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

Кафедра прикладной математики

Курсовая работа по курсу

«УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ»

Группа	ПМ-01
Студент	ЧИРКОВ АРТЁМ ЕВГЕНЬЕВИЧ

Новосибирск

2023

1. **Постановка задачи**

   МКЭ для двумерной краевой задачи для параболического уравнения в (r, z) системе координат. Использовать четырёхслойную неявною схему для аппроксимации по времени. Базисные функции билинейные на прямоугольниках. Краевые условия всех типов. Коэффициент σ разложить по билинейным функциям. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

2. **Вариационная постановка**

   Необходимо решить уравнение

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

заданное в некоторой области Ω с границей и краевыми условиями:

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

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

$$\lambda \frac{\partial u}{\partial n}\bigg|_{S_3} + \beta(u|_{S_3} - u_\beta) = 0$$

Начальное условия:

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

Перепишем дифференциальное уравнение в цилиндрических координатах:

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(r\lambda\frac{\partial u}{\partial r}\right) - \frac{\partial}{\partial z}\left(\lambda\frac{\partial u}{\partial z}\right) + \sigma\frac{\partial u}{\partial t} = f$$

3. **Дискретизация по времени**

   В нашем случае аппроксимация u по времени выглядит следующим образом:

$$u(r,z,t) \approx u^{j-3}(r,z)\eta_3^j(t) + u^{j-2}(r,z)\eta_2^j(t) + u^{j-1}(r,z)\eta_1^j(t) + u^j(r,z)\eta_0^j(t),$$

где базисные функции имеют вид:

$$\eta_0^j = \frac{(t-t_{j-3})(t-t_{j-2})(t-t_{j-1})}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}$$

$$\eta_1^j = \frac{(t-t_{j-3})(t-t_{j-2})(t-t_j)}{(t_{j-1}-t_{j-3})(t_{j-1}-t_{j-2})(t_{j-1}-t_j)}$$

$$\eta_2^j = \frac{(t-t_{j-3})(t-t_{j-1})(t-t_j)}{(t_{j-2}-t_{j-3})(t_{j-2}-t_{j-1})(t_{j-2}-t_j)}$$

$$\eta_3^j = \frac{(t-t_{j-2})(t-t_{j-1})(t-t_j)}{(t_{j-3}-t_{j-2})(t_{j-3}-t_{j-1})(t_{j-3}-t_j)}$$

Подставим в неявную схему и заменим соответствующие элементы:

$$\Delta t_{01} = t_j - t_{j-1}$$

$$\Delta t_{02} = t_j - t_{j-2}$$

$$\Delta t_{03} = t_j - t_{j-3}$$

$$\Delta t_{12} = t_{j-1} - t_{j-2}$$

$$\Delta t_{13} = t_{j-1} - t_{j-3}$$

$$\Delta t_{23} = t_{j-2} - t_{j-3}$$

Вычислим производные по t функций $\eta_i^j(t)$ при $t = t_j$ с учетом обозначений выше:

$$\left.\frac{d\eta_3^j(t)}{dt}\right|_{t=t_j} = \left.\left[\frac{(t-t_{j-2})(t-t_{j-1})(t-t_j)}{(t_{j-3}-t_{j-2})(t_{j-3}-t_{j-1})(t_{j-3}-t_j)}\right]'\right|_{t=t_j}$$

$$= \left.\left[-\frac{(t-t_{j-2})(t-t_{j-1}) + (t-t_j)\left[(t-t_{j-1}) + (t-t_{j-2})\right]}{\mathit{\Delta} t_{23}\mathit{\Delta} t_{13}\mathit{\Delta} t_{03}}\right]\right|_{t=t_j} =$$

$$= -\frac{(t_j-t_{j-2})(t_j-t_{j-1})}{\mathit{\Delta} t_{23}\mathit{\Delta} t_{13}\mathit{\Delta} t_{03}} = -\frac{\mathit{\Delta} t_{01}\mathit{\Delta} t_{02}}{\mathit{\Delta} t_{03}\mathit{\Delta} t_{13}\mathit{\Delta} t_{23}}$$

$$\left.\frac{d\eta_2^j(t)}{dt}\right|_{t=t_j} = \left.\left[\frac{(t-t_{j-3})(t-t_{j-1})(t-t_j)}{(t_{j-2}-t_{j-3})(t_{j-2}-t_{j-1})(t_{j-2}-t_j)}\right]'\right|_{t=t_j}$$

$$= \left.\left[\frac{(t-t_{j-3})(t-t_{j-1}) + (t-t_j)\left[(t-t_{j-1}) + (t-t_{j-3})\right]}{\mathit{\Delta} t_{02}\mathit{\Delta} t_{13}\mathit{\Delta} t_{23}}\right]\right|_{t=t_j} =$$

$$= \frac{(t_j-t_{j-3})(t_j-t_{j-1})}{\mathit{\Delta} t_{02}\mathit{\Delta} t_{13}\mathit{\Delta} t_{23}} = \frac{\mathit{\Delta} t_{01}\mathit{\Delta} t_{03}}{\mathit{\Delta} t_{02}\mathit{\Delta} t_{13}\mathit{\Delta} t_{23}}$$

$$\left.\frac{d\eta_1^j(t)}{dt}\right|_{t=t_j} = \left.\left[\frac{(t-t_{j-3})(t-t_{j-2})(t-t_j)}{(t_{j-1}-t_{j-3})(t_{j-1}-t_{j-2})(t_{j-1}-t_j)}\right]'\right|_{t=t_j}$$

$$= \left.\left[-\frac{(t-t_{j-3})(t-t_{j-2}) + (t-t_j)\left[(t-t_{j-2}) + (t-t_{j-3})\right]}{\mathit{\Delta} t_{13}\mathit{\Delta} t_{12}\mathit{\Delta} t_{01}}\right]\right|_{t=t_j} =$$

$$= -\frac{(t_j-t_{j-3})(t_j-t_{j-2})}{\mathit{\Delta} t_{13}\mathit{\Delta} t_{12}\mathit{\Delta} t_{01}} = -\frac{\mathit{\Delta} t_{02}\mathit{\Delta} t_{03}}{\mathit{\Delta} t_{13}\mathit{\Delta} t_{12}\mathit{\Delta} t_{01}}$$

$$\left.\frac{d\eta_0^j(t)}{dt}\right|_{t=t_j} = \left.\left[\frac{(t-t_{j-3})(t-t_{j-2})(t-t_{j-1})}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}\right]'\right|_{t=t_j}$$

$$= \left.\left[\frac{(t-t_{j-3})(t-t_{j-2}) + (t-t_{j-1})\left[(t-t_{j-2}) + (t-t_{j-3})\right]}{\mathit{\Delta} t_{01}\mathit{\Delta} t_{02}\mathit{\Delta} t_{03}}\right]\right|_{t=t_j} =$$

$$= \frac{(t_j-t_{j-3})(t_j-t_{j-2}) + (t_j-t_{j-1})\left[(t_j-t_{j-2}) + (t_j-t_{j-3})\right]}{\mathit{\Delta} t_{01}\mathit{\Delta} t_{02}\mathit{\Delta} t_{03}} = \frac{\mathit{\Delta} t_{01}\mathit{\Delta} t_{02} + \mathit{\Delta} t_{01}\mathit{\Delta} t_{03} + \mathit{\Delta} t_{02}\mathit{\Delta} t_{03}}{\mathit{\Delta} t_{01}\mathit{\Delta} t_{02}\mathit{\Delta} t_{03}}$$

Тогда параболическое уравнение может быть записано в виде:

$$\sigma u^{j-3}\left(-\frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}}\right)+\sigma u^{j-2}\left(\frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{13}\Delta t_{23}}\right)+\sigma u^{j-1}\left(-\frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}}\right)$$
$$+\sigma u^{j}\left(\frac{\Delta t_{01}\Delta t_{02}+\Delta t_{01}\Delta t_{03}+\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}}\right)-div\left(\lambda gradu^{j}\right)=f^{j}, j$$
$$=\overline{3,n}$$

Будем решать это уравнение методом Галеркина. Запишем невязку в виде:

$$R(u)=\sigma u^{j}\left(\frac{\Delta t_{01}\Delta t_{02}+\Delta t_{01}\Delta t_{03}+\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}}\right)-div\left(\lambda gradu^{j}\right)-f^{j}$$
$$+\sigma u^{j-3}\left(-\frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}}\right)+\ \sigma u^{j-2}\left(\frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{13}\Delta t_{23}}\right)$$
$$+\sigma u^{j-1}\left(-\frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}}\right)$$

Обозначим

$$\gamma=\frac{\Delta t_{01}\Delta t_{02}+\Delta t_{01}\Delta t_{03}+\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}} *\ \sigma$$

$$f\ =f^{j}+\sigma u^{j-3}\left(-\frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}}\right)+\ \sigma u^{j-2}\left(\frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{13}\Delta t_{23}}\right)$$
$$+\sigma u^{j-1}\left(-\frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}}\right)$$

Потребуем, чтобы она была ортогональна некоторому пространству пробных функций Ф, т.е.:

$$\int_{\Omega}(-div(\lambda gradu^{j})+\gamma u^{j}-f)v d\Omega=0,\ \forall v\in\Phi$$

Преобразуем выражение, при помощи формулы Грина распишем, интеграл по границе с учетом краевых условий и, для исключения из суммы интеграла по S1, потребуем, чтобы Ф $=H_0^1$, т.е. чтобы пробные функции были из пространства функций, имеющих суммируемые с квадратом производные, и равных нулю на границе S1. Решение задачи u будет принадлежать пространству H_g^1. Перепишем получившееся выражение:

$$\int_{\Omega}\lambda grad\ \ u^{j} grad\ v_0\ r d\Omega+\int_{\Omega}\gamma u^{j}v_0 r\ d\Omega+\int_{S_3}\beta u^{j}v_0\ dS=$$
$$\int_{\Omega}f v r\ d\Omega+\int_{S_2}\theta v_0\ dS+\int_{S_3}\beta u_{\beta}v_0\ dS\ \forall v_0\in H_0^1$$

4. Дискретизация и базисные функции

Получим аппроксимацию уравнения Галеркина. Для этого возьмем пространства V_0^h, V_g^h, которые аппроксимируют H_0^1 и H_g^1 соответственно. Заменим $u \epsilon \mathrm{H}_g^1$ на аппроксимирующую $u^h \epsilon \mathrm{V}_g^h$ и $v \epsilon \mathrm{H}_0^1$ на $v_0^h \epsilon \mathrm{V}_0^h$:

$$\int_\Omega \lambda grad\ u^h\ grad\ v_0{}^h\ d\Omega + \int_\Omega \gamma u^h v_0{}^h\ d\Omega + \int_{S_3} \beta u^h v_0{}^h\ dS =$$
$$\int_\Omega f v_0{}^h r\ d\Omega + \int_{S_2} \theta v_0^h\ dS + \int_{S_2} \beta u_\beta v_0{}^h\ dS\ \forall v_0{}^h \in V_0^h$$

Пусть $\{\psi_i\}$ – базис V^h, тогда $v_0^h \epsilon \mathrm{V}_0^h$ может быть представлено в виде:

$$v_0^h = \sum_{i \in N_0} q_i{}^h \psi_i\ ,\ u^h = \sum_{j=1}^{n} q_j \psi_j\ ,$$

где N_0 – множество индексов i таких, что ψ_i являются базисными функциями пространств V_0^h, V_g^h. Подставив в уравнение, получим строку СЛАУ для qj $j\epsilon N_0$

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

Недостающие уравнения для компонент вектора q могут быть получены из начального условия $u|_{S_1} = u_g$:

$$\left. \sum_{j=1}^{n} q_j \psi_j \right|_{S_1} = u_g$$

Так как мы решаем задачу на прямоугольной сетке и в цилиндрических координатах, ячейками дискретизации являются прямоугольники $\Omega_{ps} = [r_p, r_{p+1}] \times [z_s, z_{s+1}]$

Выпишем билинейные базисные функции в цилиндрических координатах. Для этого сперва построим одномерные линейные функции:

$$R_1(r) = \frac{r_{p+1} - r}{h_r}, R_2(r) = \frac{r - r_p}{h_r}, h_r = r_{p+1} - r_p$$
$$Z_1(z) = \frac{z_{s+1} - z}{z_r}, Z_2(z) = \frac{z - z_s}{h_z}, h_{rz} = z_{s+1} - z_s$$

А также локальные базисные функции:

$$\hat{\psi}_1(r,z) = R_1(r) Z_1(z), \qquad \hat{\psi}_2(r,z) = R_2(r) Z_1(z),$$
$$\hat{\psi}_3(r,z) = R_1(r) Z_2(z), \qquad \hat{\psi}_2(r,z) = R_2(r) Z_2(z)$$

Компоненты локальных матриц жесткости и массы имеют вид:

$$\hat{G}_{ij} = \int_{\Omega_k} \bar{\lambda} \left(\frac{\partial \hat{\psi}_i}{\partial r} \frac{\partial \hat{\psi}_j}{\partial r} + \frac{\partial \hat{\psi}_i}{\partial z} \frac{\partial \hat{\psi}_j}{\partial z} \right) rdrdz,$$

$$\hat{M}_{ij} = \int_{\Omega_k} \hat{\sigma}\left(\hat{\psi}_i \hat{\psi}_j\right) rdrdz$$

При этом $\bar{\gamma}$ разложим по базисным функциям: $\bar{\gamma} = \sum_{k=1}^{n} \gamma_k \psi_k \quad \gamma_k = \gamma(r_k, z_k)$

$-div\left(\lambda gradu^j\right)$ аппроксимируется вектором $G * q_j,\ \sigma u^j$- $M * q_j$

Тогда уравнение на j-ом временном слое после аппроксимации примет вид:

$$\left(\frac{\Delta t_{01}\Delta t_{02} + \Delta t_{01}\Delta t_{03} + \Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}} M + G + M^{S_3}\right) q_j = b^j + \frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}} Mq^{j-3} - \frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{12}\Delta t_{23}} Mq^{j-2} + \frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}} Mq^{j-1}$$

5. Аналитические выражения для вычисления локальных матриц

Вычислим компоненты матрицы жесткости (с учетом того, что матрица симметрична):

$$\hat{G}_{11} = \bar{\lambda} \int_{r_p}^{r_p+h_r} \int_{z_s}^{z_s+h_z} \left(\left(\frac{\partial \hat{\psi}_1}{\partial r}\right)^2 + \left(\frac{\partial \hat{\psi}_1}{\partial r}\right)^2 \right) rdrdz = \bar{\lambda}\left(\frac{r_p}{h_r} + \frac{1}{2}\right)\frac{h_z}{3} + \bar{\lambda}\left(\frac{h_r r_p}{3} + \frac{{h_r}^2}{12}\right)\frac{1}{h_z} = \bar{\lambda}\left(\frac{h_z r_p}{3h_r} + \frac{h_z}{6} + \frac{h_r r_p}{3h_z} + \frac{{h_r}^2}{12h_z}\right)$$

$$\hat{G}_{12} = \bar{\lambda} \int_{r_p}^{r_p+h_r} \frac{dR_1}{dr}\frac{dR_2}{dr} rdr \int_{z_s}^{z_s+h_z} Z_1^2\, dz + \int_{r_p}^{r_p+h_r} R_1 R_2 rdr \int_{z_s}^{z_s+h_z} \left(\frac{dZ_1}{dz}\right)^2 dz = $$
$$= \bar{\lambda}\left(-\frac{r_p}{h_r} - \frac{1}{2}\right)\frac{h_z}{3} + \bar{\lambda}\left(\frac{h_r r_p}{6} + \frac{{h_r}^2}{12}\right)\frac{1}{h_z} = \bar{\lambda}\left(-\frac{h_z r_p}{3h_r} - \frac{h_z}{6} + \frac{h_r r_p}{6h_z} + \frac{{h_r}^2}{12h_z}\right)$$

$$\hat{G}_{13} = \bar{\lambda} \int_{r_p}^{r_p+h_r} \left(\frac{dR_1}{dr}\right)^2 rdr \int_{z_s}^{z_s+h_z} Z_1 Z_2\, dz + \int_{r_p}^{r_p+h_r} R_1^2 rdr \int_{z_s}^{z_s+h_z} \frac{dZ_1}{dz}\frac{dZ_2}{dz} dz = $$
$$= \bar{\lambda}\left(\frac{r_p}{h_r} + \frac{1}{2}\right)\frac{h_z}{6} - \bar{\lambda}\left(\frac{h_r r_p}{3} + \frac{{h_r}^2}{12}\right)\frac{1}{h_z} = \bar{\lambda}\left(\frac{h_z r_p}{6h_r} + \frac{h_z}{12} - \frac{h_r r_p}{3h_z} - \frac{{h_r}^2}{12h_z}\right)$$

$$\hat{G}_{14} = \hat{G}_{23} = \bar{\lambda} \int_{r_p}^{r_p+h_r} \frac{dR_1}{dr}\frac{dR_2}{dr} rdr \int_{z_s}^{z_s+h_z} Z_1 Z_2\, dz + \int_{r_p}^{r_p+h_r} R_1 R_2 rdr \int_{z_s}^{z_s+h_z} \frac{dZ_1}{dz}\frac{dZ_2}{dz} dz = $$
$$= \bar{\lambda}\left(-\frac{r_p}{h_r} - \frac{1}{2}\right)\frac{h_z}{6} - \bar{\lambda}\left(\frac{h_r r_p}{6} + \frac{{h_r}^2}{12}\right)\frac{1}{h_z} = \bar{\lambda}\left(-\frac{h_z r_p}{6h_r} - \frac{h_z}{12} - \frac{h_r r_p}{6h_z} - \frac{{h_r}^2}{12h_z}\right)$$

$$\hat{G}_{22} = \bar{\lambda} \int\limits_{r_p}^{r_p+h_r} \left(\frac{dR_2}{dr}\right)^2 rdr \int\limits_{z_s}^{z_s+h_z} Z_1^2\, dz \ + \int\limits_{r_p}^{r_p+h_r} R_2^2 rdr \int\limits_{z_s}^{z_s+h_z} \left(\frac{dZ_1}{dz}\right)^2 dz =$$

$$= \bar{\lambda}\left(\frac{r_p}{h_r}+\frac{1}{2}\right)\frac{h_z}{3} + \bar{\lambda}\left(\frac{h_r r_p}{3}+\frac{{h_r}^2}{4}\right)\frac{1}{h_z} = \bar{\lambda}\left(\frac{h_z r_p}{3h_r}+\frac{h_z}{6}+\frac{h_r r_p}{3h_z}+\frac{{h_r}^2}{4h_z}\right)$$

$$\hat{G}_{24} = \bar{\lambda} \int\limits_{r_p}^{r_p+h_r} \left(\frac{dR_2}{dr}\right)^2 rdr \int\limits_{z_s}^{z_s+h_z} Z_1 Z_2\, dz \ + \int\limits_{r_p}^{r_p+h_r} R_2^2 rdr \int\limits_{z_s}^{z_s+h_z} \frac{dZ_1}{dz}\frac{dZ_2}{dz} dz =$$

$$= \bar{\lambda}\left(\frac{r_p}{h_r}+\frac{1}{2}\right)\frac{h_z}{6} - \bar{\lambda}\left(\frac{h_r r_p}{3}+\frac{{h_r}^2}{4}\right)\frac{1}{h_z} = \bar{\lambda}\left(\frac{h_z r_p}{6h_r}+\frac{h_z}{12}-\frac{h_r r_p}{3h_z}-\frac{{h_r}^2}{4h_z}\right)$$

$$\hat{G}_{33} = \bar{\lambda} \int\limits_{r_p}^{r_p+h_r} \left(\frac{dR_1}{dr}\right)^2 rdr \int\limits_{z_s}^{z_s+h_z} Z_2^2\, dz \ + \int\limits_{r_p}^{r_p+h_r} R_1^2 rdr \int\limits_{z_s}^{z_s+h_z} \left(\frac{dZ_2}{dz}\right)^2 dz =$$

$$= \bar{\lambda}\left(\frac{r_p}{h_r}+\frac{1}{2}\right)\frac{h_z}{3} + \bar{\lambda}\left(\frac{h_r r_p}{3}+\frac{{h_r}^2}{12}\right)\frac{1}{h_z} = \bar{\lambda}\left(\frac{h_z r_p}{3h_r}+\frac{h_z}{6}+\frac{h_r r_p}{3h_z}+\frac{{h_r}^2}{12h_z}\right)$$

$$\hat{G}_{34} = \bar{\lambda} \int\limits_{r_p}^{r_p+h_r} \frac{dR_1}{dr}\frac{dR_2}{dr} rdr \int\limits_{z_s}^{z_s+h_z} Z_2^2\, dz \ + \int\limits_{r_p}^{r_p+h_r} R_1 R_2 rdr \int\limits_{z_s}^{z_s+h_z} \left(\frac{dZ_2}{dz}\right)^2 dz =$$

$$= \bar{\lambda}\left(-\frac{r_p}{h_r}-\frac{1}{2}\right)\frac{h_z}{3} + \bar{\lambda}\left(\frac{h_r r_p}{6}+\frac{{h_r}^2}{12}\right)\frac{1}{h_z} = \bar{\lambda}\left(-\frac{h_z r_p}{3h_r}-\frac{h_z}{6}+\frac{h_r r_p}{6h_z}+\frac{{h_r}^2}{12h_z}\right)$$

$$\hat{G}_{44} = \bar{\lambda} \int\limits_{r_p}^{r_p+h_r} \left(\frac{dR_2}{dr}\right)^2 rdr \int\limits_{z_s}^{z_s+h_z} Z_2^2\, dz \ + \int\limits_{r_p}^{r_p+h_r} R_2^2 rdr \int\limits_{z_s}^{z_s+h_z} \left(\frac{dZ_2}{dz}\right)^2 dz =$$

$$= \bar{\lambda}\left(\frac{r_p}{h_r}+\frac{1}{2}\right)\frac{h_z}{3} + \bar{\lambda}\left(\frac{h_r r_p}{3}+\frac{{h_r}^2}{4}\right)\frac{1}{h_z} = \bar{\lambda}\left(\frac{h_z r_p}{3h_r}+\frac{h_z}{6}+\frac{h_r r_p}{3h_z}+\frac{{h_r}^2}{4h_z}\right)$$

$$\hat{G} = \frac{\bar{\lambda}}{6}\frac{h_z r_p}{h_r}\begin{bmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{bmatrix} + \frac{\bar{\lambda}}{12} h_z \begin{bmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{bmatrix}$$

$$+ \frac{\bar{\lambda}}{6}\frac{h_r r_p}{h_z}\begin{bmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & -2 & 1 & 2 \end{bmatrix} + \frac{\bar{\lambda}}{12}\frac{{h_r}^2}{h_z}\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 3 & -1 & -3 \\ -1 & -1 & 1 & 1 \\ -1 & -3 & 1 & 3 \end{bmatrix}$$

Элементы матрицы массы будут выглядеть следующим образом (матрица симметрична):

$$\widehat{M}_{ij} = \int_{\Omega_k} \hat{\sigma}\left(\hat{\psi}_i \hat{\psi}_j\right) rdrdz = \sum_{k=1}^{n} \sigma_k \int_{\Omega_k} \hat{\psi}_i \hat{\psi}_j \hat{\psi}_k\, rdrdz$$

$$\widehat{M}_{11} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{4} + \frac{h_r}{20}\right)\frac{h_z}{4} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{4} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{4} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{12} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{4} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{4} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{13} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{4} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{4} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{14} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{22} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{4} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{4} + \frac{h_r}{5}\right)\frac{h_z}{4} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{4} + \frac{h_r}{5}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{23} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{24} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{4} + \frac{h_r}{5}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{4} + \frac{h_r}{5}\right)\frac{h_z}{12}\right)h_r$$

$$\widehat{M}_{33} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{4} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{4} + \frac{h_r}{20}\right)\frac{h_z}{4} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{4}\right)h_r$$

$$\widehat{M}_{34} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{30}\right)\frac{h_z}{4} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{4}\right)h_r$$

$$\widehat{M}_{44} = \left(\sigma\left(r_p, z_s\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{12} + \sigma\left(r_{p+1}, z_s\right)\left(\frac{r_p}{4} + \frac{h_r}{5}\right)\frac{h_z}{12} + \sigma\left(r_p, z_{s+1}\right)\left(\frac{r_p}{12} + \frac{h_r}{20}\right)\frac{h_z}{4} + \sigma\left(r_{p+1}, z_{s+1}\right)\left(\frac{r_p}{4} + \frac{h_r}{5}\right)\frac{h_z}{4}\right)h_r$$

Локальный вектор правой части $\hat{b}$ найдем при помощи разложения f в виде билинейного интерполянта $\sum_{v=1}^{4} \widehat{f_v}\,\widehat{\psi_v}$

$$\hat{b} = \hat{C} * \hat{f}$$

где $\hat{C}$ равна матрица массы при $\gamma \equiv 1$

6. Краевые условия

Краевые условия первого рода

Благодаря краевым условиям первого рода нам известно значение решения в узлах на границе S1. Если в узле с номером i задано первое краевое условие ug, тогда диагональный элемент Aii мы заменяем на 1, а элемент вектора правой части Fi на число равному краевому условию. Все внедиагональные элементы на i-й строчке заменим на 0, то i-е уравнение фактически примет вид qi=ug, или qi=us, что соответствует первому краевому условию.

Краевые условия второго рода

Пусть на ребре Si,j задано краевое условие второго рода. Данное краевое условие вносит вклад только в правую часть СЛАУ.

$$b^{S_{i,j}} = \frac{h_{i,j}}{6}\begin{pmatrix} 2{\theta_1}^{S_{i,j}} + {\theta_2}^{S_{i,j}} \\ {\theta_1}^{S_{i,j}} + 2{\theta_2}^{S_{i,j}} \end{pmatrix}$$

Где i,j – номера узлов ребра, на котором задано краевое условие.

Краевые условия третьего рода

При учете третьих краевых условий формируются локальная матрица и вектор правой части, которые заносятся в СЛАУ аналогично локальной матрицы конечного элемента и локального вектора правой части конечного элемента.

$$M^{S_{i,j}} = \sigma\frac{h_{i,j}}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad b^{S_{i,j}} = \frac{\beta^{S_{i,j}} h_{i,j}}{6}\begin{pmatrix} 2{u_{\beta 1}}^{S_{i,j}} + {u_{\beta 2}}^{S_{i,j}} \\ {u_{\beta 1}}^{S_{i,j}} + 2{u_{\beta 2}}^{S_{i,j}} \end{pmatrix}$$

7. Тесты

Исследования на порядок аппроксимации и сходимости по времени

q0 q1 q2 – задаются аналитически

t = [0, 1, 2, 3, 4]	U = t	f = 1	r = [0, 1, 2] z = [0, 1, 2]	$\sigma = 1$	$\lambda = 1$

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3				
r	z	q	q*	| q - q* |
0,00E+00	0,00E+00	3,00E+00	3,00E+00	0,00E+00
1,00E+00	0,00E+00	3,00E+00	3,00E+00	0,00E+00
2,00E+00	0,00E+00	3,00E+00	3,00E+00	0,00E+00
0,00E+00	1,00E+00	3,00E+00	3,00E+00	0,00E+00
1,00E+00	**1,00E+00**	**3,00E+00**	**3,00E+00**	**0,00E+00**
2,00E+00	1,00E+00	3,00E+00	3,00E+00	0,00E+00
0,00E+00	2,00E+00	3,00E+00	3,00E+00	0,00E+00

1,00E+00	2,00E+00	3,00E+00	3,00E+00	0,00E+00
2,00E+00	2,00E+00	3,00E+00	3,00E+00	0,00E+00

t = [0, 1, 2, 3, 4]	U = t^2	f = 2t	r = [0, 1, 2] z = [0, 1, 2]	$\sigma = 1$	$\lambda = 1$

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3				
r	z	q	q*	| q - q* |
0,00E+00	0,00E+00	9,00E+00	9,00E+00	0,00E+00
1,00E+00	0,00E+00	9,00E+00	9,00E+00	0,00E+00
2,00E+00	0,00E+00	9,00E+00	9,00E+00	0,00E+00
0,00E+00	1,00E+00	9,00E+00	9,00E+00	0,00E+00
1,00E+00	**1,00E+00**	**9,00E+00**	**9,00E+00**	**0,00E+00**
2,00E+00	1,00E+00	9,00E+00	9,00E+00	0,00E+00
0,00E+00	2,00E+00	9,00E+00	9,00E+00	0,00E+00
1,00E+00	2,00E+00	9,00E+00	9,00E+00	0,00E+00
2,00E+00	2,00E+00	9,00E+00	9,00E+00	0,00E+00

t = [0, 1, 2, 3, 4]	U = t^3	f = 3t^2	r = [0, 1, 2] z = [0, 1, 2]	$\sigma = 1$	$\lambda = 1$

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3				
r	z	q	q*	| q - q* |
0,00E+00	0,00E+00	27,00E+00	27,00E+00	0,00E+00
1,00E+00	0,00E+00	27,00E+00	27,00E+00	0,00E+00
2,00E+00	0,00E+00	27,00E+00	27,00E+00	0,00E+00
0,00E+00	1,00E+00	27,00E+00	27,00E+00	0,00E+00
1,00E+00	**1,00E+00**	**27,00E+00**	**27,00E+00**	**0,00E+00**
2,00E+00	1,00E+00	27,00E+00	27,00E+00	0,00E+00
0,00E+00	2,00E+00	27,00E+00	27,00E+00	0,00E+00
1,00E+00	2,00E+00	27,00E+00	27,00E+00	0,00E+00
2,00E+00	2,00E+00	27,00E+00	27,00E+00	0,00E+00

t = [0, 1, 2, 3, 4]	U = t^4	f = 4t^3	r = [0, 1, 2] z = [0, 1, 2]	$\sigma = 1$	$\lambda = 1$

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3

r	z	Q	q*	| q - q* |
0,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
2,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
0,0000E+00	1,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	**1,0000E+00**	**81,0000E+00**	**82,7234E+00**	**1,7234E+00**
2,0000E+00	1,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
0,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
2,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00

Порядок аппроксимации = 3

Уменьшим шаг по t в 2 раза

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3				
r	z	Q	q*	| q - q* |
0,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
2,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
0,0000E+00	1,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	**1,0000E+00**	**81,0000E+00**	**81,2825E+00**	**2,8253E-01**
2,0000E+00	1,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
0,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
2,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00

Уменьшим шаг по t в 4 раза

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3				
r	z	Q	q*	| q - q* |
0,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
2,0000E+00	0,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
0,0000E+00	1,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	**1,0000E+00**	**81,0000E+00**	**81,0351E+00**	**3,5176E-02**
2,0000E+00	1,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
0,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
1,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00
2,0000E+00	2,0000E+00	81,0000E+00	81,0000E+00	0,0000E+00

$\|\|q^h-q^*\|\|$	1,7234E+00	$\|\|q^h-q^*\|\|/\|\|q^{h/2}-q^*\|\|$
$\|\|q^{h/2}-q^*\|\|$	2,8253E-01	6,0998E+00
$\|\|q^{h/4}-q^*\|\|$	3,5176E-02	**8,0310+00**

Порядок сходимости = 3

Функция u зависит от времени и пространственных координат

q0 q1 q2 – задаются аналитически

t = [0, 1, 2, 3, 4]	U = t*z^4	f = z^4-12*t*z^2	r = [1, 2, 3] z = [1, 2, 3]	$\sigma = 1$	$\lambda = 1$

Количество узлов сетки: 9

Количество конечных элементов: 4

t = 3				
r	z	Q	q*	| q - q* |
1,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
3,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,0000E+00	2,0000E+00	4,8000E+01	4,8000E+01	0,0000E+00
2,0000E+00	2,0000E+00	4,8000E+01	4,6277E+01	1,7234E+00
3,0000E+00	2,0000E+00	4,8000E+01	4,8000E+01	0,0000E+00
1,0000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00
2,0000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00
3,0000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00

Подробим сетку на 2

Количество узлов сетки: 25

Количество конечных элементов: 16

t = 3				
r	z	Q	q*	| q - q* |
1,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,5000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,5000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
3,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,0000E+00	1,5000E+00	1,5188E+01	1,5188E+01	0,0000E+00
1,5000E+00	1,5000E+00	1,5188E+01	1,4954E+01	2,3301E-01
2,0000E+00	1,5000E+00	1,5188E+01	1,4920E+01	2,6727E-01
2,5000E+00	1,5000E+00	1,5188E+01	1,4974E+01	2,1320E-01
3,0000E+00	1,5000E+00	1,5188E+01	1,5188E+01	0,0000E+00
1,0000E+00	2,0000E+00	4,8000E+01	4,8000E+01	0,0000E+00
1,5000E+00	2,0000E+00	4,8000E+01	4,7715E+01	2,8463E-01
2,0000E+00	2,0000E+00	4,8000E+01	4,7665E+01	3,3520E-01
2,5000E+00	2,0000E+00	4,8000E+01	4,7744E+01	2,5601E-01

3,0000E+00	2,0000E+00	4,8000E+01	4,8000E+01	0,0000E+00
1,0000E+00	2,5000E+00	1,1719E+02	1,1719E+02	0,0000E+00
1,5000E+00	2,5000E+00	1,1719E+02	1,1695E+02	2,3301E-01
2,0000E+00	2,5000E+00	1,1719E+02	1,1692E+02	2,6727E-01
2,5000E+00	2,5000E+00	1,1719E+02	1,1697E+02	2,1320E-01
3,0000E+00	2,5000E+00	1,1719E+02	1,1719E+02	0,0000E+00
1,0000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00
1,5000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00
2,0000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00
2,5000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00
3,0000E+00	3,0000E+00	2,4300E+02	2,4300E+02	0,0000E+00

Подробим сетку на 4

Количество узлов сетки: 81

Количество конечных элементов: 64

t = 3				
r	z	Q	q*	| q - q* |
1,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,2500E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,5000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,7500E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,2500E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,5000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
2,7500E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
3,0000E+00	1,0000E+00	3,0000E+00	3,0000E+00	0,0000E+00
1,0000E+00	1,2500E+00	7,3242E+00	7,3242E+00	0,0000E+00
1,2500E+00	1,2500E+00	7,3242E+00	7,2993E+00	2,4950E-02
1,5000E+00	1,2500E+00	7,3242E+00	7,2890E+00	3,5247E-02
1,7500E+00	1,2500E+00	7,3242E+00	7,2848E+00	3,9453E-02
2,0000E+00	1,2500E+00	7,3242E+00	7,2841E+00	4,0101E-02
2,2500E+00	1,2500E+00	7,3242E+00	7,2863E+00	3,7890E-02
2,5000E+00	1,2500E+00	7,3242E+00	7,2918E+00	3,2387E-02
2,7500E+00	1,2500E+00	7,3242E+00	7,3024E+00	2,1789E-02
3,0000E+00	1,2500E+00	7,3242E+00	7,3242E+00	9,7700E-15
1,0000E+00	1,5000E+00	1,5188E+01	1,5188E+01	0,0000E+00
1,2500E+00	1,5000E+00	1,5188E+01	1,5150E+01	3,7221E-02
1,5000E+00	1,5000E+00	1,5188E+01	1,5132E+01	5,5082E-02
1,7500E+00	1,5000E+00	1,5188E+01	1,5125E+01	6,2600E-02
2,0000E+00	1,5000E+00	1,5188E+01	1,5124E+01	6,3762E-02
2,2500E+00	1,5000E+00	1,5188E+01	1,5128E+01	5,9746E-02
2,5000E+00	1,5000E+00	1,5188E+01	1,5138E+01	4,9906E-02
2,7500E+00	1,5000E+00	1,5188E+01	1,5156E+01	3,1622E-02
3,0000E+00	1,5000E+00	1,5188E+01	1,5188E+01	0,0000E+00
1,0000E+00	1,7500E+00	2,8137E+01	2,8137E+01	0,0000E+00
1,2500E+00	1,7500E+00	2,8137E+01	2,8093E+01	4,3234E-02
1,5000E+00	1,7500E+00	2,8137E+01	2,8072E+01	6,5058E-02

1,7500E+00	1,7500E+00	2,8137E+01	2,8062E+01	7,4559E-02
2,0000E+00	1,7500E+00	2,8137E+01	2,8061E+01	7,6030E-02
2,2500E+00	1,7500E+00	2,8137E+01	2,8066E+01	7,0876E-02
2,5000E+00	1,7500E+00	2,8137E+01	2,8078E+01	5,8435E-02
2,7500E+00	1,7500E+00	2,8137E+01	2,8100E+01	3,6231E-02
3,0000E+00	1,7500E+00	2,8137E+01	2,8137E+01	9,9476E-14
1,0000E+00	2,0000E+00	4,8000E+01	4,8000E+01	0,0000E+00
1,2500E+00	2,0000E+00	4,8000E+01	4,7955E+01	4,5049E-02
1,5000E+00	2,0000E+00	4,8000E+01	4,7932E+01	6,8098E-02
1,7500E+00	2,0000E+00	4,8000E+01	4,7922E+01	7,8232E-02
2,0000E+00	2,0000E+00	4,8000E+01	4,7920E+01	7,9803E-02
2,2500E+00	2,0000E+00	4,8000E+01	4,7926E+01	7,4266E-02
2,5000E+00	2,0000E+00	4,8000E+01	4,7939E+01	6,0994E-02
2,7500E+00	2,0000E+00	4,8000E+01	4,7962E+01	3,7594E-02
3,0000E+00	2,0000E+00	4,8000E+01	4,8000E+01	0,0000E+00
1,0000E+00	2,2500E+00	7,6887E+01	7,6887E+01	0,0000E+00
1,2500E+00	2,2500E+00	7,6887E+01	7,6843E+01	4,3230E-02
1,5000E+00	2,2500E+00	7,6887E+01	7,6822E+01	6,5052E-02
1,7500E+00	2,2500E+00	7,6887E+01	7,6812E+01	7,4552E-02
2,0000E+00	2,2500E+00	7,6887E+01	7,6811E+01	7,6023E-02
2,2500E+00	2,2500E+00	7,6887E+01	7,6816E+01	7,0870E-02
2,5000E+00	2,2500E+00	7,6887E+01	7,6828E+01	5,8430E-02
2,7500E+00	2,2500E+00	7,6887E+01	7,6850E+01	3,6228E-02
3,0000E+00	2,2500E+00	7,6887E+01	7,6887E+01	0,0000E+00
1,0000E+00	2,5000E+00	1,1719E+02	1,1719E+02	0,0000E+00
1,2500E+00	2,5000E+00	1,1719E+02	1,1715E+02	3,7211E-02
1,5000E+00	2,5000E+00	1,1719E+02	1,1713E+02	5,5068E-02
1,7500E+00	2,5000E+00	1,1719E+02	1,1712E+02	6,2585E-02
2,0000E+00	2,5000E+00	1,1719E+02	1,1712E+02	6,3747E-02
2,2500E+00	2,5000E+00	1,1719E+02	1,1713E+02	5,9732E-02
2,5000E+00	2,5000E+00	1,1719E+02	1,1714E+02	4,9894E-02

$\|\|q^{h}-q^{*}\|\|$	1,7234E+00	$\|\|q^{h}-q^{*}\|\|/\|\|q^{h/2}-q^{*}\|\|$
$\|\|q^{h/2}-q^{*}\|\|$	2,3886E-01	7,2153E+00
$\|\|q^{h/4}-q^{*}\|\|$	2,9968E-02	7,9703E+00

Порядок сходимости = 3

Проверим неполиномальную функцию

t = [0; 0,1; 0,3; 0,7; 1; 1,2]	U = $e^t + z$	f = $2e^t$	r = [1001, 1002, 1003, 1004] z = [1, 2, 3, 4]	$\sigma = 2$	$\lambda = 1$

Количество узлов сетки: 16

Количество конечных элементов: 9

t = 1				
r	z	Q	q*	| q - q* |
1,0010E+03	1,0000E+00	3,7183E+00	3,7658E+00	-4,7520E-02
1,0020E+03	1,0000E+00	3,7183E+00	3,7658E+00	-4,7520E-02
1,0030E+03	1,0000E+00	3,7183E+00	3,7658E+00	-4,7520E-02
1,0040E+03	1,0000E+00	3,7183E+00	3,7658E+00	-4,7520E-02
1,0010E+03	2,0000E+00	4,7183E+00	4,7250E+00	-6,7600E-03
1,0020E+03	2,0000E+00	4,7183E+00	4,7250E+00	-6,7600E-03
1,0030E+03	2,0000E+00	4,7183E+00	4,7250E+00	-6,7600E-03
1,0040E+03	2,0000E+00	4,7183E+00	4,7250E+00	-6,7600E-03
1,0010E+03	3,0000E+00	5,7183E+00	5,7537E+00	-3,5460E-02
1,0020E+03	3,0000E+00	5,7183E+00	5,7537E+00	-3,5460E-02
1,0030E+03	3,0000E+00	5,7183E+00	5,7537E+00	-3,5460E-02
1,0040E+03	3,0000E+00	5,7183E+00	5,7537E+00	-3,5460E-02
1,0010E+03	4,0000E+00	6,7183E+00	6,7932E+00	-7,4930E-02
1,0020E+03	4,0000E+00	6,7183E+00	6,7932E+00	-7,4930E-02
1,0030E+03	4,0000E+00	6,7183E+00	6,7932E+00	-7,4930E-02
1,0040E+03	4,0000E+00	6,7183E+00	6,7932E+00	-7,4930E-02

8. Вывод

Благодаря исследованиям, удалось выяснить, что порядок аппроксимации четырёх-слойной неявной схемы равен трём, так как погрешность появляется на полиноме четвертой степени по t. Порядок сходимости так же равен трём

9. Текст программы

Main.cpp

```
#include <fstream>
#include <iostream>
#include <vector>
#include <math.h>
#include "Solver.h"
#include "Generate.h"

MyVector q1, q2, q3, q4;
struct node
{
   double r;
   double z;
};
struct material
{
   double lambda;
   int gamma_id;
};
struct element
{
   std::vector<int> node_loc;
   int mater;
   int f_id;
};
std::vector<node> all_nodes;
std::vector<element> all_elems;
std::vector<material> all_materials;
std::vector<std::pair<int, std::vector<int>>> S1;
std::vector<std::pair<int, std::vector<int>>> S2_r;
std::vector<std::pair<int, std::vector<int>>> S2_z;
std::vector<std::pair<int, std::vector<int>>> S3_r;
std::vector<std::pair<int, std::vector<int>>> S3_z;
std::vector<double> time_grid;
int i_t = 0;
double gamma(double r, double z, int gam_id)
{
   switch (gam_id)
   {
   case 0:
      return 1;
   default:
      std::cout << "can't find gamma № " << gam_id << "\n";
      break;
   }
}
double beta(double r, double z, int beta_id)
{
   switch (beta_id)
   {
   case 0:
      return 1;
   default:
      std::cout << "can't find gamma № " << beta_id << "\n";
      break;
   }
}
double func_f(double r, double z, int f_id)
{
   double t = time_grid[i_t];
   switch (f_id)
   {
   case 0:
      return z*z*z*z-12*t*z*z;
```

```
    default:
        std::cout << "can't find f № " << f_id << "\n";
        break;
    }
}
double func_S(double r, double z, int s_id)
{
    double t = time_grid[i_t];
    switch (s_id)
    {
    case 0:
        return t*z*z*z*z;
    default:
        std::cout << "can't find S № " << s_id << "\n";
        break;
    }
}

int Input()
{
    int N, Nmat, Kel, NS1, Ntime, NS;
    std::ifstream in;
    in.open("info.txt");
    in >> N >> Nmat >> Kel >> NS1;
    in.close();
    in.open("rz.txt");
    all_nodes.resize(N);
    for (int i = 0; i < N; i++)
    {
        in >> all_nodes[i].r >> all_nodes[i].z;
    }
    in.close();
    in.open("time.txt");
    in >> Ntime;
    time_grid.resize(Ntime);
    for (int i = 0; i < Ntime; i++)
    {
        in >> time_grid[i];
    }
    in.close();
    in.open("q0 q1 q2.txt");
    q1.Size(N);
    for (int i = 0; i < N; i++)
    {
        in >> q1.vect[i];
    }
    q2.Size(N);
    for (int i = 0; i < N; i++)
    {
        in >> q2.vect[i];
    }
    q3.Size(N);
    for (int i = 0; i < N; i++)
    {
        in >> q3.vect[i];
    }
    q4.Size(N);
    in.close();
    in.open("S1.txt");
    S1.resize(NS1);
    for (int i = 0; i < NS1; i++)
    {
        int size;
        in >> size >> S1[i].first;
        S1[i].second.resize(size);
        for (int j = 0; j < size; j++)
        {
            in >> S1[i].second[j];
```

```
        }
    }
    in.close();
    in.open("S2_r.txt");
    in >> NS;
    S2_r.resize(NS);
    for (int i = 0; i < NS; i++)
    {
        int size;
        in >> size >> S2_r[i].first;
        S2_r[i].second.resize(size);
        for (int j = 0; j < size; j++)
        {
            in >> S2_r[i].second[j];
        }
    }
    in.close();
    in.open("S2_z.txt");
    in >> NS;
    S2_z.resize(NS);
    for (int i = 0; i < NS; i++)
    {
        int size;
        in >> size >> S2_z[i].first;
        S2_z[i].second.resize(size);
        for (int j = 0; j < size; j++)
        {
            in >> S2_z[i].second[j];
        }
    }
    in.close();
    in.open("S3_r.txt");
    in >> NS;
    S3_r.resize(NS);
    for (int i = 0; i < NS; i++)
    {
        int size;
        in >> size >> S3_r[i].first;
        S3_r[i].second.resize(size);
        for (int j = 0; j < size; j++)
        {
            in >> S3_r[i].second[j];
        }
    }
    in.close();
    in.open("S3_z.txt");
    in >> NS;
    S3_z.resize(NS);
    for (int i = 0; i < NS; i++)
    {
        int size;
        in >> size >> S3_z[i].first;
        S3_z[i].second.resize(size);
        for (int j = 0; j < size; j++)
        {
            in >> S3_z[i].second[j];
        }
    }
    in.close();
    in.open("material.txt");
    all_materials.resize(Nmat);

for (int i = 0; i < Nmat; i++)
    {
        in >> all_materials[i].lambda >> all_materials[i].gamma_id;
    }
    in.close();
    in.open("elem.txt");
```

```
    all_elems.resize(Kel);
    for (int i = 0; i < Kel; i++)
    {
        all_elems[i].node_loc.resize(4);
        in >> all_elems[i].node_loc[0] >> all_elems[i].node_loc[1]
            >> all_elems[i].node_loc[2] >> all_elems[i].node_loc[3]
            >> all_elems[i].mater >> all_elems[i].f_id;
    }
    in.close();

    return 0;
}

double GetG_Loc(double rp, double lambda, double
    hr, double hz,
    std::vector<std::vector<double>>& G_loc)
{
    double a1 = (lambda * hz * rp) / (6 * hr),
        a2 = (lambda * hz) / (12),
        a3 = (lambda * hr * rp) / (6 * hz),
        a4 = (lambda * hr * hr) / (12 * hz);
    G_loc[0][0] = 2 * a1 + 2 * a2 + 2 * a3 + 1 * a4;
    G_loc[0][1] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[0][2] = 1 * a1 + 1 * a2 - 2 * a3 - 1 * a4;
    G_loc[0][3] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[1][0] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[1][1] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
    G_loc[1][2] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[1][3] = 1 * a1 + 1 * a2 - 2 * a3 - 3 * a4;
    G_loc[2][0] = 1 * a1 + 1 * a2 - 2 * a3 - 1 * a4;
    G_loc[2][1] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[2][2] = 2 * a1 + 2 * a2 + 2 * a3 + 1 * a4;
    G_loc[2][3] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[3][0] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[3][1] = 1 * a1 + 1 * a2 - 2 * a3 - 3 * a4;
    G_loc[3][2] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[3][3] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
    return 0;
}

double GetM_Loc(double rp, double zs, int gam,
    double hr, double hz, std::vector<std::vector<double>>& M_loc)
{
    double g1 = gamma(rp, zs, gam),
        g2 = gamma(rp + hr, zs, gam),
        g3 = gamma(rp, zs + hz, gam),
        g4 = gamma(rp + hr, zs + hz, gam);
    M_loc[0][0] = hr * (
        g1 * (rp / 4 + hr / 20) * hz / 4 +
        g2 * (rp / 12 + hr / 30) * hz / 4 +
        g3 * (rp / 4 + hr / 20) * hz / 12 +
        g4 * (rp / 12 + hr / 30) * hz / 12);
    M_loc[0][1] = hr * (
        g1 * (rp / 12 + hr / 30) * hz / 4 +
        g2 * (rp / 12 + hr / 20) * hz / 4 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[0][2] = hr * (
        g1 * (rp / 4 + hr / 20) * hz / 12 +
        g2 * (rp / 12 + hr / 30) * hz / 12 +
        g3 * (rp / 4 + hr / 20) * hz / 12 +
        g4 * (rp / 12 + hr / 30) * hz / 12);
    M_loc[0][3] = hr * (
        g1 * (rp / 12 + hr / 30) * hz / 12 +
        g2 * (rp / 12 + hr / 20) * hz / 12 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[1][0] = hr * (
```

```
      g1 * (rp / 12 + hr / 30) * hz / 4 +
      g2 * (rp / 12 + hr / 20) * hz / 4 +
      g3 * (rp / 12 + hr / 30) * hz / 12 +
      g4 * (rp / 12 + hr / 20) * hz / 12);
   M_loc[1][1] = hr * (
      g1 * (rp / 12 + hr / 20) * hz / 4 +
      g2 * (rp / 4 + hr / 5) * hz / 4 +
      g3 * (rp / 12 + hr / 20) * hz / 12 +
      g4 * (rp / 4 + hr / 5) * hz / 12);
   M_loc[1][2] = hr * (
      g1 * (rp / 12 + hr / 30) * hz / 12 +
      g2 * (rp / 12 + hr / 20) * hz / 12 +
      g3 * (rp / 12 + hr / 30) * hz / 12 +
      g4 * (rp / 12 + hr / 20) * hz / 12);
   M_loc[1][3] = hr * (
      g1 * (rp / 12 + hr / 20) * hz / 12 +
      g2 * (rp / 4 + hr / 5) * hz / 12 +
      g3 * (rp / 12 + hr / 20) * hz / 12 +
      g4 * (rp / 4 + hr / 5) * hz / 12);
   M_loc[2][0] = hr * (
      g1 * (rp / 4 + hr / 20) * hz / 12 +
      g2 * (rp / 12 + hr / 30) * hz / 12 +
      g3 * (rp / 4 + hr / 20) * hz / 12 +
      g4 * (rp / 12 + hr / 30) * hz / 12);
   M_loc[2][1] = hr * (
      g1 * (rp / 12 + hr / 30) * hz / 12 +
      g2 * (rp / 12 + hr / 20) * hz / 12 +
      g3 * (rp / 12 + hr / 30) * hz / 12 +
      g4 * (rp / 12 + hr / 20) * hz / 12);
   M_loc[2][2] = hr * (
      g1 * (rp / 4 + hr / 20) * hz / 12 +
      g2 * (rp / 12 + hr / 30) * hz / 12 +
      g3 * (rp / 4 + hr / 20) * hz / 4 +
      g4 * (rp / 12 + hr / 30) * hz / 4);
   M_loc[2][3] = hr * (
      g1 * (rp / 12 + hr / 30) * hz / 12 +
      g2 * (rp / 12 + hr / 20) * hz / 12 +
      g3 * (rp / 12 + hr / 30) * hz / 4 +
      g4 * (rp / 12 + hr / 20) * hz / 4);
   M_loc[3][0] = hr * (
      g1 * (rp / 12 + hr / 30) * hz / 12 +
      g2 * (rp / 12 + hr / 20) * hz / 12 +
      g3 * (rp / 12 + hr / 30) * hz / 12 +
      g4 * (rp / 12 + hr / 20) * hz / 12);
   M_loc[3][1] = hr * (
      g1 * (rp / 12 + hr / 20) * hz / 12 +
      g2 * (rp / 4 + hr / 5) * hz / 12 +
      g3 * (rp / 12 + hr / 20) * hz / 12 +
      g4 * (rp / 4 + hr / 5) * hz / 12);
   M_loc[3][2] = hr * (
      g1 * (rp / 12 + hr / 30) * hz / 12 +
      g2 * (rp / 12 + hr / 20) * hz / 12 +
      g3 * (rp / 12 + hr / 30) * hz / 4 +
      g4 * (rp / 12 + hr / 20) * hz / 4);
   M_loc[3][3] = hr * (
      g1 * (rp / 12 + hr / 20) * hz / 12 +
      g2 * (rp / 4 + hr / 5) * hz / 12 +
      g3 * (rp / 12 + hr / 20) * hz / 4 +
      g4 * (rp / 4 + hr / 5) * hz / 4);
   return 0;
}

int Getb_Loc(double rp, double zs, double hr, double hz,
   std::vector<double>& b_loc, int f_id)
{
   double f1 = func_f(rp, zs, f_id),
      f2 = func_f(rp + hr, zs, f_id),
      f3 = func_f(rp, zs + hz, f_id),
```

```
      f4 = func_f(rp + hr, zs + hz, f_id);
   b_loc[0] =
      f1 * (hr * hz / 3 * (rp / 3 + hr / 12)) +
      f2 * (hr * hz / 3 * (rp / 6 + hr / 12)) +
      f3 * (hr * hz / 6 * (rp / 3 + hr / 12)) +
      f4 * (hr * hz / 6 * (rp / 6 + hr / 12));
   b_loc[1] =
      f1 * (hr * hz / 3 * (rp / 6 + hr / 12)) +
      f2 * (hr * hz / 3 * (rp / 3 + hr / 4)) +
      f3 * (hr * hz / 6 * (rp / 6 + hr / 12)) +
      f4 * (hr * hz / 6 * (rp / 3 + hr / 4));
   b_loc[2] =
      f1 * (hr * hz / 6 * (rp / 3 + hr / 12)) +
      f2 * (hr * hz / 6 * (rp / 6 + hr / 12)) +
      f3 * (hr * hz / 3 * (rp / 3 + hr / 12)) +
      f4 * (hr * hz / 3 * (rp / 6 + hr / 12));
   b_loc[3] =
      f1 * (hr * hz / 6 * (rp / 6 + hr / 12)) +
      f2 * (hr * hz / 6 * (rp / 3 + hr / 4)) +
      f3 * (hr * hz / 3 * (rp / 6 + hr / 12)) +
      f4 * (hr * hz / 3 * (rp / 3 + hr / 4));
   return 0;
}

int Get_Loc(std::vector<std::vector<double>>& M_loc, std::vector<std::vector<double>>&
G_loc, int el_id)
{
   element el = all_elems[el_id];
   double hr = all_nodes[el.node_loc[1]].r - all_nodes[el.node_loc[0]].r,
      hz = all_nodes[el.node_loc[2]].z - all_nodes[el.node_loc[0]].z;

   GetM_Loc(all_nodes[el.node_loc[0]].r, all_nodes[el.node_loc[0]].z,
      all_materials[el.mater].gamma_id, hr, hz, M_loc);

   GetG_Loc(all_nodes[el.node_loc[0]].r, all_materials[el.mater].lambda, hr, hz, G_loc);
   return 0;
}

int Get_Loc_b(std::vector<double>& b_loc, int el_id)
{
   element el = all_elems[el_id];
   double hr = all_nodes[el.node_loc[1]].r - all_nodes[el.node_loc[0]].r,
      hz = all_nodes[el.node_loc[2]].z - all_nodes[el.node_loc[0]].z;

   Getb_Loc(all_nodes[el.node_loc[0]].r, all_nodes[el.node_loc[0]].z, hr, hz, b_loc,
el.f_id);
   return 0;
}

int GeneratePortrait(MyMatrix& A, int N, int Kel)
{
   std::vector<int>* ia = &A.ia,
      * ja = &A.ja;
   ia->resize(N + 1);
   ja->resize(16 * Kel);

std::vector<int> temp_list1(16 * Kel),
      temp_list2(16 * Kel);
   std::vector<int> listbeg(N);
   int listsize = 0;
   for (int i = 0; i < N; i++)
   {
      listbeg[i] = 0;
   }
   for (int ielem = 0; ielem < Kel; ielem++)
   {
      for (int i = 0; i < 4; i++)
      {
```

```
            int k = all_elems[ielem].node_loc[i];
            for (int j = i + 1; j < 4; j++)
            {
               int ind1 = k;
               int ind2 = all_elems[ielem].node_loc[j];
               if (ind2 < ind1)
               {
                  ind1 = ind2;
                  ind2 = k;
               }
               int iaddr = listbeg[ind2];
               if (iaddr == 0)
               {
                  listsize++;
                  listbeg[ind2] = listsize;
                  temp_list1[listsize] = ind1;
                  temp_list2[listsize] = 0;
               }
               else
               {
                  while (temp_list1[iaddr] < ind1 && temp_list2[iaddr] > 0)
                  {
                     iaddr = temp_list2[iaddr];
                  }
                  if (temp_list1[iaddr] > ind1)
                  {
                     listsize++;
                     temp_list1[listsize] = temp_list1[iaddr];
                     temp_list2[listsize] = temp_list2[iaddr];
                     temp_list1[iaddr] = ind1;
                     temp_list2[iaddr] = listsize;
                  }
                  else if (temp_list1[iaddr] < ind1)
                  {
                     listsize++;
                     temp_list2[iaddr] = listsize;
                     temp_list1[listsize] = ind1;
                     temp_list2[listsize] = 0;
                  }
               }
            }
         }
      }
      (*ia)[0] = 0;
      for (int i = 0; i < N; i++)
      {
         (*ia)[i + 1] = (*ia)[i];
         int iaddr = listbeg[i];
         while (iaddr != 0)
         {
            (*ja)[(*ia)[i + 1]] = temp_list1[iaddr];
            (*ia)[i + 1]++;
            iaddr = temp_list2[iaddr];
         }
      }
      ja->resize((*ia)[N]);
      return 0;
   }

   int AddLocal(std::vector<int>& iaM,
      std::vector<int>& jaM, std::vector<double>& diM,
      std::vector<double>& alM, std::vector<double>& auM,
      std::vector<std::vector<double>>& M_loc, int el_id)
   {
      std::vector<int> L = all_elems[el_id].node_loc;
      int k = all_elems[el_id].node_loc.size();
      for (int i = 0; i < k; i++)
      {
```

```
        diM[L[i]] += M_loc[i][i];
    }
    for (int i = 0; i < 4; i++)
    {
        int temp = iaM[L[i]];
        for (int j = 0; j < i; j++)
        {
            for (int k = temp; k < iaM[L[i] + 1]; k++)
            {
                if (jaM[k] == L[j])
                {
                    alM[k] += M_loc[i][j];
                    auM[k] += M_loc[j][i];
                    k++;
                    break;
                }
            }
        }
    }
    return 0;
}

int AddLocal_b(std::vector<double>& b,
    std::vector<double>& b_loc, int el_id)
{
    std::vector<int> L = all_elems[el_id].node_loc;
    int k = all_elems[el_id].node_loc.size();
    for (int i = 0; i < k; i++)
    {
        b[L[i]] += b_loc[i];
    }
    return 0;
}

int SetS1(std::vector<int>& ia, std::vector<int>&
    ja, std::vector<double>& di, std::vector<double>& al,
    std::vector<double>& au, std::vector<double>& b)
{
    int NS1 = S1.size();
    for (int i = 0; i < NS1; i++)
    {
        int s1_id = S1[i].first;
        for (int j = 0; j < S1[i].second.size(); j++)
        {
            int node_id = S1[i].second[j];
            di[node_id] = 1;
            b[node_id] = func_S(all_nodes[node_id].r, all_nodes[node_id].z, s1_id);
            for (int k = ia[node_id]; k < ia[node_id + 1]; k++)
            {
                al[k] = 0;
            }
            for (int k = 0; k < ja.size(); k++)
            {
                if (ja[k] == node_id)
                {
                    au[k] = 0;
                }
            }
        }
    }
    return 0;
}

double GetM_Loc_dim2_r(double rp, double hr,
    std::vector<std::vector<double>>& M_loc)
{
    M_loc[0][0] = hr / 6 * (2 * rp + hr / 2);
    M_loc[0][1] = hr / 6 * (rp + hr / 2);
```

```
    M_loc[1][0] = hr / 6 * (rp + hr / 2);
    M_loc[1][1] = hr / 6 * (2 * rp + 3 * hr / 2);
    return 0;
}

int Getb_Loc_dim2_r(double rp, double zs, double hr,
    std::vector<double>& b_loc, int f_id)
{
    double f1 = func_S(rp, zs, f_id),
        f2 = func_S(rp + hr, zs, f_id);
    b_loc[0] = f1 * (hr / 6 * (2 * rp + hr / 2)) + f2 * (hr / 6 * (rp + hr / 2));
    b_loc[1] = f1 * (hr / 6 * (rp + hr / 2)) + f2 * (hr / 6 * (2 * rp + 3 * hr / 2));
    return 0;
}

double GetM_Loc_dim2_z(double zp, double hz,
    std::vector<std::vector<double>>& M_loc)
{
    M_loc[0][0] = hz / 3;
    M_loc[0][1] = hz / 6;
    M_loc[1][0] = hz / 6;
    M_loc[1][1] = hz / 3;
    return 0;
}

int Getb_Loc_dim2_z(double rp, double zs, double hz,
    std::vector<double>& b_loc, int s_id)
{
    double f1 = func_S(rp, zs, s_id),
        f2 = func_S(rp, zs + hz, s_id);
    b_loc[0] = f1 * (hz / 3) + f2 * (hz / 6);
    b_loc[1] = f1 * (hz / 6) + f2 * (hz / 3);
    return 0;
}

int AddLocal_dim2(std::vector<int>& iaM,
    std::vector<int>& jaM, std::vector<double>& diM,
    std::vector<double>& alM,
    std::vector<double>& auM,
    std::vector<std::vector<double>>& M_loc,
    int node1, int node2)
{
    std::vector<int> L(2);
    L[0] = node1;
    L[1] = node2;
    int k = 2;
    for (int i = 0; i < k; i++)
    {
        diM[L[i]] += M_loc[i][i];
    }
    for (int i = 0; i < 2; i++)

{
        int temp = iaM[L[i]];
        for (int j = 0; j < i; j++)
        {
            for (int k = temp; k < iaM[L[i] + 1]; k++)
            {
                if (jaM[k] == L[j])
                {
                    alM[k] += M_loc[i][j];
                    auM[k] += M_loc[j][i];
                    k++;
                    break;
                }
            }
        }
    }
```

```
	return 0;
}

int Set_S2(MyMatrix& MS)
{
	std::vector<double> b_loc(2);
	int NS2 = S2_r.size();
	for (int i = 0; i < NS2; i++)
	{
		int s2_id = S2_r[i].first;
		for (int j = 0; j < S2_r[i].second.size() - 1; j++)
		{
			int node_id1 = S2_r[i].second[j],
				node_id2 = S2_r[i].second[j + 1];
			double hr = all_nodes[node_id2].r - all_nodes[node_id1].r;

			Getb_Loc_dim2_r(all_nodes[node_id1].r, all_nodes[node_id1].z, hr, b_loc, s2_id);
			MS.b.vect[node_id1] += b_loc[0];
			MS.b.vect[node_id2] += b_loc[1];
		}
	}
	NS2 = S2_z.size();
	for (int i = 0; i < NS2; i++)
	{
		int s2_id = S2_z[i].first;
		for (int j = 0; j < S2_z[i].second.size() - 1; j++)
		{
			int node_id1 = S2_z[i].second[j],
				node_id2 = S2_z[i].second[j + 1];
			double hz = all_nodes[node_id2].z - all_nodes[node_id1].z;

			Getb_Loc_dim2_z(all_nodes[node_id1].r, all_nodes[node_id1].z, hz, b_loc, s2_id);
			MS.b.vect[node_id1] += b_loc[0];
			MS.b.vect[node_id2] += b_loc[1];
		}
	}
	return 0;
}

int Set_S3(MyMatrix& MS, bool flag)
{
	std::vector<double> b_loc(2);
	std::vector<std::vector<double>> M_loc(2);
	M_loc[0].resize(2);
	M_loc[1].resize(2);
	int NS2 = S3_r.size();
	for (int i = 0; i < NS2; i++)
	{
		int s3_id = S3_r[i].first;
		for (int j = 0; j < S3_r[i].second.size() - 1; j++)
		{
			int node_id1 = S3_r[i].second[j],
				node_id2 = S3_r[i].second[j + 1],
				beta_id = 0;
			double hr = all_nodes[node_id2].r - all_nodes[node_id1].r,
				be = beta(all_nodes[node_id1].r, all_nodes[node_id1].z, beta_id);

			if (flag)
				GetM_Loc_dim2_r(all_nodes[node_id1].r, hr, M_loc);
			for (int k = 0; k < M_loc.size(); k++)
			{
				for (int l = 0; flag && l < M_loc[k].size(); l++)
					M_loc[k][l] *= be;
				b_loc[k] *= be;
			}
			if (flag)
				AddLocal_dim2(MS.ia, MS.ja,
					MS.di, MS.al, MS.au, M_loc, node_id1, node_id2);
```

```
            Getb_Loc_dim2_r(all_nodes[node_id1].r, all_nodes[node_id1].z, hr, b_loc, s3_id);
            MS.b.vect[node_id1] += b_loc[0];
            MS.b.vect[node_id2] += b_loc[1];
         }
      }
      NS2 = S3_z.size();
      for (int i = 0; i < NS2; i++)
      {
         int s3_id = S3_z[i].first;
         for (int j = 0; j < S3_z[i].second.size() - 1; j++)
         {
            int node_id1 = S3_z[i].second[j],
               node_id2 = S3_z[i].second[j + 1],
               beta_id = 0;
            double hz = all_nodes[node_id2].z - all_nodes[node_id1].z,
               be = beta(all_nodes[node_id1].r, all_nodes[node_id1].z, beta_id);

            if (flag)
               GetM_Loc_dim2_z(all_nodes[node_id1].z, hz, M_loc);
            for (int k = 0; k < M_loc.size(); k++)
            {
               for (int l = 0; flag && l < M_loc[k].size(); l++)
                  M_loc[k][l] *= be;
               b_loc[k] *= be;
            }
            if (flag)
               AddLocal_dim2(MS.ia, MS.ja,
                  MS.di, MS.al, MS.au, M_loc, node_id1, node_id2);

            Getb_Loc_dim2_z(all_nodes[node_id1].r, all_nodes[node_id1].z, hz, b_loc, s3_id);
            MS.b.vect[node_id1] += b_loc[0];
            MS.b.vect[node_id2] += b_loc[1];
         }
      }
      return 0;
   }

   int main()
   {
      Input();
      MyMatrix M, G, A, MS;
      GeneratePortrait(M, all_nodes.size(), all_elems.size());
      G.ia = M.ia;
      G.ja = M.ja;
      M.au.resize(M.ja.size());
      M.al.resize(M.ja.size());
      M.N = all_nodes.size();
      M.di.resize(M.N);
      MS.ia = M.ia;
      MS.ja = M.ja;
      MS.au.resize(MS.ja.size());
      MS.al.resize(MS.ja.size());
      MS.N = all_nodes.size();
      MS.di.resize(MS.N);
      MS.b.Size(M.N);
      G.au.resize(G.ja.size());
      G.al.resize(G.ja.size());
      G.N = all_nodes.size();
      G.di.resize(G.N);

      std::vector<std::vector<double>> M_loc(4), G_loc(4);
      for (int i = 0; i < 4; i++)
      {
         M_loc[i].resize(4);
         G_loc[i].resize(4);
      }
      std::vector<double> b_loc(4);
```

```
for (int i = 0; i < all_elems.size(); i++)
{
   Get_Loc(M_loc, G_loc, i);
   AddLocal(M.ia, M.ja, M.di, M.al, M.au, M_loc, i);
   AddLocal(G.ia, G.ja, G.di, G.al, G.au, G_loc, i);
}
std::ofstream out("result.txt");
out.imbue(std::locale("Russian"));
out.precision(15);
double
   dt01 = 0,
   dt02 = 0,
   dt03 = 0,
   dt12 = 0,
   dt13 = 0,
   dt23 = 0;
bool change_matrix;
for (i_t = 3; i_t < time_grid.size(); i_t++)
{
   change_matrix = false;
   if (dt01 != time_grid[i_t] - time_grid[i_t - 1] ||
      dt02 != time_grid[i_t] - time_grid[i_t - 2] ||
      dt03 != time_grid[i_t] - time_grid[i_t - 3] ||
      dt12 != time_grid[i_t - 1] - time_grid[i_t - 2] ||
      dt13 != time_grid[i_t - 1] - time_grid[i_t - 3] ||
      dt23 != time_grid[i_t - 2] - time_grid[i_t - 3])
      change_matrix = true;
   dt01 = time_grid[i_t] - time_grid[i_t - 1];
   dt02 = time_grid[i_t] - time_grid[i_t - 2];
   dt03 = time_grid[i_t] - time_grid[i_t - 3];
   dt12 = time_grid[i_t - 1] - time_grid[i_t - 2];
   dt13 = time_grid[i_t - 1] - time_grid[i_t - 3];
   dt23 = time_grid[i_t - 2] - time_grid[i_t - 3];
   if (change_matrix)
   {
      A = G;
      A.b.Size(G.N);
      A = A + M * ((dt01 * dt02 + dt01 * dt03 + dt02 * dt03) / (dt01 * dt02 * dt03));
   }
   for (int i = 0; i < A.b.vect.size(); i++)
   {
      A.b.vect[i] = 0;
      MS.b.vect[i] = 0;
   }
   for (int i = 0; i < all_elems.size(); i++)
   {
      Get_Loc_b(b_loc, i);
      AddLocal_b(A.b.vect, b_loc, i);
   }
   MyVector temp;
   temp.Size(all_nodes.size());
   M.Ax(q1, temp);
   A.b = A.b + temp * ((dt01 * dt02) / (dt03 * dt13 * dt23));
   M.Ax(q2, temp);
   A.b = A.b + temp * ((-dt01 * dt03) / (dt02 * dt12 * dt23));
   M.Ax(q3, temp);
   A.b = A.b + temp * ((dt02 * dt03) / (dt01 * dt12 * dt13));
   Set_S2(MS);
   Set_S3(MS, change_matrix);
   A = A + MS;
   A.b = A.b + MS.b;
   SetS1(A.ia, A.ja, A.di, A.al, A.au, A.b.vect);
   if (change_matrix)
   {
      std::fill(MS.al.begin(), MS.al.end(), 0);
      std::fill(MS.au.begin(), MS.au.end(), 0);
      std::fill(MS.di.begin(), MS.di.end(), 0);
   }
```

```
      Solver slau(A);
      slau.CGM_LU();
      slau.getx0(q4.vect);
      out << "time = " << ";" << time_grid[i_t] << "\n";
      for (int i = 0; i < all_nodes.size(); i++)
      {
         out << all_nodes[i].r << "\t" <<
            all_nodes[i].z << "\t" << q4.vect[i] << "\n";
      }
      q1.vect.swap(q2.vect);
      q2.vect.swap(q3.vect);
      q3.vect.swap(q4.vect);
   }
   return 0;
}
```

Solver.cpp

```
Solver::Solver(int size)
{
   N = size;
   A.di.resize(N);
   A.ia.resize(N + 1);
   A.ja.resize((N * N - N) / 2);
   A.au.resize((N * N - N) / 2);
   A.al.resize((N * N - N) / 2);
   A.b.Size(N);
   A.N = N;
   A.ia[0] = 0;
   A.ia[1] = 0;
   A.di[0] = 1;
   A.b.vect[0] += 1;
   for (int i = 1; i < N; i++)
   {
      for (int j = 0; j < i; j++)
      {
         A.au[A.ia[i] + j] = 1. / (i + j + 1);
         // т.к. (i + 1) + (i - k + 1 + j) - 1, k = i
         A.b.vect[j] += (i + 1) * A.au[A.ia[i]
            + j];
         A.al[A.ia[i] + j] = 1. / (i + j + 1);
         A.b.vect[i] += (j + 1) * A.al[A.ia[i]
            + j];
         A.ja[A.ia[i] + j] = j;
      }
      A.ia[i + 1] = A.ia[i] + i;
      A.di[i] = 1. / (i + i + 1);
      A.b.vect[i] += (i + 1) * A.di[i];
   }
   maxIter = 10000;
   eps = 1E-14;
   std::ofstream fout;
   fout.precision(16);
   fout.open("kuslau.txt");
   fout << N << " " << maxIter << " " << eps;
   fout.close();
   fout.open("di.txt");
   for (int i = 0; i < N; i++)
      fout << A.di[i] << " ";
   fout.close();
   fout.open("ig.txt");
   for (int i = 0; i <= N; i++)
      fout << A.ia[i] << " ";
   fout.close();
   fout.open("jg.txt");
   for (int i = 0; i < A.ia[N]; i++)
      fout << A.ja[i] << " ";
   fout.close();
```

```
    fout.open("ggu.txt");
    for (int i = 0; i < A.ia[N]; i++)
        fout << A.au[i] << " ";
    fout.close();
    fout.open("ggl.txt");
    for (int i = 0; i < A.ia[N]; i++)
        fout << A.al[i] << " ";
    fout.close();
    fout.open("pr.txt");
    for (int i = 0; i < N; i++)
        fout << A.b.vect[i] << " ";
    fout.close();
    x0.Size(N);
    r.Size(N);
    z.Size(N);
    p.Size(N);
    Ar.Size(N);
    y.Size(N);
    L.resize(A.ia[N]);
    D.resize(N);
    U.resize(A.ia[N]);
    normB = A.b.Norm();
    iter = 0;
    normR = 0;
}

Solver::Solver(std::string filename)
{
    std::ifstream in("kuslau.txt");
    in >> N >> maxIter >> eps;
    A.ReadMatrix(N);
    x0.Size(N);
    r.Size(N);
    z.Size(N);
    p.Size(N);
    Ar.Size(N);
    y.Size(N);
    L.resize(A.ia[N]);
    D.resize(N);
    U.resize(A.ia[N]);
    normB = A.b.Norm();
    iter = 0;
    normR = 0;
}

Solver::Solver(MyMatrix _A)
{
    N = _A.N;
    maxIter = 10000;
    eps = 1E-15;
    A = _A;
    x0.Size(N);
    r.Size(N);
    z.Size(N);
    p.Size(N);
    Ar.Size(N);
    y.Size(N);
    L.resize(A.ia[N]);
    D.resize(N);
    U.resize(A.ia[N]);
    normB = A.b.Norm();
    iter = 0;
    normR = 0;
}

void Solver::output(std::string filename)
{
    std::ofstream out(filename);
```

```
    out.imbue(std::locale("Russian"));
    out.precision(15);
    for (int i = 0; i < N; i++)
        out << x0.vect[i] << std::endl;
}

void Solver::getx0(std::vector<double>& x)
{
    for (int i = 0; i < N; i++)
        x[i] = x0.vect[i];
}

void Solver::CGM_LU()
{
    std::cout.precision(15);
    FactLU(L, U, D);
    double r_r = 0, Az_z = 0;
    double a = 0, B = 0;
    A.Ax(x0, r); // r0 = A*x0
    A.b - r; // r0 = B - A*x0
    Direct(L, D, r, r); // r0 = L^(-1) * (B - A* x0)
    Reverse(L, D, r, r); // r0 = L^(-T) * L^(-1) * (B - A * x0)
    A.ATx(r, y); // y0 = A^(T) * L^(-T) * L^(-1)* (B - A * x0)
    Direct(U, r, y); // r0 = U-t * A^(T) * L^(-T) * L^(-1) * (B - A * x0)
    z = r; // z0 = r0
    r_r = r * r;
    normR = sqrt(r_r) / normB;
    for (iter = 1; iter < maxIter + 1 && normR >=
        eps; iter++)
    {
        Reverse(U, y, z); // y = U^(-1) * z
        A.Ax(y, p); // p = A * U^(-1) * z
        Direct(L, D, p, p); // p = L-1 * A * U ^ (-1)* z
        Reverse(L, D, p, p); // p = L-t * p
        A.ATx(p, Ar); // Ar = At * P
        Direct(U, Ar, Ar); // Ar = U-t * Ar
        Az_z = Ar * z; // (Ar,z)
        a = r_r / Az_z;
        // x(k) = x(k-1) + z(k-1)*a(k-1)
        // r(k) = r(k-1) - AT*A*z(k-1)*a(k-1)
        for (int i = 0; i < N; i++)
        {
            x0.vect[i] = x0.vect[i] + z.vect[i] *
                a;
            r.vect[i] = r.vect[i] - Ar.vect[i] *
                a;
        }
        // B(k) = (r(k), r(k)) / (r(k-1), r(k-1))
        B = 1.0 / r_r;
        r_r = r * r;
        B *= r_r;
        // z(k) = r(k) + B(k)*z(k-1)
        for (int i = 0; i < A.N; i++)
        {
            z.vect[i] = r.vect[i] + z.vect[i] *
                B;
        }
        normR = sqrt(r_r) / normB;
        //std::cout << iter << ". " << normR << std::endl;
    }
    // x0 = U^(-1) * x0
    Reverse(U, x0, x0);
}

void Solver::LOS_LU()
{
    std::cout.precision(15);
    FactLU(L, U, D);
```

```
    double p_p = 0, p_r = 0, r_r = 0, Ar_p = 0;
    double a = 0, B = 0, eps2 = 1e-10;
    A.Ax(x0, y); // y = A * x0
    A.b - y; // y = B - A * x0
    Direct(L, D, r, y); // r0 = L^(-1) * (B - A * x0)
    Reverse(U, z, r); // z0 = U^(-1) * r0
    A.Ax(z, y); // y = A * z0
    Direct(L, D, p, y); // p0 = L^(-1) * (A * z0)
    r_r = r * r;
    normR = sqrt(r_r) / normB;
    for (iter = 1; iter < maxIter + 1 && normR >= eps; iter++)
    {
       p_p = p * p;
       p_r = p * r;
       a = p_r / p_p;
       // x(k) = x(k-1) + a(k) * z(k-1)
       // r(k) = r(k-1) - a(k) * p(k-1)
       for (int i = 0; i < N; i++)
       {
          x0.vect[i] = x0.vect[i] + z.vect[i] * a;
          r.vect[i] = r.vect[i] - p.vect[i] * a;
       }
       Reverse(U, y, r); // y = U^(-1) * r(k)
       A.Ax(y, Ar); // Ar = A * U^(-1) * r(k)
       Direct(L, D, Ar, Ar); // Ar = L^(-1) * A * U ^ (-1)* r(k)
       Ar_p = Ar * p; // (Ar, p)
       B = -(Ar_p / p_p);
       // z(k) = U^(-1) * r(k) + B(k) * z(k-1)
       // p(k) = L^(-1) * A * U^(-1) * r(k) + B(k) * p(k - 1)
       for (int i = 0; i < N; i++)
       {
          z.vect[i] = y.vect[i] + z.vect[i] * B;
          p.vect[i] = Ar.vect[i] + p.vect[i] * B;
       }
       if (r_r - (r_r - a * a * p_p) < eps2)
          r_r = r * r;
       else
          r_r = r_r - a * a * p_p;
       normR = sqrt(r_r) / normB;
       std::cout << iter << ". " << normR << std::endl;
    }
}

void Solver::FactLU(std::vector<double>& L,
    std::vector<double>& U, std::vector<double>& D)
{
    L = A.al;
    U = A.au;
    D = A.di;
    double l, u, d;
    for (int k = 0; k < N; k++)
    {
       d = 0;
       int i0 = A.ia[k], i1 = A.ia[k + 1];
       int i = i0;
       for (; i0 < i1; i0++)
       {
          l = 0;
          u = 0;
          int j0 = i, j1 = i0;
          for (; j0 < j1; j0++)
          {
             int t0 = A.ia[A.ja[i0]],
                t1 = A.ia[A.ja[i0] + 1];
             for (; t0 < t1; t0++)
             {
                if (A.ja[j0] == A.ja[t0])
                {
```

```
                    l += L[j0] * U[t0];
                    u += L[t0] * U[j0];
                 }
              }
           }
           L[i0] -= l;
           U[i0] -= u;
           U[i0] /= D[A.ja[i0]];
           d += L[i0] * U[i0];
        }
        D[k] -= d;
     }
}

// L*y = B
void Solver::Direct(std::vector<double>& L,
   std::vector<double>& D, MyVector& y, MyVector& b)
{
   y = b;
   for (int i = 0; i < N; i++)
   {
      double sum = 0;
      int k0 = A.ia[i], k1 = A.ia[i + 1];
      int j;
      for (; k0 < k1; k0++)
      {
         j = A.ja[k0];
         sum += y.vect[j] * L[k0];
      }
      double buf = y.vect[i] - sum;
      y.vect[i] = buf / D[i];
   }
}

// U^(T)*y = B
void Solver::Direct(std::vector<double>& L,
   MyVector& y, MyVector& b)
{
   y = b;
   for (int i = 0; i < N; i++)
   {
      double sum = 0;
      int k0 = A.ia[i], k1 = A.ia[i + 1];
      int j;
      for (; k0 < k1; k0++)
      {
         j = A.ja[k0];
         sum += y.vect[j] * L[k0];
      }
      y.vect[i] -= sum;
   }
}

// U*x = y
void Solver::Reverse(std::vector<double>& U,
   MyVector& x, MyVector& y)
{
   x = y;
   for (int i = N - 1; i >= 0; i--)
   {
      int k0 = A.ia[i], k1 = A.ia[i + 1];
      int j;
      for (; k0 < k1; k0++)
      {
         j = A.ja[k0];
         x.vect[j] -= x.vect[i] * U[k0];
      }
   }
```

```
}

// L^(T)*x = y
void Solver::Reverse(std::vector<double>& U,
   std::vector<double>& D, MyVector& x, MyVector& y)
{
   x = y;
   for (int i = N - 1; i >= 0; i--)
   {
      int k0 = A.ia[i], k1 = A.ia[i + 1];
      int j;
      x.vect[i] /= D[i];
      for (; k0 < k1; k0++)
      {
         j = A.ja[k0];
         x.vect[j] -= x.vect[i] * U[k0];
      }
   }
}
```

Vector.cpp

```
#include "MyVector.h"

MyVector::MyVector()
{
}

void MyVector::Size(int N)
{
   vect.resize(N);
}

// read from filename
void MyVector::ReadVector(std::string filename)
{
   if (vect.size() < 1)
      return;
   std::ifstream in(filename);
   for (int i = 0; i < vect.size(); i++)
   {
      in >> vect[i];
   }
   in.close();
}

// this = a; a = a
MyVector& MyVector::operator=(const MyVector& a)
{
   if (this != &a)
      this->vect = a.vect;
   return *this;
}

// this = a * this
MyVector& MyVector::operator*(const double a)
{
   for (int i = 0; i < this->vect.size(); i++)
      this->vect[i] *= a;
   return *this;
}

// (this, a)
double MyVector::operator* (const MyVector& a)
{
   double res = 0;
   if (this->vect.size() != a.vect.size())
      return res;
   for (int i = 0; i < this->vect.size(); i++)
```

```
      res += this->vect[i] * a.vect[i];
   return res;
}

// a = this - a;
MyVector& MyVector::operator-(MyVector& a)
{
   if (this->vect.size() != a.vect.size())
   {
      return *this;
   }
   else
   {
      for (int i = 0; i < this->vect.size(); i++)
      {
         a.vect[i] = this->vect[i] - a.vect[i];
      }
      return a;
   }
}

// this = this + a;
MyVector& MyVector::operator+(MyVector& a)
{
   if (this->vect.size() != a.vect.size())
   {
      return *this;
   }
   else
   {
      for (int i = 0; i < this->vect.size(); i++)
      {
         this->vect[i] = this->vect[i] + a.vect[i];
      }
      return *this;
   }
}

// || this ||
double MyVector::Norm()
{
   return sqrt((*this) * (*this));
}
```

Matrix.cpp

```
#include "MyMatrix.h"

MyMatrix::MyMatrix(void)
{
}

void MyMatrix::ReadMatrix(int size)
{
   N = size;
   std::ifstream in;
   in.open("ig.txt");
   ia.resize(N + 1);
   for (int i = 0; i < N + 1; i++)
   {
      in >> ia[i];
   }
   in.close();
   if (ia[0])
      for (int i = 0; i < N + 1; i++)
      {
         ia[i]--;
      }
```

```
    in.open("jg.txt");
    ja.resize(ia[N]);
    for (int i = 0; i < ia[N]; i++)
    {
        in >> ja[i];
    }
    in.close();
    if (ja[0])
        for (int i = 0; i < ia[N]; i++)
        {
            ja[i]--;
        }
    in.open("di.txt");
    di.resize(N);
    for (int i = 0; i < N; i++)
    {
        in >> di[i];
    }
    in.close();
    in.open("ggu.txt");
    au.resize(ia[N]);
    for (int i = 0; i < ia[N]; i++)
    {
        in >> au[i];
    }
    in.close();
    in.open("ggl.txt");
    al.resize(ia[N]);
    for (int i = 0; i < ia[N]; i++)
    {
        in >> al[i];
    }
    in.close();
    b.Size(N);
    b.ReadVector("pr.txt");
}

// y = Ax
void MyMatrix::Ax(MyVector& x, MyVector& y)
{
    for (int i = 0; i < N; i++)
    {
        y.vect[i] = di[i] * x.vect[i];
        for (int j = ia[i]; j < ia[i + 1]; j++)
        {
            int k = ja[j];
            y.vect[i] += al[j] * x.vect[k];
            y.vect[k] += au[j] * x.vect[i];
        }
    }
}

// y = Ax
void MyMatrix::Ax(std::vector<double>& x, std::vector<double>& y)
{
    for (int i = 0; i < N; i++)
    {
        y[i] = di[i] * x[i];
        for (int j = ia[i]; j < ia[i + 1]; j++)
        {
            int k = ja[j];
            y[i] += al[j] * x[k];
            y[k] += au[j] * x[i];
        }
    }
}

// y = A^(T)x
```

```
void MyMatrix::ATx(MyVector& x, MyVector& y)
{
   for (int i = 0; i < N; i++)
   {
      y.vect[i] = di[i] * x.vect[i];
      for (int j = ia[i]; j < ia[i + 1]; j++)
      {
         int k = ja[j];
         y.vect[i] += au[j] * x.vect[k];
         y.vect[k] += al[j] * x.vect[i];
      }
   }
}

MyMatrix& MyMatrix::operator+ (MyMatrix B)
{
   if (N != B.N)
   {
      std::cout << "A и B разного размера\n";
      return *this;
   }
   for (int i = 0; i < N; i++)
   {
      this->di[i] += B.di[i];
      for (int j = ia[i]; j < ia[i + 1]; j++)
      {
         int k = ja[j];
         if (k != B.ja[j])
         {
            std::cout << "A и B имеют разные портреты\n";
            return *this;
         }
         this->al[j] += B.al[j];
         this->au[j] += B.au[j];
      }
   }
   return *this;
}

MyMatrix MyMatrix::operator* (const double a)
{
   MyMatrix C = *this;
   for (int i = 0; i < N; i++)
   {
      C.di[i] *= a;
      for (int j = ia[i]; j < ia[i + 1]; j++)
      {
         C.al[j] *= a;
         C.au[j] *= a;
      }
   }
   return C;
}

MyMatrix& MyMatrix::operator=(const MyMatrix& B)
{
   if (this != &B)
   {
      this->al = B.al;
      this->au = B.au;
      this->b = B.b;
      this->di = B.di;
      this->ia = B.ia;
      this->ja = B.ja;
      this->N = B.N;
   }
   return *this;
}
```

Generate.cpp

```
#include "Generate.h"

void Make_grid(std::string path)
{
   std::ofstream out;
   out.precision(15);
   std::vector<double> all_R, all_Z;
   std::ifstream in(path + "grid.txt");
   double R, Z, kr, kz;
   int Nr, Nz;
   int count_r, count_z;
   in >> count_r >> count_z;
   all_R.resize(count_r);
   all_Z.resize(count_z);
   in >> all_R[0] >> all_Z[0];
   for (int curr_count_r = 0; curr_count_r < count_r - 1; )
   {
      in >> R >> Nr >> kr;
      double hx;
      if (kr == 1)
      {
         hx = (R - all_R[curr_count_r]) / Nr;
         for (int p = 1; p < Nr; p++)
         {
            all_R[curr_count_r + p] = all_R[curr_count_r] + hx * p;
         }
         curr_count_r += Nr;
      }
      else
      {
         hx = (R - all_R[curr_count_r]) * (kr - 1) / (pow(kr, Nr) - 1);
         for (int p = 0; p < Nr - 1; curr_count_r++, p++)
         {
            all_R[curr_count_r + 1] = all_R[curr_count_r] + hx * pow(kr, p);
         }
         curr_count_r++;
      }
      all_R[curr_count_r] = R;
   }
   for (int curr_count_z = 0; curr_count_z < count_z - 1; )
   {
      in >> Z >> Nz >> kz;
      double hy;
      if (kz == 1)
      {
         hy = (Z - all_Z[curr_count_z]) / Nz;
         for (int p = 1; p < Nz; p++)
         {
            all_Z[curr_count_z + p] = all_Z[curr_count_z] + hy * p;
         }
         curr_count_z += Nz;
      }
      else
      {
         hy = (Z - all_Z[curr_count_z]) * (kz - 1) / (pow(kz, Nz) - 1);
         for (int p = 0; p < Nz - 1; curr_count_z++, p++)
         {
            all_Z[curr_count_z + 1] = all_Z[curr_count_z] + hy * pow(kz, p);
         }
         curr_count_z++;
      }
      all_Z[curr_count_z] = Z;
   }
   in.close();
   out.open("rz.txt");
   for (int i = 0; i < count_z; i++)
```

```
    {
        for (int j = 0; j < count_r; j++)
        {
            out << all_R[j] << "\t" << all_Z[i] << "\n";
        }
    }
    out.close();
    // input area
    // lambda, sigma same in all area
    out.open("elem.txt");
    for (int i = 0; i < count_z - 1; i++)
    {
        for (int j = 0; j < count_r - 1; j++)
        {
            out << i * count_r + j << " " << i * count_r + j + 1 << " "
                << (i + 1) * count_r + j << " " << (i + 1) * count_r + j + 1 << " 0 0 \n";
        }
    }
    out.close();
    // bounder
    out.open("S1.txt");
    out << 2 * count_z + 2 * count_r - 4 << " 0\n";
    for (int j = 0; j < count_r - 1; j++)
    {
        out << j << " ";
    }
    for (int i = 1; i < count_z - 1; i++)
    {
        out << i * count_r - 1 << " " << i * count_r << " ";
    }
    for (int j = -1; j < count_r; j++)
    {
        out << (count_z - 1) * count_r + j << " ";
    }
    out.close();
}

void Create_time_grid()
{
    std::ofstream out;
    out.precision(15);
    std::ifstream in;
    // time grid
    in.open("time_grid.txt");
    std::vector<double> time_grid;
    double T, kt;
    int Nt;
    int count_t;
    in >> count_t;
    time_grid.resize(count_t);
    in >> time_grid[0];
    for (int curr_count_t = 0; curr_count_t < count_t - 1; )
    {
        in >> T >> Nt >> kt;
        double ht;
        if (kt == 1)
        {
            ht = (T - time_grid[curr_count_t]) / Nt;
            for (int p = 1; p < Nt; p++)
            {
                time_grid[curr_count_t + p] = time_grid[curr_count_t] + ht * p;
            }
            curr_count_t += Nt;
        }
        else
        {
            ht = (T - time_grid[curr_count_t]) * (kt - 1) / (pow(kt, Nt) - 1);
            double pow_kt = 1;
```

```
            for (int p = 0; p < Nt - 1; curr_count_t++, p++)
            {
                time_grid[curr_count_t + 1] = time_grid[curr_count_t] + ht * pow_kt;
                pow_kt *= kt;
            }
            curr_count_t++;
        }
        time_grid[curr_count_t] = T;
    }
    in.close();
    out.open("time.txt");
    out << time_grid.size() << "\n";
    for (int i = 0; i < count_t; i++)
    {
        out << time_grid[i] << " ";
    }
    out.close();
}
```