МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

(ФГБОУ ВО «ВятГУ»)

Факультет автоматики и вычислительной техники

Кафедра ЭВМ

Отчёт

Лабораторная работа № 1 по дисциплине

«Исследование операций»

Выполнил студент группы ИВТб-2301-04-00 / Жеребцов К. А.

Проверил преподаватель / Коржавина А. С.

Киров 2022

1. Цель работы: закрепить на практике знания о симплексном методе решения задач линейного программирования и получить навыки его программной реализации.
2. Задание: написать программу, реализующую симплексный метод либо одну из его модификаций – метод искусственного базиса, двойственный симплекс и т. д.

Необходимо рассмотреть все возможные случаи, например, когда задача не имеет решений, имеет множество решений и т.д.

1. Описание симплекс-метода:

Симплекс-метод позволяет эффективно найти оптимальное решение, избегая простой перебор всех возможных угловых точек. Основной принцип метода: вычисления начинаются с какого-то «стартового» базисного решения, а затем ведется поиск решений, «улучшающих» значение целевой функции. Это возможно только в том случае, если возрастание какой-то переменной приведет к увеличению значения функционала.

Необходимые условия для применения симплекс-метода:

* Задача должна иметь каноническую форму.
* У задачи должен быть явно выделенный базис.

Алгоритм симплекс-метода:

1. Выбираем переменную, которую будем вводить в базис. Это делается в соответствии с указанным ранее принципом: мы должны выбрать переменную, возрастание которой приведет к росту функционала. Выбор происходит по следующему правилу:

Если задача на минимум – выбираем максимальный положительный элемент в последней строке.

Если задача на максимум – выбираем минимальный отрицательный.

Такой выбор, действительно, соответствует упомянутому выше принципу: если задача на минимум, то чем большее число вычитаем – тем быстрее убывает функционал; для максимума наоборот – чем большее число добавляем, тем быстрее функционал растет.

2. Выбираем переменную, которую будем вводить в базис. Для этого нужно определить, какая из базисных переменных быстрее всего обратится в нуль при росте новой базисной переменной. Алгебраически это делается так:

Вектор правых частей почленно делится на ведущий столбец

Среди полученных значений выбирают минимальное положительное (отрицательные и нулевые ответы не рассматривают)

3. Ищем элемент, стоящий на пересечении ведущих строки и столбца.

4. Вместо исключаемой переменной в первом столбце (с названиями базисных переменных) записываем название переменной, которую мы вводим в базис.

5. Далее начинается процесс вычисления нового базисного решения. Он происходит с помощью метода Жордана-Гаусса.

Новая Ведущая строка = Старая ведущая строка / Ведущий элемент

Новая строка = Новая строка – Коэффициент строки в ведущем столбце \* Новая Ведущая строка

6. После этого проверяем условие оптимальности. Если полученное решение неоптимально – повторяем весь процесс снова.

1. Листинг программы:

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

struct Min\_El

{

double min;

int index;

};

// размер

int number = 0;

int str\_num = 0;

// основная матрица

double\*\*matrix;

// дополнение к основной матрице

double\*\*one;

// таблица

double\*\* Table;

double\*\* Table2;

// целевая функция

double \*F;

// Свободные члены

double\*B;

int\* Base;

double\* mas;

bool flag = true;

//Инициализация

void Start()

{

int i, j;

F = new double[number + str\_num];

// выделение памяти для матрицы

matrix = new double\* [str\_num];

if (matrix != NULL)

{

for (i = 0; i < str\_num; i++)

{

matrix[i] = new double[number];

}

}

// выделение памяти для единичной

one = new double\* [str\_num];

if (one != NULL)

{

for (i = 0; i < str\_num; i++)

{

one[i] = new double[str\_num];

}

}

B = new double[str\_num + 1];

Base = new int[str\_num];

// выделение памяти для таблицы

Table = new double\* [str\_num + 1];

if (Table != NULL)

{

for (i = 0; i < str\_num + 1; i++)

{

Table[i] = new double[str\_num + number + 1];

}

}

Table2 = new double\* [str\_num + 1];

if (Table2 != NULL)

{

for (i = 0; i < str\_num + 1; i++)

{

Table2[i] = new double[str\_num + number + 1];

}

}

mas = new double[str\_num ];

for (i = 0; i < str\_num; i++)

{

mas[i] = 0;

}

}

//Завершение

void Stop()

{

int i, j;

delete[] F;

for (i = 0; i < str\_num; i++)

{

delete[] matrix[i];

}

delete[] matrix;

for (i = 0; i < str\_num; i++)

{

delete[] one[i];

}

delete[] one;

delete[] B;

delete[] Base;

for (i = 0; i < str\_num + 1; i++)

{

delete[] Table[i];

}

delete[] Table;

for (i = 0; i < str\_num + 1; i++)

{

delete[] Table2[i];

}

delete[] Table2;

delete[] mas;

}

//Чтение из файла

void Read(ifstream &inn)

{

int i, j;

for (i = 0; i < number + str\_num; i++)

{

if (i < number)

{

inn >> F[i];

F[i] = -F[i];

}

else

{

F[i] = 0;

}

}

// заполнение матрицы

for (i = 0; i < str\_num; i++)

{

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

{

inn >> matrix[i][j];

}

}

// заполнение единичной

for (i = 0; i < str\_num; i++)

{

for (j = 0; j < str\_num; j++)

{

if (i != j)

{

one[i][j] = 0;

}

else

{

one[i][j] = 1;

}

}

}

// Свободные члены

for (i = 0; i < str\_num + 1; i++)

{

if (i < str\_num)

{

inn >> B[i];

}

else

{

B[i] = 0;

}

}

for (i = 0; i < str\_num; i++)

{

Base[i] = number + i;

}

}

Min\_El Minimum()

{

Min\_El min;

min.min = F[0];

min.index = 1;

for (int i = 1; i < number + str\_num; i++)

{

if (F[i] <= min.min)

{

min.min = F[i];

min.index = i + 1;

}

}

return min;

}

//Заполнение таблицы

void Fill(int number, int str\_num)

{

int i, j;

for (i = 0; i < str\_num + 1; i++)

{

Table[i][0] = B[i];

Table2[i][0] = B[i];

}

for (j = 1; j < (str\_num + number + 1); j++)

{

Table[str\_num][j] = F[j - 1];

Table2[str\_num][j] = F[j - 1];

}

for (i = 0; i < str\_num; i++)

{

for (j = 1; j < number + 1; j++)

{

Table[i][j] = matrix[i][j - 1];

Table2[i][j] = matrix[i][j - 1];

}

}

for (i = 0; i < str\_num; i++)

{

for (j = number + 1; j < (str\_num + number + 1); j++)

{

Table[i][j] = one[i][j - number - 1];

Table2[i][j] = one[i][j - number - 1];

}

}

}

// Симплекс

void Siplex(double\*\* Tab, double\*\* Tab2, int str, int col)

{

cout.setf(ios::fixed);

cout.precision(2);

while ((\*min\_element(F, F + number + str\_num) < 0) && (flag == true))

{

int ii = 0;

int jj = 0;

cout << "Func:\n";

for (int n = 0; n < str\_num + number; n++)

{

cout << F[n] << "\t";

}

cout << "\n\n";

cout << "Table:\n";

for (int n = 0; n < str\_num + 1; n++)

{

for (int nn = 0; nn < (str\_num + number + 1); nn++)

{

cout << "| " << Tab[n][nn] << "\t";

}

cout << "\n";

}

cout << "\n\n";

Min\_El min;

min = Minimum();

int j = min.index;

cout << "Column: " << j << "\n" << "Min: " << min.min << "\n\n";

int num\_mas = 0;

//Вывод базиса

cout << "Mas: ";

for (ii = 0; ii < str; ii++)

{

if ((Tab[ii][j] <= 0) || (Tab[ii][j] == 0))

{

mas[ii] = 1000000;

num\_mas++;

}

else

{

mas[ii] = Tab[ii][0] / Tab[ii][j];

}

cout << "| " << mas[ii] << "\t";

}

cout << "\n\n";

if (num\_mas == str)

{

cout << "Function unlimited!" << "\n\n";

flag = false;

}

else

{

int i = 0;

double m = mas[0];

for (int n = 1; n < str; n++)

{

if (mas[n] < m)

{

i = n;

m = mas[n];

}

}

cout << "Line: " << i << "\t" << "Min: " << m << "\n\n";

for (jj = 0; jj < col + 1; jj++)

{

if ((jj != j) || (Tab[i][j] != 0))

{

Tab2[i][jj] = Tab[i][jj] / Tab[i][j];

}

}

for (ii = 0; ii < str + 1; ii++)

{

if (ii != i)

{

Tab2[ii][j] = 0;

}

}

if (Tab[i][j] != 0)

{

Tab2[i][j] = 1;

}

for (ii = 0; ii < str + 1; ii++)

{

for (jj = 0; jj < col + 1; jj++)

{

if ((ii != i) && (jj != j))

{

Tab2[ii][jj] = Tab[ii][jj] - ((Tab[ii][j] \* Tab[i][jj]) / Tab[i][j]);

}

}

}

//cout << "Table2:\n";

for (int n = 0; n < str\_num + 1; n++)

{

for (int nn = 0; nn < (str\_num + number + 1); nn++)

{

cout << "| " << Tab2[n][nn] << "\t";

}

cout << "\n";

}

cout << "\n\n";

int count = 0;

for (int nn = 1; nn < (str\_num + number + 1); nn++)

{

for (int n = 0; n < str\_num; n++)

{

if (Tab[n][nn] <= 0)

{

count++;

}

}

if (count == str\_num)

{

break;

}

count = 0;

}

for (int n = 0; n < str\_num + 1; n++)

{

for (int nn = 0; nn < (str\_num + number + 1); nn++)

{

Tab[n][nn] = Tab2[n][nn];

}

}

int count\_del = 0;

for (jj = 1; jj < col + 1; jj++)

{

F[jj - 1] = Tab[str][jj];

if (F[jj - 1] == 0)

{

count\_del++;

}

}

if (count\_del > str\_num)

{

cout << "Infinite Solutions!" << "\n\n";

flag = false;

}

else

{

Base[i] = j - 1;

cout << "Base: ";

for (int n = 0; n < str\_num; n++)

{

cout << "| " << Base[n] << "\t";

}

cout << "\n\n";

for (ii = 0; ii < str\_num; ii++)

{

cout << "X" << Base[ii] << " = " << Tab[ii][0] << "\n";

}

cout << "----------------------------------------------------------\n\n";

}

}

}

}

int main()

{

int i, j;

ifstream in("D:\\ЛЕКЦИИ\\LP\\3 Курс\\ИО\\ЛР1\\Simplex\_2\\ConsoleApplication1\\x64\\Debug\\Input.txt");

if (in.is\_open())

{

// Количество переменных

in >> number;

in >> str\_num;

Start();

Read(in);

}

in.close();

// Заполнение таблицы

Fill(number, str\_num);

Siplex(Table, Table2, str\_num, str\_num + number);

Stop();

return 0;

}

1. Экранные формы:

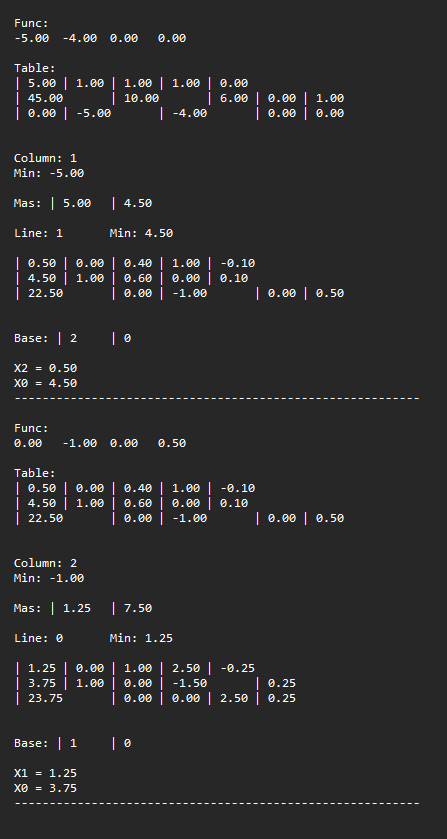


Рисунок 1 – Успешное выполнение программы.

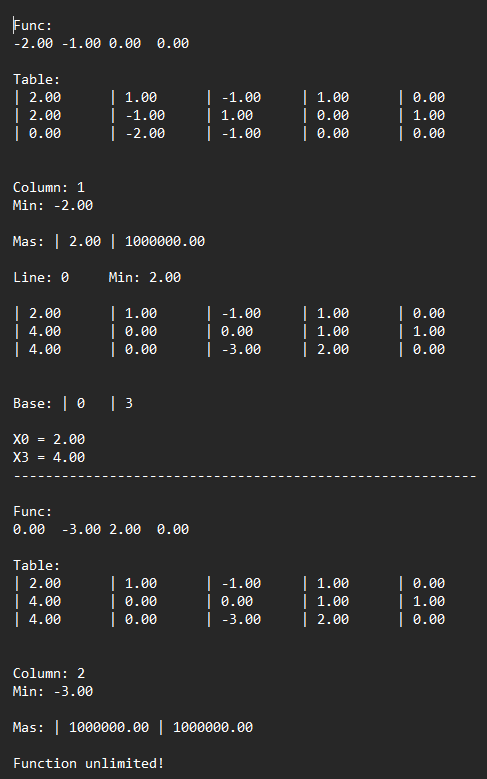


Рисунок 2 – Функция не ограничена.

1. Вывод:

В ходе выполнения лабораторной работы были закреплены на практике знания о симплексном методе решения задач линейного программирования и получены навыки его программной реализации. Была написана программа, реализующая симплекс-метод.