**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:**

проф. каф. ПЗ

Журавчак, Л. М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

ст. викл. каф. ПЗ

Івасько Н. М.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

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

**Мета роботи:** 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

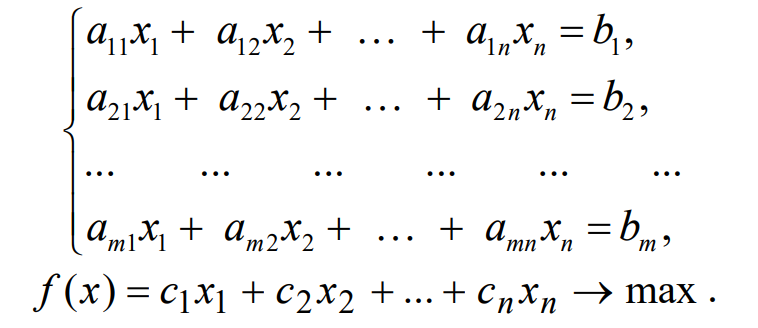
Відомим методом розв’язування задачі ЛП є **симплекс-метод**, що був опублікований Д.Б. Данцигом у 1949 р. Його ідея полягає в спрямованому переборі допустимих планів у такий спосіб, що на кожному кроці здійснюється перехід від одного опорного плану до іншого, який за значенням цільової функції був би 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**:



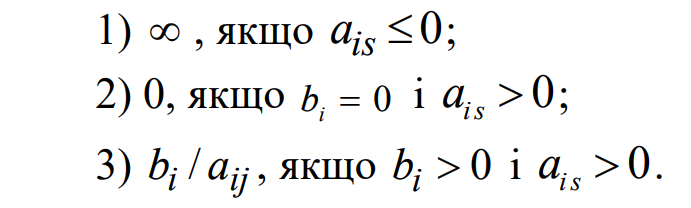
Слід мати на увазі, що всі компоненти вектора правої частини мають бути невід’ємними.

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 , для базисних 0. У першому зліва стовпці таблиці записуємо основні змінні (базис) , 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 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 = (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 Δs <0 в останньому рядку відповідає провідний стовпець s.

Обчислюємо оціночні відношення для кожного рядка за такими правилами:



Визначаємо min / {| bi /ais| }. Якщо скінченного мінімуму немає, то задача не містить скінченного оптимуму (f max = ∞) . Якщо мінімум існує, то вибираємо рядок q, на якому він досягається (будь-який, якщо їх декілька), та називаємо його провідним рядком. На перетині провідних стовпця та рядка знаходиться головний елемент a qs.

5. *Переходимо до нового опорного плану.* Базисний розв’язок можна знайти за правилом прямокутника.

*Правило прямокутника*

Заповнюємо нову симплекс таблицю за правилом:

1. у лівому стовпці записуємо новий базис: замість основної змінної xq –

змінну xs ;

1. на місці головного елемента ставимо 1, у провідному стовпчику всі

елементи, окрім головного, прирівнюємо до нуля;

1. новий рядок з номером q отримуємо із старого рядка діленням на

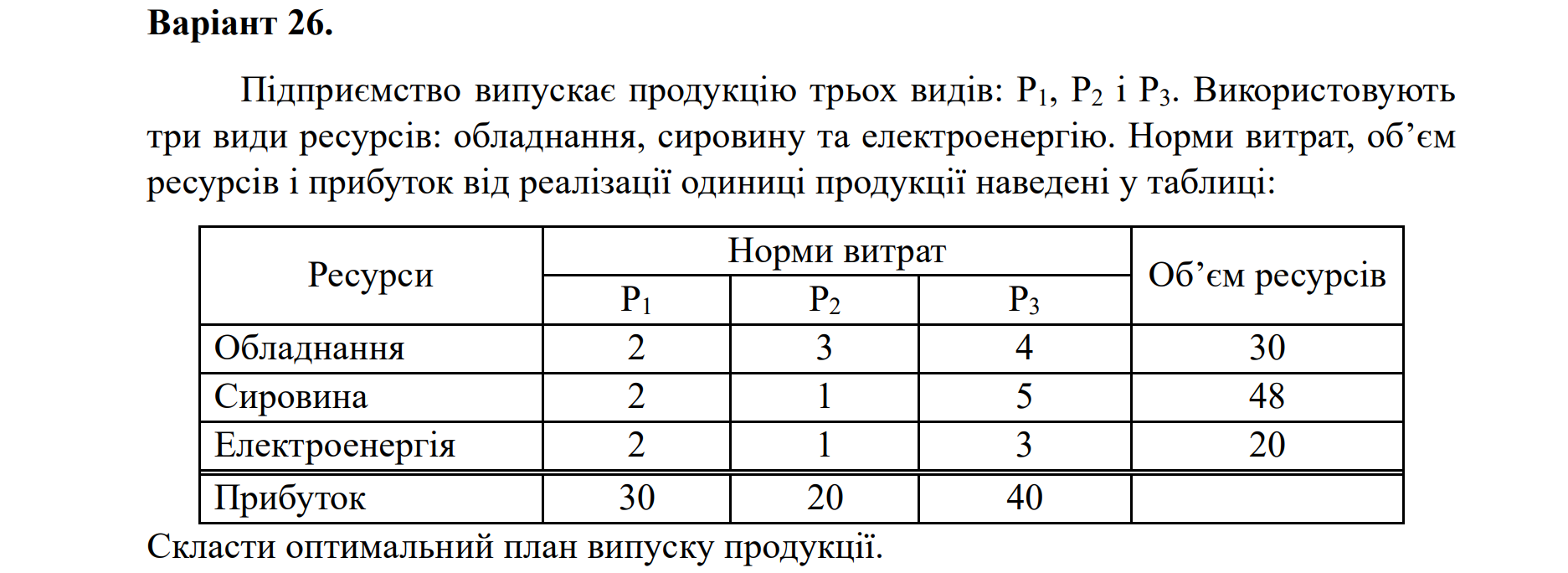
головний елемент aqs;

d) решта елементів a′ij обчислюються за правилом прямокутника

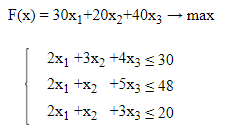
# Хід роботи

Варіант №26

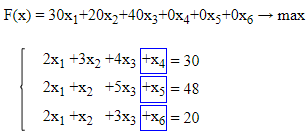
## *Завдання №1: Розв’язати задачу ЛП симплекс-методом*



1. Складемо математичну модель та запишемо систему нерівностей відповідно до обмежень:

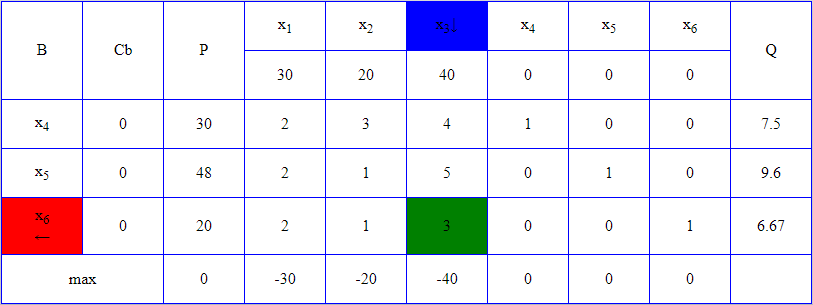


1. Введемо додаткові змінні та запишемо у канонічному вигляді:

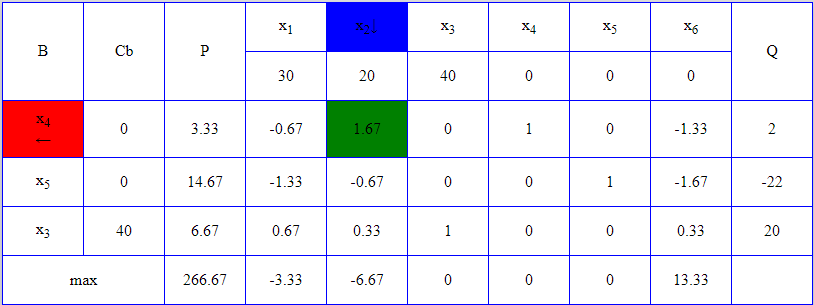


1. Заповнимо симлекс таблиці:

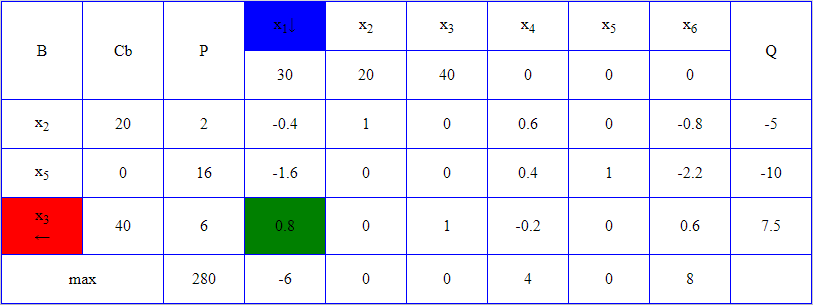
*Перша СТ*



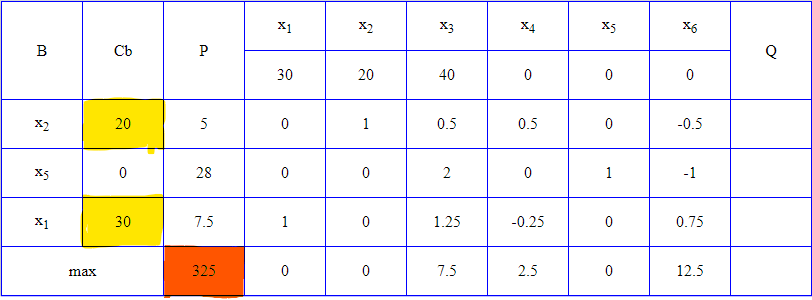
*Друга СТ*



*Третя СТ*



*Четверта СТ*

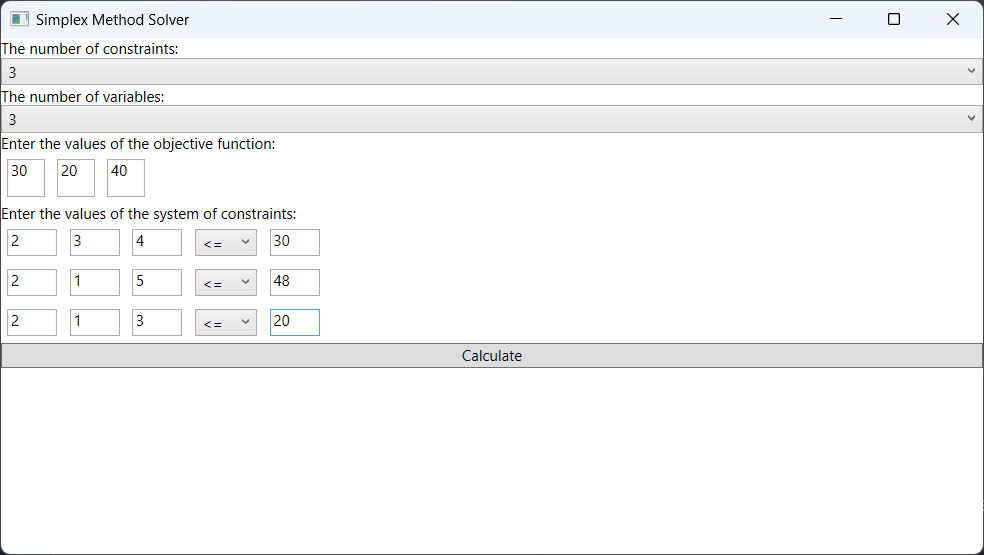


Оскільки останній рядок не містить від’ємних елементів, ми отримали оптимальний розв’язок, що при .

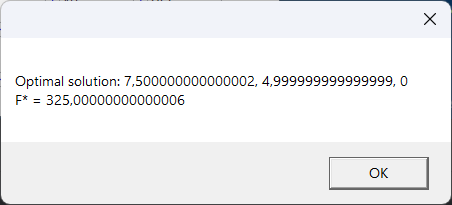
### **Код програми**

*// File: MainWindow.xaml.cs*using System;  
using System.ComponentModel;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Controls.Primitives;  
using static System.Net.Mime.MediaTypeNames;  
  
namespace Lab01\_OR  
{  
 public partial class MainWindow : Window  
 {  
 private int numberOfConstraints = 3;  
 private int numberOfVariables = 3;  
  
  
 public MainWindow()  
 {  
 InitializeComponent();  
 SetupGrid();  
 }  
  
 private void ConstraintsComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)  
 {  
 numberOfConstraints = Convert.ToInt32(((ComboBoxItem)((ComboBox)sender).SelectedItem).Content);  
 SetupGrid();  
 }  
  
 private void VariablesComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)  
 {  
 numberOfVariables = Convert.ToInt32(((ComboBoxItem)((ComboBox)sender).SelectedItem).Content);  
 SetupGrid();  
 }  
  
 private void SetupGrid()  
 {  
 *// Clear previous elements* if(ObjectiveFunctionGrid == null)  
 ObjectiveFunctionGrid = new Grid();  
  
 if (ConstraintsGrid == null)  
 ConstraintsGrid = new Grid();  
  
 ObjectiveFunctionGrid.Children.Clear();  
 ConstraintsGrid.Children.Clear();  
  
 *// Create grid for the objective function* ObjectiveFunctionGrid.ColumnDefinitions.Clear();  
 for (int i = 0; i < numberOfVariables; i++)  
 {  
 ObjectiveFunctionGrid.ColumnDefinitions.Add(new ColumnDefinition());  
  
 TextBox variableLabel = new TextBox  
 {  
 Width = 30,  
 Height = 30,  
 VerticalAlignment = VerticalAlignment.**Center**,  
 Margin = new Thickness(5)  
 };  
 ObjectiveFunctionGrid.Children.Add(variableLabel);  
 Grid.SetColumn(variableLabel, i);  
  
 }  
  
 *// Create grid for the constraints* ConstraintsGrid.RowDefinitions.Clear();  
 ConstraintsGrid.ColumnDefinitions.Clear();  
 for (int i = 0; i < numberOfConstraints; i++)  
 {  
 ConstraintsGrid.RowDefinitions.Add(new RowDefinition());  
 ConstraintsGrid.ColumnDefinitions.Add(new ColumnDefinition());  
  
 for (int j = 0; j < numberOfVariables; j++)  
 {  
 TextBox constraintTextBox = new TextBox  
 {  
 Width = 40,  
 Margin = new Thickness(5)  
 };  
 ConstraintsGrid.Children.Add(constraintTextBox);  
 Grid.SetRow(constraintTextBox, i);  
 Grid.SetColumn(constraintTextBox, j);  
 }  
  
 ConstraintsGrid.ColumnDefinitions.Add(new ColumnDefinition());  
  
 *// Add inequality selector for each row* ComboBox inequalityComboBox = new ComboBox  
 {  
 Width = 50,  
 Margin = new Thickness(5)  
 };  
 inequalityComboBox.Items.Add("<=");  
 inequalityComboBox.Items.Add(">=");  
 inequalityComboBox.Items.Add("=");  
 inequalityComboBox.SelectedIndex = 0;  
 ConstraintsGrid.Children.Add(inequalityComboBox);  
 Grid.SetRow(inequalityComboBox, i);  
 Grid.SetColumn(inequalityComboBox, numberOfVariables);  
  
 *// Add result TextBox for each row* TextBox resultTextBox = new TextBox  
 {  
 Width = 40,  
 Margin = new Thickness(5)  
 };  
 ConstraintsGrid.Children.Add(resultTextBox);  
 Grid.SetRow(resultTextBox, i);  
 Grid.SetColumn(resultTextBox, numberOfVariables + 1);  
 }  
 }  
  
 private void CalculateButton\_Click(object sender, RoutedEventArgs e)  
 {  
 *// Collect input data and call Simplex method* double[] objectiveFunction = new double[numberOfVariables];  
 for (int i = 0; i < numberOfVariables; i++)  
 {  
 objectiveFunction[i] = Convert.ToDouble(((TextBox)ObjectiveFunctionGrid.Children[i]).Text);  
 }  
  
 double[,] constraints = new double[numberOfConstraints, numberOfVariables];  
 double[] results = new double[numberOfConstraints];  
 string[] inequalities = new string[numberOfConstraints];  
  
 for (int i = 0; i < numberOfConstraints; i++)  
 {  
 for (int j = 0; j < numberOfVariables; j++)  
 {  
 constraints[i, j] = Convert.ToDouble(((TextBox)GetGridElement(ConstraintsGrid, i, j)).Text);  
 }  
  
 inequalities[i] = ((ComboBox)GetGridElement(ConstraintsGrid, i, numberOfVariables)).SelectedItem.ToString();  
 results[i] = Convert.ToDouble(((TextBox)GetGridElement(ConstraintsGrid, i, numberOfVariables + 1)).Text);  
 }  
  
 *// Call Simplex algorithm to solve* Simplex simplex = new Simplex(objectiveFunction, constraints, inequalities, results, this);  
 double[] solution = simplex.Solve();  
 double[] variables = new double[numberOfVariables];  
 for(int i = 0;i < numberOfVariables; i++)  
 {  
 variables[i] = solution[i];  
 }  
  
 *// Display the solution (can be improved to display more clearly)* MessageBox.Show("Optimal solution: " + string.Join(", ", variables) + "\nF\* = " + solution[solution.Length - 1]);  
 }  
  
 private UIElement GetGridElement(Grid grid, int row, int column)  
 {  
 foreach (UIElement element in grid.Children)  
 {  
 if (Grid.GetRow(element) == row && Grid.GetColumn(element) == column)  
 {  
 return element;  
 }  
 }  
 return null;  
 }  
 public void CreateAndAddDynamicGrid(double[,] constraints, double[] cb, Dictionary<int, double> plan, double[] deriv, double[] delta, int[] myBase)  
 {  
 *// Create a new grid* Grid dynamicGrid = new Grid  
 {  
 Margin = new Thickness(10),  
 ShowGridLines = true *// Optional: Show grid lines* };  
  
 int rows = constraints.GetLength(0) + 2;  
 int cols = constraints.GetLength(1) + 4;  
  
 *// Define rows and columns for the grid* for (int i = 0; i < rows; i++)  
 {  
 dynamicGrid.RowDefinitions.Add(new RowDefinition());  
 }  
 for (int j = 0; j < cols; j++)  
 {  
 dynamicGrid.ColumnDefinitions.Add(new ColumnDefinition());  
 }  
  
  
 CreateCell(dynamicGrid, 0, 0, "Base");  
 CreateCell(dynamicGrid, 0, 1, "Cb");  
 CreateCell(dynamicGrid, 0, 2, "Plan");  
  
 for(int i = 3; i < constraints.GetLength(1)+3; i++)  
 {  
 CreateCell(dynamicGrid, 0, i, string.Format("x" + (i - 2)));  
 }  
  
 CreateCell(dynamicGrid, 0, cols - 1, "der");  
  
 for(int i = 1; i < deriv.Length + 1; i++)  
 {  
 CreateCell(dynamicGrid, i, cols - 1, deriv[i - 1].ToString());  
 }  
  
 for(int i = 1; i < myBase.Length + 1; i++)  
 {  
 CreateCell(dynamicGrid, i, 0, string.Format("x" + myBase[i - 1]));  
 }  
  
 for(int i = 1; i < myBase.Length + 1; i++)  
 {  
 CreateCell(dynamicGrid, i, 1, string.Format(cb[i - 1].ToString()));  
 }  
  
 for(int i = 1; i < myBase.Length + 1; i++)  
 {  
 CreateCell(dynamicGrid, i, 2, string.Format(plan[myBase[i - 1]].ToString()));  
 }  
  
  
 for (int i = 1; i < constraints.GetLength(0) + 1; i++)  
 {  
 for(int j = 3; j < constraints.GetLength(1) + 3; j++)  
 {  
 CreateCell(dynamicGrid, i, j, constraints[i - 1, j -3].ToString());  
 }  
 }  
  
 CreateCell(dynamicGrid, rows - 1, 0, "F\*");  
 CreateCell(dynamicGrid, rows - 1, 1, "0");  
 CreateCell(dynamicGrid, rows - 1, 2, "0");  
  
 for(int i = 3; i < delta.Length + 3; i++)  
 {  
 CreateCell(dynamicGrid, rows - 1, i, delta[i - 3].ToString());  
 }  
  
 CreateCell(dynamicGrid, rows - 1, cols - 1, "0");  
  
 *// Add the dynamic grid to the parent container (StackPanel)* DynamicGridContainer.Children.Add(dynamicGrid);  
 }  
  
 private void CreateCell(Grid dynamicGrid, int x, int y, string text)  
 {  
 TextBox textBox = new TextBox()  
 {  
 Text = text,  
 Width = 60,  
 Height = 30,  
 Margin = new Thickness(5),  
 IsReadOnly = true *// Make the TextBox read-only to prevent editing* };  
  
 dynamicGrid.Children.Add(textBox);  
  
 Grid.SetRow(textBox, x);  
 Grid.SetColumn(textBox, y);  
 }  
 }  
  
 public class Simplex  
 {  
 private double[] \_objectiveFunction;  
 private double[,] \_constraints;  
 private string[] \_inequalities;  
 private double[] \_delta;  
 private double[] \_cb;  
 private Dictionary<int, double> \_plan;  
 private double[] \_deriv;  
 private int[] \_base;  
 MainWindow \_mainWindow;  
  
 public Simplex(double[] objectiveFunction, double[,] constraints, string[] inequalities, double[] results, MainWindow mainWindow)  
 {  
 \_mainWindow = mainWindow;  
 this.\_objectiveFunction = objectiveFunction;  
 this.\_constraints = new double[constraints.GetLength(0),constraints.GetLength(0) + objectiveFunction.Length];  
 for (int i = 0; i < constraints.GetLength(0); i++)  
 {  
 for (int j = 0; j < constraints.GetLength(0); j++)  
 {  
 \_constraints[i, j] = constraints[i, j];  
 }  
 }  
 for(int i = 0; i < constraints.GetLength(0); i++)  
 {  
 \_constraints[i, constraints.GetLength(0) + i] = 1;  
 }  
  
 this.\_delta = new double[results.Length + objectiveFunction.Length];  
 for(int i = 0;i < objectiveFunction.Length; i++)  
 {  
 \_delta[i] = -objectiveFunction[i];  
 }  
 \_deriv = new double[results.Length];  
 \_cb = new double[results.Length];  
  
 this.\_inequalities = inequalities;  
 this.\_plan = new Dictionary<int, double>();  
 this.\_base = new int[results.Length];  
 for(int i = objectiveFunction.Length; i < \_delta.Length; i++)  
 {  
 \_plan.Add(i, results[i - objectiveFunction.Length]);  
 \_base[i - objectiveFunction.Length] = i;  
 }  
 }  
  
 public double[] Solve()  
 {  
 if (CheckIfSolved())  
 {  
 double[] result = new double[\_objectiveFunction.Length + 1];  
 for (int i = 0; i < result.Length; i++)  
 {  
 result[i] = \_plan.GetValueOrDefault(i);  
 }  
  
 for(int i = 0; i < \_base.Length; i++)  
 {  
 result[result.Length - 1] += result[i] \* \_objectiveFunction[i];  
 }   
  
 return result;  
 }  
  
 double min = 0;  
 int minIndexColumn = 0;  
 for(int i = 0; i < \_delta.Length; i++)  
 {  
 if (\_delta[i] < min)  
 {  
 min = \_delta[i];  
 minIndexColumn = i;  
 }  
 }  
  
 for(int i = 0; i < \_deriv.Length; i++)  
 {  
 \_deriv[i] = \_plan.GetValueOrDefault(\_base[i]) / \_constraints[i, minIndexColumn];  
 }  
  
 min = Double.**MaxValue**;  
 int minIndexRow = 0;  
 for (int i = 0; i < \_deriv.Length; i++)  
 {  
 if (\_deriv[i] < min && \_deriv[i] > 0)  
 {  
 min = \_deriv[i];  
 minIndexRow = i;  
 }  
 }  
  
 \_mainWindow.CreateAndAddDynamicGrid(\_constraints, \_cb, \_plan, \_deriv, \_delta, \_base);  
  
 RebuildTable(minIndexRow, minIndexColumn);  
 FindDelta();  
  
 return Solve();  
 }  
  
 private void FindDelta()  
 {  
 double[] newDelta = new double[\_delta.Length];  
  
 for (int i = 0; i < newDelta.Length; i++)  
 {  
 double localDelta = 0;  
 for (int j = 0; j < \_base.Length; j++)  
 {  
 localDelta += \_cb[j] \* \_constraints[j, i];  
 }  
 if (i >= \_objectiveFunction.Length)  
 {  
 newDelta[i] = localDelta;  
 }  
 else  
 {  
 newDelta[i] = localDelta - \_objectiveFunction[i];  
 }  
 }  
  
 \_delta = newDelta;  
 }  
  
 private void RebuildTable(int minIndexRow, int minIndexColumn)  
 {  
 double rate = \_constraints[minIndexRow, minIndexColumn];  
  
 if(minIndexColumn < \_objectiveFunction.Length)  
 {  
 \_cb[minIndexRow] = \_objectiveFunction[minIndexColumn];  
 }  
  
 Dictionary<int, double> newPlan = \_plan.ToDictionary();  
 double[,] newConstraints = new double[\_constraints.GetLength(0), \_constraints.GetLength(1)];  
   
 newPlan.Add(minIndexColumn, \_plan[\_base[minIndexRow]] / rate);  
 newPlan.Remove(\_base[minIndexRow]);  
  
 for(int i = 0; i < \_delta.Length; i++)  
 {  
 newConstraints[minIndexRow, i] = \_constraints[minIndexRow, i] / rate;  
 }  
  
 for(int i = 0; i <\_base.Length; i++)  
 {  
 if (i != minIndexRow)  
 {  
 newPlan[\_base[i]] = \_plan[\_base[i]] - \_constraints[i, minIndexColumn] \* \_plan[\_base[minIndexRow]] / rate;  
   
 for (int j = 0; j < \_delta.Length; j++)  
 {  
 if (j != minIndexColumn)  
 {  
 newConstraints[i, j] = \_constraints[i, j] - \_constraints[minIndexRow, j] \* \_constraints[i, minIndexColumn] / rate;  
 }  
 else  
 {  
 newConstraints[i,j] = 0;  
 }  
 }  
 }  
 }  
  
 \_constraints = newConstraints;  
 \_plan = newPlan;  
 \_base[minIndexRow] = minIndexColumn;  
  
 }  
  
 private bool CheckIfSolved()  
 {  
 return !\_delta.Any(x => x < 0);  
 }  
 }  
}

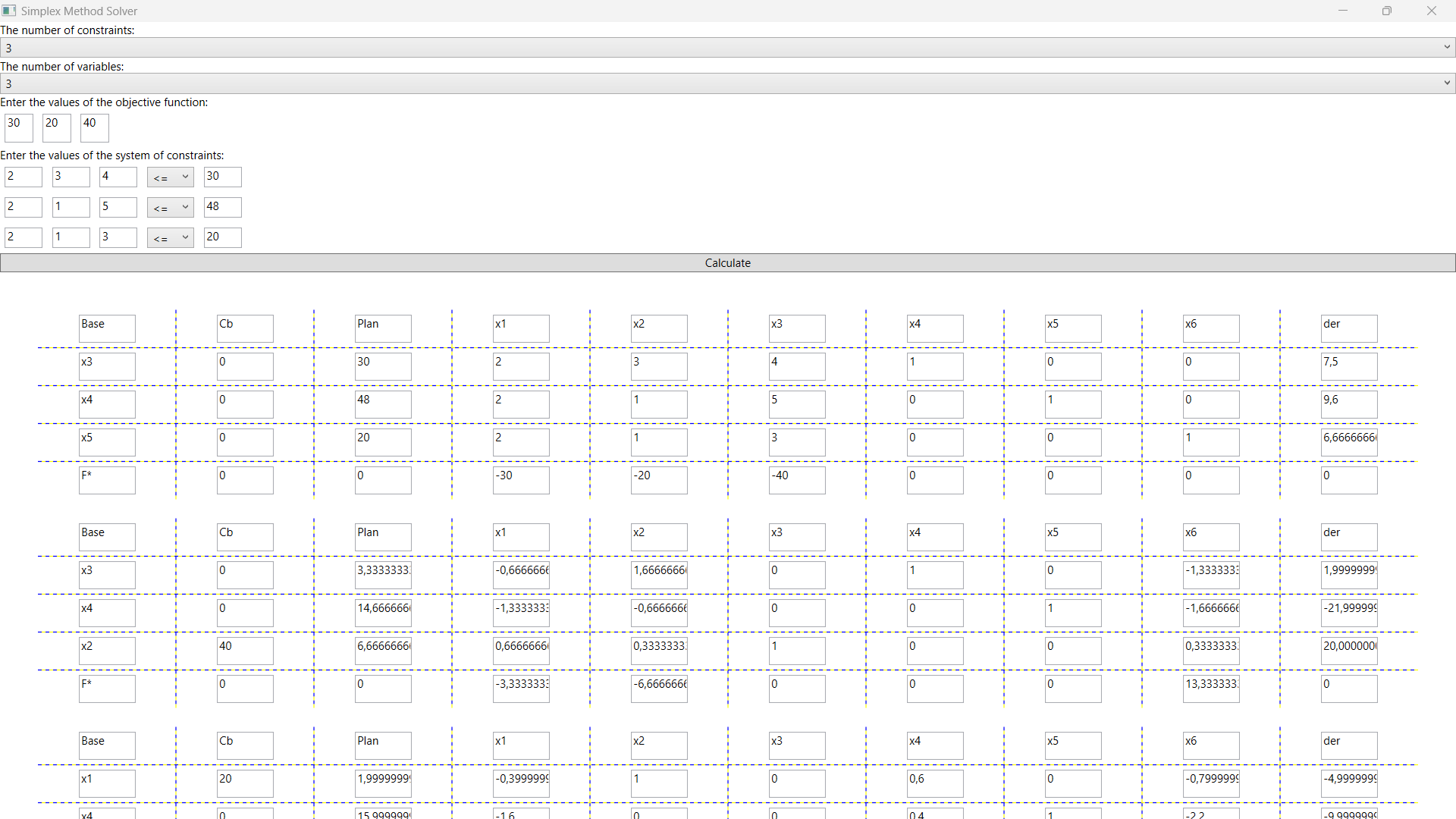
### **Хід виконання**



*Рис. 1. Інтерфейс програми із заданими вхідними даними*

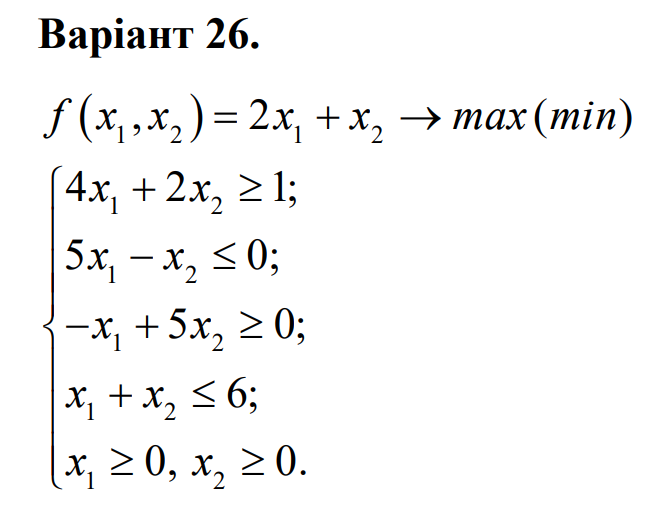
**

*Рис. 2. Результат виконання програми*

**

*Рис. 3. Виведення таблиць*

## *Завдання №2: Розв’язати задачу ЛП графічним методом (знайти максимальне та мінімальне значення функції та значення змінних, при якому вони досягаються)*



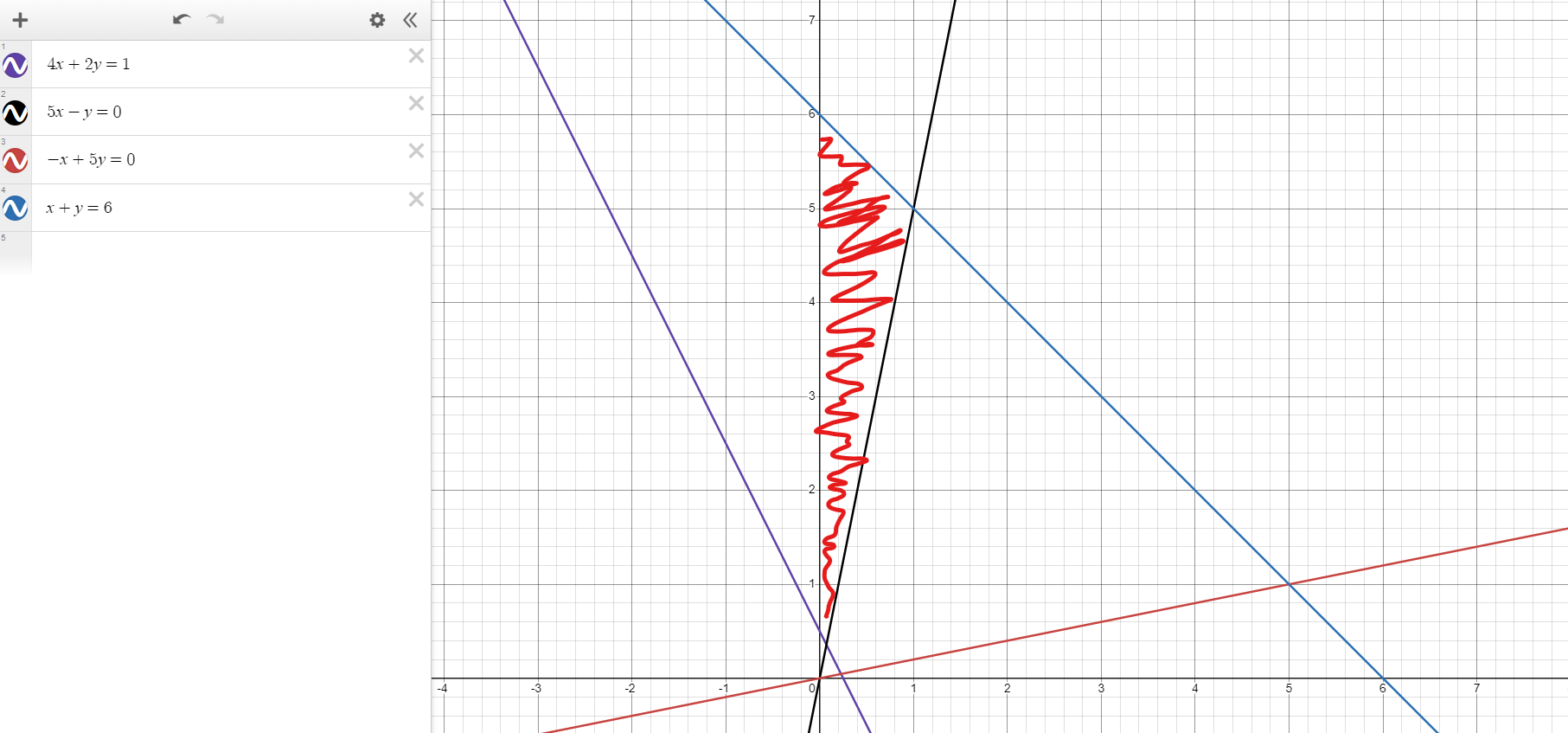
1. Знайдемо область значень, що задовольняють обмеження:

|  |  |
| --- | --- |
|  | |
|  |  |
| 1 | 1 |
|  | |

|  |  |
| --- | --- |
|  | |
|  |  |
| 0 | 0 |
|  | |

|  |  |
| --- | --- |
|  | |
|  |  |
| 0 | 0 |
|  | |

|  |  |
| --- | --- |
|  | |
|  |  |
| 1 | 1 |
|  | |



1. Оскільки функція досягає своїх граничних значень на екстремумах (вершинах отриманого багатокутника) знайдемо їх значення і визначимо min та max заданої функції .

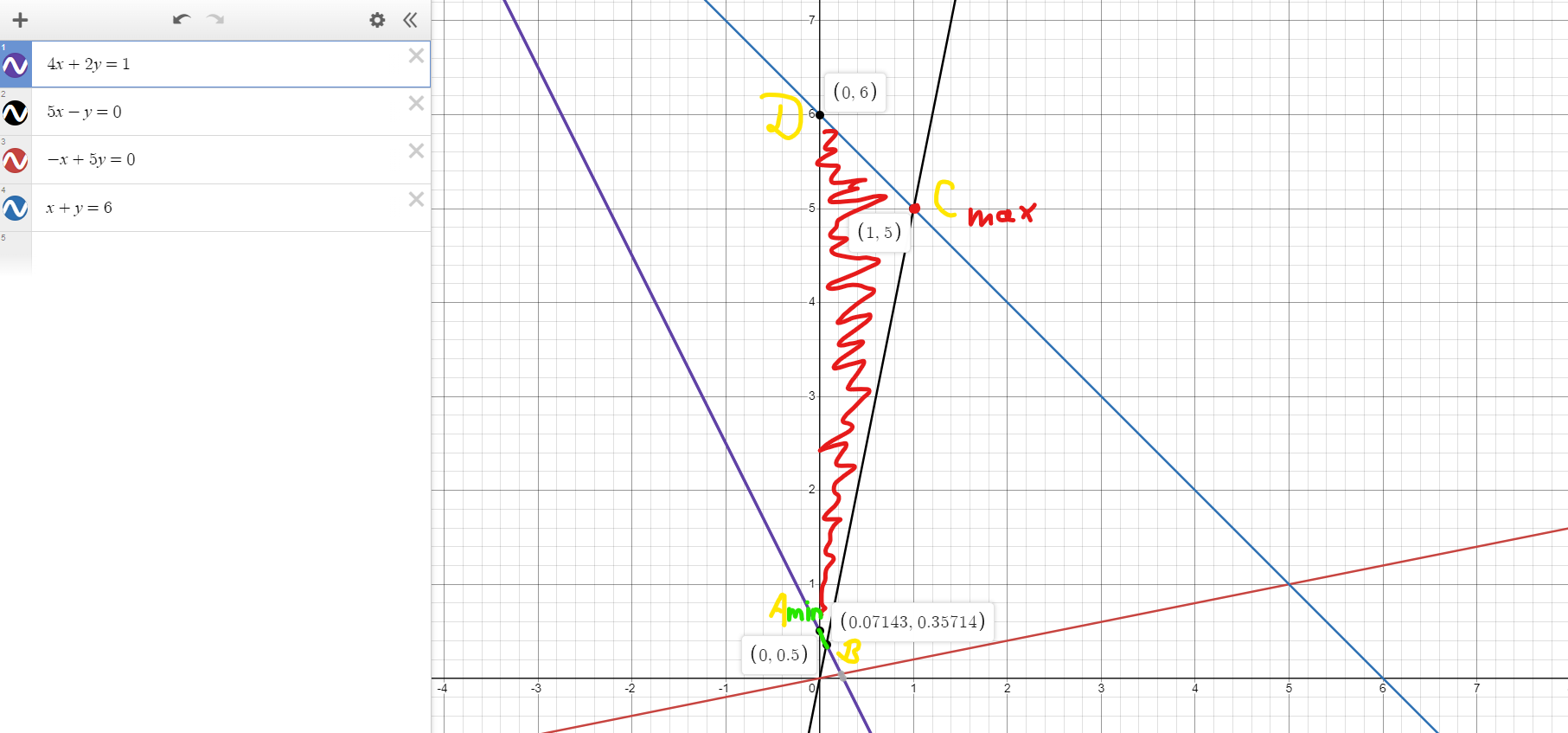
A(0; 0,5)

B(1/14; 5/14)

C(1; 5)

D(0; 6)

Звідси маємо:



**Висновок:** Під час виконання лабораторної роботи я ознайомився на практиці із основними поняттями теорії лінійного програмування (ЛП), набув навичок практичного розв’язання задач ЛП табличним симплекс-методом (СМ) та його програмною реалізацією, навчився розв’язувати задачі ЛП із двома змінними графічним методом.