Helper.cs

```
namespace Project
{
    public struct Matrix
    {
        public int N;
        public int maxIter;
        public double epsilon;
        public uint[] ig, jg;
        public double[] di, gg, pr, x, absolut_x;

        //* Умножение матрицы на вектор
        public double[] mult(double[] x) {
            var y = new double[x.Length];

            for (uint i = 0, jj = 0; i < x.Length; i++) {
                y[i] = di[i] * x[i];

                for (uint j = ig[i]; j < ig[i + 1]; j++, jj++) {
                   y[i]      += gg[jj] * x[jg[jj]];
                   y[jg[jj]] += gg[jj] * x[i];
                }
            }
            return y;
        }
    }

    public static class Helper
    {
        //* Сумма матриц
        public static double[][] SummMatrix(double[][] Mat1, double[][] Mat2)
        {
            var Mat3 = new double[3][];
            for (uint i = 0; i < 3; i++) Mat3[i] = new double[3];

            for (uint i = 0; i < Mat1.Length; i++)
                for (uint j = 0; j < Mat1.Length; j++)
                    Mat3[i][j] = Mat1[i][j] + Mat2[i][j];
            return Mat3;
        }

        //* Середина ребра (между двумя узлами)
        public static double[] MidPoints(double[] point1, double[] point2)
        {
            var midpoint = new double[2];
            midpoint[0] = (point1[0] + point2[0]) / 2.0;
            midpoint[1] = (point1[1] + point2[1]) / 2.0;
            return midpoint;
        }

        //* Вычисление скалярного произведение
        public static double scalar(double[] array1, double[] array2) {
            double res = 0;
            for (uint i = 0; i < array1.Length; i++)
                res += array1[i] * array2[i];
            return res;
        }
    }
}
```

Data.cs

```
using static System.Diagnostics.Debug;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;

namespace Project
{
    public class Data
    {

    // ************ Данные задачи ************ //
        protected (double gamma, double betta)[] materials ; /// Значения гамма и бетта

        protected uint       countNode     ; /// Количество узлов
        protected uint       countFinitEl  ; /// Количество конечных элементов
        protected uint       countAreas    ; /// Количество областей
        protected uint       countKrayCond ; /// Количество краевых условий

        protected double[][]  nodes         ; /// Координаты узлов
        protected uint[][]    finitElements ; /// Координаты конечных элементов
        protected uint[]      areaFinitEl   ; /// номера областей в которых расположен кон. эл. по порядку
        protected uint[][]    boards        ; /// Условия на границах
        protected uint[]      nodesArea     ; /// Область для каждого узла

        //? Для составления списка связанностей
        public uint[]         numNodes      ; /// Номера узлов

    // ************ Конструктор DATA (чтение данных) ************ //
         public Data(string Path) { Assert(Read(Path), "Несуществующий файл или неккоректные данные!"); }

        //* Чтение данных
        private bool Read(string path)
        {
            bool isCorr = true;

            isCorr &= readParams(path + "param.txt");

            resize(); // Выделение памяти

            isCorr &= readfiles    (path + "xy.txt",      ref nodes                        );
            isCorr &= readMaterial (path + "area.txt",    ref materials, ref areaFinitEl   );
            isCorr &= readfiles    (path + "board.txt",   ref boards                       );
            isCorr &= readNumNodes (path + "finitel.txt", ref numNodes, ref finitElements  );
            isCorr &= readNodesArea(path + "XY_area.txt", ref nodesArea                    );

            // Отсортировать массив board по номеру краевого условия
            boards = boards.OrderByDescending(l => l[3]).ToArray();

            return isCorr;
        }

        private bool readfiles(string path, ref double[][] array)
        {
            if (!File.Exists(path)) return false;
            string[] lines = File.ReadAllLines(path);

            for (uint i = 0; i < lines.Length; i++) {
                var values = lines[i].Split(" ").Select(n => double.Parse(n)).ToArray();
                for (uint j = 0; j < values.Length; j++)
                    array[i][j] = values[j];
            }
            return true;
```

```
    }

    private bool readfiles(string path, ref uint[][] array)
    {
        if (!File.Exists(path)) return false;
        string[] lines = File.ReadAllLines(path);

        for (uint i = 0; i < lines.Length; i++) {
            var values = lines[i].Split(" ").Select(n => uint.Parse(n)).ToArray();
            for (uint j = 0; j < values.Length; j++)
                array[i][j] = values[j];
        }
        return true;
    }

    private bool readNodesArea(string path, ref uint[] array)
    {
        if (!File.Exists(path)) return false;
        array = File.ReadAllText(path).Split(" ").Select(n => uint.Parse(n)).ToArray();
        return true;
    }

    private bool readNumNodes(string path, ref uint[] numNodes, ref uint[][] finEl)
    {
        if (!File.Exists(path)) return false;
        string[] lines = File.ReadAllLines(path);

        List<uint> listTemp = new List<uint>();

        for (int i = 0; i < lines.Length; i++)
        {
            var values = lines[i].Split(" ").Select(n => uint.Parse(n)).ToArray();
            for (int j = 0; j < values.Length; j++) {
                if (!listTemp.Contains(values[j]))
                    listTemp.Add(values[j]);
                finEl[i][j] = values[j];
            }
        }
        // Заполнение массива для списка связанностей
        numNodes = listTemp.OrderBy(n => ((uint)n)).ToArray();
        return true;
    }

    private bool readMaterial(string path, ref (double gamma, double betta)[] material, ref uint[] area)
    {
        if (!File.Exists(path)) return false;
        StreamReader file = new StreamReader(path);
        for (int i = 0; i < countAreas; i++) {
            double[] line = file.ReadLine().Split(" ").Select(n => double.Parse(n)).ToArray();
            material[i] = (line[0], line[1]);
        }
        areaFinitEl = file.ReadLine().Split(" ").Select(n => uint.Parse(n)).ToArray();
        file.Close();
        return true;
    }

    private bool readParams(string path)
    {
        if (!File.Exists(path)) return false;
        bool isCorr = true;
        StreamReader file = new StreamReader(path);
        string[] param = file.ReadLine().Split(" ");

        isCorr &= uint.TryParse(param[0], out countNode);
        isCorr &= uint.TryParse(param[1], out countFinitEl);
        isCorr &= uint.TryParse(param[2], out countAreas);
        isCorr &= uint.TryParse(param[3], out countKrayCond);
```

```
            file.Close();
            return isCorr;
        }

        private void resize()
        {
            nodes         = new double           [countNode]    [];
            finitElements = new uint             [countFinitEl] [];
            materials     = new (double, double) [countAreas]     ;
            areaFinitEl   = new uint             [countFinitEl]   ;
            boards        = new uint             [countKrayCond][];
            numNodes      = new uint             [countNode]      ;
            for (uint i = 0; i < countNode    ; i++) nodes[i]         = new double[2];
            for (uint i = 0; i < countFinitEl ; i++) finitElements[i] = new uint[3];
            for (uint i = 0; i < countKrayCond; i++) boards[i]        = new uint[5];
        }

        //* Данные задачи
        public string dataAll()
        {
            System.Console.OutputEncoding = Encoding.GetEncoding(65001);
            string margin = String.Join("", Enumerable.Repeat("-", 50));
            StringBuilder output = new StringBuilder();

            output.Append($"{margin}\n");
            output.Append($"(Количество узлов) countNode                = {countNode}    \n" +
                          $"(Количество конечных элементов) countFinitEl = {countFinitEl} \n" +
                          $"(Количество областей) countAreas             = {countAreas}   \n" +
                          $"(Количество граничных условий)               = {countKrayCond}\n");

            output.Append($"{margin}\n");
            output.Append("(Координаты узлов) nodes: \n");
            for (int i = 0; i < countNode; i++)
                output.Append($"({nodes[i][0]}, " +
                              $"{nodes[i][1]}) -> область {nodesArea[i]}\n");

            output.Append($"{margin}\n");
            output.Append("(Координаты конечных элементов) finitElements: \n");
            for (int i = 0; i < countFinitEl; i++)
                output.Append($"({finitElements[i][0]}, " +
                              $"{finitElements[i][1]}, "  +
                              $"{finitElements[i][2]}) -> область {areaFinitEl[i]}\n");

            output.Append($"{margin}\n");
            output.Append("(гамма и бетта для каждой области) materials: \n");
            for (int i = 0; i < countAreas; i++)
                output.Append($"{i}: \u03B3 = {materials[i].gamma}, \u03B2 = {materials[i].betta})\n");

            output.Append($"{margin}\n");
            output.Append("Данные граничных условий: \n");
            for (int i = 0; i < countKrayCond; i++)
                output.Append($"({boards[i][0]}, " +
                               $"{boards[i][1]}, " +
                               $"{boards[i][2]}, " +
                               $"{boards[i][3]}, " +
                               $"{boards[i][4]})\n");

            return output.ToString();
        }

    }
}
```

Function.cs

```
using static System.Math;

namespace Project
{
    public static class Function
    {
        public static uint NumberFunc;

        //* Функция u(x,y)
        public static double Absolut(double x, double y, uint area)
        {
            switch(NumberFunc)
            {
                case 1: /// test1
                return area switch
                {
                    0 => x,
                    _ => 0,
                };

                case 2: /// test2-SPLIT
                return area switch
                {
                    0 => 10*x + 10*y,
                    _ => 0,
                };

                case 3: /// test3-STUDY/test1
                return area switch
                {
                    0 => y*y,
                    1 => 20*y - 19,
                    _ => 0,
                };

                case 4: /// test3-STUDY/test2
                return area switch
                {
                    0 => x + 6*y - 2,
                    1 => x + 6*y - 2,
                    _ => 0,
                };

                case 5: /// test4-Decomposition
                return area switch
                {
                    0 => x,
                    _ => 0,
                };

            }
            return 0;
        }

        //* Функция f(x,y)
        public static double Func(double x, double y, uint area)
        {
            switch(NumberFunc)
            {
                case 1: /// test1
                return area switch
                {
                    0 => 2*x,
                    _ => 0,
```

```
            };

            case 2: /// test2-SPLIT
            return area switch
            {
                0 => 20*x + 20*y,
                _ => 0,
            };

            case 3: /// test3-STUDY/test1
            return area switch
            {
                0 => -20,
                1 => 0,
                _ => 0,
            };

            case 4: /// test3-STUDY/test2
            return area switch
            {
                0 => 5*x + 30*y - 10,
                1 => 0,
                _ => 0,
            };

            case 5: /// test4-Decomposition
            return area switch
            {
                0 => 2*x - 1,
                _ => 0,
            };

        }
        return 0;
    }

    //* Функция lambda(x,y)
    public static double Lambda(double x, double y, uint area)
    {
        switch(NumberFunc)
        {
            case 1: /// test1
            return area switch
            {
                0 => 1,
                _ => 0,
            };

            case 2: /// test2-SPLIT
            return area switch
            {
                0 => 4,
                _ => 0,
            };

            case 3: /// test3-STUDY/test1
            return area switch
            {
                0 => 10,
                1 => 1,
                _ => 0,
            };

            case 4: /// test3-STUDY/test2
            return area switch
            {
                0 => 1,
                1 => 1,
```

```
                _ => 0,
            };

            case 5: /// test4-Decomposition
            return area switch
            {
                0 => x,
                _ => 0,
            };
        }
        return 0;
    }

    //* Функция первого краевого условия
    public static double Func_First_Kraev(double x, double y, uint area)
    {
        switch(NumberFunc)
        {
            case 1: /// test1
            return area switch
            {
                0 => x,
                _ => 0,
            };

            case 2: /// test2-SPLIT
            return area switch
            {
                0 => 50 + 10*y,
                _ => 0,
            };

            case 3: /// test3-STUDY/test1
            return area switch
            {
                0 => y*y,
                _ => 0,
            };

            case 4: /// test3-STUDY/test2
            return area switch
            {
                0 => 6*y + 2,
                _ => 0,
            };

            case 5: /// test4-Decomposition
            return area switch
            {
                0 => x,
                _ => 0,
            };
        }
        return 0;
    }

    //* Функция второго краевого условия
    public static double Func_Second_Kraev(double x, double y, uint area, uint lam_area)
    {
        switch(NumberFunc)
        {
            case 1: /// test1
            return area switch
            {
                0 => 1,
                1 => -1,
                _ => 0,
            };
```

```
            case 2: /// test2-SPLIT
            return area switch
            {
                0 => -40,
                1 => 40,
                _ => 0,
            };

            case 3: /// test3-STUDY/test1
            return area switch
            {
                0 => 20,
                1 => 0,
                _ => 0,
            };

            case 4: /// test3-STUDY/test2
            return area switch
            {
                0 => -6,
                1 => -1,
                2 => 6,
                _ => 0,
            };

            case 5: /// test4-Decomposition
            return area switch
            {
                0 => x,
                1 => -x,
                _ => 0,
            };
        }
        return 0;
    }

    //* Функция третьего краевого условия
    public static double Func_Third_Kraev(double x, double y, uint area, uint lam_area)
    {
        switch(NumberFunc)
        {
            case 1: /// test1
            return area switch
            {
                0 => x,
                _ => 0,
            };

            case 2: /// test2-SPLIT
            return area switch
            {
                0 => 10*x + 2,
                _ => 0,
            };

            case 3: /// test3-STUDY/test1
            return area switch
            {
                0 => 20*y - 27,
                _ => 0,
            };

            case 4: /// test3-STUDY/test2
            return area switch
            {
                0 => 6*y + 2.1,
                _ => 0,
```

```
                };

                case 5: /// test4-Decomposition
                return area switch
                {
                    0 => x,
                    _ => 0,
                };
            }
            return 0;
        }


    }
}
```

LOS.cs

```
using static Project.Helper;
using static System.Math;
using System;
using System.Linq;

namespace Project
{
    public class LOS
    {
        private Matrix matrix;    /// Структура СЛАУ

    // ************ Коструктор LOS ************ //
        public LOS(Matrix matrix, int maxIter, double eps) {
            this.matrix         = matrix;
            this.matrix.maxIter = maxIter;
            this.matrix.epsilon = eps;
        }

        //* Решение СЛАУ
        public Matrix solve(bool isLog = true) {
            var r      = new double[matrix.N];
            var z      = new double[matrix.N];
            var multLr = new double[matrix.N];
            var Lr     = new double[matrix.N];
            var p      = new double[matrix.N];
            double alpha, betta, Eps;
            int iter = 0;

            double[] L = Enumerable.Range(0, matrix.N).Select(i => 1.0 / matrix.di[i]).ToArray();

            double[] multX = matrix.mult(matrix.x);
            for (int i = 0; i < r.Length; i++) {
                r[i] = L[i] * (matrix.pr[i] - multX[i]);
                z[i] = L[i] * r[i];
            }
            double[] multZ = matrix.mult(z);
            for (int i = 0; i < p.Length; i++)
                p[i] = L[i] * multZ[i];

            do
            {
                betta = scalar(p, p);
                alpha = scalar(p, r) / betta;
                for (int i = 0; i < matrix.x.Length; i++) {
                    matrix.x[i] += alpha * z[i];
                    r[i]        -= alpha * p[i];
                    Lr[i]       = L[i] * r[i];
                }

                multLr = matrix.mult(Lr);
                for (int i = 0; i < Lr.Length; i++)
                    multLr[i] = L[i] * multLr[i];
                betta = -scalar(p, multLr) / betta;
                for (int i = 0; i < z.Length; i++) {
                    z[i] = L[i] * r[i] + betta * z[i];
                    p[i] = multLr[i] + betta * p[i];
                }
                Eps = scalar(r, r);

                iter++;
                if (isLog) printLog(iter, Eps);
            } while (iter < matrix.maxIter
                && Eps > matrix.epsilon);
```

```
            return matrix;
        }

        //* Вывод невязки и количества итераций
        private void printLog(int Iter, double Eps) {
            Console.WriteLine($"Iteration = {Iter}\t\t" +
                              $"Discrepancy = {Eps}");
        }
    }
}
```

FEM.cs

```
using static System.Console;
using static System.Math;
using static Project.Function;
using static Project.Helper;
using System.Text;
using System.Linq;
using System.IO;
using System;
using System.Collections.Generic;

namespace Project
{
    public class FEM : Data
    {
        protected string Path; /// Путь к папке с задачей
        private Matrix matrix; /// Структура СЛАУ

        // ************ Коструктор FEM ************ //
        public FEM(string Path, uint Num) : base(Path) { this.Path = Path; Function.NumberFunc = Num; }

        public void solve()
        {
            portrait();                             //? Составление списка связанностей и массивов ig[] и jg[]
            global();                               //? Составление глобальный матрицы
            LOS los = new(matrix, 10000, 9e-030);   //? Создание метода LOS
            matrix = los.solve(false);              //? Решение СЛАУ методом ЛОС (диагональный)
            AbsolutSolve();                         //? Абсолютное решение СЛАУ
            WriteMatrix();                          //? Запись матрицы и вектора решения СЛАУ
            WriteTable();                           //? Запись таблички с решением и погрешностью

        }

        //* Составление портрета
        private void portrait()
        {
            // Составление списка связанностей
            var list = new uint[countNode][];

            var listI = new HashSet<uint>();
            for (uint i = 0; i < numNodes.Length; i++)
            {
                uint value = numNodes[i];
                for (uint k = 0; k < countFinitEl; k++)
                {
                    if (finitElements[k].Contains(value))
                        for (uint p = 0; p < 3; p++)
                            if (finitElements[k][p] < value)
                                listI.Add(finitElements[k][p]);
                }
                list[i] = listI.OrderBy(n => ((uint)n)).ToArray();
                listI.Clear();
            }
```

```
        // Заполнение ig[]
        matrix.ig = new uint[countNode + 1];
        matrix.ig[0] = matrix.ig[1] = 0;
        for (uint i = 1; i < countNode; i++)
            matrix.ig[i + 1] = (matrix.ig[i] + (uint)list[i].Length);

        // Заполнение jg[]
        matrix.jg = new uint[matrix.ig[countNode]];
        int jj = 0;
        for (uint i = 0; i < countNode; i++)
            for (uint j = 0; j < list[i].Length; j++, jj++)
                matrix.jg[jj] = list[i][j];

        // Размерность глобальной матрицы
        matrix.N = (int)countNode;
        resize(); // Выделение памяти
    }

    //* Построение глоабльной матрицы
    private void global()
    {
        // Для каждого конечного элемента
        for (uint index_fin_el = 0; index_fin_el < countFinitEl; index_fin_el++)
        {
            // Составим локальную матрицу и локальный вектор
            (double[][] local_matrix, double[] local_f) = local(index_fin_el);

            // Занесение в глобальную
            EntryMatInGlobalMatrix(local_matrix, finitElements[index_fin_el]);
            EntryVecInGlobalMatrix(local_f, finitElements[index_fin_el]);
        }

        // Для каждого условия на границе
        for (uint index_kraev_cond = 0; index_kraev_cond < countKrayCond; index_kraev_cond++)
        {
            uint[] curr_kraev = boards[index_kraev_cond];   // Данные краевого условия
            uint[] Node = { curr_kraev[1], curr_kraev[2] }; // Ребро на котором задано условие
            if (curr_kraev[3] == 1)
                First_Kraev(curr_kraev);
            else if (curr_kraev[3] == 2)
            {
                double[] corr_vec = Second_Kraev(curr_kraev);
                EntryVecInGlobalMatrix(corr_vec, Node);
            }
            else
            {
                (double[][] corr_mat, double[] corr_vec) = Third_Kraev(curr_kraev);
                EntryMatInGlobalMatrix(corr_mat, Node);
                EntryVecInGlobalMatrix(corr_vec, Node);
            }
        }
    }

    //* Построение локальной матрицы и вектора
    private (double[][], double[]) local(uint index_fin_el)
    {
        double[] local_f = build_F(index_fin_el);    // Построение локального вектора
        double[][] M = build_M(index_fin_el);        // Построение матрица массы
        double[][] G = build_G(index_fin_el);        // Построение матрица жесткости
        double[][] local_matrix = SummMatrix(G, M);  // Локальная матрицы (G + M)

        return (local_matrix, local_f);
    }

    //* Занесение матрицы в глоабальную матрицу
    private void EntryMatInGlobalMatrix(double[][] mat, uint[] index)
    {
```

```
        for (uint i = 0, h = 0; i < mat.GetUpperBound(0) + 1; i++)
        {
            uint ibeg = index[i];
            for (uint j = i + 1; j < mat.GetUpperBound(0) + 1; j++)
            {
                uint iend = index[j];
                uint temp = ibeg;

                if (temp < iend)
                    (iend, temp) = (temp, iend);

                h = matrix.ig[temp];
                while (matrix.jg[h++] - iend != 0);
                matrix.gg[--h] += mat[i][j];
            }
            matrix.di[ibeg] += mat[i][i];
        }
    }

    //* Занесение вектора в глолбальный вектор
    private void EntryVecInGlobalMatrix(double[] vec, uint[] index)
    {
        for (uint i = 0; i < vec.Length; i++)
            matrix.pr[index[i]] += vec[i];
    }

    //* Построение вектора правой части
    private double[] build_F(uint index_fin_el)
    {
        uint area_fin_el = areaFinitEl[index_fin_el];        // Область в которой расположек к.э.
        double detD = Abs(ComputeDet(index_fin_el)) / 24.0;  // Вычисление detD

        var f = new double[3];                               // Вычисление f - на узлах к.э.
        for (uint i = 0; i < f.Length; i++)
            f[i] = detD * Func(
                 nodes[finitElements[index_fin_el][i]][0],
                 nodes[finitElements[index_fin_el][i]][1],
                 area_fin_el
            );

        var local_f = new double[3];                         // Вычисление локального вектора
        local_f[0] = 2 * f[0] + f[1] + f[2];
        local_f[1] = 2 * f[1] + f[0] + f[2];
        local_f[2] = 2 * f[2] + f[1] + f[0];

        return local_f;
    }

    //* Построение матрицы масс
    private double[][] build_M(uint index_fin_el)
    {
        var M_matrix = new double[3][];                                  // Матрица масс
        for (uint i = 0; i < 3; i++) M_matrix[i] = new double[3];

        uint area_fin_el = areaFinitEl[index_fin_el];                    // Область в которой расположек к.э.
        double gamma = materials[area_fin_el].gamma;                     // Значение гаммы в этой области

        double value = gamma * Abs(ComputeDet(index_fin_el)) / 24.0;     // Значение матрицы массы

        for (uint i = 0; i < M_matrix.GetUpperBound(0) + 1; i++)         // Заполнение матрицы масс
            for (uint j = 0; j < M_matrix.GetUpperBound(0) + 1; j++)
                M_matrix[i][j] = i == j ? 2 * value
                                        : value;
        return M_matrix;
    }

    //* Построение матрицы жесткости
    private double[][] build_G(uint index_fin_el)
```

```
    {
        var G_matrix = new double[3][];                             // Матрица жесткости
        for (uint i = 0; i < 3; i++) G_matrix[i] = new double[3];

        double[] Node1 = nodes[finitElements[index_fin_el][0]];     // Координаты 1 узла i - конечного элемента
        double[] Node2 = nodes[finitElements[index_fin_el][1]];     // Координаты 2 узла i - конечного элемента
        double[] Node3 = nodes[finitElements[index_fin_el][2]];     // Координаты 3 узла i - конечного элемента
        double[] Mid12 = MidPoints(Node1, Node2);                   // Координаты середины ребра 1-2
        double[] Mid13 = MidPoints(Node1, Node3);                   // Координаты серидины ребра 1-3
        double[] Mid23 = MidPoints(Node2, Node3);                   // Координаты середины ребра 2-3

        uint area_fin_el = areaFinitEl[index_fin_el];               // Область в которой расположек к.э.
        double[,] a = ComputeA(index_fin_el);                       // Вычисление a-компонент

        double lambda = Lambda(Mid12[0], Mid12[1], area_fin_el) +
                        Lambda(Mid13[0], Mid13[1], area_fin_el) +
                        Lambda(Mid23[0], Mid23[1], area_fin_el);    // Подсчет лямбда разложения

        double multip = lambda / (6.0 * Abs(ComputeDet(index_fin_el)));

        for (uint i = 0; i < G_matrix.GetUpperBound(0) + 1; i++)    // Заполнение матрицы жесткости
            for (uint j = 0; j < G_matrix.GetUpperBound(0) + 1; j++)
                G_matrix[i][j] = multip * (a[i, 0] * a[j, 0] + a[i, 1] * a[j, 1]);

        return G_matrix;
    }

    //* Первое краевое условие
    private void First_Kraev(uint[] kraev)
    {
        // Ставим вместо диагонального эл. единицу
        matrix.di[kraev[1]] = 1;
        matrix.di[kraev[2]] = 1;

        // В вектор правой части ставим значение краевого условия
        matrix.pr[kraev[1]] = Func_First_Kraev(nodes[kraev[1]][0], nodes[kraev[1]][1], kraev[4]);
        matrix.pr[kraev[2]] = Func_First_Kraev(nodes[kraev[2]][0], nodes[kraev[2]][1], kraev[4]);

        // Зануляем в строке все стоящие элементы кроме диагонального и сразу делаем симметричной
        for (uint k = 1; k < 3; k++)
        {

            // Зануление в нижнем треугольнике
            for (uint i = matrix.ig[kraev[k]]; i < matrix.ig[kraev[k] + 1]; i++)
            {
                if (matrix.di[matrix.jg[i]] != 1)
                    matrix.pr[matrix.jg[i]] -= matrix.gg[i] * matrix.pr[kraev[k]];
                matrix.gg[i] = 0;
            }

            // Зануление в верхнем треугольнике, но т.к. делаем симметричную "зануление в нижнем"
            for (uint i = kraev[k] + 1; i < countNode; i++)
            {
                uint lbeg = matrix.ig[i];
                uint lend = matrix.ig[i + 1];
                for (uint p = lbeg; p < lend; p++)
                    if (matrix.jg[p] == kraev[k])
                    {
                        if (matrix.di[i] != 1)
                            matrix.pr[i] -= matrix.gg[p] * matrix.pr[kraev[k]];
                        matrix.gg[p] = 0;
                    }
            }
        }
    }

    //* Второе краевое условие
    private double[] Second_Kraev(uint[] kraev)
```

```
    {
        var corr_vec = new double[2];                               // Корректирующий вектор

        uint[] Node = { kraev[1], kraev[2] };                       // Ребро

        double betta = materials[kraev[0]].betta;                   // Значение бетты
        double multip = ComputeMesG(nodes[Node[0]], nodes[Node[1]]) / 6.0;

        for (uint i = 0, j = 1; i < corr_vec.Count(); i++, j--)     // Заполнение вектора
            corr_vec[i] = multip * (2 * Func_Second_Kraev(nodes[Node[i]][0], nodes[Node[i]][1], kraev[4], kraev[0]) +
                                       Func_Second_Kraev(nodes[Node[j]][0], nodes[Node[j]][1], kraev[4], kraev[0]));
        return corr_vec;
    }

    //* Третье краевое условие
    private (double[][], double[]) Third_Kraev(uint[] kraev)
    {
        var corr_vec = new double[2];                               // Корректирующий вектор
        var corr_mat = new double[2][];                             // Корректирующая матрица
        for (uint i = 0; i < 2; i++) corr_mat[i] = new double[2];

        uint[] Node = { kraev[1], kraev[2] };                       // Ребро
        double betta = materials[kraev[0]].betta;                   // Значение Betta

        double multip = (betta * ComputeMesG(nodes[Node[0]], nodes[Node[1]])) / 6.0;

        for (uint i = 0, k = 1; i < corr_vec.Count(); i++, k--)
        {  // Заполнение вектора и матрицы
            corr_vec[i] = multip * (2 * Func_Third_Kraev(nodes[Node[i]][0], nodes[Node[i]][1], kraev[4], kraev[0]) +
                                       Func_Third_Kraev(nodes[Node[k]][0], nodes[Node[k]][1], kraev[4], kraev[0]));

            for (uint j = 0; j < corr_mat.Count(); j++)
                corr_mat[i][j] = i == j ? 2 * multip
                                        : multip;
        }
        return (corr_mat, corr_vec);
    }

    //* Подсчет компонента detD
    private double ComputeDet(uint index_fin_el)
    {
        double[] Node1 = nodes[finitElements[index_fin_el][0]]; // Координаты 1 узла i - конечного элемента
        double[] Node2 = nodes[finitElements[index_fin_el][1]]; // Координаты 2 узла i - конечного элемента
        double[] Node3 = nodes[finitElements[index_fin_el][2]]; // Координаты 3 узла i - конечного элемента

        return (Node2[0] - Node1[0]) * (Node3[1] - Node1[1]) -
               (Node3[0] - Node1[0]) * (Node2[1] - Node1[1]);
    }

    //* Подсчет компонента mes по ребру G
    private double ComputeMesG(double[] Node1, double[] Node2)
    {
        return Sqrt(Pow((Node2[0] - Node1[0]), 2) +
                    Pow((Node2[1] - Node1[1]), 2));
    }

    //* Подсчет компонентов a
    private double[,] ComputeA(uint index_fin_el)
    {
        var a = new double[3, 2];
        double[] Node1 = nodes[finitElements[index_fin_el][0]]; // Координаты 1 узла i - конечного элемента
        double[] Node2 = nodes[finitElements[index_fin_el][1]]; // Координаты 2 узла i - конечного элемента
        double[] Node3 = nodes[finitElements[index_fin_el][2]]; // Координаты 3 узла i - конечного элемента

        // Заполнение a
        a[0, 0] = Node2[1] - Node3[1];
        a[1, 0] = Node3[1] - Node1[1];
        a[2, 0] = Node1[1] - Node2[1];
```

```
        a[0, 1] = Node3[0] - Node2[0];
        a[1, 1] = Node1[0] - Node3[0];
        a[2, 1] = Node2[0] - Node1[0];
        return a;
    }

    //* Абсолютное решение СЛАУ
    private void AbsolutSolve()
    {
        for (uint i = 0; i < countNode; i++)
            matrix.absolut_x[i] = Absolut(nodes[i][0], nodes[i][1], nodesArea[i]);
    }

    //* resize массивов глобальной матрицы
    private void resize()
    {
        matrix.di = new double[matrix.N];
        matrix.pr = new double[matrix.N];
        matrix.x = new double[matrix.N];
        matrix.absolut_x = new double[matrix.N];
        matrix.gg = new double[matrix.jg.Length];
    }

    //* Запись глобальной матрицы
    private void WriteMatrix()
    {
        Directory.CreateDirectory(Path + "matrix");
        Directory.CreateDirectory(Path + "output");
        File.WriteAllText(Path + "matrix/kuslau.txt", $"{matrix.N} {matrix.maxIter} {matrix.epsilon}");
        File.WriteAllText(Path + "matrix/ig.txt", String.Join(" ", matrix.ig));
        File.WriteAllText(Path + "matrix/jg.txt", String.Join(" ", matrix.jg));
        File.WriteAllText(Path + "matrix/di.txt", String.Join("\n", matrix.di));
        File.WriteAllText(Path + "matrix/gg.txt", String.Join("\n", matrix.gg));
        File.WriteAllText(Path + "matrix/pr.txt", String.Join("\n", matrix.pr));
        File.WriteAllText(Path + "output/x.txt", String.Join("\n", matrix.x));
        File.WriteAllText(Path + "output/x_absolut.txt", String.Join("\n", matrix.absolut_x));
    }

    //* Запись таблички с погрешностью
    private void WriteTable()
    {
        (double[] SubX, double norma) = Norm(matrix.absolut_x, matrix.x);

        StringBuilder table = new StringBuilder();
        string margin = String.Join("", Enumerable.Repeat("-", 23));

        table.Append(String.Join("", Enumerable.Repeat("-", 97)) + "\n");
        table.Append($"|X`{" ",-20} | X{" ",-20} | |X` - X|{" ",-13} | ||X` - X||  {" ",-9} |\n");
        table.Append($"|" + margin + "|" + margin + "|" + margin + "|" + margin + "|\n");

        for (uint i = 0; i < countNode; i++)
        {
            table.Append($"|{String.Format("{0,-23}", matrix.absolut_x[i])}" +
                         $"|{String.Format("{0,-23}", matrix.x[i])}" +
                         $"|{SubX[i].ToString("E3")}{String.Format("{0,-13}", "")}|");
            if (countNode / 2 == i)
                table.Append($"{norma.ToString("E3")}{String.Format("{0,-13}", "")}|");
            else
                table.Append($"{String.Format("{0,-23}", " ")}|");
            table.Append("\n");
        }
        table.Append(String.Join("", Enumerable.Repeat("-", 97)) + "\n");
        File.WriteAllText(Path + "output/table.txt", table.ToString());
    }

    //* Расчет погрешности и нормы решения
    private (double[], double) Norm(double[] x_abs, double[] x)
    {
```

```
            double norm = 0;
            double[] norm_arr = new double[x.Length];

            for (int i = 0; i < x.Length; i++)
            {
                norm_arr[i] = Abs(x_abs[i] - x[i]);
                norm += Pow(norm_arr[i], 2);
            }
            return (norm_arr, Sqrt(norm));
        }

        //* Вывод матрицы на консоль
        private void PrintMatrix()
        {
            WriteLine("Matrix: ");
            PrintArray(matrix.di, "di        = [");
            PrintArray(matrix.gg, "gg        = [");
            PrintArray(matrix.ig, "ig        = [");
            PrintArray(matrix.jg, "jg        = [");
            PrintArray(matrix.pr, "pr        = [");
        }

        //* Вывод массива на консоль
        private void PrintArray<T>(T[] array, string design = "array = [")
        {
            Write(design);
            for (int i = 0; i < array.Length; i++)
                Write(array[i] + " ");
            WriteLine("]");
        }
    }
}
```

Program.cs

```
using static System.Console;
using Project;

FEM task = new("file/test1/", 1);               //! Обычный тест с одной областью в начале координат
//FEM task = new("file/test2-SPLIT/split1/", 2);  //! Тест с разбиением #1
//FEM task = new("file/test2-SPLIT/split2/", 2);  //! Тест с разбиением #2
//FEM task = new("file/test2-SPLIT/split3/", 2);  //! Тест с разбиением #3
//FEM task = new("file/test2-SPLIT/split4/", 2);  //! Тест с разбиением #4
//FEM task = new("file/test3-STUDY/study1/", 3);  //! Тест с книги МКЭ #1
//FEM task = new("file/test3-STUDY/study2/", 4);  //! Тест с книги МКЭ #2
//FEM task = new("file/test4-Decomposition/", 5); //! Тест с книги МКЭ #2

task.solve();                                   //! Запуск решения задачи
//WriteLine(task.dataAll());                    //! Вывод всех данных задачи
```