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 «Численные методы»
Место для ввода текста.

Группа	ПМ-81
Студент	ЮРГАНОВ ЕГОР
Преподаватель	ПАТРУШЕВ ИЛЬЯ ИГОРЕВИЧ
Дата	04.06.2021

Новосибирск

1. Формулировка задачи

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции линейные на треугольниках.

2. Теоретическая часть

2.1. Краевые задачи для уравнения эллиптического типа (стр. 24)

Эллиптическая краевая задача для функции u определяется дифференциальным уравнением:

$$-div(\lambda\ grad(u))+\gamma u=f \ (1)$$

заданным в некоторой области Ω с границей $S=S1 \cup S2 \cup S3$, и краевыми условиями:

$$\mathrm{u}|_{\mathrm{s}_1}= u_g \ (2)$$

$$\lambda \frac{\partial \mathrm{u}}{\partial \mathrm{n}} \mathrm{u}\Big|_{\mathrm{s}_2} = \theta \ (3)$$

$$\lambda \frac{\partial \mathrm{u}}{\partial \mathrm{n}} \mathrm{u}\Big|_{\mathrm{s}_3} + \beta(\mathrm{u}|_{\mathrm{s}_3} - \mathrm{u}_\beta) = 0 \ (4)$$

в которых $\mathrm{u}|_{\mathrm{s_i}}$ – значение искомой функции u на границе S_i, а $\frac{\partial \mathrm{u}}{\partial \mathrm{n}} \mathrm{u}\Big|_{s_i}$ – значение на S_i производной функции u по направлению внешней нормали к поверхности S_i.

2.2. Вариационная постановка в форме уравнения Галеркина

В основе МКЭ лежат вариационные постановки, в которых решение краевых задач заменяется минимизацией некоторого функционала. Областью определения этого функционала является Гильбертово пространство функций, содержащее в качестве одного из своих элементов решение u данной краевой задачи.

В операторной форме исходное уравнение можно переписать в форме $Lu=f$, где L - оператор, действующий в Гильбертовом пространстве H. Нам нужно найти приближение к элементу $u \in H$, соответствующее заданному элементу $f \in H$.

В общем виде построение вариационной формулировки в форме уравнения Галеркина выглядит следующим образом. Если нам нужно решать краевую задачу для дифференциального уравнения

$Lu=f$ (5) то следует левую и правую часть этого уравнения домножить на функцию v из пространства пробных функций Ф и проинтегрировать по Ω. Фактически это соответствует скалярному умножению Lu и f на v в пространстве $L2(\Omega)$:

$(Lu-f,v)=0, \forall v \in \Phi, \Phi=\{v \in L2(\Omega): v|_{s_1}\}$ (6)

Для уравнения (1) постановка примет вид:

$\int -div(\lambda\ grad(u))v d\Omega + \int(\gamma u - f)v d\Omega = 0, \forall v \in \Phi$ (7)

Преобразуем слагаемое $\int -div(\lambda\ grad(u))v d\Omega$ с использованием формулы Грина:

$\int \lambda\ grad(u) grad(v) d\Omega - \int \lambda \frac{\partial u}{\partial n} v dS + \int(\gamma u - f) v d\Omega = 0, \forall v \in \Phi$ (8)

Т.к. $S=S_1 \cup S_2 \cup S_3$:

$\int \lambda \frac{\partial u}{\partial n} v dS = \int \lambda \frac{\partial u}{\partial n} v dS_1 + \int \lambda \frac{\partial u}{\partial n} v dS_2 + \int \lambda \frac{\partial u}{\partial n} v dS_3$

и поскольку $v|_{S_1}=0$, то $\int \lambda \partial \frac{\partial u}{\partial n} v dS_1 = 0$, значит, интегральное соотношение примет вид:

$\int \lambda grad(u) grad(v) d\Omega - \int \lambda \frac{\partial u}{\partial n} v dS_2 - \int \lambda \frac{\partial u}{\partial n} v dS_3 + \int(\gamma u - f) v d\Omega\ \Omega = 0, \forall v \in \Phi$ (9)

Интегралы по границам S_2 и S_3 можно преобразовать, воспользовавшись краевыми условиями (3) и (4)

Обратим внимание на то, что в уравнение входят производные пробных функций v. Поэтому в качестве Ф мы можем выбрать H_0^1 (H_0^1 – пространство функций, имеющих суммируемые с квадратом производные и равные нулю на границе S_1).

Таким образом, получаем вариационное уравнение вида:

$\int \lambda grad(u) grad(v_0) d\Omega + \int \beta u v_0 dS_3 + \int \gamma u v_0 d\Omega = \int f v_0 d\Omega + \int \theta v_0 dS_2 + \int \beta u_\beta v_0 dS_3$ $, \forall v \in H_0^1$ (10)

В пространстве H_0^1 выделим конечномерное пространство V_h, которое определяется как линейное пространство, натянутое на базисные функции ψ_i.

Заменим функцию u аппроксимирующей ее функцией u^h, а функцию v_0 - функцией v_0^h.

Т.к. v_0^h представима в виде линейной комбинации:

$v_0^h = \Sigma_i q_i^v \psi_i, i \in N_0$ (11)

Также u^h представима в виде:

$u^h = \Sigma_i q_i^v \psi_i, i \in N_0$ (12)

Причем $n-N_0$ компонент вектора весов $q=(q1,\ldots,qn)^T$ должны быть фиксированы и могут быть определены из условия

$u^h|_{S_1}=u_g$ (13)

Подставляем, получаем СЛАУ для компонент вектора весов:

$$\sum_{i=1}^{n}\left(\int_{\Omega} \lambda \mathrm{grad}(\psi_i)\mathrm{grad}(\psi_j)\, d\Omega + \int_{s3} \beta\psi_i\psi_j dS + \int_{\Omega} \gamma\psi_i\psi_j\, d\Omega\right) q_i$$

$$= \int_{\Omega} f\psi_i\, d\Omega + \int_{s3} \beta u_b \psi_i dS + \int_{s2} \theta\psi_i dS$$

Таким образом СЛАУ может записана в виде $Aq=b$, где:

$$A_{ij} = \begin{cases} \int\limits_{\Omega} \lambda \mathrm{grad}(\psi_i)\mathrm{grad}(\psi_j)\, d\Omega + \int\limits_{s3} \beta\psi_i\psi_j dS + \int\limits_{\Omega} \gamma\psi_i\psi_j\, d\Omega, \mathrm{i} \in N_0 \\ \delta_{ij}, \mathrm{i} \notin N_0 \end{cases}$$

$$b_i = \begin{cases} \int\limits_{\Omega} f\psi_i\, d\Omega + \int\limits_{s3} \beta u_b \psi_i dS + \int\limits_{s2} \theta\psi_i dS, \mathrm{i} \in N_0 \\ u_g(x_i), \mathrm{i} \notin N_0 \end{cases}$$

в которых δ_{ij}– критерий Кронекера (δ_{ii}=1 и δ_{ij}=0 при $i\neq j$).

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>& vector_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>& vector_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) {
    double temp;
    ifstream lambda_file("lambda.txt");
    lambda_file >> num_lambda;
    lambda.resize(num_lambda);
    for (int i = 0; i < num_lambda; i++) {
        lambda_file >> temp;
        lambda[i] = temp;
    }
    f.resize(num_lambda);
    for (int i = 0; i < num_lambda; i++) {
        lambda_file >> temp;
        f[i] = temp;
    }
    gamma.resize(num_lambda);
    for (int i = 0; i < num_lambda; i++) {
        lambda_file >> temp;
        gamma[i] = temp;
    }
}

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_f(int number_f, knot& knots) {
    switch (number_f) {
    case 1: {
        return knots.x * knots.x;
        //return knots[i].y * knots[i].y;
        break;
    }
    case 2: {
        return (knots.x * knots.y);
        //return (knots[i].x + knots[i].y);
        break;
    }
    case 3: {
        return (knots.x * knots.y) * (knots.x * knots.y) + 10.;
        break;
    }
    case 4: {
        return pow(knots.x, 3) - 6 * knots.x;
        break;
    }
    case 5: {
        return 3 * cos(knots.x + knots.y);
        break;
    }
    case 6: {
        return 0.;
        break;
    }
    case 7: {
        return 1.;
        break;
    }
    case 8: {
        return knots.x * knots.x - 2.;
        break;
    }
    case 9: {
        return -2.;
        break;
    }
    case 10: {
        return knots.x;
        break;
    }
    case 11: {
        return pow(knots.x, 4) - 12 * knots.x * knots.x;
        break;
    }
    }
}

double func_lambda(int number_f) {
    switch (number_f) {
```

```
    case 1: {
        return 0.;
        break;
    }
    case 2: {
        return 1.;
        break;
    }
    case 3: {
        return 5.;
        break;
    }
    }
}

double func_gamma(int number_f) {
    switch (number_f) {
    case 1: {
        return 0.;
        break;
    }
    case 2: {
        return 5.;
        break;
    }
    case 3: {
        return 1.;
        break;
    }
    }
}
```

bounds.h

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

void read_bounds_2_3(vector<local>& vector_bounds, ifstream& input, int& num_bounds) {
    input >> num_bounds;
    vector_bounds.resize(num_bounds);
    int num;
    for (int i = 0; i < num_bounds; i++) {
        local local_bound;
        local_bound.knot_nums.resize(4);
        for (int j = 0; j < 2; j++) {//первые два - номера узлов ребра, 3,4 - значение
            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];
        input >> local_bound.knot_nums[3];
        vector_bounds[i] = local_bound;
    }
}

void read_bounds_1(vector<local>& vector_bounds, ifstream& input, int& num_bounds) {
    input >> num_bounds;
    vector_bounds.resize(num_bounds);
    int num;
    for (int i = 0; i < num_bounds; i++) {
        local local_bound;
        local_bound.knot_nums.resize(3);
        for (int j = 0; j < 2; j++) {//первые два - номера узлов ребра
            input >> num;
            local_bound.knot_nums[j] = num - 1;
        }
```

```
        input >> num;
        local_bound.knot_nums[2] = num; //номер уравнения
        local_bound.b.resize(2);
        vector_bounds[i] = local_bound;
    }
}

void build_bound(vector<local>& vector_bound, int flag, vector<knot> knots, double betta) {
    int iA = 0;
    for (vector <local>::iterator iter = vector_bound.begin(); iter != vector_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 * bounds.knot_nums[2] + bounds.knot_nums[3]) / 6;
            bounds.b[1] = betta * h * (bounds.knot_nums[2] + 2 * bounds.knot_nums[3]) / 6;

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

double ub(knot knots, int num) {
    switch (num) {
    case 15: {
        return knots.x;
        break;
    }
    case 1: {
        //return (knots[i].x * knots[i].y);
        return (knots.x * knots.x);
        break;
    }
    case 3: {
        return cos(knots.x + knots.y);
        break;
    }
    case 14: {
        return pow(knots.x, 3);
```

```
            break;
        }
        case 123: {
            return pow(knots.x, 4);
            break;
        }
        case 5: {
            return 1;
            break;
        }
        default: {
            return 1;
            break;
        }
        }
}

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 все 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]] += ub(knots[bound_iter.knot_nums[k]], 1); //
последнее - номер фйункции
            }
            //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, vec-
tor<knot>& knots) {
```

```
    for (vector <local>::iterator iter = vector_bound.begin(); iter != vector_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]);
//ub(knot*& knots, int i, int num)
        }
        cout << endl;

        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < n; j++) {
                if (slae.jg[j] == bound.knot_nums[i])
                    slae.ggu[j] = 0;
            }
            cout << endl;
            //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;
        }
    }
}

main.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, vec-
tor<knot>& knots) {
    vector<vector<double>> a;   //a = D^-1
    a.resize(3);
    for (int i = 0; i < 3; i++)
        a[i].resize(2);

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

    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;
        }
}

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.;
}
```

```
void local_A(vector<local>& local_A_list, vector<triangle>& triangle_list, vector<int>
lambda_vector, vector<knot> knots, vector<int> gamma_v, vector<int> f) {
    local_A_list.resize(triangle_num);
    int iA = 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]]);
        double 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] + 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]]);
        double f2 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[1]]);
        double f3 = func_f(f[triad.knot_nums[3]], knots[triad.knot_nums[2]]);

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

        local_A_list[iA] = local_A;
    }
}

void global_A(vector<local>& Local_A, vector<set<int>>& L, SLAE& slae) {
    for (vector <local>::iterator iter = Local_A.begin(); iter != Local_A.end(); iter++) {
        local A_iter = *iter;

        //заносим высе 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]];
```

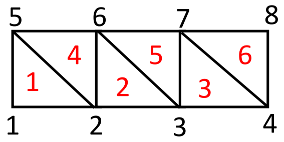
```
            for (int j = 0; j < i; j++) { // do j=1,i-1
                int iend = slae.ig[A_iter.knot_nums[i] + 1] - 1;

                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) {
    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) {
    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) {
    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 < 100000 && 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;
        for (int i = 0; i < num_knots; i++)
            cout << setprecision(15) << x[i] << " ";
        cout << endl;
    }

    vector<double> u(num_knots), ururur(num_knots);
    ofstream file("result.txt");
    for (int i = 0; i < num_knots; i++) {
        cout << x[i] << " ";
        file << setprecision(15) << x[i] << endl;
    }
    file << endl;

    for (int i = 0; i < num_knots; i++) {
        u[i] = pow(knots[i].x, 4);//= knots[i].x;
        file << setprecision(15) << u[i] << endl;
    }
    file << endl;

    for (int i = 0; i < num_knots; i++) {
        ururur[i] = x[i] - u[i];
        file << setprecision(15) << abs(x[i] - u[i]) << endl;
    }
}
```

```
int main(void) {
    double betta = 1;

    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_vector, f_vector, gamma;

    create_lambda_f_gamma(lambda_vector, f_vector, gamma);

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

    //вектор локальных матриц
    vector<local> Local_A;
    local_A(Local_A, triangles, lambda_vector, knots, gamma, f_vector);

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

    //читаем краевые и вносим в глобальную
    vector<local> bounds_1, bounds_2, bounds_3;
    ifstream input_1("boundary_conditions_1.txt");
    ifstream input_2("boundary_conditions_2.txt");
    ifstream input_3("boundary_conditions_3.txt");

    read_bounds_2_3(bounds_2, input_2, num_bounds_2);
    read_bounds_2_3(bounds_3, input_3, num_bounds_3);
    read_bounds_1(bounds_1, input_1, num_bounds_1);

    build_bound(bounds_2, 2, knots, betta);
    build_bound(bounds_3, 3, knots, betta);

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

    use_first_bounds(bounds_1, L, slae, knots);

    /*cout << "ggu" << endl;
    for (int i = 0; i < n; i++)
        cout << slae.ggu[i] << " ";
    cout << endl;
    cout << "ggl" << endl;
    for (int i = 0; i < n; i++)
        cout << slae.ggl[i] << " ";
    cout << endl;
    cout << "di" << endl;
    for (int i = 0; i < num_knots; i++)
        cout << slae.di[i] << " ";
    cout << endl;
    cout << "b" << endl;
    for (int i = 0; i < num_knots; i++)
        cout << slae.b[i] << " ";
    cout << endl;
    cout << endl;*/
    Conjugate_Gradient_Method_LOS(slae, knots);
}
```

4. Тесты

4.1.1. Проверка матрицы массы M



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

Тест 1

λ	γ	f
0	1	1

Искомое решение: u = 1

q	u	|q-u|
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000

Тест 2

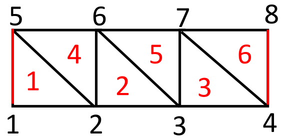
λ	γ	f
0	1	x + y

Искомое решение: u = x + y

q	u	|q-u|
6.18e-13	0.00000000000000	6.18e-13
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
5.00000000000000	5.00000000000000	0.00000000000000
8.00000000000000	8.00000000000000	0.00000000000000

4.1.2. Проверка матрицы жесткости G

красный линии – первые краевые



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

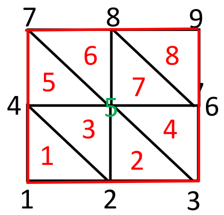
Тест 1

λ	γ	f
1	0	0

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000

Тест 2



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

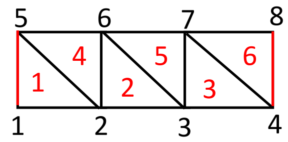
λ	γ	f
1	0	0

Искомое решение: u = x * y

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000
9.00000000000000	9.00000000000000	0.00000000000000

4.1.3. Тест линейной функции

красный линии – первые краевые



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

Тест 1

λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001

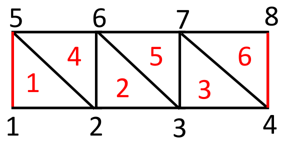
Тест 2

λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
2.50000000000000	2.50000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
4.50000000000000	4.50000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
2.50000000000001	2.50000000000000	0.00000000000001
3.00000000000000	3.00000000000000	0.00000000000000
4.50000000000000	4.50000000000000	0.00000000000000
6.00000000000000	6.00000000000000	0.00000000000000

Тест 3(равномерная сетка)



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

λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
5.99999999999999	6.00000000000000	0.00000000000001

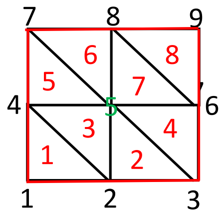
Тест 4(равномерная сетка, дробление по x в два раза)

λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
4.99999999999999	5.00000000000000	0.00000000000001
6.00000000000000	6.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
4.99999999999999	5.00000000000000	0.00000000000001
6.00000000000000	6.00000000000000	0.00000000000000

Тест 5



№ 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

λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.99999999999999	4.00000000000000	0.00000000000001
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.99999999999999	4.00000000000000	0.00000000000001
0.00000000000000	0.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.99999999999999	4.00000000000000	0.00000000000001

Тест 6(дробление шага по x в два раза)

λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000

Тест 7(дробление шага по x еще в два раза)

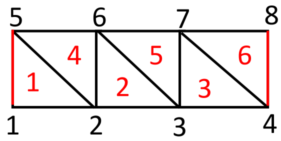
λ	γ	f
1	1	x

Искомое решение: u = x

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
0.50000000000000	0.50000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.50000000000000	1.50000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
2.50000000000000	2.50000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
3.50000000000000	3.50000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
0.50000000000000	0.50000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.50000000000000	1.50000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
2.50000000000000	2.50000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
3.50000000000000	3.50000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
0.50000000000000	0.50000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
1.50000000000000	1.50000000000000	0.00000000000000
2.00000000000000	2.00000000000000	0.00000000000000
2.50000000000000	2.50000000000000	0.00000000000000
3.00000000000000	3.00000000000000	0.00000000000000
3.50000000000000	3.50000000000000	0.00000000000000
4.00000000000000	4.00000000000000	0.00000000000000

4.1.4. Тест квадратичной функции

красный линии – первые краевые



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

Тест 1

λ	γ	f
1	1	$x^2 - 2$

Искомое решение: $u = x^2$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
3.94819546367781	4.00000000000000	0.05180453632219
9.16205400126682	9.00000000000000	0.16205400126682
35.9999999912157	36.0000000000000	8.7843e-09
0.00000000000000	0.00000000000000	0
4.05010635596975	4.00000000000000	0.05010635596975
8.85351433294374	9.00000000000000	0.14648566705626
35.9999999912157	36.0000000000000	8.7843e-09

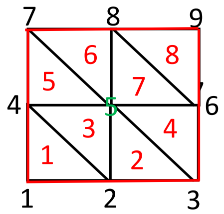
Тест 2(дробим сетку по x в два раза)

λ	γ	f
1	1	$x^2 - 2$

Искомое решение: $u = x^2$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
0.99260511443446	1.00000000000000	0.00739488556554
3.97248764705873	4.00000000000000	0.02751235394121
6.28256523415913	6.25000000000000	0.03256523415913
9.10627228527733	9.00000000000000	0.10627228527733
20.2766081859531	20.2500000000000	0.0266081859531
36.0000000686056	36.0000000000000	6.86056e-08
0.00000000000000	0.00000000000000	0.00000000000000
1.00910529878464	1.00000000000000	0.00910529878464
4.02748602982797	4.00000000000000	0.02748602982797
6.21757754922105	6.25000000000000	0.03242245077895
8.89976217801948	9.00000000000000	0.10023782198052
20.2355324451943	20.2500000000000	0.0144675548057
36.0000000686056	36.0000000000000	6.86056e-08

Тест 3



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

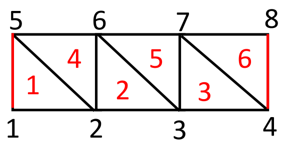
λ	γ	f
1	1	$x^2 - 2$

Искомое решение: $u = x^2$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
3.99999999534485	4.00000000000000	4.65515e-09
8.99999998952592	9.00000000000000	1.047408e-08
0.00000000000000	0.00000000000000	0.00000000000000
4.02985072911814	4.00000000000000	0.02985072911814
8.99999998952592	9.00000000000000	1.047408e-08
0.00000000000000	0.00000000000000	0.00000000000000
3.99999999534485	4.00000000000000	4.65515e-09
8.99999998952592	9.00000000000000	1.047408e-08

4.1.5. Кубическая функция

красный линии – первые краевые



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

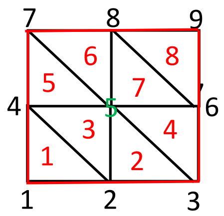
Тест 1

λ	γ	f
1	1	$x^3 - 6x$

Искомое решение: $u = x^3$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
8.94195152315054	8.00000000000000	0.94195152315054
30.2107512394543	27.0000000000000	3.2107512394543
216.000000000000	216.000000000000	8.7843e-09
0.00000000000000	0.00000000000000	0.00000000000000
6.92232378709544	8.00000000000000	1.17767621290456
24.1864659898496	27.0000000000000	2.8136340101504
216.000000000000	216.000000000000	0.00000000000000

Тест 2



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

λ	γ	f
1	1	$x^3 - 6x$

Искомое решение: $u = x^3$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
8.00000000000000	8.00000000000000	0.00000000000000
64.0000000000000	64.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
8.00000000000001	8.00000000000000	0.00000000000001
64.0000000000000	64.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
8.00000000000000	8.00000000000000	0.00000000000000
64.0000000000000	64.0000000000000	0.00000000000000

Тест 3(дробление шага по x в два раза)

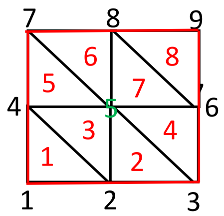
λ	γ	f
1	1	$x^3 - 6x$

Искомое решение: $u = x^3$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
8.00000000000001	8.00000000000000	0.00000000000001
27.0000000000000	27.0000000000000	0.00000000000000
64.0000000000000	64.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
0.99999999999999	1.00000000000000	0.00000000000001
8.00000000000000	8.00000000000000	0.00000000000000
27.0000000000000	27.0000000000000	0.00000000000000
64.0000000000000	64.0000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
8.00000000000001	8.00000000000000	0.00000000000001
27.0000000000000	27.0000000000000	0.00000000000000
64.0000000000000	64.0000000000000	0.00000000000000

4.1.6. X^4

Тест 1



№ 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

λ	γ	f
1	1	$x^4 - 12x^2$

Искомое решение: $u = x^4$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
16.0000000000000	16.0000000000000	0.00000000000000
256.000000000000	256.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
10.6666666666667	16.0000000000000	5.33333333333334
256.000000000000	256.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
16.0000000000000	16.0000000000000	0.00000000000000
256.000000000000	256.000000000000	0.00000000000000

Тест 2(дробление шага по x в два раза)

λ	γ	f
1	1	$x^4 - 12x^2$

Искомое решение: $u = x^4$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
16.0000000000000	16.0000000000000	0.00000000000000
80.9999999999999	81.0000000000000	0.00000000000000
256.000000000000	256.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
-0.07020872865276	1.00000000000000	1.07020872865276
14.6793168880455	16.0000000000000	1.32068311195449
79.9297912713471	81.0000000000000	1.07020872865286
256.000000000000	256.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
1.00000000000000	1.00000000000000	0.00000000000000
16.0000000000000	16.0000000000000	0.00000000000000
80.9999999999999	81.0000000000000	0.00000000000000
256.000000000000	256.000000000000	0.00000000000000

Тест 3(дробление шага по x еще в два раза)

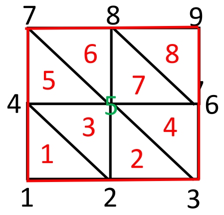
λ	γ	f
1	1	$x^4 - 12x^2$

q	u	|q-u|
0.00000000000000	0.00000000000000	0.00000000000000
0.06249999999999	0.06250000000000	0.00000000000001
0.99999999999999	1.00000000000000	0.00000000000001
5.06249999999999	5.06250000000000	0.00000000000001
16.0000000000000	16.0000000000000	0.00000000000000
39.0625000000000	39.0625000000000	0.00000000000000
80.9999999999998	81.0000000000000	0.00000000000002
150.062500000000	150.062500000000	0.00000000000000
256.000000000000	256.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
-0.10972345251115	0.06250000000000	0.17222345251115
0.73191206657013	1.00000000000000	0.26808793342986
4.74635117994889	5.06250000000000	0.31614882005111
15.6692778788444	16.0000000000000	0.3307221211556
38.7463511799489	39.0625000000000	0.3161488200511
80.7319120665700	81.0000000000000	0.2680879334300
149.890276547489	150.062500000000	0.1722234525111
256.000000000000	256.000000000000	0.00000000000000
0.00000000000000	0.00000000000000	0.00000000000000
0.06249999999999	0.06250000000000	0.00000000000001
0.99999999999999	1.00000000000000	0.00000000000001
5.06249999999999	5.06250000000000	0.00000000000001
16.0000000000000	16.0000000000000	0.00000000000000
39.0625000000000	39.0625000000000	0.00000000000000
80.9999999999998	81.0000000000000	0.00000000000002
150.062500000000	150.062500000000	0.00000000000000
256.000000000000	256.000000000000	0.00000000000000

$$\frac{5.33333333333334}{1.32068311195449} \approx \frac{1.32068311195449}{0.3307221211556} \approx 2^2$$

4.1.7. Неполиномиальная функция

Тест 1



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

λ	γ	f
1	1	3cos(x+y)

Искомое решение: u = cos(x+y)

q	u	|q-u|
1.00000000000000	1.00000000000000	0.00000000000000
-0.416146836547142	-0.416146836547142	0.00000000000000
-0.653643620863612	-0.653643620863612	0.00000000000000
-0.416146836547142	-0.416146836547142	0.00000000000000
-0.496662508775956	-0.653643620863612	0.15698111208766
0.960170286650366	0.960170286650366	0.00000000000000
-0.653643620863612	-0.653643620863612	0.00000000000000
0.960170286650366	0.960170286650366	0.00000000000000
-0.145500033808613	-0.145500033808614	0.00000000000000

Тест 2(дробление шага по x в два раза)

λ	γ	f
1	1	3cos(x+y)

Искомое решение: u = cos(x+y)

q	u	|q-u|
1.00000000000000	1.00000000000000	0.00000000000000
0.54030230586814	0.54030230586814	0.00000000000000
-0.416146836547142	-0.416146836547142	0.00000000000000
-0.989992496600445	-0.989992496600445	0.00000000000000
-0.653643620863612	-0.653643620863612	0.00000000000000
-0.416146836547142	-0.416146836547142	0.00000000000000

-0.811811857022054	-0.989992496600445	0.17818063957839
-0.508965171051856	-0.653643620863612	0.14467844981176
0.289482159099159	0.283662185463226	0.00581997363593
0.960170286650366	0.960170286650366	0.00000000000000
-0.653643620863612	-0.653643620863612	0.00000000000000
0.283662185463226	0.283662185463226	0.00000000000000
0.960170286650366	0.960170286650366	0.00000000000000
0.753902254343305	0.753902254343305	0.00000000000000
-0.145500033808614	-0.145500033808614	0.00000000000000

Тест 3(дробление шага по x еще в два раза)

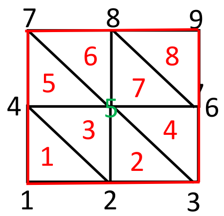
λ	γ	f
1	1	3cos(x+y)

Искомое решение: u = cos(x+y)

q	u	|q-u|
0.999999999999999	1.00000000000000	0.00000000000001
0.877582561890371	0.877582561890373	0.00000000000002
0.540302305868139	0.54030230586814	0.00000000000001
0.0707372016677028	0.0707372016677029	0.00000000000001
-0.416146836547141	-0.416146836547142	0.00000000000001
-0.801143615546933	-0.801143615546934	0.00000000000001
-0.989992496600445	-0.989992496600445	0.00000000000000
-0.936456687290795	-0.936456687290796	0.00000000000001
-0.653643620863611	-0.653643620863612	0.00000000000001
-0.416146836547141	-0.416146836547142	0.00000000000001
-0.663639453281143	-0.801143615546934	0.13750416226579
-0.778349100843985	-0.989992496600445	0.21164339575646
-0.716546553207364	-0.936456687290796	0.21991013408343
-0.481905465121884	-0.653643620863612	0.17173815574173
-0.121189982227923	-0.21079579943078	0.08960581720286
0.290377273299209	0.283662185463226	0.00671508783598
0.671432398989759	0.70866977429126	0.03723737530150
0.960170286650365	0.960170286650366	0.00000000000001
-0.653643620863611	-0.653643620863612	0.00000000000001
-0.210795799430779	-0.21079579943078	0.00000000000001
0.283662185463226	0.283662185463226	0.00000000000000
0.70866977429126	0.70866977429126	0.00000000000000
0.960170286650365	0.960170286650366	0.00000000000001
0.976587625728022	0.976587625728023	0.00000000000001
0.753902254343304	0.753902254343305	0.00000000000001
0.346635317835025	0.346635317835026	0.00000000000001
-0.145500033808613	-0.145500033808614	0.00000000000001

4.1.8. Тест в произвольных точка на полиномах разных степеней

Тест 1 (треугольник 7, с точкой без первого краевого условия)



№ 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

λ	γ	Искомая точка
1	1	(2.5, 2.5)

	q	u	|q-u|
x	2.50000000000000	2.50000000000000	0.00000000000000
x^2	7.00000000000000	6.25000000000000	0.75000000000000
x^3	22.0000000000000	15.6250000000000	6.37500000000000
x^4	73.3333333333333	39.0625000000000	34.2708333333333

Тест 2 (треугольник 1, все точкой с первым краевым условием)

λ	γ	Искомая точка
1	1	(0.5, 0.5)

	q	u	|q-u|
x	0.50000000000000	0.50000000000000	0.00000000000000
x^2	1.00000000000000	0.25000000000000	0.75000000000000
x^3	2.00000000000000	0.12500000000000	1.87500000000000
x^4	4.00000000000000	0.06250000000000	3.93750000000000

Вывод

При увеличении степени полинома искомой функции увеличивается погрешность. Это связано с тем, что базисные функции линейные, поэтому решение дает точный результат в произвольной точке лишь на полиноме первой степени. Однако, на заданных точках точное решение достигается вплоть до полинома четвертой степени. Исходя из исследований, можно утверждать, что порядок аппроксимации равен 3. Также, по тестам было определено, что порядок сходимости равен 2.