МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

НГТУ НЭТИ | Факультет прикладной математики и информатики

Кафедра прикладной математики

Курсовой проект

по дисциплине «Численные методы»

МКЭ

Студент ЧИРКОВ АРТЁМ

Группа ПМ-01

Преподаватели ПЕРСОВА МАРИНА ГЕНАДЬЕВНА

Новосибирск, 2022

1. Постановка задачи

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

2. Вариационная постановка

Необходимо решить уравнение

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

заданное в некоторой области Ω с границей и краевыми условиями:

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

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

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

Перепишем дифференциальное уравнение в цилиндрических координатах:

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

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

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

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

$$\int_\Omega \left(-\frac{1}{r}\frac{\partial}{\partial r}\left(r\lambda\frac{\partial u}{\partial r}\right) - \frac{\partial}{\partial z}\left(\lambda\frac{\partial u}{\partial z}\right) + \gamma u - f\right) v d\Omega = 0 \;\; \forall v \in \Phi$$

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

$$\int_{\Omega} \lambda grad\ u\ grad\ v\ rd\Omega + \int_{\Omega} \gamma uvrd\Omega + \int_{S_3} \beta uv_0 dS =$$

$$\int_{\Omega} fvrd\Omega + \int_{S_2} \theta v_0 dS + \int_{S_3} \beta u_\beta v_0 dS\ \ \forall v_0 \in H_0^1$$

3. Дискретизация и базисные функции

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

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

$$\int_{\Omega} fv_0{}^h rd\Omega + \int_{S_2} \theta v_0{}^h dS + \int_{S_2} \beta u_\beta v_0{}^h dS\ \ \forall v_0{}^h \in V_0^h$$

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

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

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

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

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

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

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

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

А также локальные базисные функции:

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

Компоненты локальных матриц жесткости и массы имеют вид:

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

$$\hat{M}_{ij} = \int_{\Omega_k} \hat{\gamma}\left(\hat{\psi}_i \hat{\psi}_j\right) r dr dz$$

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

4. **Аналитические выражения для вычисления локальных матриц**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\hat{M}_{ij} = \int\limits_{\Omega_k} \hat{\gamma}\left(\hat{\psi}_i \hat{\psi}_j\right) rdrdz = \sum_{k=1}^{n} \gamma_k \int\limits_{\Omega_k} \hat{\psi}_i \hat{\psi}_j \hat{\psi}_k rdrdz$$

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

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

$$\widehat{M}_{13}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{4}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{4}+\frac{h_r}{20}\right)\frac{h_z}{12}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}\right)h_r$$
$$\widehat{M}_{14}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}\right)h_r$$
$$\widehat{M}_{22}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{4}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{4}+\frac{h_r}{5}\right)\frac{h_z}{4}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{4}+\frac{h_r}{5}\right)\frac{h_z}{12}\right)h_r$$
$$\widehat{M}_{23}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}\right)h_r$$
$$\widehat{M}_{24}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{4}+\frac{h_r}{5}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{4}+\frac{h_r}{5}\right)\frac{h_z}{12}\right)h_r$$
$$\widehat{M}_{33}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{4}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{4}+\frac{h_r}{20}\right)\frac{h_z}{4}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{4}\right)h_r$$
$$\widehat{M}_{34}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{30}\right)\frac{h_z}{4}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{4}\right)h_r$$
$$\widehat{M}_{44}=\left(\gamma(r_p,z_s)\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{12}+\gamma(r_{p+1},z_s)\left(\frac{r_p}{4}+\frac{h_r}{5}\right)\frac{h_z}{12}+\gamma(r_p,z_{s+1})\left(\frac{r_p}{12}+\frac{h_r}{20}\right)\frac{h_z}{4}\right.$$
$$\left.+\gamma(r_{p+1},z_{s+1})\left(\frac{r_p}{4}+\frac{h_r}{5}\right)\frac{h_z}{4}\right)h_r$$

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 $\gamma\equiv 1$

5. Краевые условия

Краевые условия первого рода

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

Краевые условия второго рода

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

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

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

Краевые условия третьего рода

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

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

6.Тесты

Во всех случаях r и z
находятся в диапазоне
от 1 до 3

На равномерной сетке
Лямбда = гамма = 1
Проверим для константы
Истинное значение функции: u* = 1
f = 1
Количество узлов сетки: 9
Количество конечных элементов: 4

r	z	q	u*(r,z)	u*(r,z) - u
1	1	1	1	0
2	1	1	1	0
3	1	1	1	0
1	2	1	1	0
2	2	1	1	0
3	2	1	1	0
1	3	1	1	0
2	3	1	1	0
3	3	1	1	0

Проверим для линейной функции
Лямбда = гамма = 1
Истинное значение функции: u* = z
f = z
Количество узлов сетки: 9
Количество конечных элементов: 4

r	z	q	u*(r,z)	u*(r,z) - u
1	1	1	1	0
2	1	1	1	0
3	1	1	1	0
1	2	2	2	0
2	2	2	2	0
3	2	2	2	0
1	3	3	3	0
2	3	3	3	0
3	3	3	3	0

Проверим для нелинейной функции

Лямбда = гамма = 1

Истинное значение функции: u* = rz

f = rz – z/r

Количество узлов сетки: 9

Количество конечных элементов: 4

r	z	q	u*(r,z)	u*(r,z) - u
1	1	1	1	0
2	1	2	2	0
3	1	3	3	0
1	2	2	2	0
2	2	3,98214	4	0,01786
3	2	6	6	0
1	3	3	3	0
2	3	6	6	0
3	3	9	9	0

Зададим более мелкою сетку

Лямбда = 1, гамма = r + z

Истинное значение функции: u* = 3

f = 3*(r+z)

Количество узлов сетки: 25

Количество конечных элементов:16

Сетка неравномерная

r	z	q	u*(r,z)	u*(r,z) - u
1	1	3	3	0
1,5	1	3	3	0
2	1	3	3	0
2,5	1	3	3	0
3	1	3	3	0
1	1,5	3	3	0
1,5	1,5	3	3	0
2	1,5	3	3	0
2,5	1,5	3	3	0
3	1,5	3	3	0
1	2	3	3	0
1,5	2	3	3	0
2	2	3	3	0
2,5	2	3	3	0
3	2	3	3	0
1	2,3	3	3	0
1,5	2,3	3	3	0

2	2,3	3	3	0
2,5	2,3	3	3	0
3	2,3	3	3	0
1	2,7	3	3	0
1,5	2,7	3	3	0
2	2,7	3	3	0
2,5	2,7	3	3	0
3	2,7	3	3	0

Исследуем порядок сходимости:

Гамма = лямбда = 1

u* = rz

f = rz – z/r

Для шага h:

Количество узлов сетки: 9

Количество конечных элементов: 4

r	z	q	u*(r,z)	u*(r,z) - u
1	1	1	1	0
2	1	2	2	0
3	1	3	3	0
1	2	2	2	0
2	2	3,98214	4	0,01786
3	2	6	6	0
1	3	3	2	0
2	3	6	6	0
3	3	9	9	0

Относительная погрешность: 0,01786

Для шага h/2:

Количество узлов сетки: 25

Количество конечных элементов: 16

r	z	q	u*(r,z)	u*(r,z) - u
1	1	1	1	0
1,5	1	1,5	1,5	0
2	1	2	2	0
2,5	1	2,5	2,5	0
3	1	3	3	0
1	1,5	1,5	1,5	0
1,5	1,5	2,24689	2,25	0,00311
2	1,5	2,99777	3	0,00223
2,5	1,5	3,74887	3,75	0,00113
3	1,5	4,5	4,5	0
1	2	2	2	0
1,5	2	2,99554	3	0,00446
2	2	3,99669	4	0,00331
2,5	2	4,99835	5	0,00165

3	2	6	6	0
1	2,5	2,5	2,5	0
1,5	2,5	3,74556	3,75	0,00444
2	2,5	4,9971	5	0,0029
2,5	2,5	6,24856	6,25	0,00144
3	2,5	7,5	7,5	0
1	3	3	3	0
1,5	3	4,5	4,5	0
2	3	6	6	0
1	1	1	1	0
1,5	1	1,5	1,5	0

Относительная погрешность: 0,00548321
Отношение погрешностей h и h/2: 3,257216

Для шага h/4:
Количество узлов сетки: 81
Количество конечных элементов: 64

r	z	q	u*(r,z)	u*(r,z) - u
1	1	1	1	0
1,25	1	1,25	1,25	0
1,5	1	1,5	1,5	0
1,75	1	1,75	1,75	0
2	1	2	2	0
2,25	1	2,25	2,25	0
2,5	1	2,5	2,5	0
2,75	1	2,75	2,75	0
3	1	3	3	0
1	1,25	1,25	1,25	0
1,25	1,25	1,56211	1,5625	0,00039
1,5	1,25	1,87457	1,875	0,00043
1,75	1,25	2,18712	2,1875	0,00038
2	1,25	2,4997	2,5	0,0003
2,25	1,25	2,81228	2,8125	0,00022
2,5	1,25	3,12485	3,125	0,00015
2,75	1,25	3,43742	3,4375	8,00E-05
3	1,25	3,75	3,75	0
1	1,5	1,5	1,5	0
1,25	1,5	1,87439	1,875	0,00061
1,5	1,5	2,24928	2,25	0,00072
1,75	1,5	2,62435	2,625	0,00065
2	1,5	2,99947	3	0,00053
2,25	1,5	3,3746	3,375	0,0004
2,5	1,5	3,74973	3,75	0,00027
2,75	1,5	4,12486	4,125	0,00014
3	1,5	4,5	4,5	0
1	1,75	1,75	1,75	0
1,25	1,75	2,18673	2,1875	0,00077
1,5	1,75	2,62408	2,625	0,00092
1,75	1,75	3,06166	3,0625	0,00084

2	1,75	3,49931	3,5	0,00069
2,25	1,75	3,93698	3,9375	0,00052
2,5	1,75	4,37465	4,375	0,00035
2,75	1,75	4,81232	4,8125	0,00018
3	1,75	5,25	5,25	0
1	2	2	2	0
1,25	2	2,49913	2,5	0,00087
1,5	2	2,99895	3	0,00105
1,75	2	3,49905	3,5	0,00095
2	2	3,99922	4	0,00078
2,25	2	4,49942	4,5	0,00058
2,5	2	4,99961	5	0,00039
2,75	2	5,4998	5,5	0,0002
3	2	6	6	0
1	2,25	2,25	2,25	0
1,25	2,25	2,81157	2,8125	0,00093
1,5	2,25	3,37391	3,375	0,00109
1,75	2,25	3,93652	3,9375	0,00098
2	2,25	4,49921	4,5	0,00079
2,25	2,25	5,06191	5,0625	0,00059
2,5	2,25	5,6246	5,625	0,0004
2,75	2,25	6,18729	6,1875	0,00021
3	2,25	6,75	6,75	0
1	2,5	2,5	2,5	0
1,25	2,5	3,12411	3,125	0,00089
1,5	2,5	3,749	3,75	0,001
1,75	2,5	4,37413	4,375	0,00087
2	2,5	4,99931	5	0,00069
2,25	2,5	5,62449	5,625	0,00051
2,5	2,5	6,24966	6,25	0,00034
2,75	2,5	6,87482	6,875	0,00018
3	2,5	7,5	7,5	0
1	2,75	2,75	2,75	0
1,25	2,75	3,43681	3,4375	0,00069
1,5	2,75	4,12431	4,125	0,00069
1,75	2,75	4,81193	4,8125	0,00057
2	2,75	5,49956	5,5	0,00044
2,25	2,75	6,18718	6,1875	0,00032
2,5	2,75	6,87478	6,875	0,00022
2,75	2,75	7,56238	7,5625	0,00012
3	2,75	8,25	8,25	0
1	3	3	3	0
1,25	3	3,75	3,75	0
1,5	3	4,5	4,5	0
1,75	3	5,25	5,25	0
2	3	6	6	0
2,25	3	6,75	6,75	0
2,5	3	7,5	7,5	0
2,75	3	8,25	8,25	0
3	3	9	9	0

Относительная погрешность: 0,00150345

Отношение погрешностей h/2 и h/4: 3,6470857

Проверим для неполиномальной функции

Лямбда = гамма = 1

Истинное значение функции: u* = cos(z) + r

f = -1/r+2*cos(z)+r

Количество узлов сетки: 9

Количество конечных элементов: 4

r находится в диапазоне от 1001 до 1003

r	z	q	u*(r,z)	u*(r,z) - u
1001	1	1001,540	1001,724	-1,83215E-01
1002	1	1002,540	1002,052	4,88113E-01
1003	1	1003,540	1002,633	9,07694E-01
1001	2	1000,584	1001,496	-9,12212E-01
1002	2	1001,584	1001,825	-2,40883E-01
1003	2	1002,584	1002,405	1,78697E-01
1001	3	1000,010	1001,229	-1,21940E+00
1002	3	1001,010	1001,558	-5,48068E-01
1003	3	1002,010	1002,138	-1,28487E-01
1001	4	1000,346	1001,187	-8,40281E-01
1002	4	1001,346	1001,515	-1,68952E-01
1003	4	1002,346	1002,096	2,50628E-01

7. Текст программы

main.cpp

```
#include "Solver.h"
#include <fstream>
#include <iostream>
#include <vector>
struct node
{
    double r;
    double z;
};
struct material
{
    double lambda;
    int gamma_id;
};
struct element
{
    std::vector<int> node_loc;
    int mater;
    int f_id;
};
std::vector<node> all_nodes;
std::vector<element> all_elems;
std::vector<material> all_materials;
std::vector<std::vector<int>> S1;
double gamma(double r, double z, int gam_id)
{
    switch (gam_id)
    {
    case 0:
        return 1;
    default:
        break;
    }
}
double func_f(double r, double z, int f_id)
{
    switch (f_id)
    {
    case 0:
        return r * z - z / r;
    default:
        break;
    }
}
double func_S1(double r, double z, int s1_id)
{
    switch (s1_id)
    {
    case 0:
        return r * z;
    default:
        break;
    }
}
int Input()
{
    int N, Nmat, Kel, NS1;
    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("S1.txt");
    S1.resize(NS1);
    for (int i = 0; i < NS1; i++)
    {
        int size;
        in >> size;
        S1[i].resize(size);
        for (int j = 0; j < size; j++)
        {
            in >> S1[i][j];
        }
    }
    in.close();
    in.open("material.txt");
    all_materials.resize(Nmat);
    for (int i = 0; i < Nmat; i++)
    {
        in >> all_materials[i].lambda >> all_materials[i].gamma_id;
    }
    in.close();
    in.open("elem.txt");
    all_elems.resize(Kel);
    for (int i = 0; i < Kel; i++)
    {
        all_elems[i].node_loc.resize(4);
        in >> all_elems[i].node_loc[0] >> all_elems[i].node_loc[1]
            >> all_elems[i].node_loc[2] >> all_elems[i].node_loc[3]
            >> all_elems[i].mater >> all_elems[i].f_id;
    }
    in.close();
    return 0;
}
double GetG_Loc(double rp, double lambda, double hr, double hz,
    std::vector<std::vector<double>>& G_loc)
{
    double a1 = (lambda * hz * rp) / (6 * hr),
        a2 = (lambda * hz) / (12),
        a3 = (lambda * hr * rp) / (6 * hz),
        a4 = (lambda * hr * hr) / (12 * hz);
    G_loc[0][0] = 2 * a1 + 2 * a2 + 2 * a3 + 1 * a4;
    G_loc[0][1] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[0][2] = 1 * a1 + 1 * a2 - 2 * a3 - 1 * a4;
    G_loc[0][3] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[1][0] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[1][1] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
    G_loc[1][2] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[1][3] = 1 * a1 + 1 * a2 - 2 * a3 - 3 * a4;
    G_loc[2][0] = 1 * a1 + 1 * a2 - 2 * a3 - 1 * a4;
    G_loc[2][1] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[2][2] = 2 * a1 + 2 * a2 + 2 * a3 + 1 * a4;
    G_loc[2][3] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;

        G_loc[3][0] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
    G_loc[3][1] = 1 * a1 + 1 * a2 - 2 * a3 - 3 * a4;
    G_loc[3][2] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
    G_loc[3][3] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
    return 0;
}
double GetM_Loc(double rp, double zs, int gam, double hr, double hz,
```

```
    std::vector<std::vector<double>>& M_loc)
{
    double g1 = gamma(rp, zs, gam),
        g2 = gamma(rp + hr, zs, gam),
        g3 = gamma(rp, zs + hz, gam),
        g4 = gamma(rp + hr, zs + hz, gam);
    M_loc[0][0] += hr * (
        g1 * (rp / 4 + hr / 20) * hz / 4 +
        g2 * (rp / 12 + hr / 30) * hz / 4 +
        g3 * (rp / 4 + hr / 20) * hz / 12 +
        g4 * (rp / 12 + hr / 30) * hz / 12);
    M_loc[0][1] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 4 +
        g2 * (rp / 12 + hr / 20) * hz / 4 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[0][2] += hr * (
        g1 * (rp / 4 + hr / 20) * hz / 12 +
        g2 * (rp / 12 + hr / 30) * hz / 12 +
        g3 * (rp / 4 + hr / 20) * hz / 12 +
        g4 * (rp / 12 + hr / 30) * hz / 12);
    M_loc[0][3] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 12 +
        g2 * (rp / 12 + hr / 20) * hz / 12 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[1][0] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 4 +
        g2 * (rp / 12 + hr / 20) * hz / 4 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[1][1] += hr * (
        g1 * (rp / 12 + hr / 20) * hz / 4 +
        g2 * (rp / 4 + hr / 5) * hz / 4 +
        g3 * (rp / 12 + hr / 20) * hz / 12 +
        g4 * (rp / 4 + hr / 5) * hz / 12);
    M_loc[1][2] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 12 +
        g2 * (rp / 12 + hr / 20) * hz / 12 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[1][3] += hr * (
        g1 * (rp / 12 + hr / 20) * hz / 12 +
        g2 * (rp / 4 + hr / 5) * hz / 12 +
        g3 * (rp / 12 + hr / 20) * hz / 12 +
        g4 * (rp / 4 + hr / 5) * hz / 12);
    M_loc[2][0] += hr * (
        g1 * (rp / 4 + hr / 20) * hz / 12 +
        g2 * (rp / 12 + hr / 30) * hz / 12 +
        g3 * (rp / 4 + hr / 20) * hz / 12 +
        g4 * (rp / 12 + hr / 30) * hz / 12);
    M_loc[2][1] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 12 +
        g2 * (rp / 12 + hr / 20) * hz / 12 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[2][2] += hr * (
        g1 * (rp / 4 + hr / 20) * hz / 12 +

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

```
    M_loc[3][0] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 12 +
        g2 * (rp / 12 + hr / 20) * hz / 12 +
        g3 * (rp / 12 + hr / 30) * hz / 12 +
        g4 * (rp / 12 + hr / 20) * hz / 12);
    M_loc[3][1] += hr * (
        g1 * (rp / 12 + hr / 20) * hz / 12 +
        g2 * (rp / 4 + hr / 5) * hz / 12 +
        g3 * (rp / 12 + hr / 20) * hz / 12 +
        g4 * (rp / 4 + hr / 5) * hz / 12);
    M_loc[3][2] += hr * (
        g1 * (rp / 12 + hr / 30) * hz / 12 +
        g2 * (rp / 12 + hr / 20) * hz / 12 +
        g3 * (rp / 12 + hr / 30) * hz / 4 +
        g4 * (rp / 12 + hr / 20) * hz / 4);
    M_loc[3][3] += hr * (
        g1 * (rp / 12 + hr / 20) * hz / 12 +
        g2 * (rp / 4 + hr / 5) * hz / 12 +
        g3 * (rp / 12 + hr / 20) * hz / 4 +
        g4 * (rp / 4 + hr / 5) * hz / 4);
    return 0;
}
int Getb_Loc(double rp, double zs, double hr, double hz,
    std::vector<double>& b_loc, int f_id)
{
    double f1 = func_f(rp, zs, f_id),
        f2 = func_f(rp + hr, zs, f_id),
        f3 = func_f(rp, zs + hz, f_id),
        f4 = func_f(rp + hr, zs + hz, f_id);
    b_loc[0] =
        f1 * (hr * hz / 3 * (rp / 3 + hr / 12)) +
        f2 * (hr * hz / 3 * (rp / 6 + hr / 12)) +
        f3 * (hr * hz / 6 * (rp / 3 + hr / 12)) +
        f4 * (hr * hz / 6 * (rp / 6 + hr / 12));
    b_loc[1] =
        f1 * (hr * hz / 3 * (rp / 6 + hr / 12)) +
        f2 * (hr * hz / 3 * (rp / 3 + hr / 4)) +
        f3 * (hr * hz / 6 * (rp / 6 + hr / 12)) +
        f4 * (hr * hz / 6 * (rp / 3 + hr / 4));
    b_loc[2] =
        f1 * (hr * hz / 6 * (rp / 3 + hr / 12)) +
        f2 * (hr * hz / 6 * (rp / 6 + hr / 12)) +
        f3 * (hr * hz / 3 * (rp / 3 + hr / 12)) +
        f4 * (hr * hz / 3 * (rp / 6 + hr / 12));
    b_loc[3] =
        f1 * (hr * hz / 6 * (rp / 6 + hr / 12)) +
        f2 * (hr * hz / 6 * (rp / 3 + hr / 4)) +
        f3 * (hr * hz / 3 * (rp / 6 + hr / 12)) +
        f4 * (hr * hz / 3 * (rp / 3 + hr / 4));
    return 0;
}
int Get_Loc(std::vector<std::vector<double>>& A_loc, std::vector<double>& b_loc,
    int el_id)
{
    element el = all_elems[el_id];
    double hr = all_nodes[el.node_loc[1]].r - all_nodes[el.node_loc[0]].r,

        hz = all_nodes[el.node_loc[2]].z - all_nodes[el.node_loc[0]].z;
    GetG_Loc(all_nodes[el.node_loc[0]].r, all_materials[el.mater].lambda, hr, hz,
        A_loc); // A_loc = G_loc
    GetM_Loc(all_nodes[el.node_loc[0]].r, all_nodes[el.node_loc[0]].z,
all_materials[el.mater].gamma_id, hr, hz, A_loc);
    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(std::vector<int>& ia, std::vector<int>& ja,
```

```
    int N, int Kel)
{
    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>& ia, std::vector<int>& ja, std::vector<double>& di,
    std::vector<double>& al, std::vector<double>& au,
    std::vector<std::vector<double>>& A_loc,
    std::vector<double>& b, std::vector<double>& b_loc, int el_id)
{
    std::vector<int> L = all_elems[el_id].node_loc;
    int k = all_elems[el_id].node_loc.size(); // размерность локальной матрицы
    for (int i = 0; i < k; i++)
    {
        di[L[i]] += A_loc[i][i];
        b[L[i]] += b_loc[i];
    }
    for (int i = 0; i < 4; i++)
    {
        int temp = ia[L[i]];
        for (int j = 0; j < i; j++)
        {
            for (int k = temp; k < ia[L[i] + 1]; k++)
            {
                if (ja[k] == L[j])
                {
                    al[k] += A_loc[i][j];
                    au[k] += A_loc[j][i];
                    k++;
                    break;
                }
            }
        }
    }
    return 0;
}
int SetS1(std::vector<int>& ia, std::vector<int>& ja, std::vector<double>& di,
    std::vector<double>& al, std::vector<double>& au,
    std::vector<double>& b)
{
    int NS1 = S1.size();
    for (int i = 0; i < NS1; i++)
    {
        int s1_id = i;
        for (int j = 0; j < S1[i].size(); j++)
        {
            int node_id = S1[i][j];

                di[node_id] = 1;
            b[node_id] = func_S1(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;
```

```
}
int main()
{
    Input();
    std::vector<int> ia, ja;
    GeneratePortrait(ia, ja, all_nodes.size(), all_elems.size());
    std::vector<double> di(all_nodes.size()), al(ia[all_nodes.size()]),
        au(ia[all_nodes.size()]);
    std::vector<std::vector<double>> A_loc(4);
    for (int i = 0; i < 4; i++)
    {
        A_loc[i].resize(4);
    }
    std::vector<double> b(all_nodes.size()), b_loc(4);

    for (int i = 0; i < all_elems.size(); i++)
    {
        Get_Loc(A_loc, b_loc, i);
        AddLocal(ia, ja, di, al, au, A_loc, b, b_loc, i);
    }
    SetS1(ia, ja, di, al, au, b);
    Solver slau(ia, ja, di, al, au, b, all_nodes.size());
    slau.LOS_LU();
    std::vector<double> q;
    q.resize(di.size());
    slau.getx0(q);
    std::ofstream out("result.txt");
    ;
    for (int i = 0; i < all_nodes.size(); i++)
    {
        out << all_nodes[i].r << " " << all_nodes[i].z << " " << q[i] << "\n";
    }
    return 0;
}
```

Solver.cpp

```
#include "Solver.h"
Solver::Solver(std::vector<int>& ia, std::vector<int>& ja, std::vector<double>& di,
    std::vector<double>& al, std::vector<double>& au,
    std::vector<double>& b, int N_temp)
{
    maxIter = 10000;
    eps = 1E-15;
    A.ReadMatrix(ia, ja, di, al, au, b, N_temp);
        N = N_temp;
    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::getx0(std::vector<double>& x)
{
    for (int i = 0; i < N; i++)
        x[i] = x0.vect[i];
}
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;
        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);
        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];
        }
    }
}
```

matrix.cpp

```
#include "MyMatrix.h"
MyMatrix::MyMatrix()
{
}
void MyMatrix::ReadMatrix(std::vector<int>& ia_temp, std::vector<int>& ja_temp,
    std::vector<double>& di_temp,
    std::vector<double>& al_temp, std::vector<double>& au_temp,
    std::vector<double>& b_temp, int N_temp)
{
    N = N_temp;
    ia.resize(N + 1);
    for (int i = 0; i < N + 1; i++)
    {

            ia[i] = ia_temp[i];
    }
    if (ia[0])
        for (int i = 0; i < N + 1; i++)
        {
            ia[i]--;
        }
    ja.resize(ia[N]);
    for (int i = 0; i < ia[N]; i++)
    {
        ja[i] = ja_temp[i];
    }
    if (ja[0])
        for (int i = 0; i < ia[N]; i++)
        {
            ja[i]--;
        }
    di.resize(N);
    for (int i = 0; i < N; i++)
    {
        di[i] = di_temp[i];
    }
    au.resize(ia[N]);
    for (int i = 0; i < ia[N]; i++)
    {
        au[i] = au_temp[i];
    }
    al.resize(ia[N]);
    for (int i = 0; i < ia[N]; i++)
    {
        al[i] = al_temp[i];
    }
    b.Size(N);
    b.ReadVector(b_temp);
}
// 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 = 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];
        }
    }
}
```

vector.cpp

```
#include "MyVector.h"
MyVector::MyVector()
{
}
void MyVector::Size(int N)
{
    vect.resize(N);
}
void MyVector::ReadVector(std::vector<double>& b_temp)
{
    if (vect.size() < 1)
        return;
    for (int i = 0; i < vect.size(); i++)
    {
        vect[i] = b_temp[i];
    }
}
MyVector& MyVector::operator=(const MyVector& a)
{
    if (this != &a)
        this->vect = a.vect;
    return *this;
}
MyVector& MyVector::operator*(const double a)
{
    for (int i = 0; i < this->vect.size(); i++)
        this->vect[i] *= a;
    return *this;
}
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;
}
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;

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