МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

НГТУ НЭТИ

Кафедра прикладной математики

Курсовая работа
по дисциплине «Метод конечных элементов»

МЕТОД КОНЕЧНЫХ ЭЛЕМЕНТОВ

Факультет	ПМИ
Группа	ПМ-84
Студент	САТ А.А.
Преподаватели	РОЯК М.Э.
	ПАТРУШЕВ И.И.

Новосибирск

- **Постановка задачи**

Параболическая краевая задача для функции *u* определяется дифференциальным уравнением

$$-div(\lambda gradu) + \sigma\frac{\partial u}{\partial t} = f,$$

заданным в некоторой области Ω с границей S = S_1 U S_2, и краевыми условиями:

$u|_{s1} = u_g -$ первого рода,

$\lambda\frac{du}{dn}|_{s2} = \theta$ – второго рода.

Вариант: МКЭ для двумерной краевой задачи для теплового уравнения в декартовой системе координат. Базисные функции иерархические кубические на треугольниках. Трехслойная неявная схема по времени. Краевые условия первого и второго рода. Разрывный коэффициент теплопроводности

- **Теоретическая часть**

  - **Дискретизация по времени**

При построении дискретных аналогов начально-краевой задачи для дифференциального уравнения будем полагать, что ось времени t разбита на так называемые временные слои значениями $t_j, j = 1 \dots J$, а значит искомой функцией u и параметров λ, γ и f дифференциального уравнения параболического типа на j-м (при $t = t_j$) будем обозначать через $u^j, \lambda^j, \gamma^j, f^j$, которые уже не зависят от времени, но остаются функциями пространственных координат.

Помимо краевых условий, начально-краевая задача для уравнения параболического типа должна включать в себя начальное условие

$$u|_{t=t_0} u^0,$$

где u^0 – заданная функция пространственных координат.

Для описания вычислительных процедур построения конечноэлементных решений начально-краевых задач удобно ввести две глобальные матрицы, получающиеся в результате соответствующей аппроксимации отдельных членов дифференциального уравнения эллиптического типа: матрицу жёсткости G как аппроксимацию члена $-div(\lambda gradu)$, т. е

$$-div(\lambda gradu) \xrightarrow{\text{МКЭ}-\text{аппроксимация}} G \cdot q,$$

Матрицу массы M как аппроксимацию члена γu, т. е

$$\gamma \cdot u \xrightarrow{\text{МКЭ}-\text{аппроксимация}} M \cdot q,$$

где q – вектор весов разложения, конечно, элементного решения u^h по базисным функциям, зависящим только от пространственных координат. Через b будем обозначать вектор, определяющий вклад в вектор правой части, конечно, элементной СЛАУ от правой части f дифференциального уравнения и краевых условий.

Для получения, конечно, элементного решения будем использовать неявную трёхслойную схему аппроксимации по времени:

$$u(x,y,t) \approx u^{j-2}(x,y)\eta_2^j(t) + u^{j-1}(x,y)\eta_1^j(t) + u^j(x,y)\eta_0^j(t)$$

В аппроксимации функции пространственных координат u^{j-2}, u^{j-1}, u^j являются значениями искомой функции u при $t = t_{j-2}, t = t_{j-1}, t = t_j$ соответственно.

$$\eta_2^j(t) = \frac{1}{\Delta t_1 \Delta t}\big(t - t_{j-1}\big)\big(t - t_j\big),$$

$$\eta_1^j(t) = \frac{1}{\Delta t_1 \Delta t_0}\big(t - t_{j-2}\big)\big(t - t_j\big),$$

$$\eta_0^j(t) = \frac{1}{\Delta t \Delta t_0}\big(t - t_{j-2}\big)\big(t - t_{j-1}\big),$$

где $\Delta t = \big(t - t_{j-2}\big), \Delta t_1 = \big(t_{j-1} - t_{j-2}\big), \Delta t_0 = \big(t - t_{j-1}\big)$.

Применим данное представление для аппроксимации производной по времени параболического уравнения на временном слое $t = t_j$:

$$\frac{\partial}{\partial t}u^{j-2}(x,y)\eta_2^j(t) + u^{j-1}(x,y)\eta_1^j(t) + u^j(x,y)\eta_0^j(t)\big|_{t=t_j} -$$
$$-div\big(\lambda grad u^j\big) = f^j,$$

$$\frac{d\eta_2^j(t)}{dt} = \frac{\Delta t_0}{\Delta t_1 \Delta t}, \qquad \frac{d\eta_1^j(t)}{dt} = -\frac{\Delta t}{\Delta t_1 \Delta t_0}, \qquad \frac{d\eta_0^j(t)}{dt} = \frac{\Delta t + \Delta t_0}{\Delta t \Delta t_0}.$$

С учётом этого, параболическое уравнение может быть переписано в виде:

$$\frac{\Delta t_0}{\Delta t_1 \Delta t}u^{j-2} - \frac{\Delta t}{\Delta t_1 \Delta t_0}u^{j-1} + \frac{\Delta t + \Delta t_0}{\Delta t \Delta t_0}u^j - div\big(\lambda grad u^j\big) = f^j.$$

Выполняя конечноэлементную аппроксимацию краевой задачи для уравнения, получим СЛАУ следующего вида:

$$\left(\frac{\Delta t + \Delta t_0}{\Delta t \Delta t_0}M + G\right)q^j = b^j - \frac{\Delta t_0}{\Delta t_1 \Delta t}Mq^{j-2} + \frac{\Delta t}{\Delta t_1 \Delta t_0}Mq^{j-1},$$

где M – матрица массы, G -матрица жёсткости, b – вектор правой части, построенный по значениям функции f и параметра θ краевых условий на текущем временном слое, q^{j-2} и q^{j-1} – решение на двух предыдущих слоях по времени, q^j- искомое решение на текущем слое по времени.

Данной схемой мы можем воспользоваться в том случае, если известны значения u на предыдущих двух временных слоях, т.е. мы можем вычислить q^2(зная q^1 и q^0). Для вычисления q^1 (q^0 известно из начального условия) воспользуемся неявной двухслойной схемой.

Выполняя конечноэлементную аппроксимацию краевой задачи параболического уравнения, получим СЛАУ следующего вида:

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

- **Вариационная постановка задачи в форме Галеркина**

Тепловая задача определяется дифференциальным уравнением эллиптического типа

$$-div(\lambda \cdot gradu) + \gamma u = f$$

Заданным в области Ω с границей $S = S_1 \cup S_2$ и краевыми условиями:

$$u|_{S_1} = u_g$$

$$\lambda\frac{\partial u}{\partial n}\Big|_{S_2} = \theta$$

Будем называть пространством H^m множество функций v, которые вместе со своими производными до m-го порядка включительно интегрируемы с квадратом на Ω.

Потребуем, чтобы невязка $R(u) = -div(\lambda gradu) + \gamma u - f$ дифференциального уравнения была ортогональна (в смысле скалярного произведения пространства) некоторому пространству функций v, которое мы будем называть пространством пробных функций, т.е.

$$\int_\Omega (-div(\lambda \cdot gradu) + \gamma u - f)vd\Omega = 0 \quad \forall v \in \Phi$$

Воспользуемся формулой Грина интегрирования по частям:

$$\int_\Omega -div(\lambda \cdot gradu)vd\Omega = \int_\Omega \lambda \cdot gradu \cdot gradvd\Omega - \int_S \lambda\frac{\partial u}{\partial n}vdS$$

Получим уравнение в следующем виде:

$$\int_\Omega \lambda \cdot gradu \cdot gradvd\Omega - \int_S \lambda\frac{\partial u}{\partial n}vdS + \int_\Omega (\gamma u - f)vd\Omega = 0 \quad \forall v \in \Phi$$

Учитывая, что $S = S_1 \cup S_2$, получаем соотношение:

$$\int_S \lambda\frac{\partial u}{\partial n}vdS = \int_{S_1} \lambda\frac{\partial u}{\partial n}vdS + \int_{S_2} \lambda\frac{\partial u}{\partial n}vdS$$

В качестве Φ выберем H_0^1 – пространство пробных функций $v_0 \in H^1$, которые на границе S_1 удовлетворяют нулевым краевым условиям первого рода. При этом будем считать, что $u \in H_g^1$, H_g^1 – множество функций, имеющихся только первым краевым условиям на границе S_1.

С учетом этого воспользуемся заданными краевыми условиями:

$$v_0|_{S_1} = 0 \Rightarrow \int_{S_1} \lambda\frac{\partial u}{\partial n}v_0 dS = 0$$

$$\lambda\frac{\partial u}{\partial n}\Big|_{S_2} = \theta \Rightarrow \int_{S_2} \lambda\frac{\partial u}{\partial n}v_0 dS = \int_{S_2} \theta v_0 dS$$

Тогда интегральное соотношение принимает вид:

$$\int_\Omega \lambda \cdot gradu \cdot gradv_0 d\Omega + \int_\Omega \gamma u v_0 d\Omega = \int_\Omega f v_0 d\Omega + \int_{S_2} \theta v_0 dS \quad \forall v_0 \in H_0^1$$

- **Конечноэлементная дискретизация**

Для построения конечноэлементной аппроксимации по методу Галеркина пространства H_g^1 и H_0^1 заменим конечномерными пространствами V_g^h и V_0^h. При этом функции из пространств V_g^h и V_0^h являются элементами одного и того же конечномерного пространства V^h, которое определяется как линейное пространство, натянутое на базисные функции ψ_i, $i = \overline{1,n}$.

Запишем аппроксимацию полученного уравнения Галеркина на конечномерных подпространствах V_g^h и V_0^h, аппроксимирующих исходные пространства H_g^1 и H_0^1. Для этого заменим функцию $u \in H_g^1$ аппроксимирующей ее функцией $u^h \in V_g^h$, а функцию $v_0 \in H^1$ – функцией $v_0^h \in V_0^h$:

$$\int_\Omega \lambda \cdot grad u^h \cdot grad v_0^h d\Omega + \int_\Omega \gamma u^h v_0^h d\Omega = \int_\Omega f v_0^h d\Omega + \int_{S_2} \theta v_0^h dS \quad \forall v_0^h \in V_0^h$$

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

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

то вариационное уравнение эквивалентно следующей системе уравнений:

$$\int_\Omega \lambda \cdot grad u^h \cdot grad \psi_i d\Omega + \int_\Omega \gamma u^h \psi_i d\Omega = \int_\Omega f \psi_i d\Omega + \int_{S_2} \theta \psi_i dS \quad i \in N_0$$

Так как МКЭ-решение $u^h \in V_g^h$, то оно может быть предоставлено в виде линейной комбинации базисных функций пространства V^h:

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

Подставляя, окончательно получаем СЛАУ для компонент q_j вектора q:

$$\sum_{j=1}^{n} \left(\int_\Omega \lambda \cdot grad \psi_j \cdot grad \psi_i r d\Omega + \int_\Omega \gamma \psi_j \psi_i r d\Omega \right) = \int_\Omega f \psi_i r d\Omega + \int_{S_2} \theta \psi_i r dS \quad i \in N_0$$

- **Решение задачи на треугольной сетке с кубическими базисными функциями иерархического типа**

Рассмотрим построение базисных кубических функций на каждом конечном элементе. Для определения 10 коэффициентов полинома третьей степени, необходимо задать 10 точек на каждом конечном элементе Ω_k (треугольнике).

Для вычисления компонент локальных матриц на треугольных элементах очень удобно использовать L-координаты.

Рассмотрим линейные базисные функции $L_1(r,z), L_2(r,z), L_3(r,z)$ на элементе Ω_k такие, что $L_1(r,z)$ равна единице в вершине (r_1, z_1) и нулю во всех остальных вершинах $L_2(r,z)$ равна единице в вершине (r_2, z_2) и нулю во всех остальных и т.д.

Любая линейная на Ω_k функция представима в виде линейной комбинации этих базисных линейных функций. Коэффициенты – это значения функции в каждой из вершин треугольника Ω_k.

На конечном элементе Ω_k иерархическим называют такой базис, который содержит все базисные функции этого элемента более низкого порядка.
Для треугольного конечного элемента иерархические базисные функции можно записать в следующем виде:
Первые три базисные функции – это L-координаты треугольника: $\psi_1 = L1$, $\psi_2 = L2$, $\psi_3 = L3$.
В элементе второго порядка добавляются три ассоциированные с ребрами и квадратичные на них базисные функции $\psi_4 = \mathrm{L}_1 * \mathrm{L}_2$, $\psi_5 = \mathrm{L}_1 * \mathrm{L}_3$,$\psi_6 = \mathrm{L}_3 * \mathrm{L}_2$.
В элементе третьего порядка добавляются три ассоциированных с ребрами и кубические на них функции $\psi_7 = L_1 * L_2$,$* (L_1 - L_2)$,$\psi_8 = L_1 * L_3$,$* (L_1 - L_3)$,$\psi_9 = L_2 * L_3 * (L_2 - L_3)$.
Центр масс: $\psi_{10} = L_1 * L_2 * L_3$.
Поскольку мы воспользовались L-координатами для представления кубических базисных функций, то можно не вычислять интегралы от их произведений по каждой области непосредственно, а воспользоваться общей формулой:
$\int_{\Omega_m} (L_1)^{\nu_1} \cdot (L_2)^{\nu_2} \cdot (L_3)^{\nu_3} d\Omega_m = \frac{\nu_1! \nu_2! \nu_3!}{(\nu_1+\nu_2+\nu_3+2)!} \cdot |det\, D|$,
где $|det\, D|$ – удвоенная площадь треугольника
$D = \begin{pmatrix} 1 & 1 & 1 \\ r_1 & r_2 & r_3 \\ z_1 & z_2 & z_3 \end{pmatrix}$ – матрица, составленная из координат его вершин.
Учитывая построение L-функций, можно записать следующие соотношения:

$$\begin{cases} L_1 + L_2 + L_3 = 1 \\ L_1 r_1 + L_2 r_2 + L_3 r_3 = r \\ L_1 z_1 + L_2 z_2 + L_3 z_3 = z \end{cases} \Rightarrow \begin{pmatrix} 1 & 1 & 1 \\ r_1 & r_2 & r_3 \\ z_1 & z_2 & z_3 \end{pmatrix} \begin{pmatrix} L_1 \\ L_2 \\ L_3 \end{pmatrix} = \begin{pmatrix} 1 \\ r \\ z \end{pmatrix}$$

Отсюда можно найти коэффициенты линейных функций: $L_1(r,z), L_2(r,z), L_3(r,z)$.

$$\begin{pmatrix} \alpha_0^1 & \alpha_1^1 & \alpha_2^1 \\ \alpha_0^2 & \alpha_1^2 & \alpha_2^2 \\ \alpha_0^3 & \alpha_1^3 & \alpha_2^3 \end{pmatrix} = D^{-1} = \begin{pmatrix} 1 & 1 & 1 \\ r_1 & r_2 & r_3 \\ z_1 & z_2 & z_3 \end{pmatrix}^{-1} = \begin{pmatrix} r_2 z_3 - r_3 z_2 & z_2 - z_3 & r_3 - r_2 \\ r_3 z_1 - r_1 z_3 & z_3 - z_1 & r_1 - r_3 \\ r_1 z_2 - r_2 z_1 & z_1 - z_2 & r_2 - r_1 \end{pmatrix} \cdot \frac{1}{detD}$$

Перейдём теперь к решению локальной задачи на каждом конечном элементе. Полученное уравнение для области Ω представим в виде суммы интегралов по областям Ω_k (без краевых условий), тогда на каждом конечном элементе будем решать свою локальную задачу построения матриц и вектора правой части.

$$\int_{\Omega_k} \lambda \cdot grad\psi_i \cdot grad\psi_j r d\Omega_k + \int_{\Omega_k} \gamma \psi_i \psi_j r d\Omega_k = \int_{\Omega_k} f \psi_i r d\Omega_k$$

Локальная матрица, согласно этому соотношению, будет состоять из суммы двух матриц жёсткости и массы. По числу базисных функций на каждом конечном элементе, она будем иметь размерность 10×10.
Также, чтобы ассоциированные с ребром функции не оказались разрывными необходимо ввести такую локальную нумерацию вершины на каждом конечном элементе, чтобы большему локальному номеру вершины обязательно соответствовал больший главный глобальный номер.

- **Построение матриц жёсткости и масс**

Для построения матрицы жёсткости рассмотрим интеграл:

$$\int_{\Omega_k} \lambda \cdot grad\psi_i \cdot grad\psi_j r d\Omega_k$$

Будем вычислять этот интеграл программно, выразив $grad\psi_i$ через L-координаты:

$$grad\psi_1 = grad L_1$$

$$grad\psi_2 = gradL_2$$
$$grad\psi_3 = gradL_3$$
$$grad\psi_4 = L_2 \cdot gradL_1 + L_1 \cdot gradL_2$$
$$grad\psi_5 = L_3 \cdot gradL_1 + L_1 \cdot gradL_3$$
$$grad\psi_6 = L_2 \cdot gradL_3 + L_3 \cdot gradL_2$$
$$grad\psi_7 = 2L_1L_2 \cdot gradL_1 - 2L_1L_2 \cdot gradL_2 - L_2^2 \cdot gradL_1 + L_1^2 \cdot gradL_2$$
$$grad\psi_8 = 2L_1L_3 \cdot gradL_1 - 2L_1L_3 \cdot gradL_3 - L_3^2 \cdot gradL_1 + L_1^2 \cdot gradL_3$$
$$grad\psi_9 = 2L_2L_3 \cdot gradL_2 - 2L_2L_3 \cdot gradL_3 - L_3^2 \cdot gradL_2 + L_2^2 \cdot gradL_3$$
$$grad\psi_{10} = L_2L_3 \cdot gradL_1 + L_1L_3 \cdot gradL_2 + L_1L_2 \cdot gradL_3$$

Для вычислений воспользуемся указанной выше формулой:

$$\int_{\Omega_m} (L_1)^{\nu_1} \cdot (L_2)^{\nu_2} \cdot (L_3)^{\nu_3} r d\Omega_m = \frac{\nu_1!\,\nu_2!\,\nu_3!}{(\nu_1 + \nu_2 + \nu_3 + 2)!} \cdot |det\, D|$$

$$gradL_k = \left(\alpha_1^k, \alpha_2^k\right)$$

При вычислении сначала один раз посчитаем значения интегралов от всех возможных произведений $grad\psi_i \cdot grad\psi_j$, не учитывая $det\, D$, λ и r (где r раскладывается по линейному базису, т.е. $r = r_1L_1 + r_2L_2 + r_3L_3$), построив тем самым шаблонную матрицу жёсткости. После этого для каждого треугольника до множим вычисленные значения интегралов на $det\, D$, λ и r.

Интегралы для матрицы масс будем вычислять программно по аналогичной схеме без $grad\psi_i \cdot grad\psi_j$, α_1^k и α_2^k, $\lambda \rightarrow \gamma$.

- **Построение вектора правой части**

Для построения локального вектора правой части будем использовать квадратуры. Для каждой квадратуры будем вычислять локальные координаты, значение функции и значения ψ. Вычисляем вклад по формуле:

$$\int_{\Omega} f(p)\psi_i(p)dp$$

И прибавляем этот вклад в локальный вектор по номеру i.

- **Сборка глобальной матрицы и вектора правой части**

Сборка глобальной матрицы для задачи на треугольной сетке представляет собой поиск места в глобальной матрице для каждого элемента из локальной. Будем собирать глобальную матрицу, ставя в соответствие элементу $A_{ij}^{\Omega_k}$ локальной матрицы элемент A_{mk}, где m, k – номера глобальных вершин i, j на данном элементе.

Аналогичные действия нужно сделать для сборки глобального вектора правой части.

- **Учет краевых условий первого рода**

Для учета краевых условий первого рода в глобальной матрице и глобальном векторе правой части нужно найти строку, соответствующую глобальному номеру узла, находящегося на ребре, считанном из файла boundaryCondition1.txt. После этого нужно поставить вместо диагонального элемента глобальной матрицы в этой строке большое число порядка 10^{50} , а в вектор правой части в этой строке - число порядка 10^{50} , умноженное на значение функции, соответствующее заданному номеру материала.

- **Учет краевых условий второго рода**

Краевое условие второго рода, заданное на ребре с номером n учитывается, как добавление вклада от ребра с номером n к n-ой компоненте глобального вектора.
Учет происходит для каждого ребра, на котором задано второе краевой условие, считанном из файла boundaryCondition2.txt. Вклад от ребра:

$$\int_{S_2} \theta \psi_i r dS$$

Значение коэффициента θ будем брать из массива по индексу, соответствующему заданному номеру материала.
Фактически, решая задачу учета краевых условий второго рода, мы переходим к решению одномерной задачи на ребре для того, чтобы в вектор правой части занести полученные результаты. В качестве базисных функций ребра выберем четыре ненулевые на данном ребре базисные функции из функций$\psi_i, i = \overline{1,10}$, описанных выше. Например, для $\psi_1,\ \psi_2, \psi_4, \psi_5$, учитывая, что $L_1 = 1 - \xi, L_2 = \xi$:

$$h_k \int_0^1 \Psi_1 r\, d\xi = h_k \int_0^1 L_1(r_1 L_1 + r_2 L_2)\, d\xi = h_k \int_0^1 ((1-\xi)^2 r_1 + r_2(1-\xi)\xi)\, d\xi =$$

$$= h_k\left(\frac{r_1\,(1-1)^3}{3} + \frac{r_2 1^2}{2} - \frac{r_2 1^3}{3} + \frac{r_1\,(1-0)^3}{3} - \frac{r_2 0^2}{2} + \frac{r_2 0^3}{3} = h_k\left(\frac{r_2}{6} + \frac{r_1}{3}\right)\right.$$

$$h_k \int_0^1 \Psi_2 r\, d\xi = h_k \int_0^1 L_2(r_1 L_1 + r_2 L_2)\, d\xi = h_k \int_0^1 \left((1-\xi)\xi r_1 + r_2 \xi^2\right) d\xi =$$

$$= h_k\left(r_1\left(\frac{1^2}{2} - \frac{1^3}{3}\right) + \frac{r_2 1^3}{3} - \left(r_1\left(\frac{0^2}{2} - \frac{0^3}{3}\right) + \frac{r_2 0^3}{3}\right)\right) = h_k\left(\frac{r_1}{6} + \frac{r_2}{3}\right)$$

$$h_k \int_0^1 \Psi_4 r\, d\xi = h_k \int_0^1 L_1 L_2(r_1 L_1 + r_2 L_2)\, d\xi = h_k \int_0^1 ((1-\xi)^2 \xi r_1 + r_2(1-\xi)\xi^2)\, d\xi =$$

$$= h_k(r_1(1 - 2 * 1^2 + 1^3) + r_2(1^2 - 1^3) - 0) = 0$$

$$h_k \int_0^1 \Psi_5 r\, d\xi = h_k \int_0^1 L_1 L_2(L_1 - L_2)(r_1 L_1 + r_2 L_2)\, d\xi =$$

$$= h_k \int_0^1 r_1\left({L_1}^3 L_2 - {L_2}^2 {L_1}^2\right) + r_2({L_2}^2 {L_1}^2 - {L_2}^3 L_1)\, d\xi = h_k\left(\frac{r_1}{60} - \frac{r_2}{60}\right)$$

$$h_k = \sqrt{(r_{2k} - r_{1k})^2 + (z_{2k} - z_{1k})^2}$$

- **Решение СЛАУ**

Для решения СЛАУ с полученной матрицей в плотном формате будем применять Метод Гаусса. В результате получим вектор q, элементы которого и будут искомые веса при базисных функциях.

- **Исходные данные в программе**

xy.txt – количество вершин, количество узлов на расчетной области, координаты вершин по возрастанию глобальных номеров;
m.txt – количество треугольников, узлы каждого треугольника;
boundaryCondition1.txt – узлы, на которых выполняются первые краевые условия;
boundaryCondition2.txt – узлы, на которых выполняются вторые краевые условия;
field.txt – параметры поля (коэффициенты диффузии, гамма, номер функции f, номер функции для вычисления начального приближения);
time.txt – сведения о времени (количество интервалов, начальное и конечное время).

Описание разработанных программ

```
// Матрица в разреженном формате
struct matrix
назначение:
   хранение нижнего треугольника разреженной матрицы
переменные:

      vector<int> ig - номера строк в которых стоят элементы
      vector<int> jg - номера столбцов в которых стоят элементы
      vector<double> ggl - элементы матрицы
      vector<double> di - диагональные элемнты

struct Nodes
назначение:
   Хранение узла.
переменные:
   double x, y – координаты вершины (если они есть)
   int globalNumber – глобальный номер


struct CalculationArea
назначение:
   Хранение значений поля.
переменные:
   double gamma - гамма
   vector<double> lambda- лямбда


struct BoundaryCondition1
назначение:
   Хранение краевых условий 1 типа.
переменные:
   Nodes node - узел
   int formulaNum – номер формулы
   double FormulaForCondition1() – значение краевого условия

struct BoundaryCondition2
назначение:
   Хранение краевых условий 2 типа.
переменные:
   Nodes node[4] - узлы
   int formulaNum – номер формулы
   double FormulaForCondition1() – значение краевого условия

struct GradL
назначение:
   Хранение векторов gradL.
переменные:
   double component_1 – 1 компонента вектора
   double component_2 – 2 компонента вектора
```

Переменные и массивы

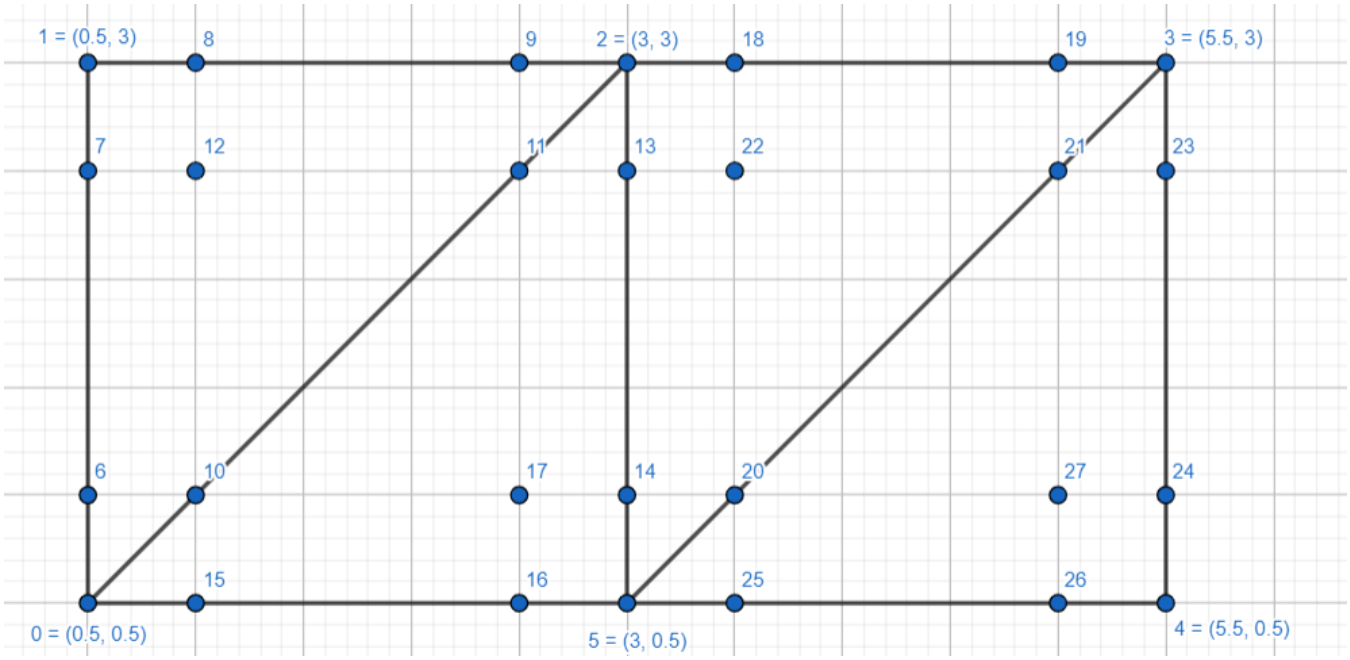
```
int n - число вершин
int m - число треугольников
int k – число узлов
vector<point> node - узлы
vector<vector<int>> triangle - трегульники (глобальные номера)
CalculationArea field - параметры поля
Matrix globalMatrixSolid – глобальная матрица
vector<double> globalVector - глобальный вектор
vector<vector<double>> localMatrix - локальная матрица
vector<double> localVector - локальный вектор
vector<BoundaryCondition1> boundCond1 - 1 краевые условия
vector<BoundaryCondition1> boundCond2 - 2 краевые условия
vector<Nodes> testNode - узлы для локальной матрицы и вектора (3 вершины трегульника)
vector<Nodes> nodeForGlobal – глобальные номера узлов, для перевода локальной в глобаль-
ную
double detD - определитель
double alpha[3][3] - матрица альфа
vector<vector<vector<int>>> psi - количество L в пси
vector<vector<double>> psiS - коэффициенты в пси
double gradLN[3] - значения градиента от L
vector<vector<vector<int>>> numL - кол-во L в каждом град от пси
double localHardMatrix [10][10] - матрица жесткости
double localMassMatrix[10][10] - матрица масс
double valIntegralDop - вспомогательная переменная для вычисления элементов матрицы жест-
кости
double valIntegral - вспомогательная переменная для вычисления элементов матрицы жестко-
сти
vector<int> sumVal - вспомогательная переменная для вычисления элементов матрицы жестко-
сти
vector<vector<GradL>> gradPsi - градиенты от пси
vector<vector<double>> numHard - коэффициенты для матрицы жесткости
double p1[21] – квадратуры (L1 = p1, L2 = p2, L3 = 1 - L2 - L1)
double p2[21] - квадратуры (L1 = p1, L2 = p2, L3 = 1 - L2 - L1)
double w[21] - веса
double LQ[3] - локальные координаты для каждой квадратуры
double rQ, zQ - значения r и z в каждой квадратуре
double s - вспомогательная переменная
vector<double> valuePsi - значения пси в каждой квадратуре
double buf - вспомогательная вершина для решения СЛАУ
double m – вспомогательная вершина для решения СЛАУ
vector<double> rs – вспомогательный вектор для решения интеграла
int currentTime - номер текущего времени
vector<double> t – время
Matrix globalMassMatrix- глобальная матрица масс
Matrix globalHardMatrix- глобальная матрица жесткости
Matrix A - глобальная матрица
vector<double> d - вектор правой части
vector<double> tempV - вспомогательный вектор
vector<vector<double>> APlot - глобальная матрица в плотном формате
double dt – шаг по времени
double dt – dt для трехслойной
double dt1 – dt1 для трехслойной
double dt0 – dt0 для трехслойной
vector<vector<double>> q - решения на времени
```

Описание основных модулей программы

```
double Function – Значение функции
void InputData - Чтение исходных данных
void InputBoundaryConditions - Чтение краевых условий
void Factorial - Подсчет факториала
void CalculateIntegralL - Подсчет интеграла от L
void CalculateBC1 - Учет краевых условий 1 рода
void CalculateBC2 - Учет краевых условий 2 рода
void CalculateLocalMatrix - Генерация локальной матрицы
void CalculateLocalVector - Генерация локального вектора
void MethodGauss – Метод Гаусса для решения СЛАУ
bool CheckInGlobal - Проверка на наличие в треугольнике
void TransformToSolidFormat - Перевод в плотный форрмат
void LocaltoGlobal - Перевод из локального в глобальное
vector<double> MultiplyVectorAndMatrix - Перемножение матрицы и вектора
void CalculateSchemes - Вычисления схем
void CalculateTwoLayerScheme - Двухслойная схема
void CalculateThreeLayerScheme - Трехслойная схема
```

- Тестирование программы

Сетка:



- Тест 1

  Назначение теста:

  Неразрывный, зависящий от времени $u = x^2 * t$

  Входные данные:

  $\lambda = 3, \quad \sigma = 1, \quad f = -6t + x^2, \qquad t \in [0; 2], \qquad \Delta t = 1$

  Краевые условия:

  - Первого рода на ребрах:

    [0, 1]: $u = 0.25 * t$

    [3, 4] $u = 30.25 * t$

Результат:

q0	q1	q2
0 (0)	0.25 (0)	0.5 (0)
0 (1)	0.25 (1)	0.5 (1)
0 (2)	9 (2)	18 (2)
0 (3)	30.25 (3)	60.5 (3)
0 (4)	30.25 (4)	60.5 (4)
0 (5)	9 (5)	18 (5)
0 (6)	2.50868e-50 (6)	4.71354e-50 (6)
0 (7)	1.44493e-64 (7)	3.32145e-64 (7)
0 (8)	-6.25 (8)	-12.5 (8)
0 (9)	3.76042e-13 (9)	9.69871e-13 (9)
0 (10)	-6.25 (10)	-12.5 (10)
0 (11)	3.55416e-13(11)	1.07033e-12 (11)
0 (12)	2.65824e-13 (12)	8.51832e-13 (12)
0 (13)	4.3785e-13 (13)	-1.46938e-13 (13)
0 (14)	-3.82479e-13 (14)	-4.93158e-14 (14)
0 (15)	-6.25 (15)	-12.5 (15)
0 (16)	3.22605e-13 (16)	1.05705e-12 (16)
0 (17)	-1.35117e-12 (17)	-1.39339e-12 (17)
0 (18)	-6.25 (18)	-12.5 (18)
0 (19)	1.1379e-12 (19)	9.09808e-13 (19)
0 (20)	-6.25 (20)	-12.5 (20)
0 (21)	-2.46835e-12 (21)	-1.57529e-12 (21)
0 (22)	2.37087e-12 (22)	2.41712e-12 (22)
0 (23)	-1.87413e-49 (23)	-4.40365e-49 (23)
0 (24)	-5.20139e-64 (24)	-3.63043e-64 (24)
0 (25)	-6.25 (25)	-12.5 (25)
0 (26)	-2.01294e-12 (26)	-1.4024e-12 (26)
0 (27)	-2.92256e-12 (27)	-2.12427e-12 (27)

- Тест 2

  Назначение теста:

  Разрывные коэффициенты, независящий от времени $u = \begin{cases} 3x^2, \ x \in [0.5, 3] \\ x^2 + 18, \ x \in [3, 5.5] \end{cases}$

  Входные данные:

  $\lambda = \begin{cases} 1, \ \Omega_1 \\ 3, \ \Omega_2 \end{cases}, \quad \sigma = 1, \quad f = \begin{cases} -6, \ \Omega_1 \\ -6, \ \Omega_2 \end{cases}$

  Краевые условия:

  - Первого рода на ребрах:

    [0, 1]: $u = 0.75$

    [3, 4] $u = 48.25$

Результат:

q0	q1	q2
0.75 (0)	0.75 (0)	0.75 (0)
0.75 (1)	0.75 (1)	0.75 (1)
27 (2)	27 (2)	27 (2)
48.25 (3)	48.25 (3)	48.25 (3)
48.25 (4)	48.25 (4)	48.25 (4)
27 (5)	27 (5)	27 (5)
-9.11458e-51 (6)	1.90104e-50 (6)	1.44531e-50 (6)
-4.74862e-65 (7)	-1.611e-66 (7)	3.00337e-64 (7)
-18.75 (8)	-18.75 (8)	-18.75 (8)
-3.73789e-11 (9)	8.61225e-13 (9)	3.05701e-12 (9)
-18.75 (10)	-18.75 (10)	-18.75 (10)
-3.08462e-11 (11)	6.30232e-13 (11)	3.51587e-12 (11)
1.69113e-11 (12)	2.02433e-12 (12)	2.77787e-12 (12)
9.54529e-11 (13)	3.00812e-12 (13)	-2.52908e-12 (13)
-8.99235e-11 (14)	-2.00982e-12 (14)	1.46222e-12 (14)
-18.75 (15)	-18.75 (15)	-18.75 (15)
-1.10506e-10 (16)	-1.23799e-13 (16)	3.89553e-12 (16)
-8.61015e-11 (17)	-5.61321e-12 (17)	-1.72759e-12 (17)
-6.25 (18)	-6.25 (18)	-6.25 (18)
3.85439e-11 (19)	2.21384e-12 (19)	-1.70586e-12 (19)
-6.25 (20)	-6.25 (20)	-6.25 (20)
-1.84943e-10 (21)	-6.02753e-12 (21)	4.45308e-12 (21)
1.7633e-10 (22)	7.92401e-13 (22)	-5.19951e-13 (22)
-1.12413e-49 (23)	-2.34288e-49 (23)	-2.90495e-49 (23)
-3.08226e-64 (24)	-1.40388e-63 (24)	6.77514e-64 (24)
-6.25 (25)	-6.25 (25)	-6.25 (25)
-2.19627e-10 (26)	-4.00352e-12 (26)	3.26884e-12 (26)
-1.2386e-10 (27)	-5.63568e-12 (27)	4.85691e-12 (27)

- Тест 3

  Назначение теста:

  Разрывные коэффициенты, зависящий от времени

$$u = \begin{cases} x^3 t + 3xt + 5t, & x \in [0.5, 3] \\ 3x^2 t + 14t, & x \in [3, 5.5] \end{cases}$$

  Входные данные:

$$\lambda = \begin{cases} 3, \ \Omega_1 \\ 5, \ \Omega_2 \end{cases}, \quad \sigma = 1, \quad f = \begin{cases} x^3 + 3x + 5 - 18xt, \ \Omega_1 \\ 3x^2 - 30t + 14, \ \Omega_2 \end{cases}, \quad t \in [1; 3], \qquad \Delta t = 1$$

  Краевые условия:

  - Первого рода на ребрах:

    [0, 1]: $u = 6.625t$

    [3, 4] $u = 104.75t$

Результат:

q0	q1	q2
6.625 (0)	13.25 (0)	19.875 (0)
6.625 (1)	13.25 (1)	19.875 (1)
41 (2)	82 (2)	123 (2)
104.75 (3)	209.5 (3)	314.25 (3)
104.75 (4)	209.5 (4)	314.25 (4)
41 (5)	82 (5)	123 (5)
-2.49256e-50 (6)	1.37649e-49 (6)	1.69085e-49 (6)
-1.27016e-64 (7)	1.21042e-63 (7)	2.43583e-63 (7)
-32.8125 (8)	-65.625 (8)	-98.4375 (8)
7.8125 (9)	15.625 (9)	23.4375 (9)
-32.8125 (10)	-65.625 (10)	-98.4375 (10)
7.8125 (11)	15.625 (11)	23.4375 (11)
15.625 (12)	31.25 (12)	46.875 (12)
1.6139e-10 (13)	1.04107e-12 (13)	-6.41176e-12 (13)
-1.56304e-10 (14)	-1.3534e-12 (14)	3.63621e-12 (14)
-32.8125 (15)	-65.625 (15)	-98.4375 (15)
7.8125 (16)	15.625 (16)	23.4375 (16)
-15.625 (17)	-31.25 (17)	-46.875 (17)
-18.75 (18)	-37.5 (18)	-56.25 (18)
6.83018e-11 (19)	3.88585e-12 (19)	-1.7441e-12 (19)
-18.75 (20)	-37.5 (20)	-56.25 (20)
-3.50047e-10 (21)	-9.19464e-12 (21)	6.20193e-12 (21)
3.33423e-10 (22)	1.02526e-11 (22)	7.58679e-12 (22)
-2.33073e-49 (23)	-1.6849e-48 (23)	-2.87695e-48 (23)
-6.72726e-64 (24)	-3.50386e-63 (24)	2.08722e-63 (24)
-18.75 (25)	-37.5 (25)	-56.25 (25)
-4.26403e-10 (26)	-7.33299e-12 (26)	3.22331e-12 (26)
-2.79236e-10 (27)	-1.05664e-11 (27)	4.1907e-12 (27)

- Приложение (текст программы)

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <algorithm>
#include <set>
using namespace std;
// время
vector<double> t;
// номер текущего времени
int currentTime = 0;
int nForQ0 = 1; // 1 q1=two , 2 q1=two slayer


#pragma region Структуры
struct matrix
{
// номера строк в которых стоят элементы
vector<int> ig;
// номера столбцов в которых стоят элементы
vector<int> jg;
// элементы матрицы
vector<double> ggl;
// диагональные элемнты
vector<double> di;
};
// Узел
struct nodes
{
double x, y;
int globalNumber;
};
// Расчетная область
struct field
{
double sigma;
vector<double> lyambda;
};
// Краевые условия 1

// градиент от L
struct gradL
{
double comp1;
double comp2;
};
#pragma endregion

#pragma region Функции и краевые
```

```
struct boundaryCondition1
{
nodes node;
int formulaNum;
double FormulaForCondition1()
{
switch (formulaNum)
{
case 0:
return 0;
case 1:
return 3;//test 1
case 2:
return 6.625 * t[currentTime]; //test 2
case 3:
return 104.75 * t[currentTime]; //test 2
}
}
};
// Краевые условия 2
struct boundaryCondition2
{
nodes node[4]; // номера узлов грани
int formulaNum;
double FormulaForCondition2()
{
switch (formulaNum)
{
case(0):
return 0;
default:
break;
}
}
};
double CalculateFunction(double x, double y, int numberTriangle)
{
if (currentTime < nForQ0) // u
{
switch (numberTriangle)
{
case (0):
return pow(x, 3) * t[currentTime] + 3 * x * t[currentTime] + 5 * t[currentTime];
case (1):
return pow(x, 3) * t[currentTime] + 3 * x * t[currentTime] + 5 * t[currentTime];
case (2):
return 3 * pow(x, 2) * t[currentTime] + 14 * t[currentTime];
case (3):
return 3 * pow(x, 2) * t[currentTime] + 14 * t[currentTime];
// тест 1 - неразрывный
//case (0):
```

```
//  return x * x * t[currentTime];
//case (1):
//  return x * x * t[currentTime];
//case (2):
//  return x * x * t[currentTime];
//case (3):
//  return x * x * t[currentTime];

//// тест 2 - разрывные без времени
//case 0:
//  return 3*pow(x,2);
//case 1:
//  return 3 * pow(x, 2);
//case 2:
//  return pow(x,2)+18;
//case 3:
//  return pow(x, 2) + 18;

}
}

else // f
{
switch (numberTriangle)
{
case 0:
return pow(x, 3) + 3 * x + 5 - 18 * x * t[currentTime];
case 1:
return pow(x, 3) + 3 * x + 5 - 18 * x * t[currentTime];
case 2:
return 3 * pow(x, 2) - 30 * t[currentTime] + 14;
case 3:
return 3 * pow(x, 2) - 30 * t[currentTime] + 14;
// тест 1 - неразрывный
//case (0):
//  return -6 * t[currentTime] + x * x;
//case (1):
//  return -6 * t[currentTime] + x * x;
//case (2):
//  return -6 * t[currentTime] + x * x;
//case (3):
//  return -6 * t[currentTime] + x * x;

//case 0:
//  return 6*pow(y,2)-12;
//case 1:
//  return 6 * pow(y, 2) - 12;
//case 2:
//  return  3 * pow(y, 2) - 2 * y + 21;
//case 3:
//  return  3 * pow(y, 2) - 2 * y + 21;
```

```
}
}

}
#pragma endregion


#pragma region ПортретнаМатрица(Разряженная)
matrix portrait(vector<vector<int>>& triangle, int n)
{
vector<set<int>> connection(n);
for (auto& lay : triangle)
{
for (int i = 1; i < 10; i++)
for (int j = 0; j < i; j++)
{
int a = lay[i];
int b = lay[j];
if (a < b)
swap(a, b);
connection[a].insert(b);
}
}
matrix globalMatrix;
globalMatrix.ig.resize(n + 1);
int* IA = &globalMatrix.ig[0];
IA[0] = IA[1] = 1;
for (int i = 2; i <= n; i++)
{
int col = IA[i - 1];
IA[i] = col + connection[i - 1].size();
}
globalMatrix.jg.resize(IA[n]);
int* JA = &globalMatrix.jg[0];
for (int i = 1, k = 0; i < n; i++)
for (int j : connection[i])
{
JA[k] = j;
k++;
}
globalMatrix.di.resize(n);
globalMatrix.ggl.resize(IA[n]);
return globalMatrix;
}
// Чтение портрета глобальной матрицы
void InputPortrait(vector<int>& ig, vector<int>& jg, int n)
{
ifstream igIn("ig.txt");
for (int i = 0; i < n + 1; i++)
igIn >> ig[i];
```

```
igIn.close();
jg.resize(ig[n] - 1);
ifstream jgIn("jg.txt");
for (int i = 0; i < ig[n] - 1; i++)
jgIn >> jg[i];
jgIn.close();
}
#pragma endregion

#pragma region IO
// Чтение исходных данных
void InputData(int& n, int& m, int& k, vector<nodes>& node, vector<vector<int>>& tri-
angle, field& field)
{
// сведения об узлах
ifstream input("xy.txt");
input >> n;
input >> k;
node.resize(n);
for (int i = 0; i < n; i++)
{
input >> node[i].x >> node[i].y;
node[i].globalNumber = i;
}
input.close();
// сведения о треугольниках
input.open("m.txt");
input >> m;
triangle.resize(m, vector<int>(10, 0));
for (int i = 0; i < m; i++)
for (int j = 0; j < 10; j++)
input >> triangle[i][j];
input.close();
// сведения о параметрах поля
input.open("field.txt");
input >> field.sigma;
field.lyambda.resize(m);
for (int i = 0; i < m; i++)
input >> field.lyambda[i];
input.close();
// чтение времени
int numTime = 0;
input.open("time.txt");
input >> numTime;
t.resize(numTime, 0);
for (int i = 0; i < numTime; i++)
input >> t[i];
input.close();
}
// Чтение краевых условий
```

```
void InputBoundaryCondition(vector<nodes>& nodes, vector<boundaryCondition1>& condi-
tion1, vector<boundaryCondition2>& condition2)
{
ifstream BC1In("boundaryCondition1.txt");
int n1;
BC1In >> n1;
condition1.resize(n1);
for (int i = 0; i < n1; i++)
BC1In >> condition1[i].node.globalNumber >> condition1[i].formulaNum;
BC1In.close();
ifstream BC2In("boundaryCondition2.txt");
int n2;
BC2In >> n2;
condition2.resize(n2);
for (int i = 0; i < n2; i++)
{
for (int j = 0; j < 4; j++)
BC2In >> condition2[i].node[j].globalNumber;
BC2In >> condition2[i].formulaNum;
for (int j = 0; j < 2; j++)
{
condition2[i].node[j].x =
nodes[condition2[i].node[j].globalNumber].x;
condition2[i].node[j].y =
nodes[condition2[i].node[j].globalNumber].y;
}
}
BC2In.close();
}
#pragma endregion

#pragma region Функции для работы с Матрицами и Векторами
// Подсчет факториала
void factorial(int num, double& res)
{
res = 1;
for (int i = 2; i <= num; i++)
res *= i;
}
// Скалярное произведение
double scalar(gradL g1, gradL g2)
{
return g1.comp1 * g2.comp1 + g1.comp2 * g2.comp2;
}
// Метод Гаусса
void MethodGauss(int N, vector<vector<double>> matrix, vector<double>& F)
{
for (int i = 1; i < N; i++)
{
for (int j = i; j < N; j++)
{
```

```
double m = matrix[j][i - 1] / matrix[i - 1][i - 1];
for (int k = 0; k < N; k++)
matrix[j][k] = matrix[j][k] - m * matrix[i - 1][k];
F[j] = F[j] - m * F[i - 1];
}
}
for (int k = N - 1; k >= 0; k--)
{
double buf = 0;
for (int j = k + 1; j < N; j++)
buf += matrix[k][j] * F[j];
F[k] = (F[k] - buf) / matrix[k][k];
}
}
bool CheckInGlobalMatrix(int num, vector<int> vectorNode, int& n)
{
for (int i = 0; i < vectorNode.size(); i++)
{
if (vectorNode[i] == num)
{
n = i;
return true;
}
}
return false;
}
void TransformMatrixToSolidFormat(matrix gl, vector<vector<double>>& matrixProf)
{
for (int i = 0; i < gl.ig.size() - 1; i++)
{
int num = 0;
for (int j = gl.ig[i] - 1; num < gl.ig[i + 1] - gl.ig[i]; j++, num++)
{
matrixProf[i][gl.jg[j]] = gl.ggl[j];
matrixProf[gl.jg[j]][i] = gl.ggl[j];
}
}
for (int i = 0; i < matrixProf.size(); i++)
matrixProf[i][i] = gl.di[i];
}
vector<nodes> VertexNumbering(vector<nodes> node, vector<vector<int>> triangle, int v)
{
vector<nodes> returnNode(3);
for (int i = 0; i < 3; i++)
{
returnNode[i].globalNumber = triangle[v][i];
returnNode[i].x = node[returnNode[i].globalNumber].x;
returnNode[i].y = node[returnNode[i].globalNumber].y;
}
return returnNode;
}
```

```
// Перемножение матрицы и вектора
vector<double> MultiplyVectorAndMatrix(matrix matrixGlobal, vector<double> vect, dou-
ble dt)
{
vector<double> res(vect.size(), 0);
for (int i = 0; i < vect.size(); i++)
{
res[i] = dt * matrixGlobal.di[i] * vect[i];
for (int j = matrixGlobal.ig[i]; j < matrixGlobal.ig[i + 1]; j++)
{
res[i] += dt * matrixGlobal.ggl[j - 1] * vect[matrixGlobal.jg[j - 1]];
res[matrixGlobal.jg[j - 1]] += dt * matrixGlobal.ggl[j - 1] * vect[i];
}
}
return res;
}
#pragma endregion

#pragma region Учет Краевых условий
// Учет краевых условий 1 рода
void CalculateFirstBoundaryCondition(vector<boundaryCondition1> condition, matrix&
globalMatrix, vector<double>& globalVector)
{
for (int i = 0; i < condition.size(); i++)
{
globalMatrix.di[condition[i].node.globalNumber] = 1.0E50;
globalVector[condition[i].node.globalNumber] = 1.0E50 * condition[i].FormulaForCondi-
tion1();
}
}
// Учет краевых условий 2 рода
void CalculateSecondBoundaryCondition(vector<boundaryCondition2> condition, vec-
tor<double>& globalVector)
{
double h = 0;
for (int i = 0; i < condition.size(); i++)
{
h = sqrt((condition[i].node[1].x - condition[i].node[0].x) * (condition[i].node[1].x -
condition[i].node[0].x)
+ (condition[i].node[1].y - condition[i].node[0].y) * (condition[i].node[1].y - condi-
tion[i].node[0].y)) * condition[i].FormulaForCondition2();
globalVector[condition[i].node[0].globalNumber] += h * ((condition[i].node[1].x / 6.0)
+ (condition[i].node[0].x / 3.0));
globalVector[condition[i].node[1].globalNumber] += h * ((condition[i].node[0].x / 6.0)
+ (condition[i].node[1].x / 3.0));
globalVector[condition[i].node[3].globalNumber] += h * ((condition[i].node[0].x /
60.0) - (condition[i].node[1].x / 60.0));
}
}
#pragma endregion
```

```
#pragma region Генерация Локальной матрицы
// Подсчет интеграла от L
void CalculateIntegralOfL(double sign, vector<int> val, double& res, vector<nodes>
node, double detD)
{
vector<double> rs(4, 0);
res = 0;
for (int i = 0; i < node.size(); i++)
{
++val[i];
factorial(val[0], rs[0]);
factorial(val[1], rs[1]);
factorial(val[2], rs[2]);
factorial(val[0] + val[1] + val[2] + 2, rs[3]);
res += (rs[0] * rs[1] * rs[2]) / (rs[3]) * sign * detD;
--val[i];
}
}
// Генерация локальной матрицы
void CalculateLocalMatrix(vector<nodes> node, field field, vector<vector<double>>& lo-
calMassMatrix, vector<vector<double>>& localHardMatrix)
{
// detD
double detD = (node[1].x - node[0].x) * (node[2].y - node[0].y) - (node[2].x - node
[0].x) * (node[1].y - node[0].y);
// матрица альфа
double alpha[3][3] =
{
{(node[1].x * node[2].y - node[2].x * node[1].y) / detD, (node[1].y - node
[2].y) / detD, (node[2].x - node[1].x) / detD},
{(node[2].x * node[0].y - node[0].x * node[2].y) / detD, (node[2].y - node
[0].y) / detD, (node[0].x - node[2].x) / detD},
{(node[0].x * node[1].y - node[1].x * node[0].y) / detD, (node[0].y - node
[1].y) / detD, (node[1].x - node[0].x) / detD}
};
detD = abs(detD);
// количество L в пси
vector<vector<vector<int>>> psi =
{
{{ 1,0,0 }},
{{ 0,1,0 }},
{{ 0,0,1 }},
{{ 1,1,0 }},
{{ 1,0,1 }},
{{ 0,1,1 }},
{{ 2,1,0 },{ 1,2,0 }},
{{ 2,0,1 },{ 1,0,2 }},
{ { 0,2,1 },{ 0,1,2 }},
{{ 1,1,1 }}
};
vector<vector<double>> psiS =
```

```
{
{{ 1 }},
{{ 1 }},
{{ 1 }},
{{ 1 }},
{{ 1 }},
{{ 1 }},
{{ 1 },{ -1 }},
{{ 1 },{ -1 }},
{{ 1 },{ -1 }},
{{ 1 }}
};
// значения градиента от L
vector<gradL> gradLN(3);
for (int i = 0; i < 3; i++)
{
gradLN[i].comp1 = alpha[i][1];
gradLN[i].comp2 = alpha[i][2];
}
// кол-во L в каждом град от пси
vector<vector<vector<int>>> numL =
{
{{ 0,0,0 }},
{{ 0,0,0 }},
{{ 0,0,0 }},
{{ 0,1,0 }, { 1,0,0 }},
{{ 0,0,1 }, { 1,0,0 }},
{{ 0,1,0 }, { 0,0,1 }},
{{ 1,1,0 }, { 1,1,0 }, { 0,2,0 }, { 2,0,0 }},
{{ 1,0,1 }, { 1,0,1 }, { 0,0,2 }, { 2,0,0 }},
{{ 0,1,1 }, { 0,1,1 }, { 0,0,2 }, { 0,2,0 }},
{{ 0,1,1 }, { 1,0,1 }, { 1,1,0 }}
};
// вспомогательнае переменные для вычисления элементов матрицы жесткости
double valIntegralDop, valIntegral = 0;
vector<int> sumVal(3, 0);
// град от пси
vector<vector<gradL>> gradPsi =
{
{gradLN[0]},
{gradLN[1]},
{gradLN[2]},
{{gradLN[0]}, {gradLN[1]}},
{{gradLN[0]}, {gradLN[2]}},
{{gradLN[2]}, {gradLN[1]}},
{{gradLN[0]}, {gradLN[1]}, {gradLN[0]}, {gradLN[1]}},
{ {gradLN[0]}, {gradLN[2]}, {gradLN[0]}, {gradLN[2]}},
{{gradLN[1]}, {gradLN[2]}, {gradLN[1]}, {gradLN[2]}},
{{gradLN[0]}, {gradLN[1]}, {gradLN[2]}}
};
// коэффиценты для матрицы жесткости
```

```
vector<vector<double>> numHard =
{
{1},
{1},
{1},
{{1}, {1}},
{{1}, {1}},
{{1}, {1}},
{{2}, {-2}, {-1}, {1}},
{{2}, {-2}, {-1}, {1}},
{{2}, {-2}, {-1}, {1}},
{{1}, {1}, {1}}
};
// алгоритм вычисления элементов матрицы жесткости
for (int i = 0; i < 10; i++)
{
for (int j = i; j < 10; j++)
{
for (int v = 0; v < numL[i].size(); v++)
{
for (int k = 0; k < numL[j].size(); k++)
{
sumVal[0] = numL[i][v][0] + numL[j][k][0];
sumVal[1] = numL[i][v][1] + numL[j][k][1];
sumVal[2] = numL[i][v][2] + numL[j][k][2];
CalculateIntegralOfL(numHard[i][v] * numHard[j][k], sumVal, valIntegralDop, node,
detD);
valIntegral += valIntegralDop * scalar(gradPsi[i][v], gradPsi[j][k]);
}
}
localHardMatrix[i][j] = valIntegral;
localHardMatrix[j][i] = valIntegral;
valIntegral = 0;
}
}
// алгоритм вычисления элементов матрицы масс
for (int i = 0; i < 10; i++)
{
for (int j = i; j < 10; j++)
{
for (int v = 0; v < psi[i].size(); v++)
{
for (int k = 0; k < psi[j].size(); k++)
{
sumVal[0] = psi[i][v][0] + psi[j][k][0];
sumVal[1] = psi[i][v][1] + psi[j][k][1];
sumVal[2] = psi[i][v][2] + psi[j][k][2];
CalculateIntegralOfL(psiS[i][v] * psiS[j][k], sumVal, valIntegralDop, node, detD);
valIntegral += valIntegralDop;
}
}
```

```
localMassMatrix[i][j] = valIntegral;
localMassMatrix[j][i] = valIntegral;
valIntegral = 0;
}
}
}
// Генерация локального вектора
void CalculateLocalVector(vector<nodes> node, vector<double>& localVector, int number-
Triangle)
{
// квадратуры
// L1 = p1, L2 = p2, L3 = 1 - L2 - L1
double p1[21] =
{ 0.0451890097844, 0.0451890097844, 0.9096219804312, 0.7475124727339,
0.2220631655373, 0.7475124727339,
0.2220631655373, 0.0304243617288, 0.0304243617288, 0.1369912012649,
0.6447187277637, 0.1369912012649, 0.2182900709714,
0.2182900709714, 0.6447187277637, 0.0369603304334, 0.4815198347833,
0.4815198347833, 0.4036039798179, 0.4036039798179,
0.1927920403641
};
double p2[21] =
{ 0.0451890097844, 0.9096219804312, 0.0451890097844, 0.0304243617288,
0.0304243617288, 0.2220631655373,
0.7475124727339, 0.7475124727339, 0.2220631655373, 0.2182900709714,
0.2182900709714, 0.6447187277637, 0.6447187277637,
0.1369912012649, 0.1369912012649, 0.4815198347833, 0.0369603304334,
0.4815198347833, 0.1927920403641, 0.4036039798179,
0.4036039798179
};
// веса
double w[21] =
{ 0.0519871420646, 0.0519871420646, 0.0519871420646, 0.0707034101784,
0.0707034101784, 0.0707034101784,
0.0707034101784, 0.0707034101784, 0.0707034101784, 0.0909390760952,
0.0909390760952, 0.0909390760952, 0.0909390760952,
0.0909390760952, 0.0909390760952, 0.1032344051380, 0.1032344051380,
0.1032344051380, 0.1881601469167, 0.1881601469167,
0.1881601469167
};
// локальные координаты для каждой квадратуры
double LQ[3];
// значения r и z в каждой квадратуре
double xQ = 0, yQ = 0;
// вспомогательная переменная
double s = 0;
double detD = abs((node[1].x - node[0].x) * (node[2].y - node[0].y) -
(node[2].x - node[0].x) * (node[1].y - node[0].y));
for (int i = 0; i < 21; i++)
{
LQ[0] = p1[i];
```

```
LQ[1] = p2[i];
LQ[2] = 1 - LQ[1] - LQ[0];
xQ = node[0].x * LQ[0] + node[1].x * LQ[1] + node[2].x * LQ[2];
yQ = node[0].y * LQ[0] + node[1].y * LQ[1] + node[2].y * LQ[2];
s = 0.25 * w[i] * CalculateFunction(xQ, yQ, numberTriangle);
//s = 0.25 * w[i] * fun(rQ, zQ);
// значения пси в каждой квадратуре
vector<double> valuePsi =
{
{LQ[0]},
{LQ[1]},
{LQ[2]},
{LQ[0] * LQ[1]},
{LQ[0] * LQ[2]},
{LQ[1] * LQ[2]},
{LQ[0] * LQ[1] * (LQ[0] - LQ[1])},
{LQ[0] * LQ[2] * (LQ[0] - LQ[2])},
{LQ[1] * LQ[2] * (LQ[1] - LQ[2])},
{LQ[0] * LQ[1] * LQ[2]}
};
for (int j = 0; j < 10; j++)
localVector[j] += valuePsi[j] * s * detD;
}
}

#pragma endregion


// Перевод из локального в глобальное
void TransformLocalMatrixToGlobalMatrix(matrix& globalMassMatrix, matrix& globalHard-
Matrix, vector<double>& globalVector, vector<nodes> node, vector<vector<int>> trian-
gle, field omega, int n, int m, int k)
{
// локальная матрица масс
vector<vector<double>> localMassMatrix(10, vector<double>(10, 0));
// локальная матрица жесткости
vector<vector<double>> localHardMatrix(10, vector<double>(10, 0));
// локальный вектор
vector<double> localVector(10, 0);
// 1 краевые условия
vector<boundaryCondition1> boundCond1;
// 2 краевые условия
vector<boundaryCondition2> boundCond2;
// v - кол-во треугольников
for (int v = 0; v < m; v++)
{
// testNode - узлы для локальной матрицы и вектора(3 вершины)
vector<nodes> localNode(3);
localNode = VertexNumbering(node, triangle, v);
/*for (int i = 0; i < 3; i++)
{
```

```
localNode[i].globalNumber = triangle[v][i];
localNode[i].x = node[localNode[i].globalNumber].x;
localNode[i].y = node[localNode[i].globalNumber].y;
}*/
CalculateLocalMatrix(localNode, omega, localMassMatrix, localHardMatrix);

for (int i = 0; i < localHardMatrix.size(); i++)
for (int j = 0; j < localHardMatrix[i].size(); j++)
localHardMatrix[i][j] *= omega.lyambda[v];

CalculateLocalVector(localNode, localVector, v);
vector<int> nodeForGlobal(10, 0);
// вектор для перевода в глобальную матрицу
for (int i = 0; i < 10; i++)
nodeForGlobal[i] = triangle[v][i];
//sort(nodeForGlobal.begin(), nodeForGlobal.end());
for (int i = 0; i < globalMassMatrix.ig.size() - 1; i++)
{
int x, y;
int num = 0;
if (CheckInGlobalMatrix(i, nodeForGlobal, y))
{
for (int j = globalMassMatrix.ig[i] - 1; num < globalMassMatrix.ig[i + 1] - global-
MassMatrix.ig[i]; j++, num++)
{
if (CheckInGlobalMatrix(globalMassMatrix.jg[j], nodeForGlobal, x))
{
globalMassMatrix.ggl[j] += localMassMatrix[x][y];
globalHardMatrix.ggl[j] += localHardMatrix[x][y];
}
}
}
}
for (int i = 0; i < 10; i++)
{
globalMassMatrix.di[nodeForGlobal[i]] += localMassMatrix[i][i];
globalHardMatrix.di[nodeForGlobal[i]] += localHardMatrix[i][i];
}
// перевод из локального в глобальный вектор
for (int i = 0; i < 10; i++)
globalVector[nodeForGlobal[i]] += localVector[i];
// обнуление локальной матрицы (возможно лишнее)
for (int i = 0; i < 10; i++)
for (int j = 0; j < 10; j++)
{
localMassMatrix[i][j] = 0;
localHardMatrix[i][j] = 0;
}
// обнуление локального вектора (возможно лишнее)
for (int i = 0; i < 10; i++)
localVector[i] = 0;
```

```
}
}

#pragma region Нестационарная задача(зависимость от времени)
vector<double> CalculateFirstSchemes(vector<nodes> node, vector<vector<int>> triangle,
field field, int n, int m, int k)
{
matrix globalMatrix = portrait(triangle, k);
// глобальный вектор
vector<double> globalVector(k, 0);
// локальная матрица
vector<vector<double>> localMassMatrix(10, vector<double>(10, 0));
// локальная матрица
vector<vector<double>> localHardMatrix(10, vector<double>(10, 0));
// локальный вектор
vector<double> localVector(10, 0);
// v - кол-во треугольников
for (int v = 0; v < m; v++)
{
// testNode - узлы для локальной матрицы и вектора(3 вершины)
vector<nodes> localNode(3);
localNode = VertexNumbering(node, triangle, v);
//for (int i = 0; i < 3; i++)
//{
// testNode[i].globalNumber = triangle[v][i];
// testNode[i].x = node[testNode[i].globalNumber].x;
// testNode[i].y = node[testNode[i].globalNumber].y;
//}
// получаем локальную матрицу
CalculateLocalMatrix(localNode, field, localMassMatrix, localHardMatrix);

for (int i = 0; i < localHardMatrix.size(); i++)
for (int j = 0; j < localHardMatrix[i].size(); j++)
localHardMatrix[i][j] *= field.lyambda[v];
// получааем локальный вектор
CalculateLocalVector(localNode, localVector, v);
vector<int> globalNode(10, 0);
// вектор для перевода в глобальную матрицу
for (int i = 0; i < 10; i++)
globalNode[i] = triangle[v][i];
for (int i = 0; i < globalMatrix.ig.size() - 1; i++)
{
int x, y;
int num = 0;
if (CheckInGlobalMatrix(i, globalNode, y))
{
for (int j = globalMatrix.ig[i] - 1; num < globalMatrix.ig[i + 1] - globalMa-
trix.ig[i]; j++, num++)
{
if (CheckInGlobalMatrix(globalMatrix.jg[j], globalNode, x))
globalMatrix.ggl[j] += localMassMatrix[x][y];
```

```
}
}
}
for (int i = 0; i < 10; i++)
globalMatrix.di[globalNode[i]] += localMassMatrix[i][i];
// перевод из локального в глобальный вектор
for (int i = 0; i < 10; i++)
globalVector[globalNode[i]] += localVector[i];
// обнуление локальной матрицы (возможно лишнее)
for (int i = 0; i < 10; i++)
for (int j = 0; j < 10; j++)
{
localMassMatrix[i][j] = 0;
localHardMatrix[i][j] = 0;
}
// обнуление локального вектора (возможно лишнее)
for (int i = 0; i < 10; i++)
localVector[i] = 0;
}
// 1 краевые условия
vector<boundaryCondition1> boundCond1;
// 2 краевые условия
vector<boundaryCondition2> boundCond2;
// чтение краевыых условий
InputBoundaryCondition(node, boundCond1, boundCond2);
// учет краевых условий
//calculateBC2(boundCond2, globalVector);
CalculateFirstBoundaryCondition(boundCond1, globalMatrix, globalVector);
vector<vector<double>> globalMatrixPlot(k, vector<double>(k, 0));
TransformMatrixToSolidFormat(globalMatrix, globalMatrixPlot);
// решение слау
MethodGauss(k, globalMatrixPlot, globalVector); // globalVector - q0
return globalVector;
}
// Двухслойная схема
void CalculateTwoLayerSchemes(vector<vector<int>> triangle, field field, vector<nodes>
node, int k, int n, int m, vector<vector<double>>& q)
{
matrix globalMassMatrix = portrait(triangle, k);
matrix globalHardMatrix = portrait(triangle, k);
matrix A = portrait(triangle, k);
// глоабльный вектор
vector<double> globalVector(k, 0);
double dt = t[currentTime] - t[currentTime - 1];
TransformLocalMatrixToGlobalMatrix(globalMassMatrix, globalHardMatrix, globalVector,
node, triangle, field, n, m, k);
for (int i = 0; i < globalMassMatrix.di.size(); i++)
A.di[i] = globalHardMatrix.di[i] + (1 / dt) * globalMassMatrix.di[i] * field.sigma;
for (int i = 0; i < globalMassMatrix.ggl.size(); i++)
A.ggl[i] = globalHardMatrix.ggl[i] + (1 / dt) * globalMassMatrix.ggl[i] * field.sigma;
vector<double> d(k, 0);
```

```
vector<double> tempV(k, 0);
tempV = MultiplyVectorAndMatrix(globalMassMatrix, q[currentTime - 1], (field.sigma /
dt));
for (int i = 0; i < k; i++)
d[i] = globalVector[i] + tempV[i];
// учитывать краевые можно здесь
// 1 краевые условия
vector<boundaryCondition1> boundCond1;
// 2 краевые условия
vector<boundaryCondition2> boundCond2;
// чтение краевыых условий
InputBoundaryCondition(node, boundCond1, boundCond2);
// учет краевых условий
//calculateBC2(boundCond2, d);
CalculateFirstBoundaryCondition(boundCond1, A, d);
vector<vector<double>> APlot(k, vector<double>(k, 0));
TransformMatrixToSolidFormat(A, APlot);
MethodGauss(k, APlot, d);
q[currentTime] = d;
}
// Трехслойная схема
void CalculateThirdLayerSchemes(int num, vector<vector<int>> triangle, field field,
vector<nodes> node, int k, int n, int m, vector<vector<double>>& q)
{
matrix globalMassMatrix = portrait(triangle, k);
matrix globalHardMatrix = portrait(triangle, k);
matrix A = portrait(triangle, k);
double dt = t[currentTime] - t[currentTime - 2];
double dt1 = t[currentTime - 1] - t[currentTime - 2];
double dt0 = t[currentTime] - t[currentTime - 1];
// глоабльный вектор
vector<double> globalVector(k, 0);
TransformLocalMatrixToGlobalMatrix(globalMassMatrix, globalHardMatrix, globalVector,
node, triangle, field, n, m, k);
for (int i = 0; i < globalMassMatrix.di.size(); i++)
A.di[i] = ((dt + dt0) / (dt * dt0)) * globalMassMatrix.di[i] * field.sigma + glob-
alHardMatrix.di[i];
for (int i = 0; i < globalMassMatrix.ggl.size(); i++)
A.ggl[i] = ((dt + dt0) / (dt * dt0)) * globalMassMatrix.ggl[i] * field.sigma + glob-
alHardMatrix.ggl[i];
vector<double> d(k, 0);
vector<double> tempV0(k, 0);
tempV0 = MultiplyVectorAndMatrix(globalMassMatrix, q[num - 2], (field.sigma * dt0 /
(dt * dt1)));
vector<double> tempV1(k, 0);
tempV1 = MultiplyVectorAndMatrix(globalMassMatrix, q[num - 1], (field.sigma * dt /
(dt1 * dt0)));
for (int i = 0; i < k; i++)
d[i] = globalVector[i] - tempV0[i] + tempV1[i];
// учитывать краевые можно здесь
// 1 краевые условия
```

```
vector<boundaryCondition1> boundCond1;
// 2 краевые условия
vector<boundaryCondition2> boundCond2;
// чтение краевыых условий
InputBoundaryCondition(node, boundCond1, boundCond2);
// учет краевых условий
CalculateSecondBoundaryCondition(boundCond2, d);
CalculateFirstBoundaryCondition(boundCond1, A, d);
vector<vector<double>> AProf(k, vector<double>(k, 0));
TransformMatrixToSolidFormat(A, AProf);
MethodGauss(k, AProf, d);
q[num] = d;
}
void CalculateParabolicType()
{
// число вершин, число треугольников, кол-во узлов (читать из файла)
int n = 0, m = 0, k = 0;
// узлы
vector<nodes> node;
// трегульники (глобальные номера)
vector<vector<int>> triangle;
// параметры поля
field omega;
//cout.setf(ios::scientific);
//cout.precision(8);
// решения на времени
vector<vector<double>> q;
// ввод начальных данных
InputData(n, m, k, node, triangle, omega);
// временный вектор
vector<double> nullQ(k, 0);
// добавляем q0
q.push_back(CalculateFirstSchemes(node, triangle, omega, n, m, k));
cout << endl << "q0" << endl;
for (int i = 0; i < q[0].size(); i++)
cout << q[0][i] << " (" << i << ")" << endl;
currentTime++;
if (nForQ0 == 2)
{
// добавляем q1 точно
q.push_back(CalculateFirstSchemes(node, triangle, omega, n, m, k));
cout << endl << "q1" << endl;
for (int i = 0; i < q[1].size(); i++)
cout << q[1][i] << " (" << i << ")" << endl;
}
else
{
if (currentTime < t.size())
{
q.push_back(nullQ);
CalculateTwoLayerSchemes(triangle, omega, node, k, n, m, q);
```

```
cout << endl << "q1" << endl;
for (int i = 0; i < q[1].size(); i++)
cout << q[1][i] << " (" << i << ")" << endl;
}
}
for (int i = 2; i < t.size(); i++)
{
currentTime++;
q.push_back(nullQ);
CalculateThirdLayerSchemes(i, triangle, omega, node, k, n, m, q);
cout << endl << "q" << i << endl;
for (int j = 0; j < q[i].size(); j++)
cout << q[i][j] << " (" << j << ")" << endl;
}
}
#pragma endregion


int main()
{
CalculateParabolicType();
return 0;
}
```