МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | **Факультет прикладной математики и информатики**

Кафедра прикладной математики

Курсовой проект
по дисциплине «Уравнения математической физики»

Группа ПМ-92 ИВАНОВ ВЛАДИСЛАВ

Вариант 7

Руководитель СОЛОВЕЙЧИК ЮРИЙ ГРИГОРЬЕВИЧ

Новосибирск, 2022

Содержание

1 Формулировка задачи

Реализовать МКЭ для гиперболического уравнения в декартовой системе координат. Схема Кранка-Николсона для аппроксимации по времени, базисные функции билинейные на прямоугольниках.

2 Анализ и дискретизация

Требуется решить гиперболическое уравнение:

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

В двумерной декартовой системе координат оно имеет вид:

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

$$u|_S = u_s$$

Конечноэлементная аппроксимация достигается путем разложения функции по базису. В качестве базисных функций используются билинейные на прямоугольных элементах:

$$\psi_1(x,y) = X_1(x)Y_1(y) \quad \psi_2(x,y) = X_1(x)Y_2(y)$$
$$\psi_3(x,y) = X_2(x)Y_1(y) \quad \psi_4(x,y) = X_2(x)Y_2(y)$$

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

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

Элементы локальных матриц будем вычислять аналитически, используя следующие формулы:

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

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

$$b_i = \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} f \psi_i \, \mathrm{d}x\, \mathrm{d}y$$

В случае с билинейными базисными функциями формулы можно представить в более простом виде:

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

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

$$\mathbf{M}^{\gamma} = \bar{\gamma}\mathbf{C}$$

$$\mathbf{b} = \mathbf{C} \cdot \mathbf{f}$$

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

Схема Кранка-Николсона для аппроксимации по времени:

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

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

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

Подставляя это в уравнение Галеркина и учитывая, что параметры γ, σ и χ являются константами, получаем глобальную систему уравнений:

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

Для неравномерной сетки производятся следующие замены:

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

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

$$\frac{\partial^2 u}{\partial t^2}=2\frac{u^j-u^{j-1}\dfrac{t_0-t_2}{t_1-t_2}+u^{j-2}\dfrac{t_0-t_1}{t_1-t_2}}{t_0\left(t_0-t_1-t_2\right)+t_1t_2}=\frac{u^j-u^{j-1}m_1+u^{j-2}m_2}{d_2}$$

$$d_1=t_0-t_2,\quad d_2=\frac{t_0\left(t_0-t_1-t_2\right)+t_1t_2}{2},\quad m_1=\frac{t_0-t_2}{t_1-t_2},\quad m_2=\frac{t_0-t_1}{t_1-t_2}$$

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

Полученный в результате решения данной системы вектор q будет являться решением поставленной задачи.

3 Исследования

Параметры, используемые во всех исследованиях:

- $\sigma : 1$
- $\lambda : 1$
- $\gamma : 1$
- $\chi : 1$
- Область : $[0,1] * [0,1]$
- Отрезок времени : $[0,1]$
- Количество узлов : 36

3.1 Сходимость на различных функциях

3.1.1 Равномерное пространство, равномерное время

	1	t	t^2	t^3	t^4	t^5	e^t	$cos(t)$
1	5.59e-17	4.28e-13	2.48e-12	3.85e-03	2.70e-02	5.97e-02	2.44e-03	9.89e-04
$x+y$	2.72e-12	2.09e-12	2.76e-12	3.85e-03	2.70e-02	5.97e-02	2.44e-03	9.89e-04
x^2+y^2	1.74e-12	8.28e-13	4.75e-13	3.85e-03	2.70e-02	5.97e-02	2.44e-03	9.89e-04
x^3+y^3	1.42e-12	3.97e-13	8.11e-13	3.85e-03	2.70e-02	5.97e-02	2.44e-03	9.89e-04
x^4+y^4	1.23e-03	1.23e-03	1.23e-03	2.62e-03	2.58e-02	5.85e-02	1.21e-03	2.45e-04
x^5+y^5	3.14e-03	3.14e-03	3.14e-03	9.59e-04	2.39e-02	5.67e-02	8.59e-04	2.17e-03
e^x+e^y	8.74e-05	8.74e-05	8.74e-05	3.77e-03	2.69e-02	5.97e-02	2.35e-03	9.02e-04

3.1.2 Равномерное пространство, неравномерное время

	1	t	t^2	t^3	t^4	t^5	e^t	$cos(t)$
1	3.20e-17	4.33e-13	2.51e-12	4.27e-03	2.80e-02	6.14e-02	2.57e-03	1.03e-03
$x+y$	2.74e-12	2.11e-12	2.79e-12	4.27e-03	2.80e-02	6.14e-02	2.57e-03	1.03e-03
x^2+y^2	1.76e-12	8.35e-13	4.79e-13	4.27e-03	2.80e-02	6.14e-02	2.57e-03	1.03e-03
x^3+y^3	1.43e-12	3.99e-13	8.18e-13	4.27e-03	2.80e-02	6.14e-02	2.57e-03	1.03e-03
x^4+y^4	1.25e-03	1.25e-03	1.25e-03	3.02e-03	2.68e-02	6.02e-02	1.32e-03	2.19e-04
x^5+y^5	3.17e-03	3.17e-03	3.17e-03	1.29e-03	2.49e-02	5.83e-02	7.92e-04	2.17e-03
e^x+e^y	8.84e-05	8.84e-05	8.84e-05	4.18e-03	2.79e-02	6.14e-02	2.48e-03	9.41e-04

3.1.3 Неравномерное пространство, равномерное время

	1	t	t^2	t^3	t^4	t^5	e^t	$cos(t)$
1	4.14e-17	4.25e-13	2.46e-12	3.82e-03	2.68e-02	5.92e-02	2.42e-03	9.80e-04
$x+y$	2.84e-12	6.24e-13	1.65e-12	3.82e-03	2.68e-02	5.92e-02	2.42e-03	9.80e-04
x^2+y^2	3.31e-12	7.45e-13	6.54e-13	3.82e-03	2.68e-02	5.92e-02	2.42e-03	9.80e-04
x^3+y^3	7.37e-13	3.52e-13	1.29e-12	3.82e-03	2.68e-02	5.92e-02	2.42e-03	9.80e-04
x^4+y^4	1.28e-03	1.28e-03	1.28e-03	2.55e-03	2.55e-02	5.80e-02	1.15e-03	3.12e-04
x^5+y^5	3.54e-03	3.54e-03	3.54e-03	9.52e-04	2.33e-02	5.58e-02	1.33e-03	2.60e-03
e^x+e^y	9.47e-05	9.47e-05	9.47e-05	3.73e-03	2.67e-02	5.91e-02	2.33e-03	8.87e-04

3.1.4 Неравномерное пространство, неравномерное время

	1	t	t^2	t^3	t^4	t^5	e^t	$cos(t)$
1	6.70e-17	4.30e-13	2.49e-12	4.23e-03	2.78e-02	6.09e-02	2.55e-03	1.02e-03
$x+y$	2.87e-12	6.28e-13	1.66e-12	4.23e-03	2.78e-02	6.09e-02	2.55e-03	1.02e-03
x^2+y^2	3.35e-12	7.52e-13	6.60e-13	4.23e-03	2.78e-02	6.09e-02	2.55e-03	1.02e-03
x^3+y^3	7.43e-13	3.55e-13	1.30e-12	4.23e-03	2.78e-02	6.09e-02	2.55e-03	1.02e-03
x^4+y^4	3.49e-17	1.29e-03	1.29e-03	2.95e-03	2.65e-02	5.96e-02	1.27e-03	2.89e-04
x^5+y^5	3.58e-03	3.58e-03	3.58e-03	1.16e-03	2.43e-02	5.74e-02	1.27e-03	2.60e-03
e^x+e^y	9.58e-05	9.58e-05	9.58e-05	4.14e-03	2.77e-02	6.08e-02	2.46e-03	9.26e-04

3.2 Точность решения при дроблении сетки

Параметры, используемые в данном исследовании:

- Шаг для неравномерных сеток (k) : 1.1
- Функция $u(t)$: 1

3.2.1 $u = 1$

i	$nodes$	$norm$
0	36	5.59e-17
1	121	4.61e-17
2	441	1.17e-16

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	3.20e-17
1	121	3.75e-17
2	441	3.12e-16

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	4.14e-17
1	121	9.80e-17
2	441	1.19e-16

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	6.70e-17
1	121	9.68e-17
2	441	6.64e-17

неравномерное пространство, неравномерное время

3.2.2 $u = x$

i	$nodes$	$norm$
0	36	5.64e-13
1	121	2.50e-13
2	441	1.64e-13

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	5.69e-13
1	121	2.51e-13
2	441	1.63e-13

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	5.91e-13
1	121	1.77e-13
2	441	1.10e-13

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	5.96e-13
1	121	1.76e-13
2	441	1.10e-13

неравномерное пространство, неравномерное время

3.2.3 $u = x^2$

i	$nodes$	$norm$
0	36	1.34e-12
1	121	7.76e-13
2	441	3.74e-13

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	1.36e-12
1	121	7.80e-13
2	441	3.71e-13

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	3.09e-13
1	121	8.38e-14
2	441	1.01e-13

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	3.11e-13
1	121	8.36e-14
2	441	1.01e-13

неравномерное пространство, неравномерное время

3.2.4 $u = x^3$

i	$nodes$	$norm$
0	36	5.47e-13
1	121	1.75e-13
2	441	9.29e-14

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	5.51e-13
1	121	1.75e-13
2	441	9.24e-14

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	4.96e-13
1	121	3.14e-13
2	441	7.44e-14

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	5.00e-13
1	121	3.15e-13
2	441	7.43e-14

неравномерное пространство, неравномерное время

3.2.5 $u = x^4$

i	$nodes$	$norm$
0	36	3.41e-04
1	121	7.59e-05
2	441	1.15e-05

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	3.44e-04
1	121	7.63e-05
2	441	1.14e-05

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	3.54e-04
1	121	1.16e-04
2	441	2.34e-05

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	3.57e-04
1	121	1.16e-04
2	441	2.32e-05

неравномерное пространство, неравномерное время

3.2.6 $u = x^5$

i	$nodes$	$norm$
0	36	8.77e-04
1	121	1.94e-04
2	441	2.91e-05

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	8.85e-04
1	121	1.95e-04
2	441	2.89e-05

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	1.00e-03
1	121	3.70e-04
2	441	7.60e-05

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	1.01e-03
1	121	3.71e-04
2	441	7.54e-05

неравномерное пространство, неравномерное время

3.2.7 $u = e^x$

i	$nodes$	$norm$
0	36	2.42e-05
1	121	5.37e-06
2	441	8.12e-07

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	2.44e-05
1	121	5.40e-06
2	441	8.06e-07

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	2.65e-05
1	121	9.29e-06
2	441	1.90e-06

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	2.67e-05
1	121	9.33e-06
2	441	1.88e-06

неравномерное пространство, неравномерное время

3.2.8 $u = cos(x)$

i	$nodes$	$norm$
0	36	1.22e-05
1	121	2.71e-06
2	441	4.11e-07

равномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	1.23e-05
1	121	2.73e-06
2	441	4.08e-07

равномерное пространство, неравномерное время

i	$nodes$	$norm$
0	36	1.23e-05
1	121	3.80e-06
2	441	7.65e-07

неравномерное пространство, равномерное время

i	$nodes$	$norm$
0	36	1.24e-05
1	121	3.82e-06
2	441	7.59e-07

неравномерное пространство, неравномерное время

4 Код программы

4.1 main.cpp

```cpp
#include "common.h"

void main() {
    lambda = 1;
    gamma = 1;
    sigma = 1;
    chi = 1;

    u_x = { [](double x, double y, double t) -> double {
        return 1;
    }};

    vector <func3D> u_t(8);
    u_t[0] = {[](double x, double y, double t) -> double { return 1; }};
    u_t[1] = {[](double x, double y, double t) -> double { return t; }};
    u_t[2] = {[](double x, double y, double t) -> double { return pow(t, 2); }};
    u_t[3] = {[](double x, double y, double t) -> double { return pow(t, 3); }};
    u_t[4] = {[](double x, double y, double t) -> double { return pow(t, 4); }};
    u_t[5] = {[](double x, double y, double t) -> double { return pow(t, 5); }};
    u_t[6] = {[](double x, double y, double t) -> double { return exp(t); }};
    u_t[7] = {[](double x, double y, double t) -> double { return cos(t); }};

    u = u_sum(u_x, u_t[0], lambda, gamma, sigma, chi);
    f = rightPart(u, lambda, gamma, sigma, chi);

    cout << "Is the grid uniform? (1/0)\n";
    cin >> answer;
```

```
    if (answer == 1)
        gridUniformFlag = true;
    else
        gridUniformFlag = false;
    cout << "Is the time uniform? (1/0)\n";
    cin >> answer;
    if (answer == 1)
        timeUniformFlag = true;
    else
        timeUniformFlag = false;
    cout << "\n";
    cout << "i nodes norm\n";
    cout << scientific << setprecision(2);

    for (coef = 0; coef < 3; coef++) {
        gridDivCoef = timeDivCoef = coef;
        readGrid();
        makeGridSpace(gridUniformFlag);
        readTime();
        makeGridTime(timeUniformFlag);
        cout << coef << " " << nodesCount << "\t" << solve() << endl;
    }
    cout << "\n";

    gridDivCoef = timeDivCoef = 0;
    for (size_t i = 0; i < u_t.size(); i++) {
        u = u_sum(u_x, u_t[i], lambda, gamma, sigma, chi);
        f = rightPart(u, lambda, gamma, sigma, chi);
        readGrid();
        makeGridSpace(gridUniformFlag);
        readTime();
        makeGridTime(timeUniformFlag);
        cout << "u(t)_" << i + 1 << ": " << solve() << "\n";
    }
}

void crankNicolson(int layer) {
    A.clear();
    A.resize(nodesCount);
    for (size_t i = 0; i < nodesCount; i++)
        A[i].resize(nodesCount, 0);

    makeGlobalG();
    makeGlobalM();
    b = makeGlobalb(layer);

    t0 = times[layer];
    t1 = times[layer - 1];
    t2 = times[layer - 2];
```

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

    double tmp = (gamma / 2.0) + (sigma / d1) + (chi / d2);
    for (size_t i = 0; i < nodesCount; i++)
        for (size_t j = 0; j < nodesCount; j++)
            A[i][j] += (G[i][j] / 2.0) + M[i][j] * tmp;
    b = ((b + b2) / 2.0) - (G * q2 / 2.0) +
        M * (q1 * (m1 * chi / d2) + q2 * (-(gamma / 2.0) + (sigma / d1)
        ↪  - (m2 * chi / d2)));

    for (size_t i = 0; i < nodesCount; i++)
        if (nodes[i].type == 1) {
            A[i].clear();
            A[i].resize(nodesCount, 0);
            A[i][i] = 1;
            b[i] = u(nodes[i].x, nodes[i].y, t0);
        }
}

double solve() {
    n = nodesCount;
    q1.resize(nodesCount);
    q2.resize(nodesCount);

    for (size_t i = 0; i < gridHeight; i++)
        for (size_t j = 0; j < gridWidth; j++) {
            k = i * gridWidth + j;
            q2[k] = u(nodes[k].x, nodes[k].y, times[0]);
            q1[k] = u(nodes[k].x, nodes[k].y, times[1]);
        }

    b2 = makeGlobalb(0);
    b1 = makeGlobalb(1);

    for (size_t layer = 2; layer < times.size(); layer++) {
        crankNicolson(layer);

        for (int j = 0; j < A.size(); ++j)
            for (int i = j + 1; i < A.size(); ++i) {
                tmp = A[i][j] / A[j][j];
                for (int k = 0; k < A.size(); ++k)
                    A[i][k] -= tmp * A[j][k];
                b[i] -= tmp * b[j];
            }

        vector <double> x;
```

```
        x.resize(A.size(), 0);

        for (int i = n - 1; i >= 0; --i) {
            tmp = 0.0;
            for (int j = i + 1; j < A.size(); ++j)
                tmp += A[i][j] * x[j];
            x[i] = (b[i] - tmp) / A[i][i];
        }

        b2 = b1;
        b1 = b;
        q = x;
        q2 = q1;
        q1 = q;
    }
    return normInNodes(q, t0);
}

void makeLocalG(int elemNum) {
    hx = nodes[elemNum + 1].x - nodes[elemNum].x;
    hy = nodes[elemNum + gridWidth].y - nodes[elemNum].y;
    GLocal = {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}};

    matTmp = {{2, -2, 1, -1}, {-2, 2, -1, 1}, {1, -1, 2, -2}, {-1, 1,
     ↪  -2, 2}};
    for (size_t i = 0; i < 4; i++)
        for (size_t j = 0; j < 4; j++)
            GLocal[i][j] += matTmp[i][j] * lambda * hy / (6 * hx);

    matTmp = {{2, 1, -2, -1}, {1, 2, -1, -2}, {-2, -1, 2, 1}, {-1, -2,
     ↪  1, 2}};
    for (size_t i = 0; i < 4; i++)
        for (size_t j = 0; j < 4; j++)
            GLocal[i][j] += matTmp[i][j] * lambda * hx / (6 * hy);
}

void makeLocalM(int elemNum) {
    hx = nodes[elemNum + 1].x - nodes[elemNum].x;
    hy = nodes[elemNum + gridWidth].y - nodes[elemNum].y;
    MLocal = {{4, 2, 2, 1}, {2, 4, 1, 2}, {2, 1, 4, 2}, {1, 2, 2, 4}};
    for (size_t i = 0; i < 4; i++)
        for (size_t j = 0; j < 4; j++)
            MLocal[i][j] *= hx * hy / 36;
}

void makeLocalb(int elemNum) {
    hx = nodes[elemNum + 1].x - nodes[elemNum].x;
    hy = nodes[elemNum + gridWidth].y - nodes[elemNum].y;
    bLocal = {0, 0, 0, 0};
```

```
147         f1 = f(nodes[elemNum].x, nodes[elemNum].y, t);
148         f2 = f(nodes[elemNum + 1].x, nodes[elemNum + 1].y, t);
149         f3 = f(nodes[elemNum + gridWidth].x, nodes[elemNum + gridWidth].y,
            ↪  t);
150         f4 = f(nodes[elemNum + gridWidth + 1].x, nodes[elemNum + gridWidth
            ↪  + 1].y, t);

151         bLocal[0] = (hx * hy / 36) * (4 * f1 + 2 * f2 + 2 * f3 + 1 * f4);
152         bLocal[1] = (hx * hy / 36) * (2 * f1 + 4 * f2 + 1 * f3 + 2 * f4);
153         bLocal[2] = (hx * hy / 36) * (2 * f1 + 1 * f2 + 4 * f3 + 2 * f4);
154         bLocal[3] = (hx * hy / 36) * (1 * f1 + 2 * f2 + 2 * f3 + 4 * f4);
155     }

156     void makeGlobalG() {
157         G.clear();
158         G.resize(nodesCount);
159         for (size_t i = 0; i < nodesCount; i++)
160             G[i].resize(nodesCount, 0);

161         for (size_t i = 0; i < gridHeight - 1; i++)
162             for (size_t j = 0; j < gridWidth - 1; j++)
163             {
164                 k = i * gridWidth + j;
165                 makeLocalG(k);
166                 vector <int> nearestNodes = {k, k + 1, k + gridWidth, k +
                    ↪  gridWidth + 1};
167                 for (size_t i1 = 0; i1 < 4; i1++)
168                     for (size_t j1 = 0; j1 < 4; j1++)
169                         G[nearestNodes[i1]][nearestNodes[j1]] +=
                            ↪  GLocal[i1][j1];
170             }
171     }

172     void makeGlobalM() {
173         M.clear();
174         M.resize(nodesCount);
175         for (size_t i = 0; i < nodesCount; i++)
176             M[i].resize(nodesCount, 0);

177         for (size_t i = 0; i < gridHeight - 1; i++)
178             for (size_t j = 0; j < gridWidth - 1; j++)
179             {
180                 k = i * gridWidth + j;
181                 makeLocalM(k);
182                 vector <int> nearestNodes = {k, k + 1, k + gridWidth, k +
                    ↪  gridWidth + 1};
183                 for (size_t i1 = 0; i1 < 4; i1++)
184                     for (size_t j1 = 0; j1 < 4; j1++)
```

```
                    M[nearestNodes[i1]][nearestNodes[j1]] +=
                     ↪  MLocal[i1][j1];
        }
}

vect makeGlobalb(int layer) {
    t = times[layer];
    vectTmp.clear();
    vectTmp.resize(nodesCount, 0);

    for (size_t i = 0; i < gridHeight - 1; i++)
        for (size_t j = 0; j < gridWidth - 1; j++)
        {
            int k = i * gridWidth + j;
            makeLocalb(k);
            vector <int> nearestNodes = { k, k + 1, k + gridWidth, k +
             ↪  gridWidth + 1 };
            for (size_t i1 = 0; i1 < 4; i1++)
                vectTmp[nearestNodes[i1]] += bLocal[i1];
        }
    return vectTmp;
}

void readGrid() {
    string filepath;
    if (gridUniformFlag)
        filepath = "uniGrid.txt";
    else
        filepath = "irrGrid.txt";
    ifstream fin(filepath);
    fin >> xLeft >> xRight >> yUp >> yDown >> gridWidth >> gridHeight;
    if (!gridUniformFlag) {
        fin >> kx >> ky;
        nx = gridWidth - 1;
        ny = gridHeight - 1;
    }
    fin.close();
}

void readTime() {
    string filepath;
    if (timeUniformFlag)
        filepath = "uniTime.txt";
    else
        filepath = "irrTime.txt";
    ifstream fin(filepath);
    fin >> tFirst >> tLast >> tNum;
    if (!timeUniformFlag) {
        fin >> kt;
```

```
228         nt = tNum - 1;
229     }
230     fin.close();
231 }

232 void makeGridSpace(bool gridUniformFlag) {
233     if (gridUniformFlag) {
234         hx = ((xRight - xLeft) / double(gridWidth - 1)) / pow(2,
            ↪ gridDivCoef);
235         hy = ((yDown - yUp) / double(gridHeight - 1)) / pow(2,
            ↪ gridDivCoef);
236         if (gridDivCoef != 0) {
237             gridWidth = (gridWidth - 1) * pow(2, gridDivCoef) + 1;
238             gridHeight = (gridHeight - 1) * pow(2, gridDivCoef) + 1;
239         }
240     }
241     else {
242         if (gridDivCoef != 0) {
243             gridWidth = (gridWidth - 1) * pow(2, gridDivCoef) + 1;
244             gridHeight = (gridHeight - 1) * pow(2, gridDivCoef) + 1;
245             nx *= pow(2, gridDivCoef);
246             ny *= pow(2, gridDivCoef);
247             kx *= pow(kx, 1.0 / gridDivCoef);
248             ky *= pow(ky, 1.0 / gridDivCoef);
249         }
250         hx = (xRight - xLeft) * (1 - kx) / (1 - pow(kx, nx));
251         hy = (yDown - yUp) * (1 - ky) / (1 - pow(ky, ny));
252     }

253     nodesCount = gridWidth * gridHeight;
254     nodes.resize(nodesCount);

255     if (gridUniformFlag) {
256         size_t i, j, elem;
257         double x, y;

258         for (size_t j = 1; j < gridHeight - 1; j++)
259         {
260             i = 1;
261             for (size_t elem = j * gridWidth + 1; elem < (j + 1) *
                ↪ gridWidth - 1; elem++, i++)
262             {
263                 x = xLeft + hx * i;
264                 y = yUp + hy * j;
265                 nodes[elem].setNodes(x, y, i, j, 0, gridDivCoef);
266                 nodes[elem].borderNum = 0; // inner nodes
267             }
268         }
269         i = j = 0;
```

```
        y = yUp;
        for (size_t elem = 0; elem < gridWidth; elem++, i++)
        {
            x = xLeft + hx * i;
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 1; // lower border
        }
        i = gridWidth - 1;
        j = 1;
        x = xRight;
        for (size_t elem = 2 * gridWidth - 1; elem < gridWidth *
        ↪  gridHeight; elem += gridWidth, j++)
        {
            y = yUp + hy * j;
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 2; // right border
        }
        i = 0;
        j = gridHeight - 1;
        y = yDown;
        for (size_t elem = gridWidth * j; elem < (j + 1) * gridWidth -
        ↪  1; elem++, i++)
        {
            x = xLeft + hx * i;
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 3; // upper border
        }
        i = 0;
        j = 1;
        x = xLeft;
        for (size_t elem = gridWidth; elem < (gridHeight - 1) *
        ↪  gridWidth; elem += gridWidth, j++)
        {
            y = yUp + hy * j;
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 4; // left border
        }
    }
    else {
        size_t i, j, elem;
        double x, y;

        dy = hy * ky;
        y = yUp + hy;
        for (size_t j = 1; j < gridHeight - 1; j++, dy *= ky)
        {
            i = 1;
            dx = hx * kx;
            x = xLeft + hx;
```

```
            for (size_t elem = j * gridWidth + 1; elem < (j + 1) *
             ↪ gridWidth - 1; elem++, i++, dx *= kx)
            {
                nodes[elem].setNodes(x, y, i, j, 0, gridDivCoef);
                nodes[elem].borderNum = 0;
                x += dx;
            }
            y += dy;
        }
        i = j = 0;
        dx = hx;
        x = xLeft;
        y = yUp;
        for (size_t elem = 0; elem < gridWidth; elem++, i++, dx *= kx)
        {
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 1;
            x += dx;
        }
        i = gridWidth - 1;
        x = xRight;
        j = 1;
        dy = hy * ky;
        y = yUp + hy;
        for (size_t elem = 2 * gridWidth - 1; elem < gridWidth *
         ↪ gridHeight; elem += gridWidth, j++, dy *= ky)
        {
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 2;
            y += dy;
        }
        i = 0;
        dx = hx;
        x = xLeft;
        j = gridHeight - 1;
        y = yDown;
        for (size_t elem = gridWidth * j; elem < (j + 1) * gridWidth -
         ↪ 1; elem++, i++, dx *= kx)
        {
            nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
            nodes[elem].borderNum = 3;
            x += dx;
        }
        i = 0;
        x = xLeft;
        j = 1;
        dy = hy * ky;
        y = yUp + hy;
        for (size_t elem = gridWidth; elem < (gridHeight - 1) *
         ↪ gridWidth; elem += gridWidth, j++, dy *= ky)
```

```
361             {
362                 nodes[elem].setNodes(x, y, i, j, 1, gridDivCoef);
363                 nodes[elem].borderNum = 4;
364                 y += dy;
365             }
366         }
367     }
368     void makeGridTime(bool timeUniformFlag) {
369         if (timeUniformFlag) {
370             ht = ((tLast - tFirst) / double(tNum - 1)) / pow(2,
                ↪  timeDivCoef);
371             if (timeDivCoef != 0)
372                 tNum = (tNum - 1) * pow(2, timeDivCoef) + 1;
373             times.resize(tNum);
374             size_t i, elem;
375             double t;
376             times[0] = tFirst;
377             i = 1;
378             for (elem = 1; elem < tNum; elem++, i++)
379                 times[elem] = tFirst + ht * i;
380             times[tNum - 1] = tLast;
381         }
382         else {
383             if (timeDivCoef != 0) {
384                 tNum = (tNum - 1) * pow(2, timeDivCoef) + 1;
385                 nt *= pow(2, timeDivCoef);
386                 kt *= pow(kt, 1.0 / timeDivCoef);
387             }
388             times.resize(tNum);
389             ht = (tLast - tFirst) * (1 - kt) / (1 - pow(kt, nt));
390             double t;
391             size_t i, elem;
392             i = 1;
393             dt = ht * kt;
394             t = tFirst + ht;
395             times[0] = tFirst;
396             for (elem = 1; elem < tNum; elem++, i++, dt *= kt)
397             {
398                 times[elem] = t;
399                 t += dt;
400             }
401             times[tNum - 1] = tLast;
402         }
403     }
```

4.2 common.h

```cpp
#pragma once
#define _CRT_SECURE_NO_WARNINGS

#include <fstream>
#include <vector>
#include <functional>
#include <cmath>
#include <string>
#include <iostream>
#include <iomanip>

using namespace std;

typedef function <double(double)> func1D;
typedef function <double(double, double, double)> func3D;
typedef vector <double> vect;
typedef vector <vector <double>> mat;

const double h = 0.001;
bool gridUniformFlag, timeUniformFlag;
int i, k, n, gridWidth, gridHeight, gridDivCoef, timeDivCoef, coef,
 ↪  nodesCount, tNum, answer;
double sum, tmp, xLeft, xRight, yUp, yDown, tFirst, tLast, hx, nx, hy,
 ↪  ny, kx, ky, ht, nt, kt, dx, dy, dt, f1, f2, f3, f4, sigma, lambda,
 ↪  gamma, chi, t, t0, t1, t2, d1, d2, m1, m2;
vect x, b, b1, b2, bLocal, vectTmp, q, q1, q2, makeGlobalb(int layer);
mat A, G, M, GLocal, MLocal, matTmp;
func1D u_t, u_xx, u_yy, u_tt;
func3D u, u_x, f;
string fp;

void readGrid();
void readTime();
void makeGridSpace(bool);
void makeGridTime(bool);
void makeLocalG(int elemNum);
void makeLocalM(int elemNum);
void makeLocalb(int elemNum);
void makeGlobalG();
void makeGlobalM();
void crankNicolson(int layer);
double solve();

vect operator + (const vect& a, const vect& b) {
    vect result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] += b[i];
```

```
    return result;
}
vect operator - (const vect& a, const vect& b) {
    vect result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] -= b[i];
    return result;
}
vect operator * (const mat& a, const vect& b) {
    vect result;
    result.resize(b.size(), 0);
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i] += a[i][j] * b[j];
    return result;
}
vect operator * (const vect& a, double b) {
    vect result = a;
    for (int i = 0; i < result.size(); i++)
        result[i] *= b;
    return result;
}
double operator * (const vect& a, const vect& b) {
    sum = 0;
    for (int i = 0; i < a.size(); i++)
        sum += a[i] * b[i];
    return sum;
}
mat operator * (const mat& a, double b) {
    mat result = a;
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i][j] *= b;
    return result;
}
mat operator * (double b, const mat& a) {
    return operator*(a, b);
}
vect operator / (const vect& a, double b) {
    vect result = a;
    for (int i = 0; i < result.size(); i++)
        result[i] /= b;
    return result;
}

struct node {
    int i, j;
    int borderNum;
    int type = -99;
```

```
    double x, y;
    bool firstNodeFlag = false;

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

vector <node> nodes;
vect times;

double normInNodes(const vect& x, int time) {
    tmp = 0;
    for (size_t i = 0; i < x.size(); i++)
        tmp += pow((x[i] - u(nodes[i].x, nodes[i].y, time)), 2);
    return sqrt(tmp) / nodes.size();
}

func1D firstDerivative(const func1D& f) {
    return [f](double x) -> double {
        return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 *
        ↪  h)) / (12 * h);
    };
}

func1D secondDerivative(const func1D& f) {
    return [f](double x) -> double {
        return (-f(x + 2 * h) + 16 * f(x + h) - 30 * f(x) + 16 * f(x -
        ↪  h) - f(x - 2 * h)) / (12 * h * h);
    };
}

func3D rightPart(const func3D& u, double lambda, double gamma, double
↪  sigma, double chi) {
    return [=](double x, double y, double t) -> double {
        using namespace placeholders;
        u_t = firstDerivative(bind(u, x, y, _1));
        u_xx = secondDerivative(bind(u, _1, y, t));
        u_yy = secondDerivative(bind(u, x, _1, t));
        u_tt = secondDerivative(bind(u, x, y, _1));
        return -lambda * (u_xx(x) + u_yy(y)) + gamma * u(x, y, t) +
        ↪  sigma * u_t(t) + chi * u_tt(t);
    };
}

```

```
137 func3D u_sum(const func3D& u_x, const func3D& u_t, double lambda,
    ↪  double gamma, double sigma, double chi) {
138     return [=](double x, double y, double t) -> double {
139         return (u_x(x, y, t) + u_t(x, y, t));
140     };
141 }
```