Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №2

по дисциплине

«Уравнения математической физики»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.А.

Преподаватели Патрушев И. В.

Вариант 12

Новосибирск 2019

1. Цель работы

Разработать программу решения нелинейной одномерной краевой задачи методом конечных элементов. Сравнить метод простой итерации и метод Ньютона для решения данной задачи.

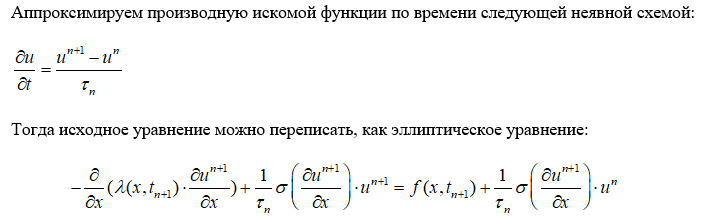
1. Задание

Дано параболическое начально-краевое уравнение относительной функции

С начальным условием

И некоторыми краевыми условиями, при x = a и при x = b

Функция u аппроксимируется квадратичными базисными функциями на каждом конечном элементе. Решение ищется при .



Выполним постановку Галеркина: для этого домножим скалярно правую и левую часть на функцию , где - множество функций, удовлетворяющих однородным первым краевым условиям.

Преобразуя слагаемое с использованием формулы Грина получаем

Т.к. у нас нет 2 и 3 краевых условий, a мы заносим как первые краевые, остается:

Где , базисные функции – квадратичные

1. Вычисление элементов матрица для метода простой итерации

Кратко метод простой итерации записывается следующим образом .

Подставив U в правую часть, и представив , как базисные функции, получаем:

Выносим q:

В матричном виде: матрица жесткости G

матрица массы М

раскладываем на базисные функции получаем,

Элементы матрицы А вычисляются

Учитывая, что базисные функции квадратичные и будут содержать 3 узла, тогда нам понадобится разбивать нашу матрицу на локальные части. Узлы на краях базиса будут находится в предыдущем и следующем базисах для из связывания и значения в глобальной матрице будет суммироваться. Поэтому получившаяся матрица будет пятидиаганальной.

Вычисление вектора правой части:

*Локальный:*

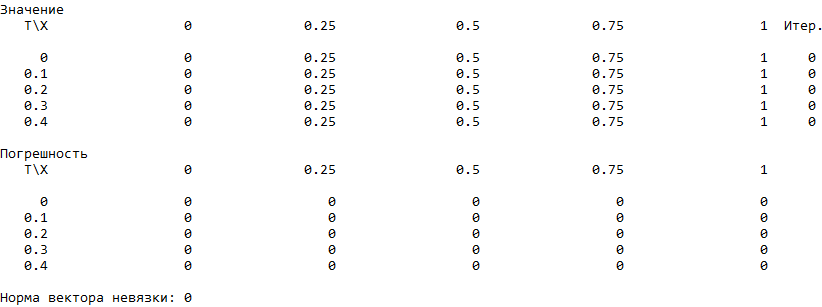
Процесс заканчивается когда невязка системы не станет меньше определенного эпсилона

1. Тесты c const σ = 1, сетка равномерная
2. *Полином 1 степени*

Искомая функция:

Уравнение:

Краевые условия: 0

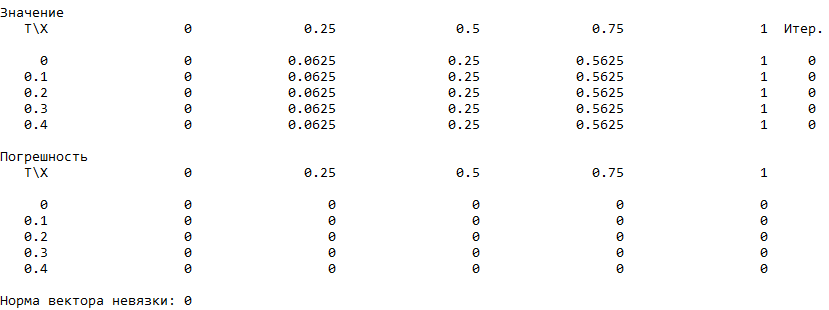


1. *Полином 2 степени*

Искомая функция:

Уравнение:

Краевые условия: 0

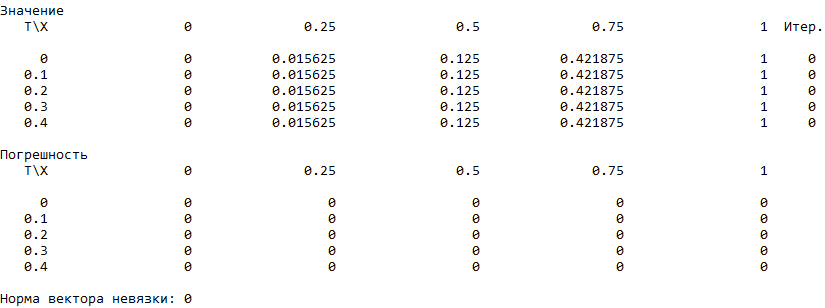


1. *Полином 3 степени*

Искомая функция:

Уравнение:

Краевые условия: 0



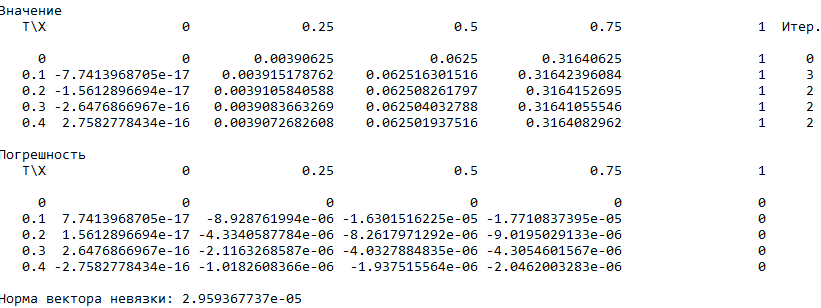
1. *Полином 4 степени*

Искомая функция:

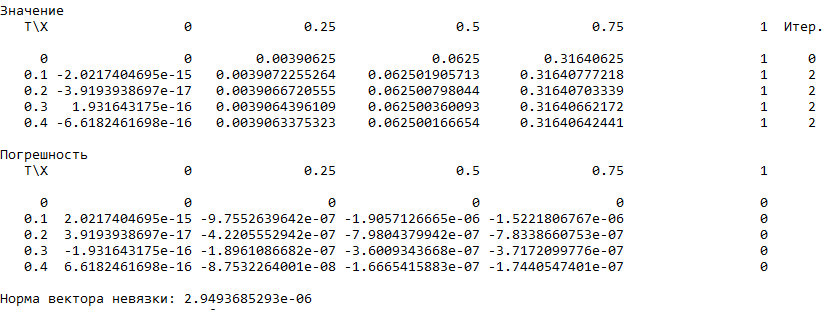
Уравнение:

Краевые условия: 0

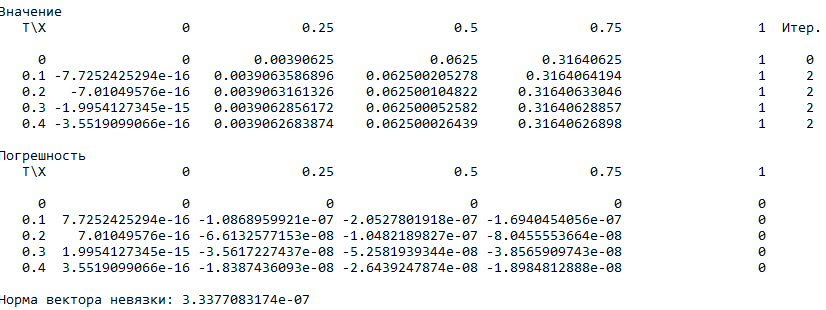
1. На сетке hx



1. На сетке hx /2



1. На сетке hx /4



Порядок сходимости: 3.3268109735545863

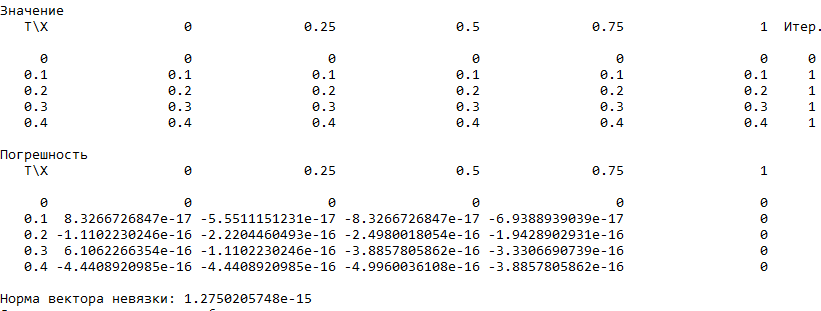
3.143476313112232

1. *Полином 1 степени по времени*

Искомая функция:

Уравнение: ,

Краевые условия:  *t*



1. *Полином 2 степени по времени*

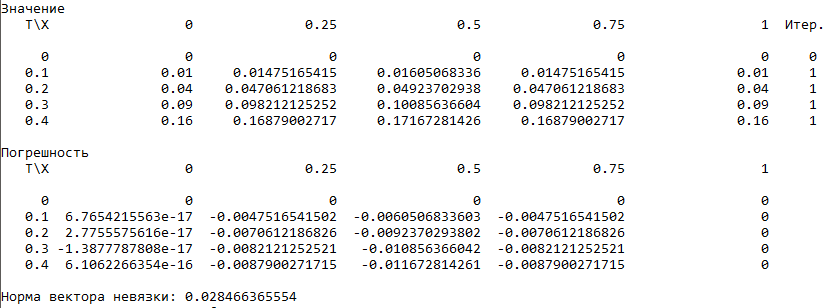
Искомая функция:

Уравнение:

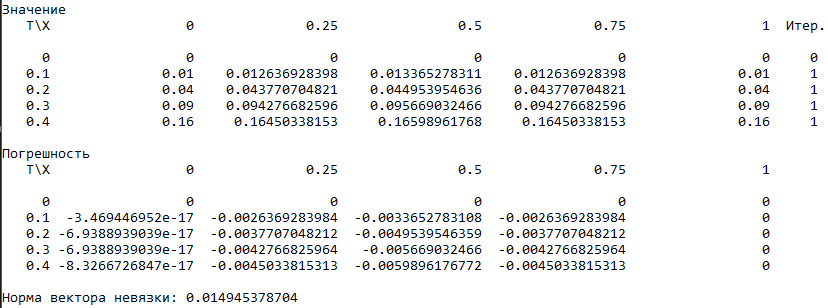
Краевые условия: t\*2

\*2

1. *Сетка с шагом ht*



1. *Сетка с шагом ht/2*



Порядок аппроксимации

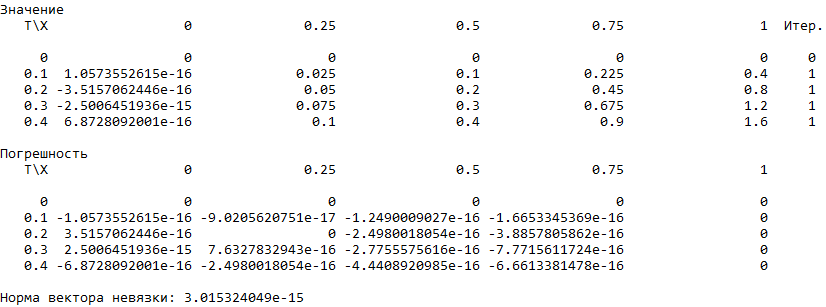
0.9295588543845414

1. *Полином*

Искомая функция:

Уравнение:

Краевые условия: 0

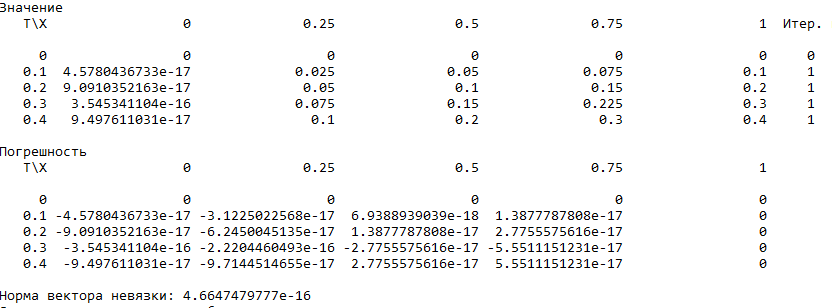


1. Тесты, сетка равномерная, σ ≠ const

Искомая функция:

Уравнение:

Краевые условия: 0

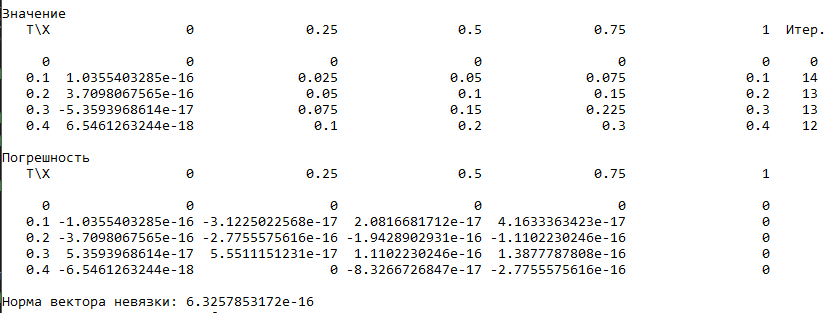




Искомая функция:

Уравнение:

Краевые условия: 0

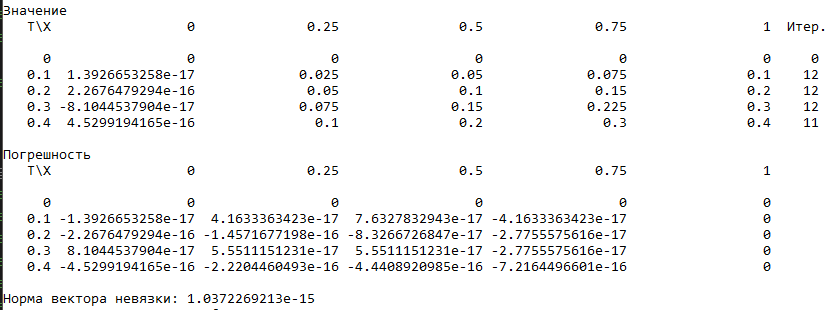




Искомая функция:

Уравнение:

Краевые условия: 0

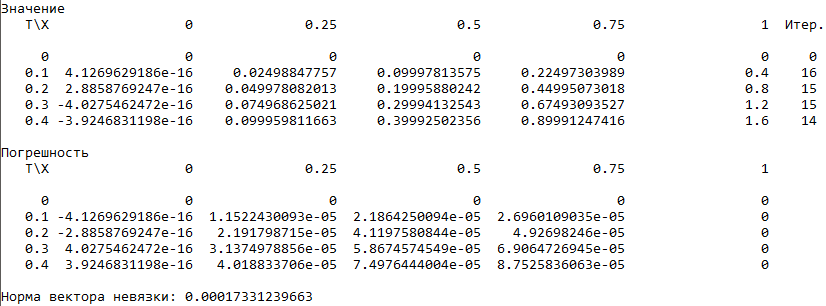


Искомая функция:

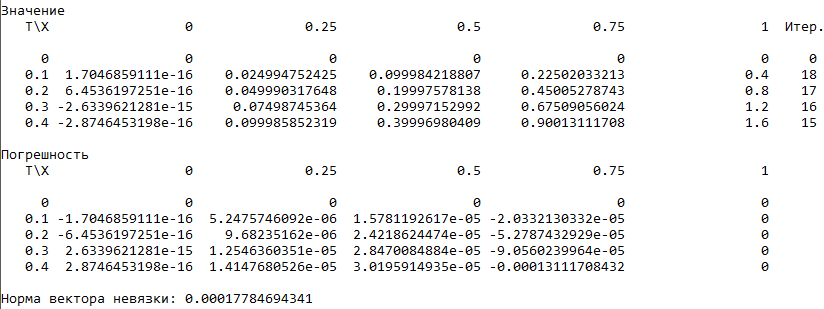
Уравнение:

Краевые условия: 0

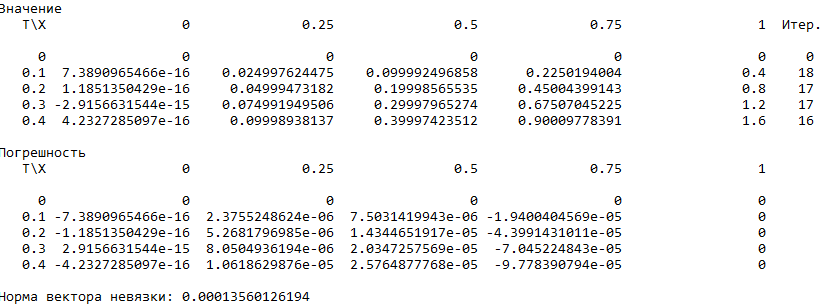
1. hx



1. hx / 2



1. hx / 4



Порядок аппроксимации  -0.037261328756131126

0.39126557491832414

1. Тесты, сетка неравномерная

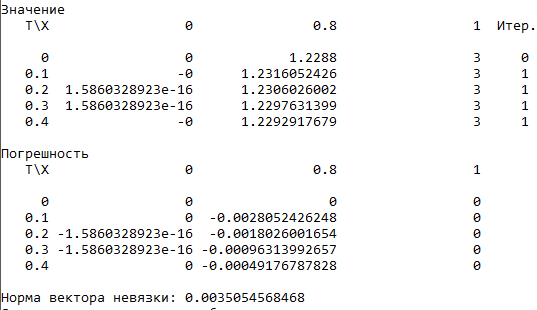
Искомая функция:

Уравнение:

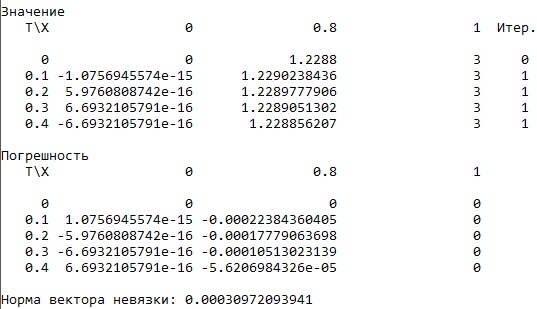
Краевые условия: 0

Сетка сходится к 1

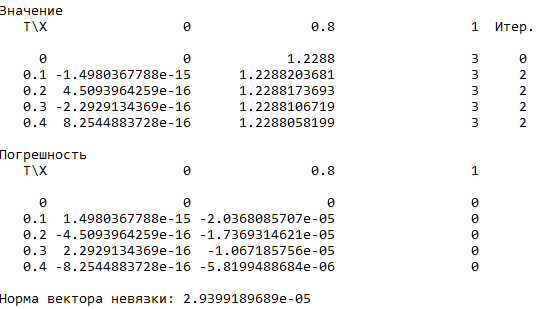
1. hx



1. hx/2



1. hx/4



Порядок сходимости 3.500561646751

3.397120626444676

*Метод считает 3 полином без погрешности, а на 4 появляется погрешность, поэтому порядок аппроксимации 3.*

1. Текст программы

*/\*Includes.h\*/*

#pragma once

#include <iostream>

#include <vector>

#include <functional>

#include <sstream>

#include <iomanip>

#include <Windows.h>

#include <math.h>

using namespace std;

*/\*FEM.h\*/*

#include "Includes.h"

vector<double> operator\*(vector<vector<double>> a, vector<double> b)

{

vector<double> c(b.size());

double sum = 0;

int m = a.size();

for (size\_t i = 0; i < m; i++)

{

sum = 0;

for (size\_t j = 0; j < m; j++)

sum += a[i][j] \* b[j];

c[i] = sum;

}

return c;

}

vector<double> operator+(vector<double> a, vector<double> b)

{

vector<double> c(a.size());

double sum = 0;

for (size\_t i = 0; i < a.size(); i++)

c[i] = a[i] + b[i];

return c;

}

vector<double> operator-(vector<double> a, vector<double> b)

{

vector<double> c(a.size());

double sum = 0;

for (size\_t i = 0; i < a.size(); i++)

c[i] = a[i] - b[i];

return c;

}

typedef void(\*Gause)(std::vector<std::vector<double>> &A, std::vector<double> &F, std::vector<double> &X);

vector<double> operator\*(vector<double> a, double b)

{

for (size\_t i = 0; i < a.size(); i++)

a[i] \*= b;

return a;

}

class FEM

{

public:

#pragma region Public struct

//Базисы

struct Basis

{

enum BasisType

{

Lagrange,

Hermite

};

BasisType type;

int order;

};

//Граничные условия

struct Border

{

public:

enum TypeBorder

{

First,

Second,

Third

};

struct Grad

{

Grad(function<double(double)> a) :x(a){}

function<double(double)>& x;

};

TypeBorder type;

function<double(double)>& U;

Grad Thetta;

//Задание первой краевой функции

Border(function<double(double)> &f, TypeBorder t) : U(f), Thetta(f)

{

type = t;

}

};

#pragma endregion

FEM() { lambda = 1;};

~FEM() {};

double eps; //Погрешнасть решения

int maxIter; //Максимальное еоличестов итераций на один слой

int lastIter; //Текущее количество итераций на слое

vector<int> It;

//Запуск метода конечных элементов

void Start()

{

checkdata();

Preprocessing();

auto hinstLib = LoadLibrary(TEXT("LDU.dll"));

Gause LDU = (Gause)GetProcAddress(hinstLib, "Gause");

vector<double> Zsigma(basis.order + 1);

vector<double> Zsigmalast(basis.order + 1);

vector<double> localF(basis.order + 1);

vector<double> qlast = q;

vector<double> qlast2 = q;

double len;

double dt;

Matrix c;

Matrix Alast;

for (size\_t l = 1; l < t.size(); l++)

{

lastIter = 0;

do

{

dt = t[l] - t[l - 1];

ClearLastIter();

for (size\_t i = 0; i < x.size() - 1; i++)

{

localF[0] = f(x[i], t[l]);

localF[1] = f((x[i + 1] + x[i]) / 2, t[l]);

localF[2] = f(x[i + 1], t[l]);

len = x[i + 1] - x[i];

Zsigma[0] = sigma(x[i], t[l], (q[2 \* i + 1] - q[2 \* i]) / (len / 2));

Zsigma[1] = sigma((x[i+1]+x[i])/2, t[l], (q[2 \* i + 2] - q[2 \* i]) / len);

Zsigma[2] = sigma(x[i+1], t[l], (q[2 \* i + 2] - q[2 \* i + 1]) / (len / 2));

M = CreateMatrixС(Zsigma \* len) / dt;

blocal = (CreateMatrixС(len)) \* localF + M \* vector<double>{ qlast2[2 \* i], qlast2[2 \* i + 1], qlast2[2 \* i + 2] };

Alocal = M + G / len;

AddLocalMatrix(i);

}

AddBorderInMatrix(t[l]);

auto kj = Norm((A \* q) - b);

if (Norm((A \* q) - b) / Norm(b) < eps) break;

qlast = q;

//Тут решаем уравнение A q = b

LDU(A, b, q);

} while (++lastIter < maxIter && Norm(q - qlast) / Norm(q) > eps);

It.push\_back(lastIter);

qlast2 = q;

//т.к. базис лагранджа то u = q

u.push\_back(q);

}

}

//Высвобождение памяти //кототрого нет!!!

void Clear()

{

}

void GetResult(vector<vector<double>> &U)

{

U = u;

}

#pragma region SetFunctions

//Задать базис

void SetBasis(Basis b)

{

basis = b;

}

//Задать начатьное приближение

void SetQ0(vector<double> q0)

{

q = q0;

}

//Задать Сигму

void SetSigma(function<double(double, double, double)> Sigma)

{

sigma = Sigma;

}

//Добавление стороны

void AddBorder(Border b)

{

border.push\_back(b);

}

//Дабавление лямбды

void SetLambda(double l)

{

lambda = l;

}

//Задание правой части

void SetF(function<double(double, double)> func)

{

f = func;

}

//Set t

void SetT(vector<double> T)

{

t = T;

}

//SetX

void SetX(vector<double> X)

{

x = X;

}

#pragma endregion

private:

#pragma region Privat struct

struct ShapeIndex

{

int ixMax; //Максимальный индекс

};

struct Matrix

{

Matrix()

{

n = 0;

}

Matrix(int N)

{

n = N;

x.resize(n);

for (size\_t i = 0; i < n; i++)

x[i].resize(n);

}

Matrix(vector<vector<double>> x0)

{

x = x0;

n = x0.size();

}

vector<vector<double>> x;

size\_t n;

void Clear()

{

x.clear();

}

Matrix operator\*(double b)

{

Matrix a(x);

for (size\_t i = 0; i < a.x.size(); i++)

for (size\_t j = 0; j < a.x[i].size(); j++)

a.x[i][j] \*= b;

return a;

}

Matrix operator/(double b)

{

Matrix a(x);

for (size\_t i = 0; i < a.x.size(); i++)

for (size\_t j = 0; j < a.x[i].size(); j++)

a.x[i][j] /= b;

return a;

}

Matrix operator+(Matrix b)

{

Matrix c(3);

for (size\_t i = 0; i < n; i++)

for (size\_t j = 0; j < n; j++)

c.x[i][j] = x[i][j] + b.x[i][j];

return c;

}

void operator\*=(double b)

{

for (size\_t i = 0; i < x.size(); i++)

for (size\_t j = 0; j < x[i].size(); j++)

x[i][j] \*= b;

}

vector<double> operator\*(vector<double> b)

{

vector<double> a(n);

for (size\_t i = 0; i < n; i++)

for (size\_t j = 0; j < n; j++)

a[i] += x[i][j] \* b[j];

return a;

}

};

#pragma endregion

//Дано

Basis basis; //Базис

vector<Border> border; //Границы

vector<double> t; //Время

vector<double> x; //Координаты точке

vector<double> q; //Начальное значение

double lambda; //Лямбда

function<double(double, double, double)> sigma; //Сигма

function<double(double, double)> f; //Правая часть

//Результат

vector<vector<double>> u; //Искомая функция

//Переходные вычисления

ShapeIndex index; //Форма в координатах интексов

int k; //Количество локальных матриц

Matrix M; //Матрица массы

Matrix G; //Матрица жесткости

Matrix Alocal; //Локальная матрица левой части

//vector<function<double(double)>> fbasis; //Базисы

vector<vector<double>> A; //Матрица левой части в диагональном виде

vector<double> b; //Вектора правой части

vector<double> blocal; //Локальный вектор правой части

//Обработка введеных данных

void Preprocessing()

{

index.ixMax = 2 \* x.size() - 1;

A.resize(index.ixMax);

for (size\_t i = 0; i < index.ixMax; i++)

A[i].resize(index.ixMax);

b.resize(index.ixMax);

blocal.resize(basis.order + 1);

Alocal = Matrix(basis.order + 1);

if (!q.size())

q.resize(index.ixMax);

k = (index.ixMax - 1.0) / 2.0; //Определяем количество локальных матриц

add\_basis(basis);

}

//Добавление базисныйх функций и матрыцы жесткости

void add\_basis(Basis b)

{

switch (b.type)

{

case b.Lagrange:

switch (b.order)

{

case 2:

{

//fbasis.resize(3);

//fbasis[0] = [](double x) { return 2 \* (x - 0.5) \* (x - 1); };

//fbasis[1] = [](double x) { return -4 \* x \* (x - 1); };

//fbasis[2] = [](double x) { return 2 \* x \* (x - 0.5); };

double c = lambda / 3;

G = Matrix({ {7, -8, 1}, {-8, 16, -8}, {1, -8, 7} });

G \*= c;

M.x.resize(3);

for (size\_t i = 0; i < M.x.size(); i++)

M.x[i].resize(3);

break;

}

default:

break;

}

break;

default:

break;

}

}

//Провкрка введенных данных

void checkdata()

{

if (border.size() < 2)

throw "all boundaries are not indicated";

if (lambda == 0)

throw "error lambda";

}

//Создание матрицы массы

Matrix CreateMatrixС(vector<double> vx)

{

Matrix C(3);

C.x[0][0] = vx[0] \* 0.09285714285714286 + vx[1] \* 0.04761904761904761 + vx[2] \* -7.1428571428571415e-3;

C.x[0][1] = vx[0] \* 0.04761904761904761 + vx[1] \* 0.0380952380952381 + vx[2] \* -0.01904761904761905;

C.x[0][2] = vx[0] \* -7.1428571428571415e-3 + vx[1]\* -0.01904761904761905 + vx[2] \* -7.1428571428571415e-3;

C.x[1][0] = vx[0] \* 0.04761904761904761 + vx[1] \* 0.0380952380952381 + vx[2] \* -0.01904761904761905;

C.x[1][1] = vx[0] \* 0.0380952380952381 + vx[1] \* 0.4571428571428571 + vx[2] \* 0.03809523809523809;

C.x[1][2] = vx[0] \* -0.01904761904761905 + vx[1] \* 0.03809523809523809 + vx[2] \* 0.04761904761904761;

C.x[2][0] = vx[0] \* -7.1428571428571415e-3 + vx[1] \* -0.01904761904761905 + vx[2] \* -7.1428571428571415e-3;

C.x[2][1] = vx[0] \* -0.01904761904761905 + vx[1] \* 0.03809523809523809 + vx[2] \* 0.04761904761904761;

C.x[2][2] = vx[0] \* -7.1428571428571415e-3 + vx[1] \* 0.04761904761904761 + vx[2] \* 0.09285714285714286;

return C;

}

//Создание матрицы массы

Matrix CreateMatrixС(double x)

{

Matrix C(3);

C.x[0][0] = x \* 0.13333333333333333;

C.x[0][1] = x \* 0.06666666666666667;

C.x[0][2] = x \* -0.03333333333333333;

C.x[1][0] = x \* 0.06666666666666667;

C.x[1][1] = x \* 0.5333333333333333;

C.x[1][2] = x \* 0.06666666666666667;

C.x[2][0] = x \* -0.03333333333333333;

C.x[2][1] = x \* 0.06666666666666667;

C.x[2][2] = x \* 0.13333333333333333;

return C;

}

//Добавление локальных матриц в глобальную

void AddLocalMatrix(int location)

{

for (size\_t i = 0, il = location \* basis.order; i <= basis.order; i++, il++)

{

for (size\_t j = 0, jl = location \* basis.order; j <= basis.order; j++, jl++)

A[il][jl] += Alocal.x[i][j];

b[il] += blocal[i];

}

}

//Очистка A и b c предыдущей итерации

void ClearLastIter()

{

for (size\_t i = 0; i < A.size(); i++)

{

for (size\_t j = 0; j < A[i].size(); j++)

A[i][j] = 0;

b[i] = 0;

}

}

//Добавление краевых условаий в матрицу А и b

void AddBorderInMatrix(double t)

{

switch (border[0].type)

{

case Border::First:

A[0][0] = 1;

for (size\_t i = 1; i < A[0].size(); i++)

A[0][i] = 0;

b[0] = border[0].U(t);

break;

default:

break;

}

switch (border[1].type)

{

case Border::First:

A[A.size() - 1][A.size() - 1] = 1;

for (size\_t i = 0; i < A[0].size() - 1; i++)

A[A.size() - 1][i] = 0;

b[A.size() - 1] = border[1].U(t);

break;

default:

break;

}

}

//Норма

double Norm(vector<double> b)

{

double sum = 0;

for (auto &i : b)

sum += pow(i, 2);

return sqrt(sum);

}

};

#include "FEM.h"

//Константы точности

#define EPS 1e-14 //Погрешность невязки

#define DELTA 1e-14 //Погрешность разности шага решений

#define MAXITER 100000 //Максимальное количество итераций на каждый метод

//X и T

//#define X {0, 0.25, 0.5, 0.75, 1}

#define T {0, 0.1, 0.2, 0.3, 0.4}

//Значение констант

#define lambda 1 //Лябда

//Значение функций

#define Sigma\_Function 1 //Функция сигма от x t и dudx

//#define U pow(x,10) - pow(x,9) //Искомая функция функция от x и t

//

//#define DuDx 10 \* pow(x,9) - 9 \* pow(x,8) //Первая производная функция от x и t

//#define DivGrad 90 \* pow(x,8) - 72 \* pow(x,7) //DivGrad функция от x и t

//

//#define DuDt 0

#pragma region Grid

//#define X {0, 0.25,0.5, 0.75, 1}

//#define X {0, 0.125, 0.25, 0.375,0.5,0.675, 0.75,0.875, 1}

//#define X {0, 0.0625, 0.125, 0.1875, 0.25, 0.3125, 0.375, 0.4375, 0.5, 0.5625, 0.675, 0.7375, 0.75, 0.8125, 0.875, 0.9375, 1}

//#define HX2

#define HX4

//#define X {0, 0.8, 1}

//#define X {0, 0.5333333333333333, 0.8, 0.9333333333333333, 1}

#define X {0, 0.3555555555555555, 0.5333333333333333, 0.711111111111111, 0.8, 0.888888888888889, 0.9333333333333333, 0.9777777777777777, 1}

//#define T {0, 0.1, 0.2, 0.3, 0.4}

//#define T {0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4}

//#define T {0, 0.025, 0.05, 0.75, 0.1, 0.125, 0.15, 0.175, 0.2, 0.225, 0.25, 0.275, 0.3, 0.325, 0.35, 0.375, 0.4}

//#define H2

//#define H4

#pragma endregion

#pragma region TESTS!

//#define testX

//#define testXX

//#define testXXX

//#define testXXXX

//#define testT

//#define testTT

//#define test4XXT

//#define testXT

#define test3XXXX

#ifdef testX

#define U x //Искомая функция функция от x и t

#define DuDx 1 //Первая производная функция от x и t

#define DivGrad 0 //DivGrad функция от x и t

#define DuDt 0

#endif

#ifdef testXX

#define U x\*x //Искомая функция функция от x и t

#define DuDx 2\*x //Первая производная функция от x и t

#define DivGrad 2 //DivGrad функция от x и t

#define DuDt 0

#endif

#ifdef testXXX

#define U x\*x\*x //Искомая функция функция от x и t

#define DuDx 3\*x\*x //Первая производная функция от x и t

#define DivGrad 6\*x //DivGrad функция от x и t

#define DuDt 0

#endif

#ifdef testXXXX

#define U x\*x\*x\*x //Искомая функция функция от x и t

#define DuDx 4\*x\*x\*x //Первая производная функция от x и t

#define DivGrad 12\*x\*x //DivGrad функция от x и t

#define DuDt 0

#endif

#ifdef testT

#define U t //Искомая функция функция от x и t

#define DuDx 0 //Первая производная функция от x и t

#define DivGrad 0 //DivGrad функция от x и t

#define DuDt 1

#endif

#ifdef testTT

#define U t\*t //Искомая функция функция от x и t

#define DuDx 0 //Первая производная функция от x и t

#define DivGrad 0 //DivGrad функция от x и t

#define DuDt 2\*t

#endif

#ifdef testXT

#define U x\*t //Искомая функция функция от x и t

#define DuDx t //Первая производная функция от x и t

#define DivGrad 0 //DivGrad функция от x и t

#define DuDt x

#endif

#ifdef test4XXT

#define U 4\*x\*x\*t //Искомая функция функция от x и t

#define DuDx 8\*x\*t //Первая производная функция от x и t

#define DivGrad 8\*t //DivGrad функция от x и t

#define DuDt 4\*x\*x

#endif

#ifdef test3XXXX

#define U 3\*x\*x\*x\*x //Искомая функция функция от x и t

#define DuDx 12\*x\*x\*x //Первая производная функция от x и t

#define DivGrad 36\*x\*x //DivGrad функция от x и t

#define DuDt 0

#endif

#pragma endregion

int main()

{

FEM m;

#pragma region Функции

//Искомая функция

function<double(double, double)> u = [](double x, double t) {return U; };

//Првая производная

function<double(double, double)> dudx = [](double x, double t) {return DuDx; };

//Првая производная по времени

function<double(double, double)> dudt = [](double x, double t) {return DuDt; };

//Сумма частных вторых производных

function<double(double, double)> divGrad = [](double x, double t) {return DivGrad; };

#pragma endregion

#pragma region Сигма

function<double(double, double, double)> sigma = [](double x, double t, double dudx) {return Sigma\_Function; };

m.SetSigma(sigma);

#pragma endregion

#pragma region Задаем базисы

FEM::Basis basis;

basis.type = basis.Lagrange;

basis.order = 2;

m.SetBasis(basis);

#pragma endregion

#pragma region Задаем константы

double NX = vector<double>X.size();

m.SetLambda(lambda);

m.eps = EPS;

m.maxIter = MAXITER;

#pragma endregion

#pragma region Задаем краевые условия

function<double(double)> f1 = [&](double t) { return u(vector<double>X[0], t); };

function<double(double)> f2 = [&](double t) { return u(vector<double>X[NX-1], t); };

FEM::Border b1(f1, FEM::Border::First), b2(f2, FEM::Border::First);

m.AddBorder(b1);

m.AddBorder(b2);

#pragma endregion

#pragma region Задаем форму

m.SetX(X);

m.SetT(T);

#pragma endregion

#pragma region Задаем начальное значение

vector<double> q(2 \* NX - 1);

double dx = (vector<double>X[NX - 1] - vector<double>X[0]) / (NX - 1);

for (size\_t i = 0; i < NX-1; i++)

{

q[2 \* i] = u(vector<double>X[i], vector<double>T[0]);

q[2 \* i + 1] = u((vector<double>X[i + 1] + vector<double>X[i]) / 2, vector<double>T[0]);

}

q[(NX - 1) \* 2] = u(vector<double>X[NX-1], vector<double>T[0]);

m.SetQ0(q);

#pragma endregion

#pragma region Задаем правую часть уравнения

function<double(double, double)> F = [&](double x, double t) {return -1 \* lambda \* divGrad(x, t) + sigma(x, t, dudx(x, t)) \* dudt(x, t);};

m.SetF(F);

#pragma endregion

m.Start();

vector<vector<double>> Result;

m.GetResult(Result);

#pragma region Вывод

#pragma region Определение выводимых строк

int h = 1, hx = 1;

#ifdef H2

h = 2;

#endif

#ifdef H4

h = 4;

#endif

#ifdef HX2

hx = 2;

#endif

#ifdef HX4

hx = 4;

#endif

#pragma endregion

cout.precision(11);

cout << "Значение" << endl;

cout << setw(6) << "T\\X";

for (size\_t j = 0; j < vector<double>X.size(); j++)

if (j % hx == 0)

cout << setw(18) << vector<double>X[j];

cout << " Итер. в ит. проц."<< endl << endl;

cout << setw(6) << vector<double>T[0];

for (size\_t j = 0; j < vector<double>X.size(); j++)

if (j % hx == 0)

cout << setw(18) << q[2 \*j];

cout << setw(6) << 0 << endl;

for (size\_t i = 0, i1 = 1; i < Result.size(); i++, i1++)

{

if (i1 % h == 0)

{

cout << setw(6) << vector<double>T[i + 1];

for (size\_t j = 0; j < vector<double>X.size(); j++)

if (j % hx == 0)

cout << setw(18) << Result[i][2 \* j];

cout << setw(6) << m.It[i] << endl;

}

}

cout << endl << "Погрешность" << endl;

cout << setw(6) << "T\\X";

for (size\_t j = 0; j < vector<double>X.size(); j++)

if (j % hx ==0)

cout << setw(18) << vector<double>X[j];

cout << endl << endl;;

cout << setw(6) << vector<double>T[0];

for (size\_t j = 0; j < vector<double>X.size(); j++)

if (j % hx == 0)

cout << setw(18) << 0;

cout << endl;

for (size\_t i = 0, i1 = 1; i < Result.size(); i++, i1++)

{

if (i1 % h == 0)

{

cout << setw(6) << vector<double>T[i1];

for (size\_t j = 0; j < vector<double>X.size(); j++)

if(j % hx == 0)

cout << setw(18) << u(vector<double>X[j], vector<double>T[i1]) - Result[i][2 \* j];

cout << endl;

}

}

cout << endl;

//Невязка

double snev = 0;

for (size\_t i = 0, i1 = 1; i < Result.size(); i++, i1++)

if (i1 % h == 0)

for (size\_t j = 0; j < vector<double>X.size(); j++)

if (j % hx == 0)

snev += pow(u(vector<double>X[j], vector<double>T[i1]) - Result[i][2 \* j] , 2);

cout << "Норма вектора невязки: " << sqrt(snev) <<endl;

#pragma endregion

system("pause");

return 0;

}