матрица смежности графа

* Точку отправления
* Точку прибытия

Импортированная матрица сразу передается на вход в nx.Graph().

**2.3.2.2. Описание алгоритма решения**

Опишем суть работы алгоритма Дейкстры на примере графа, приведенного ниже, для поиска наилучшего маршрута между Москвой и Будапештом.



Рисунок 2.12 Искомый граф

Инициализируем алгоритм:

1. Устанавливаем Москву в качестве начального узла («current node»)
2. Устанавливаем расстояния между Москвой и всеми другими городами в (бесконечность), за исключением расстояния между Москвой и Москвой, которое принимаем равным 0.

Алгоритм:

Шаг 1: выбираем узел с наименьшим значением в качестве «current node» («текущего узла») и посещаем всех соседей. При посещении каждого соседа обновляем их ориентировочное расстояние от начального узла.

Шаг 2: после посещения всех соседей текущего узла обновим их расстояния и пометим текущий узел как «visited» («посещенный»). Отметка узла как «посещенного» означает, что найдена его окончательная ценность.

Шаг 3: возвращаемся к первому шагу и повторяем до тех пор, пока не посетим все узлы графа.

В нашем примере начинаем с отметки Москва, поскольку её значение равно 0, как «текущего узла». Далее посетим два соседних узла Москвы: Минск и Харьков. В начале алгоритма их значения устанавливаются на бесконечность, но, когда мы посещаем узлы, то обновляем значение для Минска до 8 и Харькова до 9,5.



Рисунок 2.13 Первый этап подсчета маршрута

Затем отмечаем Москву как «посещенная». Мы знаем, что её окончательная стоимость равна нулю, и нам больше не нужно её посещать. Продолжаем со следующего узла с наименьшим значением, которым является Минск.

Посещаем все соседние узлы Минска, которые не помечены как «посещенные». Соседи Минска — Люблин и Киев. Обновляем значения Люблина, добавляя значение ребра, соединяющего Минск и Люблин (6,5), к значению Минск (8), что дает нам значение 14,5. Аналогично для Киева получаем значение 15.



Рисунок 2.14 Второй этап подсчета маршрута

Отмечаем Минск как посещённый и выбираем следующий узел — Харьков. Посещаем соседа Харькова Киев и обновляем его ценность.

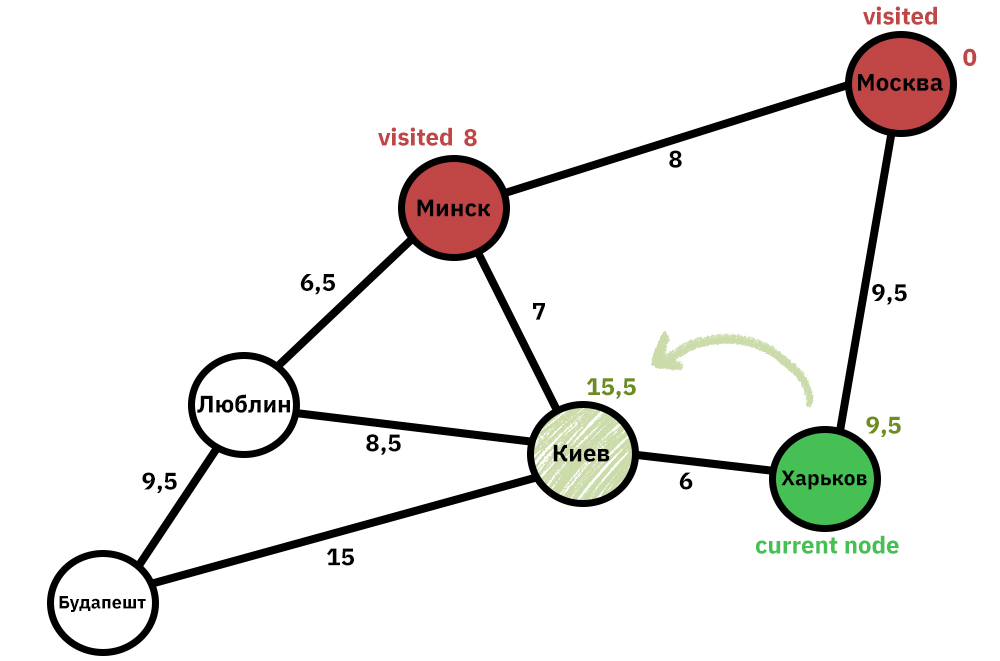


Рисунок 2.15 Третий этап подсчета маршрута

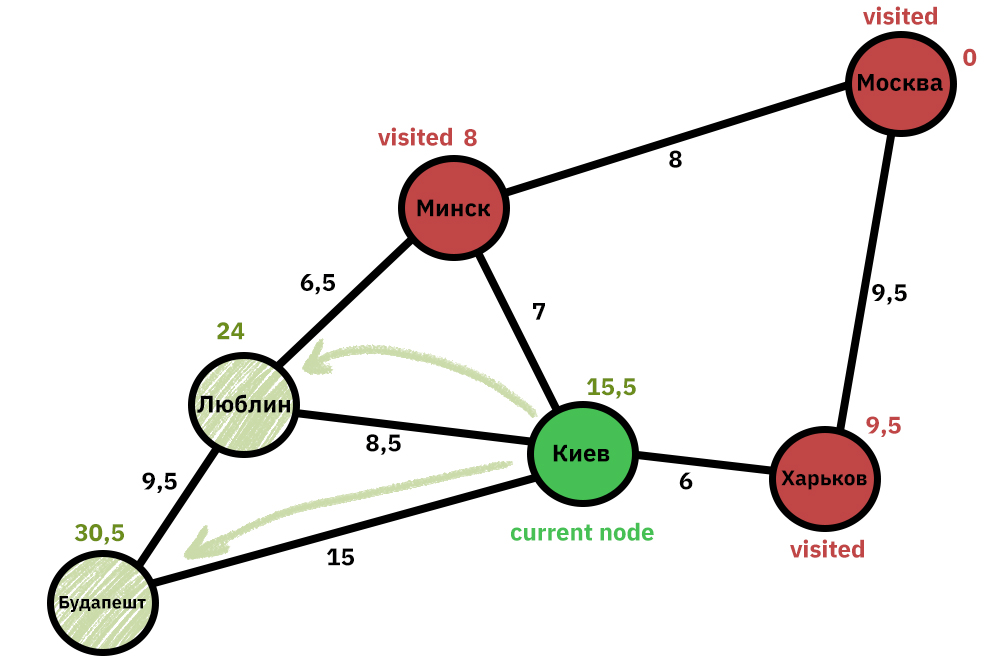
Отмечаем Харьков как посещённый. следующий узел — Киев. Соседи Киева – Минск, Люблин и Будапешт. Но Минск мы игнорируем, потому что в нём мы уже были. Вместо этого обновляем значения Люблина и Будапешта. 

Рисунок 2.16 Заключительный этап подсчета маршрута

Так, мы нашли первый путь из Москвы до Будапешта стоимостью 30,5 и, если закончить шаг с Люблином – со стоимостью 33,5. Являются ли эти пути оптимальными? После этого алгоритм возвращается к первому шагу и перебирает остальные варианты, в результате которого находит искомый оптимальный маршрут стоимостью 24. Функции, использующиеся для реализации алгоритма подробно описаны в пункте 2.5.1.

**2.3.2.3. Описание выходных данных**

Алгоритм выводит:

* Строку с полученным оптимальным путем
* Ценность полученного маршрута
* Визуализированный с помощью библиотеки nx граф, с выделенным кратчайшем путем

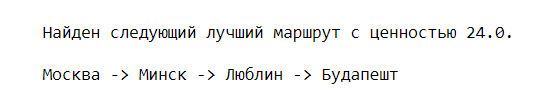


Рисунок 2.17 Строковые выходные данные алгоритма Дейкстры

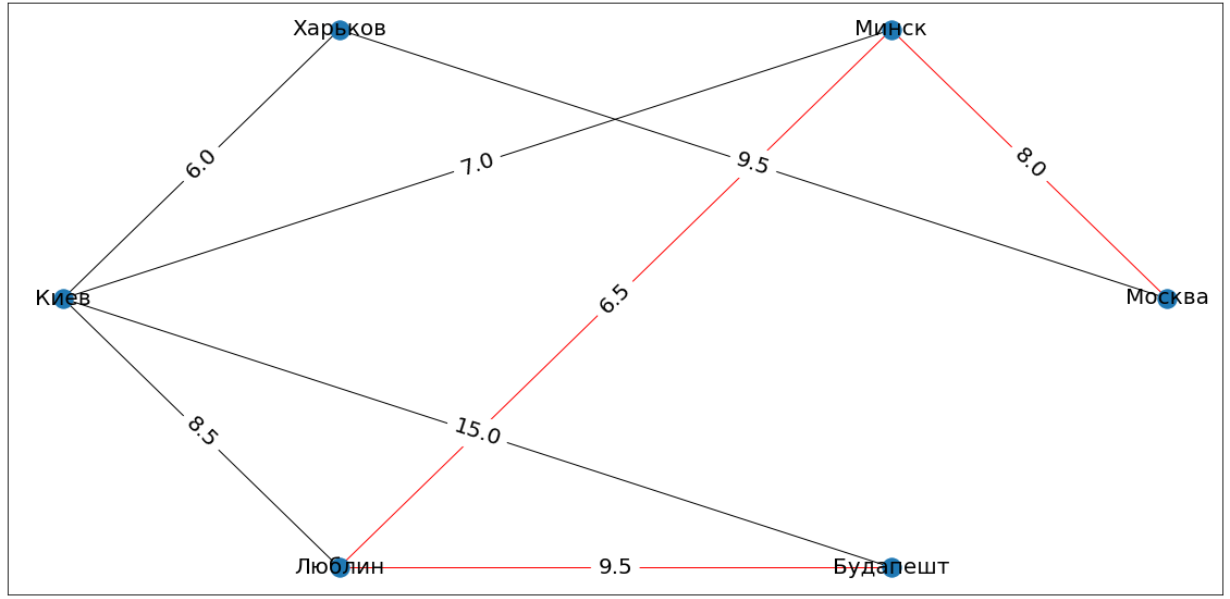


Рисунок 2.18 Визуализированный маршрут

**2.3.3. Решение в Python задачи коммивояжера муравьиным алгоритмом**

**2.3.3.1. Описание входных данных**

Входные данные зависят от выбора способа их введения. Есть три способа ввода: ручной ввод (координат точек x и y или матрицу весовых коэффициентов), случайная генерация координат, считывание из CSV-файла.

Для ввода вручную понадобится выбрать что ввести:

* Количество вершин
* Координатные точки X и Y всех вершин
* Количество муравьев
* Количество итераций

Перечисленные вершины записываются изначально в списки построчно. Такая запись позднее преобразуется позднее в вектор с помощью метода «array» из библиотеки NumPy.

Рассмотрим импорт входных данных из CSV-файла. Для вставки CSV-файла требуется ввести:

* Координаты точек, записанные в CSV-файл. Формат CSV-файла должен выглядеть следующим образом (при этом размер таблицы может увеличиваться):

Изображение выглядит как текст, кроссворд

Автоматически созданное описание

Рисунок 2.19 Таблица координат вершин для CSV файла

* Путь к CSV-файлу:
  + Название файла, при условии расположения файла с алгоритмом в одной папке с CSV-файлом.
  + Полностью путь, при ином условии.
* Количество муравьев
* Количество итераций

Алгоритм считывает координаты X и Y. В дальнейшем строки файла CSV преобразуются в вектор для дальнейших вычислений.

При генерации случайных вершин требуется только ввести количество муравьев и итераций.

**2.3.3.2. Описание алгоритма решения**

Опишем суть работы муравьиного алгоритма на примере задачи Коммивояжёра, для поиска наилучшего маршрута с обязательным посещением каждой вершины.

Инициализируем алгоритм:

1. Устанавливаем на координатной плоскости точки X и Y каждой вершины (путём выбора способа ввода входных данных).
2. Устанавливаем количество вершин для посещения.
3. Устанавливаем начальное значение α и β (влияние наносимого феромона каждым муравьем и влияние расстояния между вершинами на выбор следующей вершины соответственно).
4. Устанавливаем количество муравьев для поиска решения.
5. Устанавливаем количество итераций необходимых для поиска решения.

Алгоритм:

Шаг 1: получаются входные данные любым из способов. Затем, при помощи модуля spatial из библиотеки scipy происходит вычисление матрицы расстояний (расстояние от каждой вершины до каждой вершины).

Шаг 2: после считывания матрицы начинается запуск муравьев в соответствии с α и β.

Шаг 3: с помощью формулы рассчитывается расстояние, которое прошел каждый из муравьев, самое оптимальное значение записывается в таблицу результатов, наносится феромон на самый оптимальный путь, который в дальнейшем будет влиять на выбор вершины очередной группой муравьев в следующей итерации.

Шаг 4. муравьи возвращаются и цикл повторяется до тех пор, пока количество итераций не будут равно введенному количеству итераций.

В нашем примере мы начинаем путь с отметки 0 (при желании можно поменять начальную вершину). Далее посещаются все возможные вершины в зависимости от количества муравьев. Таблица с расстояниями равна нулю до первой итерации. После первой итерации записываются результаты и выбирается минимальный маршрут, на этот маршрут наносится феромон.

Важное значение имеют значения α и β. α – коэффициент важности феромонов, β – коэффициент важности расстояния при выборе вершины. Они влияют на выбор вершины муравьём.

После каждого цикла феромоны нанесённые на путь, проложенный муравьями, имеют свойство испаряться. Как оптимальные и неоптимальные маршруты теряют вес в вероятностном выборе вершины из-за испарения феромонов. Разница заключается в том, что на не пройденных маршрутах количество феромонов не обновляется.

В результате проделанных шагов алгоритм выводит оптимальный путь и его стоимость на определенной итерации.

**2.3.3.3. Описание выходных данных**

Алгоритм выводит:

* Строку с полученным оптимальным путем
* Ценность полученного маршрута
* Визуализированный путь муравьев на каждой из итераций
* График зависимости стоимости пути от количества итераций

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.20 Строковые входные данные

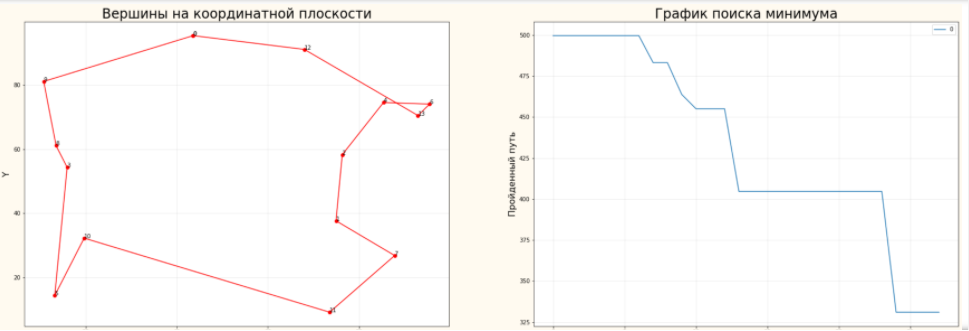


Рисунок 2.21 Выводимые графики

**2.4. Варианты использования системы**

**2.4.1. Использование алгоритма 1 – поиск решения Excel для задачи о поиске кратчайшего пути**

Шаг 1: Пользователь заменяет значения в основной таблице с весовой матрицей. Если необходимы дополнительные колонки, то пользователь должен кликнуть на столбец справа от последнего, выбрать «Вставить…», затем во всплывшем окне выбрать «столбец», те же действия для строки.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.22 Первый шаг

Шаг 2: Пользователь очищает матрицу искомых переменных, если она была заполнена значениями предыдущего поиска решений.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.23 Второй шаг

Шаг 3: Запуск поиска решений (см. Шаг 6 «Описание алгоритма решения»)

Пример полученного результата представлен на рисунке 2.24:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.24 Полученный результат

**2.4.2. Использование алгоритма 2 – Python для задачи о поиске кратчайшего пути алгоритма Дейкстры**

Пользователь может ввести данные несколькими способами: ввод данных вручную и с помощью импорта csv-файла. Перед выполнением действий для ввода данных необходимо запустить все окна кода для корректной работы программы.

**2.4.2.1. Ввод данных вручную**

Рассмотрим ввод данных вручную. При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: в строке выбора ввода данных выбрать цифру 1.



Рисунок 2.25 Выбор способа введения данных

Шаг 2: ввести все пары городов и расстояние между ними, десятичные дроби обозначая точкой (пример: 9.5), а блоки пар городов отделяя запятой (пример: Москва Минск 8,).

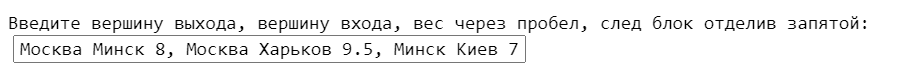


Рисунок 2.26 Ввод пар городов и расстояний между ними

Шаг 3: в поле ввода точки отбытия внести название города, который будет являться точкой отбытия. В поле ввода точки прибытия внести название города, который будет являться конечной точкой маршрута. В этом же окне будет выведен результат с наилучшим маршрутом и указанием его ценности.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 2.27 Выходные данные

Шаг 4: для просмотра матрицы запустить окно матицы смежности.

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 2.28 Весовая матрица

Шаг 5: для просмотра визуального представления маршрута запустить окно визуализации.

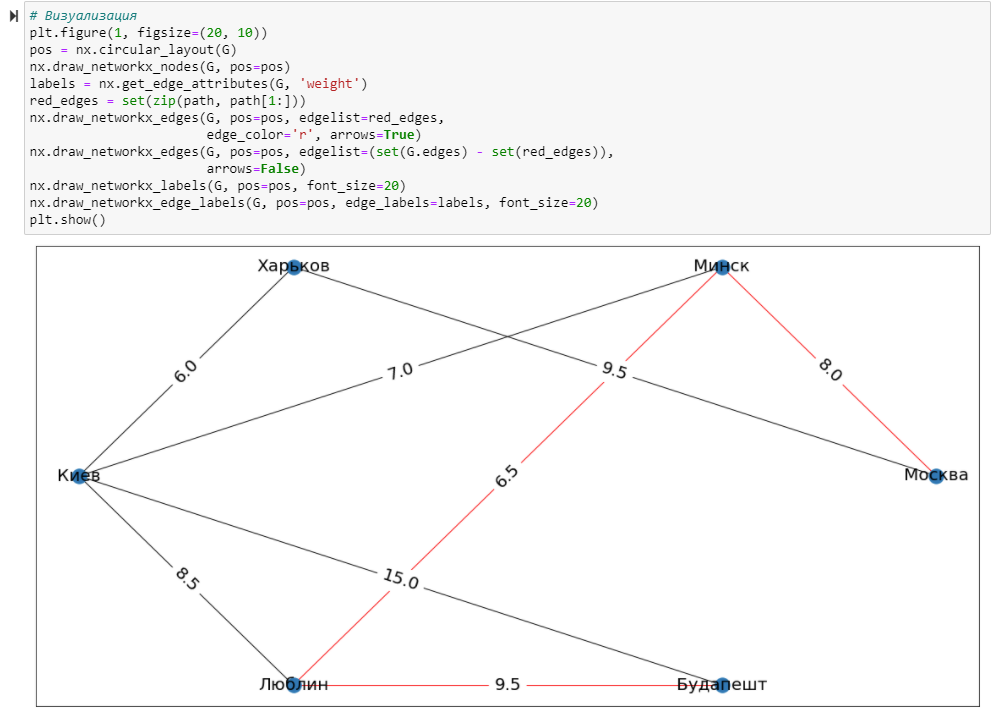


Рисунок 2.29 Блок визуализации

**2.4.2.2. Импортирование данных из CSV**

Рассмотрим импортирование данных из csv. Для данного способа необходимо поместить файл csv и файл ipynb в одну папку (любую кроме папки «Загрузки»). При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: в строке выбора ввода данных выбрать цифру 2.



Рисунок 2.30 Выбор способа ввода данных

Шаг 2: в поле ввода точки отбытия внести название города, который будет являться точкой отбытия. В поле ввода точки прибытия внести название города, который будет являться конечной точкой маршрута. В этом же окне будет выведен результат с наилучшим маршрутом и указанием его ценности.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 2.31 Полученный результат

Шаг 3: для просмотра матрицы и визуализации см. Шаг 4 и Шаг 5 способа «Ввод данных вручную».

**2.4.3. Использование алгоритма – Python для задачи коммивояжёра муравьиного алгоритма**

Пользователь может ввести данные следующими путями: ввод данных вручную, импортирование файлов из csv и случайная генерация. Во всех вариантах ввода данных кроме случайной генерации есть два варианта заполнения данных: координаты городов или расстояния между ними. Перед выполнением действий для ввода данных необходимо запустить все окна кода для корректной работы программы.

**2.4.3.1. Ввод данных вручную**

Рассмотрим ввод данных вручную. При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: в строке выбора ввода данных выбрать цифру 1.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.32 Выбор способа ввода

Шаг 2: в строке ввода количества вершин ввести количество городов, которые необходимо обойти (условие ввода: городов не может быть меньше 3 и больше 10).

Рисунок 2.33 Выбор количества вершин

Шаг 3: в строке способа ввода данных ввести точки вершины x и y.

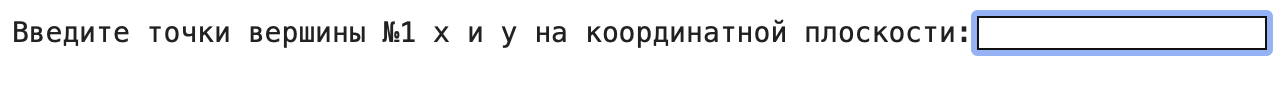


Рисунок 2.34 Ввод координат

Шаг 4: в строке ввода вершины на координатной прямой ввести через запятую координаты x и y разделяя десятичные числа точками. Если данные введены верно программа продублирует данные в виде матрицы.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.35 Введенные координаты

Шаг 5: в строке ввода данных записываем количество муравьёв.

Шаг 6: в строке ввода данных записываем количество итераций.

В итоге будет выведен результат с наилучшим маршрутом, указанием его ценности и визуализация маршрута с графиком количества итераций относительно пути.

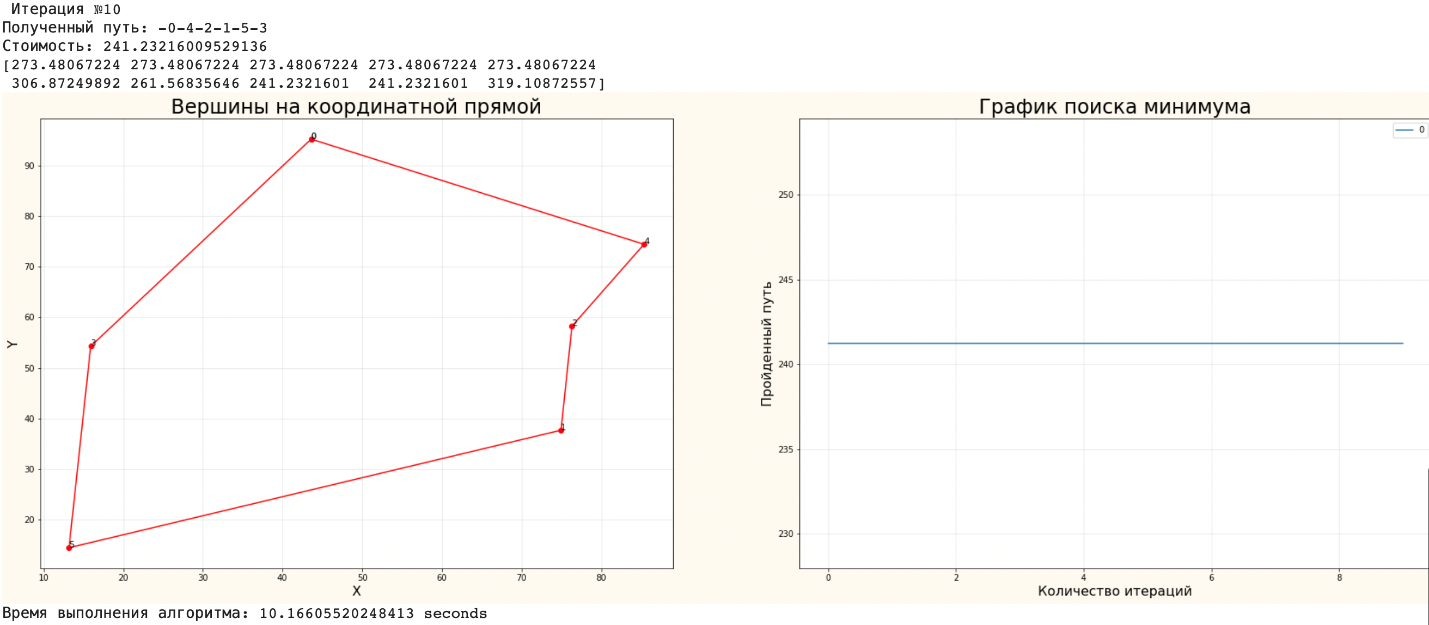


Рисунок 2.36 Полученные данные

**2.4.3.2. Импортирование файлов из CSV**

Рассмотрим импортирование файлов из CSV. При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: в строке выбора ввода данных выбрать цифру 3.

Шаг 2: в строке ввода пути к файлу указать название файла, если файл csv находится в одной папке с файлом ipynb (кроме папки «Загрузки») или путь к файлу, разделяя знаком «/».

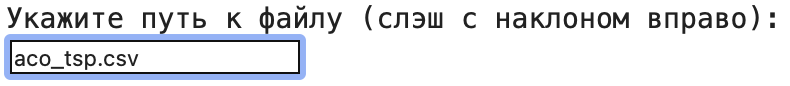


Рисунок 2.37 Ввод пути при условии нахождения кода и csv файла в одной папке

Шаг 3: в строке ввода данных записываем количество муравьёв.

Шаг 4: в строке ввода данных записываем количество итераций.

**2.4.3.3. Случайная генерация вершин**

Рассмотрим случайную генерацию. При запуске программы пользователь должен проделать следующие шаги:

Шаг 1: в строке выбора ввода данных выбрать цифру 2.

Итогом работы кода будет сгенерированные вершины и матрица весовых коэффициентов, заполненная случайными числами.

Шаг 2: в строке ввода данных записываем количество муравьёв.

Шаг 3: в строке ввода данных записываем количество итераций.

В итоге будет выведен результат с наилучшим маршрутом, указанием его ценности и визуализация маршрута с графиком количества итераций относительно пути.

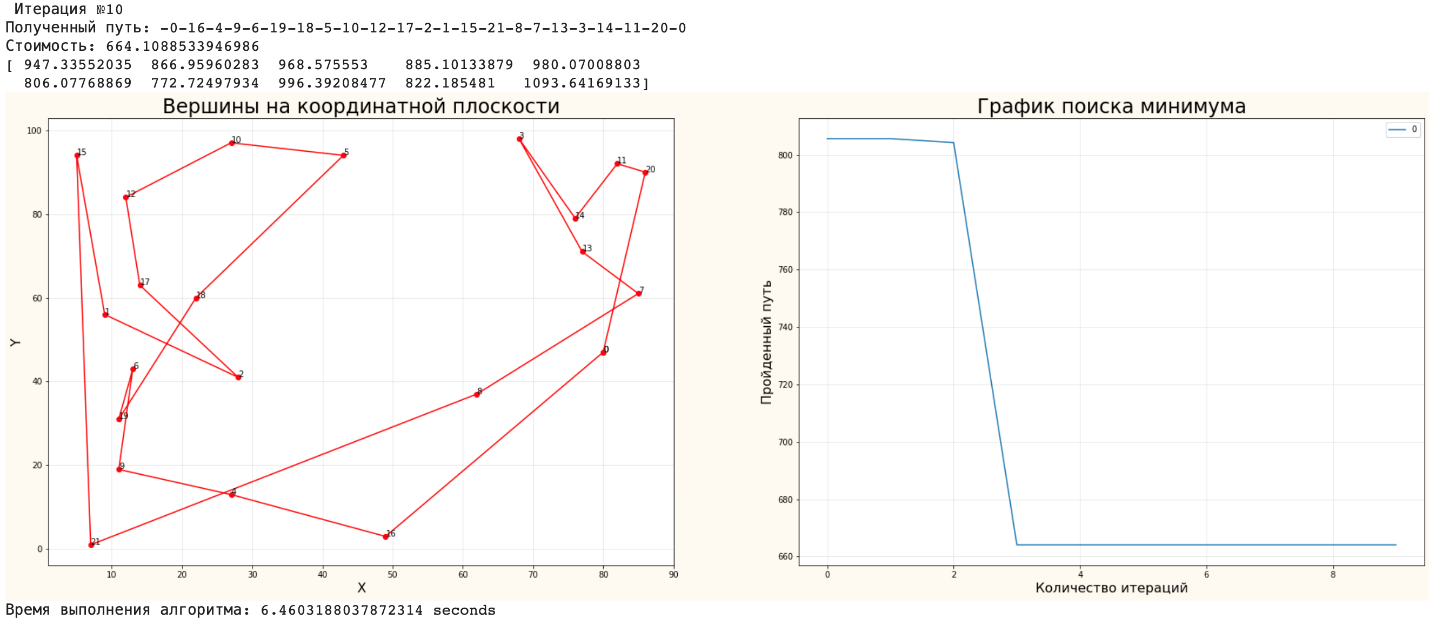


Рисунок 2.38 Полученные данные

**2.5. Архитектура решения**

**2.5.1. Архитектура решения алгоритма Дейкстры**

Для реализации алгоритма Дейкстры мы использовали следующие библиотеки:

* networkx - библиотека, специально предназначенная для работы с графами, создает сам граф как объект, визуализирует его
* pandas – библиотека для анализа данных, используется для импортирования данных из csv
* matplotlib - библиотека для визуализации данных двумерной (2D) графикой, отрисовывает граф

Опишем подробнее функции, которые содержатся в коде:

Мы реализуем алгоритм Дейкстры через функцию dijkstra\_algorithm

Входные данные:

* Переменная graph, объект библиотеки nx, в которой записан граф
* Переменная start\_node, содержащая точку отправления – узел, с которого начинается отсчет

Затем мы создаём два словаря:

* shorttest\_path будет хранить наиболее известную стоимость посещения каждого города на графике, начиная с start\_node. Вначале стоимость начинается с бесконечности, но мы будем обновлять значения по мере продвижения по графику.
* previous\_nodes будет хранить траекторию текущего наиболее известного пути для каждого узла.

То есть, мы используем словарь shorttest\_path чтобы сэкономить на посещении каждого узла и обновлять его по мере продвижения по графику, а previous\_nodes используем, чтобы сохранить кратчайший известный путь к найденному узлу.

Также мы использовали переменную max\_value для инициализации значения "бесконечности" непосещенных узлов, однако мы инициализируем значение начального узла 0.

Затем мы запустили цикл, так как мы работаем с алгоритмом Дейкстры, необходимо помнить, что данный алгоритм выполняется до тех пор, пока не посетит все узлы графика, поэтому мы представили это как условие выхода из цикла while. Находится узел с наименьшей оценкой.

После этого алгоритм посещает всех соседей узла, которые еще не посещены. Если новый путь к соседу лучше, чем текущий лучший путь, алгоритм вносит корректировки в словари shorttest\_path и previous\_nodes. После посещения всех его соседей мы можем пометить текущий узел как «посещенный». Затем мы возвращаем два словаря shorttest\_path и previous\_nodes.

Print\_result - функция, которая печатает результаты. Эта функция будет принимать два словаря, возвращаемые функцией dijskstra\_algorithm, а также имена начального и целевого узлов. Он будет использовать два словаря, чтобы найти лучший путь и сделать его оценку, т.е входными данными являются:

* словарь shorttest\_path
* словарь previous\_nodes
* start\_node-точка отправления, введенная с клавиатуры
* target\_node-точка прибытия, введенная с клавиатуры

**2.5.2. Архитектуры решения муравьиного алгоритма**

Для реализации муравьиного алгоритма мы использовали следующие библиотеки:

* Pandas – библиотека для анализа данных, используется для импортирования данных из csv
* NumPy - это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц.
* Time – модуль для работы со временем в Python.
* Scipy — это библиотека Python построенная на базе NumPy и позволяет управлять данными, а также визуализировать их с помощью разных высокоуровневых команд.
* Matplotlib - библиотека для визуализации данных двумерной (2D) графикой, рисует график.
* Sklearn - (Scikit-learn) это один из пакетов Python для Data Science и Machine Learning. Он содержит функции и алгоритмы для машинного обучения: классификации, прогнозирования или разбивки данных на группы.

Опишем подробнее функции, которые содержатся в коде:

Реализация муравьиного алгоритма происходит с помощью класса ACO\_TSP() и отдельных функций, которые требуются для промежуточного вычисления. Сам класс содержит функции вычисления и визуализации введенных данных.

Входные данные:

* Переменная func, по умолчанию содержащая функцию cal\_total\_distance, функция вне пределов класса ACO\_TSP(), которая вычисляет длину пути.
* Переменная n\_dim, содержащая количество вершин.
* Переменная size\_pop, содержащая количество муравьев.
* Переменная max\_iter, содержащая количество итераций.
* Матрица distance\_matrix, которая содержит векторы весовых коэффициентов.

Также, помимо этого, передаются матрицы distance\_matrix и points\_coordinate, последняя содержит точки X и Y на координатной плоскости.

Функции вне класса:

* def cal\_total\_distance(routine) – поитерационно передаются значения вершин.
* def main() - создание и запуск объекта алгоритма муравьиной колонии.

Исход 1. Выбор ручного ввода точек на координатной плоскости.

Создается список для хранения в нем списка списков точек X и Y.

* list\_vertex -  будет хранить в себе список списков с точками X и Y поочередно.
* points\_coordinate – на данном этапе список списков list\_vertex преобразуется в вектор с помощью метода array.

С помощью библиотеки SciPy и метода spatial.distance.cdist рассчитываются расстояния между точками и записываются в векторный список distance\_matrix, который непосредственно передается в класс. После ввода точек расчета, требуется ввести количество запускаемых муравьев и количество итераций.

Исход 2. Генерация случайных точек X и Y.

При выборе генерации случайных точек количество вершин выбирается в промежутке (5, 500) с помощью метода random.randint из библиотеки NumPy. При этом создается скелет для точек X и Y, а с помощью array, создается вектор, который передается в переменную distance\_matrix с методом spatial.distance.cdist для расчета расстояний.

Исход 3. Импорт CSV-файла с координатами городов.

При работе с CSV-файлами создается переменная list\_vertex, куда перебираются точки на координатной плоскости в виде списка списков. И с помощью метода spatial.distance.cdist рассчитываются расстояния между точками после преобразования в вектор.

Работа алгоритма класса.

После передачи в класс обязательных переменных для расчета инициируется конструктор класса \_\_init\_\_(). Конструируются основные переменные для расчета:

* self.func – переменная со ссылкой на функцию расчета расстояния
* self.n\_dim - количество городов
* self.size\_pop - количество муравьёв
* self.max\_iter - количество итераций
* self.alpha - коэффициент важности феромонов в выборе пути
* self.beta - коэффициент значимости расстояния
* self.rho - скорость испарения феромонов

Дальше следуют расчеты некоторых переменных, нужных для функции.

* self.Tau – матрица феромонов, которая будет обновляться каждую итерацию.
* self.Table – путь каждого муравья в определенном поколении
* self.y – общее расстояние пути муравья в определенном поколении
* self.generation\_best\_X, self.generation\_best\_Y – фиксирование лучших поколений

Основной функцией класса ACO\_TSP является функция def run, которая каждую итерацию производит расчет вероятности перехода без нормализации, создает список разрешённых вершин, из которых будет происходить выбор, обновляет расчет расстояния от одной точки до другой, фиксирует лучшее решение, а также подсчитывает феромон, который будет добавлен к ребру.

Входные для данной функции следующие:

* self – инициация самой себя
* max\_iter – максимальное количество итераций переданное во входные данные класса.

На каждой итерации визуализируется путь и стоимость выбранной муравьями дороги, реализуется все это с помощью библиотеки matplotlib.

**2.6. Тестирование**

**2.6.1. Тестирование задачи по поиску кратчайшего пути**

Для тестирования были использованы 5 датасетов, тесты производились следующими алгоритмами: поиск решений MS Excel и Python. Также была осуществлена проверка с помощью сервиса Онлайн-граф (<https://graphonline.ru/>). Результаты тестирования представлены в Таблице 2.1:

Таблица 2.1. Результаты тестирования задачи по поиску кратчайшего пути

|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Алгоритм 1. MS Excel | Алгоритм 2. Python | Сервис Онлайн-граф |
| Датасет 1.  Число вершин: 6  Матрица весовых коэффициентов:  Точка отправления: 1  Точка прибытия: 6 | Значение целевой функции:24  Матрица переменных: | Значение целевой функции:24  Оптимальный маршрут: 1-2-5-6 | Значение целевой функции:24  Оптимальный маршрут: 1-2-5-6 |
| Датасет 2.  Число вершин: 6  Матрица весовых коэффициентов:  Точка отправления: 1  Точка прибытия: 6 | Значение целевой функции: 11  Матрица переменных: | Значение целевой функции: 11  Оптимальный маршрут: 1-3-4-6 | Значение целевой функции: 11  Оптимальный маршрут: 1-3-4-6 |
| Датасет 3.  Число вершин: 5  Матрица весовых коэффициентов:  Точка отправления: 1  Точка прибытия: 5 | Значение целевой функции: 10  Матрица переменных: | Значение целевой функции: 10  Оптимальный маршрут: 1-3-4-5 | Значение целевой функции: 10  Оптимальный маршрут: 1-3-4-5 |
| Датасет 4:  Число вершин: 7  Матрица весовых коэффициентов:  Точка отправления: 1  Точка прибытия: 7 | Значение целевой функции: 19  Матрица переменных: | Значение целевой функции: 19  Оптимальный маршрут: 1-3-4-7 | Значение целевой функции: 19  Оптимальный маршрут: 1-3-4-7 |
| Датасет 5:  Число вершин: 8  Матрица весовых коэффициентов:  Точка отправления: 1  Точка прибытия: 8 | Значение целевой функции: 6  Матрица переменных: | Значение целевой функции: 6  Оптимальный маршрут: 1-2-4-8 | Значение целевой функции: 6  Оптимальный маршрут: 1-2-4-8 |

Заметим, что данные, полученные тремя разными способами, абсолютно идентичны. На основании этого можно сделать следующее заключение: полученное решение является единственным оптимальным маршрутом.

Однако стоит обратить внимание, что для графа, содержащего два маршрута с одинаково меньшим значением целевой функции, Python выведет только один маршрут, содержащий ближайшую по нумерации к начальному узлу промежуточную вершину.

Например, у нас есть граф, представленный на Рисунке 2.39. У данного графа значение целевой функции маршрута 1-2-4-8 и маршрута 1-5-7-8 равно шести. Все алгоритмы в качестве ответа выведут первый маршрут, так как порядок узла 2 ближе порядка узла 5.

Изображение выглядит как катается на лыжах

Автоматически созданное описание

Рисунок 2.39 Пример графа с двумя оптимальными путями

**2.6.2. Тестирование задачи по задаче коммивояжёра**

Для тестирования задачи коммивояжера, рассчитанной муравьиным алгоритмом, были использованы 3 датасета. Также производилась проверка алгоритма с помощью сервиса «Задача коммивояжёра онлайн» (<https://math.semestr.ru/kom/index.php>), который ищет оптимальный маршрут с помощью метода ветвей и границ.

Входными данными муравьиного алгоритма, как уже было сказано ранее, является координаты вершин, далее программа выводит матрицу расстояний (рис. 2.40).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.40 Матрица расстояний

Для проверки вводим эти данные на сайте (рис. 2.41).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.41 Матрица расстояний, введенная в сервис поиска решений

В ходе тестирования муравьиного алгоритма Python в датасетах 2 и 3 были введены различные значения количества муравьев и итераций.

В таблице 2.2 представлены результаты тестирования и проверки:

Таблица 2.2. Результаты тестирования задачи коммивояжёра

|  |  |  |
| --- | --- | --- |
| Входные данные | Муравьиный алгоритм Python | Проверка |
| Датасет 1.  Координаты вершин:  Вершина 1 – (43; 95)  Вершина 2 – (74; 37)  Вершина 3 – (76; 58)  Вершина 4 – (15; 54)  Вершина 5 – (85; 74)  Количество муравьев: 5  Количество итераций: 5 | Значение целевой функции:  197.45910246917103  Оптимальный маршрут: 0-4-2-1-3-0 | Значение целевой функции: 197.45910247  Оптимальный маршрут:  1-4-2-3-5-1 |
| Датасет 2.  Координаты вершин:  Вершина 1 – (1; 1)  Вершина 2 – (10; 24)  Вершина 3 – (55; 32)  Вершина 4 – (54; 22)  Вершина 5 – (76; 44)  Вершина 6 – (88; 10)   1. Количество муравьев: 5   Количество итераций: 5   1. Количество муравьев: 100   Количество итераций: 100 | 1. Значение целевой функции:   226.50004920488334  Оптимальный маршрут:  0-1-3-2-4-5-0   1. Значение целевой функции:   223.71032760769504  Оптимальный маршрут:  0-1-2-4-5-3-0 | Значение целевой функции:  223.71032759  Оптимальный маршрут:  1-4-6-5-3-2-1 |
| Датасет 3.  Координаты вершин:  Вершина 1 – (1; 1)  Вершина 2 – (3; 5)  Вершина 3 – (5; 6)  Вершина 4 – (4; 2)  Вершина 5 – (4; 7)  Вершина 6 – (7; 10)  Вершина 7 – (11; 5)  Вершина 8 – (12; 8)  Вершина 9 – (8; 12)  Вершина 10 – (9; 9)   1. Количество муравьев: 5   Количество итераций: 5   1. Количество муравьев: 100   Количество итераций: 100 | 1. Значение целевой функции:   34.866009906028964  Оптимальный маршрут:  0-1-2-4-5-8-9-7-6-3-0   1. Значение целевой функции:   36.71070811960286  Оптимальный маршрут:  0-3-1-2-4-5-8-9-7-6-0 | Значение целевой функции:  35,09550517  Оптимальный маршрут:  1-4-7-8-10-9-6-3-5-2-1 |

Проанализировав Таблицу 2.2, можно бы сделать следующие выводы: данные, полученные муравьиным алгоритмом, почти всегда совпадают с проверкой (датасеты 1 и 2), однако в датасете 3 программа нашла более оптимальный маршрут, а соответственно и меньшее значение целевой функции.

Также отметим, что чем больше количество муравьев и итераций, тем короче маршрут, то есть оптимальнее результат. Однако повышение значений муравьев и итераций влияют на скорость работы программы, например, в датасете 3, при входных данных, где количество муравьев и итераций были равны 5, время работы составило чуть больше одной секунды, если же поменять эти значения на 100, то программа выдаст результат через 31 секунду.

Заметим, что вывод оптимального маршрута у муравьиного алгоритма и проверки разный: в Python вершины нумеруются с нуля, а в сервисе «Задача коммивояжёра онлайн» с единицы, а также в обратном порядке.

**2.7. Заключение**

**Решение для предоставленной задачи о поиске кратчайшего пути мы реализовали с помощью двух алгоритмов. Первый алгоритм основан на надстройки Excel – поиск решений. Его преимуществами можно считать – наглядное представление всех массивов данных, простота редактирования и ввода новых данных, автоматическое обновление формул. Из недостатков – отсутствие пользовательского интерфейса, нет наглядного представления самого маршрута на графе, то есть визуализации и понятного для пользователя вывода.**

**Второй алгоритм, реализованный в Python при помощи библиотек networkx, matplotlib, имеет очевидное преимущество перед первым – последовательный вывод точек маршрута, его графическая визуализация. Эта программа также имеет перспективы для расширения функционала, так как можно прописать пользовательский интерфейс, добавить выгрузку данных в различных форматах и т.п.**

**Все используемые методы подсчета выдают идентичный результат – кратчайший маршрут от одной вершины до другой. Правильность кода была проверена на 5 случайно сформированных датасетах. Погрешность исключена, но одно исключение из правил в виде двух одинаковых маршрутов с наименьшей ценностью, было описано в пункте 2.6.1. Выводиться будет только один из них, то есть возможности просмотра альтернативного наименьшего пути нет.**

**Решение дополнительной задачи от заказчика было реализовано с помощью муравьиного алгоритма на языке Python. Результаты тестирования показывают высокую точность алгоритма, так как он может находить более оптимальный маршрут по сравнению с методом ветвей и границ (см. таблицу 2.2). Оптимальность достигается варьированием количества муравьев и количества итераций.**

**Данное решение имеет следующие преимущества для пользователя: визуализация пути муравьев после каждой итерации, обширный выбор способа введения данных, включающий ввод с клавиатуры, импорт из файла csv и случайную генерацию вершин. Также было выполнено условие введения координат конкретных городов. Есть перспективы развития данного решения, например создание алгоритма, который будет, как и в задаче поиска кратчайшего пути, принимать на вход весовую матрицу. Можно предусмотреть вывод полученных результатов в различных форматах данных.**

**При более глубоком изучении алгоритма есть возможность проследить закономерность между вводимым количеством муравьев, итераций и быстротой получения оптимального пути, что представляет интересный объект для исследования.**

**3. «Теория игр»**

**3.1. Постановка задачи (физическая модель)**

Наш заказчик – Gloria Jeans, российская компания, специализирующая на производстве джинсовой одежды и последующей ее продаже. В связи с уходом с российского рынка иностранных компаний, таких как Hugo Boss, в настоящее время стоит вопрос о делении данного сегмента производства между отечественными компаниями, количество потребителей в данном сегменте равняется 50 млн человек. Именно поэтому Gloria Jeans обратилась к нам с задачей о выборе оптимальной стратегии в условиях изменения рынка. Стратегиями для данной компании является производство таких вещей, как джинсы, джинсовые платья и джинсовые юбки.

Также компания-заказчик не исключает тот факт, что возможно такое деление рынка России с другим брендом «5 КармаNов», когда Gloria Jeans выходит на рынок европейской части России (30 млн человек), а 5 КармаNов занимает азиатскую часть (20 млн. человек). При этом последняя компания специализируется на продаже джинсов, джинсовых шортов и джинсовых рубашек.

Аналитики Gloria Jeans составили оперативный план, в котором определили количество покупателей при условии, что компания-заказчик будет действовать самостоятельно (таблица 3.1), а также при условии, когда Gloria Jeans и 5 КармаNов объединятся (таблица 3.2, 3.3).

Таблица 3.1. Платежная матрица Gloria Jeans при самостоятельном выходе на рынок

|  |  |  |  |
| --- | --- | --- | --- |
| А\B | **1** | **2** | **3** |
| 1 | 20 | 25 | 45 |
| 2 | 5 | 10 | 20 |
| 3 | 10 | 15 | 15 |

Таблица 3.2. Платежная матрица Gloria Jeans при совместном выходе на рынок

|  |  |  |  |
| --- | --- | --- | --- |
| А | **1** | **2** | **3** |
| **1** | 25 | 23 | 15 |
| **2** | 20 | 17 | 10 |
| **3** | 12 | 15 | 10 |

Таблица 3.3. Платежная матрица 5 КармаNов при совместном выходе на рынок

|  |  |  |  |
| --- | --- | --- | --- |
| В | 1 | 2 | 3 |
| **1** | 17 | 15 | 12 |
| **2** | 15 | 12 | 10 |
| **3** | 10 | 7 | 13 |

Также заказчик хотел бы получить данные о том, какую продукцию необходимо продавать при разном состоянии рынка. Вероятность наступления рецессии равна 0,15, стагнации – 0,25, оживления – 0,5, подъема – 0,1. В Таблице 3.4 представлено количество клиентов, при выборе разных стратегий с учетом состояния рынка.

Таблица 3.4. Платежная матрица Gloria Jeans при различном состоянии рынка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | П1 | П2 | П3 | П4 |
| А1 | 35 | 35 | 45 | 50 |
| А2 | 20 | 25 | 25 | 40 |
| А3 | 10 | 7 | 10 | 14 |

**3.2. Математическая модель**

Рассмотрим математическую модель для **антагонистической игры:**

Стратегия Максимин:

, (1)

где 1 – нижняя цена игры, a – весовая матрица игрока А

Стратегия Минимакс:

, (2)

где 2 – верхняя цена игры, a – весовая матрица игрока А

В случае если :

– оптимальное решение игры «Седловая точка»

Смешанные стратегии:

*,* (3)

Максимальная величина входного/выходного потока:

, (4)

где заменённая вероятность p

Цена игры:

(5)

Значения смеси стратегий:

, (6)

Рассмотрим математическую модель для **биматричной игры**.

Целевая функция – сумма выигрышей обоих игроков:

, (7)

Где ситуации равновесия:

(8)

(9)

Ограничения:

(10)

(11)

(12)

(13)

(14)

(15)

Рассмотрим математическую модель для **игр с природой в условии риска**.

Критерий Байеса:

(16)

Критерий Лапласа:

, (17)

Критерий Гермейера:

(18)

Приведем математическую модель для **игр с природой в условиях неопределенности.**

Критерий пессимизма или «минимин»:

(19)

Критерий оптимизма или «максимакс»:

(20)

Критерий Вальда:

(21)

Критерий Гурвица:

(22)

Критерий Сэвиджа:

(23)

**3.3. Алгоритм решения**

**3.3.1. Алгоритм 1. Решение с помощью Поиска решения в Excel**

**3.3.1.1. Решение задачи о нахождении выигрышной стратегии для антагонистической игры**

**3.3.1.1.1. Описание входных данных**

В качестве входных данных мы имеем: платежную матрицу игрока А, которая содержит в себе значения, получаемые при выборе какой-либо стратегии.

**3.3.1.1.2. Описание алгоритма решения**

*А) Оптимальные чистые стратегии*

Поиск оптимальной чистой стратегии реализуются принципами максимина и минимакса. Принцип максимина предполагает выбор той стратегии, при которой минимальный выигрыш для различных стратегий максимален. Принцип минимакса ищет для игрока В минимальный из максимальных проигрышей.

По каждой строке находится минимальное значение с помощью функции МИН(), по каждому столбцу – максимальное с помощью МАКС(). После этого в получившемся столбце min находится максимум (МАКС()), а в строке с записанными в список максимумов по столбцам, находится минимум (МИН()).

Полученные максимум среди минимумов и минимум среди максимумов являются верхней и нижней ценой игры соответственно.

*Б) Оптимальная смесь стратегий.*

Данный алгоритм принимает на вход массив переменных, ограничения, согласно математической модели, целевую функцию, которую нужно оптимизировать до минимума. Далее выбирается метод решения, в данном случае, используется метод ОПГ. Надстройка "Поиск решения" изменяет значения в ячейках переменных решения согласно пределам ячеек ограничения, а также выводит значение целевой функции. После того, как решение было найдено, в ячейках «Смесь стратегий» и «Цена игры» выводятся значения долей стратегий, которые необходимо применить для получения оптимального результата, и сам максимальный результат соответственно.

*Подробный пример реализации алгоритма для пункта А*

Шаг 1: в соответствии с описанием задачи составляем платежную матрицу для игрока А. Вводим соответствующие значения в матрицу (рис. 3.1).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.1. Платежная матрица

Шаг 2: находим минимальное значение по строке и максимальное значение по столбцу (рис. 3.2).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.2. Результаты нахождения минимума и максимума

Шаг 3: в полученном столбце min находим максимальное значение при помощи функции МАКС(), в строке max находим минимальное значение при помощи функции МИН() (рис. 3.3)

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.3. Результаты вычисления чистой стратегии по принципу максимина

*Подробный пример реализации алгоритма для пункта Б*

Шаг 1: платежную матрицу, представленную на Рисунке 3.1, транспонируем (рис.3.4).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.4. Транспонированная платежная матрица

Шаг 2: создаем ячейки для переменных и вводим в них любые значения (рис. 3.5).

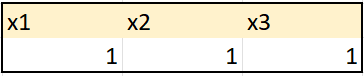


Рисунок 3.5. Ячейки переменных

Шаг 3: записываем целевую функцию согласно математической модели, используя встроенную функцию Excel СУММ(). Данное действие представлено на Рисунке 3.6:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.6. Создание целевой функции

Шаг 4: формируем ограничения согласно математической модели. Реализация данного шага представлена на Рисунке 3.7:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.7. Установка ограничений

Шаг 5: записываем в Excel формулу для нахождения цены игры (рис. 3.8).

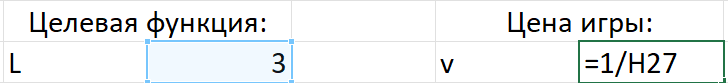


Рисунок 3.8. Создание ячейки цены игры

Шаг 6: записываем формулы для нахождения вероятностей (рис. 3.9), а также устанавливаем для этих ячеек процентный формат.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.9. Создание ячеек вероятности

Шаг 7: применяем пакет анализа «Поиск решений» (рис. 3.10), заполняем следующие параметры: целевую функцию, переменные, ограничения) и используем метод решения «Поиск решения линейных задач симплекс-методом».

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.10. Заполненное окно "Поиск решений"

**3.3.1.1.3. Описание выходных данных**

Выходными данными для пункта А) являются максимальное значение среди минимальных по строкам (нижняя цена игры) и минимальное значение среди максимальных по столбцам (верхняя цена игры). Результаты были представлены на Рисунке 3.3.

Выходные данные для пункта В): сначала надстройка сообщает о наличии решения, так как не всегда оно может быть найдено.

Алгоритм выводит:

* Значение целевой функции L;
* Цену игры *v*;
* Значения переменных;
* Смесь стратегий в процентах.

**3.3.1.2. Решение задачи о нахождении выигрышной стратегии для биматричной игры**

**3.3.1.2.1. Описание входных данных**

В качестве входных данных мы имеем: платежные матрицы игроков А и В, которые содержат в себе значения, получаемые при выборе какой-либо стратегии.

**3.3.1.2.2. Описание алгоритма решения**

*А) Оптимальные чистые стратегии.*

Для решения данной задачи воспользуемся поиском равновесия по Нэшу. Для этого на матрице игрока А находим максимальные значения по столбцам (наиболее выгодные стратегии игрока В), а в платежной матрице В находим максимальное значение по строкам (наиболее выгодные стратегии для игрока А). Далее сравниваем матрицы и выделяем элементы, которые совпадают в обоих матрицах.

Таким образом, в тех ячейках, где отмеченные выгодные стратегии варианты совпали в обоих матрицах и наблюдается ситуация равновесия по Нэшу.

*Б) Оптимальная смесь стратегий.*

Данный алгоритм принимает на вход массив переменных, ограничения, согласно математической модели, целевую функцию, которую нужно оптимизировать до минимума. Далее выбирается метод решения, в данном случае, используется метод ОПГ. Надстройка "Поиск решения" изменяет значения в ячейках переменных решения согласно пределам ячеек ограничения, а также выводит значение целевой функции. После того, как решение было найдено, в ячейках «Смесь стратегий» и «Цена игры» выводятся значения долей стратегий, которые необходимо применить для получения оптимального результата, и сам максимальный результат соответственно.

*Подробный пример реализации алгоритма для пункта А*

Шаг 1: в соответствии с описанием задачи составляем платежные матрицы для игроков А и В. Вводим соответствующие значения в матрицы (рис. 3.11).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.11. Платежные матрицы двух игроков

Шаг 2: находим максимальное значение по столбцам в матрице игрока B (рис. 3.12).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.12. Максимальные значения по столбцам

Шаг 3: находим максимальное значение по строкам в матрице игрока А (рис. 3.13).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.13. Максимальное значение по строкам

Шаг 4: пользователь отмечает найденные значения в матрице (рис. 3.14).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.14. Равновесие по Нэшу для двух игроков

*Подробный пример реализации алгоритма для пункта Б*

Шаг 1: вводим матрицы, представленные на Рисунке 3.11.

Шаг 2: определим искомые переменные p – вероятности производства определенной продукции игроком A, а также переменные q – вероятности производства определенной продукции игроком В. Также определим для этих ячеек процентный формат и введем рандомные значения. Данное действие представлено на Рисунке 3.15.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.15. Ячейки переменных

Шаг 3: далее произведем дополнительные вычисления для каждого элемента матриц, создав новые матрицы HA и НВ (рис. 3.16), также суммируем все значения матрицы.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.16. Создание матриц НА и НВ

Шаг 4: задаем целевую функцию путем суммирования двух значений цены игры.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.17. Определение целевой функции

Шаг 5: вычислим 𝑎𝑖𝑗𝑞𝑗 и 𝑏𝑖𝑗𝑝𝑖, затем просуммируем полученные значение для матрицы А и В по строкам и столбцам соответственно. Транспонируем полученные значения суммы для матрицы В (рис. 3.18).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.18. Дополнительные вычисления

Шаг 6: запишем все ограничения (рис. 3.19). Суммы 𝑎𝑖𝑗𝑞𝑗 по строкам меньше либо равны цене игры компании А. Суммы 𝑏𝑖𝑗𝑝𝑖 по столбцам меньше либо равны цены игры компании В. Сумма вероятностей каждого игрока должна равняться единице.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.19. Ограничения для биматричной игры

Шаг 7: применяем пакет анализа «Поиск решений» (рис. 3.20), заполняем следующие параметры: целевую функцию, переменные, ограничения) и используем метод решения «Поиск решения нелинейных задач методом ОПГ».

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.20. Поиск решения для биматричной игры

**3.3.1.2.3. Описание выходных данных**

А) Выходными данными являются две платежные матрицы игроков А и В, в которых отмечены максимумы. При условии, когда максимумы совпали в обоих матрицах, наблюдается равновесие по Нэшу.

Б) Сначала надстройка сообщает о наличии решения, так как не всегда оно может быть найдено.

Алгоритм выводит:

* Значение целевой функции L;
* Цену игры *v* обоих игроков;
* Значения переменных обоих игроков;
* Смесь стратегий в процентах для обоих игроков.

**3.3.1.3. Решение задачи о принятии оптимального решения в условиях риска**

**3.3.1.3.1. Описание входных данных**

В качестве входных данных мы имеем:

* платежную матрицу игрока А, которая содержит в себе значения, получаемые при выборе какой-либо стратегии;
* вероятности наступления природных состояний.

**3.3.1.3.2. Описание алгоритма решения**

А) Критерий Лапласа.

Сначала находится вероятности наступления каждого из состояний природы, согласно критерию Лапласа, они принимаются как равновероятностные.

Далее находим средневзвешенное значение выигрыша каждой стратегии (по каждой строке) с помощью СУММПРОИЗВ(). Среди полученных значений выбирается максимальное.

Б) Критерий Байеса.

В ходе реализации данного алгоритма находится средневзвешенное значение выигрыша каждой стратегии (по каждой строке) с помощью СУММПРОИЗВ(). Далее выбирается наибольшее значение по полученному столбцу. Таким образом определяется максимальная прибыль по критерию Байеса.

В) критерий Гермейера в чистых стратегиях.

Согласно математической модели, составляем матрицу Гермейера. Ценой игры является максимин Гермейера, для ее нахождения сначала находится минимум по строкам, а затем, в получившемся столбе, находится максимум. Полученное значение делим на вероятность состояния природы. В результате находится цена игры – оптимальное значени по критерию Гермейера.

Г) критерий Гермейера в смешанных стратегиях.

Используя полученную в предыдущем пункте матрицу Гермейера, согласно математической модели, задаем ограничения и целевую функцию. Далее выбирается метод решения, в данном случае, используется метод ОПГ. Надстройка "Поиск решения" изменяет значения в ячейках переменных решения согласно пределам ячеек ограничения, а также выводит значение целевой функции. После того, как решение было найдено, в ячейках «Смесь стратегий» и «Цена игры» выводятся значения долей стратегий и цена игры по Гермейеру.

*Примерный пример реализации пункта А*

Шаг 1: в соответствии с описанием задачи составляем платежную матрицу для игрока А. Вводим соответствующие значения в матрицу (рис. 3.21).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.21. Матрица для Лапласа

Шаг 2: вычисляем значение вероятностей каждого из состояний природы. Данное действие представлено на Рисунке 3.22.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.22. Вычисление вероятностей для Лапласа

Шаг 3: далее находим средневзвешенное для каждой строчки (рис. 3.23).

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 3.23. Вычисление средневзвешенных для Лапласа

Шаг 4: находим максимум в столбце средневзвешенных значений (рис. 24).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.24. Нахождение цены игры по критерию Лапласа

*Примерный пример реализации пункта Б*

Шаг 1: в соответствии с описанием задачи составляем платежные матрицы для игрока А и природы П. Вводим соответствующие значения в матрицы (рис. 3.25).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.25. Матрицы для Байеса

Шаг 2: аналогично шагу 3 из примера реализации пункта А, также находим средневзвешенное для каждой строчки (рис. 3.26).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.26. Вычисление средневзвешенных для Байеса

Шаг 3: также аналогично шагу 4 из предыдущего примера находим максимум по столбцу средневзвешенных (рис. 3.27).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.27. Нахождение цены игры по критерию Байеса

*Примерный пример реализации пункта В*

Шаг 1: аналогично шагу 1 из предыдущего примера вводим значения в матрицы.

Шаг 2: строим матрицу Гермейера, состоящую из элементов Гермейера. Данное действие представлено на Рисунке 3.28:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.28. Матрица Гермейера

Шаг 3: находим минимально значение по каждой строке матрицы Гермейера (рис. 3.29).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.29. Поиск минимального значения по строке

Шаг 4: в столбце с минимальными значениями находим максимум (рис. 3.30).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.30. Нахождение цены игры по критерию Гермейера

Шаг 5: далее находим цену игры. Для этого делим найденный максимум на вероятность (рис. 3.31).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.31. Нахождение цены игры

*Примерный пример реализации пункта Г*

Шаг 1: проделываем первые два шага из предыдущего примера реализации пункта Г.

Шаг 2: создаем ячейки для переменных и вводим в них любые значения (рис. 3.32).

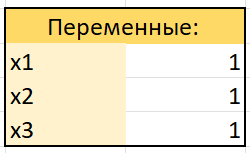


Рисунок 3.32. Создание ячеек переменных

Шаг 3: записываем целевую функцию согласно математической модели, используя встроенную функцию Excel СУММ (). Данное действие представлено на Рисунке 3.33:



Рисунок 3.33. Создание целевой функции

Шаг 4: формируем ограничения согласно математической модели. Реализация данного шага представлена на Рисунке 3.34:

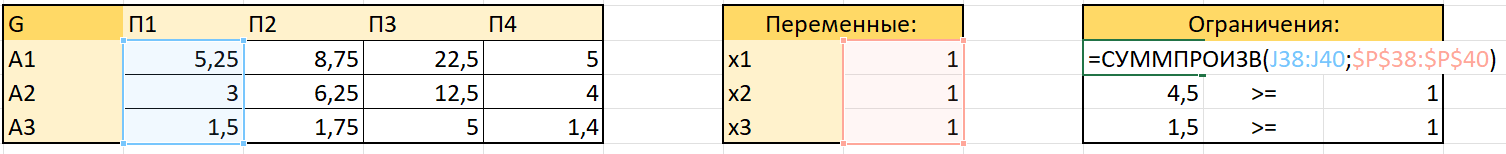


Рисунок 3.34. Установка ограничений

Шаг 5: записываем формулу для нахождения цены игры (рис. 3.35).

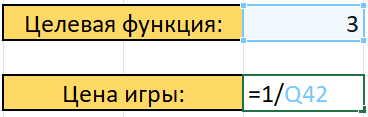


Рисунок 3.35. Установка цены игры

Шаг 6: записываем формулы для нахождения вероятностей (рис. 3.36), а также устанавливаем для этих ячеек процентный формат.

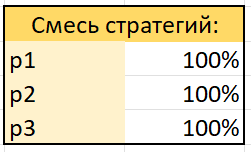


Рисунок 3.36. Создание ячеек вероятности

Шаг 7: применяем пакет анализа «Поиск решений» (рис. 3.37), заполняем следующие параметры: целевую функцию, переменные, ограничения) и используем метод решения «Поиск решения линейных задач симплекс-методом».

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.37. Заполненное окно поиска решений

Шаг 8: далее произведем дополнительные вычисления для нахождения цены игры. С помощью функции СУММПРОИЗВ() перемножим столбцы исходной матрицы со столбцом найденных вероятной. Полученные значения нужно умножить на соответствующую вероятность природы.

Шаг 9: вычислим цену игры, с помощью функции СУММ() просуммируем полученные в результате выполнения Шага 8 значения (рис. 3.38).

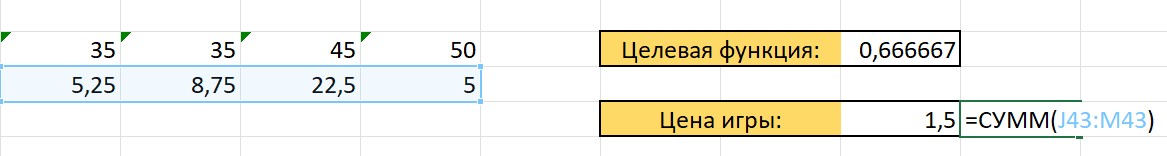


Рисунок 3.38. Нахождение цены игры

**3.3.1.3.3. Описание выходных данных**

В результате выходными данными являются:

* значение цены игры по критерию Лапласа;
* значение цены игры по критерию Байеса;
* значение цены игры для матрицы Гермейера и исходной матрицы;
* значение целевой функции L;
* значения переменных;
* смесь стратегий в процентах.

**3.3.1.4. Решение задачи о принятии оптимального решения в условиях неопределённости**

**3.3.1.4.1. Описание входных данных**

В качестве входных данных мы имеем платежную матрицу игрока А, которая содержит в себе значения, получаемые при выборе какой-либо стратегии. Для критерия Гурвица помимо платежной матрицы входными данными также являются значения риска игрока.

**3.3.1.4.2. Описание алгоритма решения**

А) критерий пессимизма;

Для нахождения заданного критерия применяется метод минимина. Для этого находим минимальные значения по строкам матрицы, а затем среди получившихся значений выбираем минимальное. При пессимистическом сценарии мы получаем самые большие издержки.

Б) критерий оптимизма;

Для нахождения заданного критерия применяется метод максимакса. Для этого находим максимальные значения по строкам матрицы, а затем среди получившихся значений выбираем максимальное.

В) критерий Вальда;

Для нахождения заданного критерия применяется метод минимакса. Для этого находим минимальные значения по строкам матрицы, а затем среди получившихся значений выбираем максимальное.

Г) критерий Гурвица;

Согласно математической модели, реализуем формулу 20. Для этого находим значения для каждой стратегии и каждой альфы, после этого находим максимальное значение в получившийся столбцах.

Д) критерий Сэвиджа.

Строим матрицу рисков (матрицу Севиджа), где находим значения согласно формуле 21. Далее реализуем метод максимина для назождения минимальной упущенной выгоды.

*Примерный пример реализации пункта А*

Шаг 1: в соответствии с описанием задачи составляем платежную матрицу для игрока А. Вводим соответствующие значения в матрицу (рис. 3.39).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.39. Исходная матрица

Шаг 2: с помощью функции МИН() находим минимум по каждой строчке. Данное действие представлено на Рисунке 3.40:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.40. Нахождение минимума по строкам

Шаг 3: находим минимальное значение по получившемуся столбцу (рис. 3.41).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.41. Нахождение цены игры по критерию "пессимизма"

*Примерный пример реализации пункта Б*

Шаг 1: используя матрицу, введенную в шаге 1 предыдущего пункта, с помощью функции МАКС() находим максимум по каждой строке (рис. 3.42).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.42. Нахождение максимума по строкам

Шаг 2: находим максимальное значение по получившемуся столбцу (рис. 3.43).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.43. Нахождение цены игры по критерию "оптимизма"

*Примерный пример реализации пункта Б*

Шаг 1: используя матрицу, введенную в первом шаге пункта А, выполняем второй шаг того же пункта.

Шаг 2: аналогично шагу 2 предыдущего пункта находим максимум по получившемуся столбцу. Данное действие продемонстрированно на Рисунке 3.44:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.44. Нахождение цены игры по критерию Вальда

*Примерный пример реализации пункта В*

Шаг 1: аналогично шагу 1 пункта А, вводим платежную матрицу.

Шаг 2: находим максимумы и минимумы по каждой строке с помощью функций МАКС() и МИН() соответственно (рис. 3.45).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 45. Нахождение максимумов и минимумов

Шаг 3: создаем строку Alpha, в которую вводим значения риска игрока (В данном примере значениями от 0 до 1 с шагом 0,1).

Шаг 4: в получившуюся матрицу вводим формулу, представленную на рисунке 3.46, а после растягиваем ее на всю матрицу.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.46. Расчеты для реализации критерия Гурвица

Шаг 5: определяем максимальное значение по каждому столбцу со значением Alpha (рис. 3.47).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.47. Нахождение максимума по столбцу

Шаг 6: пользователь окрашивает ячейку исходной матрицы, значение которой равно полученному значению в столбце «макс».

Шаг 7: для визуализации полученных данных построим график. Полученная гистограмма представлена на Рисунке 3.48:

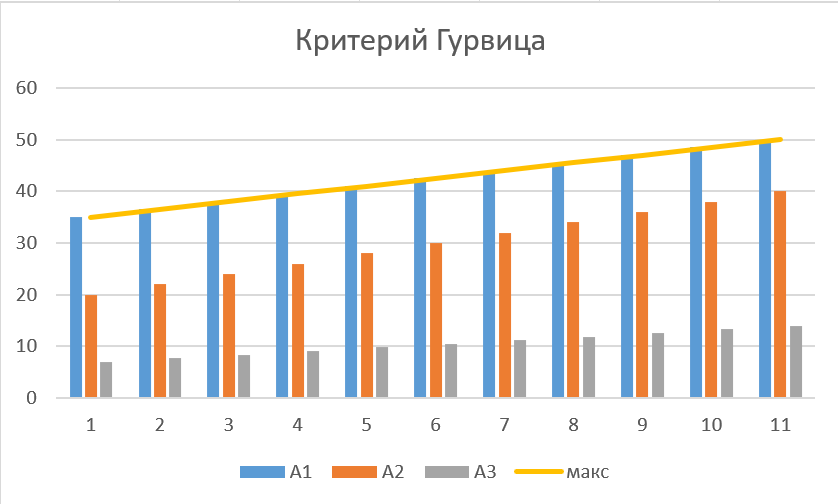


Рисунок 3.48. Гистограмма распределения оптимальной цены по критерию Гурвица

*Примерный пример реализации пункта Д*

Шаг 1: аналогично шагу 1 пункта А, вводим платежную матрицу.

Шаг 2: с помощью функции МАКС() определяем максимум по каждому столбцу (рис. 3.49)

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.49. Определение максимума по столбцу

Шаг 3: составляем матрицу Сэвиджа, используя формулу, представленную на рисунке 3.50:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.50. Формирование матрицы Сэвиджа

Шаг 4: находим максимальное значение по каждой строке (рис. 3.51).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.51. Нахождение максимального значения

Шаг 5: в полученном столбце максимумов необходимо найти минимум. Та стратегия, которая будет находиться на строчке с минимумом, и является оптимальной.

**3.3.1.4.3. Описание выходных данных**

Выходными данными являются:

* Цена игры по критерию пессимизма;
* Цена игры по критерию оптимизма;
* Цена игры по критерию Вальда;
* Линейная свертка склонности к риску по критерию Гурвица, гистограмма максимумов линейной свертки по критерию Гурвица;
* Величина минимальной недополученной прибыли по критерию Сэвиджа.

**3.3.2. Алгоритм 2. Решения в Python**

**3.3.2.1. Решение задачи о нахождении выигрышной стратегии для антагонистической игры**

**3.3.2.1.1. Описание входных данных**

Входными данными являются следующие компоненты:

* Количество стратегий каждого игрока
* Название стратегий каждого игрока
* Матрица коэффициентов при выборе определенной стратегии игроком А (платежная матрица игрока А)

Вышеперечисленные данные должны быть записаны в списки. Матрица представляет собой список со вложенными списками. Это требуется для удобного перебора элементов матрицы в циклах при нахождении чистой стратегии и для передачи матрицы в функцию nash\_equilibrium в качестве аргумента для нахождения равновесия по Нэшу в смешанных стратегиях.

**3.3.2.1.2. Описание алгоритма решения**

*Чистые стратегии*

Реализуя принципы Максимин и Минимакс (формулы 1 и 2), циклы for перебирают элементы столбцов и строк матрицы. Принцип максимина предполагает выбор той стратегии, при которой наш минимальный выигрыш для различных стратегий максимален. Принцип минимакса ищет для игрока В минимальный из максимальных проигрышей.

По каждой строке находится минимальное значение, по каждому столбцу – максимальное. После этого среди записанных в список минимумов по строкам, также при помощи цикла for среди них находится максимум, а среди записанных в список максимумов по столбцам, находится минимум.

Полученные максимум среди минимумов и минимум среди максимумов являются верхней и нижней ценой игры.

*Смешанные стратегии*

Нахождение смешанных стратегий сводится к алгоритму симплекс-метода решения пары взаимодвойственных задач линейного программирования. Алгоритм симплекс-метода заключается в том, что из множества вершин, принадлежащих границе множества решений системы неравенств, выбирается такая вершина, в которой значение целевой функции достигает максимума (минимума). По определенному правилу находится первоначальный опорный план (некоторая вершина области ограничений). Проверяется, является ли план оптимальным. Если да, то задача решена. Если нет, то переходим к другому улучшенному плану - к другой вершине.

Таким образом, решая задачу линейного программирования, можно найти оптимальную стратегию игрока A. Чтобы найти оптимальную стратегию игрока B, нужно провести аналогичные действия, с той разницей, что игрок B стремится не максимизировать, а минимизировать выигрыш (по сути - проигрыш), а значит, не минимизировать, а максимизировать величину 1/V, т.к. V → min.

Введенная пользователем весовая матрица передается в функцию nash\_equilibrium, в которой последствии эта матрица со сформированными ограничениями принимается функцией linprog, после чего непосредственно находятся искомые стратегии.

**3.3.2.1.3. Описание выходных данных**

Выходными данными алгоритма являются:

* Исходная матрица с найденными минимумами по строкам и максимумами по столбцам
* Оптимальная чистая стратегия для игрока А (название);
* Цена игры для игрока А при выборе чистой оптимальной стратегии (число);
* Оптимальная чистая стратегия для игрока Б (название);
* Цена игры для игрока Б при выборе чистой оптимальной стратегии (число);
* Таблица смешанных стратегий для игрока А-только для квадратных матриц;
* Цена игры для игрока А при выборе смешанной оптимальной стратегии-только для квадратных матриц;
* Таблица смешанных стратегий для игрока B-только для квадратных матриц

Пример выходных данных представлен на рисунке 3.52:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.52. Выходные данные

**3.3.2.2. Решение задачи о нахождение выигрышной стратегии для биматричной игры**

**3.3.2.2.1. Описание входных данных**

Входными данными являются следующие компоненты:

* Количество стратегий игрока А
* Количество стратегий игрока Б
* Список стратегий игрока А;
* Список стратегий игрока Б;
* Матрица весовых коэффициентов для игрока А;
* Матрица весовых коэффициентов для игрока Б.

**3.3.2.2.2. Описание алгоритма решения**

Нахождение смеси стратегий, так же как и чистых стратегий для биматричной игры сводится к передачи двух матриц в переменную с методом NashGame библиотеки NashPy, после чего используется метод support\_enumeration, выводящий списками оптимальную стратегию для каждого игрока в чистых стратегиях и также смесь оптимальных стратегий через запятую, если таковые имеются.

**3.3.2.2.3. Описание выходных данных**

Выходными данными являются:

* Список вида array с отмеченными единицей выигрышными стратегиями для каждого игрока
* Список вида array со смешанными стратегиями

**3.3.2.3. Решение задачи о принятии оптимального решения в условиях риска**

**3.3.2.3.1. Описание входных данных**

Входными данными являются следующие компоненты:

* Количество стратегий игрока
* Количество состояний природы
* Матрица коэффициентов при выборе определенной стратегии игроком А (весовая матрица игрока А)
* Вектор вероятностей (для критерия Байеса и Гермейера)

Вышеперечисленные данные должны быть записаны в списки. Матрица представляет собой список со вложенными списками. Это требуется для удобного перебора элементов матрицы в циклах при нахождении средневзвешенного значения по критерию Лапласа и Байеса, а также для передачи матрицы в функцию nash\_equilibrium в качестве аргумента для нахождения оптимальной стратегии по критерию Гермейера.

**3.3.2.3.2. Описание алгоритма решения**

*Критерий Лапласа*

Реализуя принцип нахождения оптимальной стратегии по критерию Лапласа (формула 17), цикл for создает список, состоящий из равных вероятностей, которые рассчитываются как 1/n, где n-число стратегий игрока А. Далее в цикле перебираются строки введенной матрицы и перемножаются на вектор вероятностей, суммы элементов получившихся массивов являются искомым средневзвешенным значением по каждой строке матрицы. Оптимальной стратегией является стратегия, по которой средневзвешенное – максимальное. Поиск максимального значения также реализуется через перебор элементов списка всех средневзвешенных значений.

*Критерий Байеса*

В отличие от критерия Лапласа, для Критерия Байеса пользователь вводит вектор вероятностей для каждого из состояний природы. Далее алгоритм повторяет алгоритм нахождения средневзвешенного значения по Лапласу, только в цикле for строки матрицы перемножаются уже на введенный пользователем вектор вероятностей. Суммы элементов получившихся массивов являются искомым средневзвешенным значением по каждой строке матрицы. Оптимальной стратегией является стратегия, по которой средневзвешенное – максимальное. Поиск максимального значения также реализуется через перебор элементов списка всех средневзвешенных значений.

*Критерий Гермейера (чистые стратегии)*

Для нахождения оптимальной стратегии игрока А по критерию Гермейера создается отдельная матрица, элементы которой являются произведением aij искомой матрицы и соответствующей для ее j состояния природы вероятности. Реализуется это умножение через цикл for, который перебирает каждую строку искомой матрицы и перемножает на вектор вероятностей. Далее находится минимум по каждой строке матрицы Гермейера (также через цикл for) и уже по описанному принципу максимина (формула 1) определяется оптимальная стратегия игрока и цена игры по матрице Гермейера. Для нахождения реальной цены игры по искомой матрице игрока, значение цены игры по Гермейеру делится на значение вероятности, соответствующей положению элемента цены игры в матрице игрока.

*Критерий Гермейера (смешанные стратегии)*

Алгоритм нахождения оптимальных смешанных стратегий схож с алгоритмом нахождения смешанных стратегий для антагонистической игры, также используется функция nash\_equilibrium, только в качестве аргумента принимается матрица Гермейера. Рассчитывается искомая смесь стратегий в процентах через функцию linprog библиотеки scipy.optimize. Сам принцип работы кода подробно описан в пункте №3.5 «Архитектура».

**3.3.2.3.3. Описание выходных данных**

Выходными данными алгоритма являются:

* Матрица с оценкой по критерию Лапласа;
* Оптимальная стратегия игрока А по критерию Лапласа (название);
* Цена игры оптимальной стратегии по критерию Лапласа (число);
* Матрица с оценкой по критерию Байеса
* Оптимальная стратегия игрока А по критерию Баеса (название);
* Цена игры оптимальной стратегии по критерию Баеса (число);
* Матрица Гермейера;
* Оптимальна чистая стратегия игрока по критерию Гермейера (название);
* Цена игры оптимальной чистой стратегии по критерию Гермейера (название);
* Таблица оптимальных смешанных стратегий для игрока А по критерию Гермейера (стратегия и процент выделения ресурсов на нее) – только для квадратных матриц;
* Цена игры для игрока А при выборе смешанной оптимальной стратегии по критерию Гермейера (число)-только для квадратных матриц.

Пример выходных данных можно видеть на рисунке 3.53:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.53. Выходные данные

**3.3.2.4. Решение задачи о принятии оптимального решения в условиях неопределённости**

**3.3.2.4.1. Описание входных данных**

Входными данными являются:

* Количество стратегий игрока
* Количество состояний природы
* Матрица коэффициентов при выборе определенной стратегии игроком А (весовая матрица игрока А)

Вышеперечисленные данные должны быть записаны в списки. Матрица представляет собой список со вложенными списками. Это требуется для удобного перебора элементов матрицы в циклах при нахождении минимального и максимального значения по критерию «оптимизма», «пессимизма» и Вальда, а также для удобного вывода таблиц и графика линейной свертки Гурвица.

**3.3.2.4.2. Описание алгоритма решения**

*Критерий оптимизма, пессимизма, Вальда*

Реализуя принцип нахождения оптимальной стратегии по критерию пессимизма, оптимизма и критерию Вальда (формулы 17,n,n), цикл for создает списки, состоящие из минимальных и максимальных значений по строкам весовой матрицы игрока А. Далее в зависимости от критерия, среди них находится максимум или минимум, которые и являются ценой игры и соответствуют оптимальной стратегии: для оптимизма – максимум среди максимумов, для пессимизма - минимум среди минимумов, для Вальда – минимум среди максимумов (принцип Минимакс). Поиск максимального или минимального значения реализуется через перебор элементов списка максимумов/минимумов.

*Критерий Гурвица*

Реализуя принцип нахождения оптимальной стратегии по критерию Гурвица, мы создаем матрицу линейной свертки с шагом 0,1. Линейная свертка заполняется при помощи цикла while, со вложенным циклом for, которые параллельно осуществляют перебор элементов искомой матрицы и вектора, содержащего коэффициенты риска, осуществляя необходимые математические операции для заполнения матрицы с линейной сверткой. После этого строится диаграмма максимумов линейной свертки для каждого значения коэффициента риска при помощи цикла for.

*Критерий Сэвиджа*

Для нахождения оптимальной стратегии игрока А по критерию Сэвиджа создается отдельная матрица, элементы которой являются разностью максимума по j столбцу и aij искомой матрицы. Реализуется это вычитание через цикл for со вложенным циклом for, который перебирает каждую строку искомой матрицы, а в каждой строке отнимает от максимума по столбцу каждый элемент строки. Далее находится максимум по каждой строке матрицы Сэвиджа (также через цикл for) и определяется минимальная недополученная прибыль, то есть минимум среди максимумов по строкам.

**3.3.2.4.3. Описание выходных данных**

Выходными данными алгоритма являются:

* Оптимальная стратегия игрока А по критерию пессимизма (название);
* Цена игры оптимальной стратегии по критерию пессимизма (число);
* Оптимальная стратегия игрока А по критерию оптимизма (название);
* Цена игры оптимальной стратегии по критерию оптимизма (число);
* Оптимальная стратегия игрока А по критерию Вальда (название);
* Цена игры оптимальной стратегии по критерию Вальда (число);
* Линейная свертка склонности к риску по критерию Гурвица шагом 10%;
* Гистограмма (диаграмма) максимумов линейной свертки по критерию Гурвица
* Матрица Сэвиджа
* Оптимальная стратегия игрока А по критерию Сэвиджа (название);
* Величина минимальной недополученной прибыли по критерию Сэвиджа (число);

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.54. Выходные данные

**3.4. Использование алгоритма**

**3.4.1. Использование алгоритма 1 – поиск решения Excel антагонистической игры**

Шаг 1: пользователь при необходимости корректирует имя компании-конкурента и товары.

Шаг 2: если пользователь желает рассчитать выгоду только для одного вида продукции, следует заменить значения в основной платёжной матрице со спросом российского рынка на различные виды товаров в чистых стратегиях.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.55. Платёжная матрица

Выигрыш (оптимальная доля рынка в млн. чел.) для обеих компаний будет высчитан автоматически и помещен в столбце «Цена игры». Исходя из цены игры находится наиболее выгодный для выпуска товар.

Шаг 3: если пользователь желает рассчитать выгоду по всем видам продукции, следует заменить значения в основной платёжной матрице со спросом российского рынка на различные виды товаров в чистых стратегиях и транспонировать её. Количество искомых переменных должно совпадать с количеством видов товара компании пользователя.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.56. Транспонированная платёжная матрица

Шаг 4: запуск поиска решений (см. «3.3.1.1.2. Описание алгоритма решения»)

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.57. Выходные данные

В столбце «Смесь стратегий» показана оптимальная доля бюджета в процентах, распределённая между товарами компании. В ячейке «Цена игры» указана доля рынка (млн. чел.), отведённая компании при использовании самой выгодной стратегии.

**3.4.2. Использование алгоритма 1 – поиск решений Excel биматричной игры**

Шаг 1: пользователь при необходимости корректирует имя компании-конкурента, товары и сегмент рынка.

Шаг 2: пользователь заменяет значения в основных платёжных матрицах со спросом российского рынка на различные виды товаров в чистых стратегиях.

Шаг 3: если пользователь желает рассчитать выгоду только для одной пары товаров, следует продублировать значения в основных платёжных матрицах в платёжные матрицы в чистых стратегиях.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.58. Чистые стратегии

Шаг 4: находим сочетания товаров с максимальной выгодой (в соответствии с колонкой и строкой «макс») и ищем пересечения в стратегиях (равновесие по Нэшу). Совпадающие сочетания товаров будут наиболее выгодными.

Шаг 5: если пользователь желает рассчитать выгоду по всем видам продукции, следует заменить значения в платёжных матрицах смешанных стратегий.

Шаг 6: запуск поиска решений (см. «3.3.1.2.2. Описание алгоритма решения»). Ограничения, цена игры и смесь стратегий будут посчитаны автоматически.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.59. Выходные данные

В столбце и строке «Смесь стратегий» показана оптимальная доля бюджета в процентах, распределённая между товарами компаний. В ячейках «Цена игры» указана доля рынка (млн. чел.), отведённая компаниям при использовании самой выгодной стратегии.

**3.4.3. Использование алгоритма 1 – поиск решения Excel для задачи о принятии оптимального решения в условиях риска**

Шаг 1: пользователь при необходимости корректирует имя компании, выбирает нужный критерий и товары в зависимости от поставленной задачи. Если необходимы дополнительные колонки в платежной матрице, то пользователь должен кликнуть на столбец справа от последнего, выбрать «Вставить…», затем во всплывшем окне выбрать «столбец», те же действия для строки.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.60. Платёжная матрица

При добавлении новых строк или столбцов нужно иметь в виду, что также нужно корректировать и формулы, использование которых подробно описано в «3.3.1.3.2. Описание алгоритма решения», также необходимо понимать, что формулы меняются из-за различных критериев (Лапласа, Байеса, Гермейера в чистых стратегиях и смешанных).

Шаг 2: пользователь очищает матрицу искомых переменных, если она была заполнена предыдущими значениями.

Шаг 3: пользователь вносит необходимые данные в платежную матрицу. Все функции уже внесены и растянуты (см. шаг 1). Пользователь получает цену игры, выбирая максимум по средневзвешенному значению выигрыша каждой стратегии.

Шаг 4: стоит обратить внимание на нахождение смешанной стратегии по Гермейру. Все перечисленные шаги выполняется также, кроме того, что при использовании данного критерия, пользователь будет использовать метод ОПГ, надстройку «Поиск решения», использование которой подробно описано в «3.1.3.2. Описание алгоритма решения».

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.61. Смешанные стратегии

**3.4.4. Использование алгоритма 1 – поиск решений Excel для задачи о принятии оптимального решения в условиях неопределённости**

Шаг 1: пользователь при необходимости корректирует имя компании, выбирает нужный критерий и товары в зависимости от поставленной задачи. Если необходимы дополнительные колонки в платежной матрице, то пользователь должен кликнуть на столбец справа от последнего, выбрать «Вставить…», затем во всплывшем окне выбрать «столбец», те же действия для строки.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.62. Платёжная матрица

При добавлении новых строк или столбцов нужно иметь в виду, что также нужно корректировать и формулы, использование которых подробно описано в «3.3.1.4.2. Описание алгоритма решения», также необходимо понимать, что формулы меняются из-за различных критериев (пессимизма, оптимизма, Вальда, Гурвица, Сэвиджа)

Шаг 2: пользователь очищает матрицу искомых переменных, если она была заполнена предыдущими значениями.

Шаг 3: пользователь вносит необходимые данные в платежную матрицу. Все функции уже внесены и растянуты (см. шаг 1). Пользователь получает цену игры, в зависимости от выбранного критерия.

К примеру:

Критерий пессимизма: для нахождения заданного критерия применяется метод минимина.

Критерий оптимизма: для нахождения заданного критерия применяется метод максимина.

Критерий Вальда: для нахождения заданного критерия применяется метод минимакса.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.63. Критерий Вальда

Описание выполнения и алгоритма остальных критериев приведено также в «3.3.1.4.2. Описание алгоритма решения».

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.64. Критерий Гурвица

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.65. Критерий Сэвиджа

**3.4.5. Использование алгоритма 2 – Python для решения антагонистической игры**

Пользователь может ввести данные путём введения с клавиатуры. Перед выполнением действий для ввода данных необходимо запустить все окна кода с библиотеками для корректной работы программы.

Шаг 1: в строке ввода количества строк ввести количество строк платёжной матрицы.



Рисунок 3.66. Количество строк

Шаг 2: аналогичное действие со строкой ввода количества столбцов.



Рисунок 3.67. Количество столбцов

Шаг 3: в строках названия стратегии игрока А ввести названия товаров компании заказчика, количество строк будет соответствовать количеству строк платёжной матрицы.

Рисунок 3.68. Название стратегии А

Шаг 4: в строках названия стратегии игрока В ввести названия товаров компании конкурента, количество строк будет соответствовать количеству столбцов платёжной матрицы.



Рисунок 3.69. Название стратегии А

Шаг 5: в строке ввода количества ожидаемых клиентов по строке через запятую ввести количество покупателей, соответствующее ожидаемому спросу на n товар при стратегиях конкурента m1, m2, m3, ... Количество строк будет соответствовать количеству строк платёжной матрицы.



Рисунок 3.70. Количество ожидаемых клиентов

В итоге будет выведена платёжная матрица, оптимальная чистая стратегия для игрока А и В, таблица смешанных стратегий для игрока А и В и цена игры для игрока А при выборе смешанной оптимальной стратегии.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.71. Выходные данные

Смешанные стратегии выводятся только в случае, если матрица квадратная.

**3.4.6. Использование алгоритма 2 – Python для решения биматричной игры**

Пользователь может ввести данные путём введения с клавиатуры. Перед выполнением действий для ввода данных необходимо запустить все окна кода с библиотеками для корректной работы программы.

Шаг 1: в строке ввода количества стратегий игрока А ввести количество строк платёжной матрицы.



Рисунок 3.72. Количество стратегий А

Шаг 2: аналогичное действие со строкой ввода стратегий игрока В.



Рисунок 3.73. Количество стратегий В

Шаг 3: в строках названия стратегии игрока А ввести названия товаров компании заказчика.

Рисунок 3.74. Название стратегии А

Шаг 4: в строках названия стратегии игрока В ввести названия товаров компании конкурента.



Рисунок 3.75. Название стратегии В

Шаг 5: в строке ввода количества ожидаемых клиентов по строке через запятую ввести количество покупателей, соответствующее ожидаемому спросу на n товар клиента при стратегиях конкурента m1, m2, m3, ...



Рисунок 3.76. Количество ожидаемых клиентов А

Шаг 6: в строке ввода количества ожидаемых клиентов по строке через запятую ввести количество покупателей, соответствующее ожидаемому спросу на n товар конкурента при стратегиях заказчика m1, m2, m3, ...



Рисунок 3.77. Количество ожидаемых клиентов В

**3.4.7. Использование алгоритма 2 – Python для решения игр с природой в условиях риска**

Пользователь может ввести данные путём введения с клавиатуры. Перед выполнением действий для ввода данных необходимо запустить все окна кода с библиотеками для корректной работы программы.

Шаг 1: в строке ввода количества стратегий ввести количество строк платёжной матрицы.



Рисунок 3.78. Количество стратегий А

Шаг 2: аналогичное действие со строкой ввода состояний природы, количество состояний будет соответствовать количеству столбцов платёжной матрицы.



Рисунок 3.79. Количество состояний природы

Шаг 3: в строках ввода названия стратегии компании А ввести названия товаров компании заказчика, количество строк будет соответствовать количеству строк платёжной матрицы. 

Рисунок 3.80. Название стратегии А

Шаг 4: в строках ввода названий состояния природы ввести названия возможных экономических ситуаций (прим. рецессия, стагнация, оживление, подъём).



Рисунок 3.81. Название состояния природы

Шаг 5: в строках ввода количества ожидаемых клиентов по строке через запятую ввести количество покупателей, соответствующее ожидаемому спросу на n товар при введённых состояниях природы. Количество строк будет соответствовать количеству строк платёжной матрицы.



Рисунок 3.82. Количество ожидаемых клиентов

Шаг 6: в строке ввода вектора вероятностей ввести через запятую ожидаемую вероятность наступления введённых состояний природы разделяя десятичные числа точками.



Рисунок 3.83. Вектор вероятностей

В итоге будут выведены матрицы, в пункте «Оценка по критерию Лапласа» оптимальная чистая стратегия компании А по критерию Лапласа и цена игры игрока А при выборе чистой оптимальной стратегии по критерию Лапласа. В пункте «Оценка по критерию Байеса» будут выведены оптимальная стратегия компании А по критерию Байеса и цена игры игрока А при выборе чистой оптимальной стратегии. В пункте «Матрица Гермейера» оптимальная чистая стратегия игрока А по критерию Гермейера, цена игры с оптимальной чистой стратегии по критерию Гермейера, таблица оптимальных смешанных стратегий для игрока А по критерию Гермейера и цена игры для игрока А при выборе смешанной оптимальной стратегии.

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 3.84. Выходные данные 1

**Изображение выглядит как стол

Автоматически созданное описание**

Рисунок 3.85. Выходные данные 2

Смешанные стратегии по Гермейеру будут выводиться только если матрица квадратная.

## **3.4.8. Использование алгоритма 2 – Python для решения игр с природой в условиях неопределенности**

Пользователь может ввести данные путём введения с клавиатуры. Перед выполнением действий для ввода данных необходимо запустить все окна кода с библиотеками для корректной работы программы.

Все шаги аналогичны шагам из предыдущего алгоритма за исключением ввода вектора вероятностей (см. 3.4.7. «Использование алгоритма 2 – Python для решения игр с природой в условиях риска»).

В итоге будут выведены линейная свертка склонности к риску по критерию Гурвица, платежная матрица для критерия Сэвиджа, матрица Сэвиджа, оптимальная стратегия игрока А по критерию Сэвиджа, величина минимальной недополученной прибыли по критерию Сэвиджа и диаграмма максимумов линейной свертки по критерию Гурвица.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.86. Выходные данные 1

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.87. Выходные данные 2

# **3.**[**5. Архитектура**](#_Toc98442922) **решения алгоритма на языке программирования Python**

Для реализации всех имеющихся алгоритмов решения мы использовали следующие библиотеки:

* Pandas – библиотека для обработки и анализа структурированных данных, с её помощью мы получаем доступ по индексу или по заданному имени строки;
* Copy – библиотека выполняет функцию глубокого копирования объектов, то есть возвращает полную копию объекта;
* Math – библиотека для использования математических формул;
* NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов (включая матрицы), поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами;
* Scipy — это библиотека Python построенная на базе NumPy и позволяет управлять данными, а также визуализировать их с помощью разных высокоуровневых команд, необходима для функции linprog из пакета scipy.optimize, который содержит в себе множество алгоритмов оптимизации;
* NashPy – это библиотека Python, используемая для вычисления равновесий в стратегических играх для двух игроков.

**3.**[**5.1. Архитектура решения антагонистической игры**](#_Toc98442922)

Опишем подробнее функции и методы библиотек, которые содержатся в коде:

linprog () помогает минимизация линейной объективной функции с линейными ограничениями равенства и неравенства;

append() добавляет элементы в массив;

array() создает массив из списка, так как с массивами можно проводить числовые операции с большим объемом информации гораздо быстрее и эффективнее, чем со списками, а также используется для создание единичных векторов;

pd.DataFrame позволяет нам сформировать таблицу и внести туда данные, через доступ по индексу или имени строки;

copy.deepcopy позволяет нам создать новый составной объект, и затем рекурсивно вставить в него копии объектов, находящихся в оригинале;

df\_matrix позволяет нам обратиться к конкретному столбцу матрицы;

df\_min помогает нам определить минимальное значение;

df\_max помогает нам определить максимальное значение;

- np.transpose() данная функция используется для транспонирования матрицы со знаком минус;

def nash\_equilibrium данная функция принимает весовую матрицу и возвращает значение игры и оптимальные стратегии первого и второго игроков.

Использование циклов for или while позволяет нам внести данные в матрицу или список.

## **3.**[**5.2. Архитектура решения биматричной**](#_Toc98442922)**задачи**

## **3.**[**5.3. Архитектура решения**](#_Toc98442922) **игр с природой в условиях риска**

Для реализации всех имеющихся алгоритмов решения мы использовали библиотеки (см. [5. Архитектура](#_Toc98442922) решения алгоритма на языке программирования Python).

Опишем подробнее функции, которые содержатся в коде, и отличаются от приведённых в «[5.1. Архитектура решения антагонистической игры»:](#_Toc98442922)

функции np.array и np.sum введены для расчета средневзвешанного для критериев Лапласа и Байеса;

mingerm.append, pd.DataFrame и цикл for in range позволяют нам создать список минимума значений по строкам матрицы Гермейера;

def nash\_equilibrium в данной задаче помогает найти реальную цены игры в чистых стартегиях определим местонахождение минимаксимума;

## **3.**[**5.4. Архитектура решения игр с природой в условиях неопределенности**](#_Toc98442922)

Для реализации всех имеющихся алгоритмов решения мы использовали библиотеки (см. [5. Архитектура](#_Toc98442922) решения алгоритма на языке программирования Python).

Дополнительно была введена библиотека matplotlib.pyplot для построения диаграммы максимумов линейной свертки по критерию Гурвица.

Опишем подробнее функции, которые содержатся в коде, и отличаются от приведённых в «[5.1. Архитектура решения антагонистической игры» и в «5.3 Архитектура решения игр с природой в условиях риска»:](#_Toc98442922)

maxvalue.append, pd.DataFrame и цикл for in range позволяют нам создать список максимума значений по строкам матрицы;

df\_g.T транспонирует матрицу по критерию Гурвица под требуемый вид;

с помощью функций из библиотеки matplotlib.pyplot мы строим диаграмму максимумов линейной свертки по критерию Гурвица;

# **3.6. Тестирование**

## **3.6.1. Тестирование задачи о нахождении выигрышной стратегии для антагонистической игры**

Для тестирования были использованы 3 датасета, тесты производились следующими алгоритмами: MS Excel и Python. Результаты тестирования представлены в Таблице 5.

Таблица 3.1 Тестирование антагонистических игр

|  |  |  |
| --- | --- | --- |
| Входные данные | Алгоритм 1. MS Excel | Алгоритм 2. Python |
| Датасет 1.  Платежная матрица: | Цена игры игрока А при выборе чистой оптимальной стратегии: 20  Цена игры игрока B при выборе чистой оптимальной стратегии: 20  Таблица смешанных стратегий для игрока А:  1–100%  2–0%  3–0%  Таблица смешанных стратегий для игрока B:  – | Оптимальная чистая стратегия для игрока А: 1  Цена игры игрока А при выборе чистой оптимальной стратегии: 20  Оптимальная чистая стратегия для игрока B: 1  Цена игры игрока B при выборе чистой оптимальной стратегии: 20  Таблица смешанных стратегий для игрока А:  1–100%  2–0%  3–0%  Таблица смешанных стратегий для игрока B:  1–100%  2–0%  3–0% |
| Датасет 2.  Платежная матрица | Цена игры игрока А при выборе чистой оптимальной стратегии: 200  Цена игры игрока B при выборе чистой оптимальной стратегии: 300  Таблица смешанных стратегий для игрока А:  1–21%  2–3%  3–76%  Таблица смешанных стратегий для игрока B:  – | Оптимальная чистая стратегия для игрока А: 3  Цена игры игрока А при выборе чистой оптимальной стратегии: 200  Оптимальная чистая стратегия для игрока B: 3  Цена игры игрока B при выборе чистой оптимальной стратегии: 300  Таблица смешанных стратегий для игрока А:  1–21%  2–3%  3–76%  Таблица смешанных стратегий для игрока B:  1–13%  2–6%  3–27% |
| Датасет 3.  Платежная матрица | Цена игры игрока А при выборе чистой оптимальной стратегии: 100  Цена игры игрока B при выборе чистой оптимальной стратегии: 100  Таблица смешанных стратегий для игрока А:  1–0%  2–0%  3–79%  4–21%  Таблица смешанных стратегий для игрока B:  – | Оптимальная чистая стратегия для игрока А: 3  Цена игры игрока А при выборе чистой оптимальной стратегии: 100  Оптимальная чистая стратегия для игрока B: 4  Цена игры игрока B при выборе чистой оптимальной стратегии: 100  Таблица смешанных стратегий для игрока А:  1–0%  2–0%  3–79%  4–21%  Таблица смешанных стратегий для игрока B:  1–0%  2–0%  3–0%  4–0% |

Как мы можем увидеть, цены игры чистых стратегий получились идентичны. Однако преимущества алгоритма Python заключается в том, что он вывел еще и номер оптимальной стратегии и ее название, а также рассчитал смесь стратегий для игрока В.

## **3.6.2. Тестирование задачи о нахождении выигрышной стратегии для биматричной игры**

## **3.6.3. Тестирование задачи о принятии оптимального решения в условиях риска**

Для тестирования были использованы 3 датасета, тесты производились следующими алгоритмами: MS Excel и Python. Результаты тестирования представлены в Таблице 6.

Таблица 3.2 Тестирование игр в условиях риска

|  |  |  |
| --- | --- | --- |
| Входные данные | Алгоритм 1. MS Excel | Алгоритм 2. Python |
| Датасет 1.  Платежная матрица:  Вероятность наступления природы:  0.15, 0.25, 0.5, 0.1 | Цена игры по Байесу: 41,5  Цена игры по Лапласу: 41,25  Цена игры по Гермейеру в чистых стратегиях: 35  Смесь стратегий по Гермейеру:  1–100%  2–0%  3–0%  Цена игры по Гермейеру в смешанных стратегиях: 5/41,5 | Оптимальная стратегия компании А по критерию Баеса: 1  Цена игры по Байесу: 41,5  Оптимальная стратегия компании А по критерию Лапласа: 1  Цена игры по Лапласу: 41,25  Цена игры по Гермейеру в чистых стратегиях: –  Смесь стратегий по Гермейеру: –  Цена игры по Гермейеру в смешанных стратегиях: 41,5 |
| Датасет 2.  Платежная матрица  Вероятность наступления природы:  0.15, 0.4, 0.35, 0.1 | Цена игры по Байесу: 330  Цена игры по Лапласу: 311,75  Цена игры по Гермейеру в чистых стратегиях: 12  Смесь стратегий по Гермейеру:  1–0%  2–0%  3–0%  4–100%  Цена игры по Гермейеру в смешанных стратегиях: 20/290,3 | Оптимальная стратегия компании А по критерию Баеса: 3  Цена игры по Байесу: 330  Оптимальная стратегия компании А по критерию Лапласа: 4  Цена игры по Лапласу: 311,75  Цена игры по Гермейеру в чистых стратегиях: –  Смесь стратегий по Гермейеру:  1–0%  2–0%  3–0%  4–100%  Цена игры по Гермейеру в смешанных стратегиях: 20/290,3 |
| Датасет 3.  Платежная матрица  Вероятность наступления природы:  0.1, 0.1, 0.8 | Цена игры по Байесу: 52  Цена игры по Лапласу: 45  Цена игры по Гермейеру в чистых стратегиях: 4  Смесь стратегий по Гермейеру:  1–33%  2–0%  3–67%  Цена игры по Гермейеру в смешанных стратегиях: 4/46,667 | Оптимальная стратегия компании А по критерию Баеса: 1  Цена игры по Байесу: 52  Оптимальная стратегия компании А по критерию Лапласа: 1  Цена игры по Лапласу: 45  Цена игры по Гермейеру в чистых стратегиях: –  Смесь стратегий по Гермейеру:  1–33%  2–0%  3–67%  Цена игры по Гермейеру в смешанных стратегиях: 4/46,667 |

Из анализа данной таблицы можно заключить, что оба метода: Python и Excel выдают идентичные значения.

Заметим, что в расчётах Excel сложно автоматически выделить оптимальную стратегию, в отличие от Python. Однако из-за некоторых технических сложностей, в методе Pyton возникают трудности с вычислением стратегий по Гермейеру для матриц не с квадратной размерностью. Преимуществом метода решения с помощью Pythoп является наличие более систематизированных резльтатов, готовых к анализу, в то время как решение с помощью Excel является более простым для проведения самих расчётов.

## **3.6.4. Тестирование задачи о принятии оптимального решения в условиях неопределенности**

Для тестирования были использованы 3 датасета, тесты производились следующими алгоритмами: MS Excel и Python. Результаты тестирования представлены в Таблице 7, а также на рисунках 3.88–3.93.

Таблица 3.3 Тестирование игр в условиях неопределенности

|  |  |  |
| --- | --- | --- |
| Входные данные | Алгоритм 1. MS Excel | Алгоритм 2. Python |
| Датасет 1.  Платежная матрица: | Цена игры по пессимизму: 7  Цена игры по оптимизму: 50  Цена игры по Вальда: 35  Величина минимальной недополученной прибыли по критерию Севиджа: 0 | Оптимальная стратегия: 3  Цена игры по пессимизму: 7  Оптимальная стратегия: 1  Цена игры по оптимизму: 50  Оптимальная стратегия: 1  Цена игры по Вальда: 35  Оптимальная стратегия: 1  Величина минимальной недополученной прибыли по критерию Севиджа: 0 |
| Датасет 2.  Платежная матрица | Цена игры по пессимизму: 0  Цена игры по оптимизму: 564  Цена игры по Вальда: 150  Величина минимальной недополученной прибыли по критерию Севиджа: 132 | Оптимальная стратегия: 2  Цена игры по пессимизму: 0  Оптимальная стратегия: 4  Цена игры по оптимизму: 564  Оптимальная стратегия: 4  Цена игры по Вальда: 150  Оптимальная стратегия: 3  Величина минимальной недополученной прибыли по критерию Севиджа: 132 |
| Датасет 2.  Платежная матрица | Цена игры по пессимизму: -10  Цена игры по оптимизму: 70  Цена игры по Вальда: 40  Величина минимальной недополученной прибыли по критерию Севиджа: 5 | Оптимальная стратегия: 1  Цена игры по пессимизму: -10  Оптимальная стратегия: 3  Цена игры по оптимизму: 70  Оптимальная стратегия: 2  Цена игры по Вальда: 40  Оптимальная стратегия: 3  Величина минимальной недополученной прибыли по критерию Севиджа: 5 |

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.88. Линейная свертка Excel датасет 1

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.89. Линейная свертка Python датасет 1

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.90. Линейная свертка Excel датасет 2

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.91. Линейная свертка Python датасет 2

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.92. Линейная свертка Excel датасет 2

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.93. Линейная свертка Python датасет 3

Заметим, что данные, полученные двумя разными способами, абсолютно идентичны. На основании этого можно сделать следующее заключение: алгоритмы работаю одинаково. Однако, опять же, преимущество алгоритма Python заключается в выводе номера и названии оптимальной стратегии, тогда как в Excel пользователь должен сам определить ее.

# **3.7. Заключение**

**Решение для предоставленной задачи мы реализовали с помощью двух алгоритмов. Первый алгоритм основан на надстройки Excel – поиск решений. Его преимуществами можно считать – наглядное представление всех массивов данных, простота редактирования и ввода новых данных, автоматическое обновление формул. Из недостатков – отсутствие пользовательского интерфейса, нет наглядного текстового вывода выигрышной стратегии.**

**Второй алгоритм, реализованный в Python при помощи библиотек pandas, NashPy, matplotlib, имеет очевидное преимущество перед первым –прописываются конкретные стратегии для конкретного игрока. Недостатком выступает тот факт, что смешанные стратегии через линпрог не оптимизированы под матрицы не квадратного типа. Эта программа имеет перспективы для расширения функционала, так как можно усовершенствовать вычислительные алгоритмы и сделать ее более универсальной для различных типов датасетов, добавить выгрузку данных в различных форматах и т.п.**

**Все используемые методы подсчета выдают идентичный результат –Правильность кода была проверена на 3 случайно сформированных датасетах для каждого вида игры. Погрешность исключена, но есть условие равенства строк и столбцов для получения смешанных стратегий в Python. Недостатком алгоритма Excel является сложная работа алгоритма ОПГ (не всегда удается вывести смешанные стратегии), в то время как в Python он всегда реализуется верно. Также в Python автоматически строится график и матрицы, пользователю не придется перебивать данные в случае, если он захочет увеличить размерность матриц.**