Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новосибирский государственный технический университет»



Кафедра прикладной математики

Курсовая работа по дисциплине   
«Численные методы»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| https://psv4.userapi.com/c848132/u275429460/docs/d16/7a48be4e8485/Logo1.png?extra=xygsIQQwad0cqCKIxugjRIy9kd3JmRBfEKfWxcAvrfIKQ1CgI-Mr8IGFc60Gm-UtH3C0t8N-ws0mwmk1iKBnRdnqgFaqqJOX8cT3bxo_ADKOGBCCZ_VicmoU5cJzeVveeKnmjlfBib7DlanMmJKw6Dk2pw | Факультет: | ПМИ |  |  |
| Студент:  Группа: | Дьячкова М.С.  ПМ-73 |  |  |
| Преподаватель: | Персова М.Г. |  |  |

Новосибирск

2019

**1)Задание**

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

**2)Постановка задачи**

Необходимо решить уравнение

,

заданное в некоторой области Ω с границей  и краевыми условиями:



Будем решать это уравнение методом Галеркина. Запишем невязку в виде:  и потребуем, чтобы она была ортогональна , в смысле скалярного произведения пространства H0  , некоторому пространству пробных функций, т.е.: 

Преобразуем выражение, при помощи формулы Грина, где . Интегралы по границам S3 и S2 можно преобразовать, воспользовавшись краевыми условиями: 

Поскольку на границе S1 краевыми условиями не определяется значение, слагаемое  следует исключить из уравнения, потребовав, чтобы пространство пробных функций содержало только функции, которые бы принимали бы нулевые значения на границе S1 .Поэтому в качестве пространства пробных функций возьмем

Представим в виде линейной комбинации базисных функций функции  и  . Подставив в уравнение, получим строку СЛАУ для qj

.

**3)Базисные функции**

Биквадратичные базисные функции строятся в виде произведений одномерных квадратичных базисных функций координат *x* и *y*. Так если *X1(x)* – квадратичная функция, равная единице при *x=x2i-1* и нулю при *x=x2i+1*, *x=x2i* и *X2(x)* – квадратичная функция, равная единице при *x=x2i* и нулю при *x=x2i-1*, и *x=x2i+1*, *X3(x)* – квадратичная функция, равная единице при x=x*2i+1* и нулю при *x=x2i*, *x=x2i-1*, а *Y1(y)* – квадратичная функция, равная единице при *y=y2i-1* и нулю при *y=y2i+1*, *y=y2i* и *Y2(y)* – квадратичная функция, равная единице при *y=y2i* и нулю при *y=y2i-1*, и *y=y2i+1*, *Y3(y)* – квадратичная функция, равная единице при y=y*2i+1* и нулю при *y=y2i*, *y=y2i-1*, то можно определить шаблонные базисные функции на элементе (0,1):

, ,

, ,

, ,

где ξ *= (x – x2i-1)/hx*, а ζ *= (y – y2j-1)/hy*.

Локальные базисные функции v(*x*,*y*) в этом случае на конечном элементе Ω*ij*=*(x2i-1,x2i+1)×(y2j-1,y2j+1)* имеют следующий вид:

|  |  |  |
| --- | --- | --- |
| 1(*x*,*y*)=*X*1(*x*)*Y*1(*y*), | 2(*x*,*y*)=*X*2(*x*)*Y*1(*y*), | 3(*x*,*y*)=*X*3(*x*)*Y*1(*y*), |
| 4(*x*,*y*)=*X*1(*x*)*Y*2(*y*), | 5(*x*,*y*)=*X*2(*x*)*Y*2(*y*), | 6(*x*,*y*)=*X*3(*x*)*Y*2(*y*), |
| 7(*x*,*y*)=*X*1(*x*)*Y*3(*y*), | 8(*x*,*y*)=*X*2(*x*)*Y*3(*y*), | 9(*x*,*y*)=*X*3(*x*)*Y*3(*y*), |

Теперь займемся формированием глобальной матрицы. Будем хранить её в разреженном формате. Первым делом сформируем портрет матрицы, т.е. массивы ig и jg. Для этого сформируем список смежности узлов друг с другом, с учетом симметричности портрета. Так для i-го узла в его список смежности попадут только те смежные вершины, номера которых больше i. Затем с помощью этих списков формируем массивы ig и jg.Когда портрет сформирован, можно заносить элементы локальных матриц в глобальные. Занесение производим с помощью соответствия локальных и глобальных номеров.

Когда матрица сформирована, далее необходимо учесть краевые условия. Первым делом учитываются естественные краевые условия(второго и третьего рода) а после этого учитываются краевые первого рода. Для учета краевых условий второго рода составим матрицу массы для ребра Г, положим β=1 и умножим матрицу на θ=(θ1,θ2). Делаем добавку локального вектора к глобальному вектору. Аналогично учитываем условия третьего рода. Полученную СЛАУ решаем методом ЛОС с неполной факторизацией. Сетку храним в виде списка узлов и списка координат в отдельных файлах. Краевые условия задаются списком ребер и указанием типа КУ.

**4) Краевые условия**

*Краевые условия первого рода*

Краевые условия первого рода означают то, что нам известно значение решения в узлах на границе S1. Если в узле с номером *i* задано первое краевое условие *ug*, тогда диагональный элемент Aii мы заменяем на большое число например, 10 30, а элемент вектора правой части *Fi* на число ug\*1030. Так как в i-й строчке все внедиагональные элементы очень малы по сравнению с диагональным, то i-е уравнение фактически примет вид *qi\*1030 =ug\*1030*, или *qi=us*, что соответствует первому краевому условию.

*Краевые условия второго рода*

Пусть на ребре Si,j задано краевое условие второго рода. Данное краевое условие вносит вклад только в правую часть СЛАУ.



Где i,j – номера узлов ребра, на котором задано краевое условие.

*Краевые условия третьего рода*

При учете третьих краевых условий формируются локальная матрица и вектор правой части, которые заносятся в СЛАУ аналогично локальной матрицы конечного элемента и локального вектора правой части конечного элемента.

**5)Тесты**

Пусть γ,λ=1.

Заданы первые краевые условия:

u=x, f=x, ug=x

u=xy, f=xy, ug=xy

2.36396e-10

1.89116e-09

1.41837e-09

3.78233e-09

1.18198e-09

0.5

1

1.5

2

2.5

1

2

3

4

5

1.5

3

4.5

6

7.5

2

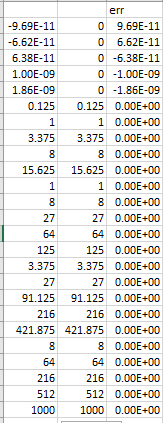
4

6

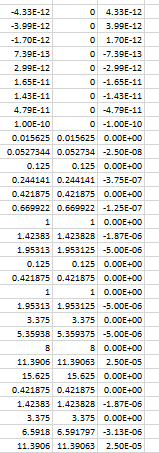
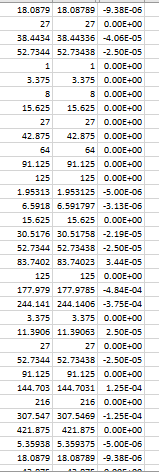
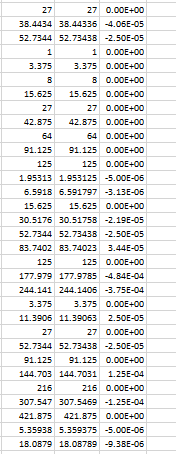
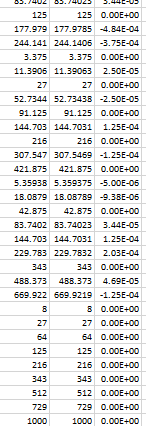
8

10

u=x3\*y3, f= -6 \* x\*y\*y\*y - 6 \* x\*x\*x\*y + x \* x\*x\*y\*y\*y;



Дробление сетки вдвое

**6) Код программы**

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

#include <ctime>

#include <vector>

#include <set>

using namespace std;

class SLAE {

public:

int nx, ny;

int nxw, nyw, l;

int n; //количество конечных элементов

int m; //количество узлов

vector<double> x, y;//прямые КЭ

vector<double> xw, yw; //прямые областей

vector<int> xi, yi;

vector<vector<int>> w; //области и их параметры

double lyam, gamma;

//краевые

int ncond1, ncond2;

vector<vector<int>> cond1;

vector<vector<int>> cond2;

vector<double> local\_cond;

vector<double> local\_cond\_vl;

//матрицы и вектора

vector<vector<double>> G;//локальные матрицы в плотном формате

vector<vector<double>> M;

vector<vector<double>> C;

vector<vector<double>> local;

vector<double> b\_loc;

vector<double> f;//вспомогательный, значения функции в координатах

vector<vector<double>> G\_small, M\_small;

vector<double> x\_loc, y\_loc;//координаты локальных узлов каждого элемента

vector<double> ggl, di,ggu,ggl\_new,di\_new,ggu\_new;

vector<int> ig, jg;

vector<double> b; //глобальная пр часть

vector<double> q; //истинное решение

vector<int> global\_num;

//slae

vector<double> r, p, z, temp1, temp2;

//подумать про оutput файл в конструкторе

SLAE(string net\_file, string area\_file, string cond\_file\_1, string cond\_file\_2) {

read\_net(net\_file);

read\_area(area\_file);

read\_cond(cond\_file\_1, cond\_file\_2);

resize\_func();

}

~SLAE(){

ggl.~vector();

ggl\_new.~vector();

ggu.~vector();

ggu.~vector();

x.~vector();

y.~vector();

xw.~vector();

yw.~vector();

xi.~vector();

yi.~vector();

w.~vector();

cond1.~vector();

cond2.~vector();

local\_cond.~vector();

local\_cond\_vl.~vector();

G.~vector();

M.~vector();

C.~vector();

local.~vector();

b\_loc.~vector();

f.~vector();

G\_small.~vector();

M\_small.~vector();

y\_loc.~vector();

x\_loc.~vector();

di.~vector();

di\_new.~vector();

ig.~vector();

jg.~vector();

b.~vector();

q.~vector();

global\_num.~vector();

r.~vector();

p.~vector();

z.~vector();

temp1.~vector();

temp2.~vector();

}

void read\_net(string net\_file) {

ifstream fin(net\_file + ".txt");

fin >> nx >> ny;

x.resize(nx);

y.resize(ny);

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

fin >> x[i];

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

fin >> y[i];

n = (nx - 1)\*(ny - 1);

m = (2 \* nx - 1)\*(2 \* ny - 1);

fin.close();

}

void read\_area(string area\_file) {

ifstream fin(area\_file + ".txt");

fin >> nxw >> nyw;

xw.resize(nxw);

yw.resize(nyw);

xi.resize(nxw);

yi.resize(nyw);

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

fin >> xw[i];

//поиск по массиву x

int k = 0;

while (x[k] != xw[i])

k++;

xi[i] = k;

}

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

fin >> yw[i];

//поиск по массиву y

int k = 0;

while (y[k] != yw[i])

k++;

yi[i] = k;

}

fin >> l;

w.resize(l);

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

w[i].resize(5);

for (int j = 0; j < 5; j++) {

fin >> w[i][j];

}

}

fin.close();

}

void resize\_func() {

G.resize(9);

M.resize(9);

C.resize(9);

local.resize(9);

for (int i = 0; i < 9; i++) {

G[i].resize(9);

M[i].resize(9);

C[i].resize(9);

local[i].resize(9);

}

b\_loc.resize(9);

f.resize(9);

G\_small.resize(3);

M\_small.resize(3);

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

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

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

}

x\_loc.resize(9);

y\_loc.resize(9);

global\_num.resize(9);

ig.resize(m + 1);

di.resize(m);

di\_new.resize(m);

b.resize(m);

q.resize(m);

local\_cond.resize(3);

local\_cond\_vl.resize(3);

//для Слау

r.resize(m);

z.resize(m);

p.resize(m);

temp1.resize(m);

temp2.resize(m);

}

double func(int num\_area,double x, double y) {

double func\_loc;

switch (num\_area) {

case 1: {

//func\_loc = x;

func\_loc = -6\*x\*y\*y\*y-6\*x\*x\*x\*y+x\*x\*x\*y\*y\*y;

break;

}

case 2: {

//func\_loc = x;

func\_loc = -6 \* x\*y\*y\*y - 6 \* x\*x\*x\*y + x \* x\*x\*y\*y\*y;

break;

}

case 3: {

//func\_loc = x;

func\_loc =- 6 \* x\*y\*y\*y - 6 \* x\*x\*x\*y + x \* x\*x\*y\*y\*y;

break;

}

}

return func\_loc;

}

void write\_small\_matrix() {

G\_small[0][0] = 7;

G\_small[0][1] = -8;

G\_small[0][2] = 1;

G\_small[1][1] = 16;

G\_small[1][2] = -8;

G\_small[2][2] = 7;

M\_small[0][0] = 4;

M\_small[0][1] = 2;

M\_small[0][2] = -1;

M\_small[1][1] = 16;

M\_small[1][2] = 2;

M\_small[2][2] = 4;

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

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

G\_small[i][j] = G\_small[j][i];

M\_small[i][j] = M\_small[j][i];

}

}

void make\_params(int num\_area) {

//явное задание параметров для каждой области

switch (num\_area) {

case 1: {

lyam = 1;

gamma = 1;

break;

}

case 2: {

lyam = 1;

gamma = 1;

break;

}

case 3: {

lyam = 1;

gamma = 1;

break;

}

}

}

void make\_local\_coor(int s, int p) {

double xp, xp\_next, ys, ys\_next, xp\_mid, ys\_mid;

xp = x[p];

xp\_next = x[p + 1];

xp\_mid = (xp + xp\_next) / 2;

ys = y[s];

ys\_next = y[s + 1];

ys\_mid = (ys + ys\_next) / 2;

x\_loc[0] = xp;

x\_loc[1] = xp\_mid;

x\_loc[2] = xp\_next;

x\_loc[3] = xp;

x\_loc[4] = xp\_mid;

x\_loc[5] = xp\_next;

x\_loc[6] = xp;

x\_loc[7] = xp\_mid;

x\_loc[8] = xp\_next;

y\_loc[0] = ys;

y\_loc[1] = ys;

y\_loc[2] = ys;

y\_loc[3] = ys\_mid;

y\_loc[4] = ys\_mid;

y\_loc[5] = ys\_mid;

y\_loc[6] = ys\_next;

y\_loc[7] = ys\_next;

y\_loc[8] = ys\_next;

}

void make\_edges\_coor(int type, int s, int p) {

double xp, xp\_next, ys, ys\_next, xp\_mid, ys\_mid;

if (type == 1) {

xp = x[p];

xp\_next = x[p + 1];

xp\_mid = (xp + xp\_next) / 2;

x\_loc[0] = xp;

x\_loc[1] = xp\_mid;

x\_loc[2] = xp\_next;

y\_loc[0] = y[s];

}

if (type == 2) {

ys = y[s];

ys\_next = y[s + 1];

ys\_mid = (ys + ys\_next) / 2;

y\_loc[0] = ys;

y\_loc[1] = ys\_mid;

y\_loc[2] = ys\_next;

x\_loc[0] = x[p];

}

}

void make\_local\_matrix(double hx, double hy, double lyam, double gamma) {

for (int i = 0; i < 9; i++) {

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

int mu\_i = (i % 3);

int mu\_j = (j % 3);

int nu\_i = ((int)(i / 3));

int nu\_j = ((int)(j / 3));

G[i][j] = lyam \* ((hy / hx)\*G\_small[mu\_i][mu\_j] \* M\_small[nu\_i][nu\_j] + (hx / hy)\*M\_small[mu\_i][mu\_j] \* G\_small[nu\_i][nu\_j]) / 90;

M[i][j] = gamma \*hx\*hy\* M\_small[mu\_i][mu\_j] \* M\_small[nu\_i][nu\_j] / 900;

C[i][j] = hx\*hy\*M\_small[mu\_i][mu\_j] \* M\_small[nu\_i][nu\_j] / 900;

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

if (i != j) {

G[j][i] = G[i][j];

M[j][i] = M[i][j];

C[j][i] = C[i][j];

local[j][i] = local[i][j];

}

}

}

}

void make\_local\_vec(int num\_area) {

for (int i = 0; i < 9; i++) {

f[i] = func(num\_area,x\_loc[i], y\_loc[i]);

b\_loc[i] = 0;

}

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) {

b\_loc[i] += f[j] \* C[i][j];

}

}

}

void add\_local(int s, int p) {

for (int i = 0; i < 9; i++)

global\_num[i] = get\_global(2 \* s, 2 \* p, i);

for (int i = 0; i < 9; i++) {

di[global\_num[i]] += local[i][i];

int beg = ig[global\_num[i]];

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

int end = ig[global\_num[i] + 1] - 1;

int ind = beg;

while (jg[ind] != global\_num[j]) {

ind++;

}

ggl[ind] += local[i][j];

ggu[ind] = ggl[ind];

}

b[global\_num[i]] += b\_loc[i];

}

}

void read\_cond(string cond\_file\_1, string cond\_file\_2) {

ifstream fin1(cond\_file\_1 + ".txt");

fin1 >> ncond1;

cond1.resize(ncond1);

for (int i = 0; i < ncond1; i++) {

cond1[i].resize(5);

for (int j = 0; j < 5; j++)

fin1 >> cond1[i][j];

}

fin1.close();

ifstream fin2(cond\_file\_2 + ".txt");

fin2 >> ncond2;

cond2.resize(ncond2);

for (int i = 0; i < ncond2; i++) {

cond2[i].resize(5);

for (int j = 0; j < 5; j++)

fin2 >> cond2[i][j];

}

fin2.close();

}

void generate\_portrait() {

//списочки для всех глоб узлов

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

for (int s = 0; s < ny - 1; s++) {

for (int p = 0; p < nx - 1; p++) {

for (int i = 0; i < 9; i++) {

for (int j = i + 1; j < 9; j++) {

int ind1 = get\_global(2 \* s, 2 \* p, i);

int ind2 = get\_global(2 \* s, 2 \* p, j);

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

list[ind1].insert(ind2);

}

}

}

}

//создание портрета по списку

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

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

int col = ig[i - 1];

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

}

jg.resize(ig[m]);

ggl.resize(ig[m]);

ggu.resize(ig[m]);

ggl\_new.resize(ig[m]);

ggu\_new.resize(ig[m]);

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

for (int j : list[i]) {

jg[k] = j;

k++;

}

}

}

int get\_area(int s, int p) {

int m = 0;

while (!(xi[w[m][1]] >= p && (p + 1) <= xi[w[m][2]] && yi[w[m][3]] >= s && (s + 1) <= yi[w[m][4]]))

m++;

return w[m][0];

}

int get\_global(int s, int p, int j) { //получение глобального номера по локальному

int k;

switch (j) {

case 0: {

k = (s)\*(2 \* nx - 1) + p;

break;

}

case 1: {

k = (s)\*(2 \* nx - 1) + p + 1;

break;

}

case 2: {

k = (s)\*(2 \* nx - 1) + p + 2;

break;

}

case 3: {

k = (s + 1) \* (2 \* nx - 1) + p;

break;

}

case 4: {

k = (s + 1) \* (2 \* nx - 1) + p + 1;

break;

}

case 5: {

k = (s + 1) \* (2 \* nx - 1) + p + 2;

break;

}

case 6: {

k = (s + 2) \* (2 \* nx - 1) + p;

break;

}

case 7: {

k = (s + 2) \* (2 \* nx - 1) + p + 1;

break;

}

case 8: {

k = (s + 2) \* (2 \* nx - 1) + p + 2;

break;

}

}

return k;

}

double get\_tetta(int num\_second, double x, double y) {

switch (num\_second) {

case 1: {

//return 0;

return x;

}

case 2: {

//return -1;

return -y;

}

}

}

double get\_Ug(int num\_first, double x, double y) {

switch (num\_first) {

case 1: {

//return 2;

//return x;

return x \*x\*x\*y\*y \*y;

}

case 2: {

//return 2;

//return x;

return x \* x\*x\*y\*y \*y;

}

}

}

void add\_second() { //двух типов-верхняя и левая границы

int s, p;

for (int i = 0; i < ncond2; i++) {

int num\_second = cond2[i][0];

int x\_num\_beg = xi[cond2[i][1]];

int x\_num\_end = xi[cond2[i][2]];

int y\_num\_beg = yi[cond2[i][3]];

int y\_num\_end = yi[cond2[i][4]];

int flag;

if (y\_num\_beg == y\_num\_end) flag = 0;//горизонт

else flag = 1; //вертик

if (flag == 0) {

s = y\_num\_beg;

for (p = x\_num\_beg; p < x\_num\_end; p++) {

make\_edges\_coor(1, s, p);

make\_params(num\_second);

local\_cond[0] = lyam \* get\_tetta(1, x\_loc[0], y\_loc[0]);

local\_cond[1] = lyam \* get\_tetta(1, x\_loc[1], y\_loc[0]);

local\_cond[2] = lyam \* get\_tetta(1, x\_loc[2], y\_loc[0]);

double h = x\_loc[2] - x\_loc[0];

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

local\_cond\_vl[i] = 0;

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

local\_cond\_vl[i] +=h/30\* M\_small[i][j] \* local\_cond[i];

}

}

b[get\_global(2 \* s, 2 \* p, 0)] += local\_cond\_vl[0];

b[get\_global(2 \* s, 2 \* p, 1)] += local\_cond\_vl[1];

b[get\_global(2 \* s, 2 \* p, 2)] += local\_cond\_vl[2]; //это верхняя, просто p уже и так для верхнего КЭ

}

}

if (flag == 1) {

p = x\_num\_beg;

for (s = y\_num\_beg; s < y\_num\_end; s++) {

make\_edges\_coor(2, s, p);

make\_params(num\_second);

local\_cond[0] = lyam \* get\_tetta(2, x\_loc[0], y\_loc[0]);

local\_cond[1] = lyam \* get\_tetta(2, x\_loc[0], y\_loc[1]);

local\_cond[2] = lyam \* get\_tetta(2, x\_loc[0], y\_loc[2]);

double h = y\_loc[2] - y\_loc[0];

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

local\_cond\_vl[i] = 0;

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

local\_cond\_vl[i] += h/30\*M\_small[i][j] \* local\_cond[i];

}

}

b[get\_global(2 \* s, 2 \* p, 0)] += local\_cond\_vl[0];

b[get\_global(2 \* s, 2 \* p, 3)] += local\_cond\_vl[1];

b[get\_global(2 \* s, 2 \* p, 6)] += local\_cond\_vl[2];

}

}

}

}

void add\_first() {

int s, p;

for (int i = 0; i < ncond1; i++) {

int num\_first = cond1[i][0];

int x\_num\_beg = xi[cond1[i][1]];

int x\_num\_end = xi[cond1[i][2]];

int y\_num\_beg = yi[cond1[i][3]];

int y\_num\_end = yi[cond1[i][4]];

long B = 1e+10;

int flag;

if (y\_num\_beg == y\_num\_end) flag = 0;//горизонтальная

else flag = 1; //вертикальная

if (flag == 0) {

s = y\_num\_beg;

for (p = x\_num\_beg; p < x\_num\_end; p++) {

make\_edges\_coor(1, s, p);

di[get\_global(2 \* s, 2 \* p, 0)] = B;

di[get\_global(2 \* s, 2 \* p, 1)] = B;

di[get\_global(2 \* s, 2 \* p, 2)] = B;

b[get\_global(2 \* s, 2 \* p, 0)] = B\*get\_Ug(num\_first,x\_loc[0],y\_loc[0]);

b[get\_global(2 \* s, 2 \* p, 1)] = B\*get\_Ug(num\_first, x\_loc[1],y\_loc[0]);

b[get\_global(2 \* s, 2 \* p, 2)] = B\*get\_Ug(num\_first, x\_loc[2],y\_loc[0]);

}

}

if (flag == 1) {

p = x\_num\_beg;

for (s = y\_num\_beg; s < y\_num\_end; s++) {

make\_edges\_coor(2, s, p);

di[get\_global(2 \* s, 2 \* p, 0)] = B;

di[get\_global(2 \* s, 2 \* p, 3)] = B;

di[get\_global(2 \* s, 2 \* p, 6)] = B;

b[get\_global(2 \* s, 2 \* p, 0)] = B\* get\_Ug(num\_first, x\_loc[0], y\_loc[0]);

b[get\_global(2 \* s, 2 \* p, 3)] = B \* get\_Ug(num\_first, x\_loc[0], y\_loc[1]);

b[get\_global(2 \* s, 2 \* p, 6)] = B \* get\_Ug(num\_first, x\_loc[0], y\_loc[2]);

}

}

}

}

void make\_global() {

double hx, hy;

for (int s = 0; s < ny - 1; s++) {

for (int p = 0; p < nx - 1; p++) {

make\_local\_coor(s,p);

hx = x\_loc[2] - x\_loc[0];

hy = y\_loc[6] - y\_loc[0];

int num\_area = 1;

// int num\_area=get\_area(s,p);

make\_params(num\_area);

make\_local\_matrix(hx, hy, lyam, gamma);

make\_local\_vec(num\_area);

add\_local(s, p);

}

}

}

//SLAE

void LU\_sq() {

//копирование-инициализация

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

di\_new[i] = di[i];

}

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

ggl\_new[i] = ggl[i];

ggu\_new[i] = ggu[i];

}

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

double sd = 0; //переменные суммирования

int i0 = ig[i];

int i1 = ig[i + 1];

for (int k = i0; k < i1; k++) {

int j = jg[k];

double sl = 0, su = 0;

int j0 = ig[j];

int j1 = ig[j + 1];

int ki = i0;

int kj = j0;

for (; ki < k && kj < j1;) {

int jl = jg[ki];

int ju = jg[kj];

if (jl == ju) {

sl += ggu\_new[kj] \* ggl\_new[ki];

su += ggl\_new[kj] \* ggu\_new[ki];

ki++; kj++;

}

else if (jl < ju) ki++;

else kj++;

}

ggu\_new[k] = (ggu\_new[k] - su) / di\_new[j];

ggl\_new[k] = (ggl\_new[k] - sl) / di\_new[j];

sd += ggu\_new[k] \* ggl\_new[k];

}

di\_new[i] = sqrt(di\_new[i] - sd);

}

}

void mult(vector<double>&MV,vector<double> vec) {

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

int k0 = ig[i];

int k1 = ig[i + 1];

MV[i] = di[i] \* vec[i];

for (int k = k0; k < k1; k++) {

int j = jg[k];

MV[i] += vec[j] \* ggl[k];

MV[j] += vec[i] \* ggu[k];

}

}

}

double skal\_mult(vector<double> vec1, vector<double> vec2) {

double s = 0;

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

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

}

return s;

}

double norm(vector<double> vec) {

double sum = 0;

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

sum += vec[i] \* vec[i];

return sqrt(sum);

}

void mult\_pr(vector<double> aa, vector<double> &y, vector<double> bb) {

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

double s = 0; //переменные суммирования

int i0 = ig[i];//индекс 1го элемента в iтой строке

int i1 = ig[i + 1];

for (int k = i0; k < i1; k++) {

int j = jg[k];

s += y[j] \* aa[k];

}

y[i] = (bb[i] - s) / di[i];

}

}

void mult\_obr(vector<double> aa, vector<double> &y, vector<double> bb) {

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

y[i] = bb[i];

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

int i0 = ig[i];//индекс 1го элемента в iтой строке

int i1 = ig[i + 1];

y[i] /= di[i];

for (int k = i1 - 1; k >= i0; k--) {

int j = jg[k];

y[j] -= y[i] \* aa[k];

}

}

}

void multU(vector<double> &MV, vector<double>vec) {

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

int k0 = ig[i];

int k1 = ig[i + 1];

MV[i] = di[i] \* vec[i];

for (int k = k0; k < k1; k++) {

int j = jg[k];

//MV[i] += vec[j] \* al[k];

MV[j] += vec[i] \* ggu\_new[k];

}

}

}

void mult\_tr(vector<double> &MV, vector<double> vec) {

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

int k0 = ig[i];

int k1 = ig[i + 1];

MV[i] = di[i] \* vec[i];

for (int k = k0; k < k1; k++) {

int j = jg[k];

MV[i] += vec[j] \* ggu[k];

MV[j] += vec[i] \* ggl[k];

}

}

}

void LOS\_sq() {

double skal1, skal2;

int max\_it = 10000;

double err = 1e-20;

LU\_sq();

// инициализация

mult(temp1, q);

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

temp2[i] = b[i] - temp1[i];

}

mult\_pr(ggl\_new, r, temp2);

mult\_obr(ggu\_new, z, r);

mult(temp1, z);

mult\_pr(ggl\_new, p, temp1);

//iteration

double nev = skal\_mult(r, r);

for (int k = 0; k < max\_it && nev > err; k++) {

cout << k + 1 << ": ";

skal1 = skal\_mult(p, r);

skal2 = skal\_mult(p, p);

double alfa = skal1 / skal2;

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

q[i] += alfa \* z[i];

r[i] -= alfa \* p[i];

}

mult\_obr(ggu\_new, temp1, r);

mult(temp2, temp1);

mult\_pr(ggl\_new, temp1, temp2);

skal1 = skal\_mult(p, temp1);

double beta = -skal1 / skal2;

mult\_obr(ggu\_new, temp2, r);

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

z[i] = temp2[i] + beta \* z[i];

}

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

p[i] = temp1[i] + beta \* p[i];

}

nev = skal\_mult(r, r);

cout << "nev=" << nev << endl;

}

};

void MSG\_sq() {

double skal1, skal2;

int max\_it = 3000;

double err = 1e-20;

LU\_sq();

// инициализация

mult(temp1, q);

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

temp1[i] = b[i] - temp1[i];

}

mult\_pr(ggl\_new, temp2, temp1);

mult\_obr(ggl\_new, temp1, temp2);

mult\_tr(temp2, temp1);

mult\_pr(ggu\_new, r, temp2);

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

z[i] = r[i];

}

multU(temp1,q);

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

q[i] = temp1[i];

}

skal1 = skal\_mult(r, r);

double norm\_f = norm(b);

//iteration

double nev = norm(r) / norm\_f;

for (int k = 0; k < max\_it && nev > err; k++) {

cout << k + 1 << ": ";

mult\_obr(ggu\_new, temp2, z);

mult(temp1, temp2);

mult\_pr(ggl\_new, temp2, temp1);

mult\_obr(ggl\_new, temp1, temp2);

mult\_tr(temp2, temp1);

mult\_pr(ggu\_new, temp1, temp2);

skal2 = skal\_mult(temp1, z);

double alfa = skal1 / skal2;

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

q[i] += alfa \* z[i];

r[i] -= alfa \* temp1[i];

}

skal2 = skal\_mult(r, r);

double beta = skal2 / skal1;

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

z[i] = r[i] + beta \* z[i];

}

skal1 = skal2;

nev = norm(r) / norm\_f;

cout << "nev=" << nev << endl;

}

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

temp1[i] = q[i];

mult\_obr(ggu\_new, q, temp1);

}

void do\_smth() {

resize\_func();

write\_small\_matrix();

generate\_portrait();

make\_global();

//add\_second();

add\_first();

LOS\_sq();

}

};

int main() {

SLAE slv = SLAE("net","area","cond1","cond2");

slv.do\_smth();

ofstream fout("out.txt");

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

fout << slv.q[i] << endl;

system("pause");

return 0;

}