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»

Факультет: ПМИ

Группа: ПМ-72

Студент: Камынин А.С.

Преподаватель: Персова М.Г.

Новосибирск
2021

1. Постановка задачи

1.1. Формулировка задания

МКЭ для двумерной тепловой задачи в цилиндрических координатах.

1.2. Постановка задачи

Имеется тепловая задача, описываема параболическим уравнением:

$$-div(\lambda \cdot grad\, T) + \sigma \frac{\partial T}{\partial t} = f$$

в цилиндрических координатах, заданным в некоторой области Ω с краевыми условиями второго и третьего рода

Коэффициент λ – теплопроводность материала

σ –Ср*р функция

Вид конечных элементов –треугольники.

Вид базисных функций – линейные.

Дискретизация по времени – по схеме Кранка-Николсона

2. Теоретическая часть

Теоретическая часть

2.1. Дискретизация по времени

Положим, что ось времени t разбита на так называемые временные слои со значениями t_j, $j = 1 \ldots J$, а значения искомой функции u и параметров λ, σ и f уравнения (1) будем обозначать соответственно через λ^j, σ^j и f^j, которые уже не зависят от времени t, но остаются функциями пространственных координат.

Схема Кранка-Николсона для уравнения (1), при условии, что λ и σ не зависят от t, а f - зависит, будет выглядеть следующим образом:

$$-div\left(\lambda grad\frac{u^i - u^{i-1}}{2}\right) + \sigma\frac{u^i - u^{i-1}}{\Delta t} = \frac{f^i + f^{i+1}}{2}, i = 1 \ldots n, \Delta t = t^i - t^{i-1}$$

2.2. Вариационная постановка

Пусть v – некоторая пробная функция из пространства $H_0{}^1$, тогда:

$$\int_\Omega \sigma\frac{u^j - u^{j-1}}{\Delta t} v d\Omega + \int_\Omega -div(\lambda grad\frac{u^j - u^{j-1}}{2}) v d\Omega = \int_\Omega \frac{f^j - f^{j-1}}{2} v d\Omega$$

Применим формулу Грина, преобразуем и перегруппируем:

$$\frac{1}{2}\int_\Omega \lambda^j grad(u^j) grad(v) d\Omega + \frac{1}{2}\int_\Omega \lambda^j grad(u^{j-1}) grad(v) d\Omega +$$
$$+\frac{1}{2}\int_{S_3} \beta^j u^j v dS_3 + \frac{1}{2}\int_{S_3} \beta^{j-1} u^{j-1} v dS_3 + \frac{1}{\Delta t}\int_\Omega \sigma^j u^j v d\Omega - \frac{1}{\Delta t}\int_\Omega \sigma^j u^{j-1} v d\Omega =$$
$$= \frac{1}{2}\int_\Omega f^j v d\Omega + \frac{1}{2}\int_\Omega f^{j-1} v d\Omega + \frac{1}{2}\int_{S_2} \theta^j v dS_2 + \frac{1}{2}\int_{S_2} \theta^{j-1} v dS_2 + \frac{1}{2}\int_{S_3} \beta^j u_\beta^j v dS_{3+}$$
$$+\frac{1}{2}\int_{S_3} \beta^{j-1} u_\beta^{j-1} v dS_3$$

2.3 Конечноэлементная дискретизация

Заменим пространство H_0^1 на конечномерное пространство V^h, которое определим как линейное пространство, натянутое на базисные функции ψ_i, $i = 1 \ldots n$. Заменим функцию u аппроксимирующей ее функцией u^h, а функцию v – функцией v^h.

Поскольку любая функция v^h может быть представлена в виде линейной комбинации $v^h = \sum_i q_i{}^v \psi_i$, получим:

$$\frac{1}{2}\int_\Omega \lambda^j grad\left(u^{h^i}\right) grad(\psi_i) d\Omega \quad + \\ + \frac{1}{2}\int_\Omega \lambda^j grad\left(u^{h^{i-1}}\right) grad(\psi_i) d\Omega \quad + \\ + + \frac{1}{2}\int_{S_3} \beta^j u^{h^i} \psi_i dS_{3+} \\ + \frac{1}{2}\int_{S_3} \beta^{j-1} u^{h^{i-1}} \psi_i dS_3 + \frac{1}{\Delta t}\int_\Omega \sigma^j u^{h^i} \psi_i d\Omega - \frac{1}{\Delta t}\int_\Omega \sigma^j u^{h^{i-1}} \psi_i d\Omega = \\ = \frac{1}{2}\int_\Omega f^j \psi_i d\Omega + \frac{1}{2}\int_\Omega f^{j-1} \psi_i d\Omega + \frac{1}{2}\int_{S_2} \theta^j \psi_i dS_2 + \frac{1}{2}\int_{S_2} \theta^{j-1} \psi_i dS_{2+} \\ + \frac{1}{2}\int_{S_3} \beta^j u_\beta^j \psi_i dS_{3+} + \frac{1}{2}\int_{S_3} \beta^{j-1} u_\beta^{j-1} \psi_i dS_3$$

Решение u^h может быть представлено в виде: $u^h = \sum_{k=1}^{n} q_k \psi_k$, причем $n - n_0$ компонент вектора весов q могут быть фиксированы и определены из условия $u^h|_{S1} = u_g$. Отсюда получим СЛАУ для вектора весов q:

$$\sum_{k=1}^{n}\left(\frac{1}{2}\int_\Omega \lambda^j grad(\psi_k) grad(\psi_i) d\Omega \quad + \frac{1}{\Delta t}\int_\Omega \sigma^j \psi_k \psi_i d\Omega + \frac{1}{2}\int_{S_3} \beta^j \psi_k \psi_i dS_3\right) \cdot q_k^j \\ = \frac{1}{2}\int_\Omega f^j \psi_i d\Omega + \frac{1}{2}\int_\Omega f^{j-1} \psi_i d\Omega + \frac{1}{2}\int_{S_2} \theta^j \psi_i dS_2 + \frac{1}{2}\int_{S_2} \theta^{j-1} \psi_i dS_{2+} \\ + \frac{1}{2}\int_{S_3} \beta^j u_\beta^j \psi_i dS_{3+} + \frac{1}{2}\int_{S_3} \beta^{j-1} u_\beta^{j-1} \psi_i dS_3 \\ + \sum_{k=1}^{n}\left(\frac{1}{\Delta t}\int_\Omega \sigma^j \psi_k \psi_i d\Omega - \frac{1}{2}\int_\Omega \lambda^j grad(\psi_k) grad(\psi_i) d\Omega \quad - \frac{1}{2}\int_{S_3} \beta^{j-1} \psi_k \psi_i dS_3\right) \\ \cdot q_k^{j-1}$$

2.4 Решение задачи на треугольной сетке с линейными базисными функциями

Так как для решения задачи используются линейные базисные функции, то на каждом конечном элементе Ω_k- треугольнике эти функции будут совпадать с функциями L_1(r, z), L_2(r, z), L_3(r, z), такими, что L_1(r, z) равна единице в вершине (r_1, z_1) и нулю во всех остальных вершинах, $L_2(x, y)$ равна единице в вершине (r_2, z_2) и нулю во всех остальных вершинах, L_3(r, z) равна единице в вершине (r_3, z_3) и нулю во всех остальных вершинах. Любая линейная на Ω_k

функция представима в виде линейной комбинации этих базисных линейных функций, коэффициентами будут значения функции в каждой из вершин треугольника Ω_k. Таким образом, на каждом конечном элементе нам понадобятся три узла – вершины треугольника.

Получаем:

$$\psi_1 = L_1(\mathrm{r}, \mathrm{z})$$
$$\psi_2 = L_2(\mathrm{r}, \mathrm{z})$$
$$\psi_3 = L_3(\mathrm{r}, \mathrm{z})$$

где $L_i = \alpha_0{}^i + \alpha_1{}^i \mathrm{r} + \alpha_2{}^i \mathrm{z}$, $i = \overline{1,3}$

из построения функций L_i получаем:

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

Обозначим $D = \begin{pmatrix} 1 & 1 & 1 \\ r_1 & r_2 & r_3 \\ z_1 & z_2 & z_3 \end{pmatrix}$

Будем использовать формулу:

$$\int_{\Omega_m} L_1^{v_1} L_2^{v_2} L_3^{v_3} dr dz = \frac{v_1! v_2! v_3!}{(v_1+v_2+v_3+2)!} |detD| \ (*)$$

$$detD = (r_2 - r_1)(z_3 - z_1) - (r_3 - r_1)(z_2 - z_1)$$

Получаем формулу:

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

J = r

Выразим как

$$\sum_{k=1}^{3} r_k L_k$$

Локальная матрица будет представлять собой сумму матриц жёсткости и массы и будет иметь размерность 3x3 (по числу узлов на конечном элементе).

Построение матрицы массы:

Пусть γ = некоторому усреднённому значению по всей области, тогда

$$M_{ij} = \frac{1}{\Delta t} \int_{\Omega_m} \sigma \Psi_i \Psi_j r d\Omega_m$$

$$= \frac{\sigma}{\Delta t} \int_{\Omega_m} \Psi_i \Psi_j r d\Omega_m = \frac{\sigma}{\Delta t} \int_{\Omega_m} L_i L_j \sum_{k=1}^{3} r_k L_k \, d\Omega_m = \frac{\sigma}{\Delta t} \sum_{k=1}^{3} r_k \int_{\Omega_m} L_i L_j L_k d\Omega_m =$$

Таким образом M = $\frac{\sigma}{\Delta t}\frac{|detD|}{24}(r1\begin{pmatrix}6&2&2\\2&2&1\\2&1&2\end{pmatrix}+r2\begin{pmatrix}2&2&1\\2&6&2\\1&2&2\end{pmatrix}+r3\begin{pmatrix}2&1&2\\1&2&1\\2&1&6\end{pmatrix})$

Построение матрицы жёсткости:

$$\psi_1 = L_1(r, z)$$
$$\psi_2 = L_2(r, z)$$
$$\psi_3 = L_3(r, z)$$

$$G_{ij} = \int \lambda\left(\frac{\partial\Psi_i}{\partial r}\frac{\partial\Psi_j}{\partial r}+\frac{\partial\Psi_i}{\partial z}\frac{\partial\Psi_j}{\partial z}\right)rd\Omega_m = \int \lambda\left(\frac{\partial\Psi_i}{\partial r}\frac{\partial\Psi_j}{\partial r}+\frac{\partial\Psi_i}{\partial z}\frac{\partial\Psi_j}{\partial z}\right)\sum_{k=1}^{3} r_k L_k\, d\Omega_m$$
$$= (\alpha_1^i\alpha_1^j+\alpha_2^i\alpha_2^j)\int_{\Omega_m} L_k * d\Omega_m = (r1+r2+r3)(\alpha_1^i\alpha_1^j+\alpha_2^i\alpha_2^j)\frac{|detD|}{6}$$

Построение вектора правой части:

$b_i = \int_{\Omega_m} f\Psi_i d\Omega_m = |f = \Psi_1 f_1 + \Psi_2 2 + \Psi_3 f_3| = \int_{\Omega_m}[\Psi_1 f_1 + \Psi_2 f_2 + \Psi_3 f_3]\Psi_i d\Omega_m$

Таким образом $b = \frac{M*\Delta t}{\sigma}\bar{f} = C * \bar{f}$

2.5 Построение Слау

Для формирования СЛАУ, необходимо просуммировать соответствующие матрицы, согласно схеме Кранка-Николсона:

$$\left(\tfrac{1}{\Delta t}\mathbf{M}+\tfrac{1}{2}\mathbf{G}+\tfrac{1}{2}\mathbf{M}^{S_3}\right)\mathbf{q}^j = \tfrac{1}{2}\left(\mathbf{b}^j+\mathbf{b}^{j-1}\right)+\left(\tfrac{1}{\Delta t}\mathbf{M}-\tfrac{1}{2}\mathbf{G}-\tfrac{1}{2}\mathbf{M}^{S_3}\right)\mathbf{q}^{j-1}.$$

Формирование глобальной матрицы из локальных:

учитываем соответствие локальной и глобальной нумераций каждого конечного элемента. Глобальная нумерация каждого конечного элемента однозначно определяет позиции вклада его локальной матрицы в глобальную. Поэтому, зная глобальные номера соответствующих узлов конечного элемента, определяем и то, какие элементы глобальной матрицы изменятся при учете текущего конечного элемента. Аналогичным образом определяется вклад локального вектора правой части в глобальный. При учете текущего локального вектора изменятся те элементы глобального вектора правой части, номера которых совпадают с глобальными номерами узлов, присутствующих в этом конечном элементе.

2.6 Учёт краевых условий Учет

первых краевых условий:

Для учета первых краевых условий, найдём максимальный по модулю элемент в матрице в глобальной матрице. Пусть $B = max * 10^{30}$. Поставим число B на главную диагональ, соответствующего узла в глобальной матрице, а для соответствующего элемента в векторе правой части запишем значение $U = U_g * B$, и наконец $qi = U_g$

Учет вторых и третьих краевых условий:

Рассмотрим краевые условия второго и третьего рода

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

$$-\lambda \frac{\partial u}{\partial n}\Big|_{S_3} = \beta(u|_{S_3} - u_\beta)$$

Отсюда получаем, что для учета краевых условий необходимо вычислить интегралы:

$$\int_{S_2} \theta \psi_i dS_2 \,, \int_{S_3} \beta u_\beta \, \psi_i dS_3, \int_{S_3} \beta \psi_i \psi_j dS_3$$

Краевые условия второго и третьего рода задаются на ребрах, т.е. определяются двумя узлами, лежащими на ребре.

где $\widetilde{\Psi_i}$ – локально занумерованные линейные базисные функции, которые имеют также свои глобальные номера во всей расчетной области, а $u_{\beta i}$ – значения функции u_β в узлах ребра.

Аналогично поступаем и при учете вторых краевых условий, раскладывая по базису ребра функцию $\theta = \theta_1 \widetilde{\Psi_1} + \theta_2 \widetilde{\Psi_2}$

Тогда приведенные выше интегралы примут вид:

$$I_1 = \int_{S_2} (\theta_1 \widetilde{\Psi_1} + \theta_2 \widetilde{\Psi_2}) \widetilde{\Psi_i} dS_2$$

$$I_2 = \beta \int_{S_3} (u_{\beta 1} \widetilde{\Psi_1} + u_{\beta 2} \widetilde{\Psi_2}) \widetilde{\Psi_i} \, dS_3$$

$$I_3 = \beta \int_{S_3} \widetilde{\Psi_i} \widetilde{\Psi_j} dS_3$$

Будем считать, что параметр β на S_3постоянен, тогда параметр u_β будем раскладывать по двум базисным функциям, определенным на этом ребре:

$u_\beta = u_{\beta 1} \widetilde{\Psi_1} + u_{\beta 2} \widetilde{\Psi_2}$.

Базисными функциями ребра являются две ненулевые на данном ребре базисные функции из $\psi_i, i = 1{,}3$ конечного элемента.

Для учета вклада вторых и третьих краевых условий рассчитываются 2 матрицы 2×2.

Интегралы I_1, I_2, I_3 будем вычислять по формуле $\int_{\Omega_m} L_j^{v_1} L_i^{v_2} dr dz = \frac{vi! v_j!}{(v_i + v_j + 1)!} mes\Gamma$, где $mes\Gamma$ длина ребра. При этом независимо от того, что на каждом из ребер присутствуют свои базисные функции, интегралы, посчитанные по приведенным выше формулам, будут равны.

$$I_1 = \begin{pmatrix} \int_{S2} \mathrm{L1L1r}\, \mathrm{drdz} & \int_{S2} \mathrm{L1L2r}\, \mathrm{drdz} \\ \int_{S2} \mathrm{L2L1r}\, \mathrm{drdz} & \int_{S2} \mathrm{L2L2r}\, \mathrm{drdz} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = \frac{mes\Gamma}{24} \begin{pmatrix} 6r1 + 2r2 & 2r1 + 2r2 \\ 2r1 + 2r2 & 2r1 + 6r2 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$$

Аналогично учитывается вектор поправок в правую часть

Поправка в левую часть:

$$I_3 = \int_{S_3} \beta \psi_i \psi_j dS_3 = \frac{\beta * mes\Gamma}{24} \begin{pmatrix} 6r1 + 2r2 & 2r1 + 2r2 \\ 2r1 + 2r2 & 2r1 + 6r2 \end{pmatrix}_3$$

2.7 Метод решения СЛАУ

Слау решается методом сопряженных градиентов с LL^T факторизацией

3. Описание разработанной программы

3.1. Структуры данных, используемые для задания расчетной области и конечноэлементной сетки

Для задания расчетной области и конечноэлементной сетки в программе используются следующие структуры (упрощенно): Для реализации конечноэлементной сетки был создан класс Net, содержащий поля:

```
vector<vector<double>> Node; //Массив узлов, номер в массиве соответствует номеру
узла, элементы массива: координаты
vector<vector<int>> Elements; //Номера узлов входящих в конечный элемент vector<int>
fields; //массив областей для элементов vector<double> t; //Массив узлов, для разбиения по
времени vector<vector<int>> firstCondi; //первые краевые vector<vector<int>>
SecondCondi; //вторые vector<vector<int>> ThirdCondi; //третьи
```

class Net имеет метод для генерации областей: BuildNet

Для внешнего хранения или ввода расчетной области и конечноэлементной сетки используются файлы следующего назначения:

Имя файла	*Содержание*	*Представление*
nodes.txt	Узлы сетки	Координаты узлов.
elements.txt	Конечные элементы	Номера узлов
Fields.txt	Подобласти	Номер подобласти элементов подряд
Condi1.txt	Первые краевые	Номер узла, на котором задано условие и номер условия (например для разных границ
Condi2.txt	Вторые краевые	Два номера узла, задающих ребро и номер условия
Condi3.txt	Третьи краевые	Два номера узла, номер условия и подобласть

3.2. Структура основных модулей программы

Основные модули программы: Для хранения элементов глобальной матрицы была использована структура:

```
struct MatrixProf
{
int size; vector<double> DI;
vector<double> AL;
vector<double> AU; vector<int>
IA; vector<int> JA;
};
```

Для задания уравнения, был создан класс Eq, который включает в себя функции:

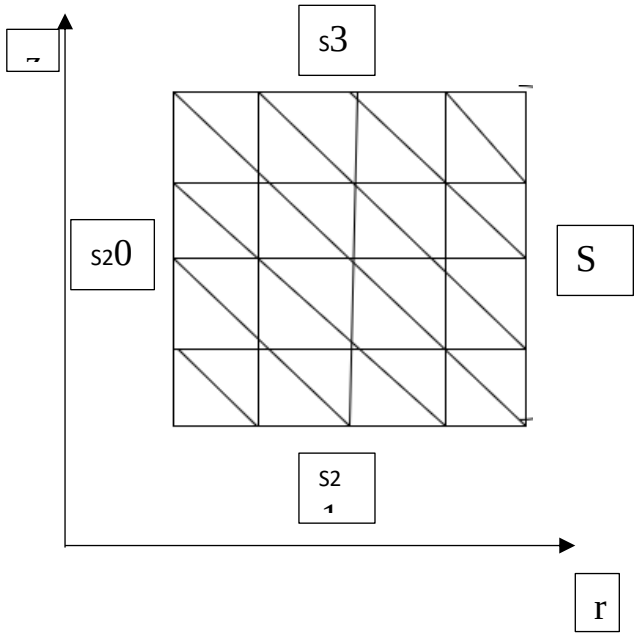
- Построение профиля матрицы
- Построение локальных матриц и векторов правой части
- Перенос локальных матриц и векторов в глобальные
- Итеративный процесс по времени

Так же содержит параметры уравнения в виде функций: Lambda, Ug, Betta, Sigma, F, UB A так же вектор решения q.

Решатель системы описан в файле Solver.h.

4. Тесты

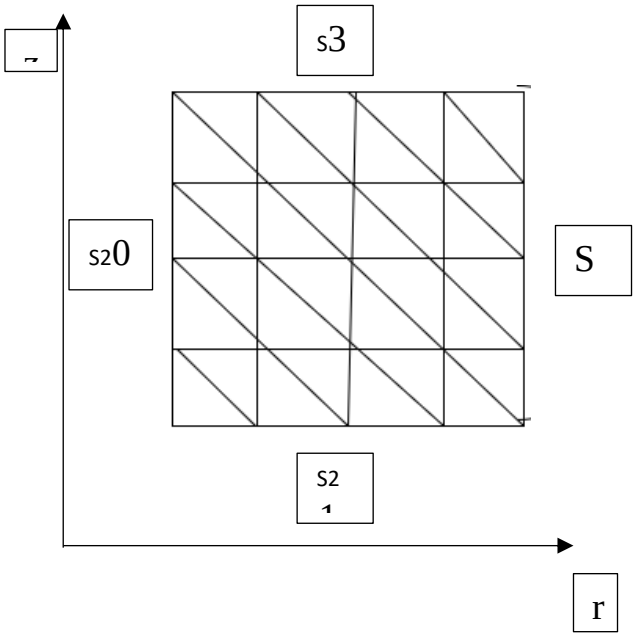
1) Линейная функция



U=z
S20 : tetta = 0
Lambda = 100
S21=-lambda
S3 : B = 1, Ub=100+z
F=0
R=0.1, 1
Z=0.1, 1

$\|u* - u_{ht}\|/\|u*\|$ = 8.161442838711795e-15

2) Квадратичная функция



U=z^2
S20 : tetta = 0
L = 100
S21=-2*z*L
S3 : B = 1,
Ub=2*z*L+z*z
F=0
R=0.1, 1
Z=0.1, 1

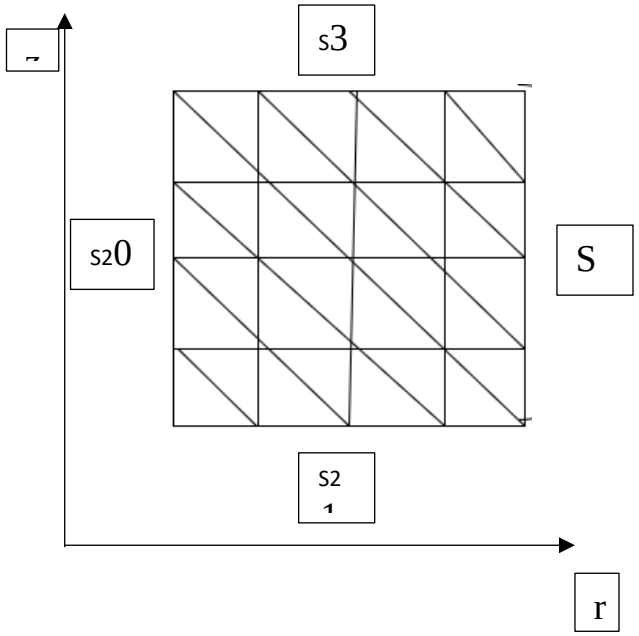
Погрешность = 4.218977585509547e-02
При дроблении:

$\|u_* - u_{ht}\|/\|u_*\|$	$\|u_* - u_{ht/2}\|/\|u_*\|$	$\|u_* - u_{ht/4}\|/\|u_*\|$
4.218977585509547e-02	1.211290755433292e-02	2.942848885954491e-03

Порядок сходимостри: 4

3) Порядок аппроксимации по времени

3.1) Вторая степень



$U = z * t\text{^}2$

F =2*sigma*z*t

Sigma = 2

L= 100

Краевые условия:

S20 : tetta=0

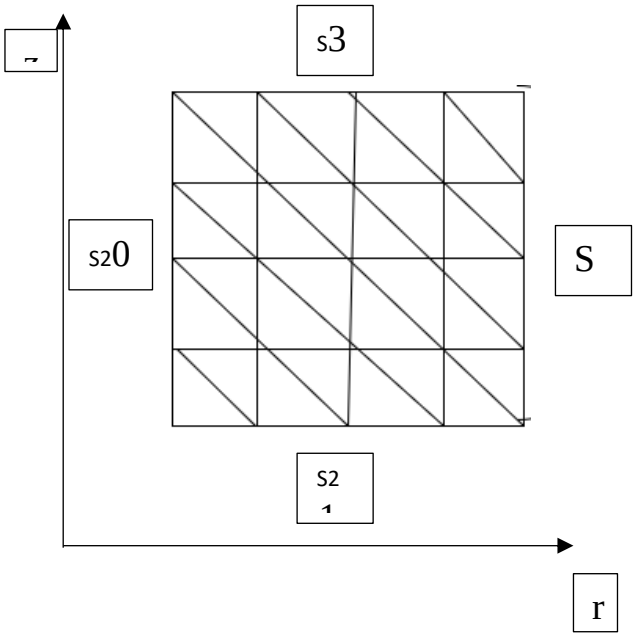
S21: tetta = -L*t

S3:L*t^2+z*t^2

t = [0,1], h = 0.1

t	$\|u * - u_{ht}\|/\|u *\|$
0.10	0.000000000000000e+00
0.20	2.240427111781592e-11
0.30	1.690540786935955e-11
0.40	7.788866798664573e-12
0.50	4.168588005795039e-12
0.60	2.824450624465520e-12
0.70	1.847619298846340e-12
0.80	1.486564821059529e-12
0.90	1.062039612050151e-12
1.00	9.681358967918325e-13

3.2) третья степень



$U = z * t\text{^}3$

F =3*z*t^2*sigma

Sigma = 2

L= 100

Краевые условия:

S20 : tetta=0

S21: tetta = -L*t^3

S3:L*t^3+z*t^3

5. Эксперимент

Пусть имеется металлический цилиндр

$$R = 1\text{м}$$

$$h = 0.5\text{м}$$

$$C\rho = 450\ \frac{\text{Дж}}{\text{Кг} * K}$$

$$\rho = 7247$$

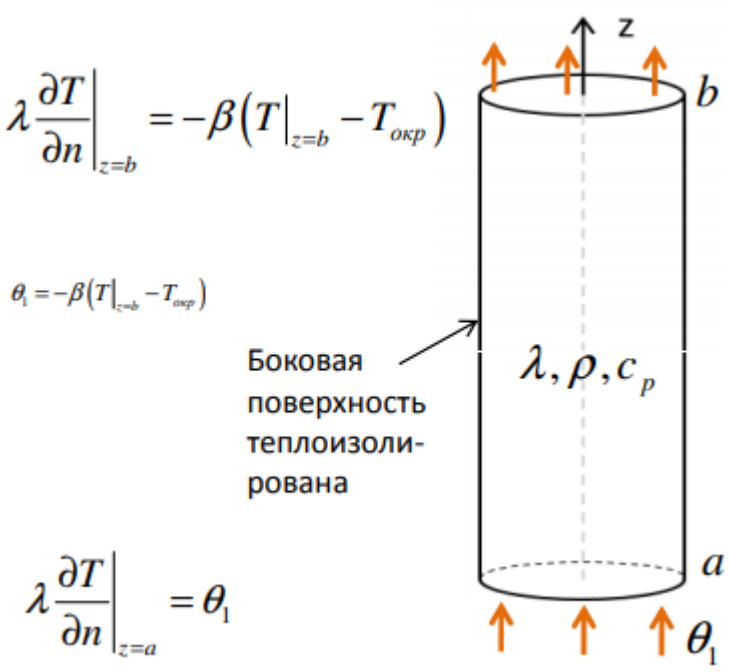
$$\lambda = 92\ \text{Вт}/(\text{м} \bullet \text{К})$$

Теплообмен с окружающей средой примем

$$\beta = 10\ \text{Вт}/(\text{м}^2\text{К})$$

Боковая поверхность теплоизолирована, на нижнюю грань цилиндра поступает постоянный тепловой поток, сверху третьи краевые условия.

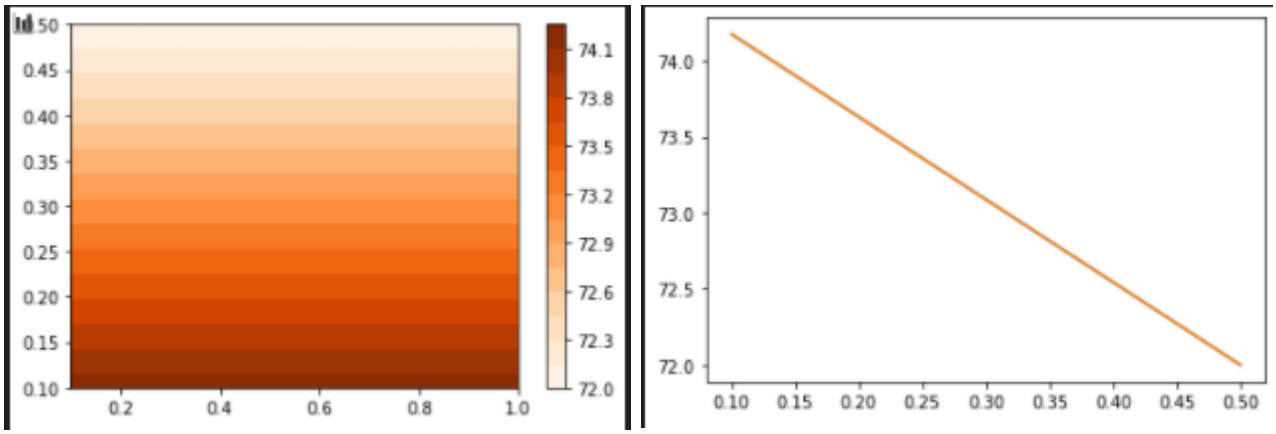
1) Стационарный процесс



θ = 500 Вт

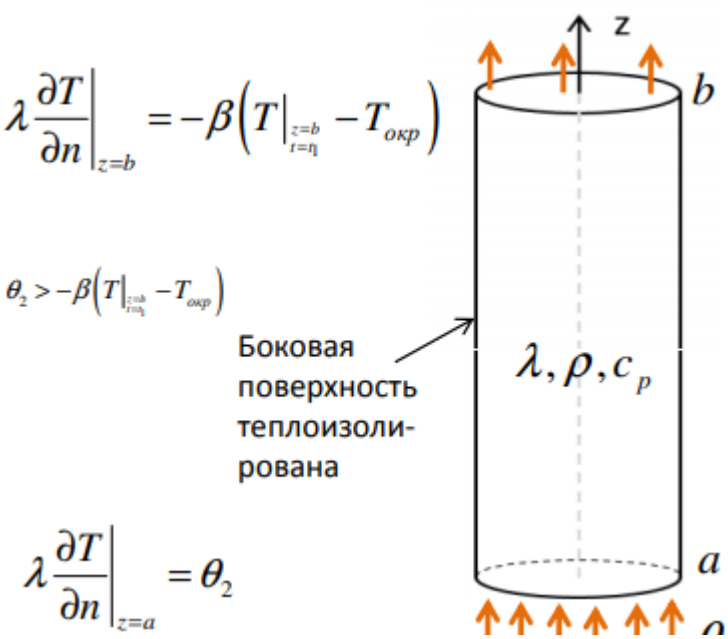
При таких значениях, температура наверху цилиндра должна быть равна ~72 С

Распределение температуры в цилиндре и график температуры



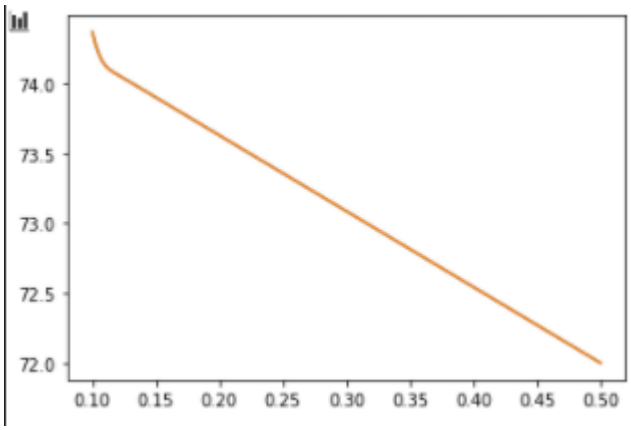
2) увеличим поток на нижней границе до

θ = 3500

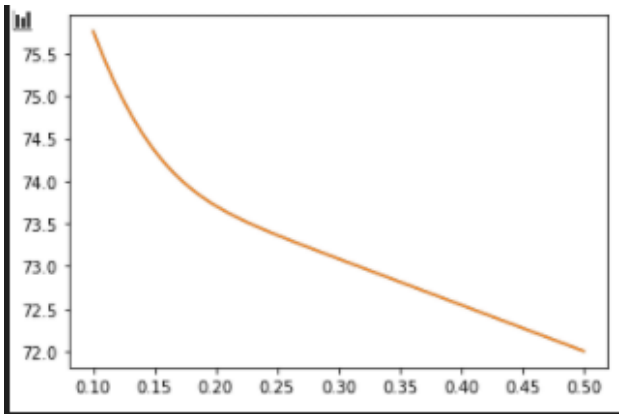


⇨ Возникает нестационарный процесс: нагрев цилиндра

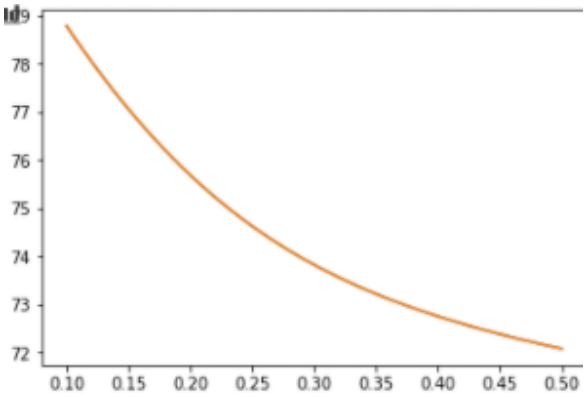
t=1c



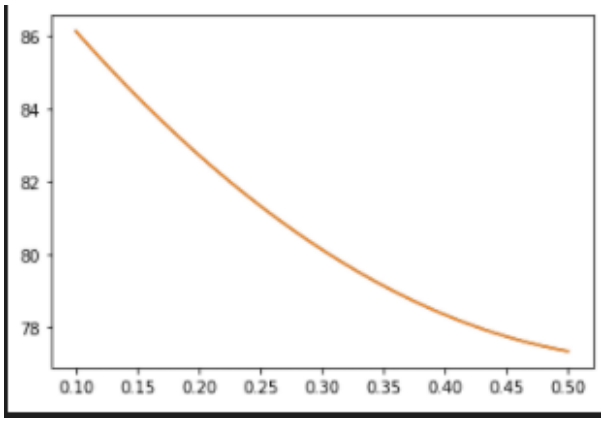
t=70c



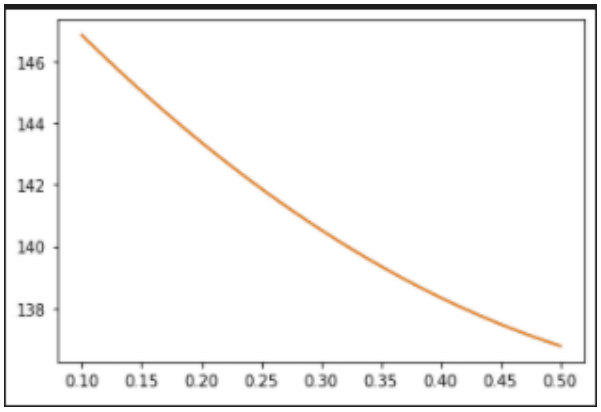
t=600c



t=3600c



t=36000



z	T	z	T
t0		t_last	
0.1	74.17391304	0.1	146.92
0.1222222222	74.0531401	0.1222222222	146.08
0.1444444444	73.93236715	0.1444444444	145.28
0.1666666667	73.8115942	0.1666666667	144.50
0.1888888889	73.69082126	0.1888888889	143.76
0.2111111111	73.57004831	0.2111111111	143.05
0.2333333333	73.44927536	0.2333333333	142.38
0.2555555556	73.32850242	0.2555555556	141.73
0.2777777778	73.20772947	0.2777777778	141.12
0.3	73.08695652	0.3	140.54
0.3222222222	72.96618357	0.3222222222	139.99
0.3444444444	72.84541063	0.3444444444	139.47
0.3666666667	72.72463768	0.3666666667	138.98
0.3888888889	72.60386473	0.3888888889	138.52
0.4111111111	72.48309179	0.4111111111	138.10
0.4333333333	72.36231884	0.4333333333	137.71
0.4555555556	72.24154589	0.4555555556	137.34
0.4777777778	72.12077295	0.4777777778	137.01
0.5	72	0.5	136.71

5. Выводы

Судя по тестам, поведение программного модуля совпадает с теоретическим.

При проведении эксперимента наблюдается чёткая картина изменения температуры вдоль цилиндра:

- При равенстве потока наблюдается линейное распределение температуры
- После увеличения потока, цилиндр нагревается
- После того как тело достаточно нагреется, из-за разности температур на верхней границе, температура перестаёт расти, а распределение поля температуры вдоль цилиндра снова стремится к линейному

6. Текст программы

Source.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <functional>
#include <iomanip>
#include <fstream>
#include "Solver.h"
// M = l * C
using namespace std;
typedef vector<vector<double>> Matrix;

struct AuxVectors
{
	vector<double> Ax;
	vector<double> r;
	vector<double> z;
	vector<double> p;
	vector<double> LU;
	vector<double> temp;
};
class Net
{
public:
	Net()
	{
	}
	Net(fstream& nodes, fstream& elements, fstream& fields, fstream& condi1, fstream&
condi2, fstream& condi3)
	{
		double x, y;
		int a, b, c, field, type;
		while (nodes >> x >> y)
		{
			Node.push_back({ x,y });
		}
		while (elements >> a >> b >> c)
		{
			Elements.push_back({ a,b,c });
		}
		while (fields >> field)
		{
			this->fields.push_back(field);
		}
		while (condi1 >> a >> type)
		{
			this->firstCondi.push_back({ a,type });
		}
		while (condi2 >> a >> b >> type)
		{
			this->SecondCondi.push_back({ a, b,type });
		}
		while (condi3 >> a >> b >> type >> field)
		{
			this->ThirdCondi.push_back({ a,b,type, field });
		}
	}
	void BuildTnet(double tmin, double tmax, int n)
	{
		double tc;
		double h = (tmax - tmin) / n;
```

```
		for (size_t i = 0; i < n + 1; i++)
		{
			t.push_back(tmin + i * h);
		}
	}
	void SaveNet(fstream & nodes, fstream & elements, fstream & fields)
	{
		int length = Node.size();
		for (size_t i = 0; i < length; i++)
		{
			nodes << Node[i][0] << " " << Node[i][1] << "\n";
		}
		length = this->Elements.size();
		for (size_t i = 0; i < length; i++)
		{
			elements << this->Elements[i][0] << " " << this->Elements[i][1] << " "
<< this->Elements[i][2] << "\n";
			fields << this->fields[i] << "\n";
		}
	}
	vector<vector<double>> Node;
	vector<vector<int>> Elements;
	vector<int> fields;
	vector<double> t;
	vector<vector<int>> firstCondi;
	vector<vector<int>> SecondCondi;
	vector<vector<int>> ThirdCondi;
	//добавление информации о подобластях
	void DevideBy2Fields()
	{
		fields = vector<int>(Elements.size());
		int middle = fields.size() / 2;
		for (int i = middle; i < fields.size(); i++)
		{
			fields[i] = 1;
		}
	}
	//Генерация первых краевых условий на всей области
	void AddCondi(int nx, int ny)
	{
		nx++;
		ny++;
		for (int j = 0; j < ny; j++)
		{
			for (int i = 0; i < nx; i++)
			{
				/*if (i == 0 || i==nx-1  )
				{
					int k = nx * j + i;
					firstCondi.push_back({ k,0 });
				}*/
				if (j == ny-1 && i!=nx-1)
				{
					int k1 = nx * j + i;
					int k2 = nx * j + i+1;
					ThirdCondi.push_back({ k1,k2,0 });
				}

				if (j == 0 && i != nx - 1)
				{
					int k1 = i;
					int k2 = i + 1;
					SecondCondi.push_back({ k1,k2,0 });
				}
```

```
                }
            }

        }
        //построение сетки на треугольниках
        void BuildNet(double xmin, double xmax, double ymin, double ymax, int nx, int ny)
        {
            double hx = (xmax - xmin) / nx;
            double hy = (ymax - ymin) / ny;
            Node = vector<vector<double>>((nx + 1) * (ny + 1));
            Node[0] = vector<double>{ xmin, ymin };
            for (int i = 0; i < ny; i++)
            {
                double y = ymin + i * hy;
                for (int j = 0; j < nx; j++)
                {
                    double x = xmin + j * hx;
                    Node[i * (nx + 1) + j + 1] = { x + hx, y };
                    Node[(i + 1) * (nx + 1) + j] = { x,y + hy };
                    Elements.push_back({ j + i * (nx + 1),j + 1 + i * (nx + 1), j +
(nx + 1) * (i + 1) });
                }
            }
            Node[Node.size() - 1] = { xmax,ymax };
            for (int i = ny; i > 0; i--)
            {
                for (int j = nx; j > 0; j--)
                {
                    Elements.push_back({ j + i * (nx + 1) - nx - 1,j - 1 + i * (nx +
1), j + i * (nx + 1) });
                }
            }
            int length = Elements.size();
            vector<vector<int>> Elementstmp(length);
            for (int j = 0, i = 0; i < length; j++, i += 2)
            {
                Elementstmp[i] = Elements[j];
            }
            for (int i = 1, j = length - 1; i < length; i += 2, j--)
            {
                Elementstmp[i] = Elements[j];
            }
            Elements = Elementstmp;
            fields.resize(Elements.size());
            //разбиение на подобласти
            //DevideBy2Fields();
        }
private:
};
class Eq
{
public:
        Net TheNet;
        vector<double> b;
        MatrixProf AProf;
        MatrixProf LU;
        Matrix A;
        vector<double>q_st;
        vector<vector<double>> q;
        double BigEl;
        void PrintPlot(Matrix& A)
        {
            int length = A.size();
            for (int i = 0; i < length; i++)
```

```
		{
			for (int j = 0; j < length; j++)
			{
				cout << A[i][j] << " \t";
			}
			cout << "\n";
		}
	}
	//параметры
	double U(double r, double z, double t, int field)
	{
		return z*t*t;
	}
	Eq()
	{
		TheNet.BuildNet(0, 2, 0, 2, 2, 2);
		A = Matrix(TheNet.Node.size());
		b = vector<double>(TheNet.Node.size());
	}
	Eq(Net net)
	{
		TheNet = net;
		A = Matrix(TheNet.Node.size());
		for (int i = 0; i < A.size(); i++)
		{
			A[i] = vector<double>(A.size());
		}
		b = vector<double>(TheNet.Node.size());
		q = vector<vector<double>>(TheNet.t.size());
		for (size_t i = 0; i < q.size(); i++)
		{
			q[i] = vector<double>(TheNet.Node.size());
		}
		q_st = vector<double>(TheNet.Node.size());
	}
	//разложение коэф дифузии по линейным базисным функциям

	vector<vector<double>> BuildG(vector<vector<double>>& D_1, double DetD, vector<int>&
el, int field)
	{
		vector<vector<double>> G(3);
		double r1 = TheNet.Node[el[0]][0];
		double r2 = TheNet.Node[el[1]][0];
		double r3 = TheNet.Node[el[2]][0];

		double multix = abs(DetD)*(r1+r2+r3) / 6.;
		for (int i = 0; i < 3; i++)
		{
			for (int j = 0; j < 3; j++)
			{

				double L = Lambda(field);
				G[i].push_back(L * multix * (D_1[i][1] * D_1[j][1] + D_1[i][2] *
D_1[j][2])); // Lambda =const;
			}
		}
		return G;
	}
	double findMax(double x1, double x2)
	{
		if (x1 > x2)
			return x1;
		else
			return x2;
```

```
}
//потроение матрицы C, M = Sigma /dt * C
Matrix BuildC(double DetD)
{
	Matrix M = Matrix{ {2,1,1 }, { 1,2,1 }, { 1,1,2 } };
	double mult = abs(DetD) / 24;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			M[i][j] *= mult;
		}
	}
	return M;
}

Matrix BuildC_cylindrical(double DetD, vector<int>& el)
{
	double r1 = TheNet.Node[el[0]][0];
	double r2 = TheNet.Node[el[1]][0];
	double r3 = TheNet.Node[el[2]][0];

	Matrix M = Matrix{
		{ 6*r1+2*r2+2*r3, 2*r1 + 2*r2 + r3, 2 * r1 + r2 + 2*r3 },
		{ 2 * r1 + 2 * r2 + r3, 2*r1+6*r2+2*r3, r1 + 2*r2 + 2*r3 },
		{ 2 * r1 + r2 + 2 * r3, r1 + 2 * r2 + 2 * r3,2 * r1 + 2 * r2 + 6 * r3}
	};
	double mult = abs(DetD) / 120;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			M[i][j] *= mult;
		}
	}
	return M;
}
//умножение матрицы на вектор
vector<double> MVecMult(Matrix& A, vector<double>& b)
{
	vector<double> result(A.size());
	int length = A.size();
	for (int i = 0; i < length; i++)
	{
		for (int j = 0; j < length; j++)
		{
			result[i] += A[i][j] * b[j];
		}
	}
	return result;
}
//Сложение векторов
vector<double> VecSum(vector<double>& a, vector<double>& b)
{
	int length = a.size();
	vector<double> rez;
	for (size_t i = 0; i < length; i++)
	{
		rez.push_back(a[i] + b[i]);
	}
	return rez;
}
//Построение локальных матриц и вектора правой части, согласно схеме Кранка-Никлсона
Matrix BuildLocalKN(vector<int>& el, int field, double t, double tpr, int tn)
```

```
	{
		double dt = t - tpr;
		double r1 = TheNet.Node[el[0]][0];
		double r2 = TheNet.Node[el[1]][0];
		double r3 = TheNet.Node[el[2]][0];
		double z1 = TheNet.Node[el[0]][1];
		double z2 = TheNet.Node[el[1]][1];
		double z3 = TheNet.Node[el[2]][1];
		vector<vector<double>> D{
		vector<double>{1,1,1},
		vector<double> {r1,r2,r3},
		vector<double> {z1,z2,z3}
		};
		double DetD = (r2 - r1) * (z3 - z1) - (r3 - r1) * (z2 - z1);
		vector<vector<double>> D_1{
		vector<double> {r2* z3 - r3 * z2, z2 - z3, r3 - r2},
		vector<double> {r3* z1 - r1 * z3, z3 - z1, r1 - r3},
		vector<double> {r1* z2 - r2 * z1, z1 - z2, r2 - r1}
		};
		for (int i = 0; i < 3; i++)
		{
			for (int j = 0; j < 3; j++)
			{
				D_1[i][j] /= DetD;
			}
		}
		Matrix G = BuildG(D_1, DetD, el, field);
		Matrix M = BuildC_cylindrical(DetD,el);
		//строим b
		vector<double> f = { F(r1,z1,t,field),F(r2,z2,t,field),F(r3,z3,t,field) };
		vector<double> fpr = {
F(r1,z1,tpr,field),F(r2,z2,tpr,field),F(r3,z3,tpr,field) };
		f = VecSum(f, fpr);
		vector<double> b = MVecMult(M, f);
		int length = b.size();
		vector<double> tmp;
		vector<double> ql = { q[tn - 1][el[0]], q[tn - 1][el[1]],q[tn - 1][el[2]] };
		tmp = MVecMult(G, ql);
		for (size_t i = 0; i < length; i++)
		{
			b[i] /= 2.;
			tmp[i] = -tmp[i] / 2.;
		}
b = VecSum(b, tmp);
		length = G.size();
		for (int i = 0; i < length; i++)
		{
			for (int j = 0; j < length; j++)
			{
				M[i][j] = M[i][j] * Sigma(field) / dt;
				G[i][j] = G[i][j] / 2. + M[i][j];
			}
		}
		tmp = MVecMult(M, ql);
		b = VecSum(b, tmp);
		ToGLobalProf(G, b, el);
		//ToGlobalPlot (G, b, el);
		return G;
	}
	Matrix BuildLocalStatic(vector<int>& el, int field)
	{
		double r1 = TheNet.Node[el[0]][0];
		double r2 = TheNet.Node[el[1]][0];
		double r3 = TheNet.Node[el[2]][0];
```

```
	double z1 = TheNet.Node[el[0]][1];
	double z2 = TheNet.Node[el[1]][1];
	double z3 = TheNet.Node[el[2]][1];
	vector<vector<double>> D{
	vector<double>{1,1,1},
	vector<double> {r1,r2,r3},
	vector<double> {z1,z2,z3}
	};
	double DetD = (r2 - r1) * (z3 - z1) - (r3 - r1) * (z2 - z1);
	vector<vector<double>> D_1{
	vector<double> {r2* z3 - r3 * z2, z2 - z3, r3 - r2},
	vector<double> {r3* z1 - r1 * z3, z3 - z1, r1 - r3},
	vector<double> {r1* z2 - r2 * z1, z1 - z2, r2 - r1}
	};
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			D_1[i][j] /= DetD;
		}
	}
	Matrix G = BuildG(D_1, DetD, el, field);
	Matrix M = BuildC_cylindrical(DetD, el);
	//строим b
	vector<double> f = { F(r1,z1,0,field),F(r2,z2,0,field),F(r3,z3,0,field) };
	vector<double> b = MVecMult(M, f);
	int length = b.size();
	for (int i = 0; i < length; i++)
	{
		for (int j = 0; j < length; j++)
		{
			M[i][j] = M[i][j] * Sigma(field);
			G[i][j] = G[i][j] /*+ M[i][j]*/;
		}
	}
	ToGLobalProf(G, b, el);
	//ToGlobalPlot (G, b, el);
	return G;
}
	//Обнуление элементов (для следующей итерации)
void RefreshMatrixProf()
{
	AProf.AL = vector<double>(AProf.IA[AProf.IA.size() - 1] - 1);
	AProf.AU = vector<double>(AProf.IA[AProf.IA.size() - 1] - 1);
	AProf.DI = vector<double>(TheNet.Node.size());
	AProf.size = AProf.DI.size();
}
void BuildGlobalStatic(int tn)
{
	RefreshMatrixProf();
	b = vector<double>(b.size());
	for (int i = 0; i < TheNet.Elements.size(); i++)
	{
		vector<int> element = TheNet.Elements[i];
		int field = TheNet.fields[i];
		BuildLocalStatic(element, field);
		//PrintPlot(A);
	}
}
//Построение глобальной матрицы в профильном формате
void BuildGlobalKN(int tn)
{
	RefreshMatrixProf();
	b = vector<double>(b.size());
```

```
		for (int i = 0; i < TheNet.Elements.size(); i++)
		{
			vector<int> element = TheNet.Elements[i];
			int field = TheNet.fields[i];
			BuildLocalKN(element, field, TheNet.t[tn], TheNet.t[tn - 1], tn);
			//PrintPlot(A);
		}
	}
	//Запуск поиска решений
	void FindSolution()
	{
		int length = TheNet.t.size();
		FindSolution_static();
		for (size_t i = 1; i < length; i++)
		{
			BuildGlobalKN(i);
			AddThirdCondiKN(i);
			AddSecondCondiKN(i);
			AddFirstKN(i);
			CalculateMSG(q[i]);
		}
	}
	void FindSolution_static()
	{
		int length = TheNet.t.size();
		BuildProfile();
		BuildGlobalStatic(0);
		AddThirdCondi_static(0);
		AddSecondCondi_static(0);
		AddFirst();
		CalculateMSG(q[0]);
	}
	//построение профиля матрицы
	void BuildProfile()
	{
		vector<vector<int>> profile(TheNet.Node.size());
		for (int i = 0; i < TheNet.Elements.size(); i++)
		{
			for (int j = 1; j < 3; j++)
			{
				for (int k = 0; k < j; k++)
				{
					int current = TheNet.Elements[i][j];
					int node = TheNet.Elements[i][k];
					if (!count(profile[current].begin(),
profile[current].end(), node))
					{
						if (profile[current].size() != 0 &&
profile[current][profile[current].size() - 1] > node)
						{
							for (int l = 0; l < profile[current].size();
l++)
							{
								if (node < profile[current][l])
								{

	profile[current].insert(profile[current].begin() + l, node);
									break;
								}
							}
						}
						else
						{
							profile[current].push_back(node);
```

```
                                        }
                                }
                        }
                }
        }
        AProf.IA.push_back(1);
        int count = 0;
        for (int i = 1; i < TheNet.Node.size(); i++)
        {
                AProf.IA.push_back(AProf.IA[i - 1] + count);
                count = 0;
                for (int j = 0; j < profile[i].size(); j++)
                {
                        AProf.JA.push_back(profile[i][j]);
                        count++;
                }
        }
        AProf.IA.push_back(AProf.IA[AProf.IA.size() - 1] + count);
        AProf.AL = vector<double>(AProf.IA[AProf.IA.size() - 1] - 1);
        AProf.AU = vector<double>(AProf.IA[AProf.IA.size() - 1] - 1);
        AProf.DI = vector<double>(TheNet.Node.size());
        AProf.size = AProf.DI.size();
}
//добавление третьих краевых
void AddThirdCondiKN(int tn)
{
        int length = TheNet.ThirdCondi.size();
        for (int i = 0; i < length; i++)
        {
                vector<int> Edge = TheNet.ThirdCondi[i];
                double r1 = TheNet.Node[Edge[0]][0];
                double z1 = TheNet.Node[Edge[0]][1];
                double r2 = TheNet.Node[Edge[1]][0];
                double z2 = TheNet.Node[Edge[1]][1];
                double hm = sqrt((r2 - r1) * (r2 - r1) + (z2 - z1) * (z2 - z1));
                Matrix MS3 = GetMS(r1,r2);
                double mult = Betta((int)Edge[2]) * hm / 24.;
                for (int i = 0; i < 2; i++)
                {
                        for (int j = 0; j < 2; j++)
                        {
                                MS3[i][j] = MS3[i][j] * mult / 2;
                        }
                }
                vector<double> Ub = { UB(TheNet.Node[Edge[0]], Edge[2],
tn),UB(TheNet.Node[Edge[1]], Edge[2], tn) };
                vector<double> UbPrev = { UB(TheNet.Node[Edge[0]], Edge[2], tn-
1),UB(TheNet.Node[Edge[1]], Edge[2], tn-1) };
                vector<double> b = MVecMult(MS3,Ub);
                vector<double> tmp = MVecMult(MS3,UbPrev);
                b = VecSum(b, tmp);
                vector<double> ql = { q[tn - 1][Edge[0]], q[tn - 1][Edge[1]] };
                tmp = MVecMult(MS3, ql);
                for (size_t i = 0; i < tmp.size(); i++)
                {
                        tmp[i] = -tmp[i];
                }
                b = VecSum(b, tmp);
                ToGLobalProf(MS3, b, Edge);
                ToGlobalPlot(MS3, b, Edge);
        }
}
//добавление первых краевых
void AddFirst()
```

```
{
	double max = 0;
	int length = AProf.AL.size();
	for (int i = 0; i < length; i++)
	{
		if (max < abs(AProf.AL[i]))
		{
			max = abs(AProf.AL[i]);
		}
	}
	max *= 1e+30;
	length = TheNet.firstCondi.size();
	for (int i = 0; i < length; i++)
	{
		int n = TheNet.firstCondi[i][0];
		AProf.DI[n] = max;
		double r = TheNet.Node[n][0];
		double z = TheNet.Node[n][1];
		b[n] = max * Ug(TheNet.Node[n], TheNet.firstCondi[i][1],0);
		q[0][n] = Ug(TheNet.Node[n], TheNet.firstCondi[i][1],0);
	}
}
void AddFirstKN(int tn)
{
	double max = 0;
	int length = AProf.AL.size();
	for (int i = 0; i < length; i++)
	{
		if (max < abs(AProf.AL[i]))
		{
			max = abs(AProf.AL[i]);
		}
	}
	max *= 1e+30;
	length = TheNet.firstCondi.size();
	for (int i = 0; i < length; i++)
	{
		int n = TheNet.firstCondi[i][0];
		AProf.DI[n] = max;
		b[n] = max * Ug(TheNet.Node[n], TheNet.firstCondi[i][1], tn);
		q[tn][n] = Ug(TheNet.Node[n], TheNet.firstCondi[i][1], tn);
	}
}
//добавление вторых краевых
void AddSecondCondiKN(int tn)
{
	double t = TheNet.t[tn];
	int length = TheNet.SecondCondi.size();
	for (int i = 0; i < length; i++)
	{
		vector<int> Edge = TheNet.SecondCondi[i];
		double r1 = TheNet.Node[Edge[0]][0];
		double z1 = TheNet.Node[Edge[0]][1];
		double r2 = TheNet.Node[Edge[1]][0];
		double z2 = TheNet.Node[Edge[1]][1];
		double hm = sqrt((r2 - r1) * (r2 - r1) + (z2 - z1) * (z2 - z1));
		double mult = hm / 24;
		Matrix M2 = GetMS(r1, r2);
		vector<double> Tet = {
			1./2.*Tetta(TheNet.Node[Edge[0]],Edge[2],tn),
			1. / 2.*Tetta(TheNet.Node[Edge[1]],Edge[2],tn) };
		vector<double> TetPrev = {
			1. / 2.*Tetta(TheNet.Node[Edge[0]],Edge[2],tn-1),
			1. / 2.*Tetta(TheNet.Node[Edge[1]],Edge[2],tn-1) };
```

```
                vector<double> b = MVecMult(M2,Tet);
                vector<double> prev = MVecMult(M2, TetPrev);
                b = VecSum(b, prev);
                this->b[Edge[0]] += b[0]*mult;
                this->b[Edge[1]] += b[1]*mult;
        }
}

Matrix GetMS(double r1,double r2)
{
        Matrix M2;
        M2.push_back({ 6 * r1 + 2 * r2,2 * (r1 + r2) });
        M2.push_back({ 2 * (r1 + r2),2 * r1 + 6 * r2 });
        return M2;
}
void AddSecondCondi_static(int tn)
{
        double t = TheNet.t[tn];
        int length = TheNet.SecondCondi.size();
        for (int i = 0; i < length; i++)
        {
                vector<int> Edge = TheNet.SecondCondi[i];
                double r1 = TheNet.Node[Edge[0]][0];
                double z1 = TheNet.Node[Edge[0]][1];
                double r2 = TheNet.Node[Edge[1]][0];
                double z2 = TheNet.Node[Edge[1]][1];
                double hm = sqrt((r2 - r1) * (r2 - r1) + (z2 - z1) * (z2 - z1));
                double mult = hm / 24;
                Matrix M2 = GetMS(r1, r2);
                vector<double> Tet;
                Tet.push_back(Tetta(TheNet.Node[Edge[0]], Edge[2], tn));
                Tet.push_back(Tetta(TheNet.Node[Edge[1]], Edge[2], tn));
                vector<double> b = MVecMult(M2,Tet);
                this->b[Edge[0]] += b[0]*mult;
                this->b[Edge[1]] += b[1]*mult;
        }
}

void AddThirdCondi_static(int tn)
{
        int length = TheNet.ThirdCondi.size();
        for (int i = 0; i < length; i++)
        {
                vector<int> Edge = TheNet.ThirdCondi[i];
                double r1 = TheNet.Node[Edge[0]][0];
                double z1 = TheNet.Node[Edge[0]][1];
                double r2 = TheNet.Node[Edge[1]][0];
                double z2 = TheNet.Node[Edge[1]][1];
                double hm = sqrt((r2 - r1) * (r2 - r1) + (z2 - z1) * (z2 - z1));
                Matrix MS3 = GetMS(r1, r2);
                double B = Betta((int)Edge[2]);
                double mult =  B*hm / 24.;
                for (int i = 0; i < 2; i++)
                {
                        for (int j = 0; j < 2; j++)
                        {
                                MS3[i][j] = MS3[i][j] * mult;
                        }
                }
                double UB1 = UB(TheNet.Node[Edge[0]], Edge[2], 0);
                double UB2 = UB(TheNet.Node[Edge[1]], Edge[2], 0);
                vector<double> Ub = { UB1,UB2 };
                vector<double> b = MVecMult(MS3, Ub);
```

```
			ToGLobalProf(MS3, b, Edge);
			ToGlobalPlot(MS3, b, Edge);
		}
	}
	void LUFactorization(MatrixProf& A, MatrixProf& LU)
	{
		int length = A.IA.size();
		for (int i = 0; i < length; i++)
		{
			A.IA[i]--;
		}
		LU.size = A.size;
		LU.IA.resize(LU.size + 1);
		for (int i = 0; i < A.size + 1; i++)
			LU.IA[i] = A.IA[i];
		LU.AL.resize(LU.IA[LU.size]);
		LU.AU.resize(LU.IA[LU.size]);
		LU.JA.resize(LU.IA[LU.size]);
		LU.DI.resize(LU.size);
		for (int i = 0; i < A.IA[A.size]; i++)
			LU.JA[i] = A.JA[i];
		for (int i = 0; i < A.size; i++)
		{
			double sumD = 0;
			int i0 = A.IA[i], i1 = A.IA[i + 1];
			for (int k = i0; k < i1; k++)
			{
				double sumL = 0, sumU = 0;
				int j = A.JA[k];
				// Calculate L[i][j], U[j][i]
				int j0 = A.IA[j], j1 = A.IA[j + 1];
				int kl = i0, ku = j0;
				for (; kl < i1 && ku < j1; )
				{
					int j_kl = A.JA[kl];
					int j_ku = A.JA[ku];
					if (j_kl == j_ku)
					{
						sumL += LU.AL[kl] * LU.AU[ku];
						sumU += LU.AU[kl] * LU.AL[ku];
						kl++;
						ku++;
					}
					if (j_kl > j_ku)
						ku++;
					if (j_kl < j_ku)
						kl++;
				}
				LU.AL[k] = A.AL[k] - sumL;
				LU.AU[k] = A.AU[k] - sumU;
				LU.AU[k] /= A.DI[j];
				// Calculate sum for DI[i]
				sumD += LU.AL[k] * LU.AU[k];
			}
			// Calculate DI[i]
			LU.DI[i] = A.DI[i] - sumD;
		}
	}
	Matrix ProfToPlot(MatrixProf& A)
	{
		Matrix Res(A.size);
		int n = A.size;
		for (int i = 0; i < n; i++)
```

```
		{
			Res[i].resize(n);
			Res[i][i] = A.DI[i];
		}
		for (int i = 0; i < n; i++)
		{
			for (int jadr = A.IA[i]; jadr < A.IA[i + 1]; jadr++)
			{
				int j = A.JA[jadr];
				Res[i][j] = A.AL[jadr];
				Res[j][i] = A.AU[jadr];
			}
		}
		return Res;
	}
	//запуск Решателя
	void Calculate(vector<double>& sol)
	{
		LUFactorization(AProf, LU);
		AuxVectors TmpSolution;
		TmpSolution.Ax = vector<double>(AProf.size);
		TmpSolution.LU = vector<double>(AProf.size);
		TmpSolution.p = vector<double>(AProf.size);
		TmpSolution.r = vector<double>(AProf.size);
		TmpSolution.z = vector<double>(AProf.size);
		TmpSolution.temp = vector<double>(AProf.size);
		LOS_LU(AProf, sol, b, LU, TmpSolution, 10000, 1e-15);
		int length = AProf.IA.size();
		for (int i = 0; i < length; i++)
		{
			AProf.IA[i]++;
		}
	}

	void CalculateMSG(vector<double>& sol)
	{
		MSG solver;
		sol = solver.Solve(AProf,b);
	}

	void LOS_LU(MatrixProf& A, vector<double>& x, vector<double>& f, MatrixProf& LU,
AuxVectors& aux, int maxiter,
		double eps)
	{
		int size = A.size;
		// Calculate r0
		Multiply(A, x, aux.Ax);
		for (int i = 0; i < size; i++)
			aux.r[i] = f[i] - aux.Ax[i];
		Forward(LU, aux.r, aux.r);
		//Calculate z0
		Backward(LU, aux.z, aux.r);
			// Calculate p0
		Multiply(A, aux.z, aux.p);
		Forward(LU, aux.p, aux.p);
		double diff = MultVecs(size, aux.r, aux.r);
		int k = 0;
		for (; k < maxiter && diff >= eps; k++)
		{
			// Calculate alpha
			double dotP = MultVecs(size, aux.p, aux.p);
			double a = MultVecs(size, aux.p, aux.r) / dotP;
			// Calculate xk, rk
			for (int i = 0; i < size; i++)
```

```
                {
                        x[i] += a * aux.z[i];
                        aux.r[i] -= a * aux.p[i];
                }
                // Calculate beta
                Backward(LU, aux.Ax, aux.r);
                Multiply(A, aux.Ax, aux.temp);
                Forward(LU, aux.Ax, aux.temp);
                double b = -MultVecs(size, aux.p, aux.Ax) / dotP;
                // Calculate zk, pk
                Backward(LU, aux.temp, aux.r);
                for (int i = 0; i < size; i++)
                {
                        aux.z[i] = aux.temp[i] + b * aux.z[i];
                        aux.p[i] = aux.Ax[i] + b * aux.p[i];
                }
                // Calculate difference
                diff = MultVecs(size, aux.r, aux.r);
        }
        maxiter = k;
    }
    //Подсчёт погрешности
    double CalculateError(int tn)
    {
        double t = TheNet.t[tn];
        double err = 0;
        int length = q[tn].size();
        double norm = 0;
        for (size_t i = 0; i < length; i++)
        {
            err += pow(U(TheNet.Node[i][0], TheNet.Node[i][1], t, 0) - q[tn][i], 2);
            norm += pow(U(TheNet.Node[i][0], TheNet.Node[i][1], t, 0), 2);
        }
        return sqrt(err / norm);
    }
private:
    //параметрыпараметры
    double Ug(vector<double>& node, int k, int tn)
    {
        double t = TheNet.t[tn];
        double r = node[0];
        double z = node[1];
        return 0;
    }
    double UB(vector<double>& node, int k, int tn)
    {
        double t = TheNet.t[tn];
        double r = node[0];
        double z = node[1];
        return 22;
    }
    double Tetta(vector<double>& node, int k, int tn)
    {
        double r = node[0];
        double z = node[1];
        double t = TheNet.t[tn];
        if (tn==0)
        {
            return 500;
        }
        return 3500;
    }
    double F(double r, double z, double t, int field)
    {
```

```
	return 0;
}
double p = 7874;
double Cp = 450;
double Lambda(int field)
{
	return 92;
}
double Betta(int field)
{
	return 10;
}
double Sigma(int field)
{
	return Cp*p;
}
//utility
void ToGlobalPlot(Matrix& L, vector<double>& b, vector<int>& el)
{
	int length = L.size();
	for (int i = 0; i < length; i++)
	{
		for (int j = 0; j < length; j++)
		{
			A[el[i]][el[j]] += L[i][j];
		}
	}
	//for (int i = 0; i < length; i++)
	//{
	// this->b[el[i]] += b[i];
	//}
}
void ToGLobalProf(Matrix& A, vector<double>& b, vector<int>& el)
{
	int length = A.size();
	for (int i = 0; i < length; i++)
	{
		AProf.DI[el[i]] = AProf.DI[el[i]] + A[i][i];
	}
	for (int i = 0; i < length; i++)
	{
		int ibeg = AProf.IA[el[i]] - 1;
		for (int j = 0; j < i; j++)
		{
			int iend = AProf.IA[el[i] + 1] - 1;
			while (AProf.JA[ibeg] != el[j])
			{
				int ind = (ibeg + iend) / 2;
				if (AProf.JA[ind] <= el[j])
				{
					ibeg = ind;
				}
				else
				{
					iend = ind;
				}
			}
			AProf.AL[ibeg] += A[i][j];
			AProf.AU[ibeg] += A[j][i];
			ibeg++;
		}
	}
	for (int i = 0; i < length; i++)
	{
```

```
                this->b[el[i]] += b[i];
        }
    }
    double MultVecs(int size, vector<double>& vec1, vector<double>& vec2)
    {
        double sum = 0;
        for (int i = 0; i < size; i++)
            sum += vec1[i] * vec2[i];
        return sum;
    }
    void Multiply(MatrixProf& A, vector<double>& vec, vector<double>& res)
    {
        int size = A.size;
        for (int i = 0; i < size; i++)
        {
            res[i] = vec[i] * A.DI[i];
            for (int k = A.IA[i]; k < A.IA[i + 1]; k++)
            {
                int j = A.JA[k];
                res[i] += A.AL[k] * vec[j];
                res[j] += A.AU[k] * vec[i];
            }
        }
    }
    void Forward(MatrixProf& A, vector<double>& x, vector<double>& b)
    {
        int size = A.size;
        for (int i = 0; i < size; i++)
        {
            double sum = 0;
            int i0 = A.IA[i], i1 = A.IA[i + 1];
            for (int k = i0; k < i1; k++)
            {
                int j = A.JA[k];
                sum += A.AL[k] * x[j];
            }
            x[i] = (b[i] - sum) / A.DI[i];
        }
    }
    void Backward(MatrixProf& A, vector<double>& x, vector<double>& b)
    {
        int size = A.size;
        for (int i = 0; i < size; i++)
            x[i] = b[i];
        for (int i = size - 1; i >= 0; i--)
        {
            int i0 = A.IA[i], i1 = A.IA[i + 1];
            for (int k = i0; k < i1; k++)
            {
                int j = A.JA[k];
                x[j] -= A.AU[k] * x[i];
            }
        }
    }
};
int main()
{
    int k = 0;

    fstream nodes;
    fstream elements;
    fstream fields;
    fstream condi1;
    fstream condi2;
```

```
	fstream condi3;
	fstream result;
	nodes.open("nodes.txt");
	elements.open("elements.txt");
	fields.open("fields.txt");
	condi1.open("condi1.txt");
	condi2.open("condi2.txt");
	condi3.open("condi3.txt");
	result.open("result.txt");

	int nx=1, ny=18;
	//Net Nett(nodes,elements,fields,condi1,condi2,condi3);
	Net Nett;
	Nett.BuildNet(0.1, 1, 0.1, 0.5, nx, ny);
	Nett.AddCondi(nx,ny);
	Nett.SaveNet(nodes, elements, fields);
	Nett.BuildTnet(0, 36000, 6000);

	Eq Equation = Eq(Nett);
	cout << scientific << setprecision(15);
	result << scientific << setprecision(15);


	vector<double> sol(Equation.q[0].size());
	/*for (size_t i = 0; i < Equation.q[0].size(); i++)
	{
		sol[i] = Equation.U(Nett.Node[i][ 0], Nett.Node[i][1], Nett.t[0], 0);
	}*/
	Equation.FindSolution();
	for (size_t i = 0; i < Equation.TheNet.t.size(); i++)
	{
		result << "t = : " << Equation.TheNet.t[i] << endl;
		for (int j = 0; j < ny+1; j++)
		{
			for (size_t k = 0; k < nx; k++)
			{
				int n = (nx+1) * j + k;
				result << Equation.q[i][n] << " ";
			}
			result << endl;
		}

	}
	std::cout << "Hello World!\n";
}
```

Файл Solver.h

```
#pragma once
#include <vector>
using namespace std;
struct MatrixProf
{
    int size;
    vector<double> DI;
    vector<double> AL;
    vector<double> AU;
    vector<int> IA;
    vector<int> JA;
};

struct RawMatrix
{
public:
```

```
    int N;
    vector<double> DI;
    vector<double> AL;
    vector<int>IA;
    vector<int>JA;
};

class MSG
{
public:
    MSG();
    ~MSG();

    int MaxIterCount = 100000;
    int IterCount = 0;
    double Eps = 1e-15;
    double Difference = 0.0;

    int N = 0;
    vector<double> Ax;
    vector<double> r;
    vector<double> z;
    vector<double> xPrev;

    double DotProduct(vector<double> a, vector<double> b)
    {
        double result = 0.0;
        for (int i = 0; i < a.size(); i++)
            result += a[i] * b[i];

        return result;
    }

    double Norm(vector<double> a)
    {
        double result = 0.0;

        for (size_t i = 0; i < a.size(); i++)
        {
            result += a[i] * a[i];
        }

        return sqrt(result);
    }

    double Error(vector<double> a, vector<double> b)
    {
        double result = 0.0;
        int N = a.size();

        for (int i = 0; i < N; i++)
            result += (a[i] - b[i]) * (a[i] - b[i]);

        return sqrt(result);
    }

    RawMatrix LLT;

    vector <double> MVMultiply(vector<double> x, MatrixProf A)
    {
        vector<double> result(x.size());


        for (int i = 0; i < A.size; i++)
```

```
    {
        result[i] = x[i] * A.DI[i];
        for (int k = A.IA[i]; k < A.IA[i + 1]; k++)
        {
            int j = A.JA[k];
            result[i] += A.AL[k] * x[j];
            result[j] += A.AU[k] * x[i];
        }
    }
    return result;
}

vector<double> Solve(MatrixProf matrix, std::vector<double> B)
{
    N = matrix.size;
    InitAuxVectors(N);
    for (size_t i = 0; i < matrix.IA.size(); i++)
    {
        matrix.IA[i] -= 1;
    }
    LLTFactorization(matrix);
    vector<double> x(N);
    Ax = MVMultiply(x, matrix);
    for (int i = 0; i < N; i++)
        r[i] = B[i] - Ax[i];

    Forward(LLT, z, r);
    Backward(LLT, z, z);

    Difference = Norm(r) / Norm(B);

    double dot1 = 0;
    double dot2 = 0;

    dot1 = DotProduct(z, r);

    while (IterCount < MaxIterCount && Difference >= Eps && Error(x, xPrev) > 1.0e-10)
    {
        Ax = MVMultiply(z, matrix);

        double a = dot1 / DotProduct(Ax, z);

        for (int i = 0; i < N; i++)
        {
            xPrev[i] = x[i];
            x[i] += a * z[i];
            r[i] -= a * Ax[i];
        }

        Forward(LLT, Ax, r);
        Backward(LLT, Ax, Ax);

        dot2 = DotProduct(Ax, r);
        double b = dot2 / dot1;
        dot1 = dot2;

        for (int i = 0; i < N; i++)
            z[i] = Ax[i] + b * z[i];

        Difference = Norm(r) / Norm(B);
        IterCount++;
    }

    return x;
```

```
}

void LLTFactorization(MatrixProf matrix)
{
    RawMatrix LLT;
    LLT.N = matrix.size;
    LLT.IA.resize(LLT.N + 1);

    for (int i = 0; i < matrix.size + 1; i++)
        LLT.IA[i] = matrix.IA[i];

    LLT.AL.resize(LLT.IA[LLT.N]);
    LLT.JA.resize(LLT.IA[LLT.N]);
    LLT.DI.resize(LLT.N);

    for (int i = 0; i < matrix.IA[matrix.size]; i++)
        LLT.JA[i] = matrix.JA[i];

    for (int i = 0; i < matrix.size; i++)
    {
        double sumD = 0;
        int i0 = matrix.IA[i], i1 = matrix.IA[i + 1];

        for (int k = i0; k < i1; k++)
        {
            double sumL = 0, sumU = 0;
            int j = matrix.JA[k];

            // Calculate L[i][j], U[j][i]
            int j0 = matrix.IA[j], j1 = matrix.IA[j + 1];

            int kl = i0, ku = j0;

            for (; kl < i1 && ku < j1;)
            {
                int j_kl = matrix.JA[kl];
                int j_ku = matrix.JA[ku];

                if (j_kl == j_ku)
                {
                    sumL += LLT.AL[kl] * LLT.AL[ku];
                    kl++;
                    ku++;
                }
                if (j_kl > j_ku)
                    ku++;
                if (j_kl < j_ku)
                    kl++;
            }

            LLT.AL[k] = (matrix.AL[k] - sumL) / LLT.DI[j];

            // Calculate sum for DI[i]
            sumD += LLT.AL[k] * LLT.AL[k];
        }

        // Calculate DI[i]
        LLT.DI[i] = sqrt(matrix.DI[i] - sumD);
    }

    this->LLT = LLT;
}

void InitAuxVectors(int N)
```

```
    {
        Ax.resize(N);
        r.resize(N);
        z.resize(N);
        xPrev.resize(N);

        for (int i = 0; i < N; i++)
            xPrev[i] = 1.0;
    }

    void Forward(RawMatrix A, vector<double> &x, vector<double> b)
    {
        vector<double> di = A.DI;
        vector<double> al = A.AL;
        vector<int> ia = A.IA;
        vector<int> ja = A.JA;
        int N = A.N;


        for (int i = 0; i < N; i++)
        {
            double sum = 0;
            int i0 = ia[i], i1 = ia[i + 1];
            for (int k = i0; k < i1; k++)
            {
                int j = ja[k];
                sum += al[k] * x[j];
            }
            x[i] = (b[i] - sum) / di[i];
        }
    }

    void Backward(RawMatrix A, vector<double> &x, vector<double> b)
    {
        vector<double> di = A.DI;
        vector<double> al = A.AL;
        vector<int> ia = A.IA;
        vector<int> ja = A.JA;
        int N = A.N;

        for (int i = 0; i < N; i++)
            x[i] = b[i];

        for (int i = N - 1; i >= 0; i--)
        {
            int i0 = ia[i], i1 = ia[i + 1];
            x[i] /= di[i];
            for (int k = i0; k < i1; k++)
            {
                int j = ja[k];
                x[j] -= al[k] * x[i];

            }
        }
    }

private:

};

MSG::MSG()
{
}
```

```
MSG::~MSG()
{
}
```

Программа визуализации

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
def drawTempGraph(nx,ny):
    with open("z.txt", "r") as fh:
        data=map(float,fh.read().split('\n'))
    z = []
    for item in data:
        tmp = []
        for i in range(0,nx):
            tmp.append(item)
        z.append(tmp)
    xval = np.linspace(0.1, 1, nx)
    yval = np.linspace(0.1, 0.5, ny)
    x, y = np.meshgrid(xval, yval)
    plt.contourf(x, y, z, 19, cmap='Oranges')
    plt.colorbar()
    return z
#распределение поля
z = drawTempGraph(2,19)

#график температуры
yval = np.linspace(0.1, 0.5, 19)
plt.plot(yval,z)
```

8. Литература:

"Метод конечных элементов для решения скалярных и векторных задач" / Ю. Г. Соловейчик, М. Э. Рояк, М. Г. Персова