**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Программный продукт**  
по курсовой работе  
по дисциплине «Теория информационных процессов и систем»  
  
**РУКОВОДСТВО ПРОГРАММИСТА**

Выполнил:  
студент гр. ИСТбд-31  
Прохоров Е.Э.  
  
Проверил:  
Новиков А.А.

Оглавление

[Необходимое программное обеспечение 4](#_Toc374384276)

[Описание основных файлов в проекте 4](#_Toc374384277)

[Описание классов в программе 4](#_Toc374384278)

[Описание методов программы 5](#_Toc374384279)

[Simplex 5](#_Toc374384280)

[SearchBase 5](#_Toc374384281)

[CheckNull 5](#_Toc374384282)

[CountIskustv 5](#_Toc374384283)

[ProverkaStroki 5](#_Toc374384284)

[PerehodArray 5](#_Toc374384285)

[RabotaSOgranich 5](#_Toc374384286)

[CheckingTargetFunctionEls 6](#_Toc374384287)

[RabotaSZeleZ 6](#_Toc374384288)

[PoiskVedushego 6](#_Toc374384289)

[PoiskMinZnach 6](#_Toc374384290)

[PoiskVedushei 6](#_Toc374384291)

[Editbase 6](#_Toc374384292)

[PrivedenieKBasisu 6](#_Toc374384293)

[PoluchenieNullVStolbe 6](#_Toc374384294)

[Writing 6](#_Toc374384295)

[GetConditionSigns 6](#_Toc374384296)

[GetConditionsCoefficients 7](#_Toc374384297)

[CreateTargetFunctionMatrix 7](#_Toc374384298)

[CreateEquationMatrix 7](#_Toc374384299)

[CheckArtificialAddition 7](#_Toc374384300)

[CheckArtificialExistence 7](#_Toc374384301)

[AddArtificialVariableToItem 7](#_Toc374384302)

[ExpressArtifialVariables 7](#_Toc374384303)

[GetNamesFectiveVariables 7](#_Toc374384304)

[ExpressArtificialFunction 7](#_Toc374384305)

[ChangeValueSide 8](#_Toc374384306)

[GetArtificialFunctionMatrix 8](#_Toc374384307)

[FillTableTitle 8](#_Toc374384308)

[FillColumnBase 8](#_Toc374384309)

[FillSimplexMatrixByNulls 8](#_Toc374384310)

[SetUsualView 8](#_Toc374384311)

[SearchAndSetValue 8](#_Toc374384312)

[FillMatrixByCoefficients 8](#_Toc374384313)

[FormTableForSimplexMethod 8](#_Toc374384314)

[PolushenieBase 9](#_Toc374384315)

[SimplexSolved 9](#_Toc374384316)

[UstanovkaZnach 9](#_Toc374384317)

[SizdanieArry 9](#_Toc374384318)

[GetUnknownVarCol 9](#_Toc374384319)

[PoiskNeizvestnZnach 9](#_Toc374384320)

[RaschetNeResheniya 9](#_Toc374384321)

[PoluchenieResheniya 9](#_Toc374384322)

[Графический интерфейс программы 10](#_Toc374384323)

[Исходный код основного класса SimplexMet 13](#_Toc374384324)

# Необходимое программное обеспечение

Для модификации кода программы или любых исправлений необходимо использовать следующее программное обеспечение:

1. Операционная система Windows.
2. Microsoft Visual C# 2010 и выше.
3. Текстовый редактор, например Notepad++.

# Описание основных файлов в проекте

В обозревателе решений Microsoft Visual C# 2010 можно увидеть следующее (Рис.1).

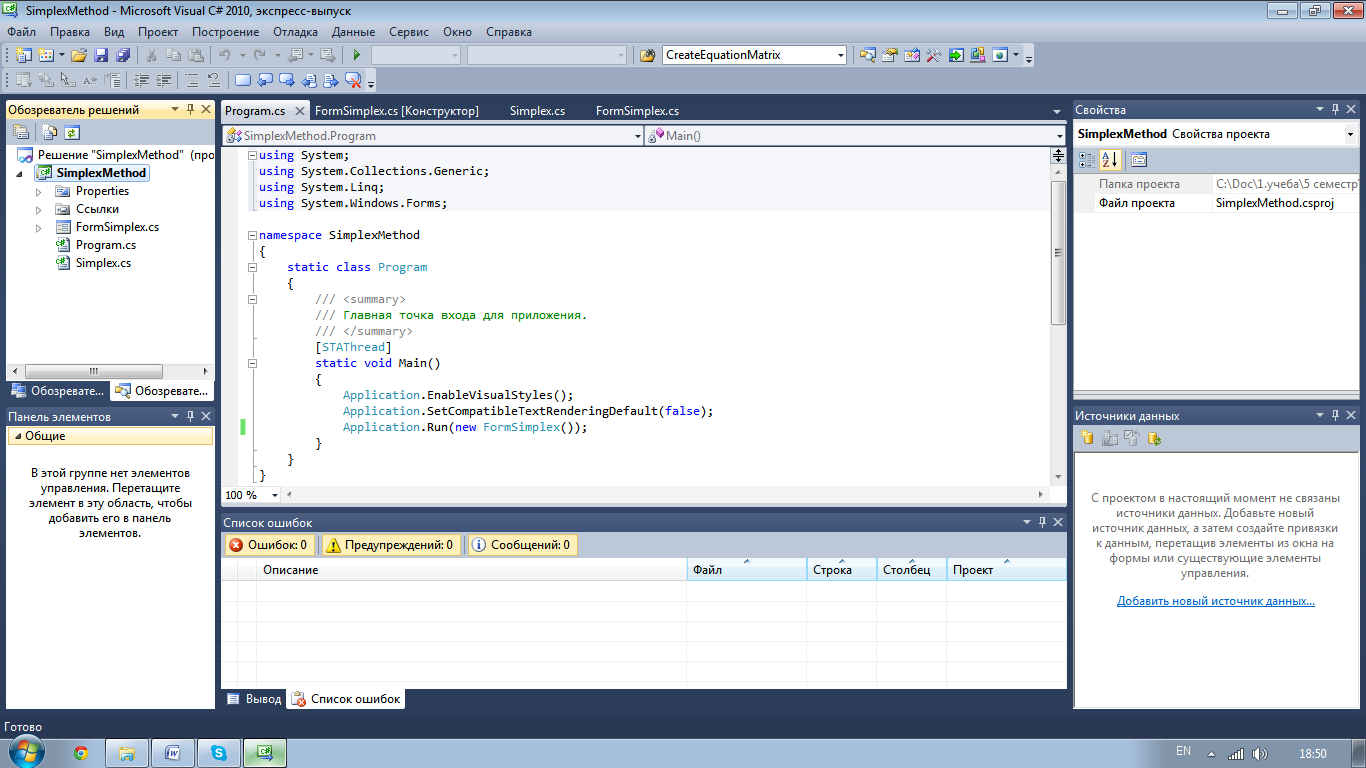


Рис.1 Обозреватель решений

Основными файлами можно считать:

* FormSimplex.cs – файл, отвечающий за отображение формы и обработку событий, создаваемых интерфейсами программы.
* Program.cs – файл, отвечающий за запуск приложения.
* Simplex.cs – файл, содержащий основной класс для вычисления решения ЗЛП симплекс методом.

# Описание классов в программе

* FormSimplex – класс для обработки событий, которые генерируются элементами управления формы (Рис.2)

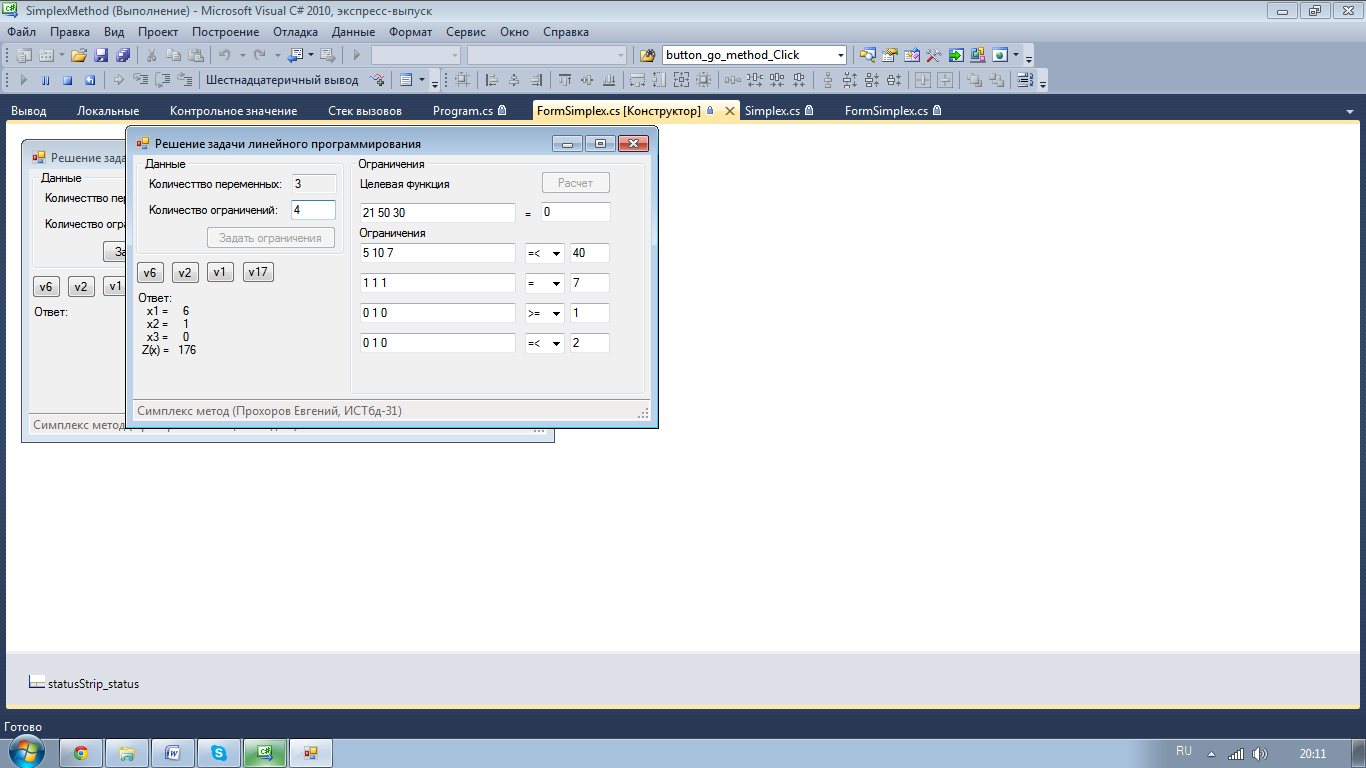


Рис.2 Внешний вид приложения

* Program – статический класс с основным методом Main(), который отвечает за запуск приложения.
* SimplexMet – класс, который отвечает за нахождение решения.
* TCT – класс, который облегчает программисту добавление на форму контролов для задания ограничений.

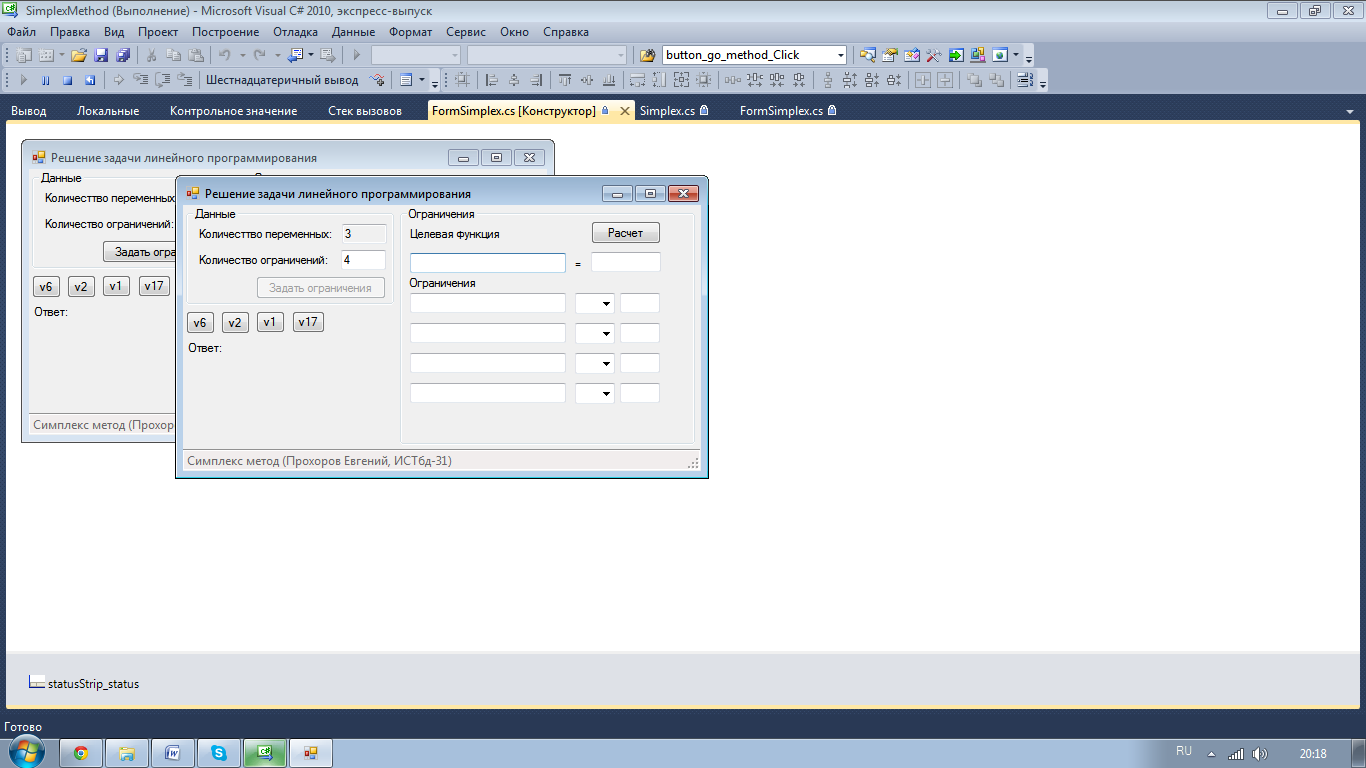


Рис.3 Генерируемый контролы с помощью TCT для задания ограничений

# Описание методов программы

Основным классом в приложении является класс – SimplexMet. Основные функции и описания к ним данного класса:

## Simplex

**Описание:**Основной метод, реализующий работу симплекс-алгоритма

**Дескриптор:**public static void Simplex(ref object[,] matrix)

**Тип возвращаемого значения:**void

## SearchBase

**Описание:**Построение и поиск по базе

**Дескриптор:**private static void SearchBase(object[,] \_matrix)

**Тип возвращаемого значения:**void

## CheckNull

**Описание:**проверка на нули в столбце

**Дескриптор:**private static bool CheckNull(object[,] \_matrix, int \_numRow, int \_numCol)

**Тип возвращаемого значения:**bool

## CountIskustv

**Описание:**Поиск числа искусственных переменных

**Дескриптор:**private static int CountIskustv(object[,] \_matrix)

**Тип возвращаемого значения:**int

## ProverkaStroki

**Описание:**Проверка на равенство 1-це исскуственных переменных, а остальных 0-лю

**Дескриптор:**private static bool ProverkaStroki(object[,] \_matrix)

**Тип возвращаемого значения:**bool

## PerehodArray

**Описание:**переход от искусственной функции к целевой

**Дескриптор:**private static object[,] PerehodArray(ref object[,] \_matrixBefore)

**Тип возвращаемого значения:**object[,]

## RabotaSOgranich

**Описание:**Отказ от искусственной переменной

**Дескриптор:**private static void RabotaSOgranich(ref object[,] \_matrix)

**Тип возвращаемого значения:**void

## CheckingTargetFunctionEls

**Описание:**Проверка на нахождение решения

**Дескриптор:**private static bool CheckingTargetFunctionEls(object[,] \_matrix)

**Тип возвращаемого значения:**bool

## RabotaSZeleZ

**Описание:**Ищем результат

**Дескриптор:**private static void RabotaSZeleZ(ref object[,] \_matrix)

**Тип возвращаемого значения:**void

## PoiskVedushego

**Описание:**поиск ведущего столбца в строке

**Дескриптор:**private static int PoiskVedushego(object[,] \_matrix)

**Тип возвращаемого значения:**int

## PoiskMinZnach

**Описание:**Поиск первого неотрицательного

**Дескриптор:**private static int PoiskMinZnach(object[,] \_matrix, int \_min\_j, int \_subtractValue)

**Тип возвращаемого значения:**int

## PoiskVedushei

**Описание:**Для поиска номера ведущей строки в матрице

**Дескриптор:**private static int PoiskVedushei(object[,] \_matrix, int \_min\_j, int \_subtractValue)

**Тип возвращаемого значения:**int

## Editbase

**Описание:**Изменение базы

**Дескриптор:**private static void Editbase(object[,] \_matrix, int \_min\_i, int \_min\_j)

**Тип возвращаемого значения:**void

## PrivedenieKBasisu

**Описание:**Приведение к базису

**Дескриптор:**private static void PrivedenieKBasisu(object[,] \_matrix, int \_min\_i, int \_min\_j)

**Тип возвращаемого значения:**void

## PoluchenieNullVStolbe

**Описание:**Получаем нули в столбце

**Дескриптор:**private static void PoluchenieNullVStolbe(object[,] \_matrix, int \_min\_i, int \_min\_j)

**Тип возвращаемого значения:**void

## Writing

**Описание:**Вывод матрицы

**Дескриптор:**public static void Writing(object[,] \_matrix)

**Тип возвращаемого значения:**void

## GetConditionSigns

**Описание:**Получение знаков условий

**Дескриптор:**private static string[] GetConditionSigns(TCT[] arry, int \_countConditions)

**Тип возвращаемого значения:**string[]

## GetConditionsCoefficients

**Описание:**Получение коэффициентов условий

**Дескриптор:**private static double[,] GetConditionsCoefficients(TCT[] arry, int \_countConditions, int \_countVariables, string[] str\_zele)

**Тип возвращаемого значения:**double[,]

## CreateTargetFunctionMatrix

**Описание:**Создание матрицы целевой функции

**Дескриптор:**public static object[,] CreateTargetFunctionMatrix(TCT[] arry, int \_countConditions, int \_countVariables, string[] str\_zell)

**Тип возвращаемого значения:**object[,]

## CreateEquationMatrix

**Описание:**Моделирование уравнений(не неравенств, т.к. сразу создаем фективные переменные) условий через матрицу

**Дескриптор:**public static void CreateEquationMatrix(TCT[] arry, int \_countConditions, int \_countVariables, ref List \_conditionSimplexMatrixes, string[] str\_zell)

**Тип возвращаемого значения:**void

## CheckArtificialAddition

**Описание:**Проверка необходимости добавления искусственной переменной

**Дескриптор:**private static bool CheckArtificialAddition(object[,] \_conditionItem)

**Тип возвращаемого значения:**bool

## CheckArtificialExistence

**Описание:**Проверка на уже добавленность искусственной переменной

**Дескриптор:**private static bool CheckArtificialExistence(object[,] \_conditionItem)

**Тип возвращаемого значения:**bool

## AddArtificialVariableToItem

**Описание:**Добавление искусственных переменных к элементу списка

**Дескриптор:**private static void AddArtificialVariableToItem(List \_conditionSimplexMatrixes, object[,] \_conditionItem, int \_countVariables)

**Тип возвращаемого значения:**void

## ExpressArtifialVariables

**Описание:**Выражаем искусственные переменные

**Дескриптор:**private static void ExpressArtifialVariables(ref List \_conditionsWithArtVars)

**Тип возвращаемого значения:**void

## GetNamesFectiveVariables

**Описание:**Ищем имена фективных переменных, участвующих в образовании искусственной ф-ии

**Дескриптор:**private static List GetNamesFectiveVariables(List \_conditionsWithArtVars)

**Тип возвращаемого значения:**List

## ExpressArtificialFunction

**Описание:**Создаем и выражаем искусственную функцию

**Дескриптор:**private static object[,] ExpressArtificialFunction(List \_conditionsWithArtVars, int \_countVariables)

**Тип возвращаемого значения:**object[,]

## ChangeValueSide

**Описание:**Перенесем значение (Value) в правую часть искусственной ф-ции(при этом не забываем учесть знак "-")

**Дескриптор:**private static object[,] ChangeValueSide(object[,] \_artificialFunction)

**Тип возвращаемого значения:**object[,]

## GetArtificialFunctionMatrix

**Описание:**Выражание искусственной функции (поэтапно)

**Дескриптор:**public static object[,] GetArtificialFunctionMatrix(List \_conditionSimplexMatrixes, int \_countVariables)

**Тип возвращаемого значения:**object[,]

## FillTableTitle

**Описание:**Заполним заголовок таблицы и первый столбец (базу)

**Дескриптор:**private static void FillTableTitle(ref object[,] \_simplexTable, int \_countVariables)

**Тип возвращаемого значения:**void

## FillColumnBase

**Описание:**Формирование столбца "Base"

**Дескриптор:**private static void FillColumnBase(ref object[,] \_simplexTable)

**Тип возвращаемого значения:**void

## FillSimplexMatrixByNulls

**Описание:**Заполним симплекс-матрицу нулями

**Дескриптор:**private static void FillSimplexMatrixByNulls(ref object[,] \_simplexTable)

**Тип возвращаемого значения:**void

## SetUsualView

**Описание:**Приводим все симплекс-условия

**Дескриптор:**private static void SetUsualView(ref List \_conditionSimplexMatrixes)

**Тип возвращаемого значения:**void

## SearchAndSetValue

**Описание:**Заносим значение через поиск по имени

**Дескриптор:**private static void SearchAndSetValue(object[,] \_simplexTable, string \_varName, int \_value, int \_rowIndex)

**Тип возвращаемого значения:**void

## FillMatrixByCoefficients

**Описание:**Заполняем матрицу коэффициентами условий, целевой ф-ии, искусственной ф-ии

**Дескриптор:**private static void FillMatrixByCoefficients(ref object[,] \_simplexTable, object[,] \_targetFunctionMatrix, List \_conditionSimplexMatrixes, object[,] \_artificialFunctionMatrix)

**Тип возвращаемого значения:**void

## FormTableForSimplexMethod

**Описание:**Формируем таблицу для симплекс-метода

**Дескриптор:**public static object[,] FormTableForSimplexMethod(object[,] \_targetFunctionMatrix, List \_conditionSimplexMatrixes, object[,] \_artificialFunctionMatrix, int \_countSimplexConditions, int \_countVariables)

**Тип возвращаемого значения:**object[,]

## PolushenieBase

**Описание:**Забор значений из столбца Base полученного решения

**Дескриптор:**private static object[,] PolushenieBase(object[,] \_simplexTable)

**Тип возвращаемого значения:**object[,]

## SimplexSolved

**Описание:**Проверка на завершенность решения симплекс-алгоритмом

**Дескриптор:**private static string SimplexSolved(object[,] \_solutionValues, object[,] \_targetFunction)

**Тип возвращаемого значения:**string

## UstanovkaZnach

**Описание:**Установка значения

**Дескриптор:**private static void UstanovkaZnach(object[,] \_findingMatrix, int \_j, object[,] \_solutionValues)

**Тип возвращаемого значения:**void

## SizdanieArry

**Описание:**Формирование матрицы для расчета неизвестного значения

**Дескриптор:**private static object[,] SizdanieArry(object[,] \_targetFunction, object[,] \_solutionValues)

**Тип возвращаемого значения:**object[,]

## GetUnknownVarCol

**Описание:**Поиск номера столбца с неизвестной переменной в матрице поиска

**Дескриптор:**private static int GetUnknownVarCol(object[,] \_findingMatrix, string \_unknownVariable)

**Тип возвращаемого значения:**int

## PoiskNeizvestnZnach

**Описание:**Поиск неизвестного значения

**Дескриптор:**private static void PoiskNeizvestnZnach(object[,] \_findingMatrix, string \_unknownVariable)

**Тип возвращаемого значения:**void

## RaschetNeResheniya

**Описание:**расчет недостающего значения и запись его в матрицу

**Дескриптор:**private static object[,] RaschetNeResheniya(object[,] \_simplexTable, object[,] \_targetFunction)

**Тип возвращаемого значения:**object[,]

## PoluchenieResheniya

**Описание:**Получить решение

**Дескриптор:**public static object[,] PoluchenieResheniya(object[,] \_simplexTable, object[,] \_targetFunction, ref double minVertex, string zele, Label lab)

**Тип возвращаемого значения:**object[,]

# Графический интерфейс программы

За создание внешнего вида приложения и обработку им соответствующих событий, которые генерируются с помощью интерфейса программы, отвечает класс – FormSimplex, который наследуется от класса Form. Код данного класса хорошо прокомментирован, поэтому дальнейшее описание не требуется:

public partial class FormSimplex : Form

{

public FormSimplex()

{

InitializeComponent();

//создание помощника для записи решения в файл

Stream myFile = File.Create("Reshenie.txt");

TextWriterTraceListener myTextListener = new

TextWriterTraceListener(myFile);

Trace.Listeners.Add(myTextListener);

}

private int \_x\_count;//сколько действительных переменных

private int \_ogranich\_count;//количество ограничений

private TCT[] \_ogr\_arry;//массив конторлов с ограничениями

private int \_fekt\_x;//количесвто фективных переменных

public static String[] textbox\_tmp = new String[2];

private void button\_get\_ogranich\_Click(object sender, EventArgs e)

{

//отрисовка формы для задания ограничений

button\_get\_ogranich.Enabled = false;

button\_go\_method.Enabled = true;

\_x\_count = Convert.ToInt32(textBox\_count\_x.Text);

\_ogranich\_count = Convert.ToInt32(textBox\_count\_ogranich.Text);

Trace.WriteLine("Кол-во переменных: " + \_x\_count + " Кол-во ограничений: " + \_ogranich\_count);

\_ogr\_arry = new TCT[\_ogranich\_count];

int hm = 86;

for (int i = 0; i < \_ogranich\_count; i++)

{

TCT tmp = new TCT(hm, i.ToString());

tmp.PasteControl(groupBox\_ogranich);

hm += 30;

if (groupBox\_ogranich.Height < hm + 10)

{

groupBox\_ogranich.Height += 30;

}

\_ogr\_arry[i] = tmp;

}

}

//определяет сколько будет фиктивных переменных

private int GetFektPerem(TCT[] sender)

{

int ret\_val = 0;

foreach (TCT m in sender)

{

if (m.c.SelectedIndex == 0 || m.c.SelectedIndex == 2)

{

ret\_val++;

}

}

return ret\_val;

}

//опеределение количество искуственных переменных

private int GetIskustvPerem(TCT[] sender)

{

int ret\_val = 0;

foreach (TCT m in sender)

{

if (m.c.SelectedIndex == 2)

{

ret\_val++;

}

}

Trace.WriteLine("Количество искусственных переменных: " + ret\_val);

return ret\_val;

}

//Кнопка старта поиска решения

private void button\_go\_method\_Click(object sender, EventArgs e)

{

textbox\_tmp[0] = textBox\_z.Text;

textbox\_tmp[1] = textBox\_z\_z.Text;

button\_go\_method.Enabled = false;

\_fekt\_x = GetFektPerem(\_ogr\_arry);

func\_zele = SimplexMet.CreateTargetFunctionMatrix(\_ogr\_arry, \_ogranich\_count, \_x\_count, textbox\_tmp);

SimplexMet.CreateEquationMatrix(\_ogr\_arry, \_ogranich\_count, \_x\_count, ref usloviya\_list, textbox\_tmp);

SimplexMet.AddArtificialVariables(ref usloviya\_list, \_x\_count);

iskustv\_function = SimplexMet.GetArtificialFunctionMatrix(usloviya\_list, \_x\_count);

simplextable\_arry = SimplexMet.FormTableForSimplexMethod(func\_zele, usloviya\_list, iskustv\_function, usloviya\_list.Count, \_x\_count);

Trace.WriteLine("\nСоставление симплекс таблицы: \n");

SimplexMet.Writing(simplextable\_arry);

Trace.WriteLine("\nГотовая симплекс таблица с базой:\n");

SimplexMet.Simplex(ref simplextable\_arry);

Trace.WriteLine("\nРешение найдено!\n");

string zele = textBox\_z.Text + " " + textBox\_z\_z.Text;

result\_arry = SimplexMet.PoluchenieResheniya(simplextable\_arry, func\_zele, ref minimum, zele, label\_otv);

TraceMatrix(result\_arry);

Trace.WriteLine("\n\nОтвет: Z(x) = " + minimum);

// Сохранение все в файл

Trace.Flush();

}

//вывод матрицы на экран

private void TraceMatrix(object[,] \_matrix)

{

for (int i = 0; i < \_matrix.GetLength(0); i++)

{

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

{

Trace.Write(String.Format("{0,16}", \_matrix[i, j]));

}

Trace.WriteLine("");

}

}

//сплит для строки

private string[] SpitTextBox(TextBox sender)

{

string[] ret\_arry = sender.Text.Split(' ');

return ret\_arry;

}

//преобразование комбокса в число

private float ComboxToFloat(ComboBox sender)

{

float ret\_val = 0F;

if (sender.SelectedIndex == 0)

{

ret\_val = -1F;

}

else

{

if (sender.SelectedIndex == 1)

{

ret\_val = 0F;

}

else

{

ret\_val = 1F;

}

}

return ret\_val;

}

private object[,] func\_zele;

private List<object[,]> usloviya\_list = new List<object[,]>();

private object[,] iskustv\_function;

private object[,] simplextable\_arry;

private object[,] result\_arry;

private double minimum;

private void button1\_Click(object sender, EventArgs e)

{

string[] val = { "8 12 13", "1 1 1", "0 0 1", "0 0 1" };

string[] val\_c = { "200", "13", "4", "5" };

for (int i = 0; i < \_ogr\_arry.Length; i++)

{

\_ogr\_arry[i].t1.Text = val[i];

\_ogr\_arry[i].t2.Text = val\_c[i];

}

\_ogr\_arry[0].c.SelectedIndex = 2;

\_ogr\_arry[1].c.SelectedIndex = 1;

\_ogr\_arry[2].c.SelectedIndex = 0;

\_ogr\_arry[3].c.SelectedIndex = 2;

textBox\_z.Text = "17 28 25";

textBox\_z\_z.Text = "0";

}

private void buttonv2\_Click(object sender, EventArgs e)

{

string[] val = { "5 10 7", "1 1 1", "0 1 0", "0 1 0" };

string[] val\_c = { "40", "7", "1", "2" };

for (int i = 0; i < \_ogr\_arry.Length; i++)

{

\_ogr\_arry[i].t1.Text = val[i];

\_ogr\_arry[i].t2.Text = val\_c[i];

}

\_ogr\_arry[0].c.SelectedIndex = 2;

\_ogr\_arry[1].c.SelectedIndex = 1;

\_ogr\_arry[2].c.SelectedIndex = 0;

\_ogr\_arry[3].c.SelectedIndex = 2;

textBox\_z.Text = "21 50 30";

textBox\_z\_z.Text = "0";

}

private void textBox\_count\_ogranich\_TextChanged(object sender, EventArgs e)

{

button\_get\_ogranich.Enabled = true;

}

private void button\_v1\_Click(object sender, EventArgs e)

{

string[] val = { "10 12 17", "1 1 1", "1 0 0", "1 0 0" };

string[] val\_c = { "50", "4", "1", "2" };

for (int i = 0; i < \_ogr\_arry.Length; i++)

{

\_ogr\_arry[i].t1.Text = val[i];

\_ogr\_arry[i].t2.Text = val\_c[i];

}

\_ogr\_arry[0].c.SelectedIndex = 2;

\_ogr\_arry[1].c.SelectedIndex = 1;

\_ogr\_arry[2].c.SelectedIndex = 0;

\_ogr\_arry[3].c.SelectedIndex = 2;

textBox\_z.Text = "15 20 31";

textBox\_z\_z.Text = "0";

}

private void button\_v5\_Click(object sender, EventArgs e)

{

string[] val = { "5 5 17", "1 1 1", "0 0 1", "0 0 1" };

string[] val\_c = { "150", "15", "2", "3" };

for (int i = 0; i < \_ogr\_arry.Length; i++)

{

\_ogr\_arry[i].t1.Text = val[i];

\_ogr\_arry[i].t2.Text = val\_c[i];

}

\_ogr\_arry[0].c.SelectedIndex = 2;

\_ogr\_arry[1].c.SelectedIndex = 1;

\_ogr\_arry[2].c.SelectedIndex = 0;

\_ogr\_arry[3].c.SelectedIndex = 2;

textBox\_z.Text = "27 23 13";

textBox\_z\_z.Text = "0";

}

}

# Исходный код основного класса SimplexMet

class SimplexMet

{

//Основной метод, реализующий работу симплекс-алгоритма

public static void Simplex(ref object[,] matrix)

{

SearchBase(matrix);

RabotaSOgranich(ref matrix);

RabotaSZeleZ(ref matrix);

}

//Посторение и поиск по базе

private static void SearchBase(object[,] \_matrix)

{

for (int i = 1; i < \_matrix.GetLength(0); i++)

{

if ((string)\_matrix[i, 0] == "x")

{

for (int j = 2 ; j < \_matrix.GetLength(1); j++)

{

if ((int)\_matrix[i, j] == 1 && CheckNull(\_matrix, i, j))

{

\_matrix[i, 0] = \_matrix[0, j];

break;

}

}

}

}

}

//проверка на нули в столбце

private static bool CheckNull(object[,] \_matrix, int \_numRow, int \_numCol)

{

if (((string)\_matrix[0, \_numCol]).Contains('x'))

return false;

for (int i = \_matrix.GetLength(0) - 1; i > 0; i--)

{

if (Convert.ToDouble(\_matrix[i, \_numCol]) == 0 || Convert.ToDouble(\_matrix[\_numRow, \_numCol]) == 1)

continue;

else

return false;

}

return true;

}

//Поиск числа искусственных переменных

private static int CountIskustv(object[,] \_matrix)

{

int artCounter = 0;

for (int i = 0; i < \_matrix.GetLength(1); i++)

if ((\_matrix[0, i] as string).Contains("art"))

artCounter++;

return artCounter;

}

//Проверка на равенство 1-це исскуственных переменных, а остальных 0-лю

private static bool ProverkaStroki(object[,] \_matrix)

{

for (int j = 2; j < \_matrix.GetLength(1) - CountIskustv(\_matrix); j++)

{

if (Convert.ToDouble(\_matrix[\_matrix.GetLength(0) - 1, j]) == 0)

continue;

else

return false;

}

return true;

}

//переход от искусственной функции к целевой

private static object[,] PerehodArray(ref object[,] \_matrixBefore)

{

object[,] matrixAfter = new object[\_matrixBefore.GetLength(0) - 1, \_matrixBefore.GetLength(1) - CountIskustv(\_matrixBefore)];

for (int i = 0; i < matrixAfter.GetLength(0); i++)

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

matrixAfter[i, j] = \_matrixBefore[i, j];

return matrixAfter;

}

//Отказ от искусственной переменной

private static void RabotaSOgranich(ref object[,] \_matrix)

{

while (!ProverkaStroki(\_matrix))

{

Writing(\_matrix);

int j\_min = PoiskVedushego(\_matrix);

int i\_min = PoiskVedushei(\_matrix, j\_min, 2);

Trace.WriteLine("Ведущим элементом был выбран элемено " + i\_min + " строки и " + j\_min+" столбца");

Editbase(\_matrix, i\_min, j\_min);

PrivedenieKBasisu(\_matrix, i\_min, j\_min);

PoluchenieNullVStolbe(\_matrix, i\_min, j\_min);

}

Writing(\_matrix);

\_matrix = PerehodArray(ref \_matrix);

Trace.WriteLine("Отбрасывание искусственных переменных и искусственной функции W");

Writing(\_matrix);

}

//Проверка на нахождение решения

private static bool CheckingTargetFunctionEls(object[,] \_matrix)

{

for (int j = 2; j < \_matrix.GetLength(1); j++)

{

if (Convert.ToDouble(\_matrix[\_matrix.GetLength(0) - 1, j]) < 0)

return false;

else

continue;

}

return true;

}

//Ищем результат

private static void RabotaSZeleZ(ref object[,] \_matrix)

{

while (!CheckingTargetFunctionEls(\_matrix))

{

Writing(\_matrix);

int j\_min = PoiskVedushego(\_matrix);

int i\_min = PoiskVedushei(\_matrix, j\_min, 1);

Trace.WriteLine("Ведущий элемент строка " + i\_min + " и столбец " + j\_min);

Editbase(\_matrix, i\_min, j\_min);

PrivedenieKBasisu(\_matrix, i\_min, j\_min);

PoluchenieNullVStolbe(\_matrix, i\_min, j\_min);

}

Trace.WriteLine("Результат получился!");

Writing(\_matrix);

}

//поиск ведущего столбца в строке

private static int PoiskVedushego(object[,] \_matrix)

{

int min\_j = 2;

for (int j = 2; j < \_matrix.GetLength(1) - CountIskustv(\_matrix); j++)

if (Convert.ToDouble(\_matrix[\_matrix.GetLength(0) - 1, j]) < Convert.ToDouble(\_matrix[\_matrix.GetLength(0) - 1, min\_j]))

min\_j = j;

return min\_j;

}

//Поиск первого неотрицательного

private static int PoiskMinZnach(object[,] \_matrix, int \_min\_j, int \_subtractValue)

{

for (int i = 1; i < \_matrix.GetLength(0) - \_subtractValue; i++)

{

if ((Convert.ToDouble(\_matrix[i, \_min\_j]) > 0) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) > 0))

{

return i;

}

}

return -1;

}

//Для поиска номера ведущей строки в матрице

private static int PoiskVedushei(object[,] \_matrix, int \_min\_j, int \_subtractValue)

{

int min\_i = PoiskMinZnach(\_matrix, \_min\_j, \_subtractValue);

double minDivisionRes = Convert.ToDouble(\_matrix[min\_i, 1]) / Convert.ToDouble(\_matrix[min\_i, \_min\_j]);

if (\_subtractValue == 2)

{

bool f\_priority = false;

if (\_matrix[min\_i, 0].ToString().Contains("art"))

f\_priority = true;

for (int i = 1; i < \_matrix.GetLength(0) - \_subtractValue; i++)

{

if (\_matrix[i, 0].ToString().Contains("art"))

f\_priority = true;

else

f\_priority = false;

if ((Convert.ToDouble(\_matrix[i, \_min\_j]) > 0) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) < minDivisionRes) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) > 0))

{

minDivisionRes = (Convert.ToDouble(\_matrix[i, 1])) / (Convert.ToDouble(\_matrix[i, \_min\_j]));

min\_i = i;

}

else

{

if ((Convert.ToDouble(\_matrix[i, \_min\_j]) > 0) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) == minDivisionRes) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) > 0) && f\_priority)

{

minDivisionRes = (Convert.ToDouble(\_matrix[i, 1])) / (Convert.ToDouble(\_matrix[i, \_min\_j]));

min\_i = i;

}

}

}

}

else

{

for (int i = 1; i < \_matrix.GetLength(0) - \_subtractValue; i++)

{

if ((Convert.ToDouble(\_matrix[i, \_min\_j]) > 0) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) < minDivisionRes) && (Convert.ToDouble(\_matrix[i, 1]) / Convert.ToDouble(\_matrix[i, \_min\_j]) > 0))

{

minDivisionRes = (Convert.ToDouble(\_matrix[i, 1])) / (Convert.ToDouble(\_matrix[i, \_min\_j]));

min\_i = i;

}

}

}

return min\_i;

}

//Изменение базы

private static void Editbase(object[,] \_matrix, int \_min\_i, int \_min\_j)

{

Trace.Write("\nПеременная " + \_matrix[\_min\_i, 0]);

\_matrix[\_min\_i, 0] = \_matrix[0, \_min\_j];

Trace.WriteLine(" заменена на " + \_matrix[\_min\_i, 0]);

}

//Приведение к базису

private static void PrivedenieKBasisu(object[,] \_matrix, int \_min\_i, int \_min\_j)

{

if (Convert.ToDouble(\_matrix[\_min\_i, \_min\_j]) != 1)

{

double devisor = Convert.ToDouble(\_matrix[\_min\_i, \_min\_j]);

for (int j = 1; j < \_matrix.GetLength(1); j++)

\_matrix[\_min\_i, j] = Convert.ToDouble(\_matrix[\_min\_i, j]) / devisor;

}

}

//Получаем нули в столбце

private static void PoluchenieNullVStolbe(object[,] \_matrix, int \_min\_i, int \_min\_j)

{

for (int i = 1; i < \_matrix.GetLength(0); i++)

{

if (i != \_min\_i)

{

double tmp = Convert.ToDouble(\_matrix[i, \_min\_j]);

if (tmp != 0)

{

for (int j = 1; j < \_matrix.GetLength(1); j++)

\_matrix[i, j] = Convert.ToDouble(\_matrix[i, j]) - tmp \* Convert.ToDouble(\_matrix[\_min\_i, j]);

}

else

continue;

}

}

}

//Вывод матрицы

public static void Writing(object[,] \_matrix)

{

for (int i = 0; i < \_matrix.GetLength(0); i++)

{

Trace.WriteLine("");

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

{

if (\_matrix[i, j].ToString().Contains("E-"))

Trace.Write(String.Format("{0,20}", "0"));

else

Trace.Write(String.Format("{0,20}", \_matrix[i, j]));

}

}

Trace.WriteLine("");

}

private static int counterFectsNames = 0;

private static int counterArtsNames = 0;

private static List<object[,]> addingArtificials = new List<object[,]>();

//Получение знаков условий

private static string[] GetConditionSigns(TCT[] arry, int \_countConditions)

{

string[] conditionSigns = new string[\_countConditions];

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

{

conditionSigns[i] = arry[i].c.SelectedItem.ToString();

}

return conditionSigns;

}

//Получение коэффициентов условий

private static double[,] GetConditionsCoefficients(TCT[] arry, int \_countConditions, int \_countVariables, string[] str\_zele)

{

double[,] coefficientsMatrix = new double[\_countConditions + 1, \_countVariables + 1];

string[] tmp\_sarry1 = str\_zele[0].Split(' ');

coefficientsMatrix[0, 0] = Convert.ToDouble(tmp\_sarry1[0]);

coefficientsMatrix[0, 1] = Convert.ToDouble(tmp\_sarry1[1]);

coefficientsMatrix[0, 2] = Convert.ToDouble(tmp\_sarry1[2]);

coefficientsMatrix[0, 3] = Convert.ToDouble(str\_zele[1]);

for (int i = 1; i < arry.Length; i++)

{

string[] tmp\_sarry = arry[i].t1.Text.Split(' ');

coefficientsMatrix[i, 0] = Convert.ToDouble(tmp\_sarry[0]);

coefficientsMatrix[i, 1] = Convert.ToDouble(tmp\_sarry[1]);

coefficientsMatrix[i, 2] = Convert.ToDouble(tmp\_sarry[2]);

coefficientsMatrix[i, 3] = Convert.ToDouble(arry[i].t2.Text);

}

return coefficientsMatrix;

}

//Создание матрицы целевой функции

public static object[,] CreateTargetFunctionMatrix(TCT[] arry, int \_countConditions, int \_countVariables, string[] str\_zell)

{

double[,] conditionCoefficients = GetConditionsCoefficients(arry, \_countConditions, \_countVariables, str\_zell);

object[,] targetMatrix = new object[3, \_countVariables];

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

{

targetMatrix[0, j] = "x" + (j + 1);

targetMatrix[1, j] = conditionCoefficients[0, j];

targetMatrix[2, j] = 0;

}

return targetMatrix;

}

//Моделирование уравнений(не неравенств, т.к. сразу создаем фективные переменные) условий через матрицу

public static void CreateEquationMatrix(TCT[] arry, int \_countConditions, int \_countVariables, ref List<object[,]> \_conditionSimplexMatrixes, string[] str\_zell)

{

string[] conditionSigns = GetConditionSigns(arry, \_countConditions);

double[,] conditionCoefficients = GetConditionsCoefficients(arry, \_countConditions, \_countVariables, str\_zell); //получили коэффициенты при переменных

bool f\_DistributionSimplex = false;

object[,] matrixItem = null;

for (int k = 0; k < \_countConditions; k++)

{

int runningJ = 0;

if (conditionSigns[k] == "=")

{

matrixItem = new object[3, \_countVariables + 1];

runningJ = \_countVariables + 1;

f\_DistributionSimplex = true;

}

if (conditionSigns[k] == "=<" || conditionSigns[k] == ">=")

{

matrixItem = new object[3, \_countVariables + 2];

runningJ = \_countVariables + 2;

f\_DistributionSimplex = true;

}

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

{

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

{

if (j >= \_countVariables + 1)

{

if (i == 0)

{

string fectName = "fect" + (j + counterFectsNames); //прибавляем число элементов списка для поддержания порядка нумерации фективных переменных

counterFectsNames++;

matrixItem[i, j] = fectName;

}

if (i == 1)

{

if (conditionSigns[k] == "=<")

matrixItem[i, j] = 1;

if (conditionSigns[k] == ">=")

matrixItem[i, j] = -1;

}

if (i == 2)

matrixItem[i, j] = 0;

}

else

{

if (i == 0)

matrixItem[i, j] = "x" + (j + 1);

if (i == 1)

{

if (j == \_countVariables)

{

matrixItem[0, j] = "Value"; //имя свободного члена

matrixItem[i, j] = conditionCoefficients[k, j] \* (-1);

}

else

matrixItem[i, j] = conditionCoefficients[k, j];

}

if (i == 2)

matrixItem[i, j] = 0;

}

}

}

if (f\_DistributionSimplex)

\_conditionSimplexMatrixes.Add(matrixItem);

}

}

//Проверка необходимости добавления искусственной переменной

private static bool CheckArtificialAddition(object[,] \_conditionItem)

{

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

{

if ((\_conditionItem[0, j].ToString()).Contains("fect") && (int)\_conditionItem[1, j] == -1)

return true;

if ((\_conditionItem[0, j].ToString()).Contains("fect") && (int)\_conditionItem[1, j] == 1)

return false;

}

return true;

}

//Проверка на уже добавленность искусственной переменной

private static bool CheckArtificialExistence(object[,] \_conditionItem)

{

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

{

if ((\_conditionItem[0, j].ToString()).Contains("art"))

return true;

}

return false;

}

//Добавление искусственных переменных к элементу списка

private static void AddArtificialVariableToItem(List<object[,]> \_conditionSimplexMatrixes, object[,] \_conditionItem, int \_countVariables)

{

if (CheckArtificialAddition(\_conditionItem) && !CheckArtificialExistence(\_conditionItem))

{

object[,] copyItem = new object[3, \_conditionItem.GetLength(1) + 1];

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

{

for (int i = 0; i < \_conditionItem.GetLength(0); i++)

{

copyItem[i, j] = \_conditionItem[i, j];

}

}

copyItem[0, copyItem.GetLength(1) - 1] = "art" + (counterArtsNames + counterFectsNames + 1 + \_countVariables);

counterArtsNames++;

copyItem[1, copyItem.GetLength(1) - 1] = 1;

copyItem[2, copyItem.GetLength(1) - 1] = 0;

addingArtificials.Add(copyItem);

}

else

addingArtificials.Add(\_conditionItem);

}

public static void AddArtificialVariables(ref List<object[,]> \_conditionSimplexMatrixes, int \_countVariables)

{

for (int i = 0; i < \_conditionSimplexMatrixes.Count; i++)

AddArtificialVariableToItem(\_conditionSimplexMatrixes, \_conditionSimplexMatrixes[i], \_countVariables);

\_conditionSimplexMatrixes = addingArtificials;

}

private static List<object[,]> LookForConditionsWithArtificialVars(List<object[,]> \_conditionSimplexMatrixes)

{

List<object[,]> conditionsWithArtVars = new List<object[,]>();

for (int k = 0; k < \_conditionSimplexMatrixes.Count; k++)

{

for (int j = \_conditionSimplexMatrixes[k].GetLength(1) - 1; j >= 0; j--)

{

if ((\_conditionSimplexMatrixes[k][0, j].ToString()).Contains("art"))

{

conditionsWithArtVars.Add(\_conditionSimplexMatrixes[k]);

break;

}

}

}

return conditionsWithArtVars;

}

//Выражаем искусственные переменные

private static void ExpressArtifialVariables(ref List<object[,]> \_conditionsWithArtVars)

{

for (int k = 0; k < \_conditionsWithArtVars.Count; k++)

{

for (int j = 0; j < \_conditionsWithArtVars[k].GetLength(1); j++)

{

if (!((\_conditionsWithArtVars[k][0, j].ToString()).Contains("art")))

{

int tmpBox = Convert.ToInt32(\_conditionsWithArtVars[k][1, j]);

\_conditionsWithArtVars[k][1, j] = 0;

\_conditionsWithArtVars[k][2, j] = Convert.ToInt32(\_conditionsWithArtVars[k][2, j]) - tmpBox;

}

}

}

}

//Ищем имена фективных переменных, участвующих в образовании искусственной ф-ии

private static List<string> GetNamesFectiveVariables(List<object[,]> \_conditionsWithArtVars)

{

List<string> actedFectiveVariables = new List<string>(); //задействованные фективные переменные

for (int k = 0; k < \_conditionsWithArtVars.Count; k++)

{

for (int j = 0; j < \_conditionsWithArtVars[k].GetLength(1); j++)

{

if ((\_conditionsWithArtVars[k][0, j].ToString()).Contains("fect") && !actedFectiveVariables.Contains((\_conditionsWithArtVars[k][0, j].ToString())))

actedFectiveVariables.Add(\_conditionsWithArtVars[k][0, j].ToString());

}

}

return actedFectiveVariables;

}

//Создаем и выражаем искусственную функцию

private static object[,] ExpressArtificialFunction(List<object[,]> \_conditionsWithArtVars, int \_countVariables)

{

List<string> actedFectiveVariables = GetNamesFectiveVariables(\_conditionsWithArtVars);

object[,] artificialFunctionMatrix = new object[3, \_countVariables + actedFectiveVariables.Count + 1];

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

{

for (int k = 0; k < \_conditionsWithArtVars.Count; k++)

{

if (k == 0)

artificialFunctionMatrix[0, n] = \_conditionsWithArtVars[k][0, n];

artificialFunctionMatrix[1, n] = Convert.ToInt32(artificialFunctionMatrix[1, n]) + Convert.ToInt32(\_conditionsWithArtVars[k][2, n]);

artificialFunctionMatrix[2, n] = 0;

}

}

for (int j = \_countVariables + 1, listCounter = 0; j < artificialFunctionMatrix.GetLength(1); j++, listCounter++)

{

artificialFunctionMatrix[0, j] = actedFectiveVariables[listCounter];

artificialFunctionMatrix[1, j] = 1;

artificialFunctionMatrix[2, j] = 0;

}

return artificialFunctionMatrix;

}

//Перенесем значение (Value) в правую часть искусственной ф-ции(при этом не забываем учесть знак "-")

private static object[,] ChangeValueSide(object[,] \_artificialFunction)

{

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

{

if (\_artificialFunction[0, j].ToString() == "Value")

{

\_artificialFunction[2, j] = (-1) \* Convert.ToInt32(\_artificialFunction[1, j]);

\_artificialFunction[1, j] = 0;

}

}

return \_artificialFunction;

}

//Выражание искусственной функции (поэтапно)

public static object[,] GetArtificialFunctionMatrix(List<object[,]> \_conditionSimplexMatrixes, int \_countVariables)

{

List<object[,]> conditionsWithArtVars = LookForConditionsWithArtificialVars(\_conditionSimplexMatrixes);

ExpressArtifialVariables(ref conditionsWithArtVars);

return ChangeValueSide(ExpressArtificialFunction(conditionsWithArtVars, \_countVariables));

}

//Заполним заголовок таблицы и первый столбец (базу)

private static void FillTableTitle(ref object[,] \_simplexTable, int \_countVariables)

{

\_simplexTable[0, 0] = "Base";

\_simplexTable[0, 1] = "Value";

for (int j = 2; j < \_countVariables + 2; j++)

{

\_simplexTable[0, j] = "x" + (j - 1);

}

for (int j = \_countVariables + 2; j < \_countVariables + 2 + counterFectsNames; j++)

{

\_simplexTable[0, j] = "fect" + (j - 1);

}

for (int j = \_countVariables + 2 + counterFectsNames; j < \_simplexTable.GetLength(1); j++)

{

\_simplexTable[0, j] = "art" + (j - 1);

}

}

//Формирование столбца "Base"

private static void FillColumnBase(ref object[,] \_simplexTable)

{

for (int i = 1; i < \_simplexTable.GetLength(0) - 2; i++)

{

\_simplexTable[i, 0] = "x";

}

\_simplexTable[\_simplexTable.GetLength(0) - 2, 0] = "-z";

\_simplexTable[\_simplexTable.GetLength(0) - 1, 0] = "-w";

}

//Заполним симплекс-матрицу нулями

private static void FillSimplexMatrixByNulls(ref object[,] \_simplexTable)

{

for (int i = 1; i < \_simplexTable.GetLength(0); i++)

for (int j = 1; j < \_simplexTable.GetLength(1); j++)

\_simplexTable[i, j] = 0;

}

//Приводим все симплекс-условия

private static void SetUsualView(ref List<object[,]> \_conditionSimplexMatrixes)

{

for (int k = 0; k < \_conditionSimplexMatrixes.Count; k++)

{

for (int j = 0; j < \_conditionSimplexMatrixes[k].GetLength(1); j++)

{

if (\_conditionSimplexMatrixes[k][0, j].ToString() != "Value")

{

if (Convert.ToInt32(\_conditionSimplexMatrixes[k][1, j]) == 0)

{

int tmpVar = Convert.ToInt32(\_conditionSimplexMatrixes[k][2, j]) \* (-1);

\_conditionSimplexMatrixes[k][2, j] = 0;

\_conditionSimplexMatrixes[k][1, j] = tmpVar;

}

}

else

{

if (Convert.ToInt32(\_conditionSimplexMatrixes[k][1, j]) != 0)

{

int tmpVar = Convert.ToInt32(\_conditionSimplexMatrixes[k][1, j]) \* (-1);

\_conditionSimplexMatrixes[k][2, j] = tmpVar;

\_conditionSimplexMatrixes[k][1, j] = 0;

}

}

}

}

}

//Заносим значение через поиск по имени

private static void SearchAndSetValue(object[,] \_simplexTable, string \_varName, int \_value, int \_rowIndex)

{

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

{

if (\_simplexTable[0, j].ToString() == \_varName)

{

\_simplexTable[\_rowIndex, j] = \_value;

break;

}

}

}

//Заполняем матрицу коэффициентами условий, целевой ф-ии, искусственной ф-ии

private static void FillMatrixByCoefficients(ref object[,] \_simplexTable, object[,] \_targetFunctionMatrix, List<object[,]> \_conditionSimplexMatrixes, object[,] \_artificialFunctionMatrix)

{

\_conditionSimplexMatrixes.Add(\_targetFunctionMatrix);

\_conditionSimplexMatrixes.Add(\_artificialFunctionMatrix);

for (int i = 1, listCounter = 0; i < \_simplexTable.GetLength(0); i++, listCounter++)

{

for (int k = 0; k < \_conditionSimplexMatrixes[listCounter].GetLength(1); k++)

{

if (Convert.ToInt32(\_conditionSimplexMatrixes[listCounter][1, k]) != 0)

SearchAndSetValue(\_simplexTable, \_conditionSimplexMatrixes[listCounter][0, k].ToString(), Convert.ToInt32(\_conditionSimplexMatrixes[listCounter][1, k]), listCounter + 1);

else

SearchAndSetValue(\_simplexTable, \_conditionSimplexMatrixes[listCounter][0, k].ToString(), Convert.ToInt32(\_conditionSimplexMatrixes[listCounter][2, k]), listCounter + 1);

}

}

\_conditionSimplexMatrixes.RemoveRange(\_conditionSimplexMatrixes.Count - 2, 2);

}

//Формируем таблицу для симплекс-метода

public static object[,] FormTableForSimplexMethod(object[,] \_targetFunctionMatrix, List<object[,]> \_conditionSimplexMatrixes, object[,] \_artificialFunctionMatrix, int \_countSimplexConditions, int \_countVariables)

{

object[,] simplexTable = new object[\_countSimplexConditions + 3, \_countVariables + counterFectsNames + counterArtsNames + 2];

FillTableTitle(ref simplexTable, \_countVariables);

FillColumnBase(ref simplexTable);

FillSimplexMatrixByNulls(ref simplexTable);

SetUsualView(ref \_conditionSimplexMatrixes);

FillMatrixByCoefficients(ref simplexTable, \_targetFunctionMatrix, \_conditionSimplexMatrixes, \_artificialFunctionMatrix);

return simplexTable;

}

//Забор значений из столбца Base полученного решения

private static object[,] PolushenieBase(object[,] \_simplexTable)

{

object[,] solutionValues = new object[2, \_simplexTable.GetLength(0) - 1];

for (int i = 1, counter = 0; i < \_simplexTable.GetLength(0); i++, counter++)

{

solutionValues[0, counter] = \_simplexTable[i, 0];

solutionValues[1, counter] = \_simplexTable[i, 1];

}

return solutionValues;

}

//Проверка на завершенность решения симплекс-алгоритмом

private static string SimplexSolved(object[,] \_solutionValues, object[,] \_targetFunction)

{

object[,] flagArray = new object[2, \_targetFunction.GetLength(1)];

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

{

flagArray[0, j] = \_targetFunction[0, j];

flagArray[1, j] = 0;

}

for (int i = 0; i < flagArray.GetLength(1); i++)

{

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

{

if (flagArray[0, i].ToString() == \_solutionValues[0, j].ToString())

flagArray[1, i] = 1;

}

}

string unknownVariable = null;

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

if (Convert.ToInt32(flagArray[1, j]) == 0)

unknownVariable = flagArray[0, j].ToString();

return unknownVariable;

}

//Установка значения

private static void UstanovkaZnach(object[,] \_findingMatrix, int \_j, object[,] \_solutionValues)

{

for (int k = 0; k < \_solutionValues.GetLength(1); k++)

{

if (\_findingMatrix[0, \_j].ToString() == \_solutionValues[0, k].ToString())

{

\_findingMatrix[1, \_j] = \_solutionValues[1, k];

}

}

}

//Формирование матрицы для расчета неизвестного значения

private static object[,] SizdanieArry(object[,] \_targetFunction, object[,] \_solutionValues)

{

object[,] findingMatrix = new object[3, \_targetFunction.GetLength(1) + 1];

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

{

if (j == findingMatrix.GetLength(1) - 1)

{

findingMatrix[0, j] = "z";

findingMatrix[1, j] = (-1) \* Convert.ToDouble(\_solutionValues[1, \_solutionValues.GetLength(1) - 1]);

findingMatrix[2, j] = 0;

}

else

{

findingMatrix[0, j] = \_targetFunction[0, j];

UstanovkaZnach(findingMatrix, j, \_solutionValues);

findingMatrix[2, j] = \_targetFunction[1, j];

}

}

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

{

if (findingMatrix[1, j] == null)

findingMatrix[1, j] = "?";

}

return findingMatrix;

}

//Поиск номера столбца с неизвестной переменной в матрице поиска

private static int GetUnknownVarCol(object[,] \_findingMatrix, string \_unknownVariable)

{

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

if (\_findingMatrix[0, j].ToString() == \_unknownVariable)

return j;

return -1;

}

//Поиск неизвестного значения

private static void PoiskNeizvestnZnach(object[,] \_findingMatrix, string \_unknownVariable)

{

double value = Convert.ToDouble(\_findingMatrix[1, \_findingMatrix.GetLength(1) - 1]);

int k = GetUnknownVarCol(\_findingMatrix, \_unknownVariable);

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

{

if (k != j)

{

value -= (Convert.ToDouble(\_findingMatrix[1, j]) \* Convert.ToDouble(\_findingMatrix[2, j]));

}

else

continue;

}

value = value / Convert.ToDouble(\_findingMatrix[2, k]);

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

{

if (\_findingMatrix[1, j].ToString() == "?")

\_findingMatrix[1, j] = value;

}

}

//расчет недостающего значения и запись его в матрицу

private static object[,] RaschetNeResheniya(object[,] \_simplexTable, object[,] \_targetFunction)

{

object[,] solutionValues = PolushenieBase(\_simplexTable);

string unknownVariable = SimplexSolved(solutionValues, \_targetFunction);

object[,] findingMatrix = SizdanieArry(\_targetFunction, solutionValues);

if (SimplexSolved(solutionValues, \_targetFunction) != null)

PoiskNeizvestnZnach(findingMatrix, unknownVariable);

Trace.WriteLine("Значения полученных переменных:");

for (int i = 0; i < solutionValues.GetLength(0); i++)

{

Trace.WriteLine("");

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

{

Trace.Write(String.Format("{0,16}", solutionValues[i, j]));

}

}

return findingMatrix;

}

//Получить решение

public static object[,] PoluchenieResheniya(object[,] \_simplexTable, object[,] \_targetFunction, ref double minVertex, string zele, Label lab)

{

object[,] solutionMatrix = new object[2, \_targetFunction.GetLength(1)];

object[,] findingMatrix = RaschetNeResheniya(\_simplexTable, \_targetFunction);

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

{

solutionMatrix[0, j] = findingMatrix[0, j];

solutionMatrix[1, j] = findingMatrix[1, j];

if (solutionMatrix[1, j].ToString().Contains("E-"))

solutionMatrix[1, j] = 0;

}

Trace.WriteLine("\nОтветы в матричном представлении:\n");

for (int i = 0; i < findingMatrix.GetLength(0); i++)

{

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

{

Trace.Write(String.Format("{0,16}", findingMatrix[i, j]));

}

Trace.WriteLine("");

}

minVertex = Convert.ToDouble(findingMatrix[1, findingMatrix.GetLength(1) - 1]);

lab.Text += "\n" + String.Format("{0,5}", "x1") + " = " + String.Format("{0,5}", (findingMatrix[1, 0].ToString().IndexOf('E') > 0) ? "0" : findingMatrix[1, 0])

+ "\n" + String.Format("{0,5}", "x2") + " = " + String.Format("{0,5}", (findingMatrix[1, 1].ToString().IndexOf('E') > 0) ? "0" : findingMatrix[1, 1])

+ "\n" + String.Format("{0,5}", "x3") + " = " + String.Format("{0,5}", (findingMatrix[1, 2].ToString().IndexOf('E') > 0) ? "0" : findingMatrix[1, 2])

+ "\n" + String.Format("{0,5}", "Z(x)") + " = " + String.Format("{0,5}", minVertex);

return solutionMatrix;

}

}