МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет математики и компьютерных наук**

**Кафедра вычислительной математики и информатики**

**КУРСОВАЯ РАБОТА**

**СИМПЛЕКСНЫЙ МЕТОД РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С.А. Бердник

(подпись, дата)

Направление подготовки 02.03.01 математика и компьютерные науки курс 3

Научный руководитель

доцент, кандидат физ.-мат. наук,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д.Г. Сокол

(подпись, дата)

Нормоконтролер

преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.С.Шимохина

(подпись, дата)

Краснодар

2020

**СОДЕРЖАНИЕ**

Введение ……………………………………….…………………………………………………………………………….3  
1 Постановка задачи ………………………………..………………………………………………………………..4  
2 Алгоритм решения задачи ..…………………………………………………………………………………..5  
3 Реализация алгоритма ……………………………………………………………………..…………………….8  
4 Тестовые примеры………………………………...……….……………………………………………………..14  
Заключение……………………………………………………………………………………………………………….18  
Список используемых источников ……………………….…………………………..………………….19

Приложение А Код программы ……………………………………………………………………………20

**ВВЕДЕНИЕ**

Линейное программирование или сокращенно ЛП – это раздел более общей теории математического программирования. Математическое программирование занимается изучением проблем принятия решений, которые могут быть математически сформулированы как задачи нахождения максимума (минимума) некоторой нелинейной функции (целевой функции) многих переменных, при заданной системе ограничений на основные переменные задачи.

Задача линейного программирования состоит в том, что необходимо максимизировать или минимизировать некоторый линейный функционал на многомерном пространстве при заданных линейных ограничениях.

Симплексный метод решения задач линейного программирования — вычислительная процедура, основанная на принципе последовательного улучшения решений — перехода от одной базисной точки к другой, для которой значение целевой функции больше (эти операции фиксируются в симплексной таблице). Доказано, что если оптимальное решение существует, то оно обязательно будет найдено через конечное число шагов (за исключением так называемой «вырожденной задачи, при которой возможно явление «зацикливания», т. е. многократного возврата к одному и тому же положению).

В качестве инструмента реализации алгоритма мною был выбран язык программирования С#.

**1 Постановка задачи**

В практике принятия решений выбор между различными вариантами (планами, решениями) предполагает поиск наилучшего. Для этого мы занимаемся поисками вариантов, требующих минимума затрат или максимума результата с учетом определенных ограничений (денег, ресурсов, времени).

Решить подобную задачу бывает непросто, особенно при наличии большого числа вариантов. Время и затраты при выборе оптимума не всегда оправданны: издержки поиска и перебора вариантов могут превысить достигнутый выигрыш.

Условия задачи на оптимум и цель, которая должна быть достигнута, могут быть выражены с помощью системы линейных уравнений. Поскольку уравнений меньше, чем неизвестных, задача обычно имеет не одно, а множество решений. Найти же нужно одно экстремальное решение.

Симплекс метод является универсальным, так как позволяет решить практически любую задачу линейного программирования, записанную в каноническом виде. Идея симплексного метода (метода последовательного улучшения плана) заключается в том, что начиная с некоторого исходного опорного решения осуществляется последовательно направленное перемещение по опорным решениям задачи к оптимальному. Значение целевой функции при этом перемещении для задач на максимум не убывает. Так как число опорных решений конечно, то через конечное число шагов получим оптимальное опорное решение. Опорным решением называется базисное неотрицательное решение.

**2 Алгоритм решения задачи**

В начале исходную задачу линейного программирования приводят к каноническому виду, затем составляют симплекс-таблицу вида:

Таблица 1 – симплекс таблица

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |

где в столбце «базис» указываются базисные переменные, а в последней строке столбца «базис» пишется f(x). В столбец «B» записываются свободные члены ограничений bi и значение целевой функции (на 1-м этапе оно равно 0, т.е. никакой прибыли).

В столбцах xj для не базисных переменных указываются коэффициенты при не базисных переменных из ограничений задачи. В столбцах базисных переменных содержится только 0 или 1 на пересечении столбца с соответствующей строкой базисной переменной.

В последней строке -cj – это коэффициенты при переменных целевой функции взятые с противоположным знаком.

Симплекс-таблица составлена, теперь опишем сам симплекс-метод.

Шаг 1**:** Выполняется проверка полученного базисного плана на оптимальность по условию: если при каком-либо ДБР (допустимое базисное решение) в симплекс-таблице все коэффициенты строки f(x) (то есть -cj) не отрицательны, то данное ДБР оптимально, следовательно конец решения. В противном случае:

Шаг 2**:** Переход к новому базисному плану. Для этого из числа небазисных переменных с отрицательными значениями в последней строке (то есть -cj < 0) выбирается переменная, вводимая в базис – xk, это переменная которой соответствует наибольшая по модулю отрицательная оценка:

Столбец, отвечающий переменной xk, называется главным, или ведущим. Элементы данного столбца обозначаются через aik.

Если окажется несколько одинаковых наибольших по модулю отрицательных оценок, то выбирается любая из соответственных переменных.

Шаг 3**:** Выбираем переменную r – переменную, которая выводится из базиса. Данная переменная находится из соотношения:

Строка таблицы, в которой получено наименьшее отношение элемента столбца «В» к соответствующему положительному элементу ведущего столбца, является ведущей, или главной.

Элементы главной строки обозначаются через arj. Выбранная переменная xr будет выводиться из базиса, то есть это исключаемая переменная.

Если окажется несколько одинаковых наименьших значений отношений, то выбирается любая из соответствующих им переменных.

Элемент, который стоит на пересечении главного столбца и строки называется главным, или ведущим, и обозначается ark.

Шаг 4: Для определения нового базисного плана проводится пересчет элементов симплекс-таблицы, и результаты заносятся в новую таблицу. Выбранные переменные среди базисных и не базисных, лежащих на главной строке и главном столбце, меняются местами.

Процедура пересчета элементов выполняется следующим образом:

а) элементы главной строки необходимо разделить на ведущий элемент:

б) элементы полученной строки умножаются на -aik, и результаты складываются с i-той строкой, причем i ≠ k:

**3 Реализация алгоритма**

Опишем код программы

Подключение библиотек

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

Класс Simplex реализует поиск оптимального решения

namespace Simplex

{

    public class Simplex

    {

На вход симплекс таблица без базисных переменных

        double[,] table;

 Симплекс таблица

        int m, n;

        List<int> basis;

 Список базисных переменных

        public Simplex(double[,] source)

        {

            m =source.GetLength(0);

            n = source.GetLength(1);

            table = new double[m, n + m - 1];

            basis = new List<int>();

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

            {

                for (int j = 0; j < table.GetLength(1); j++)

                {

                    if (j < n)

                        table[i, j] = source[i, j];

                    else

                        table[i, j] = 0;

Выставляем коэффициент 1 перед базисной переменной в строке

                if ((n + i) < table.GetLength(1))

                {

                    table[i, n + i] = 1;

                    basis.Add(n + i);

                }

            }

            n = table.GetLength(1);

        }

result - в этот массив будут записаны полученные значения X

        public double[,] Calculate(double[] result)

        {

            int mainCol, mainRow;

Ведущие столбец и строка

            while (!IsItEnd())

            {

                mainCol = findMainCol();

                mainRow = findMainRow(mainCol);

                basis[mainRow] = mainCol;

                double[,] new\_table = new double[m, n];

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

                    new\_table[mainRow, j] = table[mainRow, j] / table[mainRow, mainCol];

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

                {

                    if (i == mainRow)

                        continue;

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

                        new\_table[i, j] = table[i, j] - table[i, mainCol] \* new\_table[mainRow, j];

                }

                table = new\_table;

            }

Заносим в result найденные значения X

            for (int i = 0; i < result.Length; i++)

            {

                int k = basis.IndexOf(i + 1);

                if (k != -1)

                    result[i] = table[k, 0];

                else

                    result[i] = 0;

            }

            return table;

        }

 Проверка решения на оптимальность

        private bool IsItEnd()

        {

            bool flag = true;

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

            {

                if (table[m - 1, j] < 0)

                {

                    flag = false;

                    break;

                }

            }

            return flag;

        }

 Поиск ведущего столбца

        private int findMainCol()

        {

            int mainCol = 1;

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

                if (table[m - 1, j] < table[m - 1, mainCol])

                    mainCol = j;

            return mainCol;

        }

 Поиск ведущей строки

        private int findMainRow(int mainCol)

        {

            int mainRow = 0;

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

                if (table[i, mainCol] > 0)

                {

                    mainRow = i;

                    break;

                }

            for (int i = mainRow + 1; i < m - 1; i++)

                if ((table[i, mainCol] > 0) && ((table[i, 0] / table[i, mainCol]) < (table[mainRow, 0] / table[mainRow, mainCol])))

                    mainRow = i;

            return mainRow;

        }

    }

}

**4 Тестовые примеры**

*Пример 1*

Пример программы, использующей реализованный выше симплекс-метод. Решим, для примера, задачу с такими ограничениями и целевой функцией:

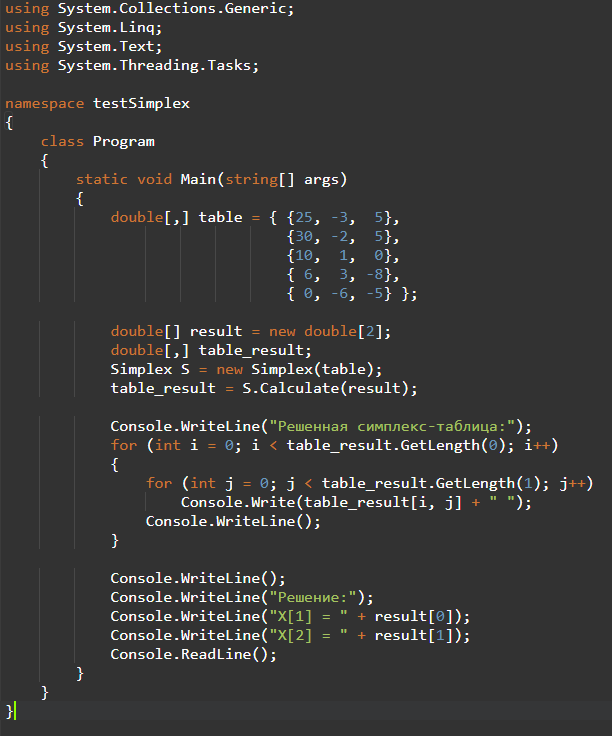


Рисунок 1 – Ввод исходных данных

При запуске программа решает задачу оптимизации целевой функции и выдает итоговый ответ.

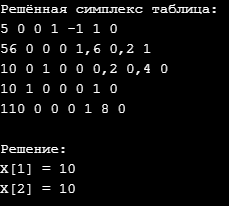


Рисунок 2 – Вывод результата на экран консоли

*Пример 2*

Решим, для примера, задачу с такими ограничениями и целевой функцией:

Пользователь вводит значения целевых функций, вектор свободных членов ограничений и матрицу коэффициентов ограничений из поставленной задачи.

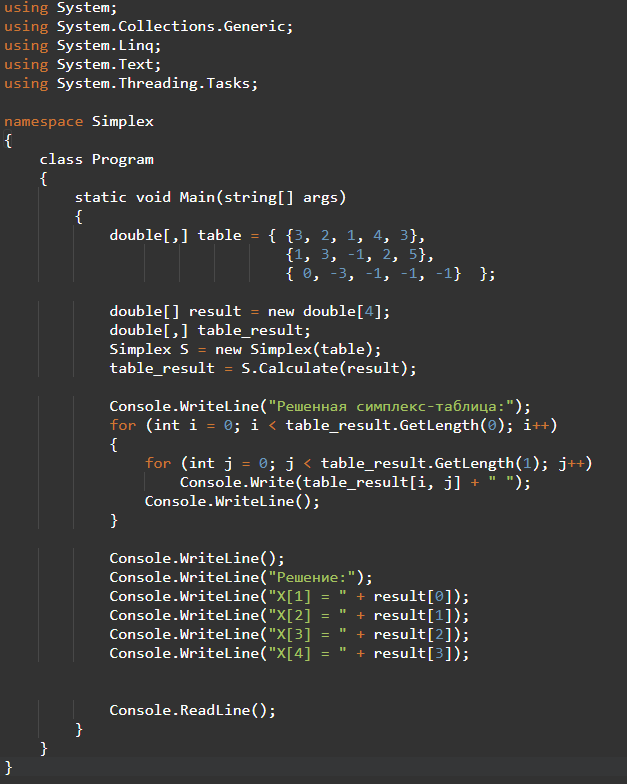


Рисунок 3 – Ввод исходных данных

При запуске программа решает задачу оптимизации целевой функции и выдает итоговый ответ.

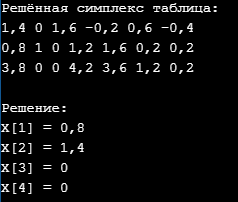


Рисунок 4 – Вывод результата на экран консоли

**ЗАКЛЮЧЕНИЕ**

В результате работы был освоен и реализован алгоритм решения базовой задачи симплекс метода . Мы рассмотрели несколько задач с разным количеством ограничений и переменных и убедились в универсальности данного метода.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1 Красс М.С., Чупрынов Б.П. Основы математики и ее приложения в экономическом образовании: Учеб. — 2-е изд., испр. 2001.

2 Лунгу К. Н. - Линейное программирование. Руководство к решению задач. 2005.  
3 Бен Албахари, Джозеф Албахари - C# 7.0. Справочник. Полное описание языка. 7-е издание. 2018.

4 Павлова Т. Н. Линейное программирование: учеб.пособие / Т. Н. Павлова, О. А. Ракова. 2002.

**ПРИЛОЖЕНИЕ А**

Код программы

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Simplex

{

    public class Simplex

        double[,] table;

        int m, n;

        List<int> basis;

        public Simplex(double[,] source)

        {

            m =source.GetLength(0);

            n = source.GetLength(1);

            table = new double[m, n + m - 1];

            basis = new List<int>();

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

            {

                for (int j = 0; j < table.GetLength(1); j++)

                {

                    if (j < n)

                        table[i, j] = source[i, j];

                    else

                        table[i, j] = 0;

                }

                if ((n + i) < table.GetLength(1))

                {

                    table[i, n + i] = 1;

                    basis.Add(n + i);

                }

            }

            n = table.GetLength(1);

        }

               public double[,] Calculate(double[] result)

        {

            int mainCol, mainRow; //ведущие столбец и строка

            while (!IsItEnd())

            {

                mainCol = findMainCol();

                mainRow = findMainRow(mainCol);

                basis[mainRow] = mainCol;

                double[,] new\_table = new double[m, n];

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

                    new\_table[mainRow, j] = table[mainRow, j] / table[mainRow, mainCol];

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

                {

                    if (i == mainRow)

                        continue;

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

                        new\_table[i, j] = table[i, j] - table[i, mainCol] \* new\_table[mainRow, j];

                }

                table = new\_table;

            }

            for (int i = 0; i < result.Length; i++)

            {

                int k = basis.IndexOf(i + 1);

                if (k != -1)

                    result[i] = table[k, 0];

                else

                    result[i] = 0;

            }

            return table;

        }

        private bool IsItEnd()

        {

            bool flag = true;

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

            {

                if (table[m - 1, j] < 0)

                {

                    flag = false;

                    break;

                }

            }

            return flag;

        }

        private int findMainCol()

        {

            int mainCol = 1;

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

                if (table[m - 1, j] < table[m - 1, mainCol])

                    mainCol = j;

            return mainCol;

        }

        private int findMainRow(int mainCol)

        {

            int mainRow = 0;

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

                if (table[i, mainCol] > 0)

                {

                    mainRow = i;

                    break;

                }

            for (int i = mainRow + 1; i < m - 1; i++)

                if ((table[i, mainCol] > 0) && ((table[i, 0] / table[i, mainCol]) < (table[mainRow, 0] / table[mainRow, mainCol])))

                    mainRow = i;

            return mainRow;

        }

    }

}