Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №2

по уравнениям математической физики

Факультет: ПМИ

Группа: ПМ-01

Студент: Ряховский М.И.

Вариант: 11

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

Новосибирск

2013

# Цель работы

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

# Постановка задачи

Дано параболическое начально-краевое уравнение относительной функции :

, 

C начальным условием:

,

И некоторыми краевыми условиями, при  и при .

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

# Дискретизация по времени

Аппроксимируем производную искомой функции по времени следующей неявной схемой:



Тогда исходное уравнение можно переписать, как эллиптическое уравнение:



Введём обозначения:







Получаем:



# Вариационная постановка и дискретизация

Рассмотрим гильбертово пространство  со своим скалярным произведением и нормой:

, .

Выполним для уравнения вариационную постановку Галёркина: домножим скалярно правую и левую часть на функцию , где - множество функций, удовлетворяющих однородным первым краевым условиям. В данном случае рассмотрим следующие краевые условия: , , тогда получим:





При этом, функция , базисные функции – линейные .

# Вычисление элементов матрицы для метода простой итерации

Коротко метод простой итерации записывается следующим выражением: 

Элементы матрицы  вычисляются по следующим формулам:



И к последнему элементу идёт добавка от краевых условий: 

Учитывая то, что базисные функции линейные и финитные эти выражения можно преобразовать и использовать локальные матрицы для сбора. Общая структура матрицы будет трёхдиагональной:





Тогда локальная матрица вычисляется следующим образом







Вычисление локального вектора правой части:



Раскрыв скобки и вычислив интегралы, получим:



# Вычисление элементов матрицы для метода Ньютона

Формирование локальных линеаризованных матриц в методе Ньютона происходит по следующей формуле:





Вычислим производные, которые входят в формулы:





Пусть зависимость  от  задана следующим образом: , введём функцию , тогда выражение для производной можно переписать, как:





В матричном виде:









# Параметры решения СЛАУ

Для повышения эффективности работы методов будем использовать параметр релаксации , при итерационном решении нелинейной системы (для обоих методов) и коэффициент демпфирования , при сборке матрицы для решения методом Ньютона. Параметр релаксации будет побираться программно, исходя из следующего условия: он должен доставлять минимум следующему функционалу:

, где 

Коэффициент демпфирования будет выбираться в ручную.

За начальное приближение на каждом временном слое будем выбирать решение этого уранения на предыдущем временном слое.

# Тесты

## Тест 1. Проверка решения линейной стационарной задачи.

Тестовая функция: 

Уравнение: 

Краевые условия: первого рода

Сетка: разбиение отрезка  c шагом 

Сетка по времени: , 

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 0.100000000000000 | 0.100000000000000 | 0.100000000000000 | 0.100000000000000 |
| 0.200000000000000 | 0.200000000000000 | 0.200000000000000 | 0.200000000000000 |
| 0.300000000000000 | 0.300000000000000 | 0.300000000000000 | 0.300000000000000 |
| 0.400000000000000 | 0.400000000000000 | 0.400000000000000 | 0.400000000000000 |
| 0.500000000000000 | 0.500000000000000 | 0.500000000000000 | 0.500000000000000 |
| 0.600000000000000 | 0.600000000000000 | 0.600000000000000 | 0.600000000000000 |
| 0.700000000000000 | 0.700000000000000 | 0.700000000000000 | 0.700000000000000 |
| 0.800000000000000 | 0.800000000000000 | 0.800000000000000 | 0.800000000000000 |
| 0.900000000000000 | 0.900000000000000 | 0.900000000000000 | 0.900000000000000 |

## Тест 2. Проверка решения линейной не стационарной задачи

Тестовая функция: 

Уравнение: 

Краевые условия: 

Сетка: разбиение отрезка  c шагом 

Сетка по времени: , 

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0.01 | 7.323e-016 | 7.323e-016 |
| 0.02 | 1.077e-015 | 1.077e-015 |
| 0.03 | 2.884e-015 | 2.884e-015 |
| 0.04 | 1.250e-015 | 1.250e-015 |
| 0.05 | 2.352e-015 | 2.352e-015 |

## Тест 3. Простой тест для нелинейной задачи

Тестовая функция: 

Уравнение: 

Краевые условия: ,

Сетка: разбиение отрезка  c шагом 

Сетка по времени: , 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | si, итерации | N, итерации |
| 0.01 | 1.24E-04 | 1.24E-04 | 3.84E-05 | 3.84E-05 | 3.23E+00 | 3.23E+00 | 5.00 | 4.00 |
| 0.02 | 2.39E-04 | 2.39E-04 | 7.35E-05 | 7.35E-05 | 3.25E+00 | 3.25E+00 | 6.00 | 7.00 |
| 0.03 | 3.46E-04 | 3.46E-04 | 1.06E-04 | 1.06E-04 | 3.27E+00 | 3.27E+00 | 6.00 | 3.00 |
| 0.04 | 4.45E-04 | 4.45E-04 | 1.36E-04 | 1.36E-04 | 3.28E+00 | 3.28E+00 | 6.00 | 4.00 |
| 0.05 | 5.39E-04 | 5.39E-04 | 1.64E-04 | 1.64E-04 | 3.30E+00 | 3.30E+00 | 6.00 | 4.00 |
| 0.06 | 6.27E-04 | 6.27E-04 | 1.90E-04 | 1.90E-04 | 3.31E+00 | 3.31E+00 | 7.00 | 4.00 |
| 0.07 | 7.10E-04 | 7.10E-04 | 2.14E-04 | 2.14E-04 | 3.32E+00 | 3.32E+00 | 7.00 | 4.00 |

## Тест 4. Задача с не полиномиальным решением

Тестовая функция: 

Уравнение: 

Краевые условия: ,

Сетка: разбиение отрезка  c шагом 

Сетка по времени:, 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | si, итерации | N, итерации |
| 0.01 | 3.06E-03 | 3.06E-03 | 2.94E-03 | 2.94E-03 | 1.04E+00 | 1.04E+00 | 99.00 | 4.00 |
| 0.02 | 7.52E-03 | 7.52E-03 | 7.34E-03 | 7.34E-03 | 1.03E+00 | 1.03E+00 | 67.00 | 4.00 |
| 0.03 | 1.30E-02 | 1.30E-02 | 1.28E-02 | 1.28E-02 | 1.02E+00 | 1.02E+00 | 52.00 | 11.00 |
| 0.04 | 1.94E-02 | 1.94E-02 | 1.92E-02 | 1.92E-02 | 1.01E+00 | 1.01E+00 | 44.00 | 47.00 |
| 0.05 | 2.65E-02 | 2.65E-02 | 2.63E-02 | 2.63E-02 | 1.01E+00 | 1.01E+00 | 56.00 | 99.00 |
| 0.06 | 3.43E-02 | 3.43E-02 | 3.40E-02 | 3.40E-02 | 1.01E+00 | 1.01E+00 | 99.00 | 99.00 |
| 0.07 | 4.26E-02 | 4.26E-02 | 4.23E-02 | 4.23E-02 | 1.01E+00 | 1.01E+00 | 99.00 | 99.00 |

На первой итерации по времени, для метода простой итерации:

|  |  |  |
| --- | --- | --- |
| Номер итерации | Относительная невязка | Параметр релаксации |
| 0 | 1.65E-03 | 1.04 |
| 1 | 8.61E-04 | 0.64 |
| 2 | 6.27E-04 | 0.45 |
| 3 | 4.68E-04 | 0.48 |
| 4 | 3.21E-04 | 0.52 |
| 5 | 2.22E-04 | 0.48 |
| 6 | 1.63E-04 | 0.46 |
| 7 | 1.18E-04 | 0.49 |
| 8 | 8.17E-05 | 0.50 |
| 9 | 5.83E-05 | 0.47 |
| 10 | 4.32E-05 | 0.46 |
| 11 | 3.14E-05 | 0.48 |
| 12 | 2.21E-05 | 0.49 |
| 13 | 1.59E-05 | 0.47 |
| 14 | 1.18E-05 | 0.46 |
| 15 | 8.55E-06 | 0.48 |
| 16 | 6.01E-06 | 0.49 |
| 17 | 4.32E-06 | 0.47 |
| 18 | 3.19E-06 | 0.46 |
| 19 | 2.33E-06 | 0.48 |
| 20 | 1.64E-06 | 0.49 |
| 21 | 1.17E-06 | 0.47 |
| 22 | 8.65E-07 | 0.46 |
| 23 | 6.32E-07 | 0.48 |
| 24 | 4.45E-07 | 0.49 |
| 25 | 3.17E-07 | 0.47 |
| 26 | 2.34E-07 | 0.46 |
| 27 | 1.71E-07 | 0.48 |
| 28 | 1.21E-07 | 0.50 |
| 29 | 8.56E-08 | 0.47 |
| 30 | 6.30E-08 | 0.46 |
| 31 | 4.63E-08 | 0.48 |
| 32 | 3.27E-08 | 0.50 |
| 33 | 2.31E-08 | 0.48 |
| 34 | 1.70E-08 | 0.46 |
| 35 | 1.25E-08 | 0.47 |
| 36 | 8.88E-09 | 0.50 |
| 37 | 6.25E-09 | 0.48 |
| 38 | 4.57E-09 | 0.46 |
| 39 | 3.38E-09 | 0.47 |
| 40 | 2.41E-09 | 0.50 |
| 41 | 1.69E-09 | 0.48 |
| 42 | 1.23E-09 | 0.46 |
| 43 | 9.12E-10 | 0.47 |
| 44 | 6.53E-10 | 0.49 |
| 45 | 4.57E-10 | 0.49 |
| 46 | 3.32E-10 | 0.46 |
| 47 | 2.46E-10 | 0.46 |
| 48 | 1.77E-10 | 0.49 |
| 49 | 1.24E-10 | 0.49 |
| 50 | 8.95E-11 | 0.46 |
| 51 | 6.64E-11 | 0.46 |
| 52 | 4.80E-11 | 0.49 |
| 53 | 3.36E-11 | 0.49 |
| 54 | 2.42E-11 | 0.46 |
| 55 | 1.79E-11 | 0.46 |
| 56 | 1.30E-11 | 0.48 |
| 57 | 9.11E-12 | 0.49 |
| 58 | 6.52E-12 | 0.46 |
| 59 | 4.83E-12 | 0.46 |
| 60 | 3.52E-12 | 0.48 |
| 61 | 2.47E-12 | 0.49 |
| 62 | 1.76E-12 | 0.47 |
| 63 | 1.30E-12 | 0.46 |
| 64 | 9.53E-13 | 0.48 |
| 65 | 6.70E-13 | 0.49 |
| 66 | 4.76E-13 | 0.47 |
| 67 | 3.50E-13 | 0.46 |
| 68 | 2.57E-13 | 0.48 |
| 69 | 1.82E-13 | 0.49 |
| 70 | 1.29E-13 | 0.47 |
| 71 | 1.40E-13 | 1.00 |
| 72 | 9.91E-14 | 0.52 |
| 73 | 1.04E-13 | 1.00 |
| 74 | 7.62E-14 | 0.47 |
| 75 | 7.70E-14 | 1.00 |
| 76 | 5.62E-14 | 0.44 |
| 77 | 4.14E-14 | 0.49 |
| 78 | 4.20E-14 | 1.00 |
| 79 | 3.09E-14 | 0.49 |
| 80 | 3.24E-14 | 1.00 |
| 81 | 3.48E-14 | 1.01 |
| 82 | 3.60E-14 | 1.00 |
| 83 | 3.84E-14 | 1.00 |
| 84 | 2.77E-14 | 0.48 |
| 85 | 2.82E-14 | 1.00 |
| 86 | 3.12E-14 | 1.00 |
| 87 | 3.15E-14 | 1.00 |
| 88 | 3.42E-14 | 1.00 |
| 89 | 3.44E-14 | 0.99 |
| 90 | 3.78E-14 | 1.00 |
| 91 | 3.79E-14 | 1.00 |
| 92 | 4.19E-14 | 1.00 |
| 93 | 4.18E-14 | 1.00 |
| 94 | 3.10E-14 | 0.45 |
| 95 | 3.28E-14 | 1.00 |
| 96 | 3.45E-14 | 1.00 |
| 97 | 3.56E-14 | 1.00 |
| 98 | 2.53E-14 | 0.45 |
| 99 | 1.88E-14 | 0.45 |

Для метода Ньютона:

|  |  |  |
| --- | --- | --- |
| Номер итерации | Относительная невязка | Параметр релаксации |
| 0 | 1.65E-03 | 1.04 |
| 1 | 2.29E-04 | 0.85 |
| 2 | 2.35E-06 | 0.99 |
| 3 | 2.59E-10 | 1.00 |
| 4 | 9.80E-15 | 1.00 |

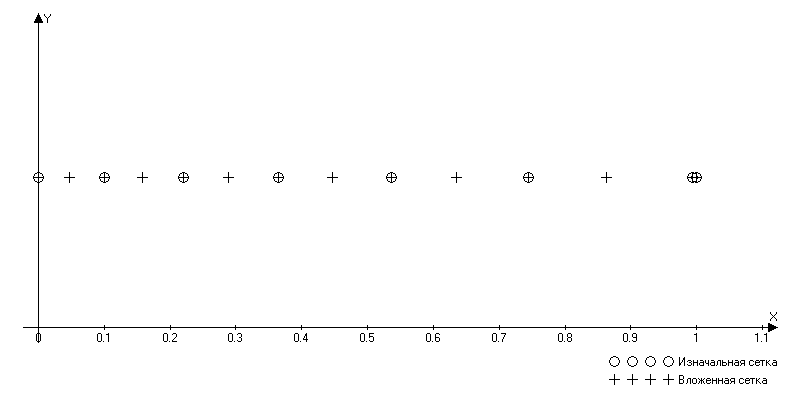
## Тест 5. Решение на неравномерной сетке

Тестовая функция: 

Уравнение: 

Краевые условия: ,

Сетка: , (коэффициент разрядки)



Сетка по времени:, , 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | si, итерации | N, итерации |
| 0.01 | 4.41E-04 | 4.41E-04 | 1.45E-04 | 1.45E-04 | 3.04E+00 | 3.04E+00 | 9.00 | 3.00 |
| 0.021 | 8.87E-04 | 8.87E-04 | 2.90E-04 | 2.90E-04 | 3.06E+00 | 3.06E+00 | 8.00 | 99.00 |
| 0.0331 | 1.34E-03 | 1.34E-03 | 4.34E-04 | 4.34E-04 | 3.08E+00 | 3.08E+00 | 99.00 | 99.00 |
| 0.04641 | 1.79E-03 | 1.79E-03 | 5.77E-04 | 5.77E-04 | 3.11E+00 | 3.11E+00 | 6.00 | 99.00 |
| 0.061051 | 2.25E-03 | 2.25E-03 | 7.20E-04 | 7.20E-04 | 3.13E+00 | 3.13E+00 | 99.00 | 7.00 |
| 0.0771561 | 2.71E-03 | 2.71E-03 | 8.62E-04 | 8.62E-04 | 3.15E+00 | 3.15E+00 | 99.00 | 4.00 |
| 0.09487171 | 3.17E-03 | 3.17E-03 | 1.00E-03 | 1.00E-03 | 3.16E+00 | 3.16E+00 | 9.00 | 8.00 |

# Выводы

На приведённых задачах, метод Ньютона оказался лучше метода простой итерации, так как в среднем, количество итераций, требуемое для решение нелинейной системы меньше, чем у метода простой итерации. Точность методов (на рассмотренных задачах) примерно одинаковая, а это значит что у метода простой итерации нет преимуществ с точки зрения получения решения, с точки зрения построения матрицы и получения формул для метода метод Ньютона значительно сложнее, однако учитывает особенности нелинейности (какой именно из коэффициентов зависит от параметра).

# Текст программы

## Файл «grid\_gen.h»

#pragma once

#include <math.h>

#include <stdio.h>

#include <string>

#include <vector>

using namespace std;

typedef vector<double> dvector;

// Генератор одномерных сеток

class grid\_gen{

public:

//===================Интерфейс класса геренации сетки===================

//============ Генерация неравномерной сетки ============

//a - начало отрезка

//b - конец отрезка

//h\_min - минимальный шаг

//k - коэффициент разрядки

//Возвращает true, если сетка может быть сгенерированна, false - в противном случае

//n - возвращаемый параметр, число узлов сетки

static bool generate\_unreg\_grid(double a, double b, double h\_min, double k, string file\_name); //генерация в файл с именем file\_name

static bool generate\_unreg\_grid(double a, double b, double h\_min, double k, dvector &grid\_vector); //генерация в вектор grid\_vector

static bool generate\_unreg\_grid(double a, double b, double h\_min, double k, int &n, double\*& grid\_mass); //генерация в массив grid\_vector

//============ Генерация равномерной сетки ============

//По сути, в принцепи тоже самое

static bool generate\_reg\_grid(double a, double b, double h, string file\_name); //генерация в файл с именем file\_name

static bool generate\_reg\_grid(double a, double b, double h, dvector &grid\_vector); //генерация в вектор grid\_vector

static bool generate\_reg\_grid(double a, double b, double h, int &n, double\*& grid\_mass); //генерация в массив grid\_vector

//============ Генерация вложенных сеток ============

//Для неравномерных

static bool generate\_unreg\_grid\_ins(double a, double b, double h\_min, double k, string file\_name);

static bool generate\_unreg\_grid\_ins(double a, double b, double h\_min, double k, dvector &grid\_vector);

static bool generate\_unreg\_grid\_ins(double a, double b, double h\_min, double k, int &n, double\*& grid\_mass);

//Для равномерных

static bool generate\_reg\_grid\_ins(double a, double b, double h, string file\_name);

static bool generate\_reg\_grid\_ins(double a, double b, double h, dvector &grid\_vector);

static bool generate\_reg\_grid\_ins(double a, double b, double h, int &n, double\*& grid\_mass);

private:

static int generate\_unreg\_grid\_pr(double a, double b, double h\_min, double k, double \*&grid\_mass); // сосбтвенно генерация сетки

//grid\_mass - здесь будет хранится сетка

static int generate\_reg\_grid\_pr(double a, double b, double h, double \*&grid\_mass); // сосбтвенно генерация сетки

//grid\_mass - здесь будет хранится сетка

};

## Файл «solver\_LU.h»

#pragma once

void solve\_LU(double\* dia\_0, double\* dia\_p, double\* dia\_m, double\* rp, int n);

//Решение слау с помощью метода прогонки для трёхдиагональной матрицы, ответ помещается на место правой части

## Файл «unlin\_FME.h»

#pragma once

#include <string>

#include <math.h>

#include <stdio.h>

#include "solver\_LU.h"

using namespace std;

//Метод конечных элементов для одномерной не стационарной нелизненой задачи

typedef double(\*func\_t)(double, double); //указалетль на функцию двух переменных (пространства и времени)

typedef double(\*func)(double); //указатель на функцию одной переменной (предположительно - времени)

//Стуркутра описывающая элемент - отрезок

struct element{

int node1, node2;

};

class unlin\_FEM{

public:

void init(string node\_file, string u0\_file, string info\_file);

//Метод для инициализации класса, выполняет чтение из файлов нужных данных

//node\_file - имя файла, где гранятся данные о количестве и координатах вершин

//u0\_file - имя файла, где хранятся начальные уловия

//info\_file - имя файла, где хранится информация о ветки по времени (начальный шаг и коэффциент разрядки), так же макс. число итераций на слое и требуемая точность

void init\_coefs(func\_t set\_lambda, func\_t set\_f);

//Устанавливает коэффициенты уравнения

void init\_sigma(double set\_c);

//Устанавливает коэффицент для сигма, который задаётся как: sigma(u\_x, x, t) = ((u\_x)^2 + c) \* x

void init\_bounds(int set\_b\_l, int set\_b\_r, func bound\_l1, func bound\_l2, func bound\_r1, func bound\_r2);

//Метод для инициаилазции краевых условий

//set\_b\_l - тип левых краевых

//set\_b\_r - при правых краевых

//Дальше параметры - это функции для краевых условий в случае первых или вторых краевых, допустим слева в третий пораметр подаётся функция, а в четвёртый - NULL

//В случае третих краевых в l1 - бетта, в l2 - u\_betta (ну или r1, r2)

double get\_current\_time();

//Метод для получения текущего времени

void set\_analytic(func\_t set\_u);

//Устанавливаем наалитическое решение

double operator () (double x);

//Перегрузка оператора, для получения значения функции на текущем временном слое

void simple\_iteration\_step();

//Сделать шаг по времени методом простой итерации

void Newton\_step();

//Сделать шаг по времени методом Ньютона

void out\_q(string file\_out);

//Вывод q в файл, с дописыванием

private:

double\* nodes; //массив координат узлов

element\* elements; //массив элементов

int nodes\_n; //количество узлов

int elements\_n; //количество эллементов

func\_t lambda, f; //коэффцииенты уравнения

double sigma\_c; //коэффициент для сигма

double sigma(double u\_x, double x); //функция для вычисления сигма

double sigma\_ux(double u\_x, double x); //производная сигма по u\_x

func\_t analytic; //Аналитическое решение

double\* q\_last; //массив весов с предыдущего временого слоя

double\* q\_new; //массив узлов, с нового временого слоя

double\* q\_temp; //массив узлов, для промежуточных итераций

double\* q\_min; //массив, используеммый для минимизации невязки

double\* q\_min\_rp; //массив, используеммый для минимизации невязки хранящий правую часть матрицы

double\* q\_min\_R; //массив, используемый для минимизации - вектор невзяки. Так же использутеся для определения невязки, при выходе из процесса решения СНУ

double \*dia\_0, \*dia\_p1, \*dia\_m1; //диагонали матрицы

double t\_now; //текущее время

double t\_new; //следующее время

double h\_t; //шаг по времени

double k\_t; //коэффицент разрядки по времени

int iter\_max; //макимальнео число итераций на одном временом слое

double eps; //точность для решения нелинейной системы

int bound\_type\_left, bound\_type\_right; // тип краевых условий справа и слева, значения соответсвнно могут быть 1, 2 или 3

func\* bounds\_l; //функции для краевых условйи слева

func\* bounds\_r; //функции для краевых условий справа

double norm\_q\_dif(double\* q1, double\* q2); // ||q1 - q2||

double norm\_q(double\* q1); // ||q||

void matrix\_gen\_si(double\* q\_n, double\* q\_t);

//Метод для генерации матрицы (для метода простой итерации) Поскольку он нужен и при оставлении СЛАУ и при нахождении параметра релаксации - чтобы не дублировать код

void matrix\_gen\_Newton(double\* q\_n, double\* q\_t);

//Метод генерации матрицы для метода Ньютона

//q\_n - значения на предыдущей итерации

//q\_t - будующая правая часть и затем значеня на новой итерации

double w\_min(); //Нахождение параметра релаксации

double w\_min\_func(double w); //Функция для минимизации

void w\_min\_find\_area(double& a, double& b, double delta); //Нахождение отрезка, содержшего минимум

double w\_min\_golden(double a, double b, double eps\_min); //Минимизация, методом золотого сечения

};

## Файл «grid\_gen.cpp»

#include "grid\_gen.h"

//======================================= Основная часть =======================================

int grid\_gen::generate\_unreg\_grid\_pr(double a, double b, double h\_min, double k, double \*&grid\_mass){

int n; //число узлов сетки

double n\_d = log(1 - (b-a)\*(1-k)/h\_min) / log(k);

n = n\_d;

n += 2;

grid\_mass = new double [n];

double h = h\_min; //текущий шаг

grid\_mass[0] = a;

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

grid\_mass[i] = grid\_mass[i-1] + h;

h \*= k;

}

grid\_mass[n-1] = b;

return n;

}

int grid\_gen::generate\_reg\_grid\_pr(double a, double b, double h, double \*&grid\_mass){

int n; //число узлов сетки

n = (b-a)/h;

n++;

grid\_mass = new double [n];

grid\_mass[0] = a;

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

grid\_mass[i] = grid\_mass[i-1] + h;

}

grid\_mass[n-1] = b;

return n;

}

//========================================== Интерефейс ============================================

bool grid\_gen::generate\_unreg\_grid(double a, double b, double h\_min, double k, int &n, double \*&grid\_mass){

if(a > b || h\_min <= 0 || k <= 1) return false; //проверка корректности данных

n = generate\_unreg\_grid\_pr(a, b, h\_min, k, grid\_mass);

return true;

}

bool grid\_gen::generate\_unreg\_grid(double a, double b, double h\_min, double k, dvector &grid\_vector){

if(a > b || h\_min <= 0 || k <= 1) return false; //проверка корректности данных

double \*grid\_mass;

int n = generate\_unreg\_grid\_pr(a, b, h\_min, k, grid\_mass);

grid\_vector.clear();

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

grid\_vector.push\_back(grid\_mass[i]);

delete[] grid\_mass;

return true;

}

bool grid\_gen::generate\_unreg\_grid(double a, double b, double h\_min, double k, std::string file\_name){

if(a > b || h\_min <= 0 || k <= 1) return false; //проверка корректности данных

double \*grid\_mass;

int n = generate\_unreg\_grid\_pr(a, b, h\_min, k, grid\_mass);

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

fprintf(out\_f, "%d\n", n);

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

fprintf(out\_f, "%.15lf\n", grid\_mass[i]);

}

delete[] grid\_mass;

fclose(out\_f);

return true;

}

bool grid\_gen::generate\_reg\_grid(double a, double b, double h, int &n, double \*&grid\_mass){

if(a > b || h <= 0) return false; //проверка корректности данных

n = generate\_reg\_grid\_pr(a, b, h, grid\_mass);

return true;

}

bool grid\_gen::generate\_reg\_grid(double a, double b, double h, dvector &grid\_vector){

if(a > b || h <= 0) return false; //проверка корректности данных

double \*grid\_mass;

int n = generate\_reg\_grid\_pr(a, b, h, grid\_mass);

grid\_vector.clear();

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

grid\_vector.push\_back(grid\_mass[i]);

delete[] grid\_mass;

return true;

}

bool grid\_gen::generate\_reg\_grid(double a, double b, double h, std::string file\_name){

if(a > b || h <= 0) return false; //проверка корректности данных

double \*grid\_mass;

int n = generate\_reg\_grid\_pr(a, b, h, grid\_mass);

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

fprintf(out\_f, "%d\n", n);

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

fprintf(out\_f, "%.15lf\n", grid\_mass[i]);

}

delete[] grid\_mass;

fclose(out\_f);

return true;

}

//========================================== Для вложенных сеток ============================================

bool grid\_gen::generate\_unreg\_grid\_ins(double a, double b, double h\_min, double k, string file\_name){

double k1 = sqrt(k);

double h\_min1 = h\_min/(1+k1);

return generate\_unreg\_grid(a, b, h\_min1, k1, file\_name);

}

bool grid\_gen::generate\_unreg\_grid\_ins(double a, double b, double h\_min, double k, dvector &grid\_vector){

double k1 = sqrt(k);

double h\_min1 = h\_min/(1+k1);

return generate\_unreg\_grid(a, b, h\_min1, k1, grid\_vector);

}

bool grid\_gen::generate\_unreg\_grid\_ins(double a, double b, double h\_min, double k, int &n, double\*& grid\_mass){

double k1 = sqrt(k);

double h\_min1 = h\_min/(1+k1);

return generate\_unreg\_grid(a, b, h\_min1, k1, n, grid\_mass);

}

bool grid\_gen::generate\_reg\_grid\_ins(double a, double b, double h, string file\_name){

return generate\_reg\_grid(a, b, h/2, file\_name);

}

bool grid\_gen::generate\_reg\_grid\_ins(double a, double b, double h, dvector &grid\_vector){

return generate\_reg\_grid(a, b, h/2, grid\_vector);

}

bool grid\_gen::generate\_reg\_grid\_ins(double a, double b, double h, int &n, double\*& grid\_mass){

return generate\_reg\_grid(a, b, h/2, n, grid\_mass);

}

## Файл «solver\_LU.cpp»

#include "solver\_LU.h"

void solve\_LU(double\* dia\_0, double\* dia\_p, double\* dia\_m, double\* rp, int n){

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

dia\_p[j] /= dia\_0[j];

dia\_0[j+1] -= dia\_m[j]\*dia\_p[j];

}

rp[0] /= dia\_0[0];

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

rp[i] = (rp[i] - dia\_m[i-1]\*rp[i-1])/dia\_0[i];

for(int i = n - 2; i >=0 ; i--)

rp[i] -= dia\_p[i]\*rp[i+1];

}

## Файл «unlim\_FME.cpp»

#include "unlin\_FEM.h"

void unlin\_FEM::init(string node\_file, string u0\_file, string info\_file){

FILE\* node\_f = fopen(node\_file.c\_str(), "r");

//Считывание узлов

fscanf(node\_f,"%d",&nodes\_n);

nodes = new double [nodes\_n];

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

fscanf(node\_f, "%lf", &nodes[i]);

fclose(node\_f);

//Генерация элементов

elements\_n = nodes\_n - 1;

elements = new element [elements\_n];

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

elements[i].node1 = i;

elements[i].node2 = i+1;

}

//Инициализация начальных условий

q\_last = new double [nodes\_n];

q\_new = new double [nodes\_n];

q\_temp = new double [nodes\_n];

q\_min = new double [nodes\_n];

q\_min\_rp = new double [nodes\_n];

q\_min\_R = new double [nodes\_n];

FILE\* u0\_f = fopen(u0\_file.c\_str(), "r");

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

fscanf(u0\_f, "%lf", &q\_last[i]);

fclose(u0\_f);

//Инициализация информации о сетки по времени

FILE\* info\_f = fopen(info\_file.c\_str(), "r");

fscanf(info\_f,"%lf %lf %lf %d %lf", &t\_now, &h\_t, &k\_t, &iter\_max, &eps);

fclose(info\_f);

//Инициализация матрицы

dia\_0 = new double [nodes\_n];

dia\_p1 = new double [nodes\_n - 1];

dia\_m1 = new double [nodes\_n - 1];

}

void unlin\_FEM::init\_coefs(func\_t set\_lambda, func\_t set\_f){

lambda = set\_lambda;

f = set\_f;

}

void unlin\_FEM::init\_sigma(double set\_c){

sigma\_c = set\_c;

}

void unlin\_FEM::set\_analytic(func\_t set\_u){

analytic = set\_u;

}

void unlin\_FEM::init\_bounds(int set\_b\_l, int set\_b\_r, func bound\_l1, func bound\_l2, func bound\_r1, func bound\_r2){

//Вот тут щас будет весело

bound\_type\_left = set\_b\_l;

bound\_type\_right = set\_b\_r;

//Разбираемся с левыми краевыми

switch(bound\_type\_left){

case 1: case 2: {

bounds\_l = new func [1];

bounds\_l[0] = bound\_l1;

}break;

case 3: {

bounds\_l = new func [2];

bounds\_l[0] = bound\_l1;

bounds\_l[1] = bound\_l2;

}break;

};

//Разбираемся с правыми краевыми

switch(bound\_type\_right){

case 1: case 2: {

bounds\_r = new func [1];

bounds\_r[0] = bound\_r1;

}break;

case 3: {

bounds\_r = new func [2];

bounds\_r[0] = bound\_r1;

bounds\_r[1] = bound\_r2;

}break;

};

}

double unlin\_FEM::get\_current\_time(){

return t\_now;

}

double unlin\_FEM::sigma(double u\_x, double x){

return (u\_x\*u\_x + sigma\_c)\*x;

}

double unlin\_FEM::sigma\_ux(double u\_x, double x){

return 2\*x\*u\_x;

}

void unlin\_FEM::simple\_iteration\_step(){

//В качестве начального приближения берём решение на предыдушем слое

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

q\_new[i] = q\_last[i];

t\_new = t\_now + h\_t; //новое время

//Общий цикл, по итерациям

bool end\_cycle = false;

for(int iter = 0; iter < iter\_max && !end\_cycle; iter++){

//Обнуление матрицы

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

dia\_0[i] = 0;

dia\_p1[i] = 0;

dia\_m1[i] = 0;

q\_temp[i] = 0;

}

dia\_0[nodes\_n-1] = 0;

q\_temp[nodes\_n-1] = 0;

//Генерация матрицы

matrix\_gen\_si(q\_new, q\_temp);

//Решаем слау

solve\_LU(dia\_0, dia\_p1, dia\_m1, q\_temp, nodes\_n);

//Релаксация

double w = w\_min();

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

q\_temp[i] = w\*q\_temp[i] + (1-w)\*q\_new[i];

//Проверка - нужно ли выходит из цикла

//В q\_min\_R запишим вектор невязки, в q\_new - вектор праовй части

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

q\_new[i] = 0;

//Обнуление матрицы

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

dia\_0[i] = 0;

dia\_p1[i] = 0;

dia\_m1[i] = 0;

}

dia\_0[nodes\_n-1] = 0;

matrix\_gen\_si(q\_temp, q\_new);

//Вычисление невязки: A(q)\*q - b(q)

q\_min\_R[0] = dia\_0[0]\*q\_temp[0] + dia\_p1[0]\*q\_temp[1];

q\_min\_R[elements\_n] = dia\_0[elements\_n]\*q\_temp[elements\_n] + dia\_m1[elements\_n-1]\*q\_temp[elements\_n-1];

for(int i = 1; i < elements\_n; i++)

q\_min\_R[i] = dia\_0[i]\*q\_temp[i] + dia\_p1[i]\*q\_temp[i+1] + dia\_m1[i-1]\*q\_temp[i-1];

double norm1 = norm\_q\_dif(q\_min\_R, q\_new);

double norm2 = norm\_q(q\_new);

double nev = norm1/norm2;

if(nev < eps)

end\_cycle = true;

//Переброс указателей

double \*ch\_p;

ch\_p = q\_new;

q\_new = q\_temp;

q\_temp = ch\_p;

printf("SI\tTime:\t%.15lf\tIter:\t%d\tNev:\t%.3e\tRelax:\t%.3e\r",t\_new, iter, nev, w);

}

printf("\n");

//Когда досчитали опять меняем, только уже переходим на новый слой

double\* ch\_pt;

ch\_pt = q\_new;

q\_new = q\_last;

q\_last = ch\_pt;

t\_now = t\_new;

h\_t \*= k\_t;

}

void unlin\_FEM::Newton\_step(){

//В качестве начального приближения берём решение на предыдушем слое

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

q\_new[i] = q\_last[i];

t\_new = t\_now + h\_t; //новое время

//Общий цикл, по итерациям

bool end\_cycle = false;

for(int iter = 0; iter < iter\_max && !end\_cycle; iter++){

//Обнуление матрицы

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

dia\_0[i] = 0;

dia\_p1[i] = 0;

dia\_m1[i] = 0;

q\_temp[i] = 0;

}

dia\_0[nodes\_n-1] = 0;

q\_temp[nodes\_n-1] = 0;

//Генерация матрицы

matrix\_gen\_Newton(q\_new, q\_temp);

//Решаем слау

solve\_LU(dia\_0, dia\_p1, dia\_m1, q\_temp, nodes\_n);

//Релаксация

double w = w\_min();

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

q\_temp[i] = w\*q\_temp[i] + (1-w)\*q\_new[i];

//Проверка - нужно ли выходит из цикла

//В q\_min\_R запишим вектор невязки, в q\_new - вектор праовй части

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

q\_new[i] = 0;

//Обнуление матрицы

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

dia\_0[i] = 0;

dia\_p1[i] = 0;

dia\_m1[i] = 0;

}

dia\_0[nodes\_n-1] = 0;

matrix\_gen\_si(q\_temp, q\_new);

//Вычисление невязки: A(q)\*q - b(q)

q\_min\_R[0] = dia\_0[0]\*q\_temp[0] + dia\_p1[0]\*q\_temp[1];

q\_min\_R[elements\_n] = dia\_0[elements\_n]\*q\_temp[elements\_n] + dia\_m1[elements\_n-1]\*q\_temp[elements\_n-1];

for(int i = 1; i < elements\_n; i++)

q\_min\_R[i] = dia\_0[i]\*q\_temp[i] + dia\_p1[i]\*q\_temp[i+1] + dia\_m1[i-1]\*q\_temp[i-1];

double norm1 = norm\_q\_dif(q\_min\_R, q\_new);

double norm2 = norm\_q(q\_new);

double nev = norm1/norm2;

if(nev < eps)

end\_cycle = true;

//Переброс указателей

double \*ch\_p;

ch\_p = q\_new;

q\_new = q\_temp;

q\_temp = ch\_p;

printf("Ne\tTime:\t%.15lf\tIter:\t%d\tNev:\t%.3e\tRelax:\t%.3e\r",t\_new, iter, nev, w);

}

printf("\n");

//Когда досчитали опять меняем, только уже переходим на новый слой

double\* ch\_pt;

ch\_pt = q\_new;

q\_new = q\_last;

q\_last = ch\_pt;

t\_now = t\_new;

h\_t \*= k\_t;

}

void unlin\_FEM::matrix\_gen\_si(double\* q\_n, double\* q\_t){

double A\_loc[2][2], b\_loc[2]; //локальная матрица и вектор правой части

//Генерация матрицы

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

double x1 = nodes[elements[i].node1]; //левый узел для элемента

double x2 = nodes[elements[i].node2]; //правый узел для элемента

double h\_k = x2 - x1; //шаг по сетки

double x\_midd = (x2+x1)/2; //центральная точка сетки

double G\_11\_val, M\_12\_val; //значения элементов в матирце жескости и массы(чтобы кучу раз не считать одно и тоже)

G\_11\_val = (lambda(x1, t\_new) + lambda(x2, t\_new))/(2\*h\_k);

double gamma\_aver = sigma((q\_n[i+1] - q\_n[i])/h\_k, x\_midd)/h\_t;

M\_12\_val = gamma\_aver \* h\_k / 6;

A\_loc[0][0] = A\_loc[1][1] = G\_11\_val + 2 \* M\_12\_val;

A\_loc[1][0] = A\_loc[0][1] = -G\_11\_val + M\_12\_val;

double f1 = f(x1, t\_new), f2 = f(x2, t\_new); //значения правой части

b\_loc[0] = h\_k\*(2\*f1 + f2)/6.0 + M\_12\_val \* (2\*q\_last[i] + q\_last[i+1]);

b\_loc[1] = h\_k\*(f1 + 2\*f2)/6.0 + M\_12\_val \* (q\_last[i] + 2\*q\_last[i+1]);

dia\_0[i] += A\_loc[0][0];

dia\_0[i+1] += A\_loc[1][1];

dia\_p1[i] += A\_loc[0][1];

dia\_m1[i] += A\_loc[1][0];

q\_t[i] += b\_loc[0];

q\_t[i+1] += b\_loc[1];

}

//Учёт краевых условий

//Левых

switch(bound\_type\_left){

case 1:{

dia\_0[0] = 1;

dia\_p1[0] = 0;

q\_t[0] = bounds\_l[0](t\_new);

}break;

case 2:{

q\_t[0] += bounds\_l[0](t\_new);

}break;

case 3:{

dia\_0[0] += bounds\_l[0](t\_now);

q\_t[0] += bounds\_l[0](t\_new) \* bounds\_l[1](t\_new);

}break;

};

//Правых

switch(bound\_type\_right){

case 1:{

dia\_0[nodes\_n-1] = 1;

dia\_m1[nodes\_n-2] = 0;

q\_t[nodes\_n-1] = bounds\_r[0](t\_new);

}break;

case 2:{

q\_t[nodes\_n-1] += bounds\_r[0](t\_new);

}break;

case 3:{

dia\_0[nodes\_n-1] += bounds\_r[0](t\_new);

q\_t[nodes\_n-1] += bounds\_r[0](t\_new) \* bounds\_r[1](t\_new);

}break;

};

}

void unlin\_FEM::matrix\_gen\_Newton(double\* q\_n, double\* q\_t){

double A\_loc[2][2], b\_loc[2]; //локальная матрица и вектор правой части

//Генерация матрицы

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

double x1 = nodes[elements[i].node1]; //левый узел для элемента

double x2 = nodes[elements[i].node2]; //правый узел для элемента

double h\_k = x2 - x1; //шаг по сетки

double x\_midd = (x2+x1)/2; //центральная точка сетки

double G\_11\_val, M\_12\_val; //значения элементов в матирце жескости и массы(чтобы кучу раз не считать одно и тоже)

G\_11\_val = (lambda(x1, t\_new) + lambda(x2, t\_new))/(2\*h\_k);

double gamma\_aver = sigma((q\_n[i+1] - q\_n[i])/h\_k, x\_midd)/h\_t;

M\_12\_val = gamma\_aver \* h\_k / 6;

//Среднее значние производной

double sigm\_ux\_aver = sigma\_ux((q\_n[i+1] - q\_n[i])/h\_k, x\_midd);

//Базовые значения

A\_loc[0][0] = A\_loc[1][1] = G\_11\_val + 2 \* M\_12\_val;

A\_loc[1][0] = A\_loc[0][1] = -G\_11\_val + M\_12\_val;

double reg\_par = 1.0;

//Прибавки от метода Ньютона

A\_loc[0][0] += reg\_par \* sigm\_ux\_aver \* (-q\_n[i]/3.0 - q\_n[i+1]/6.0 + (2\*q\_last[i] + q\_last[i+1])/6.0) / h\_t;

A\_loc[0][1] += reg\_par \* sigm\_ux\_aver \* (q\_n[i]/3.0 + q\_n[i+1]/6.0 - (2\*q\_last[i] + q\_last[i+1])/6.0) / h\_t;

A\_loc[1][0] += reg\_par \* sigm\_ux\_aver \* (-q\_n[i]/6.0 - q\_n[i+1]/3.0 + (q\_last[i] + 2\*q\_last[i+1])/6.0) / h\_t;

A\_loc[1][1] += reg\_par \* sigm\_ux\_aver \* (q\_n[i]/6.0 + q\_n[i+1]/3.0 - (q\_last[i] + 2\*q\_last[i+1])/6.0) / h\_t;

double f1 = f(x1, t\_new), f2 = f(x2, t\_new); //значения правой части

//Базовые значения

b\_loc[0] = h\_k\*(2\*f1 + f2)/6.0 + M\_12\_val \* (2\*q\_last[i] + q\_last[i+1]);

b\_loc[1] = h\_k\*(f1 + 2\*f2)/6.0 + M\_12\_val \* (q\_last[i] + 2\*q\_last[i+1]);

//Прибавки от метода Ньютона

b\_loc[0] += reg\_par \* sigm\_ux\_aver \* (-q\_n[i]\*q\_n[i]/3.0 + q\_n[i]\*q\_n[i+1]/6.0 + q\_n[i+1]\*q\_n[i+1]/6.0 + (2\*q\_last[i] + q\_last[i+1])\*(q\_n[i]-q\_n[i+1])/6.0) / h\_t;

b\_loc[1] += reg\_par \* sigm\_ux\_aver \* (-q\_n[i]\*q\_n[i]/6.0 - q\_n[i]\*q\_n[i+1]/6.0 + q\_n[i+1]\*q\_n[i+1]/3.0 + (q\_last[i] + 2\*q\_last[i+1])\*(q\_n[i]-q\_n[i+1])/6.0) / h\_t;

dia\_0[i] += A\_loc[0][0];

dia\_0[i+1] += A\_loc[1][1];

dia\_p1[i] += A\_loc[0][1];

dia\_m1[i] += A\_loc[1][0];

q\_t[i] += b\_loc[0];

q\_t[i+1] += b\_loc[1];

}

//Учёт краевых условий

//Левых

switch(bound\_type\_left){

case 1:{

dia\_0[0] = 1;

dia\_p1[0] = 0;

q\_t[0] = bounds\_l[0](t\_new);

}break;

case 2:{

q\_t[0] += bounds\_l[0](t\_new);

}break;

case 3:{

dia\_0[0] += bounds\_l[0](t\_now);

q\_t[0] += bounds\_l[0](t\_new) \* bounds\_l[1](t\_new);

}break;

};

//Правых

switch(bound\_type\_right){

case 1:{

dia\_0[nodes\_n-1] = 1;

dia\_m1[nodes\_n-2] = 0;

q\_t[nodes\_n-1] = bounds\_r[0](t\_new);

}break;

case 2:{

q\_t[nodes\_n-1] += bounds\_r[0](t\_new);

}break;

case 3:{

dia\_0[nodes\_n-1] += bounds\_r[0](t\_new);

q\_t[nodes\_n-1] += bounds\_r[0](t\_new) \* bounds\_r[1](t\_new);

}break;

};

}

double unlin\_FEM::norm\_q\_dif(double \*q1, double \*q2){

double norm = 0;

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

norm += (q1[i] - q2[i])\*(q1[i] - q2[i]);

norm = sqrt(norm);

return norm;

}

double unlin\_FEM::norm\_q(double \*q1){

double norm = 0;

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

norm += q1[i]\*q1[i];

norm = sqrt(norm);

return norm;

}

void unlin\_FEM::out\_q(string file\_out){

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

//fprintf(out\_f, "============================= Time: %.15lf ===================\n", t\_now);

double diff = 0;

double u\_norm = 0;

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

double u\_val = analytic(nodes[i], t\_now);

//fprintf(out\_f,"%d\t%.15lf\t%.15lf\n", i, q\_last[i], u\_val);

diff += (q\_last[i]-u\_val)\*(q\_last[i]-u\_val);

u\_norm += u\_val\*u\_val;

}

diff = sqrt(diff/u\_norm);

fprintf(out\_f, "Diff:\t%.3e\n", diff);

fclose(out\_f);

}

double unlin\_FEM::w\_min\_func(double w){

//Обнуление матрицы и вычисление аргумента

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

dia\_0[i] = 0;

dia\_p1[i] = 0;

dia\_m1[i] = 0;

q\_min[i] = w\*q\_temp[i] + (1-w)\*q\_new[i];

q\_min\_rp[i] = 0;

}

dia\_0[nodes\_n-1] = 0;

q\_min[nodes\_n-1] = w\*q\_temp[nodes\_n-1] + (1-w)\*q\_new[nodes\_n-1];

q\_min\_rp[nodes\_n-1] = 0;

matrix\_gen\_si(q\_min, q\_min\_rp);

//Вычисление невязки: A(q\_min)\*q\_min - b(q\_min)

q\_min\_R[0] = dia\_0[0]\*q\_min[0] + dia\_p1[0]\*q\_min[1] - q\_min\_rp[0];

q\_min\_R[elements\_n] = dia\_0[elements\_n]\*q\_min[elements\_n] + dia\_m1[elements\_n-1]\*q\_min[elements\_n-1] - q\_min\_rp[elements\_n];

for(int i = 1; i < elements\_n; i++)

q\_min\_R[i] = dia\_0[i]\*q\_min[i] + dia\_p1[i]\*q\_min[i+1]+dia\_m1[i-1]\*q\_min[i-1] - q\_min\_rp[i];

return norm\_q(q\_min\_R);

}

double unlin\_FEM::w\_min(){

double a, b; //Отрезок минимизации

w\_min\_find\_area(a, b, 1E-3); // Ищим отрезок

return w\_min\_golden(a, b, 1E-5);

}

void unlin\_FEM::w\_min\_find\_area(double &a, double &b, double delta){

//Инициализация

double x0 = 1;

double f0 = w\_min\_func(x0);

double x1 = x0-delta;

double f1 = w\_min\_func(x1);

double h = -delta;

int k = 1;

//Определяем сторону, куда идти

if(f0 < f1){

x1 = x0 + delta;

h \*= -1;

}

bool end\_cycle = false;

while(!end\_cycle){

h \*= 2;

x0 = x1 + h;

f0 = w\_min\_func(x0);

k++;

if(f1 > f0){

x1 = x0;

f1 = f0;

}

else{

end\_cycle = true;

x1 = x0;

x0 -= h + h/2;

}

};

if(x0 > x1){

a = x1;

b = x0;

}

else{

a = x0;

b = x1;

}

}

double unlin\_FEM::w\_min\_golden(double a, double b, double eps\_min){

double x1, x2, f1, f2;

const double mul = (3-sqrt((double)5.0))/2;

int point\_num;

x1 = a + mul\*(b-a);

x2 = b - mul\*(b-a);

f1 = w\_min\_func(x1);

f2 = w\_min\_func(x2);

if(f1 < f2){

b = x2;

x2 = x1;

f2 = f1;

point\_num = 1;

}

else{

a = x1;

x1 = x2;

f1 = f2;

point\_num = 2;

}

while(fabs(a-b)>eps\_min){

switch(point\_num){

case 1:{

x1 = a + mul\*(b-a);

f1 = w\_min\_func(x1);

}break;

case 2:{

x2 = b - mul\*(b-a);

f2 = w\_min\_func(x2);

}break;

};

if(f1 < f2){

b = x2;

x2 = x1;

f2 = f1;

point\_num = 1;

}

else{

a = x1;

x1 = x2;

f1 = f2;

point\_num = 2;

}

}

return (a+b)/2;

}

## Файл «launch.cpp»

#include <windows.h>

#include "grid\_gen.h"

#include "unlin\_FEM.h"

double lam1(double x, double t){return x;}

double f1(double x, double t){return -1;}

double b\_l1(double t){return 0;}

double b\_r1(double t){return 1;}

double u1(double x, double t){return x;}

double lam2(double x, double t){return x;}

double f2(double x, double t){return x\*x\*t\*t - t;}

double b\_l2(double t){return 0;}

double b\_r2(double t){return t;}

double u2(double x, double t){return x\*t;}

double lam3(double x, double t){return 1;}

double f3(double x, double t){return 0;}

double b\_l3(double t){return t;}

double b\_r3(double t){return t;}

double u3(double x, double t){return t;}

double lam4(double x, double t){return x\*t;}

double f4(double x, double t){return exp(-t)\*t\*(x\*sin(x)-cos(x))-x\*exp(-3\*t)\*cos(x)\*cos(x)\*sin(x);}

double b\_l4(double t){return 0;}

double b\_r4(double t){return 1;}

double b\_r42(double t){return exp(-t)\*(-t\*cos((double)1.0) + sin((double)1.0));}

double u4(double x, double t){return exp(-t)\*sin(x);}

int main(){

grid\_gen::generate\_unreg\_grid(0, 1, 1E-1, 1.2, "grid1.txt");

vector<double> start\_gen;

grid\_gen::generate\_unreg\_grid(0, 1, 1E-1, 1.2, start\_gen);

FILE\* out\_f = fopen("u0.txt", "w");

for(int i = 0; i < start\_gen.size(); i++)

fprintf(out\_f, "%.15lf\n", u2(start\_gen[i], 0));

fclose(out\_f);

unlin\_FEM our\_sol\_si, our\_sol\_Newton;

our\_sol\_si.init("grid1.txt", "u0.txt", "pars.txt");

our\_sol\_si.init\_coefs(lam2, f2);

our\_sol\_si.init\_sigma((double)0.0);

our\_sol\_si.init\_bounds(1, 2, b\_l2, NULL, b\_r2, NULL);

our\_sol\_si.set\_analytic(u2);

our\_sol\_Newton.init("grid1.txt", "u0.txt", "pars.txt");

our\_sol\_Newton.init\_coefs(lam2, f2);

our\_sol\_Newton.init\_sigma((double)0.0);

our\_sol\_Newton.init\_bounds(1, 2, b\_l2, NULL, b\_r2, NULL);

our\_sol\_Newton.set\_analytic(u2);

for(int i = 0 ; i < 7; i++){

our\_sol\_si.simple\_iteration\_step();

our\_sol\_Newton.Newton\_step();

our\_sol\_si.out\_q("out\_si.txt");

our\_sol\_Newton.out\_q("out\_Newton.txt");

}

system("pause");

return 0;

}