**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

**ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №1

**на тему:** “*Розв’язування задач лінійного програмування симплекс-методом та графічно”*

# з дисципліни: *“*Дослідження операцій*”*

**Лектор:**

доц. кафедри ПЗ

Журавчак Л. М.

**Виконав:**

студент групи ПЗ-33

Чернявський М.Ю.

**Прийняв:**

доц. кафедри ПЗ

Івасько Н. М.

Львів – 2024

**Тема:** Розв’язування задач лінійного програмування симплекс-методом та графічно.

**Мета:** Ознайомитись на практиці із основними поняттями теорії лінійного програмування (ЛП), набути навичок практичного розв’язання задач ЛП табличним симплекс-методом (СМ) та його програмною реалізацією, навчитись розв’язувати задачі ЛП із двома змінними графічним методом.

**Теоретичні відомості**

Під задачею лінійного програмування (ЗЛП) в загальному розуміють задачу знаходження мінімуму (максимуму) лінійної функції від n змінних на множині розв’язків системи лінійних нерівностей або лінійних рівнянь.

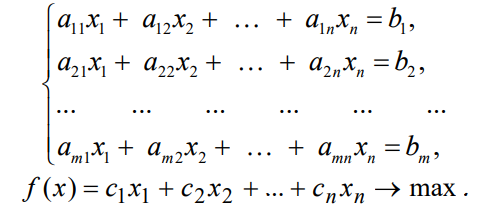
Математичну модель загальної задачі лінійного програмування (ЛП) можнаподати в такому вигляді: знайти такі числові значення змінних x1 >= 0, x2 >= 0, …, xn >= 0, в яких лінійна функція набуває екстремуму (максимуму або мінімуму) і які задовольняють систему лінійних обмежень.

Відомим методом розв’язування задачі ЛП є симплекс-метод, що був опублікований Д.Б. Данцигом у 1949 р. Його ідея полягає в спрямованому переборі допустимих планів у такий спосіб, що на кожному кроці здійснюється перехід від одного опорного плану до іншого, який за значенням цільової функції був би хоча б не гіршим за попередній. Значення функції під час переходу змінюється в потрібному напрямку: збільшується (для задачі на максимум) чи зменшується (для задачі на мінімум).

Симплекс-метод – це ітераційна обчислювальна процедура, яка дає змогу, починаючи від певного опорного плану, за скінченну кількість кроків отримати оптимальний план задачі лінійного програмування.

Алгоритм розв’язання задачі симплекс-методом:

**1.** Зводимо задачу лінійного програмування ***до канонічного вигляду***. При необхідності переходу від нерівності до рівняння вводимо додаткові змінні.

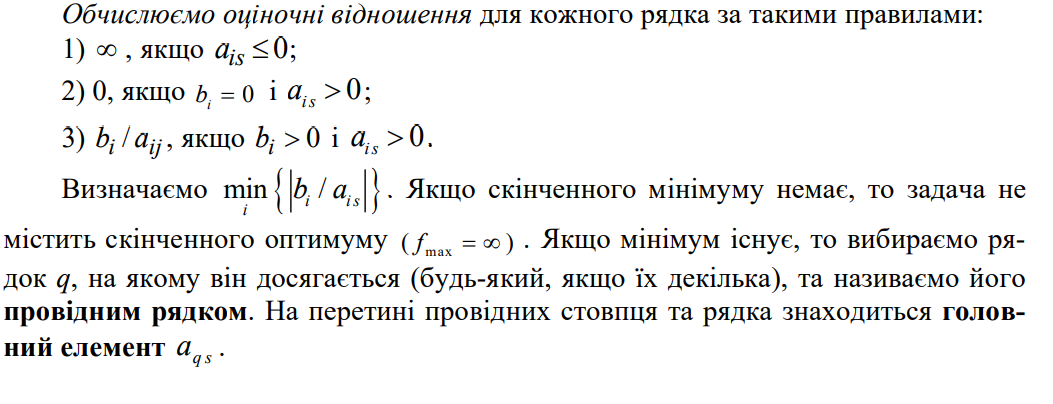
Після введення додаткових змінних систему рівнянь та лінійну функцію записуємо у вигляді розширеної системи:

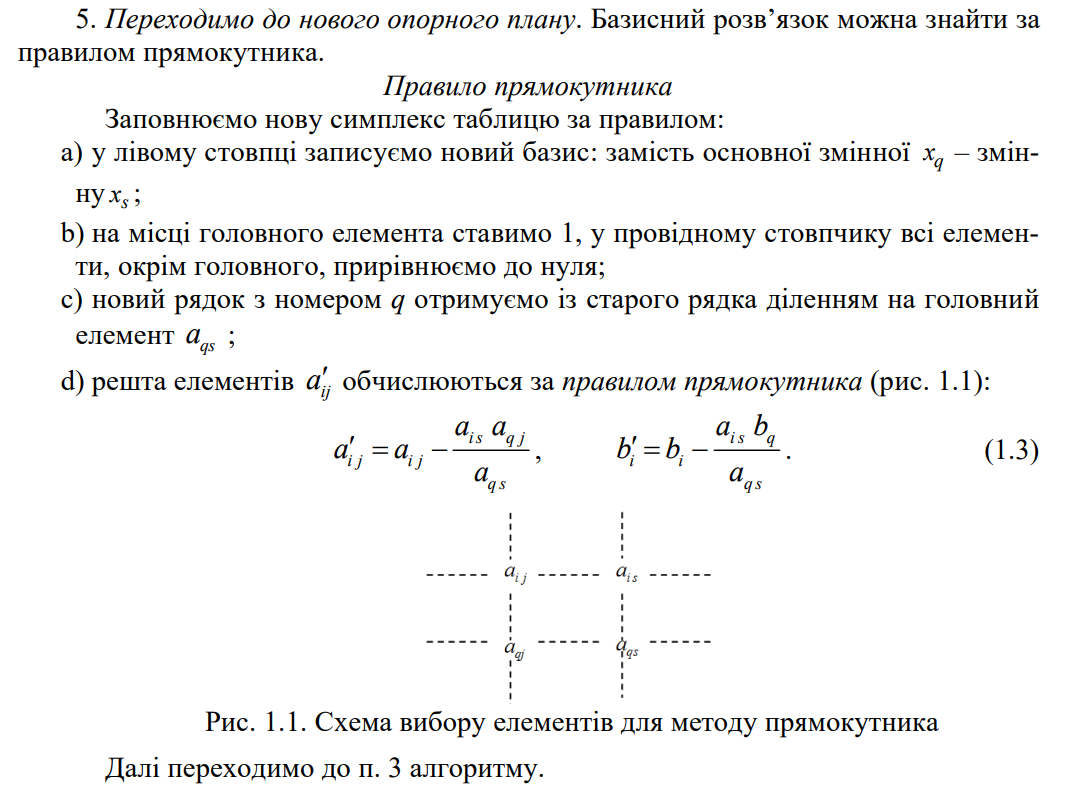
Слід мати на увазі, що всі компоненти вектора правої частини мають бути невід’ємними.

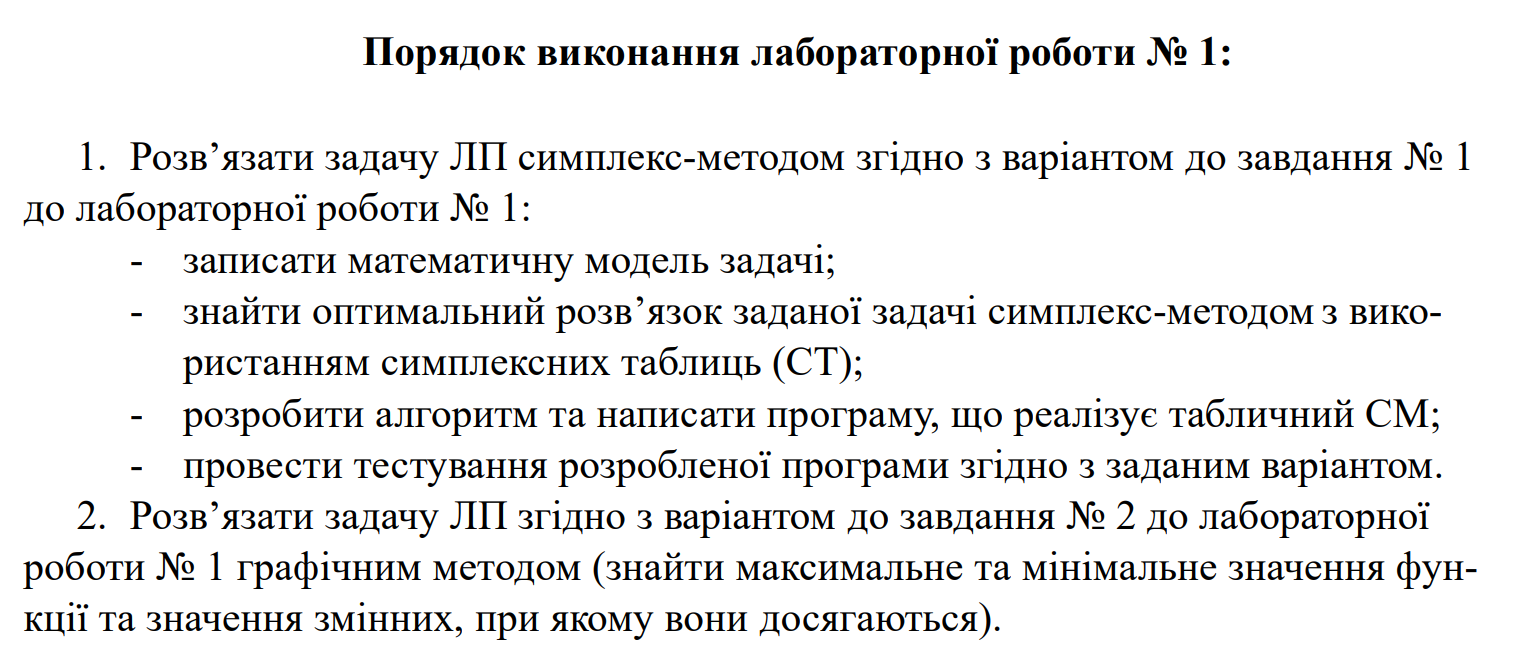
**2.** Знаходимо ***допустимий базисний розв’язок*.** Отриману розширену систему заносимо в першу симплекс-таблицю (СТ-1). Останній рядок таблиці називають оціночним. У ньому, окрім значення цільової функції (в першій таблиці рівного 0), вказуємо критерії оптимальності: для небазисних змінних коефіцієнти цільової функції з протилежним знаком -cj , для базисних 0. У першому зліва стовпці таблиці записуємо основні змінні (базис) xb , а в заголовок таблиці вносимо всі змінні; у другому стовпці – вільні члени розширеної системи b1, b2, …, bm. Останній стовпець необхідний для оціночних відношень, які використовують під час розрахунку найменшого можливого значення змінної. У робочу частину таблиці (починаючи з третього стовпця) заносимо коефіцієнти i j a при всіх змінних із розширеної системи.

**3.** Знайдений опорний план ***перевіряємо на виконання критерію оптимальності*** – для задачі максимізації на наявність в останньому рядку від’ємних коефіцієнтів. Якщо таких коефіцієнтів немає, то розв’язок оптимальний, досягнуто max f = со (в лівому нижньому куті таблиці), основні змінні приймають значення, записані в другому стовпці, а змінні, що не входять в базис, рівні 0, тобто отримуємо оптимальний базисний розв’язок.

**4.** Якщо критерій оптимальності не виконується, то найбільшому за модулем від’ємному коефіцієнту deltas < 0 в останньому рядку відповідає ***провідний стовпець s***.







Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

**2. Симплекс метод**

Зображення, що містить текст, Шрифт, знімок екрана, число

Автоматично згенерований опис

CT-1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| xb | cb | P0 | c1 = 12 | c2 = 5 | c3 = 15 | c4 = 10 | c5 = 0 | c6 = 0 | c7 = 0 | c8 = 0 | c9 = 0 |  |
| P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 |
|  | 0 | 1500 | 5 | 1 | 9 | 12 | 1 | 0 | 0 | 0 | 0 | 166,67 |
|  | 0 | 1000 | 2 | 3 | 4 | 1 | 0 | 1 | 0 | 0 | 0 | 250 |
|  | 0 | 800 | 3 | 2 | 5 | 10 | 0 | 0 | 1 | 0 | 0 | 160 |
|  | 0 | 40 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ∞ |
|  | 0 | 130 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ∞ |
| Q | = |  | -12 | -5 | -15 | -10 | 0 | 0 | 0 | 0 | 0 |  |

CT-2

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| xb | cb | P0 | c1 = 12 | c2 = 5 | c3 = 15 | c4 = 10 | c5 = 0 | c6 = 0 | c7 = 0 | c8 = 0 | c9 = 0 |  |
| P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 |
|  | 0 | 60 | -0,4 | -2,6 | 0 | -6 | 1 | 0 | -1,8 | 0 | 0 | -150 |
|  | 0 | 360 | -0,4 | 1,4 | 0 | -7 | 0 | 1 | -0,8 | 0 | 0 | -900 |
|  | 15 | 160 | 0,6 | 0,4 | 1 | 2 | 0 | 0 | 0,2 | 0 | 0 | 266,67 |
|  | 0 | 40 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 40 |
|  | 0 | 130 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ∞ |
| Q | = |  | -3 | 1 | 0 | 20 | 0 | 0 | 3 | 0 | 0 |  |

CT-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| xb | cb | P0 | c1 = 12 | c2 = 5 | c3 = 15 | c4 = 10 | c5 = 0 | c6 = 0 | c7 = 0 | c8 = 0 | c9 = 0 |  |
| P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 |
|  | 0 | 76 | 0 | -2,6 | 0 | -6 | 1 | 0 | -1,8 | 0,4 | 0 |  |
|  | 0 | 376 | 0 | 1,4 | 0 | -7 | 0 | 1 | -0,8 | 0,4 | 0 |  |
|  | 15 | 136 | 0 | 0,4 | 1 | 2 | 0 | 0 | 0,2 | -0,6 | 0 |  |
|  | 12 | 40 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  |
|  | 0 | 130 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| Q | = | 2520 | 0 | 1 | 0 | 20 | 0 | 0 | 3 | 3 | 0 |  |

Q = 2520

X\* = (40, 0, 136, 0, 76, 376, 0, 130)

B1 = 40 товарів

B2 = 0 товарів

B3 = 136 товарів

B4 = 0 товарів

A1 = 76 сировини залишилось

A2 = 376 сировини залишилось

A3 = 0 сировини залишилось

A4 = 130 сировини залишилось

Q = 2520 ум. од. вартість виготовленої продукції

**2. Графічний метод**

Зображення, що містить текст, Шрифт, почерк, знімок екрана

Автоматично згенерований опис

Зображення, що містить знімок екрана, текст, число, ряд

Автоматично згенерований опис

Рис. 1 Введені в Desmos рівняння (змінну *х* вважати за *х1*, змінну *у* вважати за *х2*)

Зображення, що містить ряд, Графік, схема, Паралель

Автоматично згенерований опис

Рис. 2 Отриманий в Desmos графік і точки перетину

F(1, 3) = 3 + 9 = 12 🡪 max

F(3, 1) = 9 + 3 = 12 🡪 max

F(0.33333 , 3) = 0.99999 + 9 = 9.99999

F(3, 0.2) = 9 + 0.6 = 9.6

F(1.14286, 0.57143) = 3.42858 + 1.71429 = 5,14287 🡪 min

**Програма, що реалізовує табличний СМ**

Наступні рисунки – це результат тестування програми згідно із даними заданими у варіанті.

Зображення, що містить текст, знімок екрана, Шрифт, чорний

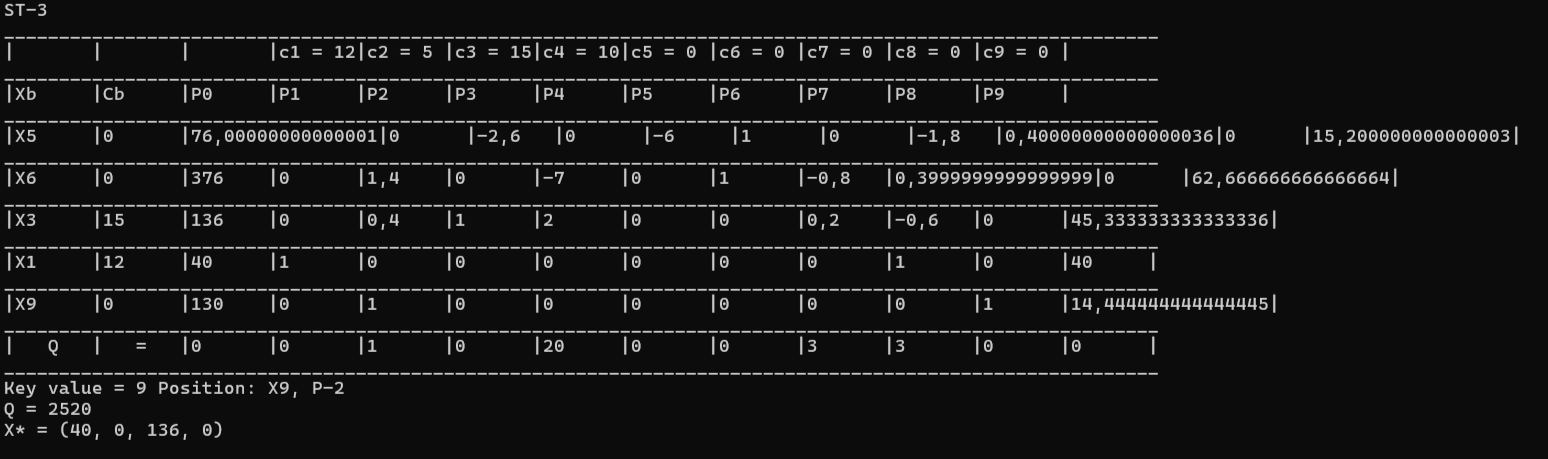
Автоматично згенерований опис

*Рис. 1. Перша симплекс таблиця*

*Зображення, що містить знімок екрана, текст, Шрифт, чорний

Автоматично згенерований опис*

*Рис. 2. Друга симплекс таблиця*

**

*Рис. 3. Третя симплекс таблиця та по сумісництву розв’язок*

**Код програми:**

using System;

using System.Globalization;

namespace Lab01

{

internal class Program

{

private static void Main(string[] args)

{

var sm = new SMCalculator();

}

}

public class SMCalculator

{

private int AmountOfFreeVariables { get; set; }

private int AmountOfBasisVariables { get; set; }

private int SystemPartsAmount { get; set; }

private double[,] \_systemCoefs;

private double[,] \_tableData;

private double[] \_goalFunctionCoefs;

private int \_tableRowCount;

private int \_tableColumnCount;

private int \_tableCount = 0;

public SMCalculator()

{

PrintInstruction();

if (GetDataFromUser())

{

PrintSystem();

PrintGoalFunction();

CreateTable();

PrintSt();

CalculateAllTables();

}

else

{

Console.WriteLine("Can`t get data from user! Wrong value!");

}

}

public SMCalculator(int amountOfFreeVariables, int systemPartsAmount)

{

AmountOfFreeVariables = amountOfFreeVariables;

SystemPartsAmount = systemPartsAmount;

AmountOfBasisVariables = SystemPartsAmount;

\_systemCoefs = new double[SystemPartsAmount, AmountOfFreeVariables + AmountOfBasisVariables + 1];

}

private static bool IsGoodString(string? str)

{

if (str != null)

for (var i = 0; i < str.Length; i++)

{

if (!char.IsDigit(str.ElementAt(i)) && !char.IsWhiteSpace(str.ElementAt(i)) &&

Convert.ToChar(str.ElementAt(i)) != '-')

return false;

}

return true;

}

public bool GetDataFromUser()

{

Console.Write("1. Enter row count: ");

var str1 = Console.ReadLine();

if (IsGoodString(str1))

{

SystemPartsAmount = Convert.ToInt32(str1);

}

else

{

Console.WriteLine("ERROR! Wrong value!");

return false;

}

Console.Write("2. Enter column count: ");

str1 = Console.ReadLine();

if (IsGoodString(str1))

{

AmountOfFreeVariables = Convert.ToInt32(str1);

}

else

{

Console.WriteLine("ERROR! Wrong value!");

return false;

}

PrepareToWork();

Console.WriteLine("3. Enter coefficients of your system WITHOUT right sides:");

for (var i = 0; i < SystemPartsAmount; i++)

{

var res = Console.ReadLine();

if (IsGoodString(res))

{

var coefs = res?.Split(" ");

if (coefs == null) continue;

for (var j = 0; j < coefs.Length; j++)

{

\_systemCoefs[i, j] = Convert.ToDouble(coefs[j]);

}

}

else

{

Console.WriteLine("ERROR! Wrong value!");

return false;

}

}

Console.WriteLine("4. Enter values of right sides of system:");

var str = Console.ReadLine();

string[]? strArr;

if (IsGoodString(str))

{

strArr = str?.Split(" ");

if (strArr != null)

for (var i = 0; i < strArr.Length; i++)

{

\_systemCoefs[i, AmountOfBasisVariables + AmountOfFreeVariables] = Convert.ToInt32(strArr[i]);

}

}

else

{

Console.WriteLine("ERROR! Wrong value!");

return false;

}

Console.WriteLine("5. Enter your goal function coefficients:");

str = Console.ReadLine();

if (IsGoodString(str))

{

strArr = str?.Split(" ");

\_goalFunctionCoefs = new double[AmountOfFreeVariables];

if (strArr != null)

{

for (var i = 0; i < strArr.Length; i++)

{

\_goalFunctionCoefs[i] = Convert.ToInt32(strArr[i]);

}

}

else

{

Console.WriteLine("ERROR! Wrong value!");

return false;

}

}

else

{

Console.WriteLine("ERROR! Wrong value!");

return false;

}

return true;

}

public void PrintInstruction()

{

Console.WriteLine("1. Enter row count!");

Console.WriteLine("2. Enter columns count!");

Console.WriteLine("3. Enter coefficients of your system WITHOUT right sides!");

Console.WriteLine("Example: ");

Console.WriteLine("Your system: ");

Console.WriteLine("1\*x1 <= 30");

Console.WriteLine("1\*x2 <= 70");

Console.WriteLine("3\*x1 + 4\*x2 <= 120");

Console.WriteLine();

Console.WriteLine("You enter:");

Console.WriteLine("1 0");

Console.WriteLine("0 1");

Console.WriteLine("3 4");

Console.WriteLine("REMEMBER! After line of coefficients you should press 'Enter' to start nem line!");

Console.WriteLine("4. Enter values of right sides of system!");

Console.WriteLine("Example: ");

Console.WriteLine("30 70 120");

Console.WriteLine("5. Enter your goal function coefficients!");

Console.WriteLine("Example: ");

Console.WriteLine("Your goal function:");

Console.WriteLine("1\*x1 + 3\*x2 - 4\*x3");

Console.WriteLine("You should enter: ");

Console.WriteLine("1 3 -4");

}

public void PrepareToWork()

{

AmountOfBasisVariables = SystemPartsAmount;

\_systemCoefs = new double[SystemPartsAmount, AmountOfFreeVariables + AmountOfBasisVariables + 1];

for (var i = 0; i < SystemPartsAmount; i++)

{

for (var j = AmountOfFreeVariables; j < AmountOfFreeVariables + AmountOfBasisVariables; j++)

{

if (j - AmountOfFreeVariables == i)

\_systemCoefs[i, j] = 1;

}

}

}

public void PrintSystem()

{

for (var i = 0; i < SystemPartsAmount; i++)

{

var xIndex = 1;

for (var j = 0; j < AmountOfFreeVariables + AmountOfBasisVariables + 1; j++)

{

if (j + 1 == AmountOfFreeVariables + AmountOfBasisVariables + 1)

{

Console.Write(" = " + \_systemCoefs[i, j] + "\n");

}

else

{

if (\_systemCoefs[i, j] != 0)

{

Console.Write(\_systemCoefs[i, j] + "\*x" + xIndex);

if (xIndex + 1 <= AmountOfFreeVariables + i + 1)

Console.Write(" + ");

}

}

xIndex++;

}

}

}

private void CalculateAllTables()

{

while (IsFinalTable() == false)

{

CalculateNextTable();

PrintSt();

}

PrintResult();

}

private void PrintResult()

{

double q = 0;

var x\_arr = new double[AmountOfFreeVariables];

for (var i = 0; i < x\_arr.Length; i++)

x\_arr[i] = 0;

for (var i = 1; i < \_tableRowCount - 1; i++)

{

if (\_tableData[i, 0] <= AmountOfFreeVariables && \_tableData[i, 0] > 0)

x\_arr[(int)\_tableData[i, 0] - 1] = \_tableData[i, 1];

q += \_tableData[i, 1] \* \_tableData[i, 2];

}

Console.WriteLine("Q = " + q);

Console.Write("X\* = (");

for (var i = 0; i < x\_arr.Length; i++)

{

if(i+1 == x\_arr.Length)

Console.Write(x\_arr[i]);

else

Console.Write(x\_arr[i] + ", ");

}

Console.Write(")\n");

}

private bool IsFinalTable()

{

for (var i = 3; i < \_tableColumnCount; i++)

{

if(\_tableData[\_tableRowCount - 1, i] < 0)

return false;

}

return true;

}

private void CalculateNextTable()

{

var keyCol = FindKeyColumn();

var keyRow = FindKeyRow();

var keyValue = \_tableData[keyRow, keyCol];

var newTable = new double[\_tableRowCount, \_tableColumnCount];

//Copying table

for (var i = 0; i < \_tableRowCount; i++)

{

for (var j = 0; j < \_tableColumnCount; j++)

{

newTable[i, j] = \_tableData[i, j];

}

}

newTable[keyRow, 0] = keyCol - 2;

//Filling key value row and col

for (var i = 2; i < \_tableColumnCount-1; i++)

{

newTable[keyRow, i] = \_tableData[keyRow, i] / keyValue;

}

for (var i = 1; i < \_tableRowCount - 1; i++)

{

newTable[i, keyCol] = 0;

}

newTable[keyRow, keyCol] = 1;

//Filling rest of table

for (int i = 1; i < \_tableRowCount-1; i++)

{

for (int j = 2; j < \_tableColumnCount-1; j++)

{

if(i != keyRow && j != keyCol)

newTable[i, j] = \_tableData[i, j] - ((\_tableData[keyRow, j] \* \_tableData[i, keyCol]) / keyValue);

}

}

//Filling basis cols

for (int i = 1; i < \_tableRowCount - 1; i++)

{

FillBasisCol((int)newTable[i, 0] + 2, newTable);

}

//Filling Cb col

newTable[keyRow, 1] = \_tableData[0, keyCol];

//Filling last row

for (var i = 3; i < \_tableColumnCount - 1; i++)

{

double sum = 0;

for (var j = 1; j < \_tableRowCount - 1; j++)

{

sum += newTable[j, 1] \* newTable[j, i];

}

newTable[\_tableRowCount - 1, i] = sum - newTable[0, i];

}

//Filling last col

keyCol = FindKeyColumn(newTable);

for (var i = 1; i < \_tableRowCount - 1; i++)

{

newTable[i, \_tableColumnCount - 1] = newTable[i, 2] / newTable[i, keyCol];

}

for (var i = 0; i < \_tableRowCount; i++)

{

for (var j = 0; j < \_tableColumnCount; j++)

{

\_tableData[i, j] = newTable[i, j];

}

}

\_tableData = newTable;

}

private void FillBasisCol(int col, double[,] table)

{

for (var i = 1; i < \_tableRowCount - 1; i++)

{

if (table[i, 0] == (col - 2))

{

table[i, col] = 1;

}

else

{

table[i, col] = 0;

}

}

}

private int FindKeyColumn()

{

var pos = 0;

double min = 0;

for (var i = 3; i < \_tableColumnCount; i++)

{

if (\_tableData[\_tableRowCount - 1, i] < min)

{

min = \_tableData[\_tableRowCount - 1, i];

pos = i;

}

}

return pos;

}

private int FindKeyColumn(double[,] table)

{

var pos = 0;

double min = 0;

for (var i = 3; i < \_tableColumnCount; i++)

{

if (table[\_tableRowCount - 1, i] < min)

{

min = table[\_tableRowCount - 1, i];

pos = i;

}

}

return pos;

}

private int FindKeyRow()

{

var pos = 0;

double min = double.MaxValue;

for (var i = 1; i < \_tableRowCount-1; i++)

{

if(\_tableData[i, \_tableColumnCount - 1] > 0)

if (\_tableData[i, \_tableColumnCount - 1] < min)

{

min = \_tableData[i, \_tableColumnCount - 1];

pos = i;

}

}

return pos;

}

private void PrintSt()

{

\_tableCount++;

Console.WriteLine("ST-" + \_tableCount);

PrintLine();

Console.Write("| | | |");

for (var i = 1; i < AmountOfFreeVariables + AmountOfBasisVariables + 1; i++)

{

var charNum = 1 + i.ToString().Length + 3 + \_tableData[0, i + 2].ToString(CultureInfo.InvariantCulture).Length;

var spaceNum = 0;

if(charNum < 7)

spaceNum = 7 - charNum;

Console.Write("c" + i + " = " + \_tableData[0, i + 2]);

if(spaceNum != 0)

for (var j = 0; j < spaceNum; j++)

Console.Write(" ");

Console.Write("|");

}

Console.Write("\n");

PrintLine();

Console.Write("|Xb |Cb |P0 |");

for (var i = 1; i < AmountOfFreeVariables + AmountOfBasisVariables + 1; i++)

{

Console.Write("P" + i + " |");

}

Console.Write("\n");

PrintLine();

for (int i = 1; i < \_tableRowCount - 1; i++)

{

for (var j = 0; j < \_tableColumnCount; j++)

{

if(j == 0)

Console.Write("|X" + \_tableData[i, j] + " |");

else

{

var numLength = \_tableData[i, j].ToString(CultureInfo.InvariantCulture).Length;

var spaceCount = 7 - numLength;

Console.Write(\_tableData[i,j]);

for (var k = 0; k < spaceCount; k++)

{

Console.Write(" ");

}

Console.Write("|");

}

}

Console.WriteLine();

PrintLine();

}

//Printing last row

Console.Write("| Q | = |");

var nl = \_tableData[\_tableRowCount-1, 2].ToString(CultureInfo.InvariantCulture).Length;

var sc = 7 - nl;

Console.Write(\_tableData[\_tableRowCount - 1, 2]);

for (var k = 0; k < sc; k++)

{

Console.Write(" ");

}

Console.Write("|");

for (var j = 3; j < \_tableColumnCount; j++)

{

var numLength = \_tableData[\_tableRowCount - 1, j].ToString(CultureInfo.InvariantCulture).Length;

var spaceCount = 7 - numLength;

Console.Write(\_tableData[\_tableRowCount - 1, j]);

for (var k = 0; k < spaceCount; k++)

{

Console.Write(" ");

}

Console.Write("|");

}

Console.WriteLine();

PrintLine();

int row = FindKeyRow();

int col = FindKeyColumn();

Console.WriteLine("Key value = " + \_tableData[row, col] + " Position: X" + \_tableData[row, 0] + ", P" + (col - 2));

}

private void PrintLine()

{

for (var i = 0; i < \_tableColumnCount; i++)

{

Console.Write(i + 1 == \_tableColumnCount ? "\_\_\_\_\_\_\_\_\_" : "\_\_\_\_\_\_\_\_");

}

Console.WriteLine();

}

public void PrintGoalFunction()

{

Console.Write("Q = ");

for (var i = 0; i < AmountOfFreeVariables; i++)

{

if (i + 1 == AmountOfFreeVariables)

{

Console.Write(\_goalFunctionCoefs[i] + "\*x" + i + " = ");

}

else

{

Console.Write(\_goalFunctionCoefs[i] + "\*x" + i + " + ");

}

}

Console.WriteLine();

}

public void CreateTable()

{

\_tableColumnCount = 3 + AmountOfBasisVariables + AmountOfFreeVariables + 1;

\_tableRowCount = 1 + SystemPartsAmount + 1;

\_tableData = new double[\_tableRowCount, \_tableColumnCount];

//Filling first column with basis variables

for (var i = 1; i < \_tableRowCount - 1; i++)

{

\_tableData[i, 0] = AmountOfFreeVariables + i;

}

//Filling first row

for (var i = 3; i < \_tableColumnCount; i++)

{

if (i - 3 < AmountOfFreeVariables)

\_tableData[0, i] = \_goalFunctionCoefs[i - 3];

else

\_tableData[0, i] = 0;

}

//Filling last row

for (var i = 3; i < \_tableColumnCount; i++)

{

if(\_tableData[0, i] != 0)

\_tableData[\_tableRowCount - 1, i] = \_tableData[0, i] \* -1;

else

{

\_tableData[\_tableRowCount - 1, i] = 0;

}

}

//Filling third column

for (var i = 1; i < \_tableRowCount - 1; i++)

{

\_tableData[i, 2] = \_systemCoefs[i-1, AmountOfFreeVariables + AmountOfBasisVariables];

}

//Filling other parts of table

for (var i = 1; i < \_tableRowCount - 1; i++)

{

for (var j = 3; j < \_tableColumnCount; j++)

{

if (j - 3 < AmountOfFreeVariables)

\_tableData[i, j] = \_systemCoefs[i - 1, j - 3];

else

{

if (j - 2 == (int)\_tableData[i, 0])

{

\_tableData[i, j] = 1;

}

else

{

\_tableData[i, j] = 0;

}

}

}

}

//Filling last column

int colNum = FindKeyColumn();

for (int i = 1; i < \_tableRowCount - 1; i++)

{

\_tableData[i, \_tableColumnCount - 1] = \_tableData[i, 2] / \_tableData[i, colNum];

}

}

}

}

**Висновки:** протягом виконання цієї лабораторної роботи я ознайомився на практиці із основними поняттями теорії лінійного програмування (ЛП), отримав навички практичного розв’язання задач ЛП табличним симплекс-методом (СМ) та навчився його реалізовувати програмно, навчився розв’язувати задачі ЛП із двома змінними графічним методом.