МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

НГТУ  НЭТИ

Кафедра прикладной математики

Курсовой проект

по дисциплине «Уравнения математической физики»

РЕШЕНИЕ ПАРАБОЛИЧЕСКОЙ КРАЕВОЙ ЗАДАЧИ МКЭ

Группа ПМ-83

Студент ЛЕОНОВИЧ ДАРЬЯНА

Преподаватель ПЕРСОВА М.Г.

Новосибирск

2021

1. **Постановка задачи**

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

2. **Дискретизация по времени**

   Выпишем задачу. Параболическая краевая задача для функции u определяется дифференциальным уравнением:

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

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

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

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

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

   и начальным условием:

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

   Перепишем дифференциальное уравнение в цилиндрических координатах:

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

   Неявная схема аппроксимирует исходной уравнение по времени в следующем виде:

$$\sigma \frac{\partial u^j}{\partial t} - div\left(\lambda\, grad\, u^j\right) = f^j, j = 3, \dots, J$$

   где j – номер текущего временного слоя.

   Так как для аппроксимации по времени используем четырехслойную схему, слои, к которым можем применить схему, начинаются с третьего. Значение нулевого слоя задается из начального условия, значения на первом слое можно найти по двухслойной схеме, на втором – по трехслойной.

   В случае четырехслойной схемы аппроксимируем *u* по времени 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:

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

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

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

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

Тогда для аппроксимации производной *u* по времени в точке $t = t_j$ вычислим эти производные для базисных функций:

$$\left.\frac{d\eta_3^j(t)}{dt}\right|_{t=t_j} = \frac{d}{dt}\left(\frac{(t-t_{j-2})(t-t_{j-1})(t-t_j)}{(t_{j-3}-t_{j-2})(t_{j-3}-t_{j-1})(t_{j-3}-t_j)}\right)_{t=t_j}$$

$$= \frac{d}{dt}\left(\frac{t_{j-2}t_jt - t_{j-2}t_{j-1}t_j - t_{j-2}t^2 + t_{j-2}t_{j-1}t - t_jt^2 + t_{j-1}t_jt + t^3 - t_{j-1}t^2}{(t_{j-3}-t_{j-2})(t_{j-3}-t_{j-1})(t_{j-3}-t_j)}\right)_{t=t_j}$$

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

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

$$\left.\frac{d\eta_2^j(t)}{dt}\right|_{t=t_j} = \frac{d}{dt}\left(\frac{(t-t_{j-3})(t-t_{j-1})(t-t_j)}{(t_{j-2}-t_{j-3})(t_{j-2}-t_{j-1})(t_{j-2}-t_j)}\right)_{t=t_j}$$

$$= \frac{d}{dt}\left(\frac{t_{j-3}t_jt - t_{j-3}t_{j-1}t_j - t_{j-3}t^2 + t_{j-3}t_{j-1}t - t_jt^2 + t_{j-1}t_jt + t^3 - t_{j-1}t^2}{(t_{j-2}-t_{j-3})(t_{j-2}-t_{j-1})(t_{j-2}-t_j)}\right)_{t=t_j}$$

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

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

$$\left.\frac{d\eta_1^j(t)}{dt}\right|_{t=t_j} = \frac{d}{dt}\left(\frac{(t-t_{j-3})(t-t_{j-2})(t-t_j)}{(t_{j-1}-t_{j-3})(t_{j-1}-t_{j-2})(t_{j-1}-t_j)}\right)_{t=t_j}$$

$$= \frac{d}{dt}\left(\frac{t_{j-3}t_j t - t_{j-3}t_{j-2}t_j - t_{j-3}t^2 + t_{j-3}t_{j-2}t - t_j t^2 + t_{j-2}t_j t + t^3 - t_{j-2}t^2}{(t_{j-1}-t_{j-3})(t_{j-1}-t_{j-2})(t_{j-1}-t_j)}\right)_{t=t_j}$$

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

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

$$\left.\frac{d\eta_0^j(t)}{dt}\right|_{t=t_j} = \frac{d}{dt}\left(\frac{(t-t_{j-3})(t-t_{j-2})(t-t_{j-1})}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}\right)_{t=t_j}$$

$$= \frac{d}{dt}\left(\frac{t_{j-3}t_{j-1}t - t_{j-3}t_{j-2}t_{j-1} - t_{j-3}t^2 + t_{j-3}t_{j-2}t - t_{j-1}t^2 + t_{j-2}t_{j-1}t + t^3 - t_{j-2}t^2}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}\right)_{t=t_j}$$

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

$$= \frac{(t_j-t_{j-1})(t_j-t_{j-3}) + (t_j-t_{j-2})(t_j-t_{j-3}) + (t_j-t_{j-1})(t_j-t_{j-2})}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}$$

Обозначим

$$\Delta t_{01} = t_j - t_{j-1}$$
$$\Delta t_{02} = t_j - t_{j-2}$$
$$\Delta t_{03} = t_j - t_{j-3}$$
$$\Delta t_{12} = t_{j-1} - t_{j-2}$$
$$\Delta t_{13} = t_{j-1} - t_{j-3}$$
$$\Delta t_{23} = t_{j-2} - t_{j-3}$$

Тогда аппроксимация производной u по времени имеет вид:

$$\frac{\partial u(r,z,t)}{\partial t} \approx u^{j-3}(r,z)\left(-\frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}}\right) + u^{j-2}(r,z)\left(\frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{12}\Delta t_{23}}\right)$$
$$+ u^{j-1}(r,z)\left(-\frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}}\right)$$
$$+ u^{j}(r,z)\left(\frac{\Delta t_{01}\Delta t_{02} + \Delta t_{01}\Delta t_{03} + \Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}}\right)$$

Подставим в неявную схему:

$$\sigma u^{j-3}(r,z)\left(-\frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}}\right)+\sigma u^{j-2}(r,z)\left(\frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{12}\Delta t_{23}}\right)$$
$$+\sigma u^{j-1}(r,z)\left(-\frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}}\right)$$
$$+\sigma u^{j}(r,z)\left(\frac{\Delta t_{01}\Delta t_{02}+\Delta t_{01}\Delta t_{03}+\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}}\right)$$
$$-div\left(\lambda\, grad\, u^{j}\right)=f^{j}, j=3,\dots,J$$

Перенесем вправо известные значения *u* на предыдущих слоях:

$$\sigma u^{j}(r,z)\left(\frac{\Delta t_{01}\Delta t_{02}+\Delta t_{01}\Delta t_{03}+\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{02}\Delta t_{03}}\right)-div\left(\lambda\, grad\, u^{j}\right)$$
$$=f^{j}-\sigma u^{j-3}(r,z)\left(-\frac{\Delta t_{01}\Delta t_{02}}{\Delta t_{03}\Delta t_{13}\Delta t_{23}}\right)-\sigma u^{j-2}(r,z)\left(\frac{\Delta t_{01}\Delta t_{03}}{\Delta t_{02}\Delta t_{12}\Delta t_{23}}\right)$$
$$-\sigma u^{j-1}(r,z)\left(-\frac{\Delta t_{02}\Delta t_{03}}{\Delta t_{01}\Delta t_{12}\Delta t_{13}}\right), j=3,\dots,J$$

3. Вариационная постановка

Запишем эквивалентную исходной задаче на j-ом временном слое подстановку Галеркина.

Для этого обозначим невязку исходного уравнения

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

Так как значения на предыдущих слоях и величина шага по временной сетке известны, обозначим

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

Тогда можем записать:

$$R(u)=-div\left(\lambda\, grad\, u^{j}\right)+\gamma u^{j}-f.$$

Потребуем, чтобы невязка была ортогональна пространству пробных функций Ф, т.е.

$$\int_{\Omega}\left(-div\left(\lambda\, grad\ u^{j}\right)+\gamma\, u^{j}-f\right)v\, d\Omega=0,\qquad \forall v\epsilon\Phi.$$

Преобразуем интеграл с использованием формулы Грина, распишем интеграл по границе с учетом краевых условий и, для исключения из

суммы интеграла по S_1, потребуем, чтобы $\Phi = H_0^1$, т.е. чтобы пробные функции были из пространства функций, имеющих суммируемые с квадратом производные, и равных нулю на границе S_1. Решение задачи u будет принадлежать пространству H_g^1. Перепишем получившееся выражение:

$$\int_\Omega \lambda\, grad\ u^j\ grad\ v_0\ d\Omega + \int_\Omega \gamma\, u^j v_0 d\Omega + \int_{S_3} \beta\, u^j v_0 dS =$$

$$= \int_\Omega f v_0 d\Omega + \int_{S_2} \theta v_0 dS + \int_{S_3} \beta u_\beta v_0 dS, \qquad \forall v_0 \epsilon \mathrm{H}_0^1.$$

4. Конечноэлементная дискретизация и переход к локальным матрицам

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

$$\int_\Omega \lambda\, grad\, u^h grad\, v_0^h\ d\Omega + \int_\Omega \gamma u^h v_0^h d\Omega + \int_{S_3} \beta u^h v_0^h dS =$$

$$= \int_\Omega f v_0^h d\Omega + \int_{S_2} \theta v_0^h dS + \int_{S_3} \beta u_\beta v_0^h dS, \qquad \forall v_0^h \epsilon \mathrm{V}_0^h.$$

Пусть $\{\psi_i\}$ – базис V^h (пространство, аппроксимирующее H^1). Тогда $v_0^h \in \mathrm{V}_0^h$ может быть представлено в виде:

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

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

где N_0 – множество индексов i таких, что ψ_i являются базисными функциями пространств V_0^h, V_g^h.

И уравнение Галеркина эквивалентно следующей СЛАУ для компонент вектора q с индексами $j \epsilon N_0$:

$$\sum_{j=1}^{n} \left(\int_\Omega \lambda\, grad\, \psi_j grad\, \psi_i\ d\Omega + \int_\Omega \gamma \psi_j \psi_i\ d\Omega + \int_{S_3} \beta \psi_j \psi_i\ dS \right) q_i =$$

$$= \int_\Omega f \psi_i\ d\Omega + \int_{S_2} \theta \psi_i\ dS + \int_{S_3} \beta u_\beta \psi_i dS, \qquad i \epsilon N_0.$$

Недостающие n-n_0 уравнений для компонент вектора q с индексами $i \notin N_0$ могут быть получены из условия $u|_{S_1} = u_g$:

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

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 на $[r_p, r_{p+1}]$, $[z_s, z_{s+1}]$:

$$R_1(r) = \frac{r_{p+1} - r}{h_r}, R_2(r) = \frac{r - r_p}{h_r}, h_r = r_{p+1} - r_p$$

$$Z_1(z) = \frac{z_{s+1} - z}{h_z}, Z_2(z) = \frac{z - z_s}{h_z}, h_z = z_{s+1} - z_s$$

И локальные базисные функции:

$$\hat{\psi}_1(r,z) = R_1(r)Z_1(z), \qquad \hat{\psi}_2(r,z) = R_2(r)Z_1(z),$$
$$\hat{\psi}_3(r,z) = R_1(r)Z_2(z), \qquad \hat{\psi}_4(r,z) = R_2(r)Z_2(z).$$

Компоненты локальных матриц жесткости и массы имеют вид:

$$\hat{G}_{ij} = \int_{\Omega_k} \bar{\lambda}\left(\frac{\partial\hat{\psi}_i}{\partial r}\frac{\partial\hat{\psi}_j}{\partial r} + \frac{\partial\hat{\psi}_j}{\partial r}\frac{\partial\hat{\psi}_j}{\partial r}\right) r dr dz\,, \qquad \hat{M}_{ij} = \int_{\Omega_k} \bar{\gamma}\hat{\psi}_i\hat{\psi}_j r dr dz$$

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

Таким образом $-div(\lambda\ grad\ u^j)$ аппроксимируется вектором G*q^j, σu^j аппроксимируется с помощью M*q^j.

Тогда вид уравнения на j-ом временном слое после конечноэлементной аппроксимации:

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

5. Аналитические выражения для вычисления локальных матриц

Запишем аналитические выражения для вычисления элементов локальных матриц. Для этого сперва вычислим вспомогательные интегралы:

$$\int_{z_s}^{z_s+h_z}\left(\frac{dZ_1}{dz}\right)^2 dz=\frac{1}{h_z},\qquad \int_{z_s}^{z_s+h_z}\left(\frac{dZ_2}{dz}\right)^2 dz=\frac{1}{h_z}$$

$$\int_{z_s}^{z_s+h_z}\frac{dZ_1}{dz}\frac{dZ_2}{dz}dz=-\frac{1}{h_z},\qquad \int_{z_s}^{z_s+h_z} Z_1Z_2dz=\frac{h_z}{6}$$

$$\int_{z_s}^{z_s+h_z}(Z_1)^2dz=\frac{h_z}{3},\qquad \int_{z_s}^{z_s+h_z}(Z_2)^2dz=\frac{h_z}{3}$$

$$\int_{r_p}^{r_p+h_r}\left(\frac{dR_1}{dr}\right)^2 rdr=\left(\frac{r_p}{h_r}+\frac{1}{2}\right),\qquad \int_{r_p}^{r_p+h_r}\left(\frac{dR_2}{dr}\right)^2 rdr=\left(\frac{r_p}{h_r}+\frac{1}{2}\right)$$

$$\int_{r_p}^{r_p+h_r}\frac{dR_1}{dr}\frac{dR_2}{dr}rdr=\left(-\frac{r_p}{h_r}-\frac{1}{2}\right),\qquad \int_{r_p}^{r_p+h_r} R_1R_2rdr=\left(\frac{h_rr_p}{6}+\frac{{h_r}^2}{12}\right)$$

$$\int_{r_p}^{r_p+h_r}(R_1)^2rdr=\left(\frac{h_rr_p}{3}+\frac{{h_r}^2}{12}\right),\qquad \int_{r_p}^{r_p+h_r}(R_2)^2rdr=\left(\frac{h_rr_p}{3}+\frac{{h_r}^2}{4}\right)$$

Теперь вычислим компоненты матрицы жесткости.

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

$$=\bar{\lambda}\int_{r_p}^{r_p+h_r}\int_{z_s}^{z_s+h_z}\left(\left(\frac{dR_1}{dr}\right)^2Z_1^2+R_1^2\left(\frac{dZ_1}{dz}\right)^2\right)rdrdz=$$

$$=\bar{\lambda}\int_{r_p}^{r_p+h_r}\left(\frac{dR_1}{dr}\right)^2 rdr\int_{z_s}^{z_s+h_z}Z_1^2\,dz+\int_{r_p}^{r_p+h_r}R_1^2rdr\int_{z_s}^{z_s+h_z}\left(\frac{dZ_1}{dz}\right)^2 dz=$$

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

Аналогично вычислим остальные компоненты матрицы (матрица симметрична, поэтому вычислим только нижний треугольник и диагональные элементы):

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

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

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

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

$$\hat{G}_{14} = \hat{G}_{23} = \bar{\lambda} \int_{r_p}^{r_p+h_r} \frac{dR_1}{dr}\frac{dR_2}{dr} r dr \int_{z_s}^{z_s+h_z} Z_1 Z_2\, dz + \int_{r_p}^{r_p+h_r} R_1 R_2 r dr \int_{z_s}^{z_s+h_z} \frac{dZ_1}{dz}\frac{dZ_2}{dz} dz =$$

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

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

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

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

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

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

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

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

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

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

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

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

Запишем элементы матрицы массы.

$$\hat{M}_{ij} = \int_{\Omega_k} \bar{\gamma}\hat{\psi}_i\hat{\psi}_j rdrdz = \sum_{k=1}^{n} \gamma_k \int_{\Omega_k} \hat{\psi}_k\hat{\psi}_i\hat{\psi}_j rdrdz$$

Вычислим вспомогательные интегралы:

$$\int_{z_s}^{z_s+h_z} Z_1^2 Z_2 dz = \frac{h_z}{12} \quad \int_{z_s}^{z_s+h_z} Z_1 Z_2^2 dz = \frac{h_z}{12}$$

$$\int_{z_s}^{z_s+h_z} (Z_1)^3 dz = \frac{h_z}{4}, \quad \int_{z_s}^{z_s+h_z} (Z_2)^3 dz = \frac{h_z}{4}$$

$$\int_{r_p}^{r_p+h_r} R_1^2 R_2 rdr = h_r\left(\frac{r_p}{12} + \frac{h_r}{30}\right) \quad \int_{r_p}^{r_p+h_r} R_1 R_2^2 rdr = h_r\left(\frac{r_p}{12} + \frac{h_r}{20}\right)$$

$$\int_{r_p}^{r_p+h_r} (R_1)^3 rdr = h_r\left(\frac{r_p}{4} + \frac{h_r}{20}\right), \quad \int_{r_p}^{r_p+h_r} (R_2)^3 rdr = h_r\left(\frac{r_p}{4} + \frac{h_r}{5}\right)$$

$$\widehat{M}_{11} = \int\limits_{\Omega_k} \bar{\sigma}\hat{\psi}_1\hat{\psi}_1 r dr dz$$

$$= \sigma(r_p, z_s) \int\limits_{\Omega_k} \hat{\psi}_1\hat{\psi}_1\hat{\psi}_1 r dr dz + \sigma(r_{p+1}, z_s) \int\limits_{\Omega_k} \hat{\psi}_2\hat{\psi}_1\hat{\psi}_1 r dr dz$$

$$+ \sigma(r_p, z_{s+1}) \int\limits_{\Omega_k} \hat{\psi}_3\hat{\psi}_1\hat{\psi}_1 r dr dz + \sigma(r_{p+1}, z_{s+1}) \int\limits_{\Omega_k} \hat{\psi}_4\hat{\psi}_1\hat{\psi}_1 r dr dz$$

$$= \sigma(r_p, z_s) \int\limits_{r_p}^{r_p+h_r} R_1^3 r dr \int\limits_{z_s}^{z_s+h_z} Z_1^3\, dz + \sigma(r_{p+1}, z_s) \int\limits_{r_p}^{r_p+h_r} R_2 R_1^2 r dr \int\limits_{z_s}^{z_s+h_z} Z_1^3\, dz$$

$$+ \sigma(r_p, z_{s+1}) \int\limits_{r_p}^{r_p+h_r} R_1^3 r dr \int\limits_{z_s}^{z_s+h_z} Z_2 Z_1^2\, dz$$

$$+ \sigma(r_{p+1}, z_{s+1}) \int\limits_{r_p}^{r_p+h_r} R_2 R_1^2 r dr \int\limits_{z_s}^{z_s+h_z} Z_2 Z_1^2\, dz$$

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

Аналогично найдем остальные элементы (с учетом симметричности матрицы)

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

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

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

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

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

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

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

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

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

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 $\gamma \equiv 1$

Вклады в глобальную матрицу и вектор правой части от краевых условий второго и третьего рода так же выпишем в виде локальных матриц и векторов, считая элементом ребро, где задано граничное условие. В случае прямоугольной сетки эти ребра ориентированы вдоль одной из осей координат.

Для учета краевого условия второго рода вдоль оси z локальный вектор $\hat{b}^{S_{2z}}$ будет иметь вид:

$$\hat{b}^{S_{2z}} = \frac{h_z}{6}\begin{pmatrix} 2\Theta(r, z_1) + \Theta(r, z_2) \\ \Theta(r, z_1) + 2\Theta(r, z_2) \end{pmatrix}$$

Для учета краевого условия второго рода вдоль оси r локальный вектор $\hat{b}^{S_{2r}}$ будет иметь вид:

$$\hat{b}^{S_{2r}} = \frac{h_r}{6}\begin{pmatrix} \left(2r_1 + \frac{h_r}{2}\right)\Theta(r_1, z) + \left(r_1 + \frac{h_r}{2}\right)\Theta(r_2, z) \\ \left(r_1 + \frac{h_r}{2}\right)\Theta(r_1, z) + \left(2r_1 + \frac{3h_r}{2}\right)\Theta(r_2, z) \end{pmatrix}$$

Для учета краевого условия третьего рода вдоль оси z локальный вектор $\hat{b}^{S_{3z}}$ будет иметь вид:

$$\hat{b}^{S_{3z}} = \beta \frac{h_z}{6} \begin{pmatrix} 2u_\beta(r, z_1) + u_\beta(r, z_2) \\ u_\beta(r, z_1) + 2u_\beta(r, z_2) \end{pmatrix}$$

а локальная матрица $\widehat{M}^{S_{3z}}$:

$$\widehat{M}^{S_{3z}} = \sigma \frac{h_z}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Для учета краевого условия второго рода вдоль оси r локальный вектор $\hat{b}^{S_{3r}}$ и локальная матрица $\widehat{M}^{S_{3z}}$ будут иметь вид:

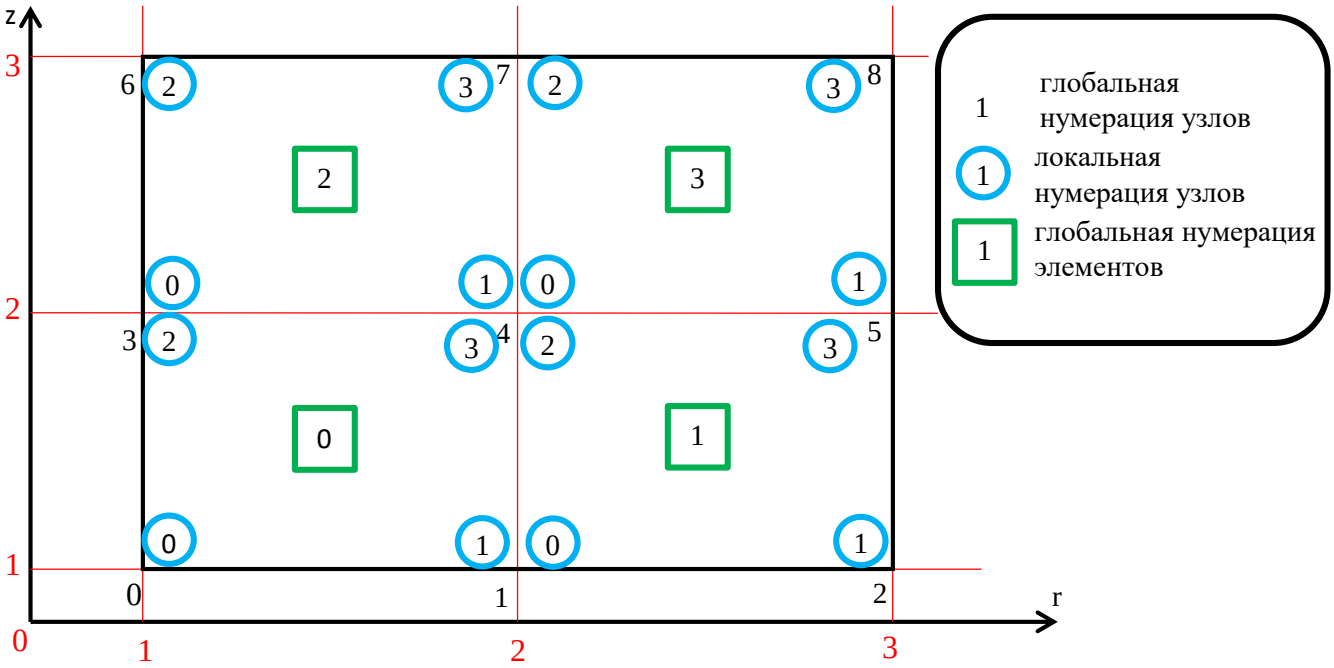
$$\hat{b}^{S_{3r}} = \beta \frac{h_r}{6} \begin{pmatrix} \left(2r_1 + \frac{h_r}{2}\right) u_\beta(r_1, z) + \left(r_1 + \frac{h_r}{2}\right) u_\beta(r_2, z) \\ \left(r_1 + \frac{h_r}{2}\right) u_\beta(r_1, z) + \left(2r_1 + \frac{3h_r}{2}\right) u_\beta(r_2, z) \end{pmatrix}$$

$$\widehat{M}^{S_{2z}} = \gamma \frac{h_z}{6} \begin{pmatrix} \left(2r_1 + \frac{h_r}{2}\right) & \left(r_1 + \frac{h_r}{2}\right) \\ \left(r_1 + \frac{h_r}{2}\right) & \left(2r_1 + \frac{3h_r}{2}\right) \end{pmatrix}$$

Для учета краевого условия первого рода заменим соответствующую узлу строку в матрице A на нулевую с единицей на диагонали, в векторе b заменим соответствующую компоненту на значение u_g

6. Структуры данных для задания расчетной области и конечноэлементной сетки

Рисунок расчетной области:



Сетка задается в следующем виде:

Имя файла	Описание	Пример
info.txt	Содержит четыре числа: кол-во узлов сетки, кол-во материалов в расчетной области, кол-во конечных элементов сетки, кол-во границ с краевыми условиями первого рода	9 1 4 1
rz.txt	Содержит координаты всех узлов, сначала по r, потом по z. В программе создана структура node, содержащий значения r и z. Вектор node описывает все узлы сетки. Глобальный номер узла соответствует индексу узла в массиве.	1 1 2 1 3 1 1 2 2 2 3 2 1 3 2 3 3 3
S1.txt	Содержит записи вида: число узлов, где задано граничное условие, номер-идентификатор функции на границе, перечисление номеров узлов в глобальной нумерации.	8 0 0 1 2 3 5 6 7 8

	Кол-во таких записей задается в info.txt. В программе хранится S1, вектор пар из идентификатора условия и вектора узлов. Можно задать несколько разных граничных условий: функции u_g задаются в программе	
material.txt	Содержит описание материала: значение $\bar{\lambda}$ для материала и индекс функции γ (сами функции внесены в программу). В программе материалы хранятся в специальной структуре material, где хранится значение $\bar{\lambda}$ и индекс функции γ	1 0
elem.txt	Содержит список глобальных номеров в порядке локальной нумерации, индекс материала, индекс правой части f дифференциального уравнения (f задаются в программе). Данные хранятся в специальной структуре element, вектор element описывает все элементы сетки. Глобальный номер элемента соответствует индексу элемента в массиве.	0 1 3 4 0 0 1 2 4 5 0 0 3 4 6 7 0 0 4 5 7 8 0 0
time.txt	Содержит количество узлов по сетке времени и сами значения этих узлов. В программе сетка времени хранится в векторе time	8 0 1 2 3 4 5 6 7
q1 q2 q3.txt	Содержит заданные на первых трех слоях значения весов. Эти значения хранятся в программе в векторах q1, q2, q3 и очередной вектор q4 на текущем слое ищем в программе, потом заменяем вектора перед переходом на новый слой времени.	1.5 1.5 1.5 3 3 3 4.5 4.5 4.5 3.5 3.5 3.5 5 5 5 6.5 6.5 6.5 5.5 5.5 5.5 7 7 7 8.5 8.5 8.5
S2_r.txt	Содержит записи вида: число границ с краевыми второго рода, параллельных оси r, число узлов, где задано граничное условие, номер-идентификатор функции на границе, перечисление номеров	1 3 1 6 7 8 (краевое условие второго рода по верхней границе)

	узлов в глобальной нумерации. В программе хранится аналогично S1	
S2_z.txt	Содержит записи вида: число границ с краевыми второго рода, параллельных оси z, число узлов, где задано граничное условие, номер-идентификатор функции на границе, перечисление номеров узлов в глобальной нумерации. В программе хранится аналогично S1	2 3 2 0 3 6 3 3 2 5 8 (разные краевые условия второго рода по боковым границе)
S3_r.txt	Содержит записи вида: число границ с краевыми третьего рода, параллельных оси r, число узлов, где задано граничное условие, номер-идентификатор функции на границе, перечисление номеров узлов в глобальной нумерации. В программе хранится аналогично S1 β задается в программе	1 3 4 6 7 8 (краевое условие третьего рода по верхней границе)
S3_z.txt	Содержит записи вида: число границ с краевыми третьего рода, параллельных оси z, число узлов, где задано граничное условие, номер-идентификатор функции на границе, перечисление номеров узлов в глобальной нумерации. В программе хранится аналогично S1 β задается в программе	2 3 5 0 3 6 3 6 2 5 8 (разные краевые условия третьего рода по боковым границе)

7. Структура основных модулей программы

main.cpp – содержит все необходимые для построения СЛАУ функции, а так же глобальные переменные, описывающие сетку.

```
std::vector<node> all_nodes;          // все узлы в порядке глобальной нумерации
std::vector<element> all_elems;       // все элементы в прядке глобальной нумерации
std::vector<material> all_materials; // все материалы по индексам
std::vector<std::pair<int,std::vector<int>>> S1; // на j-ом узле заданы краевые 1 рода
std::vector<std::pair<int,std::vector<int>>> S2_r;// параллельно r на jом узле краевые 2 рода
std::vector<std::pair<int,std::vector<int>>> S2_z;// параллельно z на jом узле краевые 2 рода
std::vector<std::pair<int,std::vector<int>>> S3_r;// параллельно r на jом узле краевые 3 рода
std::vector<std::pair<int,std::vector<int>>> S3_z;// параллельно z на jом узле краевые 3 рода
std::vector<double> time_grid; // сетка времени
int i_t = 0; // текущий временной слой
MyVector q1, q2, q3, q4; // q4 -текущий временной слой, q3, q2, q1 –веса на предыдущих слоях
double gamma(double r, double z, int gam_id); // значение гамма по индексу gam_id
```

```
double beta(double r, double z, int beta_id); // значение beta по индексу beta_id
double func_f(double r, double z, int f_id) // значение f по индексу f_id
double func_S(double r, double z, int s_id) // значение краевого S по индексу s_id
int Input() // чтение данных
double GetG_Loc(double rp, double lambda, double hr, double hz,
   std::vector<std::vector<double>> &G_loc) // получение локальной G
double GetM_Loc(double rp, double zs, int gam, double hr, double hz,
   std::vector<std::vector<double>>& M_loc) // получение локальной M
int Get_Loc(std::vector<std::vector<double>>& M_loc, std::vector<std::vector<double>>& G_loc,
    int el_id) // получение локальных матриц
int Getb_Loc(double rp, double zs, double hr, double hz,
   std::vector<double> &b_loc, int f_id) // получение локального b
int Get_Loc_b(std::vector<double>& b_loc,
    int el_id) // получить вектор правой части
int GeneratePortrait(MyMatrix &A, int N, int Kel) // генерация портрета
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) // внесение локальных A, b  в глобальную СЛАУ
int AddLocal_b(std::vector<double>& b, std::vector<double>& b_loc, int el_id)
    // внесение локальных b  в глобальную СЛАУ
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)
   // учет первых краевых
double GetM_Loc_dim2_r(double rp, double hr,
    std::vector<std::vector<double>>& M_loc) // локальная матрица для S3_r
int Getb_Loc_dim2_r(double rp, double zs, double hr,
    std::vector<double>& b_loc, int f_id) // получение локального b для S параллельных r
double GetM_Loc_dim2_z(double zp, double hz,
    std::vector<std::vector<double>>& M_loc) // локальная матрица для S3_z
int Getb_Loc_dim2_z(double rp, double zs, double hz,
    std::vector<double>& b_loc, int s_id) // получение локального b для S параллельных z
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) // внесение локальной матрицы в глобальную для одномерного случая
int Set_S2(MyMatrix &MS) // учет вторых краевых
int Set_S3(MyMatrix& MS, bool flag) // учет третьих краевых
```

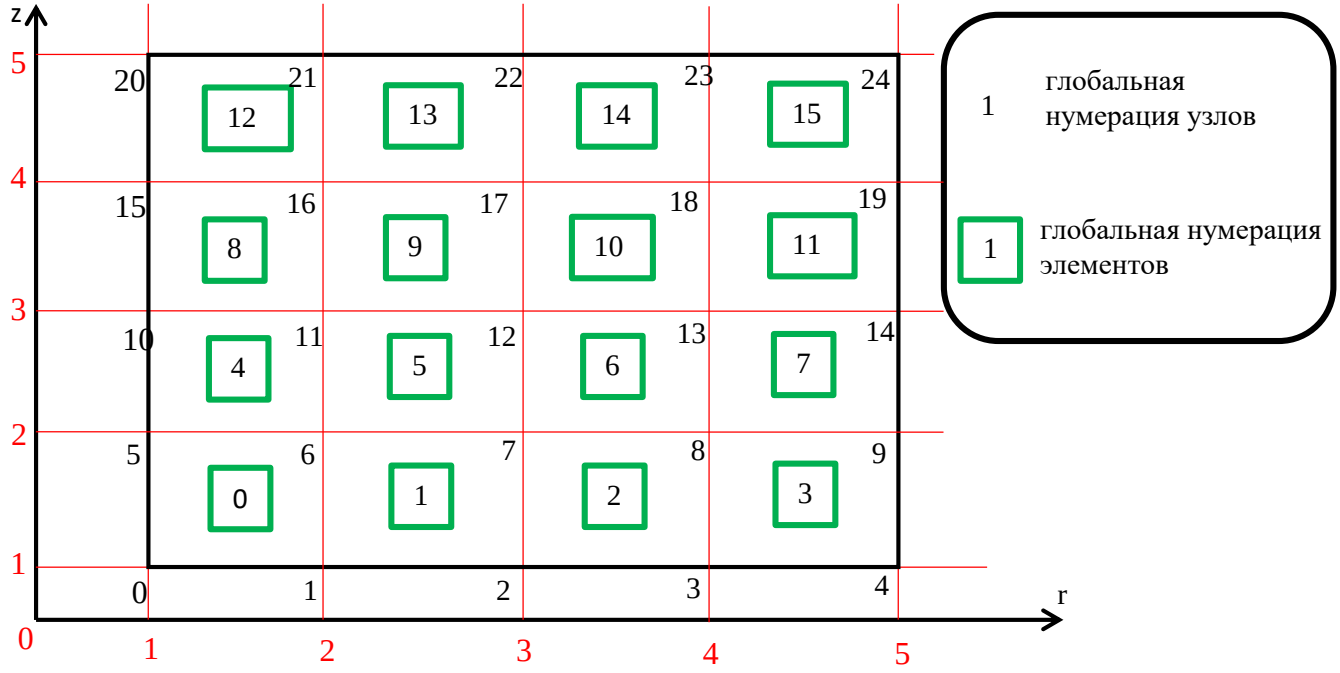
MyVector.cpp, MyVector.h, MyMatrix.cpp, MyMatrix.h –используются для облегчения работы с векторами и матрицами при решении СЛАУ.

Solver.cpp, Solver.h содержат функции для решения СЛАУ методами ЛОС и МСГ с LU-факторизацией.

Generate.h и Generate.cpp содержат функции для генерации файлов rz, elem, S1, time при задании координат начала и конца разбиения, а так же количества разбиений по каждой переменной.

8. Исследования на порядок аппроксимации

Расчетная область и сетка: (лямбда = сигма = 1; по всем границам краевые условия 1ого рода)



Сетка по времени:

time	0	0,2	0,4	0,6	0,8	1	1,2	1,4	1,6	1,8

Для нахождения порядка аппроксимации будем увеличивать степень полинома по пространству и по времени и сравнивать среднеквадратичную погрешность на временных слоях, начиная со слоя t = 0,6, где начинается применение четырехслойной неявной схемы. Чтобы исследовать только четырехслойную схему, значения на предыдущих слоях зададим из входных данных.

	среднеквадратичная погрешность						
время:	0,6	0,8	1	1,2	1,4	1,6	1,8
$u = 5$	0	7,82E-16	6,77E-16	6,77E-16	6,77E-16	6,77E-16	3,91E-16
$u = 5t$	0	5,65E-16	5,53E-16	7,82E-16	9,57E-16	1,27E-15	1,16E-15
$u = 5t^2$	4,44E-17	0	5,53E-16	8,35E-16	9,82E-16	0	0
$u = 5t^3$	4,44E-17	8,88E-17	3,91E-16	9,82E-16	7,11E-16	7,11E-16	0
$u = 5t^4$	0,003422	0,008198	0,012606	0,01608	0,018676	0,020615	0,022091
$u = r + 5t$	0,000253	0,000562	0,000804	0,000967	0,001076	0,001155	0,001216

$u = r + 5t^2$	0,000253	0,000562	0,000804	0,000967	0,001076	0,001155	0,001216
$u = r + 5t^3$	0,000253	0,000562	0,000804	0,000967	0,001076	0,001155	0,001216
$u = r + 5t^4$	0,003244	0,007782	0,011978	0,015288	0,017763	0,019613	0,021021
$u = z + 5t$	3,91E-16	8E-16	1,34E-15	9,87E-16	9,53E-16	4,26E-16	0
$u = z + 5t^2$	6,77E-16	9,57E-16	1,29E-15	9,24E-16	3,55E-16	0	3,98E-15
$u = z + 5t^3$	6,77E-16	9,33E-16	1,26E-15	3,55E-16	4,49E-16	3,9E-15	4,26E-15
$u = z + 5t^4$	0,003422	0,008198	0,012606	0,01608	0,018676	0,020615	0,022091
$u = r^2 + 5t$	3,91E-16	4E-15	4,07E-15	5,67E-15	5,63E-15	3,98E-15	6,89E-15
$u = r^2 + 5t^2$	3,91E-16	5,64E-15	4,05E-15	5,63E-15	4,15E-15	0	0
$u = r^2 + 5t^3$	3,91E-16	9,36E-16	8,9E-16	5,65E-15	5,64E-15	7,05E-15	5,83E-15
$u = r^2 + 5t^4$	0,003422	0,008198	0,012606	0,01608	0,018676	0,020615	0,022091
$u = z^2 + 5t$	3,91E-16	8,74E-16	5,7E-15	5,64E-15	3,98E-15	3,98E-15	3,98E-15
$u = z^2 + 5t^2$	5,53E-16	4,07E-15	4,08E-15	3,98E-15	4,15E-15	3,98E-15	3,98E-15
$u = z^2 + 5t^3$	6,77E-16	4,16E-15	5,68E-15	6,9E-15	5,64E-15	7,13E-15	9,16E-15
$u = z^2 + 5t^4$	0,003422	0,008198	0,012606	0,01608	0,018676	0,020615	0,022091

Так как базис по пространству используется билинейный, увеличение степени полинома по пространственным координатам приведет к возникновению погрешности из-за аппроксимации по пространству, не давая оценить порядок аппроксимации по времени.

Погрешность из-за аппроксимации по пространству появляется например при $u = r + 5t$. Это связано с особенностями цилиндрических координат – в правой части исходного уравнения появляется 1/r, являющееся неполиномиальной функцией. Однако даже на этой функции видно, что при степени t меньше 4 погрешность не меняется.

Для всех исследуемых функций погрешность от аппроксимации по времени появляется на полиноме 4-ой степени по t. Следовательно, порядок аппроксимации четырехслойной неявной схемы равен трем.

9. Исследования на порядок сходимости

Возьмем в качестве базовой сетки h такую же сетку, как при исследовании на порядок аппроксимации.

Сетку h/2 и h/4 получим, уменьшив шаг по переменным в 2 и 4 раза соответственно. Среднеквадратичную погрешность будем искать как:

$$\frac{\sqrt{\sum_{i=1}^{n}\left(u_i{}^{*}-u_i^{h}\right)^2}}{n}$$

Для определения порядка сходимости найдем значение отношения погрешностей

$$\frac{n_2\sqrt{\sum_{i=1}^{n_1}(u_i{}^{*}-u_i^{h})^2}}{n_1\sqrt{\sum_{i=1}^{n_2}(u_i{}^{*}-u_i^{h/2})^2}}\sim 2^k, \qquad k-\text{порядок сходимости}$$

1. *Сходимость по пространству*

   Возьмем *u = cos(r²+3z) - sin(t)* и будем уменьшать шаг по пространству для определения порядка сходимости билинейного базиса.

	среднеквадратичная погрешность							среднее значение погрешности на сетке
время:	0,6	0,8	1	1,2	1,4	1,6	1,8	
h	3,26414	6,036709	7,055384	7,115539	7,046677	7,094607	7,199376	6,401776
h/2	1,276438	1,538669	1,400285	1,37031	1,3992	1,408861	1,404523	1,399755
h/4	0,405573	0,486394	0,475759	0,475629	0,476899	0,476642	0,476792	0,46767
h/8	0,117256	0,144356	0,141532	0,141265	0,141605	0,141458	0,141449	0,138417

Отношение погрешностей:

	среднеквадратичная погрешность							среднее	k
время:	0,6	0,8	1	1,2	1,4	1,6	1,8		
h к h/2	2,557225	3,923332	5,038533	5,19265	5,036217	5,035705	5,12585	4,573497	2
h/2 к h/4	3,147247	3,163421	2,943266	2,881048	2,933957	2,955807	2,945778	2,993043	1
h/4 к h/8	3,458871	3,3694	3,361493	3,366933	3,367804	3,369497	3,370768	3,378693	1

Так как отношение погрешностей при первом и втором дроблении не стабильно, для нахождения порядка сходимости раздробим сетку еще раз, пока отношение погрешностей не станет стабильным.

На основе полученных в ходе исследования результатов можно увидеть, что порядок сходимости для билинейного базиса примерно равен одному, что не ниже теоретически ожидаемого.

2. *Сходимость по времени*

   Возьмем все ту же базовую сетку, но теперь шаг будем уменьшать по времени и возьмем $u = t^4$, не зависящую от r, z, чтобы убрать влияние погрешности от аппроксимации по пространству.

   По времени возьмем базовую сетку ht:

time:	0	1	2	3	4	5	6	7

   Результаты работы программы:

	среднее значение погрешности на сетке	ht к ht/2		ht/2 к ht/4	
		2^k	k	2^k	k
ht	14,40528	7,786479	2,960971		
ht/2	1,850037			7,908553	2,983414
ht/4	0,233929				

   Полученные значения подтверждают теоретическое предположение, что четырехслойная неявная схема имеет третий порядок сходимости.

3. *Сходимость по времени (если и зависит от пространственных координат)*

   Не меняя вид расчетной области, возьмем сетку с шагом по пространству h/2 = 0,5. По времени возьмем базовую сетку ht:

time:	0	1	2	3	4	5	6	7

   Возьмем $u = cos(r^2+3z) – t^4sin(t)$ и будем уменьшать шаг по времени для определения порядка сходимости по времени.

	среднее значение погрешности на сетке	ht к ht/2		ht/2 к ht/4		ht/4 к ht/8	
		2^k	k	2^k	k	2^k	k
ht	175,391	5,453558	2,447198				
ht/2	32,16083			6,219702	2,636846		
ht/4	5,170799					3,138544	1,650096
ht/8	1,647515						

   При дроблении сетки в 8 раз наблюдается снижение порядка сходимости в связи с ростом вычислительной погрешности.

   Полученный при дроблении порядок сходимости ниже теоретически ожидаемого третьего, так как из-за линейного базиса по r, z возникает погрешность от аппроксимации по пространству.

4. *Сходимость по времени и пространству*

   Возьмем расчетную область и базовую сетку h по пространству и времени как в предыдущем пункте, но теперь помимо дробления сетки времени будем уменьшать и шаг по пространству.

Возьмем $u = cos(r^2+3z) - t^4 sin(t)$ и будем уменьшать шаг по времени и пространству.

	среднее значение погрешности на сетке	h к h/2		h/2 к h/4	
		2^k	k	2^k	k
h	175,391	5,570908	2,477912		
h/2	31,48337			6,911923	2,789087
h/4	4,554937				

Наблюдается небольшое уменьшение погрешности по сравнению с вариантом, когда шаг по пространству не дробится. Однако в целом порядок сходимости не сильно отличается, в то время как размерность СЛАУ (а значит и вычислительные затраты) увеличилась в 2 и в 4 раза соответственно для сеток h/2 и h/4

10. Тестирование

1. *Краевые второго и третьего рода*

Возьмем сетку из п.9 и зададим сетку по времени

time:	0	1	2	3	4	5	6	7

Будем на верхней или правой границе расчетной области задавать краевое второго или третьего рода. Так как координаты цилиндрические, учет краевых 2 и 3 рода отличается для границ, параллельных оси z и оси r.

Пусть $u = r^2+3z-5t^3$

	среднеквадратичная погрешность					среднее значение погрешности
время:	3	4	5	6	7	
S1 везде	7,96E-15	9,15E-14	1,23E-13	0	5,66E-13	1,57639E-13
S2 сверху	3,48E-15	4,74E-15	4,81E-15	4,23E-15	7,13E-15	4,87871E-15
S2 справа	0,000413	0,000729	0,000899	0,000972	0,000998	0,000802188
S3 сверху	7,37E-16	9,94E-16	6,99E-16	4,26E-16	0	5,71245E-16
S3 справа	0,000413	0,000729	0,000899	0,000972	0,000998	0,000802188

Для краевых условий второго и третьего рода наблюдается погрешность, когда граница параллельно оси z. Это связано с тем, что берется производная по нормали в цилиндрических координатах и условие $\lambda \frac{\partial u}{\partial n}\Big|_{S_2} = \theta$ преобразуется в $\lambda r \frac{\partial u}{\partial r}\Big|_{S_2} = \theta$.

Проверим это предположение, понаблюдав за изменением погрешности при дроблении сетки по пространству для случая, когда краевые второго рода заданы на правой границе области.

	среднее значение погрешности на сетке	h к h/2		h/2 к h/4	
		2^k	k	2^k	k
h	0,000802188	3,997613	1,999139		
h/2	0,000201			3,99956	1,999841
h/4	5,02E-05				

Погрешность действительно уменьшается при дроблении сетки по пространству.

2. *Коэффициент сигма зависит от r, z*

Возьмем сетку из п.9 и зададим сетку по времени

time:	0	1	2	3	4	5	6	7

Пусть $u = r^2+3z-2,5t^2$ и $сигма = 2r^2 + z / 2$

						среднее значение погрешности
время:	3	4	5	6	7	
среднеквадратичная погрешность	3,22E-15	9E-15	1,99E-14	3,98E-14	7,73E-14	2,98E-14

Погрешность только вычислительная, значит, коэффициент верно раскладывается по базисным функциям пространства

3. *Неравномерная сетка по времени*

Пусть по пространству задана все та же сетка от 1 до 5 с шагом 1 по переменным z и r. Сетку по времени сформируем из условий t от 0 до 9, 9 разбиений с коэффициентом релаксации 1,34. Тогда сетка будет:

time	0	0,236664	0,553793	0,978746	1,548183	2,311229	3,33371	4,703836	6,539803	9

Пусть $u = r^2+3z-2,5t^2$, коэффициенты лямбда и сигма равны 1

время:	0,9787	1,5481	2,3112	3,3337	4,7038	6,5398	9	среднее значение
среднеквадратичная погрешность	5,35E-15	9,09E-15	2,09E-14	2,35E-14	9,33E-14	1,19E-13	3,07E-13	8,26068E-14

Учет меняющегося по времени шага производится корректно

4. *Неравномерная сетка по времени и пространству*

Пусть сетки формируются из условий:

- r от 1 до 10, 9 разбиений с коэффициентом релаксации 1
- z от 1 до 10, 7 разбиений с коэффициентом релаксации 1,182
- t от 0 до 9, 9 разбиений с коэффициентом релаксации 1,34

Сетки:

		r	1	2	3	4	5	6	7	8	9	10
	z	1	1,73669	2,607457	3,636703	4,853273	6,291258	7,990956	10			
time	0	0,236664	0,553793	0,978746	1,548183	2,311229	3,33371	4,703836	6,539803	9		

Пусть $u = r^2+3z-2,5t^2$, коэффициенты лямбда и сигма равны 1

время:	0,9787	1,5481	2,3112	3,3337	4,7038	6,5398	9	среднее значение
среднеквадратичная погрешность	1,23E-13	8,41E-14	7,79E-14	7,68E-14	8,64E-14	8,22E-14	1,77E-13	1,01018E-13

Неравномерный шаг по пространству тоже учитывается корректно.

11. Выводы

Успешно реализовано решение начально-краевой параболической задачи в цилиндрических координатах с использованием четырехслойной неявной схемы для аппроксимации по времени, учет краевых всех типов, разложение коэффициента сигма по базису пространства, а так же работа с неравномерными сетками.

На основе исследований экспериментальным путем удалось найти порядок аппроксимации и сходимости четырехслойной неявной схемы

Порядок аппроксимации четырехслойной неявной схемы

Для всех исследуемых функций погрешность аппроксимации по времени появляется на полиноме 4-ой степени по t. Следовательно, порядок аппроксимации четырехслойной неявной схемы равен трем, что соответствует теоретическому предположению, так как базис по времени является кубическим.

Порядок сходимости четырехслойной неявной схемы

Полученный экспериментально порядок сходимости по времени для четырехслойной неявной схемы близок к теоретически ожидаемому третьему. Если искомая функция зависит и от пространственных координат r, z, общий порядок сходимости снижается.

12.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; // number of gamma function
};

struct element
{
    std::vector<int> node_loc;
    int mater;
    int f_id;
};

std::vector<node> all_nodes;          // все узлы в порядке глобальной нумерации
std::vector<element> all_elems;       // все элементы в прядке глобальной нумерации
std::vector<material> all_materials; // все материалы по индексам
std::vector<std::pair<int,std::vector<int>>> S1; // S1[i][j] на j-ом узле заданы краевые 1 рода
std::vector<std::pair<int,std::vector<int>>> S2_r;    // граница параллельна r на j-ом узле заданы краевые 2 рода
```

```
std::vector<std::pair<int,std::vector<int>>>
S2_z;    // граница параллельна z на j-ом узле
заданы краевые 2 рода
std::vector<std::pair<int,std::vector<int>>>
S3_r;    // граница параллельна r на j-ом узле
заданы краевые 3 рода
std::vector<std::pair<int,std::vector<int>>>
S3_z;    // граница параллельна z на j-ом узле
заданы краевые 3 рода

std::vector<double> time_grid; // сетка времени
int i_t = 0; // текущий временной слой

double gamma(double r, double z, int gam_id) //
значение гамма по индексу gam_id
{
    switch (gam_id)
    {
    case 0:
        return 2;
    default:
        std::cout << "can't find gamma № " <<
gam_id << "\n";
        break;
    }
}
double beta(double r, double z, int beta_id) //
считаем, что бета везде одинаковая
{
    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) //
значение f по индексу f_id
{
    double t = time_grid[i_t];
    switch (f_id)
    {
    case 0:
        return  4
            ;
    default:
        std::cout << "can't find f № " << f_id <<
"\n";
        break;
    }
}

double func_S(double r, double z, int s_id) //
значение краевого S по индексу f_id
{
    double t = time_grid[i_t];
    switch (s_id)
    {
    case 0:
        return 1.5 * z + 2* t
            ;
    case 1:// 2_z
        return 2 * r
            ;
    case 2: // 2_r
        return 3
            ;
    case 3: // 3_z
        return 1 / beta(r,z,0) * 2 * r + r*r +3*z
- 5*t
            ;
    case 4: // 3_r
        return 1 / beta(r, z, 0) * 3 + r * r + 3
* z - 5 * t
            ;
    default:
        std::cout << "can't find S № " << s_id <<
"\n";
        break;
    }
}

int Input() // чтение данных
{
    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) //
получение локальной G
{
    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) //
прибавление локальной M
{
    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) //
получение локального b
{
    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) // получение локальной матрицы A
{
    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;

    // получить M
    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);

    // получить G
    GetG_Loc(all_nodes[el.node_loc[0]].r,
all_materials[el.mater].lambda, hr, hz, G_loc);
// A_loc = G_loc
    return 0;
}
int Get_Loc_b(std::vector<double>& b_loc,
    int el_id) // получить вектор правой части
{
```

```
    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;
    // получить f
    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) // генерация портрета
{
    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)
    // внесение локальных A, b  в глобальную СЛАУ
{
    std::vector<int> L =
all_elems[el_id].node_loc;
    int k = all_elems[el_id].node_loc.size(); //
размерность локальной матрицы
    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)
```

```
    // внесение локальных b  в глобальную СЛАУ
{
    std::vector<int> L =
all_elems[el_id].node_loc;
    int k = all_elems[el_id].node_loc.size(); //
размерность локальной матрицы
    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) // учет первых
краевых
{
    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) //
локальная матрица для S3_r
{

    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) //
получение локального b для S параллельных r
{
    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) //
локальная матрица для S3_z
{
    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) //
получение локального b для S параллельных z
{
    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) // внесение локальной
матрицы в глобальную для одномерного случая
{
    std::vector<int> L(2);
    L[0] = node1;
    L[1] = node2;
    int k = 2; // размерность локальной матрицы
    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) // учет вторых краевых
MS.b - вектор вклада от краевых
{
    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) // MS.b -
вектор вклада от краевых
{
    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);
            // mult M_loc on beta
            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)
                // add local to MS
                AddLocal_dim2(MS.ia, MS.ja,
MS.di, MS.al, MS.au, M_loc, node_id1, node_id2);
            // get MS.b

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);
            // mult M_loc on beta
            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)
                // add local to MS
                AddLocal_dim2(MS.ia, MS.ja,
MS.di, MS.al, MS.au, M_loc, node_id1, node_id2);
            // get MS.b

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()
{
    //Make_grid(""); // for creating tests
    //Create_time_grid();

    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);

    // собираем матрицы M, G
    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];

        //пересобрать матрицу, если необходимо
        if (change_matrix)
        {
            A = G;
            A.b.Size(G.N);
            A = A + M * ((dt01 * dt02 + dt01 *
dt03 + dt02 * dt03) / (dt01 * dt02 * dt03));

        }
        // пересобрать вектор правой части
        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));

        // учесть краевые
        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) // если меняли
матрицу, сбросить учтенные в А краевые
        {
            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);
        }

        // решить СЛАУ
        Solver slau(A);
        slau.CGM_LU();
        slau.getx0(q4.vect);

        // вывести ответ на временном слое
        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";
        }

        // сменить слой
        q1.vect.swap(q2.vect);
```

```
        q2.vect.swap(q3.vect);
        q3.vect.swap(q4.vect);
    }


    return 0;
}
```

MyMatrix.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;
}
```

MyMatrix.h

```
#pragma once
#include "MyVector.h"

class MyMatrix
{
public:
    std::vector<double> di;
    std::vector<double> al;
    std::vector<double> au;
    MyVector b;
    std::vector<int> ja;
    std::vector<int> ia;
    int N;

    MyMatrix(void);
    void ReadMatrix(int size);
    void Ax(std::vector<double>& X,
std::vector<double>& Y); // умножение матрицы на
вектор y = Ax
    void Ax(MyVector& X, MyVector& Y); //
умножение матрицы на вектор y = Ax
    void ATx(MyVector& X, MyVector& Y); //
умножение транспонированной матрицы на вектор y =
A^T*x
    MyMatrix& operator + (MyMatrix B);
    MyMatrix operator * (const double a);
    MyMatrix& operator=(const MyMatrix& B);
};
```

MyVector.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));
}
```

MyVector.h

```
#pragma once
#include <vector>
#include <iostream>
#include <math.h>
#include <fstream>
class MyVector
{
public:
    std::vector<double> vect;
    MyVector();
    void Size(int N);
    void ReadVector(std::string filename);
    MyVector& operator=(const MyVector& a);
    double operator* (const MyVector& a);
    MyVector& operator* (const double a);
    MyVector& operator-(MyVector& a);
    MyVector& operator+(MyVector& a);
    double Norm();
};
```

Solver.cpp

```
#include "Solver.h"

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];
        }
    }
}
```

Solver.h

```
#pragma once
#include "MyMatrix.h"
class Solver
{
private:
    MyMatrix A;
    MyVector p,
        z,
        r,
        x0,
        Ar,
        y;
    std::vector<double> L, D, U;
    int N;
    int maxIter;
    double eps;
    int iter;
    double normR;
    double normB;
public:

    Solver(int size);
    Solver(std::string filename);
    Solver(MyMatrix _A);

    void CGM_LU(); // conjugate gradient method
whith LU factorization
    void LOS_LU(); // locally optimal scheme
whith LU factorization
    void FactLU(std::vector<double>& L,
std::vector<double>& U, std::vector<double>& D);
    void Direct(std::vector<double>& L,
std::vector<double>& D, MyVector& y, MyVector&
b);
    void Direct(std::vector<double>& L, MyVector&
y, MyVector& b);
    void Reverse(std::vector<double>& U,
MyVector& x, MyVector& y);
    void Reverse(std::vector<double>& U,
std::vector<double>& D, MyVector& x, MyVector&
y);
    void output(std::string filename);
    void getx0(std::vector<double>& x);

};
```

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();
}
```

Generate.h

```
#pragma once
#include <vector>
#include <fstream>

void Make_grid(std::string path);
void Create_time_grid();
```