МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

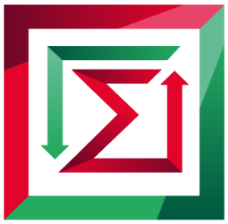
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ  НЭТИ

Кафедра прикладной математики

Курсовой проект

по дисциплине «Уравнения математической физики»



Группа	ПМ-81
Студент	ЮРГАНОВ ЕГОР
Препода-	ПАТРУШЕВ ИЛЬЯ ИГОРЕВИЧ
Дата	07.10.2021

ВВЕДЕНИЕ

ЦЕЛЬ

Приобрести навыки численного решения начально-краевых задач для уравнений гиперболического и параболического типа в неоднородных одномерных, двумерных и трехмерных областях с помощью метода конечных элементов при использовании различных схем дискретизации по времени.

ЗАДАНИЕ

МКЭ для параболического уравнения в декартовой системе координат, четырехслойная неявная схема по времени, линейные базисные функции на треугольниках.

1. ТЕОРИЯ

1.1. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ФИЗИЧЕСКИХ ПРОЦЕССОВ В ВИДЕ КРАЕВЫХ ЗАДАЧ

Параболическая краевая задача для функции u определяется дифференциальным уравнением

$$\sigma \frac{\partial u}{\partial t} - \operatorname{div}(\lambda \operatorname{grad} u) = f \tag{1.1}$$

заданным в некоторой области Ω с границей $S = S_1 \cup S_2 \cup S_3$, и краевыми условиями

$$u|_{S_1} = u_g \tag{1.2}$$

$$\lambda \frac{\partial}{\partial n}\Big|_{S_2} = \theta \tag{1.3}$$

$$\lambda \frac{\partial}{\partial n}\Big|_{S_3} + \beta\left(u|_{S_3} - u_\beta\right) = 0 \tag{1.4}$$

$$u(t)|_{t=t_0} = u^0 \tag{1.5}$$

в которых $u|_{S_i}$ – значение искомой функции u на границе S_i, а $\frac{\partial}{\partial n}\Big|_{S_i}$ – значение на S_i производной функции u по направлению внешней нормали к поверхности S_i. Коэффициент λ называют коэффициентом диффузии. u^0 – заданная функция пространственных координат.

1.2. О ДИСКРЕТИЗАЦИИ ПО ВРЕМЕНИ

При построение дискретных аналогов начально-краевая задача для дифференциального уравнения (1.1) будем полагать, что ось времени t разбита на временные слои значениями t_j, $j = 1, \dots, J$, а значения искомой функции u и параметров λ, σ, f дифференциальных уравнений на $j-$ м временном слое будем обозначать соответственно через u^j, λ^j, σ^j, f^j, которые уже не зависят от времени t, но остаются функциями пространственных координат.

Помимо краевых условий начально-краевая задача для параболического уравнения должна включать в себя начальное условие (1.5).

$$-\mathrm{div}\left(\lambda \mathrm{grad}\, u\right) \xrightarrow{\text{МКЭ-аппроксимация}} \mathbf{G}\cdot\mathbf{q},$$

$$\gamma u \xrightarrow{\text{МКЭ-аппроксимация}} \mathbf{M}\cdot\mathbf{q},$$

Рисунок 1 – матрицы M и G

Для описания вычислительных процедур построения конечноэлементного решения задачи (1.1) удобно ввести глобальные матрицы: жесткости G и массы M (Рисунок 1).

1.3. АППРОКСИМАЦИЯ ДЛЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА

Рассмотрим аппроксимацию дифференциального уравнения по времени с использованием неявной схемы:

$$\sigma \frac{u^j - u^{j-1}}{\Delta t} - \mathrm{div}\left(\lambda\, \mathrm{grad}\, u^j\right) = f^j, j = 1 \dots J \qquad (1.6)$$

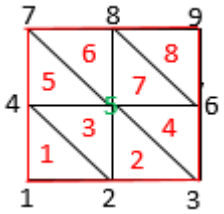
Рассмотрим процедуру построения неявной четыреххслойной схемы для решения дифференциального уравнения параболического типа.

Будем считать, что функция u является функцией двух пространственных координат x и y.

Представим искомое решение u на интервале $(t_{j-2},\ t_j)$ в следующем виде:

2. ИССЛЕДОВАНИЯ

2.1. ИССЛЕДОВАНИЯ НА ПОРЯДОК АППРОКСИМАЦИИ



(Рисунок 2 – Рассматриваемый элемент)

Таблица 1 – Координаты узлов

№ узла	Координаты узла
1	0 0
2	2 0
3	4 0
4	0 2
5	2 2
6	4 2
7	0 4
8	2 4
9	4 4

Таблица 2 – Полином 1-ой степени

u	f	time
t * x	x	0 1 2 3
q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001
12.0000000000000	12.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001
12.0000000000000	12.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001
12.0000000000000	12.0000000000000	0.00000000000000

Таблица 3 – Полином 2-ой степени

u	f	time
$t^2 * x$	2tx	0 1 2 3

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
18.0000000000000	18.0000000000000	0.00000000000000
36.0000000000000	36.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
18.0000000000000	18.0000000000000	0.00000000000000
36.0000000000000	36.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
18.0000000000000	18.0000000000000	0.00000000000000
36.0000000000000	36.0000000000000	0.00000000000000

Таблица 4 – Полином 4-ой степени

u	f	time
$t^4 * x$	$4t^3x$	0 1 2 3
q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
162.000000000000	162.000000000000	0.00000000000000
324.000000000000	324.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
168.260869565217	162.000000000000	6.26086956521700
324.000000000000	324.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
162.000000000000	162.000000000000	0.00000000000000
324.000000000000	324.000000000000	0.00000000000000

2.2. ИССЛЕДОВАНИЯ НА ПОРЯДОК СХОДИМОСТИ

Таблица 5

u	f	time
x+y+sint	cost	0 1 2 3
q	u	|q-u|
0.14112000805986	0.14112000805986	0.00000000000000
2.14112000805986	2.14112000805986	0.00000000000000
4.14112000805986	4.14112000805986	0.00000000000000
2.14112000805986	2.14112000805986	0.00000000000000
4.25432346469713	4.14112000805986	0.11320345663720
6.14112000805986	6.14112000805986	0.00000000000000
4.14112000805986	4.14112000805986	0.00000000000000
6.14112000805986	6.14112000805986	0.00000000000000
8.14112000805986	8.14112000805986	0.00000000000000

Таблица 6

u	f	time
x+y+sint	cost	0 0.5 1 1.5
q	u	|q-u|
0.99749498660405	0.99749498660405	0.00000000000000
2.99749498660406	2.99749498660406	0.00000000000000

4.99749498660406	4.99749498660406	0.00000000000000
2.99749498660406	2.99749498660406	0.00000000000000
5.00589670795262	4.99749498660406	0.00840172134856
6.99749498660406	6.99749498660406	0.00000000000000
4.99749498660406	4.99749498660406	0.00000000000000
6.99749498660406	6.99749498660406	0.00000000000000
8.99749498660405	8.99749498660405	0.00000000000000

Таблица 7

u	f	time
x+y+sint	cost	0 0.25 0.5 0.75
q	u	|q-u|
0.68163876002333	0.68163876002333	0.00000000000000
2.68163876002334	2.68163876002334	0.00000000000000
4.99749498660406	4.99749498660406	0.00000000000000
2.68163876002334	2.68163876002334	0.00000000000000
4.68200033211464	4.68163876002334	0.00036157209130
6.68163876002333	6.68163876002333	0.00000000000000
4.68163876002334	4.68163876002334	0.00000000000000
6.68163876002333	6.68163876002333	0.00000000000000
8.68163876002334	8.68163876002334	0.00000000000000

$$\frac{0.11320345663720}{0.00840172134856} \approx 14 \quad \frac{0.00840172134856}{0.00036157209130} \approx 23 \quad \frac{0.00036157209130}{0.00001305001631} \approx 28 \quad (2.1)$$

ВЫВОД

При увеличении степени полинома искомой функции увеличивается погрешность. Это связано с тем, что базисные функции линейные и на заданных точках точное решение достигается вплоть до полинома четвертой степени, это видно по Таблица 4. Порядок сходимости равен 3.

3. ТЕКСТ ПРОГРАММЫ

STRUCTS_NUMS_OPERATIONS.H

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <windows.h>
#include <iostream>
#include <set>
#include <vector>
#include <fstream>
#include <algorithm>

using namespace std;

int num_knots, num_lambda, triangle_num, num_bounds_1, num_bounds_2, num_bounds_3,
n;

struct knot {
    double x = 0, y = 0;
};

struct triangle {
    int knot_nums[4]{};  //4 - номер подобласти
};

struct local {
    vector<int> knot_nums;
    vector<vector<double>> A;
    vector<double> b;
};

struct SLAE {
    vector<int> jg, ig;
    vector<double> ggl, ggu, b, di;
    SLAE() {
        ig.resize(num_knots + 1);
        di.resize(num_knots);
        b.resize(num_knots);
        for (int i = 0; i < num_knots; i++) {
            di[i] = 0;
            b[i] = 0;
        }
    }
};

vector<double> operator + (vector<double> vector_1, const vector<double>& vec-
tor_2) {
    size_t size = vector_1.size();
    for (size_t i = 0; i < size; ++i)
        vector_1[i] -= vector_2[i];
    return vector_1;
}
```

```
vector<double>& operator += (vector<double>& vector_1, const vector<double>& vec-
tor_2) {
    size_t size = vector_1.size();
    for (size_t i = 0; i < size; ++i)
        vector_1[i] += vector_2[i];
    return vector_1;
}

vector<double> operator * (const double& w, vector<double> vector) {
    size_t size = vector.size();
    for (size_t i = 0; i < size; ++i)
        vector[i] *= w;
    return vector;
}

double scalar(vector<double> x1, vector<double> x2) {
    double sum = 0;
    for (int i = 0; i < num_knots; i++)
        sum += x1[i] * x2[i];
    return sum;
}
```

KNOTS_TRIAD_COEFF.H

```
#pragma once
#include "structs_nums_operations.h"

void read_knots(vector<knot>& knots) {
    ifstream input_knots("knots.txt");
    input_knots >> num_knots;
    knots.resize(num_knots);
    for (int i = 0; i < num_knots; i++)
        input_knots >> knots[i].x >> knots[i].y;
}

void create_lambda_f_gamma(vector<int>& lambda, vector<int>& f, vector<int>&
gamma) {
    int temp;
    ifstream lambda_file("coef.txt");
    lambda_file >> num_lambda;
    lambda.resize(num_lambda);
    for (int i = 0; i < num_lambda; i++) {
        lambda_file >> temp;
        lambda[i] = temp;
        // cout << lambda[i];
    }
    f.resize(num_lambda);
    for (int i = 0; i < num_lambda; i++) {
        lambda_file >> temp;
        f[i] = temp;
        // cout << f[i];
    }
    gamma.resize(num_lambda);
    for (int i = 0; i < num_lambda; i++) {
        lambda_file >> temp;
        gamma[i] = temp;
        // cout << gamma[i];
    }
```

```
}

void create_time_vector(vector<double>& time) {
    double temp;
    ifstream time_file("time.txt");
    for (int i = 0; i < 4; i++) {
        time_file >> temp;
        time[i] = temp;
        // cout << time[i];
    }
}

void create_triangles(vector<triangle>& triangle_list) {
    ifstream f("triangles.txt");
    f >> triangle_num;
    triangle_list.resize(triangle_num);
    int number;
    for (int i = 0; i < triangle_num; i++) {
        triangle triad;
        for (int j = 0; j < 3; j++) {
            f >> number;
            triad.knot_nums[j] = number - 1;
        }
        f >> number;
        triad.knot_nums[3] = number - 1; //подобласть
        triangle_list[i] = triad;
    }
}

double func_lambda(int number_f) {
    switch (number_f) {
    case 0: {
        return 0;
        break;
    }
    case 1: {
        return 1;
        break;
    }
    }
}

double func_f(int number_f, knot knots, double time) {
    switch (number_f) {
    case 0: {
        return 0;
        break;
    }
    case 1: {
        return cos(time);
        break;
    }
    case 2: {
        return (knots.x + knots.y);
        break;
    }
    }
}

double func_gamma(int number_f) {
```

```
    switch (number_f) {
    case 0: {
        return 0;
        break;
    }
    case 1: {
        return 1;
        break;
    }
    case 2: {
        return 2;
        break;
    }
    }
}
```

BOUNDS.H

```
#pragma once
#include "knots_triad_coeff.h"

void read_bounds(vector<local>& vector_bounds, ifstream& input, int& num_bounds,
int flag) {
    input >> num_bounds;
    vector_bounds.resize(num_bounds);
    int num;
    for (int i = 0; i < num_bounds; i++) {
        local local_bound;
        if (flag == 3)
            local_bound.knot_nums.resize(4);
        else
            local_bound.knot_nums.resize(3);
        for (int j = 0; j < 2; j++) {//первые два - номера узлов ребра, 3,4 - зна-
чение
            input >> num;                    //для 3 кр 3 и 4 - номер функции ub,
для 2 кр - тетта
            local_bound.knot_nums[j] = num - 1;
        }
        input >> local_bound.knot_nums[2];
        if (flag == 3)
            input >> local_bound.knot_nums[3];
        vector_bounds[i] = local_bound;
    }
}

double ub(knot knots, int num, double time) {
    switch (num) {
    case 1: {
        return -5 * time;
        break;
    }
    case 2: {
        return sin(time) + knots.x + knots.y;
        break;
    }
    case 3: {
        return 55 * time * time + 2 * time;
        break;
    }
```

```
    case 4: {
        return pow(knots.x * knots.y, 4);
        break;
    }
    default: {
        return 1;
        break;
    }
    }
}

void build_bound(vector<local>& vector_bound, int flag, vector<knot> knots, double
betta, double time) {
    int iA = 0;
    for (vector <local>::iterator iter = vector_bound.begin(); iter != vec-
tor_bound.end(); iter++, iA++) {
        local bounds = *iter;

        bounds.b.resize(2);

        double h = sqrt((knots[bounds.knot_nums[1]].x -
knots[bounds.knot_nums[0]].x) * (knots[bounds.knot_nums[1]].x -
knots[bounds.knot_nums[0]].x)
            + (knots[bounds.knot_nums[1]].y - knots[bounds.knot_nums[0]].y) *
(knots[bounds.knot_nums[1]].y - knots[bounds.knot_nums[0]].y));

        if (flag == 3) { //только для 3 краевых
            bounds.A.resize(2);
            for (int i = 0; i < 2; i++)
                bounds.A[i].resize(2);

            //bounds.knot_nums[2] - B и тд
            bounds.A[0][0] = bounds.A[1][1] = betta * h / 3;
            bounds.A[1][0] = bounds.A[0][1] = betta * h / 6;


            for (int i = 0; i < 2; i++) {
                for (int j = 0; j < 2; j++)
                    cout << bounds.A[i][j] << " ";
                cout << endl;
            }

            bounds.b[0] = betta * h * (2 * ub(knots[bounds.knot_nums[0]],
bounds.knot_nums[2], time) + ub(knots[bounds.knot_nums[1]], bounds.knot_nums[3],
time)) / 6;
            bounds.b[1] = betta * h * (ub(knots[bounds.knot_nums[0]],
bounds.knot_nums[2], time) + 2 * ub(knots[bounds.knot_nums[1]],
bounds.knot_nums[3], time)) / 6;

        }
        else {//вторые
            bounds.b[0] = h * (2 * ub(knots[bounds.knot_nums[0]],
bounds.knot_nums[2], time) + ub(knots[bounds.knot_nums[1]], bounds.knot_nums[3],
time)) / 6;
            bounds.b[1] = h * (ub(knots[bounds.knot_nums[0]], bounds.knot_nums[2],
time) + 2 * ub(knots[bounds.knot_nums[1]], bounds.knot_nums[3], time)) / 6;
        }
        vector_bound[iA] = bounds;
    }
}
```

```
void use_bounds(vector<local>& vector_bound, vector<set<int>>& L, SLAE& slae, int
bound_num, vector<knot> knots) {
    if (bound_num == 3) { // 3 краевые условия
        for (vector <local>::iterator iter = vector_bound.begin(); iter != vec-
tor_bound.end(); iter++) {
            local bound_iter = *iter;

            //заносим высе диагональные элементы
            for (int k = 0; k < 2; k++) {
                slae.di[bound_iter.knot_nums[k]] += bound_iter.A[k][k];
                slae.b[bound_iter.knot_nums[k]] += bound_iter.b[k];
            }
            //начинаем цикл по строкам нижнего
            for (int i = 0; i < 2; i++) {
                //устанавливаем начальное значение нижней границы поиска
                int ibeg = slae.ig[bound_iter.knot_nums[i]];

                for (int j = 0; j < i; j++) { // do j=1,i-1
                    int iend = slae.ig[bound_iter.knot_nums[i] + 1] - 1;
                    while (slae.jg[ibeg] != bound_iter.knot_nums[j]) {
                        int ind = (ibeg + iend) / 2;
                        if (slae.jg[ind] < bound_iter.knot_nums[j])
                            ibeg = ind + 1;
                        else
                            iend = ind;
                    }
                    slae.ggu[ibeg] += bound_iter.A[j][i];
                    slae.ggl[ibeg] += bound_iter.A[i][j];
                    ibeg++;
                }
            }
        }
    }
    if (bound_num == 2) { // 2 краевые условия
        for (vector <local>::iterator iter = vector_bound.begin(); iter != vec-
tor_bound.end(); iter++) {
            local bound_iter = *iter;

            //заносим высе диагональные элементы
            for (int k = 0; k < 2; k++)
                slae.b[bound_iter.knot_nums[k]] += bound_iter.b[k];
        }
    }
}

void use_first_bounds(vector<local>& vector_bound, vector<set<int>> L, SLAE& slae,
vector<knot> knots, double time) {
    for (vector <local>::iterator iter = vector_bound.begin(); iter != vec-
tor_bound.end(); iter++) {
        local bound = *iter;
        for (int i = 0; i < 2; i++) {
            slae.di[bound.knot_nums[i]] = 1;
            slae.b[bound.knot_nums[i]] = ub(knots[bound.knot_nums[i]],
bound.knot_nums[2], time); //ub(knot*& knots, int i, int num)
        }

        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < n; j++) {
```

```
                // cout << "jg" << slae.jg[j] << "  nums" << bound.knot_nums[i] <<
" ";
                if (slae.jg[j] == bound.knot_nums[i])
                    slae.ggu[j] = 0;
            }

            //устанавливаем начальное значение нижней границы поиска
            int ibeg = slae.ig[bound.knot_nums[i]];
            int iend = slae.ig[bound.knot_nums[i] + 1];
            for (int j = ibeg; j < iend; j++)
                slae.ggl[j] = 0;
        }
    }
}
```

NEW_KURSACH.CPP

```
#include "knots_triad_coeff.h"
#include "bounds.h"
#include <iomanip>

double determinant(triangle& triad, vector<knot> knots) { //detD = (x2-x1)(y3-y1)-
(x3-x1)(y2-y1)
    return (knots[triad.knot_nums[1]].x - knots[triad.knot_nums[0]].x) *
        (knots[triad.knot_nums[2]].y - knots[triad.knot_nums[0]].y) -
        (knots[triad.knot_nums[2]].x - knots[triad.knot_nums[0]].x) *
        (knots[triad.knot_nums[1]].y - knots[triad.knot_nums[0]].y);
}

void local_G(vector<vector<double>>& G, triangle& triad, double det, double
lambda, vector<knot> knots) {
    vector<vector<double>> a;   //a = D^-1
    a.resize(3);
    for (int i = 0; i < 3; i++)
        a[i].resize(2);

    // в методичке
    a[0][0] = (knots[triad.knot_nums[1]].y - knots[triad.knot_nums[2]].y) / (det);
//y2-y3 / det
    a[0][1] = (knots[triad.knot_nums[2]].x - knots[triad.knot_nums[1]].x) / (det);
//x3-x2 / det
    a[1][0] = (knots[triad.knot_nums[2]].y - knots[triad.knot_nums[0]].y) / (det);
//y3-y1 / det
    a[1][1] = (knots[triad.knot_nums[0]].x - knots[triad.knot_nums[2]].x) / (det);
//x1-x3 / det
    a[2][0] = (knots[triad.knot_nums[0]].y - knots[triad.knot_nums[1]].y) / (det);
//y1-y2 / det
    a[2][1] = (knots[triad.knot_nums[1]].x - knots[triad.knot_nums[0]].x) / (det);
//x2-x1 / det

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++) {
            G[i][j] = lambda * abs(det) * (a[i][0] * a[j][0] + a[i][1] * a[j][1])
/ 2;
        }
    /*
    cout << "matrix G\n";
    for (int i = 0; i < 3; i++) {
```

```
        for (int j = 0; j < 3; j++)
            cout << setprecision(15) << G[i][j] << " ";
        cout << "\n";
    }
    cout << "\n";*/
}

void local_M(vector<vector<double>>& M, double det, double gamma) {
    M[0][0] = M[1][1] = M[2][2] = gamma * abs(det) / 12;
    M[0][1] = M[1][0] = M[0][2] = M[2][0] = M[2][1] = M[1][2] = gamma * abs(det) /
24;
    /*
    cout << "matrix M\n";
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
        {
            cout << setprecision(15) << M[i][j] << " ";
        }
        cout << "\n";
    }*/
}

vector<double> A_on_vector(vector<vector<double>> A, vector<double> q) {
    vector<double> res;
    res.resize(q.size());
    for (int i = 0; i < res.size(); i++) {
        res[i] = 0;
        for (int j = 0; j < res.size(); j++)
            res[i] += A[i][j] * q[j];
    }
    return res;
}

void local_A_2(vector<local>& local_A_list, vector<triangle> triangle_list, vec-
tor<int> lambda_vector, vector<knot> knots, vector<int> gamma_v, vector<int> f,
vector<double> q, vector<double> time) {
    local_A_list.resize(triangle_num);
    int iA = 0;
    double delta_t = 1. / (time[1] - time[0]);
    for (vector <triangle>::iterator iter = triangle_list.begin(); iter != trian-
gle_list.end(); iter++, iA++) {
        triangle triad = *iter;
        local local_A;

        double lambda = func_lambda(lambda_vector[triad.knot_nums[3]]),
            det = determinant(triad, knots);

        vector<vector<double>> M, G;
        M.resize(3);
        G.resize(3);
        for (int i = 0; i < 3; i++) {
            M[i].resize(3);
            G[i].resize(3);
        }

        double gamma = func_gamma(gamma_v[triad.knot_nums[3]]);

        local_M(M, det, gamma);
        local_G(G, triad, det, lambda, knots);
```

```
        local_A.knot_nums.resize(3);
        for (int i = 0; i < 3; i++)
            local_A.knot_nums[i] = triad.knot_nums[i];

        local_A.A.resize(3);
        for (int i = 0; i < 3; i++) {
            local_A.A[i].resize(3);
            for (int j = 0; j < 3; j++)
                local_A.A[i][j] = G[i][j] + delta_t * M[i][j];
        }

        local_A.b.resize(3);
        //локальный вектор b = f * C
        double f1 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[0]],
time[1]),
            f2 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[1]],
time[1]),
            f3 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[2]],
time[1]);

        vector<double> q_temp(3);
        for (int i = 0; i < 3; i++)
            q_temp[i] = q[local_A.knot_nums[i]];

        q_temp = A_on_vector(M, q_temp);

        local_A.b[0] = abs(det) * (2. * f1 + f2 + f3) / 24. + delta_t * q_temp[0];
        local_A.b[1] = abs(det) * (f1 + 2. * f2 + f3) / 24. + delta_t * q_temp[1];
        local_A.b[2] = abs(det) * (f1 + f2 + 2. * f3) / 24. + delta_t * q_temp[2];

        local_A_list[iA] = local_A;
    }
}

void local_A_3(vector<local>& local_A_list, vector<triangle> triangle_list, vec-
tor<int> lambda_vector, vector<knot> knots, vector<int> gamma_v, vector<int> f,
vector<vector<double>> q, vector<double> time) {
    local_A_list.resize(triangle_num);

    int iA = 0;
    double delta_t = time[2] - time[0],
        delta_t1 = time[1] - time[0],
        delta_t0 = time[2] - time[1],
        t1 = (delta_t + delta_t0) / (delta_t * delta_t0),
        t2 = delta_t0 / (delta_t * delta_t1),
        t3 = delta_t / (delta_t0 * delta_t1);

    for (vector<triangle>::iterator iter = triangle_list.begin(); iter != trian-
gle_list.end(); iter++, iA++) {
        triangle triad = *iter;
        local local_A;

        double lambda = func_lambda(lambda_vector[triad.knot_nums[3]]),
            det = determinant(triad, knots);

        vector<vector<double>> M, G;
        M.resize(3);
        G.resize(3);
        for (int i = 0; i < 3; i++) {
            M[i].resize(3);
```

```
            G[i].resize(3);
        }

        double gamma = func_gamma(gamma_v[triad.knot_nums[3]]);

        local_M(M, det, gamma);
        local_G(G, triad, det, lambda, knots);

        local_A.knot_nums.resize(3);
        for (int i = 0; i < 3; i++)
            local_A.knot_nums[i] = triad.knot_nums[i];

        local_A.A.resize(3);
        for (int i = 0; i < 3; i++) {
            local_A.A[i].resize(3);
            for (int j = 0; j < 3; j++)
                local_A.A[i][j] = G[i][j] + t1 * M[i][j];
        }

        local_A.b.resize(3);
        //локальный вектор b = f * C
        double f1 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[0]],
time[2]),
            f2 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[1]],
time[2]),
            f3 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[2]],
time[2]);

        vector<double> q_temp0(3), q_temp1(3);

        for (int i = 0; i < 3; i++) {
            int temp = local_A.knot_nums[i];
            q_temp0[i] = q[0][temp];
            q_temp1[i] = q[1][temp];
        }

        q_temp0 = A_on_vector(M, q_temp0);
        q_temp1 = A_on_vector(M, q_temp1);

        local_A.b[0] = abs(det) * (2 * f1 + f2 + f3) / 24 - t2 * q_temp0[0] + t3 *
q_temp1[0];
        local_A.b[1] = abs(det) * (f1 + 2 * f2 + f3) / 24 - t2 * q_temp0[1] + t3 *
q_temp1[1];
        local_A.b[2] = abs(det) * (f1 + f2 + 2 * f3) / 24 - t2 * q_temp0[2] + t3 *
q_temp1[2];

        local_A_list[iA] = local_A;
    }
}

void local_A_4(vector<local>& local_A_list, vector<triangle> triangle_list, vec-
tor<int> lambda_vector, vector<knot> knots, vector<int> gamma_v, vector<int> f,
vector<vector<double>> q, vector<double> time) {
    local_A_list.resize(triangle_num);

    int iA = 0;
    double delta_t3 = (3 * time[3] * time[3] - 2 * time[3] * (time[2] + time[1] +
time[3]) + time[1] * (time[3] + time[2]) + time[3] * time[2])
        / ((time[0] - time[1]) * (time[0] - time[2]) * (time[0] - time[3])),
```

```
        delta_t2 = (3 * time[3] * time[3] - 2 * time[3] * (time[2] + time[3] +
time[0]) + time[0] * (time[3] + time[2]) + time[3] * time[2])
        / ((time[1] - time[0]) * (time[1] - time[2]) * (time[1] - time[3])),
        delta_t1 = (3 * time[3] * time[3] - 2 * time[3] * (time[3] + time[1] +
time[0]) + time[0] * (time[1] + time[3]) + time[1] * time[3])
        / ((time[2] - time[0]) * (time[2] - time[1]) * (time[2] - time[3])),
        delta_t = (3 * time[3] * time[3] - 2 * time[3] * (time[2] + time[1] +
time[0]) + time[0] * (time[1] + time[2]) + time[1] * time[2])
        / ((time[3] - time[0]) * (time[3] - time[1]) * (time[3] - time[2]));

    for (vector<triangle>::iterator iter = triangle_list.begin(); iter != trian-
gle_list.end(); iter++, iA++) {
        triangle triad = *iter;
        local local_A;

        double lambda = func_lambda(lambda_vector[triad.knot_nums[3]]),
            det = determinant(triad, knots);

        vector<vector<double>> M, G;
        M.resize(3);
        G.resize(3);
        for (int i = 0; i < 3; i++) {
            M[i].resize(3);
            G[i].resize(3);
        }

        double gamma = func_gamma(gamma_v[triad.knot_nums[3]]);

        local_M(M, det, gamma);
        local_G(G, triad, det, lambda, knots);

        local_A.knot_nums.resize(3);
        for (int i = 0; i < 3; i++)
            local_A.knot_nums[i] = triad.knot_nums[i];

        local_A.A.resize(3);
        for (int i = 0; i < 3; i++) {
            local_A.A[i].resize(3);
            for (int j = 0; j < 3; j++)
                local_A.A[i][j] = G[i][j] + delta_t * M[i][j];
        }

        local_A.b.resize(3);
        //локальный вектор b = f * C
        double f1 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[0]],
time[3]),
            f2 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[1]],
time[3]),
            f3 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[2]],
time[3]);

        vector<double> q_temp0(3), q_temp1(3), q_temp2(3);

        for (int i = 0; i < 3; i++) {
            int temp = local_A.knot_nums[i];
            q_temp0[i] = q[0][temp];
            q_temp1[i] = q[1][temp];
            q_temp2[i] = q[2][temp];
        }
```

```
        q_temp0 = A_on_vector(M, q_temp0);
        q_temp1 = A_on_vector(M, q_temp1);
        q_temp2 = A_on_vector(M, q_temp2);

        local_A.b[0] = abs(det) * (2 * f1 + f2 + f3) / 24 - delta_t3 * q_temp0[0]
- delta_t2 * q_temp1[0] - delta_t1 * q_temp2[0];
        local_A.b[1] = abs(det) * (f1 + 2 * f2 + f3) / 24 - delta_t3 * q_temp0[1]
- delta_t2 * q_temp1[1] - delta_t1 * q_temp2[1];
        local_A.b[2] = abs(det) * (f1 + f2 + 2 * f3) / 24 - delta_t3 * q_temp0[2]
- delta_t2 * q_temp1[2] - delta_t1 * q_temp2[2];

        local_A_list[iA] = local_A;
    }
}

void global_A(vector<local>& Local_A, vector<set<int>>& L, SLAE& slae) { //сборка
глоб. матрицы
    for (int k = 0; k < num_knots; k++) {
        slae.di[k] = 0;
        slae.b[k] = 0;
    }

    //алгоритм из методички
    for (vector <local>::iterator iter = Local_A.begin(); iter != Local_A.end();
iter++) {
        local A_iter = *iter;

        //заносим выse диагональные элементы
        for (int k = 0; k < 3; k++) {
            slae.di[A_iter.knot_nums[k]] += A_iter.A[k][k];
            slae.b[A_iter.knot_nums[k]] += A_iter.b[k];
        }
        //начинаем цикл по строкам нижнего
        for (int i = 0; i < 3; i++) {
            //устанавливаем начальное значение нижней границы поиска
            int ibeg = slae.ig[A_iter.knot_nums[i]];

            for (int j = 0; j < i; j++) { // do j=1,i-1
                int iend = slae.ig[A_iter.knot_nums[i] + 1] - 1;

                while (slae.jg[ibeg] != A_iter.knot_nums[j]) {
                    int ind = (ibeg + iend) / 2;
                    if (slae.jg[ind] < A_iter.knot_nums[j])
                        ibeg = ind + 1;
                    else
                        iend = ind;
                }
                slae.ggu[ibeg] += A_iter.A[j][i];
                slae.ggl[ibeg] += A_iter.A[i][j];
                ibeg++;
            }
        }
    }
}

void create_L(vector<set<int>> L, vector<triangle>& triangle_list, SLAE& slae) {
//вектор связностей
    int a[3];
    for (vector <triangle>::iterator iter = triangle_list.begin(); iter != trian-
gle_list.end(); iter++) {
```

```
        triangle triad = *iter;
        a[0] = triad.knot_nums[0];
        a[1] = triad.knot_nums[1];
        a[2] = triad.knot_nums[2];

        vector<int> abc(3);
        abc.insert(abc.begin(), a, a + 3);

        L[abc[2]].insert(abc[1]);
        L[abc[2]].insert(abc[0]);

        L[abc[1]].insert(abc[0]);
    }

    n = 0;
    for (int i = 0; i < num_knots; i++)
        n += L[i].size();

    slae.jg.resize(n);
    slae.ggu.resize(n);
    slae.ggl.resize(n);
    for (int i = 0; i < n; i++) {
        slae.ggu[i] = 0;
        slae.ggl[i] = 0;
    }

    int i = 0;
    for (vector<set<int>>::iterator it = L.begin(); it != L.end(); it++)
        for (set<int>::const_iterator cit = it->begin(); cit != it->end(); cit++,
i++)
            slae.jg[i] = (*cit);

    slae.ig[0] = 0;
    for (int i = 1; i < num_knots + 1; i++)
        slae.ig[i] = slae.ig[i - 1] + L[i - 1].size();
}

void A_mult(SLAE& slae, vector<double> f, vector<double>& res) { //иатрица на век-
тор
    for (int i = 0; i < num_knots; i++)
        res[i] = slae.di[i] * f[i];

    for (int i = 0; i < num_knots; i++) {
        for (int k = slae.ig[i]; k < slae.ig[i + 1]; k++) {
            res[i] += slae.ggl[k] * f[slae.jg[k]];
            res[slae.jg[k]] += slae.ggu[k] * f[i];
        }
    }
}

void Conjugate_Gradient_Method_LOS(SLAE slae, vector<knot> knots, vector<double>&
q) {
    double alpha, betta, residual, scalar_p, sqrt_scalar_b, eps = 1E-15;
    vector<double> z, r, x, p, temp;
    temp.resize(num_knots);
    x.resize(num_knots);
    p.resize(num_knots);
    z.resize(num_knots);
    r.resize(num_knots);
    for (int i = 0; i < num_knots; i++) {
```

```
        x[i] = 0;
        z[i] = 0;
        r[i] = 0;
        p[i] = 0;
        temp[i] = 0;
    }

    //r0 =f - A * x0
    //z0 = r0
    //p0 = A * z0
    A_mult(slae, x, temp);
    r = slae.b + (-1) * temp;
    z = r;
    A_mult(slae, z, temp);
    p = temp;
    sqrt_scalar_b = sqrt(scalar(slae.b, slae.b));
    residual = sqrt(scalar(r, r)) / sqrt_scalar_b;
    for (int k = 0; k < 10000 && residual > eps; k++) {
        scalar_p = scalar(p, p);
        alpha = scalar(p, r) / scalar_p;
        x += alpha * z;
        r += -alpha * p;
        A_mult(slae, r, temp);
        betta = -scalar(p, temp) / scalar_p;
        z = r + betta * z;
        p = temp + betta * p;
        residual = sqrt(scalar(r, r)) / sqrt_scalar_b;
    }

    q = x;

    ///////////////для таблиц погрешностей//////////////
    cout << "q" << endl;
    for (int i = 0; i < num_knots; i++)
        cout << setprecision(15) << x[i] << endl;

    cout << endl;

    for (int i = 0; i < num_knots; i++)
        cout << abs(knots[i].x + knots[i].y  + sin(.075) - x[i]) << endl;

    cout << endl;

    for (int i = 0; i < num_knots; i++)
        cout << knots[i].x + knots[i].y + sin(.05) << endl;
    ////////////////////////////////////////////////
}

int main(void) {
    double betta = 1;

    //вектор узлов
    vector<knot> knots;
    read_knots(knots);

    SLAE slae;

    //вектор элементов(треуголников)
    vector<triangle> triangles;
    create_triangles(triangles);
```

```
    //вектора, где размероность = кол-во подобластей, а эл - номер нужной функции
    vector<int> lambda, f_vector, gamma;

    vector<double> time(4); //т.к. четырехслойная

    create_lambda_f_gamma(lambda, f_vector, gamma);
    create_time_vector(time);

    //вектор связностей
    vector<set<int>> L(num_knots, set<int>()); //вектор сетов размерности = коли-
чество узлов
    create_L(L, triangles, slae);

    vector<vector<double>> q(4);
    for (int i = 0; i < 4; i++)
        q[i].resize(num_knots);

    ifstream q1_file("primary_conditions.txt");
    for (int i = 0; i < num_knots; i++)
        q1_file >> q[0][i];
    for (int i = 0; i < num_knots; i++)
        q1_file >> q[1][i];
    for (int i = 0; i < num_knots; i++)
        q1_file >> q[2][i];

    //вектор локальных матриц
    vector<local> Local_A, bounds_1, bounds_2, bounds_3;

    ifstream input_1("boundary_conditions_1.txt"),
        input_2("boundary_conditions_2.txt"),
        input_3("boundary_conditions_3.txt");

    //read_bounds(bounds_2, input_2, num_bounds_2, 2);
    //read_bounds(bounds_3, input_3, num_bounds_3, 3);
    read_bounds(bounds_1, input_1, num_bounds_1, 1);

    int i = 3; //соответсвенно двух трех и четырехслойные схемы, интересует только
четырехслойная
    switch (i) {
    case 1:
        local_A_2(Local_A, triangles, lambda, knots, gamma, f_vector, q[0], time);
        break;
    case 2:
        local_A_3(Local_A, triangles, lambda, knots, gamma, f_vector, q, time);
        break;
    case 3:
        local_A_4(Local_A, triangles, lambda, knots, gamma, f_vector, q, time);
        break;
    }

    //build_bound(bounds_2, 2, knots, betta, time[i]);
    //build_bound(bounds_3, 3, knots, betta, time[i]);

    //сборка глобальной матрицы
    global_A(Local_A, L, slae);

    //use_bounds(bounds_2, L, slae, 2, knots);
    //use_bounds(bounds_3, L, slae, 3, knots);
```

```
    use_first_bounds(bounds_1, L, slae, knots, time[i]);

    Conjugate_Gradient_Method_LOS(slae, knots, q[i]); //решаем

    cout << "\nHello World!\n";
}
```