**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

**ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №2

**на тему:** “*Двоїстий симплекс-метод. Зв’язок між розв’язками прямої та двоїстої задач ЛП”*

# 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:** Двоїстий симплекс-метод. Зв’язок між розв’язками прямої та двоїстої задач ЛП.

**Мета:** Ознайомитись на практиці із двоїстими задачами лінійного програмування та навчитись розв’язувати їх із використанням двоїстого симплекс-методу.

**Порядок виконання лабораторної роботи № 2:**

1. Побудувати двоїсту задачу до задачі ЛП (згідно з варіантом завдання № 1 до лабораторної роботи № 1).

2. Знайти розв’язок (мінімальне значення функції та значення змінних, приякому воно досягається) побудованої двоїстої задачі.

3. Розробити алгоритм та написати програму, що реалізує двоїстий симплекс-метод.

4. Провести тестування розробленої програми згідно з заданим варіантом.

5. Знайти оптимальний план прямої задачі, використовуючи зв’язок між розв’язками прямої та двоїстої задач. Порівняти отриманий розв’язок з відповідними результатами лабораторної роботи №1.

6. Зазначити економічний зміст функції мети і системи обмежень обидвох задач. Дати економічне тлумачення основних і додаткових змінних вихідної та двоїстої задач.

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

Автоматично згенерований опис

**2. Двоїстий симплекс-метод**

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

Автоматично згенерований опис

Z(y1, y2, y3, y4, y5, y6, y7, y8, y9) = (-1500, -1000, -800, -40, -130, 0 ,0 ,0 ,0)

Z = -1500y1 - 1000y2 - 800y3 - 40y4 - 130y5 → max

**Програма, що реалізовує табличний СМ**

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

Зображення, що містить знімок екрана, клавіатура

Автоматично згенерований опис

*Рис. 1. Перша таблиця*

Зображення, що містить знімок екрана, текст, клавіатура

Автоматично згенерований опис

*Рис. 2. Друга таблиця*

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

Автоматично згенерований опис*

*Рис. 3. Третя таблиця та за сумісництвом розв’язок*

Y\* = (0, 0, 0, 3, 3, 0, 1, 0, 20)

minZ\* = -maxF\* = 2520 опорний план

y1 = 0 збільшення обсягів якщо збільшити сировину виду на одиницю

y2 = 0 збільшення обсягів якщо збільшити сировину виду на одиницю

y3 = 0 збільшення обсягів якщо збільшити сировину виду на одиницю

y4 = 3 збільшення обсягів якщо збільшити сировину виду на одиницю

y5 = 3 відношення вартості сировини до вартості продукту.

y6 = 0 відношення вартості сировини до вартості продукту.

y7 = 1 відношення вартості сировини до вартості продукту.

y8 = 0 відношення вартості сировини до вартості продукту.

y9 = 20 відношення вартості сировини до вартості продукту.

**Знаходження оптимального плану:**

***Таблиця 1 – Остання симплекс-таблиця прямого розв’язку***

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

Автоматично згенерований опис

***Таблиця 2 – Остання симплекс-таблиця двоїстого розв’язку***

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

Автоматично згенерований опис

P1 = y5 = 0; P2 = y6 = 0; P3 = y7 = 6; P4 = y8 = 2

P5 = y1 = 2.5; P6 = y2 = 0; P7 = y3 = 0; P8 = y4 = 3

Z\*min = 800 ⋅ 3 + 40 ⋅ 3 = 2520 => співпадає з максимальним значенням цільової функції вихідної задачі.

**Економічний зміст функції мети і системи обмежень обидвох задач:**

1. **Функція мети (прямої задачі):**

Метою фабрики є максимізація прибутку від випуску продукції чотирьох асортиментів B1​, B2​, B3​, B4​, з умовою, що продукції виду B1 буде не більше 40 штук та продукції виду B2, не більше 130 штук. Функція мети представляє собою загальний прибуток від реалізації цієї продукції, яка обчислюється як сума прибутків від кожного асортименту. Для кожного виду продукції відомо, що прибуток за одиницю продукції становить 12 для B1, 5 для B2​, 15 для B3​, 10 для B4​.

1. **Система обмежень (прямої задачі):**

Кожен вид продукції потребує певної кількості сировини для виробництва, причому кількість сировини обмежена.

Ці обмеження вказують на кількість сировини, яку можна використовувати для кожного асортименту продукції.

**Економічний зміст основних та додаткових змінних вихідної задачі:**

Основні змінні x1, x2, x3, x4 ​ представляють кількість одиниць кожного виду продукції B1​, B2​, B3​, B4​, яку необхідно виробляти для максимізації прибутку.

Додаткові змінні (в симплекс-таблиці): вони показують недовикористання ресурсів або надлишок сировини, якщо значення змінної більше нуля.

1. **Функція мети (двоїстої задачі):**

Вартість сировини, яку підприємство готове платити за обмежений ресурс (ціну тіньову).

Функція мети вказує на мінімальну вартість ресурсів, яка відповідає максимальному прибутку для підприємства.

1. **Система обмежень (двоїстої задачі):**

Ці обмеження вказують на те, що фабрика не повинна платити за сировину більше, ніж прибуток, який можна отримати від виробництва продукції з цієї сировини.

**Основні і додаткові змінні двоїстої задачі:**

Основні змінні двоїстої задачі (ціни тіньові) показують, наскільки важлива кожна сировина для максимізації прибутку.

Додаткові змінні вказують на надлишок або дефіцит ресурсів, який можна використовувати для оптимізації виробництва.

Рішення двох задач взаємопов'язане і допомагає знайти баланс між витратами на ресурси і максимізацією прибутку підприємства.

**Код програми:**

using System.Globalization;

using System.Text.RegularExpressions;

using System.Xml.Serialization;

namespace Lab02;

internal class Lab02

{

private static void Main()

{

var dsm = new Dsm();

}

}

public class Dsm

{

private int \_xSystemRowCount;

private int \_xSystemColumnCount;

private int \_ySystemRowCount;

private int \_ySystemColumnCount;

private List<int> \_ybCol;

private List<double> \_cnRow;

private List<double> \_cbCol;

private List<List<double>> \_pnTable;

private List<double> \_p0Col;

private List<double> \_zRow;

private List<double> \_ratio;

private double \_zFinalValue;

private List<string> \_xSystemRows;

private List<string> \_xSystemOpType;

private List<List<double>> \_xSystem;

private List<List<double>> \_ySystem;

private List<string> \_ySystemRows;

private List<string> \_ySystemOpType;

private string \_fString;

private string \_zString;

private List<double> \_fCoefficients;

private List<double> \_zCoefficients;

private string \_xOptimizationType;

private string \_yOptimizationType;

private int \_additionalVars;

private int \_tableCounter;

private int \_colSize;

private string \_tableBrakingLine;

private int \_keyRow;

private int \_keyCol;

public Dsm()

{

if (!GetFStringFromUser()) return;

HandleFString();

GetXSystemFromUser();

PrintDataForSsm();

ConvertSsmDataToDsmData();

PrintDataForDsm("base");

ConvertDsmDataToCanonicalForm();

PrintDataForDsm("canonical");

CreateTable();

CalculateRatio();

PrintTable();

FindKey();

PrintKey();

CalculateAndPrintAllTables();

}

private void CalculateAndPrintAllTables()

{

while (!IsEnd())

{

WriteTableBreakingLine();

CalculateNextTable();

CalculateRatio();

CalculateZValue();

PrintTable();

FindKey();

PrintKey();

}

PrintResults();

}

private void PrintResults()

{

Console.WriteLine($"Z = {\_zFinalValue}");

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

var isPrinted = false;

for (var i = 0; i < \_ySystemColumnCount - 1; i++)

{

for (var j = 0; j < \_ybCol.Count; j++)

{

if (i + 1 == \_ybCol[j])

{

Console.Write($"{\_p0Col[j]}");

isPrinted = true;

}

}

if(isPrinted == false)

Console.Write($"0");

if(i+1 != \_ySystemColumnCount-1)

Console.Write(", ");

isPrinted = false;

}

Console.WriteLine(")");

}

private void CalculateZValue()

{

\_zFinalValue = 0;

for (var i = 0; i < \_cbCol.Count; i++)

{

\_zFinalValue += \_cbCol[i] \* \_p0Col[i];

}

}

private void PrintKey()

{

Console.WriteLine($"\nKey = {\_ySystem[\_keyRow][\_keyCol]:F2}, row = y{\_ybCol[\_keyRow]}, column = P{\_keyCol + 1}\n");

}

private bool IsEnd()

{

foreach (var num in \_p0Col)

{

if(num < 0)

{ return false; }

}

return true;

}

private void CalculateNextTable()

{

//Swap Yb item and Cb item

\_ybCol[\_keyRow] = \_keyCol + 1;

\_cbCol[\_keyRow] = \_cnRow[\_keyCol];

//Calculate new system

List<List<double>> newSystem = [];

for (var i = 0; i < \_ySystem.Count; i++)

{

newSystem.Add([]);

for(var j = 0; j < \_ySystem[i].Count; j++)

newSystem[i].Add(0);

}

for (var i = 0; i < \_ySystem.Count; i++)

{

for (var j = 0; j < \_ySystem[i].Count; j++)

{

if (i != \_keyRow)

newSystem[i][j] = \_ySystem[i][j] -

((\_ySystem[\_keyRow][j] \* \_ySystem[i][\_keyCol]) / \_ySystem[\_keyRow][\_keyCol]);

else

{

newSystem[i][j] = \_ySystem[i][j] / \_ySystem[\_keyRow][\_keyCol];

}

}

}

//Calculate new \_p0Col

List<double> newCol = [];

for (var i = 0; i < \_p0Col.Count; i++)

{

if (i != \_keyRow)

{

newCol.Add(\_p0Col[i] - ((\_p0Col[\_keyRow] \* \_ySystem[i][\_keyCol]) / \_ySystem[\_keyRow][\_keyCol]));

}

else

{

newCol.Add(\_p0Col[i] / \_ySystem[\_keyRow][\_keyCol]);

}

}

//Rewriting data

\_ySystem.Clear();

for (var i = 0; i < newSystem.Count; i++)

{

\_ySystem.Add([]);

for (var j = 0; j < newSystem[i].Count; j++)

\_ySystem[i].Add(newSystem[i][j]);

}

\_p0Col.Clear();

for(var i = 0; i < newCol.Count; i++)

\_p0Col.Add(newCol[i]);

//Calculate new \_zRow

List<double> newRow = [];

for (var i = 0; i < \_zRow.Count; i++)

{

double temp = 0;

for (var j = 0; j < \_cbCol.Count; j++)

{

temp += \_cbCol[j] \* \_ySystem[j][i];

}

newRow.Add(temp - \_cnRow[i]);

}

\_zRow.Clear();

for (var i = 0; i < newRow.Count; i++)

\_zRow.Add(newRow[i]);

}

private void CalculateRatio()

{

FindKeyRow();

\_ratio = [];

for (var i = 0; i < \_zRow.Count; i++)

{

if (\_zRow[i] != 0 && \_ySystem[\_keyRow][i] != 0)

\_ratio.Add((\_zRow[i] \* -1) / \_ySystem[\_keyRow][i]);

else

{

\_ratio.Add(0);

}

}

}

private void FindKeyRow()

{

var max = \_p0Col[0] \* -1;

var kRow = 0;

for (var i = 0; i < \_p0Col.Count; i++)

{

if ((\_p0Col[i] \* -1) > max)

{

max = \_p0Col[i] \* -1;

kRow = i;

}

}

\_keyRow = kRow;

}

private void FindKey()

{

FindKeyRow();

double min = 0;

var kCol = 0;

for (var i = 0; i < \_zRow.Count; i++)

{

if (\_ratio[i] > 0)

{

if (min == 0 || \_ratio[i] < min && i + 1 != \_ybCol[\_keyRow])

{

min = \_ratio[i];

kCol = i;

}

}

}

\_keyCol = kCol;

}

private void PrintTable()

{

//Searching column size

var temp = 0;

for (var i = 0; i < \_ySystemColumnCount - 1; i++)

{

var str = "";

str += $"C{i + 1} = {\_cnRow[i]:F2}";

if (str.Length > temp)

temp = str.Length;

}

\_colSize = temp;

//Initializing table braking line

\_tableBrakingLine = "";

for (var i = 0; i < 3 + \_ySystemColumnCount - 1; i++)

{

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

{

\_tableBrakingLine += "-";

}

}

for(var i = 0; i < 4+\_ySystemColumnCount-1; i++)

\_tableBrakingLine += "-";

\_tableCounter++;

WriteBreakingPartsLine();

Console.WriteLine($"DST-{\_tableCounter}\n");

WriteTableBreakingLine();

//Cn row printing

Console.Write("|");

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

{

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

Console.Write(" ");

}

for (var i = 0; i < 2; i++)

Console.Write(" ");

Console.Write("|");

for (var i = 0; i < \_cnRow.Count; i++)

{

var str = "";

str += $"C{i + 1} = {\_cnRow[i]:F2}";

WriteStrInColumn(str);

Console.Write("|");

}

Console.WriteLine();

WriteTableBreakingLine();

//Headers printing

Console.Write("|");

WriteStrInColumn("Yb");

Console.Write("|");

WriteStrInColumn("Cb");

Console.Write("|");

WriteStrInColumn("P0");

Console.Write("|");

for (var i = 0; i < \_ySystemColumnCount - 1; i++)

{

WriteStrInColumn($"P{i+1}");

Console.Write("|");

}

Console.WriteLine();

//Middle table filling

WriteTableBreakingLine();

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

{

Console.Write("|");

WriteStrInColumn($"y{\_ybCol[i]}");

Console.Write("|");

WriteStrInColumn($"{\_cbCol[i]}");

Console.Write("|");

WriteStrInColumn($"{\_p0Col[i]:F2}");

Console.Write("|");

for (var j = 0; j < \_ySystemColumnCount - 1; j++)

{

WriteStrInColumn($"{\_ySystem[i][j]:F2}");

Console.Write("|");

}

Console.WriteLine();

WriteTableBreakingLine();

}

//Last row printing

Console.Write("|");

WriteStrInColumn("Z");

Console.Write("|");

WriteStrInColumn("=");

Console.Write("|");

WriteStrInColumn($"{\_zFinalValue}");

Console.Write("|");

for (var i = 0; i < \_zRow.Count; i++)

{

WriteStrInColumn($"{\_zRow[i]:F2}");

Console.Write("|");

}

Console.WriteLine();

WriteTableBreakingLine();

//Ratio printing

Console.Write("|");

WriteStrInColumn("Ratio");

Console.Write("|");

WriteStrInColumn("");

Console.Write("|");

WriteStrInColumn("");

Console.Write("|");

for (var i = 0; i < \_ratio.Count; i++)

{

WriteStrInColumn($"{\_ratio[i]:F2}");

Console.Write("|");

}

Console.WriteLine();

WriteTableBreakingLine();

}

private void WriteStrInColumn(string str)

{

var strSize = str.Length;

if (strSize < \_colSize)

{

if ((\_colSize - strSize) % 2 == 0)

{

for (var k = 0; k < (\_colSize - strSize) / 2; k++)

{

Console.Write(" ");

}

Console.Write(str);

for (var k = 0; k < (\_colSize - strSize) / 2; k++)

{

Console.Write(" ");

}

}

else

{

var leftSpacing = Convert.ToInt32(((\_colSize - strSize) / 2.0) - 0.5f);

var rightSpacing = Convert.ToInt32(((\_colSize - strSize) / 2.0) + 0.5f);

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

{

Console.Write(" ");

}

Console.Write(str);

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

{

Console.Write(" ");

}

}

}

else

{

Console.Write(str);

}

}

private void WriteBreakingPartsLine()

{

Console.WriteLine("---------------------------------------------------------------------------------------------------------------------------------------------------------");

}

private void WriteTableBreakingLine()

{

Console.WriteLine(\_tableBrakingLine);

}

private void CreateTable()

{

\_tableCounter = 0;

//Cn row filling, Z row filling

\_cnRow = [];

\_zRow = [];

\_zFinalValue = 0;

for (var i = 0; i < \_ySystemColumnCount-1; i++)

{

\_cnRow.Add(\_zCoefficients[i]);

\_zRow.Add(\_cnRow[i] \* -1);

}

//Yb column filling, Cb column filling, P0 column filling

\_ybCol = [];

\_cbCol = [];

\_p0Col = [];

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

{

\_ybCol.Add(\_ySystemColumnCount - \_additionalVars + i);

\_cbCol.Add(\_cnRow[\_ybCol[i] - 1]);

\_p0Col.Add(\_ySystem[i][\_ySystemColumnCount-1]);

}

//Pn table filling

\_pnTable = [];

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

{

\_pnTable.Add([]);

for (var j = 0; j < \_ySystemColumnCount-1; j++)

{

\_pnTable[i].Add(\_ySystem[i][j]);

}

}

}

private void ConvertDsmDataToCanonicalForm()

{

//All system multiplying with -1

foreach (var list in \_ySystem)

{

for (var i = 0; i < list.Count; i++)

{

if (list[i] != 0)

list[i] \*= -1;

}

}

//Adding additional variables

foreach (var list in \_ySystem)

{

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

{

list.Add(0);

}

list[list.Count - 1] = list[list.Count - 1 - \_additionalVars];

list[list.Count - 1 - \_additionalVars] = 0;

}

\_ySystemColumnCount += \_additionalVars;

//Additional variables initializing

for (var i = 0; i < \_ySystem.Count; i++)

{

\_ySystem[i][\_ySystem[i].Count - \_additionalVars + i - 1] = 1;

}

//Goal function multiplying with -1

for (var i = 0; i < \_zCoefficients.Count; i++)

\_zCoefficients[i] \*= -1;

var newZStr = "";

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

{

\_zCoefficients.Add(0);

}

for (var i = 0; i < \_zCoefficients.Count; i++)

{

newZStr += \_zCoefficients[i].ToString(CultureInfo.InvariantCulture) + "\*y" + (i + 1);

if (i + 1 < \_ySystemColumnCount - 1)

newZStr += " + ";

else

newZStr += " -> ";

}

\_yOptimizationType = \_yOptimizationType == "max" ? "min" : "max";

newZStr += \_yOptimizationType;

\_zString = newZStr;

CreateYSystemRows("canonical");

}

private void CreateYSystemRows(string whatFor)

{

if (whatFor == "base")

{

\_ySystemOpType = [];

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

{

if (\_xSystemOpType[i] == "<=")

\_ySystemOpType.Add(">=");

else if (\_xSystemOpType[i] == ">=")

\_ySystemOpType.Add("<=");

}

}

else if (whatFor == "canonical")

{

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

{

if (\_ySystemOpType[i] == "<=")

\_ySystemOpType[i] = ">=";

else if (\_ySystemOpType[i] == ">=")

\_ySystemOpType[i] = "<=";

}

}

\_ySystemRows = [];

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

{

var rowStr = "";

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

{

if (j == \_ySystemColumnCount - 1)

rowStr += " " + \_ySystemOpType[i] + " ";

var str = \_ySystem[i][j].ToString(CultureInfo.InvariantCulture);

if (j - 1 != \_ySystemColumnCount - 1 && j + 1 != \_ySystemColumnCount)

str += "\*y" + (j + 1);

rowStr += str;

if (j + 2 < \_ySystemColumnCount)

{

rowStr += " + ";

}

}

\_ySystemRows.Add(rowStr);

}

}

private void ConvertSsmDataToDsmData()

{

\_ySystem = [];

\_ySystemRowCount = \_xSystemColumnCount-1;

\_ySystemColumnCount = \_xSystemRowCount + 1;

\_additionalVars = \_ySystemRowCount;

// Y system creating

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

{

\_ySystem.Add([]);

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

{

if (j + 1 == \_ySystemColumnCount)

{

\_ySystem[i].Add(\_fCoefficients.ElementAt(i));

}

else

{

\_ySystem[i].Add(\_xSystem[j][i]);

}

}

}

CreateYSystemRows("base");

//Z string creating

\_zCoefficients = [];

\_zString = "";

for (var i = 0; i < \_ySystemColumnCount - 1; i++)

{

\_zCoefficients.Add(\_xSystem[i][\_xSystemColumnCount-1]);

\_zString += \_zCoefficients[i].ToString(CultureInfo.InvariantCulture) + "\*y" + (i + 1);

if (i + 1 < \_ySystemColumnCount - 1)

{

\_zString += " + ";

}

else

\_zString += " -> ";

}

if (\_xOptimizationType == "max")

\_zString += "min";

else if (\_xOptimizationType == "min")

\_zString += "max";

}

private void PrintDataForDsm(string str)

{

Console.WriteLine();

switch (str)

{

case "base":

Console.WriteLine("----------------------- Data for Dual Simplex Method -----------------------");

break;

case "canonical":

Console.WriteLine("-----------------------Canonical Data for Dual Simplex Method -----------------------");

break;

}

PrintYSystem();

PrintZ();

}

private void PrintYSystem()

{

Console.WriteLine();

Console.WriteLine("Your system for Dual Simplex Method:");

foreach (var t in \_ySystemRows)

{

Console.WriteLine(t);

}

}

private void PrintDataForSsm()

{

Console.WriteLine();

Console.WriteLine("----------------------- Data for Standard Simplex Method -----------------------");

PrintXSystem();

PrintF();

}

private void PrintXSystem()

{

Console.WriteLine();

Console.WriteLine("Your system for Standard Simplex Method:");

foreach (var t in \_xSystemRows)

{

Console.WriteLine(t);

}

}

private void PrintF()

{

Console.WriteLine("F = " + \_fString);

}

private void PrintZ()

{

Console.WriteLine("Z = " + \_zString);

}

private bool GetFStringFromUser()

{

Console.WriteLine("Enter your F!\nExample: 10\*x1 + 0\*x2 - 12\*x3 -> max(min)");

var str = Console.ReadLine();

const string pattern = @"^(\s\*(-?\d+)\\*x\d+\s\*([+-]\s\*\d+\\*x\d+\s\*)\*)\s\*->\s\*(max|min)\s\*$";

if (str == null) return false;

if (!Regex.IsMatch(str, pattern, RegexOptions.IgnoreCase)) return false;

\_fString = str;

return true;

}

private void HandleFString()

{

// Регулярний вираз для пошуку коефіцієнтів при змінних

const string coeffPattern = @"(-?\d+(\.\d+)?)\\*x\d+";

// Регулярний вираз для пошуку операції max або min

const string optimizationPattern = @"->\s\*(max|min)";

// Знаходимо всі коефіцієнти

var coeffMatches = Regex.Matches(\_fString, coeffPattern);

\_fCoefficients = [];

foreach (Match match in coeffMatches)

{

// Отримуємо коефіцієнт як рядок, потім перетворюємо на double

var coeffStr = Regex.Match(match.Value, @"-?\d+(\.\d+)?").Value;

var coeff = double.Parse(coeffStr);

\_fCoefficients.Add(coeff);

}

// Знаходимо max або min

var optimizationMatch = Regex.Match(\_fString, optimizationPattern, RegexOptions.IgnoreCase);

if (optimizationMatch.Success)

{

\_xOptimizationType = optimizationMatch.Groups[1].Value;

}

}

private void GetXSystemFromUser()

{

Console.WriteLine("Enter row count in system:");

var rows = int.Parse(Console.ReadLine() ?? string.Empty);

Console.WriteLine("Enter column count in system:");

var cols = int.Parse(Console.ReadLine() ?? string.Empty);

Console.WriteLine("Enter your system rows!\nYou have: x1 + 3\*x2 - 4\*x4 <= 100\nYou should enter: 1 3 0 -4 <= 100");

\_xSystemRows = [];

\_xSystemRowCount = rows;

\_xSystemColumnCount = cols;

var separators = new[] { ' ' };

\_xSystem = [];

\_xSystemOpType = [];

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

{

\_xSystem.Add([]);

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

{

\_xSystem[i].Add(0);

}

}

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

{

var str = Console.ReadLine();

var tokens = str?.Split(separators, StringSplitOptions.RemoveEmptyEntries);

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

{

if (j == cols-1)

{

if (tokens != null)

{

\_xSystemOpType.Add(tokens.ElementAt(j));

\_xSystem[i][j] = Convert.ToDouble(tokens.ElementAt(j + 1));

}

}

else

{

if (tokens != null)

\_xSystem[i][j] = Convert.ToDouble(tokens.ElementAt(j));

}

}

}

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

{

var rowStr = "";

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

{

if (j == cols - 1)

rowStr += " " + \_xSystemOpType[i] + " ";

var str = \_xSystem[i][j].ToString(CultureInfo.InvariantCulture);

if (j - 1 != cols - 1 && j + 1 != cols)

str += "\*x" + (j + 1);

rowStr += str;

if(j + 2 < cols)

{

rowStr += " + ";

}

}

\_xSystemRows.Add(rowStr);

}

}

}

**Висновки:** протягом виконання цієї лабораторної роботи я ознайомився на практиці із двоїстими задачами лінійного програмування та навчився розв’язувати їх із використанням двоїстого симплекс-методу.