Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра прикладной математики

Лабораторная работа № 3

по дисциплине

«Уравнения математической физики»

Факультет прикладной математики и информатики

Группа ПМ-01

Студент Курочкин А.В.

Преподаватель Задорожный А.Г.

Персова М.Г.

Вариант 9

Новосибирск

2013

1. Цель работы

Разработать программу решения гармонической задачи методом конечных элементов. Сравнить прямой и итерационной методы решения получаемой в результате конечноэлементной аппроксимации СЛАУ.

1. Задание

Решить трехмерную гармоническую задачу в декартовых координатах, базисные функции – трилинейные.

1. Анализ

Пусть дана дифференциальная задача , в которой правая часть представима в виде , тогда, если прочие части не зависят от t, то решение представимо в виде , где  и  – две зависящие только от пространственных координат функции, удовлетворяющие системе уравнений









, 

, 

, 

Запишем постановку Галёркина, применим первую формулу Грина



, 







Пусть ,, тогда



1. Текст программы

**objects.h**

#ifndef OBJECTS\_MFE\_H\_

#define OBJECTS\_MFE\_H\_

struct node{double x, y, z; int number;};//узел

struct face{int node\_n[8], number, area;};//двойной двумерный КЭ

struct dCube{int node\_n[16], number, area;};//двойной трёхмерный КЭ

struct PortGenEl{int value; PortGenEl\* next;};//элемент списка

class PortMF\_list{

private:

PortGenEl \*begin, \*end, \*cash;

int ls, ms, numberline;

void add(int val);

void ElemElimination();

public:

PortMF\_list();//на самом деле construct

~PortMF\_list();

void construct();//конструктор

int size();

int size\_before(int n);

void ElemAddition(dCube &el\_a); //Добавление элемента

int popList();//берём элемент

void cashEQbegin();

void setNum(int s\_num); //Установить номер линии

int getms();

};

class CPortraitGener{

private:

int n;

PortMF\_list\* lists;

public:

void init(int n\_nodes);

void ElemAddition(dCube& el\_a);

void gen(int \*gi, int \*&gj, int &m);

CPortraitGener();

~CPortraitGener();

};

#endif

**objects.cpp**

#include "objects.h"

PortMF\_list::PortMF\_list(){construct();}

void PortMF\_list::construct(){ls =0; begin = end = cash = 0;}

void PortMF\_list::setNum(int s\_num){numberline = s\_num;}

PortMF\_list::~PortMF\_list()

{ cash = 0;

while(begin != 0)ElemElimination();}

void PortMF\_list::ElemAddition(dCube &el\_a)

{ if(numberline > 10000)numberline = numberline;

for(int i = 0; i < 16; i++)add(el\_a.node\_n[i]);}

void PortMF\_list::cashEQbegin(){cash = begin;}

int PortMF\_list::popList()

{ int Val = cash->value;

cash = cash->next;

return Val;}

int PortMF\_list::getms(){return ms;}

void PortMF\_list::add(int val)

{ if(val <= numberline){

PortGenEl \*ElemAddition;

if(begin == 0){

begin = new PortGenEl;

begin->value = val;

begin->next = 0;

end = begin;

cash = begin;}

else{

if(val < begin->value){

ElemAddition = new PortGenEl;

ElemAddition->value = val;

ElemAddition->next = begin;

begin = ElemAddition;

cash = begin;}

else{

if(val > end->value){

ElemAddition = new PortGenEl;

ElemAddition->value = val;

ElemAddition->next = 0;

end->next = ElemAddition;

end = end->next;}

else{

cash = begin;

while(cash->next != 0 && val > cash->next->value) cash = cash->next;

if(cash->next != 0 && cash->next->value != val && cash->value != val){

ElemAddition = new PortGenEl;

ElemAddition->value = val;

ElemAddition->next = cash->next;

cash->next = ElemAddition;}

}

}

}

ls++;

}

}

void PortMF\_list::ElemElimination(){

if(begin == end){

delete begin;

begin = end = cash = 0;}

else{

cash = begin;

while(cash->next != end) cash = cash->next;

delete cash->next;

cash->next = 0;

end = cash;}

}

int PortMF\_list::size\_before(int n){

int t = 0;

cash = begin;

if(begin != 0){

if(begin->value < n) t++;

while(cash->next != 0 && cash->next->value < n){

t++;

cash = cash->next;

}

}

ms = t;

return t;

}

CPortraitGener::CPortraitGener(){lists = new PortMF\_list [0];}

CPortraitGener::~CPortraitGener(){delete []lists; lists = 0;}

void CPortraitGener::init(int num){

n = num;

lists = new PortMF\_list [n];

for(int i = 0 ; i < n; i++)lists[i].setNum(i);

}

void CPortraitGener::ElemAddition(dCube &el){

for(int i = 0; i < 16; i++)lists[el.node\_n[i]].ElemAddition(el);

}

void CPortraitGener::gen(int \*gi, int \*&gj, int &m){

m = 0;

gi[0] = 0;

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

gi[i] = m;

m = m + lists[i].size\_before(i);

}

gi[n] = m;

gj = new int [m];

int shift, m1 = 0, iters\_m;

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

shift = m1;

iters\_m = lists[i].getms();

lists[i].cashEQbegin();

for(int j = 0; j < iters\_m; j++)

gj[shift+j] = lists[i].popList();

m1 = m1 + iters\_m;

}

}

**gener1.h**

typedef vector<double> dvector;

class GridGenerator{

private:

static int GridGen\_pr(double a, double b, double hmin, double k, double \*&masGrid);

public:

static int IsLegal(double a, double b, double hmin, double k);

static int numberNodes(double a, double b, double hmin, double k);

static int GridGen(double a, double b, double hmin, double k, string file\_name);

static int GridGen(double a, double b, double hmin, double k, dvector &grid\_vector);

static int GridGen(double a, double b, double hmin, double k, int &n, double\*& masGrid);

static int InsertedGridGen(double a, double b, double hmin, double k, string file\_name);

static int InsertedGridGen(double a, double b, double hmin, double k, dvector &grid\_vector);

static int InsertedGridGen(double a, double b, double hmin, double k, int &n, double\*& masGrid);

};

#endif

**gener2.h**

#ifndef GENER2\_MFE\_H\_

#define GENER2\_MFE\_H\_

#include "gener1.h"

struct Point{double x, y, z;

Point(){x = y = z = 0;}

Point(double set\_x, double set\_y)

{x = set\_x; y = set\_y; z = 0;}

Point(double set\_x, double set\_y, double set\_z)

{x = set\_x; y = set\_y; z = set\_z;}

};

class GridGenerator\_rect{

public:

static int GridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, string file\_name);

static int GridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, int &n, double \*\*&masGrid);

static int InsertedGridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, string file\_name);

static int InsertedGridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, int &n, double \*\*&masGrid);

private:

static int GridGen\_pr(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, double \*\*&masGrid);

};

#endif

**gener3.h**

#ifndef GENER3\_MFE\_H\_

#define GENER3\_MFE\_H\_

#include "gener2.h"

class GridGenerator3{

private:

static int GridGen\_pr(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double h\_zmin, double k\_x, double k\_y, double k\_z, Point \*&masGrid);

public:

static int GridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double h\_zmin, double k\_x, double k\_y, double k\_z, string file\_name);

static int GridGenMain(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double h\_zmin, double k\_x, double k\_y, double k\_z, string file\_cords, string file\_elements, string file\_faces);

};

#endif

**gener1.cpp**

#include "gener1.h"

int GridGenerator::IsLegal(double a, double b, double hmin, double k){

if((a > b)||(hmin <= 0.)||(k < 1.)) return 0;

return 1;

}

int GridGenerator::numberNodes(double a, double b, double hmin, double k){

double n;

if(k != 1) n = log(1 - (b-a)\*(k-1)/hmin) / log(k);

else n = (b-a)/hmin - 1;

return n + 2;

}

int GridGenerator::GridGen\_pr(double a, double b, double hmin, double k, double \*&masGrid){

int n = numberNodes(a, b, hmin, k);

masGrid = new double [n];

double h = hmin;

masGrid[0] = a;

for(int i = 1; i < n; i++){masGrid[i] = masGrid[i-1] + h; h = h \* k;}

masGrid[n-1] = b;

return n;

}

int GridGenerator::GridGen(double a, double b, double hmin, double k, int &n, double \*&masGrid){

if(!IsLegal(a, b, hmin, k)) return 0;

n = GridGen\_pr(a, b, hmin, k, masGrid);

return 1;

}

int GridGenerator::GridGen(double a, double b, double hmin, double k, dvector &grid\_vector){

if(!IsLegal(a, b, hmin, k))return 0;

double \*masGrid;

int n = GridGen\_pr(a, b, hmin, k, masGrid);

grid\_vector.clear();

for(int i = 0; i < n; i++)grid\_vector.push\_back(masGrid[i]);

delete[] masGrid;

return 1;

}

int GridGenerator::GridGen(double a, double b, double hmin, double k, string file\_name){

if(!IsLegal(a, b, hmin, k)) return 0;

FILE \*out\_f = fopen(file\_name.c\_str(), "w");

double \*masGrid;

int n = GridGen\_pr(a, b, hmin, k, masGrid);

fprintf(out\_f, "%d\n", n);

for(int i = 0; i < n; i++){fprintf(out\_f, "%.15lf\n", masGrid[i]);}

delete[] masGrid;

fclose(out\_f);

return 1;

}

int GridGenerator::InsertedGridGen(double a, double b, double hmin, double k, string file\_name){

double k1 = sqrt(k);

double hmin1 = hmin/(1+k1);

return GridGen(a, b, hmin1, k1, file\_name);

}

int GridGenerator::InsertedGridGen(double a, double b, double hmin, double k, dvector &grid\_vector){

double k1 = sqrt(k);

double hmin1 = hmin/(1+k1);

return GridGen(a, b, hmin1, k1, grid\_vector);

}

int GridGenerator::InsertedGridGen(double a, double b, double hmin, double k, int &n, double\*& masGrid){

double k1 = sqrt(k);

double hmin1 = hmin/(1+k1);

return GridGen(a, b, hmin1, k1, n, masGrid);

}

**gener2.cpp**

#include "gener2.h"

int GridGenerator\_rect::GridGen\_pr(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, double \*\*&masGrid){

double\* Xm;double\* Ym;

int n\_x, n\_y;

double ax = left\_bottom.x, bx = right\_top.x, ay = left\_bottom.y, by = right\_top.y;

GridGenerator::GridGen(ax, bx, h\_xmin, k\_x, n\_x, Xm);

GridGenerator::GridGen(ay, by, h\_ymin, k\_y, n\_y, Ym);

int n = n\_x \* n\_y;

masGrid = new double\* [n];

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

masGrid[i] = new double [2];

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

for(int j = 0; j < n\_y; j++){

masGrid[j\*n\_x + i][0] = Xm[i];

masGrid[j\*n\_x + i][1] = Ym[j];

}

}

delete[] Xm;

delete[] Ym;

return n;

}

int GridGenerator\_rect::GridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, int &n, double \*\*&masGrid){

if(left\_bottom.x < right\_top.y || left\_bottom.y < right\_top.y || h\_xmin < 0 || h\_ymin < 0 || k\_x <= 1 || k\_y <= 1) return 0;

n = GridGen\_pr(left\_bottom, right\_top, h\_xmin, h\_ymin, k\_x, k\_y, masGrid);

return 1;

}

int GridGenerator\_rect::GridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, string file\_name){

if(left\_bottom.x > right\_top.x || left\_bottom.y > right\_top.y || h\_xmin < 0 || h\_ymin < 0 || k\_x <= 1 || k\_y <= 1) return 0;

double \*\*masGrid;

FILE \*out\_f = fopen(file\_name.c\_str(), "w");

int n = GridGen\_pr(left\_bottom, right\_top, h\_xmin, h\_ymin, k\_x, k\_y, masGrid);

fprintf(out\_f, "%d\n", n);

for(int i = 0; i < n; i++)fprintf(out\_f, "%.15lf\t%.15lf\n", masGrid[i][0], masGrid[i][1]);

delete[] masGrid;

fclose(out\_f);

return 1;

}

int GridGenerator\_rect::InsertedGridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, int &n, double \*\*&masGrid){

double k\_x1 = sqrt(k\_x), k\_y1 = sqrt(k\_y);

double h\_x1 = h\_xmin/(1+k\_x1), h\_y1 = h\_ymin/(1+k\_y1);

return InsertedGridGen(left\_bottom, right\_top, h\_x1, h\_y1, k\_x1, k\_y1, n, masGrid);

}

int GridGenerator\_rect::InsertedGridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double k\_x, double k\_y, string file\_name){

double k\_x1 = sqrt(k\_x), k\_y1 = sqrt(k\_y);

double h\_x1 = h\_xmin/(1+k\_x1), h\_y1 = h\_ymin/(1+k\_y1);

return GridGen(left\_bottom, right\_top, h\_x1, h\_y1, k\_x1, k\_y1, file\_name);

}

**gener3.cpp**

#include "gener3.h"

int GridGenerator3::GridGen\_pr(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double h\_zmin, double k\_x, double k\_y, double k\_z, Point \*&masGrid){

int n\_x, n\_y, n\_z; //количества узлов

double\* Xm; double\* Ym; double\* Zm; //координаты

double ax = left\_bottom.x, ay = left\_bottom.y, az = left\_bottom.z;

double bx = right\_top.x, by = right\_top.y, bz = right\_top.z;

GridGenerator::GridGen(ax, bx, h\_xmin, k\_x, n\_x, Xm);

GridGenerator::GridGen(ay, by, h\_ymin, k\_y, n\_y, Ym);

GridGenerator::GridGen(az, bz, h\_zmin, k\_z, n\_z, Zm);

int n\_xy = n\_x \* n\_y;

int n = n\_xy \* n\_z;

masGrid = new Point [n];

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

for(int j = 0; j < n\_y; j++){

for(int k = 0; k < n\_z; k++){

masGrid[k\*n\_xy + j\*n\_x + i].x = Xm[i];

masGrid[k\*n\_xy + j\*n\_x + i].y = Ym[j];

masGrid[k\*n\_xy + j\*n\_x + i].z = Zm[k];

}

}

}

delete[] Xm;

delete[] Ym;

delete[] Zm;

return n;

}

int GridGenerator3::GridGen(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double h\_zmin, double k\_x, double k\_y, double k\_z, std::string file\_name){

double ax = left\_bottom.x;

double ay = left\_bottom.y;

double az = left\_bottom.z;

double bx = right\_top.x;

double by = right\_top.y;

double bz = right\_top.z;

bool check\_x = GridGenerator::IsLegal(ax, bx, h\_xmin, k\_x);

bool check\_y = GridGenerator::IsLegal(ay, by, h\_ymin, k\_y);

bool check\_z = GridGenerator::IsLegal(az, bz, h\_zmin, k\_z);

if(!(check\_x && check\_y && check\_z)) return 0;

Point\* masGrid;//сетка

int n = GridGen\_pr(left\_bottom, right\_top, h\_xmin, h\_ymin, h\_zmin, k\_x, k\_y, k\_z, masGrid);

FILE\* out\_f = fopen(file\_name.c\_str(), "w");

fprintf(out\_f, "%d\n", n);

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

fprintf(out\_f, "%.15lf\t%.15lf\t%.15lf\n", masGrid[i].x, masGrid[i].y, masGrid[i].z);

}

fclose(out\_f);

delete[] masGrid;

return 1;

}

int GridGenerator3::GridGenMain(Point left\_bottom, Point right\_top, double h\_xmin, double h\_ymin, double h\_zmin, double k\_x, double k\_y, double k\_z, string file\_cords, string file\_elements, string file\_faces){

if(!GridGen(left\_bottom, right\_top, h\_xmin, h\_ymin, h\_zmin, k\_x, k\_y, k\_z, file\_cords)) return 0;

double ax = left\_bottom.x;

double ay = left\_bottom.y;

double az = left\_bottom.z;

double bx = right\_top.x;

double by = right\_top.y;

double bz = right\_top.z;

FILE\* out\_f = fopen(file\_elements.c\_str(), "w");

int n\_x = GridGenerator::numberNodes(ax, bx, h\_xmin, k\_x);

int n\_y = GridGenerator::numberNodes(ay, by, h\_ymin, k\_y);

int n\_z = GridGenerator::numberNodes(az, bz, h\_zmin, k\_z);

int n\_xy = n\_x \* n\_y;

fprintf(out\_f, "%d\n", (n\_x-1)\*(n\_y-1)\*(n\_z-1));

for(int i = 0; i < n\_x - 1; i++){

for(int j = 0; j < n\_y - 1; j++){

for(int k = 0; k < n\_z - 1; k++){

int node\_i0 = k\*n\_xy + j\*n\_y + i;

int node\_i1 = k\*n\_xy + j\*n\_y + i+1;

int node\_i2 = k\*n\_xy + (j+1)\*n\_x + i;

int node\_i3 = k\*n\_xy + (j+1)\*n\_x + i + 1;

int node\_i4 = (k+1)\*n\_xy + j\*n\_y + i;

int node\_i5 = (k+1)\*n\_xy + j\*n\_y + i+1;

int node\_i6 = (k+1)\*n\_xy + (j+1)\*n\_x + i;

int node\_i7 = (k+1)\*n\_xy + (j+1)\*n\_x + i + 1;

fprintf(out\_f, "%d %d %d %d %d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3, node\_i4, node\_i5, node\_i6, node\_i7);

}

}

}

fclose(out\_f);

out\_f = fopen(file\_faces.c\_str(), "w");

fprintf(out\_f, "%d\n", 2\*(n\_x-1)\*(n\_y-1) + 2\*(n\_x-1)\*(n\_z-1) + 2\*(n\_y-1)\*(n\_z-1));

for(int i = 0; i < n\_x - 1; i++){

for(int j = 0; j < n\_y - 1; j++){

int node\_i0 = j\*n\_x + i;

int node\_i1 = j\*n\_x + i + 1;

int node\_i2 = (j+1)\*n\_x + i;

int node\_i3 = (j+1)\*n\_x + i + 1;

fprintf(out\_f, "%d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3);

}

}

for(int i = 0; i < n\_x - 1; i++){

for(int j = 0; j < n\_y - 1; j++){

int node\_i0 = (n\_z-1)\*n\_xy + j\*n\_x + i;

int node\_i1 = (n\_z-1)\*n\_xy + j\*n\_x + i + 1;

int node\_i2 = (n\_z-1)\*n\_xy + (j+1)\*n\_x + i;

int node\_i3 = (n\_z-1)\*n\_xy + (j+1)\*n\_x + i + 1;

fprintf(out\_f, "%d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3);

}

}

for(int i = 0; i < n\_x - 1; i++){

for(int k = 0; k < n\_z - 1; k++){

int node\_i0 = k\*n\_xy + i;

int node\_i1 = k\*n\_xy + i + 1;

int node\_i2 = (k+1)\*n\_xy + i;

int node\_i3 = (k+1)\*n\_xy + i + 1;

fprintf(out\_f, "%d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3);

}

}

for(int i = 0; i < n\_x - 1; i++){

for(int k = 0; k < n\_z - 1; k++){

int node\_i0 = k\*n\_xy + (n\_y-1)\*n\_x + i;

int node\_i1 = k\*n\_xy + (n\_y-1)\*n\_x + i + 1;

int node\_i2 = (k+1)\*n\_xy + (n\_y-1)\*n\_x + i;

int node\_i3 = (k+1)\*n\_xy + (n\_y-1)\*n\_x + i + 1;

fprintf(out\_f, "%d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3);

}

}

for(int j = 0; j < n\_y - 1; j++){

for(int k = 0; k < n\_z - 1; k++){

int node\_i0 = k\*n\_xy + j\*n\_x;

int node\_i1 = k\*n\_xy + (j+1)\*n\_x;

int node\_i2 = (k+1)\*n\_xy + j\*n\_x;

int node\_i3 = (k+1)\*n\_xy + (j+1)\*n\_x;

fprintf(out\_f, "%d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3);

}

}

for(int j = 0; j < n\_y - 1; j++){

for(int k = 0; k < n\_z - 1; k++){

int node\_i0 = k\*n\_xy + j\*n\_x + n\_x-1;

int node\_i1 = k\*n\_xy + (j+1)\*n\_x + n\_x-1;

int node\_i2 = (k+1)\*n\_xy + j\*n\_x + n\_x-1;

int node\_i3 = (k+1)\*n\_xy + (j+1)\*n\_x + n\_x-1;

fprintf(out\_f, "%d %d %d %d\n", node\_i0, node\_i1, node\_i2, node\_i3);

}

}

fclose(out\_f);

return 1;

}

**LOS.h**

#ifndef LOS\_HMFE\_H\_

#define LOS\_HMFE\_H\_

#include <math.h>

class LOS{

public:

void init(int\* s\_ig, int\* s\_jg, double\* s\_gu, double\* s\_gl, double\* s\_di, int s\_n);

void setF(double\* s\_rp);

void SolutionSLAE(double \*&solution, int &its);

private:

int n;

int \*ig, \*jg;

double \*gu, \*gl, \*di, \*rp, \*Uu, \*Ll, \*Ld;

void precond();

double dot\_prod(double \*a, double \*b);

void mull\_A(double \*f, double \*&x);

void SolutionSLAE\_L(double \*f, double \*&x);

void SolutionSLAE\_U(double \*f, double \*&x);

};

#endif

**LOS.cpp**

#include "LOS.h"

void LOS::init(int \*s\_ig, int \*s\_jg, double \*s\_gu, double \*s\_gl, double \*s\_di, int s\_n){

ig = s\_ig;

jg = s\_jg;

gu = s\_gu;

gl = s\_gl;

di = s\_di;

n = s\_n;

precond();

}

void LOS::setF(double \*s\_rp){rp = s\_rp;}

void LOS::precond(){

double sum\_l, sum\_u, sum\_d;

int copy\_end = ig[n];

Ll = new double [copy\_end];

Uu = new double [copy\_end];

Ld = new double [n];

for(int i = 0; i < copy\_end; i++){Ll[i] = gl[i]; Uu[i] = gu[i];}

for(int i = 0; i < n; i++)Ld[i] = di[i];

for(int k = 1, k1 = 0; k <= n; k++, k1++){

sum\_d = 0;

int i\_s = ig[k1], i\_e = ig[k];

for(int m = i\_s; m < i\_e; m++){

sum\_l = 0; sum\_u = 0;

int j\_s = ig[jg[m]], j\_e = ig[jg[m]+1];

for(int i = i\_s; i < m; i++){

for(int j = j\_s ; j < j\_e; j++){

if(jg[i] == jg[j]){

sum\_l += Ll[i]\*Uu[j];

sum\_u += Ll[j]\*Uu[i];

j\_s++;

}

}

}

Ll[m] = Ll[m] - sum\_l;

Uu[m] = (Uu[m] - sum\_u) / Ld[jg[m]];

sum\_d += Ll[m]\*Uu[m];

}

Ld[k1] = Ld[k1] - sum\_d;

}

}

double LOS::dot\_prod(double \*a, double \*b){

double dp = 0;

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

dp += a[i]\*b[i];

return dp;

}

void LOS::mull\_A(double \*f, double \*&x){

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

double v\_el = f[i];

x[i] = di[i]\*v\_el;

for(int k = ig[i], k1 = ig[i+1]; k < k1; k++){

int j = jg[k];

x[i] += gl[k]\*f[j];

x[j] += gu[k]\*v\_el;

}

}

}

void LOS::SolutionSLAE\_L(double \*f, double \*&x){

for(int k = 1, k1 = 0; k <= n; k++, k1++){

double sum = 0;

for(int i = ig[k1]; i < ig[k]; i++)sum += Ll[i]\*x[jg[i]];

x[k1] = (f[k1] - sum)/Ld[k1];

}

}

void LOS::SolutionSLAE\_U(double \*f, double \*&x){

double\* f1 = new double [n];

for(int i = 0; i < n; i++)f1[i] = f[i];

for(int k = n, k1 = n-1; k > 0; k--, k1--){

x[k1] = f1[k1]/Ld[k1];

double v\_el = x[k1];

for(int i = ig[k1]; i < ig[k]; i++)f1[jg[i]] -= Uu[i]\*v\_el;

}

delete[] f1;

}

void LOS::SolutionSLAE(double \*&solution, int &its){

int max\_iter = 100000;

double eps = 1E-16;

double end\_cycle = 0;

double rp\_norm = sqrt(dot\_prod(rp, rp));

double\* x0 = new double [n];

for(int i = 0; i < n; i++)x0[i] = 0;

solution = new double [n];

double\* r = new double [n];

double\* z = new double [n];

double\* p = new double [n];

double\* s = new double [n];

double\* t = new double [n];

//r0 = L^(-1) \* (f - Ax0)

mull\_A(x0, s);

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

s[i] = rp[i] - s[i];

SolutionSLAE\_L(s, r);

//z0 = U^(-1)r0

SolutionSLAE\_U(r, z);

//p0 = L^(-1)Az0

mull\_A(z, s);

SolutionSLAE\_L(s, p);

int iter;

for(iter = 0; iter < max\_iter && !end\_cycle; iter++){

double discr = sqrt(dot\_prod(r, r));

if( discr / rp\_norm > eps){

double dot1 = dot\_prod(p, p); //(p[k-1], p[k-1])

double alpha = dot\_prod(p ,r) / dot1; //a = (p[k-1], r[k-1]) / (p[k-1], p[k-1])

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

x0[i] = x0[i] + alpha\*z[i]; //x[k] = x[k-1] + a\*z[k-1]

r[i] = r[i] - alpha\*p[i]; //r[k] = r[k-1] - a\*p[k-1]

}

//betta = -(p[k-1], L^(-1)\*A\*U^(-1)r[k]) / (p[k-1], p[k-1])

SolutionSLAE\_U(r, s); // s = U^(-1)r[k]

mull\_A(s, t);

SolutionSLAE\_L(t, t);

double betta = - dot\_prod(p, t) / dot1;

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

z[i] = s[i] + betta \* z[i]; // z[k] = U^(-1)r[k] + b\*z[k-1]

p[i] = t[i] + betta \* p[i]; // p[k] = L^(-1)\*A\*U^(-1)r[k] + b\*p[k-1]

}

if(iter % n == 0){ //Обновление метода

//r0 = L^(-1) \* (f - Ax0)

mull\_A(x0, s);

for(int i = 0; i < n; i++)s[i] = rp[i] - s[i];

SolutionSLAE\_L(s, r);

//z0 = U^(-1)r0

SolutionSLAE\_U(r, z);

//p0 = L^(-1)Az0

mull\_A(z, s);

SolutionSLAE\_L(s, p);

}

}

else

end\_cycle = true;

}

for(int i = 0 ; i < n; i++)solution[i] = x0[i];

its = iter;

delete[] x0;

delete[] p;

delete[] r;

delete[] z;

delete[] s;

delete[] t;

}

**LU.h**

#ifndef LU\_HMFE\_H\_

#define LU\_HMFE\_H\_

#include <math.h>

class SLAEsolver\_LU{

public:

void init(int\* s\_ig, int\* s\_jg, double\* s\_gu, double\* s\_gl, double\* s\_di, int s\_n);

void setF(double\* s\_rp);

void SolutionSLAE(double \*&solution, int &its);

private:

int n;

int \*ig;

double \*gl, \*gu, \*di, \*rp;

void dec();

};

#endif

**LU.cpp**

#include "LU.h"

void SLAEsolver\_LU::init(int\* s\_ig, int\* s\_jg, double\* s\_gu, double\* s\_gl, double\* s\_di, int s\_n){

n = s\_n;

di = new double [n];

ig = new int [n+1];

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

di[i] = s\_di[i];

ig[0] = 0;

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

int k = s\_ig[i] - s\_ig[i-1];

if(k > 0){

int total\_n = i - s\_jg[s\_ig[i-1]];

ig[i] = ig[i-1] + total\_n;

}

else

ig[i] = ig[i-1];

}

gu = new double [ig[n]];

gl = new double [ig[n]];

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

int j\_s = ig[i];

int j\_e = ig[i+1];

int column = i - (j\_e - j\_s);

int s\_point = s\_ig[i];

for(int j = j\_s; j < j\_e; j++, column++){

if(column == s\_jg[s\_point]){

gu[j] = s\_gu[s\_point];

gl[j] = s\_gl[s\_point];

s\_point++;

}

else

gu[j] = gl[j] = 0;

}

}

dec();

}

void SLAEsolver\_LU::setF(double \*s\_rp){

rp = new double [n];

for(int i = 0; i < n; i++)rp[i] = s\_rp[i];

}

void SLAEsolver\_LU::SolutionSLAE(double \*&solution, int &its){

solution = new double [n];

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

double sum = 0;

int j\_start = ig[i], j\_end = ig[i+1];

int vect\_iter = i - (j\_end - j\_start);

for(int j = j\_start; j < j\_end; j++, vect\_iter++)sum += gl[j]\*rp[vect\_iter];

rp[i] = (rp[i] - sum)/di[i];

}

for(int i = n-1; i >= 0; i--){

int j\_start = ig[i], j\_end = ig[i+1];

int vect\_iter = i - (j\_end - j\_start);

for(int j = j\_start; j<j\_end; j++, vect\_iter++)

rp[vect\_iter] -= gu[j]\*rp[i];

}

for(int i = 0; i < n; i++)solution[i] = rp[i];

its = -1;

}

void SLAEsolver\_LU::dec(){

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

int i0 = ig[i];

int i1 = ig[i+1];

int j = i - (i1-i0);

double sd = 0;

for(int m = i0; m < i1; m++,j++){

double sl = 0;

double su = 0;

int j0 = ig[j];

int j1 = ig[j+1];

int mi = i0;

int mj = j0;

int kol\_i = m - i0;

int kol\_j = j1 - j0;

int kol\_r = kol\_i - kol\_j;

if(kol\_r < 0) mj -= kol\_r;

else mi += kol\_r;

for(mi=mi; mi<m; mi++,mj++){

sl += gl[mi]\*gu[mj];

su += gu[mi]\*gl[mj];

}

gl[m] = gl[m] - sl;

gu[m] = (gu[m] - su ) / di[j];

sd += gl[m]\*gu[m];

}

di[i] = di[i] - sd;

}

}

**method.h**

#ifndef HMFE\_H\_

#define HMFE\_H\_

#include <stdio.h>

#include <string>

#include <windows.h>

#include "objects.h"

#include "LOS.h"

#include "LU.h"

using namespace std;

typedef double(\*func)(double, double, double);

template <class SLAEsolver\_type> class HFEM{

private:

int n\_elem;

int n\_nodes;

int n\_faces1, n\_faces2, n\_faces3;

node\* nodes;

dCube\* elements;

face \*faces\_sec, \*faces\_thi;

double \*faces\_fir;

int \*n\_faces1ode;

double u\_betta\_s(double x, double y, double z, int face\_n);

double u\_betta\_с(double x, double y, double z, int face\_n);

double\* betta;

double tetta\_s(double x, double y, double z, int face\_n);

double tetta\_c(double x, double y, double z, int face\_n);

func lambda, sigma, hi;

func f\_sin, f\_cos;

double w;

SLAEsolver\_type SLAEsolver;

int SLAEsolver\_iters;

CPortraitGener port\_gen;

double time;

int \*ig, \*jg;

int SLAE\_el\_n;

double \*gl, \*gu, \*di;

double \*right\_part;

double\* solution;

func u\_sin, u\_cos;

void formigjg();

int findNumber(int i, int j);

void LocalMatrix(double \*\*A\_loc, double \*b\_loc, int el\_n);

void Local3(double \*\*A\_loc, double \*b\_loc, int face\_n);

void Local2(double \*b\_loc, int face\_n);

int mu(int i);

int nu(int i);

int v(int i);

public:

void GridT(string file\_cords, string file\_elements, string file\_faces);

void init(string file\_cords, string file\_elements, string file\_faces);

void setCf(func set\_lambda, func set\_sigma, func set\_hi);

void setF(func set\_f\_sin, func set\_f\_cos);

void setOmega(double s\_w);

void MatrixForming();

void SolutionSLAE();

void outputResult(string file\_name);

void outputDiffer(string file\_name);

void set\_sol(func us, func uc);

};

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::init(string file\_cords, string file\_elements, string file\_faces){

FILE\* inCoord = fopen(file\_cords.c\_str(), "r");

FILE\* inEls = fopen(file\_elements.c\_str(), "r");

FILE\* inFace = fopen(file\_faces.c\_str(), "r");

fscanf(inCoord, "%d", &n\_nodes);

nodes = new node [n\_nodes];

for(int i = 0; i < n\_nodes; i++)fscanf(inCoord, "%lf %lf %lf", &nodes[i].x, &nodes[i].y, &nodes[i].z);

fclose(inCoord);

fscanf(inEls, "%d", &n\_elem);

elements = new dCube [n\_elem];

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

for(int j = 0; j < 16; j++)fscanf(inEls, "%d", &elements[i].node\_n[j]);

}

fclose(inEls);

fscanf(inFace, "%d", &n\_faces1);

faces\_fir = new double [n\_faces1];

n\_faces1ode = new int [n\_faces1];

for(int i = 0; i < n\_faces1; i++)fscanf(inFace, "%d %lf", &n\_faces1ode[i], &faces\_fir[i]);

fscanf(inFace, "%d", &n\_faces2);

faces\_sec = new face [n\_faces2];

for(int i = 0; i < n\_faces2; i++){

for(int j = 0; j < 8; j++)fscanf(inFace, "%d", &faces\_sec[i].node\_n[j]);

}

fscanf(inFace, "%d", &n\_faces3);

faces\_thi = new face [n\_faces3];

for(int i = 0; i < n\_faces3; i++){

for(int j = 0; j < 8; j++)fscanf(inFace, "%d", &faces\_thi[i].node\_n[j]);

}

fclose(inFace);

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::outputDiffer(string file\_name){

FILE\* out\_f = fopen(file\_name.c\_str(), "w");

double diff1 = 0;

double u\_norm = 0;

double diff\_s = 0, diff\_c = 0;

double u\_ns = 0, u\_nc = 0;

for(int i = 0; i < n\_nodes/2; i++){

double x = nodes[2\*i].x;

double y = nodes[2\*i].y;

double z = nodes[2\*i].z;

double us = u\_sin(x,y,z);

double uc = u\_cos(x,y,z);

diff\_s = diff\_s + (us - solution[2\*i])\*(us - solution[2\*i]);

diff\_c = diff\_c + (uc - solution[2\*i+1])\*(uc - solution[2\*i+1]);

u\_ns = u\_ns + us\*us;

u\_nc = u\_nc + uc\*uc;

}

diff1 = diff\_s + diff\_c;

u\_norm = u\_ns + u\_nc;

fprintf(out\_f, "Total:\t%.3e\n", sqrt(diff1/u\_norm));

fprintf(out\_f, "Sin:\t%.3e\n", sqrt(diff\_s/u\_ns));

fprintf(out\_f, "Cos:\t%.3e\n", sqrt(diff\_c/u\_nc));

fprintf(out\_f, "Iters:\t%d\n", SLAEsolver\_iters);

fprintf(out\_f, "Time:\t%lf\n", time);

fclose(out\_f);

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::setCf(func set\_lambda, func set\_sigma, func set\_hi){

lambda = set\_lambda; sigma = set\_sigma; hi = set\_hi;}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::setF(func set\_f\_sin, func set\_f\_cos){

f\_sin = set\_f\_sin; f\_cos = set\_f\_cos;}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::setOmega(double s\_w){w = s\_w;}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::SolutionSLAE(){

LARGE\_INTEGER start, stop, timetime, fr;

QueryPerformanceFrequency(&fr);

QueryPerformanceCounter(&start);

SLAEsolver.init(ig, jg, gu, gl, di, n\_nodes);

SLAEsolver.setF(right\_part);

SLAEsolver.SolutionSLAE(solution, SLAEsolver\_iters);

QueryPerformanceCounter(&stop);

timetime.QuadPart = stop.QuadPart - start.QuadPart;

time = (double)timetime.QuadPart / (double)fr.QuadPart;

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::formigjg(){

port\_gen.init(n\_nodes);

ig = new int [n\_nodes+1];

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

port\_gen.ElemAddition(elements[i]);

port\_gen.gen(ig, jg, SLAE\_el\_n);

gl = new double [SLAE\_el\_n];

gu = new double [SLAE\_el\_n];

di = new double [n\_nodes];

right\_part = new double [n\_nodes];

solution = new double [n\_nodes];

for(int i = 0; i < SLAE\_el\_n; i++)gl[i] = gu[i] = 0;

for(int i = 0; i < n\_nodes; i++)di[i] = right\_part[i] = solution[i] = 0;

port\_gen.~CPortraitGener();

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::MatrixForming(){

double \*\*A\_loc, \*b\_loc;

b\_loc = new double [16];

A\_loc = new double\* [16];

for(int i = 0; i < 16; i++)A\_loc[i] = new double [16];

formigjg();

int cur\_row;

int pos;

for(int k = 0; k < n\_elem; k++){

LocalMatrix(A\_loc, b\_loc, k);

for(int i = 0; i < 16; i++){

cur\_row = elements[k].node\_n[i];

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

if(cur\_row > elements[k].node\_n[j]){

pos = findNumber(cur\_row, elements[k].node\_n[j]);

gl[pos] = gl[pos] + A\_loc[i][j];

gu[pos] = gu[pos] + A\_loc[j][i];

}

else{

pos = findNumber(elements[k].node\_n[j], cur\_row); //Находим позицию в gu и gl

gu[pos] = gu[pos] + A\_loc[i][j];

gl[pos] = gl[pos] + A\_loc[j][i];

}

}

di[cur\_row] += A\_loc[i][i];

right\_part[cur\_row] += b\_loc[i];

}

}

for(int k = 0; k < n\_faces3; k++){

Local3(A\_loc, b\_loc, k);

for(int i = 0; i < 8; i++){

cur\_row = faces\_thi[k].node\_n[i];

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

if(cur\_row > faces\_thi[k].node\_n[j]){ //Если элементы содержаться в строке

pos = findNumber(cur\_row, faces\_thi[k].node\_n[j]); //Находим позицию в gu и gl

gl[pos] += A\_loc[i][j];

gu[pos] += A\_loc[j][i];

}

else{

pos = findNumber(faces\_thi[k].node\_n[j], cur\_row); //Находим позицию в gu и gl

gu[pos] += A\_loc[i][j];

gl[pos] += A\_loc[j][i];

}

}

di[cur\_row] += A\_loc[i][i];

right\_part[cur\_row] += b\_loc[i];

}

}

for(int k = 0; k < n\_faces2; k++){

Local2(b\_loc, k);

for(int i = 0; i < 8; i++)right\_part[faces\_sec[k].node\_n[i]] += b\_loc[i];

}

for(int k = 0; k < n\_faces1; k++){

cur\_row = n\_faces1ode[k];

double val = faces\_fir[k];

di[cur\_row] = 1;

right\_part[cur\_row] = val;

int i\_s = ig[cur\_row], i\_e = ig[cur\_row+1];

for(int i = i\_s; i < i\_e; i++){

right\_part[jg[i]] -= gu[i]\*val;

gl[i] = 0;

gu[i] = 0;

}

for(int p = cur\_row + 1; p < n\_nodes; p++){

int i\_s = ig[p], i\_e = ig[p+1];

for(int i = i\_s; i < i\_e; i++){

if(jg[i] == cur\_row){

right\_part[p] -= gl[i]\*val;

gl[i] = 0;

gu[i] = 0;

}

}

}

}

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::LocalMatrix(double \*\*A\_loc, double \*b\_loc, int el\_n){

double G1[2][2] = {{1.0, -1.0}, {-1.0, 1.0}};

double M1[2][2] = {{1.0/3.0, 1.0/6.0}, {1.0/6.0, 1.0/3.0}};

double G[8][8], M[8][8];

double hx = fabs(nodes[elements[el\_n].node\_n[2]].x - nodes[elements[el\_n].node\_n[0]].x);

double hy = fabs(nodes[elements[el\_n].node\_n[4]].y - nodes[elements[el\_n].node\_n[0]].y);

double hz = fabs(nodes[elements[el\_n].node\_n[8]].z - nodes[elements[el\_n].node\_n[0]].z);

double lambdaA = 0;

double sigmaA = 0;

double hiA = 0;

for(int i = 0; i < 8; i++){

double x = nodes[elements[el\_n].node\_n[2\*i]].x;

double y = nodes[elements[el\_n].node\_n[2\*i]].y;

double z = nodes[elements[el\_n].node\_n[2\*i]].z;

lambdaA += lambda(x,y,z);

sigmaA += sigma(x,y,z);

hiA += hi(x, y, z);

}

lambdaA = lambdaA / 8.0;

sigmaA = sigmaA / 8.0;

hiA = hiA / 8.0;

for(int i = 0; i < 8; i++){

for(int j = 0; j < 8; j++){

G[i][j] = hy\*hz \* G1[mu(i)][mu(j)] \* M1[nu(i)][nu(j)] \* M1[v(i)][v(j)] / hx;

G[i][j] += hx\*hz \* M1[mu(i)][mu(j)] \* G1[nu(i)][nu(j)] \* M1[v(i)][v(j)] / hy;

G[i][j] += hx\*hy \* M1[mu(i)][mu(j)] \* M1[nu(i)][nu(j)] \* G1[v(i)][v(j)] / hz;

G[i][j] \*= lambdaA;

M[i][j] = hx\*hy\*hz \* M1[mu(i)][mu(j)] \* M1[nu(i)][nu(j)] \* M1[v(i)][v(j)];

}

}

for(int i = 0; i < 8; i++){

for(int j = 0; j < 8; j++){

A\_loc[2\*i+1][2\*j+1] = A\_loc[2\*i][2\*j] = G[i][j] - w\*w\*hiA\*M[i][j];

A\_loc[2\*i][2\*j+1] = -w\*sigmaA\*M[i][j];

A\_loc[2\*i+1][2\*j] = w\*sigmaA\*M[i][j];

}

}

double val\_f\_sin[8], val\_f\_cos[8], b\_sin[8], b\_cos[8];

for(int i = 0; i < 8; i++){

double x = nodes[elements[el\_n].node\_n[2\*i]].x;

double y = nodes[elements[el\_n].node\_n[2\*i]].y;

double z = nodes[elements[el\_n].node\_n[2\*i]].z;

val\_f\_sin[i] = f\_sin(x,y,z);

val\_f\_cos[i] = f\_cos(x,y,z);

}

for(int i = 0; i < 8; i++){

b\_sin[i] = b\_cos[i] = 0;

for(int j = 0; j < 8; j++){

b\_sin[i] += M[i][j]\*val\_f\_sin[j];

b\_cos[i] += M[i][j]\*val\_f\_cos[j];

}

}

for(int i = 0; i < 8; i++){

b\_loc[2\*i] = b\_sin[i];

b\_loc[2\*i+1] = b\_cos[i];

}

double vec2[16], vals[16];

for(int i = 0; i < 8; i++){

double x = nodes[elements[el\_n].node\_n[2\*i]].x;

double y = nodes[elements[el\_n].node\_n[2\*i]].y;

double z = nodes[elements[el\_n].node\_n[2\*i]].z;

vals[2\*i] = u\_sin(x,y,z);

vals[2\*i+1] = u\_cos(x,y,z);

}

for(int i = 0; i < 16; i++){

vec2[i] = 0;

for(int j = 0; j < 16; j++)

vec2[i] += A\_loc[i][j]\*vals[j];

}

double diff[16];

for(int i = 0; i < 16; i++)

diff[i] = b\_loc[i] - vec2[i];

}

template <typename SLAEsolver\_type> int HFEM<SLAEsolver\_type>::mu(int i){return i % 2;}

template <typename SLAEsolver\_type> int HFEM<SLAEsolver\_type>::nu(int i){return (i/2) % 2;}

template <typename SLAEsolver\_type> int HFEM<SLAEsolver\_type>::v(int i){return (i/4) % 2;}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::Local3(double \*\*A\_loc, double \*b\_loc, int face\_n){

double hx, hy;

if(nodes[faces\_thi[face\_n].node\_n[2]].x == nodes[faces\_thi[face\_n].node\_n[0]].x){

hx = fabs(nodes[faces\_thi[face\_n].node\_n[2]].y - nodes[faces\_thi[face\_n].node\_n[0]].y);

hy = fabs(nodes[faces\_thi[face\_n].node\_n[4]].z - nodes[faces\_thi[face\_n].node\_n[0]].z);

}

else{

if(nodes[faces\_thi[face\_n].node\_n[2]].y == nodes[faces\_thi[face\_n].node\_n[0]].y){

hx = fabs(nodes[faces\_thi[face\_n].node\_n[2]].x - nodes[faces\_thi[face\_n].node\_n[0]].x);

hy = fabs(nodes[faces\_thi[face\_n].node\_n[4]].z - nodes[faces\_thi[face\_n].node\_n[0]].z);

}

else{

hx = fabs(nodes[faces\_thi[face\_n].node\_n[2]].x - nodes[faces\_thi[face\_n].node\_n[0]].x);

hy = fabs(nodes[faces\_thi[face\_n].node\_n[4]].y - nodes[faces\_thi[face\_n].node\_n[0]].y);

}

}

double M1[4][4] = {{4, 2, 2, 1}, {2, 4, 1, 2}, {2, 1, 4 ,2}, {1, 2, 2, 4}};

double loc\_betta = betta[faces\_thi[face\_n].area];

for(int i = 0; i < 4; i++){

for(int j = 0; j < 4; j++){

A\_loc[2\*i][2\*i] = A\_loc[2\*j+1][2\*j+1] = loc\_betta\*hx\*hy\*M1[i][i]/36.0;

A\_loc[2\*i+1][2\*j] = A\_loc[2\*i][2\*j+1] = 0;

}

}

double b\_both[8];

for(int i = 0; i < 4; i++){

double x = nodes[faces\_thi[face\_n].node\_n[2\*i]].x;

double y = nodes[faces\_thi[face\_n].node\_n[2\*i]].y;

double z = nodes[faces\_thi[face\_n].node\_n[2\*i]].z;

b\_both[2\*i] = u\_betta\_s(x, y, z, face\_n);

b\_both[2\*i+1] = u\_betta\_с(x, y, z, face\_n);

}

for(int i = 0; i < 8; i++){

b\_loc[i] = 0;

for(int j = 0; j < 8; j++){

b\_loc[i] += A\_loc[i][j]\*b\_both[j];

}

}

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::Local2(double \*b\_loc, int face\_n){

double hx,hy;

double A\_loc[8][8];

if(nodes[faces\_thi[face\_n].node\_n[2]].x == nodes[faces\_thi[face\_n].node\_n[0]].x){ //Если в плоскости yOz

hx = fabs(nodes[faces\_thi[face\_n].node\_n[2]].y - nodes[faces\_thi[face\_n].node\_n[0]].y);

hy = fabs(nodes[faces\_thi[face\_n].node\_n[4]].z - nodes[faces\_thi[face\_n].node\_n[0]].z);

}

else{

if(nodes[faces\_thi[face\_n].node\_n[2]].y == nodes[faces\_thi[face\_n].node\_n[0]].y){ //Если в плоскости xOz

hx = fabs(nodes[faces\_thi[face\_n].node\_n[2]].x - nodes[faces\_thi[face\_n].node\_n[0]].x);

hy = fabs(nodes[faces\_thi[face\_n].node\_n[4]].z - nodes[faces\_thi[face\_n].node\_n[0]].z);

}

else{

hx = fabs(nodes[faces\_thi[face\_n].node\_n[2]].x - nodes[faces\_thi[face\_n].node\_n[0]].x);

hy = fabs(nodes[faces\_thi[face\_n].node\_n[4]].y - nodes[faces\_thi[face\_n].node\_n[0]].y);

}

}

double M1[4][4] = {{4, 2, 2, 1}, {2, 4, 1, 2}, {2, 1, 4 ,2}, {1, 2, 2, 4}};

for(int i = 0; i < 4; i++){

for(int j = 0; j < 4; j++){

A\_loc[2\*i][2\*i] = A\_loc[2\*j+1][2\*j+1] = hx\*hy\*M1[i][i]/36.0;

A\_loc[2\*i+1][2\*j] = A\_loc[2\*i][2\*j+1] = 0;

}

}

double b\_both[8];

for(int i = 0; i < 4; i++){

double x = nodes[faces\_thi[face\_n].node\_n[2\*i]].x;

double y = nodes[faces\_thi[face\_n].node\_n[2\*i]].y;

double z = nodes[faces\_thi[face\_n].node\_n[2\*i]].z;

b\_both[2\*i] = tetta\_s(x, y, z, face\_n);

b\_both[2\*i+1] = tetta\_c(x, y, z, face\_n);

}

for(int i = 0; i < 8; i++){

b\_loc[i] = 0;

for(int j = 0; j < 8; j++){

b\_loc[i] += A\_loc[i][j]\*b\_both[j];

}

}

}

template <typename SLAEsolver\_type> int HFEM<SLAEsolver\_type>::findNumber(int i, int j){

int k\_s = ig[i], k\_e = ig[i+1];

int cur;

bool find = false;

for(int k = k\_s; k < k\_e && !find; k++){

if(jg[k] == j){

cur = k;

find = true;

}

}

return cur;

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::GridT(string file\_cords, string file\_elements, string file\_faces){

FILE\* cords\_f\_in = fopen(file\_cords.c\_str(), "r");

FILE\* cords\_f\_out = fopen((file\_cords + "1").c\_str(), "w");

int n;

fscanf(cords\_f\_in, "%d", &n);

fprintf(cords\_f\_out, "%d\n", 2\*n);

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

double x, y, z;

fscanf(cords\_f\_in, "%lf %lf %lf",&x, &y, &z);

fprintf(cords\_f\_out, "%.15lf\t%.15lf\t%.15lf\n", x, y, z);

fprintf(cords\_f\_out, "%.15lf\t%.15lf\t%.15lf\n", x, y, z);

}

fclose(cords\_f\_in);

fclose(cords\_f\_out);

FILE\* els\_f\_in = fopen(file\_elements.c\_str(), "r");

FILE\* els\_f\_out = fopen((file\_elements+"1").c\_str(), "w");

fscanf(els\_f\_in, "%d", &n);

fprintf(els\_f\_out, "%d\n", n);

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

for(int j = 0; j < 8; j++){

int k;

fscanf(els\_f\_in, "%d", &k);

fprintf(els\_f\_out, "%d %d ", 2\*k, 2\*k+1);

}

fprintf(els\_f\_out, "\n");

}

fclose(els\_f\_in);

fclose(els\_f\_out);

FILE\* face\_f\_in = fopen(file\_faces.c\_str(), "r");

FILE\* face\_f\_out = fopen((file\_faces+"1").c\_str(), "w");

fscanf(face\_f\_in, "%d", &n);

fprintf(face\_f\_out, "%d\n", n);

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

for(int j = 0; j < 4; j++){

int k;

fscanf(face\_f\_in, "%d", &k);

fprintf(face\_f\_out, "%d %d ", 2\*k, 2\*k+1);

}

fprintf(face\_f\_out, "\n");

}

fclose(face\_f\_in);

fclose(face\_f\_out);

}

template <typename SLAEsolver\_type> double HFEM<SLAEsolver\_type>::u\_betta\_s(double x, double y, double z, int face\_n){

double val = 0;

return val;

}

template <typename SLAEsolver\_type> double HFEM<SLAEsolver\_type>::u\_betta\_с(double x, double y, double z, int face\_n){

double val = 0;

return val;

}

template <typename SLAEsolver\_type> double HFEM<SLAEsolver\_type>::tetta\_s(double x, double y, double z, int face\_n){

double val = 0;

return val;

}

template <typename SLAEsolver\_type> double HFEM<SLAEsolver\_type>::tetta\_c(double x, double y, double z, int face\_n){

double val = 0;

return val;

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::outputResult(string file\_name){

FILE\* out\_f = fopen(file\_name.c\_str(), "w");

for(int i = 0; i < n\_nodes; i++)fprintf(out\_f, "%d\t%.15lf\n", i, solution[i]);

fclose(out\_f);

}

template <typename SLAEsolver\_type> void HFEM<SLAEsolver\_type>::set\_sol(func us, func uc){

u\_sin = us; u\_cos = uc;

}

#endif

**main.cpp**

#include "method.h"

#include "gener3.h"

#include <set>

#include <math.h>

using namespace std;

void trans\_face(string file\_cord, string file\_face);

namespace test2{

double omega = 1000.;

double lambda(double x, double y, double z){ return 1000;}

double sigma(double x, double y, double z) {return 3;}

double hi(double x, double y, double z) {return 1E-11;}

double us(double x, double y, double z) { return x\*x\*x+y\*y\*y+z\*z\*z; }

double uc(double x, double y, double z) { return 2\*x\*x\*x-y\*y\*y+3\*z\*z\*z; }

double fs(double x, double y, double z){ return -omega\*omega\*hi(x,y,z)\*us(x,y,z) - omega\*sigma(x,y,z)\*uc(x,y,z) - 6\*lambda(x,y,z)\*(x+y+z);}

double fc(double x, double y, double z){ return -omega\*omega\*hi(x,y,z)\*uc(x,y,z) + omega\*sigma(x,y,z)\*us(x,y,z) - 6\*lambda(x,y,z)\*(2\*x-y+3\*z);}

}

namespace test3{

double omega = 1000.;

double lambda(double x, double y, double z){ return 1000;}

double sigma(double x, double y, double z) {return 3;}

double hi(double x, double y, double z) {return 1E-11;}

double us(double x, double y, double z) { return x+y+z; }

double uc(double x, double y, double z) { return 2\*x-y+3\*z; }

double fs(double x, double y, double z){ return -omega\*omega\*hi(x,y,z)\*us(x,y,z) - omega\*sigma(x,y,z)\*uc(x,y,z);}

double fc(double x, double y, double z){ return -omega\*omega\*hi(x,y,z)\*uc(x,y,z) + omega\*sigma(x,y,z)\*us(x,y,z);}

}

namespace test4{

double omega = 1000.;

double lambda(double x, double y, double z){ return 1000;}

double sigma(double x, double y, double z) {return 3;}

double hi(double x, double y, double z) {return 1E-11;}

double us(double x, double y, double z) { return sin(x+y+z); }

double uc(double x, double y, double z) { return sin(x-y-z); }

double fs(double x, double y, double z){ return -omega\*omega\*hi(x,y,z)\*us(x,y,z) - omega\*sigma(x,y,z)\*uc(x,y,z) + 3\*lambda(x,y,z)\*sin(x+y+z);}

double fc(double x, double y, double z){ return -omega\*omega\*hi(x,y,z)\*uc(x,y,z) + omega\*sigma(x,y,z)\*us(x,y,z) + 3\*lambda(-x,y,z)\*sin(x-y-z);}

}

using namespace test2;

int main(){

HFEM<LOS> our\_meth\_LOS;

HFEM<SLAEsolver\_LU> our\_meth\_LU;

string file\_cords = "coord";

string file\_els = "els";

string file\_face = "face";

double ax = 0, ay = 0, az = 0;

double bx = 1, by = 1, bz = 1;

double h = 0.2;

double k = 1.000;

double k1 = sqrt(k);

double h1 = h/(1+k1);

double hx = 0.2, hy = 0.2, hz = 0.2;

double kx = k, ky = k, kz = k;

Point st(ax, ay, az), en(bx, by, bz);

printf("Grid generation\n");

GridGenerator3::GridGenMain(st, en, hx, hy, hz, kx, ky, kz, file\_cords, file\_els, file\_face);

our\_meth\_LOS.GridT(file\_cords, file\_els, file\_face);

trans\_face(file\_cords+"1", file\_face+"1");

printf("LOS\n");

our\_meth\_LOS.init(file\_cords+"1", file\_els+"1", file\_face+"11");

our\_meth\_LOS.set\_sol(us, uc);

our\_meth\_LOS.setCf(lambda, sigma, hi);

our\_meth\_LOS.setOmega(omega);

our\_meth\_LOS.setF(fs, fc);

our\_meth\_LOS.MatrixForming();

our\_meth\_LOS.SolutionSLAE();

our\_meth\_LOS.outputResult("LOS.txt");

our\_meth\_LOS.outputDiffer("LU.txt");

printf("LU\n");

our\_meth\_LU.init(file\_cords+"1", file\_els+"1", file\_face+"11");

our\_meth\_LU.set\_sol(us, uc);

our\_meth\_LU.setCf(lambda, sigma, hi);

our\_meth\_LU.setOmega(omega);

our\_meth\_LU.setF(fs, fc);

our\_meth\_LU.MatrixForming();

our\_meth\_LU.SolutionSLAE();

our\_meth\_LU.outputResult("sol\_LU.txt");

our\_meth\_LU.outputDiffer("diff\_LU.txt");

return 0;

}

void trans\_face(string file\_cord, string file\_face){

FILE\* cords = fopen(file\_cord.c\_str(), "r");

int n;

fscanf(cords, "%d", &n);

node\* nodes = new node [n];

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

fscanf(cords, "%lf %lf %lf", &nodes[i].x, &nodes[i].y, &nodes[i].z);

fclose(cords);

FILE\* faces = fopen(file\_face.c\_str(), "r");

int m;

fscanf(faces, "%d", &m);

set<int> face\_node;

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

for(int j = 0; j < 8; j++){

int k;

fscanf(faces, "%d", &k);

face\_node.insert(k);

}

}

fclose(faces);

faces = fopen((file\_face+"1").c\_str(), "w");

fprintf(faces, "%d\n", face\_node.size());

for(set<int>::iterator it = face\_node.begin(); it != face\_node.end(); it++){

int k = \*it;

double x = nodes[k].x;

double y = nodes[k].y;

double z = nodes[k].z;

if(k%2 == 0)

fprintf(faces, "%d\t%.15lf\n", k , us(x,y,z));

else

fprintf(faces, "%d\t%.15lf\n", k , uc(x,y,z));

}

fprintf(faces, "%d\n%d", 0, 0);

fclose(faces);

face\_node.clear();

delete[] nodes;

}

1. Тесты

Рассчётная область – куб [0,1]x[0,1]x[0,1], шаг 0,2, 125 КЭ

1. us=x3+y3+z3, uc= 2x3-y3+3z3, ω=1000, λ=1000, σ=3, χ=10-11,

fs=- ω σ uc- ω2 χ us-6 λ(x+y+z), fs= ω σ us- ω2 χ uc-6 λ(2x-y+3z)

LU:

Погрешность – 1.536e-016

Время – 0.039235 с

LOS:

Погрешность – 3.235e-016

Время – 0.077221 с

Итераций – 50

1. us=x+y+z, uc= 2x-y+3z, ω=1000, λ=1000, σ=3, χ=10-11,

fs=- ω σ uc- ω2 χ us, fs= ω σ us- ω2 χ uc

LU:

Погрешность – 1.966e-016

Время – 0.039716 с

LOS:

Погрешность – 3.747e-016

Время – 0.079993

Итераций – 54

1. us= sin(x+y+z);, uc= sin(x-y-z), ω=1000, λ=1000, σ=3, χ=10-11,

fs=- ω σ uc- ω2 χ us+3λ sin(x+y+z), fs= ω σ us- ω2 χ uc+3λ sin(x-y-z)

LU:

Погрешность – 1.983e-004

Время – 0.039381 с

LOS:

Погрешность – 1.983e-004

Время – 0.076187

Итераций – 50

1. Исследования на сетке с небольшим числом КЭ

Используется первый тест, расчётная область прежняя, шаг 0,1 по всем осям, число КЭ 1000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | LU, время, с | LOS, iter | LOS, время, с |
| 10-4 | 5 103 | 104 | 10-11 | 2.291 | 136 | 1.237 |
| 102 | 5 103 | 104 | 10-11 | 2.290 | 583 | 4.631 |
| 109 | 5 103 | 104 | 10-11 | 2.288 | 6818 | 51.028 |
| 102 | 102 | 104 | 10-11 | 2.289 | 3168 | 23.733 |
| 102 | 105 | 104 | 10-11 | 2.287 | 344 | 2.778 |
| 102 | 5 103 | 0 | 10-11 | 2.288 | 107 | 1.031 |
| 102 | 5 103 | 106 | 10-11 | 2.294 | 6803 | 51.203 |
| 102 | 5 103 | 104 | 8.81 10-12 | 2.340 | 455 | 3.682 |
| 102 | 5 103 | 104 | 10-10 | 2.284 | 455 | 3.620 |

1. Исследования на сетке с большим числом КЭ

Используется первый тест, расчётная область прежняя, шаги 0.025, 0.025 и 0.05 – 32000 КЭ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | LU, время, с | LOS, iter | LOS, время, с |
| 10-4 | 5 103 | 104 | 10-11 | 303.683 | 5463 | 54.017 |
| 102 | 5 103 | 104 | 10-11 | 301.621 | 5530 | 50.353 |
| 102 | 105 | 104 | 10-11 | 305.761 | 3443 | 32.758 |
| 102 | 5 103 | 0 | 10-11 | 302.253 | 1723 | 18.032 |
| 102 | 5 103 | 104 | 8.81 10-12 | 302.340 | 4593 | 42.956 |
| 102 | 5 103 | 104 | 10-10 | 302.284 | 4519 | 37.210 |