Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Курсовая работа

по уравнениям математической физики

Факультет: ПМИ

Группа: ПМ-01

Студент: Александров М.Е.

Вариант: 6

Преподаватель: Персова М.Г.

Новосибирск

2013

# Постановка задачи

Решить методом конечных элементов трёхмерную гиперболическую задачу в декартовых координатах. Вид уравнения:

,  (1)

– известная кусочно-постоянная функция;

- известная функция трёх переменных;

 - известная функция трёх переменных;

Краевые условия всех типов:







Вид разностной схемы по времени: неявная, трёхслойная.

Вид конечных элементов: параллелепипеды.

Вид базисных функций: трилинейные.

Формат хранения матрицы СЛАУ: разреженный строчно-столбцовый.

# Теоретическая часть

## *Дискретизация по времени*

Введём сетку по времени, разбив необходимый отрезок точками . Рассмотрим отрезок  и представим на нём нашу функцию в виде полиномов Лагранжа:

, (2)

где  – значения функции  при  соответственно,

а  – квадратичные полиномы равные 1 при , и 0 при .

Данные функции времени выглядят следующим образом:

 где .

Подставим вид искомой функции (2) в исходное уравнение (1). Для этого надо вычислить первые и вторые производные от функций  в точке (т.к. схема неявная):

После подстановки уравнение (1) примет вид:

Перенесём все известные компоненты в правую часть:



Для краткости выкладок обозначим правую часть за, а 

## *Вариационная постановка*

Выполним вариационную постановку методом Бубнова-Галёркина. Введём гильбертово пространство . В нём скалярное произведение определяется следующим образом:

 (1.1)

Норма, ассоциированная со скалярным произведением: .

В общем виде постановка Бубнова-Галёркина для операторного уравнения записывается в следующем виде:



Где . Для уравнения (1) постановка (1.1) примет вид:

 (1.2)

Применив формулу Грина к этому выражению получим:

 (1.3)

## *Конечноэлементная дискретизация*

Разобьём область  на непересекающиеся подобласти – конечные элементы: . В соответствии с заданием  - параллелепипеды. Формулу (1.3) можно переписать в виде:

 (1.4)

В области выберем линейный финитный базис , обладающий следующими свойствами:

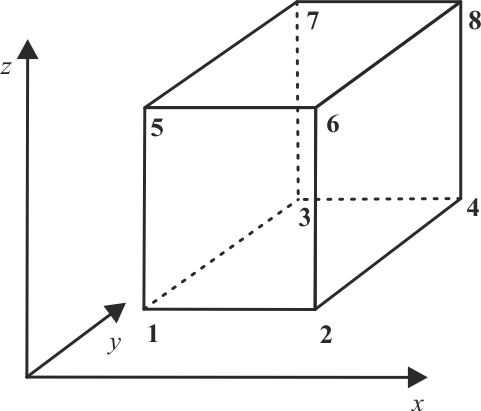
– Базисная функция ассоциирована с узлом конечноэлементной сетки , причём: 

– Базисная функция  отличная от нуля только на тех конечных элементах, которые содержат точку

Рассмотрим отдельно интегралы по конечным элементам, по граням, где заданы вторые краевые условия и граням, где заданы третьи краевые условия.

## *Локальные матрицы и вектора конечных элементов*

В качестве базисных функций на элементе  выберем:

Для конечного элемента элементы локальной матрицы жёсткости примут вид:



Элементы матрицы массы будут вычисляться :



Вектора правой части:



После вычислений итоговые матрицы примут вид:





где:





При разложении функции f по базису, получаем:



Где - вектор.

В матричном виде (1) будет выглядеть:



## *Решение СЛАУ*

Для решения СЛАУ будем использовать МСГ с неполной факторизацией.

# Описание разработанной программы

## *Структуры данных:*

Структура *local* хранит количество локальных областей, узлы из которых состоит локальная область и коэффициенты .

Структура cross хранит координаты узлов и их количество.

Класс *MATRIX* хранит матрицу в разреженном строчном формате и содержит методы для ее решения(разложения в LLT).

Класс *GLOBAL\_MATRIX* является наследником класса *MATRIX* и содержит информацию о локальных областях, координатах узлов, правой части ДУ. В классе находятся методы, которые позволяют считать данные, применить краевые условия, решить матрицу (МСГ) и вычислить значение функции в любой точке области.

## *Исходные данные:*

Файл *local.txt* в первой строке хранит количество конечных элементов, в последующих строках: номера узлов входящих в конечный элемент, параметры .

Файл cross.txt в первой строке хранит количество узлов, в последующих строках хранятся их координаты.

Файл *kraev\_1.txt* содержит номера узлов где нужно задать первое краевое условие.

Файлы *kraev\_2\_<N>.txt* содержит номера граничных узлов и значения потока в них.

Файл *kraev\_3\_<N>.txt* содержит номера граничных узлов, значения потока в них , параметр β.

*<N> -* номер временного слоя.

В функции *right* класса *GLOBAL\_MATRIX* содержится функция правой части.

В функции *Ua* класса *GLOBAL\_MATRIX* содержится аналитическое значение функции (для оценки погрешности).

В функции *sigma* класса *GLOBAL\_MATRIX* содержится функция .

В функции *chi* класса *GLOBAL\_MATRIX* содержится функция .

# Тестирование и исследования

## *Тест 1*

Цель теста: проверка вклада компоненты при .

Параметры теста:



Краевые условия: первого рода.

Количество узлов: 1331

|  |  |
| --- | --- |
|  | Относительная погрешность |
| 0.2 | 1.627936562013949400e-014 |
| 0.3 | 3.066601800718981200e-014 |
| 0.4 | 4.711112051526903800e-014 |
| 0.5 | 6.101898576211329600e-014 |
| 0.6 | 7.580112201827664800e-014 |
| 0.7 | 9.078275020397903600e-014 |
| 0.8 | 1.045652118505472800e-013 |
| 0.9 | 1.130268000282097600e-013 |
| 1.0 | 1.189072313677049800e-013 |

## *Тест 2*

Цель теста: проверить вклад дивергентной компоненты.

Параметры теста:



Краевые условия: первого рода.

Количество узлов: 1331

|  |  |
| --- | --- |
|  | Относительная погрешность |
| 0.2 | 3.784805147152752600e-015 |
| 0.3 | 3.684488527349151600e-015 |
| 0.4 | 3.568154147985066500e-015 |
| 0.5 | 3.727559174053712800e-015 |
| 0.6 | 3.899884879226070300e-015 |
| 0.7 | 3.445044959907866100e-015 |
| 0.8 | 3.591412722935076100e-015 |
| 0.9 | 3.802804842012643800e-015 |
| 1.0 | 4.392414968340299600e-015 |

## *Тест 3*

Цель теста: проверка вклада компоненты при .

Параметры теста:



Краевые условия: первого рода.

Количество узлов: 9261

|  |  |
| --- | --- |
|  | Относительная погрешность |
| 0.2 | 6.039962652561381000e-014 |
| 0.3 | 1.296282506128491700e-013 |
| 0.4 | 2.155265809353709100e-013 |
| 0.5 | 3.170570701158787900e-013 |
| 0.6 | 4.332603450132910500e-013 |
| 0.7 | 5.632646598782968200e-013 |
| 0.8 | 7.056448389061342100e-013 |
| 0.9 | 8.572295665298476700e-013 |
| 1.0 | 1.012744313016825000e-012 |

## *Тест 4*

Цель теста: определить порядок аппроксимации по времени.

Параметры теста:



Краевые условия: первого рода.

Количество узлов: 9261

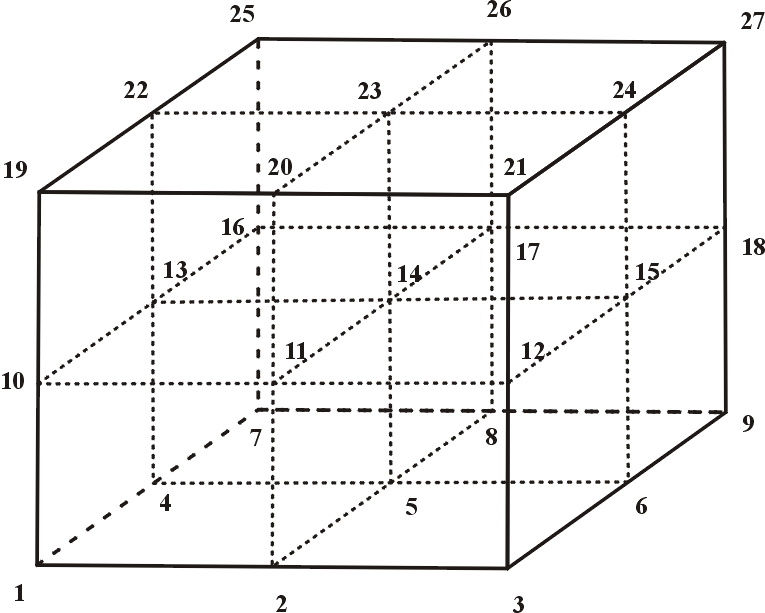
При 

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг по времени | 0.025 | 0.0125 | 0.00625 |
| Относительная погрешность | 8.464834678927092600e-003 | 2.140862021393743400e-003 | 5.383077199078433200e-004 |

Оценка порядка аппроксимации по времени: 

## *Тест 5*

Цель теста: проверка 2 и 3 краевых условий.

Параметры теста:



Краевые условия: первого, второго, третьего рода.

Количество узлов: 27

|  |  |
| --- | --- |
|  | Относительная погрешность |
| 1.0 | 3.038395362969182500e-002 |
| 1.5 | 5.504901609136000900e-002 |
| 2.0 | 7.696382346813637800e-002 |
| 2.5 | 3.946049383950815400e-001 |
| 3.0 | 5.438930906557216700e-001 |

# Вывод

На рассмотренных модельных задачах для построенной разностной схемы по времени фактический и теоретический порядок аппроксимации совпадают, что демонстрирует правильность расчетов и реализации.

# Код программы

#include "myclass.h"

#include "grid\_gen\_cube.h"

int main()

{

grid\_gen\_cube gener;

gener.lambda = 1.0;

bool success = false;

success = gener.generate\_unreg\_grid\_FEM(gen\_point(0, 0, 0), gen\_point(20, 20, 20), 1, 1, 1, 1+1e-15 , 1+1e-15, 1+1e-15, "cross.txt", "local.txt", "faces.txt");

gener.generate\_face\_nodes("faces.txt","kraev\_1.txt");

grid\_gen::generate\_unreg\_grid(2.98125,3, 0.00625, 1+1e-15,"time.txt");

GLOBAL\_MATRIX my;

my.calculate();

return 0;

}

#ifndef MYCLASS\_H\_

#define MYCLASS\_H\_

#define M 8

//#define debug\_print

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

double right(double \*x);

void mul\_c\_vector(double\*,double\*);

// Локальные области

struct local

{

static int count;

int mas[M];

double lambda;

//double gamma;

};

//Координаты узлов

struct cross

{

double mas[3];

static int count;

};

class MATRIX

{

public:

int n;

double \*d;

double \*gg;

int \*ig;

int \*jg;

void SST(MATRIX \*);

void mul\_matrix\_vector(double \*vector,double \*result);

void solution\_x(double\*,double\*);

void solution\_x\_l(double\*,double\*);

void solution\_x\_u(double\*,double\*);

~MATRIX();

};

class GLOBAL\_MATRIX : public MATRIX

{

private:

local \*matr;

cross \*set;

double \*f;

double \*x;

//double \*vect\_v;

double \*time;

int time\_size;

public:

GLOBAL\_MATRIX();

~GLOBAL\_MATRIX();

void read\_local();

void read\_cross();

void read\_time();

double U0(double \*x);

double U1(double \*x);

double \*Uprev;

double \*Uprev2;

double m\_sigma(int i);

double m\_chi(int i);

void formier\_profil();

void formier\_matrix(int i);

void add(int, int, double);

void MSG();

void kraev\_1(int it);

void kraev\_2(int it);

void kraev\_3(int it);

void calculate();

void print\_result(int i);

double U(double nx, double y, double z);

double get\_psi(int num\_fe, int num\_basis, double x,double y, double z);

char \*FILE\_LOCAL\_NUMBER;

char \*FILE\_CROSS;

char \*FILE\_TIME;

char \*FILE\_OUT;

char \*FILE\_CRAEV\_1;

char \*FILE\_CRAEV\_2;

char \*FILE\_CRAEV\_3;

double POR;

};

#endif

#include "myclass.h"

#include <string.h>

#include <vector>

using namespace std;

//основа матрицы массы

double c[M][M]={{8},

{4,8},

{4,2,8},

{2,4,4,8},

{4,2,2,1,8},

{2,4,1,2,4,8},

{2,1,4,2,4,2,8},

{1,2,2,4,2,4,4,8}};

//составные матрицы жесткости

double bx[M][M]={{4},

{-4,4},

{2,-2,4},

{-2,2,-4,4},

{2,-2,1,-1,4,4},

{-2,2,-1,1,-4,4},

{1,-1,2,-2,2,-2,4},

{-1,1,-2,2,-2,2,-4,4}};

double by[M][M]={{4},

{2,4},

{-4,-2,4},

{-2,-4,2,4},

{2,1,-2,-1,4},

{1,2,-1,-2,2,4},

{-2,-1,2,1,-4,-2,4},

{-1,-2,1,2,-2,-4,2,4}};

double bz[M][M]={{4},

{2,4},

{2,1,4},

{1,2,2,4},

{-4,-2,-2,-1,4},

{-2,-4,-1,-2,2,4},

{-2,-1,-4,-2,2,1,4},

{-1,-2,-2,-4,1,2,2,4}};

double right\_vector[M];

double c1[M/2][M/2]={{4},

{2,4},

{2,1,4},

{1,2,2,4}};

//-------------------------------

double Ua(double\* x, double t)

{

//return x[0] + x[1] + x[2] + t;

//return x[0]\*x[0] + x[1] + x[2] + t;

//return x[0] + x[1] + x[2] + t\*t;

return sin(x[0]+x[1]) + t\*t;

}

//-------------------------------

double right(double \*x, double t)

{

return 2\*sin(x[0]+x[1])+5+2\*t;

}

//--------------------------------

/\*double gamma(double \*x)

{

return 0;

}\*/

//-------------------------------

double sigma(double \*x)

{

return 1;

}

//-------------------------------

double chi(double \*x)

{

return 2;

}

//-------------------------------

double GLOBAL\_MATRIX::m\_sigma(int it)

{

double sum = 0;

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

sum += sigma(set[matr[it].mas[i]-1].mas);

return sum/M;

}

//-------------------------------

double GLOBAL\_MATRIX::m\_chi(int it)

{

double sum = 0;

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

sum += chi(set[matr[it].mas[i]-1].mas);

return sum/M;

}

//-------------------------------

double GLOBAL\_MATRIX::U0(double \*x)

{

return Ua(x, time[0]);

}

//-------------------------------

double GLOBAL\_MATRIX::U1(double \*x)

{

return Ua(x, time[1]);

}

//-------------------------------

void mul\_c\_vector(double \*vector,double \*result)

{

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

result[i]=0;

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

int i1,j1;

if(i>j) {

i1=i;

j1=j;

}

else {

i1=j;

j1=i;

}

result[i]+=c[i1][j1]\*vector[j];

}

}

}

//--------------------------------

void mul\_c1\_vector(double \*vector,double \*result)

{

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

result[i]=0;

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

int i1,j1;

if(i>j) {

i1=i;

j1=j;

}

else {

i1=j;

j1=i;

}

result[i]+=c1[i1][j1]\*vector[j];

}

}

}

//----------------------------

double scalar(double \*vector1,double \*vector2,int n)

{

double f=0;

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

f+=vector1[i]\*vector2[i];

return f;

}

//-----------------------------

double norma(double \*vector,int n)

{

double f=scalar(vector,vector,n);

return sqrt(f);

}

//-----------------------------

void sum(double \*vector1,double \*vector2,double \*result,int n)

{

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

result[i]=vector1[i]+vector2[i];

}

//-----------------------------

void sub(double \*vector1,double \*vector2,double \*result,int n)

{

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

result[i]=vector1[i]-vector2[i];

}

//-----------------------------

void mul(double a,double \*vector,double \*result,int n)

{

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

result[i] = a\*vector[i];

}

//-----------------------------

void mul2(double \*a,double \*vector,double \*result,int n)

{

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

result[i]=a[i]\*vector[i];

}

//-----------------------------

void mov(double \*vector,double \*result,int n)

{

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

result[i]=vector[i];

}

//-----------------------------

int local::count=0;

int cross::count=0;

//----------------------------------

void MATRIX::mul\_matrix\_vector(double \*vector,double \*result)

{

int i;

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

result[i]=d[i]\*vector[i];

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

for(int j=ig[i];j<ig[i+1];j++) {

result[i]+=gg[j-1]\*vector[jg[j-1]-1];

result[jg[j-1]-1]+=gg[j-1]\*vector[i];

}

}

//-------------------------------------------------------

void MATRIX::solution\_x\_l(double \*f,double \*x)

{

int i;

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

x[i]=f[i];

//SY=F

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

for(int j=ig[i]-1;j<ig[i+1]-1;j++)

x[i]-=gg[j]\*x[jg[j]-1];

x[i]/=d[i];

}

}

//------------------------------

void MATRIX::solution\_x\_u(double\*f,double \*x)

{

int i;

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

x[i]=f[i];

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

x[i]/=d[i];

for(int j=ig[i+1]-2;j>=ig[i]-1;j--)

x[jg[j]-1]-=gg[j]\*x[i];

}

}

//----------------------------------

void MATRIX::solution\_x(double \*f,double \*x)

{

int i;

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

x[i]=f[i];

//SY=F

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

for(int j=ig[i]-1;j<ig[i+1]-1;j++)

x[i]-=gg[j]\*x[jg[j]-1];

x[i]/=d[i];

}

//STX=Y

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

x[i]/=d[i];

for(int j=ig[i+1]-2;j>=ig[i]-1;j--)

x[jg[j]-1]-=gg[j]\*x[i];

}

}

//----------------------------------

void MATRIX::SST(MATRIX \*my)

{

int i,j,k,l;

my->n=n;

my->d = new double[n];

my->ig = new int[n+1];

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

my->ig[i]=ig[i];

my->jg = new int[ig[n]-1];

my->gg = new double[ig[n]-1];

for(i=0;i<ig[n]-1;i++)

my->jg[i]=jg[i];

for(j=0;j<n;j++)

{

double sum=0;

for(k=ig[j]-1;k<ig[j+1]-1;k++)

sum+=my->gg[k]\*my->gg[k];

my->d[j]=sqrt(fabs(d[j]-sum));

for(i=j+1;i<n;i++)

{

int number;

int flag=1;

for(l=ig[i]-1;l<ig[i+1]-1&&flag;l++)

if(jg[l]==j+1) flag=0;

number=l-1;

if(flag) continue;

sum=0;

for(k=ig[i]-1;k<ig[i+1]-1&&jg[k]<=j;k++)

{

flag=1;

for(l=ig[j]-1;l<ig[i+1]-1&&flag&&jg[l]<=jg[k];l++)

if(jg[l]==jg[k]) flag=0;

l--;

if(!flag)

sum+=my->gg[l]\*my->gg[k];

}

my->gg[number]=(gg[number]-sum)/my->d[j];

}

}

}

MATRIX::~MATRIX()

{

delete [] d;

delete [] ig;

delete [] gg;

delete [] jg;

}

//Считывание нумерации

void GLOBAL\_MATRIX::read\_local()

{

FILE \*f=fopen(FILE\_LOCAL\_NUMBER,"r");

fscanf(f,"%d",&local::count);

matr = new local[local::count];

for(int i=0;i<local::count;i++) {

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

fscanf(f," %d",&matr[i].mas[j]);

fscanf(f," %lf",&matr[i].lambda);

}

#ifdef debug\_print

printf("\n Local:%d\n", local::count);

for(int i=0;i<local::count;i++) {

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

printf(" %d",matr[i].mas[j]);

printf("\n %lf\n",matr[i].lambda);

}

#endif

fclose(f);

}

//----------------------------------------------------

//Считывание координат узлов

void GLOBAL\_MATRIX::read\_cross()

{

FILE \*f=fopen(FILE\_CROSS,"r");

fscanf(f," %d",&cross::count);

n=cross::count;

set = new cross[n];

for(int i=0;i<cross::count;i++)

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

fscanf(f," %lf",&set[i].mas[j]);

#ifdef debug\_print

for(int i=0;i<cross::count;i++) {

printf("\n");

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

printf("%2.1f ",set[i].mas[j]);

}

#endif

fclose(f);

}

//-----------------------------------------------------

void GLOBAL\_MATRIX::read\_time()

{

FILE \*f = fopen(FILE\_TIME,"r");

fscanf(f," %d",&time\_size);

time = new double[time\_size];

for(int i = 0; i < time\_size; i++)

fscanf(f, " %lf", &time[i]);

#ifdef debug\_print

printf("\n Time:%d\n", time\_size);

for(int i = 0; i < time\_size; i++)

printf( " %lf", time[i]);

#endif

fclose(f);

}

//------------------------------------------------------

void GLOBAL\_MATRIX::formier\_profil()

{

vector < vector < bool > > prof;

prof.resize(n-1);

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

prof[i].resize(i+1);

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

prof[i][j] = false;

}

unsigned int s=0;//Кол-во ненулевых эл-тов

for(int k=0;k<local::count;k++)

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

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

int i1 = matr[k].mas[i], j1 = matr[k].mas[j], k\_b;

if(i1<j1) {

k\_b=i1;

i1=j1;

j1=k\_b;

}

if(prof[i1-2][j1-1]==0) {

s++;

prof[i1-2][j1-1]=1;

}

}

//Формирование массива ig и jg

jg = new int[s];

gg = new double[s];

ig[0]=1;

ig[1]=1;

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

int k=0;

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

if(prof[i][j]==1) {

jg[d]=j+1;

d++;

k++;

}

ig[i+2]=ig[i+1]+k;

}

printf("\n");

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

printf(" %d",ig[i]);

printf("\n");

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

printf(" %d",jg[i]);

}

//------------------------------------------------------

void GLOBAL\_MATRIX::add(int i,int j,double x)

{

int k;

for(k=ig[i]-1;k<ig[i+1]-1;k++)

if(jg[k]==j+1)

break;

gg[k]+=x;

}

//------------------------------------------------------

void GLOBAL\_MATRIX::formier\_matrix(int it)

{

double h[3];

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

d[i] = 0.0;

f[i] = 0.0;

}

for(int i = 0; i < ig[n]-1; i++) {

gg[i] = 0.0;

}

//t=====================================

double dt = time[it] - time[it-2];

double dt1 = time[it-1] - time[it-2];

double dt0 = time[it] - time[it-1];

//======================================

double fr[M];

double um2[M];

double um1[M];

double um2t[M];

double um1t[M];

double vsig[M];

double vchi[M];

for(int k = 0; k < local::count; k++) {

//Вычисление шага

//h----------------------------------------------------------

for(int i=0;i<3;i++){ //i-по x,y,z

int flag=1;

int j;

for(j=1; j<M && flag;j++){//1 узел фиксируем,пробегаем по остальным

flag=0;

for(int l=0;l<3&&!flag;l++)//проверяем,лежат ли точки на нужном ребре

if(i!=l&&set[matr[k].mas[0]-1].mas[l]!=set[matr[k].mas[j]-1].mas[l])

flag=1;

}

if(!flag)

h[i]=fabs(set[matr[k].mas[0]-1].mas[i]-set[matr[k].mas[j-1]-1].mas[i]);

}

//------------------------------------------------------------

//формирование элементов матрицы

//заполнение мссива gg

double b\_k=matr[k].lambda\*h[0]\*h[1]\*h[2]/36.;

double c\_k=h[0]\*h[1]\*h[2]/216.;

//вектор правой части для локал. матрицы

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

right\_vector[i] = right(set[matr[k].mas[i]-1].mas, time[it]);

mul\_c\_vector(right\_vector, fr);

//====Формирование локальных предыдущих векторов======

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

um2t[p] = Uprev2[matr[k].mas[p]-1];

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

um1t[p] = Uprev[matr[k].mas[p]-1];

//====================================================

mul(dt0/(dt1\*dt), um2t, um2, M);

mul(dt/(dt0\*dt1), um1t, um1, M);

sub(um2, um1, vsig, M);

mul(2.0/(dt1\*dt), um2t, um2, M);

mul(2.0/(dt1\*dt0), um1t, um1, M);

sub(um2, um1, vchi, M);

mul\_c\_vector(vsig, um2);

mul\_c\_vector(vchi, um1);

#ifdef debug\_print

printf("\ntest gg:\n");

#endif

for(int ind = 0; ind < M; ind++) {

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

int i1 = matr[k].mas[ind] - 1;

int j1 = matr[k].mas[j] - 1;

double s = //gamma(set[matr[k].mas[]-1].mas) \* c\_k \* c[i][j] +

b\_k / (h[0]\*h[0]) \* bx[ind][j] +

b\_k / (h[1]\*h[1]) \* by[ind][j] +

b\_k / (h[2]\*h[2]) \* bz[ind][j] +

(dt0+dt)/(dt\*dt0) \* m\_sigma(k) \* c\_k \* c[ind][j] +

2.0 / (dt\*dt0) \* m\_chi(k) \* c\_k \* c[ind][j];

#ifdef debug\_print

printf("%lf ", s);

#endif

//добавка в gg

if(i1 < j1)

add(j1, i1, s);

else

add(i1, j1, s);

}

#ifdef debug\_print

printf("\n");

#endif

//добавка в диагональ

d[matr[k].mas[ind]-1] += //c\_k \* gamma(set[].mas) \* c[i][i] +

b\_k / (h[0]\*h[0]) \* bx[ind][ind]+

b\_k / (h[1]\*h[1]) \* by[ind][ind] +

b\_k / (h[2]\*h[2]) \* bz[ind][ind] +

(dt0+dt)/(dt\*dt0) \* m\_sigma(k) \* c\_k \* c[ind][ind] +

2.0 / (dt\*dt0) \* m\_chi(k) \* c\_k \* c[ind][ind];

f[matr[k].mas[ind]-1] += c\_k \* fr[ind] -

c\_k \* m\_sigma(k) \* um2[ind] -

c\_k \* m\_chi(k) \* um1[ind];

}

}

}

//------------------------------------------------------

GLOBAL\_MATRIX::GLOBAL\_MATRIX()

{

FILE\_LOCAL\_NUMBER="local.txt";

FILE\_CROSS="cross.txt";

FILE\_TIME="time.txt";

FILE\_OUT="out";

FILE\_CRAEV\_1="kraev\_1.txt";

FILE\_CRAEV\_2="kraev\_2\_";

FILE\_CRAEV\_3="kraev\_3\_";

POR=1e30;

//formier\_local();

read\_cross();

read\_local();

read\_time();

d = new double[n];

f = new double[n];

x = new double[n];

ig = new int[n+1];

Uprev = new double[n];

Uprev2 = new double[n];

formier\_profil();

//formier\_matrix();

/\*for(int i=0;i<ig[n]-1;i++)

printf("%1.2f ",gg[i]);

printf("\n=============\n");

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

printf("%1.2f ",f[i]);

printf("\n=============\n");

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

printf("%1.2f ",d[i]);\*/

}

GLOBAL\_MATRIX::~GLOBAL\_MATRIX()

{

delete [] matr;

delete [] set;

delete [] f;

delete [] x;

}

//---------------------------------------------

void GLOBAL\_MATRIX::MSG()

{

double e=1e-70;

int max=1000;

MATRIX s;

SST(&s);

double \*r=new double[n];

double \*z=new double[n];

double \*h=new double[n];

double \*h2=new double[n];

double \*h3=new double[n];

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

x[i]=0;

mul\_matrix\_vector(x,h);

mul(-1,h,h,n);

sum(f,h,r,n);

s.solution\_x(r,z);

double e2=norma(r,n)/norma(f,n);

double a,b;

int k;

for(k=1;k<max&&e2>e;k++) {

//a

s.solution\_x(r,h2);

mul\_matrix\_vector(z,h);

a=scalar(h2,r,n)/scalar(h,z,n);

//r

mov(r,h3,n);

mul(-a,h,h,n);

sum(r,h,r,n);

//x

mul(a,z,h,n);

sum(x,h,x,n);

//b

b=1/scalar(h2,h3,n);

s.solution\_x(r,h2);

b\*=scalar(h2,r,n);

//z

mul(b,z,z,n);

sum(z,h2,z,n);

e2=norma(r,n)/norma(f,n);

}

printf("k=%d e=%10.8e\n",k-1,e2);

delete [] r;

delete [] z;

delete [] h;

delete [] h2;

delete [] h3;

}

//--------------------------------------

void GLOBAL\_MATRIX::print\_result(int it)

{

char\* st;

st = new char[20];

sprintf(st,"%s%i.txt", FILE\_OUT,it);

FILE \*f=fopen(st,"w");

double msub = 0, nr = 0;

fprintf(f , "U\* \t U \t U-U\*\n");

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

fprintf(f, "%20.18e\t%20.18e\t%20.18e\n", x[i], Ua(set[i].mas, time[it]), fabs(Ua(set[i].mas, time[it]) - x[i]));

msub += (Ua(set[i].mas, time[it]) - x[i]) \* (Ua(set[i].mas, time[it]) - x[i]);

nr = Ua(set[i].mas, time[it]) \* Ua(set[i].mas, time[it]);

}

fprintf(f,"||u\*-u||/||u||= %20.18e \n||u\*-u||/N= %20.18e\n", sqrt(msub/nr), msub/n);

fclose(f);

}

//---------------------------------------

void GLOBAL\_MATRIX::kraev\_1(int it)

{

int i;

double q;

FILE \*fl=fopen(FILE\_CRAEV\_1,"r");

if (fl==NULL) return;

int flag;

while((flag=fscanf(fl," %d",&i))>0) {

i--;

q = Ua(set[i].mas, time[it]);

d[i] = POR;

f[i] = q\*POR;

}

fclose(fl);

}

//----------------------------------------

void GLOBAL\_MATRIX::kraev\_3(int it)

{

int cr[4];//граничные узлы

double teta[4];//значения потока в узлах

double betta;

char st[20];

sprintf(st, "%s%d.txt", FILE\_CRAEV\_3, it);

FILE \*fl=fopen(st,"r");

if (fl==NULL) return;

while(!feof(fl)) {

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

fscanf(fl," %d",&cr[i]);

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

fscanf(fl," %lf",&teta[i]);

fscanf(fl," %lf",&betta);

double hxyz = 1.0;

double eps = 1e-12;

double hx = 0.0, hy = 0.0;

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

double tmp = fabs(set[cr[0]-1].mas[i]-set[cr[1]-1].mas[i]);

if(tmp > eps)

hx += tmp;

tmp = fabs(set[cr[0]-1].mas[i]-set[cr[2]-1].mas[i]);

if(tmp > eps)

hy += tmp;

}

hxyz = hx \* hy;

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

d[cr[i]-1] += betta \* hxyz/36.0 \*c1[i][i];

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

int ind= ig[cr[i]-1]-1;

while(jg[ind] != cr[j])

ind++;

gg[ind] += betta \* hxyz/36.0 \*c1[i][j];

}

}

double vec\_tet[4];

mul\_c1\_vector(teta,vec\_tet);

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

f[cr[i]-1] += hxyz/36.0 \*vec\_tet[i];

}

fclose(fl);

}

//----------------------------------------

void GLOBAL\_MATRIX::kraev\_2(int it)

{

int cr[4];//граничные узлы

double teta[4];//значения потока в узлах

char st[20];

sprintf(st, "%s%d.txt", FILE\_CRAEV\_2, it);

FILE \*fl=fopen(st,"r");

if (fl==NULL) return;

while(!feof(fl)) {

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

fscanf(fl," %d",&cr[i]);

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

fscanf(fl," %lf",&teta[i]);

double hxyz = 1.0;

double eps=1e-12;

double hx = 0.0, hy = 0.0;

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

double tmp = fabs(set[cr[0]-1].mas[i]-set[cr[1]-1].mas[i]);

if(tmp > eps)

hx += tmp;

tmp = fabs(set[cr[0]-1].mas[i]-set[cr[2]-1].mas[i]);

if(tmp > eps)

hy += tmp;

}

hxyz = hx \* hy;

double vec\_tet[4];

mul\_c1\_vector(teta,vec\_tet);

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

f[cr[i]-1] += hxyz/36.0 \*vec\_tet[i];

}

fclose(fl);

}

//----------------------------------------

// получение значения базисной ф-ции

double GLOBAL\_MATRIX::get\_psi(int num\_fe, int num\_basis, double x,double y, double z)

{

int n1,n2,n3,n5;

double hx,hy,hz;

n1=matr[num\_fe].mas[0];

n2=matr[num\_fe].mas[1];

n3=matr[num\_fe].mas[2];

n5=matr[num\_fe].mas[4];

hx=set[n2-1].mas[0]-set[n1-1].mas[0];

hy=set[n3-1].mas[1]-set[n1-1].mas[1];

hz=set[n5-1].mas[2]-set[n1-1].mas[2];

switch(num\_basis) {

case(0):

return (set[n2-1].mas[0]-x)/hx\*(set[n3-1].mas[1]-y)/hy\*(set[n5-1].mas[2]-z)/hz;

break;

case(1):

return (x-set[n1-1].mas[0])/hx\*(set[n3-1].mas[1]-y)/hy\*(set[n5-1].mas[2]-z)/hz;

break;

case(2):

return (set[n2-1].mas[0]-x)/hx\*(y-set[n1-1].mas[1])/hy\*(set[n5-1].mas[2]-z)/hz;

break;

case(3):

return (x-set[n1-1].mas[0])/hx\*(y-set[n1-1].mas[1])/hy\*(set[n5-1].mas[2]-z)/hz;

break;

case(4):

return (set[n2-1].mas[0]-x)/hx\*(set[n3-1].mas[1]-y)/hy\*(z-set[n1-1].mas[2])/hz;

break;

case(5):

return (x-set[n1-1].mas[0])/hx\*(set[n3-1].mas[1]-y)/hy\*(z-set[n1-1].mas[2])/hz;

break;

case(6):

return (set[n2-1].mas[0]-x)/hx\*(y-set[n1-1].mas[1])/hy\*(z-set[n1-1].mas[2])/hz;

break;

case(7):

return (x-set[n1-1].mas[0])/hx\*(y-set[n1-1].mas[1])/hy\*(z-set[n1-1].mas[2])/hz;

break;

}

}

//----------------------------------------

double GLOBAL\_MATRIX::U(double nx, double y, double z)

{

bool flag = false;

int i;

//находим элемент

for(i = 0; i < local::count && !flag; i++) {

int v1 = matr[i].mas[0];

int v2 = matr[i].mas[1];

int v3 = matr[i].mas[2];

int v5 = matr[i].mas[4];

if(nx >= set[v1-1].mas[0] && nx <= set[v2-1].mas[0] &&

y >= set[v1-1].mas[1] && y <= set[v3-1].mas[1] &&

z >= set[v1-1].mas[2] && z <= set[v5-1].mas[2])

flag = true;

}

int lc = i;

double sum = 0;

for(i = 0; i < M; i++)

sum += x[matr[lc-1].mas[i]-1] \* get\_psi(lc-1,i,nx,y,z);

return sum;

}

//---------------------------------------

void GLOBAL\_MATRIX::calculate()

{

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

Uprev2[i] = U0(set[i].mas);

Uprev[i] = U1(set[i].mas);

}

for(int i = 2; i < time\_size; i++) {

formier\_matrix(i);

#ifdef debug\_print

printf("\n gg:\n");

for(int i=0;i<ig[n]-1;i++)

printf("%1.2f ",gg[i]);

printf("\n=====f========\n");

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

printf("%1.2f ",f[i]);

printf("\n=====d========\n");

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

printf("%1.5f ",d[i]);

#endif

kraev\_3(i);

kraev\_2(i);

kraev\_1(i);

MSG();

print\_result(i);

memcpy(Uprev2, Uprev, sizeof(double) \* n);

memcpy(Uprev, x, sizeof(double) \* n);

}

}