Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Факультет информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

**Курсовая работа по дисциплине  
 Алгоритмы и вычислительные методы оптимизации**

**Симплекс-метод**

Вариант 1

Выполнил:

студент гр. ИП-712 Алексеев С.В./\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

ФИО студента

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Проверил

Галкина М.Ю./\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

ФИО преподавателя

«06» мая 2020 г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2020 г.

Оглавление

[Задание 2](#_Toc39684293)

[№1 Перейти к канонической форме 3](#_Toc39684294)

[№2 Написать программу 4](#_Toc39684295)

[Form1.cs 4](#_Toc39684296)

[Drob.cs 17](#_Toc39684297)

[Скриншот 20](#_Toc39684298)

[№3 Решить графически 21](#_Toc39684299)

[№4 Составить двойственную задачу 22](#_Toc39684300)

# Задание

Задание на курсовую работу по дисциплине

«Алгоритмы и вычислительные методы оптимизации»

для студентов гр.ИП-711-715

Решение задачи линейного программирования, теория двойственности

1. Перейти к канонической форме записи задачи линейного программирования.

2. Написать программу, решающую задачу линейного программирования в

канонической форме (с выводом всех промежуточных таблиц, все

вычисления должны выполняться в простых дробях, для этого реализовать

класс простых дробей) одним из перечисленных способов (в соответствии с

последним столбцом приведенной ниже таблицы):

− симплекс-методом, используя в качестве начальной угловой точки

опорное решение с указанными в задании базисными переменными,

найденное методом Жордана-Гаусса (1);

3. Решить исходную задачу графически и отметить на чертеже точки,

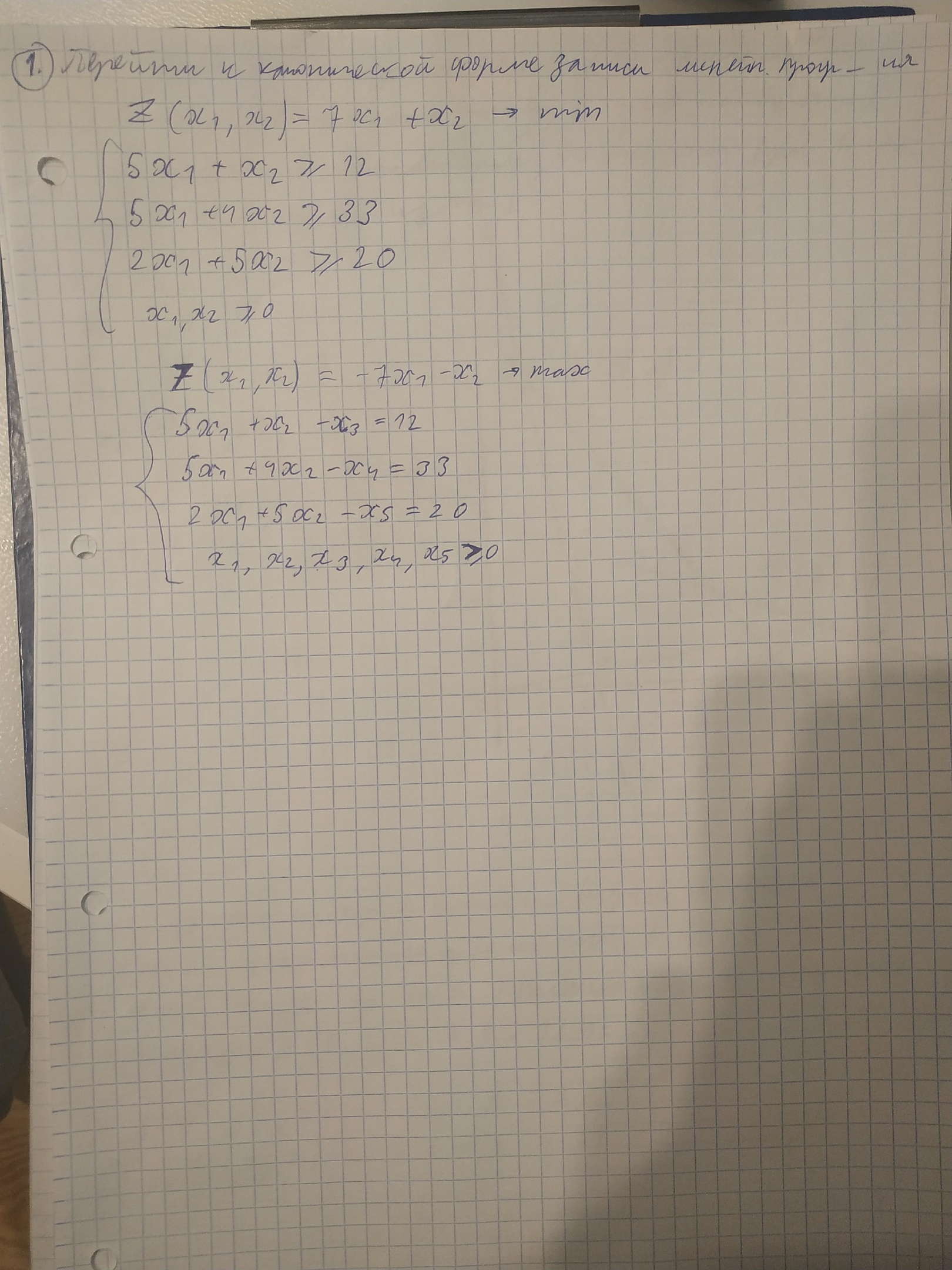
соответствующие симплексным таблицам, полученным при выполнении

программы из п.2.

4. Составить двойственную задачу к исходной и найти ее решение на основании

теоремы равновесия.

# №1 Перейти к канонической форме



# №2 Написать программу

# Form1.cs

using CW3;

using System; //Получается, что число базисных переменных всегда равно числу ограничений(числу строк)

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

//implementing the Jordan-Gauss method + rectangle method (метод Жордана-Гауса и метод прямоугольников)

/\*В разных источниках разные определения канонического вида ЗЛП. Я принял, что это когда функция исследуется на максимум и члены столбца свободных членов не отрицательны.

\* Изначально значение функции принимается нулём(правое нижнее значение матрицы(слева от Z)).

\* https://habr.com/ru/post/474286/

\* Алгоритм перехода от произвольной задачи ЛП к канонической форме:

Неравенства с отрицательными b(своб.членами) умножаем на (-1).

\* 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.

Замечание: Хотя мы и берем минимальное отрицательное число в задаче на максимум, этот коэффициент показывает направление роста функционала,

т.к. строка функционала в симплекс-таблице взята со знаком “-”. Аналогичная ситуация с минимизацией.\*/

namespace CW3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

masOld = new Drob[n1, n2];//матрица не рабочая

//инициализирую функцию

fun = new Drob[n2];//инициализирую массив коэффициентов исследуемой функции.

fun[0] = new Drob(-7, 1); fun[1] = new Drob(-1, 1); fun[2] = new Drob(0, 1); fun[3] = new Drob(0, 1); fun[4] = new Drob(0, 1); fun[5] = new Drob(0, 1);

for (int i = 0; i < n2; i++)

{

if (MAX)

fun[i] = fun[i].mul(fun[i], new Drob(-1, 1));// это всё часть какого-то плана))) но я не в курсе. Короче при максимизации мы исключаем отрицательные

//члены из Z. Поэтому надо, чтобы они там появились обязательно... Но в моём случае я поменял здесь знаки ещё при приведении к канонической форме...

}

//инициализирую матрицу коэффициентов системы в канонической форме. Свободные члены справа:

mas = new Drob[n1 + 1, n2];//ещё одну строку добавляю для значений функции

mas[0, 0] = new Drob(5, 1); mas[0, 1] = new Drob(1, 1); mas[0, 2] = new Drob(-1, 1); mas[0, 3] = new Drob(0, 1); mas[0, 4] = new Drob(0, 1); mas[0, 5] = new Drob(12, 1);

mas[1, 0] = new Drob(5, 1); mas[1, 1] = new Drob(4, 1); mas[1, 2] = new Drob(0, 1); mas[1, 3] = new Drob(-1, 1); mas[1, 4] = new Drob(0, 1); mas[1, 5] = new Drob(33, 1);

mas[2, 0] = new Drob(2, 1); mas[2, 1] = new Drob(5, 1); mas[2, 2] = new Drob(0, 1); mas[2, 3] = new Drob(0, 1); mas[2, 4] = new Drob(-1, 1); mas[2, 5] = new Drob(20, 1);

for (int i = 0; i < n2; i++)

{

mas[n1, i] = fun[i];//просто направил ссылки на функцию таким образом. Для красоты.

}

for (int i = 0; i < n1; i++)

{

basis[i] = i;//заполняю массив базисных переменных(по уcловию задачи: x1, x2, x3)

}

}

public int rowsIncrement = 0;//для прорисовки промежуточных решений(чтобы новые матрицы рисовались ниже)

public const int n1 = 3;//число строк(число уравнений системы)

public const int n2 = 6;//число столбцов(коэффициенты при переменных + 1 свободный член)(5+1)

public const int n3 = 2;//число переменных в матрице, не приведённой к каноническому виду, равное числу коэффициентов в исследуемой функции

public const int n4 = 11;//число всех названий столбцов

public Drob[] fun;// массив коэффициентов исследуемой функции

//С этого момента не буду заморачиваться с красивым оформлением, взаимодействием с пользователем в рантайме

public int[] basis = new int[n1];// базисные переменные заданные условием задачи. Их может быть ровно столько, сколько уравнений в системе

//массив для вообще всех переменных наверное не нужен....

public Drob[,] mas;//массив для хранения матрицы в коде

public int resolvRow = 0;//переменная, указывающая текущую резольвирующую строку

public int resolvColumn = 0;//переменная, указывающая текущий резольвирующий столбец

public string[] elems;//список имён вершин для подписи в матрице

public Drob d = new Drob(1, 1);//объект для вызова методов. Криво конечно, но пока так...

bool MAX = true; //Если исследую функцию на максимум, то домножаю коэффициенты функции на - 1...

int maxZindex = 0;//здесь хранятся результаты действия метода findMinMaxZ()

int minZindex = 0;//здесь хранятся результаты действия метода findMinMaxZ()

double maxZvalue = 0;//здесь хранятся результаты действия метода findMinMaxZ()

double minZvalue = 0;//здесь хранятся результаты действия метода findMinMaxZ()

public double[,] masBumaga2;//массив для запоминания вводимых матриц//остаток от лабы. не нужен

public bool load;// остаток от лабы. не нужен

public bool flagSolved = false;//остаток от лабы. не нужен

public Drob[,] masOld;// остаток от лабы. не нужен

private void printMatrixMasBumaga(object sender, EventArgs e) // <--- КНОПКА "Вывести матрицу"

{

dataGridView1.RowCount = n1 \* 300;//просто делаю всю полосу подлиннее с запасом

dataGridView1.ColumnCount = n4;

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;

initilaizeHeaders();

fillTable();// в первый раз нарисовал матричку

rctnApplyFirstBasis();//этот метод для подведения к первому заданному базису

rctnApplySimplex();//это нужно скорее в этой функции

}

private void initilaizeHeaders()//------------------------------------расписал заголовки вверху таблицы

{

elems = new string[n4];

for (int i = 0; i < n2 - 1; i++)

{

elems[i] = "x" + Convert.ToString(i);//создал строки для обозначения базисных переменных

}

elems[n2 - 1] = "своб.Чл.";

elems[n2] = "б.п.\nX";

elems[n2 + 1] = "резCтр.";

elems[n2 + 2] = "резCтолб.";

elems[n2 + 3] = "максАбсZ";

elems[n2 + 4] = "своб.Чл./резCтолб";

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

{

dataGridView1.Columns[i].HeaderCell.Value = elems[i];//расписал все заголовки в dataGridView1

}

}

private void fillTable(int row, int index)//----заполнил dataGridView1 содержимым mas, и остальными обозначениями. Этот вариант filltable вызывается

{//когда известны резольвирующие строка и столбец

for (int i = 0; i < n1 + 1; i++)

{

for (int j = 0; j < n2; j++)

{

dataGridView1.Rows[rowsIncrement + i].Cells[j].Value = mas[i, j].toStr();//ввёл значения переменных и свободных членов

}

}

for (int i = 0; i < n1; i++)

{

dataGridView1.Rows[rowsIncrement + i].Cells[n2].Value = elems[basis[i]]; //ввёл значения базисных перемнных(подписал столбцы б.п.)

}

dataGridView1.Rows[rowsIncrement - n1 - 2].Cells[n2 + 1].Value = row; // ввёл номера резольвирующих строк... но ведь потом они будут выбираться по-другому... пока НЕ ПОНЯТНО

dataGridView1.Rows[rowsIncrement - n1 - 2].Cells[n2 + 2].Value = index;// ввёл номера резольвирующих столбцов... они ведь указаны в basis...

dataGridView1.Rows[rowsIncrement + n1].Cells[n2].Value = "Z";//подписал строку функции

//запрещает сортировать содержимое столбцов кликом по хедеру, а также минимизирует длину ячеек:(пока отключил, а то иногда всё не выводится)

//dataGridView1.Columns.Cast<DataGridViewColumn>().ToList().ForEach(f => f.SortMode = DataGridViewColumnSortMode.NotSortable);

rowsIncrement += (n1 + 2);//для прорисовки матрицы

}

private void fillTable()//---------------------------------------------заполнил dataGridView1 содержимым mas, и названиями базисных переменных

{

for (int i = 0; i < n1 + 1; i++)

{

for (int j = 0; j < n2; j++)

{

dataGridView1.Rows[rowsIncrement + i].Cells[j].Value = mas[i, j].toStr();//ввёл значения переменных и свободных членов

}

}

for (int i = 0; i < n1; i++)

{

dataGridView1.Rows[rowsIncrement + i].Cells[n2].Value = elems[basis[i]]; //ввёл значения базисных перемнных(подписал столбцы б.п.)

}

dataGridView1.Rows[rowsIncrement + n1].Cells[n2].Value = "Z";//подписал строку функции

rowsIncrement += (n1 + 2);//для прорисовки матрицы(здесь +1 для создания одной пустой строки и ещё +1 для строки значений функции). При этом индексы рабочего массива сохраняются

}

//-------------теперь нужно пройтись прямоугольником по матричке, чтобы во-первых получить базис в соответствии с условием задачи(0,1,2), а во-вторых

//---методу прямоугольника надо знать и резольв. строку и резольв. столбец? Нет. Столбец нормально

public void rctn(int row/\*номер строки, в которой базисная переменная\*/, int index /\*индекс базисной переменной\*/)// <--метод прямоугольника. У меня есть массив basis, обозначающий индексы переменных, которые сейчас считаются базисными.

{ //------------------------

//в начале у меня есть index - индекс текущей базисной переменной для одного полного прохода прямоугольником по матрице. А также есть row - номер строки,

//в которой этот индекс указан.

//т.о. выбрал строку резольвирующую

//Индексы чисел массива basis - номера строк, а сами числа - индексы базисных переменных

// Видимо не важно в какой строке будет единица, важно, чтобы единицы просто были в разных строках

for (int i = 0; i < n2; i++)//n2(сейчас 6) раз надо пройтись по столбцам. Резольвирующий столбец можно не исключать из алгоритма, т.к. всё равно получаются нули. А резольвирующую строку я всё равно пропускаю.

{

for (int j = 0; j < n1 + 1; j++)//n1 раз надо пройтись по строкам, избегая резольвирующую

{

if (!(j == row || i == index))// если строка не резольвирующая, и столбец не резольвирующий то делаем прямоугольник

{

Drob multiaa; Drob divibb; Drob subcc = new Drob(1, 1);

multiaa = d.mul(mas[row, i], mas[j, index]);

divibb = d.div(multiaa, mas[row, index]);

subcc = d.sub(mas[j, i], divibb);// проблема в том, что прямоугольником нужно проходиться до того момента, когда в резольвирующем столбце появились нули.(проблема решена)

//тестирую этапы метода прямоугольника:

textBox4.Text += ("multiaa = " + multiaa.toStr() + ", divibb = " + divibb.toStr() + ", subcc = " + subcc.toStr() + " END \0");

mas[j, i] = (Drob)subcc.Clone();//не очень удачно конечно работаю с памятью... Не хватает базовых знаний сишарпа...

}

}

}//только сейчас можно занулить остальные элементы резольвирующего столбца:

for (int i = 0; i < n1 + 1; i++)

{

if (!(i == row)) mas[i, index] = new Drob(0, 0);

}

Drob temp = (Drob)mas[row, index].Clone();//склонировал резольвирующий элемент

//теперь элементы резольвирующей строки можно(нужно) разделить на mas[row, index](резольвирующий элемент):

for (int i = 0; i < n2; i++)

{

mas[row, i] = d.div(mas[row, i], temp);

}

}

public void rctnApplyFirstBasis()//применяю метод прямоугольника для каждой базисной переменной отдельно. Этот метод приводит к начальному(заданному в задаче) базису.

//Дальше Нужно написать другой метод для нахождения резольвирующих элементов уже по большим/меньшим элементам

{//здесь нужно принять решение, какие строки и столбцы будут резольвирующими. Эта информация есть в basis.

//basis[0] - индекс первой базисной переменной. Ею называем первую строку симплекс-таблицы. При этом при вызове прямоугольника 0 будет резольвирующей

//строкой, индекс будет резольвирующим столбцом.

//basis[1] - индекс второй базисной переменной. Ею называем вторую строку и т.д.

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

{

rctn(i, basis[i]);

fillTable(i, basis[i]);

}//Привели систему к единичной матрице методом жордановских преобразований. Теперь в качестве базисных переменных нужно принять

}

public void findMinMaxZ()//нахожу макс, мин элементы в строке Z и их индексы:

{

double[] arr = new double[n2 - 1];// свободные члены не рассматриваем, поэтому n2-1

for (int i = 0; i < n2 - 1; i++)

{//раз уж у меня нет нормальной функции для сравнения дробей и хочется использовать встроенный функционал поиска

//максимального числа по модулю, то поступим проще: преобразуем дроби строки в массив double

if (mas[n1, i].denominator != 0)

arr[i] = (double)mas[n1, i].numerator / (double)mas[n1, i].denominator;//делю числитель каждой дроби на знаменатель

else arr[i] = 0.0;

Console.WriteLine("arrZ[" + i + "] = " + arr[i] + ", ");

}

//тестирую поиск минимальных, максимальных элементов и их индексов:

maxZvalue = arr.Max();

minZvalue = arr.Min();

maxZindex = arr.ToList().IndexOf(maxZvalue);

minZindex = arr.ToList().IndexOf(minZvalue);

Console.WriteLine("maxZvalue = " + maxZvalue + ", maxZindex = " + maxZindex + "; minZvalue = " + minZvalue + ", minZindex = " + minZindex);

}

public void rctnApplySimplex()

{

if (MAX)//если функция на максимум, то исключамем отрицательные

{/\*Для перехода к новому опорному решению в Z-строке среди отрицательных коэффициентов ( 1, , ) j c j r n  − = + выбирают наибольший по абсолютной

величине, пусть это будет -cj0 . Тем самым выбрана свободная переменная xj0, которая будет вводиться в базис. Столбец с заголовком xj0 называется разрешающим.\*/

int it = 30;

while (it-->0)

{

findMinMaxZ();//нашёл минимумы, максимумы

if (minZvalue < 0.0)//Если минимальное найденное в Z значение меньше нуля,то план не оптимален, идём по шагам симплекс метода:

//делим свободные члены на столбец minZindex(за исключением строки функции) для нахождения min отношения

{

double[] arrFree = new double[n1];

double[] arrZ = new double[n1];

double[] arrDivided = new double[n1];

for (int i = 0; i < n1; i++)

{

if (mas[i, minZindex].denominator != 0 )//только положительные элементы разрешающего столбца

{

if(mas[i, minZindex].numerator < 0)

{

arrZ[i] = double.NaN;

}else

arrZ[i] = (double)mas[i, minZindex].numerator / (double)mas[i, minZindex].denominator;//получил массив со значениями double столбца minZindex

}

else arrZ[i] = double.NaN;

Console.WriteLine("arrZ[" + i + "] = " + arrZ[i] + ", ");

if (mas[i, n2 - 1].denominator != 0)

{

arrFree[i] = (double)mas[i, n2 - 1].numerator / (double)mas[i, n2 - 1].denominator;//получил массив со значениями double столбца свободных членов

}

else arrFree[i] = 0.0;

Console.WriteLine("arrFree[" + i + "] = " + arrFree[i] + ", ");

if (arrZ[i] != 0 && !double.IsNaN(arrZ[i]))

arrDivided[i] = arrFree[i] / arrZ[i];//находим т.н. симплексные отношения. при этом разрешающий столбец - только положительные элементы.

else arrDivided[i] = double.NaN;

Console.WriteLine("arrDivided[" + i + "] = " + arrDivided[i] + ", ");

dataGridView1.Rows[rowsIncrement + i - n1 - 2].Cells[n2 + 4].Value = arrDivided[i];//это будет красивее смотреться у матрицы выше, поэтому дописал -n1 -2

}

int minRowRelationIndex = 0;//сюда назначаем строку с минимальным отношением столбца свободных членов на столбец minZindex. Она станет резольвирующей

double minArrDividedvalue = arrDivided.Min();

minRowRelationIndex = arrDivided.ToList().IndexOf(minArrDividedvalue);//теперь у меня есть индексы резольвирующих строки и столбца. Надо это проверить.

for (int i = 0; i < n1; i++)// оказывается сишарп считает, что его NaN - это таки число и его можно сравнивать с другими и оно оказывается наименьшим...

{//поэтому надо таки найти нужное минимальное(положительное) число каким-то образом

if (double.IsNaN(minArrDividedvalue)) arrDivided[minRowRelationIndex] = double.PositiveInfinity;//если NaN, то превращаем в бесконечность

else break;//иначе прерываем цикл

minArrDividedvalue = arrDivided.Min();//берём следующее минимальное

minRowRelationIndex = arrDivided.ToList().IndexOf(minArrDividedvalue);//находим его индекс. и так n1 раз

}

Console.WriteLine("minArrDividedvalue = " + minArrDividedvalue);

basis[minRowRelationIndex] = minZindex;//загнал эти индексы куда и планировал...

dataGridView1.Rows[rowsIncrement - n1 - 2].Cells[n2 + 1].Value = minRowRelationIndex;//вывел значения резольв строки и столбца в матрицу

dataGridView1.Rows[rowsIncrement - n1 - 2].Cells[n2 + 2].Value = minZindex;//выше. Так лучше смотрится, особенно при прописанном отношении arrFree[i] / arrZ[i]

//походу время запускать прямоугольник))

rctn(minRowRelationIndex, minZindex);

fillTable(minRowRelationIndex, minZindex);

}

else

{

textBox2.Text = "План оптимальный";

break;

}

}

}

}

private void button4\_Click(object sender, EventArgs e)// "Solve"

{

}

private void button1\_Click(object sender, EventArgs e)// "Build a table"

{

}

private void button2\_Click(object sender, EventArgs e) // "Load"

{

}

private void button3\_Click(object sender, EventArgs e) // "test"

{

Drob db = new Drob(0, 0);

Drob d1 = new Drob(-94, 3);

Drob d2 = new Drob(1, 1);

Drob d3 = new Drob(28, 3);

Drob d4 = new Drob(-1, 2);

Drob dRes = new Drob(0, 0);

d1 = db.mul(d1, d2);

textBox1.AppendText(d1.toStr() + "\n");

d3 = db.mul(d3, d4);

textBox2.AppendText(d3.toStr() + "\n");

d1 = db.sub(d1, d3);

textBox3.AppendText(d1.toStr() + "\n");

//textBox1.Text = dRes.toStr();

// db.test();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

}

}

# Drob.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CW3

{

public class Drob : ICloneable

{

public int numerator;//chislitel

public int denominator;//znamenatel

//public Drob()

//{// it would be difficult to implement some actions like addition for example. So such default constructor is undesirable.

// this.numerator = 0;

// this.denominator = 1;

//}

public Drob(int numerator, int denominator)

{

if (numerator < 0 && denominator < 0)//сокращаю минусы в числителе и знаменталел

{

numerator \*= -1;

denominator \*= -1;

}

if(denominator < 0)//переношу минус из знаменателя в числитель

{

numerator \*= -1;

denominator \*= -1;

}

if (denominator != 0)

{

int t = GCD(numerator, denominator);

this.numerator = numerator / t;

this.denominator = denominator / t;

}

else

{

this.numerator = numerator;

this.denominator = denominator;

}

if (numerator == 0) denominator = 0;

if (denominator == 0) numerator = 0;

}

public Drob mul(Drob a, Drob b)//this fun returns a result of multiplication of сommon fractions a & b

{

return new Drob(a.numerator \* b.numerator, a.denominator \* b.denominator);

}

public void printDrob()//in console

{

Console.WriteLine(numerator + "/" + denominator);

}

public void test()

{

Drob d1 = new Drob(1, 2);

Drob d2 = new Drob(1, 3);

Drob d3;

Drob d4;

d3 = mul(d1, d2);

d3.printDrob();

d4 = add(d1, d2);

d4.printDrob();

}

public String toStr()

{

if (denominator == 0 && numerator != 0) return Double.PositiveInfinity.ToString();

else

if (numerator == 0) return "0";

else

if (denominator == 1) return (numerator.ToString());

else

return (numerator.ToString() + "/" + denominator.ToString());

}

public Drob div(Drob a, Drob b)//divides a by b

{

//return new Drob(a.numerator \* b.denominator, a.denominator \* b.numerator);

return mul(a, new Drob(b.denominator, b.numerator));

}

public Drob add(Drob a, Drob b)

{

if (a.denominator == 0 || b.denominator == 0)

{

if (a.denominator != 0)

{

return new Drob(a.numerator, a.denominator);

}

else

return new Drob(b.numerator, b.denominator);

}

int multiplierA = 1;

int multiplierB = 1;

int newDenominator = 1;

if ((a.denominator != b.denominator))//if denominators aren't equal, we need to find Least(lowest) common multiple (наимееьшее общее кратное)

{

if (a.denominator != 0 && b.denominator != 0)

{

newDenominator = LCM(a.denominator, b.denominator);

multiplierA = newDenominator / a.denominator;//нашёл во сколько раз надо увеличить числитель a

multiplierB = newDenominator / b.denominator;//нашёл во сколько раз надо увеличить числитель b

}

}

else newDenominator = a.denominator;

return new Drob(a.numerator \* multiplierA + b.numerator \* multiplierB, newDenominator);

}

public Drob sub(Drob a, Drob b)

{

if (a.denominator == 0 || b.denominator == 0)

{

if (a.denominator != 0)

{

return new Drob(a.numerator, a.denominator);

}

else

return new Drob(-b.numerator, b.denominator);

}

int multiplierA = 1;

int multiplierB = 1;

int newDenominator = 1;

if ((a.denominator != b.denominator))//if denominators aren't equal, we need to find Least(lowest) common multiple (наимееьшее общее кратное)

{

if (a.denominator != 0 && b.denominator != 0)

{

newDenominator = LCM(a.denominator, b.denominator);

multiplierA = newDenominator / a.denominator;//нашёл во сколько раз надо увеличить числитель a

multiplierB = newDenominator / b.denominator;//нашёл во сколько раз надо увеличить числитель b

}

}

else newDenominator = a.denominator;

return new Drob(a.numerator \* multiplierA - b.numerator \* multiplierB, newDenominator);

}

// Use Euclid's algorithm to calculate the greatest common divisor (GCD) of two numbers

private int GCD(int a, int b)

{

a = Math.Abs(a);

b = Math.Abs(b);

// Pull out remainders

for (; ; )

{

int remainder = a % b;

if (remainder == 0) return b;

a = b;

b = remainder;

};

}

// Return the least common multiple (LCM) of two numbers

private int LCM(int a, int b)

{

return a \* b / GCD(a, b);

}

public object Clone()

{

// return new Drob(numerator, denominator) { numerator = this.numerator, denominator = this.denominator };

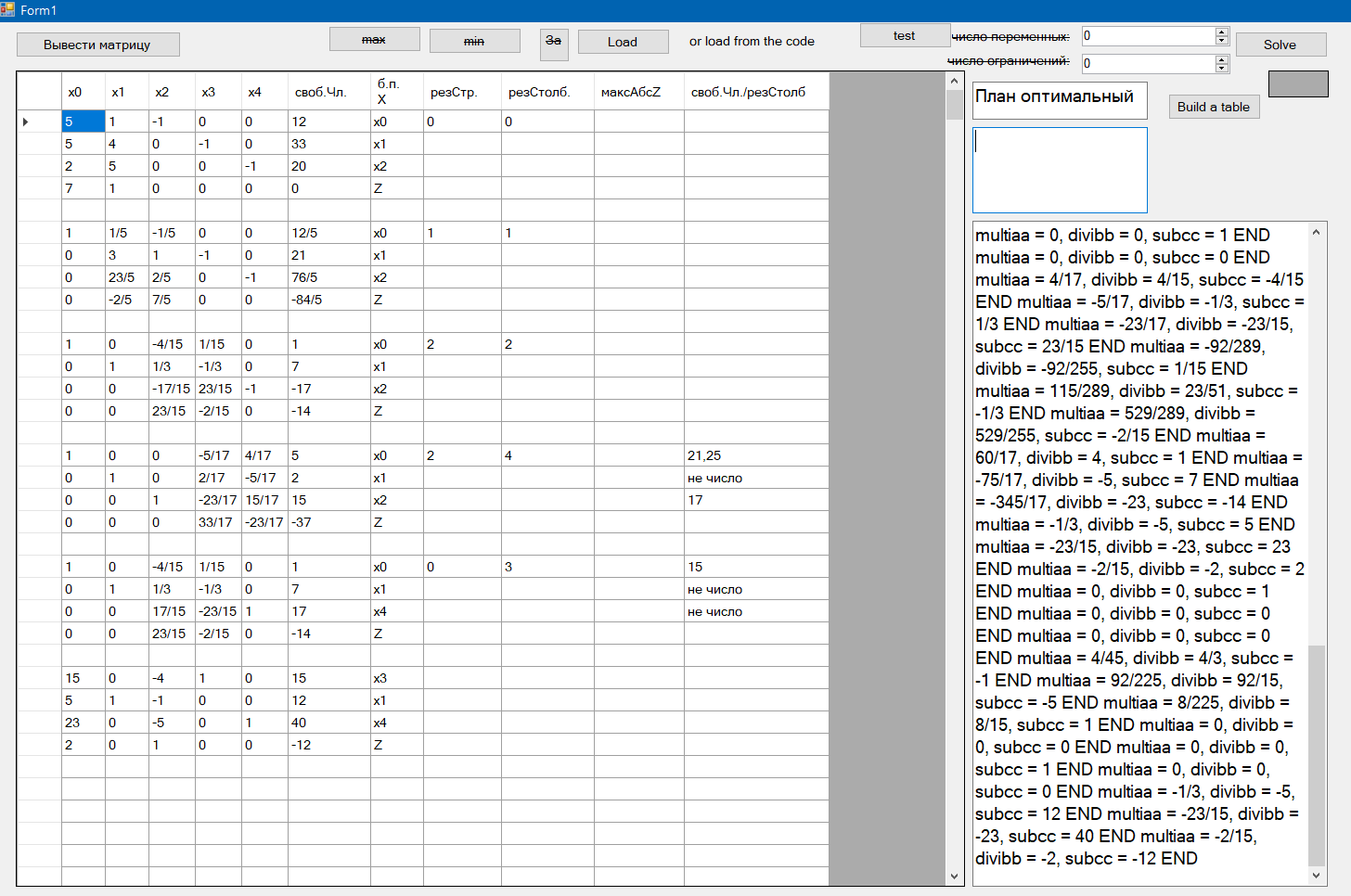
return this.MemberwiseClone();

}

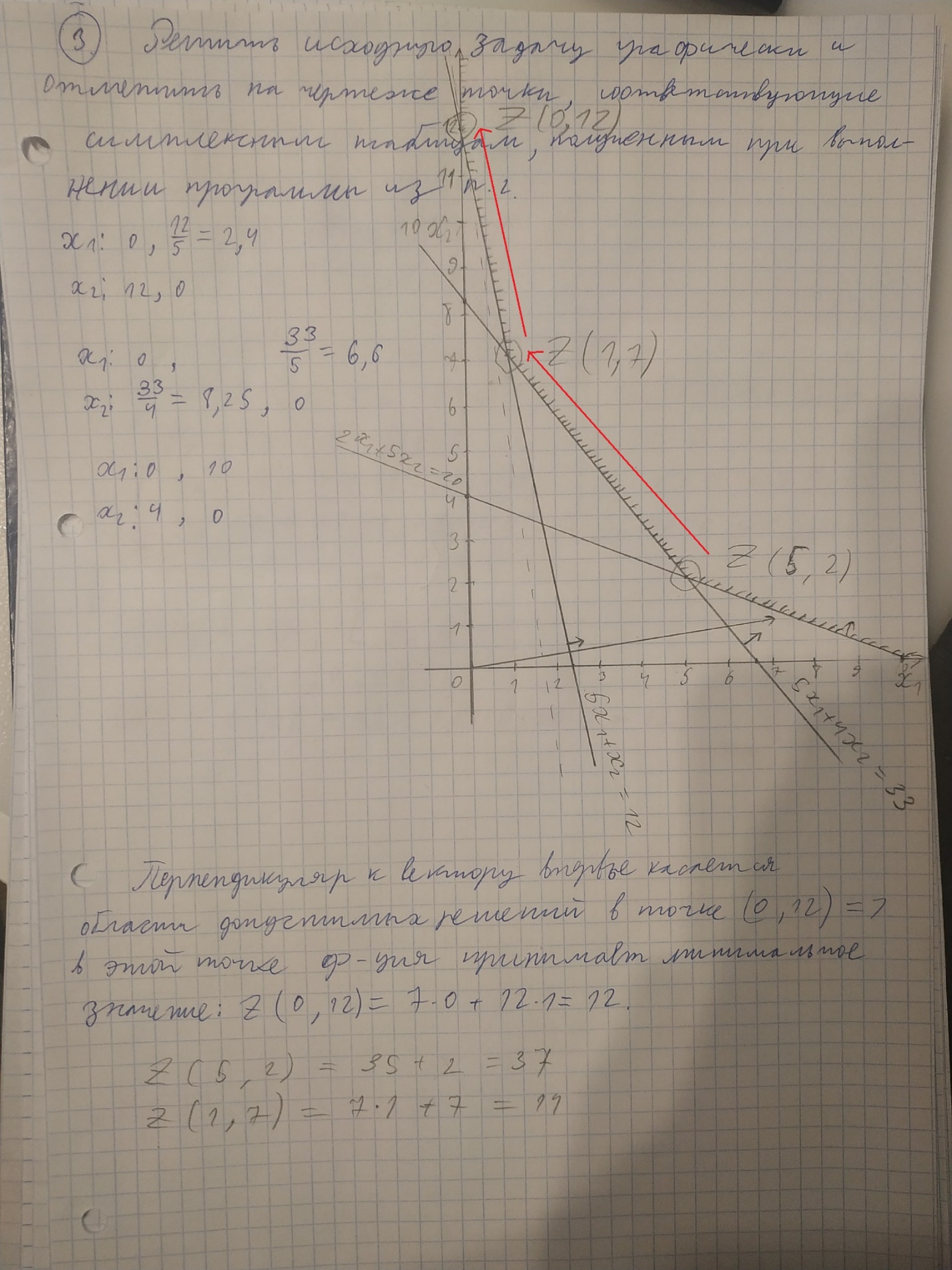
}

}

# Скриншот



# №3 Решить графически



# №4 Составить двойственную задачу

