Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторные работы №3, 4

по уравнениям математической физики

Факультет: ПМИ

Группа: ПМ-01

Студент: Ряховский М.И.

Варианты: 9, 9

Преподаватель: Задорожный А.Г., Персова М.Г.

Новосибирск

2013

# Цель работы

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

Изучить особенности реализации итерационных методов для СЛАУ с несимметричными разреженными СЛАУ. Исследовать влияние предобуславливания на сходимость этих методов.

# Постановка задачи

Дано гиперболическое начально-краевое уравнение вида:

, в 

И правая часть имеет вид: , при этом остальные коэффициенты не зависит от времени.

Решить методом конечных элементов. Вид кэ: параллелепипеды, базис: трилинейный.

# Вариационная постановка и конечноэлементная аппроксимация

При такой постановки задачи решение может быть представлено в виде: , тогда исходное уравнение эквивалентно системе:



Введём гильбертовы пространства  и , домножим скалярно каждое уравнение системы на функцию , получим систему из двух уравнений Галёркина:



Далее проведём аппроксимацию следующим образом:

, 

Подставив выражения в систему, получим СЛАУ из  уравнений.

На локальном уровне матрица системы(без учёта краевых) будет выглядеть следующим образом: это будет блочная матрица размером и каждый блок будет иметь следующий вид:



Здесь - матрица жесткости для трилинейных функция на параллелепипедах (с учётом коэффициента ), а  - аналогичная матрица массы.

Локальная правая часть тоже буде иметь блочную структуру: это будет вектор из 8 двухмерных векторов вида:

, причём вектора  и  вычисляются через соответствующие значения правой части:, .

Локальные матрицы массы краевых условий тоже будет иметь блочную структуру и блоки будут иметь вид:

, где - матрица массы для билиныйных функциях на прямоугольниках, добавки в правую часть будут вычисляться умножением этой матрицы на вектор значений функции или .

# Используемые методы решения СЛАУ

## LU-факторизация

Формат представления матрицы: профильный, выполняется конвертация из разреженного.

## Локально-оптимальная схема

Формат представления матрицы: разреженный

Предобуславливание: -факторизация

## Стабилизированный метод бисопряженных градиентов

Формат представления матрицы: разреженный

Предобуславливание: -факторизация

# Тестирование

## Тест 1. Решение представляется точно

Сетка: 

Шаг:  (одинаковый по всем переменным)

, 

, , , 

, 

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Относительная погрешность | 3.230e-017 | 2.046e-016 | 1.928e-016 |
| Число итераций | - | 11 | 9 |

## Тест 2. Оценка порядка аппроксимации

Сетка: 

Шаг:  (одинаковый по всем переменным)

, 

, , , 

, 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Относительная погрешность | 7.498e-008 | 2.039e-008 | 7.498e-008 | 2.039e-008 | 7.498e-008 | 2.039e-008 |
| Число итераций | - | - | 8 | 52 | 5 | 133 |

Оценка порядка аппроксимации: 

# Исследование на сетке с небольшим количеством узлов

Сетка: 

Шаг: , , 

Общее число узлов: 3388

Решение: , 

Параметры итерационных решателей: точность - , максимальное число итераций - 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| - | - | - | - | с | шт. | c | шт. | c |
|  |  |  |  | 0.282220 | 236 | 0.236629 | 12130 | 22.180972 |
|  |  |  |  | 0.285776 | 236 | 0.237901 | 12130 | 22.110159 |
|  |  |  |  | 0.286288 | 236 | 0.240817 | 12130 | 21.994483 |
|  |  |  |  | 0.283721 | 145 | 0.159681 | 100000 | 183.530957 |
|  |  |  |  | 0.284247 | 145 | 0.155717 | 100000 | 181.949085 |
|  |  |  |  | 0.286646 | 181 | 0.190400 | 3335 | 6.054977 |
|  |  |  |  | 0.283519 | 149 | 0.160309 | 100000 | 182.082858 |
|  |  |  |  | 0.281667 | 1692 | 1.522061 | 100000 | 182.512822 |
|  |  |  |  | 0.285897 | 236 | 0.237889 | 12130 | 22.015288 |
|  |  |  |  | 0.286370 | 236 | 0.239630 | 12130 | 22.085888 |
|  |  |  |  | 0.286333 | 145 | 0.156308 | 100000 | 183.220697 |
|  |  |  |  | 0.286338 | 153 | 0.163450 | 100000 | 183.021901 |
|  |  |  |  | 0.291898 | 176 | 0.183320 | 1190 | 2.209703 |
|  |  |  |  | 0.286142 | 184 | 0.195384 | 100000 | 182.826084 |
|  |  |  |  | 0.286088 | 1917 | 1.716556 | 100000 | 182.607631 |
|  |  |  |  | 0.291071 | 236 | 0.238965 | 12130 | 22.079427 |
|  |  |  |  | 0.283518 | 273 | 0.266690 | 10724 | 19.557291 |
|  |  |  |  | 0.281076 | 153 | 0.163126 | 100000 | 182.001991 |
|  |  |  |  | 0.282378 | 305 | 0.300440 | 100000 | 181.527346 |
|  |  |  |  | 0.290436 | 100000 | 89.017746 | 100000 | 181.731978 |
|  |  |  |  | 0.286174 | 2014 | 1.796470 | 100000 | 181.760017 |
|  |  |  |  | 0.288331 | 100000 | 88.615270 | 100000 | 183.213995 |
|  |  |  |  | 0.306571 | 236 | 0.238036 | 12130 | 22.400100 |
|  |  |  |  | 0.288710 | 3972 | 3.605054 | 100000 | 184.598956 |
|  |  |  |  | 0.280464 | 882 | 0.820093 | 100000 | 182.217881 |
|  |  |  |  | 0.293216 | 33891 | 30.116092 | 100000 | 180.628793 |
|  |  |  |  | 0.283742 | 100000 | 88.357032 | 100000 | 180.756813 |
|  |  |  |  | 0.283665 | 100000 | 88.553575 | 100000 | 182.277881 |
|  |  |  |  | 0.284541 | 100000 | 86.753897 | 100000 | 181.731978 |
|  |  |  |  | 0.280968 | 239 | 0.238946 | 100000 | 179.403057 |
|  |  |  |  | 0.278924 | 188 | 0.192079 | 100000 | 181.076163 |
|  |  |  |  | 0.279172 | 37273 | 32.658626 | 100000 | 185.470776 |
|  |  |  |  | 0.283496 | 100000 | 88.465951 | 100000 | 184.355670 |
|  |  |  |  | 0.296005 | 100000 | 89.810243 | 100000 | 186.382725 |
|  |  |  |  | 0.281300 | 100000 | 87.856849 | 100000 | 183.573062 |
|  |  |  |  | 0.282621 | 100000 | 88.365675 | 100000 | 181.731034 |
|  |  |  |  | 0.284008 | 100000 | 88.953194 | 100000 | 179.703074 |
|  |  |  |  | 0.308059 | 100000 | 90.902670 | 100000 | 181.076163 |

# Исследование на сетке с большим количеством узлов

Сетка: 

Шаг: , , 

Общее число узлов: 32428

Решение: , 

Параметры итерационных решателей: точность - максимальное число итераций - .

В целях экономии времени  запускался только для первого набора парметров.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| - | - | - | - | с | шт. | c | шт. | c |
|  |  |  |  | 277.56073 | 5463 | 52.01578 | - | более чем за 30 минут не дорешал |
|  |  |  |  | 277.75638 | 3779 | 35.83094 |  |  |
|  |  |  |  | 279.24360 | 5359 | 50.35348 |  |  |
|  |  |  |  | 282.78876 | 4451 | 42.42333 |  |  |
|  |  |  |  | 278.49579 | 5463 | 51.64678 |  |  |
|  |  |  |  | 280.47651 | 1972 | 18.59367 |  |  |
|  |  |  |  | 277.75544 | 1835 | 17.59215 |  |  |
|  |  |  |  | 281.75676 | 100000 | 933.5551 |  |  |

# Выводы

При увеличении частоты, а так же параметров  и сходимость итерационных решателей ухудшается, а при увеличении улучшается. Это может быть связанно с тем, что при увеличении указанных параметров исходный дифференциальный оператор теряет свойство положительной определённости, и как следствие его дискретный аналог – матрица системы тоже их теряет. К тому же увеличение параметров  и  делает матрицу «более несимметричной», что тоже сказывается на сходимости. Время решения с помощью -факторизации от параметров системы не зависит, только её размерности (как и должно быть для прямого метода), и в отличии от итерационных сходится всегда.

При сравнении ЛОС и BCGStab, последний, на данных тестах, не показал преимуществ. Он сходится хуже и каждая итерация выполняется за большее количество времени.

# Код программы

## Файл «harm\_FEM.h»

#pragma once

#include <stdio.h>

#include <string>

#include <windows.h> //Для замеров времени

#include "additional\_classes.h"

//Решатели

#include "solver\_LOS.h"

#include "solver\_LU.h"

#include "solver\_BCGStab.h"

using namespace std;

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

//Решение гармонической задачи с помощью МКЭ. Вид элементов - кубы

//solver\_type - тип решателя СЛАУ

template <class solver\_type> class harm\_FEM{

public:

void transf\_grid(string file\_cords, string file\_elements, string file\_faces); //преобразовывает сетку в файлах из обычной кубической в нужную

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

void set\_coefs(func set\_lambda, func set\_sigma, func set\_epsilon); //Ввод коэф. уравнения

void set\_rp(func set\_f\_sin, func set\_f\_cos); // Ввод правой части уравния

void set\_w(double s\_w); //Ввод частоты колебаний

void form\_matrix(); //Формирование матрицы СЛАУ

void solve(); //Решение СЛАУ

void out\_rez(string file\_name); //Вывод резульатата в файл file\_name

void out\_diff(string file\_name); //Вывод погрешности в файл

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

private:

int elements\_n; //Количество узлов

int nodes\_n; //Количество элементов

int faces\_fir\_n, faces\_sec\_n, faces\_thi\_n; //Количество граней с 1, 2 и 3 краевыми соответственно

node\* nodes; //Массив узлов

FE\* elements; //Массив элементов

face \*faces\_sec, \*faces\_thi; //Массивы для краевых граней

double \*faces\_fir; //Значения первых краевых

int \*faces\_fir\_node; //Узлы первых краевых

double u\_betta\_s(double x, double y, double z, int face\_n); //Вычисление краевого условия третьего рода, для синуса

double u\_betta\_с(double x, double y, double z, int face\_n); //Вычисление краевого условия третьего рода, для косинуса

double\* betta;

double tetta\_s(double x, double y, double z, int face\_n); ////Вычисление краевого условия второго рода, для синуса

double tetta\_c(double x, double y, double z, int face\_n); ////Вычисление краевого условия второго рода, для косинуса

func lambda, sigma, epsilon; //коэффициенты уравнения

func f\_sin, f\_cos; //правая часть уравнения

double w; //Частота колебаний

solver\_type solver; //Решатель СЛАУ

int solver\_iters; //количество итераций за которое было решено СЛАУ

SLAE\_port\_gen port\_gen; //Генератор портрета СЛАУ

double time; //Время решения СЛАУ

//Массивы, для хранения СЛАУ

int \*ig, \*jg;

int SLAE\_el\_n; //Количество элементов в СЛАУ(элементов в jg, gl, gu)

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

double \*right\_part; //правая часть

double\* solution; //Решение

func u\_sin, u\_cos;

void form\_gmass(); //Формирование массивов ig и jg

int find\_el\_pos(int i, int j); //Определяет положение элемента в матрице

void form\_loc(double \*\*A\_loc, double \*b\_loc, int el\_n); //Формирование локальной матрицы для элемента el\_n

void gen\_loc\_thi(double \*\*A\_loc, double \*b\_loc, int face\_n); //Формирование локальной матрицы и вектора для третих краевых, для грани face\_n

void gen\_loc\_sec(double \*b\_loc, int face\_n); //Формирование локального вектора для вторых краевых, для грани face\_n

//Вычисление специальных индексов, для матрицы жескости

int mu(int i);

int nu(int i);

int v(int i);

};

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

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

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

// ============================================= Реализации ===========================================================

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

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

template <typename solver\_type> void harm\_FEM<solver\_type>::init(string file\_cords, string file\_elements, string file\_faces){

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

//Ввод координат вершин

fscanf(inp\_cord, "%d", &nodes\_n);

nodes = new node [nodes\_n];

for(int i = 0; i < nodes\_n; i++)

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

fclose(inp\_cord);

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

//Ввод элементов

fscanf(inp\_els, "%d", &elements\_n);

elements = new FE [elements\_n];

for(int i = 0; i < elements\_n; i++){

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

fscanf(inp\_els, "%d", &elements[i].node\_n[j]);

}

}

fclose(inp\_els);

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

//Ввод первых краевых

fscanf(inp\_faces, "%d", &faces\_fir\_n);

faces\_fir = new double [faces\_fir\_n];

faces\_fir\_node = new int [faces\_fir\_n];

for(int i = 0; i < faces\_fir\_n; i++){

fscanf(inp\_faces, "%d %lf", &faces\_fir\_node[i], &faces\_fir[i]);

}

//Ввод вторых краевых

fscanf(inp\_faces, "%d", &faces\_sec\_n);

faces\_sec = new face [faces\_sec\_n];

for(int i = 0; i < faces\_sec\_n; i++){

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

fscanf(inp\_faces, "%d", &faces\_sec[i].node\_n[j]);

}

}

//Ввод третьих краевых

fscanf(inp\_faces, "%d", &faces\_thi\_n);

faces\_thi = new face [faces\_thi\_n];

for(int i = 0; i < faces\_thi\_n; i++){

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

fscanf(inp\_faces, "%d", &faces\_thi[i].node\_n[j]);

}

}

fclose(inp\_faces);

}

template <typename solver\_type> void harm\_FEM<solver\_type>::out\_diff(string file\_name){

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

double diff1 = 0; //Общая погрешность

double u\_norm = 0; //Общая

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

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

for(int i = 0; i < nodes\_n/2; i++){

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

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

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

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

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

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

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

u\_ns += us\*us;

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", solver\_iters);

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

fclose(out\_f);

}

template <typename solver\_type> void harm\_FEM<solver\_type>::set\_coefs(func set\_lambda, func set\_sigma, func set\_epsilon){

lambda = set\_lambda;

sigma = set\_sigma;

epsilon = set\_epsilon;

}

template <typename solver\_type> void harm\_FEM<solver\_type>::set\_rp(func set\_f\_sin, func set\_f\_cos){

f\_sin = set\_f\_sin;

f\_cos = set\_f\_cos;

}

template <typename solver\_type> void harm\_FEM<solver\_type>::set\_w(double s\_w){

w = s\_w;

}

template <typename solver\_type> void harm\_FEM<solver\_type>::solve(){

LARGE\_INTEGER start, stop, timetime, fr;

//Начало замера времени

QueryPerformanceFrequency(&fr);

QueryPerformanceCounter(&start);

solver.init(ig, jg, gu, gl, di, nodes\_n);

solver.set\_rp(right\_part);

solver.solve(solution, solver\_iters);

//Конец замера времени

QueryPerformanceCounter(&stop);

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

time = (double)timetime.QuadPart / (double)fr.QuadPart; //Вычисление времени решения

}

template <typename solver\_type> void harm\_FEM<solver\_type>::form\_gmass(){

port\_gen.init(nodes\_n);

//Вс собираем

ig = new int [nodes\_n+1];

for(int i = 0; i < elements\_n; i++)

port\_gen.add\_el(elements[i]);

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

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

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

di = new double [nodes\_n];

right\_part = new double [nodes\_n];

solution = new double [nodes\_n];

//Обнуление

for(int i = 0; i < SLAE\_el\_n; i++)

gl[i] = gu[i] = 0;

for(int i = 0; i < nodes\_n; i++)

di[i] = right\_part[i] = solution[i] = 0;

port\_gen.~SLAE\_port\_gen();

}

template <typename solver\_type> void harm\_FEM<solver\_type>::form\_matrix(){

double \*\*A\_loc, \*b\_loc; //локальные матрица и вектор правой части

b\_loc = new double [16];

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

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

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

form\_gmass(); //формируем массивы

int cur\_row; //текущая строка

int pos; //Позиция в gu и gl

//Генерация основной СЛАУ

for(int k = 0; k < elements\_n; k++){

form\_loc(A\_loc, b\_loc, k); //Получаем локальные матрицы

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

cur\_row = elements[k].node\_n[i]; //Определяем элемент строки

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

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

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

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

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

}

else{

pos = find\_el\_pos(elements[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];

}

}

// Учёт третьих краевых условий

for(int k = 0; k < faces\_thi\_n; k++){

gen\_loc\_thi(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 = find\_el\_pos(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 = find\_el\_pos(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];

}

}

//Учёт вторых краевых условий

for(int k = 0; k < faces\_sec\_n; k++){

gen\_loc\_sec(b\_loc, k);

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

right\_part[faces\_sec[k].node\_n[i]] += b\_loc[i];

}

//Учёт первых краевых условий

for(int k = 0; k < faces\_fir\_n; k++){

cur\_row = faces\_fir\_node[k]; //Узел, в котором заданно краевое

double val = faces\_fir[k]; //Получаем значение

di[cur\_row] = 1;

right\_part[cur\_row] = val;

//Обнуляем верхную часть столбца

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;

}

//обнуляем нижную часть столбца

for(int p = cur\_row + 1; p < nodes\_n; 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){ //Проверка - тот ли столбец

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

gl[i] = 0;

gu[i] = 0;

}

}

}

}

}

template <typename solver\_type> void harm\_FEM<solver\_type>::form\_loc(double \*\*A\_loc, double \*b\_loc, int el\_n){

//Матрицы жескости и массы для обычного МКЭ

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); //Шаг по x

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

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

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

double M1[2][2] = {{1.0/3.0, 1.0/6.0}, {1.0/6.0, 1.0/3.0}}; // Ещё одна дополнительная матрица

double lambda\_aver = 0; //Усреднённое лямбда

double sigma\_aver = 0; //Усреднённое сигма

double epsilon\_aver = 0; //Усреднённое эпсилон

//Получаем средние значения

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

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

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

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

lambda\_aver += lambda(x,y,z);

sigma\_aver += sigma(x,y,z);

epsilon\_aver += epsilon(x, y, z);

}

lambda\_aver /= 8.0;

sigma\_aver /= 8.0;

epsilon\_aver /= 8.0;

//Получение матриц G и M

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] \*= lambda\_aver;

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

}

}

//Собираем локальную матрицу, как блочную

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\*epsilon\_aver\*M[i][j];

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

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

}

}

//Теперь начинаем собрать локлаьную правую часть

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

//Вычисляем значения

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

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

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

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

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

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

}

//Вычисляем подвекторы правой части

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];

}

}

//Соединяем два вектора в один

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; //координаты точки

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

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

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 solver\_type> int harm\_FEM<solver\_type>::mu(int i){

return i % 2;

}

template <typename solver\_type> int harm\_FEM<solver\_type>::nu(int i){

return (i/2) % 2;

}

template <typename solver\_type> int harm\_FEM<solver\_type>::v(int i){

return (i/4) % 2;

}

template <typename solver\_type> void harm\_FEM<solver\_type>::gen\_loc\_thi(double \*\*A\_loc, double \*b\_loc, int face\_n){

double hx;

double hy;

//Определяем оринтацию грани

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{ //Если в плоскости xOy

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);

}

}

//Матрица массы для обычного МКЭ

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];

//Формируем матрицу массы

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;

}

}

//Формируем правую часть

double b\_both[8]; //Вектор значений

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

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

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

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

}

//Получаем правую часть

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 solver\_type> void harm\_FEM<solver\_type>::gen\_loc\_sec(double \*b\_loc, int face\_n){

double hx;

double hy;

double A\_loc[8][8]; //Большая матричка

//Определяем оринтацию грани

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{ //Если в плоскости xOy

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);

}

}

//Матрица массы для обычного МКЭ

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

//Формируем матрицу массы

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;

}

}

//Формируем правую часть

double b\_both[8]; //Вектор значений

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

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

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

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

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

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

}

//Получаем правую часть

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 solver\_type> int harm\_FEM<solver\_type>::find\_el\_pos(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 solver\_type> void harm\_FEM<solver\_type>::transf\_grid(string file\_cords, string file\_elements, string file\_faces){

//Преобразование файла координат

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

FILE\* cords\_f\_out = fopen((file\_cords + "tr").c\_str(), "w"); //Файл модифицированной сетки

int n; //количество

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; //считываем координаты

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);

//Преобразование файла элементов

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

FILE\* els\_f\_out = fopen((file\_elements+"tr").c\_str(), "w"); //Файл модифицированной сетки

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; //текущая вершина

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);

//Преобразование файла граней

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

FILE\* face\_f\_out = fopen((file\_faces+"tr").c\_str(), "w"); //Файл модифицированной сетки

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; //текущая вершина

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 solver\_type> double harm\_FEM<solver\_type>::u\_betta\_s(double x, double y, double z, int face\_n){

double val; //итоговое значение

return val;

}

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

double val; //итоговое значение

return val;

}

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

double val; //итоговое значение

return val;

}

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

double val; //итоговое значение

return val;

}

template <typename solver\_type> void harm\_FEM<solver\_type>::out\_rez(string file\_name){

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

for(int i = 0; i < nodes\_n; i++)

fprintf(out\_f, "%d\t%.15lf\n", i, solution[i]);

fclose(out\_f);

}

template <typename solver\_type> void harm\_FEM<solver\_type>::set\_sol(func us, func uc){

u\_sin = us;

u\_cos = uc;

}

## Файл «solver\_LU.h»

#pragma once

#include <math.h>

//Решатель методом LU-разложения для СЛАУ с несимметричной матрицей в профильном формате

class solver\_LU{

public:

void init(int\* s\_ig, int\* s\_jg, double\* s\_gu, double\* s\_gl, double\* s\_di, int s\_n); //инициализация, перевод из разреженного в профильный формат

void set\_rp(double\* s\_rp); //Установка правой части

void solve(double \*&solution, int &its); //Получение решения и количества итераций

private:

int n; //Размерность СЛАУ

//Массивы

int \*ig;

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

double \*rp;

void dec(); //Простроение LU-разложения

};

## Файл «solver\_LU.h»

#pragma once

#include <math.h>

//Решатель ЛОС, для СЛАУ с несимметричной матрицей в разреженном формате c LU - предобуславливанием

class LOS{

public:

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

void set\_rp(double\* s\_rp); //Установка правой части

void solve(double \*&solution, int &its); //Получение решения и количества итераций

private:

int n; //Размерность СЛАУ

//Основные массивы

int \*ig, \*jg;

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

double \*rp;

//Массивы для преобславливателя

double \*Uu, \*Ll, \*Ld;

void precond(); //Вычисление матриц L и U

double dot\_prod(double \*a, double \*b); //скалярное произведение

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

void solve\_L(double \*f, double \*&x); //Lx = f, прямой ход

void solve\_U(double \*f, double \*&x); //Ux = f, обратный ход

};

## Файл «solver\_BCGStab.h»

#pragma once

#include <math.h>

//Решатель BSGStab, для СЛАУ с несимметричной матрицей в разреженном формате c LU - предобуславливанием

class BCGStab{

public:

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

void set\_rp(double\* s\_rp); //Установка правой части

void solve(double \*&solution, int& its); //Получение решения и количества итераций

private:

int n; //Размерность СЛАУ

//Основные массивы

int \*ig, \*jg;

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

double \*rp;

//Массивы для преобславливателя

double \*Uu, \*Ll, \*Ld;

void precond(); //Вычисление матриц L и U

double dot\_prod(double \*a, double \*b); //скалярное произведение

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

void solve\_L(double \*f, double \*&x); //Lx = f, прямой ход

void solve\_U(double \*f, double \*&x); //Ux = f, обратный ход

};

## Файл «solver\_LU.cpp»

#include "solver\_LU.h"

void solver\_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];

//Перенос диагонали

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

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

//Формируем новый массив ig

ig[0] = 0;

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

int k = s\_ig[i] - s\_ig[i-1]; //Количество не нулевых элементов в строке

if(k > 0){ //Если такие есть

int total\_n = i - s\_jg[s\_ig[i-1]]; // вычисляем сколько всего элементов в строке, как в профильной

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

}

else

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

}

gu = new double [ig[n]];

gl = new double [ig[n]];

//Формируем новые gl и gu

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

int j\_s = ig[i]; // начало строки

int j\_e = ig[i+1]; //конец

int column = i - (j\_e - j\_s); //номер текущей колонки

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 solver\_LU::set\_rp(double \*s\_rp){

rp = new double [n];

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

rp[i] = s\_rp[i];

}

void solver\_LU::solve(double \*&solution, int &its){

solution = new double [n];

//Прямой ход

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];

}

//Обраный ход

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 solver\_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;

}

}

## Файл «solver\_LOS.cpp»

#include "solver\_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::set\_rp(double \*s\_rp){

rp = s\_rp;

}

void LOS::precond(){

double sum\_l, sum\_u, sum\_d; //Промежуточные переменные, для вычисления сумм

int copy\_end = ig[n];

Ll = new double [copy\_end];

Uu = new double [copy\_end];

Ld = new double [n];

//Копируем старые в новые

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::solve\_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::solve\_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::solve(double \*&solution, int &its){

//Параметры решателя

int max\_iter = 100000;

double eps = 1E-16;

double end\_cycle = false;

//Норма правой части, для выхода

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

//Начинаем решение

double\* x0 = new double [n]; //Приближение

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

x0[i] = 0;

solution = new double [n];

double\* r = new double [n]; //Вектор невязки

double\* z = new double [n];

double\* p = new double [n];

double\* s = new double [n]; //Вспомогательный вектор

double\* t = new double [n]; //Вспомогательный вектор

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

mull\_A(x0, s);

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

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

solve\_L(s, r);

//z0 = U^(-1)r0

solve\_U(r, z);

//p0 = L^(-1)Az0

mull\_A(z, s);

solve\_L(s, p);

int iter;

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

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

if( discr / rp\_norm > eps){ //Проверка условия выхода

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])

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

mull\_A(s, t);

solve\_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];

solve\_L(s, r);

//z0 = U^(-1)r0

solve\_U(r, z);

//p0 = L^(-1)Az0

mull\_A(z, s);

solve\_L(s, p);

}

}

else{

end\_cycle = true;

}

}

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

solution[i] = x0[i];

its = iter;

//И отчишаем память

delete[] x0;

delete[] p;

delete[] r;

delete[] z;

delete[] s;

delete[] t;

}

Файл «solver\_BCGStab.cpp»

#include "solver\_BCGStab.h"

void BCGStab::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 BCGStab::set\_rp(double \*s\_rp){

rp = s\_rp;

}

void BCGStab::precond(){

double sum\_l, sum\_u, sum\_d; //Промежуточные переменные, для вычисления сумм

int copy\_end = ig[n];

Ll = new double [copy\_end];

Uu = new double [copy\_end];

Ld = new double [n];

//Копируем старые в новые

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

double dp = 0;

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

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

return dp;

}

void BCGStab::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 BCGStab::solve\_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 BCGStab::solve\_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 BCGStab::solve(double \*&solution, int &its){

//Параметры решателя

int max\_iter = 100000;

double eps = 1E-16;

double end\_cycle = false;

//Норма правой части, для выхода

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

//Начинаем решение

double\* x0 = new double [n]; //Приближение

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

x0[i] = 0;

solution = new double [n];

double\* r = new double [n]; //Вектор невязки

double\* r0 = new double [n]; //Вектор начальной невязки

double\* z = new double [n];

double\* p = new double [n];

double\* s = new double [n]; //Вспомогательный вектор

double\* t = new double [n]; //Вспомогательный вектор

double\* w = new double [n]; //Вспомогательный вектор

//r0 = f - Ax0

//z0 = r0

mull\_A(x0, s);

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

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

solve\_L(w, r0);

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

r[i] = z[i] = r0[i];

double dot1, dot2;

dot1 = dot\_prod(r0, r0);

int iter;

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

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

if(discr / rp\_norm > eps){ //Проверка условия выхода

//s = L^(-1)AU^(-1)z[k-1]

solve\_U(z, s);

mull\_A(s, t);

solve\_L(t, s);

double alpha = dot1 / dot\_prod(r0, s); // a = (r[k-1], r[0]) / (r[0], L^(-1)AU^(-1)z[k-1])

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

p[i] = r[i] - alpha\*s[i]; //p[k] = r[k-1] - a\*L^(-1)AU^(-1)z[k-1]

//t = L^(-1)AU^(-1)p[k]

solve\_U(p, t);

mull\_A(t, w);

solve\_L(w, t);

double gamma = dot\_prod(p, t) / dot\_prod(t, t); //g = (p[k], L^(-1)AU^(-1)p[k]) / (L^(-1)AU^(-1)p[k], L^(-1)AU^(-1)p[k])

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

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

w[i] = r[i];

r[i] = p[i] - gamma\*t[i]; //r[k] = p[k] - g\*L^(-1)AU^(-1)p[k]

}

dot2 = dot\_prod(r, r0); // (r[k], r[0])

double betta = alpha\*dot2 / (gamma \* dot1); //b = a\*(r[k],r[0]) / (g\*(r[k-1], r[0])

dot1 = dot2;

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

z[i] = r[i] + betta\*w[i] - betta\*gamma \* s[i]; //z[k] = r[k] + b\*r[k-1] - b\*g\*L^(-1)AU^(-1)z[k-1]

}

else{

end\_cycle = true;

}

}

solve\_U(x0, solution);

its = iter;

//И отчишаем память

delete[] x0;

delete[] p;

delete[] r;

delete[] z;

delete[] s;

delete[] t;

delete[] w;

}