**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **КНІТ**
Кафедра **ПЗ**

# ЗВІТ

До лабораторної роботи №2
**На тему:** *"Двоїстий симплекс-методю. Зв'язок між розв'язками прямої та двоїстої задач ЛП "*
**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:** Двоїстий симплекс-метод. Зв'язок між розв'язками прямої та двоїстої задач ЛП.
**Мета роботи:** Ознайомитись на практиці із двоїстими задачами лінійного програмування та навчитись розв'язувати їх із використанням двоїстого симплекс-методу.

## Лабораторне завдання

1. Отримати індивідуальний варіант завдання.
2. Записати математичну модель задачі ЛП (згідно з варіантом з Додатка №1 до лабораторної роботи №1) і побудувати до неї двоїсту задачу. Зазначити економічний зміст цільової функції і системи обмежень обидвох задач.
3. Написати програмування розв'язування задачі ЛП двоїстим симплекс-методом та за її допомогою знайти розв'язок (максимальне (мінімальне) значення функції та значення змінних, при якому воно досягається) побудованої двоїстої задачі.
4. Знайти оптимальний план прямої задачі, використовуючи зв'язок між розв'язками прямої та двоїстой задачі. Порівняйте отриманий розв'язок з відповідними результатами лабораторної роботи №1.
5. Дати економічне тлумачення основним і додатковим змінним вихідної та двоїстої задач.
6. Оформити звіт про виконану роботу.
7. 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.

## Теоретичні відомості

14. Як визначити, чи ресурс є дефіцитним (недефіцитним)?
Двоїсті змінні вказують на міру дефіцитності ресурсу, вони чисельно дорівнюють зміні цільової функції при зміні відповідного ресурсу на одиницю. Якщо двоїста змінна = 0, то відповідний ресурс в надлишку і збільшення ресурсу ні до чого не приведе. А якщо двоїста змінна не дорівнює 0, то збільшення відповідного ресурсу на одиницю приведе до збільшення обсягів реалізації на значення змінної.
18. Який розв'язок називають надоптимальним?

Це розв'язок, який є оптимальний, але недопустимим, тобто оціночні коефіцієнти >= 0 (останні рядок СТ), але стовпець вільних членів (Р0) має від'ємні значення.

4. Сформулюйте основні теореми двоїстості.

Перша теорема двоїстості: Якщо одна задача з пари двоїстих задач має розв'язок, то й інша має розв'язок. Оптимальні значення функцій мети рівні між собою. Якщо функція мети однієї з задач необмежена, двоїста до неї задача взагалі не має припустимих розв'язків.

Друга теорема двоїстості (теорема про доповнюючу нежорсткість): Розв'язки $x^*=(x_1^*,\ldots, x_n^*)$, $y^*=(y_1^*,\ldots, y_n^*)$ є оптимальними тоді і лише тоді, коли виконуються співвідношення:

$$x_j^* \times (\sum_{i=1}^{m} a_{ij} y_i^* - c_j) = 0, j = \overline{1,n}, \quad y_i^* \times (\sum_{j=1}^{n} a_{ij} x_j^* - b_i) = 0, i = \overline{1,m}.$$

## Хід роботи

1. Записав математичну модель задачі ЛП:
   F(x)=315x1+278x2+573x3+370x4->max
   Економічний зміст: максимізувати прибуток від реалізації продукції.
   Записав систему обмежень:
   50x1+62x3<=6427
   4x1+5x2+2x3+2x4<=480
   9x1+11x2+15x3+5x4<=2236
   16x1+9x2+16x3+13x4<=2624
   16x2+3x3+5x4<=780
   3x1+4x2+3x3+3x4<=520
   4.5x1+4.5x2+4.5x3+4.5x4<=720
   Звівши систему до канонічної вигляду записав двоїсту задачу:
   G(y)=6427y1+480y2+2236y3+2624y4+780y5+520y6+720y7->min
   Економічний зміст: мінімізувати витрати на реалізацію плану.
   Система обмежень:
   50y1+4y2+9y3+16y4+3y6+4.5y7>=315
   5y2+11y3+9y4+16y5+4y6+4.5y7>=278
   62y1+2y2+15y3+16y4+3y5+3y6+4.5y7>=573
   2y2+5y3+13y4+5y5+3y6+4.5y7>=370
2. Написав програму, яка будує двоїсту задачу до задачі ЛП, зводить систему до канонічного вигляду, і шукає розв'язок двоїстої задачі і задачі ЛП.

Program.cs

```
var processor = new SimplexProcessor(new ConsoleReader());
Tuple<SimplexFunction, SimplexConstraint[]> data = processor.GetData();
Console.WriteLine(data.Item1.ToString('x'));
foreach (SimplexConstraint constraint in data.Item2)
{
    Console.WriteLine(constraint.ToString('x'));
}
List<DualSimplexSnapshot> steps = new DualSimplex(data.Item1, data.Item2)
     .Solve()
     .ToList();

ISimplexTableBuilder builder = new ConsoleSimplexTableBuilder(steps);
List<string> tables = builder.Build().ToList();

var firstStep = steps.First();

Tuple<SimplexFunction, SimplexConstraint[]> dualSimplexData =
firstStep.BuildDualSimplexData();

Console.WriteLine(dualSimplexData.Item1.ToString('y'));
foreach (SimplexConstraint constraint in dualSimplexData.Item2)
{
```

```
    Console.WriteLine(constraint.ToString('y'));
}

foreach (var table in tables)
{
    Console.WriteLine(table);
    Console.WriteLine();
}

var lastStep = steps.Last();

var basisIndices = lastStep.BasisIndices.ToArray();
var b = lastStep.B.ToArray();
var functionCoefficients = lastStep.FunctionCoefficients.ToArray();

var varCount = lastStep.SystemMatrix.GetLength(1);
var initialVarCount = data.Item1.VarCount;
var additionalVarCount = varCount - initialVarCount;

var y = Enumerable
    .Range(0, varCount)
    .Select(i => basisIndices.Contains(i) ? b[Array.IndexOf(basisIndices, i)] : 0);

Console.WriteLine("Y = [ {0} ] ", String.Join(", ", y));

var x = Enumerable
    .Range(0, varCount)
    .Select(i => i < initialVarCount
        ? functionCoefficients[i + additionalVarCount]
        : functionCoefficients[i - initialVarCount])
    .ToArray();

 Console.WriteLine("X = [ {0} ] ", String.Join(", ", x));
```

DualSimplex.cs

```
using Lab1;

namespace Lab2
{
    public class DualSimplex
    {
        private readonly double[,] _systemMatrix;
        private readonly int _rowCount, _columnCount;
        private readonly double[] _b;
        private readonly int[] _basisIndices;
        private readonly double[] _functionCoefficients;
        private readonly double[] _initialFCoefficients;
        private double _g;

        public DualSimplex(SimplexFunction function, SimplexConstraint[] constraints)
        {
            if (constraints.GroupBy(constraint => constraint.Count).First().Count() !=
constraints.Length)
            {
                throw new Exception();
            }

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

            var dualSimplex = Tuple.Create(
                function.ToCanonical(), constraints.ToCanonical().ToArray()
            ).ToDualSimplex();
```

```
        dualSimplex.Deconstruct(out var dualFunction, out var dualConstraints);

        dualFunction = dualFunction.ToCanonical();
        dualConstraints = dualConstraints.ToCanonical().ToArray();

        _initialFCoefficients = Enumerable
            .Range(0, dualConstraints.First().Count)
            .Select(i =>
        {
            if (i < dualFunction.VarCount)
            {
                return dualFunction[i];
            }

            return 0;
        }).ToArray();

        _functionCoefficients = _initialFCoefficients
            .Select(coefficient => -coefficient)
            .ToArray();

        _systemMatrix = dualConstraints.ToMatrix();

        _b = dualConstraints
            .Select(c => c.Value)
            .ToArray();

        _g = 0;
        _rowCount = _systemMatrix.GetLength(0);
        _columnCount = _systemMatrix.GetLength(1);
        _basisIndices = new int[_rowCount];

        SetBasisIndices();
    }

    private void SetBasisIndices()
    {
        var counter = 0;

        for (var i = 0; i < _columnCount; ++i)
        {
            if (counter == _rowCount)
            {
                break;
            }

            var hasOne = false;
            var isZeroOnly = true;
            for (var j = 0; j < _rowCount; ++j)
            {
                if (_systemMatrix[j, i].IsEqualToWithin(1.0, 1e-6) && !hasOne)
                {
                    hasOne = true;
                    continue;
                }

                if (_systemMatrix[j, i] != 0)
                {
                    isZeroOnly = false;
                    break;
                }
            }

            if (isZeroOnly && hasOne)
            {
```

```
                    _basisIndices[counter++] = i;
                }
            }
        }

        public IEnumerable<DualSimplexSnapshot> Solve()
        {
            var snapshots = new List<DualSimplexSnapshot>()
            {
                new DualSimplexSnapshot(
                    _systemMatrix,
                    _basisIndices,
                    _b,
                    _initialFCoefficients,
                    _functionCoefficients,
                    _g,
                    DualSimplexResultStatus.NotFoundYet
                )
            };

            var stepCounter = 1;

            do
            {
                DualSimplexSnapshot snapshot = NextStep();
                snapshots.Add(snapshot);
            } while (snapshots[stepCounter++].ResultStatus is
DualSimplexResultStatus.NotFoundYet);


            return snapshots;
        }

        private DualSimplexSnapshot NextStep()
        {
            int baseRow = GetBaseRow();
            int baseColumn = GetBaseColumn(baseRow);

            _basisIndices[baseRow] = baseColumn;
            DualSimplexResultStatus status = ApplyTriangleRule(baseRow, baseColumn);

            return new DualSimplexSnapshot(
                _systemMatrix,
                _basisIndices,
                _b,
                _initialFCoefficients,
                _functionCoefficients,
                _g,
                status
            );
        }

        private int GetBaseRow()
        {
            var minNegativeIndex = 0;

            for (var i = 1; i < _rowCount; ++i)
            {
                minNegativeIndex = _b[minNegativeIndex] > _b[i] ? i : minNegativeIndex;
            }

            return minNegativeIndex;
        }

        private int GetBaseColumn(int baseRow)
```

```
{
    var minRatio = double.MaxValue;
    var minRatioColumnIndex = 0;

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

        var ratio = -_functionCoefficients[i] / _systemMatrix[baseRow, i];
        if (ratio < minRatio)
        {
            minRatio = ratio;
            minRatioColumnIndex = i;
        }
    }

    return minRatioColumnIndex;
}

private DualSimplexResultStatus ApplyTriangleRule(int baseRow, int baseColumn)
{
    var baseElement = _systemMatrix[baseRow, baseColumn];
    var baseRowValues = new double[_columnCount];
    var baseColumnValues = new double[_rowCount];
    var newRow = new double[_columnCount];

    _g -= _functionCoefficients[baseColumn] * (_b[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];
    }

    _b[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 < _b.Length; ++i)
    {
        if (i == baseRow)
        {
            continue;
        }

        _b[i] -= baseColumnValues[i] * _b[baseRow];
```

```
            }

            var 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];
            }

            return _b.Count(value => value >= 0) == _rowCount
                ? DualSimplexResultStatus.Found
                : DualSimplexResultStatus.NotFoundYet;
        }
    }
}
```

DualSimplexSnapshot.cs

```
namespace Lab2
{
    public class DualSimplexSnapshot
    {
        public IEnumerable<int> BasisIndices { get; }
        public IEnumerable<double> B { get; }
        public IEnumerable<double> InitialFCoefficients { get; }
        public IEnumerable<double> FunctionCoefficients { get; }
        public double G { get; }
        public DualSimplexResultStatus ResultStatus { get; }
        public double[,] SystemMatrix { get; }

        public DualSimplexSnapshot(
            double[,] systemMatrix,
            IEnumerable<int> basisIndices,
            IEnumerable<double> b,
            IEnumerable<double> initialFCoefficients,
            IEnumerable<double> functionCoefficients,
            double g,
            DualSimplexResultStatus resultStatus
        )
        {
            SystemMatrix = (systemMatrix.Clone() as double[,])!;
            BasisIndices = basisIndices.ToArray();
            B = b.ToArray();
            InitialFCoefficients = initialFCoefficients.ToArray();
            FunctionCoefficients = functionCoefficients.ToArray();
            G = g;
            ResultStatus = resultStatus;
        }
    }

    public enum DualSimplexResultStatus
    {
        Unbounded, Found, NotFoundYet
    }
}
```

Extensions.cs

```
using System.Text;
using Lab1;

namespace Lab2
```

```
{
    public static class Extensions
    {
        public static Tuple<SimplexFunction, SimplexConstraint[]> ToDualSimplex(
            this Tuple<SimplexFunction, SimplexConstraint[]> canonicalSimplex
        )
        {
            var canonicalFunction = canonicalSimplex.Item1;
            var canonicalConstraints = canonicalSimplex.Item2;

            if (canonicalFunction.IsMaximize == false)
            {
                throw new InvalidOperationException("Simplex function must be
canonical.");
            }

            if (canonicalConstraints.Count(c => c.Sign == "=") !=
canonicalConstraints.Length)
            {
                throw new InvalidOperationException("Simplex constraints must be
canonical.");
            }

            IEnumerable<double> functionCoefficients = Enumerable
                .Range(0, canonicalConstraints.Length)
                .Select(i => canonicalConstraints[i].Value);

            var dualFunction = new SimplexFunction(functionCoefficients, false);

            var dualConstraints = Enumerable
                .Range(0, canonicalFunction.VarCount)
                .Select(i =>
                {
                    var coefficients = Enumerable
                        .Range(0, canonicalConstraints.Length)
                        .Select(j => canonicalConstraints[j][i]);

                    double value = canonicalFunction[i];

                    return new SimplexConstraint(coefficients, ">=", value);
                });

            return Tuple.Create(dualFunction, dualConstraints.ToArray());
        }
        public static SimplexFunction ToCanonical(this SimplexFunction function)
        {
            if (function.IsMaximize)
            {
                return new SimplexFunction(
                    Enumerable.Range(0, function.VarCount).Select(i => function[i]),
                    function.IsMaximize
                );
            }

            var coefficients = Enumerable.Range(0, function.VarCount).Select(i => -
function[i]);
            return new SimplexFunction(coefficients, true);
        }

        public static IEnumerable<SimplexConstraint> ToCanonical(this
IEnumerable<SimplexConstraint> constraints)
        {
            var constraintsArr = constraints as SimplexConstraint[] ??
constraints.ToArray();
```

```
            var additionalVarCount = constraintsArr.Count(constraint => constraint.Sign
!= "=");
            if (additionalVarCount == 0)
            {
                return Enumerable
                    .Range(0, constraintsArr.Length)
                    .Select(i => constraintsArr[i].Clone());
            }

            var k = 0;

            return Enumerable
                .Range(0, constraintsArr.Length)
                .Select(i => constraintsArr[i].ToCanonical(additionalVarCount, ref k));
        }

        public static SimplexConstraint ToCanonical(this SimplexConstraint constraint,
int additionalVar, ref int k)
        {
            var coefficients = new double[constraint.Count + additionalVar];
            for (var i = 0; i < constraint.Count; ++i)
            {
                coefficients[i] = constraint[i];
            }

            if (constraint.Sign == "=")
            {
                return new SimplexConstraint(coefficients, constraint.Sign,
constraint.Value);
            }

            string sign = constraint.Sign;
            double value = constraint.Value;
            if (sign == ">=")
            {
                coefficients.Inverse();
                value *= -1;
            }

            coefficients[constraint.Count + k++] = 1;
            return new SimplexConstraint(coefficients, "=", value);
        }

        public static void Inverse(this double[] array)
        {
            for (var i = 0; i < array.Length; ++i)
            {
                array[i] = -array[i];
            }
        }

        public static SimplexConstraint Clone(this SimplexConstraint constraint)
        {
            string sign = constraint.Sign;
            double value = constraint.Value;

            IEnumerable<double> coefficients = Enumerable
                .Range(0, constraint.Count)
                .Select(i => constraint[i]);

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

        public static double[,] ToMatrix(this SimplexConstraint[] constraints)
        {
```

```
            var rows = constraints.Length;
            var cols = constraints.First().Count;

            var matrix = new double[rows, cols];

            for (var i = 0; i < rows; ++i)
            {
                for (var j = 0; j < cols; ++j)
                {
                    matrix[i, j] = constraints[i][j];
                }
            }

            return matrix;
        }

        public static Tuple<SimplexFunction, SimplexConstraint[]> BuildDualSimplexData(
            this DualSimplexSnapshot snapshot
        )
        {
            var function = new SimplexFunction(snapshot.InitialFCoefficients, true);
            var b = snapshot.B.ToArray();
            var constraints = new SimplexConstraint[snapshot.SystemMatrix.GetLength(0)];

            for (var i = 0; i < constraints.Length; ++i)
            {
                var coefficients = Enumerable
                    .Range(0, snapshot.SystemMatrix.GetLength(1))
                    .Select(j => snapshot.SystemMatrix[i, j]);

                constraints[i] = new SimplexConstraint(coefficients, "=", b[i]);
            }

            return Tuple.Create(function, constraints);
        }

            public static string ToString(this SimplexFunction function, char varChar)
        {
            if (function.VarCount == 0)
            {
                return "";
            }

            var sb = new StringBuilder().Append($"F(x) = {function[0]:F3}{varChar}1");

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

                var s = usePlus ? $"+{function[i]:F3}{varChar}{i + 1}" :
$"{function[i]:F3}{varChar}{i + 1}";

                sb.Append(s);
            }

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

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

        public static string ToString(this SimplexConstraint constraint, char varChar)
        {
            if (constraint.Count == 0)
            {
                return "";
```

```
            }

            var sb = new StringBuilder().Append($"{constraint[0]:F3}{varChar}1");

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

                var s = usePlus ? $"+{constraint[i]:F3}{varChar}{i + 1}" :
$"{constraint[i]:F3}{varChar}{i + 1}";

                sb.Append(s);
            }

            return sb.Append($"{constraint.Sign}{constraint.Value:F3}").ToString();
        }

        public static bool IsEqualToWithin(this double thisArg, double value, double
tolerance)
        {
            return Math.Abs(thisArg - value) <= tolerance;
        }
    }
}
```

ISimplexTableBuilder.cs

```
namespace Lab2
{
    public interface ISimplexTableBuilder
    {
        IEnumerable<string> Build();
    }
}
```

ConsoleSimplexTableBuilder.cs

```
using System.Text;

namespace Lab2
{
    public class ConsoleSimplexTableBuilder : ISimplexTableBuilder
    {
        private List<DualSimplexSnapshot> _steps;

        public IEnumerable<DualSimplexSnapshot> Steps
        {
            get => _steps;
            set => _steps = value.ToList();
        }

        public ConsoleSimplexTableBuilder(IEnumerable<DualSimplexSnapshot> steps)
        {
            Steps = steps;
        }
        public IEnumerable<string> Build()
        {
            foreach (var step in _steps)
            {
                yield return BuildTable(step);
            }
        }

        private string BuildTable(DualSimplexSnapshot step)
        {
            var sb = new StringBuilder();
```

```
            var firstLineBuilder = new StringBuilder($"|{"yb",-12}|{"cb",-12}|{"P0",-
12}|");

            var initialCoefficients = step.InitialFCoefficients.ToArray();

            for (var i = 0; i < initialCoefficients.Length; i++)
            {
                firstLineBuilder.Append($"{$"c{i + 1}={initialCoefficients[i]:F3}",-
12}|");
            }

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

            var basisIndices = step.BasisIndices.ToArray();
            var functionCoefficients = step.FunctionCoefficients.ToArray();
            var b = step.B.ToArray();
            double[,] matrix = step.SystemMatrix;

            for (var i = 0; i < basisIndices.Length; ++i)
            {
                var c = initialCoefficients[basisIndices[i]];

                sb
                    .Append($"|{$"y{basisIndices[i] + 1}",-12}|")
                    .Append($"{$"{c:F3}",-12}|")
                    .Append($"{$"{b[i]:F3}",-12}|");

                for (var j = 0; j < matrix.GetLength(1); ++j)
                {
                    sb.Append($"{$"{matrix[i, j]:F3}",-12}|");
                }

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

            sb
                .Append($"|{$"G",-12}|")
                .Append($"{$"=",-12}|")
                .Append($"{step.G,-12:F3}|");

            foreach (var coefficient in functionCoefficients)
            {
                sb.Append($"{$"{coefficient:F3}",-12}|");
            }

            sb.AppendLine().AppendLine(emptyLine);

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

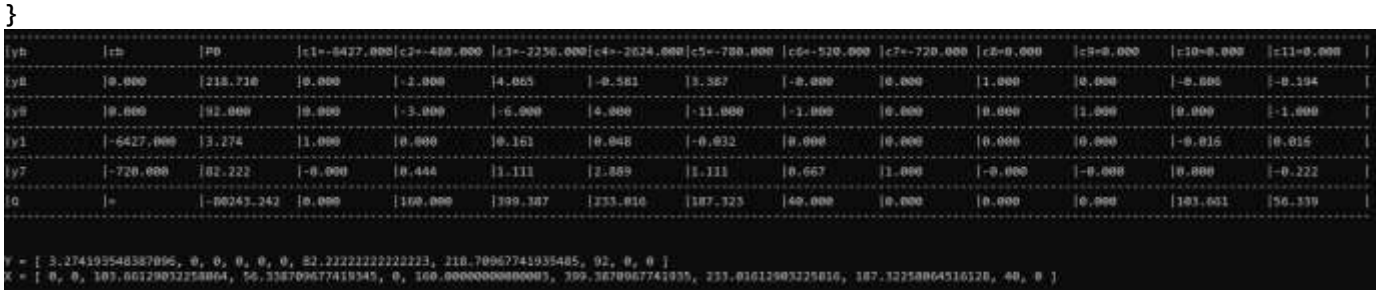

**Рис. 1. Остання СТ, розв'язки двоїстої задачі та задчі ЛП.**

## Висновки

На лабораторній роботі я ознайомився на практиці із двоїстими задачами лінійного програмування та навчився розв’язувати їх із використанням двоїстого симплекс-методу. Написав програму, яка зчитує дані про задачу ЛП, тоді зводить її до канонічного вигляду і записує двоїсту задачу до неї, тоді розв’язує, дослідив зв’язок між розв’язками прямої та двоїстої задач.