

```

namespace Project.other;

public struct Node    /// Структура Узла
{
    public double x { get; set; }    /// Координата X
    public double y { get; set; }    /// Координата Y

    public Node(double _x, double _y) {
        x = _x; y = _y;
    }

    public void Deconstructor(out double x,
                              out double y)
    {
        x = this.x;
        y = this.y;
    }

    public void Deconstructor(out double[] param) {
        param = new double[]{this.x, this.y};
    }

    public override string ToString() => $"{x,20} {y,24}";
}

public struct Elem    /// Структура КЭ
{
    public int[] Node;    /// Узлы КЭ

    public Elem(params int[] node) { Node = node; }

    public void Deconstructor(out int[] nodes) { nodes = this.Node; }

    public override string ToString() {
        StringBuilder str_elem = new StringBuilder();
        str_elem.Append($"{Node[0],5}");
        for (int i = 1; i < Node.Count(); i++)
            str_elem.Append($"{Node[i],8}");
        return str_elem.ToString();
    }
}

public struct Kraev    /// Структура краевого
{
    public int[] Node;    /// Узлы краевого
    public int NumKraev;    /// Номер краевого
    public int CountNumKraev;    /// Номер по счету этого краевого

    public Kraev(int numKraev, int counNumKraev, params int[] node) {
        Node = node;
        this.NumKraev = numKraev;
        this.CountNumKraev = counNumKraev;
    }

    public void Deconstructor(out int num, out int count, out int[] nodes) {
        nodes = this.Node;
        num = this.NumKraev;
        count = this.CountNumKraev;
    }

    public override string ToString() {
        StringBuilder str_elem = new StringBuilder();
        str_elem.Append($"{Node[0],5}");
        for (int i = 1; i < Node.Count(); i++)

```

```

        str_elem.Append($"{Node[i],8}");
        return str_elem.ToString();
    }
}

public struct SLAU    /// Структура СЛАУ
{
    public Vector di, gg;           /// Матрица
    public int[] ig, jg;           /// Массивы с индексами
    public Vector f, q;            /// Правая часть и решение
    public Vector q_absolut;       /// Абсолютные значения U-функции
    public int N;                  /// Размерность матрицы
    public int N_el;               /// Размерность g1 и gu

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

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

            for (int 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;
    }

    /** Очистка массивов
    public void Clear() {
        Vector.Clear(di);
        Vector.Clear(gg);
        Vector.Clear(f);
        Vector.Clear(q);
    }
}

public static class Helper
{
    /** Скалярное произведение векторов
    public static double Scalar(Vector first, Vector scnd) {
        double res = 0;
        for (int i = 0; i < first.Length; i++)
            res += first[i]*scnd[i];
        return res;
    }

    /** Окно помощи при запуске (если нет аргументов или по команде)
    public static void ShowHelp() {
        WriteLine("----Команды----          \n" +
            "-help                - показать справку          \n" +
            "-i                    - входной файл              \n");
    }
}

```

```

namespace Project;
public class Data
{
    /** Данные для генерации сетки
    public double[][] Nodes    { get; set; }    /// Координаты узлов
    public int[][]   Elems    { get; set; }    /// КЭ
    public int[][]   Kraevs   { get; set; }    /// Краевые условия
    public uint      N        { get; set; }    /// Номер задачи
    public double    start_t  { get; set; }    /// Начальная точка сетки по времени
    public double    end_t    { get; set; }    /// Конечная точка сетки по времени
    public double    h_t      { get; set; }    /// Шаг сетки по времени
    public double    k_t      { get; set; }    /// Коэффициент разрядки сетки по времени

    /** Деконструктор
    public void Deconstruct(out Node[]  nodes,
                           out Elem[]   elems,
                           out Kraev[]  kraevs,
                           out Vector   time)
    {
        nodes = new Node[Nodes.Count()];
        for (int i = 0; i < nodes.Length; i++)
            nodes[i] = new Node(Nodes[i][0], Nodes[i][1]);

        elems = new Elem[Elems.Count()];
        for (int i = 0; i < elems.Length; i++)
            elems[i] = new Elem(Elems[i]);

        kraevs = new Kraev[Kraevs.Count()];
        for (int i = 0; i < kraevs.Length; i++) {
            int[] temp = Kraevs[i];
            kraevs[i] = new Kraev(temp[2], temp[3], new int[]{temp[0], temp[1]});
        }
        // Сортировка краевых (первые краевые должны быть в конце)
        for (int i = 0; i < kraevs.Length; i++) {
            for (int j = i + 1; j < Kraevs.Length; j++) {
                if (kraevs[i].NumKraev < kraevs[j].NumKraev)
                    (kraevs[i], kraevs[j]) = (kraevs[j], kraevs[i]);
            }
        }

        // Генерация сетки по времени
        int n = k_t != 1
            ? (int)(Log(1 - (end_t - start_t)*(k_t - 1) / (h_t*(-1))) / Log(k_t) + 2)
            : (int)((end_t - start_t) / h_t + 1);
        time = new Vector(n);
        time[0] = start_t;
        double h = h_t;
        for (int i = 1; i < n - 1; i++, h *= k_t)
            time[i] = time[i - 1] + h;
        time[n - 1] = end_t;
    }
}

```

```

namespace Project;
public static class Function
{
    public static uint    NumberFunc;    /// Номер задачи
    public static double  betta;        /// Значение betta

    public static void Init(uint numF) {
        NumberFunc = numF;

        switch(NumberFunc) {
            case 3:                /// OneFileEl_ThirdKraev
                betta = 5;
                break;

            case 4:                /// Split-test
                betta = 4;
                break;

        }
    }

    /* Абсолютное значение U-функции
    public static double Absolut(Vector vec, double t) {
        (double x, double y) = vec;
        return NumberFunc switch
        {
            1 => 2*x + y + t,          /// OneFileEl_FirstKraev
            2 => 2*x + y + t,          /// OneFileEl_SecondKraev
            3 => 2*x + y + t,          /// OneFileEl_ThirdKraev
            4 => 10*x + 10*y + 10*t,    /// Split-test
            5 => t*t*t,                /// Approxi

            _ => 0,

        };
    }

    /* Значения F-функции
    public static double F(Vector vec, double t) {
        (double x, double y) = vec;
        return NumberFunc switch
        {
            1 => 5,                    /// OneFileEl_FirstKraev
            2 => 5,                    /// OneFileEl_SecondKraev
            3 => 5,                    /// OneFileEl_ThirdKraev
            4 => 10,                   /// Split-test
            5 => 3*t*t + 6*t,          /// Approxi

            _ => 0,

        };
    }

    /* Значение Lambda
    public static double Lambda(Vector vec) {
        (double x, double y) = vec;
        return NumberFunc switch
        {
            1 => 8,                    /// OneFileEl_FirstKraev
            2 => 8,                    /// OneFileEl_SecondKraev
            3 => 8,                    /// OneFileEl_ThirdKraev
            4 => 2,                    /// Split-test
            5 => 1,                    /// Approxi

            _ => 0,

        };
    }

```

```

}

/* Значение Sigma
public static double Sigma(Vector vec) {
    (double x, double y) = vec;
    return NumberFunc switch
    {
        1 => 5,           /// OneFileEl_FirstKraev
        2 => 5,           /// OneFileEl_SecondKraev
        3 => 5,           /// OneFileEl_ThirdKraev
        4 => 1,           /// Split-test
        5 => 1,           /// Approxi
        _ => 0,
    };
}

/* Значение Hi
public static double Hi(Vector vec) {
    (double x, double y) = vec;
    return NumberFunc switch
    {
        1 => 2,           /// OneFileEl_FirstKraev
        2 => 2,           /// OneFileEl_SecondKraev
        3 => 2,           /// OneFileEl_ThirdKraev
        4 => 3,           /// Split-test
        5 => 1,           /// Approxi
        _ => 0,
    };
}

/* Значения первого краевого
public static double Func_First_Kraev(Vector vec, double t, int count_kraev) {
    (double x, double y) = vec;
    switch (NumberFunc)
    {
        case 1:           /// OneFileEl_FirstKraev
            return count_kraev switch
            {
                0 => 2*x + 1 + t,
                1 => 10 + y + t,
                2 => 2*x + 9 + t,
                3 => 2 + y + t,
                _ => 0
            };

        case 2:           /// OneFileEl_SecondKraev
            return count_kraev switch
            {
                0 => 2*x + 1 + t,
                1 => 2*x + 9 + t,
                _ => 0
            };

        case 3:           /// OneFileEl_ThirdKraev
            return count_kraev switch
            {
                0 => 2*x + 1 + t,
                _ => 0
            };

        case 4:           /// Split-test
            return count_kraev switch
            {
                0 => 50 + 10*y + 10*t,
                _ => 0
            };
    }
}

```

```

};

case 5:                                     /// Approxi
return count_kraev switch
{
    0 => t*t*t,
    1 => t*t*t,
    2 => t*t*t,
    3 => t*t*t,
    _ => 0
};

}
return 0;
}

/* Значения второго краевого
public static double Func_Second_Kraev(Vector vec, double t, int count_kraev) {
    (double x, double y) = vec;
    switch (NumberFunc)
    {
        case 2:                             /// OneFileEl_SecondKraev
        return count_kraev switch
        {
            0 => -16,
            1 => 16,
            _ => 0
        };

        case 3:                             /// OneFileEl_ThirdKraev
        return count_kraev switch
        {
            0 => -16,
            1 => 16,
            _ => 0
        };

        case 4:                             /// Split-test
        return count_kraev switch
        {
            0 => -20,
            1 => 20,
            _ => 0
        };

    }
    return 0;
}

/* Значения третьего краевого
public static double Func_Third_Kraev(Vector vec, double t, int count_kraev) {
    (double x, double y) = vec;
    switch (NumberFunc)
    {
        case 3:                             /// OneFileEl_ThirdKraev
        return count_kraev switch
        {
            0 => 53/5.0 + 2*x + t,
            _ => 0
        };

        case 4:                             /// Split-test
        return count_kraev switch
        {
            0 => 5 + 10*x + 10*t,
            _ => 0
        };
    }
}

```

```
}  
    return 0;  
}
```

## LOS.cs

```
namespace Project;  
public class LOS  
{  
    private SLAU slau;        /// Структура СЛАУ  
  
    private int maxIter;      /// Максимальное количество итераций  
    private double EPS;       /// Точность  
  
    /// ***** Конструктор LOS ***** //  
    public LOS(SLAU slau, int maxIter, double eps) {  
        this.slau = slau;  
        this.maxIter = maxIter;  
        this.EPS = eps;  
    }  
  
    /* Решение СЛАУ  
    public Vector solve(bool isLog = true) {  
        var r = new Vector(slau.N);  
        var z = new Vector(slau.N);  
        var multLr = new Vector(slau.N);  
        var Lr = new Vector(slau.N);  
        var p = new Vector(slau.N);  
        double alpha, betta, Eps;  
        int iter = 0;  
  
        double[] L = Enumerable.Range(0, slau.N).Select(i => 1.0 / slau.di[i]).ToArray();  
  
        Vector multX = slau.mult(slau.q);  
        for (int i = 0; i < r.Length; i++) {  
            r[i] = L[i] * (slau.f[i] - multX[i]);  
            z[i] = L[i] * r[i];  
        }  
        Vector multZ = slau.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 < slau.q.Length; i++) {  
                slau.q[i] += alpha * z[i];  
                r[i] -= alpha * p[i];  
                Lr[i] = L[i] * r[i];  
            }  
  
            multLr = slau.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 < maxIter &&
            Eps > EPS);

    return slau.q;
}

/* Вывод невязки на определенной итерации
private void printLog(int Iter, double Eps) {
    WriteLine($"Iteration = {Iter}\\t\\t" +
            $"Discrepancy = {Eps}");
}
}

```

## Portrait.cs

```

namespace Project.other;

public class Portrait
{
    private int countNode;
    private int[] lportrait;

    /* Конструктор
    public Portrait(int n_nodes) {
        this.countNode = n_nodes;
        lportrait = new int[this.countNode];
        for (int i = 0; i < this.countNode; i++)
            lportrait[i] = i;
    }

    /* Генерация ig, jg (размерность - n)
    public int GenPortrait(ref int[] ig, ref int[] jg, Elem[] elems) {

        var list = new int[countNode][];

        var listI = new HashSet<int>();
        for (int i = 0; i < lportrait.Length; i++) {
            int value = lportrait[i];
            for (int k = 0; k < elems.Count(); k++) {
                if (elems[k].Node.Contains(value))
                    for (int p = 0; p < elems[k].Node.Count(); p++)
                        if (elems[k].Node[p] < value)
                            listI.Add(elems[k].Node[p]);
            }
            list[i] = listI.OrderBy(n => n).ToArray();
            listI.Clear();
        }

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

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

        return ig[countNode];
    }
}

```



```

}

namespace Project;
public class FEM
{
    private Node[]    Nodes;           /// Узлы
    private Elem[]    Elems;           /// КЭ
    private Kraev[]   Kraevs;          /// Краевые
    private SLAU      slau;             /// Структура СЛАУ
    private Vector     Time;            /// Слои

    private Vector     Uprev1;          /// Значения на слое (j-1)
    private Vector     Uprev2;          /// Значения на слое (j-2)

    public string Path { get; set; }    /// Путь к задаче

    public FEM(Data data, string path) {
        (Nodes, Elems, Kraevs, Time) = data;
        this.Path = path;
    }

    public void solve() {

        /// Запись слоев в файл
        File.WriteAllText(Path + "/layers.txt", String.Join("\n", ((double[])Time)));

        /// Подсчет функции на 2-х предыдущих слоях
        Uprev1 = new Vector(Nodes.Length);
        Uprev2 = new Vector(Nodes.Length);
        for (int i = 0; i < Nodes.Length; i++) {
            var vec = new Vector(new double[] {Nodes[i].x, Nodes[i].y});
            Uprev1[i] = Absolut(vec, Time[1]);
            Uprev2[i] = Absolut(vec, Time[0]);
        }

        /// Составление портрета матрицы
        portrait();

        /// Создание директории с решениями на слоях
        Directory.CreateDirectory(Path + "/solve_layers");

        /// Подсчет для каждого слоя (начиная с 3)
        for (int i = 2; i < Time.Length; i++) {
            global(i);                                /// Составление глобальной матрицы
            Vector.Copy(Uprev2, slau.q);                /// Начальное приближение
            LOS los = new LOS(slau, 10000, 9e-030);      /// Создание метода LOS
            los.solve(true);                             /// Решение СЛАУ методом ЛОС (диагональный)
            AbsolutSolve(i);                             /// Подсчет абсолютного значения на слое
            WriteTable(Path + "/solve_layers", i);       /// Запись таблички с погрешностью в файл

            /// Замена слоев
            Vector.Copy(Uprev1, Uprev2);
            Vector.Copy(slau.q, Uprev1);

            /// Очистка СЛАУ
            slau.Clear();
        }
    }

    /** Составление портрета матрицы (ig, jg, выделение памяти)
    private void portrait() {
        Portrait port = new Portrait(Nodes.Length);
    }

```

```

// Генерируем массивы ig и jg и размерность
slau.N_el = port.GenPortrait(ref slau.ig, ref slau.jg, Elems);
slau.N     = Nodes.Length;

// Выделяем память
slau.gg     = new Vector(slau.N_el);
slau.di     = new Vector(slau.N);
slau.f      = new Vector(slau.N);
slau.q      = new Vector(slau.N);
slau.q_absolut = new Vector(slau.N);
}

/* Построение глобальной матрицы
private void global(int index_t) {

    // Для каждого КЭ
    for (int index_fin_el = 0; index_fin_el < Elems.Length; index_fin_el++) {
        // Составляем локальную матрицу и локальный вектор
        (double[][] loc_mat, Vector local_f) = local(index_fin_el, index_t);

        // Занесение в глобальную
        EntryMatInGlobalMatrix(loc_mat, Elems[index_fin_el].Node);
        EntryVecInGlobalMatrix(local_f, Elems[index_fin_el].Node);
    }

    // Для каждого условия на границе
    for (int index_kraev_cond = 0; index_kraev_cond < Kraevs.Length; index_kraev_cond++) {
        Kraev cur_kraev = Kraevs[index_kraev_cond];
        if (cur_kraev.NumKraev == 1)
            First_Kraev(cur_kraev, index_t);
        else if (cur_kraev.NumKraev == 2) {
            Vector corr_vec = Second_Kraev(cur_kraev, index_t);
            EntryVecInGlobalMatrix(corr_vec, Kraevs[index_kraev_cond].Node);
        } else {
            (double[][] corr_mat, Vector corr_vec) = Third_Kraev(cur_kraev, index_t);
            EntryMatInGlobalMatrix(corr_mat, Kraevs[index_kraev_cond].Node);
            EntryVecInGlobalMatrix(corr_vec, Kraevs[index_kraev_cond].Node);
        }
    }
}

/* Занесение матрицы в глобальную матрицу
private void EntryMatInGlobalMatrix(double[][] mat, int[] index) {
    for (int i = 0, h = 0; i < mat.GetUpperBound(0) + 1; i++) {
        int ibeg = index[i];
        for (int j = i + 1; j < mat.GetUpperBound(0) + 1; j++) {
            int iend = index[j];
            int temp = ibeg;

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

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

/* Занесение вектора в глобальный вектор
private void EntryVecInGlobalMatrix(Vector vec, int[] index) {
    for (int i = 0; i < vec.Length; i++)
        slau.f[index[i]] += vec[i];
}

/* Построение локальной матрицы и вектора
private (double[][], Vector) local(int index_fin, int index_t) {

```

```

// Формирование векторов с предыдущих слоев для конкретного КЭ
Vector U2 = new Vector(3);
Vector U1 = new Vector(3);
for (int i = 0; i < 3; i++) {
    U2[i] = Uprev2[Elms[index_fin].Node[i]];
    U1[i] = Uprev1[Elms[index_fin].Node[i]];
}

// Подсчитывание локальных компонент
double[][] M    = build_M(index_fin);
double[][] G    = build_G(index_fin);
Vector local_f  = build_F(index_fin, index_t);

///? Подсчитывания с учетом времени

// Подсчет дельт
double delta = Time[index_t]      - Time[index_t - 2];
double delta1 = Time[index_t - 1] - Time[index_t - 2];
double delta0 = Time[index_t]      - Time[index_t - 1];

double[] Node1 = new double[2];
double[] Node2 = new double[2];
double[] Node3 = new double[2];
Nodes[Elms[index_fin].Node[0]].Deconstructor(out Node1); // Координаты 1 узла i - конечного элемента
Nodes[Elms[index_fin].Node[1]].Deconstructor(out Node2); // Координаты 2 узла i - конечного элемента
Nodes[Elms[index_fin].Node[2]].Deconstructor(out Node3); // Координаты 3 узла i - конечного элемента

// Подсчет сигмы и хи
double sigma = (Sigma(new Vector(Node1)) +
                Sigma(new Vector(Node2)) +
                Sigma(new Vector(Node3))) / 3.0;
double hi    = (Hi(new Vector(Node1)) +
                Hi(new Vector(Node2)) +
                Hi(new Vector(Node3))) / 3.0;

//: Составление локального вектора с учетом времени
// Подсчет скобочек
Vector sub1 = (delta0 / (delta1*delta)) * U2 - (delta / (delta0*delta1))*U1;
Vector sub2 = (2 / (delta1*delta)) * U2 - (2 / (delta1*delta0))*U1;

// Умножение на компоненты
sub1 = sigma * sub1;
sub2 = hi * sub2;

// Умножение на матрицу масс
Vector sub1_mull_M = new Vector(3);
Vector sub2_mull_M = new Vector(3);
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++) {
        sub1_mull_M[i] += M[i][j] * sub1[j];
        sub2_mull_M[i] += M[i][j] * sub2[j];
    }

// Локальный вектор
Vector F = local_f - sub1_mull_M - sub2_mull_M;

///? Составление локальной матрицы с учетом времени
double koef1 = (delta + delta0) / (delta*delta0);
double koef2 = 2 / (delta*delta0);
koef1 *= sigma; koef2 *= hi;

double[][] Mat = new double[3][];
for (int i = 0; i < 3; i++) Mat[i] = new double[3];

for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)

```

```

        Mat[i][j] = G[i][j] + koef1*M[i][j] + koef2*M[i][j];

    return (Mat, F);
}

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

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

    double[] Node1 = new double[2];
    double[] Node2 = new double[2];
    double[] Node3 = new double[2];
    Nodes[Elms[index_fin_el].Node[0]].Deconstructor(out Node1); // Координаты 1 узла i - конечного элемента
    Nodes[Elms[index_fin_el].Node[1]].Deconstructor(out Node2); // Координаты 2 узла i - конечного элемента
    Nodes[Elms[index_fin_el].Node[2]].Deconstructor(out Node3); // Координаты 3 узла i - конечного элемента
    double[] Mid12 = MidPoints(Node1, Node2); // Координаты середины ребра 1-2
    double[] Mid13 = MidPoints(Node1, Node3); // Координаты середины ребра 1-3
    double[] Mid23 = MidPoints(Node2, Node3); // Координаты середины ребра 2-3

    double[,] a = ComputeA(index_fin_el); // Вычисление а-компонент

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

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

    // Заполнение матрицы жесткости
    for (uint i = 0; i < G_matrix.GetUpperBound(0) + 1; i++)
        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;
}

/* Построение вектора правой части
private Vector build_F(int index_fin_el, int index_t) {
    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++) {
        Vector vec = new Vector(new double[] { Nodes[Elms[index_fin_el].Node[i]].x, Nodes[Elms[index_fin_el].Node[i]].y });
        f[i] = detD * F(vec, Time[index_t]);
    }

    var local_f = new Vector(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;
}

```

```

}

/** Подсчет компонента detD
private double ComputeDet(int index_fin_el) {
    double[] Node1 = new double[2];
    double[] Node2 = new double[2];
    double[] Node3 = new double[2];
    Nodes[Elms[index_fin_el].Node[0]].Deconstructor(out Node1); // Координаты 1 узла i - конечного элемента
    Nodes[Elms[index_fin_el].Node[1]].Deconstructor(out Node2); // Координаты 2 узла i - конечного элемента
    Nodes[Elms[index_fin_el].Node[2]].Deconstructor(out Node3); // Координаты 3 узла i - конечного элемента

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

/** Середина ребра (между двумя узлами)
public 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;
}

/** Подсчет компонентов a
private double[,] ComputeA(int index_fin_el) {
    var a = new double[3, 2];
    double[] Node1 = new double[2];
    double[] Node2 = new double[2];
    double[] Node3 = new double[2];
    Nodes[Elms[index_fin_el].Node[0]].Deconstructor(out Node1); // Координаты 1 узла i - конечного элемента
    Nodes[Elms[index_fin_el].Node[1]].Deconstructor(out Node2); // Координаты 2 узла i - конечного элемента
    Nodes[Elms[index_fin_el].Node[2]].Deconstructor(out Node3); // 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;
}

/** Подсчет компонента mes по ребру G
private double ComputeMesG(Node Node1, Node Node2) {
    return Sqrt((Node2.x - Node1.x), 2) +
        Pow((Node2.y - Node1.y), 2));
}

/** Учет первого краевого условия
private void First_Kraev(Kraev kraev, int index_t) {
    // Ставим вместо диагонального эл. единицу
    slau.di[kraev.Node[0]] = 1;
    slau.di[kraev.Node[1]] = 1;

    // В вектор правой части ставим значение краевого условия
    Vector vec0 = new Vector(new double[] {Nodes[kraev.Node[0]].x, Nodes[kraev.Node[0]].y});
    Vector vec1 = new Vector(new double[] {Nodes[kraev.Node[1]].x, Nodes[kraev.Node[1]].y});
    slau.f[kraev.Node[0]] = Func_First_Kraev(vec0, Time[index_t], kraev.CountNumKraev);
    slau.f[kraev.Node[1]] = Func_First_Kraev(vec1, Time[index_t], kraev.CountNumKraev);

    // Зануляем в строке все стоящие элементы кроме диагонального и сразу делаем симметричной
    for (int k = 0; k < 2; k++) {

        // Зануление в нижнем треугольнике
        for (int i = slau.ig[kraev.Node[k]]; i < slau.ig[kraev.Node[k] + 1]; i++) {
            if (slau.di[slau.jg[i]] != 1)
                slau.f[slau.jg[i]] -= slau.gg[i] * slau.f[kraev.Node[k]];
        }
    }
}

```

```

        slau.gg[i] = 0;
    }

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

}

/* Второе краевое условие
private Vector Second_Kraev(Kraev kraev, int index_t) {
    var corr_vec = new Vector(2); // Корректирующий вектор

    int[] Node = { kraev.Node[0], kraev.Node[1] }; // Ребро

    double multip = ComputeMesG(Nodes[Node[0]], Nodes[Node[1]]) / 6.0;

    // Заполнение вектора
    for (int i = 0, j = 1; i < corr_vec.Length; i++, j--) {
        Vector veci = new Vector(new double[] { Nodes[Node[i]].x, Nodes[Node[i]].y });
        Vector vecj = new Vector(new double[] { Nodes[Node[j]].x, Nodes[Node[j]].y });
        corr_vec[i] = multip * (2 * Func_Second_Kraev(veci, Time[index_t], kraev.CountNumKraev) +
                                Func_Second_Kraev(vecj, Time[index_t], kraev.CountNumKraev));
    }
    return corr_vec;
}

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

    int[] Node = { kraev.Node[0], kraev.Node[1] }; // Ребро

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

    // Заполнение вектора и матрицы
    for (int i = 0, k = 1; i < corr_vec.Length; i++, k--) {
        Vector veci = new Vector(new double[] { Nodes[Node[i]].x, Nodes[Node[i]].y });
        Vector veck = new Vector(new double[] { Nodes[Node[k]].x, Nodes[Node[k]].y });
        corr_vec[i] = multip * (2 * Func_Third_Kraev(veci, Time[index_t], kraev.CountNumKraev) +
                                Func_Third_Kraev(veck, Time[index_t], kraev.CountNumKraev));

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

/* Расчет погрешности и нормы решения
private (Vector, double) Norm(Vector x_abs, Vector x) {
    double norm = 0;
    Vector norm_arr = new Vector(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));
}

/* Абсолютное решение СЛАУ
private void AbsolutSolve(int index_t) {
    for (int i = 0; i < Nodes.Length; i++) {
        Vector vec = new Vector(new double[] { Nodes[i].x, Nodes[i].y });
        slau.q_absolut[i] = Absolut(vec, Time[index_t]);
    }
}

/* Запись таблички с погрешностью
private void WriteTable(string Path, int index_t) {
    (Vector SubX, double norma) = Norm(slau.q_absolut, slau.q);

    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 (int i = 0; i < Nodes.Length; i++)
    {
        table.Append($"|{String.Format("{0,-23}", slau.q_absolut[i])}" +
            $"{String.Format("{0,-23}", slau.q[i])}" +
            $"{SubX[i].ToString("E3")}{String.Format("{0,-13}", "")}|");
        if (Nodes.Length / 2 == i)
            table.Append($"{{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 + "/layer_" + index_t.ToString() + ".txt", table.ToString());
}
}

```

```
try {

    if (args.Length == 0) throw new ArgumentException("Not found arguments!");
    if (args[0] == "-help") {
        ShowHelp(); return;
    }

    string json = File.ReadAllText(args[1]);
    Data data = JsonConvert.DeserializeObject<Data>(json)!;
    if (data is null) throw new FileNotFoundException("File uncorrected!");

    // Определение функции
    Function.Init(data.N);

    // Метод МКЭ
    FEM task = new FEM(data, Path.GetDirectoryName(args[1]));
    task.solve();
}
catch (FileNotFoundException ex) {
    WriteLine(ex.Message);
}
catch (ArgumentException ex) {
    ShowHelp();
    WriteLine(ex.Message);
}
```