|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
| C:\Users\Дарья\Desktop\3 ЦВЕТ.png | | |
| Кафедра прикладной математики | | |
|  | | |
| Курсовая работа | | |
| по дисциплине «Численное методы»» | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-72 |
| Студент: | Голованова Дарья, |
| Вариант: | 22 |
| Преподаватель: | Персова М.Г. |
|  |  |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. **Постановка задачи**

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

Эллиптическая краевая задача для функции u определяется уравнением

*,*

заданным в некоторой области Ω с границей и краевыми условиями

,

где значение искомой функции на границе , значение производной функции по направлению внешней нормали к поверхности на границе

1. **Теоретическая часть**

Воспользуемся вариационной постановкой в форме уравнения Галеркина. Общая схема построения:

Исходное уравнения можно записать в операторной форме , где оператор, действующий в Гильбертовом пространстве .

Пусть - некоторая полная замкнутая система линейно независимых элементов из *H*. Ищем приближенное решение уравнения в ее первых n элементах в виде

Отсюда, исходя из того, что любой элемент из *H* может быть представлен в виде суммы элемента из и элемента, ортогонального к этому подпространству, получаем: .

Тогда, , где - финитные базисные функции.

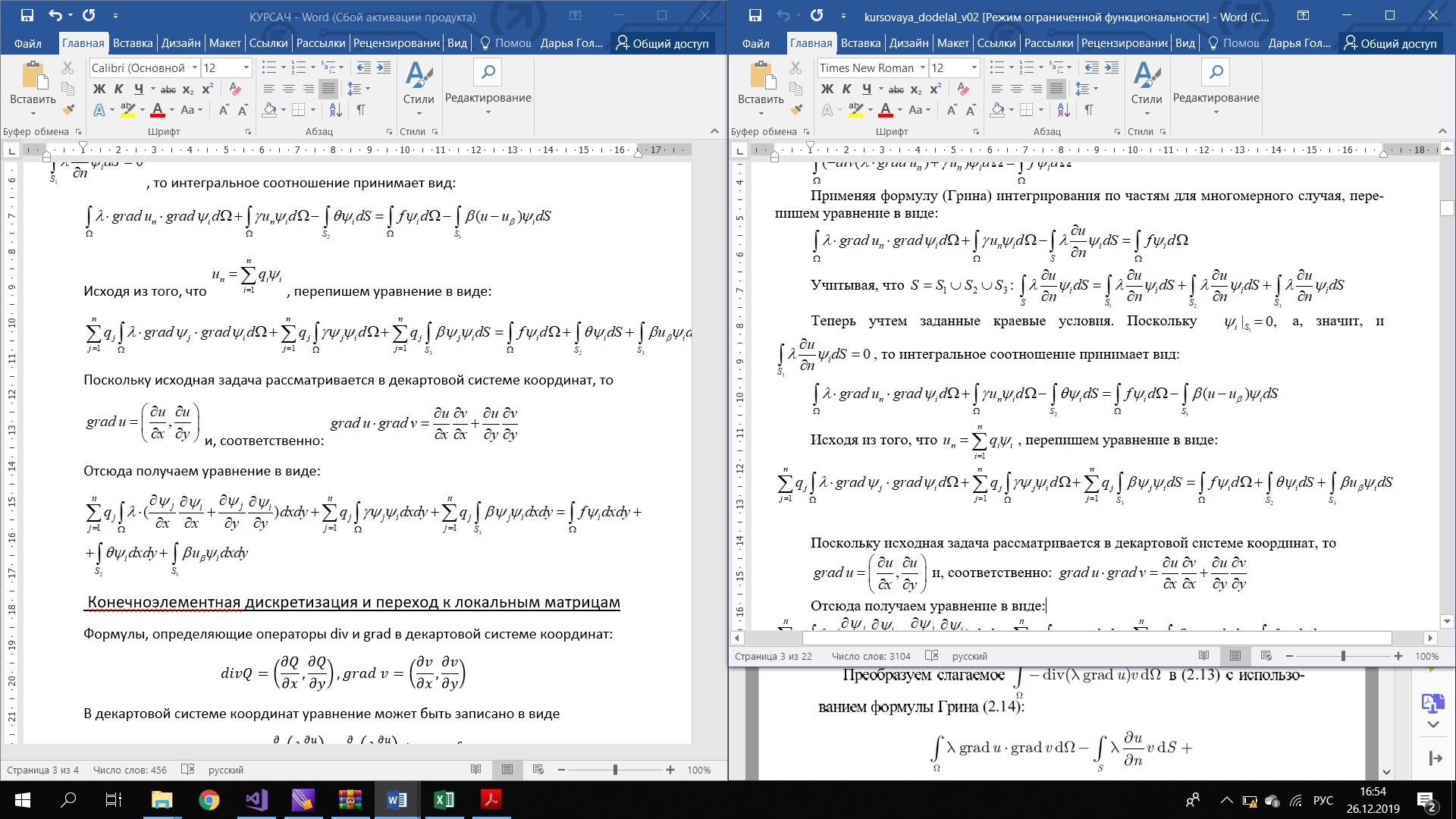
Таким образом, приближенное решение будем искать как проекцию на конечномерное подпространство гильбертова пространства , натянутого на систему базисных функций . Отсюда получаем, что То есть, будем искать приближенное решение в виде разложения по базисным функциям конечномерного подпространства гильбертова пространства.

Рассмотрим Гильбертово пространство функций, интегрируемых с квадратом, то есть, то скалярное произведение в двумерном случае можно переписать в виде:



Применяя формулу Грина, перепишем уравнение в виде:





Учитывая, что , получаем уравнение в виде:



Конечноэлементная дискретизация и переход к локальным матрицам

Так как для решения задачи используются линейные базисные функции, то на каждом конечном элементе (треугольнике) эти функции будут совпадать с функциями , такими, что  равна единице в вершине  и нулю во всех остальных вершинах и т.д.. Любая линейная на конечном элементе функция представима в виде линейной комбинации этих базисных линейных функций, коэффициентами будут значения функции в каждой из вершин треугольника. Таким образом, на каждом конечном элементе нам понадобятся три узла – вершины треугольника.

Получаем:



При дальнейшем решении задачи будем использовать соотношения:



где - это площадь треугольника,  - матрица координат его вершин.

Учитывая построение - функций, получаем следующие соотношения:



Т.е. имеем систему:



Отсюда находим коэффициенты линейных функций 





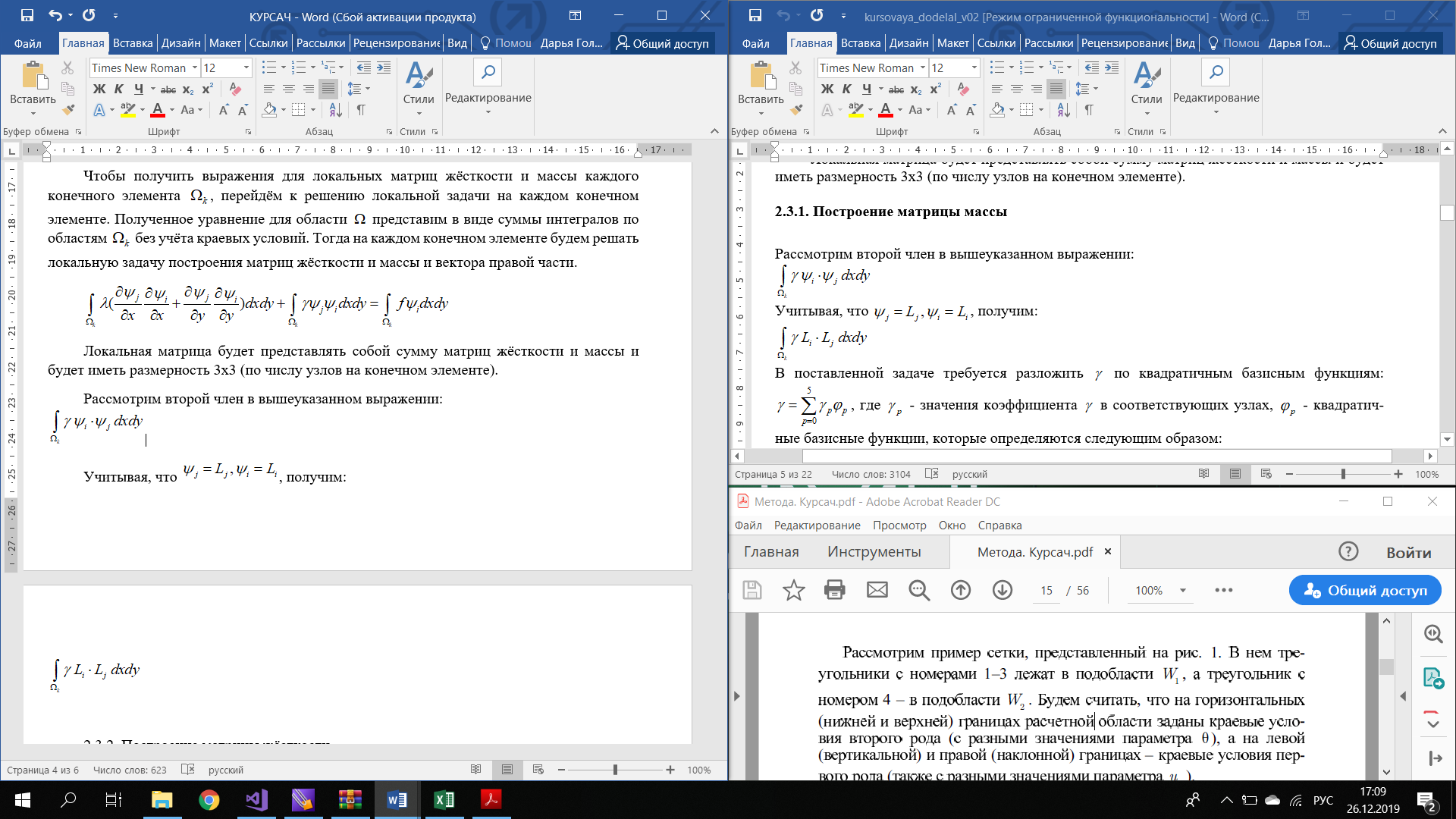
Аналитические выражения для вычисления локальных матриц

Чтобы получить выражения для локальных матриц жёсткости и массы каждого конечного элемента , перейдём к решению локальной задачи на каждом конечном элементе. Полученное уравнение для области  представим в виде суммы интегралов по областям  без учёта краевых условий. Тогда на каждом конечном элементе будем решать локальную задачу построения матриц жёсткости и массы и вектора правой части:



Локальная матрица будет представлять собой сумму матриц жёсткости и массы и будет иметь размерность 3x3 (по числу узлов на конечном элементе).

Построение матрицы массы:



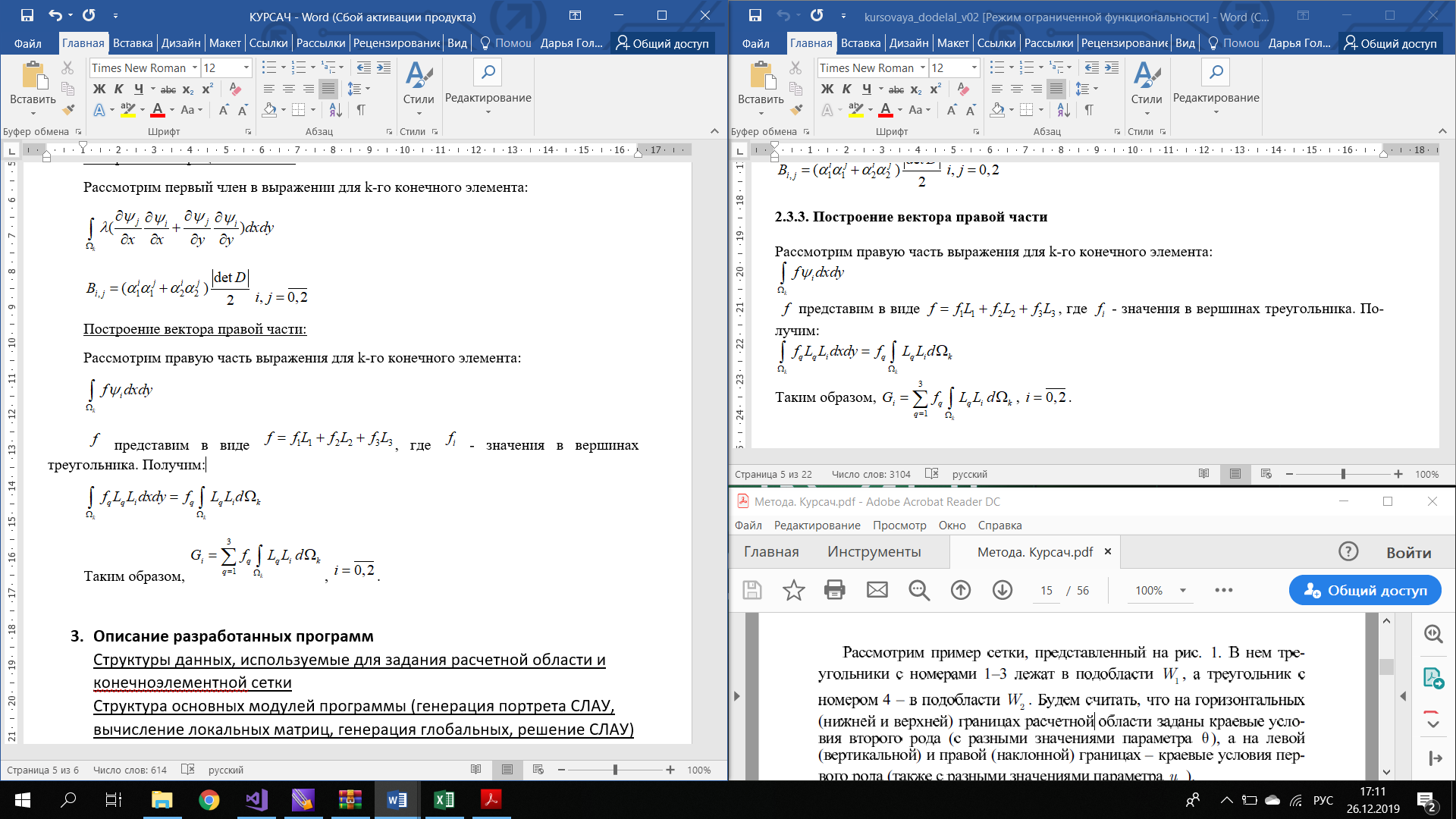
Построение матрицы жесткости:

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





Построение вектора правой части:



При формировании глобальной матрицы из локальных учитываем соответствие локальной и глобальной нумераций каждого конечного элемента.

Глобальная нумерация каждого конечного элемента однозначно определяет позиции вклада его локальной матрицы в глобальную. Поэтому, зная глобальные номера соответствующих узлов конечного элемента, определяем какие элементы глобальной матрицы изменятся при учете текущего конечного элемента. Аналогичным образом определяется вклад локального вектора правой части в глобальный.

При учете текущего локального вектора изменятся те элементы глобального вектора правой части, номера которых совпадают с глобальными номерами узлов, присутствующих в этом конечном элементе.

Учет первых краевых условий

Для учета первых краевых условий, в глобальной матрице и глобальном векторе находим соответствующую глобальному номеру краевого узла строку, и ставим вместо диагонального элемента глобальной матрицы на этой строке «большое число», а вместо элемента с таким номером в вектор правой части ­­- «большое число», умноженное на значение краевого условия, заданное в исходной задаче.

Краевые условия второго и третьего рода не учитывались.

Метод решения СЛАУ

Для решения СЛАУ с полученной матрицей в разреженно-строчном формате будем применять Локально-оптимальную схему. В результате получим вектор, элементами которого и будут искомые коэффициенты разложения нашего решения по базисным функциям, учитывая построение базисных функций, то фактически мы получим вектор, координатами которого будут значения функции-решения в узлах сетки.

1. **Описание разработанных программ**

Структуры входных данных, используемые для задания расчетной области и конечноэлементной сетки

Файл **Nodes.txt** задает количество глобальных узлов сетки и их координаты (x,y).

Файл **Elems.txt** задает количество локальных элементов сетки – треугольников, номера глобальных узлов, принадлежащих определенному элементу, и номер подобласти, которой принадлежит конечный элемент.

Файл **Cond1.txt** задает количество ребер, на которых задаются первые краевые условия, два узла для каждого ребра и номер краевого условия.

Описание заголовочных модулей программы

Модуль **Function.h** содержит функции определения краевых условий и параметры исходной задачи.

Модуль **LOS.h** содержит функции для расчета Локально-оптимальной схемы глобальной матрицы.

Структура основных модулей программы

Процедура **Portrait,** генерирующая портрет матрицы.

Процедура **G**, генерирующая матрицу жесткости.

Процедура **M**, генерирующая матрицу массы.

Процедура **AddLocalToGlobal,** собирающая глобальную матрицу и глобальный вектор правой части.

1. **Описание тестирования программ**

Тест для проверки константных краевых условий первого рода

2

1. 1

|  |  |  |
| --- | --- | --- |
| **Координаты узлов** | **Конечные элементы** | **1 краевое условие** |
| 3  0 0  1 0  1 1 | 1  0 1 2 1 | 3  0 1 0  1 0 0  1 1 0 |

|  |  |  |
| --- | --- | --- |
| **Точное решение** | **Полученное решение** | **Погрешность** |
| 5 | 5 | 0,00E+00 |
| 5 | 5 | 0,00E+00 |
| 5 | 5 | 0,00E+00 |

Тест для проверки смешанных краевых условий первого рода

2 3

0 1

|  |  |  |
| --- | --- | --- |
| **Координаты узлов** | **Конечные элементы** | **1 краевое условие** |
| 4  0 0  2 0  0 1  2 1 | 2  0 1 2 1  1 2 0 2 | 2  0 1 1  1 2 1 |

|  |  |  |
| --- | --- | --- |
| **Точное решение** | **Полученное решение** | **Погрешность** |
| 1 | 1,00E+00 | 0,00E+00  0,00E+00  0,00E+00  0,00E+00 |
| 2 | 2,00E+00 |
| 0  1 | 0,00E+00  1,00E+00 |
|  |  |

Линейные базисные функции позволяют получить достаточно точное решение полинома первой степени.

1. **Тексты основных модулей программ**

**Function.h**

#pragma once

#include<fstream>

#include<iostream>

#include<vector>

#include<cmath>

#include<string>

#include<sstream>

#include<iterator>

#include<iomanip>

#include<algorithm>

#include<conio.h>

using namespace std;

int n, m;

vector <vector<double>> nodes;

vector<vector<int>> elems, conds1, conds2, conds3;

vector<vector<double>> local\_matrix(3), G\_matrix(3), M\_matrix(3);

vector<double> local\_b(3);

vector<vector<double>> global\_A;

vector<double> global\_b;

vector<double> ggl, di;

vector<int> ig, jg;

double Kraev\_1(int num\_f, double x, double y)

{

switch (num\_f)

{

case 0:

return exp(x);

case 1:

return x+2\*y;

default: 0;

}

}

void uc\_kr1()

{

ifstream fin("Cond1.txt");

int p, v1, v2, num\_f; //кол-во ребер, на которых задано 1 кр.условие

long B = 1e+10;

fin >> p;

for (int i = 0; i < p; i++)

fin >> v1 >> v2 >> num\_f;

double x1, y1, x2, y2;

x1 = nodes[v1][0];

y1 = nodes[v1][1];

x2 = nodes[v2][0];

y2 = nodes[v2][1];

double res1 = Kraev\_1(num\_f, x1, y1);

double res2 = Kraev\_1(num\_f, x2, y2);

di[v1] = B;

di[v2] = B;

global\_b[v1] = B \* res1;

global\_b[v2] = B \* res2;

}

double Diffusion\_coef()

{ return 2;}

double Gamma\_coef()

{ return 3;}

double function(double x, double y)

{ return -exp(x+y);}

**LOS.h**

#pragma once

#include "Function.h"

vector<double> Ma;

void Mult(vector<double> &x, vector<double> &Ma)

{

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

int gi = ig[i], gi\_1 = ig[i + 1];

Ma[i] = di[i] \* x[i];

for (int j = gi; j < gi\_1; ++j) {

int column = jg[j];

Ma[i] += ggl[j] \* x[column];

Ma[column] += ggl[j] \* x[i];

}

}

}

double Mult\_scal(vector<double> vec1, vector<double> vec2)

{

double sum = 0;

for (int i = 0; i < n; i++)

sum += vec1[i] \* vec2[i];

return sum;

}

void LOS()

{

int k = 0;

double norm\_eps;

vector <double>x0(n,0);

vector <double>r0(n, 0);

vector <double>z0(n, 0);

vector <double>p0(n, 0);

vector <double>xk(n, 0);

Ma.resize(n);

double maxiter = 100;

double eps = 1e-12;

for (int i = 0; i < n; ++i)

{

x0[i] = 1;

}

int count = 0;

Mult(x0, Ma);

for (int i = 0; i < n; ++i)

{

r0[i] = global\_b[i] - Ma[i];

z0[i] = r0[i];

}

Mult(z0, p0);

double sr = Mult\_scal(r0, r0);

while (sr > eps&& count <= maxiter)

{

double pp = Mult\_scal(p0, p0);

double ak = Mult\_scal(p0, r0) / pp;

for (int i = 0; i < n; ++i)

{

x0[i] = x0[i] + ak \* z0[i];

r0[i] = r0[i] - ak \* p0[i];

}

Mult(r0, Ma);

double bk = -Mult\_scal(p0, Ma) / pp;

for (int i = 0; i < n; ++i)

{

z0[i] = r0[i] + bk \* z0[i];

p0[i] = Ma[i] + bk \* p0[i];

}

sr = sqrt(Mult\_scal(r0, r0));

++count;

}

cout << "k = " << k << '\t' << "norm\_eps = " << eps << endl;

for (int i=0;i<n;i++)

cout << x0[i] << '\n';

}

**Source.cpp**

#include "Function.h"

#include "LOS.h"

#include <Set>

void Input()

{

ifstream fnodes, felems, fcond1, fcond2, fcond3;

string str;

//кол-во глобальных узлов и их координаты

fnodes.open("Nodes.txt");

fnodes >> n; //кол-во узлов

nodes.resize(n);

for (int i = 0; i < n; i++)

{

nodes[i].resize(2);

fnodes >> nodes[i][0] >> nodes[i][1];

}

fnodes.close();

//список конечных элементов с локальными узлами и номером подобласти

felems.open("Elems.txt");

felems >> m; //кол-во конечных элементов

elems.resize(m);

for (int i = 0; i < m; i++)

{

elems[i].resize(4);

felems >> elems[i][0] >> elems[i][1] >> elems[i][2] >> elems[i][3];

}

felems.close();

}

void Portrait()

{

vector<set<int>> list(n);

for (int s = 0; s < elems.size(); s++)

for (int p = 0; p < 3; p++)

for (int j = p + 1; j < 3; j++)

{

int ind1 = elems[s][p];

int ind2 = elems[s][j];

if (ind1 < ind2) swap(ind1, ind2);

list[ind1].insert(ind2);

}

//создание портрета по списку

ig.resize(n+1);

ig[0] = ig[1] = 0;

for (int i = 2; i < n + 1; i++) {

int col = ig[i - 1];

ig[i] = col + list[i - 1].size();

}

jg.resize(ig[n]);

for (int i = 1, k = 0; i < n; i++) {

for (int j : list[i]) {

jg[k] = j;

k++;

}

}

}

void G(vector<double> &x, vector<double>&y)

{

double det = abs((x[1] - x[0])\*(y[2] - y[0]) - (x[2] - x[0])\*(y[1] - y[0]));

double mult = Diffusion\_coef() \* det / 2;

vector<vector<double>> a(3);

a[0].push\_back(x[1] \* y[2] - y[1] \* x[2]);

//a[0].push\_back(-x[0] \* y[2] + x[2] \* y[0]);

a[0].push\_back(-(y[2] - y[1]));

a[0].push\_back(x[2] - x[1]);

a[1].push\_back(-x[0] \* y[2] + x[2] \* y[0]);

a[1].push\_back(y[2] - y[0]);

a[1].push\_back(-x[2] + x[0]);

a[2].push\_back(x[0] \* y[1] - x[1] \* y[0]);

a[2].push\_back(-y[1] + y[0]);

a[2].push\_back(x[1] - x[0]);

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

{

G\_matrix[i].resize(3);

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

G\_matrix[i][j] = mult \* ((a[i][1])\*(a[j][1]) + (a[i][2])\*(a[j][2]));

}

}

void M(vector<double> &x, vector<double>&y)

{

double det = abs((x[1] - x[0])\*(y[2] - y[0]) - (x[2] - x[0])\*(y[1] - y[0]));

double \*f = new double[3], mult = Gamma\_coef() \* det / 24;

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

{

M\_matrix[i].resize(3);

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

(i == j) ? M\_matrix[i][j] = 2 \* mult : M\_matrix[i][j] = mult;

}

f[0] = mult \* function(x[0], y[0]);

f[1] = mult \* function(x[1], y[1]);

f[2] = mult \* function(x[2], y[2]);

local\_b[0] = 2 \* f[0] + f[1] + f[2];

local\_b[1] = f[0] + 2 \* f[1] + f[2];

local\_b[2] = f[0] + f[1] + 2 \* f[2];

}

void Local\_Matrix(vector<double> x, vector<double> y, vector<double> &local\_f)

{

G(x, y);

M(x, y);

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

{

local\_matrix[i].resize(3);

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

local\_matrix[i][j] = M\_matrix[i][j] + G\_matrix[i][j];

}

}

void AddLocalToGlobal(vector<int>elems)

{

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

{

di[elems[i]] += local\_matrix[i][i];

global\_b[elems[i]] += local\_b[i];

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

{

auto a = elems[i];

auto b = elems[j];

if (a < b) swap(a, b);

auto begin = jg.begin() + ig[a];

if (ig[a + 1] > ig[a]) //!= профиль не пуст

{

auto end = jg.begin() + ig[a + 1] - 1;

auto iter = lower\_bound(begin, end, b); //дихотомия

auto index = iter - jg.begin();

ggl[index] += local\_matrix[i][j];

}

}

}

}

int main()

{

vector<double>x(3), y(3);

Input();

Portrait();

di.resize(n);

global\_b.resize(n);

ggl.resize(ig[n]-ig[0]);

for (int i = 0; i < m; i++)// локальная матрица для всех элементов, но без их сохранения

{

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

{

int point = elems[i][j];

x[j] = nodes[point][0];

y[j] = nodes[point][1];

}

Local\_Matrix(x, y, local\_b);

AddLocalToGlobal(elems[i]);

}

uc\_kr1();

LOS();

\_getch();

}