**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Теория разностных схем

**Отчет по лабораторной работе № 2**

«Решение начально-краевой задачи для уравнения теплопроводности»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа МКН-315 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Халиков А.Р. |  |  |  |
| Принял | Белевцов Н.С. |  |  |  |

**Уфа 2022**

**Цель работы:** получить навык численного решения линейных и нелинейных начально-краевых задач для уравнений параболического типа с использованием различных конечно-разностных схем на примере задачи для одномерного уравнения теплопроводности с источником.

Для достижения цели необходимо выполнить следующие задачи:

1. Изучить необходимый теоретический материал.
2. Выполнить задания согласно 9 варианту.

**Теоретическая часть**

***Начально-краевая задача***

Рассматривается начально-краевая задача для нелинейного одномерного уравнения теплопроводности с источником:

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |
|  | (3) |
|  | (4) |

1. ***Сравнение конечно-разностных схем для линейной задачи***

Рассматривается линейный случай уравнения теплопроводности (1):

(5)

Параметры задачи выбираются в соответствии с индивидуальным заданием (Таблица 1).

Методом разделения переменных построить аналитическое решение задачи.

***Задача 1. Явная разностная схема.***

При решении данной задачи (1)-(5) используется явная конечно-разностная схема с шаблоном «левый уголок» на равномерной пространственно-временной сетке:

Порядок аппроксимации данной схемы: .

***Задача 2. Неявная разностная схема.***

При решении данной задачи (1)-(5) используются неявная конечно-разностная схема и схема Кранка-Николсона.

Неявная схема представляется следующим образом:

Схема Кранка-Схема Кранка-Николсона имеет вид:

Порядок аппроксимации полностью неявной схемы: , а схемы Кранка-Николсона - . Полностью неявная схема устойчива при любых соотношениях между шагами сеткисхема Кранка-Николсона обычно абсолютно устойчива, выбирается как .

**II. Решение нелинейной задачи с использованием консервативной схемы.**

Решается нелинейная задача (1)-(4) с дополнительными исходными данными и из таблицы 2, где

а функция и остальные данные берутся из таблицы 1.

***Задача 3. Консервативная схема на равномерной сетке.***

При решении данной задачи (1)-(4) используется консервативная схема на равномерной сетке:

где

***Задача 4. Модификация задачи 3.***

Модификация программы из задачи 3 реализуется путем организации внутренних итераций на каждом временном шаге для повышения точности вычисления нелинейных слагаемых. Условием остановки итерационного процесса является достижение заданного преподавателем уровня погрешности вычислений нелинейных функций.

**Практическая часть**

1. ***Сравнение конечно-разностных схем для линейной задачи***

***Задача 1.***

1. Написать вычислительную программу на языке программирования C++ решения задачи (1)-(5) с использованием явной разностной схемы на равномерной пространственно-временной сетке.
2. Непосредственными расчетами продемонстрировать условную устойчивость схемы и справедливость условия устойчивости.
3. Исследовать зависимость решения от величины шагов сетки по пространственной и временной переменным посредством сравнения с построенным аналитическим решением. Построить графики зависимости погрешности, оцениваемой в равномерной норме по пространственной переменной, от времени и шагов сетки.

В Задаче 1для решения используется явная конечно-разностная схема на равномерной пространственно-временной сетке.

Вычисления проводятся на сетке с шагами по X: 0.1, 0.02, 0.01, и , удовлетворяющем условию сходимости: 5е-3, 2е-04 и 5е-5.

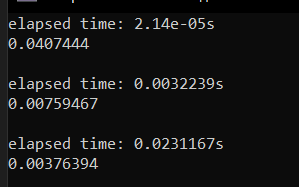


Рисунок 1 - Демонстрация сходимости схемы.

Далее исследуем зависимость решения от величины шагов сетки по пространственной переменной, посредством сравнения с аналитическим решением. Для проверки по пространственной переменной зафиксирует , и будем изменять X: 10, 50,100; h = 0.1, 0.02, 0.01.

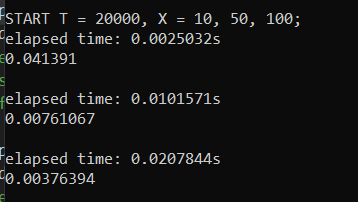
**

Рисунок 2 - зависимость решения от величины шагов сетки по пространственной переменной.

Далее исследуем зависимость решения от величины шагов сетки по временной переменной, посредством сравнения с аналитическим решением. Для проверки по пространственной переменной зафиксирует .

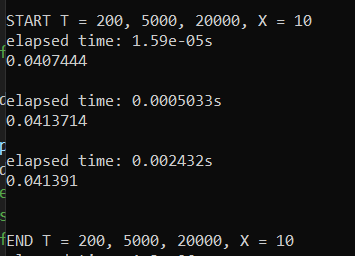


Рисунок 3 - зависимость решения от величины шагов сетки по временной переменной.

Теперь намеренно нарушим условие устойчивости (x = 10, T = 10) и получим результат:

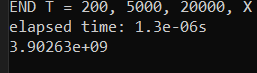


Рисунок 4 - ошибка, при нарушении условие устойчивости.

Как видно из (Рисунок 4), при нарушении условия устойчивости, схема расходится.

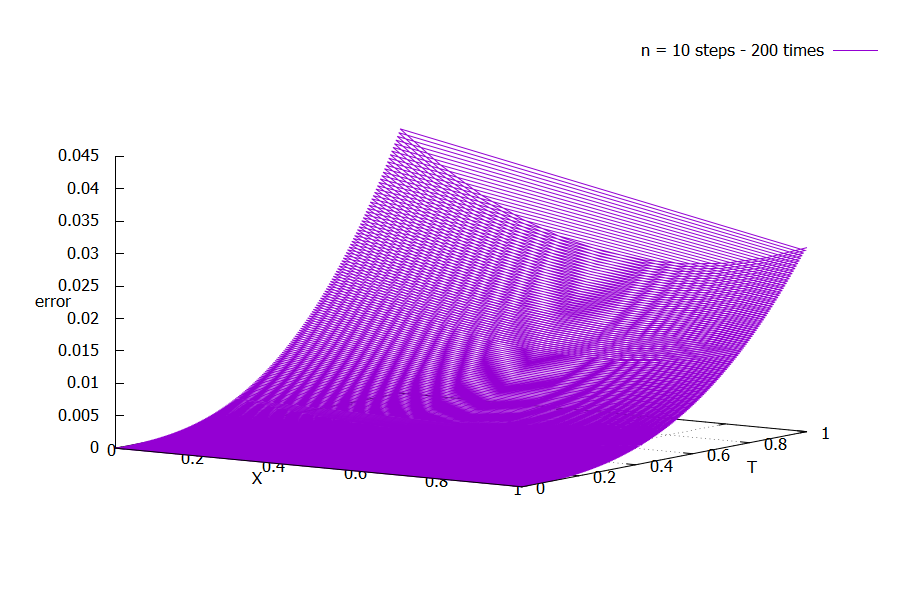


Рисунок 5 - график ошибки при x = 10; T = 200;

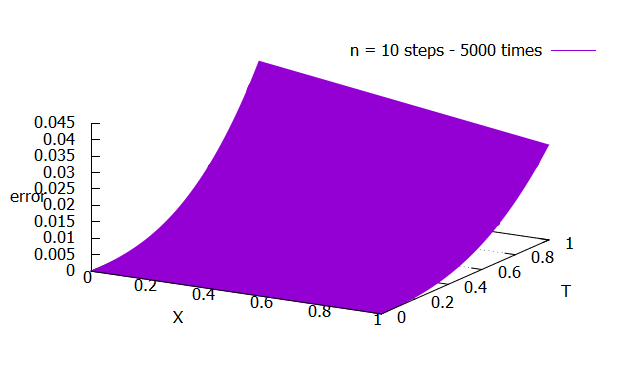


Рисунок 6 - график ошибки при x = 10; T = 5000;

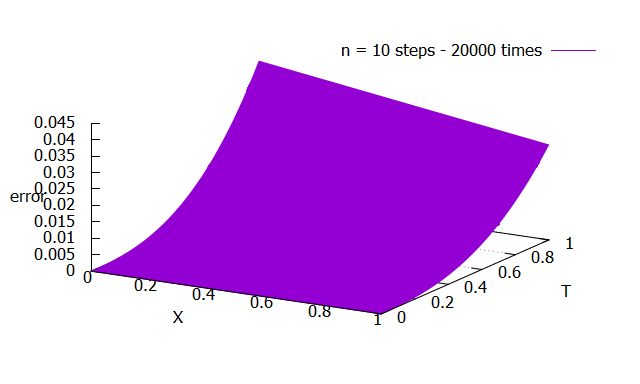


Рисунок 7 - график ошибки при x = 10; T = 20000;

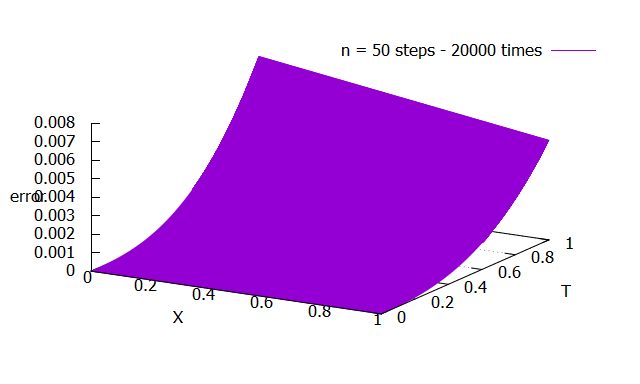


Рисунок 8 - график ошибки при x = 50; T = 20000;

Как видно из (Рисунок 5, Рисунок 6), если при фиксированном x менять разбиение по времени, то ошибка не меняется.

Из рисунков (Рисунок 7, Рисунок 8) можно сделать вывод, что при фиксированном разбиение по времени, меняя разбиение по пространству, уменьшается ошибка.

***Задача 2.***

1. Написать вычислительную программу на языке программирования C++ решения задачи (1)-(5) по полностью неявной схеме и схеме Кранка-Николсона на равномерной сетке.
2. Выполнить сравнение точности получаемого решения по двум схемам с использованием точного решения. Построить графики погрешностей как функций координат и времени, а также графики норм погрешностей как функций шагов сетки.
3. Сравнить время решения задач по трем схемам (явной, полностью неявной и Кранка-Николсона), обеспечивающих получение решения с одинаковым уровнем погрешности.

В Задаче 2для решения используется полностью не явная схема и схема Кранка-Николсона на равномерной пространственно-временной сетке.

Произведем вычисления на двух схемах при разбиении по временной и пространственной переменной T = 5000, X = 50.

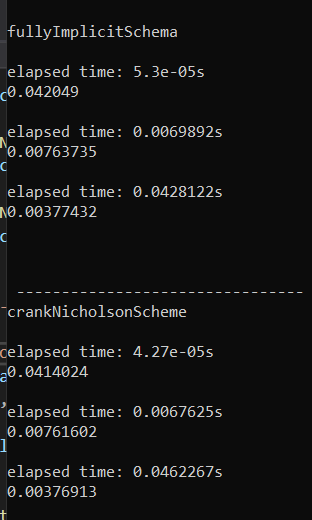


Рисунок 9 - Ошибки по норме для двух методов, при разбиениях X = 10, 50, 100; T =200, 5000, 20000;

Как видно из (Рисунок 9), схемы, фактически, имеют одинаковую точность решения.

И построим графики ошибки при X = 50, T = 5000.

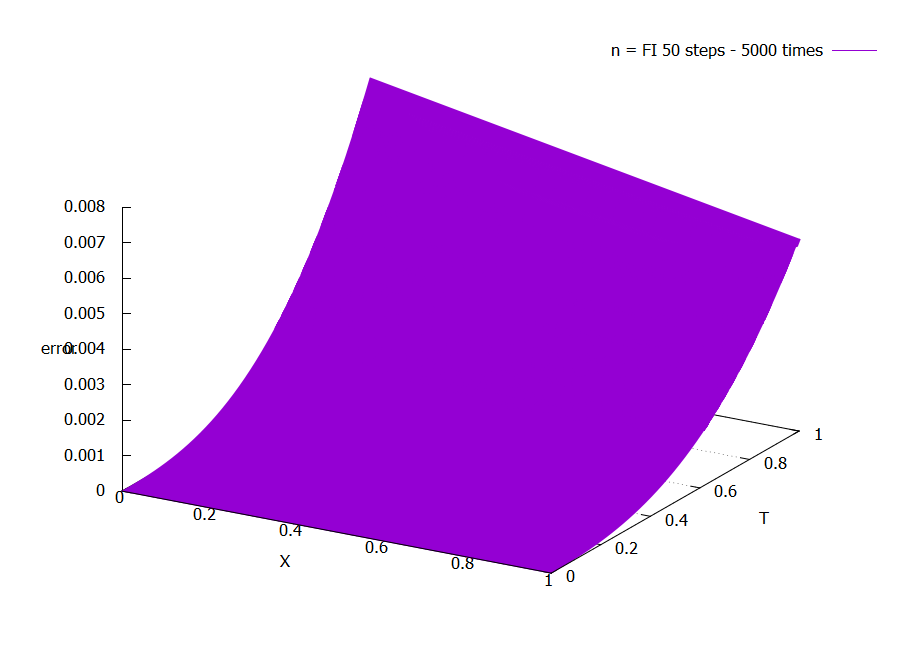


Рисунок 10 - график ошибки чисто неявной схемы.

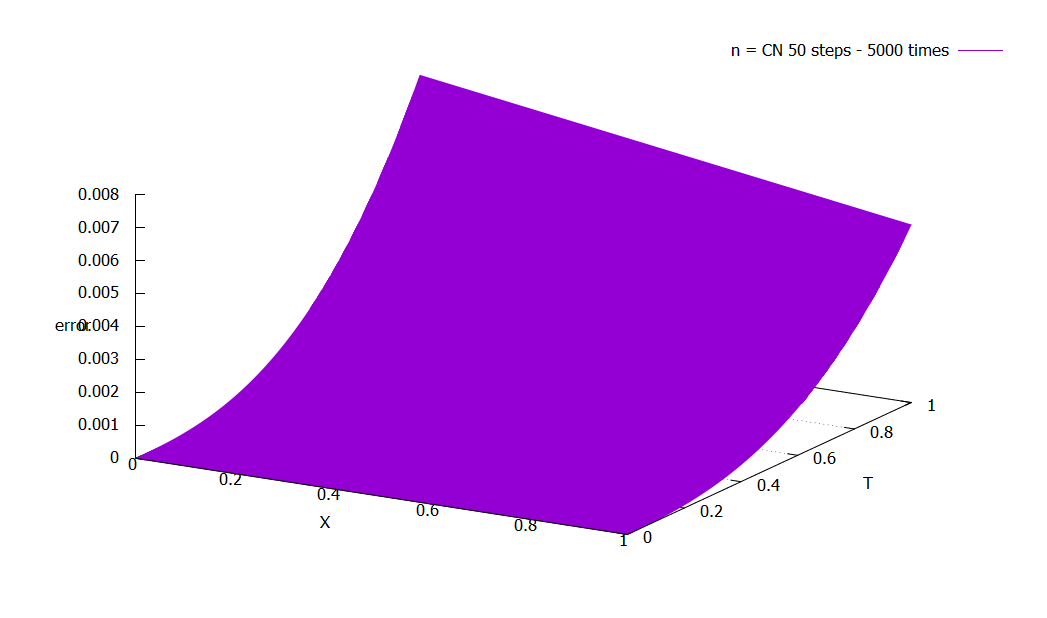


Рисунок 11 - график ошибки схемы Кранка-Николсона.

Таблица 1 - время работы разных схем, при одинаковой погрешности.

|  |  |  |
| --- | --- | --- |
| Схемы | Время работы программы | Погрешность. |
| Явная схема | 0.0093792s | 0.00759467 |
| Полностью неявная схема | 0.0142846s | 0.00763735 |
| Схема Кранка-Николсона | 0.0164354s | 0.00761602 |

Как видно из (Таблица 1), неявные схемы, при той же точности, работают гораздо дольше из-за большей сложности самих схем.

***Задача 3 (2 балла).***

1. Написать вычислительную программу на языке программирования C++ решения задачи (1)-(4) с использованием консервативной схемы на равномерной сетке.
2. Убедиться в корректности программы на примере задачи 1.
3. Исследовать зависимость получаемого решения от величины шага сетки по пространственной и временной переменным. Построить графики решений для различных значений шага.

Построим консервативную схему, и убедимся в ее работоспособности на примере задачи 1. Для этого сделаем так, чтобы функция возвращала k = 1, и F = 1;

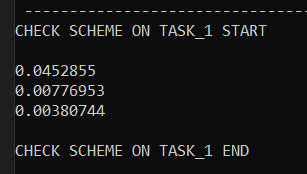


Рисунок 12 - ошибки консервативной схемы для 1-ой задачи.

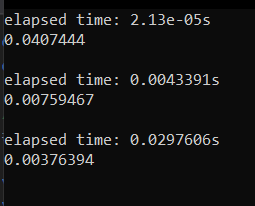


Рисунок 13 – ошибки явной схемы для 1-ой задачи.

Как видно из (Рисунок 12, Рисунок 13) погрешности одинаковы, значит консервативная схема работает верно.

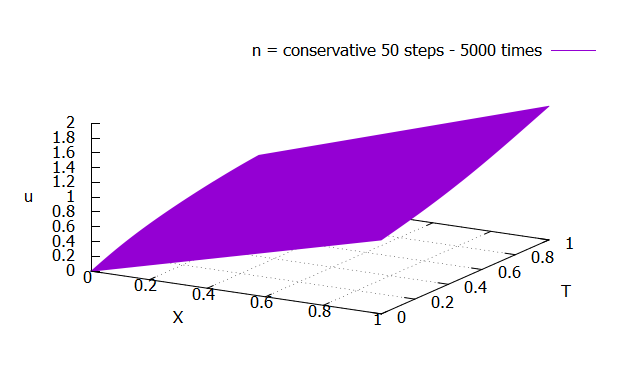
Теперь с помощью консервативной схемы решим задачу 2 и построим график решения.

Рисунок 14 - график решения при X = 50; T = 5000.

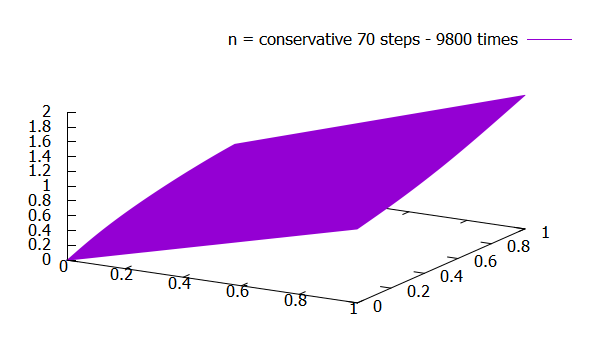


Рисунок 15 - график решения при X = 70; T = 9800.

**Вывод**

В ходе выполнения лабораторной работы был получен навык численного решения краевых задач для уравнений гиперболического типа на примере начально-краевой задачи для линейного одномерного уравнения переноса и линейного одномерного неоднородного волнового уравнения. Было проведено сравнение методов решения.

Для каждой поставленной задачи написана вычислительная программа на языке программирования С++, выполняющая необходимые построения и расчеты.

Для линейной задачи, уравнения теплопроводности, было выявлено, что при одинаковой точности явная схема считает быстрее, нежели полунеявная или неявная. Но при этом, неявная схема абсолютно устойчива.

Для ОДУ второго порядка были построены решения конечно-разностным методом. С помощью метода прогонки была решена система уравнений и получено решение при различных Было проведено сравнение и исследование зависимости для численного приближенного решения к решению, выдаваемое математическим пакетом Maple.

**Список литературы**

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы: Бином, 2018. – 636 с.
2. Самарский А.А., Гулин А. В. Численные методы: Учеб, пособие для вузов, — М.: Наука. Гл. ред. физ-мат. лит., 1989.— 432 с.

**Приложение:**

ConservativeScheme.h

#pragma once

#include "ImplicitSchema.h"

class ConservativeScheme : public ImplicitSchema

{

public:

ConservativeScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

std::function<double(double)> F,

std::function<double(double)> k,

const std::string& path,

int gridPitchT = -1.);

//~ConservativeScheme();

double virtual calculate(const bool isSaveGraph = false, const bool isSave = false) override;

private:

std::function<double(double)> F;

std::function<double(double)> k;

double function(double t, double x, double u);

};

#pragma once

ConservativeScheme.cpp

#include "ConservativeScheme.h"

ConservativeScheme::ConservativeScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

std::function<double(double)> F,

std::function<double(double)> k,

const std::string& path,

int gridPitchT) : ImplicitSchema(

T,

numberOfStepsX,

alpha\_0,

alpha\_1,

beta\_0, beta\_1,

k\_0,

psi\_0,

psi\_1,

phi,

f,

solution,

path,

gridPitchT

)

{

this->F = F;

this->k = k;

}

double ConservativeScheme::calculate(const bool isSaveGraph, const bool isSaveError)

{

double\* elementsMatrix = new double[(numberOfStepsX + 1) \* 4]; // 0 - верх диаг.; 1 - ср. диаг.; 2 - нижн. диаг.;

double k\_e;

double coeff = 1.;

for (unsigned i = 0; i <= numberOfStepsX; ++i) {

layers[0 \* (numberOfStepsT + 1) + i] = phi(i \* gridPitchX);

}

for (unsigned t = 0; t < numberOfStepsT; ++t) {

//верхн.

elementsMatrix[0 \* (numberOfStepsX + 1) + 0] = beta\_0;

//ср.

elementsMatrix[1 \* (numberOfStepsX + 1) + 0] = gridPitchX \* alpha\_0 - beta\_0;

elementsMatrix[1 \* (numberOfStepsX + 1) + numberOfStepsX] = alpha\_1 \* gridPitchX + beta\_1;

//нижн;

elementsMatrix[2 \* (numberOfStepsX + 1) + numberOfStepsX - 1] = -beta\_1;

//прав;

elementsMatrix[3 \* (numberOfStepsX + 1) + 0] = gridPitchX \* psi\_0(gridPitchT \* (t + 1));

elementsMatrix[3 \* (numberOfStepsX + 1) + numberOfStepsX] = gridPitchX \* psi\_1(gridPitchT \* (t + 1));

for (unsigned x = 1; x < numberOfStepsX; ++x) {

double V\_plus = k((layers[t \* (numberOfStepsX + 1) + x] + layers[t \* (numberOfStepsX + 1) + x + 1]) / 2.);

double V\_minus = k((layers[t \* (numberOfStepsX + 1) + x] + layers[t \* (numberOfStepsX + 1) + x - 1]) / 2.);

//double V\_plus = ( k(layers[t \* (numberOfStepsX + 1) + x]) + k(layers[t \* (numberOfStepsX + 1) + x + 1])) / 2.;

//double V\_minus = ( k(layers[t \* (numberOfStepsX + 1) + x]) + k(layers[t \* (numberOfStepsX + 1) + x - 1]) ) / 2.;

//верхн.

elementsMatrix[0 \* (numberOfStepsX + 1) + x] = gridPitchT \* coeff / (gridPitchX \* gridPitchX) \* V\_plus;

//ср.

elementsMatrix[1 \* (numberOfStepsX + 1) + x] = -1. - gridPitchT \* coeff / (gridPitchX \* gridPitchX) \* (V\_plus + V\_minus);

//нижн;

elementsMatrix[2 \* (numberOfStepsX + 1) + x - 1] = gridPitchT \* coeff / (gridPitchX \* gridPitchX) \* V\_minus;

//прав;

elementsMatrix[3 \* (numberOfStepsX + 1) + x] = -layers[t \* (numberOfStepsX + 1) + x] -

gridPitchT \* (1 - coeff) / gridPitchX \* (V\_plus \* (layers[t \* (numberOfStepsX + 1) + x + 1] - layers[t \* (numberOfStepsX + 1) + x]) / gridPitchX -

V\_minus \* (layers[t \* (numberOfStepsX + 1) + x] - layers[t \* (numberOfStepsX + 1) + x - 1]) / gridPitchX) -

gridPitchT \* function(gridPitchT \* t, gridPitchX \* x, layers[t \* (numberOfStepsX + 1) + x + 1]);

}

progonka(elementsMatrix, t + 1);

}

delete[] elementsMatrix;

if (isSaveGraph) {

saveToFile(fileName + "\_" + std::to\_string(numberOfStepsX) + "\_" + std::to\_string(numberOfStepsT) + "Solution.txt", layers);

}

if (isSaveError) {

return error(fileName);

}

else {

return error();

}

}

double ConservativeScheme::function(double t, double x, double u)

{

return F(u) \* f(t, x);

}

CrankNicholsonScheme.h

#pragma once

#include "ImplicitSchema.h"

class CrankNicholsonScheme : public ImplicitSchema

{

public:

CrankNicholsonScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path);

//~CrankNicholsonScheme();

virtual double calculate(const bool isSaveGraph = false, const bool isSaveError = false) override;

private:

};

CrankNicholsonScheme.cpp

#include "CrankNicholsonScheme.h"

CrankNicholsonScheme::CrankNicholsonScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path) : ImplicitSchema(

T,

numberOfStepsX,

alpha\_0,

alpha\_1,

beta\_0, beta\_1,

k\_0,

psi\_0,

psi\_1,

phi,

f,

solution,

path

)

{

}

double CrankNicholsonScheme::calculate(const bool isSaveGpaph, const bool isSaveError )

{

auto start = std::chrono::steady\_clock::now();

double\* elementsMatrix = new double[(numberOfStepsX + 1) \* 4]; // 0 - верх диаг.; 1 - ср. диаг.; 2 - нижн. диаг.;

for (unsigned i = 0; i <= numberOfStepsX; ++i) {

layers[0 \* (numberOfStepsT + 1) + i] = phi(i \* gridPitchX);

}

for (unsigned t = 0; t < numberOfStepsT; ++t) {

//верхн.

elementsMatrix[0 \* (numberOfStepsX + 1) + 0] = beta\_0;

//ср.

elementsMatrix[1 \* (numberOfStepsX + 1) + 0] = gridPitchX \* alpha\_0 - beta\_0;

elementsMatrix[1 \* (numberOfStepsX + 1) + numberOfStepsX] = alpha\_1 \* gridPitchX + beta\_1;

//нижн;

elementsMatrix[2 \* (numberOfStepsX + 1) + numberOfStepsX - 1] = -beta\_1;

//прав;

elementsMatrix[3 \* (numberOfStepsX + 1) + 0] = gridPitchX \* psi\_0(gridPitchT \* (t + 1));

elementsMatrix[3 \* (numberOfStepsX + 1) + numberOfStepsX] = gridPitchX \* psi\_1(gridPitchT \* (t + 1));

for (unsigned x = 1; x < numberOfStepsX; ++x) {

//верхн.

elementsMatrix[0 \* (numberOfStepsX + 1) + x] = gridPitchT \* k\_0 / (2. \* gridPitchX \* gridPitchX);

//ср.

elementsMatrix[1 \* (numberOfStepsX + 1) + x] = -1 - gridPitchT \* k\_0 / (gridPitchX \* gridPitchX);

//нижн;

elementsMatrix[2 \* (numberOfStepsX + 1) + x - 1] = gridPitchT \* k\_0 / (2. \* gridPitchX \* gridPitchX);

//прав;

elementsMatrix[3 \* (numberOfStepsX + 1) + x] = -layers[t \* (numberOfStepsX + 1) + x] -

gridPitchT \* f(gridPitchT \* (t + .5), x \* gridPitchX) -

gridPitchT \* k\_0 \* .5 / (gridPitchX \* gridPitchX) \*

(

layers[t \* (numberOfStepsX + 1) + x - 1] + layers[t \* (numberOfStepsX + 1) + x + 1] -

2 \* layers[t \* (numberOfStepsX + 1) + x]

);

}

progonka(elementsMatrix, t + 1);

}

delete[] elementsMatrix;

auto end = std::chrono::steady\_clock::now();

std::chrono::duration<double> elapsed\_seconds = end - start;

std::cout << "elapsed time: " << elapsed\_seconds.count() << "s\n";

if (isSaveError) {

return error(fileName);

}

else {

return error();

}

}

DifferenceScheme.h

#pragma once

#include <iostream>

#include <vector>

#include <functional>

#include <string>

#include <fstream>

#include <memory>

#include <chrono>

class DifferenceScheme

{

public:

DifferenceScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path,

int numberOfStepsT = -1

);

~DifferenceScheme();

void drawGraph(std::vector<std::pair<std::string, std::string>>&& pathTofiles,

std::string&& graphName, const std::string path, std::string&& labelX, std::string&& labelY, std::string&& labelZ);

void drawGraphNorm(std::vector<std::pair<std::string, std::string>>&& pathTofiles,

std::string&& graphName, const std::string path, std::string&& labelX, std::string&& labelY);

void setNumberOfStepsSpace(unsigned numberOfStepsX);

void setNumberOfStepsTimeAndSpace(unsigned numberOfStepsT, double numberOfStepsX);

protected:

virtual double calculate(const bool isSaveGraph = false, const bool isSaveError = false) { return -1; };

double error(const std::string& fileName = "");

double\* layers;

std::function<double(double)> psi\_0;

std::function<double(double)> psi\_1;

std::function<double(double)> phi;

std::function<double(double, double)> f;

std::function<double(double, double)> solution;

double alpha\_0;

double alpha\_1;

double beta\_0;

double beta\_1;

double k\_0;

double gridPitchX;

double gridPitchT;

unsigned numberOfStepsX;

unsigned numberOfStepsT;

std::string fileName;

double T;

void saveToFile(const std::string& fileName, double\* array);

private:

};

DifferenceScheme.cpp

#include "DifferenceScheme.h"

DifferenceScheme::DifferenceScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path,

int numberOfStepsT) : alpha\_0{ alpha\_0 },

T{ T },

numberOfStepsX{ numberOfStepsX },

alpha\_1{ alpha\_1 },

beta\_0{ beta\_0 },

beta\_1{ beta\_1 },

k\_0{ k\_0 },

psi\_0{ psi\_0 },

psi\_1{ psi\_1 },

phi{ phi },

f{ f },

fileName{ path },

solution{ solution }

{

if (numberOfStepsT == -1) {

this->numberOfStepsT = 2. \* numberOfStepsX \* numberOfStepsX \* T;

}

else {

this->numberOfStepsT = numberOfStepsT;

}

gridPitchX = 1. / numberOfStepsX;

gridPitchT = 1. / this->numberOfStepsT;

layers = new double[(this->numberOfStepsT + 1) \* (numberOfStepsX + 1)];

//std::cout << "X " << numberOfStepsX << "\n\n";

//std::cout << "T " << numberOfStepsT << "\n\n";

std::cout << "h = " << gridPitchX << "; t = " << gridPitchT << "\n";

//std::cout << (numberOfStepsT + 1) \* (numberOfStepsX + 1) << "\n\n";

}

void DifferenceScheme::setNumberOfStepsSpace(unsigned numberOfStepsX)

{

this->numberOfStepsX = numberOfStepsX;

numberOfStepsT = 2. \* numberOfStepsX \* numberOfStepsX;

gridPitchX = 1. / numberOfStepsX;

gridPitchT = 1. / numberOfStepsT;

delete[] layers;

std::cout <<"SET h = " << gridPitchX << "; t = " << gridPitchT << "\n";

layers = new double[(numberOfStepsT + 1) \* (numberOfStepsX + 1)];

}

void DifferenceScheme::setNumberOfStepsTimeAndSpace(unsigned numberOfStepsT, double numberOfStepsX)

{

this->numberOfStepsX = numberOfStepsX;

this->numberOfStepsT = numberOfStepsT;

gridPitchT = 1. / numberOfStepsT;

gridPitchX = 1. / numberOfStepsX;

delete[] layers;

std::cout << "SET h = " << gridPitchX << "; t = " << gridPitchT << "\n";

layers = new double[(numberOfStepsT + 1) \* (numberOfStepsX + 1)];

}

DifferenceScheme::~DifferenceScheme()

{

delete[] layers;

}

double DifferenceScheme::error(const std::string& fileName)

{

std::unique\_ptr<double[]> error{ new double[(numberOfStepsT + 1) \* (numberOfStepsX + 1)] };

double max = -1.;

for (unsigned t = 0; t <= numberOfStepsT; ++t) {

for (unsigned x = 0; x <= numberOfStepsX; ++x) {

error[t \* (numberOfStepsX + 1) + x] = fabs(layers[t \* (numberOfStepsX + 1) + x] - solution(gridPitchT \* t, gridPitchX \* x));

if (error[t \* (numberOfStepsX + 1) + x] >= max) {

max = fabs(layers[t \* (numberOfStepsX + 1) + x] - solution(gridPitchT \* t, gridPitchX \* x));

}

}

}

if (!fileName.empty()) {

saveToFile(fileName + "\_" + std::to\_string(numberOfStepsX) + "\_" + std::to\_string(numberOfStepsT) + "Error.txt", error.get());

}

return max;

}

void DifferenceScheme::drawGraph(std::vector<std::pair<std::string, std::string>>&& pathTofiles,

std::string&& graphName, const std::string path, std::string&& labelX, std::string&& labelY, std::string&& labelZ)

{

std::string command = "";

std::string lineUsing = "\" u 1:2:3";

std::string lineName = " title ";

std::string lineWith = " with lines";

command += "set ticslevel 0\n";

command += "splot";

command += "\"" + pathTofiles[0].first + lineUsing + lineName + "\"n = " + pathTofiles[0].second + "\"" + lineWith;

for (unsigned i = 1; i < pathTofiles.size(); ++i) {

command += ", ";

command += "\"" + pathTofiles[i].first + lineUsing + lineName + "\"n = " + pathTofiles[i].second + "\"" + lineWith;

}

command += ";set term wxt title \"" + graphName + "\"";

command += ";set xlabel \"" + labelX + "\"";

command += ";set ylabel \"" + labelY + "\"";

command += ";set zlabel \"" + labelZ + "\"";

std::ofstream graphic("file");

graphic << "cd \"" + path + "\"" << "\n";

graphic << command << "; pause mouse keypress" << "\n";

graphic.close();

std::system("gnuplot -persist file");

graphic.close();

}

void DifferenceScheme::drawGraphNorm(std::vector<std::pair<std::string, std::string>>&& pathTofiles,

std::string&& graphName, const std::string path, std::string&& labelX, std::string&& labelY)

{

std::string lineUsing = "\" using ";

std::string lineName = " title ";

std::string lineWith = " with lines ls ";

std::string ls = " lw 3 ";

int numLine = 1;

std::string command = "";

std::string counter;

command += "\"" + pathTofiles[0].first + lineUsing + "1:2" + lineName + "\"n = " + pathTofiles[0].second + "\"" + lineWith + "1" + ls;

for (unsigned i = 1; i < pathTofiles.size(); ++i) {

command += ", ";

counter = std::to\_string(numLine + 1);

command += "\"" + pathTofiles[i].first + lineUsing + "1:2" + lineName + "\"n = " + pathTofiles[i].second + "\"" + lineWith + counter + ls;

numLine++;

}

command += ";set term wxt title \"" + graphName + "\"";

command += ";set xlabel \"" + labelX + "\"";

command += ";set ylabel \"" + labelY + "\"";

std::ofstream graphic("file");

graphic << "cd \"" + path + "\"" << "\n";

graphic << "plot " << command << "; pause mouse keypress" << "\n";

graphic.close();

std::system("gnuplot -persist file");

graphic.close();

}

//const std::unique\_ptr<double[]> array

void DifferenceScheme::saveToFile(const std::string& fileName, double\* array)

{

std::ofstream fileToWrite(fileName);

for (unsigned t = 0; t <= numberOfStepsT; ++t) {

for (unsigned x = 0; x <= numberOfStepsX; ++x) {

fileToWrite << x \* gridPitchX << "\t" << t \* gridPitchT << "\t" << array[t \* (numberOfStepsX + 1) + x] << "\n";

}

}

fileToWrite.close();

}

ExplicitDifferenceScheme.h

#pragma once

#include "DifferenceScheme.h"

class ExplicitDifferenceScheme : public DifferenceScheme

{

public:

ExplicitDifferenceScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path

);

virtual double calculate(const bool isSaveGraph = false, const bool isSaveError = false) override;

//~ExplicitDifferenceScheme();

private:

};

ExplicitDifferenceScheme.cpp

#include "ExplicitDifferenceScheme.h"

ExplicitDifferenceScheme::ExplicitDifferenceScheme(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path) : DifferenceScheme(

T,

numberOfStepsX,

alpha\_0,

alpha\_1,

beta\_0, beta\_1,

k\_0,

psi\_0,

psi\_1,

phi,

f,

solution,

path

)

{

}

double ExplicitDifferenceScheme::calculate(const bool isSaveGraph, const bool isSaveError)

{

/\*std::cout << gridPitchX << "\t" << gridPitchT << "\n";\*/

auto start = std::chrono::steady\_clock::now();

for (unsigned i = 0; i <= numberOfStepsX; ++i) {

layers[0 \* (numberOfStepsT + 1) + i] = phi(i \* gridPitchX);

}

for (unsigned t = 0; t < numberOfStepsT; ++t) {

for (unsigned x = 1; x < numberOfStepsX; ++x) {

layers[(t + 1) \* (numberOfStepsX + 1) + x] = layers[t \* (numberOfStepsX + 1) + x] +

k\_0 \* gridPitchT / (gridPitchX \* gridPitchX) \*

(layers[(t) \* (numberOfStepsX + 1) + x - 1] - 2. \* layers[t \* (numberOfStepsX + 1) + x] +

layers[t \* (numberOfStepsX + 1) + x + 1]) +

gridPitchT \* f(t \* gridPitchT, x \* gridPitchX);

//std::cout << "1 " << layers[(t + 1) \* (numberOfStepsX + 1) + 0] << "\n";

//std::cout << "2 " << layers[(t + 1) \* (numberOfStepsX + 1) + 1] << "\n";

layers[(t + 1) \* (numberOfStepsX + 1) + 0] = (psi\_0((t + 1) \* gridPitchT) \* gridPitchX - beta\_0 \* layers[(t + 1) \* (numberOfStepsX + 1) + 1]) /

(alpha\_0 \* gridPitchX - beta\_0);

layers[(t + 1) \* (numberOfStepsX + 1) + numberOfStepsX] = (psi\_1((t + 1) \* gridPitchT) \* gridPitchX +

beta\_1 \* layers[(t + 1) \* (numberOfStepsX + 1) + numberOfStepsX - 1]) / (alpha\_1 \* gridPitchX + beta\_1);

}

}

auto end = std::chrono::steady\_clock::now();

std::chrono::duration<double> elapsed\_seconds = end - start;

std::cout << "elapsed time: " << elapsed\_seconds.count() << "s\n";

if (isSaveError) {

return error(fileName);

}

else {

return error();

}

}

FullyImplicitSchema.h

#pragma once

#include "ImplicitSchema.h"

class FullyImplicitSchema : public ImplicitSchema

{

public:

FullyImplicitSchema(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path);

//~FullyImplicitSchema();

virtual double calculate(const bool isSaveGraph = false, const bool isSaveError = false) override;

private:

};

FullyImplicitSchema.cpp

#include "FullyImplicitSchema.h"

FullyImplicitSchema::FullyImplicitSchema(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path) : ImplicitSchema(

T,

numberOfStepsX,

alpha\_0,

alpha\_1,

beta\_0, beta\_1,

k\_0,

psi\_0,

psi\_1,

phi,

f,

solution,

path

)

{

}

double FullyImplicitSchema::calculate(const bool isSaveGraph, const bool isSaveError)

{

auto start = std::chrono::steady\_clock::now();

double\* elementsMatrix = new double[(numberOfStepsX + 1) \* 4]; // 0 - верх диаг.; 1 - ср. диаг.; 2 - нижн. диаг.;

for (unsigned i = 0; i <= numberOfStepsX; ++i) {

layers[0 \* (numberOfStepsT + 1) + i] = phi(i \* gridPitchX);

}

for (unsigned t = 0; t < numberOfStepsT; ++t) {

//верхн.

elementsMatrix[0 \* (numberOfStepsX + 1) + 0] = beta\_0;

//ср.

elementsMatrix[1 \* (numberOfStepsX + 1) + 0] = gridPitchX \* alpha\_0 - beta\_0;

elementsMatrix[1 \* (numberOfStepsX + 1) + numberOfStepsX] = alpha\_1 \* gridPitchX + beta\_1;

//нижн;

elementsMatrix[2 \* (numberOfStepsX + 1) + numberOfStepsX - 1] = -beta\_1;

//прав;

elementsMatrix[3 \* (numberOfStepsX + 1) + 0] = gridPitchX \* psi\_0(gridPitchT \* (t + 1));

elementsMatrix[3 \* (numberOfStepsX + 1) + numberOfStepsX] = gridPitchX \* psi\_1(gridPitchT \* (t + 1));

for (unsigned x = 1; x < numberOfStepsX; ++x) {

//верхн.

elementsMatrix[0 \* (numberOfStepsX + 1) + x] = k\_0 \* gridPitchT / (gridPitchX \* gridPitchX);

//ср.

elementsMatrix[1 \* (numberOfStepsX + 1) + x] = -2. \* gridPitchT \* k\_0 / (gridPitchX \* gridPitchX) - 1;

//нижн;

elementsMatrix[2 \* (numberOfStepsX + 1) + x - 1] = k\_0 \* gridPitchT / (gridPitchX \* gridPitchX);

//прав;

elementsMatrix[3 \* (numberOfStepsX + 1) + x] = -layers[t \* (numberOfStepsX + 1) + x] - gridPitchT \* f(gridPitchT \* (t + 1), gridPitchX \* (x));

}

progonka(elementsMatrix, t + 1);

}

delete[] elementsMatrix;

auto end = std::chrono::steady\_clock::now();

std::chrono::duration<double> elapsed\_seconds = end - start;

std::cout << "elapsed time: " << elapsed\_seconds.count() << "s\n";

if (isSaveError) {

return error(fileName);

}

else {

return error();

}

}

ImplicitSchema.h

#pragma once

#include "DifferenceScheme.h"

class ImplicitSchema : public DifferenceScheme

{

public:

ImplicitSchema(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path,

int numberOfStepsT = -1.

);

//~ImplicitSchema();

void progonka(double\* matrix, unsigned time);

private:

};

ImplicitSchema.cpp

#include "ImplicitSchema.h"

ImplicitSchema::ImplicitSchema(

double T,

unsigned numberOfStepsX,

double alpha\_0,

double alpha\_1,

double beta\_0,

double beta\_1,

double k\_0,

std::function<double(double)> psi\_0,

std::function<double(double)> psi\_1,

std::function<double(double)> phi,

std::function<double(double, double)> f,

std::function<double(double, double)> solution,

const std::string& path,

int numberOfStepsT

) : DifferenceScheme(

T,

numberOfStepsX,

alpha\_0,

alpha\_1,

beta\_0, beta\_1,

k\_0,

psi\_0,

psi\_1,

phi,

f,

solution,

path,

numberOfStepsT

)

{

}

void ImplicitSchema::progonka(double\* matrix, unsigned time)

{

for (unsigned x = 1; x <= numberOfStepsX; x++)

{

matrix[1 \* (numberOfStepsX + 1) + x] = matrix[1 \* (numberOfStepsX + 1) + x] -

matrix[0 \* (numberOfStepsX + 1) + x - 1] \*

(matrix[2 \* (numberOfStepsX + 1) + x - 1] /

matrix[1 \* (numberOfStepsX + 1) + x - 1]);

matrix[3 \* (numberOfStepsX + 1) + x] = matrix[3 \* (numberOfStepsX + 1) + x] -

matrix[3 \* (numberOfStepsX + 1) + x - 1] \*

(matrix[2 \* (numberOfStepsX + 1) + x - 1] /

matrix[1 \* (numberOfStepsX + 1) + x - 1]);

matrix[2 \* (numberOfStepsX + 1) + x - 1] = 0;

}

layers[(numberOfStepsX + 1) \* time + numberOfStepsX] = matrix[3 \* (numberOfStepsX + 1) + numberOfStepsX] / matrix[1 \* (numberOfStepsX + 1) + numberOfStepsX];

for (int x = numberOfStepsX - 1; x >= 0; --x)

layers[(numberOfStepsX + 1) \* time + x] = (matrix[3 \* (numberOfStepsX + 1) + x] -

matrix[0 \* (numberOfStepsX + 1) + x] \*

layers[(numberOfStepsX + 1) \* time + x + 1]) /

matrix[1 \* (numberOfStepsX + 1) + x];

}

Source.cpp

#include <iostream>

#include "ExplicitDifferenceScheme.h"

#include "FullyImplicitSchema.h"

#include "CrankNicholsonScheme.h"

#include "ConservativeScheme.h"

constexpr int k\_0 = 1;

constexpr int alpha\_0 = 0;

constexpr int alpha\_1 = 1;

constexpr int beta\_0 = 1;

constexpr int beta\_1 = 1;

constexpr double T = 1.;

constexpr unsigned numberOfStepsX = 100;

double f(double t, double x);

double phi(double x);

double psi\_0(double t);

double psi\_1(double t);

double K(double u);

double F(double u);

double solution(double t, double x);

double solution\_3(double t, double x);

//#define pathToFiles "C:/Project/USATU\_Lab/TRS\_LAB/Lab\_1/Lab\_2"

#define pathToFiles "C:/Users/hp/OneDrive/Документы/УГАТУ/3 курс/ТРС\_2/Lab\_2"

//void TASK1(bool isDraw = false);

//void TASK2(bool isDraw = false);

void TASK3(bool isDraw = false);

int main() {

//TASK1(true);

//TASK2();

TASK3(true);

return 0;

}

double f(double t, double x)

{

return t \* x \* x - t \* t;

}

double phi(double x)

{

return x;

}

double psi\_0(double t)

{

return 1.;

}

double psi\_1(double t)

{

return 3. / 2. \* t \* t + 2.;

}

double solution(double t, double x)

{

return t \* t \* x \* x / 2. + x;

}

double solution\_3(double t, double x)

{

return t \* t \* x \* x / 2. + x;

}

void TASK1(bool isDraw)

{

ExplicitDifferenceScheme explicitScheme(T, numberOfStepsX, alpha\_0, alpha\_1,

beta\_0, beta\_1, k\_0, psi\_0, psi\_1, phi, f, solution, "explicitSchem");

std::cout << explicitScheme.calculate() << "\n\n";

explicitScheme.setNumberOfStepsSpace(50);

std::cout << explicitScheme.calculate() << "\n\n";

explicitScheme.setNumberOfStepsSpace(100);

std::cout << explicitScheme.calculate() << "\n\n";

//std::ofstream fileToWrite("ExplicitSchemeError\_X.txt");

double error;

std::cout << "\nSTART T = 20000, X = 10, 50, 100;\n";

explicitScheme.setNumberOfStepsTimeAndSpace(20000, 10);

std::cout << explicitScheme.calculate(true, true) << "\n\n";

//error = explicitScheme.calculate();

//std::cout << error << "\n\n";

//fileToWrite << "10\t" << error << "\n";

explicitScheme.setNumberOfStepsTimeAndSpace(20000, 50);

std::cout << explicitScheme.calculate(true, true) << "\n\n";

//error = explicitScheme.calculate();

//std::cout << error << "\n\n";

//fileToWrite << "50\t" << error << "\n";

explicitScheme.setNumberOfStepsTimeAndSpace(20000, 100);

std::cout << explicitScheme.calculate() << "\n\n";

//error = explicitScheme.calculate();

//std::cout << error << "\n\n";

//fileToWrite << "100\t" << error << "\n";

//fileToWrite.close();

std::cout << "\nEND T = 20000, X = 10, 50, 100;\n";

//fileToWrite.open("ExplicitSchemeError\_T.txt");

std::cout << "\nSTART T = 200, 5000, 20000, X = 10\n";

explicitScheme.setNumberOfStepsTimeAndSpace(200, 10);

std::cout << explicitScheme.calculate(true, true) << "\n\n";

//error = explicitScheme.calculate();

//std::cout << error << "\n\n";

//fileToWrite << "200\t" << error << "\n";

explicitScheme.setNumberOfStepsTimeAndSpace(5000, 10);

std::cout << explicitScheme.calculate(true, true) << "\n\n";

//error = explicitScheme.calculate(true, true);

//std::cout << error << "\n\n";

//fileToWrite << "5000\t" << error << "\n";

explicitScheme.setNumberOfStepsTimeAndSpace(20000, 10);

std::cout << explicitScheme.calculate() << "\n\n";

//error = explicitScheme.calculate();

//std::cout << error << "\n\n";

//fileToWrite << "20000\t" << error << "\n";

//fileToWrite.close();

std::cout << "\nEND T = 200, 5000, 20000, X = 10\n";

explicitScheme.setNumberOfStepsTimeAndSpace(10, 10);

std::cout << explicitScheme.calculate() << "\n\n";

if (isDraw) {

explicitScheme.drawGraph({ {"explicitSchem\_10\_20000Error.txt", "10 steps - 20000 times"} }, "temp",

pathToFiles, "X", "T", "error");

explicitScheme.drawGraph({ {"explicitSchem\_50\_20000Error.txt", "50 steps - 20000 times"} }, "temp",

pathToFiles, "X", "T", "error");

explicitScheme.drawGraph({ {"explicitSchem\_10\_200Error.txt", "10 steps - 200 times"} }, "temp",

pathToFiles, "X", "T", "error");

explicitScheme.drawGraph({ {"explicitSchem\_10\_5000Error.txt", "10 steps - 5000 times"} }, "temp",

pathToFiles, "X", "T", "error");

//explicitScheme.drawGraph({ {"explicitSchem\_10\_5000Error.txt", "10 steps - 5000 times"} }, "temp",

// pathToFiles, "X", "T", "error");

//explicitScheme.drawGraphNorm({ {"ExplicitSchemeError\_X.txt", "Change X; Error"} }, "t", pathToFiles, "T", "Error");

//explicitScheme.drawGraphNorm({ {"ExplicitSchemeError\_T.txt", "Change T; Error"} }, "t", pathToFiles, "T", "Error");

}

}

void TASK2(bool isDraw)

{

std::cout << "\n\n\----------------------\n\n\n";

std::cout << "fullyImplicitSchema\n\n";

FullyImplicitSchema fullyImplicitSchema(T, numberOfStepsX, alpha\_0, alpha\_1,

beta\_0, beta\_1, k\_0, psi\_0, psi\_1, phi, f, solution, "fullyImplicitSchema");

std::cout << fullyImplicitSchema.calculate() << "\n\n";

fullyImplicitSchema.setNumberOfStepsSpace(50);

std::cout << fullyImplicitSchema.calculate(true, true) << "\n\n";

fullyImplicitSchema.setNumberOfStepsSpace(100);

std::cout << fullyImplicitSchema.calculate() << "\n\n";

std::cout << "\n\n -------------------------------- \n";

std::cout << "crankNicholsonScheme\n\n";

CrankNicholsonScheme crankNicholsonScheme(T, numberOfStepsX, alpha\_0, alpha\_1,

beta\_0, beta\_1, k\_0, psi\_0, psi\_1, phi, f, solution, "crankNicholsonScheme");

std::cout << crankNicholsonScheme.calculate() << "\n\n";

crankNicholsonScheme.setNumberOfStepsSpace(50);

std::cout << crankNicholsonScheme.calculate(true, true) << "\n\n";

crankNicholsonScheme.setNumberOfStepsSpace(100);

std::cout << crankNicholsonScheme.calculate() << "\n\n";

if (isDraw) {

fullyImplicitSchema.drawGraph({{"fullyImplicitSchema\_50\_5000Error.txt", "FI 50 steps - 5000 times"}

}, "temp",

pathToFiles, "X", "T", "error");

crankNicholsonScheme.drawGraph({ {"crankNicholsonScheme\_50\_5000Error.txt", "CN 50 steps - 5000 times"}

}, "temp",

pathToFiles, "X", "T", "error");

}

}

void TASK3(bool isDraw)

{

std::cout << "\n\n -------------------------------- \n";

std::cout << "CHECK SCHEME ON TASK\_1 START\n\n";

ConservativeScheme conservativeSchemeCHECK(T, numberOfStepsX, alpha\_0, alpha\_1,

beta\_0, beta\_1, k\_0, psi\_0, psi\_1, phi, f, solution, [](double u) {return 1; }, [](double u) {return 1; }, "conservativeSchemeCHECK");

std::cout << conservativeSchemeCHECK.calculate() << "\n";

conservativeSchemeCHECK.setNumberOfStepsSpace(50);

std::cout << conservativeSchemeCHECK.calculate() << "\n";

conservativeSchemeCHECK.setNumberOfStepsSpace(100);

std::cout << conservativeSchemeCHECK.calculate() << "\n";

std::cout << "\nCHECK SCHEME ON TASK\_1 END\n\n\n";

std::cout << "START TASK\_3. Conservative Scheme\n\n";

ConservativeScheme conservativeScheme(T, numberOfStepsX, alpha\_0, alpha\_1,

beta\_0, beta\_1, k\_0, psi\_0, psi\_1, phi, f, solution\_3, F, K, "conservativeScheme");

std::cout << conservativeScheme.calculate() << "\n";

conservativeScheme.setNumberOfStepsSpace(50);

std::cout << conservativeScheme.calculate(true) << "\n";

conservativeScheme.setNumberOfStepsSpace(70);

std::cout << conservativeScheme.calculate(true) << "\n";

std::cout << "END TASK\_3. Conservative Scheme\n\n\n";

if (isDraw) {

conservativeScheme.drawGraph({ {"conservativeScheme\_50\_5000Solution.txt", "conservative 50 steps - 5000 times"}

}, "temp",

pathToFiles, "X", "T", "u");

conservativeScheme.drawGraph({ {"conservativeScheme\_70\_9800Solution.txt", "conservative 70 steps - 9800 times"}

}, "temp",

pathToFiles, "X", "T", "u");

}

//conservativeScheme.drawGraph({ {"fullyImplicitSchema\_10\_200Error.txt", "FI"}, {"crankNicholsonScheme\_10\_200Error.txt", "CN"} }, "temp",

//pathToFiles, "X", "T", "error");

}

double K(double u)

{

return sin(u);

//return 1.;

}

double F(double u)

{

return u;

//return 1.;

}