

Міністерство освіти і науки України  
Національний технічний університет України «Київський політехнічний  
інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 4 з дисципліни  
«Основи програмування 2. Модульне програмування»

«Перевантаження операторів»

Варіант 35

Виконав студент ІП-15, Шабанов Метін Шаміль огли  
(шифр, прізвище, ім'я, по батькові)

Перевірів Вечерковська Анастасія Сергіївна  
(прізвище, ім'я, по батькові)

Київ 2022

## Лабораторна робота 4

### Перевантаження операторів Варіант 35 (1)

Завдання:

1. Створити клас `TSystemLinearEquation`, який представляє систему лінійних алгебраїчних рівнянь і містить методи для знаходження коренів рівнянь та перевірки того, чи є деякий набір чисел розв'язком системи рівнянь. На основі цього класу створити класи-нащадки, які представляють системи двох та трьох лінійних рівнянь (відповідно з двома та трьома невідомими). Випадковим чином згенерувавши дані для декількох систем двох лінійних рівнянь та декількох систем трьох лінійних рівнянь. Знайти розв'язок даних систем лінійних алгебраїчних рівнянь (обох видів).

Виконання:

C#:

```
C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#
1 namespace Lab5_InheritanceAndPolymorphism
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             double[,] firstMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 2);
8             TwoSystemLinearEquation firstSystem = new TwoSystemLinearEquation(firstMatrix);
9             Printer.PrintSystem(firstSystem);
10            double[] firstRoots = firstSystem.MatrixMethod();
11            Printer.PrintRoots(firstRoots);
12
13            double[,] secondMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 2);
14            TwoSystemLinearEquation secondSystem = new TwoSystemLinearEquation(secondMatrix);
15            Printer.PrintSystem(secondSystem);
16            double[] secondRoots = secondSystem.KramerMethod();
17            Printer.PrintRoots(secondRoots);
18
19            double[,] thirdMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 3);
20            ThreeSystemLinearEquation thirdSystem = new ThreeSystemLinearEquation(thirdMatrix);
21            Printer.PrintSystem(thirdSystem);
22            double[] thirdRoots = thirdSystem.MatrixMethod();
23            Printer.PrintRoots(thirdRoots);
24
25            double[,] fourthMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 3);
26            ThreeSystemLinearEquation fourthSystem = new ThreeSystemLinearEquation(fourthMatrix);
27            Printer.PrintSystem(fourthSystem);
28            double[] fourthRoots = fourthSystem.KramerMethod();
29            Printer.PrintRoots(fourthRoots);
30        }
31    }
32 }
```

```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#
1 using System;
2
3 namespace Lab5_InheritanceAndPolymorphism
4 {
5     public class RandomSystemGenerator
6     {
7         public static double[,] GenerateRandomSystem(int variablesQuantity)
8         {
9             Random rand = new Random();
10            double[,] system = new double[variablesQuantity, variablesQuantity + 1];
11
12            for (int i = 0; i < variablesQuantity; i++)
13                for (int j = 0; j < variablesQuantity + 1; j++)
14                    system[i, j] = Math.Round(rand.NextDouble() * 9);
15
16            return system;
17        }
18    }
19 }

```

```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# Matrix.cs × C# Printer.cs
1 namespace Lab5_InheritanceAndPolymorphism
2 {
3     public abstract class TSystemLinearEquation
4     {
5         protected double[,] _coefficients;
6
7         public TSystemLinearEquation(double[,] coefficients)
8         {
9             _coefficients = coefficients;
10        }
11
12        public double[] MatrixMethod()
13        {
14            Matrix A = new Matrix(_initMatrixA());
15            Matrix B = new Matrix(_initMatrixB());
16
17            if (A.MakeInverseMatrix() != null)
18            {
19                Matrix result = new Matrix(A.MakeInverseMatrix()) * B;
20                return _convertToRootsView(result.Matrix1);
21            }
22
23            return null;
24        }
25    }

```

2 usages

```

26     public double[] KramerMethod()
27     {
28         Matrix A = new Matrix(_initMatrixA());
29         Matrix B = new Matrix(_initMatrixB());
30         int size = A.Matrix1.GetLength(dimension: 0);
31         double det = A.FindDeterminant(A.Matrix1);
32
33         if (det != 0)
34         {
35             double[] roots = new double[size];
36
37             for (int i = 0; i < roots.Length; i++)
38             {
39                 Matrix replaced = new Matrix(formNewMatrix(A.Matrix1, replacingColumn: B.Matrix1, i));
40                 roots[i] = replaced.FindDeterminant(replaced.Matrix1);
41             }
42
43             for (int i = 0; i < roots.Length; i++)
44                 roots[i] /= det;
45
46             return roots;
47         }
48
49         return null;
50     }

```

```

52     private bool checkOnRoots(double[] beleivedRoots)
53     {
54         int size = _coefficients.GetLength(dimension: 1);
55         bool areRoots = true;
56         for (int i = 0; i < size && areRoots; i++)
57         {
58             double left = 0;
59             for (int j = 0; j < size - 1 && areRoots; j++)
60                 left += _coefficients[i, j] * beleivedRoots[j];
61
62             if (left != _coefficients[i, size])
63                 areRoots = false;
64         }
65
66         return areRoots;
67     }

```

2 usages

```

70     private double[,] _initMatrixA()
71     {
72         int rows = _coefficients.GetLength(dimension: 0);
73         int cols = _coefficients.GetLength(dimension: 1) - 1;
74         double[,] initialized = new double[rows, cols];
75
76         for (int i = 0; i < rows; i++)
77             for (int j = 0; j < cols; j++)
78                 initialized[i, j] = _coefficients[i, j];
79
80         return initialized;
81     }

```

```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# Matrix.cs × C# Printer.cs
83     private double[,] _initMatrixB()
84     {
85         double[,] B = new double[_coefficients.GetLength(dimension: 0), 1];
86
87         for (int i = 0; i < B.Length; i++)
88             B[i, 0] = _coefficients[i, _coefficients.GetLength(dimension: 1) - 1];
89
90         return B;
91     }
92
93     [1 usage]
94     private double[,] formNewMatrix(double[,] mainMatrix, double[,] replacingColumn, int replacingIndex)
95     {
96         double[,] tempMatrix = _copyArray(mainMatrix);
97
98         for (int i = 0; i < tempMatrix.GetLength(dimension: 0); i++)
99         {
100             tempMatrix[i, replacingIndex] = replacingColumn[i, 0];
101         }
102
103         return tempMatrix;
104     }
105
106     [1 usage]
107     private double[] _convertToRootsView(double[,] rootsMatrix)
108     {
109         double[] roots = new double[rootsMatrix.GetLength(dimension: 0)];
110
111         for (int i = 0; i < roots.Length; i++)
112             roots[i] = rootsMatrix[i, 0];
113
114         return roots;
115     }
116
117     [1 usage]
118     private double[,] _copyArray(double[,] arr)
119     {
120         double[,] copiedArr = new double[arr.GetLength(dimension: 0), arr.GetLength(dimension: 1)];
121
122         for (int i = 0; i < copiedArr.GetLength(dimension: 0); i++)
123             for (int j = 0; j < copiedArr.GetLength(dimension: 1); j++)
124                 copiedArr[i, j] = arr[i, j];
125
126         return copiedArr;
127     }
128
129     public override string ToString()
130     {
131         string linearSystem = "";
132
133         for (int i = 0; i < _coefficients.GetLength(dimension: 0); i++)
134         {
135             int j = 0;
136             for (; j < _coefficients.GetLength(dimension: 1) - 2; j++)
137                 linearSystem += $"{_coefficients[i, j]}{(char)(j + 120)} + ";
138
139             linearSystem += $"{_coefficients[i, j]}{(char)(j + 120)}";
140             linearSystem += " = {_coefficients[i, j + 1]}";
141
142             linearSystem += "\n";
143         }
144
145         return linearSystem;
146     }

```

```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#
1 using System;
2
3 namespace Lab5_InheritanceAndPolymorphism
4 {
5     [4 usages]
6     public class TwoSystemLinearEquation : TSystemLinearEquation
7     {
8         [2 usages]
9         public TwoSystemLinearEquation(double[,] coefficients) : base(coefficients)
10        {
11            if(coefficients.GetLength(dimension: 0) == 2 && coefficients.GetLength(dimension: 1) == 3)
12                _coefficients = coefficients;
13            else
14                Console.WriteLine("This ain't a two variables system.");
15        }
16    }
17 }

```

```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#
1 using System;
2
3 namespace Lab5_InheritanceAndPolymorphism
4 {
5     [4 usages]
6     public class ThreeSystemLinearEquation : TSystemLinearEquation
7     {
8         [2 usages]
9         public ThreeSystemLinearEquation(double[,] coefficients) : base(coefficients)
10        {
11            if(coefficients.GetLength(dimension: 0) == 3 && coefficients.GetLength(dimension: 1) == 4)
12                _coefficients = coefficients;
13            else
14                Console.WriteLine("This ain't a three variables system.");
15        }
16    }
17 }

```

```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#
1 using System;
2
3 namespace Lab5_InheritanceAndPolymorphism
4 {
5     [24 usages] [3 exposing APIs]
6     public class Matrix
7     {
8         private double[,] _matrix;
9         private double[,] _transponated;
10        private double[,] _minorsMatrix;
11
12        [10 usages]
13        public Matrix(double[,] matrix)
14        {
15            _matrix = matrix;
16            if (_matrix.GetLength(dimension: 0) == _matrix.GetLength(dimension: 1))
17            {
18                _transponated = MakeTransponated(_matrix);
19                _getMinorsMatrix();
20            }
21        }
22
23        [6 usages]
24        public double[,] Matrix1 => _matrix;
25
26        public double[,] Transponated => _transponated;
27        public double[,] Minors => _minorsMatrix;
28    }
29 }

```

1 usage

```
26 public double[,] MakeTransponated(double[,] matrix)
27 {
28     double[,] transponated = new double[matrix.GetLength( dimension: 0), matrix.GetLength( dimension: 1)];
29
30     for (int i = 0; i < matrix.GetLength( dimension: 0); i++)
31     {
32         for (int j = 0; j < matrix.GetLength( dimension: 1); j++)
33         {
34             transponated[j, i] = matrix[i, j];
35         }
36     }
37
38     return transponated;
39 }
40
```

2 usages

```
41 public double[,] MakeInverseMatrix()
42 {
43     double det = FindDeterminant(_matrix);
44
45     if (det == 0)
46         return null;
47
48     Matrix deltaOnCofactor = new Matrix(_minorsMatrix) * (1/det);
49
50     double[,] inverseMatrix = deltaOnCofactor._transponated;
51
52     return inverseMatrix;
53 }
54
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# M

```
55 private void _getMinorsMatrix()
56 {
57     _minorsMatrix = new double[_matrix.GetLength( dimension: 0), _matrix.GetLength( dimension: 1)];
58
59     for (int i = 0; i < _matrix.GetLength( dimension: 0); i++)
60         for (int j = 0; j < _matrix.GetLength( dimension: 1); j++)
61             _minorsMatrix[i, j] = Math.Pow(-1, (i + j)) * FindDeterminant(_formMatrix(i, j));
62 }
63
```

1 usage

```
64 private double[,] _formMatrix(int indexI, int indexJ)
65 {
66     int size = _matrix.GetLength( dimension: 0) - 1;
67     double[,] subMatrix = new double[size, size];
68
69     int counterI = 0;
70     int counterJ = 0;
71
72     for (int i = 0; i < _matrix.GetLength( dimension: 0); i++)
73     {
74         for (int j = 0; j < _matrix.GetLength( dimension: 1); j++)
75         {
76             if (counterJ ≥ size)
77             {
78                 counterI++;
79                 counterJ = 0;
80             }
81
82             if (i ≠ indexI && j ≠ indexJ)
83             {
84                 subMatrix[counterI, counterJ] = _matrix[i, j];
85                 counterJ++;
86             }
87         }
88     }
89 }
```

4 usages

```
94 public double FindDeterminant(double[,] matrix)
95 {
96     double determinant = 0;
97
98     int size = matrix.GetLength(dimension: 0);
99
100     if (size == 1)
101         determinant += matrix[0, 0];
102
103     if (size == 2)
104         determinant += matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0];
105
106     if (size == 3)
107     {
108         determinant += matrix[0, 0] * matrix[1, 1] * matrix[2, 2];
109         determinant += matrix[0, 1] * matrix[1, 2] * matrix[2, 0];
110         determinant += matrix[0, 2] * matrix[1, 0] * matrix[2, 1];
111         determinant -= matrix[0, 2] * matrix[1, 1] * matrix[2, 0];
112         determinant -= matrix[0, 1] * matrix[1, 0] * matrix[2, 2];
113         determinant -= matrix[0, 0] * matrix[1, 2] * matrix[2, 1];
114     }
115
116     return determinant;
117 }
```

```
119 public static Matrix operator *(Matrix matrix1, Matrix matrix2)
120 {
121     double[,] table1 = matrix1._matrix;
122     double[,] table2 = matrix2._matrix;
123
124     double[,] resultingTable = new double[table1.GetLength(dimension: 0), table2.GetLength(dimension: 1)];
125
126     for (var i = 0; i < table1.GetLength(dimension: 0); i++)
127     {
128         for (var j = 0; j < table2.GetLength(dimension: 1); j++)
129         {
130             resultingTable[i, j] = 0;
131
132             for (var k = 0; k < table1.GetLength(dimension: 1); k++)
133                 resultingTable[i, j] += table1[i, k] * table2[k, j];
134         }
135     }
136
137     return new Matrix(resultingTable);
138 }
139
140 public static Matrix operator *(Matrix matrix1, double number)
141 {
142     double[,] table1 = matrix1._matrix;
143
144     for (int i = 0; i < table1.GetLength(dimension: 0); i++)
145     {
146         for (int j = 0; j < table1.GetLength(dimension: 1); j++)
147         {
148             table1[i, j] *= number;
149         }
150     }
151
152     return new Matrix(table1);
153 }
154
```



```

155     public static Matrix operator *(Matrix matrix1, double[] number)
156     {
157         double[,] table1 = matrix1._matrix;
158
159         for (int i = 0; i < table1.GetLength( dimension: 0); i++)
160         {
161             for (int j = 0; j < table1.GetLength( dimension: 1); j++)
162             {
163                 table1[j, i] *= number[i];
164             }
165         }
166
167         return new Matrix(table1);
168     }
169
170 }
171 }

```

## Python:

```

main.py × LinearSystem.py × TwoVarSystem.py × ThreeVarSystem.py × RandomCoefficientsGenerator.py × Printer.py ×
1  from RandomCoefficientsGenerator import RandSystem
2  from ThreeVarSystem import ThreeSystemLinearEquation
3  from TwoVarSystem import TwoSystemLinearEquation
4  from Printer import SystemsPrinter
5
6  first_matrix = RandSystem.generate_random_system(2)
7  first_sys = TwoSystemLinearEquation(first_matrix)
8  SystemsPrinter.print_system(first_sys)
9  first_roots = first_sys.matrix_method()
10 SystemsPrinter.print_roots(first_roots)
11
12 second_matrix = RandSystem.generate_random_system(2)
13 second_sys = TwoSystemLinearEquation(second_matrix)
14 SystemsPrinter.print_system(second_sys)
15 second_roots = second_sys.kramer_method()
16 SystemsPrinter.print_roots(second_roots)
17
18 third_matrix = RandSystem.generate_random_system(3)
19 third_sys = ThreeSystemLinearEquation(third_matrix)
20 SystemsPrinter.print_system(third_sys)
21 third_roots = third_sys.matrix_method()
22 SystemsPrinter.print_roots(third_roots)
23
24 fourth_matrix = RandSystem.generate_random_system(3)
25 fourth_sys = ThreeSystemLinearEquation(fourth_matrix)
26 SystemsPrinter.print_system(fourth_sys)
27 fourth_roots = fourth_sys.matrix_method()
28 SystemsPrinter.print_roots(fourth_roots)

```

main.py × LinearSystem.py × TwoVarSystem.py × ThreeVarSystem.py × RandomCoefficientsGenerator.py × Printer.py ×

```
1 from abc import ABC
2 import numpy.linalg
3 from numpy import array
4
5
6 class TSystemLinearEquation(ABC):
7     def __init__(self, coefficients):
8         self.__coefficients = array(coefficients)
9
10    def kramer_method(self):
11        A, B = self.__coefficients[:, :-1], self.__coefficients[:, -1]
12        roots = []
13        determinant = self.count_determinant(A)
14        if determinant != 0:
15            index = 0
16            while index < len(A):
17                replaced = self.make_replaced_matrix(A, index, B)
18                roots.append(self.count_determinant(replaced))
19                roots[index] /= determinant
20                index += 1
21        return roots
22
23
24    def matrix_method(self):
25        A, B = self.__coefficients[:, :-1], self.__coefficients[:, -1]
26        determinant = numpy.linalg.det(A)
27        if determinant != 0:
28            inverted_matrix = numpy.linalg.inv(A)
29            roots = numpy.matmul(inverted_matrix, B)
30            return roots
31
32    def count_determinant(self, matrix):
33        det = 0
34        if len(matrix) == 2:
35            det += matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0]
36        else:
37            det += matrix[0, 0] * matrix[1, 1] * matrix[2, 2]
```

main.py × LinearSystem.py × TwoVarSystem.py × ThreeVarSystem.py × RandomCoefficientsGenerator.py × Printer.py ×

```
18        roots.append(self.count_determinant(replaced))
19        roots[index] /= determinant
20        index += 1
21
22        return roots
23
24    def matrix_method(self):
25        A, B = self.__coefficients[:, :-1], self.__coefficients[:, -1]
26        determinant = numpy.linalg.det(A)
27        if determinant != 0:
28            inverted_matrix = numpy.linalg.inv(A)
29            roots = numpy.matmul(inverted_matrix, B)
30            return roots
31
32    def count_determinant(self, matrix):
33        det = 0
34        if len(matrix) == 2:
35            det += matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0]
36        else:
37            det += matrix[0, 0] * matrix[1, 1] * matrix[2, 2]
38            det += matrix[0, 1] * matrix[1, 2] * matrix[2, 0]
39            det += matrix[0, 2] * matrix[1, 0] * matrix[2, 1]
40            det -= matrix[0, 2] * matrix[1, 1] * matrix[2, 0]
41            det -= matrix[0, 1] * matrix[1, 0] * matrix[2, 2]
42            det -= matrix[0, 0] * matrix[1, 2] * matrix[2, 1]
43        return det
44
45    def make_replaced_matrix(self, main_matrix, index, replacer):
46        temp = main_matrix
47        i = 0
48        while i < len(temp):
49            main_matrix[i, index] = replacer[i]
50            i += 1
51        return temp
```

main.py × LinearSystem.py × TwoVarSystem.py × ThreeVarSystem.py × RandomCoefficientsGenerator.py × Printer.py ×

```
1 from numpy import array
2 from LinearSystem import TSystemLinearEquation
3
4
5 class TwoSystemLinearEquation(TSystemLinearEquation):
6     def __init__(self, coefficients):
7         super().__init__(coefficients)
8         self.__coefficients = array(coefficients)
9
10     def __str__(self):
11         lin_system = ''
12         for row in self.__coefficients:
13             lin_system += f'{row[0]}x + {row[1]}y = {row[2]}\n'
14
15         return lin_system
16
```

main.py × LinearSystem.py × TwoVarSystem.py × ThreeVarSystem.py × RandomCoefficientsGenerator.py × Printer.py ×

```
1 from numpy import array
2 from LinearSystem import TSystemLinearEquation
3
4
5 class ThreeSystemLinearEquation(TSystemLinearEquation):
6     def __init__(self, coefficients):
7         super().__init__(coefficients)
8         self.__coefficients = array(coefficients)
9
10     def __str__(self):
11         lin_system = ''
12         for row in self.__coefficients:
13             lin_system += f'{row[0]}x + {row[1]}y '
14             lin_system += f'{row[2]}z = {row[3]}\n'
15
16         return lin_system
17
```

main.py × LinearSystem.py × TwoVarSystem.py × ThreeVarSystem.py × RandomCoefficientsGenerator.py × Printer.py ×

```
1 import random
2
3
4 class RandSystem:
5     @staticmethod
6     def generate_random_system(size):
7         lin_system = []
8         for i in range(size):
9             col = []
10             for j in range(size + 1):
11                 col.append(random.randint(0, 20))
12             lin_system.append(col)
13         return lin_system
14
```

```

1 class SystemsPrinter:
2     @staticmethod
3     def print_system(lin_system):
4         print('System:')
5         print(f'{lin_system}')
6
7     @staticmethod
8     def print_roots(roots):
9         if roots is not None:
10             i = 0
11             for root in roots:
12                 print(f'{chr(i + 120)} = {round(root, 3)}')
13                 i += 1
14             print('\n')
15         else:
16             print('Equation has infinite roots')
17

```

Тестування:

C#:

```

System:
7x + 5y = 3
4x + 7y = 5

x = -0,00475624256837099
y = 0,027348394768133177

System:
3x + 9y = 1
8x + 6y = 1

x = 0,05555555555555555
y = 0,09259259259259259

System:
0x + 3y + 3z = 1
5x + 5y + 5z = 8
2x + 1y + 3z = 4

x = -0,04222222222222223
y = 0,007777777777777772
z = -0,018888888888888886

```

System:

$$4x + 5y + 6z = 9$$

$$2x + 0y + 2z = 4$$

$$4x + 9y + 6z = 5$$

$$x = -1$$

$$y = -1$$

$$z = 3$$

## Python:

System:

$$9x + 14y = 19$$

$$11x + 11y = 2$$

$$x = -3.291$$

$$y = 3.473$$

System:

$$17x + 20y = 8$$

$$0x + 2y = 2$$

$$x = -0.706$$

$$y = 0.0$$

System:

$$2x + 9y + 2z = 3$$

$$15x + 5y + 15z = 3$$

$$16x + 13y + 16z = 14$$

Equation has infinite roots

System:

$$11x + 16y + 5z = 0$$

$$16x + 1y + 7z = 17$$

$$5x + 18y + 1z = 9$$

$$x = 6.456$$

$$y = -0.613$$

$$z = -12.241$$