

Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Уравнения математической физики

Курсовая работа

Тема: Решение двумерной гиперболической задачи при помощи
трёхслойной неявной схемы. Базисные функции билинейные

Факультет:	ФПМИ
Группа:	ПМ-63
Студент:	Утюганов Д.С.
Вариант:	42, 6

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 42, 6: Решить одномерную гармоническую задачу в декартовых координатах, базисные функции - билинейные, трёхслойная неявная схема.

3. Анализ

3.1. Постановка задачи

Дано гиперболическое уравнение в декартовой системе координат:

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

3.2. Дискретизация по времени

Представим искомое решение u на интервале (t_{j-2}, t_j) :

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

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

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

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

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

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

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

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 3-х слойную неявную схему:

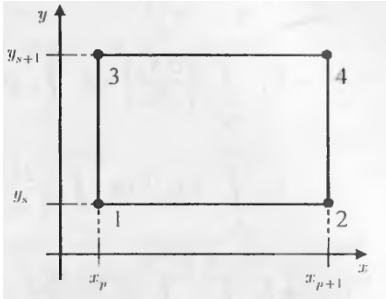
$$\begin{aligned} & \left(\left[\frac{2\chi}{t_{01}t_{02}} + \sigma \frac{t_{02} + t_{01}}{t_{01}t_{02}} + \gamma \right] M + G \right) q^j = b^j \\ & - \left[\frac{2\chi}{t_{01}t_{02}} + \sigma \frac{t_{01}}{t_{02}t_{12}} \right] M q^{j-2} \\ & + \left[\frac{2\chi}{t_{01}t_{12}} + \sigma \frac{2}{t_{01}t_{12}} \right] M q^{j-1} \end{aligned}$$

3.3. Конечноэлементная дискретизация

Формулы для билинейных базисных функций прямоугольных элементов:

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

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



3.4. Локальные матрицы и вектора

Аналитические выражения для вычисления элементов локальных матриц:

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

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

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

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

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

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

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

$\begin{matrix} & time(t) \\ space(x, y) \end{matrix}$	1	t	t^2	t^3	t^4	t^5	$\sin(t)$	e^t
1	3.60e-17	3.94e-13	1.65e-12	1.49e-02	4.15e-02	7.82e-02	1.87e-03	5.08e-03
$x + y$	1.46e-12	1.71e-12	2.44e-12	1.49e-02	4.15e-02	7.82e-02	1.87e-03	5.08e-03
$x^2 + y^2$	9.45e-13	4.74e-13	7.57e-13	1.49e-02	4.15e-02	7.82e-02	1.87e-03	5.08e-03
$x^3 + y^3$	1.03e-12	4.07e-13	9.69e-13	1.49e-02	4.15e-02	7.82e-02	1.87e-03	5.08e-03
$x^4 + y^4$	6.81e-04	6.81e-04	6.81e-04	1.42e-02	4.08e-02	7.75e-02	2.55e-03	4.40e-03
$x^5 + y^5$	1.73e-03	1.73e-03	1.73e-03	1.32e-02	3.98e-02	7.65e-02	3.58e-03	3.39e-03
$\sin(x) + \sin(y)$	1.34e-05	1.34e-05	1.34e-05	1.49e-02	4.15e-02	7.82e-02	1.88e-03	5.07e-03
$e^x + e^y$	4.82e-05	4.82e-05	4.82e-05	1.49e-02	4.15e-02	7.82e-02	1.91e-03	5.03e-03

Равномерная сетка по пространству

Неравномерная сетка по времени

$\begin{matrix} & time(t) \\ space(x, y) \end{matrix}$	1	t	t^2	t^3	t^4	t^5	$\sin(t)$	e^t
1	3.85e-17	4.01e-13	1.67e-12	1.58e-02	4.34e-02	8.12e-02	1.99e-03	5.33e-03
$x + y$	1.48e-12	1.74e-12	2.49e-12	1.58e-02	4.34e-02	8.12e-02	1.99e-03	5.33e-03
$x^2 + y^2$	9.58e-13	4.80e-13	7.69e-13	1.58e-02	4.34e-02	8.12e-02	1.99e-03	5.33e-03
$x^3 + y^3$	1.04e-12	4.10e-13	9.83e-13	1.58e-02	4.34e-02	8.12e-02	1.99e-03	5.33e-03
$x^4 + y^4$	6.93e-04	6.93e-04	6.93e-04	1.51e-02	4.27e-02	8.05e-02	2.68e-03	4.64e-03
$x^5 + y^5$	1.76e-03	1.76e-03	1.76e-03	1.40e-02	4.17e-02	7.94e-02	3.73e-03	3.61e-03
$\sin(x) + \sin(y)$	1.37e-05	1.37e-05	1.37e-05	1.57e-02	4.34e-02	8.12e-02	2.00e-03	5.32e-03
$e^x + e^y$	4.91e-05	4.91e-05	4.91e-05	1.57e-02	4.33e-02	8.11e-02	2.03e-03	5.28e-03

Неравномерная сетка по пространству

Равномерная сетка по времени

$\begin{matrix} & time(t) \\ space(x, y) \end{matrix}$	1	t	t^2	t^3	t^4	t^5	$\sin(t)$	e^t
1	2.54e-17	3.81e-13	1.59e-12	1.44e-02	4.02e-02	7.58e-02	1.81e-03	4.92e-03
$x + y$	1.01e-12	9.25e-13	1.75e-12	1.44e-02	4.02e-02	7.58e-02	1.81e-03	4.92e-03
$x^2 + y^2$	8.85e-13	7.53e-13	1.39e-12	1.44e-02	4.02e-02	7.58e-02	1.81e-03	4.92e-03
$x^3 + y^3$	9.01e-13	5.61e-13	1.14e-12	1.44e-02	4.02e-02	7.58e-02	1.81e-03	4.92e-03
$x^4 + y^4$	7.68e-04	7.68e-04	7.68e-04	1.37e-02	3.95e-02	7.50e-02	2.57e-03	4.16e-03
$x^5 + y^5$	2.26e-03	2.26e-03	2.26e-03	1.23e-02	3.80e-02	7.36e-02	4.03e-03	2.82e-03
$\sin(x) + \sin(y)$	1.73e-05	1.73e-05	1.73e-05	1.44e-02	4.02e-02	7.58e-02	1.82e-03	4.90e-03
$e^x + e^y$	5.89e-05	5.89e-05	5.89e-05	1.44e-02	4.02e-02	7.57e-02	1.87e-03	4.86e-03

Неравномерная сетка по пространству

Неравномерная сетка по времени

$\begin{matrix} & time(t) \\ space(x, y) \end{matrix}$	1	t	t^2	t^3	t^4	t^5	$\sin(t)$	e^t
1	5.45e-17	3.88e-13	1.62e-12	1.53e-02	4.20e-02	7.86e-02	1.92e-03	5.17e-03
$x + y$	1.02e-12	9.38e-13	1.78e-12	1.53e-02	4.20e-02	7.86e-02	1.92e-03	5.17e-03
$x^2 + y^2$	8.96e-13	7.65e-13	1.41e-12	1.53e-02	4.20e-02	7.86e-02	1.92e-03	5.17e-03
$x^3 + y^3$	9.15e-13	5.67e-13	1.16e-12	1.53e-02	4.20e-02	7.86e-02	1.92e-03	5.17e-03
$x^4 + y^4$	7.82e-04	7.82e-04	7.82e-04	1.45e-02	4.13e-02	7.79e-02	2.70e-03	4.39e-03
$x^5 + y^5$	2.30e-03	2.30e-03	2.30e-03	1.31e-02	3.98e-02	7.64e-02	4.18e-03	3.02e-03
$\sin(x) + \sin(y)$	1.76e-05	1.76e-05	1.76e-05	1.52e-02	4.20e-02	7.86e-02	1.94e-03	5.15e-03
$e^x + e^y$	5.99e-05	5.99e-05	5.99e-05	1.52e-02	4.20e-02	7.86e-02	1.98e-03	5.11e-03

4.1. Вывод

Если порядок полинома по пространству не превышает порядка используемых базисных функций, а порядок полинома по времени не соответствует 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 36, а конечных элементов 25

Для неравномерных сеток по времени и пространству коэффициент $k=1.2$

$$u = 1$$

пространство время	равномерное				не равномерное			
	i	nodes	norm		i	nodes	norm	
равномерное	0	36	3.596e-17		0	36	2.543e-17	
	0	121	4.606e-17		0	121	1.046e-16	
	0	441	1.170e-16		0	441	1.584e-16	
не равномерное	i	nodes	norm		i	nodes	norm	
	0	36	3.852e-17		0	36	5.447e-17	
	0	121	9.434e-17		0	121	1.145e-16	
	0	441	6.420e-17		0	441	1.461e-16	

$$u = x$$

пространство время	равномерное				не равномерное			
	i	nodes	norm		i	nodes	norm	
равномерное	0	36	1.711e-12		0	36	9.249e-13	
	0	121	5.844e-13		0	121	1.529e-13	
	0	441	1.804e-13		0	441	1.702e-13	
не равномерное	i	nodes	norm		i	nodes	norm	
	0	36	1.739e-12		0	36	9.385e-13	
	0	121	5.869e-13		0	121	1.290e-13	
	0	441	2.360e-13		0	441	7.445e-14	

$$u = x^2$$

пространство время	равномерное				не равномерное			
	i	nodes	norm		i	nodes	norm	
равномерное	0	36	7.573e-13		0	36	1.389e-12	
	0	121	2.039e-13		0	121	1.876e-13	
	0	441	4.655e-14		0	441	4.047e-14	
не равномерное	i	nodes	norm		i	nodes	norm	
	0	36	7.694e-13		0	36	1.414e-12	
	0	121	3.159e-13		0	121	2.652e-13	
	0	441	1.839e-13		0	441	9.062e-14	

$$u = x^3$$

пространство время	равномерное				не равномерное			
	i	nodes	norm		i	nodes	norm	
равномерное	0	36	1.490e-02		0	36	1.444e-02	
	0	121	6.170e-03		0	121	4.000e-03	
	0	441	1.797e-03		0	441	8.269e-04	
не равномерное	i	nodes	norm		i	nodes	norm	
	0	36	1.575e-02		0	36	1.526e-02	
	0	121	7.622e-03		0	121	4.974e-03	
	0	441	2.825e-03		0	441	1.322e-03	

$$u = x^4$$

пространство время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	4.082e-02	0	36	3.945e-02
	0	121	1.614e-02	0	121	1.030e-02
	0	441	4.960e-03	0	441	2.251e-03
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	4.270e-02	0	36	4.127e-02
	0	121	2.007e-02	0	121	1.292e-02
	0	441	7.821e-03	0	441	3.631e-03

$$u = x^5$$

пространство время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	7.652e-02	0	36	7.362e-02
	0	121	2.927e-02	0	121	1.837e-02
	0	441	9.059e-03	0	441	4.061e-03
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	7.944e-02	0	36	7.642e-02
	0	121	3.698e-02	0	121	2.346e-02
	0	441	1.470e-02	0	441	6.776e-03

$$u = \sin(x)$$

пространство время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	1.879e-03	0	36	1.825e-03
	0	121	7.971e-04	0	121	5.214e-04
	0	441	2.278e-04	0	441	1.057e-04
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	1.999e-03	0	36	1.940e-03
	0	121	9.785e-04	0	121	6.437e-04
	0	441	3.550e-04	0	441	1.669e-04

$$u = e^x$$

пространство время	равномерное			не равномерное		
равномерное	i	nodes	norm	i	nodes	norm
	0	36	5.032e-03	0	36	4.864e-03
	0	121	2.023e-03	0	121	1.297e-03
	0	441	6.053e-04	0	441	2.759e-04
не равномерное	i	nodes	norm	i	nodes	norm
	0	36	5.283e-03	0	36	5.107e-03
	0	121	2.516e-03	0	121	1.626e-03
	0	441	9.596e-04	0	441	4.465e-04

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

```

1 #pragma once
2 #define _CRT_SECURE_NO_WARNINGS
3 #include <fstream>
4 #include <iostream>
5 #include <vector>
6 #include <string>

```

```

7 #include <iomanip>
8 #include <functional>
9 #include <cmath>
10
11 using namespace std;
12
13
14 typedef std::function<double(double)> function1D;
15 typedef std::function<double(double, double)> function2D;
16 typedef std::function<double(double, double, double)> function3D;
17
18 typedef vector<double> vector1D;
19 typedef vector<vector<double>> matrix2D;
20
21
22 // Сравнение векторов
23 inline bool operator==(const vector1D& a, const vector1D& b) {
24 #ifdef _DEBUG
25     if (a.size() != b.size())
26         throw std::exception();
27 #endif
28     for (int i = 0; i < a.size(); ++i)
29         if (a[i] != b[i])
30             return false;
31
32     return true;
33 }
34
35 // Сложение векторов
36 inline vector1D operator+(const vector1D& a, const vector1D& b) {
37 #ifdef _DEBUG
38     if (a.size() != b.size())
39         throw std::exception();
40 #endif
41     vector1D result = a;
42     for (int i = 0; i < b.size(); i++)
43         result[i] += b[i];
44     return result;
45 }
46 // Сложение матриц
47 inline matrix2D operator+(const matrix2D& a, const matrix2D& b) {
48 #ifdef _DEBUG
49     if (a.size() != b.size())
50         throw std::exception();
51 #endif
52     matrix2D result = a;
53     for (int i = 0; i < b.size(); i++)
54         for (int j = 0; j < b.size(); j++)
55             result[i][j] += b[i][j];
56     return result;
57 }
58
59
60 // Деление матрицы на число
61 inline matrix2D operator/(const matrix2D& a, const double& b) {
62
63     matrix2D result = a;
64     for (int i = 0; i < a.size(); i++)
65         for (int j = 0; j < a.size(); j++)
66             result[i][j] /= b;

```



```

67     return result;
68 }
69
70
71 // Вычитание векторов
72 inline vector1D operator-(const vector1D& a, const vector1D& b) {
73 #ifdef _DEBUG
74     if (a.size() != b.size())
75         throw std::exception();
76 #endif
77     vector1D result = a;
78     for (int i = 0; i < b.size(); i++)
79         result[i] -= b[i];
80     return result;
81 }
82 // Обратный знак вектора
83 inline vector1D operator-(const vector1D& a) {
84     vector1D result = a;
85     for (int i = 0; i < a.size(); i++)
86         result[i] = -result[i];
87     return result;
88 }
89
90
91
92 // Умножение матрицы на вектор
93 inline vector1D operator*(const matrix2D& a, const vector1D& b) {
94     vector1D result;
95     result.resize(b.size(), 0);
96     for (int i = 0; i < a.size(); i++)
97         for (int j = 0; j < a.size(); j++)
98             result[i] += a[i][j] * b[j];
99     return result;
100 }
101
102 // Умножение матрицы на вектор
103 inline matrix2D operator*(const matrix2D& a, double b) {
104     matrix2D result = a;
105     for (int i = 0; i < a.size(); i++)
106         for (int j = 0; j < a.size(); j++)
107             result[i][j] *= b;
108     return result;
109 }
110 // Умножение на число
111 inline matrix2D operator*(double b, const matrix2D& a) {
112     return operator*(a, b);
113 }
114
115
116 // Умножение на число
117 inline vector1D operator*(const vector1D& a, double b) {
118     vector1D result = a;
119     for (int i = 0; i < result.size(); i++)
120         result[i] *= b;
121     return result;
122 }
123 // Умножение на число
124 inline vector1D operator*(double b, const vector1D& a) {
125     return operator*(a, b);
126 }

```

```

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 // Поточковый 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 // Поточковый 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 // Поточковый вывод вектора для TeX
174 inline void printTeXVector(std::ofstream &fout, const vector1D &v, int coefGrid)
175 {
176     fout << "$(";
177     for (int i = 0; i < v.size() - 1; ++i)
178         if (i % int(pow(2, coefGrid)) == 0)
179             fout << v[i] << ", ";
180     fout << v.back() << ")^T$";
181 }

```

grid.h

```

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

```

```

5 struct NODE {
6
7     bool isFirstNode = false;
8     int i, j;
9     double x, y;
10    int type = -9000;           // -9000    начение при инициализации
11                                   // -1      фиктивный узел
12                                   // 0       внутренний узел
13                                   // n       номер границы
14    int border;                // номер границы
15
16    void setNodesData(double _x, double _y, int _i, int _j, int _type, double
17    _coef) {
18        x = _x;
19        y = _y;
20        i = _i;
21        j = _j;
22        type = _type;
23        if (i % int(pow(2, _coef)) == 0 && j % int(pow(2.0, _coef)) == 0)
24            isFirstNode = true;
25    }
26 };
27
28 class GRID
29 {
30 public:
31     void inputGrid();
32     void inputTime();
33     void buildGrid();
34     void buildTimeGrid();
35     void showGrid();
36     void saveGridAndBorder(const string &filepathGrid, const string &
37     filepathGridBorder);
38 protected:
39
40     int coefGrid, // Сколько раз дробили сетку по пространству
41         coefTime; // Сколько раз дробили сетку по времени
42
43     // Пространство
44     int width;
45     double xLeft, xRight;
46     double hx, nx, kx;
47     double dx;
48
49     int heigth;
50     double yLower, yUpper;
51     double hy, ny, ky;
52     double dy;
53
54     bool isGridUniform;
55     int nodesCount, finiteElementsCount;
56
57     // Время
58     bool isTimeUniform;
59     int tCount;
60     double tFirst, tLast;
61     double ht, nt, kt;
62     double dt;

```

```

63
64     // Узлы
65     vector <NODE> nodes;
66     vector<1D 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;

```

```

55         height = (height - 1) * pow(2, coefGrid) + 1;
56     }
57 }
58 else {
59     if (coefGrid != 0) {
60         width = (width - 1) * pow(2, coefGrid) + 1;
61         height = (height - 1) * pow(2, coefGrid) + 1;
62         nx *= pow(2, coefGrid);
63         ny *= pow(2, coefGrid);
64         kx *= pow(kx, 1.0 / coefGrid);
65         ky *= pow(ky, 1.0 / coefGrid);
66     }
67     hx = (xRight - xLeft) * (1 - kx) / (1 - pow(kx, nx));
68     hy = (yUpper - yLower) * (1 - ky) / (1 - pow(ky, ny));
69 }
70
71
72 nodesCount = width * height;
73 finiteElementsCount = (width - 1) * (height - 1);
74 nodes.resize(nodesCount);
75
76 if (isGridUniform) {
77
78     size_t i, j, elem;
79     double x, y;
80
81     // Внутренние узлы
82     for (size_t j = 1; j < height - 1; j++)
83     {
84         i = 1;
85         for (size_t elem = j * width + 1; elem < (j + 1) * width - 1; elem
++, i++)
86         {
87             x = xLeft + hx * i;
88             y = yLower + hy * j;
89             nodes[elem].setNodesData(x, y, i, j, 0, coefGrid);
90             nodes[elem].border = 0;
91         }
92     }
93
94
95     // 1 Нижняя граница
96     i = 0;
97     j = 0;
98     y = yLower;
99     for (size_t elem = 0; elem < width; elem++, i++)
100     {
101         x = xLeft + hx * i;
102         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
103         nodes[elem].border = 1;
104     }
105
106
107     // 2 Правая граница
108     i = width - 1;
109     j = 1;
110     x = xRight;
111     for (size_t elem = 2 * width - 1; elem < width * height; elem += width,
j++)
112     {

```

```

113         y = yLower + hy * j;
114         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
115         nodes[elem].border = 2;
116     }
117
118
119     // 3 Верхняя граница
120     i = 0;
121     j = height - 1;
122     y = yUpper;
123     for (size_t elem = width * j; elem < (j + 1) * width - 1; elem++, i++)
124     {
125         x = xLeft + hx * i;
126         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
127         nodes[elem].border = 3;
128     }
129
130
131     // 4 Левая граница
132     i = 0;
133     j = 1;
134     x = xLeft;
135     for (size_t elem = width; elem < (height - 1) * width; elem += width, j
136 ++, i++)
137     {
138         y = yLower + hy * j;
139         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
140         nodes[elem].border = 4;
141     }
142 }
143 else {
144
145     size_t i, j, elem;
146     double x, y;
147
148     // Внутренние узлы
149     dy = hy * ky;
150     y = yLower + hy;
151     for (size_t j = 1; j < height - 1; j++, dy *= ky)
152     {
153         i = 1;
154         dx = hx * kx;
155         x = xLeft + hx;
156         for (size_t elem = j * width + 1; elem < (j + 1) * width - 1; elem
157 ++, i++, dx *= kx)
158         {
159             nodes[elem].setNodesData(x, y, i, j, 0, coefGrid);
160             nodes[elem].border = 0;
161             x += dx;
162         }
163         y += dy;
164     }
165
166     // 1 Нижняя граница
167     i = 0;
168     dx = hx;
169     x = xLeft;
170

```

```

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 * height; elem += width,
190 j++, dy *= ky)
191     {
192         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
193         nodes[elem].border = 2;
194         y += dy;
195     }
196
197     // 3 Верхняя граница
198     i = 0;
199     dx = hx;
200     x = xLeft;
201
202     j = height - 1;
203     y = yUpper;
204     for (size_t elem = width * j; elem < (j + 1) * width - 1; elem++, i++,
205 dx *= kx)
206     {
207         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
208         nodes[elem].border = 3;
209         x += dx;
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 < (height - 1) * width; elem += width, j
221 ++, dy *= ky)
222     {
223         nodes[elem].setNodesData(x, y, i, j, 1, coefGrid);
224         nodes[elem].border = 4;
225         y += dy;
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 // Отображение сетки на экран
284 void GRID::showGrid() {
285
286     for (size_t i = 0; i < width; i++)
287         cout << nodes[i].x << " ";

```



```

288 }
289
290
291 // Сохранение внутренних и внешних узлов в 2 файлах
292 void GRID::saveGridAndBorder(const string &filepathGrid, const string &
    filepathGridBorder) {
293
294     ofstream grid(filepathGrid);
295     ofstream border(filepathGridBorder);
296     for (size_t i = 0; i < nodesCount; i++)
297     {
298         if (nodes[i].type > 0)
299             border << nodes[i].x << " " << nodes[i].y << endl;
300         else
301             grid << nodes[i].x << " " << nodes[i].y << endl;
302     }
303
304     border.close();
305     grid.close();
306 }

```

fem.h

```

1 #pragma once
2 #include "head.h"
3 #include "grid.h"
4 #include "solver.h"
5
6
7 class FEM : public GRID, public SOLVER {
8 public:
9
10     void init(
11         function3D _u,
12         function3D _f,
13         double _lambda,
14         double _gamma,
15         double _sigma,
16         double _chi,
17         bool _isGridUniform,
18         bool _isTimeUniform,
19         int _coefGrid,
20         int _coefTime
21     );
22     double solve();
23     inline int getNodesCount() { return nodesCount; }
24     void convAToDense();
25     void outputA();
26     void outputG();
27     void outputM();
28
29
30 protected:
31     double p00, p01, p10, p11,
32         c00, c01, c10, c11;
33     function3D u, f;
34     double lambda, gamma, sigma, chi;
35     double t, dt;
36     double t0, t1, t2;
37     double t01, t02, t12;
38
39     double calcNormAtMainNodes(const vector1D &x, int time) {

```

```

40     double normE = 0;
41     for (size_t i = 0; i < x.size(); i++)
42         normE += pow((x[i] - u(nodes[i].x, nodes[i].y, time)), 2);
43     return sqrt(normE) / nodes.size();
44 }
45
46 void implicitCheme3(int timeLayer);
47 void buildGlobalMatrixG();
48 void buildGlobalMatrixM();
49 vector1D buildGlobalVectorb(int timeLayer);
50
51 void buildLocalVectorb(int elemNumber);
52 void buildLocalmatrixG(int elemNumber);
53 void buildLocalmatrixM(int elemNumber);
54
55 double f1, f2, f3, f4;
56 matrix2D A, G, M;
57 matrix2D Glocal, Mlocal, tmpMatrix;
58 vector1D blocal, tmpVector;
59 };

```

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
110     for (size_t i = 0; i < heigth; i++)
111         for (size_t j = 0; j < width; j++)
112             {
113                 int k = i * width + j;
114                 q2[k] = u(nodes[k].x, nodes[k].y, times[0]);
115                 q1[k] = u(nodes[k].x, nodes[k].y, times[1]);
116             }
117
118     b2 = buildGlobalVectorb(0);
119     b1 = buildGlobalVectorb(1);
120
121     for (size_t timelayer = 2; timelayer < times.size(); timelayer++) {
122         implicitCheme3(timelayer);
123
124         for (int j = 0; j < A.size(); ++j) {
125             for (int i = j + 1; i < A.size(); ++i) {
126
127                 double toMult = A[i][j] / A[j][j];
128
129                 for (int k = 0; k < A.size(); ++k)
130                     A[i][k] -= toMult * A[j][k];
131
132                 b[i] -= toMult * b[j];
133             }
134         }
135
136         vector <double> x;
137         x.resize(A.size(), 0);
138
139         for (int i = n - 1; i >= 0; --i) {
140
141             double tmp = 0.0;
142             for (int j = i + 1; j < A.size(); ++j) {
143                 tmp += A[i][j] * x[j];
144             }
145             x[i] = (b[i] - tmp) / A[i][i];
146         }
147         q = x;
148
149         b2 = b1;
150         b1 = b;
151
152         q2 = q1;
153         q1 = q;
154
155         //cout << endl << "q:" << endl << q << endl;
156         //cout << endl << "Residual: " << endl << calcNormAtMainNodes(q, t0);
157     }
158 }
159 return calcNormAtMainNodes(q, t0);

```

```

160 }
161
162
163
164 //
165 //
166 //
167
168
169 void FEM::implicitCheme3(int timeLayer)
170 {
171     A.clear();
172     A.resize(nodesCount);
173     for (size_t i = 0; i < nodesCount; i++)
174         A[i].resize(nodesCount, 0);
175
176     buildGlobalMatrixG();
177     buildGlobalMatrixM();
178     b = buildGlobalVectorb(timeLayer);
179
180     t0 = times[timeLayer];
181     t1 = times[timeLayer - 1];
182     t2 = times[timeLayer - 2];
183
184
185     t01 = t0 - t1;
186     t02 = t0 - t2;
187     t12 = t1 - t2;
188
189
190     // Собираем левую часть
191     double tmp = 2.0 * chi / (t01 * t02)
192         + sigma * (t02 + t01) / (t02 * t01) + gamma;
193     for (size_t i = 0; i < nodesCount; i++)
194         for (size_t j = 0; j < nodesCount; j++)
195             A[i][j] += M[i][j] * tmp + G[i][j];
196
197     // Собираем правую часть
198     b = b - (2 * chi / (t02 * t12) + sigma * t01 / (t02 * t12)) * M * q2
199         + (2 * chi / (t01 * t12) + sigma * t02 / (t01 * t12)) * M * q1;
200
201     //outputA();
202     //cout << endl << "b:" << endl << b << endl;
203
204     // Добавляем краевые условия
205     for (size_t i = 0; i < nodesCount; i++)
206     {
207         if (nodes[i].type == 1) {
208             A[i].clear();
209             A[i].resize(nodesCount, 0);
210             A[i][i] = 1;
211             b[i] = u(nodes[i].x, nodes[i].y, t0);
212         }
213     }
214
215     //cout << endl << "Added 1st boundary conditions:" << endl;
216     //cout << endl << "A:" << endl;
217     //outputA();
218     //cout << endl << "b:" << endl << b << endl;
219 }

```

```

220
221
222
223
224 void FEM::buildGlobalMatrixG()
225 {
226     G.clear();
227     G.resize(nodesCount);
228     for (size_t i = 0; i < nodesCount; i++)
229         G[i].resize(nodesCount, 0);
230
231
232     for (size_t i = 0; i < height - 1; i++)
233     {
234         for (size_t j = 0; j < width - 1; j++)
235         {
236             int k = i * width + j;
237             buildLocalmatrixG(k);
238
239             vector <int> nearestNodesIndexes = { k, k + 1, k + width, k + width
+ 1 };
240             for (size_t i1 = 0; i1 < 4; i1++)
241                 for (size_t j1 = 0; j1 < 4; j1++)
242                     G[nearestNodesIndexes[i1]][nearestNodesIndexes[j1]] +=
GLocal[i1][j1];
243         }
244     }
245 }
246
247
248
249 void FEM::buildGlobalMatrixM()
250 {
251     M.clear();
252     M.resize(nodesCount);
253     for (size_t i = 0; i < nodesCount; i++)
254         M[i].resize(nodesCount, 0);
255
256     for (size_t i = 0; i < height - 1; i++)
257     {
258         for (size_t j = 0; j < width - 1; j++)
259         {
260             int k = i * width + j;
261             buildLocalmatrixM(k);
262
263             vector <int> nearestNodesIndexes = { k, k + 1, k + width, k + width
+ 1 };
264             for (size_t i1 = 0; i1 < 4; i1++)
265                 for (size_t j1 = 0; j1 < 4; j1++)
266                     M[nearestNodesIndexes[i1]][nearestNodesIndexes[j1]] +=
MLocal[i1][j1];
267         }
268     }
269 }
270
271
272
273
274 // Строим глобальный вектор правой части системы нелинейных уравнений
275 vector1D FEM::buildGlobalVectorb(int timeLayer)

```

```

276 {
277     t = times[timeLayer];
278     tmpVector.clear();
279     tmpVector.resize(nodesCount, 0);
280
281
282     for (size_t i = 0; i < height - 1; i++)
283     {
284         for (size_t j = 0; j < width - 1; j++)
285         {
286             int k = i * width + j;
287             buildLocalVectorb(k);
288
289             vector<int> nearestNodesIndexes = { k, k + 1, k + width, k + width
+ 1 };
290             for (size_t i1 = 0; i1 < 4; i1++)
291                 tmpVector[nearestNodesIndexes[i1]] += bLocal[i1];
292         }
293     }
294
295     return tmpVector;
296 }
297
298
299 //-----
300 //-----
301 //-----
302
303
304
305 // Построение локальной матрицы жёсткости
306 void FEM::buildLocalmatrixG(int elemNumber)
307 {
308     hx = nodes[elemNumber + 1].x - nodes[elemNumber].x;
309     hy = nodes[elemNumber + width].y - nodes[elemNumber].y;
310
311     GLocal = { {0, 0, 0, 0},
312                {0, 0, 0, 0},
313                {0, 0, 0, 0},
314                {0, 0, 0, 0} };
315
316     tmpMatrix = { {2, -2, 1, -1},
317                   {-2, 2, -1, 1},
318                   {1, -1, 2, -2},
319                   {-1, 1, -2, 2} };
320
321     for (size_t i = 0; i < 4; i++)
322         for (size_t j = 0; j < 4; j++)
323             GLocal[i][j] += tmpMatrix[i][j] * lambda * hy / (6 * hx);
324
325     tmpMatrix = { {2, 1, -2, -1},
326                   {1, 2, -1, -2},
327                   {-2, -1, 2, 1},
328                   {-1, -2, 1, 2} };
329
330
331     for (size_t i = 0; i < 4; i++)
332         for (size_t j = 0; j < 4; j++)
333             GLocal[i][j] += tmpMatrix[i][j] * lambda * hx / (6 * hy);
334 }

```

```

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

```

solver.h

```

1 #pragma once
2 #include "head.h"
3
4
5 // Система линейных алгебраических уравнений
6 class SOLVER {
7
8 public:
9     void initSLAE();
10    void BiCG();
11
12
13    void getVectX(vector1D &x) { x = b; };
14    void generateVectX(int size);
15    void writeXTToFile(const char *fileName);
16    void writeXTostream(std::ofstream& fout);
17    void writeFTToFile(const char *fileName);
18
19
20    int getDimention() { return n; }
21    void decompositionD();
22    void decompositionLUSq();
23    vector1D execDirectTraversal(const vector1D &_F);

```



```

24     vector1D execReverseTraversal(const vector1D &_y);
25
26     pair<int, double> LOS();
27     pair<int, double> LOSfactD();
28     pair<int, double> LOSfactLUsq();
29     void clearAll();
30     void setMaxiter(int new_maxiter) { maxiter = new_maxiter; }
31     void setE(double new_E) { E = new_E; }
32
33 protected:
34     vector1D q, q1, q2, q3;
35     vector1D di, al, au, di_f, al_f, au_f;
36     vector<int> ia, ja;
37     int n, maxiter;
38     double E, delta;
39     vector1D x, r, z, p, b, b1, b2, b3, bTmp;
40
41
42     vector1D multA(const vector1D&x);
43     vector1D multD(const vector1D&x);
44     double calcRelativeDiscrepancy();
45     double calcNormE(const vector1D &x) { return sqrt(x*x); }
46
47
48     vector1D xPrev, xPrevPrev; // решение на k-1 итерации
49
50     vector1D s; // вспомогательный вектор
51     double alpha, beta;
52
53
54     void generateInitialGuess();
55     vector1D multAOn(const vector1D &v);
56     vector1D multAtOn(const vector1D &v);
57     bool doStop(int i);
58 };

```

solver.cpp

```

1  #include "solver.h"
2
3
4
5
6  // A*x = b, где x — произвольный вектор
7  vector1D SOLVER::multA(const vector1D& x) {
8
9      vector1D result(n);
10     for (int i = 0; i < n; ++i) {
11
12         result[i] = di[i] * x[i];
13         int i0 = ia[i];
14         int i1 = ia[i + 1];
15
16         for (int k = i0; k < i1; ++k) {
17
18             result[i] += al[k] * x[ja[k]];
19             result[ja[k]] += au[k] * x[i];
20         }
21     }
22
23     return result;
24 }

```

```

25
26 //  $A \cdot x = b$ , где  $x$  — произвольный вектор
27 vector1D SOLVER::multD(const vector1D&x) {
28
29     vector1D result(n);
30
31     for (int i = 0; i < n; ++i)
32         result[i] = di_f[i] * x[i];
33
34     return result;
35 }
36
37
38 // Создаём вектор  $x^* = (1, 2, \dots, n)'$ 
39 void SOLVER::generateVectX(int size) {
40
41     x.resize(size);
42     for (int i = 0; i < size; ++i) {
43         x[i] = i + 1;
44     }
45 }
46
47
48 // Вывод вектора  $b$  в файл
49 void SOLVER::writeFToFile(const char *fileName) {
50
51     std::ofstream fout;
52     fout.open(fileName);
53
54     for (int i = 0; i < b.size(); ++i)
55         fout << b[i] << endl;
56     fout.close();
57 }
58
59
60 // Вывод вектора  $x$  в файл
61 void SOLVER::writeXToFile(const char * fileName) {
62
63     std::ofstream fout;
64     fout.open(fileName);
65     for (int i = 0; i < x.size(); ++i)
66         fout << x[i] << " ";
67     fout << "\t";
68     fout.close();
69 }
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::decompositionD() {
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 эквивалентно
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
126                    // перемножить
127                    if (ja[ki] > ja[kj]) kj++;
128                    else ki++;
129                }
130
131                al_f[k] = (al_f[k] - sum_l) / di_f[j];
132                au_f[k] = (au_f[k] - sum_u) / di_f[j];
133            }
134
135            // Рассчёт диагонального элемента
136            sum_d = 0.0;
137            for (int k = i0; k < i1; ++k)
138                sum_d += al_f[k] * au_f[k];
139            di_f[i] = sqrt(di_f[i] - sum_d);
140        }
141    }
142 }
143

```

```

144
145 // Прямой ход     $L y = b \implies y = L^{-1} b$ 
146 vector1D SOLVER::execDirectTraversal(const vector1D &_F) {
147
148     vector1D y;
149     y.resize(n, 0);
150
151     for (int i = 0; i < n; ++i) {
152         double sum = 0;
153         int i0 = ia[i];
154         int i1 = ia[i + 1];
155         for (int k = i0; k < i1; ++k)
156             sum += al_f[k] * y[ja[k]];
157
158         y[i] = (_F[i] - sum) / di_f[i];
159     }
160     return y;
161 }
162
163
164 // Обратный ход     $U(sq) x = y \implies x = U(sq)^{-1} y$ 
165 vector1D SOLVER::execReverseTraversal(const vector1D &_y) {
166
167     vector1D x, y = _y;
168     x.resize(n);
169     for (int i = n - 1; i >= 0; --i) {
170
171         x[i] = y[i] / di_f[i];
172         int i0 = ia[i];
173         int i1 = ia[i + 1];
174         for (int k = i0; k < i1; ++k)
175             y[ja[k]] -= au_f[k] * x[i];
176     }
177
178     return x;
179 }
180
181
182
183 // Полная очистка СЛАУ
184 void SOLVER::clearAll() {
185
186     n = 0;
187     E = 0.0;
188     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         double relativeDiscrepancy = calcRelativeDiscrepancy();
246         if (x == xprev || relativeDiscrepancy < E) {
247             return make_pair(i, relativeDiscrepancy);
248         }
249         xprev = x;
250     }
251 }
252
253
254
255 // Локально — оптимальная схема с неполной диагональной факторизацией
256 pair<int, double> SOLVER::LOSfactD() {
257
258     x.clear();           // Задаём начальное приближение
259     x.resize(n, 0);      // x_0 = (0, 0, ...)
260     vector1D xprev = x;
261     decompositionD();
262
263     r = b - multA(x);    // r_0 = b - A*x_0

```

```

264     r = multD(r);
265     z = multD(r);           // z = U-1 r
266     p = multA(z);           // p = A*z
267     p = multD(p);           // p = L-1 A*z
268
269
270     for (int i = 0; i < maxiter; ++i) {
271
272         double pp = p * p;
273         double alpha = (p*r) / pp;
274         x = x + alpha * z;
275         r = r - alpha * p;
276
277         vector1D tmp = multD(r);
278         tmp = multA(tmp);
279         tmp = multD(tmp);
280         double beta = -(p * tmp) / pp;
281         p = tmp + beta * p;
282
283         tmp = multD(r);
284         z = tmp + beta * z;
285
286
287         double relativeDiscrepancy = calcRelativeDiscrepancy();
288         if (x == xprev || relativeDiscrepancy < E) {
289             return make_pair(i, relativeDiscrepancy);
290         }
291         xprev = x;
292     }
293 }
294
295
296
297
298
299 // Локально — оптимальная схема с неполной факторизацией LU(sq)
300 pair<int, double> SOLVER::LOSfactLUsq() {
301
302     x.clear();                // Задаём начальное приближение
303     x.resize(n, 0);           // x_0 = (0, 0, ...)
304     vector1D xprev = x;
305     decompositionLUsq();
306
307     r = b - multA(x);          // r_0 = b - A*x_0
308     r = execDirectTraversal(r); // r = L-1 (b - A*x_0)
309     z = execReverseTraversal(r); // z = U-1 r
310     p = multA(z);              // p = A*z
311     p = execDirectTraversal(p); // p = L-1 A*z
312
313
314     for (int i = 0; i < maxiter; ++i) {
315
316         double pp = p * p;
317         double alpha = (p*r) / pp;
318         x = x + alpha * z;
319         r = r - alpha * p;
320
321         vector1D tmp = execReverseTraversal(r);
322         tmp = multA(tmp);
323         tmp = execDirectTraversal(tmp);

```

```

324     double beta = -(p * tmp) / pp;
325     p = tmp + beta * p;
326
327     tmp = execReverseTraversal(r);
328     z = tmp + beta * z;
329
330
331     double relativeDiscrepancy = calcRelativeDiscrepancy();
332     if (x == xprev || relativeDiscrepancy < E) {
333         return make_pair(i, relativeDiscrepancy);
334     }
335     xprev = x;
336 }
337 }
338
339
340
341
342 // Считываем все СЛАУ и её параметры из файлов
343 void SOLVER::initSLAE()
344 {
345     ifstream finMatrix("input/matrix.txt");
346     ifstream finVector("input/vector.txt");
347     ifstream finParams("input/SLAE_parameters.txt");
348
349     finParams >> maxiter >> E >> delta;
350     finMatrix >> n;
351     di.resize(n);
352     ia.resize(n + 1);
353
354     for (size_t i = 0; i < n; i++)
355         finMatrix >> di[i];
356     for (size_t i = 0; i < n + 1; i++)
357         finMatrix >> ia[i];
358
359     ja.resize(ia[n]);
360     al.resize(ia[n]);
361     au.resize(ia[n]);
362
363     for (size_t i = 0; i < ja.size(); i++)
364         finMatrix >> ja[i];
365     for (size_t i = 0; i < al.size(); i++)
366         finMatrix >> al[i];
367     for (size_t i = 0; i < au.size(); i++)
368         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  $x_0 = (0, \dots, 0)'$ 
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)
429 {
430     vector1D result(n);
431     for (size_t i = 0; i < n; i++)
432     {
433         result[i] = di[i] * v[i];
434         int i0 = ia[i];
435         int i1 = ia[i + 1];
436
437         for (size_t k = i0; k < i1; k++)
438         {
439             int j = ja[k];
440             result[i] += al[k] * v[j];
441             result[j] += au[k] * v[i];
442         }
443     }

```



```

444     return result;
445 }
446
447
448
449 // Умножение транспонированной матрицы A на вектор
450 vector1D SOLVER::multAtOn(const vector1D &v)
451 {
452     vector1D result(n);
453     for (size_t i = 0; i < n; i++)
454     {
455         result[i] = di[i] * v[i];
456         int i0 = ia[i];
457         int i1 = ia[i + 1];
458         for (size_t k = i0; k < i1; k++)
459         {
460             int j = ja[k];
461             result[j] += al[k] * v[i];
462             result[i] += au[k] * v[j];
463         }
464     }
465     return result;
466 }
467
468
469
470 // Проверяем условие выхода
471 bool SOLVER::doStop(int i)
472 {
473     // Выход по числу итераций
474     if (i > maxiter)
475         return true;
476
477     // Выход шагу
478     if (calcNormE(x - xPrev) / calcNormE(x) < delta)
479         return true;
480
481     // Выход по относительной невязке
482     if (calcNormE(multAOn(x) - b) / calcNormE(b) < E)
483         return true;
484
485     return false;
486 }

```

main.cpp

```

1 #include "fem.h"
2 #include <thread>
3
4 function1D calcFirstDerivative(const function1D& f) {
5     return [f](double x) -> double {
6         const double h = 0.001;
7         return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) /
8             (12 * h);
9     };
10 }
11
12 function1D calcSecondDerivative(const function1D& f) {
13     return [f](double x) -> double {
14         const double h = 0.001;
15         return (-f(x + 2 * h) + 16 * f(x + h) - 30 * f(x) + 16 * f(x - h) - f(x

```

```

    - 2 * h)) / (12 * h*h);
16     };
17 }
18
19
20 function3D calc_f(
21     const function3D& u,
22     double lambda,
23     double gamma,
24     double sigma,
25     double chi
26 ) {
27     return [=](double x, double y, double t) -> double
28     {
29         using namespace placeholders;
30         auto u_t = calcFirstDerivative(bind(u, x, y, _1));
31
32         auto u_xx = calcSecondDerivative(bind(u, _1, y, t));
33         auto u_yy = calcSecondDerivative(bind(u, x, _1, t));
34         auto u_tt = calcSecondDerivative(bind(u, x, y, _1));
35
36         return -lambda * (u_xx(x) + u_yy(y)) + gamma * u(x, y, t) + sigma * u_t(
37             t) + chi * u_tt(t);
38     };
39 }
40
41 function3D sum_u(
42     const function3D& u_x,
43     const function3D& u_t,
44     double lambda,
45     double gamma,
46     double sigma,
47     double chi
48 ) {
49     return [=](double x, double y, double t) -> double
50     {
51         return (u_x(x, y, t) + u_t(x, y, t));
52     };
53 }
54
55
56 void main() {
57
58     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) + ".
73     txt";

```



```
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 }
```