Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра ПМ

УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Практическое задание № 2

# Решение нелинейных начально-краевых задач

Факультет: ПМИ Преподаватели:

Задорожный А. Г.,

Патрушев И.И.

Группа: ПМ-81

Студенты: Ефремов А.А.,

Ртищева К. С.

Бригада: 1

Вариант: 6

Новосибирск

2021

1. **Цель работы**

Разработать программу решения нелинейной одномерной краевой задачи методом конечных элементов. Провести сравнение метода простой итерации и метода Ньютона для решения данной задачи.

1. **Задание**

Уравнение: Базисные функции – квадратичные. Начальное условие и первые краевые условия.

1. **Анализ задачи**

Параболическое уравнение для одномерной задачи:

Будем аппроксимируем производную искомой функции по времени на каждом временном слое s двухслойной неявной схемой:

В результате конечно элементной аппроксимации нелинейной задачи получается система нелинейных уравнений где

Квадратичные базисные функции:

Получаем матрицу жесткости:

Получаем матрицу массы:

Вектор правой части:

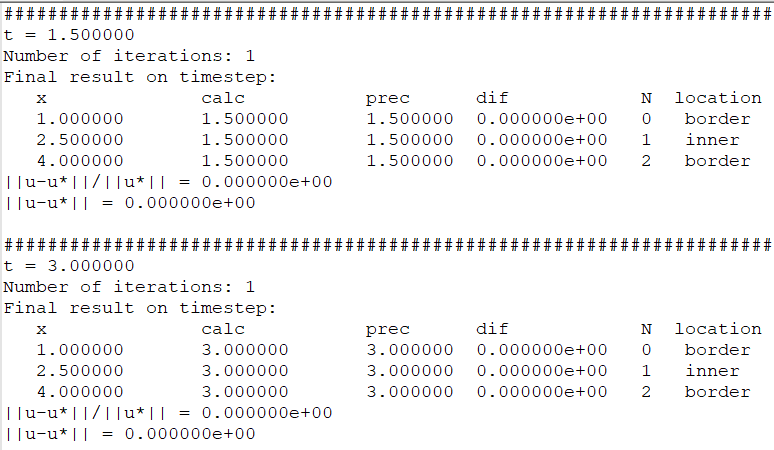
**Учет первых краевых условий:**

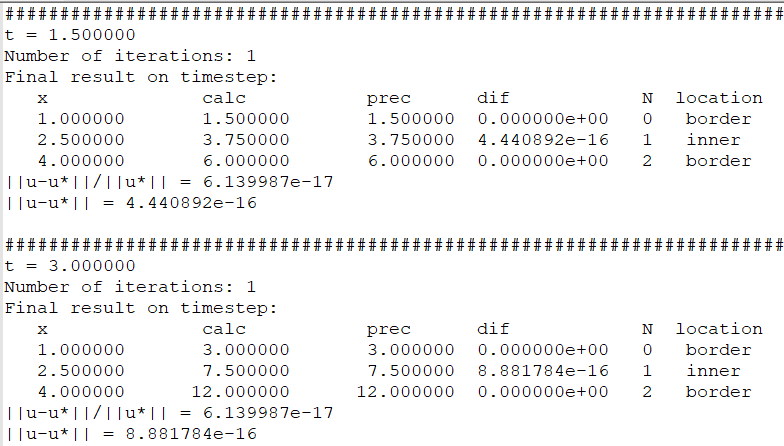
в матрице СЛАУ в строке на место диагонального элемента ставится единица, все остальные элементы этой строки матрицы обнуляются, а компоненте вектора правой части присваивается значение .

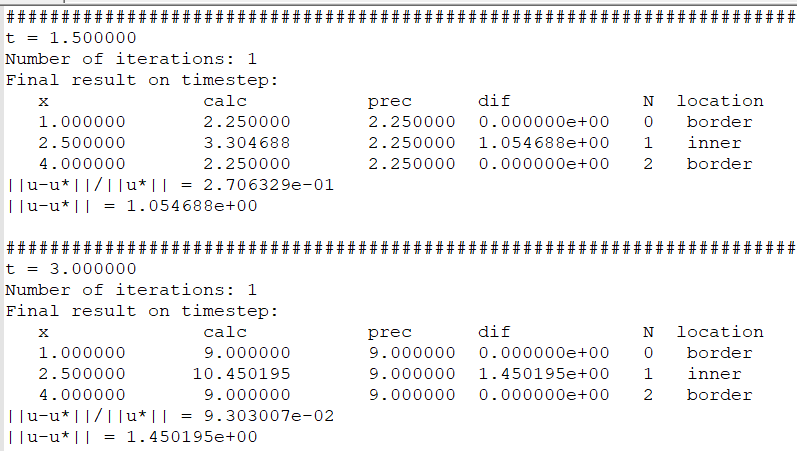
1. **Исследование порядка аппроксимации для линейной задачи на равномерной сетке**

|  |  |  |
| --- | --- | --- |
| 0  1  2.5  4 | ***Файл time.txt***  0 3  1 2 | ***Файл regions.txt***  1  1 4  -1-  0 1  1 2  1 1 |

**Исследование порядка аппроксимации по времени:**

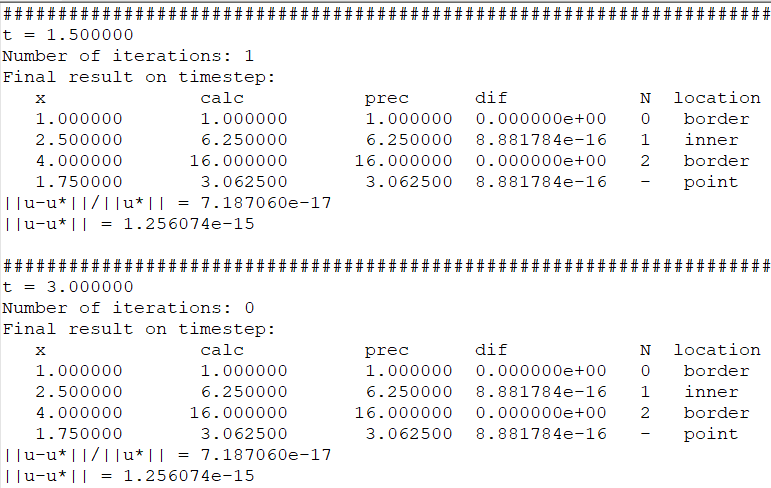


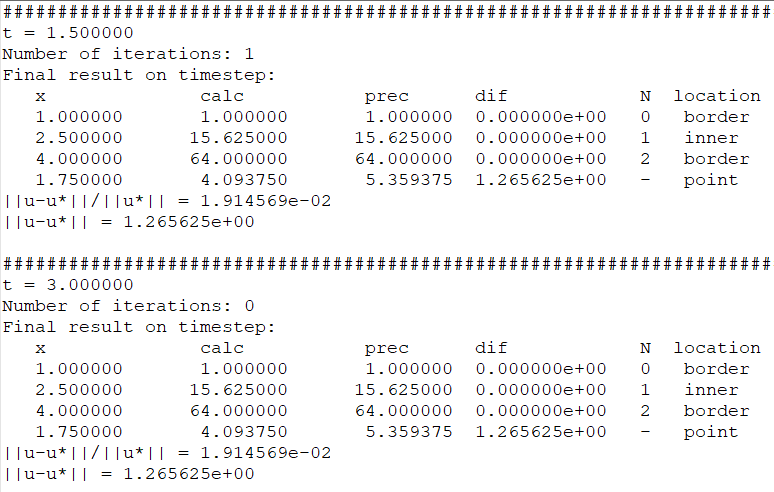


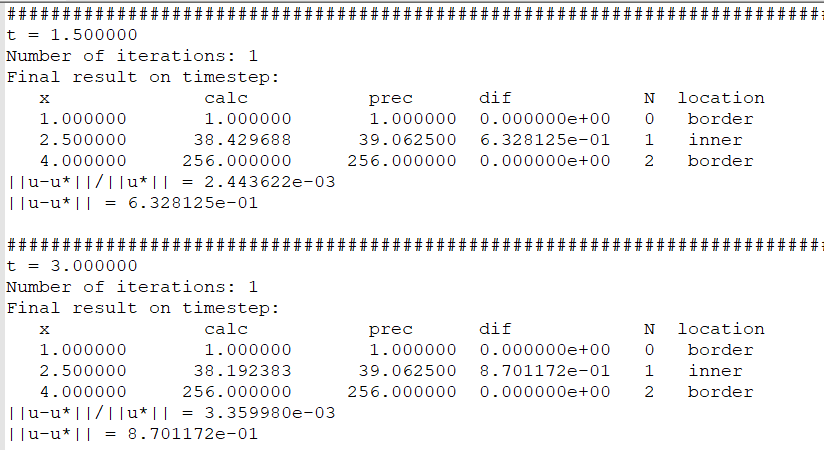


При увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

**Исследование порядка аппроксимации по пространству:**







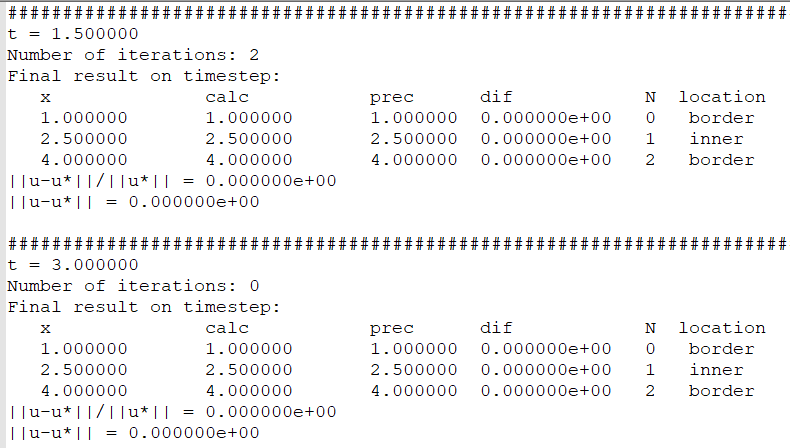
При увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

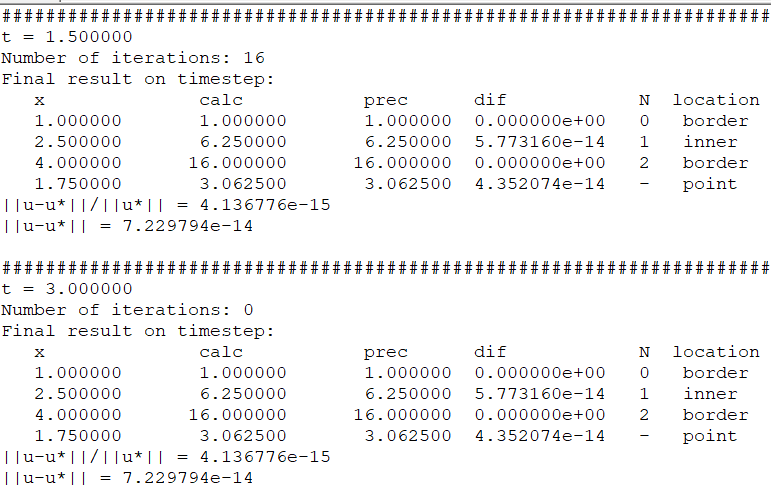
**Вывод: порядок аппроксимации метода для линейной задачи на равномерной сетке по пространству = 2, по времени = 1.**

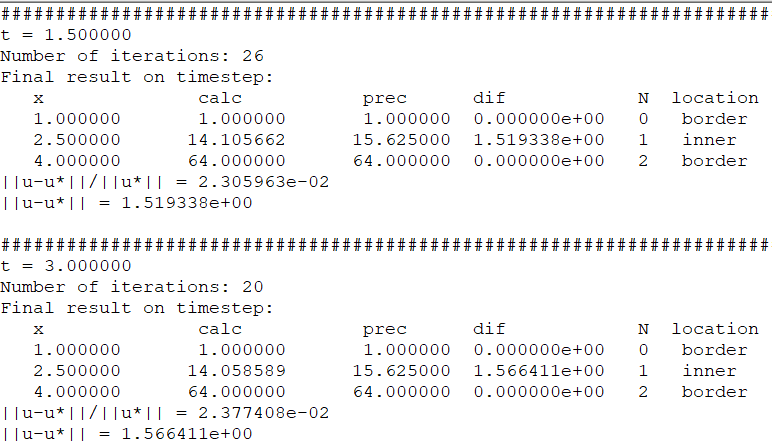
1. **Исследование порядка аппроксимации метода для нелинейной задачи на равномерной сетке**

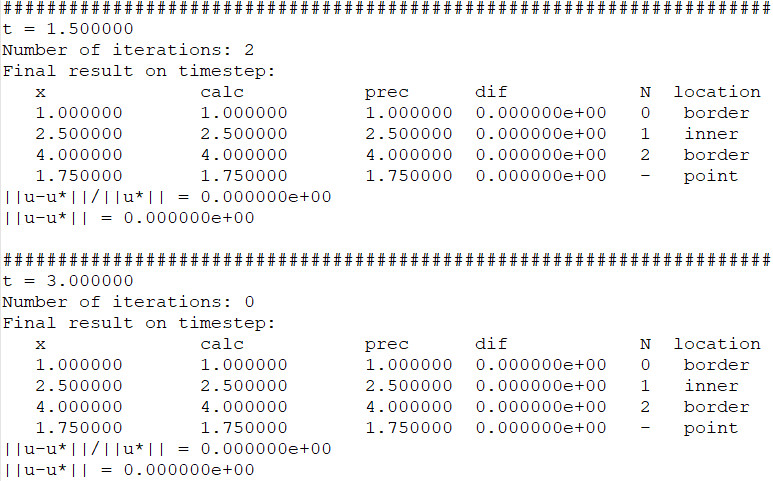
|  |  |  |
| --- | --- | --- |
| 0  1  2.5  4 | ***Файл time.txt***  0 3  1 2 | ***Файл regions.txt***  1  1 4  -1-  0 1  1 2  1 1 |

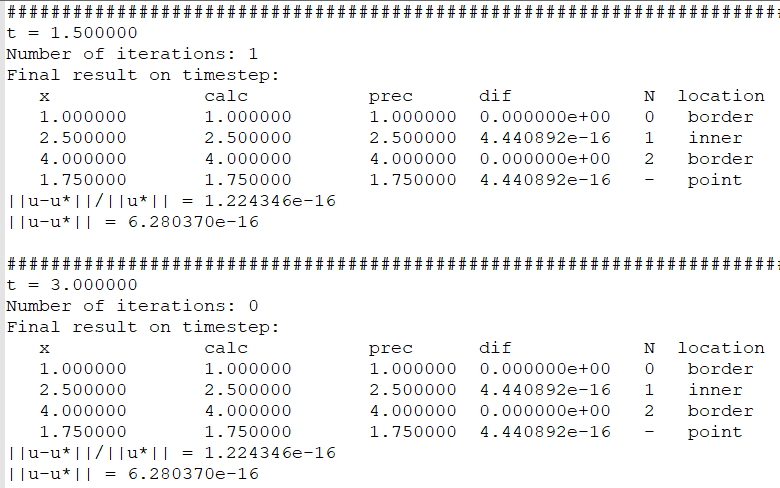
**Исследование порядка аппроксимации по пространству:**

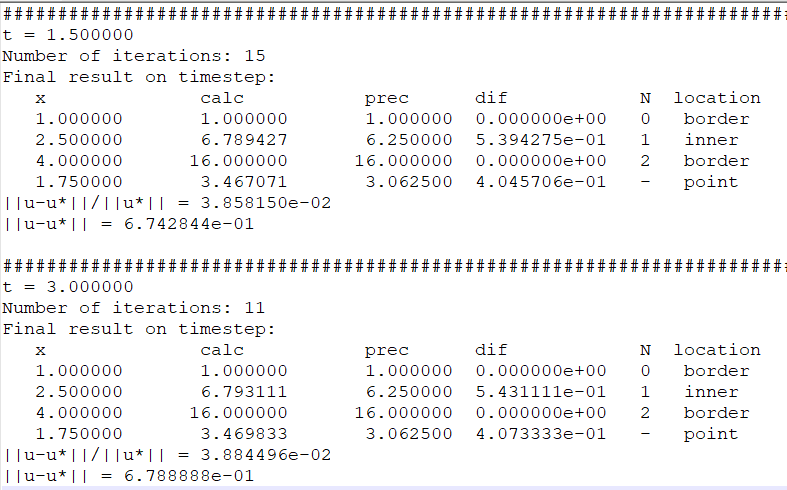








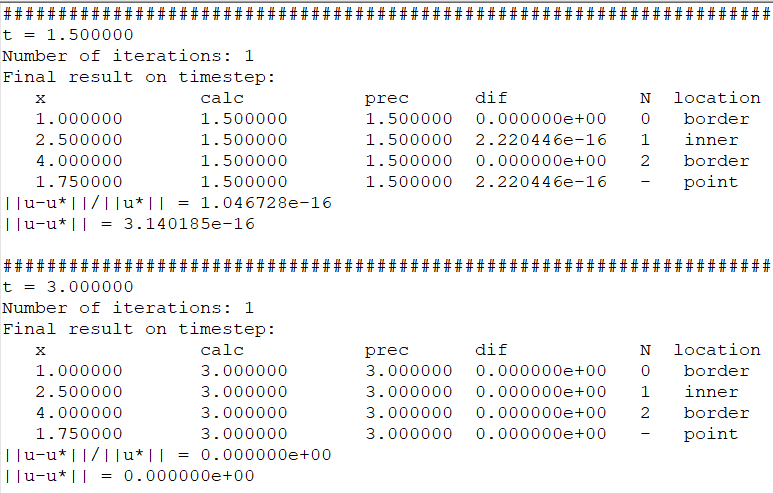


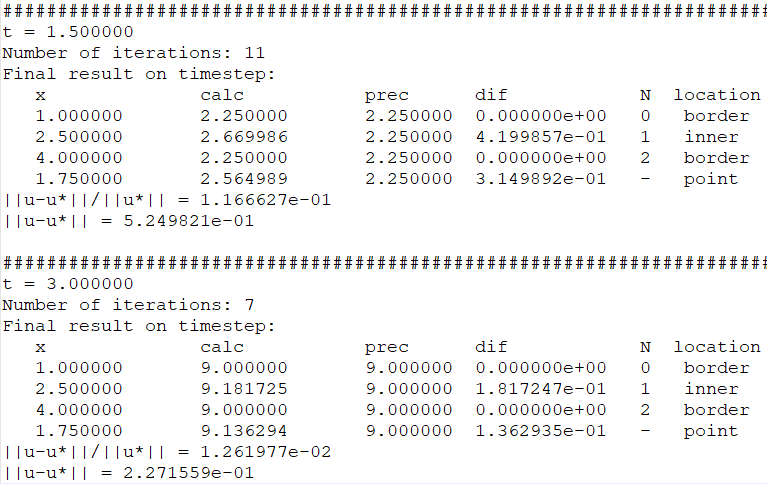


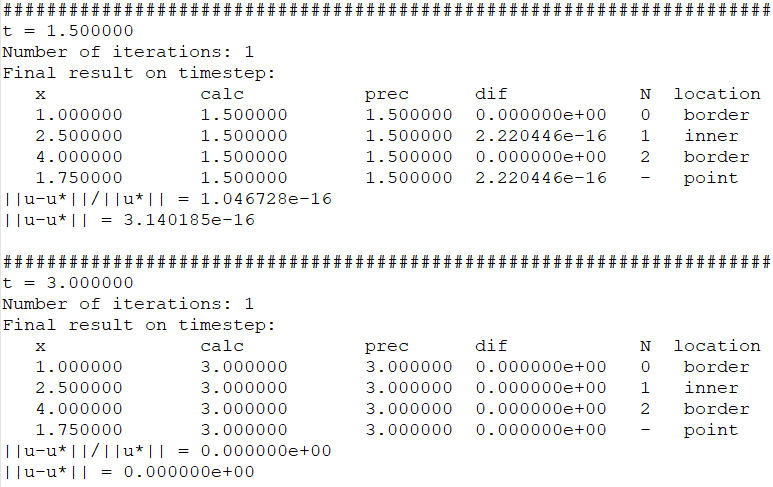
При при увеличении степени искомой функции, начиная с , происходит увеличение погрешности;

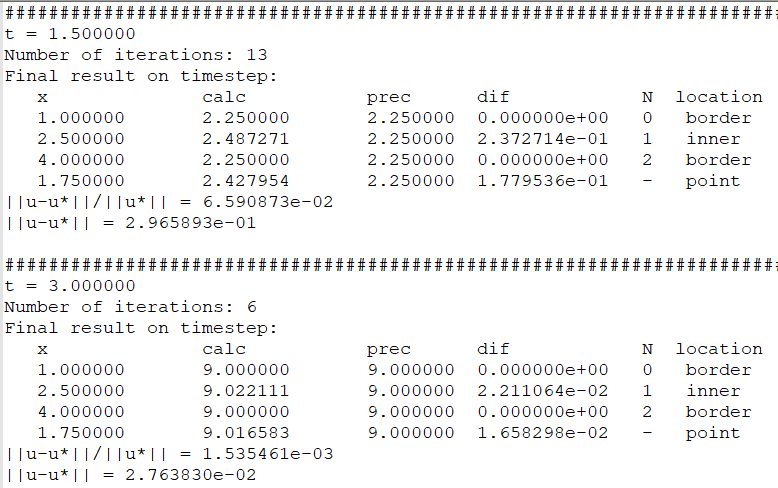
При при увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

**Исследование порядка аппроксимации по времени:**



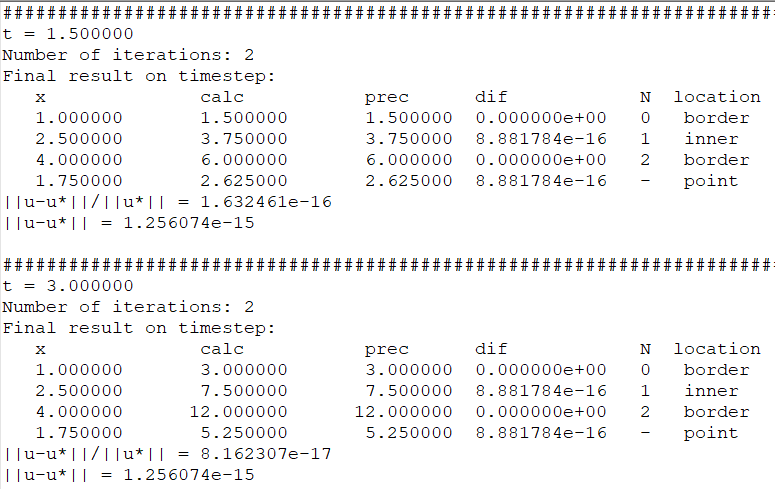


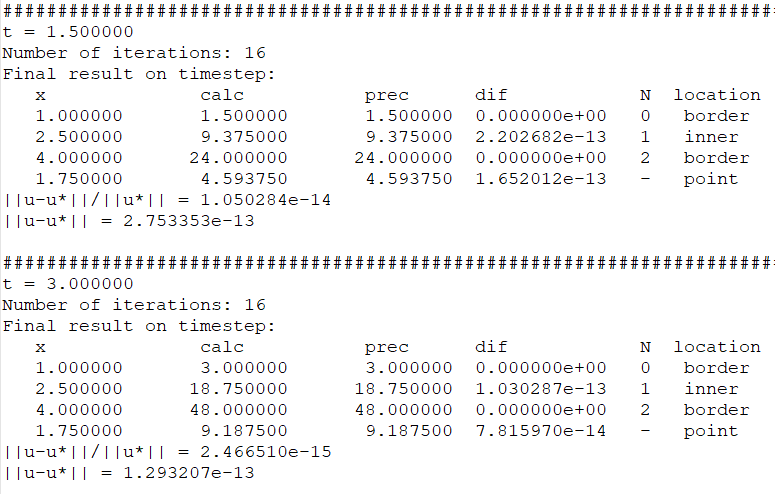


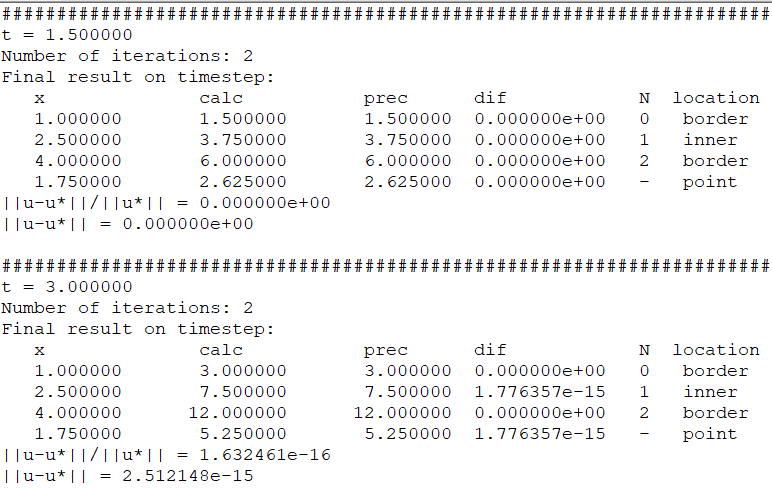


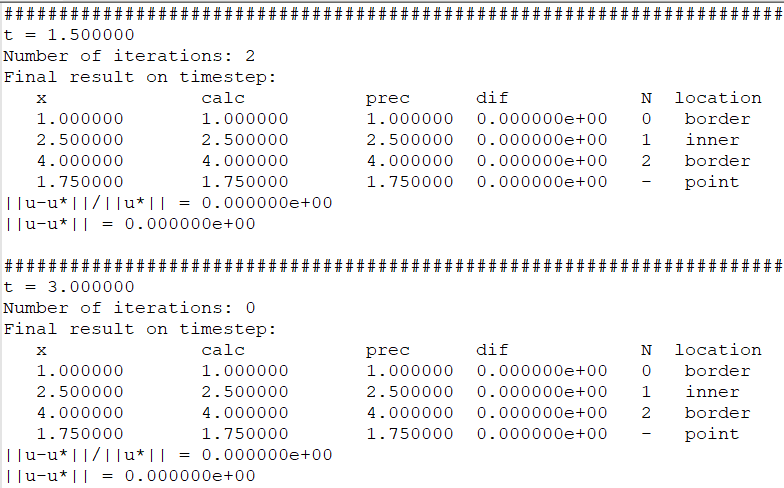
При и при увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

**Проверка:**









При при увеличении степени искомой функции, начиная с , происходит увеличение погрешности;

При при увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

При и при увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

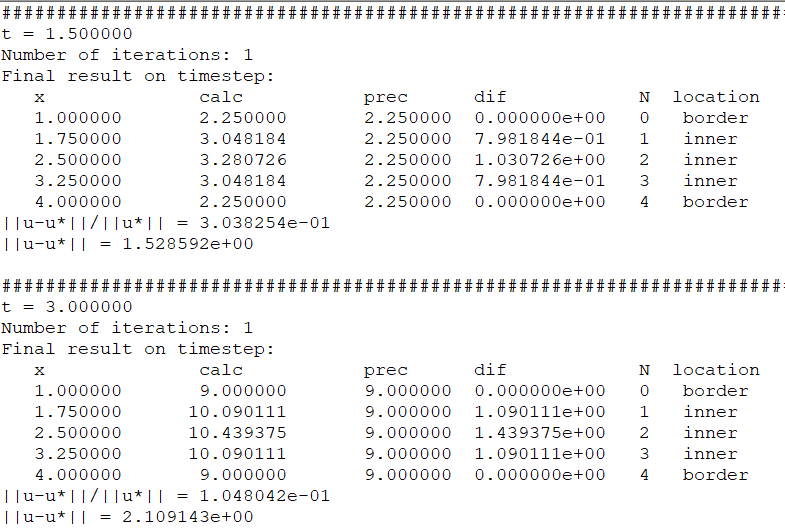
**Вывод: порядок аппроксимации методо для нелинейной задачи на равномерной сетке при линейной зависимости параметра от решения по пространству и по времени совпадает с порядком аппроксимации метода для линейной задачи: по пространству = 2, по времени = 1.**

1. **Исследование порядка сходимости метода для линейной задачи на равномерной сетке**

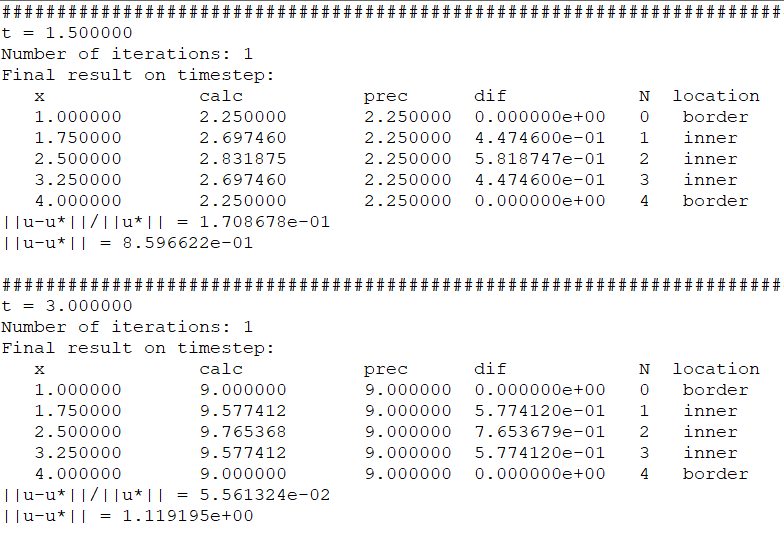
**Исследование порядка сходимости по времени:**

|  |  |  |
| --- | --- | --- |
| 0  1  2.5  4  1.75  3.25 | ***Файл time.txt***  0 3  1 2 | ***Файл regions.txt***  1  1 4  -1-  0 1  1 4  1 1 |

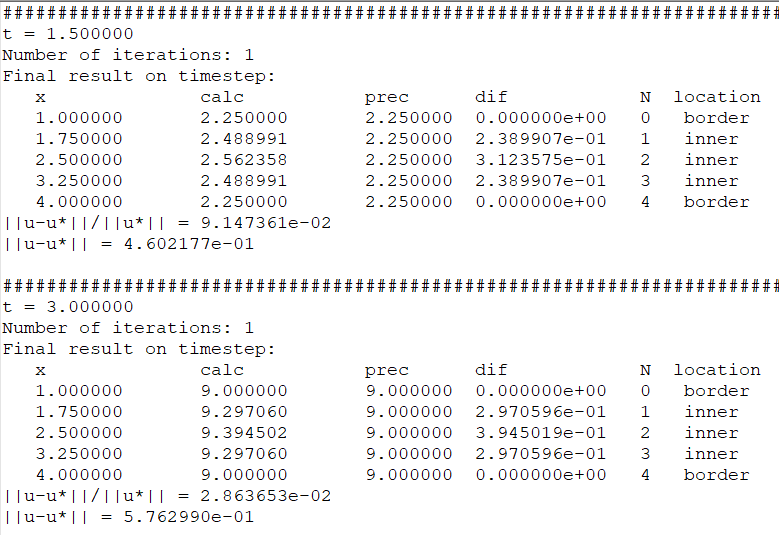
n = 2 (2 временных слоя)



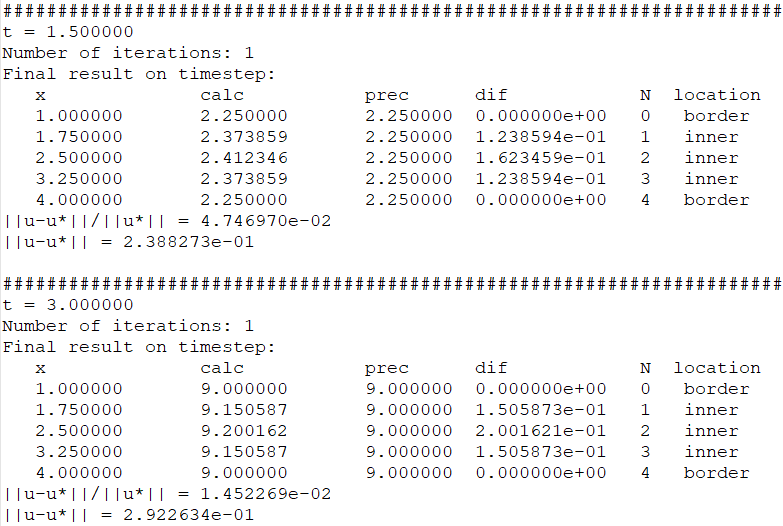
Поделим сетку по t в 2 раза (n = 4)



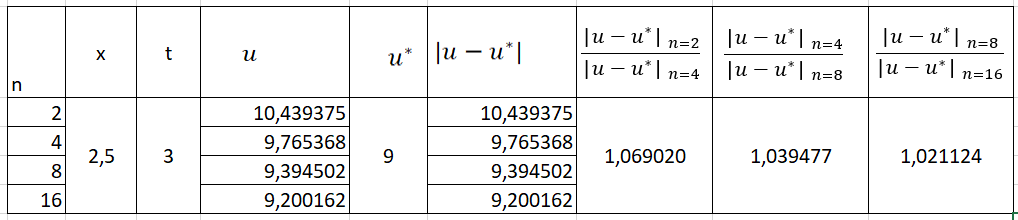
Поделим сетку по t еще в 2 раза (n=8)

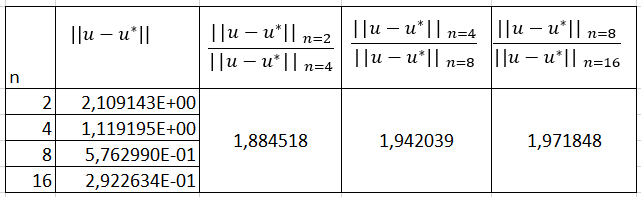


Поделим сетку по t еще в 2 раза (n=16)



**Рассмотрим значения численного и аналитического решения**

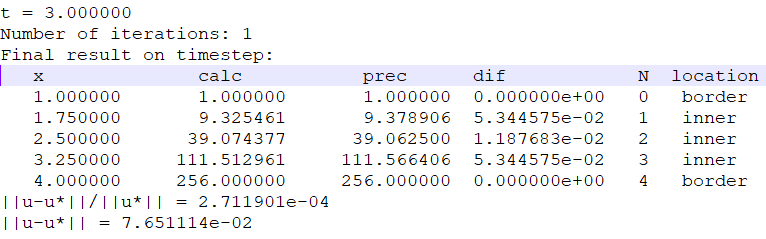


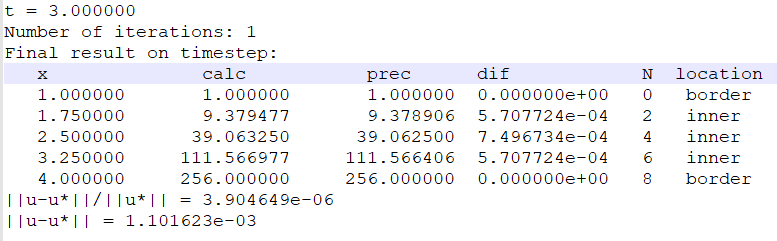


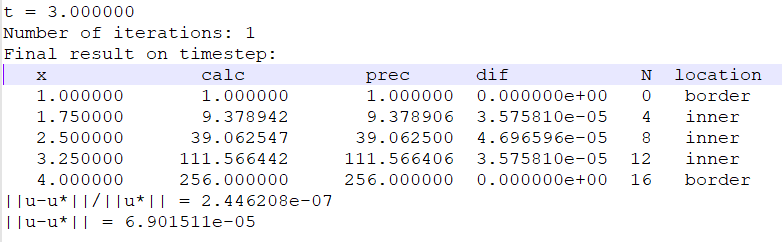
Таким образом, при дроблении сетки по времени в 2 раза погрешность решения падает в 2 раза, следовательно, **порядок сходимости метода по времени для линейной задачи на равномерной сетке равен 1.**

**Исследование порядка сходимости по пространству:**

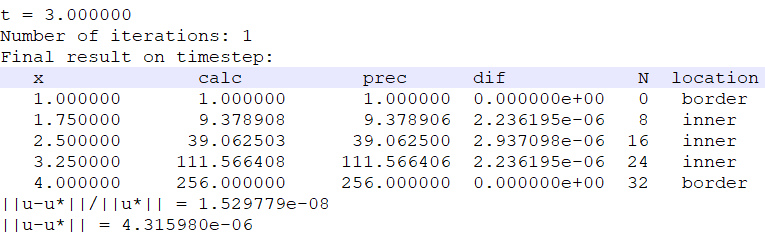
|  |  |  |
| --- | --- | --- |
| 0  1  2.5  4  1.75  3.25 | ***Файл time.txt***  0 3  1 1 | ***Файл regions.txt***  1  1 4  -1-  0 1  1 4  1 1 |

n=2 (2 конечных элемента)

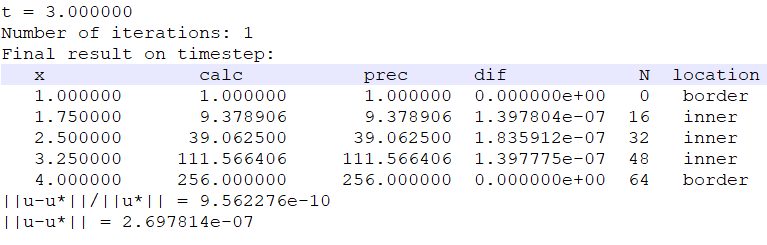
Поделим сетку по x в 2 раза (n=4)

Поделим сетку по x еще в 2 раза (n=8)

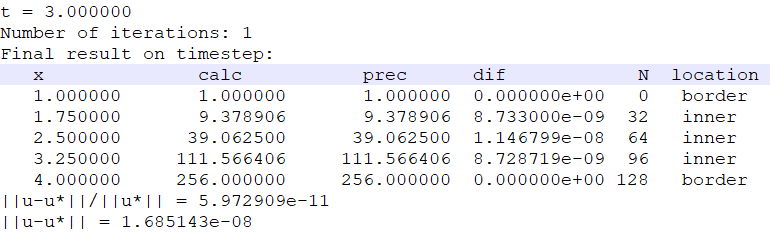
Поделим сетку по x еще в 2 раза (n=16)



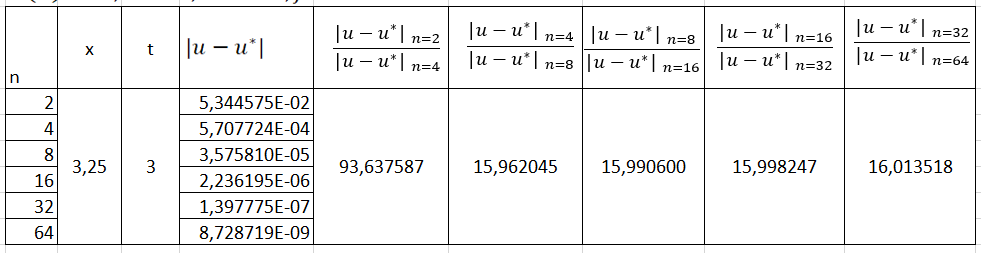
Поделим сетку по x еще в 2 раза (n=32)

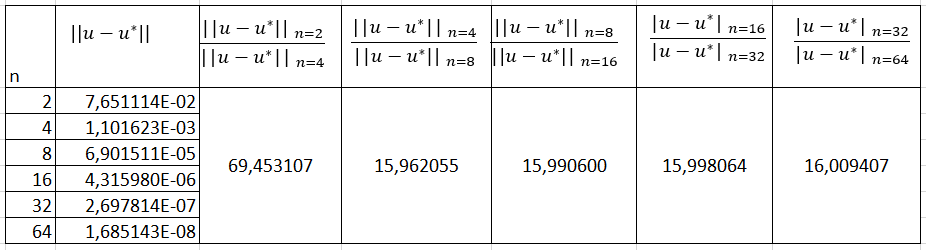


Поделим сетку по x еще в 2 раза (n=64)



**Рассмотрим значения численного и аналитического решения**





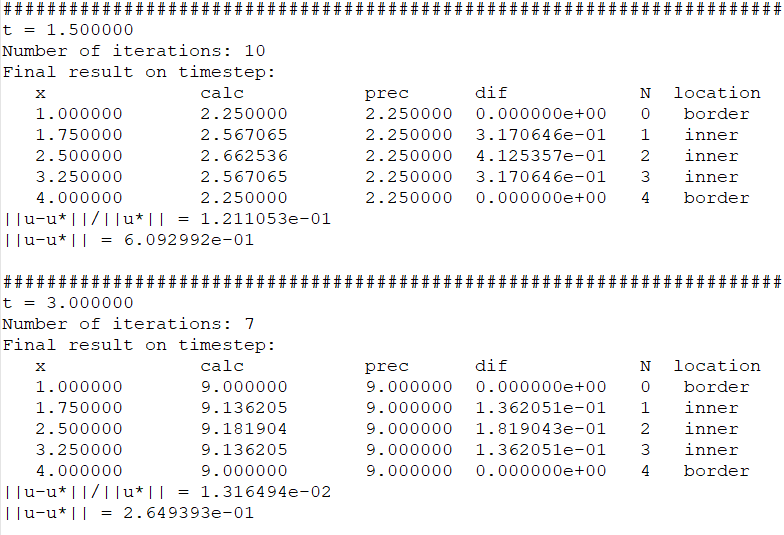
Таким образом, при дроблении сетки по времени в 2 раза погрешность решения падает в 16 раз, следовательно, **порядок сходимости метода по пространству для линейной задачи на равномерной сетке равен 4.**

1. **Исследование порядка сходимости метода для нелинейной задачи на равномерной сетке**

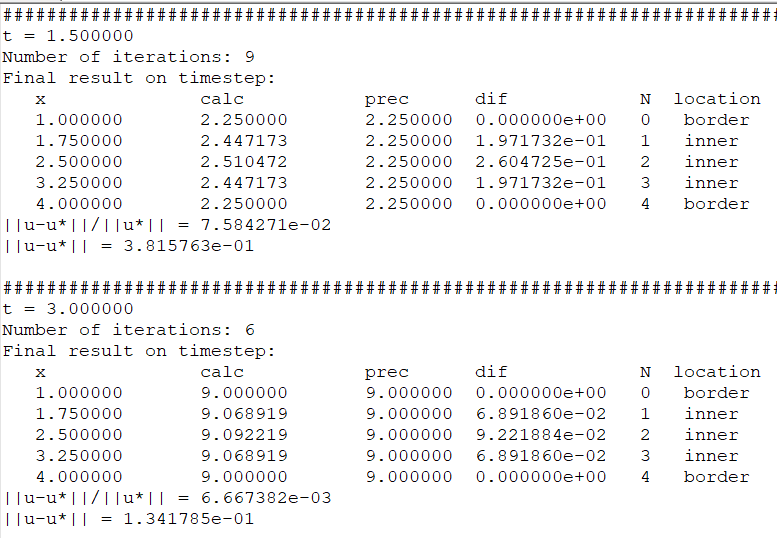
**Исследование порядка сходимости по времени:**

|  |  |  |
| --- | --- | --- |
| 0  1  2.5  4  1.75  3.25 | ***Файл time.txt***  0 3  1 2 | ***Файл regions.txt***  1  1 4  -1-  0 1  1 4  1 1 |

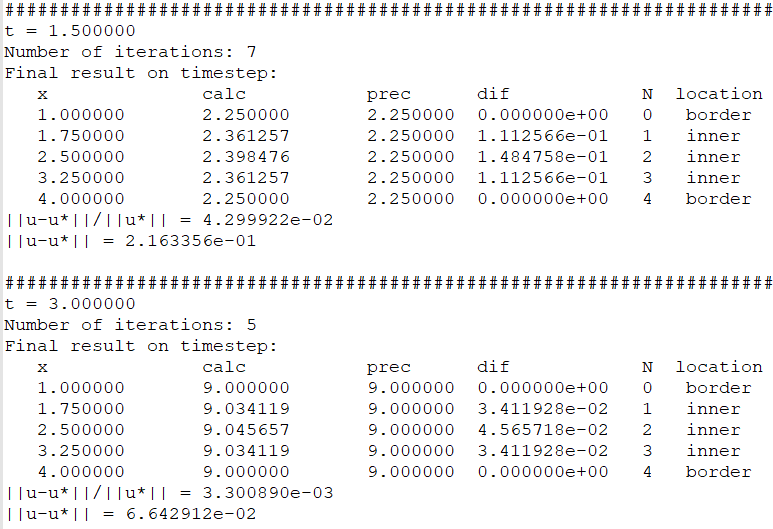
**n = 2 (2 временных слоя)**



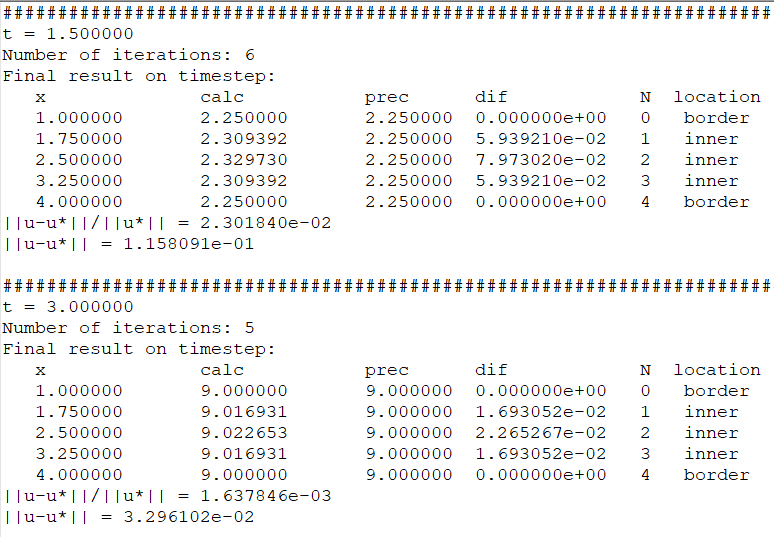
**Поделим сетку в два раза по t (n = 4)**



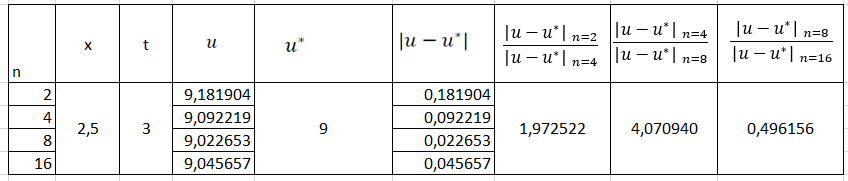
**Поделим сетку еще в два раза по t (n = 8)**

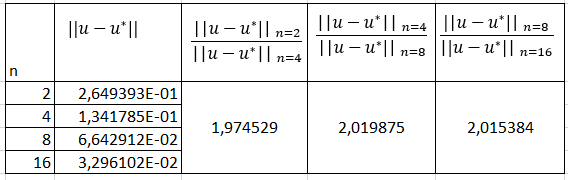


**Поделим сетку еще в два раза по t (n = 16)**



**Рассмотрим значения численного и аналитического решения**



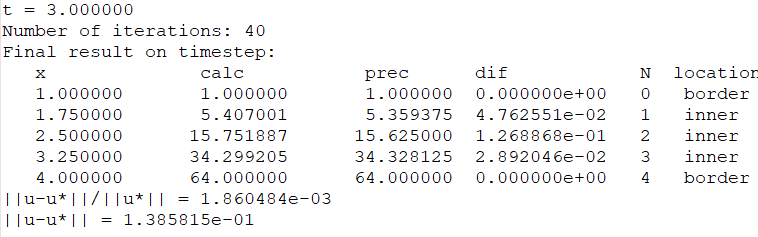


Таким образом, при дроблении сетки по времени в 2 раза погрешность решения падает в 2 раза, следовательно, **порядок сходимости метода по времени для нелинейной задачи на равномерной сетке равен 1.**

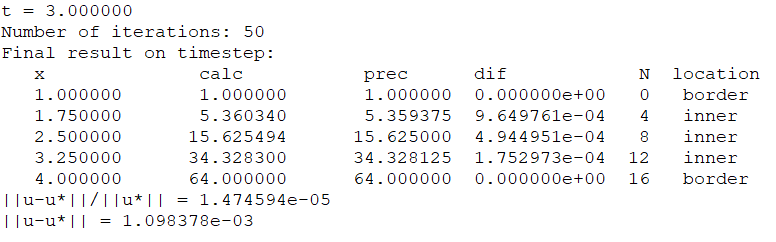
**Исследование порядка сходимости по пространству:**

|  |  |  |
| --- | --- | --- |
| 0  1  2.5  4  1.75  3.25 | ***Файл time.txt***  0 3  1 1 | ***Файл regions.txt***  1  1 4  -1-  0 1  1 4  1 1 |

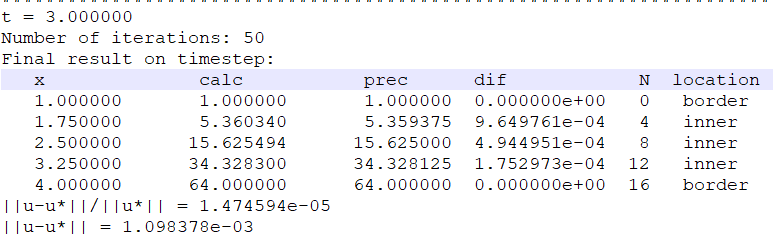
**n = 2 (2 конечных элемента)**



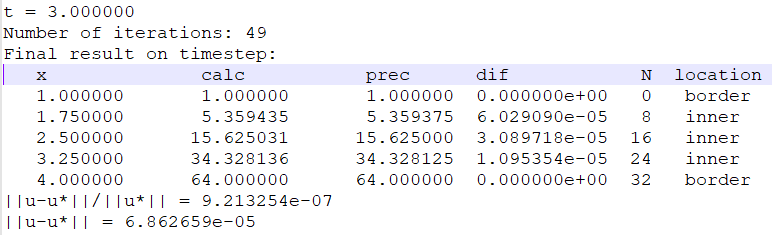
**Поделим сетку еще в два раза по x (n = 4)**



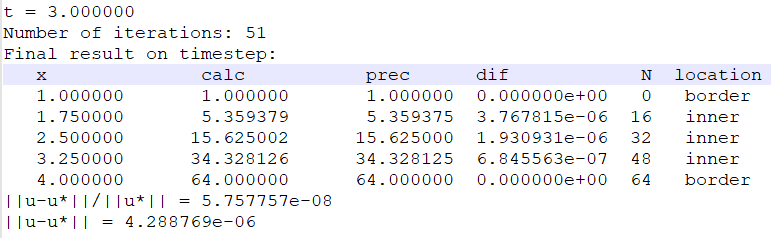
**Поделим сетку еще в два раза по x (n = 8)**



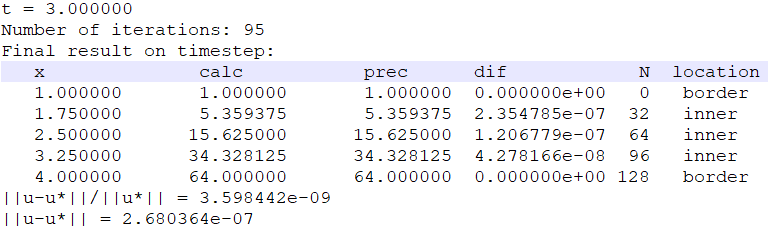
**Поделим сетку еще в два раза по x (n = 16)**



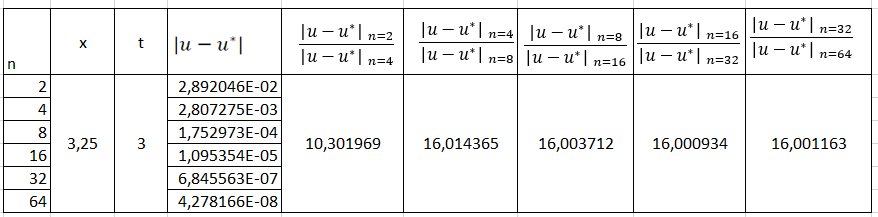
**Поделим сетку еще в два раза по x (n = 32)**

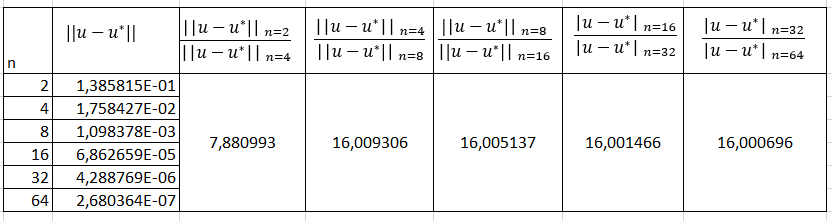


**Поделим сетку еще в два раза по x (n = 64)**



**Рассмотрим значения численного и аналитического решения**





Таким образом, при дроблении сетки по пространству в 2 раза погрешность решения падает в 16 раз, следовательно, **порядок сходимости метода по пространству для нелинейной задачи на равномерной сетке равен 4.**

1. **Выводы**
2. **Исследование на порядок аппроксимации**

Порядок аппроксимации метода для линейной задачи на равномерной сетке по пространству = 2, по времени = 1.

Порядок аппроксимации метода для нелинейной задачи на равномерной сетке при линейной зависимости параметра от решения по пространству и по времени совпадает с порядком аппроксимации метода для линейной задачи.

1. **Исследование на порядок сходимости**

Порядок сходимости метода по времени для линейной и нелинейной задач на равномерной сетке равен 1.

Порядок сходимости метода по пространству для линейной и нелинейной задач на равномерной сетке равен 4.

1. **Текст программы**

**Файл “Matrix.h”**

#pragma once

#include <vector>

using namespace std;

// Класс матрицы в профильном формате

class Matrix

{

public:

vector<double> bot\_tr; // Нижний треугольник матрицы системы

vector<double> top\_tr; // Верхний треугольник матрицы системы

vector<double> di; // Диагональ матрицы системы

vector<int> ind; // Вектор указателей начала строк

int size; // Размер матрицы

// Умножение матрицы на вектор vec, результат в res

void MatVecMult(const vector<double>& vec, vector<double>& res)

{

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

{

res[i] = vec[i] \* di[i];

int i0 = ind[i + 0], i1 = ind[i + 1];

for(int j = i - (i1 - i0), k = i0; j < i; j++, k++)

{

res[i] += vec[j] \* bot\_tr[k];

res[j] += vec[i] \* top\_tr[k];

}

}

}

};

**Файл “SLAE.h”**

#pragma once

#include <vector>

#include "Matrix.h"

using namespace std;

// Класс решателя СЛАУ методом LU - разложения

class SLAE

{

public:

Matrix m; // Матрица системы

vector<double> b; // Вектор правой части

// Алгоритм прямого прохода

void ForwardSolver()

{

for(int i = 0; i < m.size; i++)

{

int i0 = m.ind[i + 0], i1 = m.ind[i + 1];

double s = 0;

for(int j = i - (i1 - i0), k = i0; j < i; j++, k++)

s += b[j] \* m.bot\_tr[k];

b[i] = (b[i] - s) / m.di[i];

}

}

// Алгоритм обратного прохода

void BackwardSolver()

{

for(int i = m.size - 1; i >= 0; i--)

{

int i0 = m.ind[i + 0], i1 = m.ind[i + 1];

double xi = b[i];

for(int j = i - (i1 - i0), k = i0; j < i; j++, k++)

b[j] -= xi \* m.top\_tr[k];

b[i] = xi;

}

}

// LU - разложение матрицы системы

void LUDecomp()

{

for(int i = 0; i < m.size; i++)

{

int i0 = m.ind[i + 0], i1 = m.ind[i + 1];

double sd = 0;

for(int j = i - (i1 - i0), k = i0; j < i; j++, k++)

{

double sl = 0, su = 0;

int j0 = m.ind[j + 0], j1 = m.ind[j + 1];

int kol\_i = k - i0, kol\_j = j1 - j0;

int kol\_r = kol\_i - kol\_j, ki = i0, kj = j0;

if(kol\_r > 0)

ki += kol\_r;

else

kj -= kol\_r;

for(; ki < k; ki++, kj++)

{

sl += m.bot\_tr[ki] \* m.top\_tr[kj];

su += m.bot\_tr[kj] \* m.top\_tr[ki];

}

m.bot\_tr[k] = m.bot\_tr[k] - sl;

m.top\_tr[k] = m.top\_tr[k] - su;

m.top\_tr[k] /= m.di[j];

sd += m.bot\_tr[k] \* m.top\_tr[k];

}

m.di[i] = m.di[i] - sd;

}

}

};

**Файл “Region.h”**

#pragma once

#include <vector>

using namespace std;

struct Region

{

vector<double> nodes; // Координаты узлов

int nodes\_count; // Количество узлов

int elems\_count; // Количество конечных элементов

int first\_i; // Индекс первого узла в глобальной нумерации

int left\_bord, right\_bord; // Информация о краевых условиях

};

**Файл “NonLinearBVP.h”**

#pragma once

#include <vector>

#include <fstream>

#include <iomanip>

#include "Region.h"

#include "SLAE.h"

#include "Test.h"

using namespace std;

class NonLinearBVP

{

public:

vector<Region> regions; // Регионы расчетной области

vector<double> grid; // Исходная сетка

int regions\_count = 0; // Количество регионов

int nodes\_count = 0; // Общее количество узлов

int elems\_count = 0; // Общее количество конечных элементов

double big\_num = 1E+20; // Большое число для учета первого краевого условия

vector<double> q\_1; // Приближение функции u на предыдущей итерации по времени

vector<double> q; // Приближение функции u на текущей итерации по времени

//vector<double> u\_1; // Приближение функции u по нелинейности на предыдущей итерации

vector<double> u; // Приближение функции u по нелинейности на текущей итерации

Matrix G, M; // Матрицы жесткости и массы

SLAE slae; // СЛАУ

Test test; // Тестовая информация

vector<double> time\_grid; // Сетка по времени

vector<double> vec\_1; // Вспомогательный вектор для метода

// простой итерации

vector<double> vec\_2; // Вспомогательный вектор для явной схемы

// по времени

// Вспомогательная матрица для построения матрицы массы конечного элемента

vector<vector<int>> C;

NonLinearBVP()

{

C = {

{4, 2, -1},

{2, 16, 2},

{-1, 2, 4}

};

}

// Функция считывания областей из файла file\_name и формирования сетки

void ReadFormGrid(const string& file\_name)

{

ifstream fin(file\_name);

fin >> regions\_count;

regions.resize(regions\_count);

grid.resize(regions\_count \* 2);

for(int grid\_i = 0; grid\_i < regions\_count \* 2; grid\_i++)

fin >> grid[grid\_i];

string s;

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

fin >> s;

Region\* r = &regions[reg\_i];

int left, right;

// Считывание границы области

fin >> left;

fin >> right;

// Генерация координат узлов по X

int n;

double h, q;

fin >> q >> n;

r->nodes\_count = n + 1;

r->nodes.resize(r->nodes\_count);

h = grid[right] - grid[left];

if(q != 1)

h \*= (1 - q) / (1 - pow(q, n));

else

h /= n;

r->nodes[0] = grid[left];

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

r->nodes[i + 1] = r->nodes[i] + h \* pow(q, i);

// Считывание информации о краевых условиях

fin >> r->left\_bord;

fin >> r->right\_bord;

if(reg\_i > 0)

if(grid[reg\_i \* 2] == grid[(reg\_i - 1) \* 2 + 1])

r->first\_i = regions[reg\_i - 1].nodes\_count - 1;

else

r->first\_i = regions[reg\_i - 1].nodes\_count;

nodes\_count += r->nodes\_count;

r->elems\_count = r->nodes\_count / 2;

elems\_count += r->elems\_count;

}

fin.close();

q.resize(nodes\_count, 0);

q\_1.resize(nodes\_count, 0);

u.resize(nodes\_count, 1);

//u\_1.resize(nodes\_count, 0);

/\*for(int i = 0; i < nodes\_count; i++)

q[i] = pow(11 + 0.25 \* (i), 2) ;\*/

vec\_1.resize(nodes\_count);

}

// Функция считывания и формирования сетки по времени из файла file\_name

void ReadFormtimeGrid(const string& file\_name)

{

ifstream fin(file\_name);

double t0, tn;

fin >> t0;

fin >> tn;

// Генерация координат узлов по X

int n;

double h, q;

fin >> q >> n;

h = tn - t0;

if(q != 1)

h \*= (1 - q) / (1 - pow(q, n));

else

h /= n;

time\_grid.resize(n + 1);

time\_grid[0] = t0;

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

time\_grid[i + 1] = time\_grid[i] + h \* pow(q, i);

fin.close();

}

// Функция инициализации матрицы

void InitMatrix(Matrix& m)

{

m.size = 0;

m.bot\_tr = vector<double>(elems\_count \* 3);

m.top\_tr = vector<double>(elems\_count \* 3);

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

m.size += regions[reg\_i].elems\_count \* 2 + 1;

if(reg\_i + 1 < regions\_count && grid[reg\_i \* 2 + 1] == grid[(reg\_i + 1) \* 2])

m.size--;

}

m.di = vector<double>(m.size);

m.ind = vector<int>(m.size + 1);

}

// Функция формирования портрета глобальной матрицы

void FormPortrait(Matrix& m)

{

m.ind[0] = 0;

m.ind[1] = 0;

m.ind[2] = 1;

m.ind[slae.m.size] = m.top\_tr.size();

int global\_i = 3;

int val = 3;

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

Region\* r = &regions[reg\_i];

for(int elem\_i = reg\_i == 0 ? 1 : 0; elem\_i < r->elems\_count; elem\_i++, global\_i +=

2, val += 2)

{

m.ind[global\_i] = val++;

m.ind[global\_i + 1] = val;

}

if(reg\_i + 1 < regions\_count && grid[reg\_i \* 2 + 1] != grid[(reg\_i + 1) \* 2])

m.ind[global\_i++] = val;

}

}

// Функция заполнения матриц жесткости и массы

void FillMatrices(const double& t)

{

// Индекс очередного элемента в треугольнике матрицы

int to\_add\_i\_tr = 0;

// Индекс очередного элемента на диагонали матрицы

int to\_add\_i\_di = 0;

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

Region\* r = &regions[reg\_i];

for(int elem\_i = 0; elem\_i < r->elems\_count; elem\_i++)

{

// Индекс первого узла элемента

int elem\_beg\_i = elem\_i \* 2;

// Координаты узлов

vector<double> x\_elem = { r->nodes[elem\_beg\_i], r->nodes[elem\_beg\_i + 1],

r->nodes[elem\_beg\_i + 2] };

double h = x\_elem[2] - x\_elem[0];

// Индекс первого узла элемента в глобальной нумерации

int elem\_beg\_i\_glob = elem\_i \* 2 + r->first\_i;

vector<double> q\_elem = { u[elem\_beg\_i\_glob], u[elem\_beg\_i\_glob + 1],

u[elem\_beg\_i\_glob + 2] };

vector<double> lambda = { test.lambda(q\_elem[0], x\_elem[0], t),

test.lambda(q\_elem[1], x\_elem[1], t), test.lambda(q\_elem[2], x\_elem[2], t) };

// Заполнение диагонали матрицы жесткости

G.di[to\_add\_i\_di++] += (lambda[0] \* 37 / 30 + lambda[1] \* 1.2 - lambda[2] \* 0.1) / h;

G.di[to\_add\_i\_di++] += (lambda[0] \* 1.6 + lambda[1] \* 32 / 15 + lambda[2] \* 1.6) / h;

G.di[to\_add\_i\_di] += (-lambda[0] \* 0.1 + lambda[1] \* 1.2 + lambda[2] \* 37 / 30) / h;

// Заполнение нижнего треугольника матрицы жесткости

G.bot\_tr[to\_add\_i\_tr++] += (-lambda[0] \* 22 / 15 - lambda[1] \* 16 / 15 –

lambda[2] \* 2 / 15) / h;

G.bot\_tr[to\_add\_i\_tr++] += (lambda[0] \* 7 / 30 - lambda[1] \* 2 / 15 +

lambda[2] \* 7 / 30) / h;

G.bot\_tr[to\_add\_i\_tr++] += (-lambda[0] \* 2 / 15 - lambda[1] \* 16 / 15 –

lambda[2] \* 22 / 15) / h;

to\_add\_i\_di -= 2;

to\_add\_i\_tr -= 3;

// Заполнение диагонали матрицы массы

M.di[to\_add\_i\_di++] += test.sigma() \* h / 30.0 \* C[0][0];

M.di[to\_add\_i\_di++] += test.sigma() \* h / 30.0 \* C[1][1];

M.di[to\_add\_i\_di] += test.sigma() \* h / 30.0 \* C[2][2];

// Заполнение нижнего треугольника массы

M.bot\_tr[to\_add\_i\_tr++] += test.sigma() \* h / 30.0 \* C[1][0];

M.bot\_tr[to\_add\_i\_tr++] += test.sigma() \* h / 30.0 \* C[2][0];

M.bot\_tr[to\_add\_i\_tr++] += test.sigma() \* h / 30.0 \* C[2][1];

vector<double> f\_elem = { test.f(x\_elem[0], t), test.f(x\_elem[1], t),

test.f(x\_elem[2], t) };

// Заполнение вектора правой части

slae.b[to\_add\_i\_di - 2] += h / 30.0 \* (C[0][0] \* f\_elem[0] + C[0][1] \* f\_elem[1]

+ C[0][2] \* f\_elem[2]);

slae.b[to\_add\_i\_di - 1] += h / 30.0 \* (C[1][0] \* f\_elem[0] + C[1][1] \* f\_elem[1]

+ C[1][2] \* f\_elem[2]);

slae.b[to\_add\_i\_di - 0] += h / 30.0 \* (C[2][0] \* f\_elem[0] + C[2][1] \* f\_elem[1]

+ C[2][2] \* f\_elem[2]);

}

}

// Заполнение верхних треуголников

G.top\_tr = G.bot\_tr;

M.top\_tr = M.bot\_tr;

}

// Функция сбора глобальной матрицы системы

void GlobalMatrixAssemble()

{

int tr\_size = slae.m.bot\_tr.size();

for(int i = 0; i < tr\_size; i++)

slae.m.bot\_tr[i] = G.bot\_tr[i] + M.bot\_tr[i];

slae.m.top\_tr = slae.m.bot\_tr;

for(int i = 0; i < slae.m.size; i++)

slae.m.di[i] = G.di[i] + M.di[i];

}

// Функция учета краевых условий

void AccountBound(const double& t)

{

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

Region\* r = &regions[reg\_i];

if(r->left\_bord == 1)

{

/\*slae.m.di[r->first\_i] = big\_num;

slae.b[r->first\_i] = big\_num \* test.u\_prec(r->nodes[0], t);\*/

slae.m.di[r->first\_i] = 1.0;

slae.b[r->first\_i] = test.u\_prec(r->nodes[0], t);

slae.m.top\_tr[slae.m.ind[r->first\_i + 1]] = 0;

slae.m.top\_tr[slae.m.ind[r->first\_i + 2]] = 0;

}

if(r->right\_bord == 1)

{

/\*slae.m.di[r->first\_i + r->nodes\_count - 1] = big\_num;

slae.b[r->first\_i + r->nodes\_count - 1] = big\_num \* test.u\_prec(r->nodes[r->nodes\_count - 1], t);\*/

slae.m.di[r->first\_i + r->nodes\_count - 1] = 1.0;

slae.b[r->first\_i + r->nodes\_count - 1] = test.u\_prec(r->nodes[r->nodes\_count

- 1], t);

slae.m.bot\_tr[slae.m.ind[r->first\_i + r->nodes\_count - 1]] = 0;

slae.m.bot\_tr[slae.m.ind[r->first\_i + r->nodes\_count - 1] + 1] = 0;

}

}

}

// Функция расчета нормы вектора

static double CalcNorm(const vector<double>& vec)

{

double res = 0;

int size = vec.size();

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

res += vec[i] \* vec[i];

return sqrt(res);

}

// Линеаризация методом простой итераций, вывод итераций в файл file\_name

void SimpleIterations(const double& t, const double& eps, const double& delta,

const int& max\_iter, ofstream& fout)

{

int n\_iter = 0;

double eps\_residual = 1;

double delta\_residual = 1;

do

{

// Инициализация матриц

InitMatrix(slae.m);

InitMatrix(G);

InitMatrix(M);

slae.b = vector<double>(slae.m.size);

// Сборка матриц жесткости и массы

FormPortrait(slae.m);

FormPortrait(G);

FormPortrait(M);

FillMatrices(t);

// Сборка матрицы системы

GlobalMatrixAssemble();

AccountBound(t);

vec\_1 = vector<double>(nodes\_count);

slae.m.MatVecMult(u, vec\_1);

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

vec\_1[i] -= slae.b[i];

eps\_residual = CalcNorm(vec\_1) / CalcNorm(slae.b);

if(eps\_residual < eps)

{

break;

}

else

{

slae.LUDecomp();

slae.ForwardSolver();

slae.BackwardSolver();

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

vec\_1[i] = slae.b[i] - u[i];

delta\_residual = CalcNorm(vec\_1) / CalcNorm(slae.b);

if(delta\_residual < delta)

{

break;

}

u = slae.b;

fout << endl << "Non-linear iteration " << n\_iter << endl;

PrintSolution(u, t, fout);

n\_iter++;

}

} while(0 < n\_iter && n\_iter < max\_iter);

}

// Явная разностная схема по времени, вывод итераций в поток fout

void ExplicitScheme(const double& eps, const double& delta, const int& max\_iter,

const string& file\_name)

{

ofstream fout(file\_name);

// Расчет вектора приближения при t=0

q\_1.resize(nodes\_count);

int to\_add\_i = 0;

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

Region\* r = &regions[reg\_i];

for(int elem\_i = 0; elem\_i < r->elems\_count; elem\_i++)

{

q\_1[to\_add\_i++] = test.u\_prec(r->nodes[elem\_i \* 2], 0.0);

q\_1[to\_add\_i++] = test.u\_prec(r->nodes[elem\_i \* 2 + 1], 0.0);

q\_1[to\_add\_i] = test.u\_prec(r->nodes[elem\_i \* 2 + 2], 0.0);

}

}

for(int time\_i = 1; time\_i < time\_grid.size(); time\_i++)

{

for(int i = 0; i < 100; i++)

fout << "#";

fout << endl;

const double t = time\_grid[time\_i];

const double dt = t - time\_grid[time\_i - 1];

fout << "t = " << fixed << t << endl;

int n\_iter = 0;

double eps\_residual = 1;

double delta\_residual = 1;

do

{

// Инициализация матриц

InitMatrix(slae.m);

InitMatrix(G);

InitMatrix(M);

slae.b = vector<double>(slae.m.size);

vec\_2 = vector<double>(slae.m.size);

// Сборка матриц жесткости и массы

FormPortrait(slae.m);

FormPortrait(G);

FormPortrait(M);

FillMatrices(t);

// Сборка матрицы системы

const int tr\_size = slae.m.bot\_tr.size();

for(int i = 0; i < tr\_size; i++)

slae.m.bot\_tr[i] = G.bot\_tr[i] + M.bot\_tr[i] / dt;

slae.m.top\_tr = slae.m.bot\_tr;

for(int i = 0; i < slae.m.size; i++)

slae.m.di[i] = G.di[i] + M.di[i] / dt;

M.MatVecMult(q\_1, vec\_2);

for(int i = 0; i < slae.m.size; i++)

slae.b[i] += vec\_2[i] / dt;

AccountBound(t);

vec\_1 = vector<double>(nodes\_count);

slae.m.MatVecMult(u, vec\_1);

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

vec\_1[i] -= slae.b[i];

eps\_residual = CalcNorm(vec\_1) / CalcNorm(slae.b);

if(eps\_residual < eps)

{

break;

}

else

{

slae.LUDecomp();

slae.ForwardSolver();

slae.BackwardSolver();

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

vec\_1[i] = slae.b[i] - u[i];

delta\_residual = CalcNorm(vec\_1) / CalcNorm(slae.b);

if(delta\_residual < delta)

{

break;

}

u = slae.b;

//fout << endl << "Non-linear iteration " << n\_iter << endl;

//PrintSolution(u, t, fout);

n\_iter++;

}

} while(0 < n\_iter && n\_iter < max\_iter);

q = u;

q\_1 = q;

//if (time\_i % 8 == 0)

{

fout << "Number of iterations: " << n\_iter << endl;

fout << "Final result on timestep:" << endl;

PrintSolution(q, t, fout);

}

}

fout.close();

}

// Вывод решения на основе вектора текущего приближения solution

// во временной точке t в поток fout

void PrintSolution(const vector<double>& solution, const double& t, ofstream& fout)

{

double norm = 0., norm\_u = 0.;

fout << " x calc prec dif N location" << endl << fixed;

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

Region\* r = &regions[reg\_i];

for(int node\_i = 0; node\_i < r->nodes\_count; node\_i++)

{

//if (node\_i % 16 == 0)

{

// Индекс узла в глобальной нумерации

int elem\_beg\_i = node\_i + r->first\_i;

fout << setw(11) << r->nodes[node\_i];

double help\_1 = solution[elem\_beg\_i];

fout << setw(15) << help\_1;

double help\_2 = test.u\_prec(r->nodes[node\_i], t);

fout << setw(15) << help\_2;

fout << setw(14) << scientific << abs(help\_1 - help\_2) << fixed;

fout << setw(4) << elem\_beg\_i << " ";

if (node\_i == 0 || node\_i == r->nodes\_count - 1)

fout << " border";

else

fout << " inner";

fout << endl;

norm\_u += help\_2 \* help\_2;

norm += abs(help\_1 - help\_2) \* abs(help\_1 - help\_2);

}

}

//PrintSolutionInside(&regions[reg\_i], solution, t, fout, norm\_u, norm);

}

fout << "||u-u\*||/||u\*|| = " << scientific << sqrt(norm) / sqrt(norm\_u) << endl;

fout << "||u-u\*|| = " << scientific << sqrt(norm) << endl << endl;

}

// Вывод решения на основе вектора текущего приближения solution

// во временной точке t в поток fout

void PrintSolutionInside(Region \*r, const vector<double>& solution, const double& t,

ofstream& fout, double &norm\_u, double &norm)

{

// Индекс узла в глобальной нумерации

int i = r->first\_i;

double h = (r->nodes[2] - r->nodes[0]);

double x = r->nodes[0] + h / 4;

double e = (x - r->nodes[0]) / h;

double fi1 = 2 \* (e - 0.5) \* (e - 1);

double fi2 = - 4 \* e \* (e - 1);

double fi3 = 2 \* (e - 0.5) \* e;

double res = fi1 \* solution[i] + fi2 \* solution[i + 1] + fi3 \* solution[i + 2];

fout << setw(11) << x;

fout << setw(15) << res;

double help\_2 = test.u\_prec(x, t);

fout << setw(15) << help\_2;

fout << setw(14) << scientific << abs(res - help\_2) << fixed;

fout << setw(5) << "- ";

fout << " point" << endl;

norm\_u += help\_2 \* help\_2;

norm += abs(res - help\_2) \* abs(res - help\_2);

}

};

**Файл “Test.h”**

#pragma once

#pragma once

using namespace std;

class Test

{

public:

int N, M;

Test(const int& t\_N, const int& t\_M) : N(t\_N), M(t\_M) {};

Test() : N(0), M(0) {};

double f(const double& x, const double& t)

{

return -1 \* (dudx(x, t) \* dlambdadx(x, t) + d2udx2(x, t) \* lambda(u\_prec(x, t), x, t))

+ sigma() \* dudt(x, t);

}

double sigma()

{

return 1;

}

double lambda(const double& u, const double& x, const double& t)

{

switch(M)

{

case 0: return 1;

//case 1: return u \* u + 1;

case 1: return u + 1;

//case 1: return u + t \* x;

}

}

// Производная lambda по x после подстановки x в функцию u

double dlambdadx(const double& x, const double& t)

{

switch(M)

{

case 0: return 0;

case 1:

{

switch(N)

{

case(0): return 0;

//case(1): return 1;

//case(1): return 2 \* x;

case(1): return 1 + t;

//case(2): return t;

case(2): return 2 \* x \* t \* t;

//case(3): return 2 \* x;

case(3): return 4 \* x \* x \* x;

case(4): return 3 \* x \* x;

//case(4): return 2 \* x + 1;

case(5): return 4 \* x \* x \* x;

case(6): return 0;

//case(7): return 2 \* x \* t \* t;

case(7): return 0;

case(8): return 2 \* x \* t;

}

}

}

}

// Точное решение

double u\_prec(const double& x, const double& t)

{

switch(N)

{

case(0): return 2.0;

case(1): return x;

case(2): return x \* t;

case(3): return x \* x;

case(4): return x \* x \* x;

case(5): return x \* x \* x \* x;

//case(5): return x \* x \* x \* x \* t;

case(6): return t;

case(7): return t \* t;

case(8): return x \* x \* t;

};

}

// Первая производная точного решения по х

double dudx(const double& x, const double& t)

{

switch(N)

{

case(1): return 1;

case(2): return t;

case(3): return 2 \* x;

case(4): return 3 \* x \* x;

case(5): return 4 \* x \* x \* x;

case(8): return 2 \* x \* t;

default: return 0;

};

}

// Вторая производная точного решения по х

double d2udx2(const double& x, const double& t)

{

switch(N)

{

case(3): return 2;

case(4): return 6 \* x;

case(5): return 12 \* x \* x;

case(8): return 2 \* t;

default: return 0;

};

}

// Производная точного решения по t

double dudt(const double& x, const double& t)

{

switch(N)

{

case(2): return x;

case(6): return 1;

//case(5): return x \* x \* x \* x;

case(7): return 2\*t;

case(8): return x \* x;

default: return 0;

};

}

};

**Файл “main.cpp”**

#include <iostream>

#include "NonLinearBVP.h"

int main()

{

NonLinearBVP nl = NonLinearBVP();

nl.test = Test(4, 1);

nl.ReadFormGrid("regions.txt");

nl.ReadFormtimeGrid("time.txt");

nl.ExplicitScheme(1e-14, 1e-14, 100, "result.txt");

}