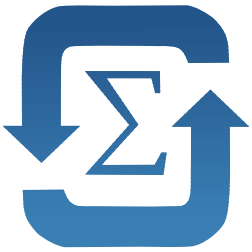
Министерство науки и высшего образования Российской Федерации

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:

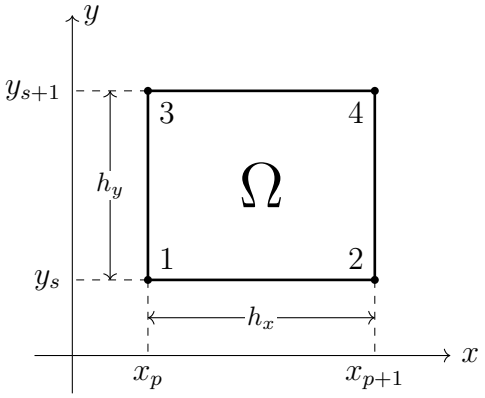
$$-\frac{\partial}{\partial x}\left(\lambda \frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(\lambda \frac{\partial u}{\partial y}\right) + \gamma u + \sigma \frac{\partial u}{\partial t} + \chi \frac{\partial^2 u}{\partial t^2} = f$$

Первые краевые условия:

$$u|_S = u_s$$

Формулы для билинейных базисных функций прямоугольных элементов:

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



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

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

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

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

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:

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

$$M_{ij}^{\gamma} = \int\limits_{x_p}^{x_{p+1}} \int\limits_{y_s}^{y_{s+1}} \gamma \psi_i \psi_j \,\mathrm{d}x\,\mathrm{d}y, \quad b_i = \int\limits_{x_p}^{x_{p+1}} \int\limits_{y_s}^{y_{s+1}} f \psi_i \,\mathrm{d}x\,\mathrm{d}y$$

Вычисленные матрицы для билинейных прямоугольных элементов:

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

$$\mathbf{C} = \frac{h_x h_y}{36}\begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix} \qquad \begin{gathered} \mathbf{M}^{\gamma} = \bar{\gamma}\mathbf{C} \\ \mathbf{f} = (f_1, f_2, f_3, f_4)^t \\ \mathbf{b} = \mathbf{C}\cdot\mathbf{f} \end{gathered}$$

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:

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

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:

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

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

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

Эти выражения были упрощены при помощи замен:

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

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:

```
class grid_generator_t
{
public:
    grid_generator_t(double a, double b, int n, double t = 0);
    double operator()(int i) const;
    int size(void) const;
    double back(void) const;
private:
    double a, len, t, n1;
};
```

Для задания узла конечного элемента структура:

```
struct basic_elem_t
{
    int i; /// Номер узла

    double x, y; /// Координаты узла

    basic_elem_t *up, *down, *left, *right; /// Указатели на соседей узла

    /** Проверяет, является ли элемент граничным. Он таким является, если у него нет хотя бы одного
    ↪  соседа. */
    bool is_boundary(void) const;
};
```

Для задания конечного элемента используется структура:

```
struct elem_t
{
    int i; /// Номер конечного элемента
    basic_elem_t* e[4]; /// Указатели на все 4 элемента конечного узла, нумерация такая:
    /**
         Y
         ^  3 +-----+ 4
         |    |     |
         |  1 +-----+ 2
        -+--------------> X
         |
     */

    double get_hx(void) const; /// Ширина конечного элемента
    double get_hy(void) const; /// Высота конечного элемента

    /** Рассчитать значение внутри конечного элемента. q - вектор рассчитыванных весов. */
    double value(double x, double y, const vector_t& q) const;
};
```

Прямоугольная сетка задается и вычисляется с помощью класса:

```
class grid_t
{
public:
    vector<elem_t> es; /// Массив конечных элементов сетки
    vector<basic_elem_t> bes; /// Массив узлов сетки
    int n; /// Число узлов

    /** Рассчитать неравномерную сетку. */
    void calc(const grid_generator_t& gx, const grid_generator_t& gy);
};
```

Локальные матрицы формируются, получая на вход конечный элемент `elem_t`.
Для генерации разреженной матрицы используется класс с возможностью произвольного доступа к элементам:

```cpp
class matrix_sparse_ra_t
{
public:
    matrix_sparse_ra_t(int n);

    /** Установить значение в позиции (i, j) */
    double& operator()(int i, int j);

    /** Получить значение в позиции (i, j). Если туда ещё не устанавливалось значение, вызывается
    ↪  исключение. */
    const double& operator()(int i, int j) const;

    /** Преобразует текущую матрицу к разреженной матрице. */
    matrix_sparse_t to_sparse(void) const;
private:
    int n;
    vector<double> dm;
    vector<map<int, double>> lm, um;
};
```

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-элементоной 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	$0.36 \cdot 10^{-15}$ $0.4 \cdot 10^{-16}$ 86	$0.36 \cdot 10^{-11}$ $0.31 \cdot 10^{-12}$ 157	$0.34 \cdot 10^{-11}$ $0.29 \cdot 10^{-12}$ 91	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 69	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 42	$0.61 \cdot 10^{-3}$ $0.56 \cdot 10^{-4}$ 104
$x+y$	$0.22 \cdot 10^{-11}$ $0.24 \cdot 10^{-12}$ 66	$0.35 \cdot 10^{-11}$ $0.32 \cdot 10^{-12}$ 54	$0.24 \cdot 10^{-11}$ $0.27 \cdot 10^{-12}$ 56	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 73	$0.76 \cdot 10^{-3}$ $0.69 \cdot 10^{-4}$ 60	$0.61 \cdot 10^{-3}$ $0.56 \cdot 10^{-4}$ 60
x^2+y	$0.28 \cdot 10^{-2}$ $0.17 \cdot 10^{-12}$ 54	$0.28 \cdot 10^{-2}$ $0.1 \cdot 10^{-12}$ 53	$0.28 \cdot 10^{-2}$ $0.16 \cdot 10^{-12}$ 64	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 84	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 89	$0.34 \cdot 10^{-2}$ $0.56 \cdot 10^{-4}$ 64
x^2y+y^3	$0.28 \cdot 10^{-2}$ $0.12 \cdot 10^{-12}$ 108	$0.28 \cdot 10^{-2}$ $0.16 \cdot 10^{-12}$ 143	$0.28 \cdot 10^{-2}$ $0.26 \cdot 10^{-12}$ 52	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 52	$0.35 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 62	$0.34 \cdot 10^{-2}$ $0.56 \cdot 10^{-4}$ 65
xy^2	$0.69 \cdot 10^{-3}$ $0.65 \cdot 10^{-13}$ 62	$0.69 \cdot 10^{-3}$ $0.12 \cdot 10^{-12}$ 70	$0.69 \cdot 10^{-3}$ $0.92 \cdot 10^{-13}$ 71	$0.14 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 55	$0.14 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 73	$0.13 \cdot 10^{-2}$ $0.56 \cdot 10^{-4}$ 64
x^4+y^4	$0.43 \cdot 10^{-2}$ $0.14 \cdot 10^{-3}$ 56	$0.43 \cdot 10^{-2}$ $0.14 \cdot 10^{-3}$ 48	$0.43 \cdot 10^{-2}$ $0.14 \cdot 10^{-3}$ 54	$0.48 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 49	$0.48 \cdot 10^{-2}$ $0.69 \cdot 10^{-4}$ 50	$0.47 \cdot 10^{-2}$ $0.83 \cdot 10^{-4}$ 51
e^{xy}	$0.68 \cdot 10^{-3}$ $0.12 \cdot 10^{-5}$ 51	$0.68 \cdot 10^{-3}$ $0.12 \cdot 10^{-5}$ 42	$0.68 \cdot 10^{-3}$ $0.12 \cdot 10^{-5}$ 42	$0.14 \cdot 10^{-2}$ $0.68 \cdot 10^{-4}$ 43	$0.14 \cdot 10^{-2}$ $0.68 \cdot 10^{-4}$ 39	$0.13 \cdot 10^{-2}$ $0.54 \cdot 10^{-4}$ 33

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	$0.16 \cdot 10^{-13}$ $0.37 \cdot 10^{-15}$ 6474	$0.47 \cdot 10^{-12}$ $0.1 \cdot 10^{-13}$ 6609	$0.34 \cdot 10^{-11}$ $0.94 \cdot 10^{-13}$ 8229	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 7419	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 8962	$0.27 \cdot 10^{-4}$ $0.6 \cdot 10^{-6}$ 8846
$x+y$	$0.22 \cdot 10^{-11}$ $0.54 \cdot 10^{-13}$ 7875	$0.43 \cdot 10^{-11}$ $0.96 \cdot 10^{-13}$ 7012	$0.19 \cdot 10^{-11}$ $0.45 \cdot 10^{-13}$ 8480	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 6367	$0.32 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 7514	$0.27 \cdot 10^{-4}$ $0.6 \cdot 10^{-6}$ 8530
x^2+y	$0.13 \cdot 10^{-3}$ $0.56 \cdot 10^{-14}$ 8647	$0.13 \cdot 10^{-3}$ $0.85 \cdot 10^{-14}$ 9075	$0.13 \cdot 10^{-3}$ $0.25 \cdot 10^{-13}$ 7615	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 5630	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 8021	$0.16 \cdot 10^{-3}$ $0.6 \cdot 10^{-6}$ 8169
x^2y+y^3	$0.13 \cdot 10^{-3}$ $0.8 \cdot 10^{-14}$ 7083	$0.13 \cdot 10^{-3}$ $0.12 \cdot 10^{-13}$ 4340	$0.13 \cdot 10^{-3}$ $0.19 \cdot 10^{-13}$ 8451	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 7423	$0.16 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 6780	$0.16 \cdot 10^{-3}$ $0.6 \cdot 10^{-6}$ 9024
xy^2	$0.32 \cdot 10^{-4}$ $0.21 \cdot 10^{-14}$ 8623	$0.32 \cdot 10^{-4}$ $0.41 \cdot 10^{-14}$ 7928	$0.32 \cdot 10^{-4}$ $0.5 \cdot 10^{-14}$ 7910	$0.64 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 5137	$0.64 \cdot 10^{-4}$ $0.69 \cdot 10^{-6}$ 4013	$0.59 \cdot 10^{-4}$ $0.6 \cdot 10^{-6}$ 5455
x^4+y^4	$0.2 \cdot 10^{-3}$ $0.14 \cdot 10^{-5}$ 5086	$0.2 \cdot 10^{-3}$ $0.14 \cdot 10^{-5}$ 4879	$0.2 \cdot 10^{-3}$ $0.14 \cdot 10^{-5}$ 4342	$0.23 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 4702	$0.23 \cdot 10^{-3}$ $0.69 \cdot 10^{-6}$ 3857	$0.22 \cdot 10^{-3}$ $0.79 \cdot 10^{-6}$ 4179
e^{xy}	$0.31 \cdot 10^{-4}$ $0.12 \cdot 10^{-7}$ 4024	$0.31 \cdot 10^{-4}$ $0.12 \cdot 10^{-7}$ 4891	$0.31 \cdot 10^{-4}$ $0.12 \cdot 10^{-7}$ 3713	$0.63 \cdot 10^{-4}$ $0.68 \cdot 10^{-6}$ 4633	$0.63 \cdot 10^{-4}$ $0.68 \cdot 10^{-6}$ 4508	$0.59 \cdot 10^{-4}$ $0.59 \cdot 10^{-6}$ 4586

Вывод: предыдущие выводы не опровеглись.
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 $j \leqslant i \Rightarrow x_j \leqslant x_i$.

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.

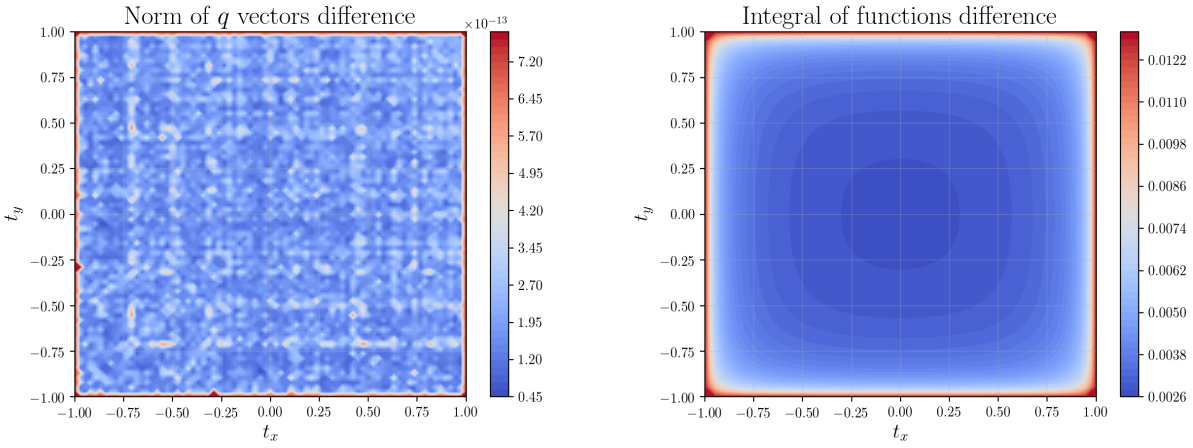
Таким образом, можно исследовать различные неравномерные сетки, изменяя параметр от -1 до 1, где точка $t = 0$ будет являться результатом на равномерной сетке.

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

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

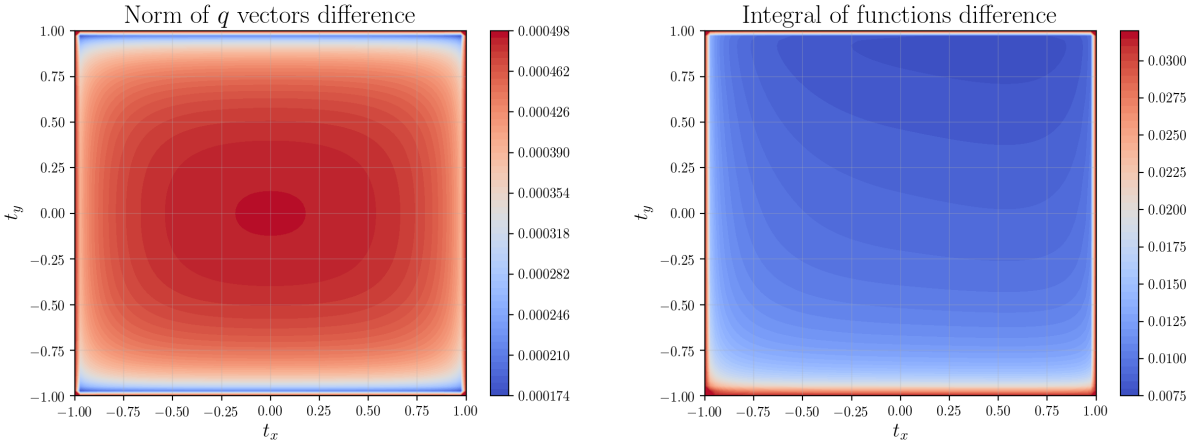


Вывод: так как эта функция полностью аппроксимируется данным методом в узлах, то не имеет значения насколько сетка неравномерна, примерно во всех элементах она

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 раномерная сетка согласно правому графику.

4.2.2.2 Функция 2

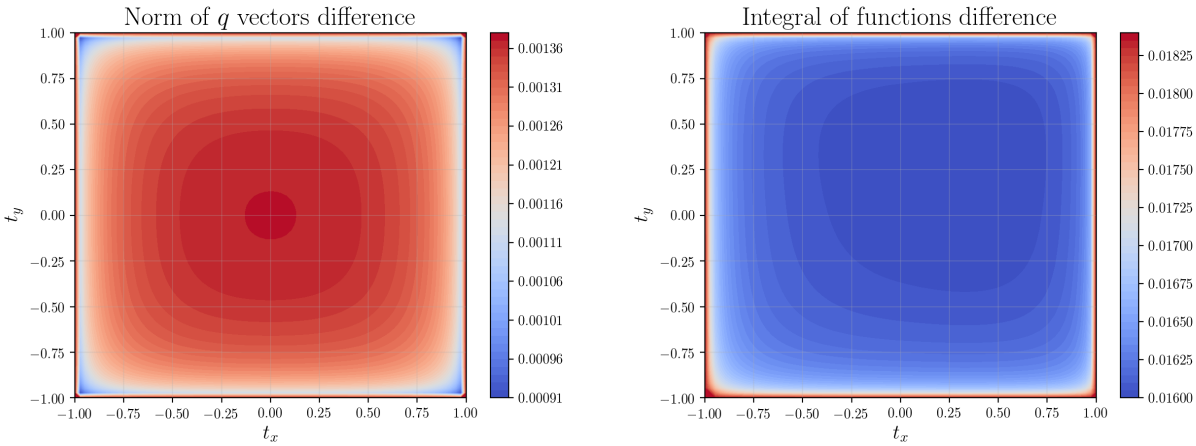
$$u = x^4 + y^3x + t^4$$



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

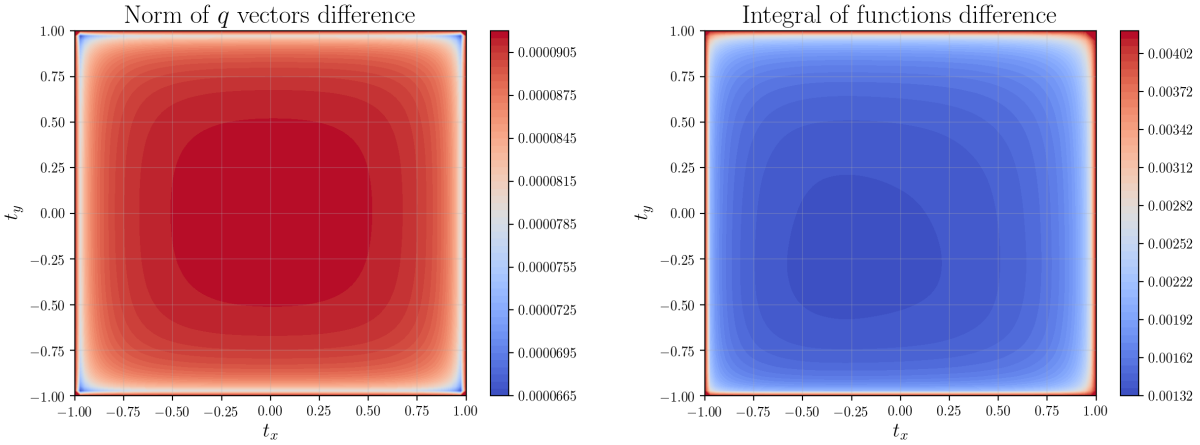
$$u = e^{xy} + e^{t^2}$$



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

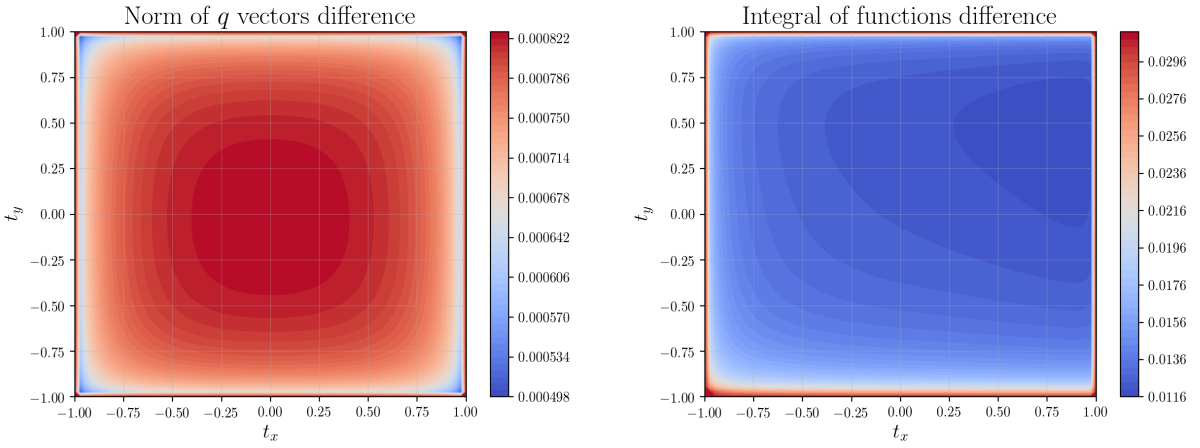
$$u = e^{(1-x)(1-y)} + e^{(1-t)^2}$$



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

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



Вывод: всё аналогично предыдущим выводам и функциям.

4.2.2.6 Общие выводы

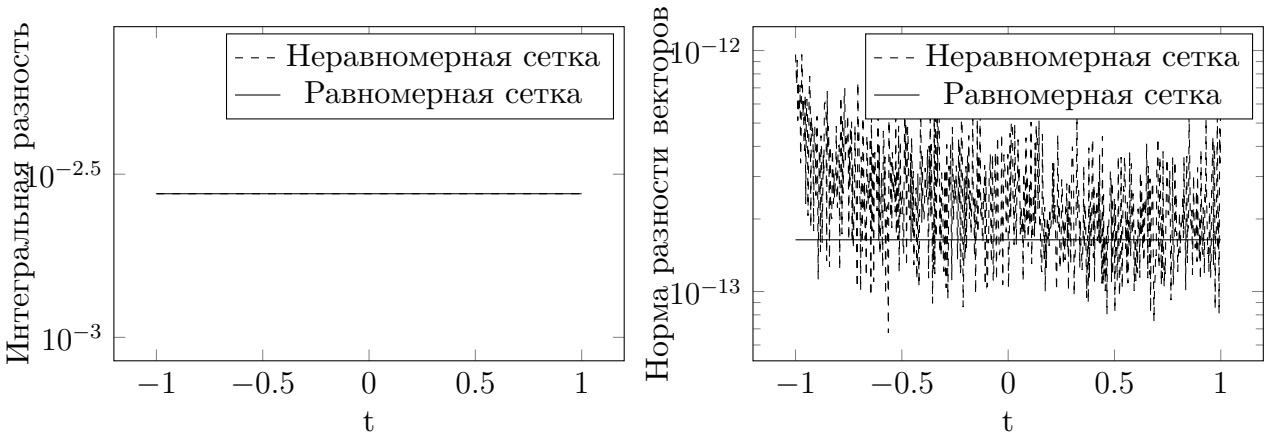
Вывод: хорошая аппроксимация в узлах $\neq$ хорошая аппроксимация по интегральной норме.
Вывод: согласно интегральной норме для неполиномиальных функций существует некоторый набор параметров t_x и t_y, при которых нелинейная сетка оптимальным образом аппроксимирует функцию.
Вывод: согласно норме в узлах 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

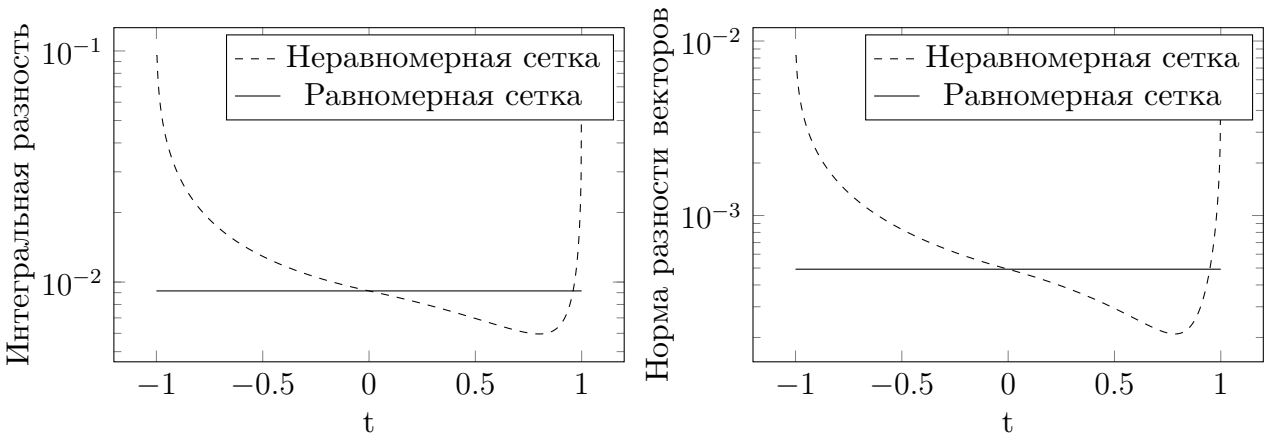
$$u = x^2 + y^2 + t^2$$



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

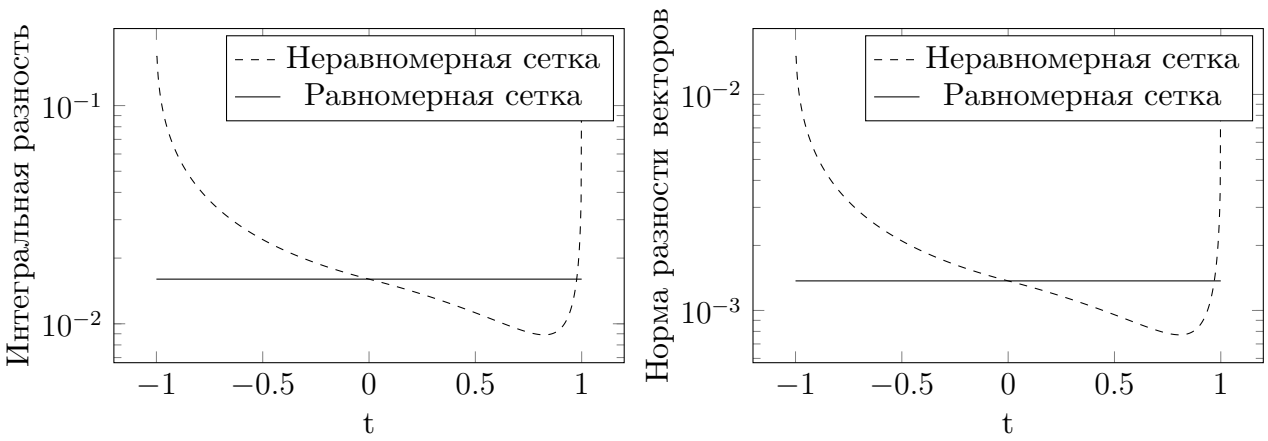
$$u = x^4 + y^3x + t^4$$



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

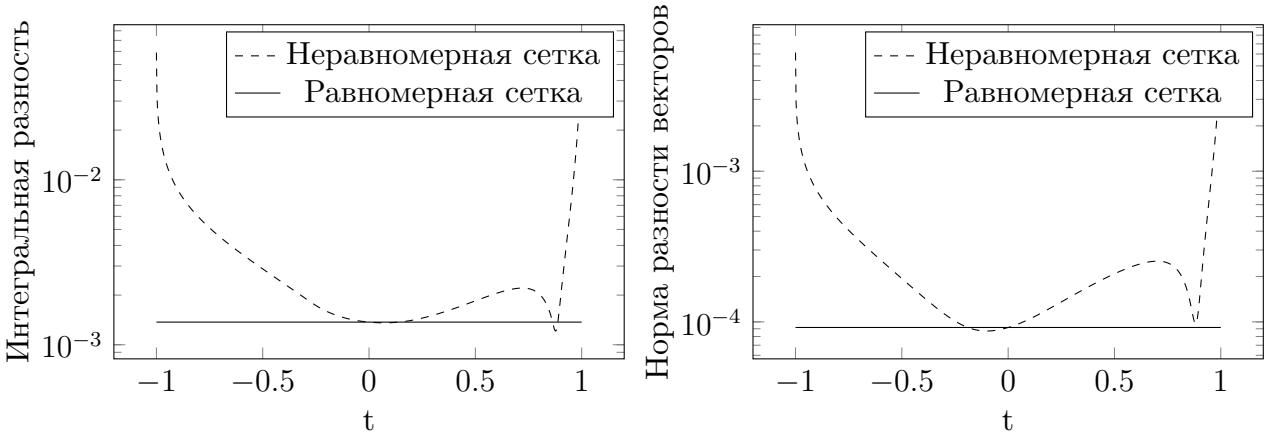
$$u = e^{xy} + e^{t^2}$$



Вывод: всё аналогично предыдущему.

4.2.3.4 Функция 4

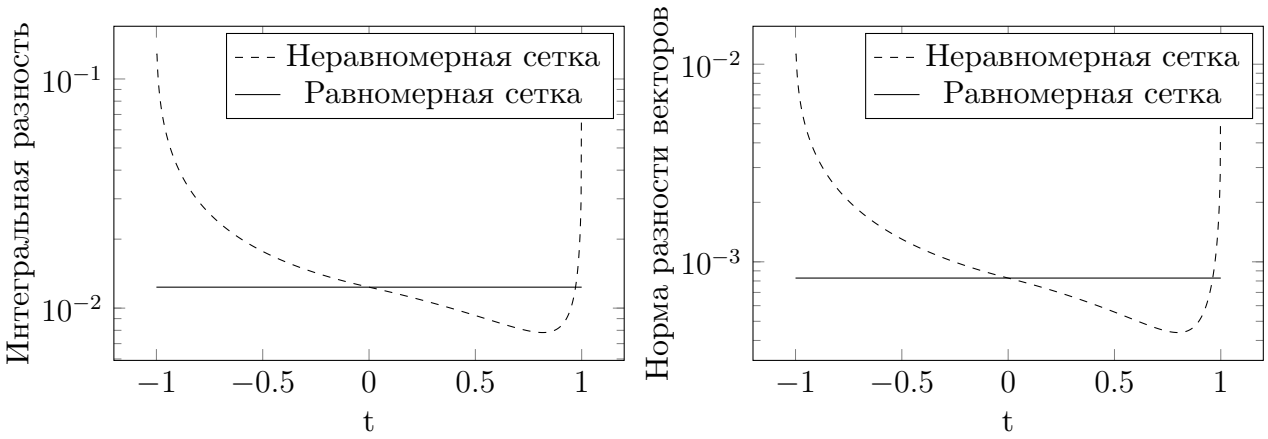
$$u = e^{(1-x)(1-y)} + e^{(1-t)^2}$$



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

4.2.3.5 Функция 5

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



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.

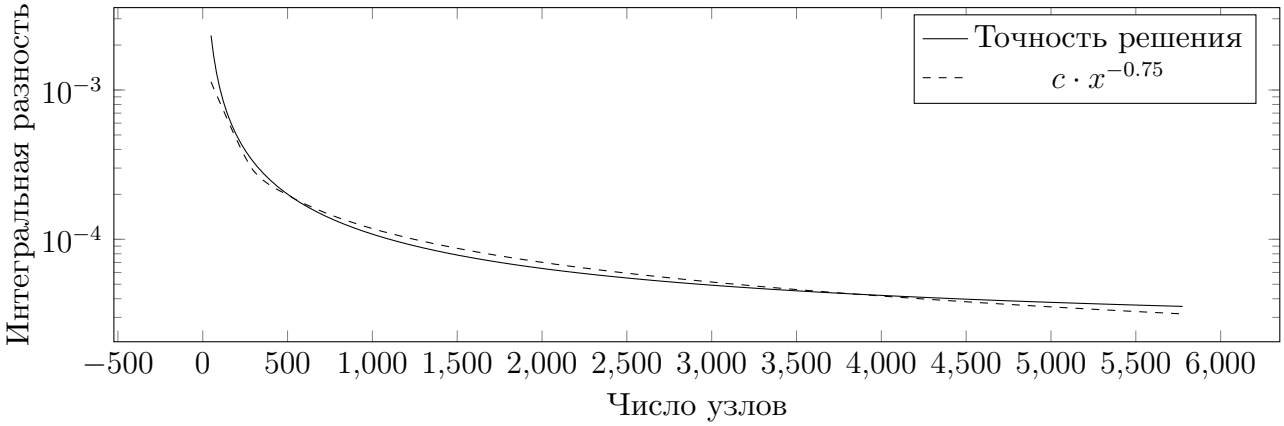
4.3 Порядок сходимости

Исследуется на функции:

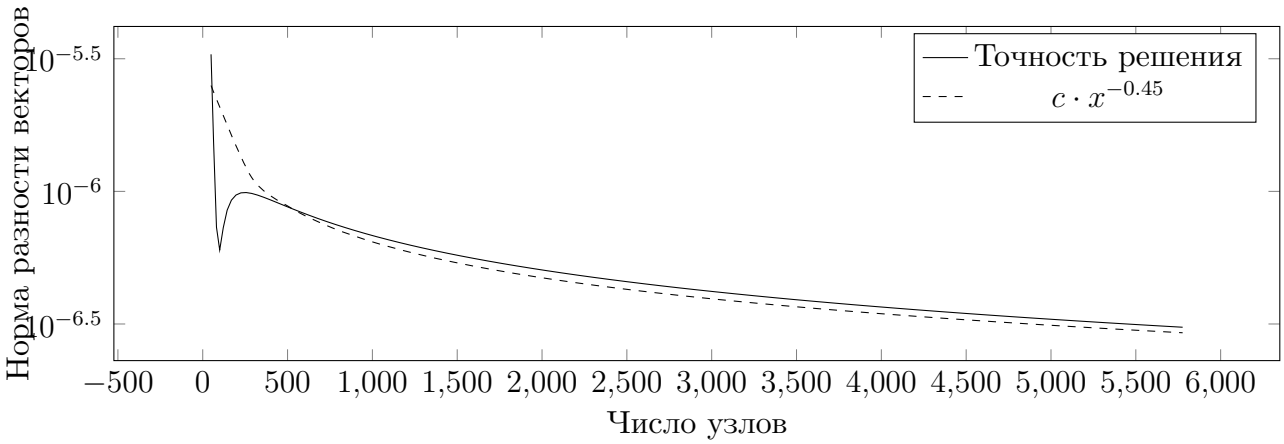
$$u(x, y, t) = e^{xy} + e^{t^2}$$

4.3.1 Увеличение размерности по пространству

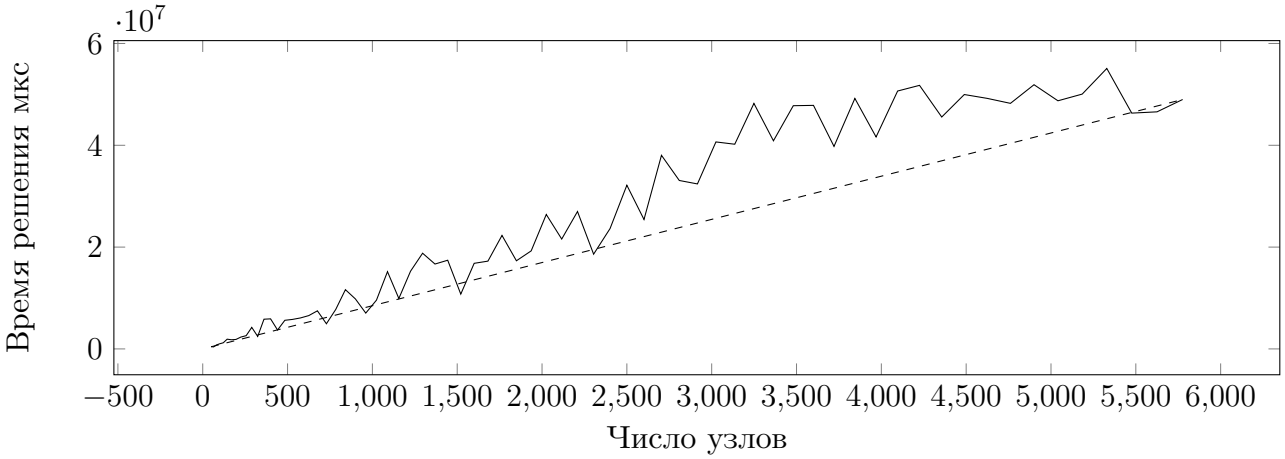
Сетка по пространству: $(x, y) \in [0, 1] \times [0, 1]$. Сетка по времени: $t \in [0, 1]$, количество элементов сетки 200.



Вывод: согласно интегральной норме, порядок сходимости ≈ 0.75.



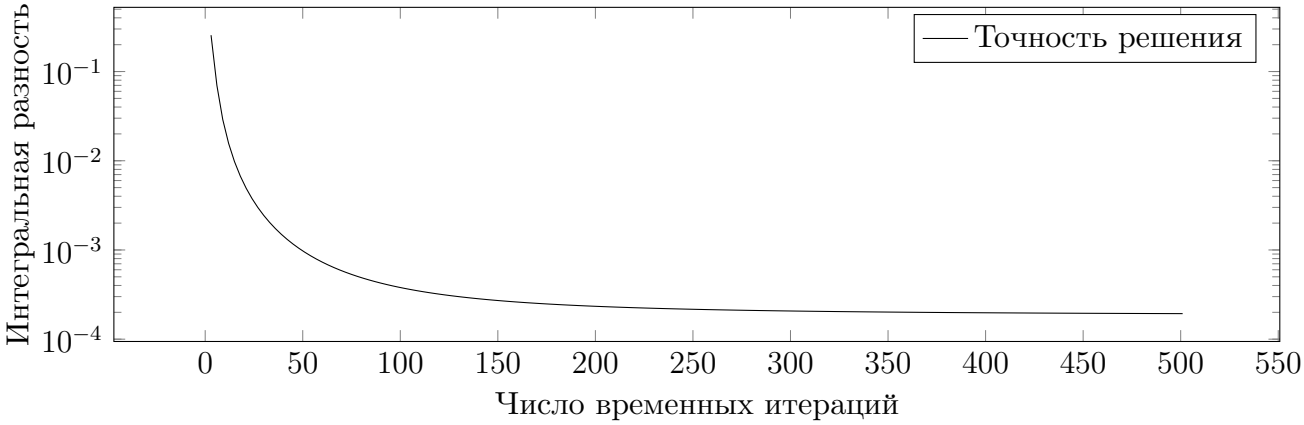
Вывод: согласно норме разности векторов, порядок сходимости ≈ 0.45.



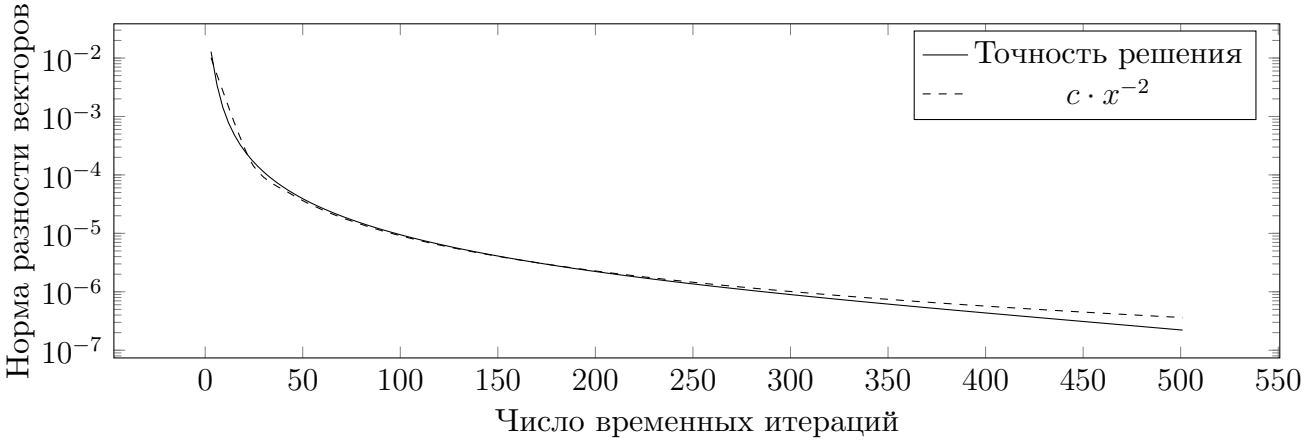
Вывод: время решения почти линейно зависит от числа узлов (с учетом погрешности, вносимой многопоточностью).

4.3.2 Увеличение размерности по времени

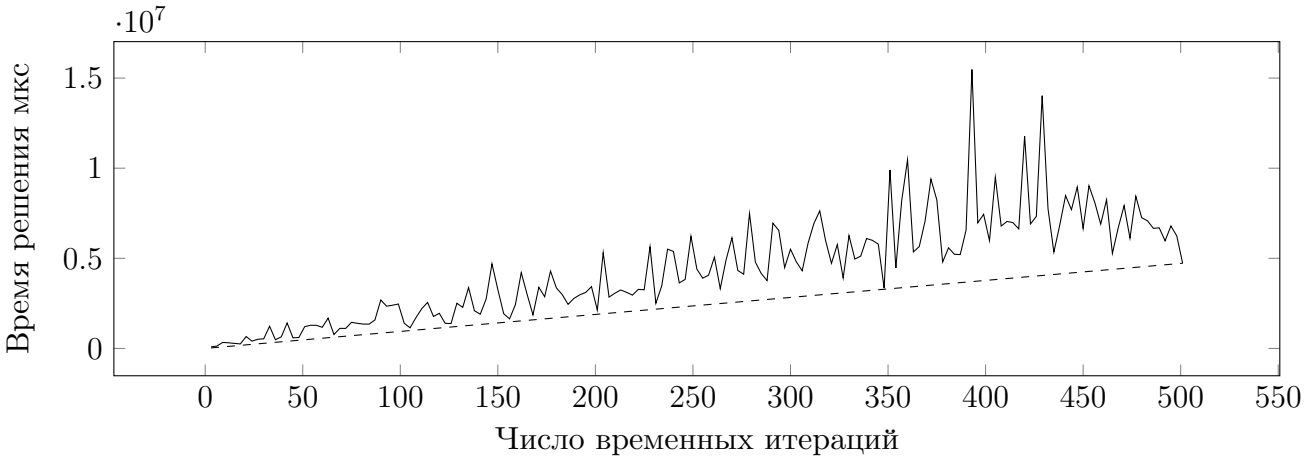
Сетка по пространству: $(x, y) \in [0, 1] \times [0, 1]$, строк и столбцов 20. Сетка по времени: $t \in [0, 1]$.



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 Файлы заголовков

FILE **lib.h**

```cpp
#pragma once

/** Определения типов. */
#include <functional>
#include <chrono>
#include <mutex>
#include <iomanip>
#include <iostream>
#include <string>
#include <Eigen/Dense>

using namespace std;
using namespace placeholders;

/* Для плотных матриц и векторов используется библиотека Eigen. */
typedef Eigen::MatrixXd matrix_t; /// Плотная матрица
typedef Eigen::VectorXd vector_t; /// Вектор

/* Тип 1D, 2D, 3D функций. */
typedef function<double(double)> function_1d_t;
typedef function<double(double, double)> function_2d_t;
typedef function<double(double, double, double)> function_3d_t;

/** Считает время выполнения функции f в микросекундах. */
inline double calc_time_microseconds(const function<void(void)>& f) {
    using namespace chrono;
    auto start = high_resolution_clock::now();
    f();
    auto end = high_resolution_clock::now();
    return duration_cast<microseconds>(end - start).count();;
}

/** Выводит на экран процент завершенной работы. Использует мьютексы для защиты cout при использовании
↪  несколькими потоками */
inline void write_percent(double percent) {
    static mutex m;
    lock_guard<mutex> g(m);
    cout << "\r" << setprecision(2) << fixed << setw(5) << percent * 100 << "%";
}

//-----------------------------------------------------------------------------
inline string write_for_latex_double(double v, int precision) {
    int power = log(std::fabs(v)) / log(10.0);
    double value = v / pow(10.0, power);

    if (v != v) return "nan";

    if (v == 0) {
        power = 0;
        value = 0;
    }
```

```
    stringstream sout;
    sout.precision(precision);
    if (power == -1 || power == 0 || power == 1) {
        sout << v;
    } else {
        sout << value << "\\cdot 10^{" << power << "}";
    }

    return sout.str();
}
```

FILE **sparse.h**

```
#pragma once

/** Файл для работы с матрицей в разряженном формате и решении */

#include <vector>
#include <map>
#include <iostream>
#include "lib.h"

//---------------------------------------------------------------------------
/** Класс квадратной разряженной матрицы в строчно-столбцовом формате с симметричным профилем.
↪  Примечание: ненулевым считается элемент, который имеется в профиле, неважно что в массивах l, u он
↪  может иметь значение 0. */
class matrix_sparse_t
{
public:
    int n;            /// Размерность матрицы
    vector<double> d; /// Диагональные элементы матрицы
    vector<double> l; /// Элементы матрицы из нижнего треугольника
    vector<double> u; /// Элементы матрицы из верхнего треугольника
    vector<int> i;    /// Массив начала строк в формате (ia в методичке)
    vector<int> j;    /// Массив столбцов каждого элемента (ja в методичке)

    matrix_sparse_t(int n);

    /** Преобразование разреженной матрицы к плотному формату. */
    void to_dense(matrix_t& m) const;

    void clear_line(int line);

    int line_elem_start(int line) const; /// Получить позицию в массивах l, u элемента, с которого
    ↪  начинается строка line
    int line_elem_row(int line, int elem) const; /// Получить столбец ненулевого элемента в строке
    ↪  line под номером elem
    int line_elem_count(int line) const; /// Получить количество ненулевых элементов в строке

    /** Раскладывает текущую матрицу в неполное LU разложение и хранит результат в матрице lu.
    ↪  Неполное разложение - это когда были применены формулы для получения LU матрицы, но только к
    ↪  существующим ненулевым элементам, без перестройки формата. Иными словами называется "неполная
    ↪  факторизация". */
    void decompose_lu_partial(matrix_sparse_t& lu) const;

    /* Методы для умножения разряженной матрицы на вектор. */
    void mul(vector_t& x_y) const;         // x_y = a * x_y
    void mul_t(vector_t& x_y) const;       // x_y = a^t * x_y

    /* Представляет, что текущая матрица хранит LU разложение, и соответственно можно каждую матрицу
    ↪  этого разложения умножить на соответствующие вектора. */
    void mul_l_inv_t(vector_t& x_y) const; // x_y = l^-t * x_y
    void mul_u_inv_t(vector_t& x_y) const; // x_y = u^-t * x_y
    void mul_l_inv(vector_t& x_y) const;   // x_y = l^-1 * x_y
    void mul_u_inv(vector_t& x_y) const;   // x_y = u^-1 * x_y
    void mul_u(vector_t& x_y) const;       // x_y = u * x_y
};

ostream& operator<<(ostream& out, const matrix_sparse_t& m);

//---------------------------------------------------------------------------
/** Квадратная матрица с произвольным доступом к любому элементу. Предполагается, что матрица будет
↪  разряженная. Далее можно перегенерировать её в разряженную матрицу. */
class matrix_sparse_ra_t
{
public:
    matrix_sparse_ra_t(int n);

    /** Установить значение в позиции (i, j) */
    double& operator()(int i, int j);

    /** Получить значение в позиции (i, j). Если туда ещё не устанавливалось значение, вызывается
    ↪  исключение. */
    const double& operator()(int i, int j) const;

```

```
    /** Преобразует текущую матрицу к разреженной матрице. */
    matrix_sparse_t to_sparse(void) const;
private:
    int n;
    vector<double> dm;
    vector<map<int, double>> lm, um;
};

//---------------------------------------------------------------------------
/* Функции для умножения векторов. */
void mul(const vector_t& d, vector_t& x_y);     // x_y = d * x_y
void mul_inv(const vector_t& d, vector_t& x_y); // x_y = x_y / d

//---------------------------------------------------------------------------
/** Решает СЛАУ с матрицей в разряженном формате при помощи Локально-Оптимальной Схемы (ЛОС) с
 ↪  предобуславливанием на основе неполной LU факторизации. */
vector_t solve_by_los_lu(
    const matrix_sparse_t& a,
    const vector_t& b,
    int maxiter,
    double eps,
    bool is_log = false
);
```

FILE **fem.h**

```
#pragma once

/* Заголовок функций для реализации Метода Конечных Элементов (МКЭ) в 2D пространстве. Уравнение
 ↪  гиперболическое (с второй производной по времени). Схема для аппроксимации по времени:
 ↪  Кранка-Николсона. Базисные элементы: билинейные. Форма сетки: прямоугольники. */

#include "lib.h"
#include "sparse.h"

//---------------------------------------------------------------------------
/** Узел конечного элемента. Другими словами, вес, домноженный на базовую функцию. Из сумм этих
 ↪  элементов образуется конечный элемент. */
struct basic_elem_t
{
    int i; /// Номер узла

    double x, y; /// Координаты узла

    basic_elem_t *up, *down, *left, *right; /// Указатели на соседей узла

    /** Проверяет, является ли элемент граничным. Он таким является, если у него нет хотя бы одного
     ↪  соседа. */
    bool is_boundary(void) const;
};

/** Прямоугольный конечный элемент на основе билинейных базисных функций. Образуется из четырех узлов.
 ↪  */
struct elem_t
{
    int i; /// Номер конечного элемента
    basic_elem_t* e[4]; /// Указатели на все 4 элемента конечного узла, нумерация такая:
    /**
         Y
         ^  3 +-----+ 4
         |    |     |
         |  1 +-----+ 2
        -+--------------> X
         |
     */

    double get_hx(void) const; /// Ширина конечного элемента
    double get_hy(void) const; /// Высота конечного элемента

    /** Рассчитать значение внутри конечного элемента. q - вектор рассчитыванных весов. */
    double value(double x, double y, const vector_t& q) const;
};

/** Все константы решаемого уравнения. */
struct constants_t
{
    double lambda; /// Коэффициент внутри div
    double gamma;  /// Коэффициент при u
    double sigma;  /// Коэффициент при du/dt
    double chi;    /// Коэффициент при d^2u/dt^2
};

//---------------------------------------------------------------------------
```

```
/** 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 неравномернной сетки. */
double non_linear_offset(double x, double t);

/** 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. */
class grid_generator_t
{
public:
    grid_generator_t(double a, double b, int n, double t = 0);
    double operator()(int i) const;
    int size(void) const;
    double back(void) const;
private:
    double a, len, t, n1;
};

/** Класс двумерной неравномерной сетки по пространству в виде прямоугольника. */
class grid_t
{
public:
    vector<elem_t> es; /// Массив конечных элементов сетки
    vector<basic_elem_t> bes; /// Массив узлов сетки
    int n; /// Число узлов

    /** Рассчитать неравномерную сетку. */
    void calc(const grid_generator_t& gx, const grid_generator_t& gy);
};

/** Рассчитать веса идеальной аппроксимации функции u при помощи узлов bes. */
vector_t calc_true_approx(const function_2d_t& u, const vector<basic_elem_t>& bes);

/** Рассчитать интегральную норму между конечно-элементной аппроксимацией и истинной функцией. */
double calc_integral_norm(const function_2d_t& u, const vector<elem_t>& es, const vector_t& q);

//---------------------------------------------------------------------------
/* Расчет локальных матриц для конечного элемента. */
matrix_t calc_local_matrix_g(const elem_t& e, const constants_t& cs);
matrix_t calc_local_matrix_c(const elem_t& e);
vector_t calc_local_vector_b(const elem_t& e, const function_2d_t& f);

//---------------------------------------------------------------------------
/** Рассчитать глобальный вектор из локальных векторов для всех конечных элементов. */
vector_t calc_global_vector(
    const vector<elem_t>& es,
    const function<vector_t(const elem_t&)> calc_local_vector,
    int n
);

/** Рассчитать глобальную матрицу из функции построения локальных матриц. */
matrix_sparse_t calc_global_matrix(
    const vector<elem_t>& es,
    const function<matrix_t(const elem_t&)> calc_local_matrix,
    int n
);

//---------------------------------------------------------------------------
/* Численный расчет определенных интегралов. */
double calc_integral_gauss3(
    double a, double b, int n, // n - количество внутренних узлов
    const function_1d_t& f
);
double calc_integral_gauss3(
    double ax, double bx, int nx,
    double ay, double by, int ny,
    const function_2d_t& f
);

//---------------------------------------------------------------------------
/* Численный расчет производной. */
function_1d_t calc_first_derivative(const function_1d_t& f);
function_1d_t calc_second_derivative(const function_1d_t& f);

//---------------------------------------------------------------------------
/** Для гиперболического дифференциального уравнения и функции u считает каким должно быть f, чтобы
↪  решением этого диф. уравнения была функци u. Делает это численно. */
function_3d_t calc_right_part(const function_3d_t& u, const constants_t& cs);

//---------------------------------------------------------------------------
/** Использует схему Кранка-Николсона для получения СЛАУ. Предполагается, что разряженные матрицы
↪  имеют одинаковый формат. */
void calc_crank_nicolson_method(
    const matrix_sparse_t& c,
    const matrix_sparse_t& g,
    const vector_t& b0,  // b current (b_j)
    const vector_t& bl,  // b last (b_{j-1})
    const vector_t& bll, // b last last (b_{j-2})
```

```
    const vector_t& ql,  // q last (q_{j-1})
    const vector_t& qll, // q last last (q_{j-2})
    const constants_t& cs,
    const grid_generator_t& time_grid,
    int time_i,
    matrix_sparse_t& a,
    vector_t& b
);

//---------------------------------------------------------------------------
/** 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. */
typedef function<void(matrix_sparse_t&, vector_t&, const vector<basic_elem_t>&, double)> boundary_setter_t;

/** Записывает первые краевые условия в матрицу a и вектор b. Для этой записи ему необходимо получить истинную функцию. */
void write_first_boundary_conditions(
    matrix_sparse_t& a,
    vector_t& b,
    const vector<basic_elem_t>& bes,
    double t,
    const function_3d_t& u
);

//---------------------------------------------------------------------------
/** 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 разряженной строчно-столбцовой матрице для решения СЛАУ. */
vector<vector_t> solve_differential_equation(
    const function_3d_t& f,
    const boundary_setter_t& set_boundary_conditions,
    const vector_t& q0,
    const vector_t& q1,
    const constants_t& cs,
    const grid_t& grid,
    const grid_generator_t& time_grid
);
```

5.2 Исходные файлы

FILE **sparse.cpp**

```
#include "sparse.h"

//---------------------------------------------------------------------------
matrix_sparse_t::matrix_sparse_t(int n) : n(n) {
    d.resize(n);
    i.resize(n+1, 0);
}

//---------------------------------------------------------------------------
void matrix_sparse_t::to_dense(matrix_t& m) const {
    m = matrix_t(n, n);
    m.fill(0);
    for (int _i = 0; _i < n; ++_i) {
        m(_i, _i) = d[_i];
        for (int _j = 0; _j < line_elem_count(_i); ++_j) {
            m(_i, line_elem_row(_i, _j)) = l[line_elem_start(_i) + _j];
            m(line_elem_row(_i, _j), _i) = u[line_elem_start(_i) + _j];
        }
    }
}

//---------------------------------------------------------------------------
void matrix_sparse_t::clear_line(int line) {
    d[line] = 0;
    for (int _i = 0; _i < i.size()-1; _i++) {
        for (int pj = i[_i]; pj < i[_i+1]; pj++) {
            int _j = j[pj];
            if (_j == line) u[pj] = 0;
            if (_i == line) l[pj] = 0;
        }
    }
}

//---------------------------------------------------------------------------
int matrix_sparse_t::line_elem_start(int line) const {
    return i[line];
}
//---------------------------------------------------------------------------
```

```
int matrix_sparse_t::line_elem_row(int line, int elem) const {
    return j[line_elem_start(line) + elem];
}

//---------------------------------------------------------------------------
int matrix_sparse_t::line_elem_count(int line) const {
    return i[line+1]-i[line];
}

//---------------------------------------------------------------------------
void matrix_sparse_t::decompose_lu_partial(matrix_sparse_t& lu) const {
    const matrix_sparse_t& a = *this;

    lu = a;
    for (int i = 0; i < lu.n; ++i) {
        // Заполняем нижний треугольник
        int line_start = lu.line_elem_start(i);
        int line_end = lu.line_elem_start(i+1);
        for (int j = line_start; j < line_end; ++j) {
            double sum = 0;

            int row = lu.j[j];
            int row_start = lu.line_elem_start(row);
            int row_end = lu.line_elem_start(row+1);

            int kl = line_start;
            int ku = row_start;

            while (kl < j && ku < row_end) {
                if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
                    sum += lu.l[kl] * lu.u[ku];
                    ku++;
                    kl++;
                } else if (lu.j[kl] < lu.j[ku]) {
                    kl++;
                } else {
                    ku++;
                }
            }

            lu.l[j] = (a.l[j] - sum) / lu.d[row];
        }

        // Заполняем верхний треугольник
        int row_start = lu.line_elem_start(i);
        int row_end = lu.line_elem_start(i+1);
        for (int j = line_start; j < line_end; ++j) {
            double sum = 0;

            int line = lu.j[j];
            int line_start = lu.line_elem_start(line);
            int line_end = lu.line_elem_start(line+1);

            int kl = line_start;
            int ku = row_start;

            while (kl < line_end && ku < j) {
                if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
                    sum += lu.l[kl] * lu.u[ku];
                    ku++;
                    kl++;
                } else if (lu.j[kl] < lu.j[ku]) {
                    kl++;
                } else {
                    ku++;
                }
            }

            lu.u[j] = (a.u[j] - sum) / lu.d[line];
        }

        // Расчитываем диагональный элемент
        double sum = 0;
        int line_row_start = lu.line_elem_start(i);
        int line_row_end = lu.line_elem_start(i+1);
        for (int j = line_row_start; j < line_row_end; ++j)
            sum += lu.l[j] * lu.u[j];

        lu.d[i] = sqrt(a.d[i] - sum);
    }
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul(vector_t& x_y) const {
    const matrix_sparse_t& a = *this;

    vector_t result(a.n); result.fill(0);

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

```
        int start = a.line_elem_start(i);
        int size = a.line_elem_count(i);
        for (int j = 0; j < size; j++) {
            result[i] += a.l[start + j] * x_y[a.line_elem_row(i, j)];
            result[a.line_elem_row(i, j)] += a.u[start + j] * x_y[i];
        }
    }

    // Умножение диагональных элементов на вектор
    for (int i = 0; i < a.n; ++i)
        result[i] += a.d[i] * x_y[i];

    x_y = result;
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul_t(vector_t& x_y) const {
    const matrix_sparse_t& a = *this;

    vector_t result(a.n); result.fill(0);

    for (int i = 0; i < a.n; ++i) {
        int start = a.line_elem_start(i);
        int size = a.line_elem_count(i);
        for (int j = 0; j < size; j++) {
            result(i) += a.u[start + j] * x_y[a.line_elem_row(i, j)];
            result(a.line_elem_row(i, j)) += a.l[start + j] * x_y[i];
        }
    }

    // Умножение диагональных элементов на вектор
    for (int i = 0; i < a.n; ++i)
        result(i) += a.d[i] * x_y[i];

    x_y = result;
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul_l_inv_t(vector_t& x_y) const {
    const matrix_sparse_t& l = *this;

    for (int i = l.n - 1; i >= 0; i--) {
        int start = l.line_elem_start(i);
        int size = l.line_elem_count(i);

        x_y[i] /= l.d[i];
        for (int j = 0; j < size; ++j)
            x_y[l.line_elem_row(i, j)] -= x_y[i] * l.l[start + j];
    }
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul_u_inv_t(vector_t& x_y) const {
    const matrix_sparse_t& u = *this;

    for (int i = 0; i < u.n; ++i) {
        int start = u.line_elem_start(i);
        int size = u.line_elem_count(i);

        double sum = 0;
        for (int j = 0; j < size; ++j)
            sum += u.u[start + j] * x_y[u.line_elem_row(i, j)];
        x_y[i] = (x_y[i] - sum) / u.d[i];
    }
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul_l_inv(vector_t& x_y) const {
    const matrix_sparse_t& l = *this;

    for (int i = 0; i < l.n; ++i) {
        int start = l.line_elem_start(i);
        int size = l.line_elem_count(i);

        double sum = 0;
        for (int j = 0; j < size; ++j)
            sum += l.l[start + j] * x_y[l.line_elem_row(i, j)];
        x_y[i] = (x_y[i] - sum) / l.d[i];
    }
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul_u_inv(vector_t& x_y) const {
    const matrix_sparse_t& u = *this;

    for (int i = u.n-1; i >= 0; i--) {
        int start = u.line_elem_start(i);
        int size = u.line_elem_count(i);

```

```
        x_y[i] /= u.d[i];
        for (int j = 0; j < size; ++j)
            x_y[u.line_elem_row(i, j)] -= x_y[i] * u.u[start + j];
    }
}

//---------------------------------------------------------------------------
void matrix_sparse_t::mul_u(vector_t& x_y) const {
    const matrix_sparse_t& u = *this;

    vector_t result(u.n); result.fill(0);

    for (int i = 0; i < u.n; ++i) {
        int start = u.line_elem_start(i);
        int size = u.line_elem_count(i);
        for (int j = 0; j < size; j++) {
            result[u.line_elem_row(i, j)] += u.u[start + j] * x_y[i];
        }
    }

    // Умножение диагональных элементов на вектор
    for (int i = 0; i < u.n; ++i)
        result[i] += u.d[i] * x_y[i];

    x_y = result;
}

//---------------------------------------------------------------------------
ostream& operator<<(ostream& out, const matrix_sparse_t& m) {
    matrix_t dense;
    m.to_dense(dense);
    out << dense;
    return out;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
matrix_sparse_ra_t::matrix_sparse_ra_t(int n) : n(n), dm(n, 0), lm(n), um(n) {}

//---------------------------------------------------------------------------
double& matrix_sparse_ra_t::operator()(int i, int j) {
    if (i == j) {
        return dm[i];
    } else if (i > j) {
        um[i][j] += 0;
        return lm[i][j];
    } else {
        lm[j][i] += 0;
        return um[j][i];
    }
}

//---------------------------------------------------------------------------
const double& matrix_sparse_ra_t::operator()(int i, int j) const {
    if (i == j) {
        return dm[i];
    } else if (i > j) {
        if (lm[i].find(j) != lm[i].end())
            return lm[i].at(j);
    } else {
        if (um[j].find(i) != um[i].end())
            return um[j].at(j);
    }

    throw exception();
}

//---------------------------------------------------------------------------
matrix_sparse_t matrix_sparse_ra_t::to_sparse(void) const {
    matrix_sparse_t result(n);
    result.n = dm.size();
    result.d = dm;
    for (int i = 0; i < lm.size(); ++i) {
        result.i[i+1] = result.i[i] + lm[i].size();
        for (auto& j : lm[i]) {
            result.j.push_back(j.first);
            result.l.push_back(j.second);
            result.u.push_back(um[i].at(j.first));
        }
    }
    return result;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

```
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) <<
        ↪  setprecision(16) << residual << endl;
378         if (fabs(residual) < eps || i > maxiter)
379             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 {
```

```
    return
        up == nullptr ||
        down == nullptr ||
        left == nullptr ||
        right == nullptr;
}

//---------------------------------------------------------------------------
double elem_t::get_hx(void) const {
    return e[1]->x - e[0]->x;
}

//---------------------------------------------------------------------------
double elem_t::get_hy(void) const {
    return e[3]->y - e[0]->y;
}

//---------------------------------------------------------------------------
double elem_t::value(double x, double y, const vector_t& q) const {
    double xp = e[0]->x;
    double xp1 = e[1]->x;
    double ys = e[0]->y;
    double ys1 = e[2]->y;
    double hx = get_hx();
    double hy = get_hy();
    auto x1 = [xp1, hx](double x) -> double { return (xp1 - x) / hx; };
    auto x2 = [xp, hx](double x) -> double { return (x - xp) / hx; };
    auto y1 = [ys1, hy](double y) -> double { return (ys1 - y) / hy; };
    auto y2 = [ys, hy](double y) -> double { return (y - ys) / hy; };

    auto psi1 = [&]() -> double { return x1(x) * y1(y); };
    auto psi2 = [&]() -> double { return x2(x) * y1(y); };
    auto psi3 = [&]() -> double { return x1(x) * y2(y); };
    auto psi4 = [&]() -> double { return x2(x) * y2(y); };

    double v1 = psi1() * q[e[0]->i];
    double v2 = psi2() * q[e[1]->i];
    double v3 = psi3() * q[e[2]->i];
    double v4 = psi4() * q[e[3]->i];

    return v1 + v2 + v3 + v4;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
double non_linear_offset(double x, double t) {
    int signt = (t > 0) ? 1 : -1;
    t *= signt;
    t = 1.0 - t;
    t = (signt == -1) ? 1.0/t : t;
    if (t == 1.0) return x;
    return (1.0 - pow(t, x))/(1.0 - t);
}

//---------------------------------------------------------------------------
grid_generator_t::grid_generator_t(double a, double b, int n, double t) : a(a), len(b-a), n1(n+1.0),
↪  t(t) {}

//---------------------------------------------------------------------------
double grid_generator_t::operator()(int i) const {
    return a + len * non_linear_offset(i/n1, t);
}

//---------------------------------------------------------------------------
int grid_generator_t::size(void) const {
    return n1+1;
}

//---------------------------------------------------------------------------
double grid_generator_t::back(void) const {
    return operator()(size()-1);
}

//---------------------------------------------------------------------------
void grid_t::calc(const grid_generator_t& gx, const grid_generator_t& gy) {
    bes.clear();
    bes.resize(gx.size() * gy.size());
    int counter = 0;
    for (int j = 0; j < gy.size(); ++j) {
        double y = gy(j);
        for (int i = 0; i < gx.size(); ++i) {
            double x = gx(i);
            basic_elem_t* down = (counter >= gx.size()) ? &bes[counter-gx.size()] : nullptr;
            basic_elem_t* left = (counter % gx.size() > 0) ? &bes[counter-1] : nullptr;
            bes[counter] = {counter, x, y,
```

```
                nullptr,
                down,
                left,
                nullptr
            };
            if (down != nullptr) down->up = &bes[counter];
            if (left != nullptr) left->right = &bes[counter];
            counter++;
        }
    }

    es.clear();
    counter = 0;
    for (auto& i : bes) {
        if (i.right != nullptr && i.up != nullptr && i.up->right == i.right->up && i.up->right !=
        ↪  nullptr) {
            es.push_back({counter,
                i.right->left,
                i.right,
                i.up,
                i.up->right
            });
            counter++;
        }
    }

    n = bes.size();
}

//---------------------------------------------------------------------------
vector_t calc_true_approx(const function_2d_t& u, const vector<basic_elem_t>& bes) {
    vector_t result(bes.size());
    for (int i = 0; i < bes.size(); ++i)
        result(i) = u(bes[i].x, bes[i].y);
    return result;
}

//---------------------------------------------------------------------------
double calc_integral_norm(const function_2d_t& u, const vector<elem_t>& es, const vector_t& q) {
    double sum = 0;
    for (auto& i : es) {
        sum += calc_integral_gauss3(
            i.e[0]->x, i.e[1]->x, 5,
            i.e[0]->y, i.e[2]->y, 5,
            [&](double x, double y) -> double {
                return fabs(u(x, y) - i.value(x, y, q));
            }
        );
    }
    return sum;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
matrix_t calc_local_matrix_g(
    const elem_t& e,
    const constants_t& cs
) {
    double hx = e.get_hx();
    double hy = e.get_hy();
    matrix_t result;
    matrix_t a(4, 4), b(4, 4);
    a <<
        2, -2, 1, -1,
        -2, 2, -1, 1,
        1, -1, 2, -2,
        -1, 1, -2, 2;
    b <<
        2, 1, -2, -1,
        1, 2, -1, -2,
        -2, -1, 2, 1,
        -1, -2, 1, 2;
    result = cs.lambda/6.0*(hy/hx*a + hx/hy*b);
    return result;
}

//---------------------------------------------------------------------------
matrix_t calc_local_matrix_c(
    const elem_t& e
) {
    double hx = e.get_hx();
    double hy = e.get_hy();
    matrix_t result;
    matrix_t c(4, 4);
    c <<
        4, 2, 2, 1,
```

```
        2, 4, 1, 2,
        2, 1, 4, 2,
        1, 2, 2, 4;
    result = hx*hy/36.0*c;
    return result;
}

//---------------------------------------------------------------------------
vector_t calc_local_vector_b(
    const elem_t& e,
    const function_2d_t& f
) {
    vector_t fv(4);
    fv <<
        f(e.e[0]->x, e.e[0]->y),
        f(e.e[1]->x, e.e[1]->y),
        f(e.e[2]->x, e.e[2]->y),
        f(e.e[3]->x, e.e[3]->y);
    return calc_local_matrix_c(e) * fv;
}

//---------------------------------------------------------------------------
vector_t calc_global_vector(
    const vector<elem_t>& es,
    const function<vector_t(const elem_t&)> calc_local_vector,
    int n
) {
    vector_t result(n);
    result.fill(0);
    for (auto& e : es) {
        auto b = calc_local_vector(e);
        for (int i = 0; i < 4; ++i) {
            result(e.e[i]->i) += b(i);
        }
    }
    return result;
}

//---------------------------------------------------------------------------
matrix_sparse_t calc_global_matrix(
    const vector<elem_t>& es,
    const function<matrix_t(const elem_t&)> calc_local_matrix,
    int n
) {
    matrix_sparse_ra_t result(n);
    for (auto& e : es) {
        auto m = calc_local_matrix(e);
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                result(e.e[i]->i, e.e[j]->i) += m(i, j);
            }
        }
    }
    return result.to_sparse();
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
double calc_integral_gauss3(
    double a, double b, int n, // n - количество внутренних узлов
    const function_1d_t& f
) {
    const double x1 = -sqrt(3.0/5.0);
    const double x2 = 0;
    const double x3 = -x1;
    const double q1 = 5.0/9.0;
    const double q2 = 8.0/9.0;
    const double q3 = q1;
    double sum = 0;
    double xk = 0;
    double h = (b-a)/double(n+1);
    double h2 = h/2.0;

    for (int i = 0; i < n+1; ++i) {
        xk = a + h*i + h2;
        sum += q1 * f(xk + x1 * h2);
        sum += q2 * f(xk + x2 * h2);
        sum += q3 * f(xk + x3 * h2);
    }

    sum *= h;
    sum /= 2.0;
    return sum;
}

//---------------------------------------------------------------------------
```

```
double calc_integral_gauss3(
    double ax, double bx, int nx,
    double ay, double by, int ny,
    const function_2d_t& f
) {
    return calc_integral_gauss3(ax, bx, nx, [ay, by, ny, f](double x)->double {
        return calc_integral_gauss3(ay, by, ny, bind(f, x, _1));
    });
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
function_1d_t calc_first_derivative(const function_1d_t& f) {
    return [f](double x) -> double {
        const double h = 0.001;
        return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) / (12 * h);
    };
}

//---------------------------------------------------------------------------
function_1d_t calc_second_derivative(const function_1d_t& f) {
    return [f](double x) -> double {
        const double h = 0.001;
        return (-f(x+2*h) + 16*f(x+h) - 30*f(x) + 16*f(x-h) - f(x-2*h))/(12*h*h);
    };
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
function_3d_t calc_right_part(
    const function_3d_t& u,
    const constants_t& cs
) {
    // f = -div(lambda * grad u) + gamma * u + sigma * du/dt + chi * d^2 u/dt^2
    return [=](double x, double y, double t) -> double {
        using namespace placeholders;
        auto ut = calc_first_derivative(bind(u, x, y, _1));

        auto uxx = calc_second_derivative(bind(u, _1, y, t));
        auto uyy = calc_second_derivative(bind(u, x, _1, t));
        auto utt = calc_second_derivative(bind(u, x, y, _1));

        return -cs.lambda * (uxx(x) + uyy(y)) + cs.gamma * u(x, y, t) + cs.sigma * ut(t) + cs.chi *
        ↪  utt(t);
    };
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void calc_crank_nicolson_method(
    const matrix_sparse_t& c,
    const matrix_sparse_t& g,
    const vector_t& b0,
    const vector_t& bl,
    const vector_t& bll,
    const vector_t& ql,
    const vector_t& qll,
    const constants_t& cs,
    const grid_generator_t& time_grid,
    int time_i,
    matrix_sparse_t& a,
    vector_t& b
) {
    // Схема Кранка-Николсона

    // Константы для вычислений с неравномерной сеткой по времени
    double t0 = time_grid(time_i);
    double t1 = time_grid(time_i-1);
    double t2 = time_grid(time_i-2);

    double d1 = t0-t2;
    double d2 = (t0*(t0-t2-t1)+t2*t1)/2.0;
    double m1 = (t0-t2)/(t1-t2);
    double m2 = (t0-t1)/(t1-t2);

    // Вычисляемт матрицу a
    a = c;
    double c1 = cs.gamma/2.0 + cs.sigma/d1 + cs.chi/d2;
    for (int i = 0; i < a.d.size(); i++)
        a.d[i] = g.d[i]/2.0 + c.d[i]*c1;
```

```
    for (int i = 0; i < a.l.size(); i++) {
        a.l[i] = g.l[i]/2.0 + c.l[i]*c1;
        a.u[i] = g.u[i]/2.0 + c.u[i]*c1;
    }

    // Рассчитываем вектор b
    b = (b0 + bll)/2.0;
    vector_t temp = qll;
    g.mul(temp);
    b = b - temp/2.0;

    temp = ql*(m1*cs.chi/d2) + qll*(-cs.gamma/2.0 + cs.sigma/d1 - m2*cs.chi/d2);
    c.mul(temp);

    b = b + temp;
}

//---------------------------------------------------------------------------
void write_first_boundary_conditions(
    matrix_sparse_t& a,
    vector_t& b,
    const vector<basic_elem_t>& bes,
    double t,
    const function_3d_t& u
) {
    for (int i = 0; i < bes.size(); ++i) {
        if (bes[i].is_boundary()) {
            a.clear_line(bes[i].i);
            a.d[bes[i].i] = 1;
            b(bes[i].i) = u(bes[i].x, bes[i].y, t);
        }
    }
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
vector<vector_t> solve_differential_equation(
    const function_3d_t& f,
    const boundary_setter_t& set_boundary_conditions,
    const vector_t& q0,
    const vector_t& q1,
    const constants_t& cs,
    const grid_t& grid,
    const grid_generator_t& time_grid
) {
    auto c = calc_global_matrix(grid.es, calc_local_matrix_c, grid.n);
    auto g = calc_global_matrix(grid.es, bind(calc_local_matrix_g, _1, cs), grid.n);

    auto calc_global_vector_b = [&] (int i) {
        return calc_global_vector(
            grid.es,
            bind(
                calc_local_vector_b,
                _1,
                function_2d_t(bind(f, _1, _2, time_grid(i)))
            ),
            grid.n
        );
    };

    vector_t bll = calc_global_vector_b(0);
    vector_t bl = calc_global_vector_b(1);
    vector_t b0;

    vector_t qll = q0;
    vector_t ql = q1;
    vector_t q;

    vector<vector_t> result;
    result.push_back(q0);
    result.push_back(q1);

    matrix_sparse_t a(grid.n);
    vector_t b(grid.n);
    for (int i = 2; i < time_grid.size(); ++i) {
        b0 = calc_global_vector_b(i);
        calc_crank_nicolson_method(c, g, b0, bl, bll, ql, qll, cs, time_grid, i, a, b);
        set_boundary_conditions(a, b, grid.bes, time_grid(i));

        q = solve_by_los_lu(a, b, 1000, 1e-16, false);

        result.push_back(q);
```

```
446
447         bll = bl;
448         bl = b0;
449
450         qll = ql;
451         ql = q;
452     }
453
454     return result;
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,
            ↪  _4, u);
33
34         grid_t grid;
35         grid.calc(x_grid, y_grid);
36
37         vector_t q0 = calc_true_approx(bind(u, _1, _2, time_grid(0)), grid.bes);
38         vector_t q1 = calc_true_approx(bind(u, _1, _2, time_grid(1)), grid.bes);
39         vector_t q = calc_true_approx(bind(u, _1, _2, time_grid.back()), grid.bes);
40
41         auto steps = solve_differential_equation(f, set_boundary_conditions, q0, q1, c, grid,
            ↪  time_grid);
42
43         res.integral_residual = calc_integral_norm(bind(u, _1, _2, time_grid.back()), grid.es,
            ↪  steps.back());
44         res.norm_residual = (q-steps.back()).norm() / q.size();
45     });
46     return res;
47 }
48
49 //--------------------------------------------------------------------------
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(f);
60     }
61
62     void finish(void) {
63         int counter = 0;
64         for (auto i = mf.rbegin(); i != mf.rend(); ++i) {
65             if (counter % (mf.size()/10000 + 1) == 0)
```

```
                write_percent(double(counter)/mf.size());
            auto value = i->second.get();
            m[i->first] = value;
            counter++;
        }
        cout << "\r       \r";
    }

    auto begin(void) { return m.begin(); }
    auto end(void) { return m.end(); }

    Ret& operator[](const Key& key) { return m[key]; }
    const Ret& operator[](const Key& key) const { return m[key]; }
private:
    map<Key, future<Ret>> mf;
    map<Key, Ret> m;
};

//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
template<class ForwardIt, class GetValue>
double max_element_ignore_nan(ForwardIt first, ForwardIt last, GetValue get) {
    return get(*max_element(first, last, [get] (auto& a, auto& b) -> bool {
        if (isnan(get(a)))
            return true;
        else
            return get(a) < get(b);
    }));
}

//----------------------------------------------------------------------------
void investigate_t_changing(
    int n,
    const string& filename,
    const function<fem_result_t(double)>& ft
) {
    auto uniform_value = ft(0);

    async_performer_t<fem_result_t, int> performer;

    grid_generator_t grid(-1, 1, n);
    for (int i = 0; i < grid.size(); i++) {
        performer.add([i, ft, grid] () -> fem_result_t {
            return ft(grid(i));
        }, i);
    }

    performer.finish();

    int counter = 0;
    auto integral_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto&
    ↪  a) -> double { return a.second.integral_residual; });
    auto norm_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto& a)
    ↪  -> double { return a.second.norm_residual; });

    ofstream fout(filename + ".txt");
    fout << "t\tintegral\tnorm\tuniform_integral\tuniform_norm\ttime" << endl;
    for (int i = 0; i < grid.size(); i++) {
        auto v = performer[i];
        fout
            << grid(i) << "\t"
            << (isnan(v.integral_residual) ? integral_residual_max : v.integral_residual) << "\t"
            << (isnan(v.norm_residual) ? norm_residual_max : v.norm_residual) << "\t"
            << uniform_value.integral_residual << "\t"
            << uniform_value.norm_residual << "\t"
            << v.time << endl;
    }
    fout.close();
}

//----------------------------------------------------------------------------
void investigate_t2_changing(
    int n,
    const string& filename,
    const function<fem_result_t(double, double)>& ft
) {
    async_performer_t<fem_result_t, pair<int, int>> performer;

    grid_generator_t grid(-1, 1, n);
    for (int i = 0; i < grid.size(); i++) {
        for (int j = 0; j < grid.size(); j++) {
            performer.add([i, j, ft, grid] () -> fem_result_t {
                return ft(grid(i), grid(j));
            }, {i, j});
        }
```

```
    }

    performer.finish();

    auto integral_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto&
    ↪  a) -> double { return a.second.integral_residual; });
    auto norm_residual_max = max_element_ignore_nan(performer.begin(), performer.end(), [] (auto& a)
    ↪  -> double { return a.second.norm_residual; });

    ofstream fout(filename + ".integral.txt");
    ofstream fout2(filename + ".norm.txt");
    ofstream fout3(filename + ".time.txt");
    int last_line = 0;
    for (int i = 0; i < grid.size(); i++) {
        for (int j = 0; j < grid.size(); j++) {
            auto v = performer[{i, j}];
            fout << (isnan(v.integral_residual) ? integral_residual_max : v.integral_residual) << "\t";
            fout2 << (isnan(v.norm_residual) ? norm_residual_max : v.norm_residual) << "\t";
            fout3 << v.time << "\t";
        }
        fout << endl;
        fout2 << endl;
        fout3 << endl;
    }
    fout.close();
    fout2.close();

    fout.open(filename + ".x.txt");
    for (int i = 0; i < grid.size(); i++)
        fout << grid(i) << endl;
    fout.close();

    fout.open(filename + ".y.txt");
    for (int i = 0; i < grid.size(); i++)
        fout << grid(i) << endl;
    fout.close();
}

//------------------------------------------------------------------------------
void investigate_functions(
    const string& filename,
    const function<fem_result_t(const function_3d_t&)>& f
) {
    vector<pair<function_3d_t, string>> spaces, times;

    spaces.push_back({[] (double x, double y, double t) -> double { return 1; }, "$1$"});
    spaces.push_back({[] (double x, double y, double t) -> double { return x+y; }, "$x+y$"});
    spaces.push_back({[] (double x, double y, double t) -> double { return x*x+y*y; }, "$x^2+y$"});
    spaces.push_back({[] (double x, double y, double t) -> double { return x*x*y+y*y*y; },
    ↪  "$x^2y+y^3$"});
    spaces.push_back({[] (double x, double y, double t) -> double { return x*y*y; }, "$xy^2$"});
    spaces.push_back({[] (double x, double y, double t) -> double { return x*x*x*x+y*y*y*y; },
    ↪  "$x^4+y^4$"});
    spaces.push_back({[] (double x, double y, double t) -> double { return exp(x*y); }, "$e^{xy}$"});

    times.push_back({[] (double x, double y, double t) -> double { return 0; }, "$0$"});
    times.push_back({[] (double x, double y, double t) -> double { return t; }, "$t$"});
    times.push_back({[] (double x, double y, double t) -> double { return t*t; }, "$t^2$"});
    times.push_back({[] (double x, double y, double t) -> double { return t*t*t; }, "$t^3$"});
    times.push_back({[] (double x, double y, double t) -> double { return t*t*t; }, "$t^4$"});
    times.push_back({[] (double x, double y, double t) -> double { return exp(t); }, "$e^t$"});

    async_performer_t<fem_result_t, pair<string, string>> performer;

    for (auto& i : spaces) {
        for (auto& j : times) {
            performer.add([i, j, &f]() -> fem_result_t {
                return f(function_3d_t([&] (double x, double y, double t) -> double { return
                ↪  i.first(x, y, t) + j.first(x, y, t); }));
            }, {i.second, j.second});
        }
    }

    performer.finish();

    ofstream fout(filename);
    fout << "a\t";
    for (auto& i : times)
        fout << i.second << "\t";
    for (auto& i : spaces) {
        fout << endl << i.second << "\t";
        for (auto& j : times) {
            auto v = performer[{i.second, j.second}];
            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";
        }
```

```
    }
    fout.close();
}

//-------------------------------------------------------------------------------
void investigate_grid_changing(
    const string& filename,
    const function<pair<fem_result_t, int>(int)>& fi,
    int n
) {
    async_performer_t<pair<fem_result_t, int>, int> performer;

    for (int i = 0; i < n; i+=3) {
        performer.add([i, fi] () -> pair<fem_result_t, int> {
            return fi(i);
        }, i);
    }

    performer.finish();

    ofstream fout(filename + ".txt");
    fout << "i\tintegral\tnorm\ttime" << endl;
    for (int i = 0; i < n; i+=3) {
        auto v = performer[i];
        fout
            << v.second << "\t"
            << v.first.integral_residual << "\t"
            << v.first.norm_residual << "\t"
            << v.first.time << endl;
    }
    fout.close();
}

//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------

int main() {
    cout << calc_time_microseconds([](){
        investigate_grid_changing(
            "space_sgrid",
            [] (int sz) -> pair<fem_result_t, int> {
                return {
                    calc_fem_residual(
                        [] (double x, double y, double t) -> double { return exp(x*y) + exp(t*t); },
                        grid_generator_t(0, 1, 5+sz),
                        grid_generator_t(0, 1, 5+sz),
                        grid_generator_t(0, 1, 300)
                    ),
                    (7+sz)*(7+sz)
                };
            },
            70
        );

        investigate_grid_changing(
            "time_sgrid",
            [] (int sz) -> pair<fem_result_t, int> {
                return {
                    calc_fem_residual(
                        [] (double x, double y, double t) -> double { return exp(x*y) + exp(t*t); },
                        grid_generator_t(0, 1, 20),
                        grid_generator_t(0, 1, 20),
                        grid_generator_t(0, 1, 1+sz)
                    ),
                    3+sz
                };
            },
            500
        );

        investigate_functions(
            "functions_table_10_10_10.txt",
            [] (const function_3d_t& u) -> fem_result_t {
                return calc_fem_residual(u, grid_generator_t(0, 1, 10), grid_generator_t(0, 1, 10),
                ↪  grid_generator_t(0, 1, 10));
            }
        );

        investigate_functions(
            "functions_table_50_50_50.txt",
            [](const function_3d_t& u) ->  fem_result_t {
                return calc_fem_residual(u, grid_generator_t(0, 1, 50), grid_generator_t(0, 1, 50),
                ↪  grid_generator_t(0, 1, 50));
            }
        );

        vector<pair<function_3d_t, int>> u_space_mas;
```

```
        u_space_mas.push_back({[] (double x, double y, double t) -> double { return x*x + y*y + t*t; }, 0});
        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});
        u_space_mas.push_back({[] (double x, double y, double t) -> double { return exp(x*y) + exp(t*t); }, 2});
        u_space_mas.push_back({[] (double x, double y, double t) -> double { return exp((1-x)*(1-y)) + exp((1-t)*(1-t)); }, 3});
        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});

        for (auto& i : u_space_mas) {
            auto& u = i.first;
            investigate_t_changing(
                750,
                "time_tgrid_" + to_string(i.second),
                [u] (double tt) -> fem_result_t {
                    return calc_fem_residual(u, grid_generator_t(0, 1, 10), grid_generator_t(0, 1, 10), grid_generator_t(0, 1, 10, tt));
                }
            );

            investigate_t2_changing(
                75,
                "space_tgrid_" + to_string(i.second),
                [u] (double tx, double ty) -> fem_result_t {
                    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));
                }
            );
        }
    })/1000/1000 << "s" << endl;
    system("pause");
}
```

5.4 Визуализация

FILE **plot.py**

```
import math
import pylab
import numpy
import sys
import matplotlib.pyplot as plt
import matplotlib.lines as lines
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import ticker, cm

DPI = 200

def make_image(xpath, ypath, zpath, resultpath, mytitle):
    x = numpy.loadtxt(xpath)
    y = numpy.loadtxt(ypath)
    z = numpy.loadtxt(zpath)

    X, Y = np.meshgrid(x, y)

    fig, ax = plt.subplots()
    #locator=ticker.LogLocator(base=math.pow(10, 1/10000))
    cs = ax.contourf(X, Y, z, 55, cmap=cm.coolwarm)
    cbar = fig.colorbar(cs)

    plt.title(mytitle, fontsize=19)
    plt.xlabel(r'$t_x$', fontsize=15)
    plt.ylabel(r'$t_y$', fontsize=15)
    plt.tick_params(axis='both', labelsize=10)
    plt.grid(alpha=0.25)
    plt.savefig(resultpath, dpi=DPI)
    plt.clf()

def make_images(i):
    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");
    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");
    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");

```

```
if __name__ == '__main__':
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')

    make_images(0)
    make_images(1)
    make_images(2)
    make_images(3)
    make_images(4)
```