

Министерство науки и высшего образования Российской
Федерации

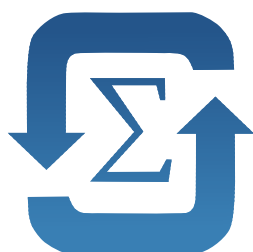
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ**»



Кафедра прикладной математики

Метод конечных элементов

Пояснительная записка к курсовому проекту



Факультет:	ПМИ
Группа:	ПМ-61
Студент:	Шепрут И.И.
Преподаватель:	Персова М.Г.

Новосибирск
2020

1 Задание

Реализовать МКЭ для двумерной задачи для гиперболического уравнения в декартовой системе координат. Базисные функции — биквадратичные. 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:

$$u|_S = u_s$$

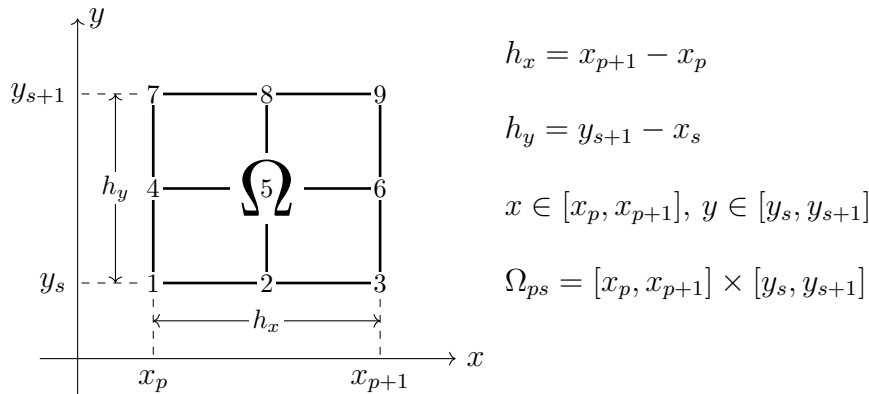
Формулы для билинейных базисных функций прямоугольных элементов:

$$X_1(x) = 1 + \frac{3x_p}{h_x} + \frac{2x^2}{h_x^2} - \frac{x(3h_x + 4x_p)}{h_x^2} + \frac{2x_p^2}{h_x^2}$$

$$X_2(x) = -\frac{4x^2}{h_x^2} + \frac{4x(h_x + 2x_p)}{h_x^2} - \frac{4x_p(h_x + x_p)}{h_x^2}$$

$$X_3(x) = \frac{2x^2}{h_x^2} - \frac{x(h_x + 4x_p)}{h_x^2} + \frac{x_p(h_x + 2x_p)}{h_x^2}$$

Формулы $Y_{1,2,3}(y)$ выглядят аналогично $X_{1,2,3}(x)$, только с заменой $x \rightarrow y$, $h_x \rightarrow h_y$. Конечный элемент для биквадратичных базисов представляется так:



Значения функций в узлах:

$$\begin{array}{lll} \psi_1(x, y) = X_1(x)Y_1(y) & \psi_2(x, y) = X_2(x)Y_1(y) & \psi_3(x, y) = X_3(x)Y_1(y) \\ \psi_4(x, y) = X_1(x)Y_2(y) & \psi_5(x, y) = X_2(x)Y_2(y) & \psi_6(x, y) = X_3(x)Y_2(y) \\ \psi_7(x, y) = X_1(x)Y_3(y) & \psi_8(x, y) = X_2(x)Y_3(y) & \psi_9(x, y) = X_3(x)Y_3(y) \end{array}$$

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

$$u_{ps}^*(x, y) = \sum_{i=1}^9 q_i \psi_i(x, y)$$

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:

$$\mathbf{G} = \frac{\lambda}{90} \frac{h_x}{h_y} \cdot \begin{bmatrix} 28 & 14 & -7 & -32 & -16 & 8 & 4 & 2 & -1 \\ 14 & 112 & 14 & -16 & -128 & -16 & 2 & 16 & 2 \\ -7 & 14 & 28 & 8 & -16 & -32 & -1 & 2 & 4 \\ -32 & -16 & 8 & 64 & 32 & -16 & -32 & -16 & 8 \\ -16 & -128 & -16 & 32 & 256 & 32 & -16 & -128 & -16 \\ 8 & -16 & -32 & -16 & 32 & 64 & 8 & -16 & -32 \\ 4 & 2 & -1 & -32 & -16 & 8 & 28 & 14 & -7 \\ 2 & 16 & 2 & -16 & -128 & -16 & 14 & 112 & 14 \\ -1 & 2 & 4 & 8 & -16 & -32 & -7 & 14 & 28 \end{bmatrix} + \frac{\lambda}{90} \frac{h_y}{h_x} \cdot \begin{bmatrix} 28 & -32 & 4 & 14 & -16 & 2 & -7 & 8 & -1 \\ -32 & 64 & -32 & -16 & 32 & -16 & 8 & -16 & 8 \\ 4 & -32 & 28 & 2 & -16 & 14 & -1 & 8 & -7 \\ 14 & -16 & 2 & 112 & -128 & 16 & 14 & -16 & 2 \\ -16 & 32 & -16 & -128 & 256 & -128 & -16 & 32 & -16 \\ 2 & -16 & 14 & 16 & -128 & 112 & 2 & -16 & 14 \\ -7 & 8 & -1 & 14 & -16 & 2 & 28 & -32 & 4 \\ 8 & -16 & 8 & -16 & 32 & -16 & -32 & 64 & -32 \\ -1 & 8 & -7 & 2 & -16 & 14 & 4 & -32 & 28 \end{bmatrix}$$

$$\mathbf{C} = \gamma \frac{h_x h_y}{900} \cdot \begin{bmatrix} 16 & 8 & -4 & 8 & 4 & -2 & -4 & -2 & 1 \\ 8 & 64 & 8 & 4 & 32 & 4 & -2 & -16 & -2 \\ -4 & 8 & 16 & -2 & 4 & 8 & 1 & -2 & -4 \\ 8 & 4 & -2 & 64 & 32 & -16 & 8 & 4 & -2 \\ 4 & 32 & 4 & 32 & 256 & 32 & 4 & 32 & 4 \\ -2 & 4 & 8 & -16 & 32 & 64 & -2 & 4 & 8 \\ -4 & -2 & 1 & 8 & 4 & -2 & 16 & 8 & -4 \\ -2 & -16 & -2 & 4 & 32 & 4 & 8 & 64 & 8 \\ 1 & -2 & -4 & -2 & 4 & 8 & -4 & 8 & 16 \end{bmatrix} \quad \begin{aligned} \mathbf{M}^\gamma &= \bar{\gamma} \mathbf{C} \\ \mathbf{f} &= (f_1, f_2, f_3, f_4)^t \\ \mathbf{b} &= \mathbf{C} \cdot \mathbf{f} \end{aligned}$$

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 γ , σ , χ 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   Igrid_generator_t(double a, double b, int n, double t = 0);
5   Idouble operator()(int i) const;
6   Iint size(void) const;
7   Idouble back(void) const;
8 private:
9   Idouble a, len, t, n1;
10 };

```

Для задания узла конечного элемента структура:

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

```

Для задания конечного элемента используется структура:

```
1 struct elem_t
2 {
3   Iint i; /// Номер конечного элемента
4   Ibasic_elem_t* e[4]; /// Указатели на все 4 элемента конечного узла, нумерация такая:
5   I/**
6   I^
7   I^ 3 +-----+ 4
8   I^ |   |   |
9   I^ | 1 +-----+ 2
10  I^+-----+ X
11  I^ |
12  I^ */
13
14   Idouble get_hx(void) const; /// Ширина конечного элемента
15   Idouble get_hy(void) const; /// Высота конечного элемента
16
17   I/** Рассчитать значение внутри конечного элемента. q - вектор рассчитываемых весов. */
18   Idouble 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   ^^Ivector<elem_t> es; /// Массив конечных элементов сетки
5   ^^Ivector<basic_elem_t> bes; /// Массив узлов сетки
6   ^^Int n; /// Число узлов
7
8   ^^I/** Рассчитать неравномерную сетку. */
9   ^^Ivoid 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   ^^Imatrix_sparse_ra_t(int n);
5
6   ^^I/** Установить значение в позиции (i, j) */
7   ^^Idouble& operator()(int i, int j);
8
9   ^^I/** Получить значение в позиции (i, j). Если туда ещё не устанавливалось значение, вызывается
10  ↪ исключение. */
11   ^^Iconst double& operator()(int i, int j) const;
12
13   ^^I/** Преобразует текущую матрицу к разреженной матрице. */
14   ^^Imatrix_sparse_t to_sparse(void) const;
15 private:
16   ^^Int n;
17   ^^Ivector<double> dm;
18   ^^Ivector<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.

time(t) space(x, y)	0	t	t^2	t^3	t^4	e^t
1	$0.42 \cdot 10^{-13}$ $0.13 \cdot 10^{-15}$ 73	$0.46 \cdot 10^{-12}$ $0.34 \cdot 10^{-13}$ 74	$0.66 \cdot 10^{-11}$ $0.4 \cdot 10^{-12}$ 71	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 72	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 72	$0.13 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 104
$x+y$	$0.19 \cdot 10^{-11}$ $0.13 \cdot 10^{-12}$ 73	$0.39 \cdot 10^{-11}$ $0.23 \cdot 10^{-12}$ 73	$0.21 \cdot 10^{-11}$ $0.14 \cdot 10^{-12}$ 73	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 76	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 74	$0.13 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 107
x^2+y	$0.53 \cdot 10^{-12}$ $0.34 \cdot 10^{-13}$ 72	$0.52 \cdot 10^{-12}$ $0.33 \cdot 10^{-13}$ 73	$0.68 \cdot 10^{-12}$ $0.48 \cdot 10^{-13}$ 71	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 73	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 72	$0.13 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 103
x^2y+y^3	$0.24 \cdot 10^{-4}$ $0.44 \cdot 10^{-13}$ 71	$0.24 \cdot 10^{-4}$ $0.67 \cdot 10^{-13}$ 73	$0.24 \cdot 10^{-4}$ $0.96 \cdot 10^{-13}$ 72	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 74	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 73	$0.13 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 106
xy^2	$0.6 \cdot 10^{-12}$ $0.37 \cdot 10^{-13}$ 71	$0.74 \cdot 10^{-12}$ $0.47 \cdot 10^{-13}$ 73	$0.65 \cdot 10^{-12}$ $0.44 \cdot 10^{-13}$ 72	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 73	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 75	$0.13 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 105
x^4+y^4	$0.67 \cdot 10^{-4}$ $0.7 \cdot 10^{-7}$ 74	$0.67 \cdot 10^{-4}$ $0.7 \cdot 10^{-7}$ 75	$0.67 \cdot 10^{-4}$ $0.7 \cdot 10^{-7}$ 72	$0.17 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 73	$0.17 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 74	$0.14 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 106
e^{xy}	$0.21 \cdot 10^{-5}$ $0.12 \cdot 10^{-8}$ 103	$0.21 \cdot 10^{-5}$ $0.12 \cdot 10^{-8}$ 105	$0.21 \cdot 10^{-5}$ $0.12 \cdot 10^{-8}$ 102	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 107	$0.16 \cdot 10^{-3}$ $0.95 \cdot 10^{-5}$ 107	$0.13 \cdot 10^{-3}$ $0.76 \cdot 10^{-5}$ 135

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.

time(t) space(x, y)	0	t	t^2	t^3	t^4	e^t
1	$0.86 \cdot 10^{-12}$ $0.8 \cdot 10^{-15}$ 9704	$0.3 \cdot 10^{-11}$ $0.25 \cdot 10^{-13}$ 9747	$0.19 \cdot 10^{-11}$ $0.83 \cdot 10^{-14}$ 9737	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 9726	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 9735	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 12939
$x+y$	$0.14 \cdot 10^{-11}$ $0.11 \cdot 10^{-13}$ 10007	$0.51 \cdot 10^{-11}$ $0.59 \cdot 10^{-13}$ 10076	$0.21 \cdot 10^{-11}$ $0.11 \cdot 10^{-13}$ 10064	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10083	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10093	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 13396
x^2+y	$0.14 \cdot 10^{-11}$ $0.13 \cdot 10^{-13}$ 10053	$0.19 \cdot 10^{-11}$ $0.13 \cdot 10^{-13}$ 10097	$0.19 \cdot 10^{-11}$ $0.11 \cdot 10^{-13}$ 10064	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10098	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10074	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 13426
x^2y+y^3	$0.24 \cdot 10^{-6}$ $0.28 \cdot 10^{-14}$ 10157	$0.24 \cdot 10^{-6}$ $0.56 \cdot 10^{-14}$ 10123	$0.24 \cdot 10^{-6}$ $0.49 \cdot 10^{-14}$ 10078	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10138	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10089	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 13466
xy^2	$0.25 \cdot 10^{-12}$ $0.11 \cdot 10^{-14}$ 10125	$0.11 \cdot 10^{-11}$ $0.17 \cdot 10^{-14}$ 10152	$0.14 \cdot 10^{-11}$ $0.97 \cdot 10^{-14}$ 10225	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10190	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10133	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 13438
x^4+y^4	$0.68 \cdot 10^{-6}$ $0.18 \cdot 10^{-10}$ 10123	$0.68 \cdot 10^{-6}$ $0.18 \cdot 10^{-10}$ 10133	$0.68 \cdot 10^{-6}$ $0.18 \cdot 10^{-10}$ 10116	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10135	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 10166	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 13447
e^{xy}	$0.23 \cdot 10^{-7}$ $0.24 \cdot 10^{-12}$ 12915	$0.23 \cdot 10^{-7}$ $0.23 \cdot 10^{-12}$ 12958	$0.23 \cdot 10^{-7}$ $0.24 \cdot 10^{-12}$ 12919	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 13309	$0.77 \cdot 10^{-5}$ $0.89 \cdot 10^{-7}$ 13310	$0.66 \cdot 10^{-5}$ $0.77 \cdot 10^{-7}$ 16487

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.

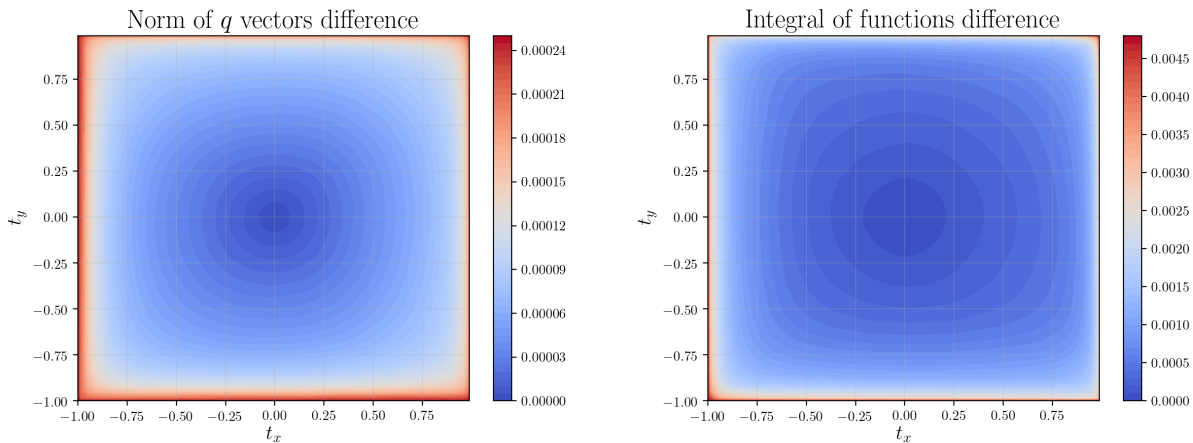
Таким образом, можно исследовать различные неравномерные сетки, изменяя параметр от -1 до 1 , где точка $t = 0$ будет являться результатом на равномерной сетке.

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

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



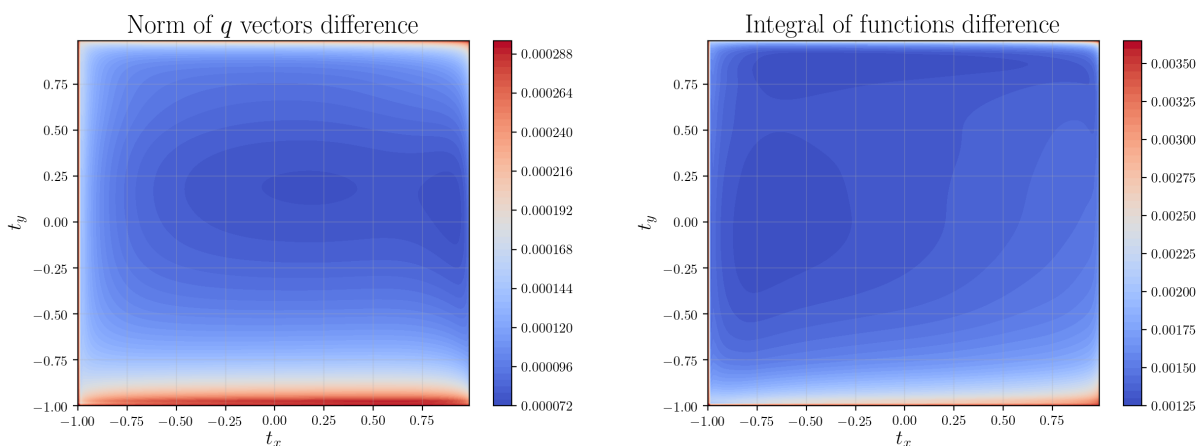
Вывод: так как эта функция полностью аппроксимируется данным методом в узлах, то не имеет значения насколько сетка неравномерна, примерно во всех элементах она

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

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



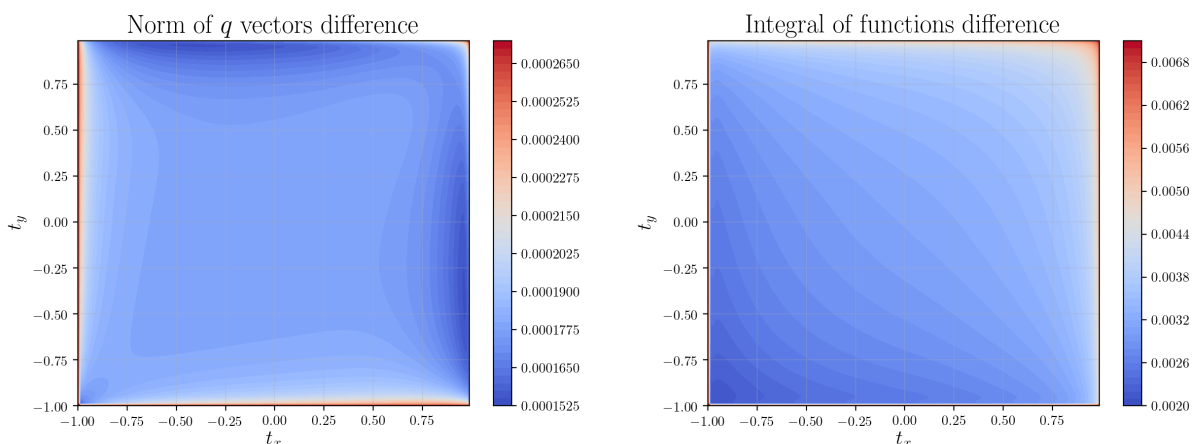
Вывод: согласно левому графику норма в узлах лучше всего аппроксимируется при сгущении по y в одну или другую сторону. По x же неравномерность сетки практически ни на что не влияет.

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

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

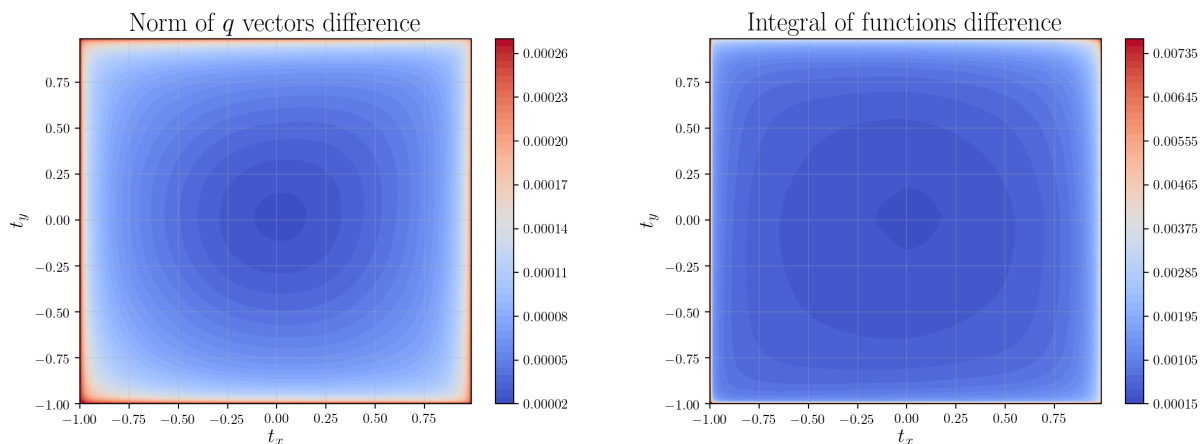


Вывод: согласно левому графику аппроксимация в узлах тоже имеет некоторые оптимальные значения, причем точность увеличивается на порядок.

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

4.2.2.4 Функция 4

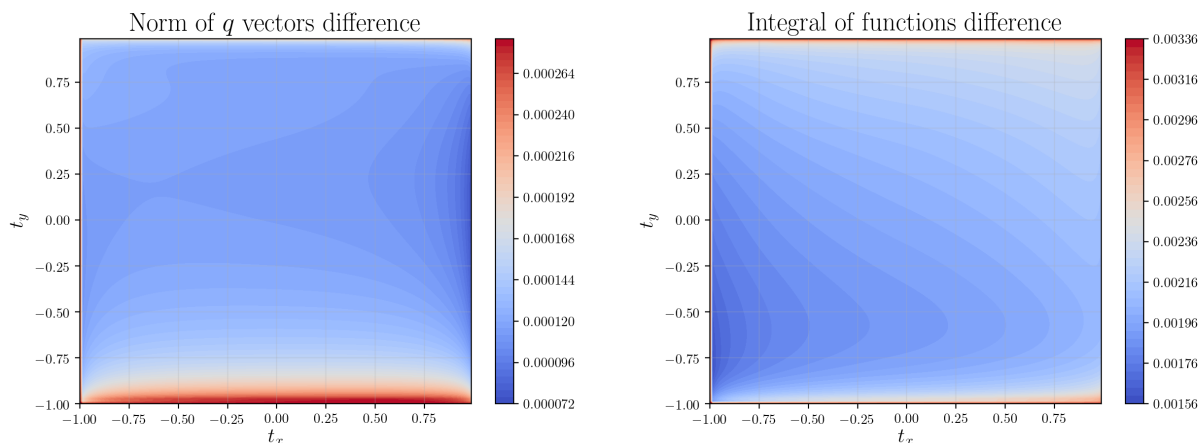
$$u = e^{(1-x)(1-y)} + e^{(1-t)^2}$$



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.

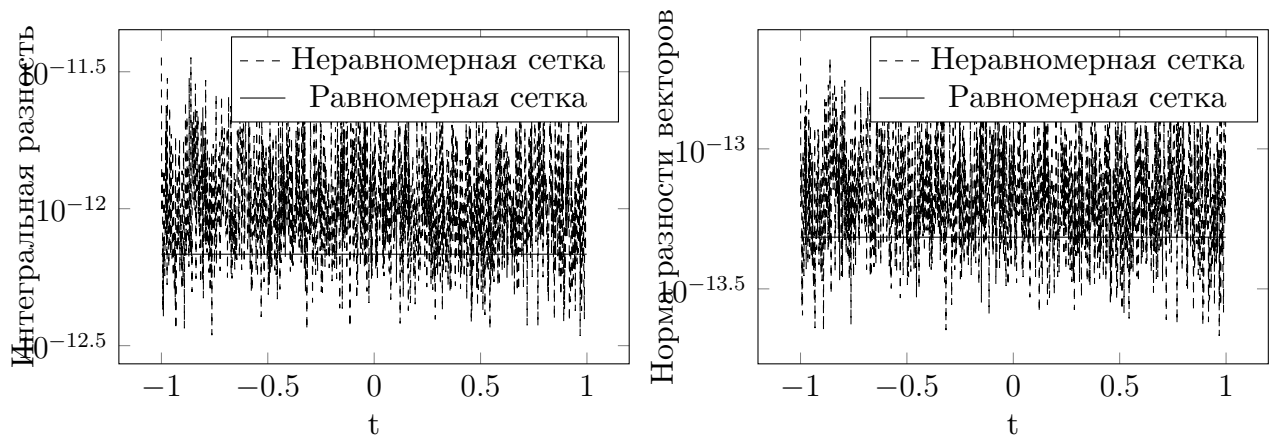
Вывод: согласно норме в узлах для неполиномиальных функций оптимальными являются параметры в окрестности ± 1 .

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

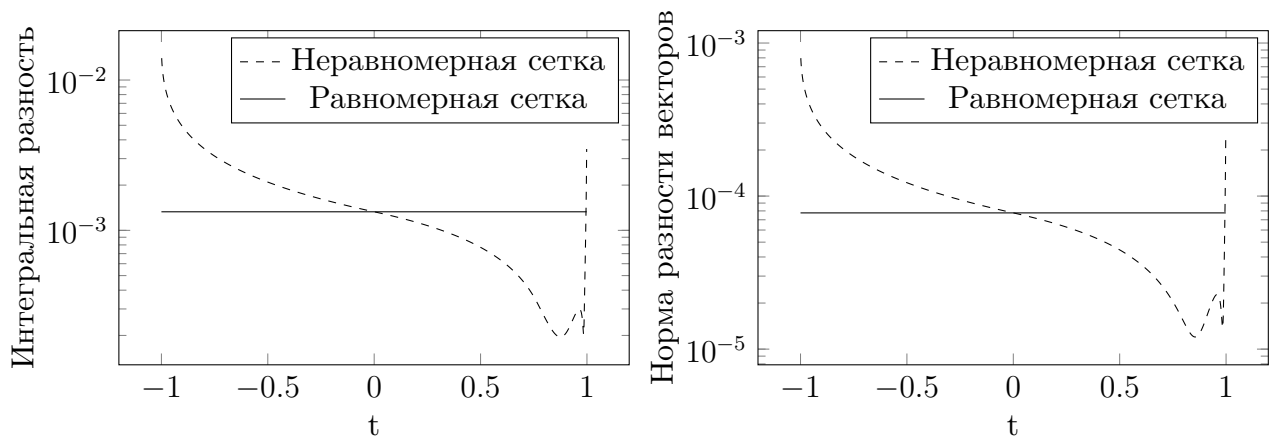
$$u = x^2 + y^2 + t^2$$



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

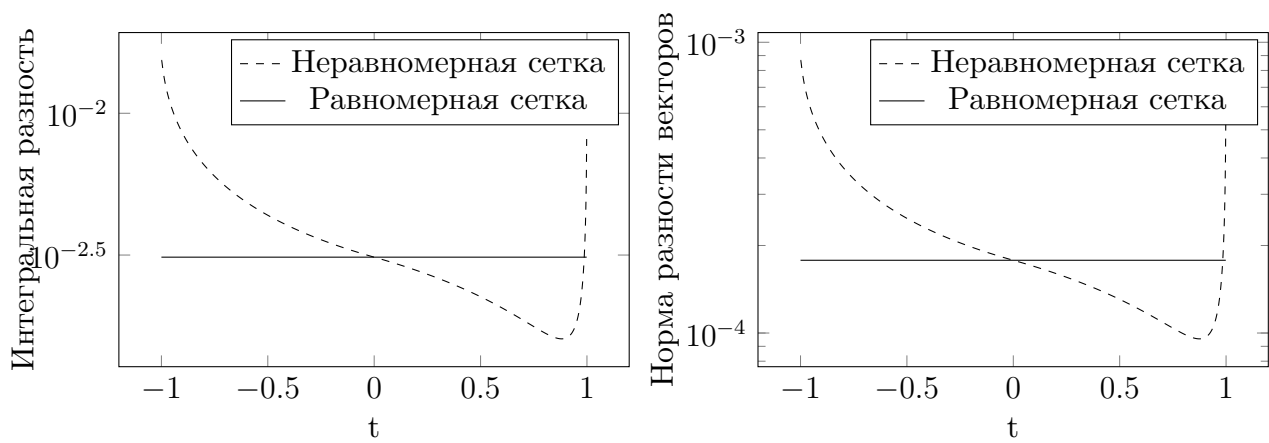
$$u = x^4 + y^3x + t^4$$



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

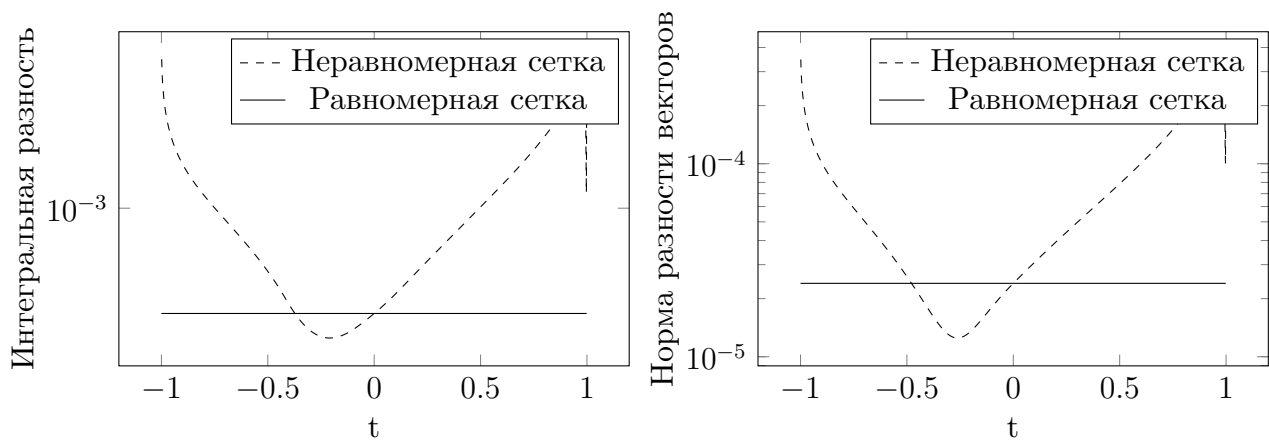
$$u = e^{xy} + e^{t^2}$$



Вывод: всё аналогично предыдущему.

4.2.3.4 Функция 4

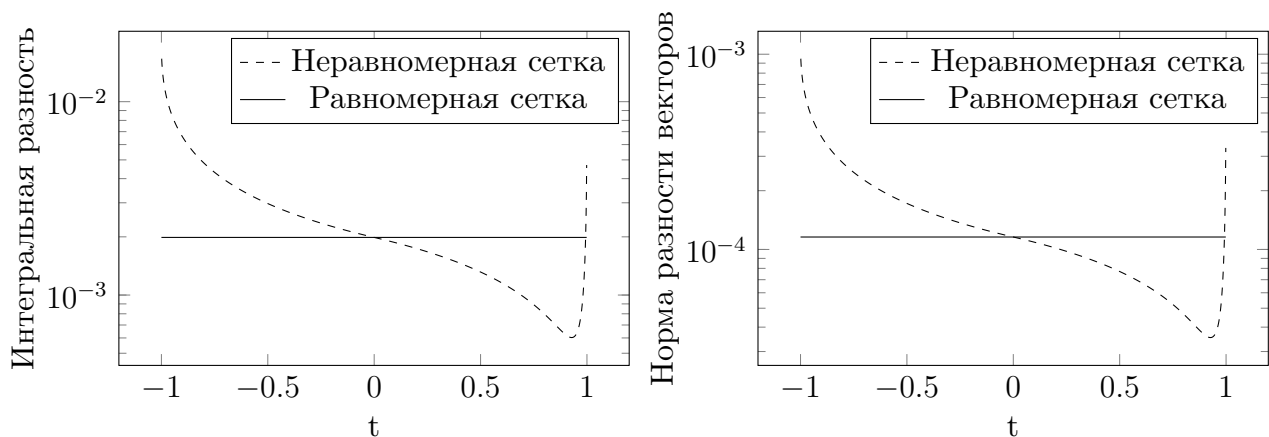
$$u = e^{(1-x)(1-y)} + e^{(1-t)^2}$$



Вывод: эта функция является перевернутой версией предыдущей, но график аналогично не перевернулся, 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.

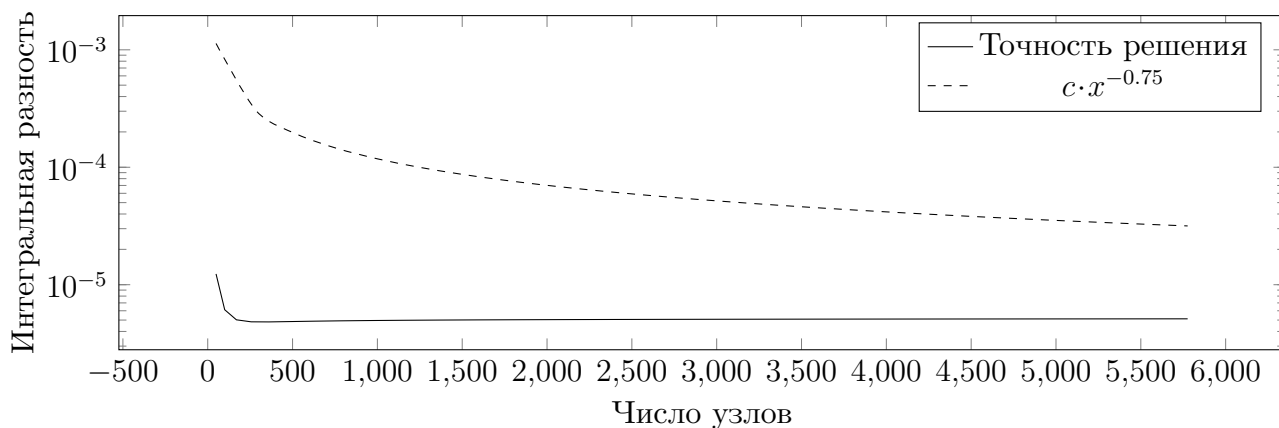
4.3 Порядок сходимости

Исследуется на функции:

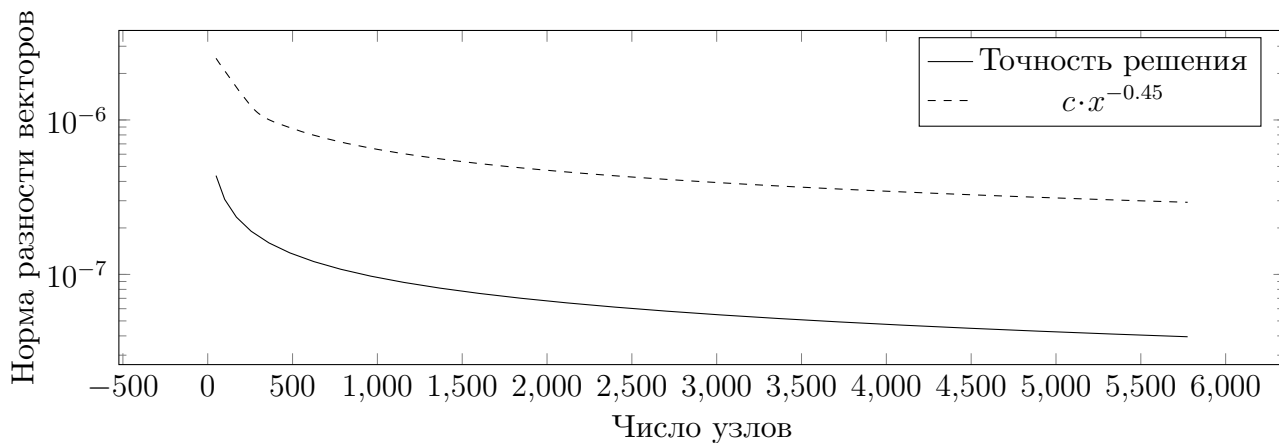
$$u(x, y, t) = e^{xy} + e^{t^2}$$

4.3.1 Увеличение размерности по пространству

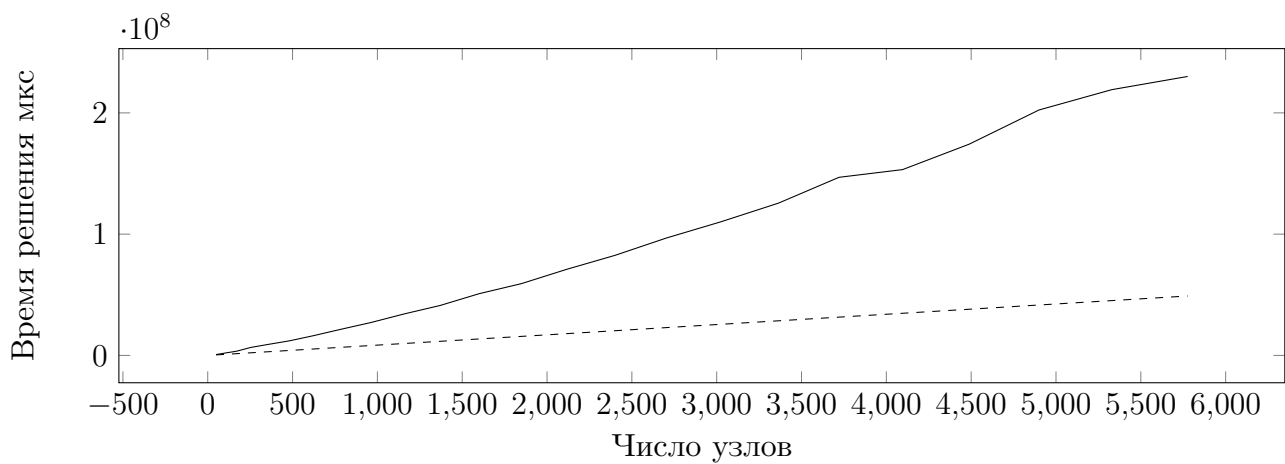
Сетка по пространству: $(x, y) \in [0, 1] \times [0, 1]$. Сетка по времени: $t \in [0, 1]$, количество элементов сетки 200.



Вывод: согласно интегральной норме, порядок сходимости ≈ 0.75 .



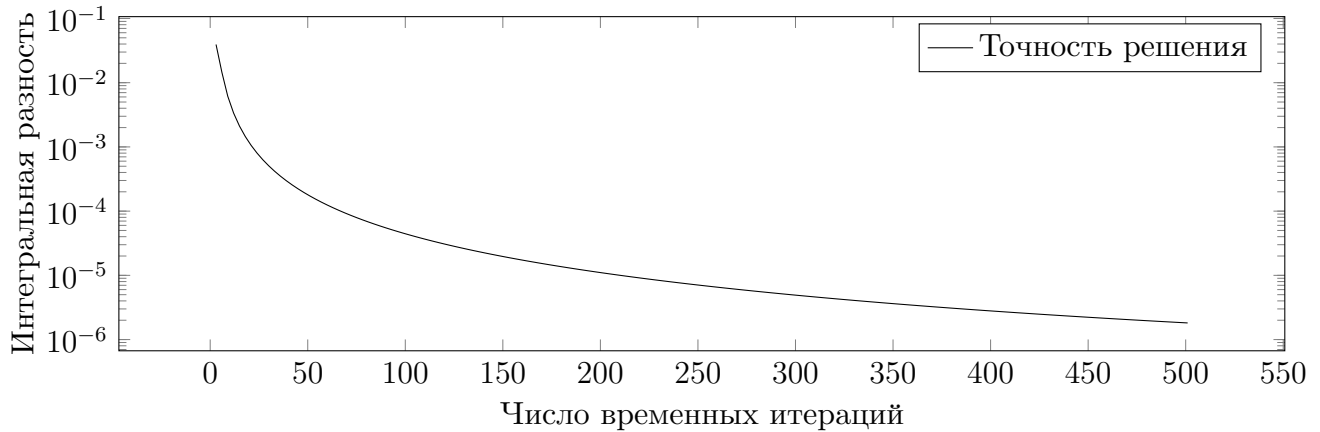
Вывод: согласно норме разности векторов, порядок сходимости ≈ 0.45 .



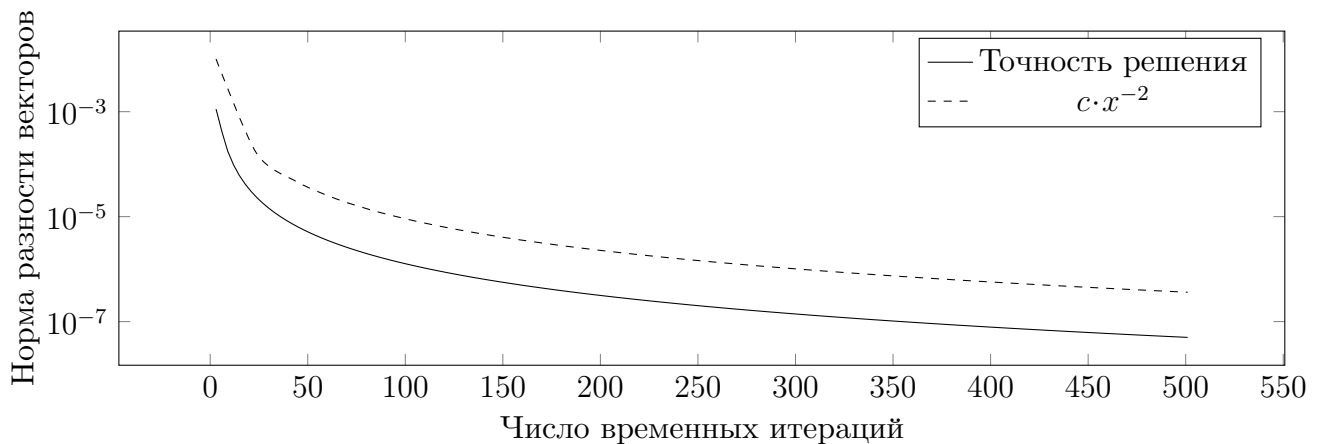
Вывод: время решения почти линейно зависит от числа узлов (с учетом погрешности, вносимой многопоточностью).

4.3.2 Увеличение размерности по времени

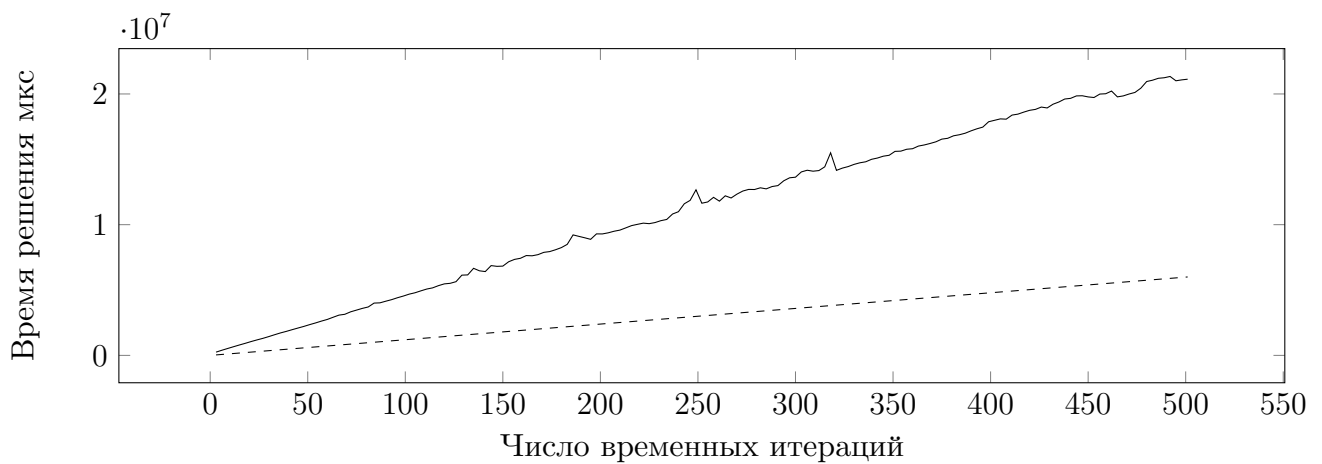
Сетка по пространству: $(x, y) \in [0, 1] \times [0, 1]$, строк и столбцов 20. Сетка по времени: $t \in [0, 1]$.



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 Аналитический расчёт формул

Для расчётов формул была использована библиотека `sympy` на `Python`, благодаря ей были вычислены матрицы G , C размера 9×9 ; и билинейные базисные функции.

В следующем файле с вычисляются эти функции и матрицы для билинейных базисов:

FILE `integrate_fem_2.py`

```

1 from sympy import *
2 import copy
3
4 x, y = symbols("x y")
5
6 x_p, h_x = symbols("x_p h_x")
7 x_p1 = x_p + h_x
8
9 y_p, h_y = symbols("y_p h_y")
10 y_p1 = y_p + h_y
11
12 X1 = (x_p1 - x)/h_x
13 X2 = (x - x_p)/h_x
14
15 Y1 = (y_p1 - y)/h_y
16 Y2 = (y - y_p)/h_y
17
18 f = []
19
20 for i in [Y1, Y2]:
21     for j in [X1, X2]:
22         f.append(j*i)
23
24 G1 = zeros(len(f))
25 G2 = zeros(len(f))
26
27 for i, a in enumerate(f):
28     for j, b in enumerate(f):
29         temp = simplify(
30             integrate(
31                 integrate(
32                     diff(a, x) * diff(b, x) + diff(a, y) * diff(b, y),
33                     (y, y_p, y_p + h_y)
34                 ),
35                 (x, x_p, x_p + h_x)
36             )
37         )
38         (first, second) = temp.args
39         (coef, up, down) = first.args
40         if up == h_x:
41             G1[i, j], G2[i, j] = first, second
42         else:
43             G1[i, j], G2[i, j] = second, first

```

```

44 M = zeros(len(f))
45 for i, a in enumerate(f):
46     for j, b in enumerate(f):
47         M[i, j] = simplify(
48             integrate(
49                 integrate(
50                     a * b,
51                     (y, y_p, y_p + h_y)
52                 ),
53                 (x, x_p, x_p + h_x)
54             )
55         )
56
57 first = h_x/(h_y*6)
58 second = h_y/(h_x*6)
59 G1 = simplify(G1 / first)
60 G2 = simplify(G2 / second)
61 l = Symbol("lambda")
62
63 print("G = {}\\cdot{} + {}\\cdot{}".format(latex(l * first), latex(G1), latex(l * second), latex(G2)))
64
65 multiplier = h_x*h_y/36
66 M = simplify(M / multiplier)
67 g = Symbol("gamma")
68
69 print("M = {}\\cdot{}".format(latex(g * multiplier), latex(M)))
70

```

А здесь вычисляются для биквадратичных базисов:

FILE integrate_fem_3.py

```

1 from sympy import *
2 import copy
3
4 def get_function(list, var, xy, h):
5     a, b, c = symbols("a b c")
6     x, y = symbols("x y")
7     a1 = solve([
8         a*xy*xy + b*xy + c - list[0],
9         a*(xy + h/2)*(xy + h/2) + b*(xy + h/2) + c - list[1],
10        a*(xy + h)*(xy + h) + b*(xy + h) + c - list[2]
11    ], [a, b, c])
12    return a1[a]*var*var + a1[b]*var + a1[c]
13
14 x, y = symbols("x y")
15
16 x_p, h_x = symbols("x_p h_x")
17 x_p1 = x_p + h_x
18
19 y_p, h_y = symbols("y_p h_y")
20 y_p1 = y_p + h_y
21
22 X1 = get_function([1, 0, 0], x, x_p, h_x)
23 X2 = get_function([0, 1, 0], x, x_p, h_x)
24 X3 = get_function([0, 0, 1], x, x_p, h_x)
25
26 Y1 = get_function([1, 0, 0], y, y_p, h_y)
27 Y2 = get_function([0, 1, 0], y, y_p, h_y)
28 Y3 = get_function([0, 0, 1], y, y_p, h_y)
29
30 print("X1: {}\\nX2: {}\\nX3: {}".format(X1, X2, X3))
31 print("Y1: {}\\nY2: {}\\nY3: {}".format(Y1, Y2, Y3))
32
33 print("latex:")
34 print("X1: {}\\nX2: {}\\nX3: {}".format(latex(X1), latex(X2), latex(X3)))
35 print("Y1: {}\\nY2: {}\\nY3: {}".format(latex(Y1), latex(Y2), latex(Y3)))
36
37 f = []
38
39 for i in [Y1, Y2, Y3]:
40     for j in [X1, X2, X3]:
41         f.append(j*i)
42
43 G1 = zeros(len(f))
44 G2 = zeros(len(f))
45
46 for i, a in enumerate(f):
47     for j, b in enumerate(f):
48         temp = simplify(
49             integrate(
50                 integrate(
51                     diff(a, x) * diff(b, x) + diff(a, y) * diff(b, y),
52                     (y, y_p, y_p + h_y)
53                 ),
54                 (x, x_p, x_p + h_x)

```

```

55     )
56     )
57     (first, second) = temp.args
58     (coef, up, down) = first.args
59     if up == h_x:
60         G1[i, j], G2[i, j] = first, second
61     else:
62         G1[i, j], G2[i, j] = second, first
63
64 M = zeros(len(f))
65 for i, a in enumerate(f):
66     for j, b in enumerate(f):
67         M[i, j] = simplify(
68             integrate(
69                 integrate(
70                     a * b,
71                     (y, y_p, y_p + h_y)
72                 ),
73                 (x, x_p, x_p + h_x)
74             )
75         )
76
77 first = h_x/(h_y*6*15)
78 second = h_y/(h_x*6*15)
79 G1 = simplify(G1 / first)
80 G2 = simplify(G2 / second)
81 l = Symbol("lambda")
82
83 print("G = {\}\cdot{} + {\}\cdot{}".format(latex(l * first), latex(G1), latex(l * second), latex(G2)))
84
85 multiplier = h_x*h_y/36/25
86 M = simplify(M / multiplier)
87 g = Symbol("gamma")
88
89 print("M = {\}\cdot{}".format(latex(g * multiplier), latex(M)))

```

Программу вполне можно обобщить для любых би- n -базисов.

5.2 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 завершённой 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 inline string write_for_latex_double(double v, int precision) {
42     int power = log(std::fabs(v)) / log(10.0);
43     double value = v / pow(10.0, power);
44
45     if (v != v) return "nan";
46
47     if (v == 0) {
48         power = 0;
49         value = 0;
50     }
51
52     stringstream sout;
53     sout.precision(precision);
54     if (power == -1 || power == 0 || power == 1) {
55         sout << v;
56     } else {
57         sout << value << "\\cdot 10^{ " << power << " }";
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 {

```

```

54 public:
55     matrix_sparse_ra_t(int n);
56
57     /** Установить значение в позиции (i, j) */
58     double& operator()(int i, int j);
59
60     /** Получить значение в позиции (i, j). Если туда ещё не устанавливалось значение, вызывается
61     ↪ исключение. */
62     const double& operator()(int i, int j) const;
63
64     /** Преобразует текущую матрицу к разреженной матрице. */
65     matrix_sparse_t to_sparse(void) const;
66 private:
67     int n;
68     vector<double> dm;
69     vector<map<int, double>> lm, um;
70 };
71
72 //-----
73 /** Функции для умножения векторов. */
74 void mul(const vector_t& d, vector_t& x_y); // x_y = d * x_y
75 void mul_inv(const vector_t& d, vector_t& x_y); // x_y = x_y / d
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: биквадратичные. 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     bool is_part_of_elem; // Является ли данный узел началом конечного элемента
19
20     /** Проверяет, является ли элемент граничным. Он таким является, если у него нет хотя бы одного
21     ↪ соседа. */
22     bool is_boundary(void) const;
23 };
24
25 /** Прямоугольный конечный элемент на основе билинейных базисных функций. Образуется из четырех узлов.
26 ↪ */
27 struct elem_t
28 {
29     int i; /// Номер конечного элемента
30     basic_elem_t* e[9]; /// Указатели на все 9 элементов конечного узла, нумерация такая:
31     /**
32     Y
33     |
34     | 7----8----9
35     | |   |   |
36     | 4----5----6
37     | |   |   |
38     | 1----2----3
39     |
40     +-----> X
41     */
42     double get_hx(void) const; /// Ширина конечного элемента
43     double get_hy(void) const; /// Высота конечного элемента
44
45     /** Рассчитать значение внутри конечного элемента. q - вектор рассчитываемых весов. */

```

```

46     double value(double x, double y, const vector_t& q) const;
47 };
48
49 /** Все константы решаемого уравнения. */
50 struct constants_t
51 {
52     double lambda; /// Коэффициент внутри div
53     double gamma;   /// Коэффициент при u
54     double sigma;   /// Коэффициент при du/dt
55     double chi;     /// Коэффициент при d^2u/dt^2
56 };
57
58 //-----
59 /** 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. */
60 double non_linear_offset(double x, double t);
61
62 /** 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. */
63 class grid_generator_t
64 {
65 public:
66     grid_generator_t(double a, double b, int n, double t = 0);
67     double operator()(int i) const;
68     int size(void) const;
69     double back(void) const;
70 private:
71     double a, len, t, n1;
72 };
73
74 /** Класс двумерной неравномерной сетки по пространству в виде прямоугольника. */
75 class grid_t
76 {
77 public:
78     vector<elem_t> es; /// Массив конечных элементов сетки
79     vector<basic_elem_t> bes; /// Массив узлов сетки
80     int n; /// Число узлов
81
82     /** Рассчитать неравномерную сетку. */
83     void calc(const grid_generator_t& gx, const grid_generator_t& gy);
84 };
85
86 /** Рассчитать веса идеальной аппроксимации функции u при помощи узлов bes. */
87 vector_t calc_true_approx(const function_2d_t& u, const vector<basic_elem_t>& bes);
88
89 /** Рассчитать интегральную норму между конечно-элементной аппроксимацией и истинной функцией. */
90 double calc_integral_norm(const function_2d_t& u, const vector<elem_t>& es, const vector_t& q);
91
92 //-----
93 /** Расчет локальных матриц для конечного элемента. */
94 matrix_t calc_local_matrix_g(const elem_t& e, const constants_t& cs);
95 matrix_t calc_local_matrix_c(const elem_t& e);
96 vector_t calc_local_vector_b(const elem_t& e, const function_2d_t& f);
97
98 //-----
99 /** Рассчитать глобальный вектор из локальных векторов для всех конечных элементов. */
100 vector_t calc_global_vector(
101     const vector<elem_t>& es,
102     const function<vector_t(const elem_t)> calc_local_vector,
103     int n
104 );
105
106 /** Рассчитать глобальную матрицу из функции построения локальных матриц. */
107 matrix_sparse_t calc_global_matrix(
108     const vector<elem_t>& es,
109     const function<matrix_t(const elem_t)> calc_local_matrix,
110     int n
111 );
112
113 //-----
114 /** Численный расчет определенных интегралов. */
115 double calc_integral_gauss3(
116     double a, double b, int n, // n - количество внутренних узлов
117     const function_1d_t& f
118 );
119 double calc_integral_gauss3(
120     double ax, double bx, int nx,
121     double ay, double by, int ny,
122     const function_2d_t& f
123 );
124
125 //-----
126 /** Численный расчет производной. */
127 function_1d_t calc_first_derivative(const function_1d_t& f);
128 function_1d_t calc_second_derivative(const function_1d_t& f);
129
130 //-----

```

```

131 /** Для гиперболического дифференциального уравнения и функции u считает каким должно быть f, чтобы
132     ↪ решением этого диф. уравнения была функции u. Делает это численно. */
133 function_3d_t calc_right_part(const function_3d_t& u, const constants_t& cs);
134
135 /**-----
136 ** Использует схему Кранка-Николсона для получения СЛАУ. Предполагается, что разреженные матрицы
137     ↪ имеют одинаковый формат. */
138 void calc_crank_nicolson_method(
139     const matrix_sparse_t& c,
140     const matrix_sparse_t& g,
141     const vector_t& b0, // b current (b_j)
142     const vector_t& b1, // b last (b_{j-1})
143     const vector_t& b11, // b last last (b_{j-2})
144     const vector_t& q1, // q last (q_{j-1})
145     const vector_t& q11, // q last last (q_{j-2})
146     const constants_t& cs,
147     const grid_generator_t& time_grid,
148     int time_i,
149     matrix_sparse_t& a,
150     vector_t& b
151 );
152
153 /**-----
154 ** Функция, которая устанавливает краевые условия для задачи в СЛАУ. Сделана для того, чтобы не
155     ↪ посылать в функцию решения МКЭ истинную функцию, а чтобы посылать красивую оболочку, которую
156     ↪ потенциально можно использовать в реальных задачах. */
157 typedef function<void(matrix_sparse_t&, vector_t&, const vector<basic_elem_t>&, double)>
158     ↪ boundary_setter_t;
159
160 /** Записывает первые краевые условия в матрицу a и вектор b. Для этой записи ему необходимо получить
161     ↪ истинную функцию. */
162 void write_first_boundary_conditions(
163     matrix_sparse_t& a,
164     vector_t& b,
165     const vector<basic_elem_t>& bes,
166     double t,
167     const function_3d_t& u
168 );
169
170 /**-----
171 ** Решает при помощи МКЭ дифференциальное уравнение с функцией правой части f, заданными константами,
172     ↪ прямоугольной сеткой grid и функцией выставления краевых условий. Использует схему
173     ↪ Кранка-Николсона для аппроксимации по времени, и ЛОС в разреженной строчно-столбцовой матрице для
174     ↪ решения СЛАУ. */
175 vector<vector_t> solve_differential_equation(
176     const function_3d_t& f,
177     const boundary_setter_t& set_boundary_conditions,
178     const vector_t& q0,
179     const vector_t& q1,
180     const constants_t& cs,
181     const grid_t& grid,
182     const grid_generator_t& time_grid
183 );

```

5.3 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             lu.l[j] = (a.l[j] - sum) / lu.d[row];
79         }
80
81         // Заполняем верхний треугольник
82         int row_start = lu.line_elem_start(i);
83         int row_end = lu.line_elem_start(i+1);
84         for (int j = line_start; j < line_end; ++j) {
85             double sum = 0;
86
87             int line = lu.j[j];
88             int line_start = lu.line_elem_start(line);
89             int line_end = lu.line_elem_start(line+1);
90
91             int kl = line_start;
92             int ku = row_start;
93
94             while (kl < line_end && ku < j) {
95                 if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
96                     sum += lu.l[kl] * lu.u[ku];
97                     ku++;
98                     kl++;
99                 } else if (lu.j[kl] < lu.j[ku]) {
100                     kl++;
101                 } else {
102                     ku++;
103                 }
104             }
105             lu.u[j] = (a.u[j] - sum) / lu.d[line];
106         }
107
108         // Расчитываем диагональный элемент
109         double sum = 0;
110         int line_row_start = lu.line_elem_start(i);
111         int line_row_end = lu.line_elem_start(i+1);
112         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 //-----
308 void mul(const vector_t& d, vector_t& x_y) {
309     for (int i = 0; i < d.size(); i++)
310         x_y[i] *= d[i];
311 }
312
313 //-----
314 void mul_inv(const vector_t& d, vector_t& x_y) {
315     for (int i = 0; i < d.size(); i++)
316         x_y[i] /= d[i];
317 }
318
319 //-----
320 //-----
321 //-----
322
323 //-----
324 vector_t solve_by_los_lu(
325     const matrix_sparse_t& a,
326     const vector_t& b,
327     int maxiter,
328     double eps,
329     bool is_log
330 ) {
331     matrix_sparse_t lu(a.n);
332     vector_t r, z, p;
333     vector_t x, t1, t2;
334
335     int n = a.n;
336
337     a.decompose_lu_partial(lu);
338     x = vector_t(n); x.fill(0);
339
340     r = x;
341     a.mul(r);
342     for (int i = 0; i < n; i++)
343         r[i] = b[i] - r[i];
344     lu.mul_l_inv(r);
345
346     z = r;
347     lu.mul_u_inv(z);
348
349     p = z;
350     a.mul(p);
351     lu.mul_l_inv(p);
352
353     double flen = sqrt(b.dot(b));
354     double residual;
355
356     int i = 0;
357     while (true) {
358         double pp = p.dot(p);
359         double alpha = (p.dot(r)) / pp;
360         for (int i = 0; i < n; ++i) {
361             x[i] += alpha * z[i];
362             r[i] -= alpha * p[i];
363         }
364         t1 = r;
365         lu.mul_u_inv(t1);
366         t2 = t1;
367         a.mul(t2);
368         lu.mul_l_inv(t2);
369         double beta = -(p.dot(t2)) / pp;
370         for (int i = 0; i < n; ++i) {
371             z[i] = t1[i] + beta * z[i];
372             p[i] = t2[i] + beta * p[i];
373         }
374         residual = r.norm() / flen;
375         i++;
376
377         //if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
378         ↪ setprecision(16) << residual << endl;
379         if (fabs(residual) < eps || i > maxiter)
380             break;

```



```

380 }
381
382 return x;
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[2]->x - e[0]->x;
15 }
16
17 //-----
18 double elem_t::get_hy(void) const {
19     return e[6]->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 yp = e[0]->y;
26     double hx = get_hx();
27     double hy = get_hy();
28     auto x1 = [&](double x) -> double { return 1.0 + 3.0*xp/hx + 2.0*(x*x)/(hx*hx) - x*(3.0*hx +
29     ↪ 4.0*xp)/(hx*hx) + 2.0*(xp*xp)/(hx*hx); };
30     auto x2 = [&](double x) -> double { return -4.0*(x*x)/(hx*hx) + 4.0*x*(hx + 2.0*xp)/(hx*hx) -
31     ↪ 4.0*xp*(hx + xp)/(hx*hx); };
32     auto x3 = [&](double x) -> double { return 2.0*(x*x)/(hx*hx) - x*(hx + 4.0*xp)/(hx*hx) + xp*(hx +
33     ↪ 2.0*xp)/(hx*hx); };
34     auto y1 = [&](double y) -> double { return 1.0 + 3.0*yp/hy + 2.0*(y*y)/(hy*hy) - y*(3.0*hy +
35     ↪ 4.0*yp)/(hy*hy) + 2.0*(yp*yp)/(hy*hy); };
36     auto y2 = [&](double y) -> double { return -4.0*(y*y)/(hy*hy) + 4.0*y*(hy + 2.0*yp)/(hy*hy) -
37     ↪ 4.0*yp*(hy + yp)/(hy*hy); };
38     auto y3 = [&](double y) -> double { return 2.0*(y*y)/(hy*hy) - y*(hy + 4.0*yp)/(hy*hy) + yp*(hy +
39     ↪ 2.0*yp)/(hy*hy); };
40
41     auto psi1 = [&]() -> double { return x1(x) * y1(y); };
42     auto psi2 = [&]() -> double { return x2(x) * y1(y); };
43     auto psi3 = [&]() -> double { return x3(x) * y1(y); };
44     auto psi4 = [&]() -> double { return x1(x) * y2(y); };
45     auto psi5 = [&]() -> double { return x2(x) * y2(y); };
46     auto psi6 = [&]() -> double { return x3(x) * y2(y); };
47     auto psi7 = [&]() -> double { return x1(x) * y3(y); };
48     auto psi8 = [&]() -> double { return x2(x) * y3(y); };
49     auto psi9 = [&]() -> double { return x3(x) * y3(y); };
50
51     double v1 = psi1() * q[e[0]->i];
52     double v2 = psi2() * q[e[1]->i];
53     double v3 = psi3() * q[e[2]->i];
54     double v4 = psi4() * q[e[3]->i];
55     double v5 = psi5() * q[e[4]->i];
56     double v6 = psi6() * q[e[5]->i];
57     double v7 = psi7() * q[e[6]->i];
58     double v8 = psi8() * q[e[7]->i];
59     double v9 = psi9() * q[e[8]->i];
60
61     return v1 + v2 + v3 + v4 + v5 + v6 + v7 + v8 + v9;
62 }
63
64 //-----
65 //-----
66 //-----
67 //-----
68 double non_linear_offset(double x, double t) {
69     int sign = (t > 0) ? 1 : -1;
70     t *= sign;
71     t = 1.0 - t;
72     t = (sign == -1) ? 1.0/t : t;
73     if (t == 1.0) return x;
74     return (1.0 - pow(t, x))/(1.0 - t);
75 }
76
77 //-----

```

```

73 grid_generator_t::grid_generator_t(double a, double b, int n, double t) : a(a), len(b-a), n1(2 *
74   ↳ n+1.0), t(t) {}
75
76 //-----
77 double grid_generator_t::operator()(int i) const {
78     return a + len * non_linear_offset(i/n1, t);
79 }
80
81 //-----
82 int grid_generator_t::size(void) const {
83     return n1;
84 }
85
86 //-----
87 double grid_generator_t::back(void) const {
88     return operator()(size()-1);
89 }
90
91 //-----
92 void grid_t::calc(const grid_generator_t& gx, const grid_generator_t& gy) {
93     bes.clear();
94     bes.resize(gx.size() * gy.size());
95     int counter = 0;
96     for (int j = 0; j < gy.size(); ++j) {
97         double y = gy(j);
98         for (int i = 0; i < gx.size(); ++i) {
99             double x = gx(i);
100             basic_elem_t* down = (counter >= gx.size()) ? &bes[counter-gx.size()] : nullptr;
101             basic_elem_t* left = (counter % gx.size() > 0) ? &bes[counter-1] : nullptr;
102             bes[counter] = {counter, x, y,
103                 nullptr,
104                 down,
105                 left,
106                 nullptr,
107                 false
108             };
109             if (down != nullptr) down->up = &bes[counter];
110             if (left != nullptr) left->right = &bes[counter];
111             counter++;
112         }
113     }
114     n = bes.size();
115
116     es.clear();
117     counter = 0;
118     for (auto& i : bes) {
119         bool is_has_next_elems =
120             i.right != nullptr &&
121             i.right->right != nullptr &&
122             i.up != nullptr &&
123             i.up->up != nullptr &&
124             i.up->up->right->right != nullptr;
125         bool is_not_part_of_elem = i.is_part_of_elem != true;
126         if (is_has_next_elems && is_not_part_of_elem) {
127             es.push_back({counter,
128                 i.right->left,
129                 i.right,
130                 i.right->right,
131                 i.up,
132                 i.up->right,
133                 i.up->right->right,
134                 i.up->up,
135                 i.up->up->right,
136                 i.up->up->right->right,
137             });
138             i.right->left->is_part_of_elem = true;
139             i.right->is_part_of_elem = true;
140             //i.right->right->is_part_of_elem = true;
141             i.up->is_part_of_elem = true;
142             i.up->right->is_part_of_elem = true;
143             i.up->right->right->is_part_of_elem = true;
144             //i.up->up->is_part_of_elem = true;
145             i.up->up->right->is_part_of_elem = true;
146             //i.up->up->right->right->is_part_of_elem = true;
147             counter++;
148         }
149     }
150 }
151
152 //-----
153 vector_t calc_true_approx(const function_2d_t& u, const vector<basic_elem_t>& bes) {
154     vector_t result(bes.size());
155     for (int i = 0; i < bes.size(); ++i)
156         result(i) = u(bes[i].x, bes[i].y);
157     return result;
158 }
159
160 //-----

```

```

161 double calc_integral_norm(const function_2d_t& u, const vector<elem_t>& es, const vector_t& q) {
162     double sum = 0;
163     for (auto& i : es) {
164         sum += calc_integral_gauss3(
165             i.e[0]->x, i.e[2]->x, 10,
166             i.e[0]->y, i.e[6]->y, 10,
167             [&](double x, double y) -> double {
168                 return fabs(u(x, y) - i.value(x, y, q));
169             }
170         );
171     }
172     return sum;
173 }
174
175 //-----
176 //-----
177 //-----
178
179 //-----
180 matrix_t calc_local_matrix_g(
181     const elem_t& e,
182     const constants_t& cs
183 ) {
184     double hx = e.get_hx();
185     double hy = e.get_hy();
186     matrix_t result;
187     matrix_t a(9, 9), b(9, 9);
188     a <<
189         28, -32, 4, 14, -16, 2, -7, 8, -1,
190         -32, 64, -32, -16, 32, -16, 8, -16, 8,
191         4, -32, 28, 2, -16, 14, -1, 8, -7,
192         14, -16, 2, 112, -128, 16, 14, -16, 2,
193         -16, 32, -16, -128, 256, -128, -16, 32, -16,
194         2, -16, 14, 16, -128, 112, 2, -16, 14,
195         -7, 8, -1, 14, -16, 2, 28, -32, 4,
196         8, -16, 8, -16, 32, -16, -32, 64, -32,
197         -1, 8, -7, 2, -16, 14, 4, -32, 28;
198     b <<
199         28, 14, -7, -32, -16, 8, 4, 2, -1,
200         14, 112, 14, -16, -128, -16, 2, 16, 2,
201         -7, 14, 28, 8, -16, -32, -1, 2, 4,
202         -32, -16, 8, 64, 32, -16, -32, -16, 8,
203         -16, -128, -16, 32, 256, 32, -16, -128, -16,
204         8, -16, -32, -16, 32, 64, 8, -16, -32,
205         4, 2, -1, -32, -16, 8, 28, 14, -7,
206         2, 16, 2, -16, -128, -16, 14, 112, 14,
207         -1, 2, 4, 8, -16, -32, -7, 14, 28;
208     result = cs.lambda/90.0*(hy/hx*a + hx/hy*b);
209     return result;
210 }
211
212 //-----
213 matrix_t calc_local_matrix_c(
214     const elem_t& e
215 ) {
216     double hx = e.get_hx();
217     double hy = e.get_hy();
218     matrix_t result;
219     matrix_t c(9, 9);
220     c <<
221         16, 8, -4, 8, 4, -2, -4, -2, 1,
222         8, 64, 8, 4, 32, 4, -2, -16, -2,
223         -4, 8, 16, -2, 4, 8, 1, -2, -4,
224         8, 4, -2, 64, 32, -16, 8, 4, -2,
225         4, 32, 4, 32, 256, 32, 4, 32, 4,
226         -2, 4, 8, -16, 32, 64, -2, 4, 8,
227         -4, -2, 1, 8, 4, -2, 16, 8, -4,
228         -2, -16, -2, 4, 32, 4, 8, 64, 8,
229         1, -2, -4, -2, 4, 8, -4, 8, 16;
230     result = hx*hy/900.0*c;
231     return result;
232 }
233
234 //-----
235 vector_t calc_local_vector_b(
236     const elem_t& e,
237     const function_2d_t& f
238 ) {
239     vector_t fv(9);
240     fv <<
241         f(e.e[0]->x, e.e[0]->y),
242         f(e.e[1]->x, e.e[1]->y),
243         f(e.e[2]->x, e.e[2]->y),
244
245         f(e.e[3]->x, e.e[3]->y),
246         f(e.e[4]->x, e.e[4]->y),
247         f(e.e[5]->x, e.e[5]->y),
248
249         f(e.e[6]->x, e.e[6]->y),

```

```

250         f(e.e[7]->x, e.e[7]->y),
251         f(e.e[8]->x, e.e[8]->y);
252     return calc_local_matrix_c(e) * fv;
253 }
254
255 //-----
256 vector_t calc_global_vector(
257     const vector<elem_t>& es,
258     const function<vector_t(const elem_t)> calc_local_vector,
259     int n
260 ) {
261     vector_t result(n);
262     result.fill(0);
263     for (auto& e : es) {
264         auto b = calc_local_vector(e);
265         for (int i = 0; i < 9; ++i) {
266             result(e.e[i]->i) += b(i);
267         }
268     }
269     return result;
270 }
271
272 //-----
273 matrix_sparse_t calc_global_matrix(
274     const vector<elem_t>& es,
275     const function<matrix_t(const elem_t)> calc_local_matrix,
276     int n
277 ) {
278     matrix_sparse_ra_t result(n);
279     for (auto& e : es) {
280         auto m = calc_local_matrix(e);
281         for (int i = 0; i < 9; ++i) {
282             for (int j = 0; j < 9; ++j) {
283                 result(e.e[i]->i, e.e[j]->j) += m(i, j);
284             }
285         }
286     }
287     return result.to_sparse();
288 }
289
290 //-----
291 //-----
292 //-----
293
294 //-----
295 double calc_integral_gauss3(
296     double a, double b, int n, // n - количество внутренних узлов
297     const function_1d_t& f
298 ) {
299     const double x1 = -sqrt(3.0/5.0);
300     const double x2 = 0;
301     const double x3 = -x1;
302     const double q1 = 5.0/9.0;
303     const double q2 = 8.0/9.0;
304     const double q3 = q1;
305     double sum = 0;
306     double xk = 0;
307     double h = (b-a)/double(n+1);
308     double h2 = h/2.0;
309
310     for (int i = 0; i < n+1; ++i) {
311         xk = a + h*i + h2;
312         sum += q1 * f(xk + x1 * h2);
313         sum += q2 * f(xk + x2 * h2);
314         sum += q3 * f(xk + x3 * h2);
315     }
316
317     sum *= h;
318     sum /= 2.0;
319     return sum;
320 }
321
322 //-----
323 double calc_integral_gauss3(
324     double ax, double bx, int nx,
325     double ay, double by, int ny,
326     const function_2d_t& f
327 ) {
328     return calc_integral_gauss3(ax, bx, nx, [ay, by, ny, f](double x)->double {
329         return calc_integral_gauss3(ay, by, ny, bind(f, x, _1));
330     });
331 }
332
333 //-----
334 //-----
335 //-----
336
337 //-----
338 function_1d_t calc_first_derivative(const function_1d_t& f) {

```

```

339     return [f](double x) -> double {
340         const double h = 0.001;
341         return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) / (12 * h);
342     };
343 }
344
345 //-----
346 function_1d_t calc_second_derivative(const function_1d_t& f) {
347     return [f](double x) -> double {
348         const double h = 0.001;
349         return (-f(x+2*h) + 16*f(x+h) - 30*f(x) + 16*f(x-h) - f(x-2*h))/(12*h*h);
350     };
351 }
352
353 //-----
354 //-----
355 //-----
356
357 //-----
358 function_3d_t calc_right_part(
359     const function_3d_t& u,
360     const constants_t& cs
361 ) {
362     // f = -div(lambda * grad u) + gamma * u + sigma * du/dt + chi * d^2 u/dt^2
363     return [=](double x, double y, double t) -> double {
364         using namespace placeholders;
365         auto ut = calc_first_derivative(bind(u, x, y, _1));
366
367         auto uxx = calc_second_derivative(bind(u, _1, y, t));
368         auto uyy = calc_second_derivative(bind(u, x, _1, t));
369         auto utt = calc_second_derivative(bind(u, x, y, _1));
370
371         return -cs.lambda * (uxx(x) + uyy(y)) + cs.gamma * u(x, y, t) + cs.sigma * ut(t) + cs.chi *
372             ↪ utt(t);
373     };
374 }
375
376 //-----
377 //-----
378 //-----
379
380 void calc_crank_nicolson_method(
381     const matrix_sparse_t& c,
382     const matrix_sparse_t& g,
383     const vector_t& b0,
384     const vector_t& b1,
385     const vector_t& bll,
386     const vector_t& ql,
387     const vector_t& qll,
388     const constants_t& cs,
389     const grid_generator_t& time_grid,
390     int time_i,
391     matrix_sparse_t& a,
392     vector_t& b
393 ) {
394     // Схема Кранка-Николсона
395
396     // Константы для вычислений с неравномерной сеткой по времени
397     double t0 = time_grid(time_i);
398     double t1 = time_grid(time_i-1);
399     double t2 = time_grid(time_i-2);
400
401     double d1 = t0-t2;
402     double d2 = (t0*(t0-t2-t1)+t2*t1)/2.0;
403     double m1 = (t0-t2)/(t1-t2);
404     double m2 = (t0-t1)/(t1-t2);
405
406     // Вычисляем матрицу a
407     a = c;
408     double c1 = cs.gamma/2.0 + cs.sigma/d1 + cs.chi/d2;
409     for (int i = 0; i < a.d.size(); i++)
410         a.d[i] = g.d[i]/2.0 + c.d[i]*c1;
411     for (int i = 0; i < a.l.size(); i++) {
412         a.l[i] = g.l[i]/2.0 + c.l[i]*c1;
413         a.u[i] = g.u[i]/2.0 + c.u[i]*c1;
414     }
415
416     // Рассчитываем вектор b
417     b = (b0 + bll)/2.0;
418     vector_t temp = qll;
419     g.mul(temp);
420     b = b - temp/2.0;
421
422     temp = ql*(m1*cs.chi/d2) + qll*(-cs.gamma/2.0 + cs.sigma/d1 - m2*cs.chi/d2);
423     c.mul(temp);
424
425     b = b + temp;
426 }

```

```

427 //-----
428
429 void write_first_boundary_conditions(
430     matrix_sparse_t& a,
431     vector_t& b,
432     const vector<basic_elem_t>& bes,
433     double t,
434     const function_3d_t& u
435 ) {
436     for (int i = 0; i < bes.size(); ++i) {
437         if (bes[i].is_boundary()) {
438             a.clear_line(bes[i].i);
439             a.d[bes[i].i] = 1;
440             b(bes[i].i) = u(bes[i].x, bes[i].y, t);
441         }
442     }
443 }
444
445 //-----
446 //-----
447 //-----
448 //-----
449 //-----
450 //-----
451 //-----
452 //-----
453 //-----
454 vector<vector_t> solve_differential_equation(
455     const function_3d_t& f,
456     const boundary_setter_t& set_boundary_conditions,
457     const vector_t& q0,
458     const vector_t& q1,
459     const constants_t& cs,
460     const grid_t& grid,
461     const grid_generator_t& time_grid
462 ) {
463     auto c = calc_global_matrix(grid.es, calc_local_matrix_c, grid.n);
464     auto g = calc_global_matrix(grid.es, bind(calc_local_matrix_g, _1, cs), grid.n);
465
466     auto calc_global_vector_b = [&] (int i) {
467         return calc_global_vector(
468             grid.es,
469             bind(
470                 calc_local_vector_b,
471                 _1,
472                 function_2d_t(bind(f, _1, _2, time_grid(i)))
473             ),
474             grid.n
475         );
476     };
477
478     vector_t bll = calc_global_vector_b(0);
479     vector_t bl = calc_global_vector_b(1);
480     vector_t b0;
481
482     vector_t qll = q0;
483     vector_t ql = q1;
484     vector_t q;
485
486     vector<vector_t> result;
487     result.push_back(q0);
488     result.push_back(q1);
489
490     matrix_sparse_t a(grid.n);
491     vector_t b(grid.n);
492     for (int i = 2; i < time_grid.size(); ++i) {
493         b0 = calc_global_vector_b(i);
494         calc_crank_nicolson_method(c, g, b0, bl, bll, ql, qll, cs, time_grid, i, a, b);
495         set_boundary_conditions(a, b, grid.bes, time_grid(i));
496
497         q = solve_by_los_lu(a, b, 1000, 1e-16, false);
498
499         result.push_back(q);
500
501         bll = bl;
502         bl = b0;
503
504         qll = ql;
505         ql = q;
506     }
507
508     return result;
509 }

```

5.4 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\tnorm\tuniform_integral\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
156     performer.finish();
157
158     auto integral_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto&
159     ↪ a) -> double { return a.second.integral_residual; });
160     auto norm_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto& a)
161     ↪ -> double { return a.second.norm_residual; });
162
163     ofstream fout(filename + ".integral.txt");
164     ofstream fout2(filename + ".norm.txt");
165     ofstream fout3(filename + ".time.txt");
166     int last_line = 0;
167     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; },
199         ↪ "$x^2y+y^3$"});
200     spaces.push_back({[] (double x, double y, double t) -> double { return x*y*y; }, "$xy^2$"});
201     spaces.push_back({[] (double x, double y, double t) -> double { return x*x*x+y*y*y*y; },
202         ↪ "$x^4+y^4$"});
203     spaces.push_back({[] (double x, double y, double t) -> double { return exp(x*y); }, "$e^{xy}$"});
204
205     times.push_back({[] (double x, double y, double t) -> double { return 0; }, "$0$"});
206     times.push_back({[] (double x, double y, double t) -> double { return t; }, "$t$"});
207     times.push_back({[] (double x, double y, double t) -> double { return t*t; }, "$t^2$"});
208     times.push_back({[] (double x, double y, double t) -> double { return t*t*t; }, "$t^3$"});
209     times.push_back({[] (double x, double y, double t) -> double { return t*t*t*t; }, "$t^4$"});
210     times.push_back({[] (double x, double y, double t) -> double { return exp(t); }, "$e^t$"});
211
212     async_performer_t<fem_result_t, pair<string, string>> performer;
213
214     for (auto& i : spaces) {
215         for (auto& j : times) {
216             performer.add([i, j, &f]() -> fem_result_t {
217                 return f(function_3d_t([&] (double x, double y, double t) -> double { return
218                     ↪ i.first(x, y, t) + j.first(x, y, t); }));
219             }, {i.second, j.second});
220         }
221     }
222     performer.finish();
223
224     ofstream fout(filename);
225     fout << "a\t";
226     for (auto& i : times)
227         fout << i.second << "\t";
228     for (auto& i : spaces) {
229         fout << endl << i.second << "\t";
230         for (auto& j : times) {
231             auto v = performer[{i.second, j.second}];
232             fout << "\\scalebox{.75}{\\tcell{$" << write_for_latex_double(v.integral_residual, 2) <<
233                 ↪ "$\\\\\\$" << write_for_latex_double(v.norm_residual, 2) << "$\\\\\\$" << int(v.time/1000)
234                 ↪ << "$"}\\t$";
235         }
236     }
237     fout.close();
238 }
239
240 //-----
241 void investigate_grid_changing(
242     const string& filename,
243     const function<pair<fem_result_t, int>>(int)>& fi,
244     int n
245 ) {
246     async_performer_t<pair<fem_result_t, int>, int> performer;
247
248     for (int i = 0; i < n; i+=3) {
249         performer.add([i, fi] () -> pair<fem_result_t, int> {
250             return fi(i);
251         });
252     }
253 }

```

```

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;
320     u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x + y*y + t*t;
321         ↪ }, 0});
322     u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x*x*x + y*y*y*y*x
323         ↪ + t*t*t*t; }, 1});
324     u_space_mas.push_back({[] (double x, double y, double t) -> double { return exp(x*y) +
325         ↪ exp(t*t); }, 2});
326     u_space_mas.push_back({[] (double x, double y, double t) -> double { return exp((1-x)*(1-y)) +
327         ↪ exp((1-t)*(1-t)); }, 3});
328     u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x*x +
329         ↪ y*y*y*y*x*x*t + t*t*exp(t); }, 4});
330
331     for (auto& i : u_space_mas) {
332         auto& u = i.first;
333         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,
331                 ↪ 10), grid_generator_t(0, 1, 10, tt));
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,
339                 ↪ 10, ty), grid_generator_t(0, 1, 10));
340         }
341     );
342 }/1000/1000 << "s" << endl;
343 system("pause");
344 }

```

5.5 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",
37         ↪ f"space_tgrid_{i}_integral.png", r"Integral of functions difference");
38     make_image(f"space_tgrid_{i}.x.txt", f"space_tgrid_{i}.y.txt", f"space_tgrid_{i}.norm.txt",
39         ↪ f"space_tgrid_{i}_norm.png", r"Norm of $q$ vectors difference");
40     make_image(f"space_tgrid_{i}.x.txt", f"space_tgrid_{i}.y.txt", f"space_tgrid_{i}.time.txt",
41         ↪ f"space_tgrid_{i}_time.png", r"Solving time");
42
43 if __name__ == '__main__':
44     plt.rc('text', usetex=True)
45     plt.rc('font', family='serif')
46
47     make_images(0)
48     make_images(1)
49     make_images(2)
50     make_images(3)
51     make_images(4)

```