

Министерство науки и высшего образования Российской  
Федерации

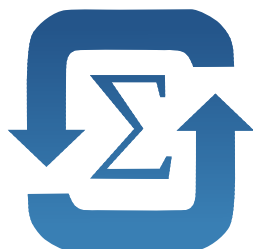
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ**»



Кафедра прикладной математики

Уравнения математической физики

**Пояснительная записка к курсовому проекту**



Факультет:	ПМИ
Группа:	ПМ-63
Студент:	Шепрут И.И.
Преподаватель:	Персова М.Г.

Новосибирск  
2019

# 1 Задание

Реализовать МКЭ для двумерной задачи для гиперболического уравнения в декартовой системе координат. Базисные функции — билинейные. Схема Кранка-Николсона.

# 2 Теория

Решаемое уравнение в общем виде:

$$-\operatorname{div}(\lambda \operatorname{grad} u) + \gamma u + \sigma \frac{\partial u}{\partial t} + \chi \frac{\partial^2 u}{\partial t^2} = f$$

Решаемое уравнение в декартовой двумерной системе координат:

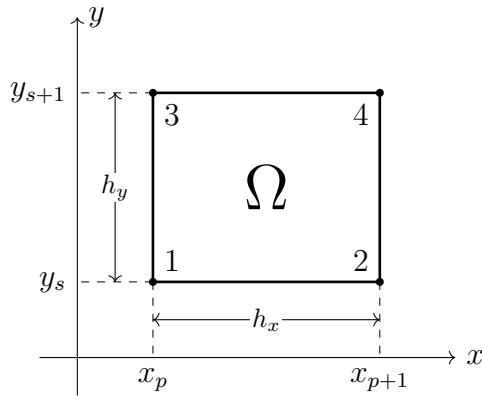
$$-\frac{\partial}{\partial x} \left( \lambda \frac{\partial u}{\partial x} \right) - \frac{\partial}{\partial y} \left( \lambda \frac{\partial u}{\partial y} \right) + \gamma u + \sigma \frac{\partial u}{\partial t} + \chi \frac{\partial^2 u}{\partial t^2} = f$$

Первые краевые условия:

$$u|_S = u_s$$

Формулы для билинейных базисных функций прямоугольных элементов:

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



$$\psi_4(x, y) = X_2(x)Y_2(y)$$

$$\psi_3(x, y) = X_2(x)Y_1(y)$$

$$\psi_2(x, y) = X_1(x)Y_2(y)$$

$$\psi_1(x, y) = X_1(x)Y_1(y)$$

И значение конечно-элементной аппроксимации на этом конечном элементе равно:

$$u_{ps}^*(x, y) = \sum_{i=1}^4 q_i \psi_i(x, y)$$

Аналитические выражения для вычисления элементов локальных матриц:

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

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

Вычисленные матрицы для билинейных прямоугольных элементов:

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

$$\mathbf{C} = \frac{h_x h_y}{36} \begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix} \quad \mathbf{M}^\gamma = \bar{\gamma} \mathbf{C}$$

$$\mathbf{f} = (f_1, f_2, f_3, f_4)^t \quad \mathbf{b} = \mathbf{C} \cdot \mathbf{f}$$

Схема Кранка-Николсона:

$$\frac{\partial u}{\partial t} = \frac{u^j - u^{j-2}}{2\Delta t}, \quad \frac{\partial^2 u}{\partial t^2} = \frac{u^j - 2u^{j-1} + u^{j-2}}{\Delta t^2}$$

$$u = \frac{u^j + u^{j-2}}{2}, \quad f = \frac{f^j + f^{j-2}}{2}$$

$$-\operatorname{div} \left( \lambda \operatorname{grad} \frac{u^j + u^{j-2}}{2} \right) + \gamma \frac{u^j + u^{j-2}}{2} + \sigma \frac{u^j - u^{j-2}}{2\Delta t} + \chi \frac{u^j - 2u^{j-1} + u^{j-2}}{\Delta t^2} = \frac{f^j + f^{j-2}}{2}$$

Подставляя это в уравнение Галёркина, получаем СЛАУ из глобальных матриц:

$$\left( \frac{\mathbf{G}}{2} + \frac{\mathbf{M}^\gamma}{2} + \frac{\mathbf{M}^\sigma}{2\Delta t} + \frac{\mathbf{M}^\chi}{\Delta t^2} \right) \mathbf{q}^j = \frac{(\mathbf{b}^j + \mathbf{b}^{j-2})}{2} - \frac{\mathbf{G} \mathbf{q}^{j-2}}{2} - \frac{\mathbf{M}^\gamma \mathbf{q}^{j-2}}{2} + \frac{\mathbf{M}^\sigma \mathbf{q}^{j-2}}{2\Delta t} - \frac{\mathbf{M}^\chi (-2\mathbf{q}^{j-1} + \mathbf{q}^{j-2})}{\Delta t^2}$$

В нашем случае, так как  $\gamma, \sigma, \chi$  являются константами, можно записать:

$$\left( \frac{\mathbf{G}}{2} + \mathbf{C} \left( \frac{\gamma}{2} + \frac{\sigma}{2\Delta t} + \frac{\chi}{\Delta t^2} \right) \right) \mathbf{q}^j = \frac{(\mathbf{b}^j + \mathbf{b}^{j-2})}{2} - \frac{\mathbf{G} \mathbf{q}^{j-2}}{2} + \mathbf{C} \left( \mathbf{q}^{j-1} \frac{2\chi}{\Delta t^2} + \mathbf{q}^{j-2} \left( -\frac{\gamma}{2} + \frac{\sigma}{2\Delta t} - \frac{\chi}{\Delta t^2} \right) \right)$$

Для неравномерной же сетки по времени имеем только отличие в:

$$t_2 = t^{j-2}, \quad t_1 = t^{j-1}, \quad t_0 = t^j$$

$$\frac{\partial u}{\partial t} = \frac{u^j - u^{j-2}}{t_2 - t_1} = \frac{u^j - u^{j-2}}{d_1}$$

$$\frac{\partial^2 u}{\partial t^2} = 2 \frac{u^j - u^{j-1} \frac{t_0 - t_2}{t_1 - t_2} + u^{j-2} \frac{t_0 - t_1}{t_1 - t_2}}{t_0(t_0 - t_1 - t_2) + t_1 t_2} = \frac{u^j - u^{j-1} m_1 + u^{j-2} m_2}{d_2}$$

Эти выражения были упрощены при помощи замен:

$$d_1 = t_0 - t_2, \quad d_2 = \frac{t_0(t_0 - t_1 - t_2) + t_1 t_2}{2}, \quad m_1 = \frac{t_0 - t_2}{t_1 - t_2}, \quad m_2 = \frac{t_0 - t_1}{t_1 - t_2}$$

И итоговый результат будет:

$$\left( \frac{\mathbf{G}}{2} + \mathbf{C} \left( \frac{\gamma}{2} + \frac{\sigma}{d_1} + \frac{\chi}{d_2} \right) \right) \mathbf{q}^j = \frac{(\mathbf{b}^j + \mathbf{b}^{j-2})}{2} - \frac{\mathbf{G} \mathbf{q}^{j-2}}{2} + \mathbf{C} \left( \mathbf{q}^{j-1} \frac{m_1 \chi}{d_2} + \mathbf{q}^{j-2} \left( -\frac{\gamma}{2} + \frac{\sigma}{d_1} - \frac{m_2 \chi}{d_2} \right) \right)$$

### 3 Структуры данных

Для задания сетки используется класс:

```
1 class grid_generator_t
2 {
3 public:
4     grid_generator_t(double a, double b, int n, double t = 0);
5     double operator()(int i) const;
6     int size(void) const;
7     double back(void) const;
8 private:
9     double a, len, t, n1;
10 };
```

Для задания узла конечного элемента структура:

```
1 struct basic_elem_t
2 {
3     int i; /// Номер узла
4
5     double x, y; /// Координаты узла
6
7     basic_elem_t *up, *down, *left, *right; /// Указатели на соседей узла
8
9     /** Проверяет, является ли элемент граничным. Он таким является, если у него нет хотя бы одного
10     ↪ соседа. */
11     bool is_boundary(void) const;
12 };
```

Для задания конечного элемента используется структура:

```
1 struct elem_t
2 {
3     int i; /// Номер конечного элемента
4     basic_elem_t* e[4]; /// Указатели на все 4 элемента конечного узла, нумерация такая:
5     /**
6         Y
7         ^ 3 +-----+ 4
8         |   |       |
9         |   1 +-----+ 2
10        +-----+ X
11        |
12        */
13
14     double get_hx(void) const; /// Ширина конечного элемента
15     double get_hy(void) const; /// Высота конечного элемента
16
17     /** Рассчитать значение внутри конечного элемента. q - вектор рассчитываемых весов. */
18     double value(double x, double y, const vector_t& q) const;
19 };
```

Прямоугольная сетка задается и вычисляется с помощью класса:

```
1 class grid_t
2 {
3 public:
4     vector<elem_t> es; /// Массив конечных элементов сетки
5     vector<basic_elem_t> bes; /// Массив узлов сетки
6     int n; /// Число узлов
7
8     /** Рассчитать неравномерную сетку. */
9     void calc(const grid_generator_t& gx, const grid_generator_t& gy);
10 };
```

Локальные матрицы формируются, получая на вход конечный элемент **elem\_t**.

Для генерации разреженной матрицы используется класс с возможностью произвольного доступа к элементам:

```

1 class matrix_sparse_ra_t
2 {
3 public:
4     matrix_sparse_ra_t(int n);
5
6     /** Установить значение в позиции (i, j) */
7     double& operator()(int i, int j);
8
9     /** Получить значение в позиции (i, j). Если туда ещё не устанавливалось значение, вызывается
10     ↪ исключение. */
11     const double& operator()(int i, int j) const;
12
13     /** Преобразует текущую матрицу к разреженной матрице. */
14     matrix_sparse_t to_sparse(void) const;
15 private:
16     int n;
17     vector<double> dm;
18     vector<map<int, double>> lm, um;
19 };

```

## 4 Исследования

Во всех исследованиях заданы следующие параметры  $\lambda = \gamma = \sigma = \chi = 1$ .

СЛАУ решается при помощи Локально-Оптимальной Схемы (ЛОС) с неполным LU предобуславливанием.

### 4.1 Таблицы

Далее в таблицах будут указаны две функции:  $\text{space}(x, y)$  и  $\text{time}(t)$ , итоговая функция  $u$  будет формироваться из них:  $u(x, y, t) = \text{space}(x, y) + \text{time}(t)$ .

В таблицах для каждой функции указано три значения:

- Интеграл разности между истинной функцией и конечно-элементной аппроксимацией.
- Норма разности векторов  $q$  для найденного решения и  $q$ , полученного из истинного значения функции.
- Время решения в миллисекундах.

#### 4.1.1 10 на 10 на 10

Сетка по пространству:  $(x, y) \in [0, 1] \times [0, 1]$ , строк и столбцов 10. Сетка по времени:  $t \in [0, 1]$ , количество элементов сетки 10. Все сетки равномерные.

space(x, y) \ time(t)	0	$t$	$t^2$	$t^3$	$t^4$	$e^t$
1	$0.36 \cdot 10^{-15}$ $0.4 \cdot 10^{-16}$ 25	$0.36 \cdot 10^{-11}$ $0.31 \cdot 10^{-12}$ 15	$0.34 \cdot 10^{-11}$ $0.29 \cdot 10^{-12}$ 16	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 15	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 19	$0.61 \cdot 10^{-3}$ $0.56 \cdot 10^{-4}$ 34
$x + y$	$0.22 \cdot 10^{-11}$ $0.24 \cdot 10^{-12}$ 18	$0.35 \cdot 10^{-11}$ $0.32 \cdot 10^{-12}$ 19	$0.24 \cdot 10^{-11}$ $0.27 \cdot 10^{-12}$ 19	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 20	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 15	$0.61 \cdot 10^{-3}$ $0.56 \cdot 10^{-4}$ 36
$x^2 + y$	$0.28 \cdot 10^{-2}$ $0.17 \cdot 10^{-12}$ 16	$0.28 \cdot 10^{-2}$ $0.1 \cdot 10^{-12}$ 16	$0.28 \cdot 10^{-2}$ $0.16 \cdot 10^{-12}$ 26	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 19	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 16	$0.34 \cdot 10^{-2}$ $0.56 \cdot 10^{-4}$ 44
$x^2y + y^3$	$0.28 \cdot 10^{-2}$ $0.12 \cdot 10^{-12}$ 15	$0.28 \cdot 10^{-2}$ $0.16 \cdot 10^{-12}$ 15	$0.28 \cdot 10^{-2}$ $0.26 \cdot 10^{-12}$ 16	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 16	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 17	$0.34 \cdot 10^{-2}$ $0.56 \cdot 10^{-4}$ 35
$xy^2$	$0.69 \cdot 10^{-3}$ $0.65 \cdot 10^{-13}$ 19	$0.69 \cdot 10^{-3}$ $0.12 \cdot 10^{-12}$ 16	$0.69 \cdot 10^{-3}$ $0.92 \cdot 10^{-13}$ 15	$0.14 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 15	$0.14 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 15	$0.13 \cdot 10^{-2}$ $0.56 \cdot 10^{-4}$ 32
$x^4 + y^4$	$0.43 \cdot 10^{-2}$ $0.14 \cdot 10^{-3}$ 18	$0.43 \cdot 10^{-2}$ $0.14 \cdot 10^{-3}$ 20	$0.43 \cdot 10^{-2}$ $0.14 \cdot 10^{-3}$ 16	$0.48 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 16	$0.48 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 16	$0.47 \cdot 10^{-2}$ $0.83 \cdot 10^{-4}$ 32
$e^{xy}$	$0.68 \cdot 10^{-3}$ $0.12 \cdot 10^{-5}$ 32	$0.68 \cdot 10^{-3}$ $0.12 \cdot 10^{-5}$ 45	$0.68 \cdot 10^{-3}$ $0.12 \cdot 10^{-5}$ 41	$0.14 \cdot 10^{-2}$ $0.68 \cdot 10^{-4}$ 31	$0.14 \cdot 10^{-2}$ $0.68 \cdot 10^{-4}$ 32	$0.13 \cdot 10^{-2}$ $0.54 \cdot 10^{-4}$ 47

**Вывод:** полностью (на всей области конечных элементов, а не только в узлах) аппроксимируются только линейные функции по пространству и для степени  $t$  равной 0, 1 или 2.

**Вывод:** значения в узлах полностью аппроксимируются только до полиномов 3 степени включительно по пространству.

**Вывод:** порядок аппроксимации по пространству — 3, порядок аппроксимации по времени — 2.

**Вывод:** все функции считаются примерно за одинаковое время.

#### 4.1.2 50 на 50 на 50

Сетки аналогичны предыдущему пункту, только число элементов по всем сеткам равно 50.

space(x, y) \ time(t)	0	$t$	$t^2$	$t^3$	$t^4$	$e^t$
1	$0.16 \cdot 10^{-13}$ $0.37 \cdot 10^{-15}$ 1620	$0.47 \cdot 10^{-12}$ $0.1 \cdot 10^{-13}$ 1594	$0.34 \cdot 10^{-11}$ $0.94 \cdot 10^{-13}$ 1631	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 1662	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 1547	$0.27 \cdot 10^{-4}$ $0.6 \cdot 10^{-6}$ 2979
$x + y$	$0.22 \cdot 10^{-11}$ $0.54 \cdot 10^{-13}$ 1648	$0.43 \cdot 10^{-11}$ $0.96 \cdot 10^{-13}$ 1731	$0.19 \cdot 10^{-11}$ $0.45 \cdot 10^{-13}$ 1630	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 1654	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 1678	$0.27 \cdot 10^{-4}$ $0.6 \cdot 10^{-6}$ 3047
$x^2 + y$	$0.13 \cdot 10^{-3}$ $0.56 \cdot 10^{-14}$ 1621	$0.13 \cdot 10^{-3}$ $0.85 \cdot 10^{-14}$ 1732	$0.13 \cdot 10^{-3}$ $0.25 \cdot 10^{-13}$ 1640	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 1695	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 1621	$0.16 \cdot 10^{-3}$ $0.6 \cdot 10^{-6}$ 2974
$x^2y + y^3$	$0.13 \cdot 10^{-3}$ $0.8 \cdot 10^{-14}$ 1586	$0.13 \cdot 10^{-3}$ $0.12 \cdot 10^{-13}$ 1663	$0.13 \cdot 10^{-3}$ $0.19 \cdot 10^{-13}$ 1594	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 1568	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 1609	$0.16 \cdot 10^{-3}$ $0.6 \cdot 10^{-6}$ 2943
$xy^2$	$0.32 \cdot 10^{-4}$ $0.21 \cdot 10^{-14}$ 1634	$0.32 \cdot 10^{-4}$ $0.41 \cdot 10^{-14}$ 1617	$0.32 \cdot 10^{-4}$ $0.5 \cdot 10^{-14}$ 1697	$0.64 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 1689	$0.64 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 1624	$0.59 \cdot 10^{-4}$ $0.6 \cdot 10^{-6}$ 2984
$x^4 + y^4$	$0.2 \cdot 10^{-3}$ $0.14 \cdot 10^{-5}$ 1638	$0.2 \cdot 10^{-3}$ $0.14 \cdot 10^{-5}$ 1686	$0.2 \cdot 10^{-3}$ $0.14 \cdot 10^{-5}$ 1643	$0.23 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 1626	$0.23 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 1604	$0.22 \cdot 10^{-3}$ $0.79 \cdot 10^{-6}$ 2930
$e^{xy}$	$0.31 \cdot 10^{-4}$ $0.12 \cdot 10^{-7}$ 2960	$0.31 \cdot 10^{-4}$ $0.12 \cdot 10^{-7}$ 2958	$0.31 \cdot 10^{-4}$ $0.12 \cdot 10^{-7}$ 2936	$0.63 \cdot 10^{-4}$ $0.68 \cdot 10^{-6}$ 2980	$0.63 \cdot 10^{-4}$ $0.68 \cdot 10^{-6}$ 2904	$0.59 \cdot 10^{-4}$ $0.59 \cdot 10^{-6}$ 4230

**Вывод:** предыдущие выводы не опроверглись.

**Вывод:** время вычислений выросло примерно в 110 раз.

## 4.2 Неравномерные сетки

### 4.2.1 Функции нелинейной сетки

В ходе выполнения лабораторной работы была обнаружена функция, позволяющая легко задавать неравномерную сетку, сгущающуюся к одному из концов.

Если у нас задано начало —  $a$  и конец сетки —  $b$ , а количество элементов  $n$ , тогда сетку можно задать следующим образом:

$$x_i = a + m\left(\frac{i}{n}\right) \cdot (b - a), i = \overline{0, n}$$

где  $m(x)$  — некоторая функция, задающая неравномерную сетку. При этом  $x$  обязан принадлежать области  $[0, 1]$ , а функция  $m$  возвращать значения из той же области, и при этом быть монотонной на этом участке. Тогда гарантируется условие монотонности сетки, то есть что при  $j \leq i \Rightarrow x_j \leq x_i$ .

*Пример:* при  $m(x) = x$ , сетка становится равномерной.

Найденная функция зависят от параметра неравномерности  $t$ :

$$m_t(x) = \frac{1 - (1 - |t|)^{x \operatorname{sign} t}}{1 - (1 - |t|)^{\operatorname{sign} t}}$$

Эта функции вырождается в  $x$  при  $t = 0$ ; при  $t = -1$ , она вырождается в сетку, полностью находящуюся в 0; а при  $t = 1$  она полностью сгущается к 1.

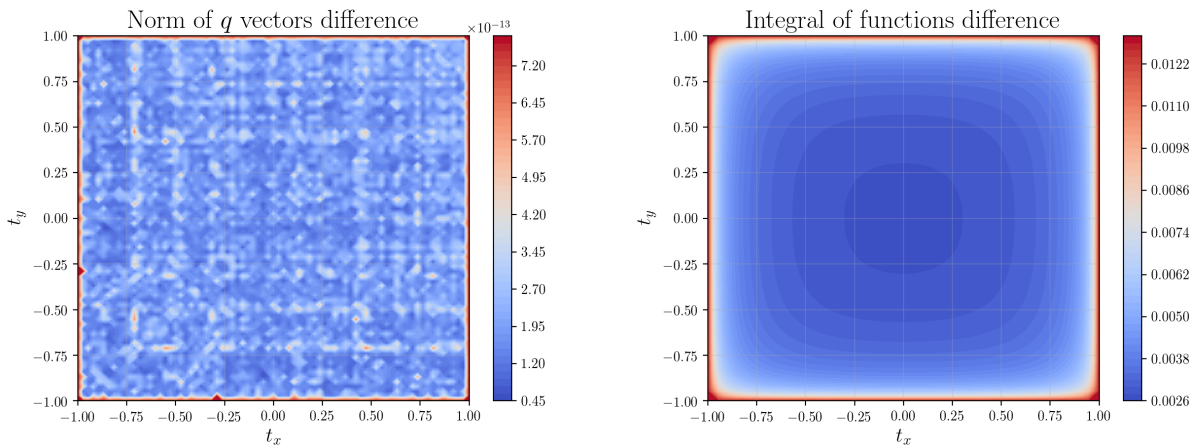
Таким образом, можно исследовать различные неравномерные сетки, изменяя параметр от  $-1$  до  $1$ , где точка  $t = 0$  будет являться результатом на равномерной сетке.

### 4.2.2 По пространству

Сетка по пространству:  $(x, y) \in [0, 1] \times [0, 1]$ , строк и столбцов 10. Сетка по времени:  $t \in [0, 1]$ , количество элементов сетки 10.

#### 4.2.2.1 Функция 1

$$u = x^2 + y^2 + t^2$$



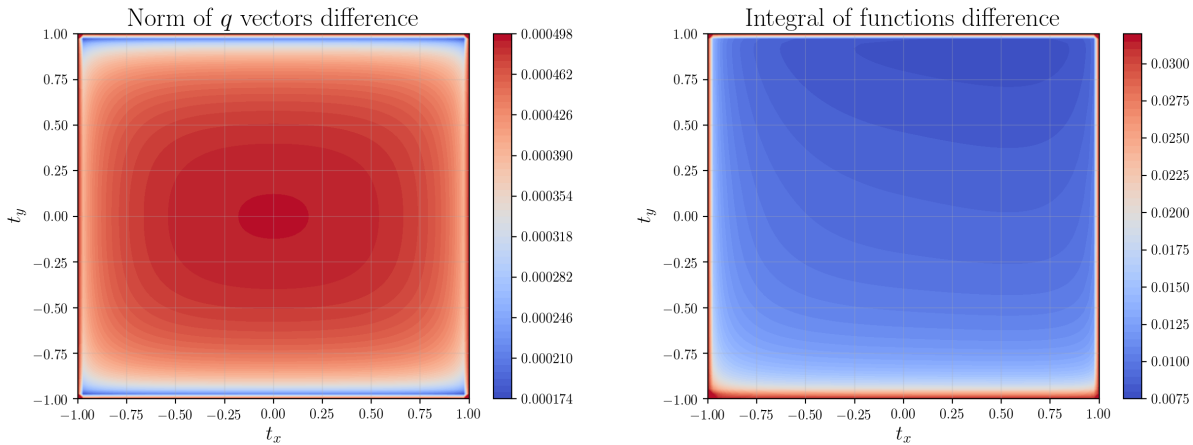
**Вывод:** так как эта функция полностью аппроксимируется данным методом в узлах, то не имеет значения насколько сетка неравномерна, примерно во всех элементах она

имеет одинаковую невязку, согласно левому графику. Разве что в сильно неравномерных сетках, где элементы сильно сгущены к одному из концов, точностью страдает на несколько порядков.

**Вывод:** а по интегральной норме лучшей сеткой является равномерная сетка согласно правому графику.

#### 4.2.2.2 Функция 2

$$u = x^4 + y^3x + t^4$$



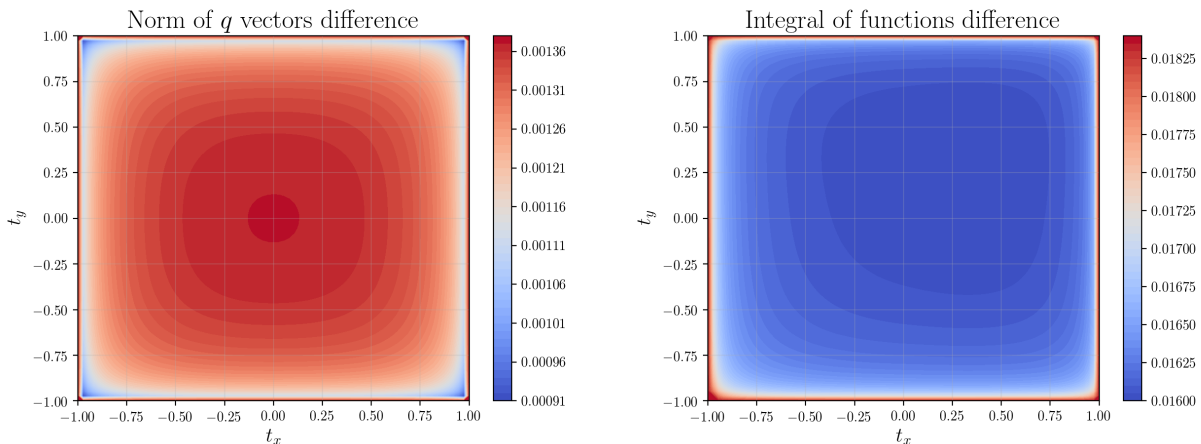
**Вывод:** согласно левому графику норма в узлах лучше всего аппроксимируется при сгущении по  $y$  в одну или другую сторону. По  $x$  же неравномерность сетки практически ни на что не влияет.

**Вывод:** лучшая точность, даваемая неравномерной сетки примерно на полпорядка лучше, чем при равномерной.

**Вывод:** по интегральной же норме существует некоторая комбинация параметров, при которых сетка получается оптимальной. Но различия от неравномерной сетки ничтожны.

#### 4.2.2.3 Функция 3

$$u = e^{xy} + e^{t^2}$$



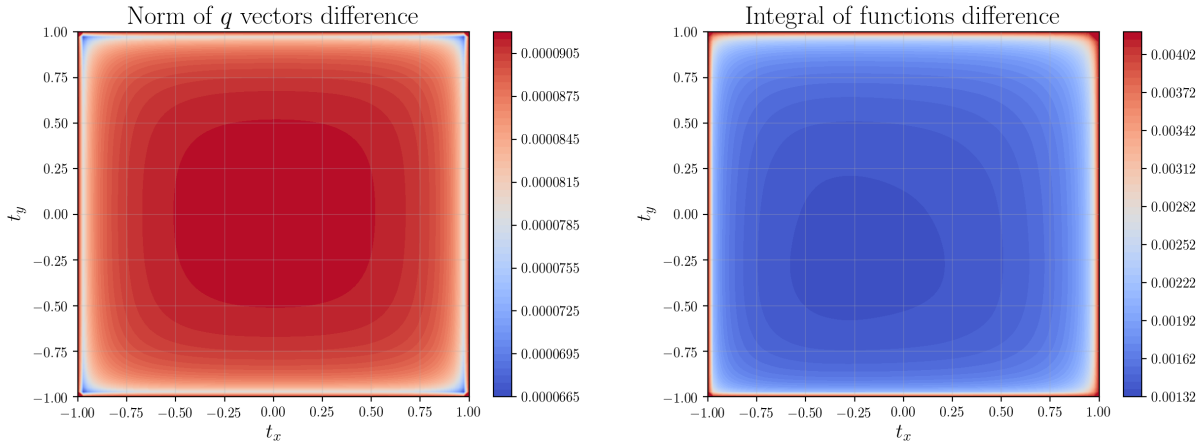
**Вывод:** согласно левому графику аппроксимация в узлах тоже имеет некоторые оптимальные значения, причем точность увеличивается на порядок.

**Вывод:** для интегральной же нормы различия же от равномерной сетки ничтожны при любых параметрах сетки.



#### 4.2.2.4 Функция 4

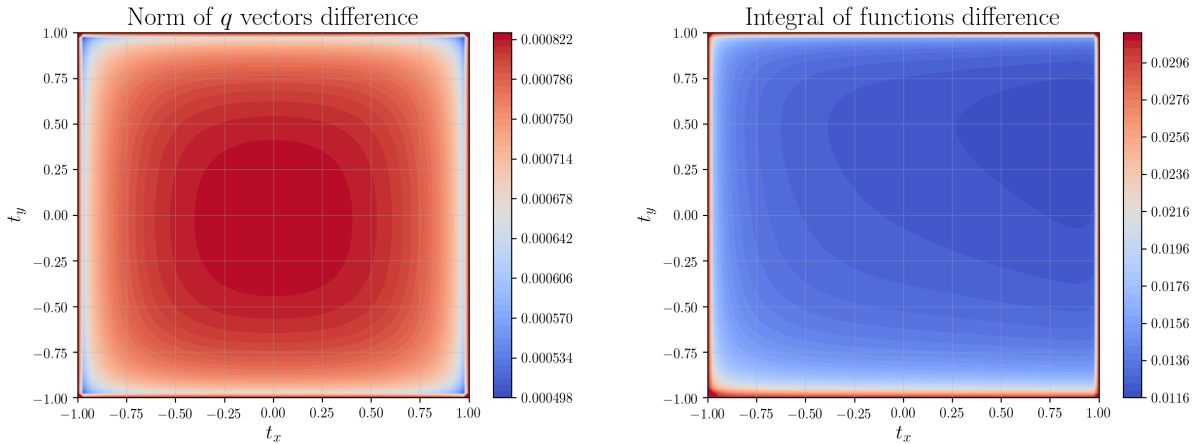
$$u = e^{(1-x)(1-y)} + e^{(1-t)^2}$$



**Вывод:** эта функция отличается от предыдущей, что для неё инвертировано положение  $x$  и  $y$ , график по интегральной норме соответственно изменился.

#### 4.2.2.5 Функция 5

$$u = x^3 + y^4 x^2 t + t^2 e^t$$



**Вывод:** всё аналогично предыдущим выводам и функциям.

#### 4.2.2.6 Общие выводы

**Вывод:** хорошая аппроксимация в узлах  $\neq$  хорошая аппроксимация по интегральной норме.

**Вывод:** согласно интегральной норме для неполиномиальных функций существует некоторый набор параметров  $t_x$  и  $t_y$ , при которых нелинейная сетка оптимальным образом аппроксимирует функцию.

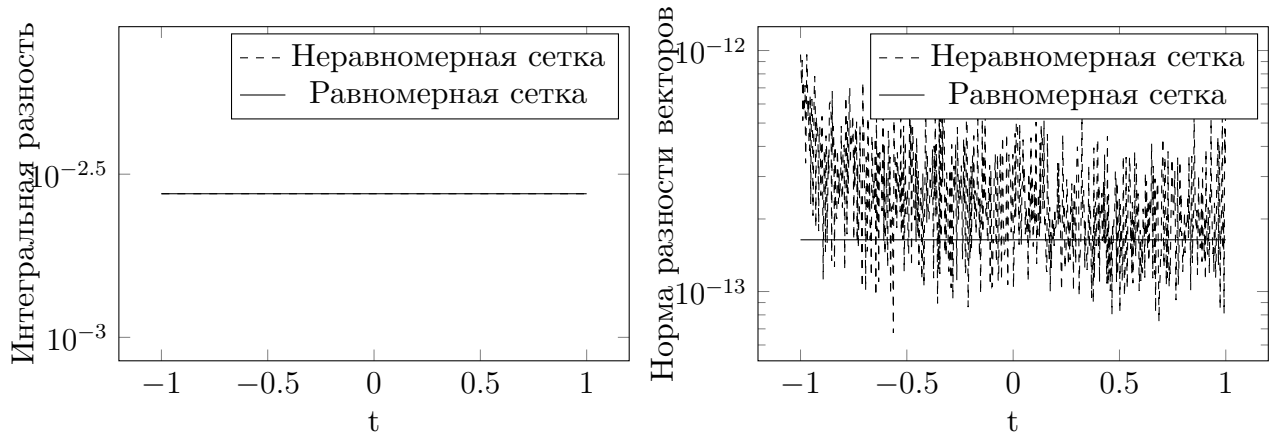
**Вывод:** согласно норме в узлах для неполиномиальных функций оптимальными являются параметры в окрестности  $\pm 1$ .

#### 4.2.3 По времени

Сетка по пространству:  $(x, y) \in [0, 1] \times [0, 1]$ , строк и столбцов 10. Сетка по времени:  $t \in [0, 1]$ , количество элементов сетки 10.

#### 4.2.3.1 Функция 1

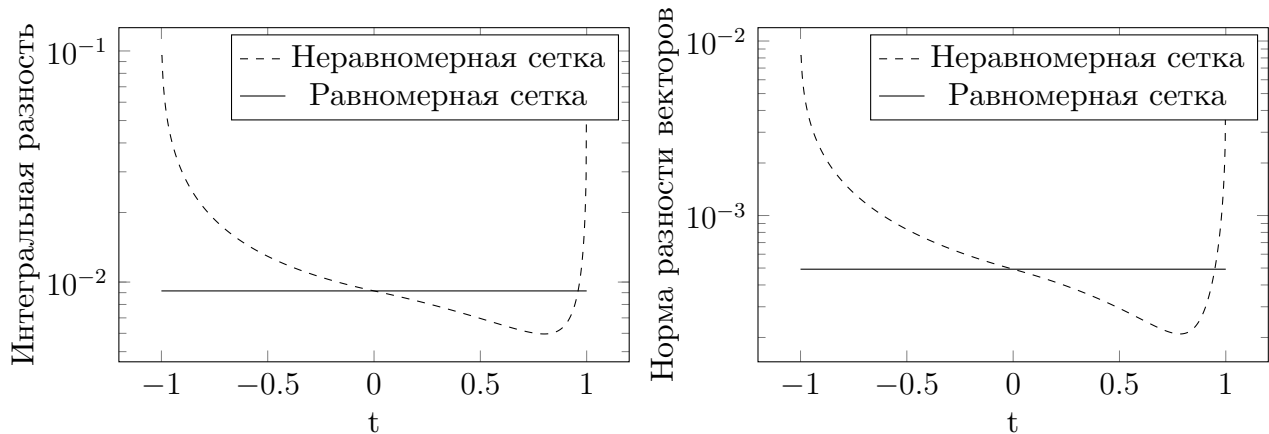
$$u = x^2 + y^2 + t^2$$



**Вывод:** так как по времени эта функция аппроксимируется точно, то неравномерность сетки никак не влияет на точность. Правый график колеблется в пределах максимальной точности, левый же абсолютно не меняется.

#### 4.2.3.2 Функция 2

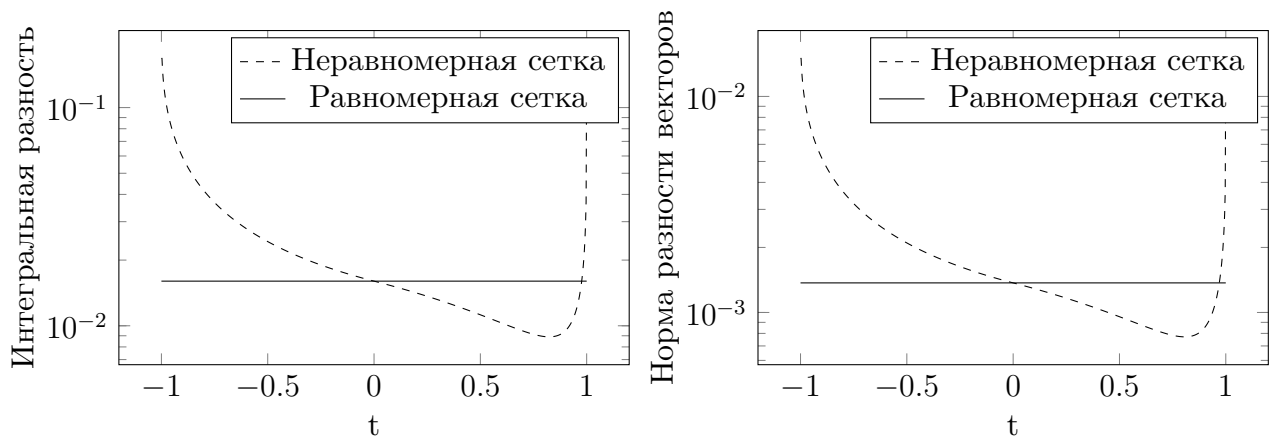
$$u = x^4 + y^3x + t^4$$



**Вывод:** для данной функции в сетки есть выраженный минимум в окрестности  $t = 0.7$ , но улучшение точности на нем примерно полпорядка.

#### 4.2.3.3 Функция 3

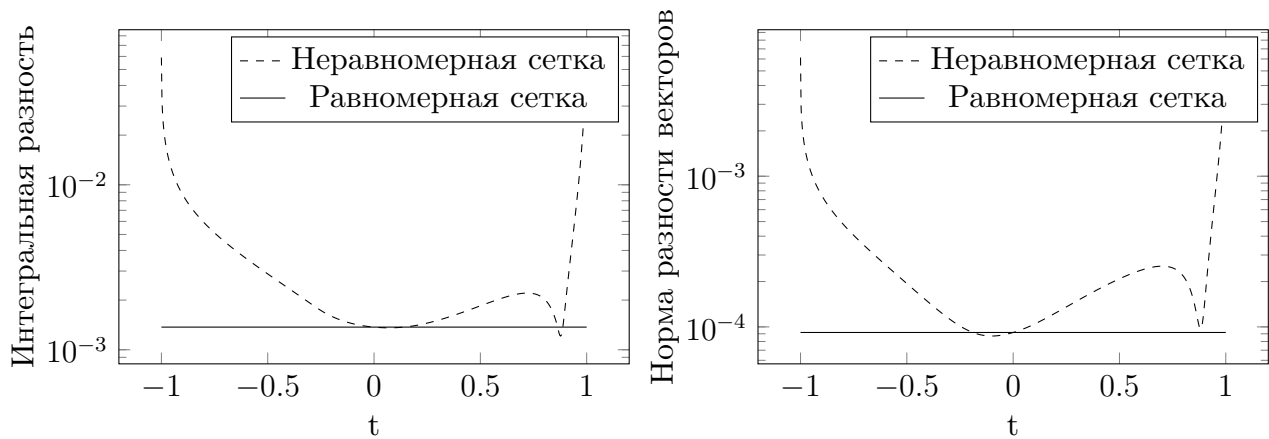
$$u = e^{xy} + e^{t^2}$$



**Вывод:** всё аналогично предыдущему.

#### 4.2.3.4 Функция 4

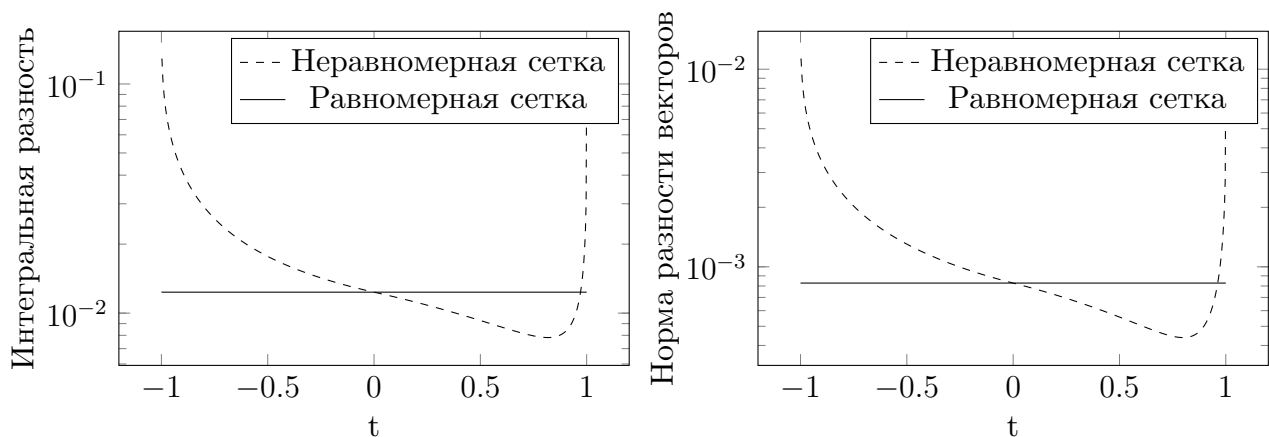
$$u = e^{(1-x)(1-y)} + e^{(1-t)^2}$$



**Вывод:** эта функция является перевернутой версией предыдущей, но график аналогично не переверлся, а наблюдается более сложная зависимость. Для данной функции неравномерная сетка по времени практически везде дает отрицательный эффект по сравнению с равномерной сеткой.

#### 4.2.3.5 Функция 5

$$u = x^3 + y^4 x^2 t + t^2 e^t$$



#### 4.2.3.6 Общие выводы

**Вывод:** у множества функций наблюдалось схожее поведение на неравномерной сетке по времени, с наличием ярко выраженного минимума, и использование сетки с данным оптимальным параметром может улучшить точность решения на полпорядка.

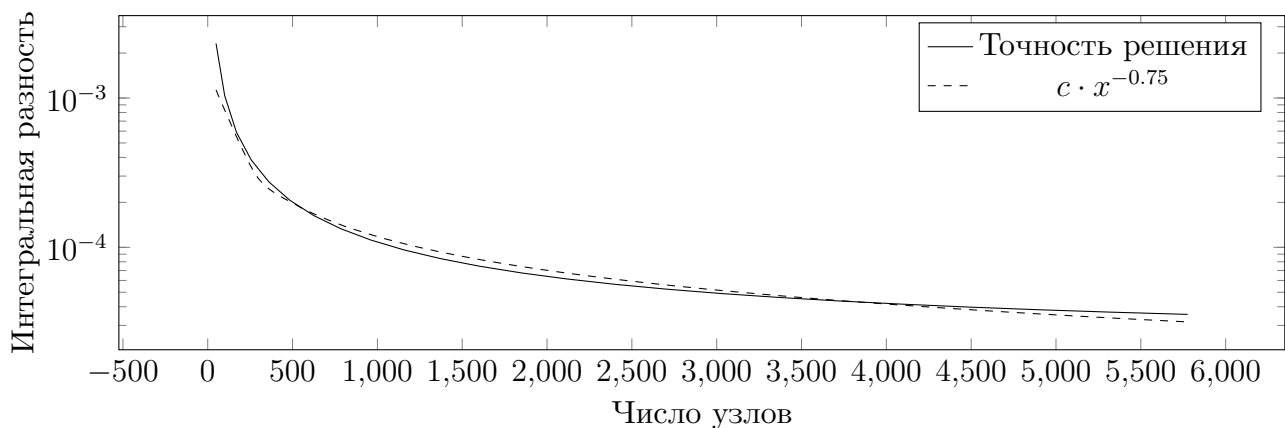
### 4.3 Порядок сходимости

Исследуется на функции:

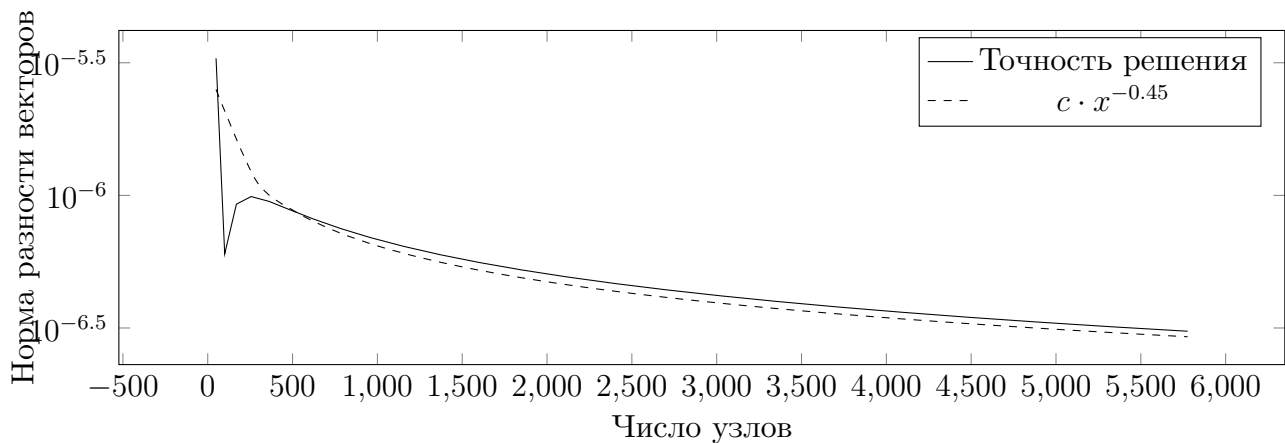
$$u(x, y, t) = e^{xy} + e^{t^2}$$

#### 4.3.1 Увеличение размерности по пространству

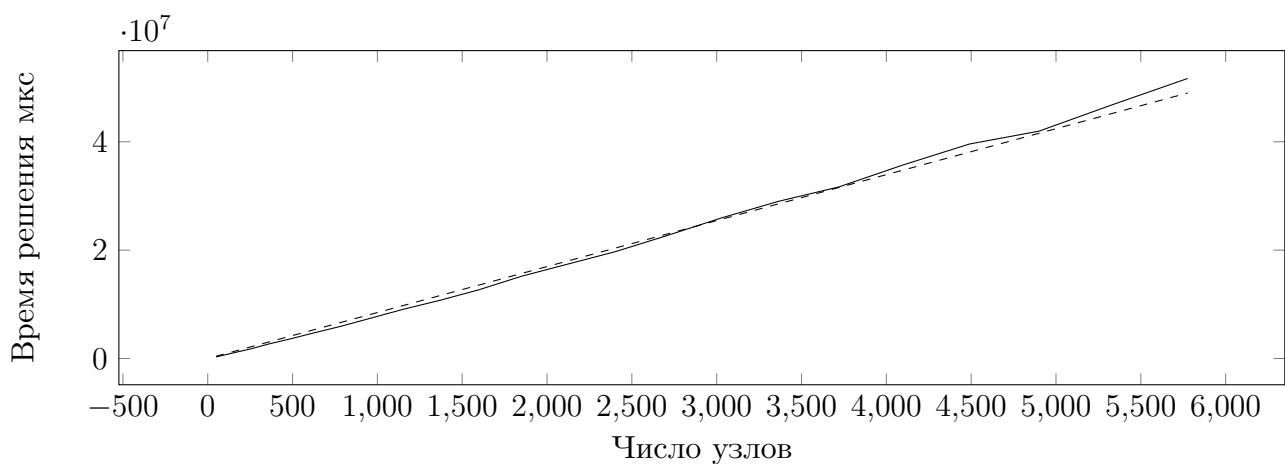
Сетка по пространству:  $(x, y) \in [0, 1] \times [0, 1]$ . Сетка по времени:  $t \in [0, 1]$ , количество элементов сетки 200.



**Вывод:** согласно интегральной норме, порядок сходимости  $\approx 0.75$ .



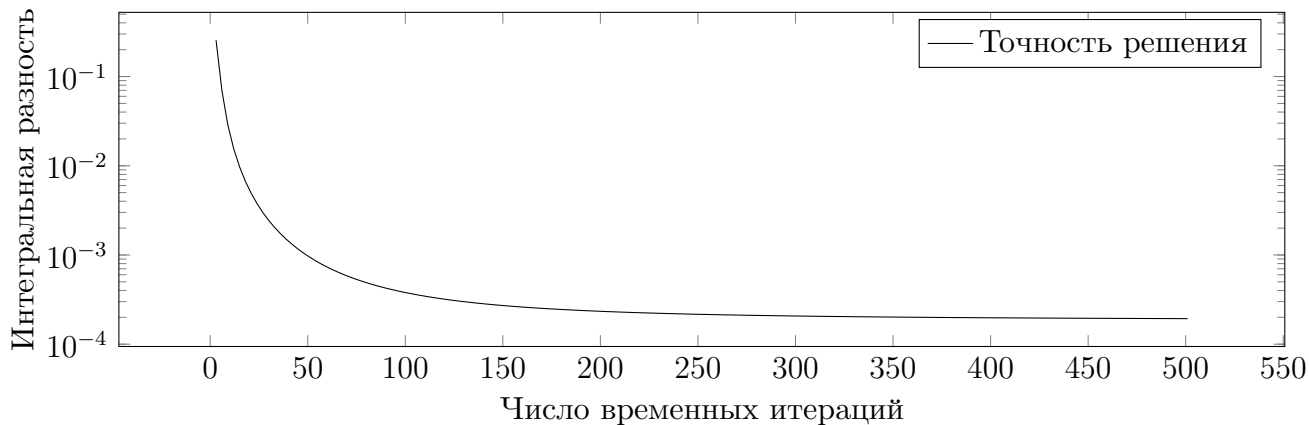
**Вывод:** согласно норме разности векторов, порядок сходимости  $\approx 0.45$ .



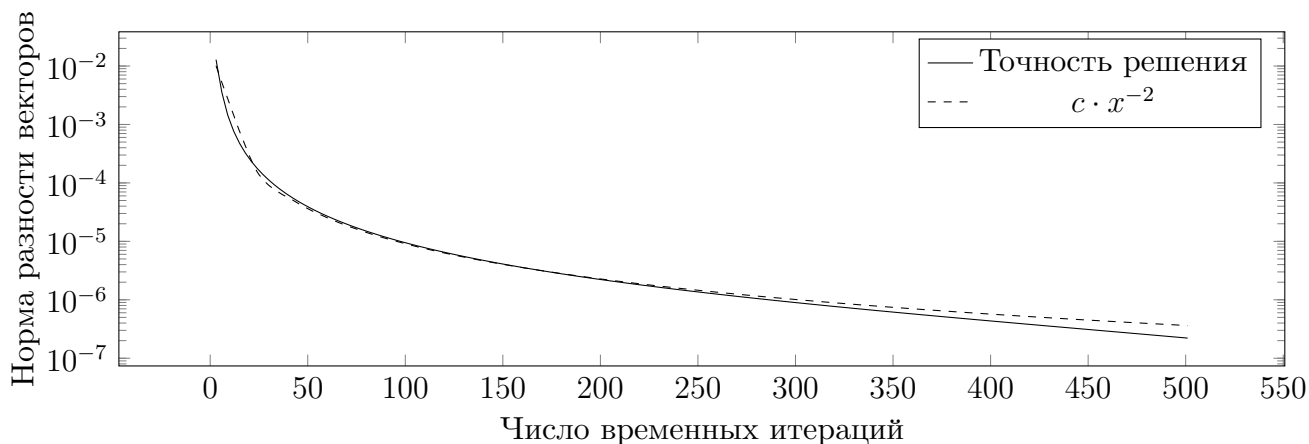
**Вывод:** время решения почти линейно зависит от числа узлов (с учетом погрешности, вносимой многопоточностью).

#### 4.3.2 Увеличение размерности по времени

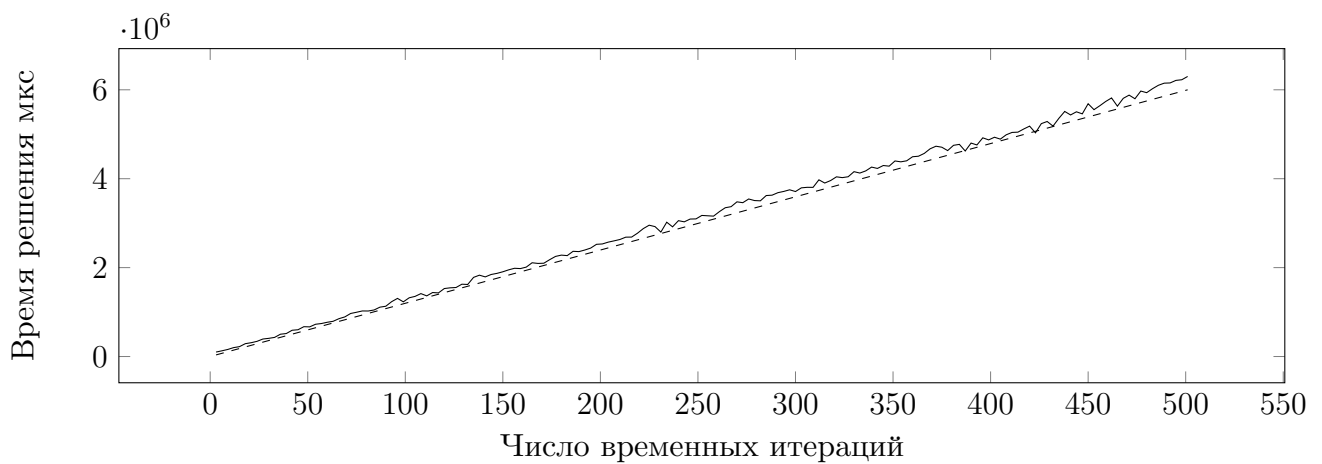
Сетка по пространству:  $(x, y) \in [0, 1] \times [0, 1]$ , строк и столбцов 20. Сетка по времени:  $t \in [0, 1]$ .



**Вывод:** при увеличении числа итераций по времени, увеличивается интегральная норма, но потом она достигает предельного значения и далее не увеличивается, потому что количество элементов сетки неизменно, и они не способны точно аппроксимировать данную функцию, поскольку она непредставима в линейных элементах.



**Вывод:** при увеличении числа итераций по времени, увеличивается точность, и порядок аппроксимации по времени равен 2.



**Вывод:** время решения почти линейно зависит от числа итераций по времени (с учетом погрешности, вносимой многопоточностью).

## 5 Код

### 5.1 Файлы заголовков

FILE **lib.h**

```
1 #pragma once
2
3 /** Определения типов. */
4 #include <functional>
5 #include <chrono>
6 #include <mutex>
7 #include <iomanip>
8 #include <iostream>
9 #include <string>
10 #include <Eigen/Dense>
11
12 using namespace std;
13 using namespace placeholders;
14
15 /* Для плотных матриц и векторов используется библиотека Eigen. */
16 typedef Eigen::MatrixXd matrix_t; /// Плотная матрица
17 typedef Eigen::VectorXd vector_t; /// Вектор
18
19 /* Тип 1D, 2D, 3D функций. */
20 typedef function<double(double)> function_1d_t;
21 typedef function<double(double, double)> function_2d_t;
22 typedef function<double(double, double, double)> function_3d_t;
23
24 /** Считает время выполнения функции f в микросекундах. */
25 inline double calc_time_microseconds(const function<void(void)>& f) {
26     using namespace chrono;
27     auto start = high_resolution_clock::now();
28     f();
29     auto end = high_resolution_clock::now();
30     return duration_cast<microseconds>(end - start).count();
31 }
32
33 /** Выводит на экран процент завершенной работы. Использует мьютексы для защиты cout при использовании
34     ↪ несколькими потоками */
35 inline void write_percent(double percent) {
36     static mutex m;
37     lock_guard<mutex> g(m);
38     cout << "\r" << setprecision(2) << fixed << setw(5) << percent * 100 << "%";
39 }
40
41 //-----
42 inline string write_for_latex_double(double v, int precision) {
43     int power = log(std::fabs(v)) / log(10.0);
44     double value = v / pow(10.0, power);
45
46     if (v != v) return "nan";
47
48     if (v == 0) {
49         power = 0;
50         value = 0;
51     }
52 }
```

```

51     stringstream sout;
52     sout.precision(precision);
53     if (power == -1 || power == 0 || power == 1) {
54         sout << v;
55     } else {
56         sout << value << "\\cdot 10^{ " << power << " }";
57     }
58 }
59
60 return sout.str();
61 }

```

## FILE sparse.h

```

1 #pragma once
2
3 /** Файл для работы с матрицей в разреженном формате и решении */
4
5 #include <vector>
6 #include <map>
7 #include <iostream>
8 #include "lib.h"
9
10 //-----
11 /** Класс квадратной разреженной матрицы в строчно-столбцовом формате с симметричным профилем.
12     ↳ Примечание: ненулевым считается элемент, который имеется в профиле, неважно что в массивах l, u он
13     ↳ может иметь значение 0. */
14 class matrix_sparse_t
15 {
16 public:
17     int n; // Размерность матрицы
18     vector<double> d; // Диагональные элементы матрицы
19     vector<double> l; // Элементы матрицы из нижнего треугольника
20     vector<double> u; // Элементы матрицы из верхнего треугольника
21     vector<int> i; // Массив начала строк в формате (ia в методичке)
22     vector<int> j; // Массив столбцов каждого элемента (ja в методичке)
23
24     matrix_sparse_t(int n);
25
26     /** Преобразование разреженной матрицы к плотному формату. */
27     void to_dense(matrix_t& m) const;
28
29     void clear_line(int line);
30
31     int line_elem_start(int line) const; // Получить позицию в массивах l, u элемента, с которого
32     ↳ начинается строка line
33     int line_elem_row(int line, int elem) const; // Получить столбец ненулевого элемента в строке
34     ↳ line под номером elem
35     int line_elem_count(int line) const; // Получить количество ненулевых элементов в строке
36
37     /** Раскладывает текущую матрицу в неполное LU разложение и хранит результат в матрице lu.
38     ↳ Неполное разложение - это когда были применены формулы для получения LU матрицы, но только к
39     ↳ существующим ненулевым элементам, без перестройки формата. Иными словами называется "неполная
40     ↳ факторизация". */
41     void decompose_lu_partial(matrix_sparse_t& lu) const;
42
43     /** Методы для умножения разреженной матрицы на вектор. */
44     void mul(vector_t& x_y) const; // x_y = a * x_y
45     void mul_t(vector_t& x_y) const; // x_y = a^t * x_y
46
47     /** Представляет, что текущая матрица хранит LU разложение, и соответственно можно каждую матрицу
48     ↳ этого разложения умножить на соответствующие вектора. */
49     void mul_l_inv_t(vector_t& x_y) const; // x_y = l^-t * x_y
50     void mul_u_inv_t(vector_t& x_y) const; // x_y = u^-t * x_y
51     void mul_l_inv(vector_t& x_y) const; // x_y = l^-1 * x_y
52     void mul_u_inv(vector_t& x_y) const; // x_y = u^-1 * x_y
53     void mul_u(vector_t& x_y) const; // x_y = u * x_y
54 };
55
56 ostream& operator<<(ostream& out, const matrix_sparse_t& m);
57
58 //-----
59 /** Квадратная матрица с произвольным доступом к любому элементу. Предполагается, что матрица будет
60     ↳ разреженная. Далее можно регенерировать её в разреженную матрицу. */
61 class matrix_sparse_ra_t
62 {
63 public:
64     matrix_sparse_ra_t(int n);
65
66     /** Установить значение в позиции (i, j) */
67     double& operator()(int i, int j);
68
69     /** Получить значение в позиции (i, j). Если туда ещё не устанавливалось значение, вызывается
70     ↳ исключение. */
71     const double& operator()(int i, int j) const;
72
73 }

```

```

63  /** Преобразует текущую матрицу к разреженной матрице. */
64  matrix_sparse_t to_sparse(void) const;
65 private:
66  int n;
67  vector<double> dm;
68  vector<map<int, double>> lm, um;
69 };
70
71 //-----
72 /* Функции для умножения векторов. */
73 void mul(const vector_t& d, vector_t& x_y); //  $x_y = d * x_y$ 
74 void mul_inv(const vector_t& d, vector_t& x_y); //  $x_y = x_y / d$ 
75
76 //-----
77 /** Решает СЛАУ с матрицей в разреженном формате при помощи Локально-Оптимальной Схемы (ЛОС) с
78     ↳ предобуславливанием на основе неполной LU факторизации. */
79 vector_t solve_by_los_lu(
80     const matrix_sparse_t& a,
81     const vector_t& b,
82     int maxiter,
83     double eps,
84     bool is_log = false
85 );

```

## FILE fem.h

```

1 #pragma once
2
3 /* Заголовок функций для реализации Метода Конечных Элементов (МКЭ) в 2D пространстве. Уравнение
4     ↳ гиперболическое (с второй производной по времени). Схема для аппроксимации по времени:
5     ↳ Кранка-Николсона. Базисные элементы: билинейные. Форма сетки: прямоугольники. */
6
7 #include "lib.h"
8 #include "sparse.h"
9
10 //-----
11 /** Узел конечного элемента. Другими словами, вес, домноженный на базовую функцию. Из сумм этих
12     ↳ элементов образуется конечный элемент. */
13 struct basic_elem_t
14 {
15     int i; /// Номер узла
16     double x, y; /// Координаты узла
17     basic_elem_t *up, *down, *left, *right; /// Указатели на соседей узла
18     /** Проверяет, является ли элемент граничным. Он таким является, если у него нет хотя бы одного
19         ↳ соседа. */
20     bool is_boundary(void) const;
21 };
22
23 /** Прямоугольный конечный элемент на основе билинейных базисных функций. Образуется из четырех узлов.
24     ↳ */
25 struct elem_t
26 {
27     int i; /// Номер конечного элемента
28     basic_elem_t* e[4]; /// Указатели на все 4 элемента конечного узла, нумерация такая:
29     /**
30         Y
31         ^ 3 +-----+ 4
32         |   |       |
33         |   1 +-----+ 2
34         +-----+ X
35     */
36     double get_hx(void) const; /// Ширина конечного элемента
37     double get_hy(void) const; /// Высота конечного элемента
38     /** Рассчитать значение внутри конечного элемента. q - вектор рассчитываемых весов. */
39     double value(double x, double y, const vector_t& q) const;
40 };
41
42 /** Все константы решаемого уравнения. */
43 struct constants_t
44 {
45     double lambda; /// Коэффициент внутри div
46     double gamma; /// Коэффициент при u
47     double sigma; /// Коэффициент при du/dt
48     double chi; /// Коэффициент при  $d^2u/dt^2$ 
49 };
50
51 //-----
52

```



```

53 /** t in [-1, 1]. x in [0, 1] При t=-1 возвращаемое значение полностью смещается к 0, при t=1
    ↪ возвращаемое значение полностью смещается к 1, при t=0 возвращаемое значение равно x. Между этими
    ↪ значениями используются формулы, чтобы решение сгущалось постепенно к одному из концов. Функция
    ↪ используется для генерации неравномерной сетки. */
54 double non_linear_offset(double x, double t);
55
56 /** Генерирует неравномерную сетку по заданным параметрам. n - число внутренних узлов. То есть если n
    ↪ будет равно 0, то узел под номером 0 будет a, а под номером 1 будет b. */
57 class grid_generator_t
58 {
59 public:
60     grid_generator_t(double a, double b, int n, double t = 0);
61     double operator()(int i) const;
62     int size(void) const;
63     double back(void) const;
64 private:
65     double a, len, t, n1;
66 };
67
68 /** Класс двумерной неравномерной сетки по пространству в виде прямоугольника. */
69 class grid_t
70 {
71 public:
72     vector<elem_t> es; /// Массив конечных элементов сетки
73     vector<basic_elem_t> bes; /// Массив узлов сетки
74     int n; /// Число узлов
75
76     /** Рассчитать неравномерную сетку. */
77     void calc(const grid_generator_t& gx, const grid_generator_t& gy);
78 };
79
80 /** Рассчитать веса идеальной аппроксимации функции u при помощи узлов bes. */
81 vector_t calc_true_approx(const function_2d_t& u, const vector<basic_elem_t>& bes);
82
83 /** Рассчитать интегральную норму между конечно-элементной аппроксимацией и истинной функцией. */
84 double calc_integral_norm(const function_2d_t& u, const vector<elem_t>& es, const vector_t& q);
85
86 //-----
87 /** Расчет локальных матриц для конечного элемента. */
88 matrix_t calc_local_matrix_g(const elem_t& e, const constants_t& cs);
89 matrix_t calc_local_matrix_c(const elem_t& e);
90 vector_t calc_local_vector_b(const elem_t& e, const function_2d_t& f);
91
92 //-----
93 /** Рассчитать глобальный вектор из локальных векторов для всех конечных элементов. */
94 vector_t calc_global_vector(
95     const vector<elem_t>& es,
96     const function<vector_t(const elem_t)> calc_local_vector,
97     int n
98 );
99
100 /** Рассчитать глобальную матрицу из функции построения локальных матриц. */
101 matrix_sparse_t calc_global_matrix(
102     const vector<elem_t>& es,
103     const function<matrix_t(const elem_t)> calc_local_matrix,
104     int n
105 );
106
107 //-----
108 /** Численный расчет определенных интегралов. */
109 double calc_integral_gauss3(
110     double a, double b, int n, // n - количество внутренних узлов
111     const function_1d_t& f
112 );
113 double calc_integral_gauss3(
114     double ax, double bx, int nx,
115     double ay, double by, int ny,
116     const function_2d_t& f
117 );
118
119 //-----
120 /** Численный расчет производной. */
121 function_1d_t calc_first_derivative(const function_1d_t& f);
122 function_1d_t calc_second_derivative(const function_1d_t& f);
123
124 //-----
125 /** Для гиперболического дифференциального уравнения и функции u считает каким должно быть f, чтобы
    ↪ решением этого диф. уравнения была функции u. Делает это численно. */
126 function_3d_t calc_right_part(const function_3d_t& u, const constants_t& cs);
127
128 //-----
129 /** Использует схему Кранка-Николсона для получения СЛАУ. Предполагается, что разреженные матрицы
    ↪ имеют одинаковый формат. */
130 void calc_crank_nicolson_method(
131     const matrix_sparse_t& c,
132     const matrix_sparse_t& g,
133     const vector_t& b0, // b current (b_j)
134     const vector_t& b1, // b last (b_{j-1})
135     const vector_t& b11, // b last last (b_{j-2})

```

```

136     const vector_t& q1, // q last (q_{j-1})
137     const vector_t& q11, // q last last (q_{j-2})
138     const constants_t& cs,
139     const grid_generator_t& time_grid,
140     int time_i,
141     matrix_sparse_t& a,
142     vector_t& b
143 );
144
145 //-----
146 /** Функция, которая устанавливает краевые условия для задачи в СЛАУ. Сделана для того, чтобы не
    ↪ посылать в функцию решения МКЭ истинную функцию, а чтобы посылать красивую оболочку, которую
    ↪ потенциально можно использовать в реальных задачах. */
147 typedef function<void(matrix_sparse_t&, vector_t&, const vector<basic_elem_t>&, double)>
    ↪ boundary_setter_t;
148
149 /** Записывает первые краевые условия в матрицу a и вектор b. Для этой записи ему необходимо получить
    ↪ истинную функцию. */
150 void write_first_boundary_conditions(
151     matrix_sparse_t& a,
152     vector_t& b,
153     const vector<basic_elem_t>& bes,
154     double t,
155     const function_3d_t& u
156 );
157
158 //-----
159 /** Решает при помощи МКЭ дифференциальное уравнение с функцией правой части f, заданными константами,
    ↪ прямоугольной сеткой grid и функцией выставления краевых условий. Использует схему
    ↪ Кранка-Николсона для аппроксимации по времени, и ЛОС в разреженной строчно-столбцовой матрице для
    ↪ решения СЛАУ. */
160 vector<vector_t> solve_differential_equation(
161     const function_3d_t& f,
162     const boundary_setter_t& set_boundary_conditions,
163     const vector_t& q0,
164     const vector_t& q1,
165     const constants_t& cs,
166     const grid_t& grid,
167     const grid_generator_t& time_grid
168 );

```

## 5.2 Исходные файлы

### FILE sparse.cpp

```

1 #include "sparse.h"
2
3 //-----
4 matrix_sparse_t::matrix_sparse_t(int n) : n(n) {
5     d.resize(n);
6     i.resize(n+1, 0);
7 }
8
9 //-----
10 void matrix_sparse_t::to_dense(matrix_t& m) const {
11     m = matrix_t(n, n);
12     m.fill(0);
13     for (int _i = 0; _i < n; ++_i) {
14         m(_i, _i) = d[_i];
15         for (int _j = 0; _j < line_elem_count(_i); ++_j) {
16             m(_i, line_elem_row(_i, _j)) = l[line_elem_start(_i) + _j];
17             m(line_elem_row(_i, _j), _i) = u[line_elem_start(_i) + _j];
18         }
19     }
20 }
21
22 //-----
23 void matrix_sparse_t::clear_line(int line) {
24     d[line] = 0;
25     for (int _i = 0; _i < i.size()-1; _i++) {
26         for (int pj = i[_i]; pj < i[_i+1]; pj++) {
27             int _j = j[pj];
28             if (_j == line) u[pj] = 0;
29             if (_i == line) l[pj] = 0;
30         }
31     }
32 }
33
34 //-----
35 int matrix_sparse_t::line_elem_start(int line) const {
36     return i[line];
37 }
38 //-----

```

```

39 int matrix_sparse_t::line_elem_row(int line, int elem) const {
40     return j[line_elem_start(line) + elem];
41 }
42
43 //-----
44 int matrix_sparse_t::line_elem_count(int line) const {
45     return i[line+1]-i[line];
46 }
47
48 //-----
49 void matrix_sparse_t::decompose_lu_partial(matrix_sparse_t& lu) const {
50     const matrix_sparse_t& a = *this;
51
52     lu = a;
53     for (int i = 0; i < lu.n; ++i) {
54         // Заполняем нижний треугольник
55         int line_start = lu.line_elem_start(i);
56         int line_end = lu.line_elem_start(i+1);
57         for (int j = line_start; j < line_end; ++j) {
58             double sum = 0;
59
60             int row = lu.j[j];
61             int row_start = lu.line_elem_start(row);
62             int row_end = lu.line_elem_start(row+1);
63
64             int kl = line_start;
65             int ku = row_start;
66
67             while (kl < j && ku < row_end) {
68                 if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
69                     sum += lu.l[kl] * lu.u[ku];
70                     ku++;
71                     kl++;
72                 } else if (lu.j[kl] < lu.j[ku]) {
73                     kl++;
74                 } else {
75                     ku++;
76                 }
77             }
78
79             lu.l[j] = (a.l[j] - sum) / lu.d[row];
80         }
81
82         // Заполняем верхний треугольник
83         int row_start = lu.line_elem_start(i);
84         int row_end = lu.line_elem_start(i+1);
85         for (int j = line_start; j < line_end; ++j) {
86             double sum = 0;
87
88             int line = lu.j[j];
89             int line_start = lu.line_elem_start(line);
90             int line_end = lu.line_elem_start(line+1);
91
92             int kl = line_start;
93             int ku = row_start;
94
95             while (kl < line_end && ku < j) {
96                 if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
97                     sum += lu.l[kl] * lu.u[ku];
98                     ku++;
99                     kl++;
100                 } else if (lu.j[kl] < lu.j[ku]) {
101                     kl++;
102                 } else {
103                     ku++;
104                 }
105             }
106
107             lu.u[j] = (a.u[j] - sum) / lu.d[line];
108         }
109
110         // Расчитываем диагональный элемент
111         double sum = 0;
112         int line_row_start = lu.line_elem_start(i);
113         int line_row_end = lu.line_elem_start(i+1);
114         for (int j = line_row_start; j < line_row_end; ++j)
115             sum += lu.l[j] * lu.u[j];
116
117         lu.d[i] = sqrt(a.d[i] - sum);
118     }
119 }
120
121 //-----
122 void matrix_sparse_t::mul(vector_t& x_y) const {
123     const matrix_sparse_t& a = *this;
124
125     vector_t result(a.n); result.fill(0);
126
127     for (int i = 0; i < a.n; ++i) {

```

```

128     int start = a.line_elem_start(i);
129     int size = a.line_elem_count(i);
130     for (int j = 0; j < size; j++) {
131         result[i] += a.l[start + j] * x_y[a.line_elem_row(i, j)];
132         result[a.line_elem_row(i, j)] += a.u[start + j] * x_y[i];
133     }
134 }
135
136 // Умножение диагональных элементов на вектор
137 for (int i = 0; i < a.n; ++i)
138     result[i] += a.d[i] * x_y[i];
139
140 x_y = result;
141 }
142
143 //-----
144 void matrix_sparse_t::mul_t(vector_t& x_y) const {
145     const matrix_sparse_t& a = *this;
146
147     vector_t result(a.n); result.fill(0);
148
149     for (int i = 0; i < a.n; ++i) {
150         int start = a.line_elem_start(i);
151         int size = a.line_elem_count(i);
152         for (int j = 0; j < size; j++) {
153             result(i) += a.u[start + j] * x_y[a.line_elem_row(i, j)];
154             result(a.line_elem_row(i, j)) += a.l[start + j] * x_y[i];
155         }
156     }
157
158     // Умножение диагональных элементов на вектор
159     for (int i = 0; i < a.n; ++i)
160         result(i) += a.d[i] * x_y[i];
161
162     x_y = result;
163 }
164
165 //-----
166 void matrix_sparse_t::mul_l_inv_t(vector_t& x_y) const {
167     const matrix_sparse_t& l = *this;
168
169     for (int i = l.n - 1; i >= 0; i--) {
170         int start = l.line_elem_start(i);
171         int size = l.line_elem_count(i);
172
173         x_y[i] /= l.d[i];
174         for (int j = 0; j < size; ++j)
175             x_y[l.line_elem_row(i, j)] -= x_y[i] * l.l[start + j];
176     }
177 }
178
179 //-----
180 void matrix_sparse_t::mul_u_inv_t(vector_t& x_y) const {
181     const matrix_sparse_t& u = *this;
182
183     for (int i = 0; i < u.n; ++i) {
184         int start = u.line_elem_start(i);
185         int size = u.line_elem_count(i);
186
187         double sum = 0;
188         for (int j = 0; j < size; ++j)
189             sum += u.u[start + j] * x_y[u.line_elem_row(i, j)];
190         x_y[i] = (x_y[i] - sum) / u.d[i];
191     }
192 }
193
194 //-----
195 void matrix_sparse_t::mul_l_inv(vector_t& x_y) const {
196     const matrix_sparse_t& l = *this;
197
198     for (int i = 0; i < l.n; ++i) {
199         int start = l.line_elem_start(i);
200         int size = l.line_elem_count(i);
201
202         double sum = 0;
203         for (int j = 0; j < size; ++j)
204             sum += l.l[start + j] * x_y[l.line_elem_row(i, j)];
205         x_y[i] = (x_y[i] - sum) / l.d[i];
206     }
207 }
208
209 //-----
210 void matrix_sparse_t::mul_u_inv(vector_t& x_y) const {
211     const matrix_sparse_t& u = *this;
212
213     for (int i = u.n-1; i >= 0; i--) {
214         int start = u.line_elem_start(i);
215         int size = u.line_elem_count(i);
216

```

```

217     x_y[i] /= u.d[i];
218     for (int j = 0; j < size; ++j)
219         x_y[u.line_elem_row(i, j)] -= x_y[i] * u.u[start + j];
220 }
221 }
222
223 //-----
224 void matrix_sparse_t::mul_u(vector_t& x_y) const {
225     const matrix_sparse_t& u = *this;
226
227     vector_t result(u.n); result.fill(0);
228
229     for (int i = 0; i < u.n; ++i) {
230         int start = u.line_elem_start(i);
231         int size = u.line_elem_count(i);
232         for (int j = 0; j < size; ++j) {
233             result[u.line_elem_row(i, j)] += u.u[start + j] * x_y[i];
234         }
235     }
236
237     // Умножение диагональных элементов на вектор
238     for (int i = 0; i < u.n; ++i)
239         result[i] += u.d[i] * x_y[i];
240
241     x_y = result;
242 }
243
244 //-----
245 ostream& operator<<(ostream& out, const matrix_sparse_t& m) {
246     matrix_t dense;
247     m.to_dense(dense);
248     out << dense;
249     return out;
250 }
251
252 //-----
253 //-----
254 //-----
255
256 //-----
257 matrix_sparse_ra_t::matrix_sparse_ra_t(int n) : n(n), dm(n, 0), lm(n), um(n) {}
258
259 //-----
260 double& matrix_sparse_ra_t::operator()(int i, int j) {
261     if (i == j) {
262         return dm[i];
263     } else if (i > j) {
264         um[i][j] += 0;
265         return lm[i][j];
266     } else {
267         lm[j][i] += 0;
268         return um[j][i];
269     }
270 }
271
272 //-----
273 const double& matrix_sparse_ra_t::operator()(int i, int j) const {
274     if (i == j) {
275         return dm[i];
276     } else if (i > j) {
277         if (lm[i].find(j) != lm[i].end())
278             return lm[i].at(j);
279     } else {
280         if (um[j].find(i) != um[j].end())
281             return um[j].at(j);
282     }
283
284     throw exception();
285 }
286
287 //-----
288 matrix_sparse_t matrix_sparse_ra_t::to_sparse(void) const {
289     matrix_sparse_t result(n);
290     result.n = dm.size();
291     result.d = dm;
292     for (int i = 0; i < lm.size(); ++i) {
293         result.i[i+1] = result.i[i] + lm[i].size();
294         for (auto& j : lm[i]) {
295             result.j.push_back(j.first);
296             result.l.push_back(j.second);
297             result.u.push_back(um[i].at(j.first));
298         }
299     }
300     return result;
301 }
302
303 //-----
304 //-----
305 //-----

```

```

306 //-----
307 void mul(const vector_t& d, vector_t& x_y) {
308     for (int i = 0; i < d.size(); i++)
309         x_y[i] *= d[i];
310 }
311 //-----
312 void mul_inv(const vector_t& d, vector_t& x_y) {
313     for (int i = 0; i < d.size(); i++)
314         x_y[i] /= d[i];
315 }
316 //-----
317 //-----
318 //-----
319 //-----
320 //-----
321 //-----
322 //-----
323 vector_t solve_by_los_lu(
324     const matrix_sparse_t& a,
325     const vector_t& b,
326     int maxiter,
327     double eps,
328     bool is_log
329 ) {
330     matrix_sparse_t lu(a.n);
331     vector_t r, z, p;
332     vector_t x, t1, t2;
333
334     int n = a.n;
335
336     a.decompose_lu_partial(lu);
337     x = vector_t(n); x.fill(0);
338
339     r = x;
340     a.mul(r);
341     for (int i = 0; i < n; i++)
342         r[i] = b[i] - r[i];
343     lu.mul_l_inv(r);
344
345     z = r;
346     lu.mul_u_inv(z);
347
348     p = z;
349     a.mul(p);
350     lu.mul_l_inv(p);
351
352     double flen = sqrt(b.dot(b));
353     double residual;
354
355     int i = 0;
356     while (true) {
357         double pp = p.dot(p);
358         double alpha = (p.dot(r)) / pp;
359         for (int i = 0; i < n; ++i) {
360             x[i] += alpha * z[i];
361             r[i] -= alpha * p[i];
362         }
363         t1 = r;
364         lu.mul_u_inv(t1);
365         t2 = t1;
366         a.mul(t2);
367         lu.mul_l_inv(t2);
368         double beta = -(p.dot(t2)) / pp;
369         for (int i = 0; i < n; ++i) {
370             z[i] = t1[i] + beta * z[i];
371             p[i] = t2[i] + beta * p[i];
372         }
373         residual = r.norm() / flen;
374         i++;
375
376         //if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
377         //    setprecision(16) << residual << endl;
378         if (fabs(residual) < eps || i > maxiter)
379             break;
380     }
381     return x;
382 }
383 }

```

FILE fem.cpp

```

1 #include "fem.h"
2
3 //-----
4 bool basic_elem_t::is_boundary(void) const {

```

```

5     return
6         up == nullptr ||
7         down == nullptr ||
8         left == nullptr ||
9         right == nullptr;
10 }
11
12 //-----
13 double elem_t::get_hx(void) const {
14     return e[1]->x - e[0]->x;
15 }
16
17 //-----
18 double elem_t::get_hy(void) const {
19     return e[3]->y - e[0]->y;
20 }
21
22 //-----
23 double elem_t::value(double x, double y, const vector_t& q) const {
24     double xp = e[0]->x;
25     double xp1 = e[1]->x;
26     double ys = e[0]->y;
27     double ys1 = e[2]->y;
28     double hx = get_hx();
29     double hy = get_hy();
30     auto x1 = [xp1, hx](double x) -> double { return (xp1 - x) / hx; };
31     auto x2 = [xp, hx](double x) -> double { return (x - xp) / hx; };
32     auto y1 = [ys1, hy](double y) -> double { return (ys1 - y) / hy; };
33     auto y2 = [ys, hy](double y) -> double { return (y - ys) / hy; };
34
35     auto psi1 = [&]() -> double { return x1(x) * y1(y); };
36     auto psi2 = [&]() -> double { return x2(x) * y1(y); };
37     auto psi3 = [&]() -> double { return x1(x) * y2(y); };
38     auto psi4 = [&]() -> double { return x2(x) * y2(y); };
39
40     double v1 = psi1() * q[e[0]->i];
41     double v2 = psi2() * q[e[1]->i];
42     double v3 = psi3() * q[e[2]->i];
43     double v4 = psi4() * q[e[3]->i];
44
45     return v1 + v2 + v3 + v4;
46 }
47
48 //-----
49 //-----
50 //-----
51
52 //-----
53 double non_linear_offset(double x, double t) {
54     int sign = (t > 0) ? 1 : -1;
55     t *= sign;
56     t = 1.0 - t;
57     t = (sign == -1) ? 1.0/t : t;
58     if (t == 1.0) return x;
59     return (1.0 - pow(t, x))/(1.0 - t);
60 }
61
62 //-----
63 grid_generator_t::grid_generator_t(double a, double b, int n, double t) : a(a), len(b-a), n1(n+1.0),
64     ↪ t(t) {}
65
66 //-----
67 double grid_generator_t::operator()(int i) const {
68     return a + len * non_linear_offset(i/n1, t);
69 }
70
71 //-----
72 int grid_generator_t::size(void) const {
73     return n1+1;
74 }
75
76 //-----
77 double grid_generator_t::back(void) const {
78     return operator()(size()-1);
79 }
80
81 //-----
82 void grid_t::calc(const grid_generator_t& gx, const grid_generator_t& gy) {
83     bes.clear();
84     bes.resize(gx.size() * gy.size());
85     int counter = 0;
86     for (int j = 0; j < gy.size(); ++j) {
87         double y = gy(j);
88         for (int i = 0; i < gx.size(); ++i) {
89             double x = gx(i);
90             basic_elem_t* down = (counter >= gx.size()) ? &bes[counter-gx.size()] : nullptr;
91             basic_elem_t* left = (counter % gx.size() > 0) ? &bes[counter-1] : nullptr;
92             bes[counter] = {counter, x, y,

```

```

92         nullptr,
93         down,
94         left,
95         nullptr
96     };
97     if (down != nullptr) down->up = &bes[counter];
98     if (left != nullptr) left->right = &bes[counter];
99     counter++;
100 }
101 }
102
103 es.clear();
104 counter = 0;
105 for (auto& i : bes) {
106     if (i.right != nullptr && i.up != nullptr && i.up->right == i.right->up && i.up->right !=
        ↪ nullptr) {
107         es.push_back({counter,
108             i.right->left,
109             i.right,
110             i.up,
111             i.up->right
112         });
113         counter++;
114     }
115 }
116
117 n = bes.size();
118 }
119
120 //-----
121 vector_t calc_true_approx(const function_2d_t& u, const vector<basic_elem_t>& bes) {
122     vector_t result(bes.size());
123     for (int i = 0; i < bes.size(); ++i)
124         result[i] = u(bes[i].x, bes[i].y);
125     return result;
126 }
127
128 //-----
129 double calc_integral_norm(const function_2d_t& u, const vector<elem_t>& es, const vector_t& q) {
130     double sum = 0;
131     for (auto& i : es) {
132         sum += calc_integral_gauss3(
133             i.e[0]->x, i.e[1]->x, 5,
134             i.e[0]->y, i.e[2]->y, 5,
135             [&](double x, double y) -> double {
136                 return fabs(u(x, y) - i.value(x, y, q));
137             }
138         );
139     }
140     return sum;
141 }
142
143 //-----
144 //-----
145 //-----
146
147 //-----
148 matrix_t calc_local_matrix_g(
149     const elem_t& e,
150     const constants_t& cs
151 ) {
152     double hx = e.get_hx();
153     double hy = e.get_hy();
154     matrix_t result;
155     matrix_t a(4, 4), b(4, 4);
156     a <<
157         2, -2, 1, -1,
158         -2, 2, -1, 1,
159         1, -1, 2, -2,
160         -1, 1, -2, 2;
161     b <<
162         2, 1, -2, -1,
163         1, 2, -1, -2,
164         -2, -1, 2, 1,
165         -1, -2, 1, 2;
166     result = cs.lambda/6.0*(hy/hx*a + hx/hy*b);
167     return result;
168 }
169
170 //-----
171 matrix_t calc_local_matrix_c(
172     const elem_t& e
173 ) {
174     double hx = e.get_hx();
175     double hy = e.get_hy();
176     matrix_t result;
177     matrix_t c(4, 4);
178     c <<
179         4, 2, 2, 1,

```



```

180         2, 4, 1, 2,
181         2, 1, 4, 2,
182         1, 2, 2, 4;
183     result = hx*hy/36.0*c;
184     return result;
185 }
186
187 //-----
188 vector_t calc_local_vector_b(
189     const elem_t& e,
190     const function_2d_t& f
191 ) {
192     vector_t fv(4);
193     fv <<
194         f(e.e[0]->x, e.e[0]->y),
195         f(e.e[1]->x, e.e[1]->y),
196         f(e.e[2]->x, e.e[2]->y),
197         f(e.e[3]->x, e.e[3]->y);
198     return calc_local_matrix_c(e) * fv;
199 }
200
201 //-----
202 vector_t calc_global_vector(
203     const vector<elem_t>& es,
204     const function<vector_t(const elem_t&)> calc_local_vector,
205     int n
206 ) {
207     vector_t result(n);
208     result.fill(0);
209     for (auto& e : es) {
210         auto b = calc_local_vector(e);
211         for (int i = 0; i < 4; ++i) {
212             result(e.e[i]->i) += b(i);
213         }
214     }
215     return result;
216 }
217
218 //-----
219 matrix_sparse_t calc_global_matrix(
220     const vector<elem_t>& es,
221     const function<matrix_t(const elem_t&)> calc_local_matrix,
222     int n
223 ) {
224     matrix_sparse_ra_t result(n);
225     for (auto& e : es) {
226         auto m = calc_local_matrix(e);
227         for (int i = 0; i < 4; ++i) {
228             for (int j = 0; j < 4; ++j) {
229                 result(e.e[i]->i, e.e[j]->j) += m(i, j);
230             }
231         }
232     }
233     return result.to_sparse();
234 }
235
236 //-----
237 //-----
238 //-----
239
240 //-----
241 double calc_integral_gauss3(
242     double a, double b, int n, // n - количество внутренних узлов
243     const function_1d_t& f
244 ) {
245     const double x1 = -sqrt(3.0/5.0);
246     const double x2 = 0;
247     const double x3 = -x1;
248     const double q1 = 5.0/9.0;
249     const double q2 = 8.0/9.0;
250     const double q3 = q1;
251     double sum = 0;
252     double xk = 0;
253     double h = (b-a)/double(n+1);
254     double h2 = h/2.0;
255
256     for (int i = 0; i < n+1; ++i) {
257         xk = a + h*i + h2;
258         sum += q1 * f(xk + x1 * h2);
259         sum += q2 * f(xk + x2 * h2);
260         sum += q3 * f(xk + x3 * h2);
261     }
262
263     sum *= h;
264     sum /= 2.0;
265     return sum;
266 }
267
268 //-----

```

```

269 double calc_integral_gauss3(
270     double ax, double bx, int nx,
271     double ay, double by, int ny,
272     const function_2d_t& f
273 ) {
274     return calc_integral_gauss3(ax, bx, nx, [ay, by, ny, f](double x)->double {
275         return calc_integral_gauss3(ay, by, ny, bind(f, x, _1));
276     });
277 }
278
279 //-----
280 //-----
281 //-----
282
283 //-----
284 function_1d_t calc_first_derivative(const function_1d_t& f) {
285     return [f](double x) -> double {
286         const double h = 0.001;
287         return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) / (12 * h);
288     };
289 }
290
291 //-----
292 function_1d_t calc_second_derivative(const function_1d_t& f) {
293     return [f](double x) -> double {
294         const double h = 0.001;
295         return (-f(x+2*h) + 16*f(x+h) - 30*f(x) + 16*f(x-h) - f(x-2*h))/(12*h*h);
296     };
297 }
298
299 //-----
300 //-----
301 //-----
302
303 //-----
304 function_3d_t calc_right_part(
305     const function_3d_t& u,
306     const constants_t& cs
307 ) {
308     // f = -div(lambda * grad u) + gamma * u + sigma * du/dt + chi * d^2 u/dt^2
309     return [=](double x, double y, double t) -> double {
310         using namespace placeholders;
311         auto ut = calc_first_derivative(bind(u, x, y, _1));
312
313         auto uxx = calc_second_derivative(bind(u, _1, y, t));
314         auto uyy = calc_second_derivative(bind(u, x, _1, t));
315         auto utt = calc_second_derivative(bind(u, x, y, _1));
316
317         return -cs.lambda * (uxx(x) + uyy(y)) + cs.gamma * u(x, y, t) + cs.sigma * ut(t) + cs.chi *
318             ↪ utt(t);
319     };
320 }
321
322 //-----
323 //-----
324 //-----
325
326 //-----
327 void calc_crank_nicolson_method(
328     const matrix_sparse_t& c,
329     const matrix_sparse_t& g,
330     const vector_t& b0,
331     const vector_t& b1,
332     const vector_t& bll,
333     const vector_t& ql,
334     const vector_t& qll,
335     const constants_t& cs,
336     const grid_generator_t& time_grid,
337     int time_i,
338     matrix_sparse_t& a,
339     vector_t& b
340 ) {
341     // Схема Кранка-Николсона
342
343     // Константы для вычислений с неравномерной сеткой по времени
344     double t0 = time_grid(time_i);
345     double t1 = time_grid(time_i-1);
346     double t2 = time_grid(time_i-2);
347
348     double d1 = t0-t2;
349     double d2 = (t0*(t0-t2-t1)+t2*t1)/2.0;
350     double m1 = (t0-t2)/(t1-t2);
351     double m2 = (t0-t1)/(t1-t2);
352
353     // Вычисляем матрицу a
354     a = c;
355     double c1 = cs.gamma/2.0 + cs.sigma/d1 + cs.chi/d2;
356     for (int i = 0; i < a.d.size(); i++)
357         a.d[i] = g.d[i]/2.0 + c.d[i]*c1;

```

```

357     for (int i = 0; i < a.l.size(); i++) {
358         a.l[i] = g.l[i]/2.0 + c.l[i]*c1;
359         a.u[i] = g.u[i]/2.0 + c.u[i]*c1;
360     }
361
362     // Рассчитываем вектор b
363     b = (b0 + b11)/2.0;
364     vector_t temp = q11;
365     g.mul(temp);
366     b = b - temp/2.0;
367
368     temp = q1*(m1*cs.chi/d2) + q11*(-cs.gamma/2.0 + cs.sigma/d1 - m2*cs.chi/d2);
369     c.mul(temp);
370
371     b = b + temp;
372 }
373
374 //-----
375 void write_first_boundary_conditions(
376     matrix_sparse_t& a,
377     vector_t& b,
378     const vector<basic_elem_t>& bes,
379     double t,
380     const function_3d_t& u
381 ) {
382     for (int i = 0; i < bes.size(); ++i) {
383         if (bes[i].is_boundary()) {
384             a.clear_line(bes[i].i);
385             a.d[bes[i].i] = 1;
386             b(bes[i].i) = u(bes[i].x, bes[i].y, t);
387         }
388     }
389 }
390
391 //-----
392 //-----
393 //-----
394
395 //-----
396 //-----
397 //-----
398
399 //-----
400 vector<vector_t> solve_differential_equation(
401     const function_3d_t& f,
402     const boundary_setter_t& set_boundary_conditions,
403     const vector_t& q0,
404     const vector_t& q1,
405     const constants_t& cs,
406     const grid_t& grid,
407     const grid_generator_t& time_grid
408 ) {
409     auto c = calc_global_matrix(grid.es, calc_local_matrix_c, grid.n);
410     auto g = calc_global_matrix(grid.es, bind(calc_local_matrix_g, _1, cs), grid.n);
411
412     auto calc_global_vector_b = [&] (int i) {
413         return calc_global_vector(
414             grid.es,
415             bind(
416                 calc_local_vector_b,
417                 _1,
418                 function_2d_t(bind(f, _1, _2, time_grid(i)))
419             ),
420             grid.n
421         );
422     };
423
424     vector_t b11 = calc_global_vector_b(0);
425     vector_t b1 = calc_global_vector_b(1);
426     vector_t b0;
427
428     vector_t q11 = q0;
429     vector_t q1 = q1;
430     vector_t q;
431
432     vector<vector_t> result;
433     result.push_back(q0);
434     result.push_back(q1);
435
436     matrix_sparse_t a(grid.n);
437     vector_t b(grid.n);
438     for (int i = 2; i < time_grid.size(); ++i) {
439         b0 = calc_global_vector_b(i);
440         calc_crank_nicolson_method(c, g, b0, b1, b11, q1, q11, cs, time_grid, i, a, b);
441         set_boundary_conditions(a, b, grid.bes, time_grid(i));
442
443         q = solve_by_los_lu(a, b, 1000, 1e-16, false);
444
445         result.push_back(q);

```

```

446         b1l = bl;
447         bl = b0;
448
449         ql1 = ql;
450         ql = q;
451     }
452
453     return result;
454 }
455

```

## 5.3 Исследования

FILE main.cpp

```

1  #include <iostream>
2  #include <cmath>
3  #include <string>
4  #include <fstream>
5  #include <thread>
6  #include <future>
7  #include "lib.h"
8  #include "fem.h"
9
10 using namespace std;
11 using namespace placeholders;
12
13 //-----
14 struct fem_result_t
15 {
16     double integral_residual;
17     double norm_residual;
18     double time;
19 };
20
21 //-----
22 fem_result_t calc_fem_residual(
23     const function_3d_t& u,
24     const grid_generator_t& x_grid,
25     const grid_generator_t& y_grid,
26     const grid_generator_t& time_grid,
27     const constants_t& c = {1, 1, 1, 1}
28 ) {
29     fem_result_t res;
30     res.time = calc_time_microseconds([&]() {
31         auto f = calc_right_part(u, c);
32         boundary_setter_t set_boundary_conditions = bind(write_first_boundary_conditions, _1, _2, _3,
33             ↪ _4, u);
34
35         grid_t grid;
36         grid.calc(x_grid, y_grid);
37
38         vector_t q0 = calc_true_approx(bind(u, _1, _2, time_grid(0)), grid.bes);
39         vector_t q1 = calc_true_approx(bind(u, _1, _2, time_grid(1)), grid.bes);
40         vector_t q = calc_true_approx(bind(u, _1, _2, time_grid.back()), grid.bes);
41
42         auto steps = solve_differential_equation(f, set_boundary_conditions, q0, q1, c, grid,
43             ↪ time_grid);
44
45         res.integral_residual = calc_integral_norm(bind(u, _1, _2, time_grid.back()), grid.es,
46             ↪ steps.back());
47         res.norm_residual = (q.steps.back()).norm() / q.size();
48     });
49     return res;
50 }
51
52 //-----
53
54 template<class Ret, class Key>
55 class async_performer_t
56 {
57 public:
58     void add(const function<Ret(void)>& f, const Key& key) {
59         mf[key] = async(std::launch::deferred, f);
60     }
61
62     void finish(void) {
63         int counter = 0;
64         for (auto i = mf.begin(); i != mf.end(); ++i) {
65             //if (counter % (mf.size()/10000 + 1) == 0)

```

```

66         write_percent(double(counter)/mf.size());
67         auto value = i->second.get();
68         m[i->first] = value;
69         counter++;
70     }
71     cout << "\r      \r";
72 }
73
74 auto begin(void) { return m.begin(); }
75 auto end(void) { return m.end(); }
76
77 Ret& operator[](const Key& key) { return m[key]; }
78 const Ret& operator[](const Key& key) const { return m[key]; }
79 private:
80     map<Key, future<Ret>> mf;
81     map<Key, Ret> m;
82 };
83
84 //-----
85 //-----
86 //-----
87 //-----
88
89 template<class ForwardIt, class GetValue>
90 double max_element_ignore_nan(ForwardIt first, ForwardIt last, GetValue get) {
91     return get(*max_element(first, last, [get] (auto& a, auto& b) -> bool {
92         if (isnan(get(a)))
93             return true;
94         else
95             return get(a) < get(b);
96     }));
97 }
98
99 //-----
100 void investigate_t_changing(
101     int n,
102     const string& filename,
103     const function<fem_result_t(double)>& ft
104 ) {
105     auto uniform_value = ft(0);
106
107     async_performer_t<fem_result_t, int> performer;
108
109     grid_generator_t grid(-1, 1, n);
110     for (int i = 0; i < grid.size(); i++) {
111         performer.add([i, ft, grid] () -> fem_result_t {
112             return ft(grid(i));
113         }, i);
114     }
115
116     performer.finish();
117
118     int counter = 0;
119     auto integral_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto&
120     ↪ a) -> double { return a.second.integral_residual; });
121     auto norm_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto& a)
122     ↪ -> double { return a.second.norm_residual; });
123
124     ofstream fout(filename + ".txt");
125     fout << "t\tintegral\t\norm\tuniform\tintegral\tuniform_norm\ttime" << endl;
126     for (int i = 0; i < grid.size(); i++) {
127         auto v = performer[i];
128         fout
129             << grid(i) << "\t"
130             << (isnan(v.integral_residual) ? integral_residual_max : v.integral_residual) << "\t"
131             << (isnan(v.norm_residual) ? norm_residual_max : v.norm_residual) << "\t"
132             << uniform_value.integral_residual << "\t"
133             << uniform_value.norm_residual << "\t"
134             << v.time << endl;
135     }
136     fout.close();
137 }
138
139 //-----
140 void investigate_t2_changing(
141     int n,
142     const string& filename,
143     const function<fem_result_t(double, double)>& ft
144 ) {
145     async_performer_t<fem_result_t, pair<int, int>> performer;
146
147     grid_generator_t grid(-1, 1, n);
148     for (int i = 0; i < grid.size(); i++) {
149         for (int j = 0; j < grid.size(); j++) {
150             performer.add([i, j, ft, grid] () -> fem_result_t {
151                 return ft(grid(i), grid(j));
152             }, {i, j});
153         }
154     }
155 }

```

```

152 }
153
154 performer.finish();
155
156 auto integral_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto&
    ↪ a) -> double { return a.second.integral_residual; });
157 auto norm_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto& a)
    ↪ -> double { return a.second.norm_residual; });
158
159 ofstream fout(filename + ".integral.txt");
160 ofstream fout2(filename + ".norm.txt");
161 ofstream fout3(filename + ".time.txt");
162 int last_line = 0;
163 for (int i = 0; i < grid.size(); i++) {
164     for (int j = 0; j < grid.size(); j++) {
165         auto v = performer[{i, j}];
166         fout << (isnan(v.integral_residual) ? integral_residual_max : v.integral_residual) << "\t";
167         fout2 << (isnan(v.norm_residual) ? norm_residual_max : v.norm_residual) << "\t";
168         fout3 << v.time << "\t";
169     }
170     fout << endl;
171     fout2 << endl;
172     fout3 << endl;
173 }
174 fout.close();
175 fout2.close();
176
177 fout.open(filename + ".x.txt");
178 for (int i = 0; i < grid.size(); i++)
179     fout << grid(i) << endl;
180 fout.close();
181
182 fout.open(filename + ".y.txt");
183 for (int i = 0; i < grid.size(); i++)
184     fout << grid(i) << endl;
185 fout.close();
186 }
187
188 //-----
189 void investigate_functions(
190     const string& filename,
191     const function<fem_result_t(const function_3d_t&)>& f
192 ) {
193     vector<pair<function_3d_t, string>> spaces, times;
194
195     spaces.push_back({[] (double x, double y, double t) -> double { return 1; }, "$1$"});
196     spaces.push_back({[] (double x, double y, double t) -> double { return x+y; }, "$x+y$"});
197     spaces.push_back({[] (double x, double y, double t) -> double { return x*x+y*y; }, "$x^2+y^2$"});
198     spaces.push_back({[] (double x, double y, double t) -> double { return x*x*y+y*y*y; },
    ↪ "$x^2y+y^3$"});
199     spaces.push_back({[] (double x, double y, double t) -> double { return x*y*y; }, "$xy^2$"});
200     spaces.push_back({[] (double x, double y, double t) -> double { return x*x*x*x+y*y*y*y; },
    ↪ "$x^4+y^4$"});
201     spaces.push_back({[] (double x, double y, double t) -> double { return exp(x*y); }, "$e^{xy}$"});
202
203     times.push_back({[] (double x, double y, double t) -> double { return 0; }, "$0$"});
204     times.push_back({[] (double x, double y, double t) -> double { return t; }, "$t$"});
205     times.push_back({[] (double x, double y, double t) -> double { return t*t; }, "$t^2$"});
206     times.push_back({[] (double x, double y, double t) -> double { return t*t*t; }, "$t^3$"});
207     times.push_back({[] (double x, double y, double t) -> double { return t*t*t*t; }, "$t^4$"});
208     times.push_back({[] (double x, double y, double t) -> double { return exp(t); }, "$e^t$"});
209
210     async_performer_t<fem_result_t, pair<string, string>> performer;
211
212     for (auto& i : spaces) {
213         for (auto& j : times) {
214             performer.add([i, j, &f]() -> fem_result_t {
215                 return f(function_3d_t([&] (double x, double y, double t) -> double { return
    ↪ i.first(x, y, t) + j.first(x, y, t); }));
216             }, {i.second, j.second});
217         }
218     }
219
220     performer.finish();
221
222     ofstream fout(filename);
223     fout << "a\t";
224     for (auto& i : times)
225         fout << i.second << "\t";
226     for (auto& i : spaces) {
227         fout << endl << i.second << "\t";
228         for (auto& j : times) {
229             auto v = performer[{i.second, j.second}];
230             fout << "\\scalebox{.75}{\\tcell{$" << write_for_latex_double(v.integral_residual, 2) <<
    ↪ "$\\\\\\$" << write_for_latex_double(v.norm_residual, 2) << "$\\\\\\$" << int(v.time/1000)
    ↪ << "}}\\t";
231         }

```

```

232     }
233     fout.close();
234 }
235
236 //-----
237 void investigate_grid_changing(
238     const string& filename,
239     const function<pair<fem_result_t, int>(int)>& fi,
240     int n
241 ) {
242     async_performer_t<pair<fem_result_t, int>, int> performer;
243
244     for (int i = 0; i < n; i+=3) {
245         performer.add([i, fi] () -> pair<fem_result_t, int> {
246             return fi(i);
247         }, i);
248     }
249
250     performer.finish();
251
252     ofstream fout(filename + ".txt");
253     fout << "i\tintegral\tnorm\ttime" << endl;
254     for (int i = 0; i < n; i+=3) {
255         auto v = performer[i];
256         fout
257             << v.second << "\t"
258             << v.first.integral_residual << "\t"
259             << v.first.norm_residual << "\t"
260             << v.first.time << endl;
261     }
262     fout.close();
263 }
264
265 //-----
266 //-----
267 //-----
268
269 int main() {
270     cout << calc_time_microseconds([]){
271         investigate_grid_changing(
272             "space_sgrid",
273             [] (int sz) -> pair<fem_result_t, int> {
274                 return {
275                     calc_fem_residual(
276                         [] (double x, double y, double t) -> double { return exp(x*y) + exp(t*t); },
277                         grid_generator_t(0, 1, 5+sz),
278                         grid_generator_t(0, 1, 5+sz),
279                         grid_generator_t(0, 1, 300)
280                     ),
281                     (7+sz)*(7+sz)
282                 };
283             },
284             70
285         );
286
287     investigate_grid_changing(
288         "time_sgrid",
289         [] (int sz) -> pair<fem_result_t, int> {
290             return {
291                 calc_fem_residual(
292                     [] (double x, double y, double t) -> double { return exp(x*y) + exp(t*t); },
293                     grid_generator_t(0, 1, 20),
294                     grid_generator_t(0, 1, 20),
295                     grid_generator_t(0, 1, 1+sz)
296                 ),
297                 3+sz
298             };
299         },
300         500
301     );
302
303     investigate_functions(
304         "functions_table_10_10_10.txt",
305         [] (const function_3d_t& u) -> fem_result_t {
306             return calc_fem_residual(u, grid_generator_t(0, 1, 10), grid_generator_t(0, 1, 10),
307                 ↪ grid_generator_t(0, 1, 10));
308         }
309     );
310
311     investigate_functions(
312         "functions_table_50_50_50.txt",
313         [] (const function_3d_t& u) -> fem_result_t {
314             return calc_fem_residual(u, grid_generator_t(0, 1, 50), grid_generator_t(0, 1, 50),
315                 ↪ grid_generator_t(0, 1, 50));
316         }
317     );
318
319     vector<pair<function_3d_t, int>> u_space_mas;

```

```

318 u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x + y*y + t*t;
    ↪ }, 0});
319 u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x*x*x + y*y*y*y
    ↪ + t*t*t*t; }, 1});
320 u_space_mas.push_back({[] (double x, double y, double t) -> double { return exp(x*y) +
    ↪ exp(t*t); }, 2});
321 u_space_mas.push_back({[] (double x, double y, double t) -> double { return exp((1-x)*(1-y)) +
    ↪ exp((1-t)*(1-t)); }, 3});
322 u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x*x +
    ↪ y*y*y*y*x*x*t + t*t*exp(t); }, 4});
323
324 for (auto& i : u_space_mas) {
325     auto& u = i.first;
326     investigate_t_changing(
327         750,
328         "time_tgrid_" + to_string(i.second),
329         [u] (double tt) -> fem_result_t {
330             return calc_fem_residual(u, grid_generator_t(0, 1, 10), grid_generator_t(0, 1,
    ↪ 10), grid_generator_t(0, 1, 10, tt));
331         }
332     );
333
334     investigate_t2_changing(
335         75,
336         "space_tgrid_" + to_string(i.second),
337         [u] (double tx, double ty) -> fem_result_t {
338             return calc_fem_residual(u, grid_generator_t(0, 1, 10, tx), grid_generator_t(0, 1,
    ↪ 10, ty), grid_generator_t(0, 1, 10));
339         }
340     );
341 }
342 }/1000/1000 << "s" << endl;
343 system("pause");
344 }

```

## 5.4 Визуализация

### FILE plot.py

```

1 import math
2 import pylab
3 import numpy
4 import sys
5 import matplotlib.pyplot as plt
6 import matplotlib.lines as lines
7 import matplotlib as mpl
8 from mpl_toolkits.mplot3d import Axes3D
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from matplotlib import ticker, cm
12
13 DPI = 200
14
15 def make_image(xpath, ypath, zpath, resultpath, mytitle):
16     x = numpy.loadtxt(xpath)
17     y = numpy.loadtxt(ypath)
18     z = numpy.loadtxt(zpath)
19
20     X, Y = np.meshgrid(x, y)
21
22     fig, ax = plt.subplots()
23     #locator=ticker.LogLocator(base=math.pow(10, 1/10000))
24     cs = ax.contourf(X, Y, z, 55, cmap=cm.coolwarm)
25     cbar = fig.colorbar(cs)
26
27     plt.title(mytitle, fontsize=19)
28     plt.xlabel(r'$t_x$', fontsize=15)
29     plt.ylabel(r'$t_y$', fontsize=15)
30     plt.tick_params(axis='both', labelsize=10)
31     plt.grid(alpha=0.25)
32     plt.savefig(resultpath, dpi=DPI)
33     plt.clf()
34
35 def make_images(i):
36     make_image(f"space_tgrid_{i}.x.txt", f"space_tgrid_{i}.y.txt", f"space_tgrid_{i}.integral.txt",
    ↪ f"space_tgrid_{i}_integral.png", r"Integral of functions difference");
37     make_image(f"space_tgrid_{i}.x.txt", f"space_tgrid_{i}.y.txt", f"space_tgrid_{i}.norm.txt",
    ↪ f"space_tgrid_{i}_norm.png", r"Norm of $q$ vectors difference");
38     make_image(f"space_tgrid_{i}.x.txt", f"space_tgrid_{i}.y.txt", f"space_tgrid_{i}.time.txt",
    ↪ f"space_tgrid_{i}_time.png", r"Solving time");
39

```



```
40 if __name__ == '__main__':  
41     plt.rc('text', usetex=True)  
42     plt.rc('font', family='serif')  
43  
44     make_images(0)  
45     make_images(1)  
46     make_images(2)  
47     make_images(3)  
48     make_images(4)
```