Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ
НЭТИ

Кафедре прикладной математики

Курсовой проект

по дисциплине «Численные методы»

Вариант 43

Факультет:	ПМИ
Группа:	ПМ-72
Студент:	Ким С.Е.
Преподаватель:	Патрушев И. И. Персова М. Г.

Новосибирск

1. Задание

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции биквадратичные на прямоугольниках. Краевые условия всех типов. Коэффициент диффузии λ разложить по билинейным базисным функциям. Матрицу СЛАУ генерировать в разряженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

2. Постановка задачи

Эллиптическая краевая задача для функции $\boldsymbol{u}$ определяется дифференциальным уравнением:

$$-div(\lambda gradu) + \gamma u = f \tag{1}$$

Заданным в некоторой области $\boldsymbol{\Omega}$ с границей $\boldsymbol{S = S1 \cup S2 \cup S3}$, и краевыми условиями:

$$u|_{s_1} = u_g \tag{2}$$

$$\lambda \frac{\partial u}{\partial n}\Big|_{s_2} = \theta \tag{3}$$

$$\lambda \frac{\partial u}{\partial n}\Big|_{s_3} + \beta\left(u|_{s_3} - u_\beta\right) = 0 \tag{4}$$

Для краевой задачи в декартовой системе координат уравнение представимо в виде:

$$-\frac{\partial}{\partial x}\left(\lambda \frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(\lambda \frac{\partial u}{\partial y}\right) + \gamma u = f \tag{5}$$

3. Теоретическая часть

3.1. Вариационная постановка

Выполним вариационную постановку методом Бубнова-Галёркина. Введём гильбертово пространство $\mathbb{H} = \boldsymbol{L_2(\Omega)}$. Тогда скалярное произведение и норма определены:

$$(f, g) = \int_{\Omega} f(\vec{r}) g(\vec{r}) d\,\Omega \tag{6}$$

$$\|f\|_{L_2} = \sqrt{(f, f)} \tag{7}$$

В общем виде постановка Бубнова-Галёркина для операторного уравнения $\boldsymbol{Au = f}$ представима в виде:

$$(Au - f, v) = 0, \forall v \in \mathbb{H}_0, \mathbb{H}_0 = \{v \in \mathbb{H}: v|_{s_1} = 0\} \quad (8)$$

Для уравнения постановка примет вид:

$$\int_\Omega -div(\lambda gradu)vd\Omega + \int_\Omega (\gamma u - f)vd\Omega = 0, \forall v \in \mathbb{H}_0 \quad (9)$$

Далее применяя формулу Грина получим выражение:

$$\int_\Omega \lambda gradu * gradvd\Omega - \int_S \lambda \frac{\partial u}{\partial n} vdS + \int_\Omega (\gamma u - f)vd\Omega = 0 \quad (10)$$

где $\boldsymbol{S = S_1 \cup S_2 \cup S_3}$.

Подставим в формулу (10) вторые и третьи краевые условия (3), (4):

$$\int_\Omega \lambda gradu * gradvd\Omega - \int_{S_1} \lambda \frac{\partial u}{\partial n} vdS - \int_{S_2} \theta vdS - \int_{S_3} \beta(u - u_\beta)vdS + \int_\Omega (\gamma u - f)vd\Omega = 0 \quad (11)$$

Так как первые краевые условия не определяют значение $\boldsymbol{\lambda \frac{\partial u}{\partial n}}$, слагаемое $\boldsymbol{\int_{S_1} \lambda \frac{\partial u}{\partial n} vdS}$ необходимо исключить из уравнения (11), взамен потребовав, чтобы пространство пробных функций содержало только функции, которые бы принимали бы нулевые значения на границе $\boldsymbol{S_1}$. Поэтому в качестве пространства пробных функций возьмем $\boldsymbol{v_0} \in \mathbb{H}_0^1$.

В пространстве $\mathbb{H}_0^1$ выделим конечномерное подпространство $\boldsymbol{V^h}$, которое определяется как линейное пространство, натянутое на базисные функции $\boldsymbol{\psi_i, i = \overline{1,n}}$.

Заменим функцию $\boldsymbol{u}$ аппроксимирующей ее функцией $\boldsymbol{u^h}$, а функцию $\boldsymbol{v_0}$ - функцией $\boldsymbol{v_0^h}$, то получим аппроксимацию уравнения Галеркина:

$$\int_\Omega \lambda gradu^h * gradv_0^h d\Omega + \int_\Omega \gamma u^h v_0^h d\Omega + \int_{S_3} \beta u^h v_0^h dS = \int_\Omega f v_0^h d\Omega + \int_{S_2} \theta v_0^h dS + \int_{S_3} \beta u_\beta v_0^h dS \quad (12)$$

Так как $\boldsymbol{v_0^h}$ представима в виде линейной комбинации:

$$v_0^h = \sum_{i \in N_0} q_i^v \psi_i \tag{13}$$

Уравнение (12) эквивалентно следующей системе уравнений:

$$\begin{gathered} \int_\Omega \lambda grad u^h * grad\psi_i d\Omega + \int_\Omega \gamma u^h \psi_i d\Omega + \int_{S_3} \beta u^h \psi_i dS \\ = \int_\Omega f\psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS \, , i \in N_0 \end{gathered} \tag{14}$$

Так же $\boldsymbol{u^h}$ представим в виде:

$$u^h = \sum_{j=1}^{n} q_j \psi_j \tag{15}$$

Причём $\boldsymbol{n - N_0}$ компонент вектора весов $\boldsymbol{q = (q_1, \dots, q_n)^T}$ могут быть фиксированы и определены из первого краевого условия $\boldsymbol{u^h|_{s_1} = u_g}$. Подставим разложение (15) в систему (14) и получим СЛАУ для компонент $\boldsymbol{q_j}$ вектора весов $\boldsymbol{q}$:

$$\begin{gathered} \sum_{j=1}^{n} \left(\int_\Omega \lambda grad\psi_j * grad\psi_i d\Omega + \int_\Omega \gamma \psi_j \psi_i d\Omega + \int_{S_3} \beta \psi_j \psi_i dS \right) q_j \\ = \int_\Omega f\psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS \, , i \in N_0 \end{gathered} \tag{16}$$

Таким образом СЛАУ может записана в виде $\boldsymbol{Aq = b}$, где:

$$\begin{gathered} A_{ij} = \begin{cases} \int_\Omega \lambda grad\psi_j * grad\psi_i d\Omega + \int_\Omega \gamma \psi_j \psi_i d\Omega + \int_{S_3} \beta \psi_j \psi_i dS \, , j = \overline{1,n} \\ \delta_{ij}, i \notin N_0, j = \overline{1,n} \end{cases} \\ b_i = \begin{cases} \int_\Omega f\psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS \, , i \in N_0 \\ u_g(x_i), i \notin N_0 \end{cases} \end{gathered} \tag{17}$$

3.2. Конечно элементная дискретизация

Чтобы реализовать МКЭ для данной задачи, разобьём область $\boldsymbol{\Omega}$ на прямоугольные конечные элементы $\boldsymbol{\Omega} = [\boldsymbol{x_p}, \boldsymbol{y_s}] \times [\boldsymbol{x_{p+1}}, \boldsymbol{y_{s+1}}]$.

Введем на них двумерные базисные биквадратичные функции:

$$\begin{array}{lll} \psi_1 = X_1(x)Y_1(y) & \psi_2 = X_1(x)Y_2(y) & \psi_3 = X_1(x)Y_3(y) \\ \psi_4 = X_2(x)Y_1(y) & \psi_5 = X_2(x)Y_2(y) & \psi_6 = X_2(x)Y_3(y) \\ \psi_7 = X_3(x)Y_1(y) & \psi_8 = X_3(x)Y_2(y) & \psi_9 = X_3(x)Y_3(y) \end{array} \quad (18)$$

Где $\boldsymbol{X}$ и $\boldsymbol{Y}$ – одномерные биквадратичные базисные функции.

$$\begin{aligned} X_i(x) &= \varphi_i(\xi), \text{где } \xi = \frac{x - x_p}{x_{p+1} - x_p} \\ Y_i(y) &= \varphi_i(\chi), \text{где } \chi = \frac{y - y_s}{y_{s+1} - y_s} \end{aligned} \quad (19)$$

Одномерные базисные функции получены из шаблонных базисных биквадратичных функций:

$$\begin{aligned} \varphi_1(\xi) &= 2(\xi - \frac{1}{2})(\xi - 1) \\ \varphi_2(\xi) &= -4\xi(\xi - 1) \\ \varphi_3(\xi) &= 2\xi(\xi - \frac{1}{2}) \end{aligned} \quad (20)$$

$$\begin{aligned} G_{ij} &= \int_{\Omega} \lambda grad\psi_j * grad\psi_i d\Omega \\ &= \sum_k \int_{\Omega_k} \lambda grad\psi_j * grad\psi_i d\Omega, i \in N_0 \end{aligned} \quad (21)$$

Введём матрицу массы:

$$M_{ij} = \int_{\Omega} \gamma\psi_j\psi_i d\Omega = \sum_k \int_{\Omega_k} \gamma\psi_j\psi_i d\Omega, i \in N_0 \quad (22)$$

$$M_{ij}^{s_3} = \int_{S_3} \beta\psi_j\psi_i dS = \sum_l \int_{S_3^l} \beta\psi_j\psi_i dS, i \in N_0 \quad (23)$$

Таким образом СЛАУ (17) будет иметь вид:

$$A_{ij} = \begin{cases} \sum_k \int_{\Omega_k} \lambda grad\psi_j * grad\psi_i d\Omega + \sum_k \int_{\Omega_k} \gamma\psi_j\psi_i d\Omega + \sum_l \int_{S_3^l} \beta\psi_j\psi_i dS, j = \overline{1,n} \\ \delta_{ij}, i \notin N_0, j = \overline{1,n} \end{cases}$$
$$b_i = \begin{cases} \sum_k \int_{\Omega_k} f\psi_i d\Omega + \sum_n \int_{S_2^n} \theta\psi_i dS + \sum_m \int_{S_3^m} \beta u_\beta \psi_i dS, i \in N_0 \\ u_g(x_i), i \notin N_0 \end{cases} \quad (24)$$

По заданию так же необходимо разложить $\boldsymbol{\lambda}$ по билинейным базисным функциям.

$$\begin{aligned} \mu_1 &= X_1(x)Y_1(y) \quad \mu_2 = X_1(x)Y_2(y) \\ \mu_3 &= X_2(x)Y_1(y) \quad \mu_4 = X_2(x)Y_2(y) \end{aligned} \quad (25)$$

Где $\boldsymbol{X}$ и $\boldsymbol{Y}$ – одномерные билинейные базисные функции.

$$\begin{aligned} X_i(x) &= \phi_i(\xi), \text{где } \xi = \frac{x - x_p}{x_{p+1} - x_p} \\ Y_i(y) &= \phi_i(\chi), \text{где } \chi = \frac{y - y_s}{y_{s+1} - y_s} \end{aligned} \quad (26)$$

Одномерные базисные функции получены из шаблонных базисных билинейных функций:

$$\begin{aligned} \phi_1(\xi) &= \xi - 1 \\ \phi_2(\xi) &= \xi \end{aligned} \quad (27)$$

После подстановки в СЛАУ (24) получаем:

$$A_{ij} = \begin{cases} \sum_k \int_{\Omega_k} \sum_{p=1}^{4} (\lambda_p\mu_p)\, grad\psi_j * grad\psi_i d\Omega + \sum_k \int_{\Omega_k} \gamma\psi_j\psi_i d\Omega + \sum_l \int_{S_3^l} \beta\psi_j\psi_i dS, j = \overline{1,n} \\ \delta_{ij}, i \notin N_0, j = \overline{1,n} \end{cases}$$
$$b_i = \begin{cases} \sum_k \int_{\Omega_k} f\psi_i d\Omega + \sum_n \int_{S_2^n} \theta\psi_i dS + \sum_m \int_{S_3^m} \beta u_\beta \psi_i dS, i \in N_0 \\ u_g(x_i), i \notin N_0 \end{cases} \quad (28)$$

3.3. Локальная матрица, локальные матрицы массы и жёсткости

Так как функция раскладывается по биквадратичному базису, локальная матрица массы и жёсткости имеет размерность $[9 \times 9]$.

Локальные массы и жёсткости можно получить из одномерных[1] матриц массы и жёсткости, следующим образом:

$$\eta(i) = (i-1) mod\ 3 + 1$$
$$\vartheta(i) = \left[\frac{i-1}{3}\right] + 1 \tag{29}$$

Так как одномерные биквадратичные и билинейные базисные шаблоны одинаковые по $\boldsymbol{x}$ и $\boldsymbol{y}$, локальную одномерную матрицу жёсткости можно выразить:

$$G_{ij}^{xy} = \int_0^1 \varphi_i \varphi_j \phi_1 d\xi\ , i = \overline{1,3}, j = \overline{1,3}$$
$$G^{xy} = \frac{1}{6}\begin{pmatrix} 11 & -12 & 1 \\ -12 & 16 & -4 \\ 1 & -4 & 3 \end{pmatrix} \tag{30}$$

$$M_{ij}^{xy} = \int_0^1 \phi_i \phi_j \varphi_1 d\xi\ , i = \overline{1,3}, j = \overline{1,3}$$
$$M^{xy} = \frac{1}{60}\begin{pmatrix} 7 & 4 & -1 \\ 4 & 16 & 0 \\ -1 & 0 & 1 \end{pmatrix} \tag{31}$$

Так же зададим матрицы:

$\boldsymbol{D^{xy}}$ – отображение матрицы $\boldsymbol{M^{xy}}$ относительно побочной диагонали;

$\boldsymbol{R^{xy}}$ – отображение матрицы $\boldsymbol{G^{xy}}$ относительно побочной диагонали;

$\boldsymbol{\lambda^{i,j}}$ – определяет значение параметра в угловых узлах конечного элемента

Тогда локальная двумерная матрица жёсткости будет выглядеть:

$$\begin{aligned} G_{ij}^{k} = \frac{1}{360}\Bigg(&\frac{h_y}{h_x}\Big[M_{\vartheta(i)\vartheta(j)}^{xy}\left(\lambda^{k,k} G_{\eta(i)\eta(j)}^{xy} + \lambda^{k+1,k} R_{\eta(i)\eta(j)}^{xy}\right) \\ &+ D_{\vartheta(i)\vartheta(j)}^{xy}\left(\lambda^{k,k+1} G_{\eta(i)\eta(j)}^{xy} + \lambda^{k+1,k+1} R_{\eta(i)\eta(j)}^{xy}\right)\Big] \\ &+ \frac{h_x}{h_y}\Big[M_{\eta(i)\eta(j)}^{xy}\left(\lambda^{k,k} G_{\vartheta(i)\vartheta(j)}^{xy} + \lambda^{k+1,k} R_{\vartheta(i)\vartheta(j)}^{xy}\right) \\ &+ D_{\eta(i)\eta(j)}^{xy}\left(\lambda^{k,k+1} G_{\vartheta(i)\vartheta(j)}^{xy} + \lambda^{k+1,k+1} R_{\vartheta(i)\vartheta(j)}^{xy}\right)\Big]\Bigg), \\ &i = \overline{1,9}, j = \overline{1,9} \end{aligned} \tag{32}$$

Матрица массы без учёта $\boldsymbol{\lambda}$ имеет вид:

[1] Для удобства представления в программе одномерные матрицы массы и жёсткости представлены в виде двумерных матриц $[3 \times 3]$, где строка и столбец это разложения по $\boldsymbol{X_i}$ и $\boldsymbol{Y_j}$ соответственно.

$$M_{ij}^{xy} = \int_0^1 \phi_i \phi_j d\xi \, , i = \overline{1,3}, j = \overline{1,3}$$
$$M^{xy} = \frac{1}{30}\begin{pmatrix} 7 & 4 & -1 \\ 4 & 16 & 0 \\ -1 & 0 & 1 \end{pmatrix} \quad (33)$$

Тогда, локальная двумерная матрица массы представима в виде:

$$M_{ij} = \gamma h_x h_y M_{\eta(i)\eta(j)}^{xy} M_{\vartheta(i)\vartheta(j)}^{xy}, i = \overline{1,9}, j = \overline{1,9}$$
$$M = \frac{1}{30}\begin{pmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{pmatrix} \quad (34)$$

И в итоге локальная матрица будет иметь вид:

$$A_{ij}^k = G_{ij}^k + \gamma M_{ij}, i = \overline{1,9}, j = \overline{1,9} \quad (35)$$

А локальная правая часть:

$$\hat{b}_i^k = f_i^k \psi_i, i = \overline{1,9} \quad (36)$$

Таким образом глобальная матрица будет иметь вид, без учёта краевых:

$$A_{ij} = \begin{cases} \sum_k A_{i_k j_k}^k \, , j = \overline{1,n} \\ \delta_{ij}, i \notin N_0, j = \overline{1,n} \end{cases}$$
$$b_i = \begin{cases} \sum_k \hat{b}_{i_k}^k = \sum_k f_{i_k}^k \psi_{i_k} \, , i \in N_0 \\ u_g(x_i), i \notin N_0 \end{cases} \quad (37)$$

3.4. Учёт краевых условий

3.4.1. Учёт первых краевых условий

Так как первое краевое условие фактически означает, что мы знаем значение функции в узле. По этой причине, чтобы не терять симметричность матрицы мы, поставим большое число на диагональный элемент, соответствующий этому узлу, и в правой части поставим значение первого краевого умноженное на это же большое число.

3.4.2. Учёт вторых краевых условий

Рассмотрим краевые условия второго рода (3):

$$\lambda \frac{\partial u}{\partial n}\Big|_{s_2} = \theta$$

Для учёта вторых условий вычислим интеграл:

$$\int_{S_2} \theta \psi_j dxdy \quad (38)$$

Р

а

з

Г

д

И

м

$$\theta = \theta_1 \psi_1 + \theta_2 \psi_2 + \theta_3 \psi_3 \quad (39)$$

И учёт вторых краевых будет представлен в виде:

$\boldsymbol{\theta i}$ значение функции в точке, где соответствующая базисная функция принимает **1**.

$$b^{S_2} = h \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} M \quad (40)$$

3.4.3. Учёт третьих краевых условий

Рассмотрим краевые условия третьего рода (4):

$$\boldsymbol{\lambda \left. \frac{\partial u}{\partial n} \right|_{s_3} + \beta \left(u|_{s_3} - u_\beta \right) = 0}$$

Тогда для учета третьего краевого условия нужно вычислить интегралы:

$$\int_{S_3} \beta u_\beta \psi_j dxdy$$
$$\int_{S_3} \beta \psi_i \psi_j dxdy \quad (41)$$

Б

у

$$u_\beta = u_{\beta 1} \psi_1 + u_{\beta 2} \psi_2 + u_{\beta 3} \psi_3 \quad (42)$$

К

с

М

с

ч

и

Который будем раскладывать по двум базисным функциям, определённым на этом ребре.

Тогда добавления к глобальной матрице будут представлены в виде:

$$b^{S_3} = h\beta \begin{pmatrix} u_{\beta 1} \\ u_{\beta 2} \\ u_{\beta 3} \end{pmatrix} M$$
$$A^{S_3} = h\beta M \quad (43)$$

3.5. Алгоритм формирования глобальной матрицы

Глобальную матрицу по заданию необходимо хранить в разряженном формате. Она состоит из элементов локальной матрицы.

Д

л

Н

Необходимо сформировать список смежности узлов друг с другом, с учетом того что портрет матрицы будет симметричным.

Д

ч

т

о

б

а

п

равен количеству элементов в списке смежности для $i-$узла и количеству элементов в списке смежности находящихся до $i-$узла; в массив jg заносятся все элементы списка смежности.

Когда портрет сформировали, можно заносить локальные элементы в соответствии с глобальной нумерацией. Краевые условия задаются списком узлов для 1 краевых, списком узлов и номером функции для 2 краевых, и для третьих списком узлов, номером функции и значением коэффициента.

Полученную систему линейных алгебраических уравнений будем решать методом ЛОС.

Сетка храниться в виде двух списков, первый список узлов конечного элемента в глобальной нумерации, второй координаты вершин, в порядке слева на право, снизу-вверх.

4. Описание разработанной программы

4.1. Структура данных, используемая для задания расчётной области, и конечной элементной сетки

Для задания расчетной области и конечной элементной сетки в программе используются следующие структуры:

1) Описания локальной области, содержит номера узлов и номер области

```
struct numberNode
{
   int x[9];
   int num_sqr;
};
```

2) Координаты узлов

```
struct point
{
   double x;
   double y;
   point();
};
```

3) Структура первых краевых условий

```
struct firstBoundCondition
{
   int el;
   double value;
};
```

4) Структура вторых краевых условий

```
struct secondBoundCondition
{
   int nodes[3];
   int numFunction;
};
```

5) Структура третих краевых условий

```
struct thirdBoundCondition
{
   int nodes[3];
```

```
    int numFunction;
    double b;
};
```

6) Узлы хранятся в массиве xy со структурой point
7) Хранение конечных элементов происходит в массиве nvtr типа

8) Первые краевые условия хранятся в массиве nvk1, с типом

9) Вторые краевые условия хранятся в массиве nvk2, с типом

10) П

Для хранения, вывода результатов и внешнего ввода дробления сетки, расчётной области, конечной элементной сетки, и параметров для МКЭ используются файлы:

Имя файла	Предназначение	Способ задания
x.txt	Формирование сетки по оси $\boldsymbol{x}$	Количество значений $\boldsymbol{n}$ Перечисление $\boldsymbol{x_i, i = \overline{1,n}}$
y.txt	Формирование сетки по оси $\boldsymbol{y}$	Количество значений $\boldsymbol{m}$ Перечисление $\boldsymbol{y_i, i = \overline{1,m}}$
hxhy.txt	Дробление сетки, по итогу дробления получаться файлы **xy.txt, n**	В о В д и
xy.txt	Узлы сетки	Первая строка – число узлов, последующие – их координаты.
	Конечные элементы	Первая строка – число конечных элементов, далее номера узлов, номер области.
nvk.txt	Первые краевые	Номер узла, на котором задано условие и значение краевого условия в этом узле.

nvk1.txt	Вторые краевые	Три номера узла, задающих грань, и номер функции
nvk2.txt	Третьи краевые	Три номера узла, задающих грань, номер функции , значение
options.txt	Настройки для ЛОС	Количество итераций Точность решения
out.txt	Результат работы программы	Выводятся координаты узла и значение в этом узле

4.2. Структура основных модулей программы

1) FEM - Основной модуль программы, содержит описание и реализацию основного класса программы, основные и вспомогательные типы данных для представления расчетной области и конечной элементной сетки, все необходимые данные, такие как локальные матрицы.

- процедура создания класса, чтение всех необходимых данных

процедура сборки глобальной матрицы, учета краевых условий, решение СЛАУ

процедура вывода результата решения

процедура создания портрета

1.5) d o u - процедура выполняющая инициализацию одномерных локальных матриц жесткости и массы

1.7)

1.8)

1.9)

1.10)

1.11)

1.12)

1.13)

функция вычисляющая значение параметра третьего краевого условия в точке
процедура расчета локального вектора правой части
процедура расчета локальной двумерной матрицы жесткости и массы для данного элемента

1.17) d
o
1.18) u
b - процедуры учета краевых условий первого рода
l - процедуры учета краевых условий второго рода
d - процедуры учета краевых условий третьего рода
матрица для хранения локальной матрицы жесткости
матрица для хранения локальной матрицы массы
матрица для хранения локальной одномерной матрицы жесткости
матрица для хранения локальной одномерной матрицы массы
матрица для хранения локальной одномерной матрицы массы
m
локальный вектор правой части функции
1.28) double bLocal[9]; - локальный вектор правой части функции разложенной по базисным функциям
массивы для хранения матрицы в разряженном формате
- вектор правой части
1.32) int size_gg; - количество элементов матрицы в массиве ggl или ggu
количество первых краевых условий
количество вторых краевых условий
количество третьих краевых условий
количество узлов
1.37) процедура добавления к глобальной матрице в (i,j) элемент d

2) LosSolver - модуль программы, обеспечивающий решение СЛАУ в разряженном строчно-столбцовом формате методом локально оптимальной схемы с факторизованным элементом
   2.1) L
   2.2) O
   2.3) S
   2.4) double norm(vectorD a); - норма вектора
   2.5) M

3) fragmentationGrid - модуль программы, обеспечивающий генерацию сетки

количество элементов на сколько нужно разбить промежутки

3.2) i

3.3) n

3.4) h

3.5) w

3.6) double hx; double hy; переменная для хранения шага между промежутками

3.7) d

3.8) d

3.9) d

3.10) d

3.11) d

3.12) d

3.13) y

5. Тесты

Сетка для равномерных тестов имеет вид:

4	20	21	22	23	24
3	15	16	17	18	19
2	10	11	12	13	14
1	5	6	7	8	9
0	0	1	2	3	4
y / x	0	1	2	3	4

Сетка для неравномерных тестов имеет вид:

6	10	11	12	13	14
3	5	6	7	8	9
0	0	1	2	3	4
y / x	0	2	3	5	6

Формула для расчёта относительной нормы:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}\left(q_i^h - q^*\right)^2}{\sum_{i=1}^{n}(q_i^*)^2}} \qquad (44)$$

1) $u = x + y, \gamma = 1, \lambda = 1$

$u_ч$

4	4	5	6	7	8
3	3	4	5	6	7
2	2	3	4	5	6

- аналитическая функция в точке, для генерации первых краевых условий на границах

- вертикальные асимптоты по $\boldsymbol{x}$

- вертикальные асимптоты по $\boldsymbol{y}$

- переменная в которой храниться итоговое количество получившихся узлов по оси $\boldsymbol{x}$

переменная в которой храниться итоговое количество получившихся узлов по оси $\boldsymbol{y}$

вспомогательный массив для формирования сетки по оси $\boldsymbol{y}$

; все точки разбиения по $\boldsymbol{x}$

все точки разбиения по $\boldsymbol{y}$

функция необходимая для расчета глобальной нумерации

процедура считывания входных данных

1	1	2	3	4	5
0	0	1	2	3	4
	0	1	2	3	4

u^*

4	4	5	6	7	8
3	3	4	5	6	7
2	2	3	4	5	6
1	1	2	3	4	5
0	0	1	2	3	4
	0	1	2	3	4

$|u_ч - u^*|$

4	0	0	0	0	0
3	0	0	0	0	0
2	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

Относительная норма: 0.0

Комментарий: Так как данное уравнение полностью представимо в базисе, то погрешность практически отсутствует. (выводилось 12 знаков в экспоненциальном формате)

2) $u = x + y, \gamma = 1, \lambda = x + y$

$u_ч$

4	4	5	6	7	8
3	3	4	5	6	7
2	2	3	4	5	6
1	1	2	3	4	5
0	0	1	2	3	4
	0	1	2	3	4

u^*

4	4	5	6	7	8
3	3	4	5	6	7
2	2	3	4	5	6
1	1	2	3	4	5
0	0	1	2	3	4
	0	1	2	3	4

$|u_ч - u^*|$

4	0	0	0	0	0
3	0	0	0	0	0
2	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

Относительная норма: 0.0

К

3) $u = x^2 + y^2, \gamma = 1, \lambda = 1$

м $u_ч$

4	16	17	20	25	32
3	9	10	13	18	25
2	4	5	8	13	20

1	1	2	5	10	17
0	0	1	4	9	16
	0	1	2	3	4

u^*

4	16	17	20	25	32
3	9	10	13	18	25
2	4	5	8	13	20
1	1	2	5	10	17
0	0	1	4	9	16
	0	1	2	3	4

$|u_ч - u^*|$

4	0	0	0	0	0
3	0	0	0	0	0
2	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

Относительная норма: 0.0

Комментарий: Аналогично тесту 1.

4) $u = xy^3, \gamma = 1, \lambda = x + y$

$u_ч$

4	0	64	128	192	256
3	0	27.01070549	54.01252712	81.0109238	108
2	0	8.009361632	16.01171184	24.01083645	32
1	0	1.007259581	2.008606122	3.008564323	4
0	0	0	0	0	0
	0	1	2	3	4

u^*

4	0	64	128	192	256
3	0	27	54	81	108
2	0	8	16	24	32
1	0	1	2	3	4
0	0	0	0	0	0
	0	1	2	3	4

$|u_ч - u^*|$

4	0E+00	0E+00	0E+00	0E+00	0E+00
3	0E+00	1E-02	1E-02	1E-02	0E+00
2	0E+00	9E-03	1E-02	1E-02	0E+00
1	0E+00	7E-03	9E-03	9E-03	0E+00
0	0E+00	0E+00	0E+00	0E+00	0E+00
	0	1	2	3	4

Относительная норма: 8.0E-05

Комментарий: Уравнение не представимо базисными функциями, поэтому решение дает приблизительный результат (аппроксимированная функция).

5) $u = x^2 + y^2, \gamma = 1, \lambda = 1$

$u_ч$

6	36	40	45	61	72
3	9	13	18	34	45

u^*

6	36	40	45	61	72
3	9	13	18	34	45
0	0	4	9	25	36
	0	2	3	5	6

0	0	4	9	25	36
	0	2	3	5	6

$|u_ч - u^*|$

6	0	0	0	0	0
3	0	0	0	0	0
0	0	0	0	0	0
	0	2	3	5	6

Относительная норма: 0.0

Комментарий: Аналогично тесту 1. Только на неравномерной сетке.

6) $u = x, \gamma = 1, \lambda = x$

$u_ч$

6	0	2	3	5	6
3	0	2	3	5	6
0	0	2	3	5	6
	0	2	3	5	6

u^*

6	0	2	3	5	6
3	0	2	3	5	6
0	0	2	3	5	6
	0	2	3	5	6

$|u_ч - u^*|$

6	0	0	0	0	0
3	0	0	0	0	0
0	0	0	0	0	0
	0	2	3	5	6

Относительная норма: 0.0

Комментарий: Аналогично тесту 2. Только на неравномерной сетке.

7) $u = x, \gamma = 1, \lambda = x^2$

$u_ч$

6	0	2	3	5	6
3	0	1.950951412	3.004707771	4.99018826	6
0	0	2	3	5	6
	0	2	3	5	6

u^*

6	0	2	3	5	6
3	0	2	3	5	6

0	0	2	3	5	6
	0	2	3	5	6

$|u_ч - u^*|$

6	0E+00	0E+00	0E+00	0E+00	0E+00
3	0E+00	5E-02	5E-03	1E-02	0E+00
0	0E+00	0E+00	0E+00	0E+00	0E+00
	0	2	3	5	6

Относительная норма: 0.0033

Комментарий Конечно-разностная схема

8) $u = x/y, \gamma = 1, \lambda = 1$, с учётом краевых условий (только 1 − ые)

u^*

4	0	0.5	0.75	1.25	1.5
3	0	0.666666667	1	1.666666667	2
1	0	2	3	5	6
	0	2	3	5	6

Решение

$u_ч$

4	0	0.5	0.75	1.25	1.5
3	0	0.67018938	1.005179574	1.672676283	2
1	0	2	3	5	6
	0	2	3	5	6

Дифференциальной

$|u_ч - u^*|$

4	0E+00	0E+00	0E+00	0E+00	0E+00
3	0E+00	4E-03	5E-03	6E-03	0E+00
1	0E+00	0E+00	0E+00	0E+00	0E+00
	0	2	3	5	6

уравнении

Относительная норма: 0.00093

не представим в билинейном базисе, поэтому решение не точное (аппроксимированная функция).

Комментарий: Была взята функция не представимая в биквадратичном базисе, для анализа влияния краевых условий на точность решения.

С учётом краевых условий (1 − ых и 2 − ых краевых условий)

$u_ч$

4	0.00	0.50	0.75	1.25	1.50
3	0.00	0.68	1.00	1.55	2.00
1	0.00	2.00	3.00	5.00	6.00
	0	2	3	5	6

$|u_ч - u^*|$

4	3E-17	0E+00	0E+00	0E+00	0E+00
3	7E-17	1E-02	1E-03	1E-01	0E+00
1	2E-17	0E+00	0E+00	0E+00	0E+00
	0	2	3	5	6

Относительная норма: 0.0125

Комментарий: 2 краевые условия были добавлены на верхней и нижней грани области, при этом первые краевые на этих гранях были удалены.

С учётом краевых условий (1 − ых и 3 − ых краевых условий)

$u_ч$

4	0.00	0.50	0.75	1.25	1.50
3	0.00	0.68	1.01	1.68	2.00
1	0.00	2.00	3.00	5.00	6.00
	0	2	3	5	6

$|u_ч - u^*|$

4	3E-17	0E+00	0E+00	0E+00	0E+00
3	8E-17	1E-02	8E-03	9E-03	0E+00
1	2E-17	0E+00	0E+00	0E+00	0E+00
	0	2	3	5	6

Относительная норма: 0.0019

Комментарий: 2 краевые условия были заменены 3-ми краевыми условиями.

Вывод: Третьи краевые условия дают результат точнее, чем вторые краевые, и приблизительные по точности к первым. Однако первые являются самыми точными.

Следующие тесты будут на дробление сетки. Дробление сетки происходило путём дробления области на 4 подобласти.

9) $u = {}^{x}/_{y}, \gamma = 1, \lambda = 1$

u^*

6	0.333333333	0.666666667	1	1.333333333	1.666666667
4	0.5	1	1.5	2	2.5
2	1	2	3	4	5
	2	4	6	8	10

$u_ч$

6	0.33	0.67	1.00	1.33	1.67
4	0.50	1.01	1.51	2.01	2.50
2	1.00	2.00	3.00	4.00	5.00
	2	4	6	8	10

$|u_ч - u^*|$

6	3E-08	3E-08	0E+00	3E-07	3E-07
4	0E+00	6E-03	1E-02	1E-02	0E+00
2	0E+00	0E+00	0E+00	0E+00	0E+00
	2	4	6	8	10

Относительная норма: 0.00194

Комментарий: Для исследования уменьшения погрешности путём дробления сетки, была выбрана равномерная сетка и функция не представимая биквадратичным базисом.

$u_ч$

6	0.33	0.67	1.00	1.33	1.67
4	0.50	1.00	1.50	2.00	2.50
2	1.00	2.00	3.00	4.00	5.00
	2	4	6	8	10

$|u_ч - u^*|$

6	3E-08	3E-08	0E+00	3E-07	3E-07
4	0E+00	4E-04	6E-04	8E-04	0E+00
2	0E+00	0E+00	0E+00	0E+00	0E+00
	2	4	6	8	10

Относительная норма: 0.000128

Комментарий: Относительная норма уменьшилась в ~ 15.24 раза, при первом дроблении.

$u_ч$

6	0.33	0.67	1.00	1.33	1.67
4	0.50	1.00	1.50	2.00	2.50
2	1.00	2.00	3.00	4.00	5.00
	2	4	6	8	10

$|u_ч - u^*|$

6	3E-08	3E-08	0E+00	3E-07	3E-07
4	0E+00	3E-05	4E-05	5E-05	0E+00
2	0E+00	0E+00	0E+00	0E+00	0E+00
	2	4	6	8	10

Относительная норма: 8.6E-06

Комментарий: Относительная норма уменьшилась в ~ 14.85 раза, при первом дроблении.

Вывод: Погрешность при дроблении сетки падает приблизительно в 16 раз.

6. Приложение

Код программы:

Fem.h

```
#pragma once
#include <string>
#include <vector>
#include "LOSSolver.h"
using namespace std;
struct numberNode
{
	int x[9];
	int num_sqr;
};
struct point
{
	double x;
	double y;
	point();
};
struct firstBoundCondition
{
	int el;
	double value;
};
struct secondBoundCondition
{
	int nodes[3];
	int numFunction;
};
```

```
struct thirdBoundCondition
{
	int nodes[3];
	int numFunction;
	double b;
};
class FEM
{
private:
	static const int countBasicFuncs = 9;
	void initMatrix();
	double basicFunc(double x, double xpStart, double xpMid, double xpEnd, int
num);
	void initGM();
	int globalIndex(int elem, int num);
	int getNumberElement(double x, double y);
	double lambda(int elem, double x, double y);
	double gamma(int elem, double x, double y);
	double function(double x, double y);
	double funcsKr2(int number, double x, double y);
	double funcsKr3(int number, double x, double y);
	void calcFLocal(int elem);
	void calcGM(int elem);
	double bisquar(int def_elem, int number, double x, double y);
	void addElem(int i, int j, double a);
	void ukr1(int number);
	void ukr2(int number);
	void ukr3(int number);
	numberNode* nvtr;
	point* xy;
	firstBoundCondition* nvk1;
	secondBoundCondition* nvk2;
	thirdBoundCondition* nvk3;
	double GLocal[9][9];
	double MLocal[9][9];
	double Gxy[3][3];
	double M[3][3];
	double Mxy[3][3];
	double fLocal[9];
	double bLocal[9];
	int* ig, * jg;
	double* ggl, * ggu, * di;
	int size_gg;
	int nk1;
	int nk2;
	int nk3;
	double* b;
	int nuz, nel;
public:
	FEM(string direct);
	void solveSystem();
	double solveInPoint(double x, double y);
	void output(string dir);
	~FEM();
};
```

Fem.cpp

```
#include "FEM.h"
point::point()
{
	x = 0;
	y = 0;
}
FEM::FEM(string direct)
{
```

```
	string dir = direct + "/";
	FILE* f;
	myfopen(&f, dir + "xy.txt", "r");
	fscanf_s(f, "%d", &nuz);
	xy = new point[nuz];
	for (int i = 0; i < nuz; ++i)
	{
		fscanf_s(f, "%lf%lf", &xy[i].x, &xy[i].y);
	}
	fclose(f);
	myfopen(&f, dir + "nvtr.txt", "r");
	fscanf_s(f, "%d", &nel);
	nvtr = new numberNode[nel];
	for (int i = 0; i < nel; ++i)
	{
		for (int j = 0; j < 9; ++j) {
			fscanf_s(f, "%d", &nvtr[i].x[j]);
		}
	}
	fclose(f);
	myfopen(&f, dir + "nvk.txt", "r");
	fscanf_s(f, "%d", &nk1);
	nvk1 = new firstBoundCondition[nk1];
	for (int i = 0; i < nk1; ++i)
		fscanf_s(f, "%d%lf", &nvk1[i].el, &nvk1[i].value);
	fclose(f);
	myfopen(&f, dir + "nvk1.txt", "r");
	fscanf_s(f, "%d", &nk2);
	nvk2 = new secondBoundCondition[nk2];
	for (int i = 0; i < nk2; ++i)
	{
		fscanf_s(f, "%d%d%d%d", &nvk2[i].nodes[0], &nvk2[i].nodes[1],
&nvk2[i].nodes[2], &nvk2[i].numFunction);
	}
	fclose(f);
	myfopen(&f, dir + "nvk2.txt", "r");
	fscanf_s(f, "%d", &nk3);
	nvk3 = new thirdBoundCondition[nk3];
	for (int i = 0; i < nk3; ++i)
		fscanf_s(f, "%d%d%d%d%lf", &nvk3[i].nodes[0], &nvk3[i].nodes[1],
&nvk3[i].nodes[2], &nvk3[i].numFunction, &nvk3[i].b);
	fclose(f);
	b = new double[nuz];
	for (int i = 0; i < nuz; ++i)
		b[i] = 0;
	initMatrix();
	initGM();
}

void FEM::solveSystem()
{
	double g = gamma(0, 0, 0);
	for (int i = 0; i < nel; ++i)
	{
		calcGM(i);
		for (int k = 0; k < 9; ++k)
		{
			for (int j = 0; j < 9; ++j)
			{
				addElem(nvtr[i].x[k], nvtr[i].x[j], GLocal[k][j] + g *
MLocal[k][j]);
			}
		}
		calcFLocal(i);
```

```
                for (int k = 0; k < 9; ++k)
                {
                        b[nvtr[i].x[k]] += bLocal[k];
                }
        }
        for (int i = 0; i < nk2; ++i)
        {
                ukr2(i);
        }
        for (int i = 0; i < nk3; ++i)
        {
                ukr3(i);
        }
        for (int i = 0; i < nk1; ++i)
        {
                ukr1(i);
        }
        LOS solver(nuz, ig, jg, di, ggu, ggl, b);
        solver.solve();
}

void FEM::output(string dir)
{
        FILE* f;
        myfopen(&f, dir + "/out.txt", "w");
        for (int i = 0; i < nuz; ++i) {
                fprintf(f, "%lf %lf %.12le\n", xy[i].x, xy[i].y, b[i]);
        }
        fclose(f);
}

FEM::~FEM()
{
        delete[]xy;
        delete[]nvtr;
        delete[]nvk1;
        delete[]nvk2;
        delete[]nvk3;
        delete[]b;
        delete[]ig;
        delete[]jg;
        delete[]ggl;
        delete[]ggu;
        delete[]di;
}

void FEM::initMatrix()
{
        di = new double[nuz];
        for (int i = 0; i < nuz; ++i)
        {
                di[i] = 0;
        }
        ig = new int[nuz + 1];
        vector<vector<int>> list(nuz);
        int listsize = 0;
        for (int elem = 0; elem < nel; elem++)
        {
                for (int i = 0; i < countBasicFuncs; ++i) {
                        int k = nvtr[elem].x[i];
                        for (int j = i + 1; j < countBasicFuncs; j++) {
                                int ind1 = k;
                                int ind2 = nvtr[elem].x[j];
                                if (ind2 < ind1) {
```

```
                                        ind1 = ind2;
                                        ind2 = k;
                                }
                                int iaddr = list[ind2].size();
                                if (iaddr == 0) {
                                        listsize++;
                                        list[ind2].push_back(ind1);
                                }
                                else
                                {
                                        int l = 0;
                                        for (l = 0; l < iaddr && list[ind2][l] < ind1;
l++);
                                        if (l == iaddr)
                                        {
                                                listsize++;
                                                list[ind2].push_back(ind1);
                                        }
                                        else
                                        {
                                                if (list[ind2][l] > ind1) {
                                                        listsize++;
                                                        list[ind2].emplace(list[ind2].begin()
+ l, ind1);
                                                }
                                        }
                                }
                        }
                }
        }
        jg = new int[listsize];
        ig[0] = 0;
        size_gg = listsize;
        int size_sum = 0;
        for (int i = 0; i < nuz; ++i)
        {
                int size = list[i].size();
                ig[i + 1] = ig[i] + list[i].size();
                for (int j = 0; j < size; j++)
                        jg[size_sum + j] = list[i][j];
                size_sum += size;
        }
        ggl = new double[listsize];
        ggu = new double[listsize];
        for (int i = 0; i < listsize; ++i)
        {
                ggl[i] = 0;
                ggu[i] = 0;
        }
        list.~vector();
}

double FEM::funcsKr2(int numFunc, double x, double y)
{
        switch (numFunc)
        {
        case 1: {return x / (y * y); }
        case 2: {return -x / (y * y); }
        }
}

double FEM::funcsKr3(int numFunc, double x, double y)
{
        switch (numFunc)
```

```
        {
        case 1: {return x / y - x / (y * y); }
        case 2: {return x / y + x / (y * y); }
        }
}

void FEM::ukr1(int number)
{
        int elem = nvk1[number].el;
        di[elem] = 1e+16;
        b[elem] = nvk1[number].value * 1e+16;
}

void FEM::ukr2(int number)
{
        int* uzel = nvk2[number].nodes;
        double h = xy[uzel[2]].y - xy[uzel[0]].y;
        if (h == 0)
        {
                h = xy[uzel[2]].x - xy[uzel[0]].x;
        }
        double buff[3];
        for (int i = 0; i < 3; ++i) {
                buff[i] = 0;
                for (int j = 0; j < 3; j++)
                        buff[i] += funcsKr2(nvk2[number].numFunction, xy[uzel[j]].x,
xy[uzel[j]].y) * Mxy[i][j];
                b[uzel[i]] += buff[i] * h / 30.;
        }
}

void FEM::ukr3(int number)
{
        int* uzel = nvk3[number].nodes;
        double h = xy[uzel[2]].y - xy[uzel[0]].y;
        double beta = nvk3[number].b;
        if (h == 0) h = xy[uzel[2]].x - xy[uzel[0]].x;
        double buff[3];
        for (int i = 0; i < 3; ++i) {
                buff[i] = 0;
                for (int j = 0; j < 3; j++)
                {
                        buff[i] += funcsKr3(nvk3[number].numFunction, xy[uzel[j]].x,
xy[uzel[j]].y) * Mxy[i][j];
                }
                b[uzel[i]] += buff[i] * h * beta / 30.;
        }
        for (int k = 0; k < 3; k++)
        {
                for (int j = 0; j < 3; j++)
                {
                        addElem(nvk3[number].nodes[k], nvk3[number].nodes[j], Mxy[k][j] *
beta * h / 30.0);
                }
        }
    }

void FEM::initGM()
{
        Gxy[0][0] =  11.; Gxy[0][1] = -12.; Gxy[0][2] =  1.;
        Gxy[1][0] = -12.; Gxy[1][1] =  16.; Gxy[1][2] = -4.;
        Gxy[2][0] =   1.; Gxy[2][1] =  -4.; Gxy[2][2] =  3.;

        Mxy[0][0] =  7.; Mxy[0][1] =  4.; Mxy[0][2] = -1.;
```

```
	Mxy[1][0] =  4.; Mxy[1][1] = 16.; Mxy[1][2] =   0;
	Mxy[2][0] = -1.; Mxy[2][1] =  0.; Mxy[2][2] =  1.;

	M[0][0] =  4.; M[0][1] =  2.; M[0][2] = -1.;
	M[1][0] =  2.; M[1][1] = 16.; M[1][2] =  2.;
	M[2][0] = -1.; M[2][1] =  2.; M[2][2] =  4.;
}

double FEM::basicFunc(double x, double xpStart, double xpMid, double xpEnd, int num)
{
	double hx1 = xpMid - xpStart;
	double hx2 = xpEnd - xpStart;
	double hx3 = xpEnd - xpMid;
	switch (num)
	{
	case 0: {return((xpMid - x) * (xpEnd - x)) / (hx1 * hx2); }
	case 1: {return((x - xpStart) * (xpEnd - x)) / (hx1 * hx3); }
	case 2: {return((x - xpStart) * (x - xpMid)) / (hx2 * hx3); }
	}
}

int FEM::globalIndex(int elem, int num)
{
	return nvtr[elem].x[num];
}

double FEM::solveInPoint(double x, double y)
{
	double u = 0;
	int numElem = getNumberElement(x, y);
	for (int i = 0; i < countBasicFuncs; ++i) {
		u += b[globalIndex(numElem, i)] * bisquar(numElem, i, x, y);
	}
	return u;
}

int FEM::getNumberElement(double x, double y)
{
	for (int i = 0; i < nel; ++i)
	{
		int x0 = globalIndex(i, 0), x8 = globalIndex(i, 8);
		if (xy[x0].x <= x && xy[x0].y <= y && xy[x8].x >= x && xy[x8].y >= y)
			return i;
	}
	return -1;
}

double FEM::lambda(int elem, double x, double y)
{
	return 1;
	//return x + y;
	//return x * x;
}

double FEM::gamma(int elem, double x, double y)
{
	return 1;
}

double FEM::function(double x, double y)
{
	//TEST1
	//return x + y;
	//return x + y - 2;
```

```
	//TEST2
	//return x * x + y * y - 4;
	//return x * x + y * y - 4 * x - 2 * y;
	//TEST3
	//return -y * y * y - 6 * x * x * y - 9 * y * y * x + x * y * y * y;
	//TEST4
	//return -x;
	//TEST5
	return x / y - 2 * x / (y * y * y);
}

void FEM::calcFLocal(int elem)
{
	int node;
	for (int i = 0; i < countBasicFuncs; ++i)
	{
		node = globalIndex(elem , i);
		fLocal[i] = function(xy[node].x, xy[node].y);
	}
	for (int i = 0; i < countBasicFuncs; ++i)
	{
		bLocal[i] = 0;
		for (int j = 0; j < countBasicFuncs; j++)
		{
			bLocal[i] += MLocal[i][j] * fLocal[j];
		}
	}
}

void FEM::calcGM(int elem)
{
	int xy0 = globalIndex(elem, 0);
	int xy2 = globalIndex(elem, 2);
	int xy6 = globalIndex(elem, 6);
	int xy8 = globalIndex(elem, 8);
	double hx = xy[xy8].x - xy[xy0].x;
	double hy = xy[xy8].y - xy[xy0].y;
	double lk0k0 = lambda(elem, xy[xy0].x, xy[xy0].y);
	double lk1k0 = lambda(elem, xy[xy2].x, xy[xy2].y);
	double lk0k1 = lambda(elem, xy[xy6].x, xy[xy6].y);
	double lk1k1 = lambda(elem, xy[xy8].x, xy[xy8].y);
	int nui, mui, nuj, muj;
	for (int i = 0; i < countBasicFuncs; ++i) {
		nui = (i) % 3;
		mui = (i) / 3;
		for (int j = 0; j < countBasicFuncs; ++j)
		{
			nuj = (j) % 3;
			muj = (j) / 3;
			GLocal[i][j] = ((hy / hx) * (lk0k0 * Gxy[nui][nuj] * M[mui][muj]
+ lk1k0 * Gxy[2 - nui][2 - nuj] * M[mui][muj]
				+ lk0k1 * Gxy[nui][nuj] * M[2 - mui][2 - muj] + lk1k1 *
Gxy[2 - nui][2 - nuj] * M[2 - mui][2 - muj])
				+ (hx / hy) * (lk0k0 * Gxy[mui][muj] * M[nui][nuj] + lk1k0
* Gxy[2 - mui][2 - muj] * M[nui][nuj]
				+ lk0k1 * Gxy[mui][muj] * M[2 - nui][2 - nuj] + lk1k1 *
Gxy[2 - mui][2 - muj] * M[2 - nui][2 - nuj])) / 360.;
			MLocal[i][j] = hx * hy * (M[mui][muj] * M[nui][nuj]) / 900.;
		}
	}
}

double FEM::bisquar(int elem, int num, double x, double y)
{
```

```
	int xySt = globalIndex(elem, 0);
	int xyMid = globalIndex(elem, 4);
	int xyEnd = globalIndex(elem, 8);
	double xpEnd = xy[xyEnd].x, xpSt = xy[xySt].x, xpMid = xy[xyMid].x;
	double ypEnd = xy[xyEnd].y, ypSt = xy[xySt].y, ypMid = xy[xyMid].y;
	switch (num)
	{
	case 0: {return basicFunc(x, xpSt, xpMid, xpEnd, 0) * basicFunc(y, ypSt,
ypMid, ypEnd, 0); }
	case 1: {return basicFunc(x, xpSt, xpMid, xpEnd, 1) * basicFunc(y, ypSt,
ypMid, ypEnd, 0); }
	case 2: {return basicFunc(x, xpSt, xpMid, xpEnd, 2) * basicFunc(y, ypSt,
ypMid, ypEnd, 0); }
	case 3: {return basicFunc(x, xpSt, xpMid, xpEnd, 0) * basicFunc(y, ypSt,
ypMid, ypEnd, 1); }
	case 4: {return basicFunc(x, xpSt, xpMid, xpEnd, 1) * basicFunc(y, ypSt,
ypMid, ypEnd, 1); }
	case 5: {return basicFunc(x, xpSt, xpMid, xpEnd, 2) * basicFunc(y, ypSt,
ypMid, ypEnd, 1); }
	case 6: {return basicFunc(x, xpSt, xpMid, xpEnd, 0) * basicFunc(y, ypSt,
ypMid, ypEnd, 2); }
	case 7: {return basicFunc(x, xpSt, xpMid, xpEnd, 1) * basicFunc(y, ypSt,
ypMid, ypEnd, 2); }
	case 8: {return basicFunc(x, xpSt, xpMid, xpEnd, 2) * basicFunc(y, ypSt,
ypMid, ypEnd, 2); }
	};
	return 0;
}

void FEM::addElem(int i, int j, double a)
{
	if (i == j)
	{
		di[i] += a;
	}
	else
	{
		if (i < j)
		{
			int ind;
			for (ind = ig[j]; ind < ig[j + 1]; ++ind)
			{
				if (jg[ind] == i) break;
			}
			ggu[ind] = ggu[ind] + a;
		}
		else
		{
			int ind;
			for (ind = ig[i]; ind < ig[i + 1]; ++ind)
			{
				if (jg[ind] == j)
					break;
			}
			ggl[ind] = ggl[ind] + a;
		}
	}
}
```

LOSSolver.h

```
#pragma once
#include <cstdio>
#include <cmath>
#define myfopen(file, path, mode) fopen_s(file, (path).c_str(), mode)
```

```
typedef double* vectorD;
class LOS {
private:
	double eps;
	int *ig, *jg;
	vectorD di, gu, gl;
	vectorD b;
	vectorD x;
	vectorD mv;
	vectorD z;
	vectorD r;
	vectorD p;
	vectorD diag;
	int n, maxiter;
public:
	void multyMatrixVector(vectorD x, vectorD res);
	double scal(vectorD x, vectorD y);
	double norm(vectorD a);
	void solve();
	LOS(int locn, int* lig, int* ljg, vectorD ld, vectorD ggu, vectorD ggl,
vectorD lb);
	~LOS();
    };
```

LOSSolver.cpp

```
#include "LOSSolver.h"
void LOS::multyMatrixVector(vectorD x, vectorD res)
{
	for (int i = 0; i < n; ++i) {
		int gi = ig[i], gi_1 = ig[i + 1];
		res[i] = di[i] * x[i];
		for (int j = gi; j < gi_1; ++j) {
			int column = jg[j];
			res[i] += gl[j] * x[column];
			res[column] += gu[j] * x[i];
		}
	}
}

double LOS::scal(vectorD a, vectorD b)
{
	double s = 0.0;
	for (int i = 0; i < n; i++)
		s += a[i] * b[i];
	return s;
}

double LOS::norm(vectorD a)
{
	return sqrt(scal(a, a));
}


LOS::LOS(int locn, int* lig, int* ljg, vectorD ld, vectorD ggu, vectorD ggl, vectorD
lb)
```

```
{
	FILE* f;
	fopen_s(&f, "options.txt", "r");
	fscanf_s(f, "%d%lf", &maxiter, &eps);
	n = locn;
	ig = lig;
	jg = ljg;
	di = ld;
	gl = ggl; gu = ggu;
	b = lb;
	mv = new double[n];
	z = new double[n];
	r = new double[n];
	p = new double[n];
	x = new double[n];
	diag = new double[n];
}

LOS::~LOS()
{
	delete[] mv;
	delete[] z;
	delete[] r;
	delete[] p;
	delete[] x;
	delete[] diag;
}

void LOS::solve()
{
	int count = 0;
	for (int i = 0; i < n; ++i)
	{
		x[i] = 1;
	}
	multyMatrixVector(x, mv);
	for (int i = 0; i < n; ++i)
	{
		r[i] = b[i] - mv[i];
		z[i] = r[i];
	}
	multyMatrixVector(z, p);
	double sr = scal(r, r);
	while (sr > eps&& count <= maxiter)
	{
		double pp = scal(p, p);
		double ak = scal(p, r) / pp;
		for (int i = 0; i < n; ++i)
		{
			x[i] = x[i] + ak * z[i];
			r[i] = r[i] - ak * p[i];
		}
		multyMatrixVector(r, mv);
		double bk = -scal(p, mv) / pp;
		for (int i = 0; i < n; ++i)
		{
			z[i] = r[i] + bk * z[i];
			p[i] = mv[i] + bk * p[i];
		}
		sr = sqrt(scal(r, r));
		++count;
	}
	for (int i = 0; i < n; ++i)
	{
```

```
                b[i] = x[i];
        }
}
```

fragmentationGrid.h

```
#pragma once
#include <vector>
#include <string>
using namespace std;
class fragmentationGrid
{
        int *ihx, *ihy;
        int howx;
        int howy;
        int xsize;
        int ysize;
        double hx; double hy;
        double* xw;
        double* yw;
        vector<double> x;
        vector<double> y;
        void read(string dir);
        double NodeValue(double x, double y);
        int K(int i);
public:
        void process(string dir);
};
```

fragmentationGrid.cpp

```
#include "fragmentationGrid.h"
void fragmentationGrid::read(string dir)
{
        FILE* f;
        fopen_s(&f, (dir + "/x.txt").c_str(), "r");
        fscanf_s(f, "%d", &howx);
        xw = new double[howx];
        ihx = new int[howx - 1];
        for (int i = 0; i < howx; ++i) {
                fscanf_s(f, "%lf", &xw[i]);
        }
        fclose(f);
        fopen_s(&f, (dir + "/y.txt").c_str(), "r");
        fscanf_s(f, "%d", &howy);
        yw = new double[howy];
        ihy = new int[howy - 1];
        for (int i = 0; i < howy; ++i) {
                fscanf_s(f, "%lf", &yw[i]);
        }
        fclose(f);
        fopen_s(&f, (dir + "/hxhy.txt").c_str(), "r");
        for (int i = 0; i < howx - 1; ++i) {
                fscanf_s(f, "%d", &ihx[i]);
                ihx[i] *= 2;
                hx = (xw[i + 1] - xw[i]) / ihx[i];
                for (int j = 0; j < ihx[i]; ++j)
                {
                        x.push_back(xw[i] + j * hx);
                }
        }
```

```
	x.push_back(xw[howx - 1]);
	for (int i = 0; i < howy - 1; ++i) {
		fscanf_s(f, "%d", &ihy[i]);
		ihy[i] *= 2;
		hy = (yw[i + 1] - yw[i]) / ihy[i];
		for (int j = 0; j < ihy[i]; j++)
		{
			y.push_back(yw[i] + j * hy);
		}
	}
	y.push_back(yw[howy - 1]);
	xsize = x.size();
	ysize = y.size();
	fclose(f);
}

double fragmentationGrid::NodeValue(double x, double y)
{
	//return x + y;
	//return x * x + y * y;
	//return x * y * y * y;
	return x / y;
}

void fragmentationGrid::process(string dir)
{
	read(dir);
	FILE* f;

	fopen_s(&f, (dir + "/xy.txt").c_str(), "w");
	fprintf(f, "%d\n", ysize * xsize);
	for (int i = 0; i < ysize; i++)
		for (int j = 0; j < xsize; j++)
			fprintf(f, " %0.15lg %0.15lg\n", x[j], y[i]);
	fclose(f);
	fopen_s(&f, (dir + "/nvtr.txt").c_str(), "w");
	fprintf(f, "%d\n", (xsize / 2) * (ysize / 2));

	for (int i = 0; i < (xsize / 2) * (ysize / 2); i++) {
		int k = K(i);
		fprintf(f, "%d %d %d ", k, k + 1, k + 2);
		fprintf(f, "%d %d %d ", k + 2 * xsize / 2, k + 2 * xsize / 2 + 1, k + 2
* xsize / 2 + 2);
		fprintf(f, "%d %d %d\n", k + 2 * (2 * xsize / 2), k + 2 * (2 * xsize /
2) + 1, k + 2 * (2 * xsize / 2) + 2);
	}
	fclose(f);
	fopen_s(&f, (dir + "/nvk.txt").c_str(), "w");
	fprintf(f, "%d\n", xsize * 2 + ysize * 2 - 4);

	for (int i = 0; i < xsize; i++)
	{
		fprintf(f, "%d %le\n", i, NodeValue(x[i], y[0]));
	}
	for (int i = 0; i < xsize; i++)
	{
		fprintf(f, "%d %le\n", i + (2 * xsize / 2) * (ysize - 1),
NodeValue(x[i], y[ysize - 1]));
	}
	for (int i = 1; i < ysize - 1; i++)
	{
		fprintf(f, "%d %le\n", i * (2 * xsize / 2), NodeValue(x[0], y[i]));
	}
	for (int i = 1; i < ysize - 1; i++)
```

```
	{
		fprintf(f, "%d %le\n", (i + 1) * (2 * xsize / 2) - 1, NodeValue(x[xsize
- 1], y[i]));
	}
	fclose(f);
	delete[]xw;
	delete[]yw;
	delete[]ihx;
	delete[]ihy;
	x.~vector();
	y.~vector();
}

int fragmentationGrid::K(int i)
{
	return 2 * ((i) / (xsize / 2)) * (xsize) + 2 * (i % (xsize / 2));
}
```