

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
Факультет: «Информационные технологии и прикладная математика»  
Кафедра: 806 «Вычислительная математика и программирование»

Отчет по лабораторным работам  
по курсу «Численные методы»

Выполнил:  
Студент группы 80-304Б  
Сорокин Денис Михайлович

Вариант: 15

Преподаватель:  
Сластушенский Ю.В.

Москва  
2018

# Лабораторная работа №1

## 1.1

### Задание

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

$$15. \begin{cases} -9 \cdot x_1 + 8 \cdot x_2 + 8 \cdot x_3 + 6 \cdot x_4 = -81 \\ -7 \cdot x_1 - 9 \cdot x_2 + 5 \cdot x_3 + 4 \cdot x_4 = -50 \\ -3 \cdot x_1 - x_2 + 8 \cdot x_3 = -69 \\ 3 \cdot x_1 - x_2 - 4 \cdot x_3 - 5 \cdot x_4 = 48 \end{cases}$$

### Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    class LUdecomposition
    {
        public LESystem Sys;
        public Matrix L;
        public Matrix U;

        public LUdecomposition(LESystem s)
        {
            Sys = s;
        }

        public LUdecomposition(Matrix stand, List<double> right)
        {
            Sys = new LESystem(stand, right);
        }

        public void FindSolution()
        {
            var n = U.columns;
            Sys.Solution = new List<double>();

            for (int i = 0; i < n; ++i)
                Sys.Solution.Add(0);

            for (int i = n - 1; i >= 0; --i)
            {
                if (i == n - 1)
                {
                    Sys.Solution[i] = Sys.Right[i] / U.GetElement(i, i);
                    continue;
                }

                double sum = 0;
```

```

        for (int j = i + 1; j < n; ++j)
            sum += U.GetElement(i, j) * Sys.Solution[j];

        Sys.Solution[i] = (Sys.Right[i] - sum) / U.GetElement(i, i);
    }
}

public void LU_Algorithm()
{
    var n = Sys.Mtx.columns;
    U = new Matrix(Sys.Mtx);

    L = new Matrix(n);

    for (int j = 0; j < n; ++j)
    {
        for (int i = j + 1; i < n; ++i)
        {
            L.SetElement(i, j, U.GetElement(i, j) / U.GetElement(j, j));

            for (int k = j; k < n; ++k)
            {
                U.SetElement(i, k, U.GetElement(i, k) - U.GetElement(j, k) * L.GetElement(i, j));
            }
        }
    }

    //Console.WriteLine("L Matrix:");
    //L.Print();
    //Console.WriteLine("U Matrix:");
    //U.Print();

    for (int i = 0; i < n; ++i)
    {
        double sum = 0;

        for (int j = 0; j < i; ++j)
            sum += L.GetElement(i, j) * Sys.Right[j];

        Sys.Right[i] -= sum;
    }

    //Console.WriteLine("Right side:");
    //PrintRight();

    FindSolution();
}

public void FindDeter()
{
    U.det = 1;

    for (int i = 0; i < U.columns; ++i)
    {
        U.det *= U.GetElement(i, i);
    }

    Console.WriteLine("Deter:");
    Console.WriteLine(U.det);
}

public void FindInvertMtx()
{
    var n = Sys.Mtx.columns;
    var InvertMtx = new Matrix(n);

    for (int i = 0; i < n; ++i)
    {
        var tempList = new List<double>();

```

```

        for (int j = 0; j < n; ++j)
        {
            if (j == i)
                tempList.Add(1);
            else
                tempList.Add(0);
        }

        var tempSys = new LSystem(Sys.Mtx, tempList);
        var tempLU = new LUdecomposition(tempSys);
        tempLU.LU_Algorithm();

        for (int j = 0; j < n; ++j)
        {
            //InvertMtx.SetElement(j, i, tempSys.Solution[j]);
            InvertMtx.mtx[j][i] = tempSys.Solution[j];
        }
    }
    Console.WriteLine("Invert mtx:");
    InvertMtx.Print();
}
}
}

```

## Результат выполнения

Решаем систему LU - методом:

```

*****
-9x1 +8x2 +8x3 +6x4 = -81
-7x1 -9x2 +5x3 +4x4 = -50
-3x1 -1x2 +8x3 = -69
+3x1 -1x2 -4x3 -5x4 = 48
*****

```

L Matrix:

```

*****
0,0000  0,0000  0,0000  0,0000
0,7778  0,0000  0,0000  0,0000
0,3333  0,2409  0,0000  0,0000
-0,333  -0,109  -0,261  0,0000
*****

```

U Matrix:

```

*****
-9,000  8,0000  8,0000  6,0000
0,0000 -15,222 -1,222 -0,667
0,0000  0,0000  5,6277 -1,839
0,0000  0,0000  0,0000 -3,553
*****

```

Right side:

```

-81
13
-45,1313868613139
10,6575875486381

```

Solution:

```

X1 = -1,00
X2 = 0,00
X3 = -9,00
X4 = -3,00

```

Deter:

```

-2739

```

Invert mtx:

```

*****
-0,140  -0,105  0,0796  -0,252
0,0573  -0,061  -0,009  0,0197
-0,045  -0,047  0,1537  -0,092
-0,059  -0,013  -0,073  -0,281

```

## 1.2

### Задание

1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

$$15. \quad \begin{cases} 16 \cdot x_1 - 8 \cdot x_2 = 0 \\ -7 \cdot x_1 - 16 \cdot x_2 + 5 \cdot x_3 = -123 \\ 4 \cdot x_2 + 12 \cdot x_3 + 3 \cdot x_4 = -68 \\ -4 \cdot x_3 + 12 \cdot x_4 - 7 \cdot x_5 = 104 \\ -x_4 + 7 \cdot x_5 = 20 \end{cases}$$

### Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    static class ThomasAlgorythm
    {
        public static void TA(Matrix matr)
        {
            var P = new List<double>();
            var Q = new List<double>();
            var X = new List<double>();

            for (var i = 0; i < matr.rows; ++i)
            {
                double a = 0;
                double b = 0;
                double c = 0;
                double d = matr.GetElement(i, matr.columns - 1);

                if (i - 1 >= 0)
                    a = matr.GetElement(i, i - 1);

                b = matr.GetElement(i, i);

                if (i + 1 < matr.rows)
                    c = matr.GetElement(i, i + 1);

                P.Add(CalcP(a, b, c, P, i));
                Q.Add(CalcQ(a, b, c, d, P, Q, i));
                X.Add(0);
            }

            X[matr.rows - 1] = Q[matr.rows - 1];

            for (int i = matr.rows - 2; i >= 0; --i)
```

```

        X[i] = P[i] * X[i + 1] + Q[i];

        Console.WriteLine("Ответ:");
        for (int i = 0; i < matr.rows; ++i)
        {
            Console.WriteLine("x" + (i + 1) + " = " + X[i]);
        }
    }

    private static double CalcP(double a, double b, double c, List<double> p, int i)
    {
        if (i == 0)
            return -c / b;

        return -c / (b + a * p[i - 1]);
    }

    private static double CalcQ(double a, double b, double c, double d, List<double>
p, List<double> q, int i)
    {
        if (i == 0)
            return d / b;

        return (d - a * q[i - 1]) / (b + a * p[i - 1]);
    }
}

```

## Результат выполнения

Решаем систему методом прогонки:

\*\*\*\*\*

```

+16x1 -8x2 = 0
-7x1 -16x2 +5x3 = -123
+4x2 +12x3 +3x4 = -68
-4x3 +12x4 -7x5 = 104
-1x4 +7x5 = 20
*****

```

Ответ:

```

x1 = 2
x2 = 4
x3 = -9
x4 = 8
x5 = 4-0,059 -0,013 -0,073 -0,281

```

## 1.3

### Задание

1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

$$15. \quad \begin{cases} -14 \cdot x_1 + 6 \cdot x_2 + x_3 - 5 \cdot x_4 = 95 \\ -6 \cdot x_1 + 27 \cdot x_2 + 7 \cdot x_3 - 6 \cdot x_4 = -41 \\ 7 \cdot x_1 - 5 \cdot x_2 - 23 \cdot x_3 - 8 \cdot x_4 = 69 \\ 3 \cdot x_1 - 8 \cdot x_2 - 7 \cdot x_3 + 26 \cdot x_4 = 27 \end{cases}$$

### Код программы

#### SimpleIterationMethod.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    class SimpleIterationMethod
    {
        protected LESystem sys;
        protected Matrix Alpha;
        protected List<double> Beta = new List<double>();
        protected double eps = 0;

        public SimpleIterationMethod(Matrix m, double e)
        {
            sys = new LESystem(m);
            eps = e;

            CalcBeta();
            CalcAlpha();
            Console.WriteLine("Alpha Matrix:");
            Alpha.Print();
            Console.WriteLine("Beta Vector:");
            PrintBeta();
            Console.WriteLine("Alpha measure:");
            Alpha.CalcMeasure();

            Console.WriteLine("||Alpha||c = " + Alpha.measure);

            if (Alpha.measure >= 1)
            {
                Console.WriteLine("Не выполнено достаточное условие сходимости метода");
                return;
            }

            Console.WriteLine("Выполнено достаточное условие сходимости метода");
        }
    }
}
```

```

        IterationProcces();
    }

    virtual protected void IterationProcces()
    {
        var X = Beta;
        var X0 = new List<double>();

        for (int i = 0; i < Beta.Count; ++i)
            X0.Add(0);

        var eps0 = CalcEps(Vector.SumVec(X, Vector.DigitOnVector(-1, X0)));

        Console.WriteLine("Start iterations:");

        for (int i = 0; eps0 > eps; ++i)
        {
            Console.WriteLine("Iteration #" + i + "\n");
            Console.WriteLine(eps0 + " > " + eps);
            Console.Write("X = (");

            for (int j = 0; j < X.Count; ++j)
            {
                if (j + 1 == X.Count)
                    Console.Write("{0:0.00}", X[j]);
                else
                    Console.Write("{0:0.00 }", X[j]);
            }
            Console.WriteLine(")");

            X0 = X;
            X = Vector.SumVec(Beta, Alpha * X);
            eps0 = CalcEps(Vector.SumVec(X, Vector.DigitOnVector(-1, X0)));
            Console.WriteLine("*****");
        }

        sys.Solution = X;
        sys.PrintSolution();
    }

    virtual protected double CalcEps(List<double> list)
    {
        double max = 0;

        foreach (var el in list)
        {
            if (Math.Abs(el) > max)
                max = Math.Abs(el);
        }

        return max;
    }

    private void PrintBeta()
    {
        foreach (var el in Beta)
        {
            Console.WriteLine(el);
        }
    }

    private void CalcAlpha()
    {
        var n = sys.Mtx.columns;
    }

```



```

        Alpha = new Matrix(n);

        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                if (i == j)
                    continue;

                Alpha.mtx[i][j] = - sys.Mtx.mtx[i][j] / sys.Mtx.mtx[i][i];
            }
        }

        private void CalcBeta()
        {
            var n = sys.Mtx.columns;

            for (int i = 0; i < n; ++i)
                Beta.Add(sys.Right[i] / sys.Mtx.mtx[i][i]);
        }
    }
}

```

## SeidelMethod.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    class SeidelMethod : SimpleIterationMethod
    {
        public SeidelMethod(Matrix m, double e) : base(m, e)
        {
        }

        protected override void IterationProcces()
        {
            var X = new List<double>();

            for (int k = 0; k < Beta.Count; ++k)
                X.Add(Beta[k]);

            var X0 = new List<double>();

            for (int i = 0; i < Beta.Count; ++i)
                X0.Add(0);

            var eps0 = CalcEps(Vector.SumVec(X, Vector.DigitOnVector(-1, X0)));

            Console.WriteLine("Start iterations:");

            for (int i = 0; eps0 > eps; ++i)
            {
                Console.WriteLine("Iteration #" + i + "\n");
                Console.WriteLine(eps0 + " > " + eps);
            }
        }
    }
}

```

```

        Console.Write("X = (");

        for (int j = 0; j < X.Count; ++j)
        {
            if (j + 1 == X.Count)
                Console.Write("{0:0.00}", X[j]);
            else
                Console.Write("{0:0.00 }", X[j]);
        }
        Console.WriteLine(")");

        for (int k = 0; k < X.Count; ++k)
            X0[k] = X[k];

        X = SeidelIter(X);
        eps0 = CalcEps(Vector.SumVec(X, Vector.DigitOnVector(-1, X0)));
        Console.WriteLine("*****");
    }

    sys.Solution = X;
    sys.PrintSolution();
}

private List<double> SeidelIter(List<double> x)
{
    for (int i = 0; i < x.Count; ++i)
        x[i] = Beta[i] + Vector.Scalar(Alpha.mtx[i], x);

    return x;
}
}
}

```

## Результат выполнения

Решаем систему методом простых итераций:

```

*****
-14x1 +6x2 +x3 -5x4 = 95
-6x1 +27x2 +7x3 -6x4 = -41
+7x1 -5x2 -23x3 -8x4 = 69
+3x1 -8x2 -7x3 +26x4 = 27
*****

```

Введите точность:

0,01

Alpha Matrix:

```

*****
0,0000  0,4286  0,0714  -0,357
0,2222  0,0000  -0,259  0,2222
0,3043  -0,217  0,0000  -0,348
-0,115  0,3077  0,2692  0,0000
*****

```

Beta Vector:

```

-6,78571428571429
-1,51851851851852
-3
1,03846153846154

```

Alpha measure:

||Alpha||<sub>c</sub> = 0,869565217391304

Выполнено достаточное условие сходимости метода

Start iterations:

```

Iteration #0

6,78571428571429 > 0,01
X = (-6,79 -1,52 -3,00 1,04)
*****

Iteration #1

2,0963086832652 > 0,01
X = (-8,02 -2,02 -5,10 0,55)
*****

Iteration #2

0,575438512311422 > 0,01
X = (-8,21 -1,86 -5,19 -0,03)
*****

Iteration #3

0,266981594764017 > 0,01
X = (-7,94 -2,00 -5,08 0,02)
*****

Iteration #4

0,0971181876118719 > 0,01
X = (-8,01 -1,96 -4,99 -0,03)
*****

Iteration #5

0,0510532698118156 > 0,01
X = (-7,97 -2,01 -5,00 0,02)
*****

Iteration #6

0,0396640972777993 > 0,01
X = (-8,01 -1,99 -4,99 -0,01)
*****

Iteration #7

0,0191686093519117 > 0,01
X = (-7,99 -2,01 -5,00 0,01)
*****

Iteration #8

0,0124113333096254 > 0,01
X = (-8,00 -2,00 -5,00 0,00)
*****

Solution:
X1 = -8,00
X2 = -2,00
X3 = -5,00
X4 = 0,00
Решаем систему методом Зейделя:
*****
-14x1 +6x2 +x3 -5x4 = 95
-6x1 +27x2 +7x3 -6x4 = -41
+7x1 -5x2 -23x3 -8x4 = 69
+3x1 -8x2 -7x3 +26x4 = 27
*****

Alpha Matrix:
*****
0,0000 0,4286 0,0714 -0,357
0,2222 0,0000 -0,259 0,2222
0,3043 -0,217 0,0000 -0,348
-0,115 0,3077 0,2692 0,0000
*****

Beta Vector:

```

```

-6,78571428571429
-1,51851851851852
-3
1,03846153846154
Alpha measure:
||Alpha||c = 0,869565217391304
Выполнено достаточное условие сходимости метода
Start iterations:
Iteration #0

6,78571428571429 > 0,01
X = (-6,79 -1,52 -3,00 1,04)
*****
Iteration #1

2,30419888753222 > 0,01
X = (-8,02 -2,29 -5,30 -0,17)
*****
Iteration #2

0,331992215941065 > 0,01
X = (-8,09 -1,98 -4,97 0,02)
*****
Iteration #3

0,0893526671933458 > 0,01
X = (-8,00 -2,00 -5,01 0,00)
*****
Solution:
X1 = -8,00
X2 = -2,00
X3 = -5,00
X4 = 0,00

```

## 1.4

### Задание

1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

$$15. \begin{pmatrix} -3 & -1 & 3 \\ -1 & 8 & 1 \\ 3 & 1 & 5 \end{pmatrix}$$

### Код программы

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    static class JacobiMethod
    {
        static Matrix A;
        static double Eps;
        static int N;

        public static void JM(Matrix mtx, double e)
        {
            A = mtx;
            Eps = e;
            N = A.columns;

            IterationProccess();
        }

        private static void IterationProccess()
        {
            Console.WriteLine("Start iterations:");

            double eps0 = CalcEps();

            var finalU = new Matrix(A.columns);
            finalU.MakeUnitMatrix();

            for (int i = 0; eps0 > Eps; ++i)
            {
                Console.WriteLine("Iteration #" + i + "\n");
                Console.WriteLine(eps0 + " > " + Eps);

                var U = MakeU();
                U.Print("U");

                A = U.FindTransparent() * A * U;

                A.Print("A");
                eps0 = CalcEps();

                Console.WriteLine("eps = " + eps0 + "\n");
            }
        }
    }
}
```

```

        finalU *= U;
    }

    WriteEigenvalues();

    Console.WriteLine("Eigenvectors:");

    Console.WriteLine("x1      x2      x3");
    finalU.Print();
}

private static void WriteEigenvalues()
{
    Console.WriteLine("Eigenvalues:");

    for (int i = 0; i < A.columns; ++i)
        Console.WriteLine("Lyambda" + (i + 1) + " = " + A.mtx[i][i]);

    Console.WriteLine();
}

private static Matrix MakeU()
{
    double maxEl = 0;

    int iMax = 0;
    int jMax = 0;

    for (int i = 0; i < N; ++i)
    {
        for (int j = i + 1; j < N; ++j)
        {
            if (Math.Abs(A.mtx[i][j]) > maxEl)
            {
                iMax = i;
                jMax = j;
                maxEl = Math.Abs(A.mtx[i][j]);
            }
        }
    }

    double phi = FindPhi(iMax, jMax);
    Console.WriteLine("\nSin: " + Math.Sin(phi));
    Console.WriteLine("Cos: " + Math.Cos(phi));
    Console.WriteLine("Phi: " + phi + "\n");
    Matrix U = new Matrix(N);

    U.mtx[iMax][jMax] = -Math.Sin(phi);
    U.mtx[jMax][iMax] = Math.Sin(phi);

    for (int i = 0; i < N; ++i)
        U.mtx[i][i] = (i == iMax || i == jMax) ? Math.Cos(phi) : 1;

    return U;
}

private static double FindPhi(int iMax, int jMax)
{
    if (A.mtx[iMax][iMax] == A.mtx[jMax][jMax])
        return Math.PI / 4;

    return 0.5 * Math.Atan(2.0 * A.mtx[iMax][jMax] / (A.mtx[iMax][iMax] -
A.mtx[jMax][jMax]));
}

```

```

    }

    private static double CalcEps()
    {
        double sum = 0;

        for(int i = 0; i < N; ++i)
        {
            for (int j = i + 1; j < N; ++j)
            {
                sum += Math.Pow(A.mtx[i][j], 2);
            }
        }

        return Math.Pow(sum, 0.5);
    }
}

```

## Результат выполнения

4

Ищем собственные значения и собственные векторы методом вращений:

Введите точность:

0,01

\*\*\*\*\*

4,0000 2,0000 1,0000

2,0000 5,0000 3,0000

1,0000 3,0000 6,0000

\*\*\*\*\*

Start iterations:

Iteration #0

3,74165738677394 > 0,01

Sin: -0,646374896130196

Cos: 0,763019982472726

Phi: -0,702823824690135

Matrix U:

\*\*\*\*\*

1,0000 0,0000 0,0000

0,0000 0,7630 0,6464

0,0000 -0,646 0,7630

\*\*\*\*\*

Matrix A:

\*\*\*\*\*

4,0000 0,8797 2,0558

0,8797 2,4586 0,0000

2,0558 0,0000 8,5414

\*\*\*\*\*

eps = 2,23606797749979

Iteration #1

2,23606797749979 > 0,01

Sin: -0,35963951199841

Cos: 0,933091325331848

Phi: -0,367881527339301

Matrix U:

\*\*\*\*\*

0,9331 0,0000 0,3596

```
0,0000 1,0000 0,0000
-0,360 0,0000 0,9331
*****
```

```
Matrix A:
*****
3,2076 0,8208 0,0000
0,8208 2,4586 0,3164
0,0000 0,3164 9,3337
*****
```

eps = 0,879665068815256

Iteration #2

0,879665068815256 > 0,01

```
Sin: 0,540783028649252
Cos: 0,841162122259997
Phi: 0,571367720244857
```

```
Matrix U:
*****
0,8412 -0,541 0,0000
0,5408 0,8412 0,0000
0,0000 0,0000 1,0000
*****
```

```
Matrix A:
*****
3,7353 0,0000 0,1711
0,0000 1,9309 0,2661
0,1711 0,2661 9,3337
*****
```

eps = 0,316362316070766

Iteration #3

0,316362316070766 > 0,01

```
Sin: -0,0358779793814643
Cos: 0,999356178044396
Phi: -0,0358856810418495
```

```
Matrix U:
*****
1,0000 0,0000 0,0000
0,0000 0,9994 0,0359
0,0000 -0,036 0,9994
*****
```

```
Matrix A:
*****
3,7353 -0,006 0,1710
-0,006 1,9214 0,0000
0,1710 0,0000 9,3433
*****
```

eps = 0,171083371435241

Iteration #4

0,171083371435241 > 0,01

```
Sin: -0,0304452992909225
Cos: 0,999536434429024
```



Phi: -0,03045000462761

Matrix U:

```
*****
0,9995  0,0000  0,0304
0,0000  1,0000  0,0000
-0,030  0,0000  0,9995
*****
```

Matrix A:

```
*****
3,7301  -0,006  0,0000
-0,006  1,9214  0,000
0,0000  0,000  9,3485
*****
```

eps = 0,00613812567286496

Eigenvalues:

Lyambda1 = 3,73013831329282  
Lyambda2 = 1,92136775705923  
Lyambda3 = 9,34849392964795

Eigenvectors:

```
x1      x2      x3
*****
0,7741  -0,517  0,3650
0,1978  0,7454  0,6366
-0,601  -0,421  0,6793
*****
```

## 1.5

### Задание

1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

$$15. \begin{pmatrix} 1 & 7 & -1 \\ -2 & 2 & -2 \\ 9 & -7 & 3 \end{pmatrix}$$

### Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    public static class QRAlgorhythm
    {
        private static Matrix A;
        private static Matrix Q;
        private static Matrix R;
        private static double eps;

        public static void QR(Matrix m, double e)
        {
            A = m;
            eps = e;

            ItarationProccess();
        }

        public static void QR(Matrix m)
        {
            A = m;

            FindQR();

            (R * Q).Print("R * Q");
        }

        private static void ItarationProccess()
        {
            Console.WriteLine("Start iterations:");

            FindQR();

            for (int i = 0; FindEps(A.GetColumn(0)) > eps; i++)
            {
                Console.WriteLine("Iteration #" + i + "\n");
            }
        }
    }
}
```

```

        A = R * Q;

        A.Print("A");

        FindQR();
    }

    A.Print("Final A");

    var l1 = A.mtx[0][0];

    var c = A.mtx[2][2] * A.mtx[1][1] - A.mtx[1][2] * A.mtx[2][1];
    var b = -A.mtx[2][2] - A.mtx[1][1];

    var l2R = -b / 2;
    var l3R = l2R;

    var l2I = Math.Sqrt(Math.Abs(b * b - 4 * c)) / 2;
    var l3I = - l2I;

    Console.WriteLine("Eigenvalues:");
    Console.WriteLine("Lyambda 1 = " + l1);
    Console.WriteLine($"Lyambda 2 = {l2R} + {l2I}i");
    Console.WriteLine($"Lyambda 3 = {l3R} - {l2I}i");

}

public static double FindEps(List<double> l)
{
    double sum = 0;

    //for (int i = 1; i < l.Count; i++)
    //    sum += Math.Pow(l[i], 2);

    //return Math.Pow(sum, 0.5);

    return l[l.Count - 1];
}

private static void FindQR()
{
    R = new Matrix(A);
    Q = new Matrix(A.columns);
    Q.MakeUnitMatrix();

    for (int i = 0; i < R.rows - 1; i++)
    {
        var b = R.GetColumn(i);
        var v = CalcV(b, i);
        var H = HouseholderCalc(v);

        Q *= H;

        R = H * R;
    }

    Q.Print("Q");
    R.Print("R");
}

public static Matrix HouseholderCalc(List<double> v)

```

```

    {
        Matrix E = new Matrix(v.Count);
        E.MakeUnitMatrix();

        var koef = -2 / Vector.Scalar(v, v);

        Matrix H = E + koef * Vector.MultiplyVectors(v, v);

        return H;
    }

    public static double CalcNorm(List<double> b)
    {
        double norm = 0;

        foreach (var el in b)
            norm += Math.Pow(el, 2);

        norm = Math.Pow(norm, 0.5);

        return norm;
    }

    public static List<double> CalcV(List<double> b, int i)
    {
        var e = new List<double>();
        var v = b;

        for (int j = 0; j < b.Count; j++)
        {
            if (i > j)
                v[j] = 0;

            if (i == j)
                e.Add(1);
            else
                e.Add(0);
        }

        var res = Vector.SumVec(v, Vector.DigitOnVector(Math.Sign(b[i]) *
CalcNorm(b), e));

        return res;
    }
}

```

## Результат выполнения

5  
Ищем собственные значения и собственные векторы QR методом:  
Введите точность:  
0,001

```

*****
1,0000  7,0000  -1,000
-2,000  2,0000  -2,000
9,0000  -7,000  3,0000
*****

```

Start iterations:

Matrix Q:

```

*****
-0,108  -0,993  -0,056
0,2157  -0,078  0,9733
-0,970  0,0930  0,2225
*****

```

```
Matrix R:
*****
-9,274  6,4700  -3,235
0,0000  -7,755  1,4274
0,0000  0,0000  -1,224
*****
```

Iteration #0

```
Matrix A:
*****
5,5349  8,3999  6,0936
-3,058  0,7374  -7,231
1,1875  -0,114  -0,272
*****
```

```
Matrix Q:
*****
-0,860  -0,510  -0,017
0,4753  -0,813  0,3356
-0,185  0,2807  0,9419
*****
```

```
Matrix R:
*****
-6,434  -6,855  -8,628
0,0000  -4,912  2,6995
0,0000  0,0000  -2,785
*****
```

Iteration #1

```
Matrix A:
*****
3,8697  6,4315  -10,320
-2,833  4,7530  0,8943
0,5139  -0,782  -2,623
*****
```

```
Matrix Q:
*****
-0,802  -0,597  -0,006
0,5873  -0,791  0,1704
-0,107  0,1331  0,9854
*****
```

```
Matrix R:
*****
-4,823  -2,285  9,0841
0,0000  -7,704  5,1029
0,0000  0,0000  -2,368
*****
```

Iteration #2

```
Matrix A:
*****
1,5595  5,8959  8,5915
-5,068  6,7743  3,7157
0,2524  -0,315  -2,334
*****
```

```
Matrix Q:
*****
-0,294  -0,956  -0,003
0,9547  -0,294  0,0489
```

```

-0,048  0,0117  0,9988
*****

Matrix R:
*****
-5,309  4,7503  1,1344
0,0000  -7,628  -9,330
0,0000  0,0000  -2,173
*****

Iteration #3

Matrix A:
*****
6,0407  3,6933  1,3800
-6,839  2,1298  -9,692
0,1033  -0,025  -2,171
*****

Matrix Q:
*****
-0,662  -0,750  -0,001
0,7494  -0,662  0,0140
-0,011  0,0084  0,9999
*****

Matrix R:
*****
-9,125  -0,848  -8,153
0,0000  -4,178  5,3629
0,0000  0,0000  -2,308
*****

Iteration #4

Matrix A:
*****
5,4971  7,3328  -8,153
-3,192  2,8107  5,3037
0,0261  -0,019  -2,308
*****

Matrix Q:
*****
-0,865  -0,502  0,000
0,5021  -0,865  0,0077
-0,004  0,0065  1,0000
*****

Matrix R:
*****
-6,357  -4,930  9,7230
0,0000  -6,113  -0,507
0,0000  0,0000  -2,265
*****

Iteration #5

Matrix A:
*****
2,9817  7,5181  9,6868
-3,068  5,2828  -0,554
0,0093  -0,015  -2,265
*****

Matrix Q:
*****

```

```

-0,697  -0,717  0,000
0,7171  -0,697  0,0029
-0,002  0,0020  1,0000
*****

```

```

Matrix R:
*****
-4,278  -1,452  -7,144
0,0000  -9,073  -6,564
0,0000  0,0000  -2,267
*****

```

Iteration #6

```

Matrix A:
*****
1,9561  4,0656  -7,148
-6,492  6,3111  -6,591
0,0049  -0,004  -2,267
*****

```

```

Matrix Q:
*****
-0,289  -0,957  -0,0001
0,9575  -0,289  0,0007
-0,001  0,0002  1,0000
*****

```

```

Matrix R:
*****
-6,780  4,8699  -4,247
0,0000  -5,713  8,7451
0,0000  0,0000  -2,272
*****

```

Iteration #7

```

Matrix A:
*****
6,6219  5,0861  -4,243
-5,477  1,6498  8,7409
0,0017  0,000  -2,272
*****

```

```

Matrix Q:
*****
-0,771  -0,637  0,0000
0,6373  -0,771  0,0003
0,000  0,0002  1,0000
*****

```

```

Matrix R:
*****
-8,593  -2,868  8,8407
0,0000  -4,513  -4,032
0,0000  0,0000  -2,269
*****

```

Iteration #8

```

Matrix A:
*****
4,7925  7,6886  8,8400
-2,875  3,4767  -4,033
0,0004  0,000  -2,269
*****

```

```

Matrix Q:
*****
-0,857  -0,514  0,0000
0,5145  -0,857  0,0001
-0,0001  0,0001  1,0000
*****

Matrix R:
*****
-5,589  -4,804  -9,655
0,0000  -6,937  -1,090
0,0000  0,0000  -2,270
*****

Matrix Final A:
*****
4,7925  7,6886  8,8400
-2,875  3,4767  -4,033
0,0004  0,000  -2,269
*****

Eigenvalues:
Lyambda 1 = -2,26917686605184
Lyambda 2 = 4,13458843302592 + 4,65565448033191i
Lyambda 3 = 4,13458843302592 - 4,65565448033191i

```

## Вспомогательные классы

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    public class Matrix
    {
        public List<List<double>> mtx = new List<List<double>>();
        public int rows;
        public int columns;
        public double det;
        public double measure;

        public Matrix()
        {
            rows = 0;
            columns = 0;
        }

        public Matrix(int a, int b)
        {
            rows = a;
            columns = b;

            for (int i = 0; i < a; ++i)
            {
                mtx.Add(new List<double>());

                for (int j = 0; j < b; ++j)
                    mtx[i].Add(0);
            }
        }

        public Matrix(int a)
        {
            rows = a;
            columns = a;
        }
    }
}

```



```

        for (int i = 0; i < a; ++i)
        {
            mtx.Add(new List<double>());

            for (int j = 0; j < a; ++j)
                mtx[i].Add(0);
        }
    }

    public Matrix(Matrix m)
    {
        rows = m.rows;
        columns = m.columns;

        for (int i = 0; i < rows; ++i)
        {
            mtx.Add(new List<double>());

            for (int j = 0; j < columns; ++j)
                mtx[i].Add(m.GetElement(i, j));
        }
    }

    public void MakeUnitMatrix()
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                if (i == j)
                    mtx[i][j] = 1;
                else
                    mtx[i][j] = 0;
            }
        }
    }

    #region Math Operations

    // equalation

    // matrix multiplication
    public static Matrix operator *(Matrix left, Matrix right)
    {
        if (left.columns != right.rows)
        {
            Console.WriteLine("Несовместимые матрицы!");
            return null;
        }

        Matrix res = new Matrix(left.rows, right.columns);

        for (int i = 0; i < left.rows; ++i)
        {
            for (int j = 0; j < right.columns; ++j)
            {
                for (int k = 0; k < left.columns; ++k)
                    res.mtx[i][j] += left.mtx[i][k] * right.mtx[k][j];
            }
        }
        return res;
    }

    public List<double> GetColumn(int i)
    {
        if (i + 1 > columns)
            return null;

        var col = new List<double>();

        for (int j = 0; j < rows; j++)
            col.Add(mtx[j][i]);

        return col;
    }
}

```

```

public static Matrix operator *(double digit, Matrix right)
{
    Matrix res = new Matrix(right.rows, right.columns);

    for (int i = 0; i < right.rows; ++i)
    {
        for (int j = 0; j < right.columns; ++j)
            res.mtx[i][j] = digit * right.mtx[i][j];
    }
    return res;
}

public static Matrix operator +(Matrix left, Matrix right)
{
    if (left.columns != right.columns || left.rows != right.rows)
    {
        Console.WriteLine("Несовместимые матрицы!");
        return null;
    }

    Matrix res = new Matrix(left.rows, left.columns);

    for (int i = 0; i < left.rows; ++i)
    {
        for (int j = 0; j < left.columns; ++j)
            res.mtx[i][j] = left.mtx[i][j] + right.mtx[i][j];
    }

    return res;
}

// matrix on vector multiplication
public static List<double> operator *(Matrix left, List<double> right)
{
    if (left.columns != right.Count)
    {
        Console.WriteLine("Несовместимые матрица и вектор!");
        return null;
    }

    var res = new List<double>(right.Count);

    for (int i = 0; i < right.Count; ++i)
    {
        res.Add(0);
        for (int j = 0; j < left.columns; ++j)
            res[i] += left.mtx[i][j] * right[j];
    }
    return res;
}

#endregion

public void ReadFromFile(string path)
{
    var lines = File.ReadAllLines(path).ToList();
    int i = 0;

    foreach (var line in lines)
    {
        mtx.Add(new List<double>());

        var str = line.Split(' ');

        foreach (var s in str)
        {
            mtx[i].Add(Convert.ToDouble(s));
        }
        ++i;
    }

    rows = mtx.Count;
    columns = mtx[0].Count;
}

```

```

    }

    public void Print(string str = " ")
    {
        if (mtx == null)
            return;

        if (str != " ")
            Console.WriteLine("Matrix " + str + ":");
        Console.WriteLine("*****");

        for (int i = 0; i < mtx.Count; ++i)
        {
            for (int j = 0; j < mtx[i].Count; ++j)
            {
                double eps = 0.0001;
                if (mtx[i][j] + eps < 0)
                    Console.Write("{0:0.000 }", mtx[i][j]);
                else
                    Console.Write("{0:0.0000 }", mtx[i][j]);
            }

            Console.WriteLine();
        }

        Console.WriteLine("*****\n");
    }

    public void PrintLikeSystem()
    {
        if (mtx == null)
            return;

        Console.WriteLine("*****");
        for (int i = 0; i < mtx.Count; ++i)
        {
            for (int j = 0; j < mtx[i].Count; ++j)
            {
                if (j == mtx[i].Count - 1)
                {
                    Console.Write("=" + mtx[i][j]);
                    continue;
                }

                if (mtx[i][j] == 0)
                    continue;

                if (mtx[i][j] > 0)
                    Console.Write("+");

                if (mtx[i][j] == 1)
                {
                    Console.Write("x" + (j + 1) + " ");
                    continue;
                }

                Console.Write(mtx[i][j] + "x" + (j + 1) + " ");
            }

            Console.WriteLine();
        }
        Console.WriteLine("*****\n");
    }

    public void SwipeRows(int f, int s)
    {
        var tmp = new List<double>();

        tmp = mtx[s];
        mtx[s] = mtx[f];
        mtx[f] = tmp;
    }

    public double GetElement(int a, int b)
    {
        return mtx[a][b];
    }

```

```

    }

    public void SetElement(int a, int b, double val)
    {
        mtx[a][b] = val;
    }

    public void CalcMeasure()
    {
        double max = 0;

        for (int i = 0; i < rows; ++i)
        {
            double temp = 0;

            for (int j = 0; j < columns; ++j)
            {
                temp += Math.Abs(mtx[i][j]);
            }

            if (temp > max)
                max = temp;
        }

        measure = max;
    }

    public Matrix FindTransparent()
    {
        var trans = new Matrix(columns, rows);

        for (int i = 0; i < trans.rows; ++i)
        {
            for (int j = 0; j < trans.columns; ++j)
            {
                trans.mtx[i][j] = mtx[j][i];
            }
        }

        return trans;
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab1
{
    static class Vector
    {
        public static List<double> DigitOnVector(double d, List<double> right)
        {
            var res = new List<double>();

            foreach (var el in right)
                res.Add(d * el);

            return res;
        }

        public static List<double> SumVec(List<double> left, List<double> right)
        {
            if (left.Count != right.Count)
            {
                Console.WriteLine("Несовместимые векторы");
                return null;
            }
        }
    }
}

```

```

    }

    var res = new List<double>();

    for (int i = 0; i < left.Count; ++i)
        res.Add(left[i] + right[i]);

    return res;
}

public static double Scalar(List<double> a, List<double> x)
{
    double res = 0;

    for (int i = 0; i < a.Count; ++i)
        res += a[i] * x[i];

    return res;
}

public static Matrix MultiplyVectors(List<double> left, List<double> right)
{
    if (left.Count != right.Count)
    {
        Console.WriteLine("Несовместимые векторы!");
        return null;
    }

    var res = new Matrix(left.Count);

    for (int i = 0; i < left.Count; i++)
    {
        for (int j = 0; j < right.Count; j++)
        {
            res.mtx[i][j] = left[i] * right[j];
        }
    }

    return res;
}
}
}
}

```

## Выводы

Мною были реализованы такие методы решения систем линейных алгебраических уравнений как LU-разложение, метод прогонки, метод простых итераций и метод Зейделя. Программирование этих методов оказалось довольно увлекательным занятием, они помогут мне в дальнейших лабораторных.

Также были реализованы методы нахождения собственных векторов и собственных значений матриц. В них входят методы вращения Якоби и методы QR-разложения. Больше всего пришлось повозиться с методом QR-разложения: долгое время не мог разобраться, почему закидывается моя программа. Но решение было найдено пересмотром условия остановки цикла.

# Лабораторная работа №2

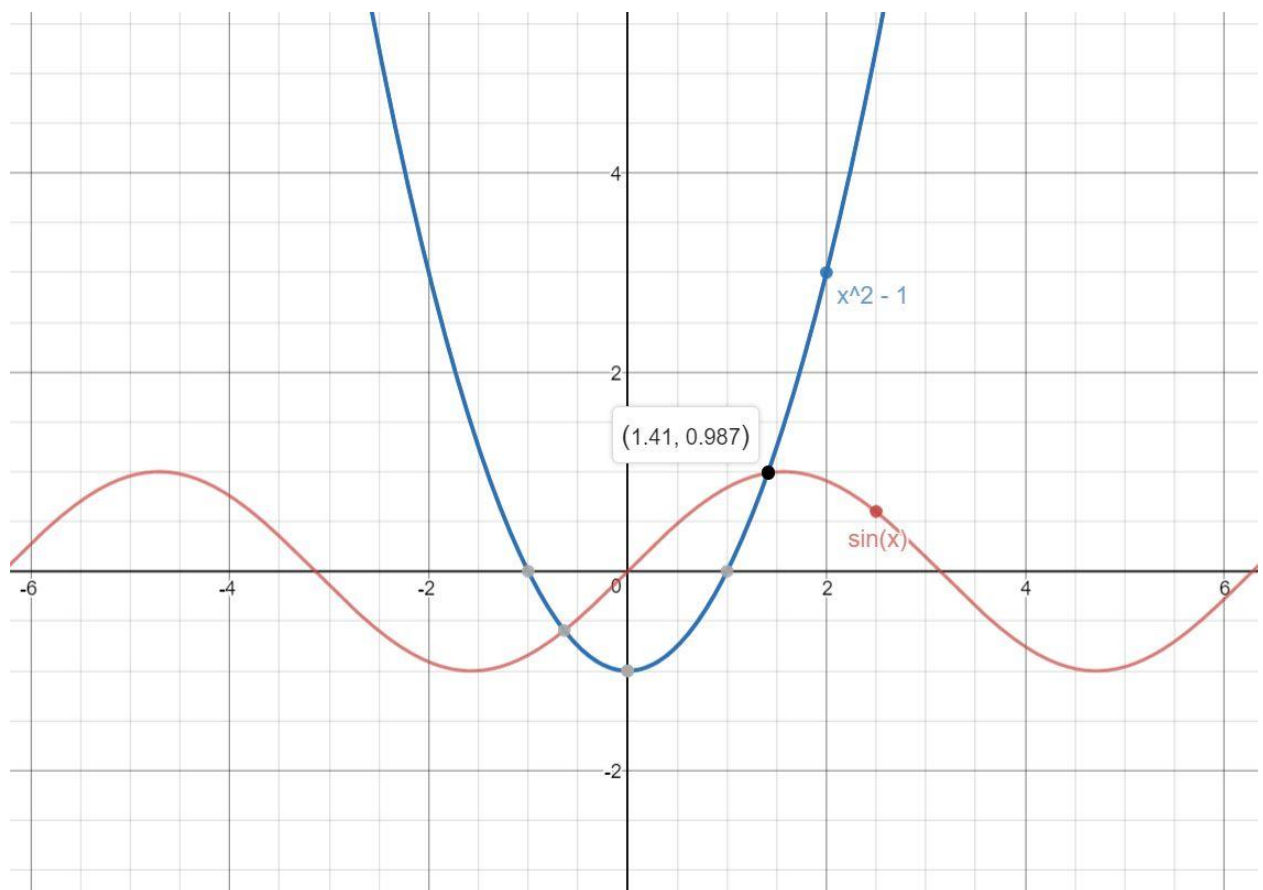
## 2.1

### Задание

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

15.  $\sin x - x^2 + 1 = 0.$

Начальное приближение определим графически:



### Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab2
{
    static class Newton
    {
        private static double a = 0.0;
```

```

private static double b = 2.0;
private static double eps = 0.001;

public static void Start(double e)
{
    Console.WriteLine("*****");

    eps = e;

    var xk = b;

    for (int i = 0; i < 100; i++)
    {
        var fx = Math.Sin(xk) - xk * xk + 1;
        var f1x = Math.Cos(xk) - 2 * xk;
        var dlt = -fx / f1x;

        Console.WriteLine();
        Console.WriteLine($"x{i} = {xk}");
        Console.WriteLine($"f(x){i} = {fx}");
        Console.WriteLine($"f'(x){i} = {f1x}");
        Console.WriteLine($"delta{i} = {dlt}");

        Console.WriteLine();

        if (Math.Abs(dlt) < eps)
            break;

        xk = xk + dlt;
    }

    Console.WriteLine($"Answer:\nx = {xk}");

    Console.WriteLine("*****");
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab2
{
    static class SimpleIteration
    {
        private static double a = -3.0;
        private static double b = 2.0;
        private static double eps = 0.001;

        public static void Start(double e)
        {
            Console.WriteLine("*****");

            eps = e;

            var xk = (a + b) / 2;

```

```

var q1 = Math.Cos(a) / (2 * Math.Sqrt(Math.Sin(a) + 1));
var q2 = Math.Cos(b) / (2 * Math.Sqrt(Math.Sin(b) + 1));

var q = Math.Min(Math.Abs(q1), Math.Abs(q2));

for (int i = 0; i < 100; i++)
{
    var fx = Math.Sqrt(Math.Sin(xk) + 1);

    Console.WriteLine();
    Console.WriteLine($"x{i} = {xk}");
    Console.WriteLine($"f(x){i} = {fx}");
    Console.WriteLine();

    if ((q * Math.Abs(fx - xk)) / (1 - q) < eps) break;

    xk = fx;
}

Console.WriteLine($"Answer:\nx = {xk}");

Console.WriteLine("*****");
}
}

```

## Результат выполнения

1

Введите точность: 0,0001

Newton method:

\*\*\*\*\*

$x_0 = 2$   
 $f(x)_0 = -2,09070257317432$   
 $f'(x)_0 = -4,41614683654714$   
 $\text{delta}_0 = -0,473422340913143$

$x_1 = 1,52657765908686$   
 $f(x)_1 = -0,331416835222043$   
 $f'(x)_1 = -3,00895105911431$   
 $\text{delta}_1 = -0,110143644316899$

$x_2 = 1,41643401476996$   
 $f(x)_2 = -0,0181755419823058$   
 $f'(x)_2 = -2,67911800466749$   
 $\text{delta}_2 = -0,00678415133287926$

$x_3 = 1,40964986343708$   
 $f(x)_3 = -6,87553536056207\text{E}-05$   
 $f'(x)_3 = -2,6588498053038$   
 $\text{delta}_3 = -2,58590588563782\text{E}-05$

Answer:

$x = 1,40964986343708$

\*\*\*\*\*



```

Simple iteration method:
*****

x0 = -0,5
f(x)0 = 0,721508462456122

x1 = 0,721508462456122
f(x)1 = 1,28861087689684

x2 = 1,28861087689684
f(x)2 = 1,40016041305137

x3 = 1,40016041305137
f(x)3 = 1,40906954486193

x4 = 1,40906954486193
f(x)4 = 1,40959238926248

Answer:
x = 1,40906954486193
*****

```

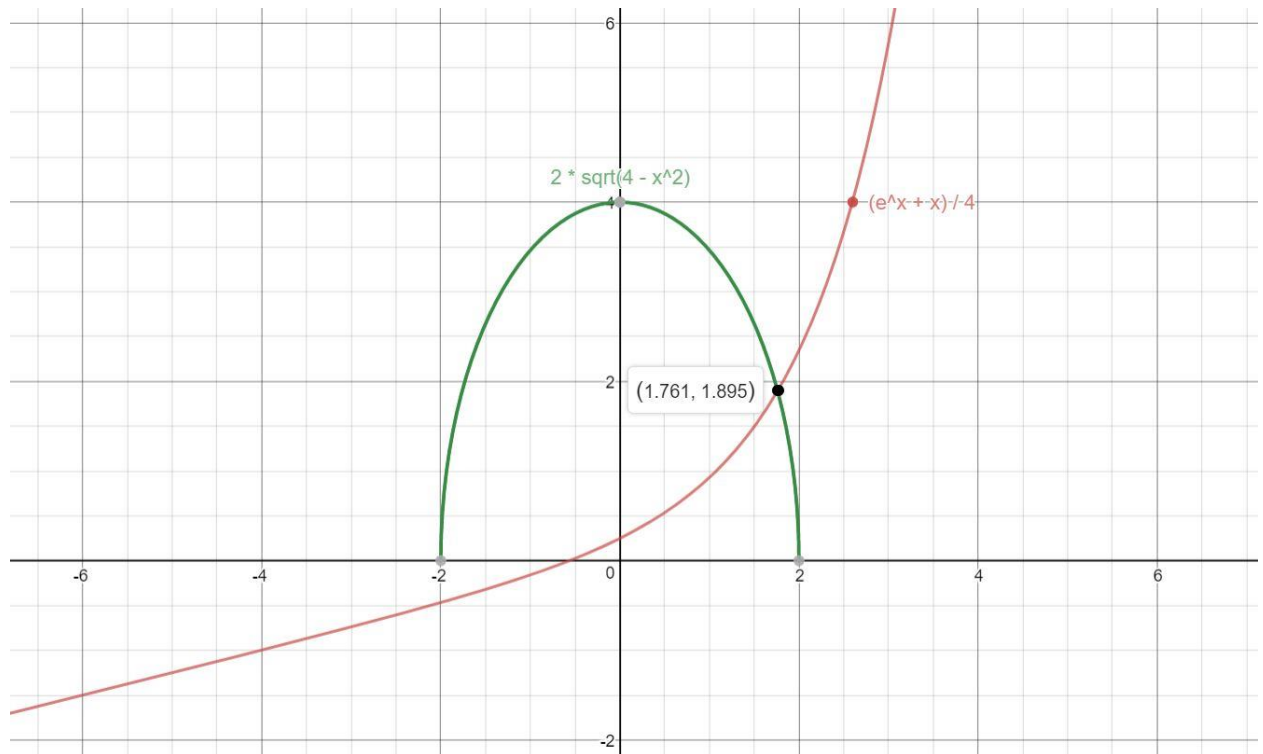
## 2.2

### Задание

2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

13	2	$\begin{cases} x_1^2/a^2 + x_2^2/(a/2)^2 - 1 = 0, \\ ax_2 - e^{x_1} - x_1 = 0. \end{cases}$
14	3	
15	4	

Начальное приближение определим графически



## Код программы

```
using NMLab1;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab2
{
    static class NewtonSys
    {
        public static void Start(double eps)
        {
            Console.WriteLine("*****");

            var x1k = 2.0;
            var x2k = 2.0;

            for (int i = 0; i < 100; i++)
            {
                var A1 = FindDetA1(x1k, x2k);
                var A2 = FindDetA2(x1k, x2k);
                var J = FindDetJ(x1k, x2k);

                var tmp1 = A1 / J;
                var tmp2 = A2 / J;

                Console.WriteLine("*****");
                Console.WriteLine("Iteration #" + i);
                Console.WriteLine();
                Console.WriteLine($"x1 = {x1k}");
                Console.WriteLine($"x2 = {x2k}");
            }
        }
    }
}
```

```

        Console.WriteLine();
        Console.WriteLine($"det A1 = {A1}");
        Console.WriteLine($"det A2 = {A2}");
        Console.WriteLine($"det J = {J}");
        Console.WriteLine($"dlt1 = {tmp1}");
        Console.WriteLine($"dlt2 = {tmp2}");
        Console.WriteLine("*****");

        if (Math.Max(Math.Abs(tmp1), Math.Abs(tmp2)) < eps)
            break;

        x1k -= tmp1;
        x2k -= tmp2;

    }

    Console.WriteLine($"Answer:\n");
    Console.WriteLine($"x1 = {x1k}");
    Console.WriteLine($"x2 = {x2k}");
    Console.WriteLine("*****");
}

private static double FindDetJ(double x1k, double x2k)
{
    Matrix m = new Matrix(2);

    m.mtx[0][0] = F11(x1k, x2k);
    m.mtx[0][1] = F12(x1k, x2k);
    m.mtx[1][0] = F21(x1k, x2k);
    m.mtx[1][1] = F22(x1k, x2k);

    return m.CalcDet2();
}

private static double FindDetA1(double x1k, double x2k)
{
    Matrix m = new Matrix(2);

    m.mtx[0][0] = F1(x1k, x2k);
    m.mtx[0][1] = F12(x1k, x2k);
    m.mtx[1][0] = F2(x1k, x2k);
    m.mtx[1][1] = F22(x1k, x2k);

    return m.CalcDet2();
}

private static double F22(double x1k, double x2k)
{
    return 4;
    //return 1 - 0.1 * x1k;
}

private static double F12(double x1k, double x2k)
{
    return x2k / 2;
    //return 0.4 * x2k;
}

private static double FindDetA2(double x1k, double x2k)
{
    Matrix m = new Matrix(2);

    m.mtx[0][0] = F11(x1k, x2k);
    m.mtx[0][1] = F1(x1k, x2k);

```

```

        m.mtx[1][0] = F21(x1k, x2k);
        m.mtx[1][1] = F2(x1k, x2k);

        return m.CalcDet2();
    }

    private static double F21(double x1k, double x2k)
    {
        return -Math.Pow(Math.E, x1k) - 1;
        //return 0.4 * x1k - 0.1 * x2k;
    }

    private static double F11(double x1k, double x2k)
    {
        return x1k / 8;
        //return 0.2 * x1k + 1;
    }

    private static double F2(double x1k, double x2k)
    {
        return 4.0 * x2k - Math.Pow(Math.E, x1k) - x1k;
        //return 0.2 * 0.2 * x1k + x2k - 0.1 * x1k * x2k - 0.7;
    }

    private static double F1(double x1k, double x2k)
    {
        return x1k * x1k / 16.0 + x2k * x2k / 4.0 - 1.0;
        //return 0.1 * x1k * x1k + x1k + 0.2 * x2k * x2k - 0.3;
    }
}

```

## Результат выполнения

2

Введите точность: 0,001

Newton method for system:

\*\*\*\*\*

\*\*\*\*\*

Iteration #0

x1 = 2

x2 = 2

det A1 = 2,38905609893065

det A2 = 1,75

det J = 9,38905609893065

dlt1 = 0,254451147565595

dlt2 = 0,186387213108601

\*\*\*\*\*

\*\*\*\*\*

Iteration #1

x1 = 1,74554885243441

x2 = 1,8136127868914

det A1 = 0,250553363846451

det A2 = 0,0376380111626785

det J = 6,97471546576725

```

dlt1 = 0,035923094651846
dlt2 = 0,00539635076834466
*****
*****
Iteration #2

x1 = 1,70962575778256
x2 = 1,80821643612305

det A1 = 0,00365418042509474
det A2 = -0,000206656606541401
det J = 6,75583020190419
dlt1 = 0,000540892875617979
dlt2 = -3,05893724923923E-05
*****
Answer:

x1 = 1,70962575778256
x2 = 1,80821643612305
*****

Simple iteration method for system:
*****
*****
Iteration #0

x1 = 1,5
x2 = 2

phi1 = 1,58385316345286
phi2 = 2,99749498660405
*****
*****
Iteration #1

x1 = 1,58385316345286
x2 = 2,99749498660405

phi1 = 1,01036411668589
phi2 = 2,99991476071923
*****
*****
Iteration #2

x1 = 1,01036411668589
x2 = 2,99991476071923

phi1 = 1,01001953596404
phi2 = 2,84702544784007
*****
*****
Iteration #3

x1 = 1,01001953596404
x2 = 2,84702544784007

phi1 = 1,04307211675717

```

```

phi2 = 2,84684223486834
*****
*****
Iteration #4

x1 = 1,04307211675717
x2 = 2,84684223486834

phi1 = 1,04312532425621
phi2 = 2,86395532289233
*****
*****
Iteration #5

x1 = 1,04312532425621
x2 = 2,86395532289233

phi1 = 1,03829430769284
phi2 = 2,86398211528805
*****
*****
Iteration #6

x1 = 1,03829430769284
x2 = 2,86398211528805

phi1 = 1,03828696466504
phi2 = 2,86153951713034
*****
*****
Iteration #7

x1 = 1,03828696466504
x2 = 2,86153951713034

phi1 = 1,0389592476067
phi2 = 2,86153578912154
*****
*****
Iteration #8

x1 = 1,0389592476067
x2 = 2,86153578912154

phi1 = 1,03896027806004
phi2 = 2,86187691033097
*****
*****
Iteration #9

x1 = 1,03896027806004
x2 = 2,86187691033097

phi1 = 1,03886604456177
phi2 = 2,86187743289147
*****
*****

```

Iteration #10

x1 = 1,03886604456177  
x2 = 2,86187743289147

phi1 = 1,03886590029212  
phi2 = 2,86182964168942

\*\*\*\*\*

\*\*\*\*\*

Iteration #11

x1 = 1,03886590029212  
x2 = 2,86182964168942

phi1 = 1,03887909567792  
phi2 = 2,86182956851615

\*\*\*\*\*

\*\*\*\*\*

Iteration #12

x1 = 1,03887909567792  
x2 = 2,86182956851615

phi1 = 1,0388791158831  
phi2 = 2,8618362611142

\*\*\*\*\*

Answer:

x1 = 1,03887909567792  
x2 = 2,86182956851615

\*\*\*\*\*

## Выводы

В ходе лабораторной работы мной были реализованы методы простых итераций и Ньютона для решение нелинейных уравнений, а также систем нелинейных уравнений. Данные навыки пригодятся мне в дальнейших лабораторных и курсовых работах. Проблемы возникли лишь с методом простых итераций для систем уравнений. Никак не получалось, чтобы метод сходиллся.

# Лабораторная работа №3

## 3.1

### Задание

3.1. Используя таблицу значений  $Y_i$  функции  $y = f(x)$ , вычисленных в точках  $X_i$ ,  $i = 0, \dots, 3$  построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки  $\{X_i, Y_i\}$ . Вычислить значение погрешности интерполяции в точке  $X^*$ .

$$15. \text{ } y = \text{ctg}(x) + x, a) X_i = \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}, \frac{4\pi}{8}; б) X_i = \frac{\pi}{8}, \frac{\pi}{3}, \frac{3\pi}{8}, \frac{\pi}{2}; \quad X^* = \frac{3\pi}{16}.$$

### Код программы

#### Lagrange.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab3
{
    public static class Lagrange
    {
        static List<double> X = new List<double>();
        static List<double> Fi = new List<double>();
        static List<double> W = new List<double>();

        static double X_ = 0.0;

        public static void Start(string path)
        {
            Console.WriteLine("*****");
            Parse(path);

            FillFi();
            FillW();

            MakePolynomial();
            Console.WriteLine("*****");
        }

        private static void MakePolynomial()
        {
            string s = "L(x) = ";
            double L = 0;

            for (int i = 0; i < X.Count; ++i)
            {
                var tmp = Fi[i] / W[i];

                if (i == 0)
```



```

        s += "[";
    else
        s += "          [";

    s += tmp.ToString("0.000");

    for (int j = 0; j < X.Count; ++j)
    {
        if (j == i)
            continue;

        s += "(x ";

        if (X[j] > 0)
        {
            s += "- (";
            s += X[j].ToString("0.000") + ")";
        }

        s += ")";

        tmp *= (X_ - X[j]);
    }

    if (i == X.Count - 1)
        s += "]";
    else
        s += "] +\n";

    L += tmp;
}

Console.WriteLine(s + "\n");
Console.WriteLine("L(" + X_ + ") = " + L);
Console.WriteLine("y(" + X_ + ") = " + MyFun(X_));

Console.WriteLine("delta(" + X_ + ") = " + Math.Abs(Math.Abs(L) -
Math.Abs(MyFun(X_))));
}

private static void FillW()
{
    for (int i = 0; i < X.Count; ++i)
    {
        double tmp = 1;

        for (int j = 0; j < X.Count; ++j)
        {
            if (i == j)
                continue;

            tmp *= (X[i] - X[j]);
        }

        W.Add(tmp);
    }
}

private static void PrintList(List<double> fi)
{
    foreach (var el in fi)
    {
        Console.WriteLine(el);
    }
}

```

```

private static void FillFi()
{
    foreach (var x in X)
    {
        Fi.Add(MyFun(x));
    }
}

private static double MyFun(double x)
{
    return 1 / Math.Tan(x) + x;
}

private static void Parse(string path)
{
    var lines = File.ReadAllLines(path);

    var str = lines[0].Split(' ');
    X_ = Convert.ToDouble(lines[1]);

    foreach (var s in str)
        X.Add(Convert.ToDouble(s));
}
}
}

```

## Newton.cs

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab3
{
    public static class Newton
    {
        static List<double> X = new List<double>();
        static List<double> Fi = new List<double>();

        static double X_ = 0.0;

        public static void Start(string path)
        {
            Console.WriteLine("*****");
            Parse(path);

            FillFi();

            MakePolynomial();
            Console.WriteLine("*****");
        }

        private static double DividedDifference(List<double> l)
        {
            if (l.Count < 2)
                return 0.0;
        }
    }
}

```

```

        if (l.Count == 2)
            return (MyFun(l[0]) - MyFun(l[1])) / (l[0] - l[1]);

        var f = new List<double>();
        var s = new List<double>();

        for (int i = 0; i < l.Count - 1; ++i)
            f.Add(l[i]);

        for (int i = 1; i < l.Count; ++i)
            s.Add(l[i]);

        return (DividedDifference(f) - DividedDifference(s)) / (l[0] - l[l.Count -
1]);
    }

    private static void MakePolynomial()
    {
        string s = "P(x) = ";
        double P = 0;

        for (int i = 0; i < X.Count; ++i)
        {
            if (i == 0)
            {
                s += MyFun(X[i]).ToString("0.000") + " + ";

                P += MyFun(X[i]);
                continue;
            }

            var tmp = 1.0;

            var l = MakeList(i);

            if (i == 1)
                l.Reverse();

            var DD = DividedDifference(l);

            s += "(" + DD.ToString("0.000");

            tmp *= DD;

            for (int j = 0; j < i; ++j)
            {
                s += "(x - " + X[j].ToString("0.000") + ")";
                tmp *= (X_ - X[j]);
            }

            s += ")";

            if (i != X.Count - 1)
                s += " + ";

            P += tmp;
        }

        Console.WriteLine(s + "\n");
        Console.WriteLine("P(" + X_ + ") = " + P);
        Console.WriteLine("y(" + X_ + ") = " + MyFun(X_));

        Console.WriteLine("delta(" + X_ + ") = " + Math.Abs(Math.Abs(P) -
Math.Abs(MyFun(X_))));
    }

```

```

private static List<double> MakeList(int i)
{
    var l = new List<double>();

    for (int j = 0; j <= i; ++j)
        l.Add(X[j]);

    return l;
}

private static void PrintList(List<double> fi)
{
    foreach (var el in fi)
    {
        Console.WriteLine(el);
    }
}

private static void FillFi()
{
    foreach (var x in X)
    {
        Fi.Add(MyFun(x));
    }
}

private static double MyFun(double x)
{
    return 1 / Math.Tan(x) + x;
}

private static void Parse(string path)
{
    var lines = File.ReadAllLines(path);

    var str = lines[0].Split(' ');
    X_ = Convert.ToDouble(lines[1]);

    foreach (var s in str)
        X.Add(Convert.ToDouble(s));
}
}
}

```

## Результат выполнения

Choose part:

- 1 - 3.1
- 2 - 3.2
- 3 - 3.3
- 4 - 3.4
- 5 - 3.5

1

Lagrange method:

\*\*\*\*\*

$$\begin{aligned}
 L(x) = & [-7,725(x - (0,785))(x - (1,178))(x - (1,571))] + \\
 & [14,741(x - (0,393))(x - (1,178))(x - (1,571))] + \\
 & [-13,147(x - (0,393))(x - (0,785))(x - (1,571))] + \\
 & [4,323(x - (0,393))(x - (0,785))(x - (1,178))]
 \end{aligned}$$

```

L(0,58904) = 2,15157242907422
y(0,58904) = 2,08567369860569
delta(0,58904) = 0,0658987304685303
*****

```

Newton method:

```
*****
```

```

P(x) = 2,807 + (-1,807(x - 0,393)) + (1,987(x - 0,393)(x - 1,047)) + (-1,376(x - 0,393)(x - 1,047)(x - 1,178))

```

```

P(0,58904) = 2,2006195678141
y(0,58904) = 2,08567369860569
delta(0,58904) = 0,114945869208412
*****

```

## 3.2

### Задание

3.2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при  $x = x_0$  и  $x = x_4$ . Вычислить значение функции в точке  $x = X^*$ .

15.  $X^* = 2.666666667$

$i$	0	1	2	3	4
$x_i$	1.0	1.9	2.8	3.7	4.6
$f_i$	2.8069	1.8279	1.6091	1.5713	1.5663

### Код программы

```
using NMLab1;
using System;
using System.Collections.Generic;
using System.ComponentModel.Design.Serialization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab3
{
    public static class CubicSpline
    {
        static List<double> X = new List<double>();
        static List<double> F = new List<double>();
        static List<double> A = new List<double>();
        static List<double> B = new List<double>();
        static List<double> C = new List<double>();
        static List<double> D = new List<double>();
        static List<double> H = new List<double>();

        static Matrix CMtx;

        static double X_ = 0.0;

        public static void Start(string path)
        {
            Console.WriteLine("*****");
            Parse(path);

            FillH();
            MakeMatrixC();

            var ans = ThomasAlgorythm.TA(CMtx);

            C = ans;

            C.Insert(0, 0.0);
            C.Insert(0, 0.0);

            FillUnknown();

            MakePolynomial();
            Console.WriteLine("*****");
        }
    }
}
```

```

    }

    private static void MakePolynomial()
    {
        var iForX = 0;
        var iRight = 0;

        for (int i = 1; i < X.Count; i++)
        {
            if (X[i - 1] <= X_ && X[i] >= X_)
            {
                iForX = i;
                iRight = i - 1;
                break;
            }
        }

        Console.WriteLine($"X[{iForX}-1] = {X[iForX - 1]} <= X = {X_} <= X[{iForX}]
= {X[iForX]}\n");

        Console.WriteLine($"A[{iForX}] = {A[iRight]}");
        Console.WriteLine($"B[{iForX}] = {B[iRight]}");
        Console.WriteLine($"C[{iForX}] = {C[iRight]}");
        Console.WriteLine($"D[{iForX}] = {D[iRight]}");

        var xTmp = X_ - X[iForX - 1];

        var ans0 = A[iRight];
        var ans1 = B[iRight] * xTmp;
        var ans2 = C[iRight] * Math.Pow(xTmp, 2);
        var ans3 = D[iRight] * Math.Pow(xTmp, 3);

        var ans = ans0 + ans1 + ans2 + ans3;

        Console.WriteLine($"F({X_}) = {ans}");
    }

    private static void FillUnknown()
    {
        var n = X.Count;
        for (int i = 1; i < n; i++)
            A.Add(F[i - 1]);

        for (int i = 1; i < n - 1; i++)
        {
            B.Add((F[i] - F[i-1]) / H[i] - H[i]*(C[i+1] + 2 * C[i]) / 3.0);
            D.Add((C[i + 1] - C[i]) / (3 * H[i]));
        }

        B.Add((F[n - 1] - F[n - 2]) / H[n - 1] - H[n - 1] * C[n - 1] * 2.0 / 3.0);
        D.Add(- C[n - 1] / (3 * H[n - 1]));

        C.RemoveAt(0);
    }

    private static void MakeMatrixC()
    {
        CMtx = new Matrix(X.Count - 2, X.Count - 1);

        // Н здесь номер берется как в методичке, у всех остальных переменных i
        // сдвигается на один меньше

```

```

CMtx.mtx[0][0] = 2 * (H[1] + H[2]);
CMtx.mtx[0][1] = H[2];

var tmp1 = (F[2] - F[1]) / H[2] - (F[1] - F[0]) / H[1];
CMtx.mtx[0][CMtx.columns - 1] = 3 * (tmp1);

//for (int i = 1; i < n; ++i)
//{
//    // однако тут у H на один больше, т.к. i с нуля...
//    я запутался в индексах и спустился до такого...
CMtx.mtx[1][0] = H[2];
CMtx.mtx[1][1] = 2 * (H[2] + H[3]);
CMtx.mtx[1][2] = H[3];

var tmp2 = (F[3] - F[2]) / H[3] - (F[2] - F[1]) / H[2];
CMtx.mtx[1][3] = 3 * (tmp2);
//}

CMtx.mtx[2][1] = H[3];
CMtx.mtx[2][2] = 2 * (H[3] + H[4]);

var tmp3 = (F[4] - F[3]) / H[4] - (F[3] - F[2]) / H[3];
CMtx.mtx[2][3] = 3 * (tmp3);
}

private static void FillH()
{
    H.Add(0);

    for (int i = 1; i < X.Count; ++i)
        H.Add(X[i] - X[i - 1]);
}

private static void Parse(string path)
{
    var lines = File.ReadAllLines(path);

    var str1 = lines[0].Split(' ');
    var str2 = lines[1].Split(' ');

    foreach (var s in str1)
        X.Add(Convert.ToDouble(s));

    foreach (var s in str2)
        F.Add(Convert.ToDouble(s));

    X_ = Convert.ToDouble(lines[2]);
}
}
}

```

## Результат выполнения

2

Cubic spline:

\*\*\*\*\*

Решение системы:

c2 = 0,708452380952381

c3 = -0,0182539682539682

c4 = 0,0349338624338626

X[2-1] = 1,9 <= X = 2,66666667 <= X[2] = 2,8

i = 2



```
A[2] = 1,8279  
B[2] = -0,662706349206349  
C[2] = 0,708452380952381  
D[2] = -0,269150499706055  
  
F(2,66666667) = 1,61495050824548
```

## 3.3

### Задание

3.3. Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

15.

$i$	0	1	2	3	4	5
$x_i$	1.0	1.9	2.8	3.7	4.6	5.5
$y_i$	3.4142	2.9818	3.3095	3.8184	4.3599	4.8318

### Код программы

```
using NMLab1;
using System;
using System.Collections.Generic;
using System.ComponentModel.Design.Serialization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab3
{
    public static class LeastSquares
    {
        static List<double> X = new List<double>();
        static List<double> Y = new List<double>();
        static List<double> A1 = new List<double>();
        static List<double> A2 = new List<double>();

        static List<double> F1 = new List<double>();
        static List<double> F2 = new List<double>();

        public static void Start(string path)
        {
            Console.WriteLine("*****");
            Parse(path);

            ListHelper.Print(X, "X");
            ListHelper.Print(Y, "Y");

            Console.WriteLine("Find First power polynome: ");

            FindFirstPower();
            Console.WriteLine("Find Second power polynome: ");

            FindSecondPower();
            Console.WriteLine("*****");
        }

        private static void FindFirstPower()
        {
            var A = new Matrix(2, 3);
```

```

A.mtx[0][0] = X.Count;
A.mtx[0][1] = FindSum("X", 1);
A.mtx[0][2] = FindSum("Y", 1);
A.mtx[1][0] = FindSum("X", 1);
A.mtx[1][1] = FindSum("X", 2);
A.mtx[1][2] = FindSum("XY", 1);

A.Print("AAA");

var aSys = new LESystem(A);

var sol = new LUDecomposition(aSys);

sol.LU_Algorithm();
A1 = aSys.Solution;

ListHelper.Print(A1, "A1");
FillF1();
ListHelper.Print(F1, "F1");
//foreach (var el in F1)
//{
//    Console.WriteLine(el);
//}

var bigF = 0.0;

for (int i = 0; i < X.Count; i++)
    bigF += Math.Pow(F1[i] - Y[i], 2);

Console.WriteLine("Sum of quadro error: " + bigF + "\n");
}

private static void FillF1()
{
    for (int i = 0; i < X.Count; i++)
    {
        F1.Add(A1[0] + A1[1] * X[i]);
    }
}

private static double FindSum(string s, int i)
{
    double sum = 0.0;

    if (s == "X")
    {
        for (int j = 0; j < X.Count; j++)
            sum += Math.Pow(X[j], i);
    }

    if (s == "Y")
    {
        for (int j = 0; j < X.Count; j++)
            sum += Math.Pow(Y[j], i);
    }

    if (s == "XY")
    {
        for (int j = 0; j < X.Count; j++)
        {

```

```

        sum += Y[j] * Math.Pow(X[j], i);
    }
}

return sum;
}

private static void FindSecondPower()
{
    var A = new Matrix(3, 4);

    A.mtx[0][0] = X.Count;
    A.mtx[0][1] = FindSum("X", 1);
    A.mtx[0][2] = FindSum("X", 2);
    A.mtx[0][3] = FindSum("Y", 1);
    A.mtx[1][0] = FindSum("X", 1);
    A.mtx[1][1] = FindSum("X", 2);
    A.mtx[1][2] = FindSum("X", 3);
    A.mtx[1][3] = FindSum("XY", 1);
    A.mtx[2][0] = FindSum("X", 2);
    A.mtx[2][1] = FindSum("X", 3);
    A.mtx[2][2] = FindSum("X", 4);
    A.mtx[2][3] = FindSum("XY", 2);

    A.Print("AAAA");

    var aSys = new LSystem(A);

    var sol = new LUdecomposition(aSys);

    sol.LU_Algorithm();
    A2 = aSys.Solution;

    ListHelper.Print(A2, "A2");
    FillF2();
    ListHelper.Print(F2, "F2");
    //foreach (var el in F1)
    //{
    //    Console.WriteLine(el);
    //}

    var bigF = 0.0;

    for (int i = 0; i < X.Count; i++)
        bigF += Math.Pow(F2[i] - Y[i], 2);

    Console.WriteLine("Sum of quadro error: " + bigF + "\n");
}

private static void FillF2()
{
    for (int i = 0; i < X.Count; i++)
        F2.Add(A2[0] + A2[1] * X[i] + A2[2] * X[i] * X[i]);
}

private static void Parse(string path)
{
    var lines = File.ReadAllLines(path);

    var str = lines[0].Split(' ');
    var str1 = lines[1].Split(' ');

```

```

        foreach (var s in str)
            X.Add(Convert.ToDouble(s));

        foreach (var s in str1)
            Y.Add(Convert.ToDouble(s));
    }
}

```

## Результат выполнения

3

Least Squares method:

\*\*\*\*\*

X[0] = 1  
 X[1] = 1,9  
 X[2] = 2,8  
 X[3] = 3,7  
 X[4] = 4,6  
 X[5] = 5,5

Y[0] = 3,4142  
 Y[1] = 2,9818  
 Y[2] = 3,3095  
 Y[3] = 3,8184  
 Y[4] = 4,3599  
 Y[5] = 4,8318

Find First power polynome:

Matrix AAA:

\*\*\*\*\*

6,0000 19,5000 22,7156  
 19,5000 77,5500 79,1047

\*\*\*\*\*

A1[0] = 2,57557142857143  
 A1[1] = 0,372419047619046

F1[0] = 2,94799047619048  
 F1[1] = 3,28316761904762  
 F1[2] = 3,61834476190476  
 F1[3] = 3,9535219047619  
 F1[4] = 4,28869904761905  
 F1[5] = 4,62387619047619

Sum of quadro error: 0,470118664190477

Find Second power polynome:

Matrix AAAA:

\*\*\*\*\*

6,0000 19,5000 77,5500 22,7156  
 19,5000 77,5500 344,1750 79,1047  
 77,5500 344,1750 1625,7219 330,8163

\*\*\*\*\*

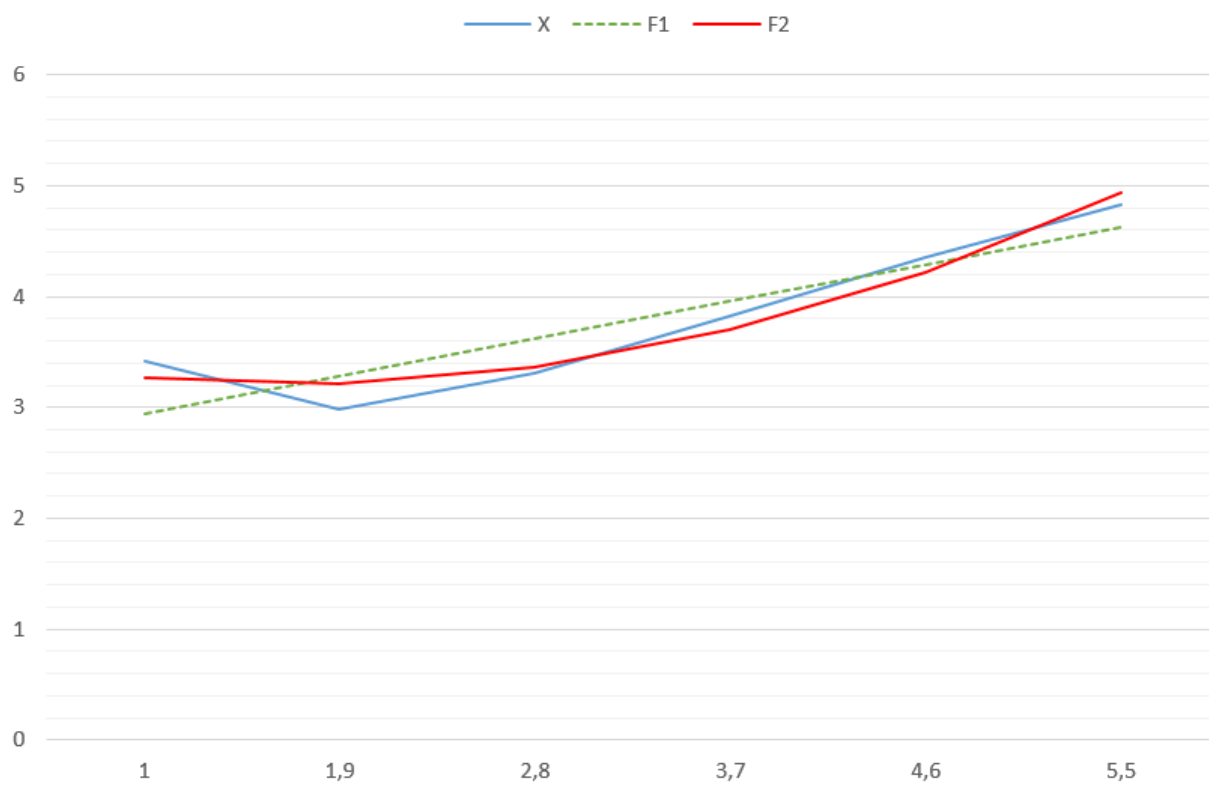
```
A2[0] = 3,5475498236332  
A2[1] = -0,398051631393333  
A2[2] = 0,118533950617289
```

```
F2[0] = 3,26803214285716  
F2[1] = 3,21915928571429  
F2[2] = 3,36231142857142  
F2[3] = 3,69748857142856  
F2[4] = 4,22469071428571  
F2[5] = 4,94391785714287
```

Sum of quadro error: 0,125965058357143

\*\*\*\*\*

## График



## 3.4

### Задание

3.4. Вычислить первую и вторую производную от таблично заданной функции

$y_i = f(x_i)$ ,  $i = 0, 1, 2, 3, 4$  в точке  $x = X^*$ .

15.  $X^* = 0.4$

i	0	1	2	3	4
$x_i$	0.0	0.2	0.4	0.6	0.8
$y_i$	1.0	1.4214	1.8918	2.4221	3.0255

### Код программы

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab3
{
    public static class NumericDiff
    {
        static List<double> X = new List<double>();
        static List<double> Y = new List<double>();
        static double X_ = 0.0;

        public static void Start(string path)
        {
            Console.WriteLine("*****");
            Parse(path);

            int i = FindNum();

            double left = CalcLeftSide(i);
            double right = CalcRightSide(i);
            double halfSum = (left + right) / 2;
            double secondAccuracy = CalcSecondAccuracy(i, left, right);
            double secondDerivative = CalcSecondDerivative(i, left, right);

            Console.WriteLine("Left side derivative:    " + left);
            Console.WriteLine("Right side derivative:    " + right);
            Console.WriteLine("Half-sum of left and right:    " + halfSum);
            Console.WriteLine("Second order of accuracy derivative:    " +
secondAccuracy);
            Console.WriteLine("Second derivative:    " + secondDerivative);

            Console.WriteLine("*****");

            Console.WriteLine("*****");
            Console.WriteLine("By five dots:");
            var KF = CalcKoeff();
            ByFiveDots(KF);
        }

        private static void ByFiveDots(List<double> kF)
```

```

{
    var fd = FirstDirByFive(kF);

    Console.WriteLine("First");
    Console.WriteLine(fd);

    var sd = SecondDirByFive(kF);

    Console.WriteLine("Second");
    Console.WriteLine(sd/2.45);
}

private static double FirstDirByFive(List<double> kF)
{
    var sum1 = kF[1] * ((X_ - X[0]) + (X_ - X[1]));
    var sum2 = kF[2] * ((X_ - X[0]) * (X_ - X[1]) + (X_ - X[0]) * (X_ - X[2]) +
(X_ - X[1]) * (X_ - X[2]));
    var sum3 = kF[2] * kF[3] * ((X_ - X[0]) * (X_ - X[1]) * (X_ - X[2]) +
(X_ - X[0]) * (X_ - X[1]) * (X_ - X[3]) +
(X_ - X[0]) * (X_ - X[2]) * (X_ - X[3]) +
(X_ - X[1]) * (X_ - X[2]) * (X_ - X[3]));

    return kF[0] + sum1 + sum2 + sum3;
}

private static double SecondDirByFive(List<double> kF)
{
    var sum1 = -kF[2] * 2 * (-3 * X_ + X[0] + X[1] + X[2]);
    var sum2 = kF[3] * 2 * (X[2] * (X[3] - 3 * X_ + X[0] + X[1]) +
X[3] * (-3 * X_ + X[0] + X[1]) + 6 * X_ * X_ - 3 * X_
* X[0] -
3 * X_ * X[1] + X[0] * X[1]);

    return 2 * kF[1] + sum1 + sum2;
}

private static double DividedDifference(List<List<double>> l, int j)
{
    if (l[0].Count < 2)
        return 0.0;

    if (l[0].Count == 2)
        return (l[1][0] - l[1][1]) / (l[0][0] - l[0][1]);

    var f = new List<List<double>>();

    f.Add(new List<double>());
    f.Add(new List<double>());

    var s = new List<List<double>>();

    s.Add(new List<double>());
    s.Add(new List<double>());

    for (int i = 0; i < l[0].Count - 1; ++i)
    {
        f[0].Add(l[0][i]);
        f[1].Add(l[1][i]);
    }

    for (int i = 1; i < l[0].Count; ++i)
    {
        s[0].Add(l[0][i]);
        s[1].Add(l[1][i]);
    }
}

```



```

    }

    return (DividedDifference(f, j) - DividedDifference(s, j + 1)) / (l[0][0] -
l[0][l.Count - 1]);
}

private static List<double> CalcKoeff()
{
    List<double> KF = new List<double>();

    for (int i = 0; i < X.Count; ++i)
    {
        if (i == 0)
            continue;

        var l = MakeLists(i);

        if (i == 1)
        {
            l[0].Reverse();
            l[1].Reverse();
        }

        var DD = DividedDifference(l, i);
        KF.Add(DD);
    }
    return KF;
}

private static List<List<double>> MakeLists(int i)
{
    var l = new List<List<double>>();

    l.Add(new List<double>());
    l.Add(new List<double>());

    for (int j = 0; j <= i; ++j)
    {
        l[0].Add(X[j]);
        l[1].Add(Y[j]);
    }

    return l;
}

private static double CalcSecondDerivative(int i, double left, double right)
{
    return 2 * ((right - left) / (X[3] - X[1]));
}

private static double CalcSecondAccuracy(int i, double left, double right)
{
    return left + (right - left) / (X[3] - X[1]) * (2 * X_ - X[1] - X[2]);
}

private static double CalcRightSide(int i)
{
    return (Y[i + 1] - Y[i]) / (X[i + 1] - X[i]);
}

private static double CalcLeftSide(int i)
{
    return (Y[i] - Y[i - 1]) / (X[i] - X[i - 1]);
}

```

```

    }

    private static int FindNum()
    {
        int i = 0;

        for (int j = 0; j < X.Count; j++)
        {
            if (X_ == X[j])
                i = j;
        }

        return i;
    }

    private static void Parse(string path)
    {
        var lines = File.ReadAllLines(path);

        var str = lines[0].Split(' ');
        var str1 = lines[1].Split(' ');

        X_ = Convert.ToDouble(lines[2]);

        foreach (var s in str)
            X.Add(Convert.ToDouble(s));

        foreach (var s in str1)
            Y.Add(Convert.ToDouble(s));
    }
}

```

## Результат выполнения

4

```

Numeric diff:
*****
Left side derivative:    2,352
Right side derivative:   2,6515
Half-sum of left and right:    2,50175
Second order of accuracy derivative:    2,50175
Second derivative:    1,49750000000001
*****
*****
By five dots:
First
2,91966250000001
Second
1,62040816326533
*****

```

## 3.5

### Задание

3.5. Вычислить определенный интеграл  $F = \int_{x_0}^{x_1} y dx$ , методами прямоугольников, трапеций, Симпсона с шагами  $h_1, h_2$ . Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

$$15. \quad y = \frac{x}{x^4 + 81}, \quad X_0 = 0, \quad X_k = 2, \quad h_1 = 0.5, \quad h_2 = 0.25;$$

### Код программы

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab3
{
    public static class NumericIntegr
    {
        static double X0 = 0.0;
        static double Xk = 0.0;
        static double H1 = 0.0;
        static double H2 = 0.0;

        public static void Start(string path)
        {
            Console.WriteLine("*****");
            Parse(path);

            double rect1 = RectangleMethod(H1);
            double trap1 = TrapezMethod(H1);
            double simps1 = SimpsonMethod(H1);

            Console.WriteLine("Step = " + H1);
            Console.WriteLine("Rectangle = " + rect1);
            Console.WriteLine("Trapez = " + trap1);
            Console.WriteLine("Simpson = " + simps1);
            Console.WriteLine();

            double rect2 = RectangleMethod(H2);
            double trap2 = TrapezMethod(H2);
            double simps2 = SimpsonMethod(H2);

            Console.WriteLine("Step = " + H2);
            Console.WriteLine("Rectangle = " + rect2);
            Console.WriteLine("Trapez = " + trap2);
            Console.WriteLine("Simpson = " + simps2);
            Console.WriteLine();

            Console.WriteLine("Runge-Romberg-Richardson method adjustment:");

            Console.WriteLine("Rectangle");
            RRR(rect1, rect2);
        }
    }
}
```

```

        Console.WriteLine("Trapez");
        RRR(trap1, trap2);
        Console.WriteLine("Simpson");
        RRR(simps1, simps2);

        Console.WriteLine("*****");
    }

    private static void RRR(double x1, double x2)
    {
        var k = 0.0;
        var p = 2;
        var rrr = 0.0;

        if (H1 >= H2)
        {
            k = H1 / H2;
            rrr = x2 + (x2 - x1) / (Math.Pow(k, p) - 1);
        }
        else
        {
            k = H2 / H1;
            rrr = x1 + (x1 - x2) / (Math.Pow(k, p) - 1);
        }

        Console.WriteLine(rrr);
    }

    private static double SimpsonMethod(double h)
    {
        Console.WriteLine("Simpson method with step " + h + "\n");

        var x = SplitX(h);
        var y = FindY(x);

        ListHelper.Print(x, "X");
        ListHelper.Print(y, "Y");

        var steps = (Xk - X0) / h + 1;
        double ans = 0;

        ans += h / 3 * y[0];

        var simps = new List<double>{ans};

        for (int i = 1; i < steps - 1; ++i)
        {
            if (i % 2 == 0)
            {
                ans += h / 3 * y[i] * 2;
                simps.Add(ans);
            }
            else
            {
                ans += h / 3 * y[i] * 4;
                simps.Add(ans);
            }
        }

        ans += h / 3 * y[y.Count - 1];
        simps.Add(ans);

        ListHelper.Print(simps, "Simpson");
    }

```

```

        return ans;
    }

    private static double TrapezMethod(double h)
    {
        Console.WriteLine("Trapez method with step " + h + "\n");

        var x = SplitX(h);
        var y = FindY(x);

        ListHelper.Print(x, "X");
        ListHelper.Print(y, "Y");

        var steps = (Xk - X0) / h + 1;
        double ans = 0;

        var trap = new List<double>();

        trap.Add(ans);

        for (int i = 1; i < steps; i++)
        {
            ans += h / 2 * (y[i - 1] + y[i]);
            trap.Add(ans);
        }

        ListHelper.Print(trap, "Trapez");
        return ans;
    }

    private static double RectangleMethod(double h)
    {
        Console.WriteLine("Recatangle method with step " + h + "\n");

        var x = SplitX(h);
        var y = FindY(x);

        ListHelper.Print(x, "X");
        ListHelper.Print(y, "Y");

        var steps = (Xk - X0) / h + 1;
        double ans = 0;

        var rect = new List<double>();

        rect.Add(ans);

        for (int i = 1; i < steps; i++)
        {
            ans += MyFunc((x[i - 1] + x[i]) / 2) * h;
            rect.Add(ans);
        }

        ListHelper.Print(rect, "Rectangle");
        return ans;
    }

    private static List<double> FindY(List<double> x)
    {
        var Y = new List<double>();

        foreach (var el in x)
            Y.Add(MyFunc(el));
    }

```

```

        return Y;
    }

    private static List<double> SplitX(double h)
    {
        var steps = (Xk - X0) / h + 1;
        var X = new List<double>();

        for (int i = 0; i < steps; i++)
            X.Add(X0 + h * i);

        return X;
    }

    private static double MyFunc(double x)
    {
        return x / (Math.Pow(x, 4) + 81);
    }

    private static void CalcInterg()
    {
    }

    private static void Parse(string path)
    {
        var lines = File.ReadAllLines(path);

        X0 = Convert.ToDouble(lines[0]);
        Xk = Convert.ToDouble(lines[1]);
        H1 = Convert.ToDouble(lines[2]);
        H2 = Convert.ToDouble(lines[3]);
    }
}

```

## Результат выполнения

5

Numeric integration:

\*\*\*\*\*

Recatangle method with step 0,5

X[0] = 0

X[1] = 0,5

X[2] = 1

X[3] = 1,5

X[4] = 2

Y[0] = 0

Y[1] = 0,00616808018504241

Y[2] = 0,0121951219512195

Y[3] = 0,0174291938997821

Y[4] = 0,0206185567010309

Rectangle[0] = 0

Rectangle[1] = 0,00154313545835945

Rectangle[2] = 0,0061547509649166

Rectangle[3] = 0,0136450370002146

Rectangle[4] = 0,0233264995925991

Trapez method with step 0,5

```
X[0] = 0
X[1] = 0,5
X[2] = 1
X[3] = 1,5
X[4] = 2
```

```
Y[0] = 0
Y[1] = 0,00616808018504241
Y[2] = 0,0121951219512195
Y[3] = 0,0174291938997821
Y[4] = 0,0206185567010309
```

```
Trapez[0] = 0
Trapez[1] = 0,0015420200462606
Trapez[2] = 0,00613282058032608
Trapez[3] = 0,0135388995430765
Trapez[4] = 0,0230508371932798
```

Simpson method with step 0,5

```
X[0] = 0
X[1] = 0,5
X[2] = 1
X[3] = 1,5
X[4] = 2
```

```
Y[0] = 0
Y[1] = 0,00616808018504241
Y[2] = 0,0121951219512195
Y[3] = 0,0174291938997821
Y[4] = 0,0206185567010309
```

```
Simpson[0] = 0
Simpson[1] = 0,00411205345669494
Simpson[2] = 0,00817709410710144
Simpson[3] = 0,0197965567069562
Simpson[4] = 0,0232329828237947
```

Step = 0,5

```
Rectangle = 0,0233264995925991
Trapez = 0,0230508371932798
Simpson = 0,0232329828237947
```

Recatangle method with step 0,25

```
X[0] = 0
X[1] = 0,25
X[2] = 0,5
X[3] = 0,75
X[4] = 1
X[5] = 1,25
X[6] = 1,5
X[7] = 1,75
X[8] = 2
```

```

Y[0] = 0
Y[1] = 0,00308627091671891
Y[2] = 0,00616808018504241
Y[3] = 0,00922323101311428
Y[4] = 0,0121951219512195
Y[5] = 0,0149805720705959
Y[6] = 0,0174291938997821
Y[7] = 0,019362925184769
Y[8] = 0,0206185567010309

```

```

Rectangle[0] = 0
Rectangle[1] = 0,000385801306299111
Rectangle[2] = 0,00154292621250871
Rectangle[3] = 0,00346831151520034
Rectangle[4] = 0,00614952536295168
Rectangle[5] = 0,00955441457104893
Rectangle[6] = 0,0136188802294895
Rectangle[7] = 0,0182367795848152
Rectangle[8] = 0,0232576869569193

```

Trapez method with step 0,25

```

X[0] = 0
X[1] = 0,25
X[2] = 0,5
X[3] = 0,75
X[4] = 1
X[5] = 1,25
X[6] = 1,5
X[7] = 1,75
X[8] = 2

```

```

Y[0] = 0
Y[1] = 0,00308627091671891
Y[2] = 0,00616808018504241
Y[3] = 0,00922323101311428
Y[4] = 0,0121951219512195
Y[5] = 0,0149805720705959
Y[6] = 0,0174291938997821
Y[7] = 0,019362925184769
Y[8] = 0,0206185567010309

```

```

Trapez[0] = 0
Trapez[1] = 0,000385783864589864
Trapez[2] = 0,00154257775231003
Trapez[3] = 0,00346649165207961
Trapez[4] = 0,00614378577262134
Trapez[5] = 0,00954074752534827
Trapez[6] = 0,0135919682716455
Trapez[7] = 0,0181909831572144
Trapez[8] = 0,0231886683929394

```

Simpson method with step 0,25

```

X[0] = 0
X[1] = 0,25

```



```

X[2] = 0,5
X[3] = 0,75
X[4] = 1
X[5] = 1,25
X[6] = 1,5
X[7] = 1,75
X[8] = 2

Y[0] = 0
Y[1] = 0,00308627091671891
Y[2] = 0,00616808018504241
Y[3] = 0,00922323101311428
Y[4] = 0,0121951219512195
Y[5] = 0,0149805720705959
Y[6] = 0,0174291938997821
Y[7] = 0,019362925184769
Y[8] = 0,0206185567010309

Simpson[0] = 0
Simpson[1] = 0,00102875697223964
Simpson[2] = 0,00205677033641337
Simpson[3] = 0,00513118067411813
Simpson[4] = 0,00716370099932138
Simpson[5] = 0,0121572250228534
Simpson[6] = 0,0150620906728171
Simpson[7] = 0,02151639906774
Simpson[8] = 0,0232346121261593

Step = 0,25
Rectangle = 0,0232576869569193
Trapez = 0,0231886683929394
Simpson = 0,0232346121261593

Runge-Romberg-Richardson method adjustment:
Rectangle
0,0232347494116928
Trapez
0,0232346121261593
Simpson
0,0232351552269475
*****

```

## Выводы

В данной лабораторной работе мною были реализованы:

1. Построение интерполяционных многочленов Лагранжа и Ньютона. Реализация многочлена Ньютона помогла мне в 3.4 при нахождении производной по 5-ти точкам.
2. Построение кубического сплайна для функции заданной в узлах интерполяции.
3. Методом наименьших квадратов были найдены приближающие многочлены первой и второй степени. Построены графики, демонстрирующие сравнение полученных многочленов и приближающей функции.

4. Вычисление первой и второй производных от таблично заданных функций по трем и по пяти точкам.
5. Вычисление определенного интеграла методами прямоугольников, трапеций и Симпсона с различными шагами. Оценена погрешности вычислений с помощью метода Рунге-Ромберга.

Данный раздел понравился мне больше всего, было интересно программировать сложные математические операции и глубже погрузиться в их сущность.

## 4.1

### Задание

4.1. Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки  $h$ . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

15	$xy'' + y' = 0,$ $y(1) = 1,$ $y'(1) = 1,$ $x \in [1, 2], h = 0.1$	$y = 1 + \ln x $
----	--	------------------

### Код программы

```
using NMLab3;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab4
{
    public static class ERKA
    {
        static double h = 0;
        static double y0t1 = 1;
        static double y10t1 = 1;
        static double a = 1;
        static double b = 2;

        public static void Start()
        {
            Console.WriteLine("\nEnter step:");

            h = Convert.ToDouble(Console.ReadLine().Replace('.', ','));

            Console.WriteLine("\nEuler method:");
            Euler();
            RungeKuttaAndAdams();
        }

        private static void RungeKuttaAndAdams()
        {
            var steps = Convert.ToInt16((b - a) / h);

            var xk = a;
            var yk = y0t1;
            var zk = y10t1;

            #region
            Console.WriteLine("\nRunge-Kutta method:");

            var X = new List<double> { xk };
```

```

var Xhalf = new List<double> { xk };
var Y = new List<double> { yk };
var Z = new List<double> { zk };
var YIst = new List<double> { OriginalFunc(xk) };
var Eps = new List<double>();

for (int i = 0; i < steps; ++i)
{
    var k1 = h * zk;
    var l1 = h * MyFunc(xk, zk);
    var k2 = h * (zk + 0.5 * l1);
    var l2 = h * MyFunc(xk + 0.5 * h, zk + 0.5 * l1);
    var k3 = h * (zk + 0.5 * l2);
    var l3 = h * MyFunc(xk + 0.5 * h, zk + 0.5 * l2);
    var k4 = h * (zk + l3);
    var l4 = h * MyFunc(xk + h, zk + l3);

    xk += h;
    yk += (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    zk += (l1 + 2 * l2 + 2 * l3 + l4) / 6;

    X.Add(xk);
    Y.Add(yk);
    Z.Add(zk);
    YIst.Add(OriginalFunc(xk));
}

var runge = h / 2;
var rungeSteps = Convert.ToInt16((b - a) / runge);

for (int i = 0; i <= rungeSteps; ++i)
{
    var k1 = h * zk;
    var l1 = h * MyFunc(xk, zk);
    var k2 = h * (zk + 0.5 * l1);
    var l2 = h * MyFunc(xk + 0.5 * h, zk + 0.5 * l1);
    var k3 = h * (zk + 0.5 * l2);
    var l3 = h * MyFunc(xk + 0.5 * h, zk + 0.5 * l2);
    var k4 = h * (zk + l3);
    var l4 = h * MyFunc(xk + h, zk + l3);

    xk += h;
    yk += (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    zk += (l1 + 2 * l2 + 2 * l3 + l4) / 6;

    if (i % 2 == 0)
        Xhalf.Add(yk);
}

var Acc = new List<double>();

for (int i = 0; i < X.Count; ++i)
{
    Acc.Add(YIst[i] - Y[i]);
    Eps.Add((X[i] - Xhalf[i]) / (2 * 2 - 1));
}

var ll = new List<List<double>>();
var ss = new List<string>();

ll.Add(X);
ss.Add("X");
ll.Add(Y);
ss.Add("Y");
ll.Add(Z);

```

```

        ss.Add("Z");
        ll.Add(YIst);
        ss.Add("YIst");
        ll.Add(Acc);
        ss.Add("YIst-Y");
        ll.Add(Eps);
        ss.Add("Runge Eps");

        ListHelper.PrintTable(ll, ss);
        #endregion

        Console.WriteLine("\nAdams method:");

        xk = X[2];
        yk = Y[2];
        zk = Z[2];

        for (int i = 3; i < steps; ++i)
        {
            zk += h / 24 * (55 * MyFunc(X[i], Z[i]) -
                           59 * MyFunc(X[i - 1], Z[i - 1]) +
                           37 * MyFunc(X[i - 2], Z[i - 2]) -
                           9 * MyFunc(X[i - 3], Z[i - 3]));

            yk += h / 24 * (55 * Z[i] - 59 * Z[i - 1] +
                           37 * Z[i - 2] - 9 * Z[i - 3]);

            xk += h;

            X[i] = xk;
            Y[i] = yk;
            Z[i] = zk;
        }

        Acc = new List<double>();

        for (int i = 0; i < X.Count; ++i)
            Acc.Add(YIst[i] - Y[i]);

        ll = new List<List<double>>();
        ss = new List<string>();

        ll.Add(X);
        ss.Add("X");
        ll.Add(Y);
        ss.Add("Y");
        ll.Add(Z);
        ss.Add("Z");
        ll.Add(YIst);
        ss.Add("YIst");
        ll.Add(Acc);
        ss.Add("YIst-Y");

        ListHelper.PrintTable(ll, ss);
    }

    private static void Euler()
    {
        var steps = Convert.ToInt16((b - a) / h);

        var xk = a;
        var yk = y0t1;
        var y1k = y10t1;

```

```

var X = new List<double> { xk };
var Xhalf = new List<double> { xk };
var Y = new List<double> { yk };
var YIst = new List<double> { OriginalFunc(xk) };
var Eps = new List<double>();

for (int i = 0; i < steps; ++i)
{
    y1k += h * MyFunc(xk, y1k);
    yk += h * y1k;
    xk += h;

    X.Add(xk);
    Y.Add(yk);
    YIst.Add(OriginalFunc(xk));
}

var runge = h / 2;
var rungeSteps = Convert.ToInt16((b - a) / runge);

for (int i = 0; i <= rungeSteps; ++i)
{
    y1k += runge * MyFunc(xk, y1k);
    yk += runge * y1k;
    xk += runge;
    if (i % 2 == 0)
        Xhalf.Add(yk);
}

var Acc = new List<double>();

for (int i = 0; i < X.Count; ++i)
{
    Acc.Add(YIst[i] - Y[i]);
    Eps.Add((X[i] - Xhalf[i]) / (2 * 2 - 1));
}

var ll = new List<List<double>>>();
var ss = new List<string>();

ll.Add(X);
ss.Add("X");
ll.Add(Y);
ss.Add("Y");
ll.Add(YIst);
ss.Add("YIst");
ll.Add(Acc);
ss.Add("YIst-Y");
ll.Add(Eps);
ss.Add("Runge Eps");

ListHelper.PrintTable(ll, ss);

//ListHelper.Print(X, "X");
//ListHelper.Print(Y, "Y");
//ListHelper.Print(YIst, "YIst");

}

private static double OriginalFunc(double x)
{
    return 1.0 + Math.Log(Math.Abs(x));
}

```

```

        private static double MyFunc(double x, double y1)
        {
            return - y1 / x;
        }
    }
}

```

## Результат выполнения

Choose part:

1 - 4.1

2 - 2.2

1

Enter step:

0.1

Euler method:

i	X	Y	YIst	YIst-Y	Runge Eps
0	1,00000	1,00000	1,00000	0,00000	0,00000
1	1,10000	1,09000	1,09531	0,00531	-0,19000
2	1,20000	1,17182	1,18232	0,01050	-0,17150
3	1,30000	1,24682	1,26236	0,01555	-0,15233
4	1,40000	1,31605	1,33647	0,02042	-0,13253
5	1,50000	1,38033	1,40547	0,02513	-0,11216
6	1,60000	1,44033	1,47000	0,02967	-0,09127
7	1,70000	1,49658	1,53063	0,03404	-0,06989
8	1,80000	1,54953	1,58779	0,03826	-0,04807
9	1,90000	1,59953	1,64185	0,04233	-0,02584
10	2,00000	1,64689	1,69315	0,04625	-0,00321

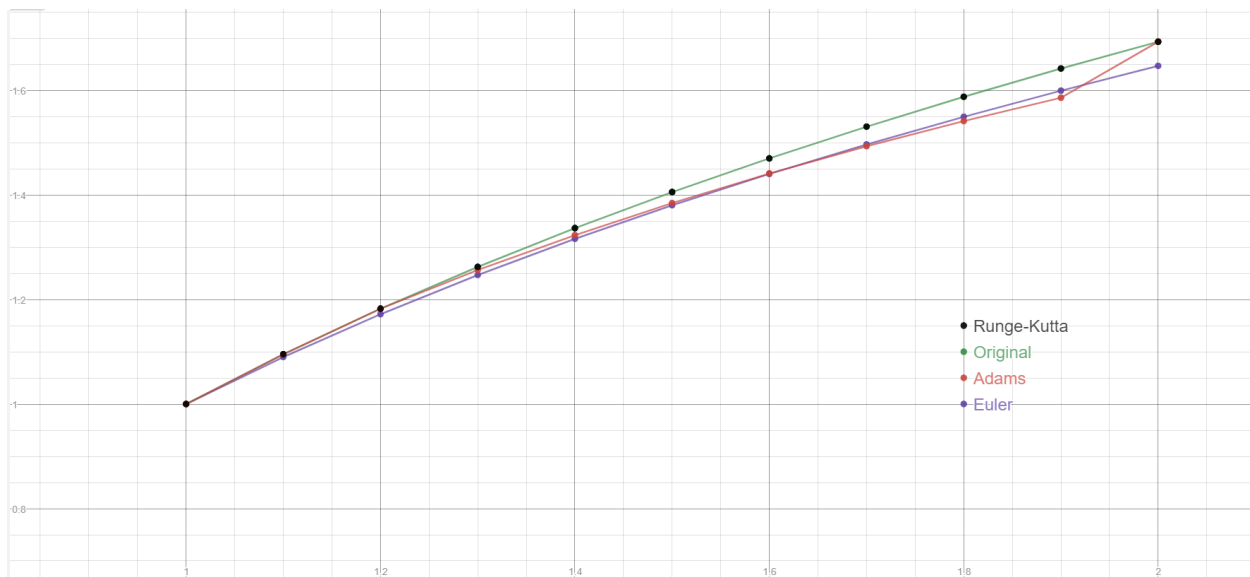
Runge-Kutta method:

i	X	Y	Z	YIst	YIst-Y	Runge Eps
0	1,00000	1,00000	1,00000	1,00000	0,00000	0,00000
1	1,10000	1,09531	0,90909	1,09531	0,00000	-0,21398
2	1,20000	1,18232	0,83333	1,18232	0,00000	-0,21097
3	1,30000	1,26236	0,76923	1,26236	0,00000	-0,20543
4	1,40000	1,33647	0,71429	1,33647	0,00000	-0,19775
5	1,50000	1,40546	0,66667	1,40547	0,00000	-0,18824
6	1,60000	1,47000	0,62500	1,47000	0,00000	-0,17713
7	1,70000	1,53063	0,58824	1,53063	0,00000	-0,16464
8	1,80000	1,58779	0,55556	1,58779	0,00000	-0,15092
9	1,90000	1,64185	0,52632	1,64185	0,00000	-0,13611
10	2,00000	1,69315	0,50000	1,69315	0,00000	-0,12033

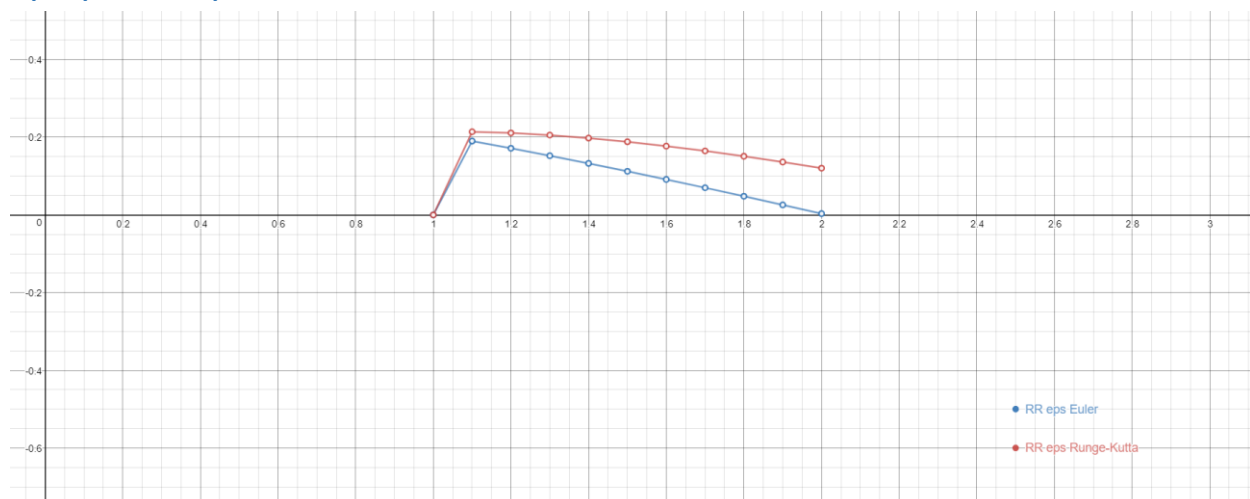
Adams method:

i	X	Y	Z	YIst	YIst-Y
0	1,00000	1,00000	1,00000	1,00000	0,00000
1	1,10000	1,09531	0,90909	1,09531	0,00000
2	1,20000	1,18232	0,83333	1,18232	0,00000
3	1,30000	1,25639	0,77854	1,26236	0,00597
4	1,40000	1,32307	0,73277	1,33647	0,01340
5	1,50000	1,38449	0,69331	1,40547	0,02098
6	1,60000	1,44105	0,65918	1,47000	0,02895
7	1,70000	1,49321	0,62953	1,53063	0,03741
8	1,80000	1,54139	0,60366	1,58779	0,04639
9	1,90000	1,58594	0,58098	1,64185	0,05591
10	2,00000	1,69315	0,50000	1,69315	0,00000

## График для сравнения методов



## График погрешностей


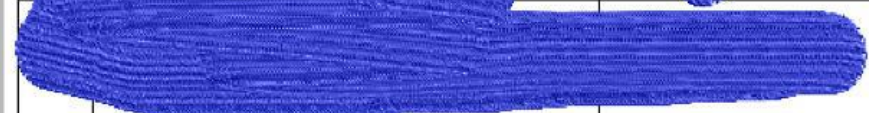




## 4.2

### Задание

4.2. Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Краевая задача	Точное решение
		
		
15	$x^2 \ln x y'' - xy' + y = 0,$ $y'(-1) = 0,$ $y'(1) - y(1) = 0$	$y(x) = 1 + x + \ln x$

### Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NMLab4
{
    class SKR
    {
        public SKR(double t0, double tk, double stepOfRg4)
        {
            this.t0 = t0;
            this.tk = tk;
            this.stepOfRg4 = stepOfRg4;
            razm = 3;
            double temp1 = 1 / Math.E;
            double temp2 = 1 / (2 * Math.E * Math.E);
            x0 = new double[3] { temp1, 0, t0 };
            xk = new double[3] { temp2, 0, tk };
            nnode = (int)((tk - t0) / stepOfRg4);
            n1 = 1;
            n2 = 0.9;
        }

        private double t0, tk, stepOfRg4, n1, n2;
    }
}
```

```

private int nnode, razm;
private double[] x0, xk;

public void calcMS(double[,] res, double[,] RRpogr)
{
    for (int i = 0; true; i++)
    {
        double nImin1 = n2;
        double nImin2 = n1;
        double n = 0;
        if (i < 2) { if (i == 0) { n = n1; } else n = n2; }
        else
        {
            double Fimin1, Fimin2;
            x0[1] = nImin1;
            rg4(nnode, stepOfRg4, t0, x0, res, razm);
            Fimin1 = res[nnode - 1, 0] - xk[0];
            x0[1] = nImin2;
            rg4(nnode, stepOfRg4, t0, x0, res, razm);
            Fimin2 = res[nnode - 1, 0] - xk[0];

            n = nImin1 - (nImin1 - nImin2) * Fimin1 / (Fimin1 - Fimin2);
        }
        x0[1] = n;
        rg4(nnode, stepOfRg4, t0, x0, res, razm);
        if (Math.Abs(res[nnode - 1, 0] - xk[0]) < 0.0001) { break; }
        else { if (i > 1) { n1 = n2; n2 = n; } }
    }
    double[,] tempXout = new double[nnode / 2, 3];
    rg4(nnode / 2, stepOfRg4 * 2, t0, x0, tempXout, razm);
    for (int i = 0; i < nnode / 2; i++)
    {
        double temp = (res[2 * i, 0] - tempXout[i, 0]) / (2 * 2 - 1);
        RRpogr[i, 0] = temp;
        RRpogr[i, 1] = t0 + 2 * i * stepOfRg4;
    }
}

public void calcKR(double[,] res, double[,] RRpogr)
{
    Matrix A = new Matrix(nnode, '0');
    A.matrix[0, 0] = -2 - stepOfRg4 * stepOfRg4;
    A.matrix[0, 1] = 1 + stepOfRg4 / ((t0 + stepOfRg4));
    A.matrix[nnode - 1, nnode - 2] = 1 - stepOfRg4 / ((t0 + stepOfRg4 *
(nnode)));
    A.matrix[nnode - 1, nnode - 1] = -2 - stepOfRg4 * stepOfRg4;

    for (int i = 1; i < nnode - 1; i++)
    {
        A.matrix[i, i - 1] = 1 - stepOfRg4 / ((t0 + stepOfRg4 * (i + 1)));
        A.matrix[i, i] = -2 - stepOfRg4 * stepOfRg4;
        A.matrix[i, i + 1] = 1 + stepOfRg4 / ((t0 + stepOfRg4 * (i + 1)));
    }

    Vector b = new Vector(nnode);
    b.vector[0] = -(1 - stepOfRg4 / ((t0 + stepOfRg4))) * x0[0];
    b.vector[nnode - 1] = -(1 + stepOfRg4 / ((t0 + stepOfRg4 * (nnode)))) *
xk[0];

    SystemSol slau = new SystemSol(A, b);
    Vector vecRes = slau.ProgonCalculate();

    res[0, 0] = x0[0];
    res[0, 2] = t0;

```

```

        for (int i = 0; i < nnode; i++)
        {
            res[i + 1, 0] = vecRes.vector[i];
            res[i + 1, 2] = t0 + (i + 1) * stepOfRg4;
        }

        Matrix A1 = new Matrix(nnode / 2, '0');
        A1.matrix[0, 0] = -2 - stepOfRg4 * 2 * stepOfRg4 * 2;
        A1.matrix[0, 1] = 1 + stepOfRg4 * 2 / ((t0 + stepOfRg4 * 2));
        A1.matrix[nnode / 2 - 1, nnode / 2 - 2] = 1 - stepOfRg4 * 2 / ((t0 +
stepOfRg4 * 2 * (nnode / 2)));
        A1.matrix[nnode / 2 - 1, nnode / 2 - 1] = -2 - stepOfRg4 * 2 * stepOfRg4 * 2;

        for (int i = 1; i < nnode / 2 - 1; i++)
        {
            A1.matrix[i, i - 1] = 1 - stepOfRg4 * 2 / ((t0 + stepOfRg4 * 2 * (i +
1)));
            A1.matrix[i, i] = -2 - stepOfRg4 * 2 * stepOfRg4 * 2;
            A1.matrix[i, i + 1] = 1 + stepOfRg4 * 2 / ((t0 + stepOfRg4 * 2 * (i +
1)));
        }

        Vector b1 = new Vector(nnode / 2);
        b1.vector[0] = -(1 - stepOfRg4 * 2 / ((t0 + stepOfRg4 * 2))) * x0[0];
        b1.vector[nnode / 2 - 1] = -(1 + stepOfRg4 * 2 / ((t0 + stepOfRg4 * 2 *
(nnode / 2)))) * xk[0];

        SystemSol slau1 = new SystemSol(A1, b1);
        Vector vecRes1 = slau1.ProgonCalculate();

        RRpogr[0, 0] = 0;
        RRpogr[0, 2] = t0;

        for (int i = 0; i < nnode / 2; i++)
        {
            double temp = (vecRes.vector[i] - vecRes1.vector[i]) / (2 * 4 - 1);
            RRpogr[i + 1, 0] = temp;
            RRpogr[i + 1, 2] = t0 + 2 * (i + 1) * stepOfRg4;
        }
    }

    public void OdeSystem(double t, double[] x, double[] f)
    {
        f[0] = x[1];
        f[1] = x[0] - 2 * x[1] / t;
    }

    private void rg4(int nnode, double stepOfRg4, double t0, double[] x0, double[, ]
xout, int razm)
    {
        double t1, t2, t3, t4;
        double[] x1, x2, x3, x4;
        x1 = new double[razm];
        x2 = new double[razm];
        x3 = new double[razm];
        x4 = new double[razm];
        double[] k1, k2, k3, k4;
        k1 = new double[razm];
        k2 = new double[razm];
        k3 = new double[razm];
        k4 = new double[razm];

        for (int i = 0; i < razm; i++)
        {

```

```

        x1[i] = x0[i];
        xout[0, i] = x0[i];
    }

    for (int j = 1; j < nnode; j++)
    {
        t1 = t0 + (j - 1) * stepOfRg4;
        OdeSystem(t1, x1, k1);

        t2 = t1 + stepOfRg4 / 2;
        for (int i = 0; i < razm; i++) { x2[i] = x1[i] + stepOfRg4 * k1[i] / 2.0; }

        OdeSystem(t2, x2, k2);

        t3 = t2;
        for (int i = 0; i < razm; i++) { x3[i] = x1[i] + stepOfRg4 * k2[i] / 2.0; }

        OdeSystem(t3, x3, k3);

        t4 = t1 + stepOfRg4;
        for (int i = 0; i < razm; i++) { x4[i] = x1[i] + stepOfRg4 * k3[i]; }
        OdeSystem(t4, x4, k4);

        for (int i = 0; i < razm; i++) { x1[i] = x1[i] + stepOfRg4 * (k1[i] + 2 *
k2[i] + 2 * k3[i] + k4[i]) / 6.0; }
        for (int i = 0; i < razm; i++) { xout[j, i] = x1[i]; }
        xout[j, razm - 1] = t4;
    }
}
}
}

```

## Результат выполнения

2

Shooting method:

\*\*\*\*\*

i	xi	yi
0	0,367879441171442	-0,756224971017089
1	0,300749245808227	-0,594718386725093
2	0,247565556464	-0,474698836378558
3	0,204850516436243	-0,38367582064763
4	0,170141046084027	-0,313475361707456
5	0,141648594754939	-0,25857248236766
6	0,118046621678599	-0,215129006541448
7	0,0983332103634192	-0,180415307745296
8	0,0817395900604036	-0,152450445944707
9	0,0676676416183055	-0,12977131799336

i rreps

0	0
2	-2,45363411218169E-05
4	-2,89914352859059E-05

\*\*\*\*\*

Konechno-raznostniy method:

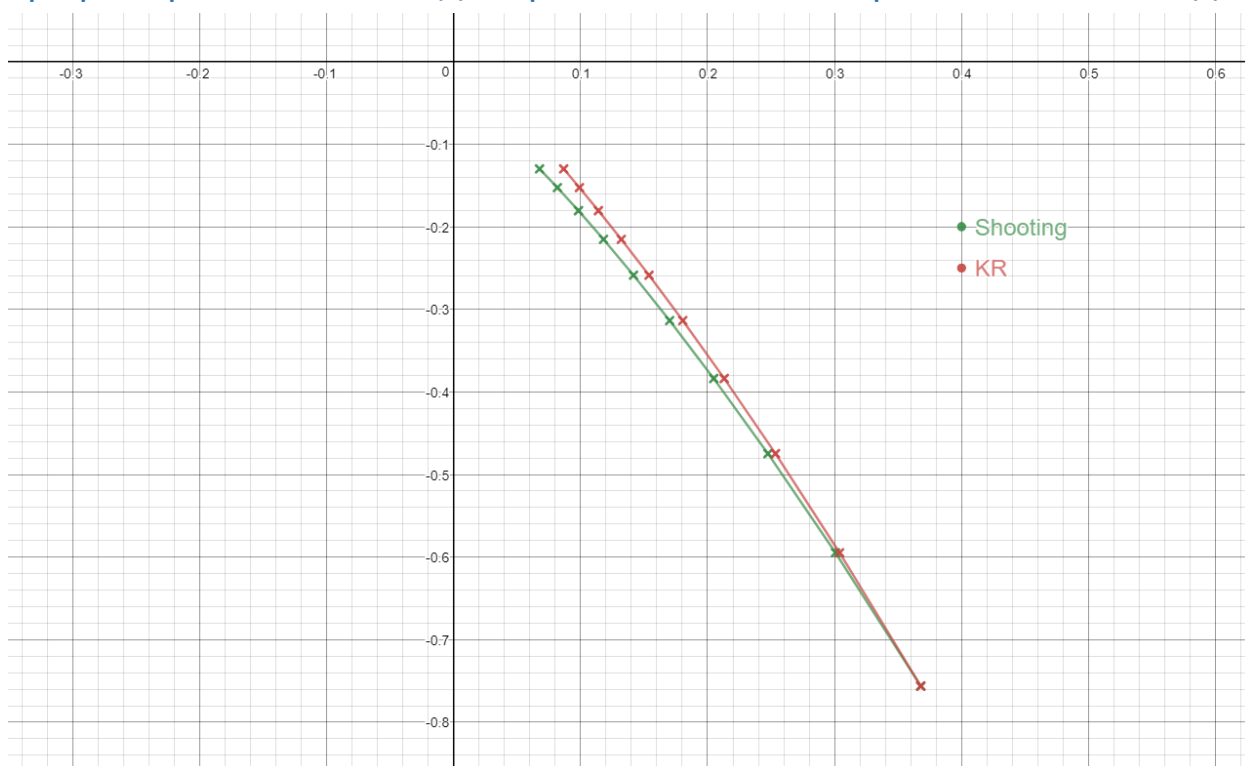
\*\*\*\*\*

i	xi	yi
0	0,367879441171442	-0,756224971017089
1	0,303958436927885	-0,594718386725093
2	0,253477219063425	-0,474698836378558
3	0,213102132123315	-0,38367582064763
4	0,18047372025865	-0,313475361707456

5	0,153880184698355	-0,25857248236766
6	0,132053467814644	-0,215129006541448
7	0,114037456143743	-0,180415307745296
8	0,0991002439665215	-0,152450445944707
9	0,086674214329743	-0,12977131799336
i	rreps	
0	0	
2	0,00695292089933302	
4	0,00997985171344356	
6	0,0109727908898561	
8	0,0108777161399645	
10	0,0101949884071078	

\*\*\*\*\*

## График сравнения метода стрельбы и конечно-разностного метода



## Выводы

В последнем разделе данного семестра предлагается реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка для решения задачи Коши для ОДУ 2-го порядка. С заданием я справился, для наглядности построил графики. С их помощью наглядно можно сравнить результаты работы каждого из методов. Также были реализованы метод стрельбы и конечно-разностный метод для решения краевой задачи ОДУ 2-го порядка и построены графики для их сравнения.