МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 1
На тему: *"Розв'язування задачі лінійного програмування із двома змінними графічно та за симплекс-методом"*
З дисципліни: *"Дослідження операцій"*

Лектор:
Доктор технічних наук
Журавчак Л. М.

Виконав:
студент групи ПЗ-22
Павлів М. Я.

Прийняв:
Кандидат ф.-м. наук
Івасько Н. М.

«__»_____ 2022 р.

∑= ____ .....

Львів – 2022

Тема роботи: Розв'язування задачі лінійного програмування із двома змінними графічно та за симплекс-методом.

Мета роботи: Ознайомитись на практиці із основними поняттями теорії лінійного програмування, навчитись знаходити оптимальні плани задач лінійного програмування графічно, за допомогою симплекс методу.

Лабораторне завдання

1. Отримати індивідуальний варіант завдання.
2. Записати математичну модель задачі ЛП та зазначити економічний зміст цільової функції і системи обмежень. Написати програму розв'язування та з її допомогою знайти розв'язок (максимальне (мінімальне) значення функції та значення змінних, при якому воно досягається) задачі ЛП згідно з варіантом з Додатка № 1 до лабораторної роботи № 1.
3. Розв'язати задачу ЛП з Додатка № 2 графічним методом (знайти максимальне та мінімальне значення функції та значення змінних, при якому вони досягаються).
4. Оформити звіт про виконану роботу.
5. Продемонструвати викладачеві результати, відповісти на запитання стосовно виконання роботи.

Індивідуальне завдання

14. Підприємство для виготовлення чотирьох видів продукції використовує токарне, фрезерне, свердлильне, розточувальне і шліфувальне устаткування, а також комплектуючі вироби, збирання яких потребує виконання певних складально-налагоджувальних робіт. Норми витрат усіх видів ресурсів на виготовлення кожного виду виробу, а також наявний фонд кожного з ресурсів, прибуток від реалізації одиниці продукції певного виду наведені в таблиці:

Ресурси	Норми затрат на виготовлення одного виробу				Обсяг ресурсів
	1	2	3	4	
Продуктивність верстатів, людино-год:					
токарного	50	–	62	–	6 427
фрезерного	4	3	2	2	480
свердлильного	9	11	15	5	2 236
розточувального	16	9	16	13	2 624
шліфувального	–	16	3	5	780
Комплектуючі вироби, шт.	3	4	3	3	520
Складально-налагоджувальні роботи, людино-год	4,5	4,5	4,5	4,5	720
Прибуток, грн/шт.	315	278	573	370	–

Знайти оптимальну програму випуску продукції, яка дасть максимальний прибуток від реалізації продукції.

14	$F = x_1 + 2x_2 \to \max;$ $x_1 + x_2 \leqslant 4;$ $3x_1 + x_2 \geqslant 4;$ $x_1 + 5x_2 \geqslant 4,$ $0 \leqslant x_1 \leqslant 3,$ $0 \leqslant x_2 \leqslant 3.$

Теоретичні відомості

14. Що таке провідний рядок (стовпець) симплекс таблиці?

Провідний стовпець – стовпець, який відповідає найбільшому за модулем від'ємному коефіцієнту Δ_s<0. Провідний рядок – рядок, на якому досягається мінімум $\min_i\{|b_i/a_{is}|\}$.

18. Чи може задача ЛП мати більше, ніж один оптимальний розв’язок?

Задача ЛП може мати єдиний оптимальний план або нескінченну множину оптимальних планів, а може і не мати жодного.

2. Що таке цільова функція?

Це функція, яка моделює поставлену в задачі мету.

Хід роботи

1. Записав математичну модель задачі ЛП: F(x)=315x1+278x2+573x3+370x4->.
   Економічний зміст: максимізувати прибуотк від реалізації продукції.
   Записав систему обмежень:
   50x1+0x2+62x3+0x4<6427
   4x1+3x2+2x3+2x4<=480
   9x1+11x2+15x3+5x4<=2236
   16x1+9x2+16x3+13x4<=2624
   0x1+16x2+3x3+5x4<=520
   3x1+4x2+3x3+3x4<=520
   4.5x1+4.5x1+4.5x1<=720
2. Написав програму розв’язування задачі симплекс методом:

IReader.cs

```
namespace Lab1
{
    public interface IReader
    {
        string ReadLine(string prompt = "");
    }

    public class ConsoleReader : IReader
    {
        public string ReadLine(string prompt = "")
        {
            Console.Write(prompt);

            return Console.ReadLine()!;
        }
    }
}
```

SimplexFunction.cs

```
using System.Text;

namespace Lab1
{
    public class SimplexFunction
    {
        private readonly double[] _coefficients;
        public bool IsMaximize { get; private set; }
        public int VarCount => _coefficients.Length;
        public double this[int index] => _coefficients[index];

        public SimplexFunction(IEnumerable<double> coefficients, bool isMaximize)
        {
            _coefficients = coefficients.ToArray();

            IsMaximize = isMaximize;
        }

        public void Canonize()
        {
            if (IsMaximize)
            {
                return;
```

```
            }

            Invert();

            IsMaximize = true;
        }

        public void Invert()
        {
            for (var i = 0; i < VarCount; i++)
            {
                _coefficients[i] *= -1;
            }
        }

        public override string ToString()
        {
            if (VarCount == 0)
            {
                return "";
            }

            var sb = new StringBuilder().Append($"F(x) = {_coefficients[0]}x1");

            for (var i = 1; i < VarCount; i++)
            {
                bool usePlus = _coefficients[i] >= 0;

                var s = usePlus ? $"+{_coefficients[i]}x{i + 1}" :
$"{_coefficients[i]}x{i + 1}";

                sb.Append(s);
            }

            var extr = IsMaximize ? " -> max" : " -> min";

            return sb.Append(extr).ToString();
        }
    }
}
```

SimplexConstraint.cs

```
using System.Text;

namespace Lab1
{
    public class SimplexConstraint
    {
        private static readonly string[] s_allowedSigns = new[] {"<=", ">=", "="};

        private double[] _coefficients;
        private string _sign;
        private double _value;

        public string Sign => _sign;
        public double Value => _value;
        public int Count => _coefficients.Length;
        public double this[int index] => _coefficients[index];

        public SimplexConstraint(IEnumerable<double> coefficients, string sign, double
value)
        {
            _coefficients = coefficients.ToArray();
            _value = value;

            if (!s_allowedSigns.Contains(sign.Trim()))
```

```
            {
                throw new ArgumentException();
            }

            _sign = sign.Trim();
        }

        public override string ToString()
        {
            if (Count == 0)
            {
                return "";
            }

            var sb = new StringBuilder().Append($"{_coefficients[0]}x1");

            for (var i = 1; i < Count; i++)
            {
                bool usePlus = _coefficients[i] >= 0;

                var s = usePlus ? $"+{_coefficients[i]}x{i + 1}" :
$"{_coefficients[i]}x{i + 1}";

                sb.Append(s);
            }

            return sb.Append($"{_sign}{_value}").ToString();
        }
    }
}
```

SimplexProcessor.cs

```
using System.Text.RegularExpressions;

namespace Lab1
{
    public class SimplexProcessor
    {
        private static readonly string s_doublePattern = @"^[-]?(([1-9][0-9]*\.?[0-
9]*)|([0]\.[0-9]*)|([0]))$";
        private static readonly string s_intPattern = @"^[1-9][0-9]{0,7}$";

        private readonly IReader _reader;
        private int _varCount;


        public SimplexProcessor(IReader reader)
        {
            _reader = reader;
        }

        public Simplex Get(Action<string, bool>? outputProvider = null)
        {
            SimplexFunction function = GetFunction();
            SimplexConstraint[] constraints = GetConstraints().ToArray();

            return new Simplex(function, constraints, outputProvider);
        }

        private SimplexFunction GetFunction()
        {
            string s = ReadUntilMatch(s_intPattern, "Enter count of variables: ");

            _varCount = Int32.Parse(s);
            var coefficients = new double[_varCount];
```

```
            for (var i = 0; i < _varCount; i++)
            {
                s = ReadUntilMatch(s_doublePattern, $"x{i + 1} * ");

                coefficients[i] = Double.Parse(s);
            }

            s = ReadUntilMatch(@"^(min|max)$", "Min / Max ? :");

            return new SimplexFunction(
                coefficients,
                s.Contains("max", StringComparison.InvariantCultureIgnoreCase)
            );
        }

        private IEnumerable<SimplexConstraint> GetConstraints()
        {
            string s = ReadUntilMatch(s_intPattern, "Enter count of rows: ");

            var rowCount = Int32.Parse(s);
            var constraints = new SimplexConstraint[rowCount];

            for (var i = 0; i < rowCount; i++)
            {
                constraints[i] = GetConstraint(i);
            }

            return constraints;
        }

        private SimplexConstraint GetConstraint(int index)
        {
            string s;

            var coefficients = new double[_varCount];

            for (var i = 0; i < _varCount; ++i)
            {
                s = ReadUntilMatch(s_doublePattern, $"x{i + 1} * ");

                coefficients[i] = Double.Parse(s);
            }

            string sign = ReadUntilMatch(@"^(<=|>=|=)$", "Sign (<=|>=|=): ");

            s = ReadUntilMatch(s_doublePattern, $"b{index + 1} = ");

            var value = Double.Parse(s);

            return new SimplexConstraint(coefficients, sign, value);
        }

        private string ReadUntilMatch(string pattern, string prompt = "")
        {
            string s;
            do
            {
                s = _reader.ReadLine(prompt);
            } while (Regex.IsMatch(s, pattern) == false);

            return s;
        }
    }
}
```

Simplex.cs

```
using System.Text;

namespace Lab1
{
    public class Simplex
    {
        private readonly int _rowCount, _columnCount;
        private readonly double[,] _systemMatrix;
        private readonly double[] _basis;
        private readonly double[] _functionCoefficients;
        private readonly double[] _initialCoefficients;
        private readonly List<int> _basisIndices = new();
        private double Q;
        private bool _isUnbounded;
        private readonly Action<string, bool>? _outputProvider;

        public Simplex(
            SimplexFunction function,
            SimplexConstraint[] constraints,
            Action<string, bool>? outputProvider = null
        )
        {
            if (constraints.GroupBy(constraint => constraint.Count).First().Count() !=
constraints.Length)
            {
                throw new Exception();
            }

            if (function.VarCount != constraints.First().Count)
            {
                throw new Exception();
            }

            _outputProvider = outputProvider;

            PrintInitial(function, constraints);
            Canonize(function, constraints);
            PrintInitial(function, constraints);
            function.Invert();
            Q = 0;
            _isUnbounded = false;
            _functionCoefficients = new double[function.VarCount];
            _initialCoefficients = new double[function.VarCount];
            _rowCount = constraints.Length;
            _columnCount = constraints.First().Count;
            _systemMatrix = new double[_rowCount, _columnCount];
            _basis = new double[_rowCount];
            int varCount = _initialCoefficients.Length;

            for (var i = 0; i < function.VarCount; ++i)
            {
                _functionCoefficients[i] = function[i];
                _initialCoefficients[i] = -function[i];
            }

            for (var i = 0; i < _rowCount; ++i)
            {
                for (var j = 0; j < _columnCount; ++j)
                {
                    _systemMatrix[i, j] = constraints[i][j];
                }

                _basis[i] = constraints[i].Value;
            }
```

```
            Array.Resize(ref _functionCoefficients, _columnCount);
            _basisIndices.AddRange(Enumerable.Range(varCount, _columnCount));
        }

        private void Canonize(SimplexFunction function, SimplexConstraint[] constraints)
        {
            function.Canonize();

            var additionalVarCount = constraints.Count(constraint => constraint.Sign !=
"=");
            var k = 0;
            for (var i = 0; i < constraints.Length; ++i)
            {
                var constraint = constraints[i];
                var newCoefficients = new double[constraint.Count + additionalVarCount];

                for (var j = 0; j < constraint.Count; j++)
                {
                    newCoefficients[j] = constraint[j];
                }

                double value = constraint.Value;
                string newSign = constraint.Sign;

                if (value < 0)
                {
                    value = -value;
                    for (var j = 0; j < constraint.Count; ++j)
                    {
                        newCoefficients[j] *= -1;
                    }

                    newSign = newSign switch
                    {
                        ">=" => "<=",
                        "<=" => "=>",
                        _ => "="
                    };
                }

                if (newSign == "<=")
                {
                    newCoefficients[constraint.Count + k++] = 1;
                }
                else if (newSign == ">=")
                {
                    newCoefficients[constraint.Count + k++] = -1;
                }

                newSign = "=";
                constraints[i] = new SimplexConstraint(newCoefficients, newSign, value);
            }
        }

        public Tuple<IEnumerable<double>, double> Solve()
        {
            var solutionFound = false;

            PrintStep();
            while (solutionFound == false)
            {
                solutionFound = NextStep();
                PrintStep();
            }
```

```
        var roots = new double[_initialCoefficients.Length];
        for (var i = 0; i < roots.Length; ++i)
        {
            var countOf0 = 0;
            var indexOfRoot = 0;
            for (var j = 0; j < _rowCount; ++j)
            {
                if (_systemMatrix[j, i] == 0)
                {
                    countOf0++;
                }
                else if (Math.Abs(_systemMatrix[j, i] - 1) < 1e-8)
                {
                    indexOfRoot = j;
                }
            }

            roots[i] = countOf0 == _rowCount - 1 ? _basis[indexOfRoot] : 0;
        }

        return Tuple.Create<IEnumerable<double>, double>(roots, Q);
    }
    private bool NextStep()
    {
        if (IsOptimal())
        {
            return true;
        }

        var baseColumn = GetBaseColumn();

        if (_isUnbounded)
        {
            _outputProvider?.Invoke("Error: unbounded.", true);
            return true;
        }

        var baseRow = GetBaseRow(baseColumn);

        NextCalculation(baseRow, baseColumn);

        return false;
    }

    private void NextCalculation(int baseRow, int baseColumn)
    {
        _basisIndices[baseRow] = baseColumn;
        double baseElement = _systemMatrix[baseRow, baseColumn];

        var baseRowValues = new double[_columnCount];
        var baseColumnValues = new double[_rowCount];
        var newRow = new double[_columnCount];

        Q -= _functionCoefficients[baseColumn] * (_basis[baseRow] / baseElement);

        for (var i = 0; i < _columnCount; ++i)
        {
            baseRowValues[i] = _systemMatrix[baseRow, i];
            newRow[i] = baseRowValues[i] / baseElement;
        }

        for (var i = 0; i < _rowCount; ++i)
        {
            baseColumnValues[i] = _systemMatrix[i, baseColumn];
```

```
        }

        _basis[baseRow] /= baseElement;

        for (var i = 0; i < _rowCount; ++i)
        {
            if (i == baseRow)
            {
                continue;
            }

            for (var j = 0; j < _columnCount; ++j)
            {
                _systemMatrix[i, j] -= baseColumnValues[i] * newRow[j];
            }
        }

        for (var i = 0; i < _basis.Length; ++i)
        {
            if (i == baseRow)
            {
                continue;
            }

            _basis[i] -= baseColumnValues[i] * _basis[baseRow];
        }

        double multiplyValue = _functionCoefficients[baseColumn];
        for (var i = 0; i < _functionCoefficients.Length; ++i)
        {
            _functionCoefficients[i] -= multiplyValue * newRow[i];
        }

        for (var i = 0; i < _columnCount; ++i)
        {
            _systemMatrix[baseRow, i] = newRow[i];
        }
    }

    private int GetBaseRow(int pivotColumn)
    {
        var positiveValues = new double[_rowCount];
        var result = new double[_rowCount];

        var negativeValueCount = 0;
        for (var i = 0; i < _rowCount; ++i)
        {
            if (_systemMatrix[i, pivotColumn] > 0)
            {
                positiveValues[i] = _systemMatrix[i, pivotColumn];
                continue;
            }

            positiveValues[i] = 0;
            negativeValueCount++;
        }

        if (negativeValueCount == _rowCount)
        {
            _isUnbounded = true;
        }
        else
        {
            for (var i = 0; i < _rowCount; ++i)
            {
```

```
            var value = positiveValues[i];
            if (value > 0)
            {
                result[i] = _basis[i] / value;
            }
            else
            {
                result[i] = 0;
            }
        }
    }

    var min = double.MaxValue;
    var baseRow = 0;
    for (var i = 0; i < result.Length; ++i)
    {
        if (!(result[i] > 0) || !(result[i] < min))
        {
            continue;
        }

        min = result[i];
        baseRow = i;
    }

    return baseRow;
}

private int GetBaseColumn()
{
    var baseColumn = 0;
    var min = _functionCoefficients[0];

    for (var i = 1; i < _functionCoefficients.Length; ++i)
    {
        if (!(_functionCoefficients[i] < min))
        {
            continue;
        }

        min = _functionCoefficients[i];
        baseColumn = i;
    }

    return baseColumn;
}

private bool IsOptimal()
{
    var isOptimal = false;
    var positiveValueCount = _functionCoefficients.Count(value => value >= 0);

    if (positiveValueCount != _functionCoefficients.Length)
    {
        return isOptimal;
    }

    isOptimal = true;

    return isOptimal;
}

private void PrintStep()
{
    if (_outputProvider == null)
```

```
            {
                return;
            }

            const int columnWidth = 10;
            var varCount = _columnCount;

            var sb = new StringBuilder();

            string firstLine = $"|{"xb",-columnWidth}|{"cb",-columnWidth}|{"P0",-
columnWidth}|";


            for (var i = 0; i < varCount; ++i)
            {
                if (i < _initialCoefficients.Length)
                {
                    firstLine += $"{$"c{i + 1}={_initialCoefficients[i]:F3}",-
columnWidth}|";
                }
                else
                {
                    firstLine += $"{$"c{i + 1}=0",-columnWidth}|";
                }
            }

            var emptyLine = new string('-', firstLine.Length);
            sb.AppendLine(emptyLine).AppendLine(firstLine).AppendLine(emptyLine);

            for (var i = 0; i < _rowCount; ++i)
            {
                var c = _basisIndices[i] >= _initialCoefficients.Length
                    ? 0
                    : _initialCoefficients[_basisIndices[i]];

                sb
                    .Append($"|{$"x{_basisIndices[i] + 1}",-columnWidth}|")
                    .Append($"{$"{c:F3}",-columnWidth}|")
                    .Append($"{$"{_basis[i]:F3}",-columnWidth}|");
                for (var j = 0; j < varCount; ++j)
                {
                    sb.Append($"{$"{_systemMatrix[i, j]:F3}",-columnWidth}|");
                }

                sb.AppendLine().AppendLine(emptyLine);
            }

            sb
                .Append($"|{$"Q",-columnWidth}|")
                .Append($"{$"=",-columnWidth}|")
                .Append($"{Q,-columnWidth:F3}|");

            for (var i = 0; i < varCount; ++i)
            {
                if (i < _functionCoefficients.Length)
                {
                    sb.Append($"{$"{_functionCoefficients[i]:F3}",-columnWidth}|");
                }
                else
                {
                    sb.Append($"{"0.0000",-columnWidth}|");
                }
            }

            sb.AppendLine().AppendLine(emptyLine);
```

```
            _outputProvider.Invoke(sb.ToString(), true);
            _outputProvider.Invoke("", true);
        }

        private void PrintInitial(SimplexFunction function, SimplexConstraint[]
constraints)
        {
            if (_outputProvider == null)
            {
                return;
            }

            var sConstraints = String.Join<SimplexConstraint>("\n", constraints);

            _outputProvider.Invoke(sConstraints, true);
            _outputProvider.Invoke("", true);
            _outputProvider.Invoke(function.ToString(), true);
        }
    }
}
```

Program.cs

```
using Lab1;

IReader reader = new ConsoleReader();
var processor = new SimplexProcessor(reader);

Simplex simplex = processor.Get((str, insertLine) =>
{
    if (insertLine)
    {
        Console.WriteLine(str);
        return;
    }

    Console.Write(str);
});

Tuple<IEnumerable<double>, double> result = simplex.Solve();
Console.WriteLine($"Roots: {String.Join(", ", result.Item1)}");
Console.WriteLine($"F(x) = {result.Item2}");
```

3. Згідно з програмою отримав такі розв'язки: x1=0, x2=0, x3=103.661, x4=56.339. F(x)= 80243.242.

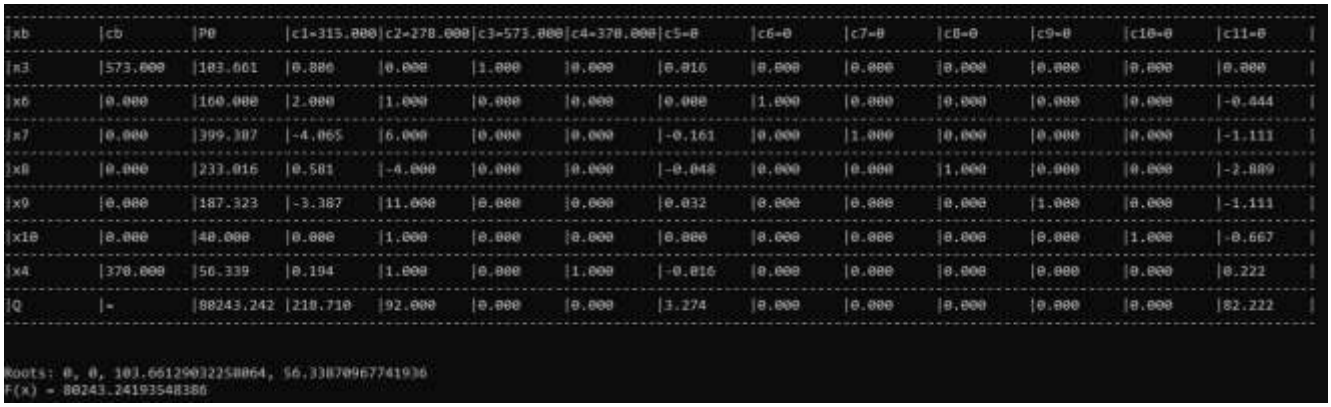

Рис. 1. Результат виконання програми

4. Розв’язав задачу ЛП з Додатка №2 графічним методом:

Множина розв’язків задачі позначена червоним заштрихованим чотирикутником:

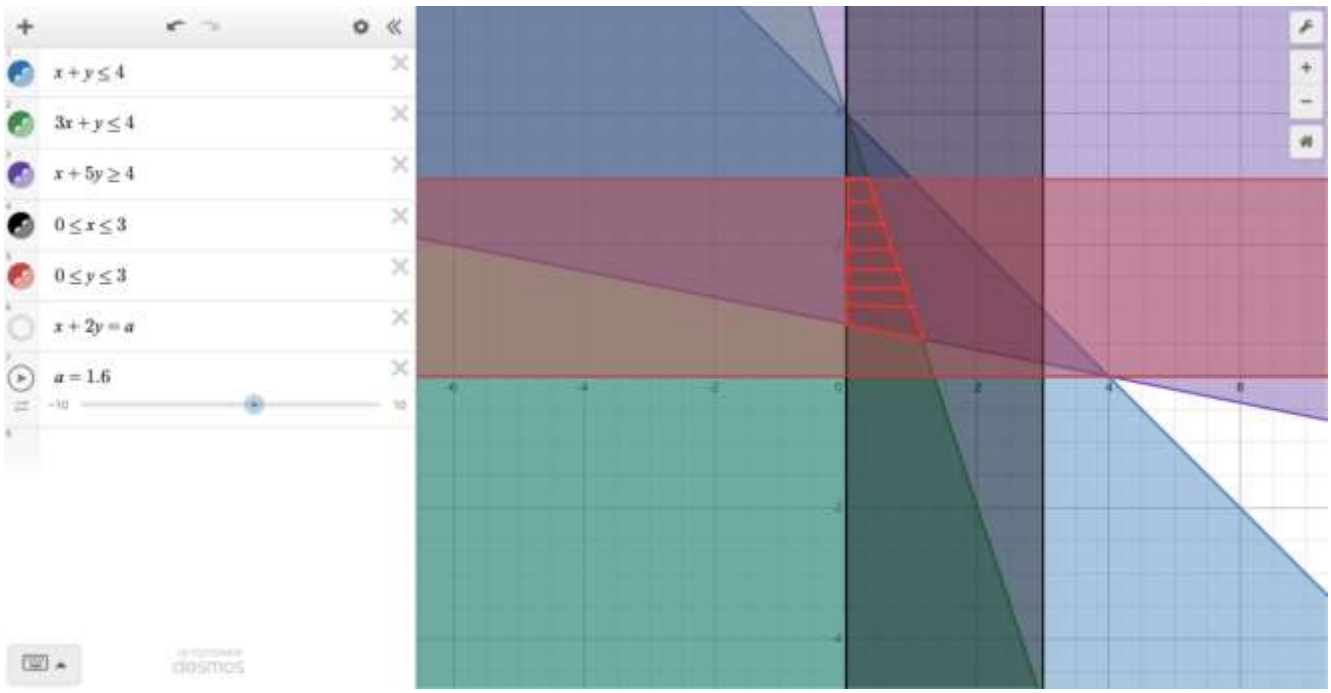

Рис. 2. Множина припустимих розв'язків.

Мінімуму функція досягає в точці (0, 0.8) і набуває значення 1.6:

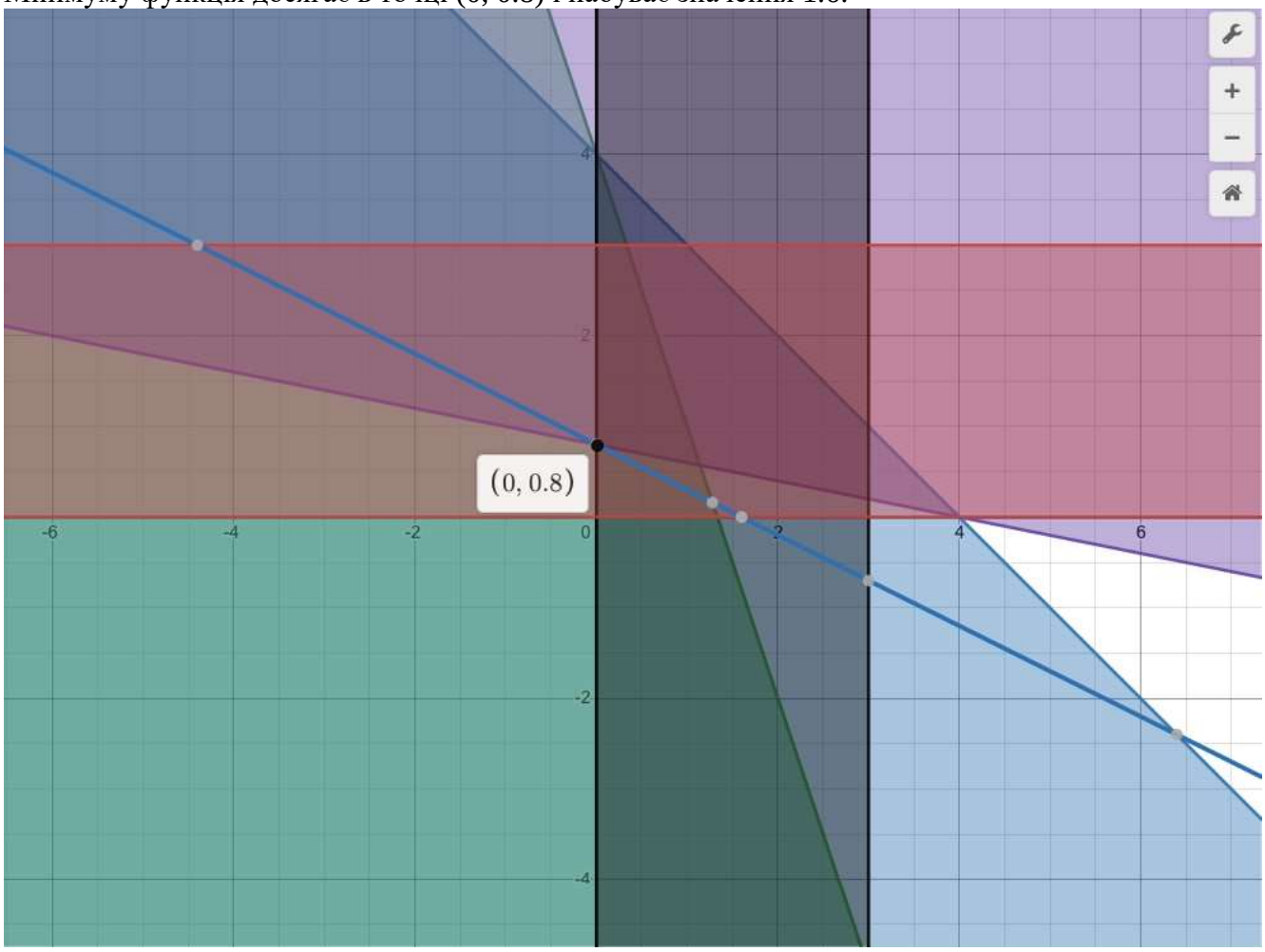

Рис. 3. Мінімум функції.

Максимум функція набуває в точці (0.35, 3) і набуває при цьому значення 6.35.

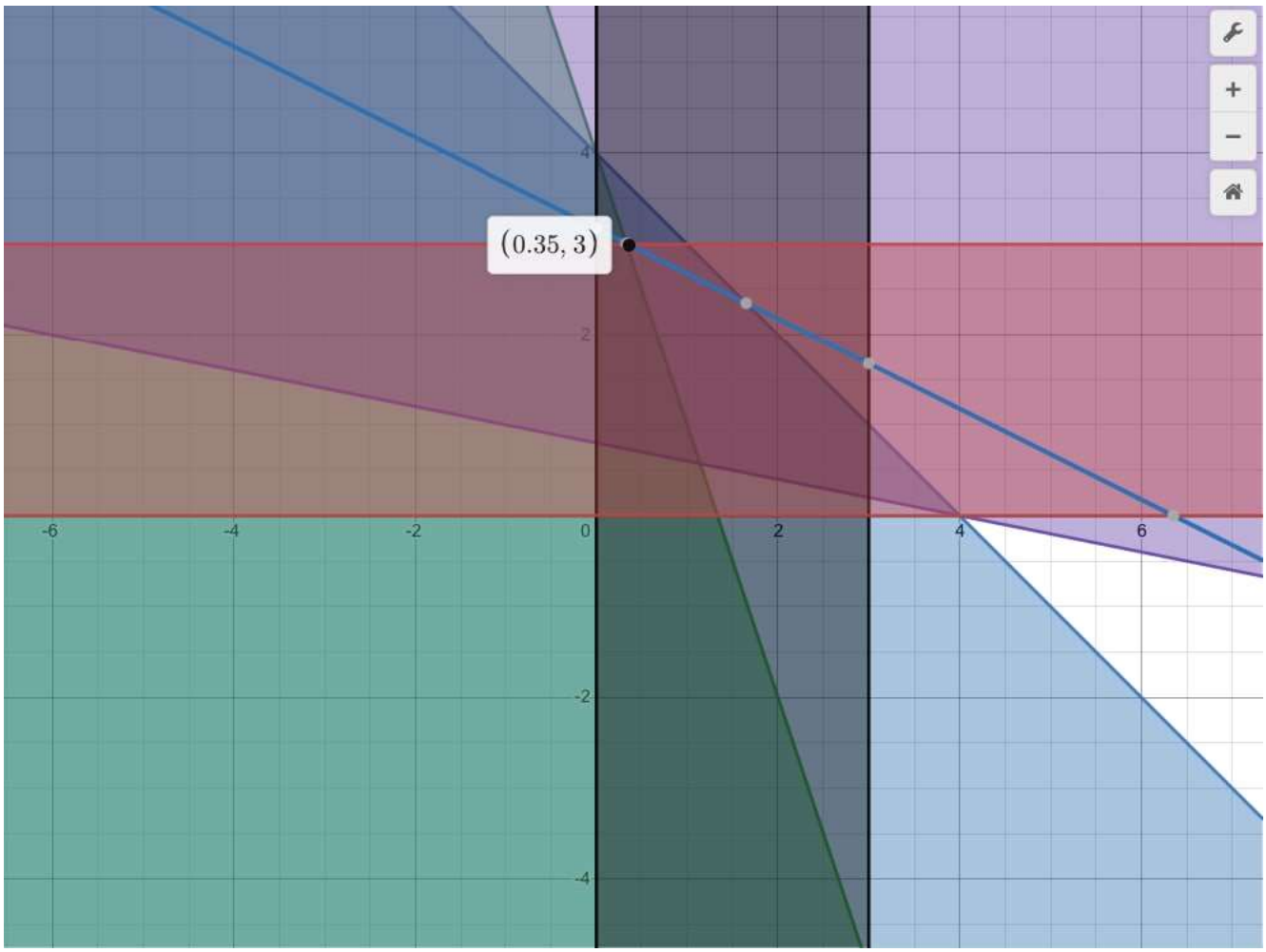

Рис. 4. Максимум функції.

Висновки

В ході виконання лабораторної роботи я ознайомився на практиці із основними поняттями теорії лінійного програмування, навчився знаходити оптимальні плани задач лінійного програмування графічно та використовуючи симплекс метод. Для поставленої задачі записав математичну модель, економічний зміст та систему обмежень і розв’язав її за допомогою симплекс методу. Навчився також застосовувати графічний метод для розв’язування задач ЛП.