Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Уравнения математической физики
Курсовая работа
Тема: Решение двумерной гармонической задачи при помощи четырёхслойной неявной схемы. Базисные функции билинейные

Факультет: ФПМИ
Группа: ПМ-63
Студент: Кожекин М.В.
Вариант: 70, 9

Новосибирск
2019

1. Цель работы

Приобрести навыки численного решения начально-краевых задач для уравнений гиперболического и параболического типа в неоднородных одномерных, двумерных и трехмерных областях с помощью метода конечных элементов при использовании различных схем дискретизации по времени.

2. Задание

Разработать программу решения двумерной гиперболической задачи методом конечных элементов. Сравнить прямой и итерационные методы решения получаемой в результате конечноэлементной аппроксимации СЛАУ.

Вариант 70, 9: Решить двумерную гиперболическую задачу в декартовых координатах, базисные функции - билинейные, четырёхслойная неявная схема.

3. Анализ

3.1. Постановка задачи

Дано гиперболическое уравнение в декартовой системе координат:

$$div(\lambda gradu) + \gamma u + \sigma \frac{du}{dt} + \chi \frac{d^2u}{dt^2} = f$$

3.2. Дискретизация по времени

Представим искомое решение u на интервале (t_{j-3}, t_j) :

$$\mathrm{u(x,y,t)} = \mathrm{u}^{j-3}\eta_3^j(t) + u^{j-2}\eta_2^j(t) + u^{j-1}\eta_1^j(t) + u^{j-0}\eta_0^j(t)$$

где функции $\eta_\nu^j(t)$ являются базисными кубическими полиномами Лагранжа и имеют следующий вид

$$\eta_3^j(t) = \frac{(t-t_{j-2})(t-t_{j-1})(t-t_j)}{(t_{j-3}-t_{j-2})(t_{j-3}-t_{j-1})(t_{j-3}-t_j)}$$

$$\eta_2^j(t) = \frac{(t-t_{j-3})(t-t_{j-1})(t-t_j)}{(t_{j-2}-t_{j-3})(t_{j-2}-t_{j-1})(t_{j-2}-t_j)}$$

$$\eta_1^j(t) = \frac{(t-t_{j-2})(t-t_{j-2})(t-t_j)}{(t_{j-1}-t_{j-3})(t_{j-1}-t_{j-2})(t_{j-1}-t_j)}$$

$$\eta_0^j(t) = \frac{(t-t_{j-3})(t-t_{j-2})(t-t_{j-1})}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}$$

Возьмём первые и вторые производные от полиномов Лагранжа в точке $t = t_j$ (т.к. схема неявная)

	полином Лагранжа	1ая производная	2ая производная
$\eta_3^j(t)$	$\frac{t_{02}t_{01}t_{00}}{t_{23}t_{13}t_{03}}$	$-\frac{t_{01}t_{02}}{t_{23}t_{13}t_{03}}$	$-2\cdot\frac{t_{01}+t_{02}}{t_{23}t_{13}t_{03}}$
$\eta_2^j(t)$	$\frac{t_{03}t_{01}t_{00}}{t_{23}t_{12}t_{02}}$	$\frac{t_{01}t_{03}}{t_{23}t_{12}t_{02}}$	$2\cdot\frac{t_{01}+t_{03}}{t_{23}t_{12}t_{02}}$
$\eta_1^j(t)$	$\frac{t_{03}t_{02}t_{00}}{t_{13}t_{12}t_{01}}$	$-\frac{t_{02}t_{03}}{t_{13}t_{12}t_{01}}$	$-2\cdot\frac{t_{02}+t_{03}}{t_{13}t_{12}t_{01}}$
$\eta_0^j(t)$	$\frac{t_{03}t_{02}t_{01}}{t_{03}t_{02}t_{01}}$	$\frac{t_{01}t_{02}+t_{01}t_{03}+t_{02}t_{03}}{t_{03}t_{02}t_{01}}$	$2\cdot\frac{t_{01}+t_{02}+t_{03}}{t_{03}t_{02}t_{01}}$

где:

$t_{01} = t_0 - t_1, t_0 = t_j, t_1 = t_{j-1},$

$t_{02} = t_0 - t_2, t_0 = t_j, t_2 = t_{j-2},$
...

Подставим их в исходное уравнение, а затем выведем из него 4-х слойную неявную схему:

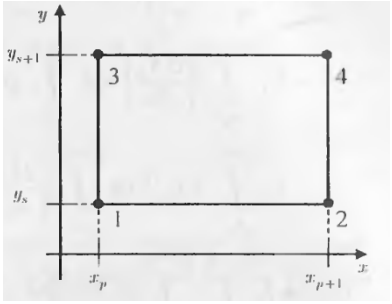
$$\left(\left[2\chi\frac{t_{01}+t_{02}+t_{03}}{t_{03}t_{02}t_{01}}+\sigma\frac{t_{01}t_{02}+t_{01}t_{03}+t_{02}t_{03}}{t_{03}t_{02}t_{01}}+\gamma\right]M+G\right)q^j=b^j$$
$$+\left[2\chi\frac{t_{01}+t_{02}}{t_{03}t_{13}t_{23}}+\sigma\frac{t_{01}t_{02}}{t_{03}t_{13}t_{23}}\right]Mq^{j-3}$$
$$-\left[2\chi\frac{t_{01}+t_{t03}}{t_{02}t_{12}t_{23}}+\sigma\frac{t_{01}t_{t03}}{t_{02}t_{12}t_{23}}\right]Mq^{j-2}$$
$$+\left[2\chi\frac{t_{02}+t_{03}}{t_{01}t_{12}t_{13}}+\sigma\frac{t_{02}+t_{03}}{t_{01}t_{12}t_{13}}\right]Mq^{j-1}$$

3.3. Конечноэлементная дискретизация

Формулы для билинейных базисных функций прямоугольных элементов:

$$X_1(x)=\frac{x_{p+1}-x}{h_x} \qquad h_x = x_{p+1}-x_p$$
$$X_2(x)=\frac{x-x_p}{h_x} \qquad h_y = y_{s+1}-x_s$$
$$Y_1(y)=\frac{y_{s+1}-y}{h_y} \qquad x\in[x_p,x_{p+1}],\ y\in[y_s,y_{s+1}]$$
$$Y_2(y)=\frac{y-y_s}{h_y} \qquad \Omega_{ps}=[x_p,x_{p+1}]\times[y_s,y_{s+1}]$$

Каждый из базисных функций равна единице в соответсвующем узле и нулю во всех остальных. Например, функция $\psi_1(x,y)=1$ в узле (x_p, x_s).



3.4. Локальные матрицы и вектора

Аналитические выражения для вычисления элементов локальных матриц:

$$G_{ij} = \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} \lambda \left(\frac{\psi_i}{x} \frac{\psi_j}{x} + \frac{\psi_i}{y} \frac{\psi_j}{y} \right) dxdy$$

$$M_{ij}^{\gamma} = \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} \gamma \psi_i \psi_j dxdy$$

$$b_i = \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} f \psi_i x dy$$

$$G = \frac{\lambda}{6} \frac{h_y}{h_x} \begin{pmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{pmatrix} + \frac{\lambda}{6} \frac{h_x}{h_y} \begin{pmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & 2 & 1 & 2 \end{pmatrix}$$

$$M = \frac{h_x h_y}{36} \begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix}$$

$$b = \frac{h_x h_y}{36} \begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}$$

3.5. Решатели

Для решения полученных СЛАУ использовался метод Гаусса.

4. Исследования

Проверим сходимость метода на разных функциях.

В ходе следующего исследования использовались следующие параметры:
$\lambda = \gamma = \sigma = \chi = 1$
Область пространства $\Omega = [0, 1] * [0, 1]$
Время задано на отрезке [0, 1]
Первоначальное число узлов 36, а конечных элементов 25
Для неравномерных сеток по времени и пространству коэффициент k=1.2

Равномерная сетка по пространству
Равномерная сетка по времени

$space(x,y)$ \ $time(t)$	1	t	t^2	t^3	t^4	t^5	$sin(t)$	e^t
1	4.66e-17	2.51e-13	1.05e-12	7.36e-13	7.23e-03	2.29e-02	1.76e-04	5.75e-04
$x+y$	1.06e-12	1.14e-12	1.59e-12	1.61e-12	7.23e-03	2.29e-02	1.76e-04	5.75e-04
x^2+y^2	6.64e-13	3.58e-13	5.08e-13	3.99e-13	7.23e-03	2.29e-02	1.76e-04	5.75e-04
x^3+y^3	7.60e-13	3.20e-13	6.71e-13	6.54e-13	7.23e-03	2.29e-02	1.76e-04	5.75e-04
x^4+y^4	4.34e-04	4.34e-04	4.34e-04	4.34e-04	6.80e-03	2.25e-02	2.58e-04	1.42e-04
x^5+y^5	1.11e-03	1.11e-03	1.11e-03	1.11e-03	6.15e-03	2.18e-02	9.39e-04	5.62e-04
$sin(x)+sin(y)$	8.61e-06	8.61e-06	8.61e-06	8.61e-06	7.22e-03	2.29e-02	1.67e-04	5.67e-04
e^x+e^y	3.08e-05	3.08e-05	3.08e-05	3.08e-05	7.20e-03	2.29e-02	1.45e-04	5.45e-04

Равномерная сетка по пространству
Неравномерная сетка по времени

$space(x,y)$ \ $time(t)$	1	t	t^2	t^3	t^4	t^5	$sin(t)$	e^t
1	8.72e-17	2.74e-13	1.14e-12	8.03e-13	9.18e-03	2.81e-02	2.17e-04	7.16e-04
$x+y$	1.13e-12	1.23e-12	1.73e-12	1.74e-12	9.18e-03	2.81e-02	2.17e-04	7.16e-04
x^2+y^2	7.09e-13	3.77e-13	5.48e-13	4.28e-13	9.18e-03	2.81e-02	2.17e-04	7.16e-04
x^3+y^3	8.06e-13	3.35e-13	7.20e-13	6.95e-13	9.18e-03	2.81e-02	2.17e-04	7.16e-04
x^4+y^4	4.73e-04	4.73e-04	4.73e-04	4.73e-04	8.70e-03	2.76e-02	2.57e-04	2.43e-04
x^5+y^5	1.21e-03	1.21e-03	1.21e-03	1.21e-03	8.00e-03	2.69e-02	9.98e-04	5.29e-04
$sin(x)+sin(y)$	9.38e-06	9.38e-06	9.38e-06	9.38e-06	9.17e-03	2.81e-02	2.07e-04	7.07e-04
e^x+e^y	3.36e-05	3.36e-05	3.36e-05	3.36e-05	9.14e-03	2.81e-02	1.83e-04	6.83e-04

Неравномерная сетка по пространству
Равномерная сетка по времени

$space(x,y)$ \ $time(t)$	1	t	t^2	t^3	t^4	t^5	$sin(t)$	e^t
1	4.66e-17	2.44e-13	1.02e-12	7.15e-13	7.02e-03	2.22e-02	1.71e-04	5.59e-04
$x+y$	7.33e-13	6.45e-13	1.14e-12	1.20e-12	7.02e-03	2.22e-02	1.71e-04	5.59e-04
x^2+y^2	6.48e-13	5.01e-13	8.87e-13	1.07e-12	7.02e-03	2.22e-02	1.71e-04	5.59e-04
x^3+y^3	6.06e-13	4.23e-13	7.52e-13	6.24e-13	7.02e-03	2.22e-02	1.71e-04	5.59e-04
x^4+y^4	4.90e-04	4.90e-04	4.90e-04	4.90e-04	6.54e-03	2.18e-02	3.23e-04	1.18e-04
x^5+y^5	1.46e-03	1.46e-03	1.46e-03	1.46e-03	5.68e-03	2.09e-02	1.30e-03	9.50e-04
$sin(x)+sin(y)$	1.12e-05	1.12e-05	1.12e-05	1.12e-05	7.01e-03	2.22e-02	1.60e-04	5.49e-04
e^x+e^y	3.77e-05	3.77e-05	3.77e-05	3.77e-05	6.99e-03	2.22e-02	1.35e-04	5.23e-04

Неравномерная сетка по пространству
Неравномерная сетка по времени

$space(x,y)$ \ $time(t)$	1	t	t^2	t^3	t^4	t^5	$sin(t)$	e^t
1	8.46e-17	2.66e-13	1.11e-12	7.80e-13	8.91e-03	2.73e-02	2.10e-04	6.95e-04
$x+y$	7.77e-13	6.90e-13	1.24e-12	1.29e-12	8.91e-03	2.73e-02	2.10e-04	6.95e-04
x^2+y^2	6.88e-13	5.42e-13	9.67e-13	1.17e-12	8.91e-03	2.73e-02	2.10e-04	6.95e-04
x^3+y^3	6.52e-13	4.46e-13	8.14e-13	6.66e-13	8.91e-03	2.73e-02	2.10e-04	6.95e-04
x^4+y^4	5.34e-04	5.34e-04	5.34e-04	5.34e-04	8.39e-03	2.68e-02	3.29e-04	1.93e-04
x^5+y^5	1.58e-03	1.58e-03	1.58e-03	1.58e-03	7.43e-03	2.58e-02	1.39e-03	9.55e-04
$sin(x)+sin(y)$	1.21e-05	1.21e-05	1.21e-05	1.21e-05	8.90e-03	2.73e-02	1.99e-04	6.84e-04
e^x+e^y	4.11e-05	4.11e-05	4.11e-05	4.11e-05	8.87e-03	2.73e-02	1.71e-04	6.56e-04

4.1. Вывод

Если порядок полинома по пространству не превышает порядка используемых базисных функций, а порядок полинома по времени не соответсвует порядку точности используемой временной схемы, то получаемое численное решение должно полностью совпадать с точным решением задачи.

5. Точность решения при дроблении сетки

В ходе следующего исследования использовались следующие параметры:
$\lambda = \gamma = \sigma = \chi = 1$
Область пространства $\Omega = [0, 1] * [0, 1]$
Время задано на отрезке [0, 1]
Первоначальное число узлов 66, а конечных элементов 25
Для неравномерных сеток по времени и пространству коэффициент k=1.2

$u = 1$

время \ пространство	равномерное			не равномерное		
	i	nodes	norm	i	nodes	norm
равномерное	0	36	4.657e-17	0	36	4.657e-17
	1	121	8.182e-17	1	121	8.922e-17
	2	441	1.064e-16	2	441	1.173e-16
	i	nodes	norm	i	nodes	norm
не равномерное	0	36	8.723e-17	0	36	8.457e-17
	1	121	1.923e-16	1	121	2.150e-16
	2	441	4.370e-16	2	441	1.088e-16

$u = x$

время \ пространство	равномерное			не равномерное		
	i	nodes	norm	i	nodes	norm
равномерное	0	36	1.138e-12	0	36	6.446e-13
	1	121	3.605e-13	1	121	1.709e-13
	2	441	1.405e-13	2	441	1.704e-13
	i	nodes	norm	i	nodes	norm
не равномерное	0	36	1.230e-12	0	36	6.897e-13
	1	121	3.321e-13	1	121	1.307e-13
	2	441	1.609e-13	2	441	9.378e-14

$u = x^2$

время \ пространство	равномерное			не равномерное		
	i	nodes	norm	i	nodes	norm
равномерное	0	36	5.076e-13	0	36	8.871e-13
	1	121	1.711e-13	1	121	1.662e-13
	2	441	1.639e-13	2	441	1.234e-13
	i	nodes	norm	i	nodes	norm
не равномерное	0	36	5.477e-13	0	36	9.672e-13
	1	121	2.510e-13	1	121	2.951e-13
	2	441	8.165e-14	2	441	5.299e-14

$u = x^3$

время \ пространство	равномерное			не равномерное		
	i	nodes	norm	i	nodes	norm
равномерное	0	36	6.542e-13	0	36	6.242e-13
	1	121	1.650e-13	1	121	1.944e-13
	2	441	5.553e-14	2	441	1.247e-13
	i	nodes	norm	i	nodes	norm
не равномерное	0	36	6.955e-13	0	36	6.655e-13
	1	121	2.500e-13	1	121	2.611e-13
	2	441	4.195e-13	2	441	7.886e-14

$u = x^4$

пространство / время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	6.795e-03	0	36	6.543e-03
	1	121	2.654e-03	1	121	1.407e-03
	2	441	4.194e-04	2	441	9.653e-05
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	8.702e-03	0	36	8.385e-03
	1	121	4.801e-03	1	121	2.801e-03
	2	441	1.448e-03	2	441	5.921e-04

$u = x^5$

пространство / время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	2.181e-02	0	36	2.088e-02
	1	121	7.219e-03	1	121	3.688e-03
	2	441	1.238e-03	2	441	2.677e-04
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	2.693e-02	0	36	2.581e-02
	1	121	1.334e-02	1	121	7.628e-03
	2	441	4.358e-03	2	441	1.782e-03

$u = sin(x)$

пространство / время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	1.674e-04	0	36	1.603e-04
	1	121	5.624e-05	1	121	2.887e-05
	2	441	9.591e-06	2	441	2.098e-06
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	2.075e-04	0	36	1.989e-04
	1	121	1.034e-04	1	121	5.929e-05
	2	441	3.361e-05	2	441	1.375e-05

$u = e^x$

пространство / время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	5.449e-04	0	36	5.229e-04
	1	121	1.938e-04	1	121	1.005e-04
	2	441	3.195e-05	2	441	7.038e-06
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	6.828e-04	0	36	6.558e-04
	1	121	3.559e-04	1	121	2.052e-04
	2	441	1.120e-04	2	441	4.573e-05

5.1. Вывод

Т.к. **порядок сходимости** - это степень того, насколько сильно увеличивается точность при дроблении сетки. Он определяется из степени x.

Исходя из исследований можно заметить, что порядок сходимости второй.

6. Исходный код программы

head.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
```

```cpp
#include <iomanip>
#include <functional>
#include <cmath>

using namespace std;


typedef std::function<double(double)> function1D;
typedef std::function<double(double, double)> function2D;
typedef std::function<double(double, double, double)> function3D;

typedef vector <double> vector1D;
typedef vector <vector <double>> matrix2D;


// Сравнение векторов
inline bool operator==(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    for (int i = 0; i < a.size(); ++i)
        if (a[i] != b[i])
            return false;

    return true;
}

// Сложение векторов
inline vector1D operator+(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    vector1D result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] += b[i];
    return result;
}
// Сложение матриц
inline matrix2D operator+(const matrix2D& a, const matrix2D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    matrix2D result = a;
    for (int i = 0; i < b.size(); i++)
        for (int j = 0; j < b.size(); j++)
            result[i][j] += b[i][j];
    return result;
}


// Деление матрицы на число
inline matrix2D operator/(const matrix2D& a, const double& b) {

    matrix2D result = a;
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i][j] /= b;
```

```
    return result;
}


// Вычитание векторов
inline vector1D operator-(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    vector1D result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] -= b[i];
    return result;
}
// Обратный знак вектора
inline vector1D operator-(const vector1D& a) {
    vector1D result = a;
    for (int i = 0; i < a.size(); i++)
        result[i] = -result[i];
    return result;
}



// Умножение матрицы на вектор
inline vector1D operator*(const matrix2D& a, const vector1D& b) {
    vector1D result;
    result.resize(b.size(), 0);
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i] += a[i][j] * b[j];
    return result;
}

// Умножение матрицы на вектор
inline matrix2D operator*(const matrix2D& a, double b) {
    matrix2D result = a;
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i][j] *= b;
    return result;
}
// Умножение на число
inline matrix2D operator*(double b, const matrix2D& a) {
    return operator*(a, b);
}


// Умножение на число
inline vector1D operator*(const vector1D& a, double b) {
    vector1D result = a;
    for (int i = 0; i < result.size(); i++)
        result[i] *= b;
    return result;
}
// Умножение на число
inline vector1D operator*(double b, const vector1D& a) {
    return operator*(a, b);
}
```

```
127
128
129
130 // Деление на число
131 inline vector1D operator/(const vector1D& a, double b) {
132     vector1D result = a;
133     for (int i = 0; i < result.size(); i++)
134         result[i] /= b;
135     return result;
136 }
137 // Деление на число
138 inline vector1D operator/(double b, const vector1D& a) {
139     return operator/(a, b);
140 }
141
142
143
144 // Скалярное произведение
145 inline double operator*(const vector1D& a, const vector1D& b) {
146 #ifdef _DEBUG
147     if (a.size() != b.size())
148         throw std::exception();
149 #endif
150     double sum = 0;
151     for (int i = 0; i < a.size(); i++)
152         sum += a[i] * b[i];
153     return sum;
154 }
155
156
157 // Потоковый вывод вектора
158 inline std::ostream& operator<<(std::ostream& out, const vector1D& v) {
159     for (int i = 0; i < v.size() - 1; ++i)
160         out << v[i] << ", ";
161     out << v.back();
162     return out;
163 }
164 // Потоковый вывод матрицы
165 inline std::ostream& operator<<(std::ostream& out, const matrix2D& v) {
166     for (int i = 0; i < v.size() - 1; ++i)
167         out << v[i] << "  ";
168     out << v.back();
169     return out;
170 }
171
172
173 // Потоковый вывод вектора для TeX
174 inline void printTeXVector(std::ofstream &fout, const vector1D &v, int coefGrid)
      {
175     fout << "$(";
176     for (int i = 0; i < v.size() - 1; ++i)
177         if (i % int(pow(2, coefGrid)) == 0)
178             fout << v[i] << ", ";
179     fout << v.back() << ")^T$";
180 }
```

grid.h

```
1 #pragma once
2 #include "head.h"
3
4
```

```
struct NODE {

    bool isFirstNode = false;
    int i, j;
    double x, y;
    int type = -9000;        // -9000     начение при инициализации
                             // -1        фиктивный узел
                             // 0         внутренний узел
                             // n         номер границы
    int border;              // номер границы

    void setNodesData(double _x, double _y, int _i, int _j, int _type, double
    _coef) {
        x = _x;
        y = _y;
        i = _i;
        j = _j;
        type = _type;
        if (i % int(pow(2, _coef)) == 0 && j % int(pow(2.0, _coef)) == 0)
            isFirstNode = true;
    }
};


class GRID
{
public:
    void inputGrid();
    void inputTime();
    void buildGrid();
    void buildTimeGrid();
    void showGrid();
    void saveGridAndBorder(const string &filepathGrid, const string &
    filepathGridBorder);

protected:

    int coefGrid, // Сколько раз дробили сетку по пространству
        coefTime; // Сколько раз дробили сетку по времени

    // Пространство
    int width;
    double xLeft, xRight;
    double hx, nx, kx;
    double dx;

    int heigth;
    double yLower, yUpper;
    double hy, ny, ky;
    double dy;

    bool isGridUniform;
    int nodesCount, finiteElementsCount;

    // Время
    bool isTimeUniform;
    int tCount;
    double tFirst, tLast;
    double ht, nt, kt;
    double dt;
```

```
63
64      // Узлы
65      vector <NODE> nodes;
66      vector1D times;
67  };
```

grid.cpp

```
1   #include "grid.h"
2
3
4   void GRID::inputGrid()
5   {
6       string filepath;
7       if (isGridUniform)
8           filepath = "input/uniform_grid.txt";
9       else
10          filepath = "input/nonuniform_grid.txt";
11
12      std::ifstream fin(filepath);
13      fin >> xLeft >> xRight
14          >> yLower >> yUpper;
15
16      fin >> width >> heigth;
17      if (!isGridUniform) {
18          fin >> kx >> ky;
19          nx = width - 1;
20          ny = heigth - 1;
21      }
22      fin.close();
23  }
24
25
26  void GRID::inputTime()
27  {
28      string filepath;
29      if (isTimeUniform)
30          filepath = "input/uniform_time.txt";
31      else
32          filepath = "input/nonuniform_time.txt";
33
34      std::ifstream fin(filepath);
35      fin >> tFirst >> tLast;
36      fin >> tCount;
37      if (!isTimeUniform) {
38          fin >> kt;
39          nt = tCount - 1;
40      }
41      fin.close();
42  }
43
44
45  void GRID::buildGrid()
46  {
47      //  xLeft          xRight
48      //    *--------------*
49      //    0    width     1
50      if (isGridUniform) {
51          hx = ((xRight - xLeft) / double(width - 1)) / pow(2, coefGrid);
52          hy = ((yUpper - yLower) / double(heigth - 1)) / pow(2, coefGrid);
53          if (coefGrid != 0) {
54              width = (width - 1) * pow(2, coefGrid) + 1;
```

```
            heigth = (heigth - 1) * pow(2, coefGrid) + 1;
        }
    }
    else {
        if (coefGrid != 0) {
            width = (width - 1) * pow(2, coefGrid) + 1;
            heigth = (heigth - 1) * pow(2, coefGrid) + 1;
            nx *= pow(2, coefGrid);
            ny *= pow(2, coefGrid);
            kx *= pow(kx, 1.0 / coefGrid);
            ky *= pow(ky, 1.0 / coefGrid);
        }
        hx = (xRight - xLeft) * (1 - kx) / (1 - pow(kx, nx));
        hy = (yUpper - yLower) * (1 - ky) / (1 - pow(ky, ny));
    }


    nodesCount = width * heigth;
    finiteElementsCount = (width - 1) * (heigth - 1);
    nodes.resize(nodesCount);

    if (isGridUniform) {

        size_t i, j, elem;
        double x, y;

        // Внутренние узлы
        for (size_t j = 1; j < heigth - 1; j++)
        {
            i = 1;
            for (size_t elem = j * width + 1; elem < (j + 1) * width - 1; elem
++, i++)
            {
                x = xLeft + hx * i;
                y = yLower + hy * j;
                nodes[elem].setNodesData(x, y, i, j, 0, coefGrid);
                nodes[elem].border = 0;
            }
        }


        // 1 Нижняя граница
        i = 0;
        j = 0;
        y = yLower;
        for (size_t elem = 0; elem < width; elem++, i++)
        {
            x = xLeft + hx * i;
            nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
            nodes[elem].border = 1;
        }


        // 2 Правая граница
        i = width - 1;
        j = 1;
        x = xRight;
        for (size_t elem = 2 * width - 1; elem < width * heigth; elem += width,
j++)
        {
```

```
            y = yLower + hy * j;
            nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
            nodes[elem].border = 2;
        }


        // 3 Верхняя граница
        i = 0;
        j = heigth - 1;
        y = yUpper;
        for (size_t elem = width * j; elem < (j + 1) * width - 1; elem++, i++)
        {
            x = xLeft + hx * i;
            nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
            nodes[elem].border = 3;
        }


        // 4 Левая граница
        i = 0;
        j = 1;
        x = xLeft;
        for (size_t elem = width; elem < (heigth - 1) * width; elem += width, j
    ++)
        {
            y = yLower + hy * j;
            nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
            nodes[elem].border = 4;
        }

    }
    else {

        size_t i, j, elem;
        double x, y;

        // Внутренние узлы
        dy = hy * ky;
        y = yLower + hy;
        for (size_t j = 1; j < heigth - 1; j++, dy *= ky)
        {
            i = 1;
            dx = hx * kx;
            x = xLeft + hx;
            for (size_t elem = j * width + 1; elem < (j + 1) * width - 1; elem
    ++, i++, dx *= kx)
            {
                nodes[elem].setNodesData(x, y, i, j, 0, coefGrid);
                nodes[elem].border = 0;
                x += dx;
            }
            y += dy;
        }


        // 1 Нижняя граница
        i = 0;
        dx = hx;
        x = xLeft;

```

```
171         j = 0;
172         y = yLower;
173         for (size_t elem = 0; elem < width; elem++, i++, dx *= kx)
174         {
175             nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
176             nodes[elem].border = 1;
177             x += dx;
178         }
179 
180 
181         // 2 Правая граница
182         i = width - 1;
183         x = xRight;
184 
185         j = 1;
186         dy = hy * ky;
187         y = yLower + hy;
188 
189         for (size_t elem = 2 * width - 1; elem < width * heigth; elem += width,
    j++, dy *= ky)
190         {
191             nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
192             nodes[elem].border = 2;
193             y += dy;
194         }
195 
196 
197         // 3 Верхняя граница
198         i = 0;
199         dx = hx;
200         x = xLeft;
201 
202         j = heigth - 1;
203         y = yUpper;
204         for (size_t elem = width * j; elem < (j + 1) * width - 1; elem++, i++,
    dx *= kx)
205         {
206             nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
207             nodes[elem].border = 3;
208             x += dx;
209         }
210 
211 
212         // 4 Левая граница
213         i = 0;
214         x = xLeft;
215 
216         j = 1;
217         dy = hy * ky;
218         y = yLower + hy;
219 
220         for (size_t elem = width; elem < (heigth - 1) * width; elem += width, j
    ++, dy *= ky)
221         {
222             nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
223             nodes[elem].border = 4;
224             y += dy;
225         }
226     }
227 }
```

```
228
229
230 void GRID::buildTimeGrid()
231 {
232     //  tFirst          tLast
233     //    *--------------*
234     //    0    tCount    1
235
236
237     if (isTimeUniform) {
238
239         ht = ((tLast - tFirst) / double(tCount - 1)) / pow(2, coefTime);
240         if (coefTime != 0)
241             tCount = (tCount - 1) * pow(2, coefTime) + 1;
242
243         times.resize(tCount);
244         size_t i, elem;
245         double t;
246         // Первый элемент
247         times[0] = tFirst;
248         i = 1;
249         for (elem = 1; elem < tCount; elem++, i++)
250             times[elem] = tFirst + ht * i;
251
252         // Последний элемент
253         times[tCount - 1] = tLast;
254     }
255
256     else {
257
258         if (coefTime != 0) {
259             tCount = (tCount - 1) * pow(2, coefTime) + 1;
260             nt *= pow(2, coefTime);
261             kt *= pow(kt, 1.0 / coefTime);
262         }
263         times.resize(tCount);
264         ht = (tLast - tFirst) * (1 - kt) / (1 - pow(kt, nt));
265         double t;
266         size_t i, elem;
267         i = 1;
268         dt = ht * kt;
269         t = tFirst + ht;
270         // Первый элемент
271         times[0] = tFirst;
272         for (elem = 1; elem < tCount; elem++, i++, dt *= kt)
273         {
274             times[elem] = t;
275             t += dt;
276         }
277         // Последний элемент
278         times[tCount - 1] = tLast;
279     }
280 }
281
282
283 // Отображние сетки на экран
284 void GRID::showGrid() {
285
286     for (size_t i = 0; i < width; i++)
287         cout << nodes[i].x << " ";
```

```
}


// Сохранение внутренних и внешних узлов в 2 файлах
void GRID::saveGridAndBorder(const string &filepathGrid, const string &
    filepathGridBorder) {

    ofstream grid(filepathGrid);
    ofstream border(filepathGridBorder);
    for (size_t i = 0; i < nodesCount; i++)
    {
        if (nodes[i].type > 0)
            border << nodes[i].x << " " << nodes[i].y << endl;
        else
            grid << nodes[i].x << " " << nodes[i].y << endl;
    }

    border.close();
    grid.close();
}
```

fem.h

```
#pragma once
#include "head.h"
#include "grid.h"
#include "solver.h"


class FEM : public GRID, public SOLVER {
public:

    void init(
        function3D _u,
        function3D _f,
        double _lambda,
        double _gamma,
        double _sigma,
        double _chi,
        bool _isGridUniform,
        bool _isTimeUniform,
        int _coefGrid,
        int _coefTime
    );
    double solve();
    inline int getNodesCount() { return nodesCount; }
    void convAToDense();
    void outputA();
    void outputG();
    void outputM();


protected:
    double  p00, p01, p10, p11,
            c00, c01, c10, c11;
    function3D u, f;
    double lambda, gamma, sigma, chi;
    double t, dt;
    double t0, t1, t2, t3;
    double  t01, t02, t03,
            t23, t12, t13;
    double d1, d2, m1, m2;
```

```
40
41      double calcNormAtMainNodes(const vector1D &x, int time) {
42          double normE = 0;
43          for (size_t i = 0; i < x.size(); i++)
44              normE += pow((x[i] - u(nodes[i].x, nodes[i].y, time)), 2);
45          return sqrt(normE) / nodes.size();
46      }
47
48      void implicitCheme4(int timeLayer);
49
50      void buildGlobalMatrixG();
51      void buildGlobalMatrixM();
52      vector1D buildGlobalVectorb(int timeLayer);
53
54      void buildLocalVectorb(int elemNumber);
55      void buildLocalmatrixG(int elemNumber);
56      void buildLocalmatrixM(int elemNumber);
57
58      double f1, f2, f3, f4;
59      matrix2D A, G, M;
60      matrix2D GLocal, MLocal, tmpMatrix;
61      vector1D bLocal, tmpVector;
62  };
```

fem.cpp

```
1  #include "fem.h"
2
3
4
5
6
7  //void FEM::outputALocal() {
8  //  cout << endl;
9  //  for (int i = 0; i < ALocal.size(); ++i) {
10 //      for (int j = 0; j < ALocal.size(); ++j) {
11 //          cout << ALocal[i][j] << "  ";
12 //      }
13 //      cout << endl;
14 //  }
15 //}
16
17 void FEM::convAToDense() {
18
19      A.clear();
20      A.resize(n);
21      for (int i = 0; i < n; ++i) {
22          A[i].resize(n, 0);
23      }
24
25      for (int i = 0; i < n; ++i) {
26
27          A[i][i] = di[i];
28          int i0 = ia[i];
29          int i1 = ia[i + 1];
30
31          for (int k = i0; k < i1; ++k) {
32              A[i][ja[k]] = al[k];
33              A[ja[k]][i] = au[k];
34          }
35      }
36 }
```

```
37
38
39
40
41 void FEM::outputA() {
42     cout << endl;
43     for (int i = 0; i < A.size(); ++i) {
44         for (int j = 0; j < A.size(); ++j) {
45             cout << A[i][j] << "\t";
46         }
47         cout << endl;
48     }
49 }
50
51
52 void FEM::outputG() {
53     cout << endl;
54     for (int i = 0; i < G.size(); ++i) {
55         for (int j = 0; j < G.size(); ++j) {
56             cout << G[i][j] << "\t";
57         }
58         cout << endl;
59     }
60 }
61
62 void FEM::outputM() {
63     cout << endl;
64     for (int i = 0; i < M.size(); ++i) {
65         for (int j = 0; j < M.size(); ++j) {
66             cout << M[i][j] << "\t";
67         }
68         cout << endl;
69     }
70 }
71
72
73
74
75 // Инициализируем модель, задавая функции u, f и тип сетки
76 void FEM::init(
77     function3D _u,
78     function3D _f,
79     double _lambda,
80     double _gamma,
81     double _sigma,
82     double _chi,
83     bool _isGridUniform,
84     bool _isTimeUniform,
85     int _coefGrid,
86     int _coefTime
87 ) {
88     ifstream fin("input/SLAE_parameters.txt");
89     fin >> E >> delta >> maxiter;
90     fin.close();
91     u = _u;
92     f = _f;
93     lambda = _lambda;
94     gamma = _gamma;
95     sigma = _sigma;
96     chi = _chi;
```

```
97      isGridUniform = _isGridUniform;
98      isTimeUniform = _isTimeUniform;
99      coefGrid = _coefGrid;
100     coefTime = _coefTime;
101 }
102
103 double FEM::solve()
104 {
105     n = nodesCount;
106
107     q1.resize(nodesCount);
108     q2.resize(nodesCount);
109     q3.resize(nodesCount);
110
111     for (size_t i = 0; i < heigth; i++)
112         for (size_t j = 0; j < width; j++)
113         {
114             int k = i * width + j;
115             q3[k] = u(nodes[k].x, nodes[k].y, times[0]);
116             q2[k] = u(nodes[k].x, nodes[k].y, times[1]);
117             q1[k] = u(nodes[k].x, nodes[k].y, times[2]);
118         }
119
120     b3 = buildGlobalVectorb(0);
121     b2 = buildGlobalVectorb(1);
122     b1 = buildGlobalVectorb(2);
123
124     for (size_t timeLayer = 3; timeLayer < times.size(); timeLayer++) {
125         implicitCheme4(timeLayer);
126
127         for (int j = 0; j < A.size(); ++j) {
128             for (int i = j + 1; i < A.size(); ++i) {
129
130                 double toMult = A[i][j] / A[j][j];
131
132                 for (int k = 0; k < A.size(); ++k)
133                     A[i][k] -= toMult * A[j][k];
134
135                 b[i] -= toMult * b[j];
136             }
137         }
138
139
140         vector <double> x;
141         x.resize(A.size(), 0);
142
143         for (int i = n - 1; i >= 0; --i) {
144
145             double tmp = 0.0;
146             for (int j = i + 1; j < A.size(); ++j) {
147                 tmp += A[i][j] * x[j];
148             }
149             x[i] = (b[i] - tmp) / A[i][i];
150         }
151         q = x;
152
153         b3 = b2;
154         b2 = b1;
155         b1 = b;
156
```

```
157         q3 = q2;
158         q2 = q1;
159         q1 = q;
160 
161         //cout << endl << "q:" << endl << q << endl;
162         cout << endl << "Residual: " << endl << calcNormAtMainNodes(q, t0);
163     }
164     return calcNormAtMainNodes(q, t0);
165 }
166 
167 //---------------------------------------------------------------------------
168 //---------------------------------------------------------------------------
169 //---------------------------------------------------------------------------
170 
171 // Строим глобальную матрицу системы нелинейных уравнений см(. с. 239)
172 void FEM::implicitCheme4(int timeLayer)
173 {
174     A.clear();
175     A.resize(nodesCount);
176     for (size_t i = 0; i < nodesCount; i++)
177         A[i].resize(nodesCount, 0);
178 
179     buildGlobalMatrixG();
180     buildGlobalMatrixM();
181     b = buildGlobalVectorb(timeLayer);
182 
183     t0 = times[timeLayer];
184     t1 = times[timeLayer - 1];
185     t2 = times[timeLayer - 2];
186     t3 = times[timeLayer - 3];
187 
188     t01 = t0 - t1;
189     t02 = t0 - t2;
190     t03 = t0 - t3;
191     t12 = t1 - t2;
192     t23 = t2 - t3;
193     t13 = t1 - t3;
194 
195 
196 
197     // Собираем левую часть
198     double tmp = 2 * chi * (t01 + t02 + t03) / (t01 * t02 * t03)
199                 + sigma * (1.0 / t01 + 1.0 / t02 + 1.0 / t03) + gamma;
200     for (size_t i = 0; i < nodesCount; i++)
201         for (size_t j = 0; j < nodesCount; j++)
202             A[i][j] += M[i][j] * tmp + G[i][j];
203 
204 
205     // Собираем правую часть
206     b = b
207         + (2 * chi * (t01 + t02) / (t03 * t13 * t23) + sigma * (t01 * t02) / (
    t03 * t13 * t23)) * M * q3
208         - (2 * chi * (t01 + t03) / (t02 * t12 * t23) + sigma * (t01 * t03) / (
    t02 * t12 * t23)) * M * q2
209         + (2 * chi * (t02 + t03) / (t01 * t12 * t13) + sigma * (t02 * t03) / (
    t01 * t12 * t13)) * M * q1;
210 
211     //outputA();
212     //cout << endl << "b:" << endl << b << endl;
213 
```

```
214     // Добавляем краевые условия
215     for (size_t i = 0; i < nodesCount; i++)
216     {
217         if (nodes[i].type == 1) {
218             A[i].clear();
219             A[i].resize(nodesCount, 0);
220             A[i][i] = 1;
221             b[i] = u(nodes[i].x, nodes[i].y, t0);
222         }
223     }
224 
225     //cout << endl << "Added 1st boundary conditions:" << endl;
226     //cout << endl << "A:" << endl;
227     //outputA();
228     //cout << endl << "b:" << endl << b << endl;
229 }
230 
231 
232 
233 void FEM::buildGlobalMatrixG()
234 {
235     G.clear();
236     G.resize(nodesCount);
237     for (size_t i = 0; i < nodesCount; i++)
238         G[i].resize(nodesCount, 0);
239 
240 
241     for (size_t i = 0; i < heigth - 1; i++)
242     {
243         for (size_t j = 0; j < width - 1; j++)
244         {
245             int k = i * width + j;
246             buildLocalmatrixG(k);
247 
248             vector <int> nearestNodesIndexes = { k, k + 1, k + width, k + width
    + 1 };
249             for (size_t i1 = 0; i1 < 4; i1++)
250                 for (size_t j1 = 0; j1 < 4; j1++)
251                     G[nearestNodesIndexes[i1]][nearestNodesIndexes[j1]] +=
    GLocal[i1][j1];
252         }
253     }
254 }
255 
256 
257 
258 void FEM::buildGlobalMatrixM()
259 {
260     M.clear();
261     M.resize(nodesCount);
262     for (size_t i = 0; i < nodesCount; i++)
263         M[i].resize(nodesCount, 0);
264 
265     for (size_t i = 0; i < heigth - 1; i++)
266     {
267         for (size_t j = 0; j < width - 1; j++)
268         {
269             int k = i * width + j;
270             buildLocalmatrixM(k);
271 
```

```
272             vector <int> nearestNodesIndexes = { k, k + 1, k + width, k + width
    + 1 };
273             for (size_t i1 = 0; i1 < 4; i1++)
274                 for (size_t j1 = 0; j1 < 4; j1++)
275                     M[nearestNodesIndexes[i1]][nearestNodesIndexes[j1]] +=
    MLocal[i1][j1];
276         }
277     }
278 }
279
280
281
282
283 // Строим глобальный вектор правой части системы нелинейных уравнений
284 vector1D FEM::buildGlobalVectorb(int timeLayer)
285 {
286     t = times[timeLayer];
287     tmpVector.clear();
288     tmpVector.resize(nodesCount, 0);
289
290
291     for (size_t i = 0; i < heigth - 1; i++)
292     {
293         for (size_t j = 0; j < width - 1; j++)
294         {
295             int k = i * width + j;
296             buildLocalVectorb(k);
297
298             vector <int> nearestNodesIndexes = { k, k + 1, k + width, k + width
    + 1 };
299             for (size_t i1 = 0; i1 < 4; i1++)
300                 tmpVector[nearestNodesIndexes[i1]] += bLocal[i1];
301         }
302     }
303
304     return tmpVector;
305 }
306
307
308 //-----------------------------------------------------------------------------
309 //-----------------------------------------------------------------------------
310 //-----------------------------------------------------------------------------
311
312
313
314 // Построение локальной матрицы жёсткости
315 void FEM::buildLocalmatrixG(int elemNumber)
316 {
317     hx = nodes[elemNumber + 1].x - nodes[elemNumber].x;
318     hy = nodes[elemNumber + width].y - nodes[elemNumber].y;
319
320     GLocal = { {0, 0, 0, 0},
321                {0, 0, 0, 0},
322                {0, 0, 0, 0},
323                {0, 0, 0, 0} };
324
325     tmpMatrix = { {2, -2, 1, -1},
326                   {-2, 2, -1, 1},
327                   {1, -1, 2, -2},
328                   {-1, 1, -2, 2} };
```

```
329
330     for (size_t i = 0; i < 4; i++)
331         for (size_t j = 0; j < 4; j++)
332             GLocal[i][j] += tmpMatrix[i][j] * lambda * hy / (6 * hx);
333
334     tmpMatrix = { {2, 1, -2, -1},
335                   {1, 2, -1, -2},
336                   {-2, -1, 2, 1},
337                   {-1, -2, 1, 2} };
338
339
340     for (size_t i = 0; i < 4; i++)
341         for (size_t j = 0; j < 4; j++)
342             GLocal[i][j] += tmpMatrix[i][j] * lambda * hx / (6 * hy);
343 }
344
345
346 // Построение локальной матрицы масс
347 void FEM::buildLocalmatrixM(int elemNumber)
348 {
349     hx = nodes[elemNumber + 1].x - nodes[elemNumber].x;
350     hy = nodes[elemNumber + width].y - nodes[elemNumber].y;
351
352     MLocal = { {4, 2, 2, 1},
353                   {2, 4, 1, 2},
354                   {2, 1, 4, 2},
355                   {1, 2, 2, 4} };
356
357     for (size_t i = 0; i < 4; i++)
358         for (size_t j = 0; j < 4; j++)
359             MLocal[i][j] *= hx * hy / 36;
360 }
361
362
363 // Построение локального вектора b
364 void FEM::buildLocalVectorb(int elemNumber)
365 {
366     hx = nodes[elemNumber + 1].x - nodes[elemNumber].x;
367     hy = nodes[elemNumber + width].y - nodes[elemNumber].y;
368
369     bLocal = { 0, 0, 0, 0 };
370     f1 = f(nodes[elemNumber].x, nodes[elemNumber].y, t);
371     f2 = f(nodes[elemNumber + 1].x, nodes[elemNumber + 1].y, t);
372     f3 = f(nodes[elemNumber + width].x, nodes[elemNumber + width].y, t);
373     f4 = f(nodes[elemNumber + width + 1].x, nodes[elemNumber + width + 1].y, t);
374
375     bLocal[0] = (hx*hy / 36) * (4 * f1 + 2 * f2 + 2 * f3 + 1 * f4);
376     bLocal[1] = (hx*hy / 36) * (2 * f1 + 4 * f2 + 1 * f3 + 2 * f4);
377     bLocal[2] = (hx*hy / 36) * (2 * f1 + 1 * f2 + 4 * f3 + 2 * f4);
378     bLocal[3] = (hx*hy / 36) * (1 * f1 + 2 * f2 + 2 * f3 + 4 * f4);
379 }
```

solver.h

```
1 #pragma once
2 #include "head.h"
3
4
5 // Система линейных алгебраических уравнений
6 class SOLVER {
7
8 public:
```

```
    void initSLAE();
    void BiCG();


    void getVectX(vector1D &x) { x = b; };
    void generateVectX(int size);
    void writeXToFile(const char *fileName);
    void writeXToStream(std::ofstream& fout);
    void writeFToFile(const char *fileName);


    int getDimention() { return n; }
    void decomposionD();
    void decomposionLUsq();
    vector1D execDirectTraversal(const vector1D &_F);
    vector1D execReverseTraversal(const vector1D &_y);

    pair<int, double> LOS();
    pair<int, double> LOSfactD();
    pair<int, double> LOSfactLUsq();
    void clearAll();
    void setMaxiter(int new_maxiter) { maxiter = new_maxiter; }
    void setE(double new_E) { E = new_E; }

protected:
    vector1D q, q1, q2, q3;
    vector1D di, al, au, di_f, al_f, au_f;
    vector <int> ia, ja;
    int n, maxiter;
    double E, delta;
    vector1D x, r, z, p, b, b1, b2, b3, bTmp;


    vector1D multA(const vector1D&x);
    vector1D multD(const vector1D&x);
    double calcRelativeDiscrepancy();
    double calcNormE(const vector1D &x) { return sqrt(x*x); }


    vector1D xPrev, xPrevPrev;  // решение на k-1 итерации

    vector1D s; // вспомогательный вектор
    double alpha, beta;


    void generateInitialGuess();
    vector1D multAOn(const vector1D &v);
    vector1D multAtOn(const vector1D &v);
    bool doStop(int i);
};
```

solver.cpp

```
#include "solver.h"




// A*x = b, где x – произвольный вектор
vector1D SOLVER::multA(const vector1D& x) {

    vector1D result(n);
```

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

        result[i] = di[i] * x[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];

        for (int k = i0; k < i1; ++k) {

            result[i] += al[k] * x[ja[k]];
            result[ja[k]] += au[k] * x[i];
        }
    }

    return result;
}

// A*x = b, где x — произвольный вектор
vector1D SOLVER::multD(const vector1D&x) {

    vector1D result(n);

    for (int i = 0; i < n; ++i)
        result[i] = di_f[i] * x[i];

    return result;
}


// Создаём вектор x* = (1,2,...n)'
void SOLVER::generateVectX(int size) {

    x.resize(size);
    for (int i = 0; i < size; ++i) {
        x[i] = i + 1;
    }
}


// Вывод вектора b в файл
void SOLVER::writeFToFile(const char *fileName) {

    std::ofstream fout;
    fout.open(fileName);

    for (int i = 0; i < b.size(); ++i)
        fout << b[i] << endl;
    fout.close();
}


// Вывод вектора x в файл
void SOLVER::writeXToFile(const char * fileName) {

    std::ofstream fout;
    fout.open(fileName);
    for (int i = 0; i < x.size(); ++i)
        fout << x[i] << " ";
    fout << " \t";
    fout.close();
}
```

```
70
71
72  // Вывод вектора x в поток
73  void SOLVER::writeXToStream(std::ofstream& fout) {
74
75      for (int i = 0; i < x.size(); ++i)
76          fout << x[i] << "\n";
77      fout << "\n";
78  }
79
80
81  // Диагональное предобуславливание M = D
82  void SOLVER::decomposionD() {
83
84      di_f.clear();
85      di_f.resize(n);
86      for (int i = 0; i < n; ++i)
87          di_f[i] = 1.0 / sqrt(di[i]);
88  }
89
90
91  // LU_sq разложение матрицы A
92  void SOLVER::decomposionLUsq() {
93
94      double sum_u, sum_l, sum_d;
95      di_f = di;
96      al_f = al;
97      au_f = au;
98
99      // Идём построчно в верхнем треугольнике, что экививалентно
100     // Обходу нижнего треугольника по столбцам вниз, начиная с первого
101     for (int i = 0; i < n; ++i) {
102
103         int i0 = ia[i];
104         int i1 = ia[i + 1];
105
106         // Рассчёт элементов нижнего треугольника
107         for (int k = i0; k < i1; ++k) {
108
109             int j = ja[k]; // текущий j
110             int j0 = ia[j]; // i0 строки j
111             int j1 = ia[j + 1]; // i1 строки j
112             sum_l = 0;
113             sum_u = 0;
114             int ki = i0; // Индекс l_ik
115             int kj = j0; // Индекс u_kj
116
117             while (ki < k && kj < j1) {
118
119                 if (ja[ki] == ja[kj]) { // l_ik * u_kj
120                     sum_l += al_f[ki] * au_f[kj];
121                     sum_u += au_f[ki] * al_f[kj];
122                     ki++;
123                     kj++;
124                 }
125                 else { // Ищем следующие элементы i и j строки, которые можем
    перемножить
126                     if (ja[ki] > ja[kj]) kj++;
127                     else ki++;
128                 }
```

```
            }

            al_f[k] = (al_f[k] - sum_l) / di_f[j];
            au_f[k] = (au_f[k] - sum_u) / di_f[j];
        }


        // Рассчёт диагонального элемента
        sum_d = 0.0;
        for (int k = i0; k < i1; ++k)
            sum_d += al_f[k] * au_f[k];
        di_f[i] = sqrt(di_f[i] - sum_d);
    }
}


// Прямой ход    L y = b    ==>    y = L^-1 b
vector1D SOLVER::execDirectTraversal(const vector1D &_F) {

    vector1D y;
    y.resize(n, 0);

    for (int i = 0; i < n; ++i) {
        double sum = 0;
        int i0 = ia[i];
        int i1 = ia[i + 1];
        for (int k = i0; k < i1; ++k)
            sum += al_f[k] * y[ja[k]];

        y[i] = (_F[i] - sum) / di_f[i];
    }
    return y;
}


// Обратный ход    U(sq) x = y    ==>    x = U(sq)^-1 y
vector1D SOLVER::execReverseTraversal(const vector1D &_y) {

    vector1D x, y = _y;
    x.resize(n);
    for (int i = n - 1; i >= 0; --i) {

        x[i] = y[i] / di_f[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];
        for (int k = i0; k < i1; ++k)
            y[ja[k]] -= au_f[k] * x[i];
    }

    return x;
}



// Полная очистка СЛАУ
void SOLVER::clearAll() {

    n = 0;
    E = 0.0;
    maxiter = 0;
```

```
189
190     di.clear();
191     ia.clear();
192     ja.clear();
193     al.clear();
194     au.clear();
195
196     di_f.clear();
197     al_f.clear();
198     au_f.clear();
199
200     x.clear();
201     r.clear();
202     z.clear();
203     p.clear();
204     b.clear();
205     bTmp.clear();
206 }
207
208
209
210 // Рассчёт относительной невязки
211 double SOLVER::calcRelativeDiscrepancy() {
212     //return calcNormE(r) / calcNormE(b);
213     return (r*r);
214 }
215
216
217 // Локально – оптимальная схема
218 pair<int, double> SOLVER::LOS() {
219
220
221     x.clear();          // Задаём начальное приближение
222     x.resize(n, 0);     // x_0 = (0, 0, ...)
223     r.resize(n);
224
225     vector1D xprev = x;
226     r = b – multA(x);   // r_0 = b – A*x_0
227     z = r;              // z_0 = r_0
228     p = multA(z);       // p_0 = A*z_0
229
230
231     for (int i = 0; i < maxiter; ++i) {
232
233         double pp = (p * p);
234         double alpha = (p * r) / pp;
235
236         x = x + alpha * z;
237         r = r – alpha * p;
238
239         bTmp = multA(r);
240         double beta = –(p * bTmp) / pp;
241
242         z = r + beta * z;
243         p = bTmp + beta * p;
244
245
246         double relativeDiscrepancy = calcRelativeDiscrepancy();
247         if (x == xprev || relativeDiscrepancy < E) {
248             return make_pair(i, relativeDiscrepancy);
```

```
        }
        xprev = x;
    }
}


// Локально – оптимальная схема с неполной диагональной факторизацией
pair<int, double> SOLVER::LOSfactD() {

    x.clear();          // Задаём начальное приближение
    x.resize(n, 0);     // x_0 = (0, 0, ...)
    vector1D xprev = x;
    decomposionD();

    r = b – multA(x);   // r_0 = b – A*x_0
    r = multD(r);
    z = multD(r);       // z = U^–1 r
    p = multA(z);       // p = A*z
    p = multD(p);       // p = L^–1 A*z


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

        double pp = p * p;
        double alpha = (p*r) / pp;
        x = x + alpha * z;
        r = r – alpha * p;

        vector1D tmp = multD(r);
        tmp = multA(tmp);
        tmp = multD(tmp);
        double beta = –(p * tmp) / pp;
        p = tmp + beta * p;

        tmp = multD(r);
        z = tmp + beta * z;


        double relativeDiscrepancy = calcRelativeDiscrepancy();
        if (x == xprev || relativeDiscrepancy < E) {
            return make_pair(i, relativeDiscrepancy);
        }
        xprev = x;
    }
}




// Локально – оптимальная схема с неполной факторизацией LU(sq)
pair<int, double> SOLVER::LOSfactLUsq() {

    x.clear();                      // Задаём начальное приближение
    x.resize(n, 0);                 // x_0 = (0, 0, ...)
    vector1D xprev = x;
    decomposionLUsq();

    r = b – multA(x);               // r_0 = b – A*x_0
    r = execDirectTraversal(r);     // r = L^–1 (b – A*x_0)
```

```
    z = execReverseTraversal(r);    // z = U^-1 r
    p = multA(z);                   // p = A*z
    p = execDirectTraversal(p);     // p = L^-1 A*z


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

        double pp = p * p;
        double alpha = (p*r) / pp;
        x = x + alpha * z;
        r = r - alpha * p;

        vector1D tmp = execReverseTraversal(r);
        tmp = multA(tmp);
        tmp = execDirectTraversal(tmp);
        double beta = -(p * tmp) / pp;
        p = tmp + beta * p;

        tmp = execReverseTraversal(r);
        z = tmp + beta * z;


        double relativeDiscrepancy = calcRelativeDiscrepancy();
        if (x == xprev || relativeDiscrepancy < E) {
            return make_pair(i, relativeDiscrepancy);
        }
        xprev = x;
    }
}



// Считываем все СЛАУ и её параметры из файлов
void SOLVER::initSLAE()
{
    ifstream finMatrix("input/matrix.txt");
    ifstream finVector("input/vector.txt");
    ifstream finParams("input/SLAE_parameters.txt");

    finParams >> maxiter >> E >> delta;
    finMatrix >> n;
    di.resize(n);
    ia.resize(n + 1);

    for (size_t i = 0; i < n; i++)
        finMatrix >> di[i];
    for (size_t i = 0; i < n + 1; i++)
        finMatrix >> ia[i];

    ja.resize(ia[n]);
    al.resize(ia[n]);
    au.resize(ia[n]);

    for (size_t i = 0; i < ja.size(); i++)
        finMatrix >> ja[i];
    for (size_t i = 0; i < al.size(); i++)
        finMatrix >> al[i];
    for (size_t i = 0; i < au.size(); i++)
        finMatrix >> au[i];
```

```
369
370     b.resize(n);
371     for (size_t i = 0; i < n; i++)
372         finVector >> b[i];
373
374     finMatrix.close();
375     finVector.close();
376     finParams.close();
377 }
378
379
380
381 // Метод бисопряжённых градиентов
382 void SOLVER::BiCG()
383 {
384     ofstream fout("output/result.txt");
385     int i = 0;
386     double prPrev, pr;
387     vector1D Az, Ats;
388     generateInitialGuess();
389     r = b - multAOn(x);
390     p = z = s = r;
391     do {
392         xPrev = x;
393         prPrev = (p*r);
394         Az = multAOn(z);
395         Ats = multAtOn(s);
396
397         alpha = prPrev / (s*Az);
398
399         x = x + alpha * z;
400         r = r - alpha * Az;
401         p = p - alpha * Ats;
402
403         beta = (p*r) / prPrev;
404
405         z = r + beta * z;
406         s = p + beta * s;
407
408         i++;
409     } while (!doStop(i));
410
411     fout << x << endl
412         << "Iterations: " << i;
413
414     fout.close();
415 }
416
417
418
419 // Создание начального приближения x0 = (0,...,0)'
420 void SOLVER::generateInitialGuess()
421 {
422     x.resize(n, 1);
423 }
424
425
426
427 // Умножение матрицы A на вектор
428 vector1D SOLVER::multAOn(const vector1D &v)
```

```
{
    vector1D result(n);
    for (size_t i = 0; i < n; i++)
    {
        result[i] = di[i] * v[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];

        for (size_t k = i0; k < i1; k++)
        {
            int j = ja[k];
            result[i] += al[k] * v[j];
            result[j] += au[k] * v[i];
        }
    }
    return result;
}



// Умножение транспонированной матрицы A на вектор
vector1D SOLVER::multAtOn(const vector1D &v)
{
    vector1D result(n);
    for (size_t i = 0; i < n; i++)
    {
        result[i] = di[i] * v[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];
        for (size_t k = i0; k < i1; k++)
        {
            int j = ja[k];
            result[j] += al[k] * v[i];
            result[i] += au[k] * v[j];
        }
    }
    return result;
}



// Проверяем условие выхода
bool SOLVER::doStop(int i)
{
    // Выход по числу итераций
    if (i > maxiter)
        return true;

    // Выход шагу
    if (calcNormE(x - xPrev) / calcNormE(x) < delta)
        return true;

    // Выход по относительной невязке
    if (calcNormE(multAOn(x) - b) / calcNormE(b) < E)
        return true;

    return false;
}
```

main.cpp

```
#include "fem.h"
```

```
#include <thread>

function1D calcFirstDerivative(const function1D& f) {
    return [f](double x) -> double {
        const double h = 0.001;
        return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) / (12 * h);
    };
}


function1D calcSecondDerivative(const function1D& f) {
    return [f](double x) -> double {
        const double h = 0.001;
        return (-f(x + 2 * h) + 16 * f(x + h) - 30 * f(x) + 16 * f(x - h) - f(x - 2 * h)) / (12 * h*h);
    };
}


function3D calc_f(
    const function3D& u,
    double lambda,
    double gamma,
    double sigma,
    double chi
) {
    return [=](double x, double y, double t) -> double
    {
        using namespace placeholders;
        auto u_t = calcFirstDerivative(bind(u, x, y, _1));

        auto u_xx = calcSecondDerivative(bind(u, _1, y, t));
        auto u_yy = calcSecondDerivative(bind(u, x, _1, t));
        auto u_tt = calcSecondDerivative(bind(u, x, y, _1));

        return -lambda * (u_xx(x) + u_yy(y)) + gamma * u(x, y, t) + sigma * u_t(t) + chi * u_tt(t);
    };
}


function3D sum_u(
    const function3D& u_x,
    const function3D& u_t,
    double lambda,
    double gamma,
    double sigma,
    double chi
) {
    return [=](double x, double y, double t) -> double
    {
        return (u_x(x, y, t) + u_t(x, y, t));
    };
}


void main() {

    int coefGrid = 0;
```

```
59      int coefTime = 0;
60      bool isGridUniform = true;
61      bool isTimeUniform = true;
62      double lambda = 1;
63      double gamma = 1;
64      double  sigma = 1;
65      double chi = 1;
66
67      cout << setprecision(3) << scientific;
68
69      string prefix = "";
70      if (!isGridUniform)
71          prefix = "Non";
72      string gridFile = "grids/" + prefix + "Uniform_" + to_string(coefGrid) + ".
        txt";
73      string gridBorderFile = "grids/Border" + prefix + "Uniform_" + to_string(
        coefGrid) + ".txt";
74
75
76
77      vector <function3D> u_x(8), u_t(8), u(64), f(64);
78      u_x[0] = { [](double x, double y, double t) -> double { return 1; } };
79      u_x[1] = { [](double x, double y, double t) -> double { return x + y; } };
80      u_x[2] = { [](double x, double y, double t) -> double { return pow(x, 2) +
        pow(y, 2); } };
81      u_x[3] = { [](double x, double y, double t) -> double { return pow(x, 3) +
        pow(y, 3); } };
82      u_x[4] = { [](double x, double y, double t) -> double { return pow(x, 4) +
        pow(y, 4); } };
83      u_x[5] = { [](double x, double y, double t) -> double { return pow(x, 5) +
        pow(y, 5); } };
84      u_x[6] = { [](double x, double y, double t) -> double { return sin(x) + sin(
        y); } };
85      u_x[7] = { [](double x, double y, double t) -> double { return exp(x) + exp(
        y); } };
86
87      u_t[0] = { [](double x, double y, double t) -> double { return 1; } };
88      u_t[1] = { [](double x, double y, double t) -> double { return t; } };
89      u_t[2] = { [](double x, double y, double t) -> double { return pow(t, 2); }
        };
90      u_t[3] = { [](double x, double y, double t) -> double { return pow(t, 3); }
        };
91      u_t[4] = { [](double x, double y, double t) -> double { return pow(t, 4); }
        };
92      u_t[5] = { [](double x, double y, double t) -> double { return pow(t, 5); }
        };
93      u_t[6] = { [](double x, double y, double t) -> double { return sin(t); } };
94      u_t[7] = { [](double x, double y, double t) -> double { return exp(t); } };
95
96      vector <string> u_x_names = {
97          "$1$",
98          "$x+y$",
99          "$x^2+y^2$",
100         "$x^3+y^3$",
101         "$x^4+y^4$",
102         "$x^5+y^5$",
103         "$sin(x)+sin(y)$",
104         "$e^x+e^y$"
105     };
106
```

```
107
108     // ▯▯▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ ▯▯ ▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯ ▯▯ ▯▯▯▯▯▯▯▯ ▯ ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
109     auto researchTable = [&](bool isGridUniform, bool isTimeUniform) {
110         ofstream table1("report/table_" + to_string(isGridUniform) + to_string(
    isTimeUniform) + ".txt");
111         table1 << setprecision(2) << scientific;
112         table1 << "a\t$1$\t$t$\t$t^2$\t$t^3$\t$t^4$\t$t^5$\t$sin(t)$\t$e^t$" <<
    endl;
113         for (size_t i = 0; i < u_x.size(); i++)
114         {
115             table1 << u_x_names[i] << "\t";
116             for (size_t j = 0; j < u_x.size(); j++)
117             {
118                 int k = 8 * i + j;
119                 u[k] = sum_u(u_x[i], u_t[j], lambda, gamma, sigma, chi);
120                 f[k] = calc_f(u[k], lambda, gamma, sigma, chi);
121
122                 FEM fem;
123                 fem.init(u[k], f[k], lambda, gamma, sigma, chi, isGridUniform,
    isTimeUniform, coefGrid, coefTime);
124                 fem.inputGrid();
125                 fem.buildGrid();
126                 fem.inputTime();
127                 fem.buildTimeGrid();
128                 if (j + 1 != u_x.size())
129                     table1 << fem.solve() << "\t";
130                 else
131                     table1 << fem.solve() << endl;
132
133                 /*fem.saveGridAndBorder(gridFile, gridBorderFile);
134                 string runVisualisation = "python plot.py " + gridFile + " " +
    gridBorderFile;
135                 system(runVisualisation.c_str());*/
136             }
137         }
138         table1.close();
139     };
140     cout << "Research Table 1 1" << endl;
141     researchTable(true, true);
142     cout << "Research Table 1 0" << endl;
143     researchTable(true, false);
144     cout << "Research Table 0 1" << endl;
145     researchTable(false, true);
146     cout << "Research Table 0 0" << endl;
147     researchTable(false, false);
148
149
150
151 // ▯▯▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ ▯▯ ▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯ ▯▯ ▯▯▯▯▯▯▯▯ ▯ ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
152     auto researchConvergence = [&](bool isGridUniform, bool isTimeUniform) {
153         for (size_t i = 0; i < u_x.size(); i++)
154         {
155             ofstream fout("report/file_u" + to_string(i) + "." + to_string(
    isGridUniform) + to_string(isTimeUniform) + ".txt");
156             fout << scientific << setprecision(3);
157             fout << "i\tnodes\tnorm\n";
158             for (int coef = 0; coef < 3; coef++)
159             {
160                 u[i] = sum_u(u_x[i], u_t[i], lambda, gamma, sigma, chi);
161                 f[i] = calc_f(u[i], lambda, gamma, sigma, chi);
```

```
162
163                 FEM fem;
164                 fem.init(u[i], f[i], lambda, gamma, sigma, chi, isGridUniform,
    isTimeUniform, coef, coef);
165                 fem.inputGrid();
166                 fem.buildGrid();
167                 fem.inputTime();
168                 fem.buildTimeGrid();
169                 fout << coef << "\t"
170                     << fem.getNodesCount() << "\t"
171                     << fem.solve() << endl;
172             }
173         }
174     };
175
176     researchConvergence(true, true);
177     cout << "Research 2.1: convergence with grid crushing" << endl;
178     researchConvergence(true, false);
179     cout << "Research 2.2: convergence with grid crushing" << endl;
180     researchConvergence(false, true);
181     cout << "Research 2.3: convergence with grid crushing" << endl;
182     researchConvergence(false, false);
183     cout << "Research 2.4: convergence with grid crushing" << endl;
184
185 }
```