**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**им. Н.Э. БАУМАНА**

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

**ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ**

**Лабораторная работа № 5 на тему:**

«Критерии выбора оптимальных стратегий в неантагонистических играх и свойства оптимальных решений»

Вариант 4

**Преподаватель:**

Коннова Н.С.

**Студент**:

Куликова А.В.

**Группа:**

ИУ8-21М

**Цель работы**

Изучить критерии выбора стратегий в неантагонистической бескоалиционной игре двух игроков на основе равновесия Нэша и оптимальности по Парето. Проверить данные критерии на примере известных игр. Исследовать свойства оптимальных решений неантагонистических бескоалиционных игр на примере биматричных (2 х 2)-игр.

**Постановка задачи**

1) Сгенерировать случайную биматричную игру (10х10). Найти ситуации, равновесные по Нэшу и оптимальные по Парето, пересечение данных множеств. Произвести проверку реализованных алгоритмов на примере трёх известных игр: **семейный спор, перекрёсток со смещением, дилемма заключённого**.

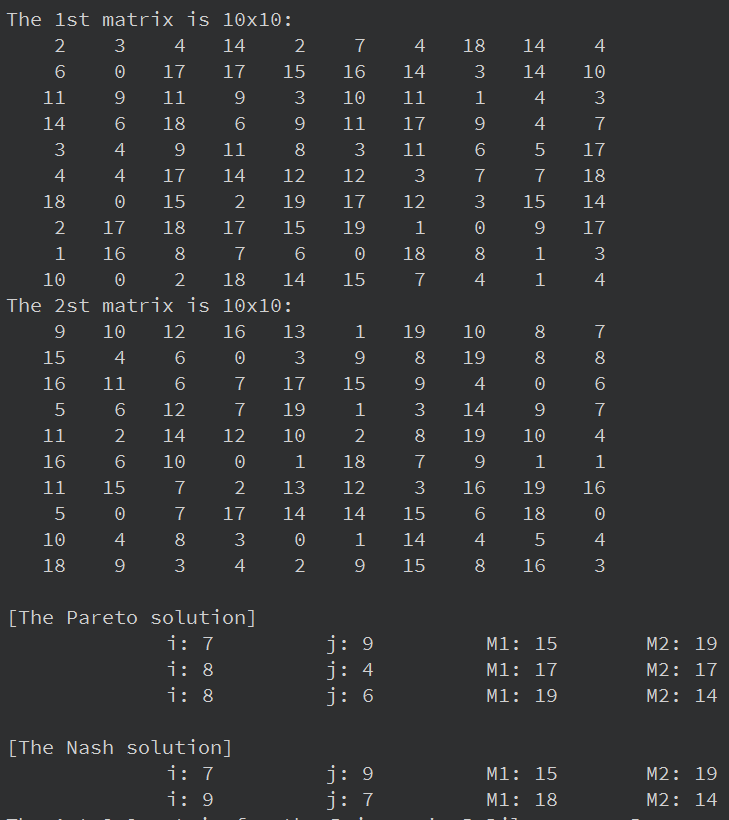
2) Для заданной по варианту Г(𝐴,𝐵) – биматричной (2 х 2)-игры, пользуясь теоремами о свойствах оптимальных решений, найдите ситуации, равновесные по Нэшу, для исходной игры и для ее смешанного расширения..

**Ход работы**

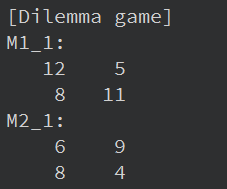
Ниже приведены биматричные игры для второй части задания. Первая часть задания подразумевает генерацию случайной платежной матрицы.

Биматрическая игра:

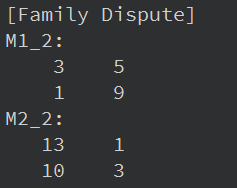
Случайная биматричная игра:



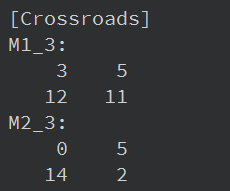
Случайная матрица для игры «Дилемма заключенного»



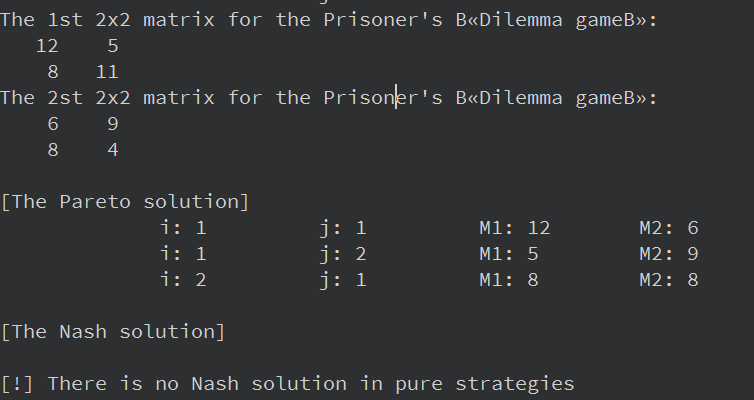
Случайная матрица для игры «Семейный спор»



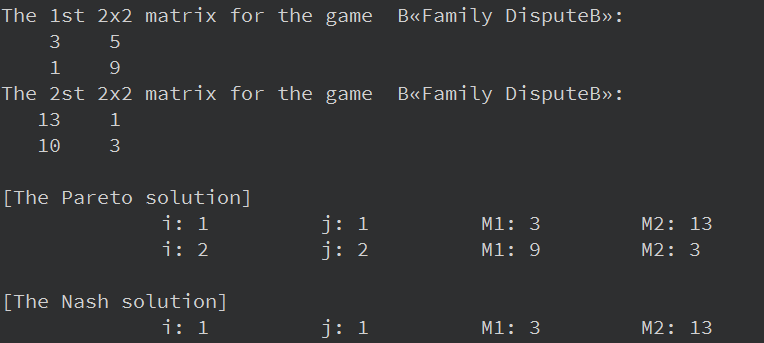
Случайная матрица для игры «Перекресток»



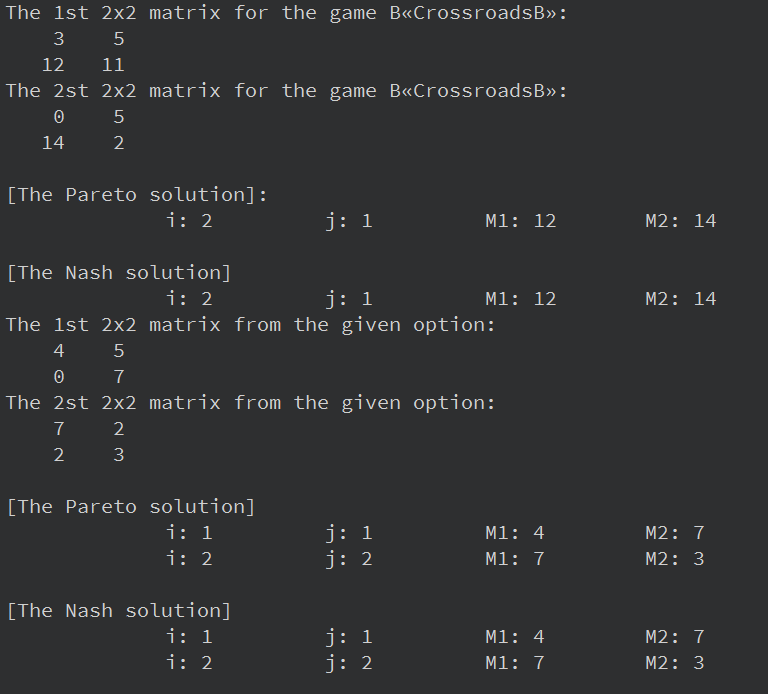
**матрица для игры «Дилемма заключенного»:**



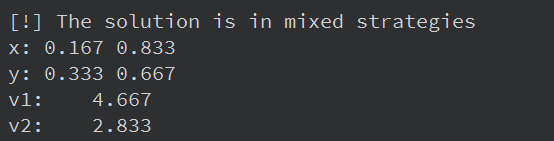
**матрица для игры «Семейный спор»:**



**матрица для игры «Перекресток»:**



Как видим, у этой игры есть две равновесные по Нэшу ситуации в чистых стратегиях, поэтому в смешанном дополнении игры существует ещё одна вполне смешанная ситуация равновесия, которую мы можем рассчитать по формулам из методических указаний к заданию:



Результат программы, представленной в приложении А.

**Выводы**

В ходе проделанной работы было найдено решение по Нэшу и Парето, а также исследован ряд теоретических моментов.

Один игрок имеет строго доминирующую стратегию, существует только одно решение по Нэшу.

**Контрольные вопросы**

1. Определение равновесных ситуаций в игре.

2. Оптимальность по Парето в бескоалиционных неантагонистических играх нескольких игроков?

3. Нахождение равновесия в смешанных стратегиях?

4. Пусть Г(𝐴,𝐵) – биматричная (𝑚 х 𝑚)-игра. Всегда ли существуют смешанные стратегии 𝑥∗, 𝑦∗ игроков 1 и 2 соответственно, такие что (𝑥∗, 𝑦∗) – ситуация равновесия по Нэшу??

.

**Приложение А**

\*\*\*\*GENERATION START\*\*\*\*

[Dilemma game]

M1\_1:

12 5

8 11

M2\_1:

6 9

8 4

[Family Dispute]

M1\_2:

3 5

1 9

M2\_2:

13 1

10 3

[Crossroads]

M1\_3:

3 5

12 11

M2\_3:

0 5

14 2

\*\*\*\*GENERATION END\*\*\*\*

The 1st matrix is 10x10:

2 3 4 14 2 7 4 18 14 4

6 0 17 17 15 16 14 3 14 10

11 9 11 9 3 10 11 1 4 3

14 6 18 6 9 11 17 9 4 7

3 4 9 11 8 3 11 6 5 17

4 4 17 14 12 12 3 7 7 18

18 0 15 2 19 17 12 3 15 14

2 17 18 17 15 19 1 0 9 17

1 16 8 7 6 0 18 8 1 3

10 0 2 18 14 15 7 4 1 4

The 2st matrix is 10x10:

9 10 12 16 13 1 19 10 8 7

15 4 6 0 3 9 8 19 8 8

16 11 6 7 17 15 9 4 0 6

5 6 12 7 19 1 3 14 9 7

11 2 14 12 10 2 8 19 10 4

16 6 10 0 1 18 7 9 1 1

11 15 7 2 13 12 3 16 19 16

5 0 7 17 14 14 15 6 18 0

10 4 8 3 0 1 14 4 5 4

18 9 3 4 2 9 15 8 16 3

[The Pareto solution]

i: 7 j: 9 M1: 15 M2: 19

i: 8 j: 4 M1: 17 M2: 17

i: 8 j: 6 M1: 19 M2: 14

[The Nash solution]

i: 7 j: 9 M1: 15 M2: 19

i: 9 j: 7 M1: 18 M2: 14

The 1st 2x2 matrix for the Prisoner's «Dilemma game»:

12 5

8 11

The 2st 2x2 matrix for the Prisoner's «Dilemma game»:

6 9

8 4

[The Pareto solution]

i: 1 j: 1 M1: 12 M2: 6

i: 1 j: 2 M1: 5 M2: 9

i: 2 j: 1 M1: 8 M2: 8

[The Nash solution]

[!] There is no Nash solution in pure strategies

The 1st 2x2 matrix for the game «Family Dispute»:

3 5

1 9

The 2st 2x2 matrix for the game «Family Dispute»:

13 1

10 3

[The Pareto solution]

i: 1 j: 1 M1: 3 M2: 13

i: 2 j: 2 M1: 9 M2: 3

[The Nash solution]

i: 1 j: 1 M1: 3 M2: 13

The 1st 2x2 matrix for the game «Crossroads»:

3 5

12 11

The 2st 2x2 matrix for the game «Crossroads»:

0 5

14 2

[The Pareto solution]:

i: 2 j: 1 M1: 12 M2: 14

[The Nash solution]

i: 2 j: 1 M1: 12 M2: 14

The 1st 2x2 matrix from the given option:

4 5

0 7

The 2st 2x2 matrix from the given option:

7 2

2 3

[The Pareto solution]

i: 1 j: 1 M1: 4 M2: 7

i: 2 j: 2 M1: 7 M2: 3

[The Nash solution]

i: 1 j: 1 M1: 4 M2: 7

i: 2 j: 2 M1: 7 M2: 3

[!] The solution is in mixed strategies

x: 0.167 0.833

y: 0.333 0.667

v1: 4.667

v2: 2.833

**Приложение Б**

исходный код **CMakeLists.txt**:

cmake\_minimum\_required(VERSION 2.8)

add\_executable(main

        main.cpp main.define.h

        matrix.h matrix.cpp

        vector.h vector.cpp

    )

исходный код **main.cpp**:

#include <iostream>

#include <iomanip>

#include <fstream>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include "main.define.h"

#include "vector.h"

#include "matrix.h"

namespace gameTheoryAndOperationsResearch {

    void \_gameTheoryAndOperationsResearch\_readfile() {

        std::string line;std::ifstream in(gameTheoryAndOperationsResearch\_filename);

        while (std::getline(in, line))

            std::cout << line << std::endl;

    }

    void \_gameTheoryAndOperationsResearch\_deletefile() {

        std::remove(gameTheoryAndOperationsResearch\_filename);

    }

    gameTheoryAndOperationsResearch\_matrix \_gameTheoryAndOperationsResearch\_matrix\_main;

    gameTheoryAndOperationsResearch\_vector \_gameTheoryAndOperationsResearch\_vector\_main;

    // 9 Вариант

    double M1[DEFINE\_N1][DEFINE\_N1] = { { 4, 5 },

                                        { 0, 7 } }; // Матрица для 1-го игрока

    double M2[DEFINE\_N1][DEFINE\_N1] = { { 7, 2 },

                                        { 2, 3} }; // Матрица для 2-го игрока

    // Проверка принадлежит ли решение с индексами i1 и j1 множеству Парето

    // Исходные матрицы передаются через указатели

    template<typename T>

    bool is\_lot\_of\_Pareto(T \*M1, T \*M2, int n, int i1, int j1)

    {

        T Val1 = \*(M1 + i1 \* n + j1);

        T Val2 = \*(M2 + i1 \* n + j1);

        for (int i = 0; i < n; i++)

            for (int j = 0; j < n; j++)

                if (!((i == i1) && (j == j1))) // Просматриваем все элементы кроме заданного

                    if ((\*(M1 + i \* n + j) >= Val1 && \*(M2 + i \* n + j) > Val2) ||

                        (\*(M1 + i \* n + j) > Val1 && \*(M2 + i \* n + j) >= Val2))

                        return false;  // Нашли решение, которое доминирует, решение не принадлежит множеству Парето

        return true; // Не нашли доминирующего решения, значит решение принадлежит множеству Парето

    }

    // Проверка принадлежит ли решение с индексами i1 и j1 множеству Неша

    template<typename T>

    bool is\_lot\_of\_Nesh(T \*M1, T \*M2, int n, int i1, int j1)

    {

        T Val1 = \*(M1 + i1 \* n + j1);

        T Val2 = \*(M2 + i1 \* n + j1);

        // Проверяем может ли первый игрок улучшить свое решение

        for (int i = 0; i < n; i++)

            if (i != i1)

                if (\*(M1 + i \* n + j1) > Val1)

                    return false; // Первый игрок может улучшить свое решение при заданном решении 2-го игрока, решение не принадлешит Нешу

        // Проверяем может ли второй игрок улучшить свое решение

        for (int j = 0; j < n; j++)

            if (j != j1)

                if (\*(M2 + i1 \* n + j) > Val2)

                    return false; // Второй игрок может улучшить свое решение при заданном решении 1-го игрока, решение не принадлехит Нешу

        return true; // Решение принадлежит Нешу

    }

    // Печать матрицы c заголовком

    template<typename T>

    void  print\_matrix(std::ostream & out, char \*text, T \*M, int n)

    {

        out << text << ":" << std::endl; // Печать заголовка

        for (int i = 0; i < n; i++)

        {

            for (int j = 0; j < n; j++) {

                out.width(5);

                out << \*(M + i \* n + j);

            }

            out << std::endl;

        }

    }

    int \_main(int argc, char\* argv[])

    {

        srand(time(0));

        int M22\_1[DEFINE\_N2][DEFINE\_N2], M22\_2[DEFINE\_N2][DEFINE\_N2];   // Матрицы 10 х 10 для заполнения псевдослуч числами

        int N = 15;

        int M1\_1[DEFINE\_N1][DEFINE\_N1] = { { rand() % N, rand() % N },

                                           { rand() % N, rand() % N } }; // The 1st 2x2 matrix for the Prisoner's «Dilemma game»

        int M1\_2[DEFINE\_N1][DEFINE\_N1] = { { rand() % N, rand() % N },

                                           { rand() % N, rand() % N} }; // The 1st 2x2 matrix for the game  «Family Dispute»

        int M1\_3[DEFINE\_N1][DEFINE\_N1] = { { rand() % N, rand() % N },

                                           { rand() % N, rand() % N }}; // The 1st 2x2 matrix for the game "Crossroads"

        int M2\_1[DEFINE\_N1][DEFINE\_N1] = { { rand() % N, rand() % N },

                                           { rand() % N, rand() % N } }; // The 2st 2x2 matrix for the Prisoner's «Dilemma game»

        int M2\_2[DEFINE\_N1][DEFINE\_N1] = { { rand() % N, rand() % N },

                                           { rand() % N, rand() % N } }; // The 2st 2x2 matrix for the game  «Family Dispute»

        int M2\_3[DEFINE\_N1][DEFINE\_N1] = { { rand() % N, rand() % N },

                                           { rand() % N, rand() % N } }; // The 2st 2x2 matrix for the game «Crossroads»

        // Удаляем файл

        \_gameTheoryAndOperationsResearch\_deletefile();

        std::ofstream fout(gameTheoryAndOperationsResearch\_filename);

        fout << "\*\*\*\*GENERATION START\*\*\*\*" << std::endl;

        fout << "[Dilemma game]" << std::endl;

        print\_matrix(fout,  (char \*)"M1\_1", (int \*)M1\_1, (int)DEFINE\_N1);

        print\_matrix(fout,  (char \*)"M2\_1", (int \*)M2\_1, (int)DEFINE\_N1);

        fout << "[Family Dispute]" << std::endl;

        print\_matrix(fout,  (char \*)"M1\_2", (int \*)M1\_2, (int)DEFINE\_N1);

        print\_matrix(fout,  (char \*)"M2\_2", (int \*)M2\_2, (int)DEFINE\_N1);

        fout << "[Crossroads]" << std::endl;

        print\_matrix(fout,  (char \*)"M1\_3", (int \*)M1\_3, (int)DEFINE\_N1);

        print\_matrix(fout,  (char \*)"M2\_3", (int \*)M2\_3, (int)DEFINE\_N1);

        fout << "\*\*\*\*GENERATION END\*\*\*\*\n" << std::endl;

        // заполняем матрцы ПСЧ

        for(int i=0; i<DEFINE\_N2; i++) {

            for (int j = 0; j < DEFINE\_N2; j++)

            {

                M22\_1[i][j] = rand() % 20;

                M22\_2[i][j] = rand() % 20;

            }

        }

        print\_matrix(fout,  (char \*)"The 1st matrix is 10x10", (int \*)M22\_1, (int)DEFINE\_N2);

        print\_matrix(fout, (char \*)"The 2st matrix is 10x10", (int \*)M22\_2, DEFINE\_N2);

        // Поиск множества Парето

        fout << "\n[The Pareto solution]" << std::endl;

        for (int i = 0; i < DEFINE\_N2; i++)

            for (int j = 0; j < DEFINE\_N2; j++)

                if (is\_lot\_of\_Pareto((int \*)M22\_1, (int \*)M22\_2, DEFINE\_N2, i, j)) {// Нашли решение из множества Парето

                    fout << "\ti: " << (i + 1) << "\tj: " << (j + 1) << "\tM1: " << M22\_1[i][j] << "\tM2: " << M22\_2[i][j] << std::endl;

                }

        fout << "\n[The Nash solution]" << std::endl;

         bool flag = false;

        for (int i = 0; i < DEFINE\_N2; i++)

            for (int j = 0; j < DEFINE\_N2; j++)

                if (is\_lot\_of\_Nesh((int \*)M22\_1, (int \*)M22\_2, DEFINE\_N2, i, j)) // Нашли решение из множества Неша

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M22\_1[i][j]

                         << "\tM2: " << M22\_2[i][j]

                         << std::endl;

                    flag = true;

                }

        if (!flag) {

                fout << "\n[!] There is no Nash solution in pure strategies" << std::endl;

        }

        print\_matrix(fout, (char \*)"The 1st 2x2 matrix for the Prisoner's «Dilemma game»", (int \*)M1\_1, DEFINE\_N1);

        print\_matrix(fout, (char \*)"The 2st 2x2 matrix for the Prisoner's «Dilemma game»", (int \*)M2\_1, DEFINE\_N1);

        // Поиск множества Парето

        fout << "\n[The Pareto solution]" << std::endl;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Pareto((int \*)M1\_1, (int \*)M2\_1, DEFINE\_N1, i, j)) // Нашли решение из множества Парето

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1\_1[i][j]

                         << "\tM2: " << M2\_1[i][j]

                            << std::endl;

                }

        fout << "\n[The Nash solution]" << std::endl;

        flag = false;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Nesh((int \*)M1\_1, (int \*)M2\_1, DEFINE\_N1, i, j)) // Нашли решение из множества Неша

                {

                    fout << "\ti: " << (i + 1) << "\tj: " << (j + 1) << "\tM1: " << M1\_1[i][j] << "\tM2: " << M2\_1[i][j] << std::endl;

                    flag = true;

                }

        if (!flag) {

            fout << "\n[!] There is no Nash solution in pure strategies" << std::endl;

        }

        print\_matrix(fout, (char \*)"The 1st 2x2 matrix for the game  «Family Dispute»", (int \*)M1\_2, DEFINE\_N1);

        print\_matrix(fout, (char \*)"The 2st 2x2 matrix for the game  «Family Dispute»", (int \*)M2\_2, DEFINE\_N1);

        // Поиск множества Парето

        fout << "\n[The Pareto solution]" << std::endl;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Pareto((int \*)M1\_2, (int \*)M2\_2, DEFINE\_N1, i, j)) // Нашли решение из множества Парето

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1\_2[i][j]

                         << "\tM2: " << M2\_2[i][j]

                         << std::endl;

                }

        fout << "\n[The Nash solution]" << std::endl;

        flag = false;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Nesh((int \*)M1\_2, (int \*)M2\_2, DEFINE\_N1, i, j)) // Нашли решение из множества Неша

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1\_2[i][j]

                         << "\tM2: " << M2\_2[i][j]

                         << std::endl;

                    flag = true;

                }

        if (!flag) {

                fout << "\n[!] There is no Nash solution in pure strategies" << std::endl;

        }

        print\_matrix(fout, (char \*)"The 1st 2x2 matrix for the game «Crossroads»", (int \*)M1\_3, DEFINE\_N1);

        print\_matrix(fout, (char \*)"The 2st 2x2 matrix for the game «Crossroads»", (int \*)M2\_3, DEFINE\_N1);

        // Поиск множества Парето

        fout << "\n[The Pareto solution]:" << std::endl;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Pareto((int \*)M1\_3, (int \*)M2\_3, DEFINE\_N1, i, j)) // Нашли решение из множества Парето

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1\_3[i][j]

                         << "\tM2: " << M2\_3[i][j]

                         << std::endl;

                }

        fout << "\n[The Nash solution]" << std::endl;

        flag = false;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Nesh((int \*)M1\_3, (int \*)M2\_3, DEFINE\_N1, i, j)) // Нашли решение из множества Неша

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1\_3[i][j]

                         << "\tM2: " << M2\_3[i][j]

                         << std::endl;

                    flag = true;

                }

        if (!flag) {

                fout << "\n[!] There is no Nash solution in pure strategies" << std::endl;

        }

        print\_matrix(fout, (char \*)"The 1st 2x2 matrix from the given option", (double \*)M1, DEFINE\_N1);

        print\_matrix(fout, (char \*)"The 2st 2x2 matrix from the given option", (double \*)M2, DEFINE\_N1);

        // Поиск множества Парето

        fout << "\n[The Pareto solution]" << std::endl;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Pareto((double \*)M1, (double \*)M2, DEFINE\_N1, i, j)) // Нашли решение из множества Парето

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1[i][j]

                         << "\tM2: " << M2[i][j]

                         << std::endl;

                }

        fout << "\n[The Nash solution]" << std::endl;

        flag = false;

        int Num=0;

        for (int i = 0; i < DEFINE\_N1; i++)

            for (int j = 0; j < DEFINE\_N1; j++)

                if (is\_lot\_of\_Nesh((double \*)M1, (double \*)M2, DEFINE\_N1, i, j)) // Нашли решение из множества Неша

                {

                    fout << "\ti: " << (i + 1)

                         << "\tj: " << (j + 1)

                         << "\tM1: " << M1[i][j]

                         << "\tM2: " << M2[i][j]

                         << std::endl;

                    flag = true;

                    Num++;

                }

        if (!flag) {

                fout << "\n[!] There is no Nash solution in pure strategies" << std::endl;

        }

        fout<<"\n[!] The solution is in mixed strategies"<<std::endl;

        if (Num!=1)

        {

            double \*p = new double[DEFINE\_N1]; double \*q = new double[DEFINE\_N1];

            double \*\*c = new double \*[DEFINE\_N1]; // Обратная матрица

            for (int i = 0; i < DEFINE\_N1; i++)

                c[i] = new double[DEFINE\_N1];

            \_gameTheoryAndOperationsResearch\_matrix\_main.get\_inverse\_matrix(M2, c, DEFINE\_N1); // Получаем обратную матрицу

            double \* u = new double[DEFINE\_N1]; // Вспомогательный вектор из всех 1

            for (int i = 0; i < DEFINE\_N1; i++)

                u[i] = 1;

            double \* r1 = new double[DEFINE\_N1]; // Вектор для промежуточных вычислений

            \_gameTheoryAndOperationsResearch\_vector\_main.composition\_vector\_matrix(u, c, r1, DEFINE\_N1, DEFINE\_N1);

            double v2 = \_gameTheoryAndOperationsResearch\_vector\_main.composition\_vector\_vector(r1, u, DEFINE\_N1);

            \_gameTheoryAndOperationsResearch\_vector\_main.composition\_vector\_matrix(u, c, p, DEFINE\_N1, DEFINE\_N1);

            for (int i = 0; i < DEFINE\_N1; i++)

                p[i] /= v2;

            \_gameTheoryAndOperationsResearch\_vector\_main.print\_vector(fout, (char \*)"x", p, DEFINE\_N1);

            \_gameTheoryAndOperationsResearch\_matrix\_main.get\_inverse\_matrix(M1, c, DEFINE\_N1); // Получаем обратную матрицу

            \_gameTheoryAndOperationsResearch\_vector\_main.composition\_vector\_matrix(u, c, r1, DEFINE\_N1, DEFINE\_N1);

            double v1 = \_gameTheoryAndOperationsResearch\_vector\_main.composition\_vector\_vector(r1, u, DEFINE\_N1);

            \_gameTheoryAndOperationsResearch\_vector\_main.composition\_matrix\_vector(c, u, q, DEFINE\_N1, DEFINE\_N1);

            for (int i = 0; i < DEFINE\_N1; i++)

                q[i] /= v1;

            \_gameTheoryAndOperationsResearch\_vector\_main.print\_vector(fout, (char \*)"y", q, DEFINE\_N1);

            fout << std::fixed << std::setprecision(3);

            fout << "v1:    " << 1/v1 << std::endl;

            fout << "v2:    " << 1/v2 << std::endl;

        }

        else

        {

            fout<<"\n[!] One player has a strictly dominant strategy, there is only one Nash solution"<<std::endl;

        }

        // Считываем файл

        \_gameTheoryAndOperationsResearch\_readfile();

        return 1;

        return 0;

    }

}

int main(int argc, char\* argv[])

{

    gameTheoryAndOperationsResearch::\_main(argc, argv);

    return 0;

}

исходный код **main.define.h**:

#ifndef MAIN\_DEFINE\_H

#define MAIN\_DEFINE\_H

#define DEFINE\_N1  2

#define DEFINE\_N2 10

#define gameTheoryAndOperationsResearch\_filename "gameTheoryAndOperationsResearch\_l4.txt"

#endif // MAIN\_DEFINE\_H

исходный код **matrix.cpp**:

#include "matrix.h"

namespace gameTheoryAndOperationsResearch {

    // Далее функции для аналитического метода обратной матрицы

    double gameTheoryAndOperationsResearch\_matrix::get\_determinant\_matrix(double \*\*p, int n) // Вычисление определителя матрицы

    {

        if (n == 1) {

            return p[0][0];

        }

        if (n == 2) {

            return p[0][0] \* p[1][1] - p[0][1] \* p[1][0];

        }

        else

        {

            double \*\*p2 = new double \*[n - 1];

            for (int i = 0; i < n - 1; i++)

                p2[i] = new double[n - 1];

            int zn = 1;

            double Det = 0;

            for (int k = 0; k < n; k++, zn \*= -1)

            {

                if (p[0][k] == 0)

                    continue;

                for (int i = 1; i < n; i++) {

                    for (int j = 0; j < n; j++)

                    {

                        if (j < k) p2[i - 1][j] = p[i][j];

                        if (j > k) p2[i - 1][j - 1] = p[i][j];

                    }

                }

                Det += zn \* p[0][k] \* get\_determinant\_matrix(p2, n - 1);

            }

            for (int i = 0; i < n - 1; i++)

                delete[] p2[i];

            delete[] p2;

            return Det;

        }

    }

    void gameTheoryAndOperationsResearch\_matrix::get\_inverse\_matrix(double p[DEFINE\_N1][DEFINE\_N1], double \*\*pObr, int n) // Вычисление обратной матрицы для матрицы

    {

        double Det = 0;

        if (n == 2)

        {

            Det = p[0][0] \* p[1][1] - p[0][1] \* p[1][0];

            pObr[0][0] = p[1][1];

            pObr[0][1] = p[1][0] \* -1;

            pObr[1][0] = p[0][1] \* -1;

            pObr[1][1] = p[0][0];

        }

        else

        {

            double \*\*p2 = new double \*[n - 1];

            for (int i = 0; i < n - 1; i++)

                p2[i] = new double[n - 1];

            int zn = 1;

            Det = 0;

            for (int k0 = 0; k0 < n; k0++)

            {

                for (int k = 0; k < n; k++)

                {

                    if ((k + k0) % 2)

                        zn = -1;

                    else

                        zn = 1;

                    //  if (p[0][k] == 0) continue;

                    for (int i = 0; i < n; i++) {

                        for (int j = 0; j < n; j++)

                        {

                            if (i > k0)

                            {

                                if (j < k) p2[i - 1][j] = p[i][j];

                                if (j > k) p2[i - 1][j - 1] = p[i][j];

                            }

                            if (i < k0)

                            {

                                if (j < k) p2[i][j] = p[i][j];

                                if (j > k) p2[i][j - 1] = p[i][j];

                            }

                        }

                    }

                    double opr = get\_determinant\_matrix(p2, n - 1);

                    pObr[k0][k] = zn \* opr;

                    if (k0 == 0)

                        Det += p[k0][k] \* pObr[k0][k];

                }

            }

            for (int i = 0; i < n - 1; i++)

                delete[] p2[i];

            delete[] p2;

        }

        // Транспонируем матрицу

        for (int i = 0; i < n - 1; i++) for (int j = i + 1; j < n; j++)

        {

            double buf = pObr[i][j]; pObr[i][j] = pObr[j][i];

            pObr[j][i] = buf;

        }

        for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {

            pObr[i][j] /= Det;

            if (fabs(pObr[i][j]) < 1e-9) pObr[i][j] = 0;

        }

    }

}

исходный код **matrix.h**:

#ifndef MATRIX\_H

#define MATRIX\_H

#include "main.define.h"

#include <iostream>

#include <iomanip>

#include <fstream>

#include <math.h>

#include <stdlib.h>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_matrix

    {

        public:

            double get\_determinant\_matrix(double\*\*, int);

            void get\_inverse\_matrix(double p[DEFINE\_N1][DEFINE\_N1], double\*\*, int);

        };

}

#endif // MATRIX\_H

исходный код **vector.cpp**:

#include "vector.h"

namespace gameTheoryAndOperationsResearch {

    // Произведение вектора на матрицу

    void gameTheoryAndOperationsResearch\_vector::composition\_vector\_matrix(double \*v, double \*\*matr, double\* v\_out, int n, int m)

    {

        for (int i = 0; i < m; i++)

        {

            v\_out[i] = 0;

            for (int j = 0; j < n; j++)

                v\_out[i] += v[j] \* matr[j][i];

        }

    }

    // Произведение матрицы на вектор

    void gameTheoryAndOperationsResearch\_vector::composition\_matrix\_vector(double \*\*matr, double \*v,  double\* v\_out, int n, int m)

    {

        for (int i = 0; i < n; i++)

        {

            v\_out[i] = 0;

            for (int j = 0; j < m; j++)

                v\_out[i] += v[j] \* matr[i][j];

        }

    }

    // Произведение вектор на вектор - на выходе число (скалярное произведение)

    double gameTheoryAndOperationsResearch\_vector::composition\_vector\_vector(double \*v1, double \*v2, int n)

    {

        double rez = 0;

        for (int i = 0; i < n; i++)

            rez += v1[i] \* v2[i];

        return rez;

    }

    void gameTheoryAndOperationsResearch\_vector::print\_vector(std::ostream& out, char \* str, double \*p, int n)

    {

        out.precision(3);

        out << str << ": ";

        for (int i = 0; i < n; i++)

            out << p[i] << " ";

        out << std::endl;

    }

}

исходный код **vector.h**:

#ifndef VECTOR\_H

#define VECTOR\_H

#include <iostream>

#include <iomanip>

#include <fstream>

#include <math.h>

#include <stdlib.h>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_vector

    {

    public:

        void composition\_vector\_matrix(double \*, double \*\*, double\*, int, int);

        void composition\_matrix\_vector(double \*\*, double \*,  double\*, int, int);

        double composition\_vector\_vector(double \*, double \*, int);

        void print\_vector(std::ostream& , char \*, double \*, int );

    };

}

#endif // VECTOR\_H