



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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



Кафедра прикладной математики
Курсовая работа
по дисциплине «Уравнения математической физики»

МЕТОД КОНЕЧНЫХ ЭЛЕМЕНТОВ

Группа ПМ-21 ДУДКИНА МАРИЯ

Новосибирск, 2025

1. Задание

МКЭ для двумерной краевой задачи для параболического уравнения в декартовой системе координат, схема Кранка-Николсона для аппроксимации по времени. Базисные функции билинейные на прямоугольниках. Краевые условия всех типов. Коэффициент диффузии λ разложить по билинейным базисным функциям. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

2. Постановка задачи

Параболическая краевая задача для функции u определяется дифференциальным уравнением

$$\sigma \frac{\partial u}{\partial t} - \operatorname{div}(\lambda \operatorname{grad} u) = f$$

заданным в некоторой области Ω с границей $S = S_1 \cup S_2 \cup S_3$, и краевыми условиями

$$\begin{aligned} u|_{S_1} &= u_g, \\ \lambda \frac{\partial u}{\partial n}|_{S_2} &= \theta, \\ \lambda \frac{\partial u}{\partial n}|_{S_3} + \beta(u|_{S_3} - u_\beta) &= 0, \end{aligned}$$

в которых $u|_{S_i}$ — значение искомой функции u на границе S_i , а $\frac{\partial u}{\partial n}|_{S_i}$ — значение на S_i производной функции u по направлению внешней нормали к поверхности S_i , λ — коэффициент диффузии.

В декартовой системе координат уравнение имеет вид:

$$\sigma \frac{\partial u}{\partial t} - \frac{d}{dx} \left(\lambda \frac{du}{dx} \right) - \frac{d}{dy} \left(\lambda \frac{du}{dy} \right) = f$$

3. Теоретическая часть

3.1. Вариационная постановка в форме уравнения Галеркина

Решение u нужно искать не во всём пространстве H^2 , а в его сужении – пространстве функций, имеющих суммируемые с квадратом вторые производные и удовлетворяющие краевым условиям на границе S области Ω , которое мы будем обозначать H_S^2 .

Исходную краевую задачу можно представить в виде $Lu = f$, где оператор L – взаимно однозначное отображение H_S^2 в H_0 (т.е. для всякой функции $f \in H_0$ уравнения (1.1) имеет единственное решение $u \in H_S^2$).

Запишем эквивалентную вариационную постановку в форме уравнения Галеркина.

$$R(u) = \sigma \frac{\partial u}{\partial t} - \operatorname{div}(\lambda \operatorname{grad} u) - f = 0$$

Потребуем, чтобы невязка $R(u)$ дифференциального уравнения была ортогональна (в смысле скалярного произведения пространства $L_2(\Omega) \equiv H^0$) некоторому пространству Φ функции v , которое мы будем называть пространством пробных функций:

$$\int_{\Omega} \left(\sigma \frac{\partial u}{\partial t} - \operatorname{div}(\lambda \operatorname{grad} u) - f \right) v d\Omega = 0, \forall v \in \Phi$$

Воспользуемся формулой Грина для пространственных производных:

$$-\int_{\Omega} \left(\frac{\partial}{\partial x} \left(\lambda \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial u}{\partial y} \right) \right) d\Omega = \int_{\Omega} \lambda \operatorname{grad}(u) \operatorname{grad}(v) d\Omega - \int_S \lambda \frac{\partial u}{\partial n} v dS, \forall v \in \Phi$$

Преобразуем слагаемое $-\int_{\Omega} (\operatorname{div}(\lambda \operatorname{grad} u)) v d\Omega$ с использованием формулы Грина:

$$\int_{\Omega} \left(\sigma \frac{\partial u}{\partial t} \right) v d\Omega + \int_{\Omega} \lambda \operatorname{grad}(u) \operatorname{grad}(v) d\Omega = \int_S \lambda \frac{\partial u}{\partial n} v dS + \int_{\Omega} f v d\Omega, \forall v \in \Phi$$

Где $S = S_1 \cup S_2 \cup S_3$ как и ранее, граница Ω .

$$\int_S \lambda \frac{\partial u}{\partial n} v dS = \int_{S_1} \lambda \frac{\partial u}{\partial n} v dS + \int_{S_2} \lambda \frac{\partial u}{\partial n} v dS + \int_{S_3} \lambda \frac{\partial u}{\partial n} v dS$$

Воспользовавшись краевыми условиями, преобразуем интегралы S_2 и S_3 :

$$\int_S \lambda \frac{\partial u}{\partial n} v dS = \int_{S_1} \lambda \frac{\partial u}{\partial n} v dS + \int_{S_2} \Theta v dS + \int_{S_3} (\beta(u - u_{\beta})) v dS$$

В качестве Φ выберем H_1^0 – пространство пробных функций $v_0 \in H^1$, которые на границе S_1 удовлетворяют нулевым первым краевым условиям. С учётом этого уравнение примет вид:

$$\begin{aligned} \int_{\Omega} \left(\sigma \frac{\partial u}{\partial t} \right) v_0 d\Omega + \int_{\Omega} \lambda \operatorname{grad}(u) \operatorname{grad}(v_0) d\Omega = \\ = \int_{S_2} \Theta v_0 dS + \int_{S_3} (\beta(u - u_\beta)) v_0 dS + \int_{\Omega} f v_0 d\Omega, \forall v \in \Phi \end{aligned}$$

Итоговый вид уравнения:

$$\begin{aligned} \int_{\Omega} \left(\sigma \frac{\partial u}{\partial t} \right) v_0 d\Omega + \int_{\Omega} \lambda \operatorname{grad}(u) \operatorname{grad}(v_0) d\Omega + \int_{S_3} \beta u v_0 dS = \\ = \int_{\Omega} f v_0 d\Omega + \int_{S_2} \Theta v_0 dS + \int_{S_3} \beta u_\beta v_0 dS, \forall v_0 \in H_0^1 \end{aligned}$$

3.2. Конечноэлементная дискретизация и переход к локальным матрицам

При построении конечноэлементных аппроксимаций по методу Галеркина пространства H_g^1 и H_0^1 заменяются конечноизмерными пространствами V_g^h и V_0^h , которые являются элементами одного и того же конечноизмерного пространства V^h . В МКЭ базисом пространства V^h является набор финитных кусочно-полиномиальных функций $\psi_i, i = \overline{1, n}$.

На одном элементе каждая функция $\psi_i, i = \overline{1, n}$ равна 1 в своём узле и 0 в других элементах.

Получим аппроксимацию уравнения Галеркина на конечноизмерных подпространствах V_g^h и V_0^h , аппроксимирующих исходные пространства H_g^1 и H_0^1 . Для этого заменим в вариационном уравнении функцию $u \in H_g^1$ аппроксимирующей её функцией $u^h \in V_g^h$, а функцию $v_0 \in H_0^1$ – функцией $v_0^h \in V_0^h$:

$$\begin{aligned} \int_{\Omega} \left(\sigma \frac{\partial u^h}{\partial t} \right) v_0^h d\Omega + \int_{\Omega} \lambda \operatorname{grad}(u^h) \operatorname{grad}(v_0^h) d\Omega + \int_{S_3} \beta u^h v_0^h dS = \\ = \int_{\Omega} f v_0^h d\Omega + \int_{S_2} \Theta v_0^h dS + \int_{S_3} \beta u_\beta v_0^h dS \end{aligned}$$

В декартовой системе координат:

$$\begin{aligned} \int_{\Omega} \left(\sigma \frac{\partial u^h}{\partial t} \right) v_0^h dx dy + \int_{\Omega} \lambda \left(\frac{\partial u^h}{\partial x} \frac{\partial v_0^h}{\partial x} + \frac{\partial u^h}{\partial y} \frac{\partial v_0^h}{\partial y} \right) dx dy + \int_{S_3} \beta u^h v_0^h dS = \\ = \int_{\Omega} f v_0^h dx dy + \int_{S_2} \Theta v_0^h dS + \int_{S_3} \beta u_\beta v_0^h dS \end{aligned}$$

Поскольку любая функция $v_0^h \in V_0^h$ может быть представлена в виде линейной комбинации

$$v_0^h = \sum_{i \in N_0} q_i^h \psi_i$$

Вариационное уравнение эквивалентно следующей системе уравнений:

$$\begin{aligned} \int_{\Omega} \left(\sigma \frac{\partial u^h}{\partial t} \right) \psi_i d\Omega + \int_{\Omega} \lambda \operatorname{grad}(u^h) \operatorname{grad}(\psi_i) d\Omega + \int_{S_3} \beta u^h \psi_i dS = \\ = \int_{\Omega} f \psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS, i \in N_0 \end{aligned}$$

N_0 – множество индексов i таких, что базисные функции ψ_i , пространства V^h являются и базисными функциями пространств V_g^h и V_0^h .

Так как $u^h \in V_g^h$, оно может быть представлено в виде линейной комбинации базисных функций пространства V^h :

$$u^h = \sum_{j=1}^n q_j \psi_j,$$

Причём $n - n_0$ компонент вектора весов $q = (q_1, \dots, q_1)^T$ должны быть фиксированы и могут быть определены из условия

$$u^h|_{S_1} = u_g.$$

Временной член может быть представлен:

$$\frac{\partial u^h}{\partial t} = \sum_{j=1}^n \frac{\partial q_j}{\partial t} \psi_j,$$

Подставляя формулы выше, получаем СЛАУ для компонент q_j вектора q с индексом $j \in N_0$:

$$\sum_{j=1}^n \left(\frac{\partial q_j}{\partial t} \int_{\Omega} \sigma \psi_i \psi_j d\Omega + \int_{\Omega} \lambda \operatorname{grad}(\psi_j) \operatorname{grad}(\psi_i) d\Omega + \int_{S_3} \beta \psi_j \psi_i dS \right) q_j = \int_{\Omega} f \psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS, i \in N_0$$

В декартовой системе координат:

$$\sum_{j=1}^n \left(\frac{\partial q_j}{\partial t} \int_{\Omega} \sigma \psi_i \psi_j d\Omega + \int_{\Omega} \lambda \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) d\Omega + \int_{S_3} \beta \psi_j \psi_i dS \right) q_j = \int_{\Omega} f \psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS, i \in N_0$$

Система уравнений содержит n_0 уравнений, в то время как вектор q имеет n компонент.

$$\sum_j q_j \psi_j \Big|_{S_1} = u_g$$

Конечноэлементная СЛАУ для вектора весов q может быть записана в матричном виде:

$$Aq = b,$$

Где компоненты матрицы A и вектора b определяются соотношением:

$$A_{ij} = \begin{cases} \frac{\partial q_j}{\partial t} \int_{\Omega} \sigma \psi_i \psi_j d\Omega + \int_{\Omega} \lambda \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) d\Omega + \int_{S_3} \beta \psi_j \psi_i dS, & i \in N_0 \\ \delta_{ij}, & i \notin N_0 \end{cases}$$

$$b_i = \begin{cases} \int_{\Omega} f \psi_i d\Omega + \int_{S_2} \theta \psi_i dS + \int_{S_3} \beta u_{\beta} \psi_i r dS, & i \in N_0 \\ u_g(x_i), & i \notin N_0 \end{cases}$$

В которых δ_{ij} – символ Кронекера ($\delta_{ii} = 1$ и $\delta_{ij} = 0$ при $i \neq j$).

3.3. Аналитическое выражение для вычисления локальных матриц

Исходное дифференциальное уравнение в декартовой системе координат для двумерной задачи примет вид:

$$\sigma \frac{\partial u}{\partial t} - \frac{d}{dx} \left(\lambda \frac{du}{dx} \right) - \frac{d}{dy} \left(\lambda \frac{du}{dy} \right) = f$$

3.3.1. Базисные функции

Рассмотрим аппроксимацию краевой задачи на прямоугольной сетке. $\Omega_{ps} = [x_p, x_{p+1}] \times [y_p, y_{p+1}]$ (ячейки этой сетки строятся в виде прямого (декартова) произведения независимых друг от друга одномерных сеток).

На отрезке $[x_p, x_{p+1}]$ задаются две одномерные линейные функции:

$$X_1(x) = \frac{x_{p+1} - x}{h_x}, \quad X_2(x) = \frac{x - x_p}{h_x}, \quad h_x = x_{p+1} - x_p$$

Аналогично на интервале (y_s, y_{s+1}) задаются линейные функции:

$$Y_1(y) = \frac{y_{s+1} - y}{h_y}, \quad Y_2(y) = \frac{y - y_s}{h_y}, \quad h_y = y_{s+1} - y_s$$

Локальные базисные функции на конечном элементе $\Omega_{ps} = [x_p, x_{p+1}] \times [y_p, y_{p+1}]$ представляются в виде произведения функций:

$$\begin{aligned} \hat{\psi}_1(x, y) &= X_1(x)Y_1(y), & \hat{\psi}_2(x, y) &= X_2(x)Y_1(y) \\ \hat{\psi}_3(x, y) &= X_1(x)Y_2(y), & \hat{\psi}_4(x, y) &= X_2(x)Y_2(y) \end{aligned}$$

3.3.2. Локальные матрицы

Предварительно вычислим одномерные интегралы:

$$\int_{x_p}^{x_{p+1}} \left(\frac{dX_1}{dx} \right)^2 dx = \frac{1}{h_x}, \quad \int_{x_p}^{x_{p+1}} \frac{dX_1}{dx} \frac{dX_2}{dx} dx = -\frac{1}{h_x}$$

$$\int_{x_p}^{x_{p+1}} \left(\frac{dX_2}{dx} \right)^2 dx = \frac{1}{h_x}, \quad \int_{x_p}^{x_{p+1}} (X_1)^2 dx = \frac{h_x}{3}$$

$$\int_{x_p}^{x_{p+1}} (X_2)^2 dx = \frac{h_x}{3}, \quad \int_{x_p}^{x_{p+1}} X_1 X_2 dx = \frac{h_x}{6}$$

Аналогичный вид имеют и интегралы от произведений функций $Y_v(y)$ и их производных.

- **Матрица жёсткости**

Рассмотрим построение локальной матрицы жёсткости \hat{G} , предварительно заменив параметр λ на конечном элементе Ω_{ps} его усреднённым значением $\bar{\lambda}$. Тогда компоненты матрицы определяются выражением:

$$\begin{aligned}\hat{G}_{ij} &= \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} \bar{\lambda} \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) dx dy \\ \hat{G}_{11} &= \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \bar{\lambda} \left(\left(\frac{\partial \widehat{\psi}_1}{\partial x} \right)^2 + \left(\frac{\partial \widehat{\psi}_1}{\partial y} \right)^2 \right) dx dy = \\ &= \bar{\lambda} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \left(\left(\frac{dX_1}{dx} \right)^2 (Y_1)^2 + \left(\frac{dX_2}{dx} \right)^2 (Y_2)^2 \right) dx dy = \\ &= \bar{\lambda} \left(\int_{x_p}^{x_p+h_x} \left(\frac{dX_1}{dx} \right)^2 dx \int_{y_s}^{y_s+h_y} (Y_1)^2 dy + \int_{x_p}^{x_p+h_x} \left(\frac{dX_2}{dx} \right)^2 dx \int_{y_s}^{y_s+h_y} (Y_2)^2 dy \right) = \frac{\bar{\lambda}}{3} \left(\frac{h_y}{h_x} + \frac{h_x}{h_y} \right)\end{aligned}$$

$$\begin{aligned}\hat{G}_{12} &= \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \bar{\lambda} \left(\frac{\partial \psi_1}{\partial x} \frac{\partial \psi_2}{\partial x} + \frac{\partial \psi_1}{\partial y} \frac{\partial \psi_2}{\partial y} \right) dx dy = \\ &= \bar{\lambda} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \left(\frac{dX_1}{dx} Y_1 \frac{dX_2}{dx} Y_1 + X_1 \frac{dY_1}{dy} X_2 \frac{dY_1}{dy} \right) dx dy = \\ &= \bar{\lambda} \left(\int_{x_p}^{x_p+h_x} \frac{dX_1}{dx} \frac{dX_2}{dx} dx \int_{y_s}^{y_s+h_y} (Y_1)^2 dy + \int_{x_p}^{x_p+h_x} X_1 X_2 dx \int_{y_s}^{y_s+h_y} \left(\frac{dY_1}{dy} \right)^2 dy \right) \\ &= \frac{\bar{\lambda}}{3} \left(-\frac{h_y}{h_x} + \frac{h_x}{2h_y} \right)\end{aligned}$$

$$\begin{aligned}\hat{G}_{13} &= \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \bar{\lambda} \left(\frac{\partial \psi_1}{\partial x} \frac{\partial \psi_3}{\partial x} + \frac{\partial \psi_1}{\partial y} \frac{\partial \psi_3}{\partial y} \right) dx dy = \\ &= \bar{\lambda} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \left(\frac{dX_1}{dx} Y_1 \frac{dX_1}{dx} Y_2 + X_1 \frac{dY_1}{dy} X_1 \frac{dY_2}{dy} \right) dx dy = \\ &= \bar{\lambda} \left(\int_{x_p}^{x_p+h_x} \left(\frac{dX_1}{dx} \right)^2 dx \int_{y_s}^{y_s+h_y} Y_1 Y_2 dy + \int_{x_p}^{x_p+h_x} (X_1)^2 dx \int_{y_s}^{y_s+h_y} \frac{dY_1}{dy} \frac{dY_2}{dy} dy \right) = \frac{\bar{\lambda}}{3} \left(\frac{h_y}{2h_x} - \frac{h_x}{h_y} \right)\end{aligned}$$

$$\begin{aligned}\hat{G}_{14} &= \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \bar{\lambda} \left(\frac{\partial \psi_1}{\partial x} \frac{\partial \psi_4}{\partial x} + \frac{\partial \psi_1}{\partial y} \frac{\partial \psi_4}{\partial y} \right) dx dy = \\ &= \bar{\lambda} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} \left(\frac{dX_1}{dx} Y_1 \frac{dX_2}{dx} Y_2 + X_1 \frac{dY_1}{dy} X_2 \frac{dY_2}{dy} \right) dx dy = \\ &= \bar{\lambda} \left(\int_{x_p}^{x_p+h_x} \frac{dX_1}{dx} \frac{dX_2}{dx} dx \int_{y_s}^{y_s+h_y} Y_1 Y_2 dy + \int_{x_p}^{x_p+h_x} X_1 X_2 dx \int_{y_s}^{y_s+h_y} \frac{dY_1}{dy} \frac{dY_2}{dy} dy \right) \\ &= \frac{\bar{\lambda}}{6} \left(-\frac{h_y}{h_x} - \frac{h_x}{h_y} \right)\end{aligned}$$

и так далее.

В результате матрица жёсткости имеет вид:

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

Коэффициент диффузии λ на элементе Ω_m заменим билинейным интерполянтом:

$$\lambda = \sum_{k=1}^4 \hat{\lambda}_k \psi_k$$

Тогда компоненты матрицы будут определяться выражением:

$$\begin{aligned} \hat{G}_{ij} &= \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} \sum_{k=1}^4 \hat{\lambda}_k \psi_k \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) dx dy \\ &= \sum_{k=1}^4 \hat{\lambda}_k \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} \psi_k \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) dx dy \end{aligned}$$

○ Матрица масс

Предварительно заменим параметр σ на Ω_{ps} его осреднённым значением $\bar{\sigma}$. Тогда компоненты матрицы определяются следующим образом:

$$\hat{M}_{ij} = \int_{x_p}^{x_{p+1}} \int_{y_s}^{y_{s+1}} \bar{\sigma} \hat{\psi}_i \hat{\psi}_j dx dy$$

$$\hat{M}_{11} = \bar{\sigma} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} X_1^2 Y_1^2 dx dy = \sigma \int_{x_p}^{x_p+h_x} X_1^2 dx \int_{y_s}^{y_s+h_y} Y_1^2 dy = \bar{\sigma} \frac{h_x h_y}{9}$$

$$\hat{M}_{12} = \bar{\sigma} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} X_1 X_2 Y_1^2 dx dy = \bar{\sigma} \int_{x_p}^{x_p+h_x} X_1 X_2 dx \int_{y_s}^{y_s+h_y} Y_1^2 dy = \bar{\sigma} \frac{h_x h_y}{18}$$

$$\hat{M}_{13} = \bar{\sigma} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} X_1^2 Y_1 Y_2 dx dy = \bar{\sigma} \int_{x_p}^{x_p+h_x} X_1^2 dx \int_{y_s}^{y_s+h_y} Y_1 Y_2 dy = \bar{\sigma} \frac{h_x h_y}{18}$$

$$\hat{M}_{14} = \bar{\sigma} \int_{x_p}^{x_p+h_x} \int_{y_s}^{y_s+h_y} X_1 X_2 Y_1 Y_2 dx dy = \bar{\sigma} \int_{x_p}^{x_p+h_x} X_1 X_2 dx \int_{y_s}^{y_s+h_y} Y_1 Y_2 dy = \bar{\sigma} \frac{h_x h_y}{36}$$

И так далее.

В результате матрица масс имеет вид:

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

- **Вектор правой части**

Локальный вектор правой части \hat{b} конечного элемента Ω_{ps} вычислим, представив на Ω_{ps} правую часть f дифференциального уравнения в виде билинейного интерполянта $\sum_{v=1}^4 \hat{f}_v \hat{\psi}_v$. Локальный вектор \hat{b} в этом случае легко вычисляется через матрицу \hat{C} .

$$\hat{b} = \hat{C} \hat{f} = \frac{h_x h_y}{36} \begin{pmatrix} 4\hat{f}_1 + 2\hat{f}_2 + 2\hat{f}_3 + \hat{f}_4 \\ 2\hat{f}_1 + 4\hat{f}_2 + \hat{f}_3 + 2\hat{f}_4 \\ 2\hat{f}_1 + \hat{f}_2 + 4\hat{f}_3 + 2\hat{f}_4 \\ \hat{f}_1 + 2\hat{f}_2 + 2\hat{f}_3 + 4\hat{f}_4 \end{pmatrix}$$

3.4. Краевые условия

Краевые условия третьего и второго рода также могут быть представлены в виде локальных матриц и векторов рёбер, на которых заданы эти краевые условия. При этом на каждом ребре ненулевым являются только две базисные функции, причём они линейны на этом ребре.

3.4.1. Краевые условия третьего рода

Локальная матрица ребра Γ длины h с заданным на нём краевым условием третьего рода фактически является матрицей масс одномерного конечного элемента с линейными базисными функциями (считаем, что $\beta = \text{const}$ на Γ):

$$A^{S_3} = \begin{pmatrix} \int_{\Gamma} \beta (\hat{\psi}_1)^2 d\Gamma & \int_{\Gamma} \beta \hat{\psi}_1 \hat{\psi}_2 d\Gamma \\ \int_{\Gamma} \beta \hat{\psi}_1 \hat{\psi}_2 d\Gamma & \int_{\Gamma} \beta (\hat{\psi}_2)^2 d\Gamma \end{pmatrix} = \frac{\beta h}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Локальный вектор \hat{b}^{S_3} этого ребра Γ при представлении u_{β} на Γ в виде разложения по базисным функциям $\hat{\psi}_1$ и $\hat{\psi}_2$ имеет вид:

$$\hat{b}^{S_3} = \begin{pmatrix} \int_{\Gamma} \beta (\hat{u}_{\beta 1} \hat{\psi}_1 + \hat{u}_{\beta 2} \hat{\psi}_2) \hat{\psi}_1 d\Gamma \\ \int_{\Gamma} \beta (\hat{u}_{\beta 1} \hat{\psi}_1 + \hat{u}_{\beta 2} \hat{\psi}_2) \hat{\psi}_2 d\Gamma \end{pmatrix} = \frac{\beta h}{6} \begin{pmatrix} 2\hat{u}_{\beta 1} + \hat{u}_{\beta 2} \\ \hat{u}_{\beta 1} + 2\hat{u}_{\beta 2} \end{pmatrix}$$

3.4.2. Краевые условия второго рода

Локальный вектор \hat{b}^{S_2} этого ребра Γ длины h с заданным на нём краевым условием второго рода при представлении Θ на Γ в виде разложения по базисным функциям вычисляется аналогично \hat{b}^{S_3} :

$$\hat{b}^{S_2} = \frac{h}{6} \begin{pmatrix} 2\hat{\theta}_1 + \hat{\theta}_2 \\ \hat{\theta}_1 + 2\hat{\theta}_2 \end{pmatrix}$$

Для учёта краевых условий 2 и 3 рода, дополним левую и правую части соответствующими матрицами на соответствующих номерах узлов.

3.4.3. Краевые условия первого рода

Учёт первых краевых условий проводится после сборки матрицы и учёта естественных краевых условий. Для каждого узла i , где задано краевое условие первого рода, на диагональ ставим единицу, обнуляем строку, соответствующую данному узлу, а соответствующий элемент правой части приравниваем к $u_g(x_i, y_i)$ – значению краевого условия.

3.5. Построение портрета матрицы

Для работы с матрицей в разреженном формате необходимо построить её портрет, т.е. сформировать массивы ig, jg , где ig – индексы ненулевых строк, jg – номера столбцов, в которых хранятся элементы матрицы по строкам ig .

Построение портрета матрицы означает нахождение в конечноэлементной сетке для каждой глобальной базисной функции ψ_i номеров j всех базисных функций, для которых $\int \psi_i \psi_j \neq 0$, т.е. номеров всех глобальных базисных функций, не равных нулю на хотя бы одном из тех конечных элементов, на которых не равна нулю глобальная базисная функция ψ_i .

3.6. Схема Кранка-Николсона для аппроксимации по времени

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

$$u = \frac{u^j + u^{j-1}}{2}, f = \frac{f^j + f^{j-1}}{2}$$

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

Подставим в полое ранее уравнение Галеркина и получим итоговую систему **без учёта краевых условий**:

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

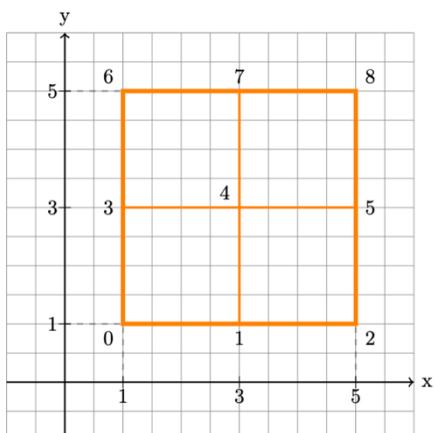
4. Описание разработанных программ

- IndexOfUnknown – получение номера глобального узла
- GridBuilder – построение сетки, чтение информации о подобластях, заполнение массива узлов
- LocalG_matrix, LocalM_matrix, LocalB_vector – формирование локальных матриц жёсткости и масс, локального вектора
- LocalMatrix – вычисление локальной матрицы
- Portret_builders – построение портрета глобальной матрицы
- GlobalMatrix, BuildGlobalVector – построение глобальной матрицы (сборка массивов di, al, au из локальных матриц), глобального вектора
- Kraevye, Third_K, Second_K, First_K – чтение краевых условий из файла, учёт третьих, вторых и первых краевых соответственно
- BuildGlobalMatrix – сборка глобальных матрицы и вектора, учёт всех краевых условий
- LOS_Dd – ЛОС с диагональным предобуславливанием
- SLAU – решение СЛАУ
- UG, Tetta, u_Betta, BettaV, gammaV, lambda, Func – входные параметры

5. Описание тестирования программ

1) Задано только первое краевое условие на всех границах

$$\begin{aligned}u &= 1 \\f &= 0 \\\lambda &= 1, \quad \sigma = 1\end{aligned}$$



номер узла	0	0.1	0.2	0.3	0.4	0.5
0	1	1	1	1	1	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	1	1	1	1	1	1

2) Заданы все краевые условия зависимость от времени

$$u = xt$$

$$f = y$$

$$\lambda = 1, \quad \sigma = 1$$

$$u|_{S_1^1} = xt$$

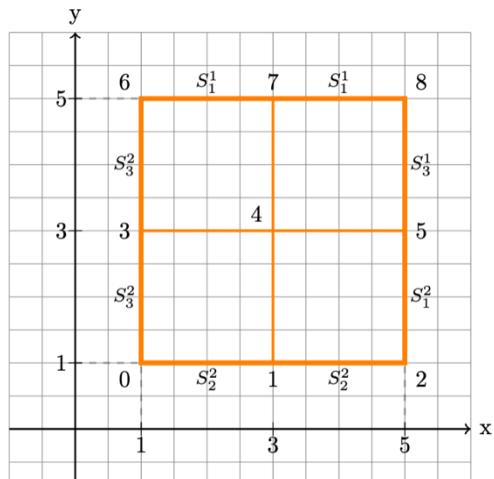
$$u|_{S_1^2} = 5t$$

$$\lambda \frac{du}{dn}|_{S_2^1} = 0$$

$$\lambda \frac{du}{dn}|_{S_2^0} = 0$$

$$u_\beta|_{S_3^1} = 6t$$

$$u_\beta|_{S_3^2} = 4t$$



x	y	t	u	u*	u-u*
1	1	0.1	0.100000000000	0.100000000000	0.00E+00
3	1		0.300000000000	0.300000000000	5.55E-17
5	1		0.500000000000	0.500000000000	0.00E+00
1	3		0.100000000000	0.100000000000	0.00E+00
3	3		0.300000000000	0.300000000000	5.55E-17
5	3		0.500000000000	0.500000000000	0.00E+00
1	5		0.100000000000	0.100000000000	0.00E+00
3	5		0.300000000000	0.300000000000	5.55E-17
5	5		0.500000000000	0.500000000000	0.00E+00
1	1	0.2	0.200000000000	0.200000000000	0.00E+00
3	1		0.600000000000	0.600000000000	1.11E-16
5	1		1.000000000000	1.000000000000	0.00E+00
1	3		0.200000000000	0.200000000000	0.00E+00
3	3		0.600000000000	0.600000000000	1.11E-16
5	3		1.000000000000	1.000000000000	0.00E+00
1	5		0.200000000000	0.200000000000	0.00E+00
3	5		0.600000000000	0.600000000000	1.11E-16
5	5		1.000000000000	1.000000000000	0.00E+00
1	1	0.3	0.300000000000	0.300000000000	0.00E+00
3	1		0.900000000000	0.900000000000	1.11E-16
5	1		1.500000000000	1.500000000000	0.00E+00
1	3		0.300000000000	0.300000000000	0.00E+00
3	3		0.900000000000	0.900000000000	1.11E-16
5	3		1.500000000000	1.500000000000	0.00E+00
1	5		0.300000000000	0.300000000000	0.00E+00
3	5		0.900000000000	0.900000000000	1.11E-16
5	5		1.500000000000	1.500000000000	0.00E+00
1	1	0.4	0.400000000000	0.400000000000	0.00E+00
3	1		1.200000000000	1.200000000000	2.22E-16
5	1		2.000000000000	2.000000000000	0.00E+00

1	3		0.400000000000	0.400000000000	0.00E+00
3	3		1.200000000000	1.200000000000	2.22E-16
5	3		2.000000000000	2.000000000000	0.00E+00
1	5		0.400000000000	0.400000000000	0.00E+00
3	5		1.200000000000	1.200000000000	2.22E-16
5	5		2.000000000000	2.000000000000	0.00E+00
1	1	0.5	0.500000000000	0.500000000000	0.00E+00
3	1		1.500000000000	1.500000000000	0.00E+00
5	1		2.500000000000	2.500000000000	0.00E+00
1	3		0.500000000000	0.500000000000	0.00E+00
3	3		1.500000000000	1.500000000000	0.00E+00
5	3		2.500000000000	2.500000000000	0.00E+00
1	5		0.500000000000	0.500000000000	0.00E+00
3	5		1.500000000000	1.500000000000	0.00E+00
5	5		2.500000000000	2.500000000000	0.00E+00

3) Заданы все краевые условия

$$u = x + yt$$

$$f = y$$

$$\lambda = 1, \quad \sigma = 1$$

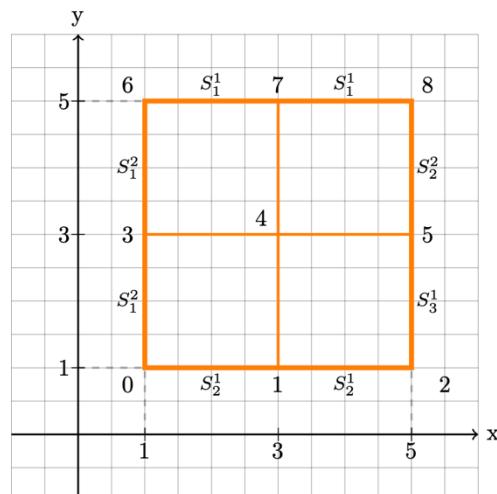
$$u|_{S_1^1} = x + 5t$$

$$u|_{S_1^2} = 1 + yt$$

$$u|_{S_1^3} = 5 + yt$$

$$\lambda \frac{du}{dn}|_{S_2^1} = -t$$

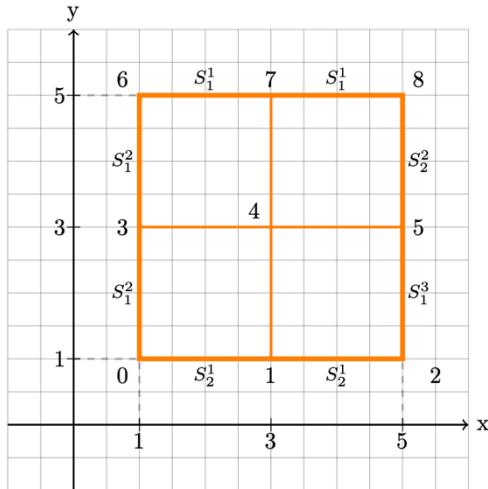
$$u_\beta|_{S_2^1} = 6 + yt$$



x	y	t	u	u*	u-u*
1	1	0.1	1.100000000000	1.100000000000	0.00E+00
3	1		3.100000000000	3.13350551288	3.35E-02
5	1		5.100000000000	5.100000000000	9.77E-15
1	3		1.300000000000	1.300000000000	0.00E+00
3	3		3.300000000000	3.31866301020	1.87E-02
5	3		5.300000000000	5.300000000000	0.00E+00
1	5		1.500000000000	1.500000000000	0.00E+00
3	5		3.500000000000	3.500000000000	0.00E+00
5	5		5.500000000000	5.500000000000	0.00E+00
1	1	0.2	1.200000000000	1.200000000000	0.00E+00
3	1		3.200000000000	3.25745545031	5.75E-02
5	1		5.200000000000	5.200000000000	0.00E+00
1	3		1.600000000000	1.600000000000	0.00E+00

3	3		3.600000000000	3.64024086833	4.02E-02
5	3		5.600000000000	5.600000000000	0.00E+00
1	5		2.000000000000	2.000000000000	0.00E+00
3	5		4.000000000000	4.000000000000	0.00E+00
5	5		6.000000000000	6.000000000000	0.00E+00
1	1	0.3	1.300000000000	1.300000000000	0.00E+00
3	1		3.300000000000	3.37328518735	7.33E-02
5	1		5.300000000000	5.300000000000	0.00E+00
1	3		1.900000000000	1.900000000000	0.00E+00
3	3		3.900000000000	3.96393140119	6.39E-02
5	3		5.900000000000	5.900000000000	0.00E+00
1	5		2.500000000000	2.500000000000	0.00E+00
3	5		4.500000000000	4.500000000000	0.00E+00
5	5		6.500000000000	6.500000000000	0.00E+00
1	1	0.4	1.400000000000	1.400000000000	9.99E-15
3	1		3.400000000000	3.48218176247	8.22E-02
5	1		5.400000000000	5.400000000000	1.07E-14
1	3		2.200000000000	2.200000000000	1.02E-14
3	3		4.200000000000	4.28910473901	8.91E-02
5	3		6.200000000000	6.200000000000	9.77E-15
1	5		3.000000000000	3.000000000000	1.02E-14
3	5		5.000000000000	5.000000000000	9.77E-15
5	5		7.000000000000	7.000000000000	9.77E-15
1	1	0.5	1.500000000000	1.500000000000	9.99E-15
3	1		3.500000000000	3.58512985331	8.51E-02
5	1		5.500000000000	5.500000000000	9.77E-15
1	3		2.500000000000	2.500000000000	1.02E-14
3	3		4.500000000000	4.61526873871	1.15E-01
5	3		6.500000000000	6.500000000000	9.77E-15
1	5		3.500000000000	3.500000000000	1.02E-14
3	5		5.500000000000	5.500000000000	9.77E-15
5	5		7.500000000000	7.500000000000	9.77E-15

5.1. Порядок аппроксимации



$$u = t$$

$$f = 1$$

$$\lambda = 1, \quad \sigma = 1$$

$$u|_{S_1^1} = t$$

$$u|_{S_1^2} = t$$

$$u|_{S_1^3} = t$$

$$\lambda \frac{du}{dn}|_{S_2^1} = 0$$

Nº	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
1	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
2	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
3	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
4	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
5	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
6	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
7	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000
8	0.1000000	0.2000000	0.3000000	0.4000000	0.5000000	0.6000000	0.7000000	0.8000000	0.9000000	1.0000000

$$u = t^2$$

$$\begin{aligned} f &= 2t \\ \lambda &= 1, \quad \sigma = 1 \\ u|_{s_1^1} &= t^2 \\ u|_{s_1^2} &= t^2 \\ u|_{s_1^3} &= t^2 \\ \lambda \frac{du}{dn}|_{s_2^1} &= 0 \end{aligned}$$

No	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
1	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
2	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
3	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
4	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
5	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
6	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
7	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000
8	0.0100000	0.0400000	0.0900000	0.1600000	0.2500000	0.3600000	0.4900000	0.6400000	0.8100000	1.0000000

$$u = t^3$$

$$\begin{aligned} f &= 3t^2 \\ \lambda &= 1, \quad \sigma = 1 \\ u|_{s_1^1} &= t^3 \\ u|_{s_1^2} &= t^3 \\ u|_{s_1^3} &= t^3 \\ \lambda \frac{du}{dn}|_{s_2^1} &= 0 \end{aligned}$$

No	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000
1	0.0016424	0.0092870	0.0289243	0.0665473	0.1281511	0.2197321	0.3472880	0.5168174	0.7343197	1.0057948
2	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000
3	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000
4	0.0019027	0.0097031	0.0294159	0.0670531	0.1286251	0.2201402	0.3476056	0.5170274	0.7344107	1.0057598
5	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000
6	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000
7	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000
8	0.0010000	0.0080000	0.0270000	0.0640000	0.1250000	0.2160000	0.3430000	0.5120000	0.7290000	1.0000000

5.2. Исследование на сходимость

$$u = \sin(t), t \in [0, 5]$$

$$f = \cos(t)$$

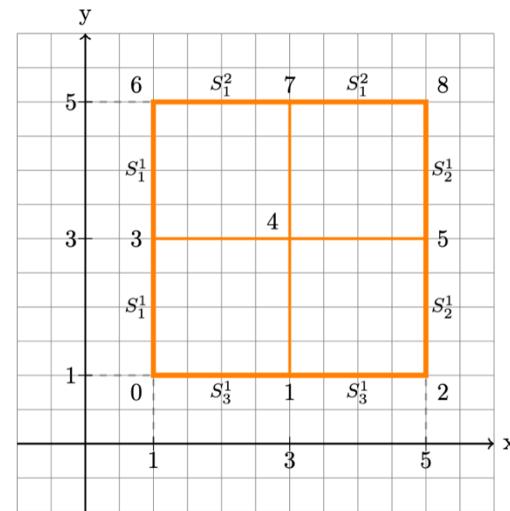
$$\lambda = 1, \sigma = 1$$

$$u|_{S_1^1} = \sin(t)$$

$$u|_{S_1^2} = \sin(t)$$

$$\lambda \frac{du}{dn}|_{S_2^1} = 0$$

$$u_\beta|_{S_3^1} = \sin(t)$$



Результат будем смотреть на слое $t = 1$

	6	11	21	41	81	161
	$\Delta t = 1$	$\Delta t = 0,5$	$\Delta t = 0,25$	$\Delta t = 0,125$	$\Delta t = 0,0625$	$\Delta t = 0,03125$
номер узла	u_h	u_h/2	u_h/4	u_h/8	u_h/16	u_h/32
0	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848
1	0.80394693196	0.8311470183	0.8387637599	0.8407770929	0.8412952723	0.8414267636
2	0.8084524831	0.831794365	0.8389063627	0.8408115349	0.8413037795	0.8414288904
3	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848
4	0.7592678267	0.8220236968	0.836638806	0.8402609097	0.8411678537	0.841395019
5	0.7660611293	0.8231256427	0.8368970945	0.8403259876	0.8411843664	0.8413992476
6	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848
7	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848
8	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848	0.8414709848

Узел 4 - внутренний

N	h	$\ u-u^*\ $	отношение
6	1	0.1950023796	
11	0.5	0.072483989718	1.427757456
21	0.25	0.02643763204	1.455069404
41	0.125	0.009510474337	1.475003761
81	0.0625	0.003393289187	1.486832911
161	0.03125	0.001205625266	1.49290283

Порядок сходимости стремится к 2

6. Текст программы

```
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <set>
#include <math.h>
#include <iomanip>
using namespace std;

struct Rectangle
{
    int position; //подобласть
    int x_left;
    int x_right; //координаты краевого
    int y_down;
    int y_up;
};

int x_size = 0; //кол-во элементов по оси x
int y_size = 0; //кол-во элементов по оси y
int global_size = 0; //общее кол-во элементов
double h_x = 0; //шаг сетки по x
double h_y = 0; //шаг сетки по y
int L = 0; //кол-во подобластей
int cntKraev = 0; //кол-во краевых условий

vector<int> X_grid; //разбиение на сетку по x (координаты)
vector<int> Y_grid; //разбиение на сетку по y (координаты)
vector<vector <int> > nodes;
vector<vector<double> > G_l; //матрица жесткости
vector<vector<double> > M_l; //матрица масс
vector<double> b_l; //вектор правых частей
vector<vector<double> > localA; //матрица коэффициентов
vector<double> localB; //вектор коэффициентов
vector<double> B; //вектор глобальный
vector<double> b_prev; //предыдущая правая часть  $b^{j-1}$ 
vector<Rectangle> rectangles;
vector<double> gridt; //временная сетка

//хранение матрицы в разреженном строчном формате
vector<double> di; //диагональ
vector<double> al;
vector<double> au;
vector<int> ig; //индексы строк ненулевых компонент
vector<int> jg; //индексы столбцов ненулевых компонент

//краевые условия
vector<vector<int> > kraevye_uslov; //краевые условия
vector<vector<double> > val_A3; //значения 3 краевых условий для матрицы
vector<double> val_B; //значения 3 и 2 краевых условий для вектора b

//решение слаг
```

```

vector<double> x0; //начальное приближение
vector<double> y;
vector<double> z;
vector<double> t;
vector<double> r;
vector<double> p;
vector<double> x;
vector<double> u_prev; //предыдущее решение  $u^{n-1}$ 

int maxiter = 10000;
double e = 1e-18;

//параметры для Кранка-Николсона
double T = 5; //конечное время
double tau = 1/64.; //шаг по времени
int nt = static_cast<int>(T / tau) + 1; //кол-во временных шагов
double sigma = 1.0; //коэффициент sigma

double Func(int num, double x, double y, double t)
{
    return cos(t);
}

double lambdaV (int num)
{
    switch(num){
        case 1: return 1.0;
        case 2: return 1.0;
        case 3: return 1.;
    }
}

double BettaV(int num)
{
    switch(num)
    {
        case 1: return 1.;
    }
}

double u_Betta(int num, double x, double y, double t)
{
    switch(num)
    {
        case 1: return sin(t);
    }
}

double Tetta(int num, double x, double y, double t)
{
    return 0;
    // switch(num)
    // {

```

```

//      case 1: return -t;
//      //case 2: return 0.0;
// }
}

double UG(int num, double x, double y, double t)
{
    return sin(t);
    // switch(num)
    // {
    //     case 1: return x + 5 *t;
    //     case 2: return 1 + y * t;
    //     case 3: return 5 + y * t;
    // }
}

double u_g(int num, double x, double y, double t) {
    return sin(t);
}

/*получение глобального номера узла*/
int IndexOfUnknown(int i, int j) {
    int elementsPerRow = x_size - 1;
    int row = i / elementsPerRow;
    int col = i % elementsPerRow;
    int globalIndex;
    switch (j) {
        case 0: globalIndex = row * x_size + col; break;
        case 1: globalIndex = row * x_size + col + 1; break;
        case 2: globalIndex = (row + 1) * x_size + col; break;
        case 3: globalIndex = (row + 1) * x_size + col + 1; break;
        default: cout << "Invalid local index for element" << endl; return -1;
    }
    return globalIndex;
}

/*построение сетки и временной сетки*/
void GridBuilder()
{
    ifstream input("./test/sxod/matrix.txt");
    input >> x_size >> y_size;
    X_grid.resize(x_size);
    Y_grid.resize(y_size);
    for (int i = 0; i < x_size; i++)
        input >> X_grid[i];
    for (int i = 0; i < y_size; i++)
        input >> Y_grid[i];
    global_size = x_size * y_size;

    //построение подобластей
    input >> L;
    rectangles.resize(L);
    for (int i = 0; i < L; i++)

```

```

        input >> rectangles[i].position >> rectangles[i].x_left >> rectangles[i].x_right >> rec-
tangles[i].y_down >> rectangles[i].y_up;
input.close();

nodes.resize(global_size);
int i = 0;
for (double y_c : Y_grid) {
    for (double x_c : X_grid) {
        nodes[i].push_back(x_c);
        nodes[i].push_back(y_c);
        i++;
    }
}

//инициализация временной сетки
gridt.resize(nt);
for (int i = 0; i < nt; i++)
    gridt[i] = i * tau;
}

/*построение локальной матрицы жесткости*/
vector<vector<double> > LocalG_matrix(int k, int num_L)
{
    vector<vector<double> > G(4, vector<double>(4));
    double lambda = lambdaV(k);

    h_x = rectangles[num_L].x_right - rectangles[num_L].x_left;
    h_y = rectangles[num_L].y_up - rectangles[num_L].y_down;
    double a1 = (lambda/6)*(h_y/h_x);
    double a2 = lambda*h_x/(6*h_y);

    G[0][1] = -2 * a1 + a2;
    G[0][2] = a1 - 2 * a2;
    G[0][3] = -a1 - a2;
    G[1][2] = -a1 - a2;
    G[1][3] = a1 - 2 * a2;
    G[2][3] = -2 * a1 + a2;

    G[0][0] = 2 * a1 + 2 * a2;
    G[1][1] = 2 * a1 + 2 * a2;
    G[2][2] = 2 * a1 + 2 * a2;
    G[3][3] = 2 * a1 + 2 * a2;

    G[1][0] = -2 * a1 + a2;
    G[2][0] = a1 - 2 * a2;
    G[2][1] = -a1 - a2;
    G[3][0] = -a1 - a2;
    G[3][1] = a1 - 2 * a2;
    G[3][2] = -2 * a1 + a2;

    return G;
}

```

```

/*построение локальной матрицы масс*/
vector<vector<double> > LocalM_matrix(int k, int num_L)
{
    vector<vector<double> > M(4, vector<double>(4));
    h_x = rectangles[num_L].x_right - rectangles[num_L].x_left;
    h_y = rectangles[num_L].y_up - rectangles[num_L].y_down;
    double a = (h_x * h_y) / 36;
    M[0][1] = 2 * a;
    M[0][2] = 2 * a;
    M[0][3] = a;
    M[1][2] = a;
    M[1][3] = 2 * a;
    M[2][3] = 2 * a;

    M[0][0] = 4 * a;
    M[1][1] = 4 * a;
    M[2][2] = 4 * a;
    M[3][3] = 4 * a;

    M[1][0] = 2 * a;
    M[2][0] = 2 * a;
    M[2][1] = a;
    M[3][0] = a;
    M[3][1] = 2 * a;
    M[3][2] = 2 * a;

    return M;
}

/*построение локального вектора правых частей*/
vector<double> LocalB_vector(int num_L, double t)
{
    h_x = rectangles[num_L].x_right - rectangles[num_L].x_left;
    h_y = rectangles[num_L].y_up - rectangles[num_L].y_down;
    vector<double> b(4);
    double f1 = Func(rectangles[num_L].position, rectangles[num_L].x_left, rectangles[num_L].y_down, t);
    double f2 = Func(rectangles[num_L].position, rectangles[num_L].x_right, rectangles[num_L].y_down, t);
    double f3 = Func(rectangles[num_L].position, rectangles[num_L].x_left, rectangles[num_L].y_up, t);
    double f4 = Func(rectangles[num_L].position, rectangles[num_L].x_right, rectangles[num_L].y_up, t);
    double a = (h_x * h_y) / 36;
    b[0] = a * (4 * f1 + 2 * f2 + 2 * f3 + f4);
    b[1] = a * (2 * f1 + 4 * f2 + f3 + 2 * f4);
    b[2] = a * (2 * f1 + f2 + 4 * f3 + 2 * f4);
    b[3] = a * (f1 + 2 * f2 + 2 * f3 + 4 * f4);

    cout << "VECTOR number of elem: " << num_L + 1 << " at t = " << t << endl;
    for (int i = 0; i < 4; i++) cout << b[i] << " ";
    cout << endl;
    return b;
}

```

```

}

/*Вычисление локальной матрицы для Кранка-Николсона*/
vector<vector<double>> LocalMatrix(int num_L, int k)
{
    vector<vector<double>> A(4, vector<double>(4));
    G_l = LocalG_matrix(num_L, k);
    M_l = LocalM_matrix(num_L, k);
    //матрица для левой части: ( $\sigma/\tau$ ) * M + (1/2) * G
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++)
            A[i][j] = (sigma / tau) * M_l[i][j] + (1.0 / 2.0) * G_l[i][j];
    }
    return A;
}

/*Вычисление правой части для Кранка-Николсона*/
vector<double> LocalRHS(int num_L, int k, const vector<double>& u_prev, double t_n, double t_np1)
{
    vector<double> rhs(4, 0.0);
    G_l = LocalG_matrix(num_L, k);
    M_l = LocalM_matrix(num_L, k);
    vector<double> b_n = LocalB_vector(num_L, t_n);      //  $b^{j-1}$  на  $t_n$ 
    vector<double> b_np1 = LocalB_vector(num_L, t_np1); //  $b^j$  на  $t_{n+1}$ 

    // Осреднение:  $(b^j + b^{j-1})/2$ 
    vector<double> b_avg(4);
    for (int i = 0; i < 4; i++) {
        b_avg[i] = (b_np1[i] + b_n[i]) / 2.0;
    }

    //глобальные индексы узлов элемента
    vector<int> global_indices(4);
    for (int p = 0; p < 4; p++)
        global_indices[p] = IndexOfUnknown(k, p);

    //вычисление  $((\sigma/\tau) * M - (1/2) * G) * u^n$ 
    vector<double> temp(4, 0.0);
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            double mat_coeff = (sigma / tau) * M_l[i][j] - (1.0 / 2.0) * G_l[i][j];
            temp[i] += mat_coeff * u_prev[global_indices[j]];
        }
    }

    //правая часть:  $((\sigma/\tau) * M - (1/2) * G) * u^n + (b^j + b^{j-1})/2$ 
    for (int i = 0; i < 4; i++) {
        rhs[i] = temp[i] + b_avg[i];
    }

    return rhs;
}

```

```

/*построение портрета глобальной матрицы*/
void Portret_builders()
{
    vector<set<int>> list;
    list.resize(global_size);
    int m = 0;
    for (; m < L; m++) {
        vector<int> tmp;
        for (int p = 3; p >= 0; p--)
            tmp.push_back(IndexOfUnknown(m, p));
        for (int i = 0; i < 4; i++) {
            int ind1 = tmp[i];
            for (int j = i + 1; j < 4; j++) {
                int ind2 = tmp[j];
                list[ind1].insert(ind2);
            }
        }
    }

    ig.resize(global_size + 1);
    ig[0] = 0;
    ig[1] = 0;
    for (int i = 0; i < global_size; i++) {
        ig[i + 1] = ig[i] + list[i].size();
    }

    jg.resize(ig[global_size]);
    int k = 0;
    jg[k] = 0;
    for (int i = 0; i < global_size; i++) {
        for (int elem : list[i]) {
            jg[k] = elem;
            k++;
        }
    }

    di.resize(global_size);
    al.resize(ig[global_size]);
    au.resize(ig[global_size]);
}

/*построение глобальной матрицы*/
void GlobalMatrix(vector<vector<double>> &localA, int k)
{
    vector<int> tmp;
    for (int p = 0; p < 4; p++) {
        tmp.push_back(IndexOfUnknown(k, p));
        di[tmp[p]] += localA[p][p];
    }
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (i == j) continue;
            int igg = tmp[i];

```

```

        int jgg = tmp[j];
        int row = max(igg, jgg);
        int col = min(igg, jgg);
        int index = ig[row];
        while (index < ig[row + 1] && jg[index] != col) index++;
        if (index == ig[row + 1]) {
            cout << "Error: Index not found in GlobalMatrix for row " << row << ", col " <<
col << endl;
            continue;
        }
        if (igg > jgg) {
            al[index] += localA[i][j];
        } else {
            au[index] += localA[i][j];
        }
    }
}
}

/*Вывод глобальной матрицы*/
void PrintGlobalMatrix(int global_size) {
    // Создаем временную двумерную матрицу для удобного отображения
    vector<vector<double>> fullMatrix;
    fullMatrix.resize(global_size, vector<double>(global_size, 0));

    // Заполнение диагональных элементов
    for (int i = 0; i < global_size; i++) {
        fullMatrix[i][i] = di[i];
    }

    // Заполнение недиагональных элементов
    for (int i = 0; i < global_size; i++) {
        for (int j = ig[i]; j < ig[i + 1]; j++) {
            int col = jg[j];           // Столбец из разреженной структуры
            fullMatrix[i][col] = al[j]; // Значение из массива al
            fullMatrix[col][i] = au[j]; // Симметричность матрицы
        }
    }

    // Вывод матрицы в консоль
    cout << "Global Matrix:" << endl;
    for (int i = 0; i < global_size; i++) {
        for (int j = 0; j < global_size; j++) {
            cout << setw(10) << fullMatrix[i][j] << " ";
        }
        cout << endl;
    }

    cout << "Global Vector:" << endl;
    for (int i = 0; i < global_size; i++) {
        cout << setw(10) << B[i] << " ";
    }
    cout << endl;
}

```

```

}

/*сборка глобального вектора*/
vector<double> BuildGlobalVector(vector<double> &l_B, int k)
{
    vector<double> tmp(4);
    for (int i = 0; i < 4; i++) tmp[i] = IndexOfUnknown(k, i);
    for (int i = 0; i < 4; i++) {
        int ind = tmp[i];
        B[ind] += l_B[i];
    }
    return B;
}

/*построение краевых условий*/
void Kraevye()
{
    ifstream input("./test/sxod/kraevie.txt");
    input >> cntKraev;
    kraevye_uslov.resize(cntKraev);
    for (int i = 0; i < cntKraev; i++) {
        kraevye_uslov[i].resize(4);
        for (int j = 0; j < 4; j++)
            input >> kraevye_uslov[i][j];
    }
    input.close();
}

/*учёт 3х краевых условий*/
void Third_K(const vector<int> &kraevye_uslov, double t) {
    vector<int> tmp;
    int i = kraevye_uslov[2];
    int j = kraevye_uslov[3];
    double betta = BettaV(kraevye_uslov[1]);
    double u_betta1 = u_Betta(kraevye_uslov[1], nodes[i][0], nodes[i][1], t);
    double u_betta2 = u_Betta(kraevye_uslov[1], nodes[j][0], nodes[j][1], t);
    tmp.push_back(i);
    tmp.push_back(j);
    double h;
    double x1 = nodes[i][0];
    double x2 = nodes[j][0];
    double y1 = nodes[i][1];
    double y2 = nodes[j][1];
    h = (x1 == x2) ? abs(y2 - y1) : abs(x2 - x1);
    double a = (betta * h) / 6.0;
    val_A3.resize(2, vector<double>(2));
    val_A3[0][0] = 2 * a;
    val_A3[0][1] = a;
    val_A3[1][0] = a;
    val_A3[1][1] = 2 * a;
    val_B.resize(2);
    val_B[0] = a * (2 * u_betta1 + u_betta2);
    val_B[1] = a * (u_betta1 + 2 * u_betta2);
}

```

```

B[i] += val_B[0];
B[j] += val_B[1];
di[i] += val_A3[0][0];
di[j] += val_A3[1][1];
int ia = j, ja = i;
int index = ig[ia];
int flag = 1;
for (; index < ig[ia + 1] && flag; index++)
    if (jg[index] == ja) flag = 0;
index--;
al[index] += val_A3[1][0];
au[index] += val_A3[0][1];
}

/*учёт 2x краевых условий*/
void Second_K(const vector<int> &kraevye_uslov, double t)
{
vector<int> tmp;
double h;
int i = kraevye_uslov[2];
int j = kraevye_uslov[3];
double tetta1 = Tetta(kraevye_uslov[1], nodes[i][0], nodes[i][1], t);
double tetta2 = Tetta(kraevye_uslov[1], nodes[j][0], nodes[j][1], t);
tmp.push_back(i);
tmp.push_back(j);
double x1 = nodes[i][0];
double x2 = nodes[j][0];
double y1 = nodes[i][1];
double y2 = nodes[j][1];
h = (x1 == x2) ? abs(y2 - y1) : abs(x2 - x1);
double a = h / 6.0;
val_B.resize(2);
val_B[0] = a * (2 * tetta1 + tetta2);
val_B[1] = a * (tetta1 + 2 * tetta2);
B[i] += val_B[0];
B[j] += val_B[1];
}

/*учёт 1x краевых условий*/
void First_K(const vector<int> &kraevye_uslov, double t) {
vector<int> tmp;
int p = kraevye_uslov[2];
int s = kraevye_uslov[3];
tmp.push_back(p);
tmp.push_back(s);
for (int globalNode : tmp) {
    double x_coord = nodes[globalNode][0];
    double y_coord = nodes[globalNode][1];
    double u_g = UG(kraevye_uslov[1], x_coord, y_coord, t);
    di[globalNode] = 1;
    B[globalNode] = u_g;
    for (int i = ig[globalNode]; i < ig[globalNode + 1]; i++) {
        al[i] = 0;
    }
}
}

```

```

    }
    for (int i = ig[globalNode]; i < ig[global_size]; i++) {
        if (jg[i] == globalNode)
            au[i] = 0;
    }
}

void AddKraevye(int num, vector<int> &kraevye_uslov, double t)
{
    switch(num) {
        case 1: First_K(kraevye_uslov, t); break;
        case 2: Second_K(kraevye_uslov, t); break;
        case 3: Third_K(kraevye_uslov, t); break;
        default: cout << "Incorrect Boundary Condition Number" << endl; break;
    }
}

/*сборка глобальной матрицы и вектора*/
void BuildGlobalMatrixAndVector(const vector<double>& u_prev, double t_n, double t_np1) {
    //обнуление глобальных массивов
    fill(di.begin(), di.end(), 0.0);
    fill(al.begin(), al.end(), 0.0);
    fill(au.begin(), au.end(), 0.0);
    fill(B.begin(), B.end(), 0.0);

    //сборка матрицы и вектора
    for (int k = 0; k < L; k++) {
        localA = LocalMatrix(rectangles[k].position, k);
        GlobalMatrix(localA, k);
        localB = LocalRHS(k, k, u_prev, t_n, t_np1);
        B = BuildGlobalVector(localB, k);
    }

    //PrintGlobalMatrix(global_size);

    //учёт краевых условий на t_{n+1}
    for (int i = 0; i < cntKraev; i++)
        AddKraevye(kraevye_uslov[i][0], kraevye_uslov[i], t_np1);
}

double vector_multiplication(vector<double> &v1, vector<double> &v2)
{
    double sum = 0;
    for (int i = 0; i < global_size; i++)
        sum += v1[i] * v2[i];
    return sum;
}

vector<double> matrix_on_vector_multiplication(vector<double> &v1, vector<double> &v2)
{
    for (int i = 0; i < global_size; i++)
        v2[i] = di[i] * v1[i];
}

```

```

    for (int i = 0; i < global_size; i++) {
        for (int j = ig[i]; j < ig[i+1]; j++) {
            v2[i] += al[j] * v1[jg[j]];
            v2[jg[j]] += au[j] * v1[i];
        }
    }
    return v2;
}

vector<double> vector_sum(vector<double> &v1, vector<double> &v2, vector<double> &v3, double a)
{
    for (int i = 0; i < global_size; i++)
        v3[i] = v1[i] + a * v2[i];
    return v3;
}

double Norma(vector<double> &v)
{
    double norma = 0;
    for (int i = 0; i < global_size; i++)
        norma += v[i] * v[i];
    return sqrt(norma);
}

void Calc_Zk(double a, vector<double> &v1)
{
    for (int i = 0; i < global_size; i++)
        z[i] = v1[i] + a * z[i];
}

void Calc_Pk(double a, vector<double> &v1)
{
    for (int i = 0; i < global_size; i++)
        p[i] = v1[i] + a * p[i];
}

void Calc_Xk_Rk_L(double alphaK)
{
    for (int i = 0; i < global_size; i++) {
        x[i] = x[i] + alphaK * z[i];
        r[i] = r[i] - alphaK * p[i];
    }
}

double calcResidual() {
    double normb = 0;
    for (int i = 0; i < global_size; i++) {
        normb += B[i] * B[i];
        r[i] = B[i] - di[i] * x[i];
        int m = ig[i + 1];
        for (int k = ig[i]; k < m; k++) {
            int j = jg[k];
            r[i] -= al[k] * x[j];
        }
    }
}

```

```

        r[j] -= au[k] * x[i];
    }
}
double mul = vector_multiplication(r, r);
return sqrt(mul/normb);
}

/*ЛОС с диагональным предобуславливанием*/
void LOS_Dd()
{
    double alpha, betta;
    double normPr = Norma(B);
    matrix_on_vector_multiplication(x0, y);
    for (int i = 0; i < global_size; i++) r[i] = B[i] - y[i];
    for (int i = 0; i < global_size; i++) z[i] = r[i];
    for (int i = 0; i < global_size; i++) x[i] = x0[i];
    matrix_on_vector_multiplication(z, p);
    int k = 0;
    double discrepancy = calcResidual();
    for (; k < maxiter && discrepancy > e; k++) {
        double scal_pk_rk = vector_multiplication(r, p);
        double scal_pk = vector_multiplication(p, p);
        alpha = scal_pk_rk / scal_pk;
        Calc_Xk_Rk_L(alpha);
        matrix_on_vector_multiplication(r, y);
        double Ar_p = vector_multiplication(y, p);
        betta = -Ar_p / scal_pk;
        Calc_Zk(betta, r);
        Calc_Pk(betta, y);
        discrepancy = calcResidual();
    }
    cout << "Iteration: " << k << " RelDiscrepancy: " << discrepancy << endl;
}

void Output(ofstream& out, int time_step)
{
    out << fixed << setprecision(15);
    out << "Time step: " << time_step << " t = " << gridt[time_step] << endl;
    for (int i = 0; i < global_size; i++) {
        out << x[i] << endl;
    }
}

void SLAU()
{
    x0.resize(global_size);
    x.resize(global_size);
    y.resize(global_size);
    z.resize(global_size);
    t.resize(global_size);
    r.resize(global_size);
    p.resize(global_size);
    LOS_Dd();
}

```

```

}

int main() {
    GridBuilder();
    B.resize(global_size);
    u_prev.resize(global_size);
    for (int i = 0; i < global_size; i++) {
        u_prev[i] = 0.;
        //u_prev[i] = u_g(rectangles[0].position, nodes[i][0], nodes[i][1], 0.);
        //u_prev[i] = UG(rectangles[0].position, nodes[i][0], nodes[i][1], 0.0); // u(t=0) no u_g
    }
    b_prev.resize(global_size, 0.0); //инициализация b^{j-1}
    Kraevye();
    Portret_builders();

    ofstream out("out.txt");

    //вычисление b^{j-1} на t = 0
    for (int k = 0; k < L; k++) {
        vector<double> b_tmp = LocalB_vector(k, gridt[0]);
        B = BuildGlobalVector(b_tmp, k);
    }
    b_prev = B; //сохранение b^{j-1}

    //временной цикл
    for (int n = 0; n < nt - 1; n++) {
        double t_n = gridt[n];
        double t_np1 = gridt[n + 1];
        cout << "Time step: " << n + 1 << " t = " << t_np1 << endl;
        BuildGlobalMatrixAndVector(u_prev, t_n, t_np1);
        //PrintGlobalMatrix(global_size);
        SLAU();
        Output(out, n + 1);

        //обновление b^{j-1} для следующего шага
        fill(B.begin(), B.end(), 0.0);
        for (int k = 0; k < L; k++) {
            vector<double> b_tmp = LocalB_vector(k, t_np1);
            B = BuildGlobalVector(b_tmp, k);
        }
        b_prev = B;
        u_prev = x; //обновление решения для следующего шага
    }

    out.close();
    return 0;
}

```