МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | Факультет прикладной
математики и информатики

Кафедра прикладной математики

Ознакомительная практика

Группа ПМ-92 АРТЮХОВ РОМАН

Преподаватели ДОМНИКОВ ПЕТР АЛЕКСАНДРОВИЧ

Новосибирск, 2022

Задача:

МКЭ для двумерной краевой задачи с гармоническими по времени источниками в декартовой системе координат. Базисные функции билинейные на прямоугольниках. Краевые условия первого рода. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

1. Постановка задачи

Уравнение имеет следующий вид:

$$-div\left(\lambda \cdot gradu\right)+i\gamma u = f$$

Уравнение заданно в некоторой области Ω с границей S и краевым условием:

$$u\big|_S = u_g$$

В декартовой системе координат это уравнение может быть записано в виде:

$$-\frac{\partial}{\partial x}\left(\lambda\frac{\partial u}{\partial x}\right)-\frac{\partial}{\partial y}\left(\lambda\frac{\partial u}{\partial y}\right)+i\gamma u = f$$

2. Теоретическая часть

a. Вариационная постановка

Выполняем вариационную постановку методом Галёркина.

В общем виде постановка Галёркина для операторного уравнения $Lu = f$ записывается в следующем виде:

$$\left(Lu - f, v\right) = 0 \quad \forall v \in H_0$$

Где $H_0 = \left\{v \in H,\ v\big|_{S_1} = 0\right\}$

$$\int_\Omega -div\left(\lambda \cdot gradu\right)vd\Omega + \int_\Omega i\gamma uvd\Omega = \int_\Omega fvd\Omega$$

Применяя формулу Грина, перепишем уравнение в виде:

$$\int_\Omega \lambda \cdot gradu \cdot gradvd\Omega + \int_\Omega i\gamma uvd\Omega = \int_\Omega fvd\Omega$$

Исходя из того, что $u = \sum_j^n q_i\psi_i$ перепишем уравнение в виде:

$$\sum_{j=1}^n q_j\int_\Omega \lambda \cdot grad\psi_j \cdot grad\psi_i d\Omega + \sum_{j=1}^n q_j\int_\Omega i\gamma\psi_j\psi_i d\Omega = \int_\Omega f\psi_i d\Omega$$

Исходная задача рассматривается в декартовой системе координат, то

$$gradu = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right)$$

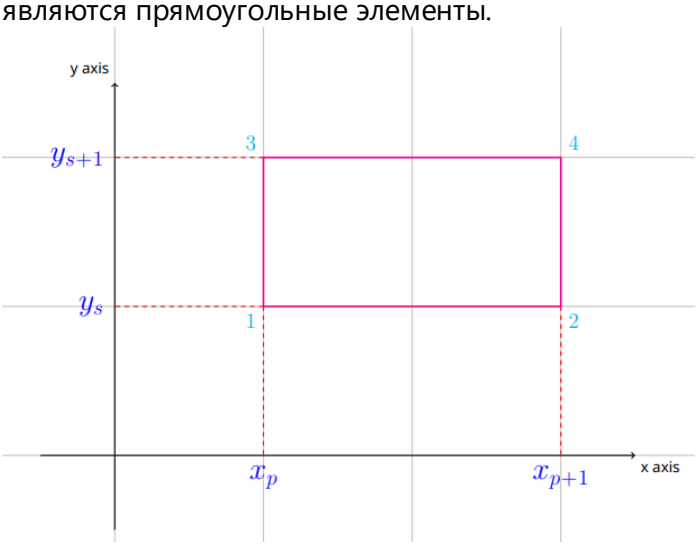
$$gradu \cdot grad\upsilon = \frac{\partial u}{\partial x}\frac{\partial \upsilon}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial \upsilon}{\partial y}$$

Отсюда получаем уравнение в виде:

$$\sum_{j=1}^{n} q_j \int_{\Omega} \lambda \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) d\Omega + \sum_{j=1}^{n} q_j \int_{\Omega} i\gamma \psi_j \psi_i d\Omega = \int_{\Omega} f \psi_i d\Omega$$

b. Конечноэлементная дискретизация

По условию задачи, область исследования разбивается на подобласти, которыми являются прямоугольные элементы.



На каждом элементе мы строим 4 билинейных функции из тех соображений, что в каждом узле только одна функции принимает значение 1, а другие 0.

Билинейные базисные функции определяются следующим образом:

На отрезке (x_p, x_{p+1}) задаются две одномерные линейные функции:

$$X_1(x) = \frac{x_{p+1} - x}{h_x},\ X_2(x) = \frac{x - x_p}{h_x},\ h_x = x_{p+1} - x_p$$

Аналогичные линейные функции задаются и на интервале (y_s, y_{s+1})

$$Y_1(y) = \frac{y_{s+1} - y}{h_y},\ Y_2(y) = \frac{y - y_s}{h_y},\ h_y = y_{s+1} - y_s$$

Локальные базисные функции на конечном элементе представляются в виде произведения вышеперечисленных функций:

$$\hat{\psi}_1(x,y) = X_1(x)Y_1(y),\quad \hat{\psi}_2 = X_2(x)Y_1(y),$$
$$\hat{\psi}_3(x,y) = X_1(x)Y_2(y),\quad \hat{\psi}_4 = X_2(x)Y_2(y).$$

c. Переход к локальным матрицам

Чтобы получить выражения для локальных матриц жёсткости G и массы M каждого конечного элемента Ω_k, перейдем к решению локальной задачи на каждом конечном элементе. Полученное уравнение для области Ω представим в виде суммы интегралов по областям Ω_k без учёта краевых условий.

$$\int\limits_{\Omega_k} \lambda \cdot \left(\frac{\partial \psi_j}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y}\frac{\partial \psi_i}{\partial y} \right) dxdy + \int\limits_{\Omega_k} i\gamma\psi_j\psi_i dxdy = \int\limits_{\Omega_k} f\psi_i dxdy$$

Локальная матрица будет представлять собой сумму матриц жёсткости и массы и будет иметь размерность 4x4.

i. Матрица жёсткости

Рассмотрим первый член в вышеуказанном выражении для k-го конечного элемента:

$$\hat{G}_{ij} = \int\limits_{\Omega_k} \lambda \cdot \left(\frac{\partial \hat{\psi}_j}{\partial x}\frac{\partial \hat{\psi}_i}{\partial x} + \frac{\partial \hat{\psi}_j}{\partial y}\frac{\partial \hat{\psi}_i}{\partial y} \right) dxdy = \int\limits_{x_p}^{x_{p+1}} \int\limits_{y_s}^{y_{s+1}} \bar{\lambda} \left(\frac{\partial \hat{\psi}_j}{\partial x}\frac{\partial \hat{\psi}_i}{\partial x} + \frac{\partial \hat{\psi}_j}{\partial y}\frac{\partial \hat{\psi}_i}{\partial y} \right) dxdy$$

Вычислим одномерные интегралы:

$$\int\limits_{x_p}^{x_p+h_x} \frac{dX_v}{dx} \cdot \frac{dX_u}{dx} dx$$

$$\int\limits_{x_p}^{x_p+h_x} \left(\frac{dX_1}{dx} \right)^2 dx = \frac{1}{h_x},$$

$$\int\limits_{x_p}^{x_p+h_x} \left(\frac{dX_2}{dx} \right)^2 dx = \frac{1}{h_x},$$

$$\int\limits_{x_p}^{x_p+h_x} \frac{dX_1}{dx} \cdot \frac{dX_2}{dx} dx = -\frac{1}{h_x},$$

Аналогичный вид имеют и интегралы от произведений функций $Y(y)$

В результате получим матрицу жесткости, которая имеет следующий вид:

$$\hat{G} = \frac{\bar{\lambda}}{6}\frac{h_y}{h_x}\begin{pmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{pmatrix} + \frac{\bar{\lambda}}{6}\frac{h_x}{h_y}\begin{pmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & -2 & 1 & 2 \end{pmatrix}$$

ii. Построение матрицы массы

Рассмотрим второй член в выражении для k-го конечного элемента:

$$\int_{\Omega_k} i\gamma\psi_j\psi_i dxdy$$

Вычислим одномерные интегралы:

$$\int_{x_p}^{x_p+h_x} X_v X_u dx$$

$$\int_{x_p}^{x_p+h_x} \left(X_1\right)^2 dx = \frac{h_x}{3}$$

$$\int_{x_p}^{x_p+h_x} \left(X_2\right)^2 dx = \frac{h_x}{3}$$

$$\int_{x_p}^{x_p+h_x} X_1 X_2 dx = \frac{h_x}{6}$$

Аналогичный вид имеют и интегралы от произведений функций $Y(y)$

В результате получим матрицу масс, которая имеет следующий вид:

$$\hat{M} = \overline{\gamma}\frac{h_x h_y}{36}\begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix}$$

iii. Построение вектора правой части

Рассмотрим правую часть выражения для k-го конечного элемента:

$$\int_{\Omega_k} f\psi_i dxdy$$

f можно представить в виде: $f = \sum_{v=1}^{4} \hat{f}_v\hat{\psi}_v$

f_v - значение в вершинах прямоугольника.

Получим:

$$\hat{F} = \frac{h_x h_y}{36}\begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix}\cdot\begin{pmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \end{pmatrix}$$

3. Метод решение СЛАУ

Локально-оптимальная схема с диагональным предобуславливанием матрицы

Тести и исследования

Тест №1 (Полином первой степени)

$$\begin{cases} Сетка : [0,1]\times[0,1] \\ Шаги : (h_x, h_y) = (0.2, 0.1) \\ Коэффициент\ \ разрядки : (k_x, k_y) = (1.5, 1.5) \\ u = (x + y,\ 2x + y) \\ \lambda = 2 \\ \gamma = (1,\ 2) \\ f = \gamma(-2x - y,\ x + y) \end{cases}$$

Число итераций	Относительная погрешность
14	2,483310E-015

Тест №2 (Полином второй степени)

$$\begin{cases} Сетка : [0,1]\times[0,1] \\ Шаги : (h_x, h_y) = (0.2, 0.1) \\ Коэффициент\ \ разрядки : (k_x, k_y) = (1.5, 1.5) \\ u = (x^2 + y^2,\ 2x^2 + y^2) \\ \lambda = 2 \\ \gamma = (1,\ 2) \\ f = \gamma(-2x^2 - y^2,\ x^2 + y^2) - (8,\ 12) \end{cases}$$

Число итераций	Относительная погрешность
14	2,324613E-015

Тест №3 (Полином третьей степени)

$$\begin{cases} Сетка: [0,1]\times[0,1] \\ Шаги: (h_x, h_y) = (0.2, 0.1) \\ Коэффициент\ \ разрядки: (k_x, k_y) = (1.5, 1.5) \\ u = \left(x^3 + y^3,\ 2x^3 + y^3\right) \\ \lambda = 2 \\ \gamma = (1,\ 2) \\ f = \gamma\left(-2x^3 - y^3,\ x^3 + y^3\right) - (12x + 12y,\ 24x + 12y) \end{cases}$$

Число итераций	Относительная погрешность
14	2,300400E-015

Тест №4 (Полином четвертой степени)

$$\begin{cases} Сетка: [0,1]\times[0,1] \\ Шаги: (h_x, h_y) = (0.2, 0.1) \\ Коэффициент\ \ разрядки: (k_x, k_y) = (1.5, 1.5) \\ u = \left(x^4 + y^4,\ 2x^4 + y^4\right) \\ \lambda = 2 \\ \gamma = (1,\ 2) \\ f = \gamma\left(-2x^4 - y^4,\ x^4 + y^4\right) - \left(24x^2 + 24y^2,\ 48x^2 + 24y^2\right) \end{cases}$$

Число итераций	Относительная погрешность
14	1,210116E-001

Тест №5 (Оценка порядка аппроксимации; не полином)

$$\begin{cases} Сетка : [0,1]\times[0,1] \\ Шаги : (h_x, h_y) = (0.4, 0.5) = (0.2, 0.25) = (0.1, 0.125) = (0.05, 0.0625) \\ Коэффициент\ \ разрядки : (k_x, k_y) = (1,1) \\ u = (\mathrm{Sin}(x+y), \mathrm{Cos}(2x+y)) \\ \lambda = 2 \\ \gamma = (1,\ 2) \\ f = \gamma(-\mathrm{Cos}(2x+y),\ \mathrm{Sin}(x+y)) - (-4\mathrm{Sin}(x+y), -10\mathrm{Cos}((2x+y))) \end{cases}$$

Координаты центрального узла (x = 0.4, y = 0.5)

Точное значение функции в центральном узле (7,8333E-001, 2,6750E-001)

(h_x,h_y)	Погрешность	Отношение погрешностей	Порядок
(0.4, 0.5)	4,1072E-03	-	-
(0.2, 0.25)	8,7702E-04	4,68317	2,2
(0.1, 0.125)	2,1309E-04	4,11575	2,04
(0.05, 0.0625)	5,2934E-05	4,02556	2,009

Helper.cs

```
using Microsoft.Win32.SafeHandles;
namespace Practice.other;

public struct Grid    /// Структура сетки
{
    public int Count_Node  { get; set; }    /// Общее количество узлов
    public int Count_Elem  { get; set; }    /// Общее количество КЭ
    public int Count_Kraev { get; set; }    /// Количество краевых

    public Node[]  Nodes;      /// Узлы
    public Elem[]  Elems;      /// КЭ
    public Kraev[] Kraevs;     /// Краевые

    public Grid(int count_node, int count_elem, int count_kraev, Node[] nodes, Elem[] elem, Kraev[] kraevs) {
        this.Count_Node  = count_node;
        this.Count_Elem  = count_elem;
        this.Count_Kraev = count_kraev;
        this.Nodes       = nodes;
        this.Elems       = elem;
        this.Kraevs      = kraevs;
    }

    public void Deconstruct(out Node[]  nodes,
                            out Elem[]  elems,
                            out Kraev[] kraevs) {
        nodes  = this.Nodes;
        elems  = this.Elems;
        kraevs = this.Kraevs;
    }
}

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 Deconstruct(out double x,
                            out double y)
    {
        x = this.x;
        y = this.y;
    }

    public void Deconstruct(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) { this.Node = node; }

    public void Deconstruct(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 (ребро)
    public int      numKraev;    /// Номер краевого
    public int      numSide;     /// Номер стороны на котором задано краевое


    public Kraev(int num, int side, params int[] node) {
        this.numKraev = num;
        this.numSide  = side;
        this.Node     = node;
    }

    public void Deconstruct(out int[] nodes, out int num, out int side) {
        nodes = this.Node;
        num   = this.numKraev;
        side  = this.numSide;
    }

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

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

    //* Умножение матрицы на вектор
    public ComplexVector mult(ComplexVector x) {
        var y = new ComplexVector(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;
    }
}

public static class Helper
{

    //* Скалярное произведение векторов
    public static Complex Scalar(ComplexVector frst, ComplexVector scnd) {
        Complex res = 0;
        for (int i = 0; i < frst.Length; i++)
            res += frst[i]*scnd[i];
```

```
        return res;
    }

    //* Модуль комплексного вектора
    public static double Norm(ComplexVector vec) {
        double norm = 0;
        for (int i = 0; i < vec.Length; i++)
            norm += vec[i].Real*vec[i].Real + vec[i].Imaginary*vec[i].Imaginary;
        return Sqrt(norm);
    }

    //* Модуль комплексного числа
    public static double Norm(Complex ch) {
        return Sqrt(ch.Real*ch.Real + ch.Imaginary*ch.Imaginary);
    }

    //* 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");
    }
}
```

Data.cs

```
namespace Practice;
public class Data
{
    //* Данные для генерации сетки
    public double[] start { get; set; }       /// Начальная точка
    public double[] end   { get; set; }       /// Конечная точка
    public double   hx    { get; set; }       /// Шаг по Оси X
    public double   hy    { get; set; }       /// Шаг по Оси Y
    public double   kx    { get; set; }       /// Коэффициент разрядки по Оси X
    public double   ky    { get; set; }       /// Коэффициент разрядки по Оси Y
    public uint     N     { get; set; }       /// Номер функции

    //* Деконструктор
    public void Deconstruct(out Vector<double>   Start,
                            out Vector<double>   End,
                            out double           Hx,
                            out double           Hy,
                            out double           Kx,
                            out double           Ky)
    {
        Start = new Vector<double>(start);
        End   = new Vector<double>(end);
        Hx    = hx;
        Hy    = hy;
        Kx    = kx;
        Ky    = ky;
    }

    //* Проверка входных данных
    public bool Incorrect(out string mes) {
        StringBuilder errorStr = new StringBuilder("");

        if (start[0] > end[0])
            errorStr.Append($"Incorrect data (start[0] > end[0]): {start[0]} > {end[0]}\n");

        if (start[1] > end[1])
            errorStr.Append($"Incorrect data (start[1] > end[1]): {start[1]} > {end[1]}\n");

        if (hx <= 0)
```

```
            errorStr.Append($"Incorrect data (hx <= 0): {hx} <= {0}\n");

        if (hy <= 0)
            errorStr.Append($"Incorrect data (hy <= 0): {hy} <= {0}\n");

        if (hx == hy)
            errorStr.Append($"Incorrect data (hx == hy): {hx} == {hy}\n");

        if (kx < 1)
            errorStr.Append($"Incorrect data (kx < 1): {kx} < {1}\n");

        if (ky < 1)
            errorStr.Append($"Incorrect data (ky < 1): {ky} < {1}\n");

        if (!errorStr.ToString().Equals("")) {
            mes = errorStr.ToString();
            return false;
        }

        mes = errorStr.ToString();
        return true;
    }
}
```

Portrait.cs

```
namespace Practice.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 Vector<int> ig, ref Vector<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 Vector<int>(countNode + 1);
        ig[0] = ig[1] = 0;
        for (int i = 1; i < countNode; i++)
            ig[i + 1] = (ig[i] + list[i].Length);
```

```
        // Заполнение jg[]
        jg = new Vector<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];
    }
}
```

Function.cs

```
namespace Practice;
public static class Function
{
    public static uint      numberFunc;     /// Номер задачи
    public static double    lambda;         /// Значение Lambda
    public static Complex   gamma;          /// Значение gamma
    public static double    betta;          /// Значение betta

    //* Инициализация констант
    public static void Init(uint numF) {
        numberFunc = numF;

        switch(numberFunc) {
            case 1:                                         /// Полином первой степени
                lambda = 2;
                gamma  = new Complex(1, 2);
            break;

            case 2:                                         /// Полином второй степени
                lambda = 2;
                gamma  = new Complex(1, 2);
            break;

            case 3:                                         /// Полином третьей степени
                lambda = 2;
                gamma  = new Complex(1, 2);
            break;

            case 4:                                         /// Полином четвертой степени
                lambda = 2;
                gamma  = new Complex(1, 2);
            break;

            case 5:                                         /// Не полином
                lambda = 2;
                gamma  = new Complex(1, 2);
            break;

        }
    }

    //* Абсолютное значение U-функции
    public static Complex Absolut(Vector<double> vec) {
        (double x, double y) = (vec[0], vec[1]);
        return numberFunc switch
        {
            1 => new Complex(x + y, 2*x + y),                                   /// Полином первой степени
            2 => new Complex(x*x + y*y, 2*x*x + y*y),                           /// Полином второй степени
            3 => new Complex(Pow(x, 3) + Pow(y, 3), 2*Pow(x, 3) + Pow(y, 3)),   /// Полином третьей степени
            4 => new Complex(Pow(x, 4) + Pow(y, 4), 2*Pow(x, 4) + Pow(y, 4)),   /// Полином четвертой степени
            5 => new Complex(Sin(x + y), Cos(2*x + y)),                         /// Не полином
```

```
            _ => 0,
        };
    }

    //* Значения F-функции
    public static Complex F(Vector<double> vec) {
        (double x, double y) = (vec[0], vec[1]);
        return numberFunc switch
        {
            1 => gamma * (new Complex(-2*x - y, x + y)),
            2 => gamma * (new Complex(-2*x*x - y*y, x*x + y*y)) - new Complex(8, 12),
            3 => gamma * (new Complex(-2*Pow(x, 3) - Pow(y, 3), Pow(x, 3) + Pow(y, 3))) - new Complex(12*x + 12*y, 24*x + 12
            4 => gamma * (new Complex(-2*Pow(x, 4) - Pow(y, 4), Pow(x, 4) + Pow(y, 4))) - new Complex(24*x*x + 24*y*y, 48*x
            5 => gamma * (new Complex(-Cos(2*x + y), Sin(x + y))) - new Complex(-4*Sin(x + y), -10*Cos(2*x + y)),

            _ => 0,
        };
    }


}
```

Generate.cs

```
namespace Practice;
public class Generate
{
    protected Vector<double> start  { get; set; }      /// Начальная точка
    protected Vector<double> end    { get; set; }      /// Конечная точка
    protected double hx             { get; set; }      /// Шаг по Оси X
    protected double hy             { get; set; }      /// Шаг по Оси Y
    protected double kx             { get; set; }      /// Коэффициент деления по Оси X
    protected double ky             { get; set; }      /// Коэффициент деления по Оси Y

    protected string Path           { get; set; }      /// Путь к папке с задачей

    private int N_X;                                        /// Количество узлов по Оси X
    private int N_Y;                                        /// Количество узлов по Оси Y
    private int Count_Node  => N_X * N_Y;                   /// Общее количество узлов
    private int Count_Elem  => (N_X - 1)*(N_Y - 1);         /// Общее количество КЭ
    private int Count_Kraev => 2*(N_X - 1) + 2*(N_Y - 1);   /// Количество краевых
    private int[]? SideKraev;                               /// Номера краевых на сторонах

    //* Конструктор
    public Generate(Data data, string Path) {
        (start, end, hx, hy, kx, ky) = data;
        this.Path = Path;

        // Подсчет количества узлов на Осях
        N_X = kx != 1
            ? (int)(Log(1 - (end[0] - start[0])*(kx - 1) / (hx*(-1))) / Log(kx) + 2)
            : (int)((end[0] - start[0]) / hx + 1);
        N_Y = ky != 1
            ? (int)(Log(1 - (end[1] - start[1])*(ky - 1) / (hy*(-1))) / Log(ky) + 2)
            : (int)((end[1] - start[1]) / hy + 1);
    }

    //* Инициализации сторон краевыми
    public void SetKraev(int side0, int side1, int side2, int side3) {
        SideKraev = new int[] {side0, side1, side2, side3};
    }

    //* Основная функция генерации сетки
    public Grid generate() {
        if (SideKraev == null) throw new ArgumentException($"Boundary conditions are not set!\nUse the function \"SetKraev\
```

```
        Path += "/grid";
        Directory.CreateDirectory(Path);
        Node[]  nodes  = generate_coords(); //? Генерация координат
        Elem[]  elems  = generate_elems();  //? Генерация КЭ
        Kraev[] kraevs = generate_kraevs(); //? Генерация краевых

        return new Grid(Count_Node, Count_Elem, Count_Kraev, nodes, elems, kraevs);
    }

    //* Генерация координат
    private Node[] generate_coords() {
        Vector<double> X_vec = generate_array(start[0], end[0], hx, kx, N_X);
        Vector<double> Y_vec = generate_array(start[1], end[1], hy, ky, N_Y);

        Node[] nodes = new Node[Count_Node];

        for (int i = 0; i < N_X; i++)
            for (int j = 0; j < N_Y; j++)
                    nodes[j*N_X + i] = new Node(X_vec[i], Y_vec[j]);

        File.WriteAllText(Path + "/coords.txt", String.Join("\n", nodes));
        return nodes;
    }

    //* Генерация массива по Оси (с шагом и коэффицентом разрядки)
    private Vector<double> generate_array(double start, double end, double h, double k, int n) {
        var coords = new Vector<double>(n);
        coords[0]     = start;
        coords[n - 1] = end;
        for (int i = 1; i < n - 1; i++, h *= k)
            coords[i] = coords[i - 1] + h;


        return coords;
    }

    //* Генерация КЭ
    private Elem[] generate_elems() {
        Elem[] elems = new Elem[Count_Elem];

        for (int i = 0, id = 0; i < N_Y - 1; i++)
            for (int j = 0; j < N_X - 1; j++, id++) {
                    elems[id] = new Elem(
                          i   *N_X + j,
                          i   *N_X + j + 1,
                         (i+1)*N_X + j,
                         (i+1)*N_X + j + 1
                    );
            }

        File.WriteAllText(Path + "/elems.txt", String.Join("\n", elems));
        return elems;
    }

    //* Генерация краевых
    private Kraev[] generate_kraevs() {
        Kraev[] kraevs = new Kraev[Count_Kraev];
        int id = 0;

        // Нижняя сторона
        for (int i = 0; i < N_X - 1; i++, id++)
            kraevs[id] = new Kraev(
                SideKraev![0],
                0,
                i,
                i + 1
            );
```

```
        // Правая сторона
        for (int i = 0; i < N_Y - 1; i++, id++)
            kraevs[id] = new Kraev(
                 SideKraev![1],
                 1,
                 i     *N_X + (N_X - 1),
                (i + 1)*N_X + (N_X - 1)
            );

        // Верхняя сторона
        for (int i = 0; i < N_X - 1; i++, id++)
            kraevs[id] = new Kraev(
                SideKraev![2],
                2,
                N_X*(N_Y - 1) + (i + 1),
                N_X*(N_Y - 1) + i
            );

        // Левая сторона
        for (int i = 0; i < N_Y - 1; i++, id++)
            kraevs[id] = new Kraev(
                 SideKraev![3],
                 3,
                (i + 1)*N_X,
                 i     *N_X
            );

        File.WriteAllText(Path + "/kraevs.txt", String.Join("\n", kraevs));
        return kraevs;
    }

}
```

LOS.cs

```
namespace Practice;
public class LOS
{
    private SLAU slau;      /// Структура СЛАУ

    private int maxIter;    /// Максимальное количество итераций
    private double EPS;     /// Точность

// ************ Коструктор LOS ************ //
    public LOS(SLAU slau, int maxIter, double eps) {
        this.slau    = slau;
        this.maxIter = maxIter;
        this.EPS     = eps;
    }

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

        ComplexVector L = new ComplexVector(Enumerable.Range(0, slau.N).Select(i => new Complex(1, 0) / slau.di[i]).ToArray

        ComplexVector 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];
        }
        ComplexVector 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 = Norm(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}");
    }
}
```

FEM.cs

```
namespace Practice;
public class FEM
{
    private Node[]     Nodes;               /// Узлы
    private Elem[]     Elems;               /// КЭ
    private Kraev[]    Kraevs;              /// Краевые
    private SLAU       slau;                /// Структура СЛАУ
    private Matrix part_M;                  /// Неполная матрица масс (M)
    private Matrix part_G_left;             /// Неполная левая матрица жесткости (G)
    private Matrix part_G_right;            /// Неполная правая матрица жесткости (G)

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

    //* Конструктор
    public FEM(Grid grid, string path) {
        (Nodes, Elems, Kraevs) = grid;
        this.Path = path;

        part_G_left = new Matrix(new double[4, 4]{
            {2, -2, 1, -1},
```

```
            {-2, 2, -1, 1},
            {1, -1, 2, -2},
            {-1, 1, -2, 2}
        });

        part_G_right = new Matrix(new double[4, 4]{
            {2, 1, -2, -1},
            {1, 2, -1, -2},
            {-2, -1, 2, 1},
            {-1, -2, 1, 2}
        });

        part_M = new Matrix(new double[4, 4]{
            {4, 2, 2, 1},
            {2, 4, 1, 2},
            {2, 1, 4, 2},
            {1, 2, 2, 4}
        });
    }

    //* Основной метод решения
    public void solve() {
        portrait();                                             //? Составление портрета матрицы
        global();                                               //? Составление глобальной матрицы
        LOS los = new(slau, 10000, 9e-30);                      //? Создание метода LOS
        los.solve(true);                                        //? 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 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 ComplexVector(slau.N_el);
        slau.di         = new ComplexVector(slau.N);
        slau.f          = new ComplexVector(slau.N);
        slau.q          = new ComplexVector(slau.N);
        slau.q_absolut  = new ComplexVector(slau.N);
    }

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

        // Для каждого КЭ
        for (int index_fin_el = 0; index_fin_el < Elems.Length; index_fin_el++) {
            // Состовляем локальную матрицы и локальный вектор
            (ComplexMatrix loc_mat, ComplexVector local_f) = local(index_fin_el);

            // 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, cur_kraev.numSide);
            // 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 глоабальную матрицу
    private void EntryMatInGlobalMatrix(ComplexMatrix mat, int[] index) {
        for (int i = 0, h = 0; i < mat.Dim; i++) {
            int ibeg = index[i];
            for (int j = i + 1; j < mat.Dim; 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 глолбальный вектор
    private void EntryVecInGlobalMatrix(ComplexVector vec, int[] index) {
        for (int i = 0; i < vec.Length; i++)
            slau.f[index[i]] += vec[i];
    }

    //* Построение локальной матрицы и вектора
    private (ComplexMatrix, ComplexVector) local(int index_fin_el) {
        // Подсчет компонент
        double hx   = Nodes[Elems[index_fin_el].Node[1]].x - Nodes[Elems[index_fin_el].Node[0]].x;
        double hy   = Nodes[Elems[index_fin_el].Node[2]].y - Nodes[Elems[index_fin_el].Node[0]].y;

        ComplexVector local_f      = build_F(index_fin_el, hx, hy);    // Построение локальной правой части
        ComplexMatrix M            = build_M(index_fin_el, hx, hy);    // Построение матрицы массы (M)
        Matrix G                   = build_G(index_fin_el, hx, hy);    // Построение матрицы жесткости (G)
        ComplexMatrix local_matrix = G + (new Complex(0, 1))*M;

        return (local_matrix, local_f);
    }

    //* Построение вектора правой части
    private ComplexVector build_F(int index_fin_el, double hx, double hy) {
        // Подсчет коэффициента
        double coef = (hx * hy) / 36.0;

        // Вычисление f - на узлах КЭ
        var f = new ComplexVector(4);
        for (int i = 0; i < f.Length; i++) {
            Vector<double> vec = new Vector<double>(new double[]{ Nodes[Elems[index_fin_el].Node[i]].x,
                                                                  Nodes[Elems[index_fin_el].Node[i]].y});
            f[i] = F(vec);
        }

        // Вычисление локального вектора
        var local_f = part_M * (coef * f);

        return local_f;
    }

    //* Построение матрицы масс
    private ComplexMatrix build_M(int index_fin_el, double hx, double hy) {
```

```
        // Подсчет коэффициента
        Complex coef = (Function.gamma * hx * hy) / 36.0;

        // Матрица масс
        var M_matrix = coef * part_M;

        return M_matrix;
    }

    //* Построение матрицы жесткости
    private Matrix build_G(int index_fin_el, double hx, double hy) {
        // Подсчет коэффициентов
        double coef_left  = (lambda * hy) / (6 * hx);
        double coef_right = (lambda * hx) / (6 * hy);

        // Матрица жесткости
        var G_matrix = coef_left * part_G_left + coef_right * part_G_right;

        return G_matrix;
    }

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

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

        // 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;
                    }
            }
        }
    }

    //* Расчет погрешности и нормы решения
    private (ComplexVector, double) Norm(ComplexVector x_abs, ComplexVector x) {
        ComplexVector norm_arr = new ComplexVector(x.Length);

        for (int i = 0; i < x.Length; i++) {
            norm_arr[i] = x_abs[i] - x[i];
            norm_arr[i] = new Complex(Abs(norm_arr[i].Real), Abs(norm_arr[i].Imaginary));
        }
```

```
        return (norm_arr, Helper.Norm(norm_arr));
    }

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

    //* Запись глобальной матрицы и решения
    private void WriteMatrix() {
        Directory.CreateDirectory(Path + "/matrix");
        Directory.CreateDirectory(Path + "/output");
        File.WriteAllText(Path + "/matrix/ig.txt", String.Join("\n", slau.ig));
        File.WriteAllText(Path + "/matrix/jg.txt", String.Join("\n", slau.jg));
        File.WriteAllText(Path + "/matrix/di.txt", String.Join("\n", slau.di));
        File.WriteAllText(Path + "/matrix/gg.txt", String.Join("\n", slau.gg));
        File.WriteAllText(Path + "/matrix/pr.txt", String.Join("\n", slau.f));
        File.WriteAllText(Path + "/output/x.txt",  String.Join("\n", slau.q));
        File.WriteAllText(Path + "/output/x_absolut.txt", String.Join("\n", slau.q_absolut));
    }

    //* Запись таблички с погрешностью
    private void WriteTable() {
        (ComplexVector SubX, double norma) = Norm(slau.q_absolut, slau.q);

        StringBuilder table = new StringBuilder();
        string margin = String.Join("", Enumerable.Repeat("-", 35));

        table.Append(String.Join("", Enumerable.Repeat("-", 145)) + "\n");
        table.Append($"|X^{" ",-32} | X{" ",-32} | |X^ - X|{" ",-25} | ||X^ - X||  {" ",-21} |\n");
        table.Append($"|" + margin + "|" + margin + "|" + margin + "|" + margin + "|\n");

        for (int i = 0; i < Nodes.Length; i++) {
            table.Append($"|{String.Format("{0,35}", slau.q_absolut[i].ToString("E4"))}" +
                         $"|{String.Format("{0,35}", slau.q[i].ToString("E4"))}" +
                         $"|{String.Format("{0,35}", SubX[i].ToString("E6"))}|");
            if (Nodes.Length / 2 == i)
                table.Append($"{String.Format("{0,35}", norma.ToString("E6"))}|");
            else
                table.Append($"{String.Format("{0,35}", " ")}|");
            table.Append("\n");
        }
        table.Append(String.Join("", Enumerable.Repeat("-", 145)) + "\n");
        File.WriteAllText(Path + "/output/table.txt", table.ToString());
    }

}
```

Program.cs

```
try {
    // Проверка аргументов
    if (args.Length == 0) throw new ArgumentNullException("Not found arguments!");
    if (args[0] == "-help") {
        ShowHelp(); return;
    }

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

```
    // Проверка входных данных
    if (!data.Incorrect(out string mes)) throw new ArgumentException(mes);

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

    // Генерация сетки
    Generate generator = new Generate(data, Path.GetDirectoryName(args[1])!);
    generator.SetKraev(1, 1, 1, 1);
    Grid grid = generator.generate();

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