**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное**

**учреждение Высшего профессионального образования**

**Московский авиационный институт**

**(национальный исследовательский университет)**

Кафедра 304

Вычислительные машины, системы и сети

**Отчет по лабораторной работе №2**

**по учебной дисциплине «Математическое программирование»**

**на тему**

***«Решение задач линейного программирования»***

Выполнил студент группы М3О-310Б-21:

Пысларь А.И.

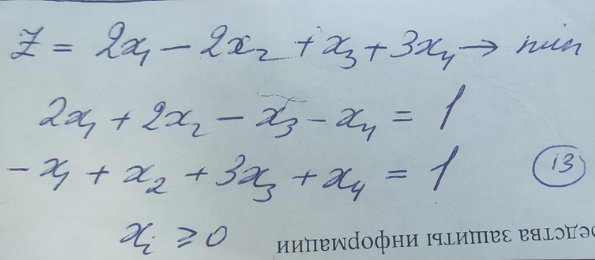
Приняли:

Доцент каф. 311, к.ф.м.н, [Григоревский Н.В.](https://mai.ru/education/schedule/ppc.php?guid=578c1789-1d99-11e0-9baf-1c6f65450efa)

Москва 2023

Задание:

Реализовать программу, решающую заданную задачу симплекс-методом и двойственную этой задаче



**Текст программы на языке C++**

#include <iostream>

#include <iomanip>

using namespace std;

int symplex\_func(double\*\* symplex\_matrix, int\* Basis, bool action);//Функция расчета симплекс-метода

double\* Dvoystvennost(double\*\* Matrix, int\* Basis); //Функция нахождения и решения двойственной задачи

int Obr\_Matrix\_func(double\*\* Matrix); //Функция нахождения обратной матрицы

int main(void) {

setlocale(LC\_ALL, "Russian");

system("color F0");

//Матрицы содержащие коэф. уравнений системы

int Amatrix[2][4] = { -1, 2, 1, -1, 3, 1, -1 , -1 };

double Matrix[3][5] = { 2, -1, 2, 1, -1, 1, 3,1,-1,-1,0,6,9,3,0 };

int Bmatrix[5] = { 0,6,9,3,0 };

//Массивы, содержащий базисные переменные

int Basis[2];

int CpBasis[2];

//Массивы, содержащий координаты

double X[5];

double X1[5];

double\* Cpointer;//Указатель

//Матрицы для симплекс-метода

double\*\* symplex\_matrix;

double\*\* cpmatrix;

//Матрицы для двойственной задачи

double\*\* pointer;

//Вспомогательные переменные

int i;

int j;

//Начальный базис

Basis[0] = 4; Basis[1] = 3;

symplex\_matrix = new double\* [3];

for (i = 0; i < 3; i++)

symplex\_matrix[i] = new double[5];

//Заполнение матриц для симплекс-метода

for (i = 0; i < 2; i++)

{

for (j = 0; j < 4; j++)

{

symplex\_matrix[i][j + 1] = Amatrix[i][j];

X[j] = 0;

X1[j] = 0;

}

CpBasis[i] = Basis[i];

}

symplex\_matrix[0][0] = 4;

symplex\_matrix[1][0] = 8;

cpmatrix = new double\* [3];

for (i = 0; i < 3; i++)

cpmatrix[i] = new double[5];

for (j = 0; j < 5; j++)

symplex\_matrix[2][j] = Bmatrix[j] - (symplex\_matrix[0][j] \* Bmatrix[Basis[0]] + symplex\_matrix[1][j] \* Bmatrix[Basis[1]]);//Расчёт симплекс-разности

for (i = 0; i < 3; i++)

for (j = 0; j < 5; j++)

cpmatrix[i][j] = symplex\_matrix[i][j];

j = 0;

while (symplex\_matrix[2][j] >= 0) //Проверка результатов разности

j++;

if (j == 5) { //Если не осталось отрицательных элементов в строке, т.е. нашли минимум

for (i = 0; i < 2; i++)

for (j = 0; j < 4; j++)

if (j == Basis[i] - 1)

X[j] = symplex\_matrix[i][0]; //Сохраняем координаты

X[4] = X[0] + 2 \* X[1] - X[2] + X[3]; //Рассчитываем значение функции

symplex\_func(cpmatrix, CpBasis, true); //Поиск максимума симплекс-методом

for (i = 0; i < 2; i++)

for (j = 0; j < 4; j++)

if (j == CpBasis[i] - 1)

X1[j] = cpmatrix[i][0]; //Сохраняем координаты

X1[4] = X1[0] + 2 \* X1[1] - X1[2] + X1[3]; //Рассчитываем значение функции

}

else

{

j = 0;

while (symplex\_matrix[2][j] <= 0) //Проверка результатов разности

j++;

if (j == 5) //Если не осталось положительных элементов в строке,т.е. нашли максимум

{

for (i = 0; i < 2; i++)

for (j = 0; j < 4; j++)

if (j == Basis[i] - 1)

X1[j] = symplex\_matrix[i][0]; //Сохраняем координаты

X1[4] = X1[0] + 2 \* X1[1] - X1[2] + X1[3]; //Рассчитываем значение функции

symplex\_func(cpmatrix, CpBasis, false); //Поиск минимума симплекс-методом

for (i = 0; i < 2; i++)

for (j = 0; j < 4; j++)

if (j == CpBasis[i])

X[j] = cpmatrix[i][0]; //Сохраняем координаты

X[4] = X[0] + 2 \* X[1] - X[2] + X[3]; //Рассчитываем значение функции

}

else //Если после 1го подсчета не было найдено ни минимума, ни максимума

{

symplex\_func(symplex\_matrix, Basis, false); //Поиск минимума симплекс-методом

for (i = 0; i < 2; i++)

for (j = 0; j < 4; j++)

if (j == Basis[i] - 1)

X[j] = symplex\_matrix[i][0];//Сохраняем координаты

X[4] = X[0] + 2 \* X[1] - X[2] + X[3]; //Рассчитываем значение функции

symplex\_func(cpmatrix, CpBasis, true); //Поиск максимума симплекс-методом

for (i = 0; i < 2; i++)

for (j = 0; j < 4; j++)

if (j == CpBasis[i] - 1)

X1[j] = cpmatrix[i][0]; //Сохраняем координаты

X1[4] = X1[0] + 2 \* X1[1] - X1[2] + X1[3]; //Рассчитываем значение функции

}

}

cout << "Минимум функции равен:" << X[4] << " в точке X(";

for (i = 0; i < 4; i++)

cout << X[i] << " ";

cout << ")\n";

cout << "Максимум функции равен:" << X1[4] << " в точке X(";

for (i = 0; i < 4; i++)

cout << X1[i] << " ";

cout << ")\n";

pointer = new double\* [3];

for (i = 0; i < 3; i++)

pointer[i] = new double[5];

for (i = 0; i < 3; i++)

for (j = 0; j < 5; j++)

pointer[i][j] = Matrix[i][j];

//Освобождение памяти

for (i = 0; i < 3; i++)

{

delete[] symplex\_matrix[i];

delete[] cpmatrix[i];

}

delete[] symplex\_matrix;

delete[] cpmatrix;

Cpointer = Dvoystvennost(pointer, CpBasis); //Решение двойственной задачи, поиск ее минимума

if (Cpointer == (double\*)1)

{

cout << "Ошибка решения двойственной задачи";

system("pause");

return 1;

}

X[4] = 4 \* Cpointer[0] + 8 \* Cpointer[1]; //Расчёт значения функции

X[0] = Cpointer[0]; X[1] = Cpointer[1]; //Сохранение координат

cout << "Минимум двойственной функции равен:" << X[4] << " в точке Y(";

for (i = 0; i < 2; i++)

cout << X[i] << " ";

cout << ")\n";

Cpointer = Dvoystvennost(pointer, Basis); //Решение двойственной задачи, поиск ее максимума

if (Cpointer == (double\*)1)

{

cout << "Ошибка решения двойственной задачи\n";

system("pause");

return 2;

}

X1[4] = 4 \* Cpointer[0] + 8 \* Cpointer[1]; //Расчёт значения функции

X1[0] = Cpointer[0]; X1[1] = Cpointer[1]; //Сохранение координат

cout << "Максимум двойственной функции равен:" << X1[4] << " в точке Y(";

for (i = 0; i < 2; i++)

cout << X1[i] << " ";

cout << ")\n";

system("pause");

//Освобождение памяти

for (i = 0; i < 3; i++)

delete[] pointer[i];

delete[] pointer;

return 0;

}

//Симплекс-метод

int symplex\_func(double\*\* symplex\_matrix, int\* Basis, bool action)

{

//Вспомогательные переменные

int j;

int k;

int i;

double min;

int temp = 0;

int temp1 = 0;

//Вспомогательная матрица

double\*\* TempMatr;

TempMatr = new double\* [3];

for (i = 0; i < 3; i++)

TempMatr[i] = new double[5];

for (i = 0; i < 3; i++)

for (j = 0; j < 5; j++)

TempMatr[i][j] = symplex\_matrix[i][j];

bool flag = false;//Флаг

if (action == true) //Если ищем максимум

{

min = 10000;

for (j = 0; j < 5; j++) //Ищем минимальное положительное значение

{

if ((symplex\_matrix[2][j] < min) && (symplex\_matrix[2][j] > 0))

{

min = symplex\_matrix[2][j]; temp = j; //Сохраняем номер столбца

}

}

}

else //Если ищем минимум

{

min = -10000;

for (j = 0; j < 5; j++) //Ищем максимальное отрицательное значение

{

if ((symplex\_matrix[2][j] > min) && (symplex\_matrix[2][j] < 0))

{

min = symplex\_matrix[2][j]; temp = j; //Сохраняем номер столбца

}

}

}

if ((symplex\_matrix[0][0] / symplex\_matrix[0][temp] > symplex\_matrix[1][0] / symplex\_matrix[1][temp]) && (symplex\_matrix[1][0] / symplex\_matrix[1][temp] > 0))//Определяем строку,которую будем заменять

{

temp1 = 1; //Сохраняем номер строки

for (j = 0; j < 5; j++)

{

TempMatr[1][j] = symplex\_matrix[1][j] / symplex\_matrix[temp1][temp];//Пересчет строки

}

}

else

{

if (symplex\_matrix[0][0] / symplex\_matrix[0][temp] < 0)

{

temp1 = 1;

for (j = 0; j < 5; j++)

{

TempMatr[1][j] = symplex\_matrix[1][j] / symplex\_matrix[temp1][temp];//Пересчет строки

}

}

else

{

temp1 = 0; //Сохраняем номер строки

for (j = 0; j < 5; j++)

{

TempMatr[0][j] = symplex\_matrix[0][j] / symplex\_matrix[temp1][temp];//Пересчет строки

}

}

}

for (i = 0; i < 3; i++)

{

for (j = 0; j < 5; j++) {

if (i == temp1) { //Пропускаем опорную строку

flag = true;

break;

}

TempMatr[i][j] = symplex\_matrix[i][j] - ((symplex\_matrix[temp1][j] \* symplex\_matrix[i][temp]) / symplex\_matrix[temp1][temp]); //Пересчет строк

}

if (flag == true)

{

flag = false;

continue;

}

}

for (i = 0; i < 3; i++)

{

for (j = 0; j < 5; j++)

{

symplex\_matrix[i][j] = TempMatr[i][j];

}

}

Basis[temp1] = temp; //Изменяем базис

j = 0;

k = 0;

for (i = 0; i < 5; i++) //Проверяем обнаружили ли минимум или максимум

{

if (symplex\_matrix[2][i] <= 0)

j++;

if (symplex\_matrix[2][i] >= 0)

k++;

}

for (i = 0; i < 3; i++)

{

delete[] TempMatr[i];

}

delete[] TempMatr;

if (action == true) //Если искали максимум

if (j == 5) //Нашли его

goto Exit; //Выход

else

{

symplex\_func(symplex\_matrix, Basis, action);//Иначе следующая итерация

goto Exit;

}

else // Если искали минимум

if (k == 5) //Нашли его

goto Exit; //Выход

else

symplex\_func(symplex\_matrix, Basis, action);//Иначе следующая итерация

Exit:

return 0;

}

//Двойственность

double\* Dvoystvennost(double\*\* Matrix, int\* Basis)

{

double\*\* Matrix2;

Matrix2 = new double\* [2];

for (int i = 0; i < 2; i++)

Matrix2[i] = new double[2];

double C[2];

double C2[2];

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

Matrix2[j][i] = Matrix[j][Basis[i]];

C[i] = Matrix[2][Basis[i]];

C2[i] = 0;

}

if (Obr\_Matrix\_func(Matrix2) == 1)

return (double\*)1;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

C2[i] = C[j] \* Matrix2[j][i] + C2[i];

for (int i = 0; i < 2; i++)

{

delete[] Matrix2[i];

}

delete[] Matrix2;

return C2;

}

//Обратная матрица

int Obr\_Matrix\_func(double\*\* Matrix)

{

double Det;

int i;

int j;

double\*\* Matrix2;

Matrix2 = new double\* [2];

for (int i = 0; i < 2; i++)

Matrix2[i] = new double[2];

double temp;

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

if (Det == 0)

{

cout << "Определитель = 0. Обратной матрицы не существует\n";

return 1;

}

for (i = 0; i < 2; i++)

for (j = 0; j < 2; j++)

Matrix2[i][j] = Matrix[j][i];

for (i = 0; i < 2; i++)

for (j = 0; j < 2; j++)

Matrix[i][j] = pow((-1), i + j + 2) \* Matrix2[1 - i][1 - j] / Det;

for (int i = 0; i < 2; i++)

{

delete[] Matrix2[i];

}

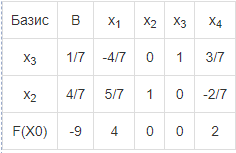
delete[] Matrix2;

return 0;

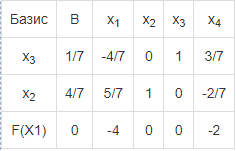
}

**Результаты работы программы**

Симплекс метод:

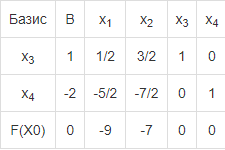


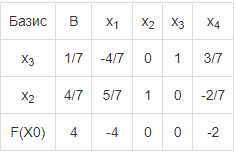


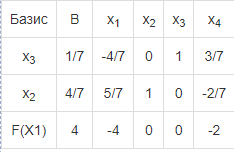




Двойственная задача:









**Вывод**

В данной ЛР была реализована программа, решающая определенную задачу линейного программирования симплекс-методом и двойственную задачу.