|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Курсовая работа | | |
| по дисциплине «Численные методы» | | |
|  | | |
|  | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-72 |
| Вариант: | 26 |
| Студент: | Максимова Софья |
|  |  |
|  |  |
| Преподаватели: | Персова М. Г. |
|  | Патрушев И.И. |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. Постановка задачи

1.1. Формулировка задания

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

1.2. Постановка задачи

Эллиптическая краевая задача для функции u определяется уравнением

,

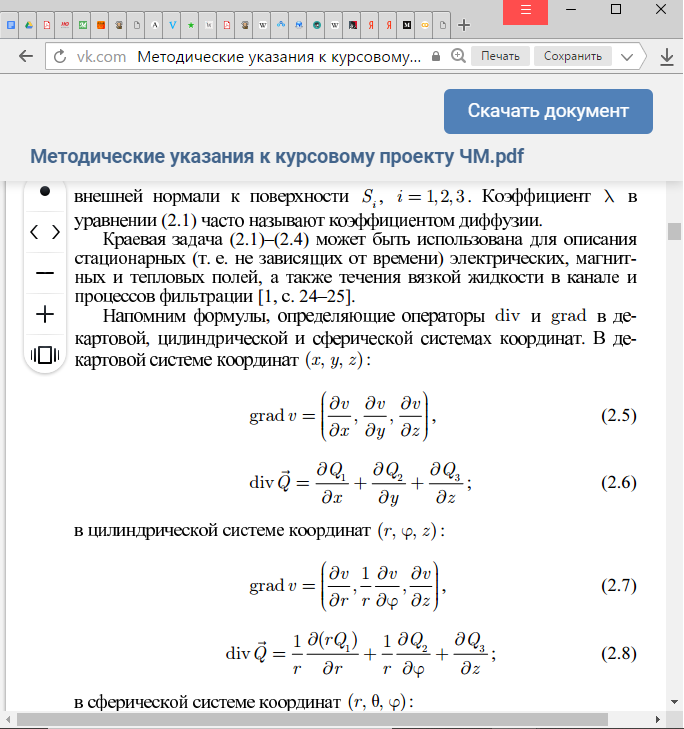
заданным в некоторой области  с границей , и краевыми условиями



В декартовой системе координат (x,y) уравнение в виде :

,

где



2. Теоретическая часть

2.1. Вариационная постановка в форме уравнения Галеркина

Эллиптическое уравнение будем решать методом Галёркина. В операторной форме исходное уравнение можно переписать в виде: , где  - оператор, действующий в Гильбертовом пространстве . Нам нужно найти приближение к элементу .

Пусть  – некоторая полная замкнутая система линейно независимых элементов из . Ее первые  элементов выделяют в  конечномерное подпространство , в котором и ищется приближенное решение уравнения в виде . Поскольку рассматриваемый оператор  - оператор Лапласа, то . Отсюда, исходя из того, что любой элемент из  может быть представлен в виде суммы элемента из  и элемента, ортогонального к этому подпространству, получаем: . Тогда  и, соответственно, , где  - финитные базисные функции (Т.е. такие функции, каждая из которых отлична от нуля лишь на нескольких малых подобластях расчётной области. Эти подобласти в совокупности с ненулевыми базисными функциями называются конечными элементами).

Таким образом, приближенное решение будем искать как проекцию на конечномерное подпространство гильбертова пространства , натянутого на систему базисных функций . Отсюда получаем, что  или .

Применяя формулу (Грина) интегрирования по частям для многомерного случая, перепишем уравнение в виде:



Учитывая, что : 

Теперь учтем заданные краевые условия. Поскольку  а, значит, и , то интегральное соотношение принимает вид:



Исходя из того, что :



Поскольку исходная задача рассматривается в декартовой системе координат, то

 и, соответственно: 

Отсюда получаем уравнение в виде:



Так, при решении краевой задачи с использованием базисных функций, принимающих нулевые значения во всех узлах сетки, кроме одного, конечноэлементная СЛАУ для вектора весов q может быть записана в матричном виде:

Aq=b

2.2. Конечноэлементная дискретизация

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 (*L1, L2* соответственно). Любая линейная на  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 будут находиться три узла - вершины треугольников.

Получаем:



При дальнейшем решении задачи следует использовать соотношения:



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

Учитывая построение - функций, получаем следующее:

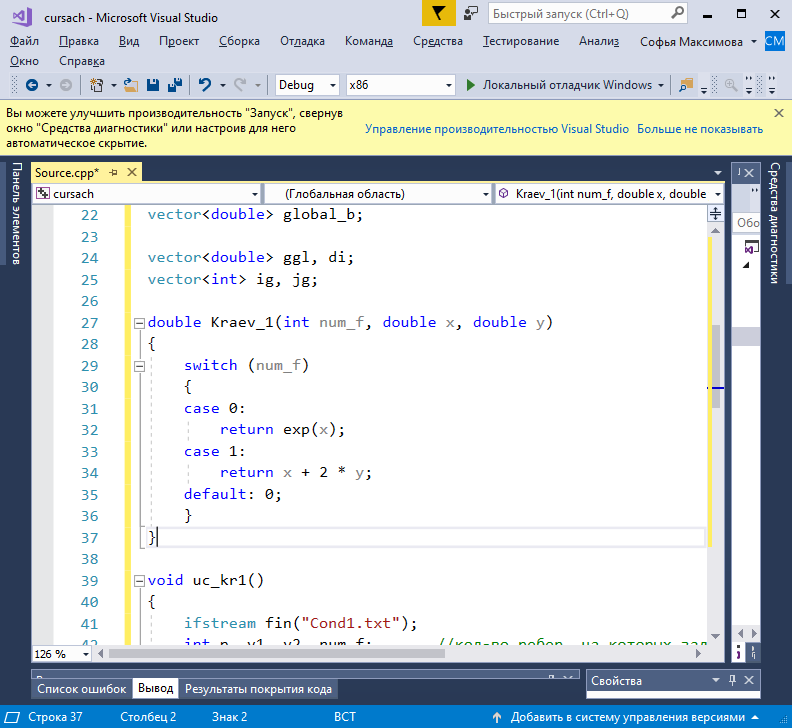


Отсюда находим коэффициенты линейных функций 





Структуры данных для хранения информации о краевых условиях организованы массивами целых чисел, где сначала указано количество рёбер, затем первое число - номер узла начала ребра, второе – номер узла конца ребра, а третье – номер формулы, которая будет вычислять значение параметра ug с помощью указанной ниже подпрограммы.



2.3. Переход к локальным матрицам

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 (G) и массы (M) и будет иметь размерность 3x3 (по числу узлов на конечном элементе, в нашем случае для треугольников).

2.3.1. Построение матриц массы и жёсткости

У первого члена вышеуказанного выражения:





У второго:



Учитывая, что , получим:



В поставленной задаче требуется разложить  по квадратичным базисным функциям: , где  - значения коэффициента  в соответствующих узлах,  - квадратичные базисные функции, которые определяются следующим образом:

Так, при расчёте локальной матрицы по локальной нумерации узлов конечного элемента однозначно определяется локальная нумерация соответствующих базисных функций, которая в свою очередь, определяет расположение компонент в локальной матрице.

Структуры данных, используемые для локальных матриц, представлены в виде массивов глобальных узлов (первое число – количество узлов, второе и третье - координаты) и конечных элементов (первое число – количество элементов, последующие четыре числа обозначают набор глобальных узлов с номером подобласти, в которой они находятся).

**2.3.3. Построение вектора правой части**

Рассмотрим правую часть выражения для k-го конечного элемента:

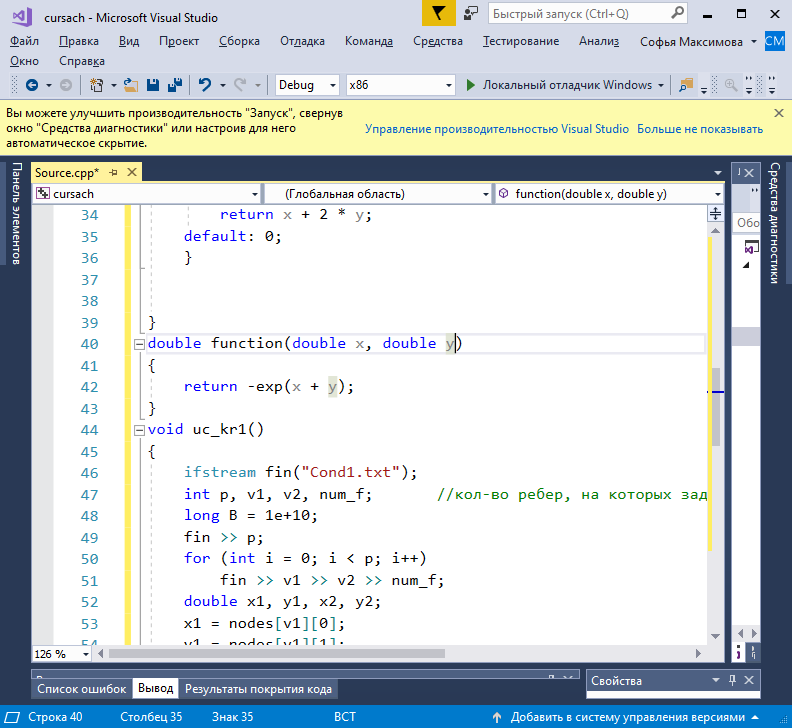


 представим в виде , где  - значения в вершинах треугольника. Получим:



Таким образом, , .

Вектор правой части определяется функцией или какой-то константой, высчитываемой для каждого узла элемента.



**2.3.4. Сборка глобальной матрицы и глобального вектора правой части**

Глобальная матрица формируется благодаря полученным локальным (из матриц жёсткости и массы) с учётом соответствия локальной нумерации глобальной для каждого конечного элемента. Поэтому, зная глобальные номера соответствующих узлов конечного элемента, можно определить и то, 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.

2.4. Учет краевых условий

**2.4.1. 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.

**2.4.1. Учет вторых и третьих краевых условий**

Рассмотрим краевые условия второго и третьего рода



Отсюда получаем, что для учета краевых условий необходимо вычислить интегралы:



Краевые условия второго и третьего рода задаются на ребрах, т.е. определяются двумя узлами, лежащими на ребре. В данной работе они не реализованы.

2.5. Метод решения СЛАУ

Для решения СЛАУ с полученной матрицей в разреженном строчно - столбцовом формате будем применять ЛОС. В результате получим вектор u, который будет для первых краевых условий равен истинному значению.

3. Текст программы

Source.cpp

#include "Function.h"

#include "LOS.h"

#include <Set>

void Input()

{

ifstream fnodes, felems, fcond1;

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(-(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])

{

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

{

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();

}

LOS.h

#pragma once

#include "Function.h"

vector<double> ggl\_new, di\_new, ggu\_new, 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>rk(n, 0);

vector <double>zk(n, 0);

vector <double>pk(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 << "norm\_eps = " << eps << endl;

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

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

}

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 x\*x+y;

case 1:

return 0;

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 -4+3\*(x\*x+y);

}

4. Тесты

u = x\*x+y

f = -4+3\*(x\*x+y);

ƛ = 2

ɤ = 3

|  |  |  |
| --- | --- | --- |
| Краевые условия: | Конечномерные элементы: | Список глобальных узлов: |
| 1  0 1 0 | 1  0 1 0 | 3  0 0  0 1  1 0 |

norm\_eps = 1e-12

0

1

1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

u = x+y

f = 4\*(x+y);

ƛ = 2

ɤ = 4

|  |  |  |
| --- | --- | --- |
| Краевые условия: | Конечномерные элементы: | Список глобальных узлов: |
| 2  0 1 0  1 2 0 | 2  0 1 4 0  1 2 3 1 | 5  0 0  0 1  6 -1  2 3  1 0 |

norm\_eps = 1e-12

1

5

2.02479

2.51724