АННОТАЦИЯ

Отчет 133 с., 4 ч., 51 рис, 10 источников, 1 прил.

РЕШЕНИЕ ЗАДАЧИ С ВЫДЕЛЕНИЕМ ЧАСТИ ПОЛЯ, РАСЧЁТ АНОМАЛЬНОГО СТАЦИОНАРНОГО ЭЛЕКТРИЧЕСКОГО ПОЛЯ В ГОРИЗОНТАЛЬНО СЛОИСТОЙ СРЕДЕ, СРАВНЕНИЕ МЕТОДОВ УЧЁТА ПОЛЯ ГОРИЗОНТАЛЬНО СЛОИСТОЙ СРЕДЫ.

Цель работы – сравнить различные способы учёта части стационарного электрического поля в слоистой среде.

Методом конечных элементов решается краевая задача с точечным источником на поверхности в горизонтально слоистой трёхмерной среде с аномальными объектами.

Разработаны программные модули для решения поставленной задачи различными способами учета поля горизонтально слоистой среды.

Разработанные модули тестируются на полиномиальных функциях и на тривиальной задаче с однородной средой.

Сравнение точности для реализованных методов на различных задачах, с точным решением, полученным аналитически, либо, с решениями, полученными с более высокой точностью.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день существует множество различных методов и подходов к расчёту электрического поля, в основном они относятся к аналитическим или численным методам. Аналитические методы позволяют вычислять точное решение, хотя и не всегда оно может быть получено: это может быть связано со сложной структурой моделируемого объекта. Чаще на практике приходится сталкиваться именно со сложной геометрией, поэтому эффективным инструментом для решения практических задач становятся численные методы.

Одним из широко используемых численных методов является метод конечных элементов. МКЭ – мощный и гибкий инструмент для решения целого ряда проблем, относящихся к моделям в виде краевых и начально-краевых задач для дифференциальных уравнений с частными производными.

Однако вычислительная сложность, которая неизбежно возникает при моделировании сложных структур, порой не позволяет получать решение с необходимой точностью за разумное время.

Существуют подходы, позволяющие значительно ускорить расчеты. Таковым является метод выделения части поля. Для многих практических задач существуют хорошие приближения, которые могут быть получены с использованием более простых моделей. Этим можно воспользоваться для построения менее вычислительно затратных процедур решения.

В этой работе мы рассмотрим различные способы учета выделенной части поля и сравним их точность.

1. ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Рассмотрим математическую модель, описывающую стационарное электрическое поле. Она включает в себя единственное уравнение:

$$-div(\sigma \cdot gradV) = J, \tag{1}$$

где $\sigma(x, y, z)$ – удельная электрическая проводимость среды, V – электрический потенциал, J – функция распределения зарядов в пространстве.

Далее будем рассматривать сосредоточенный в точке $p(x_0, y_0, z_0)$ поверхностный источник с функцией распределения

$$J = I\delta(x - x_0, y - y_0, z - z_0), \tag{2}$$

где I – величина тока, а $\delta(x, y, z)$ – дельта функция.

На удаленной границе S_1 считаем значение потенциала $V = 0$, на поверхности среды S_2 заданы однородные вторые нулевые краевые условия:

$$V\big|_{S_1} = 0, \tag{3}$$

$$\left.\frac{dV}{dn}\right|_{S_2} = 0. \tag{4}$$

1.1.1. ТРЕХМЕРНАЯ СКАЛЯРНАЯ ЗАДАЧА

В декартовой системе координат {x,y,z} дифференциальное уравнение (1) может быть записано в виде:

$$-\frac{\partial}{\partial x}\left(\sigma\frac{\partial V}{\partial x}\right)-\frac{\partial}{\partial y}\left(\sigma\frac{\partial V}{\partial y}\right)-\frac{\partial}{\partial z}\left(\sigma\frac{\partial V}{\partial z}\right)=J\,. \qquad (5)$$

Будем считать, что расчетная область Ω – представляет собой прямоугольную область. Ячейками дискретизации являются параллелепипеды, которые строится из независимых одномерных сеток $\{x_1,\ldots,x_{n_x}\}$, $\{y_1,\ldots,y_{n_y}\}$ и $\{z_1,\ldots,z_{n_z}\}$ путем их декартова умножения.

1.2. ТЕХНОЛОГИЯ ВЫДЕЛЕНИЯ ЧАСТИ ПОЛЯ

В задачах со слоистой средой параметр σ является кусочно-постоянной функцией, зависящей от координат. Обозначим σ^N проводимость среды без учета трехмерных объектов, располагаемых внутри среды. В таком случае проводимость $\sigma^N(z)$ будет зависеть только от одной координаты и совпадать с σ там, где нет трехмерных объектов, а дифференциальное уравнение для среды без трехмерных объектов запишем как:

$$-div(\sigma^N\cdot gradV^N)=J\,, \qquad (6)$$

где V^N – электрический потенциал «нормального» поля.

Представим искомую функцию в виде $V=V^N+V^A$, тогда уравнение (1) примет вид:

$$-div(\sigma\cdot grad(V^N+V^A))=J\,. \qquad (7)$$

Вычтем (6) из (7) и получим:

$$-div(\sigma\cdot gradV^A)=-div((\sigma^N-\sigma)\cdot gradV^N)\,. \qquad (8)$$

Искомую функцию можно получить, последовательно решив задачи (6) и (8), и сложив решения.

Так как для задачи (6) среда оказывается симметричной для поворота относительно оси z, распределение потенциала может быть получено в результате решения осесимметричной задачи в цилиндрических координатах {r,z}.

Такой переход к задаче более низкой размерности позволит получать лучшее решение, при меньших вычислительных затратах.

1.3. КОНЕЧНОЭЛЕМЕНТНОЕ СЛАУ

Получим эквивалентную вариационную формулировку для задач (6) и (8). Для этого умножим обе части уравнения на пробную функцию v, проинтегрируем по области Ω и применим формулу Грина (интегрирования по частям). Для задачи (6):

$$\int_{\Omega}(\sigma^N-\sigma)grad(V^N)\cdot grad(v)d\Omega=\int_{\Omega}Jv,\ \forall v\in H_0^1, \tag{9}$$

где v – пробная функция, H_0^1 – множество функций v, которые вместе со своими производными суммируемы с квадратом на Ω. Нижний индекс «0» у H_0^1 указывает на то, что входящие в него функции удовлетворяют первому краевому условию. Подробнее узнать про теоретические аспекты МКЭ можно в работе [1] (раздел 3).

Вариационная формулировка для задачи (8):

$$\int_{\Omega}\sigma grad(V^A)\cdot grad(v)d\Omega=\int_{\Omega}(\sigma^N-\sigma)grad(V^N)\cdot grad(v)d\Omega,\ \forall v\in H_0^1. \tag{10}$$

Представив V^N и V^A в виде линейных комбинаций $\sum_j q_j^N\psi_j$ и $\sum_j q_j^A\psi_j$ (q_j^N – значения V^N в узлах сетки, аналогично для q_j^A, ψ_j – базисные функции) и

заменив пробную функцию на базисную ψ_i, получим i-ые строки для конечно-элементных СЛАУ:

$$\sum_j \left(\int_\Omega (\sigma^N - \sigma) grad(\psi_j) \cdot grad(\psi_i) d\Omega \right) q_i^N = \sum_j \left(\int_\Omega J \psi_i d\Omega \right), \quad (11)$$

$$\sum_j \left(\int_\Omega \sigma grad(\psi_j) \cdot grad(\psi_i) d\Omega \right) q_j^N = \sum_j \left(\int_\Omega (\sigma^N - \sigma) grad(\psi_j) \cdot grad(\psi_i) d\Omega \right) q_j^N . \quad (12)$$

1.4. БАЗИСНЫЕ ФУНКЦИИ И ЛОКАЛЬНЫЕ ЭЛЕМЕНТЫ

Рассмотрим построение билинейных базисных функций. На отрезке $\left[r_p, r_{p+1} \right]$ задаются две одномерные базисные функции:

$$R_1(r) = \frac{r_{p+1} - r}{h_r}, R_2(r) = \frac{r - r_p}{h_r}, h_r = r_{p+1} - r_p . \quad (13)$$

Аналогично на отрезке $\left[z_t, z_{t+1} \right]$ задаются линейные функции:

$$Z_1(z) = \frac{z_{t+1} - z}{h_z}, Z_2(z) = \frac{z - z_t}{h_z}, h_z = z_{t+1} - z_t . \quad (14)$$

Локальные базисные функции на конечном элементе

$$\Omega_{pt} = \left[r_p, r_{p+1} \right] \times \left[z_t, z_{t+1} \right] \quad (15)$$

представляются в виде произведения функций (13) и (14):

$$\hat{\psi}_1(r,z) = R_1(r)Z_1(z),\ \hat{\psi}_2(r,z) = R_2(r)Z_1(z),$$
$$\hat{\psi}_3(r,z) = R_1(r)Z_2(z),\ \hat{\psi}_4(r,z) = R_2(r)Z_2(z). \quad (16)$$

Выражение для вычисления локальной матрицы жёсткости в цилиндрической {r,z} системе координат:

$$\hat{Q}_{ij} = \int_{\Omega_k} \sigma_k \left(\frac{\partial \hat{\psi}_i}{\partial r}\frac{\partial \hat{\psi}_j}{\partial r} + \frac{\partial \hat{\psi}_i}{\partial z}\frac{\partial \hat{\psi}_j}{\partial z} \right) drdz\,. \quad (17)$$

После вычислений она принимает вид:

$$\hat{Q} = \frac{\sigma}{12 h_r h_z}\left(h_z{}^2\left(\hat{Q}_h^1 + \hat{Q}_r^1\right) + h_r{}^2\left(\hat{Q}_h^2 + \hat{Q}_r^2\right)\right), \quad (18)$$

где:

$$\hat{Q}_h^1 = \begin{pmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{pmatrix}, \hat{Q}_r^1 = \begin{pmatrix} 4 & -4 & 2 & -2 \\ -4 & 4 & -2 & 2 \\ 2 & -2 & 4 & -4 \\ -2 & 2 & -4 & 4 \end{pmatrix},$$

$$\hat{Q}_h^2 = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}, \hat{Q}_r^2 = \begin{pmatrix} 4 & 2 & -4 & -2 \\ 2 & 4 & -2 & -4 \\ -4 & -2 & 4 & 2 \\ -2 & -4 & 2 & 4 \end{pmatrix}.$$

Трилинейные базисные функции на элементе

$$\Omega_{pst} = \left[x_p, x_{p+1}\right] \times \left[y_s, y_{s+1}\right] \times \left[z_t, z_{t+1}\right] \quad (19)$$

строятся аналогично билинейным, покажем, как выглядит выражение для вычислений локальной матрицы жёсткости в декартовой {x,y,z} системе координат:

$$\hat{G}_{ij} = \int_{\Omega_k} \sigma_k \left(\frac{\partial \hat{\psi}_i}{\partial x}\frac{\partial \hat{\psi}_j}{\partial x} + \frac{\partial \hat{\psi}_i}{\partial y}\frac{\partial \hat{\psi}_j}{\partial y} + \frac{\partial \hat{\psi}_i}{\partial z}\frac{\partial \hat{\psi}_j}{\partial z} \right) dxdydz, \qquad (20)$$

после вычислений она принимает вид:

$$\hat{G} = \frac{\sigma_k h_x h_y h_z}{36} \left(\frac{1}{h_x^{\,2}}\hat{G}^x + \frac{1}{h_y^{\,2}}\hat{G}^y + \frac{1}{h_z^{\,2}}\hat{G}^z \right), \qquad (21)$$

где:

$$\hat{G}^x = \begin{pmatrix} 4 & -4 & 2 & -2 & 2 & -2 & 1 & -1 \\ -4 & 4 & -2 & 2 & -2 & 2 & -1 & 1 \\ 2 & -2 & 4 & -4 & 1 & -1 & 2 & -2 \\ -2 & 2 & -4 & 4 & -1 & 1 & -2 & 2 \\ 2 & -2 & 1 & -1 & 4 & -4 & 2 & -2 \\ -2 & 2 & -1 & 1 & -4 & 4 & -2 & 2 \\ 1 & -1 & 2 & -2 & 2 & -2 & 4 & -4 \\ -1 & 1 & -2 & 2 & -2 & 2 & -4 & 4 \end{pmatrix},$$

$$\hat{G}^y = \begin{pmatrix} 4 & 2 & -4 & -2 & 2 & 1 & -2 & -1 \\ 2 & 4 & -2 & -4 & 1 & 2 & -1 & -2 \\ -4 & -2 & 4 & 2 & -2 & -1 & 2 & 1 \\ -2 & -4 & 2 & 4 & -1 & -2 & 1 & 2 \\ 2 & 1 & -2 & -1 & 4 & 2 & -4 & -2 \\ 1 & 2 & -1 & -2 & 2 & 4 & -2 & -4 \\ -2 & -1 & 2 & 1 & -4 & -2 & 4 & 2 \\ -1 & -2 & 1 & 2 & -2 & -4 & 2 & 4 \end{pmatrix},$$

$$\hat{G}^y = \begin{pmatrix} 4 & 2 & 2 & 1 & -4 & -2 & -2 & -1 \\ 2 & 4 & 1 & 2 & -2 & -4 & -1 & -2 \\ 2 & 1 & 4 & 2 & -2 & -1 & -4 & -2 \\ 1 & 2 & 2 & 4 & -1 & -2 & -2 & -4 \\ -4 & -2 & -2 & -1 & 4 & 2 & 2 & 1 \\ -2 & -4 & -1 & -2 & 2 & 4 & 1 & 2 \\ -2 & -1 & -4 & -2 & 2 & 1 & 4 & 2 \\ -1 & -2 & -2 & -4 & 1 & 2 & 2 & 4 \end{pmatrix}.$$

1.5. УЧЕТ НОРМАЛЬНОГО ПОЛЯ

Одним из способов учета нормального поля, которым мы воспользовались, является представление V^N в виде линейной комбинации $\sum_j q_j^N \psi_j$. Таким образом правая часть для уравнения ((12) является стандартной матрицей жёсткости, умноженной на вектор q_i^N:

$$G^{\Delta\sigma} = \int_\Omega (\sigma^N - \sigma) grad(\psi_j) \cdot grad(\psi_i) d\Omega. \qquad (22)$$

Мы рассмотрим два способа формирования вектора q_i^N:

1. Значение V^N при формировании локального элемента будет вычисляться в его центре.
2. Значения V^N при формировании локального элемента будут вычисляться для каждого из узлов локального элемента.

Помимо этого, можно вычислять правую часть уравнения ((10) другими способами. Например, численно вычислив интеграл. Для этого мы используем метод Гаусса.

Также имея решение V^N, мы легко можем посчитать его производную и использовать для вычисления интеграла в правой части уравнения ((10). Применим

два подхода к учёту $gradV^N$: в первом случае будем вычислять значение производной в центре элемента, во втором для каждого из узлов локального элемента.

1.6. РЕШЕНИЕ СЛАУ

Для решения СЛАУ будем использовать метод сопряженных градиентов. Алгоритм для системы уравнений

$$Ax = b \tag{23}$$

с симметричной матрицей A, выглядит следующим образом.

Выбирается начальное приближение x^0 и полагается

$$r^0 = f - Ax^0, \tag{24}$$

$$z^0 = r^0. \tag{25}$$

Далее для $k = 1,2,...$ производятся следующие вычисления:

$$\alpha_k = \frac{(r^{k-1}, r^{k-1})}{(Az^{k-1}, z^{k-1})}, \tag{26}$$

$$x^k = x^{k-1} + \alpha_k z^{k-1}, \tag{27}$$

$$r^k = r^{k-1} - \alpha_k Az^{k-1}, \tag{28}$$

$$\beta_k = \frac{(r^k, r^k)}{(r^{k-1}, r^{k-1})}, \tag{29}$$

$$z^k = r^k + \beta_k z^{k-1}, \tag{30}$$

где x^0 – вектор начального приближения; x^k – вектор решения на k -й (текущей) итерации; r^k – вектор невязки на k -й (текущей) итерации; z^k – вектор спуска (сопряженное направление) на k -й итерации; α_k, β_k – коэффициенты.

Выход из итерационного процесса осуществляется либо по условию малости относительной невязки:

$$\frac{\|r^k\|}{\|f\|} < \varepsilon, \tag{31}$$

либо по достижению максимального количества итераций.

Для ускорения сходимости итерационного процесса применяют метод предобуславливания матрицы. Одним из таких методов является метод неполной факторизации матрицы. Он заключается в замене матрицы A такой матрицей M, что $M^{-1} \approx A^{-1}$ и при этом процедура решения СЛАУ с матрицей M, является менее трудоёмкой.

Для предобуславливания неполным разложением Холесского выбирается $M = SS^T$, которая строится по формулам полного разложение Холесского, но с тем отличием, что портрет нижнетреугольной матрицы S, совпадает с портретом матрицы A, т.е. все ненулевые элементы, которые могли бы получится при разложении, становятся принудительно равными нулю.

2. ОПИСАНИЕ ПРОГРАММНОГО МОДУЛЯ

2.1. ОБЩИЕ СВЕДЕНИЯ

Программная реализация решения задачи представляет собой класс с заголовочным файлом на языке C++.

Ключевым является класс FEM. Для начала работы необходимо создать объект класса FEM, указав название папки, в которой находятся входные файлы (описание входных файлов в разделе 2.2). Затем выполнить расчет поля можно использовав любой из методов:

1. Решение задачи без выделения поля, функция
   void solve(int max_iter, double eps, double relax, std::string solution_filename).
2. Решение с выделением поля первым методом, функция
   void solve_segregation1(int max_iter, double eps, double relax, std::string solution_filename).
3. Решение с выделением поля вторым методом, функция
   void solve_segregation2(int max_iter, double eps, double relax, std::string solution_filename);
4. Решение с выделением поля третьим методом, функция
   void solve_segregation3(int max_iter, double eps, double relax, std::string solution_filename);
5. Решение с выделением поля четвёртым методом, функция
   void solve_segregation4(int max_iter, double eps, double relax, std::string solution_filename);
6. Решение с выделением поля пятым методом, функция
   void solve_segregation5(int max_iter, double eps, double relax, std::string solution_filename);

7. Решение двумерной задачи, функция
   void solve_rz(int max_iter, double eps, double relax, std::string solution_filename).

Описание входных параметров функций: max_iter – максимальное количество итераций решателя, eps – требуемое значение относительной невязки решения СЛАУ, relax – параметр релаксации решателя (по умолчанию лучше ставить равным 1, solution_filename – имя бинарного файла, в которое будет записано решение в узлах сетки.

2.2. ВХОДНЫЕ ФАЙЛЫ

Файл **sreda** – содержит описание сетки и материалов среды:

1. В первой строке число n – количество элементов. Затем n строчек в формате: [x0] [x1] [y0] [y1] [z0] [z1] [n_mat] [anomal], где x0 – левая граница по x, x1 – правая граница по x, аналогично для y и z, n_mat – номер материала, указанного в файле materials. Важно: описание элементов идет сверху вниз в порядке вложенности.
2. Затем идет описание сетки по x:
   - [a] – количество узлов в сетке по x;
   - [x0] [x1] ... [xa] – координаты узлов сетки;
   - [h0] [h1] ... [ha-1] – длина первого шага на элементе;
   - [k0] [k1] ... [ka-1] – коэффициент разрядки;
   - [0/1] * (a-1): 0 – разрядка происходит слева направо, 1 – разрядка происходит справа налево.
3. Повторить пункт 2 для сеток по y и z.
4. [d1] [d2] [d3] – параметры для дробления сетки: 0 – сетка без удвоения узлов, 1 – строится дважды вложенная сетка, 2 – строится четырежды вложенная сетка и так далее.

Файл **sreda_rz** содержит описание сетки двумерной задачи («нормального» поля). В каждой из двух строчек (описание сетки по r и по z) содержится 4 числа, обозначающие координату последнего узла, начальный шаг, максимальный шаг и коэффициент разрядки. Начало координат совпадает с началом одномерных сеток по r и z.

В файле **materials** перечислены свойства для каждого материала. В первой строке указано число n – количество различных материалов. В следующих n строках значения коэффициента проводимости каждого из материалов.

В файле **edge_conditions** 6 чисел обозначающих тип краевых условий на каждой грани расчетной области: x0 – левая грань перпендикулярная оси x, x1 – правая грань перпендикулярная оси x, аналогично для y и z. Число 1 – обозначает задание первых нулевых краевых условий на границе, число 0 – обозначает задание вторых нулевых краевых условий.

Файл **current_sources** содержит описание источников тока. В первой строке число n – обозначающее число источников. Далее n строк по 4 числа: координаты x, y и z; сила тока.

Файл **points** в каждой строчке содержит координаты x,y и z точек, в которых, если потребуется, будет получено решение задачи.

2.3. ВЫХОДНЫЕ ФАЙЛЫ

Описанные в разделе 2.1 функции решения задачи содержат входной параметр solution_filename, в которое будет экспортировано решение.

Формат экспортируемого бинарного файла для трехмерной задачи: первые три числа a,b и c – это четырехбайтовые числа типа unsigned int, обозначающие количество узлов в одномерных сетках x,y и z. Затем подряд записаны координаты узлов одномерных сеток (восьмибайтовые числа типа double). После этого идет запись решения в узлах сетки (восьмибайтовые числа типа double). Всего c*b*a узлов.

Для двумерной задачи описание выходного файла аналогично, лишь с тем отличием что одномерными сетками являются r и z.

Метод solution_in_points, экспортирует решение в файл, который был передан в качестве параметра, для точек указанных во входном файле с именем points.

3. ТЕСТИРОВАНИЕ

3.1. ТЕСТЫ НА ПОЛИНОМАХ

Для того, чтобы убедится в корректности процедур сборок глобальных матриц, непосредственно решения СЛАУ и т.д. необходимо провести ряд тестирований на полиномиальных функциях.

3.1.1. ТЕСТИРОВАНИЕ РЕШЕНИЯ ДВУМЕРНОЙ ЗАДАЧИ

Для тестирования решения двумерной задачи мы будем использовать равномерную сетку (рисунок 1). Для одномерных сеток по r и z будем использовать одинаковые параметры: начальный шаг равный 0.5, граница равная 30 и коэффициент разрядки равный 1. Всего узлов – 3721.

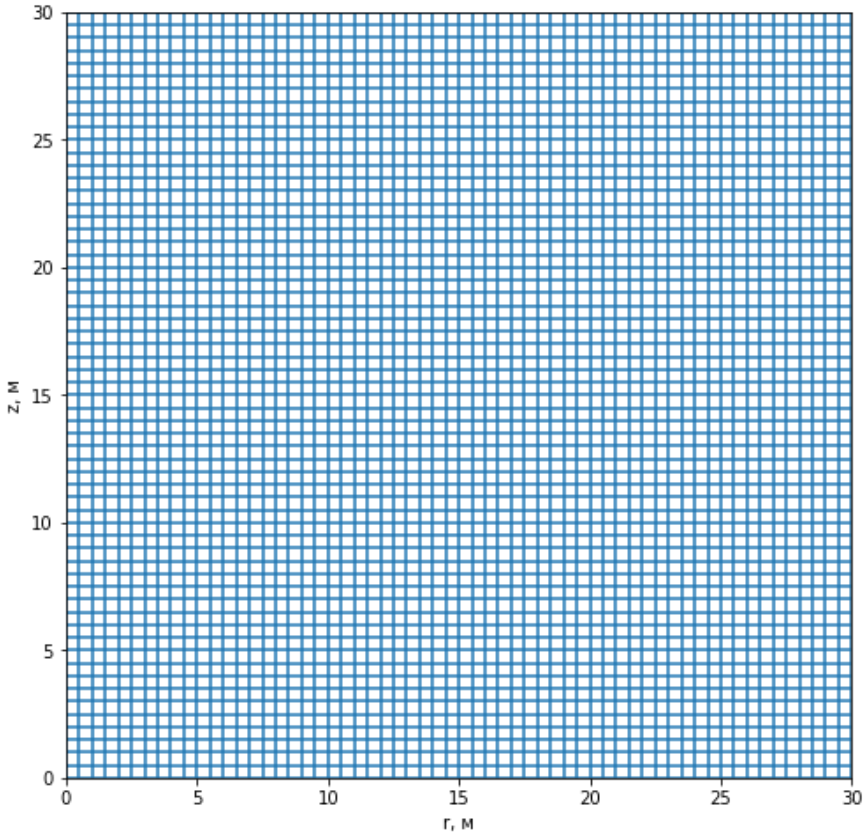


Рисунок 1 – Вид сетки для тестирования двумерной задачи

Покажем точность решения для различных функций. Искомые функции будут представлять собой полиномы различной степени.

Из таблицы 1 видно, что точность решения для тех функций, которые точно представимы выбранными базисными полиномами, сопоставима с точностью представления числа типа double (15 знаков), следовательно, можно полагать, что процедуры построения матриц и решения СЛАУ работают корректно.

Таблица 1 – Результаты тестирования двумерной задачи на полиномах

Искомая функция u	**Правая часть дифференциального уравнения f**	**Параметр σ среды**	**Количество итераций**	**Относительная невязка**	**Норма относительной погрешности**
10	0	1	73	4.108841e-15	1.281123e-14
r	$-\frac{\sigma}{r}$	1	82	2.891056e-15	6.811311e-04
z	0	1	81	3.291777e-15	1.198551e-14
z	0	2	81	3.428631e-15	1.188214e-14
$2z$	0	2	81	3.428631e-15	1.188214e-14
$r+z$	$-\frac{\sigma}{r}$	1	82	3.336932e-15	3.647264e-04
$2r+2z$	$-2\frac{\sigma}{r}$	1	82	3.336932e-15	3.647264e-04
rz	$-\frac{\sigma z}{r}$	1	85	2.989073e-15	6.598747e-04
r^2	-4σ	1	87	2.674403e-15	1.552825e-04
z^2	-2σ	1	85	2.892919e-15	1.855562e-03
r^2+z^2	-6σ	1	79	2.894108e-15	1.053531e-03
r^2z^2	$-2\sigma(2z^2-r^2)$	1	79	2.807637e-15	2.995912e-01
r^3	$-9r$	1	92	2.598646e-15	2.942591e-04

Проверим порядок сходимости решения, для этого на каждой итерации мы будем удваивать количество узлов в одномерной сетке по r. В качестве искомой функции возьмем $u = r^3$. Из таблицы 2 видно, что для дробления сетки по r имеем второй порядок сходимости решения.

Таблица 2 – Тестирование порядка сходимости для $u = r^3$

Номер итерации	Количество узлов	Количество итераций	Относительная невязка	Норма относительной погрешности	Отношение погрешностей между предыдущим и текущим шагом
1	3721	92	2.598646e-15	2.942591e-04	
2	7381	158	3.411847e-15	7.355847e-05	4.00
3	14701	237	5.419534e-15	1.838842e-05	4.00
4	29341	329	8.888586e-15	4.596958e-06	4.00
5	58621	405	1.365300e-14	1.149234e-06	4.00
6	117181	460	2.063558e-14	2.873130e-07	4.00
7	234301	483	2.994951e-14	7.183033e-08	4.00

3.1.2. ТЕСТИРОВАНИЕ РЕШЕНИЯ ТРЕХМЕРНОЙ ЗАДАЧИ

Для тестирования решения трехмерной задачи будет использоваться равномерная сетка. Одномерные сетки x и y с границами [-15,15] и равномерным шагом равным 3. Одномерная сетка z задана границами [0,30] и равномерным шагом равным 3. Всего узлов 1331. Из таблицы 3 видим, что решение точное для полиномиальных функций первого порядка, то есть совпадает с порядком выбранных базисных функций.

Также проведем тест на порядок сходимости решения. В качестве искомых функций будем рассматривать $u = x^2$, $u = y^2$, $u = z^2$. Вновь на каждой итерации будем удваивать количество узлов для соответствующей одномерной сетки. Из таблиц 4-6 видим второй порядок сходимости для трёхмерной задачи, что удовлетворяет постановке.

Таблица 3 – Результаты тестирования трёхмерной задачи на полиномах

Искомая функция u	**Правая часть дифференциального уравнения f**	**Параметр σ среды**	**Количество итераций**	**Относительная невязка**	**Норма относительной погрешности**
10	0	1	19	8.217068e-16	1.822835e-16
x	0	1	29	6.195568e-16	2.229974e-16
y	0	1	29	5.961269e-16	1.568617e-16
z	0	1	19	7.188952e-16	1.932452e-16
x+y+z	0	1	28	6.939795e-16	1.730760e-16
x+y+z	0	2	28	7.635412e-16	2.304664e-16
$xy + yz + zx$	0	1	21	6.214529e-16	8.598783e-17
xyz	0	1	30	3.337471e-16	1.188077e-16
x^2	-2λ	1	20	5.358223e-16	1.960128e-01
y^2	-2λ	1	20	5.667160e-16	1.960128e-01
z^2	-2λ	1	21	6.859797e-16	2.111839e-02
$x^2 + y^2 + z^2$	-6λ	1	20	6.037270e-16	7.764688e-02
$x^2y^2 + y^2z^2 + z^2x^2$	$-2\lambda(x^2 + y^2 + z^2)$	1	19	5.263560e-16	8.090810e-02

Таблица 4 – Тестирование порядка сходимости с дроблением x, $u = x^2$

Номер итерации	**Количество узлов**	**Количество итераций**	**Относительная невязка**	**Норма относительной погрешности**	**Отношение погрешностей между предыдущим и текущим шагом**
1	1331	20	5.358223e-16	1.960128e-01	
2	2541	23	7.789277e-16	3.735391e-01	0.52
3	4961	27	1.221346e-15	1.663876e-01	2.24
4	9801	31	1.561622e-15	4.798749e-02	3.47
5	19481	33	1.742003e-15	1.247042e-02	3.85
6	38841	33	1.781546e-15	3.158728e-03	3.95
7	77561	33	1.788276e-15	7.938272e-04	3.98
8	155001	33	1.786503e-15	1.989171e-04	3.99
9	309881	33	1.774972e-15	4.978335e-05	4.00

Таблица 5 – Тестирование порядка сходимости с дроблением y, $u = y^2$

Номер итерации	**Количество узлов**	**Количество итераций**	**Относительная невязка**	**Норма относительной погрешности**	**Отношение погрешностей между предыдущим и текущим шагом**
1	1331	20	5.358223e-16	1.960128e-01	
2	2541	23	7.789277e-16	3.735391e-01	0.52
3	4961	27	1.221346e-15	1.663876e-01	2.24
4	9801	31	1.561622e-15	4.798749e-02	3.47
5	19481	33	1.742003e-15	1.247042e-02	3.85
6	38841	33	1.781546e-15	3.158728e-03	3.95
7	77561	33	1.788276e-15	7.938272e-04	3.98
8	155001	31	1.786503e-15	1.989171e-04	3.99
9	309881	32	1.780593e-15	4.978335e-05	4.00

Таблица 6 – Тестирование порядка сходимости с дроблением z, $u = z^2$

Номер итерации	**Количество узлов**	**Количество итераций**	**Относительная невязка**	**Норма относительной погрешности**	**Отношение погрешностей между предыдущим и текущим шагом**
1	1331	20	5.649918e-16	8.661796e-02	
2	2541	22	6.093692e-16	1.874397e-01	0.46
3	4961	25	7.907620e-16	6.540260e-02	2.87
4	9801	31	8.842238e-16	1.766767e-02	3.70
5	19481	34	9.462515e-16	4.526321e-03	3.90
6	38841	34	9.693584e-16	1.142508e-03	3.96
7	77561	35	9.610412e-16	2.868443e-04	3.98
8	155001	35	9.678764e-16	7.185460e-05	3.99
9	309881	32	9.700570e-16	1.798107e-05	4.00

3.2. ТЕСТ НА ЗАДАЧЕ С ТОКОМ

Для случая, когда электрическое поле порождается точечным источником в однородной среде, можно проверить решение, сравнив полученный потенциал с аналитическим выражением:

$$V = \frac{I}{2\pi r\sigma}, \tag{32}$$

где r – расстояние до источника.

Для тестов будем использовать однородную среду с проводимостью $\sigma = 0.01$.

3.2.1. ТЕСТИРОВАНИЕ РЕШЕНИЯ ДВУМЕРНОЙ ЗАДАЧИ

Для тестирования решения двумерной задачи мы будем использовать разряжающуюся сетку. Для одномерных сеток по r и z будем использовать одинаковые параметры: начальный шаг равный 1 мм, границы равные 100 км и коэффициент разрядки равный 1.05. Всего узлов – 101124. Сравним полученное решение с аналитическим на интервалах от 0 до 10 м и от 900 м до 1100 м.

На рисунках 2 и 3 видим, что численное решение совпадает с аналитическим.

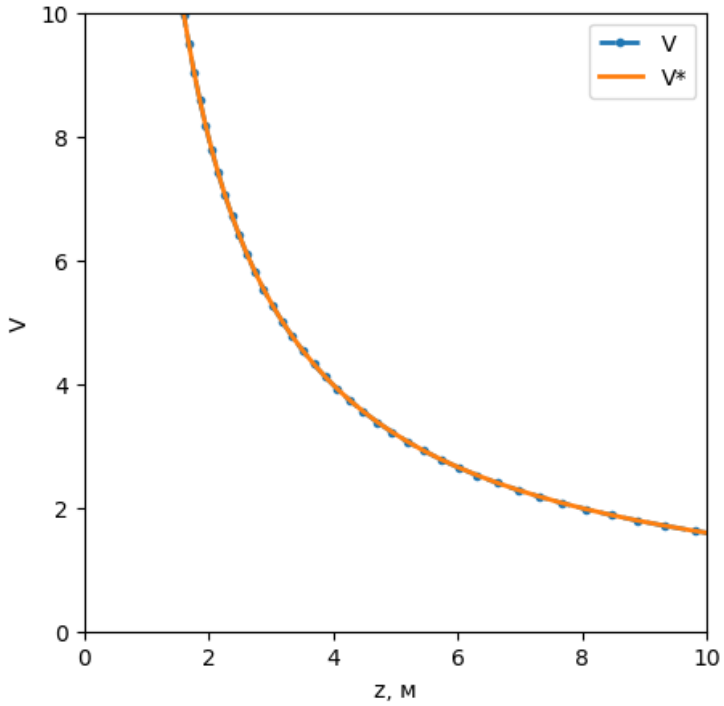


Рисунок 2 – Решение двумерной задачи с током, первый интервал. V – полученное решение, V* – аналитическое решение

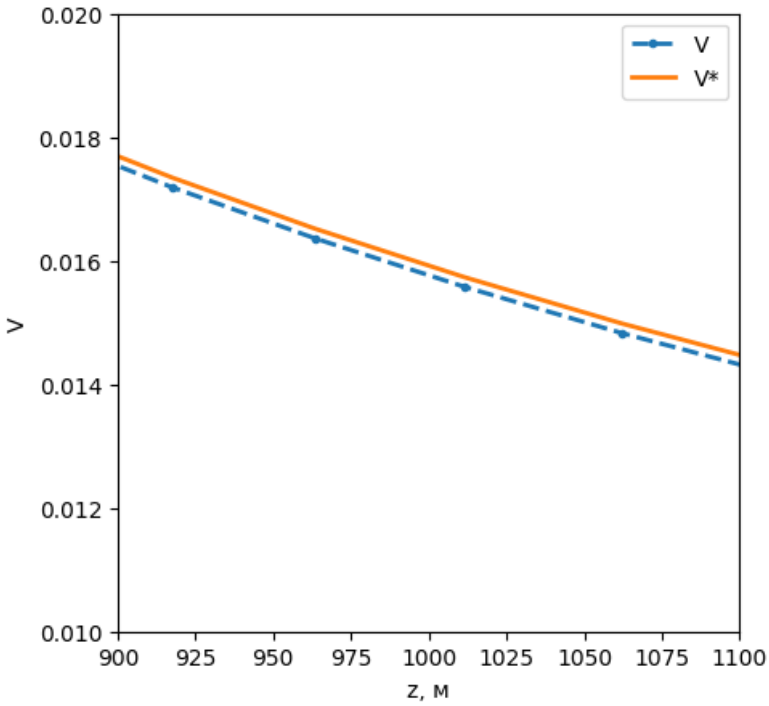


Рисунок 3 – Решение двумерной задачи с током, второй интервал

Относительная погрешность на интервале до 1 км не превышает 1% (рисунок 4).

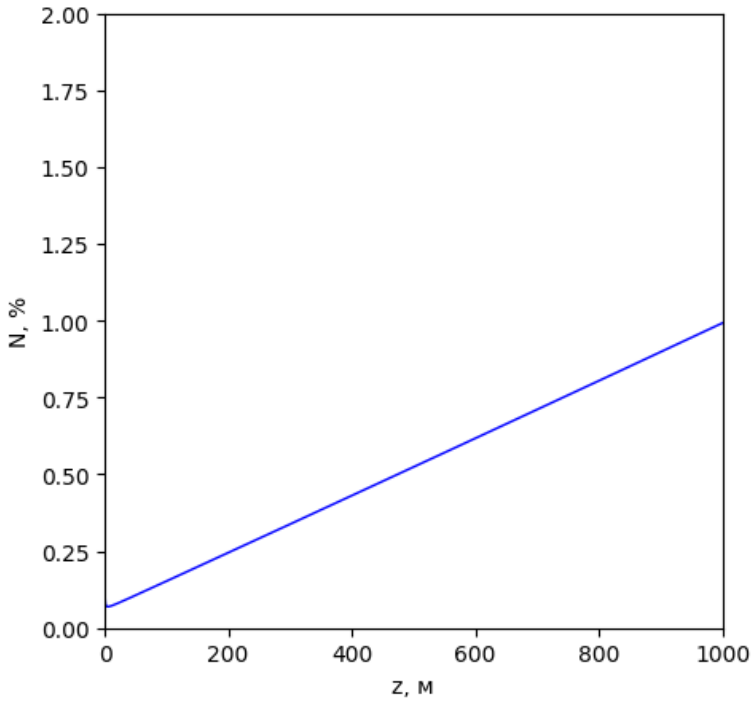


Рисунок 4 – Относительная погрешность решения.

3.2.2. ТЕСТИРОВАНИЕ РЕШЕНИЯ ТРЁХМЕРНОЙ ЗАДАЧИ

Для тестирования решения трёхмерной задачи мы будем использовать разряжающуюся сетку. Для одномерных сеток будем использовать одинаковые параметры: начальный шаг равный 1 мм, границы равные 100 км и коэффициент разрядки равный 1.5. Всего узлов – 372287. Сравним полученное решение с аналитическим на интервалах от 0 до 10 м и от 900 м до 1100 м.

На рисунках 5 и 6 видим, что численное решение совпадает с аналитическим.

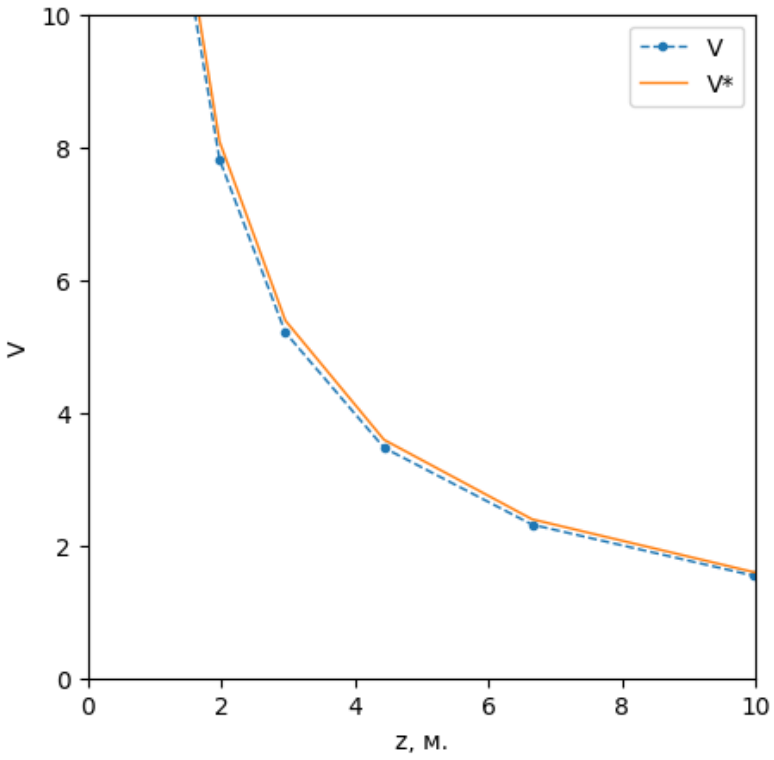


Рисунок 5 – Решение трёхмерной задачи с током, первый интервал. V – полученное решение, V* – аналитическое решение

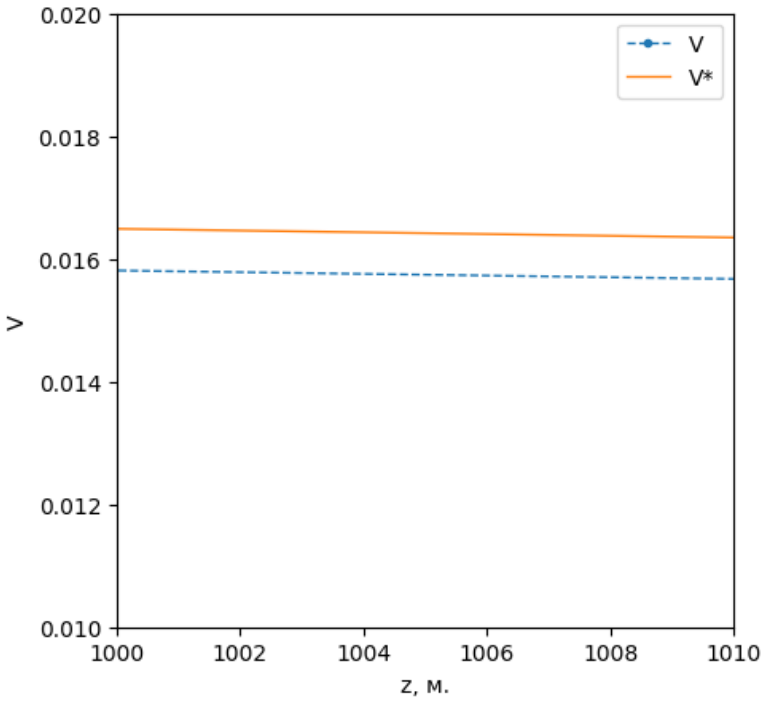


Рисунок 6 – Решение трёхмерной задачи с током, второй интервал

Убеждаемся, что решение совпадает с аналитическим. Относительная погрешность на интервале до 1 км не превысила 4,5% (рисунок 7).

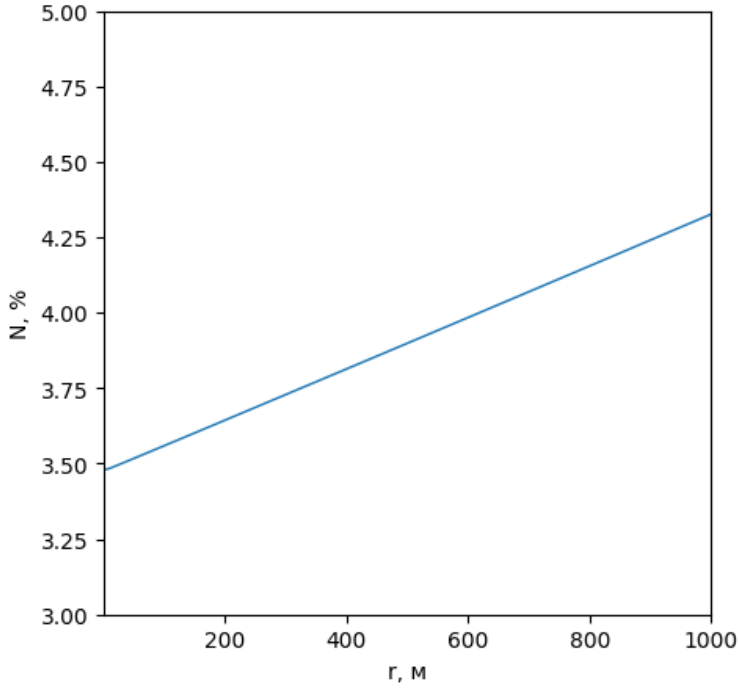


Рисунок 7 – Относительная погрешность решения

4. ИССЛЕДОВАНИЯ

Сравним решения для каждого способа учёта нормального поля и их точность вдоль линии $x = 0, y = 0$.

4.1. ИССЛЕДОВАНИЕ 1. ОДНОРОДНАЯ СРЕДА

Для первого исследования построим область следующим образом. Среда будет представлять три слоя: $h_1 = 10, h_2 = 10, h_3 = \infty$. Для этих слоев зададим следующие параметры проводимости $\sigma_1 = 0.01, \sigma_2 = 0.0001, \sigma_3 = 0.01$. Во втором слое поместим замещающий его элемент с проводимостью $\sigma^A = 0.01$. Таким образом, получим однородную среду с параметром проводимости $\sigma = 0.01$. В качестве точного решения будет использовано аналитическое решение.

На рисунках 8-12: Vn – решение соответствующей двумерной задачи, Va – решение задачи на добавочное поле, V – полученное численное решение, V* – аналитическое решение.

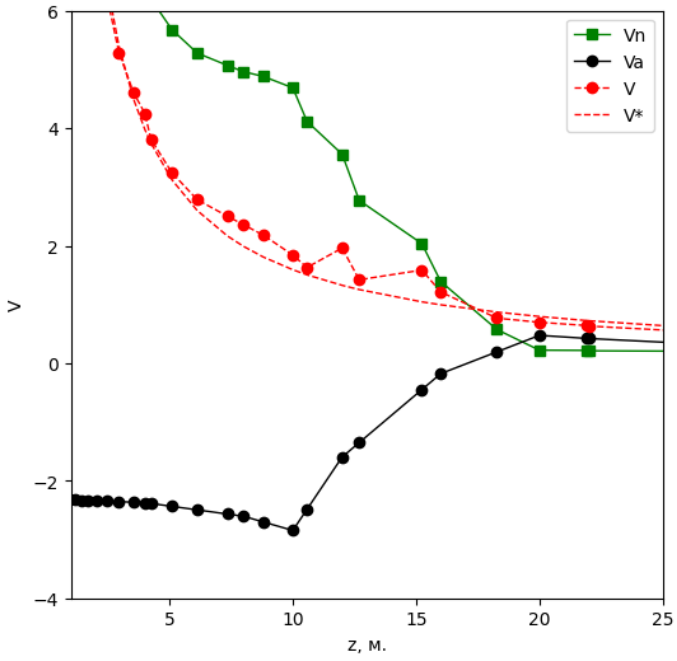


Рисунок 8 – График решения **первым** способом

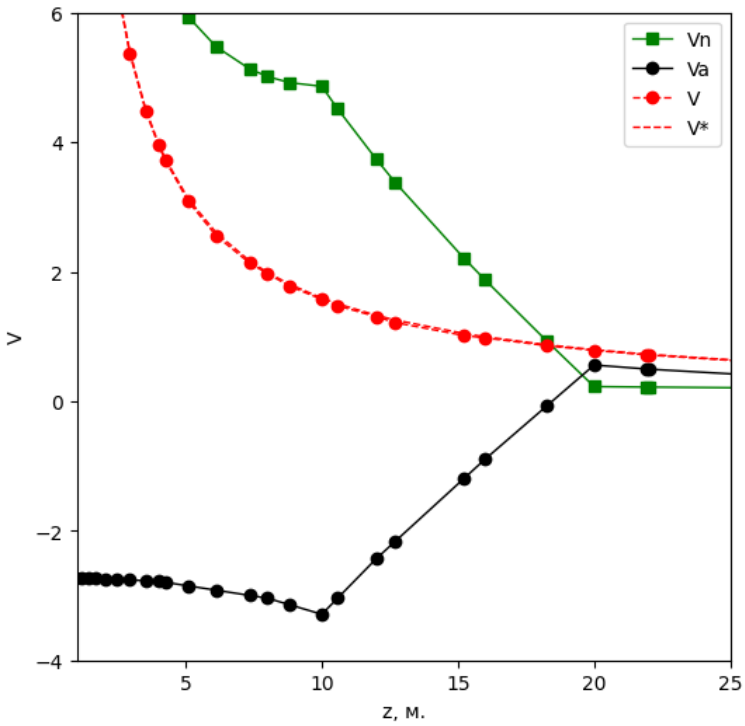


Рисунок 9 – График решения **вторым** способом

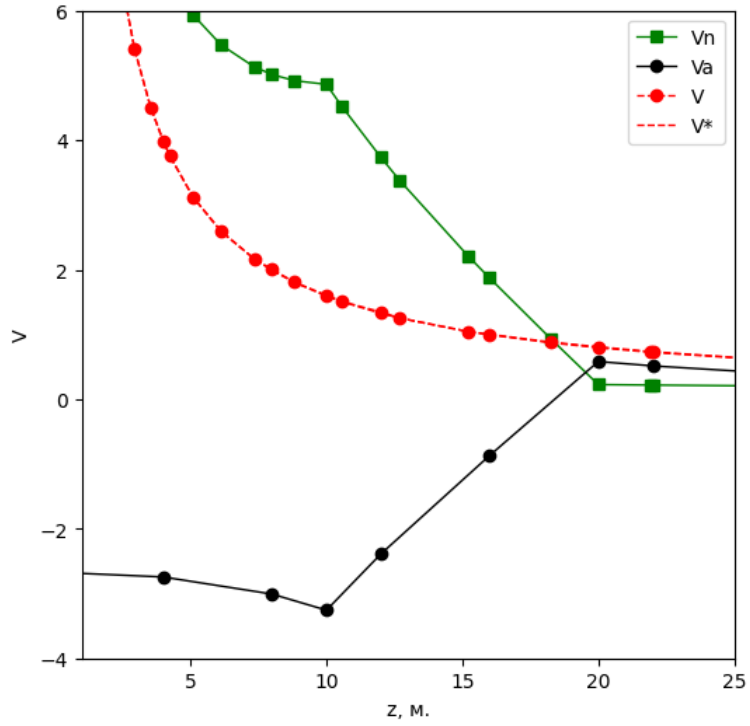


Рисунок 10 – График решения **третьим** способом

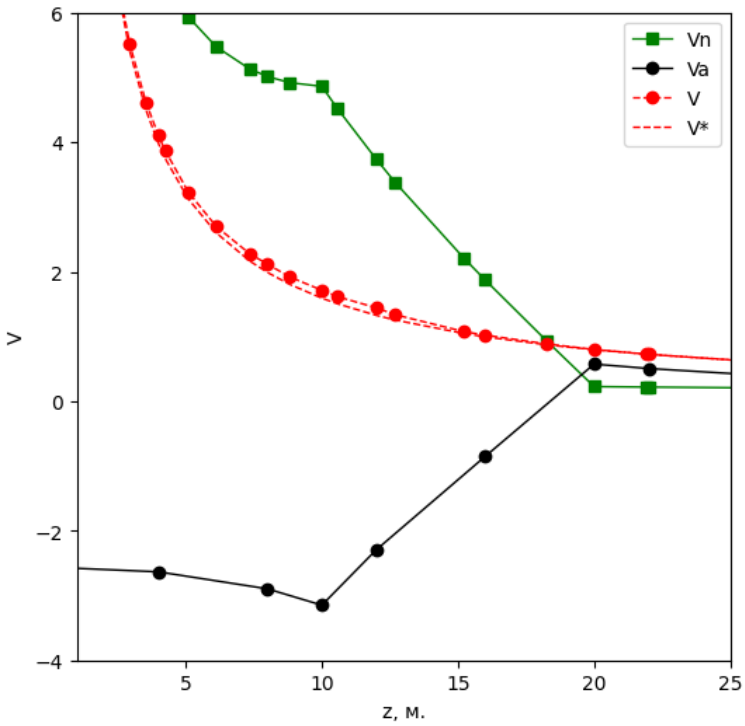


Рисунок 11 – График решения **четвертым** способом

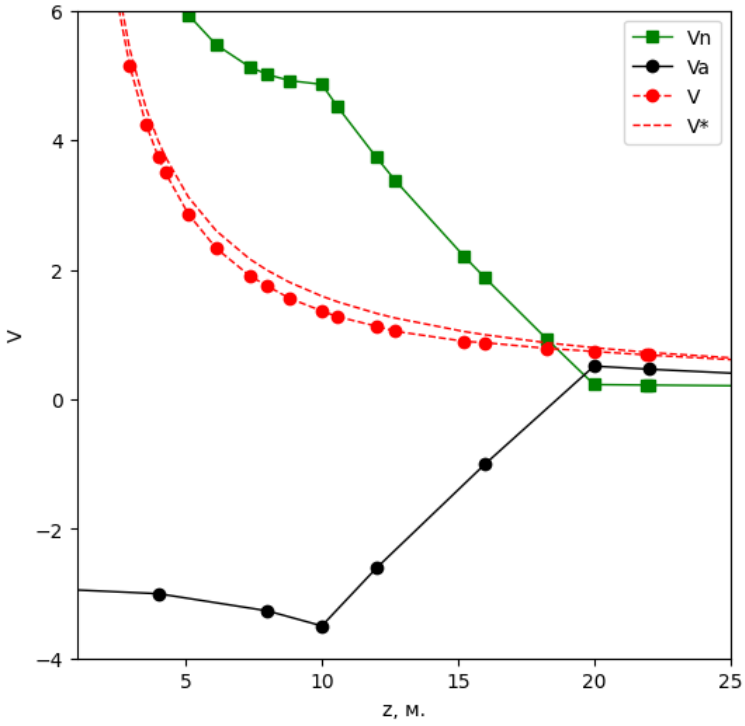


Рисунок 12 – График решения **пятым** способом

На рисунках 13 и 14: V1 – решение с первым способом выделения части поля (формирование правой части СЛАУ умножением матрицы жесткости на вектор значения нормального поля **в центре элемента**), V2 – второй способ

(матрица жесткости умножается на вектор значений нормального поля **в узлах элемента**), V3 – третий способ (численное интегрирование методом **Гаусса**), V4 – четвёртый способ ($gradV^N$ для элемента считается в его центре), V5 – пятый способ ($gradV^N$ для элемента считается в его узлах), V* – аналитическое решение.

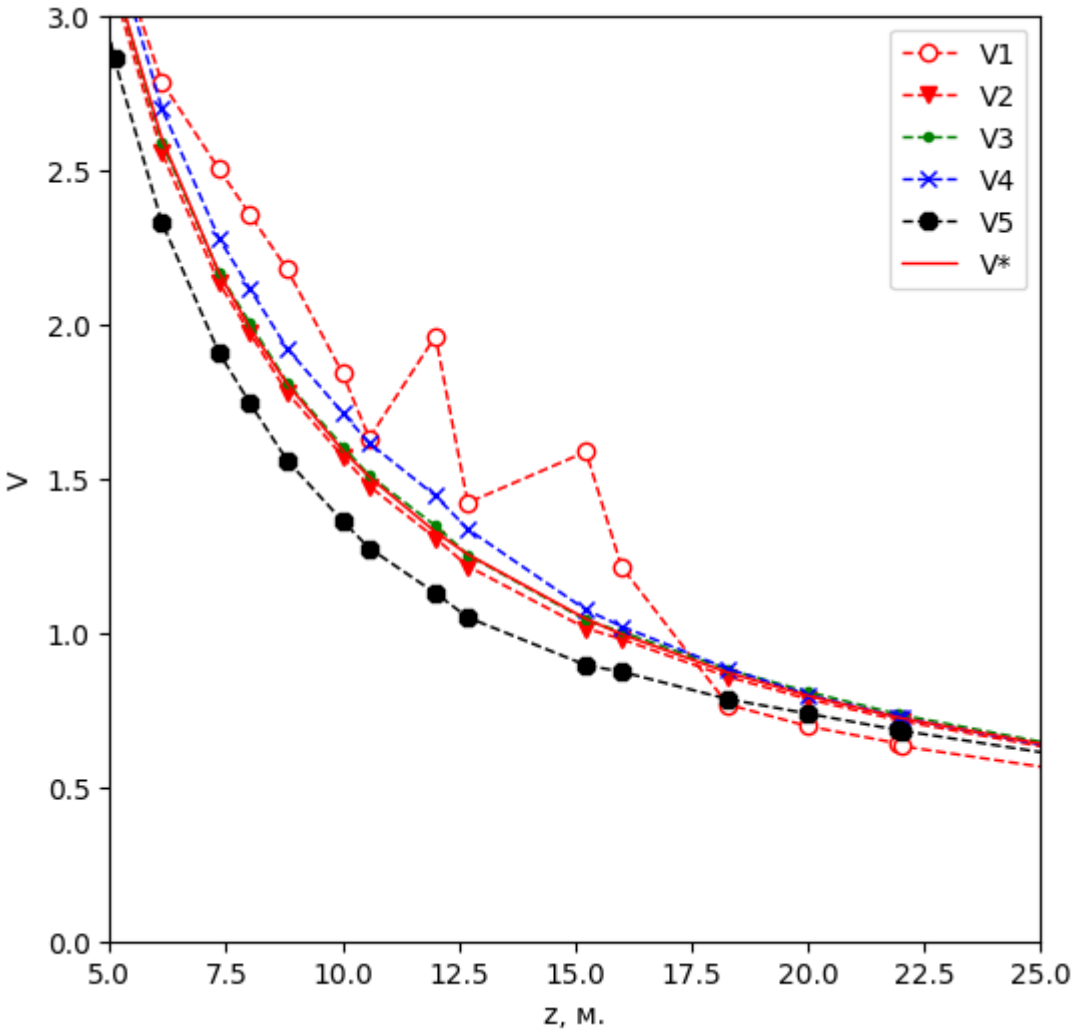


Рисунок 13 – Сравнение решений

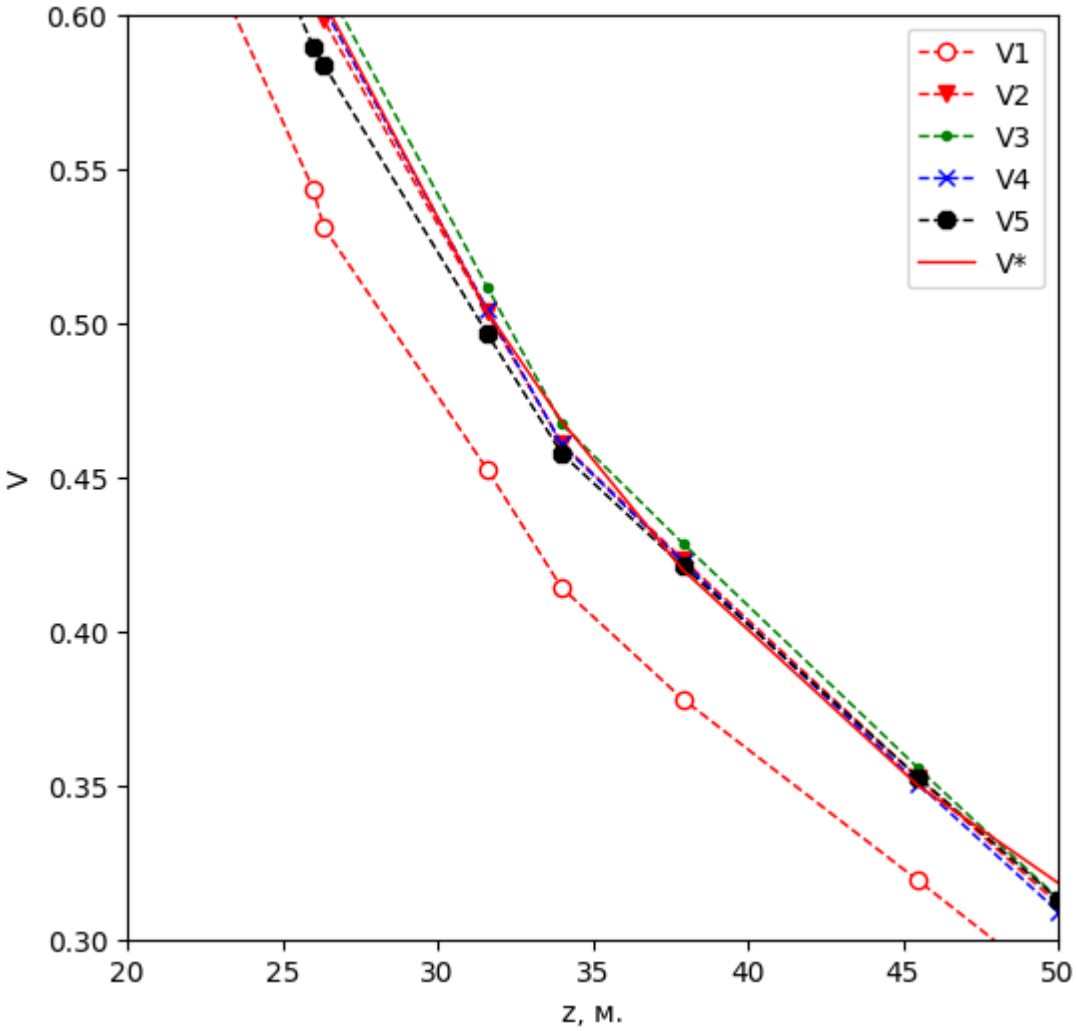


Рисунок 14 – Сравнение решений, крупный масштаб

Посчитаем относительную погрешность решения на поверхности. Будем рассматривать линию z = 0, y = 0 и x в интервале от 0 до 100 м (рисунок 15).

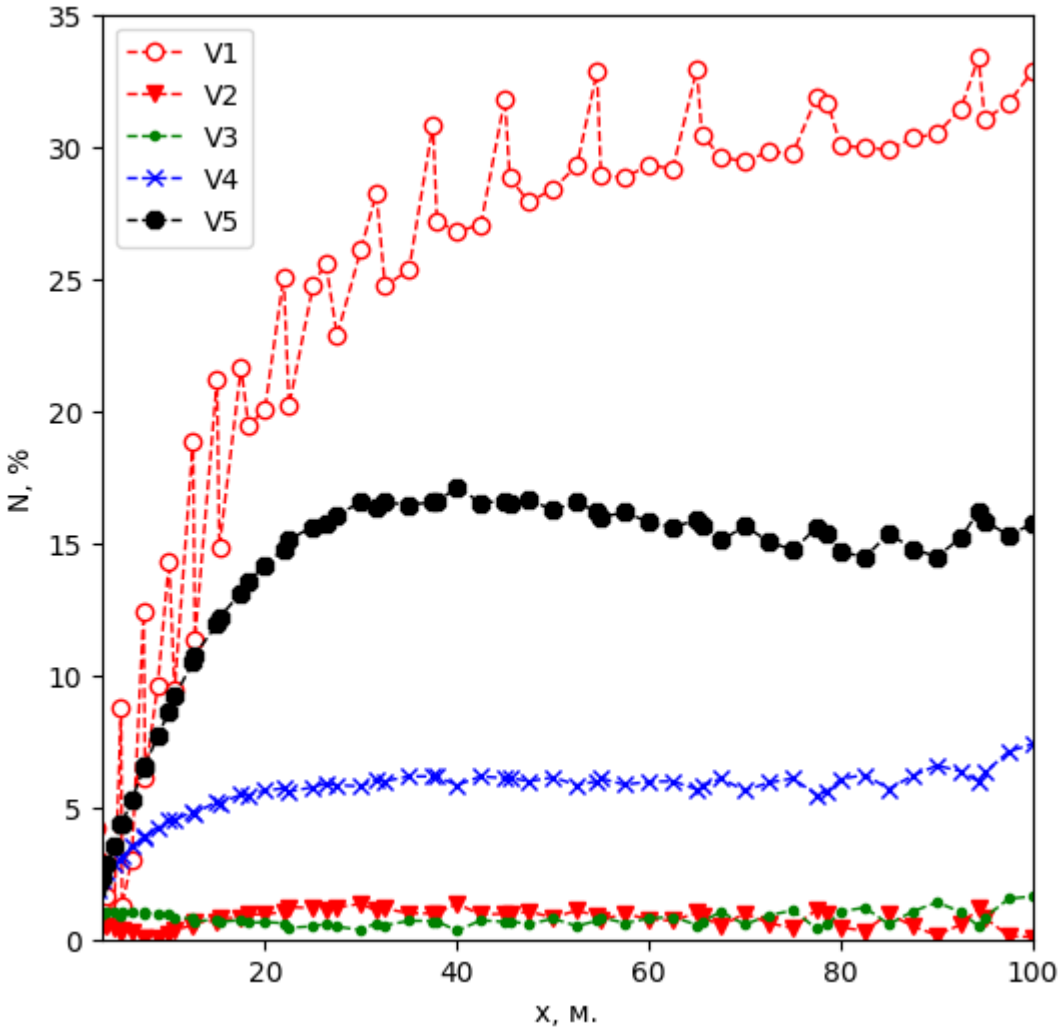


Рисунок 15 – Относительная погрешность решений

Посчитаем норму относительной погрешности (таблица 7) для каждого из решений по формуле

$$\sqrt{\sum_{i=1}^{n} \frac{\left|V_i - V_i^*\right|}{\left|V_i\right|}}, \tag{33}$$

где n – количество узлов, V_i – численное решение в i-ом узле, V^* – точное решение в i-ом узле.

Лучшим оказался метод Гаусса. Следующий по точности, хотя практически не уступающий Гауссу, метод: формирование правой части умножением локальной матрицы жесткости на вектор значений нормального поля.

Вычисление производной в центре третий по точности результат с отличием от Гаусса в 7.7 раз.

Методы с формированием правой части умножением матрицы жёсткости на значение нормального поля в центре и подсчётом градиента в узлах элемента оказались наихудшими с отличием от Гаусса в 36 и 20 раз соответственно.

Таблица 7 – Норма относительной погрешности решений

	q в центре	q в узлах	$gradV^N$ в точках Гаусса	$gradV^N$ в центре	$gradV^N$ в узлах
Норма относительной погрешности, %	188.1	6.23	5.26	40.52	105.09

4.2. ИССЛЕДОВАНИЕ 2. ОДИН СЛОЙ

Для второго исследования построим область следующим образом. В среде с проводимостью $\sigma = 0.01$, будет находится слой на глубине 30 м, толщиной 40 м с проводимостью $\sigma^A = 0.0001$. В качестве точного решения будет использовано соответствующее решение двумерной задачи на подробной сетке.

На рисунках 16-20: Vn – решение соответствующей двумерной задачи, Va – решение задачи на добавочное поле, V – полученное численное решение, V* – решение двумерной задачи на подробной сетке.

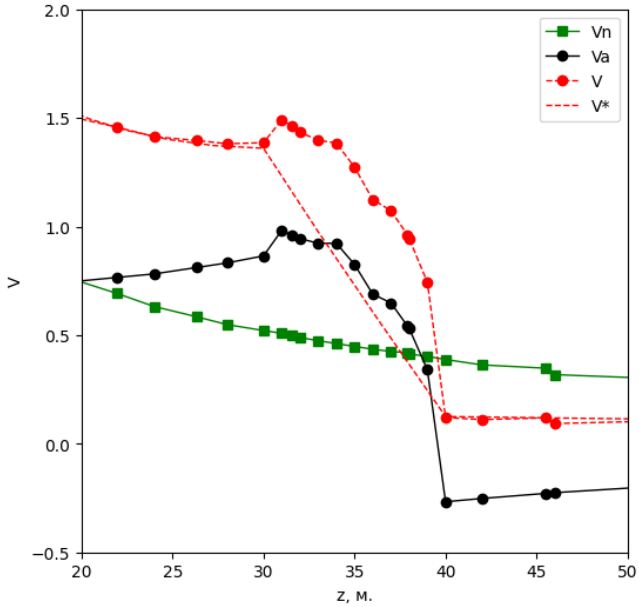


Рисунок 16 – График решения **первым** способом

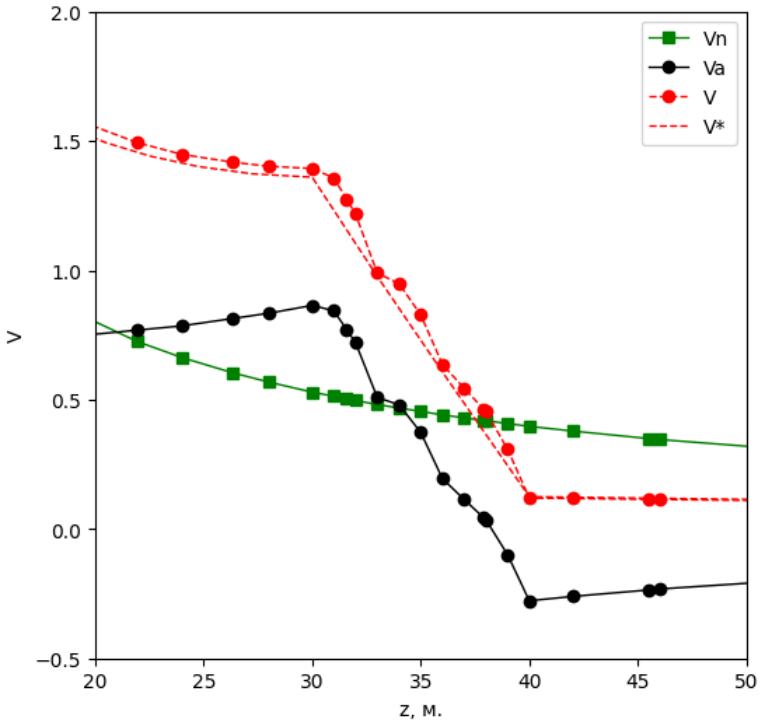


Рисунок 17 – График решения **вторым** способом

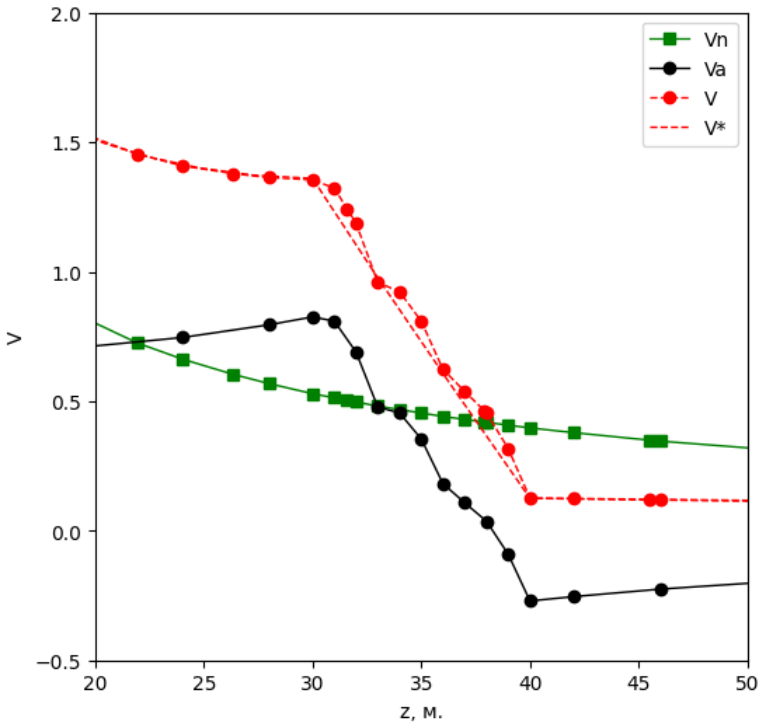


Рисунок 18 – График решения **третьим** способом

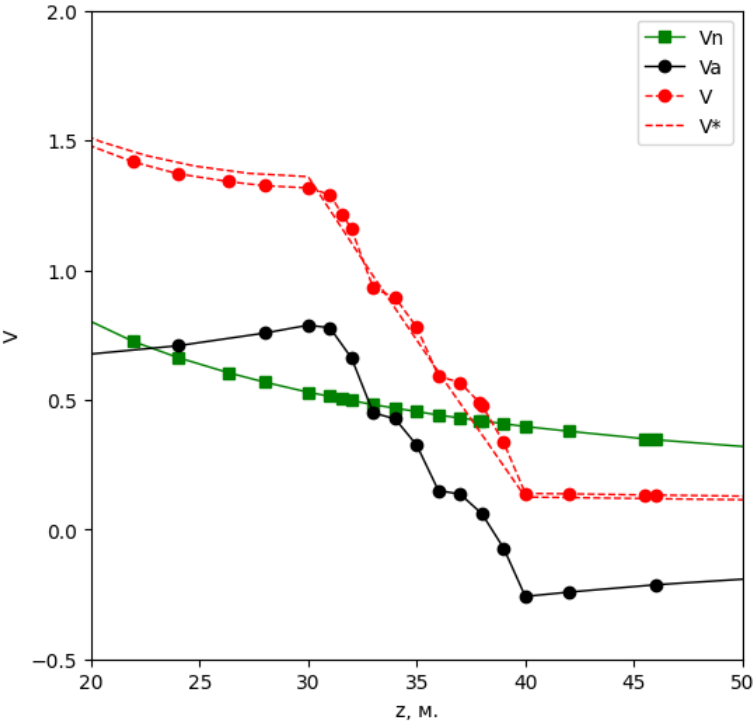


Рисунок 19 – График решения **четвертым** способом

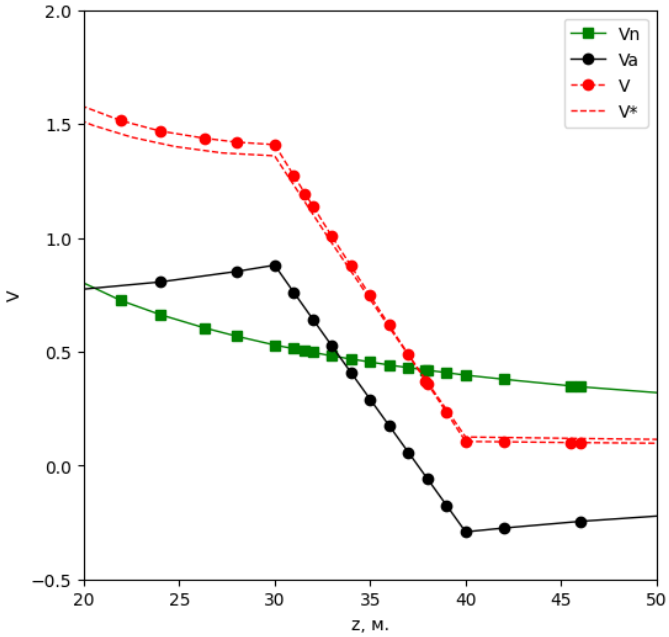


Рисунок 20 – График решения **пятым** способом

На рисунках 21 и 22: V1 – решение с первым способом выделения части поля (формирование правой части СЛАУ умножением матрицы жесткости на вектор значения нормального поля **в центре элемента**), V2 – второй способ (матрица жесткости умножается на вектор значений нормального поля **в узлах элемента**), V3 – третий способ (численное интегрирование методом **Гаусса**), V4 – четвёртый способ ($gradV^N$ для элемента считается в его центре), V5 – пятый способ ($gradV^N$ для элемента считается в его узлах), V* – аналитическое решение.

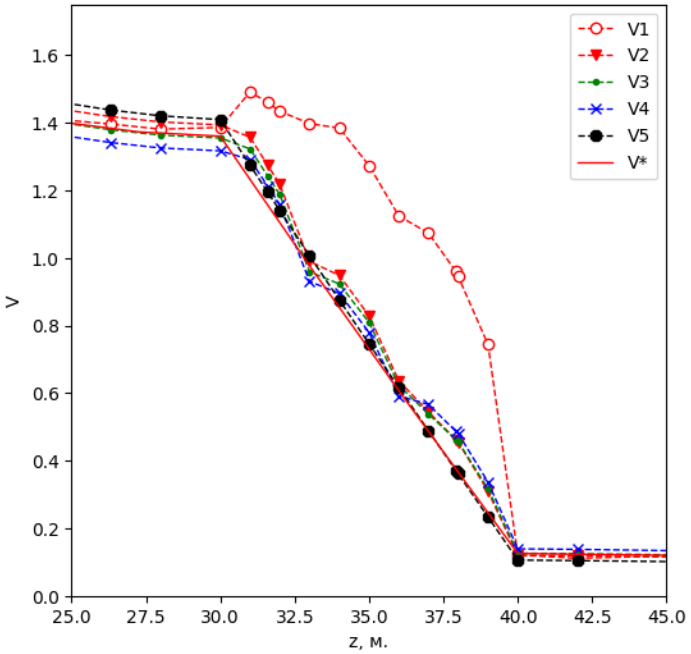


Рисунок 21 – Сравнение решений

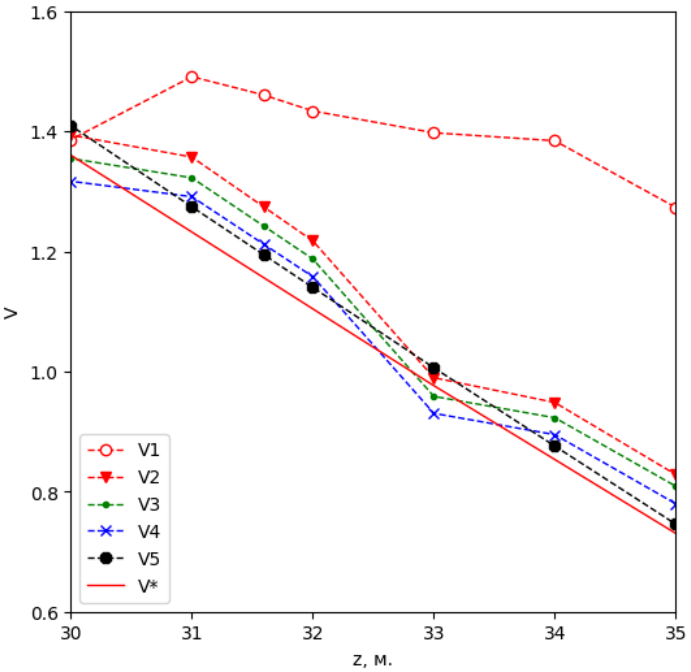


Рисунок 22 – Сравнение решений, крупный масштаб

Посчитаем относительную погрешность решения на поверхности. Будем рассматривать линию z = 0, y = 0 и x в интервале от 0 до 100 м (рисунок 23).

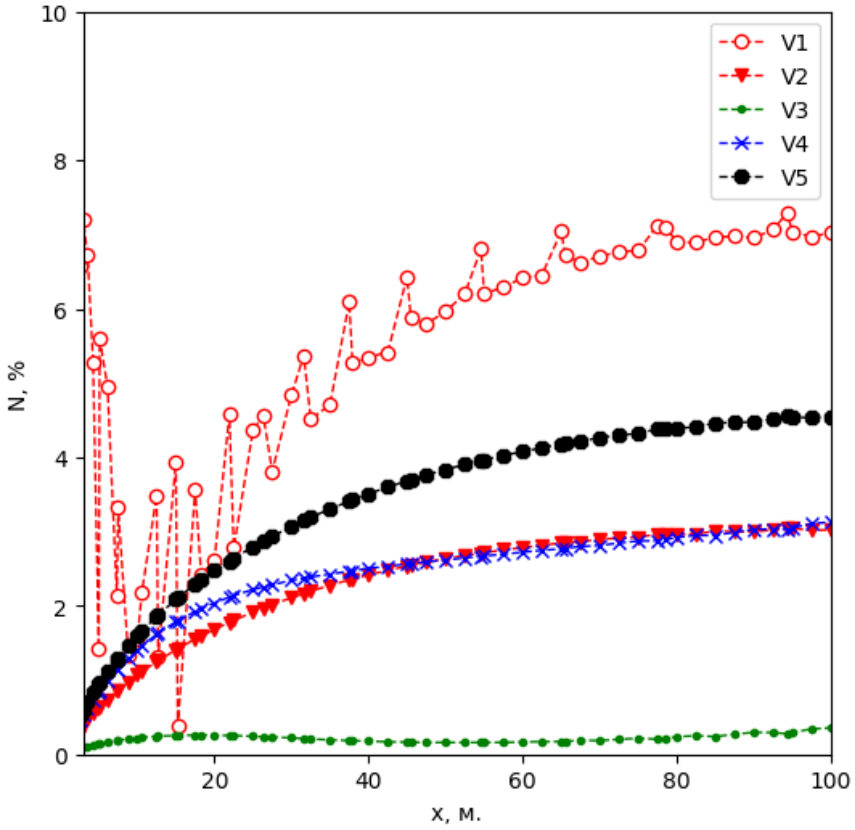


Рисунок 23 – Относительная погрешность решений

Посчитаем норму относительной погрешности для каждого из решений. Как видно из таблицы 8 лучшим методом оказывается Гаусс с разницей со вторым и третьим методом в 11 раз. Отличие с первым и пятым методом в 25 и 16 раз соответственно.

Таблица 8 – Норма относительной погрешности решений

	q в центре	q в узлах	$gradV^N$ в точках Гаусса	$gradV^N$ в центре	$gradV^N$ в узлах
Норма относительной погрешности, %	39.05	17.02	1.53	17.52	25.04

4.3. ИССЛЕДОВАНИЕ 3. ДВА СЛОЯ

Для третьего исследования построим область следующим образом. В среде с проводимостью $\sigma = 0.01$, первый слой будет находится на глубине 30 м,

толщиной 10 м с проводимостью $\sigma^{A_1} = 0.1$, второй слой будет находится на глубине 40 м, толщиной 60 м и с проводимостью $\sigma^{A_1} = 0.001$. В качестве точного решения будет использовано соответствующее решение двумерной задачи на подробной сетке.

На рисунках 24-28: Vn – решение соответствующей двумерной задачи, Va – решение задачи на добавочное поле, V – полученное численное решение, V* – решение соответствующей двумерной задачи на подробной сетке.

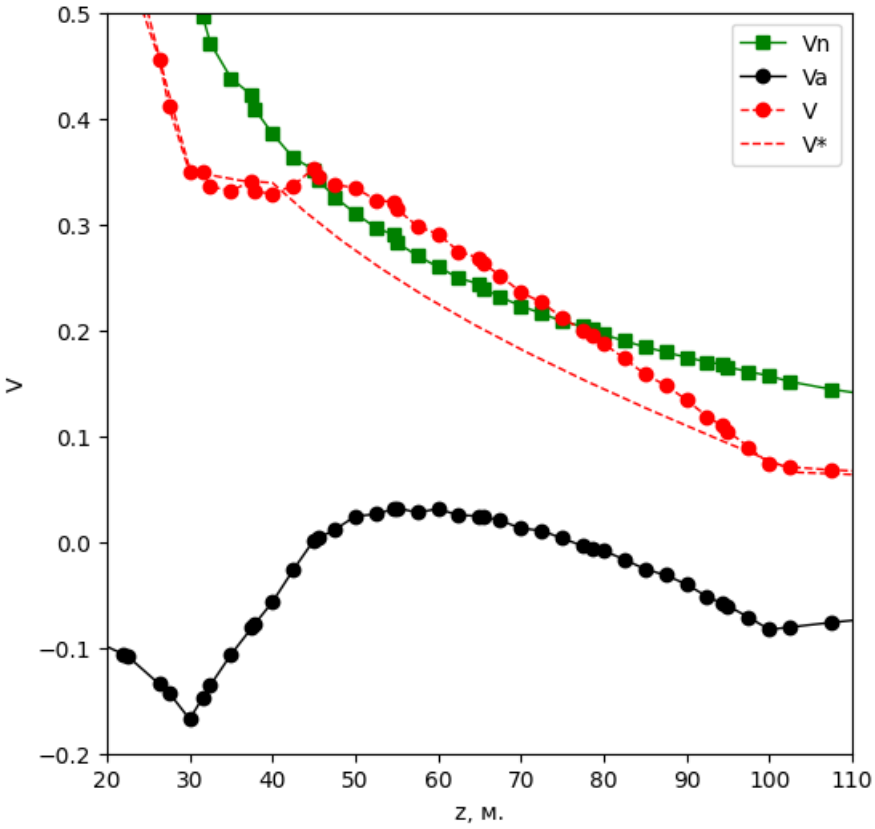


Рисунок 24 – График решения **первым** способом

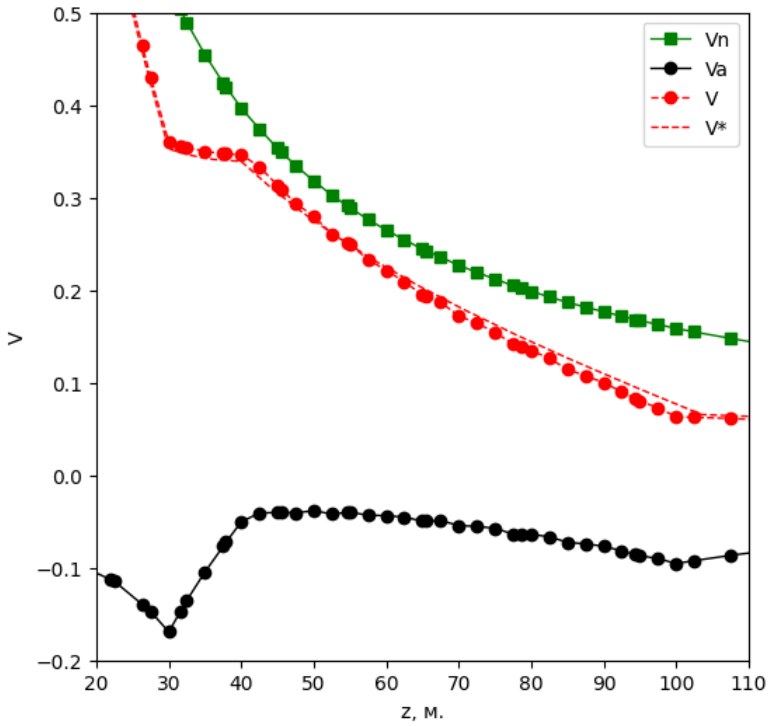


Рисунок 25 – График решения **вторым** способом

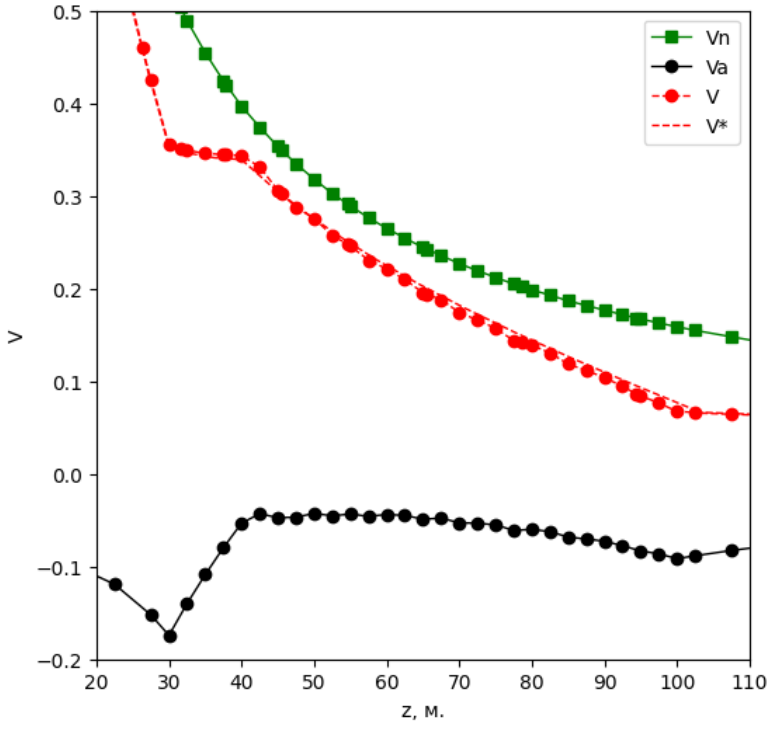


Рисунок 26 – График решения **третьим** способом

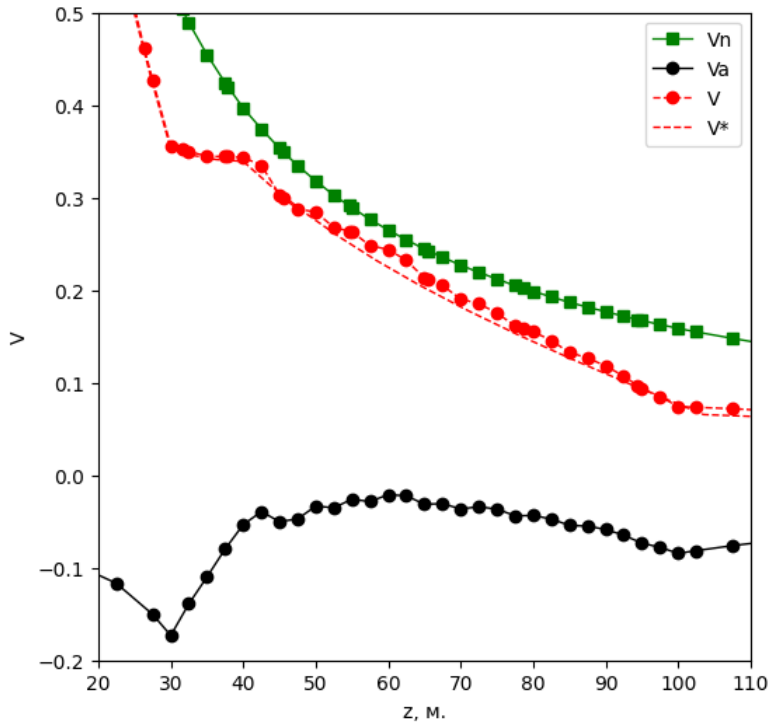


Рисунок 27 – График решения **четвертым** способом

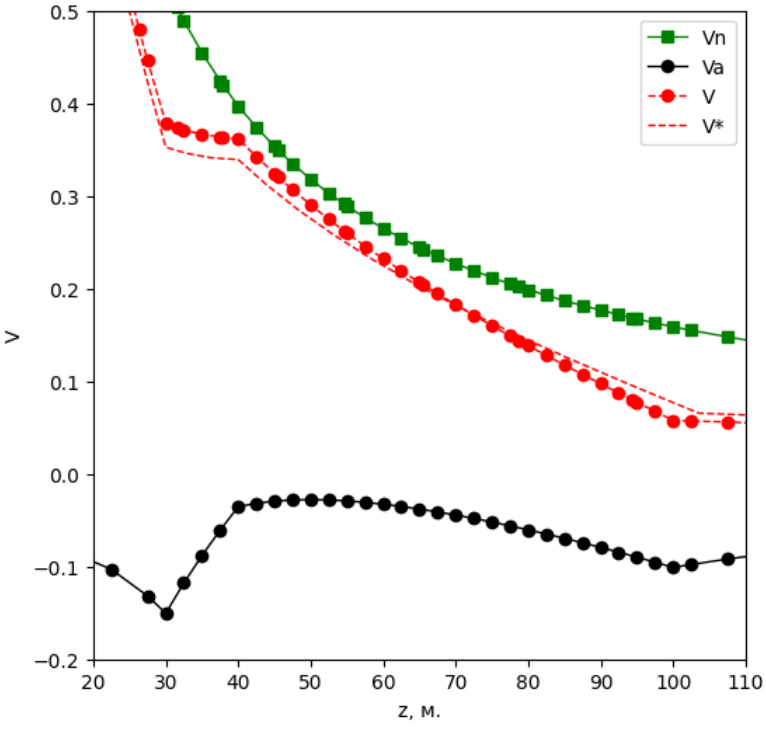


Рисунок 28 – График решения **пятым** способом

На рисунках 29 и 30: V1 – решение с первым способом выделения части поля (формирование правой части СЛАУ умножением матрицы жесткости на вектор значения нормального поля **в центре элемента**), V2 – второй способ

(матрица жесткости умножается на вектор значений нормального поля **в узлах элемента**), V3 – третий способ (численное интегрирование методом **Гаусса**), V4 – четвёртый способ ($gradV^N$ для элемента считается в его центре), V5 – пятый способ ($gradV^N$ для элемента считается в его узлах), V* – аналитическое решение.

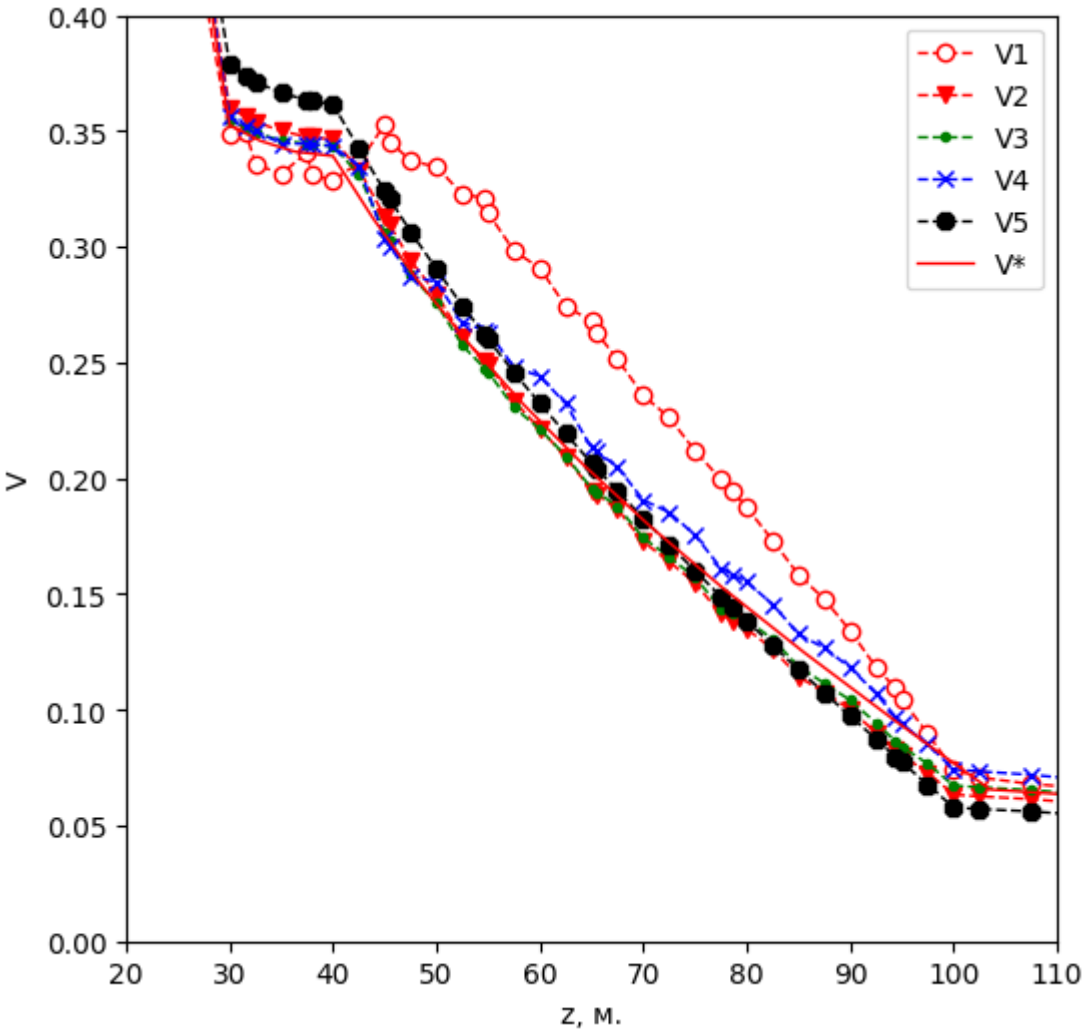


Рисунок 29 – Сравнение решений

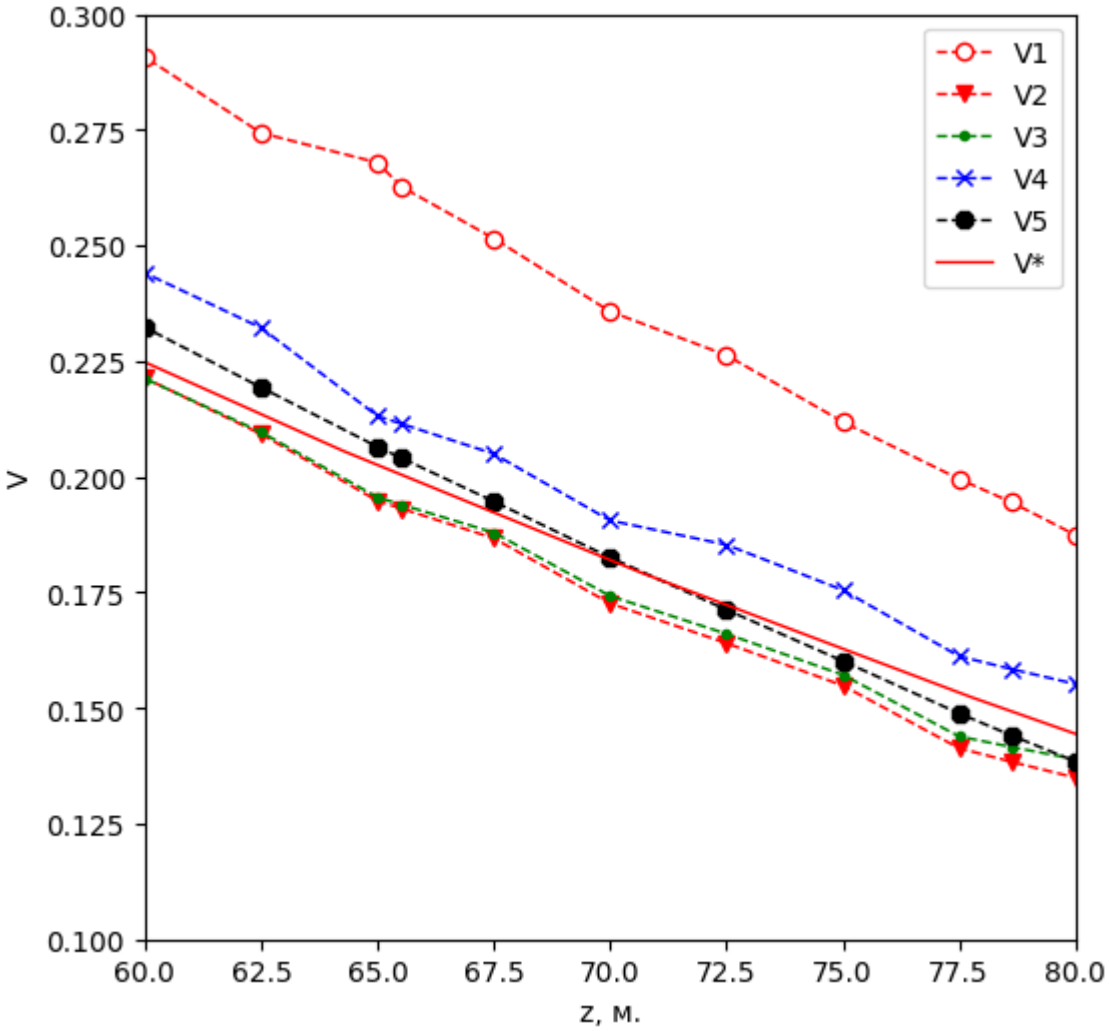


Рисунок 30 – Сравнение решений, крупный масштаб

Посчитаем относительную погрешность решения на поверхности. Будем рассматривать линию z = 0, y = 0 и x в интервале от 0 до 100 м (рисунок 31).

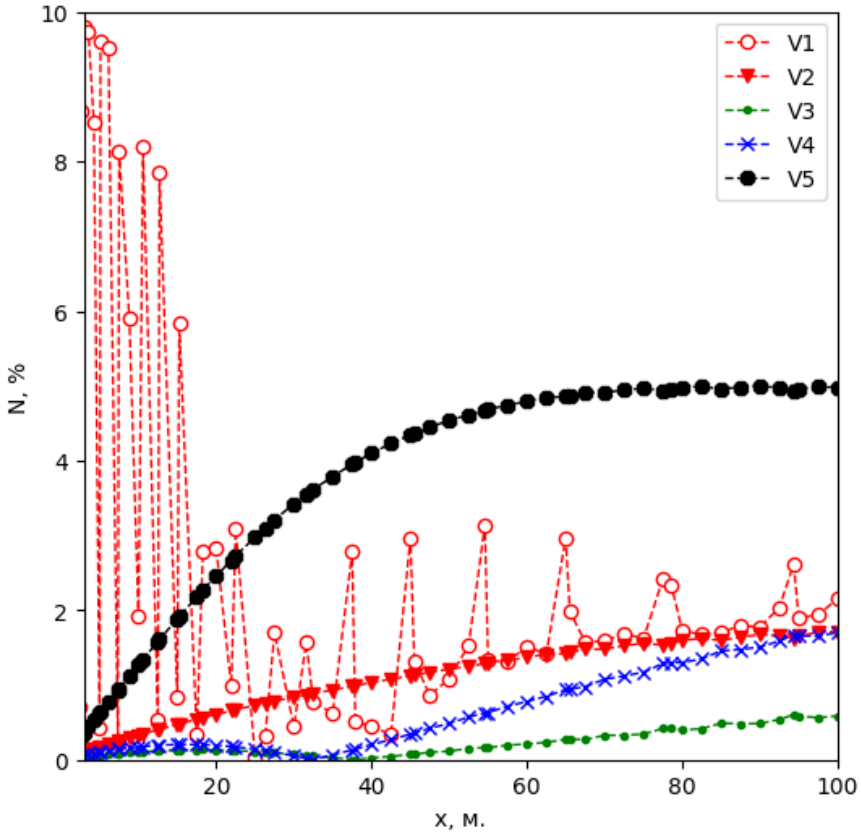


Рисунок 31 – Относительная погрешность решений

Посчитаем норму относительной погрешности для каждого из решений. Из таблицы 9 видно: лучший метод – Гаусс с разницей со вторым и третьим методом в 4.5 и 3 раза соответственно. Отличие с первым и пятым методом в 10 и 15 раз соответственно.

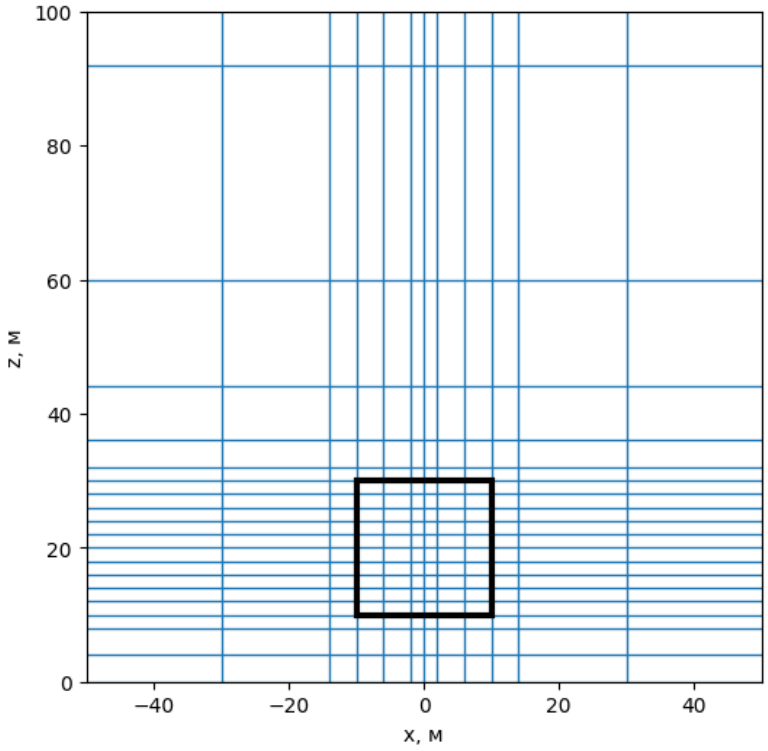
Таблица 9 – Норма относительной погрешности решений

	q в центре	q в узлах	$gradV^N$ в точках Гаусса	$gradV^N$ в центре	$gradV^N$ в узлах
Норма относительной погрешности, %	18.44	8.2	1.84	5.67	28.24

4.4. ИССЛЕДОВАНИЕ 4. ОБЪЕКТ В ОДНОРОДНОЙ СРЕДЕ

Для четвёртого исследования построим область следующим образом. В среде с проводимостью $\sigma = 0.01$, разместим объект на глубине 20 м, со

сторонами 20 м и проводимостью $\sigma^{A_1} = 0.1$. В качестве точного решения будет использовано соответствующее решение трёхмерной задачи на подробной (рисунок 33) сетке третьим способом. Относительно грубая сетка представлена на рисунке 32.



v

Рисунок 32 – Вид сетки с трехмерным объектом, срез среды y = 0, 54450 узлов

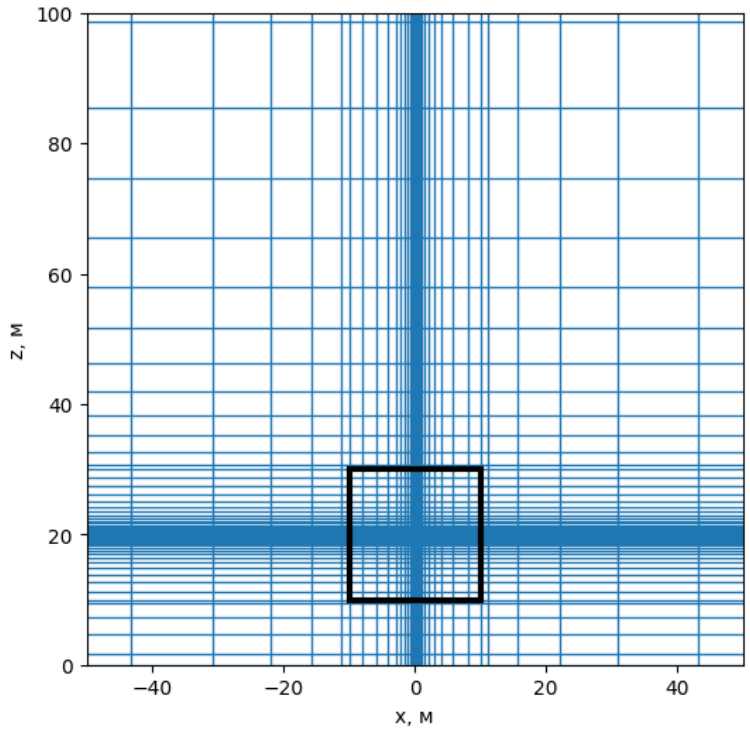


Рисунок 33 – Вид **подробной** сетки с трехмерным объектом, срез среды y = 0, 2203285 узлов

На рисунках 34-38: Vn – решение соответствующей двумерной задачи, Va – решение задачи на добавочное поле, V – полученное численное решение, V* – решение полученное третьим способом на подробной сетке.

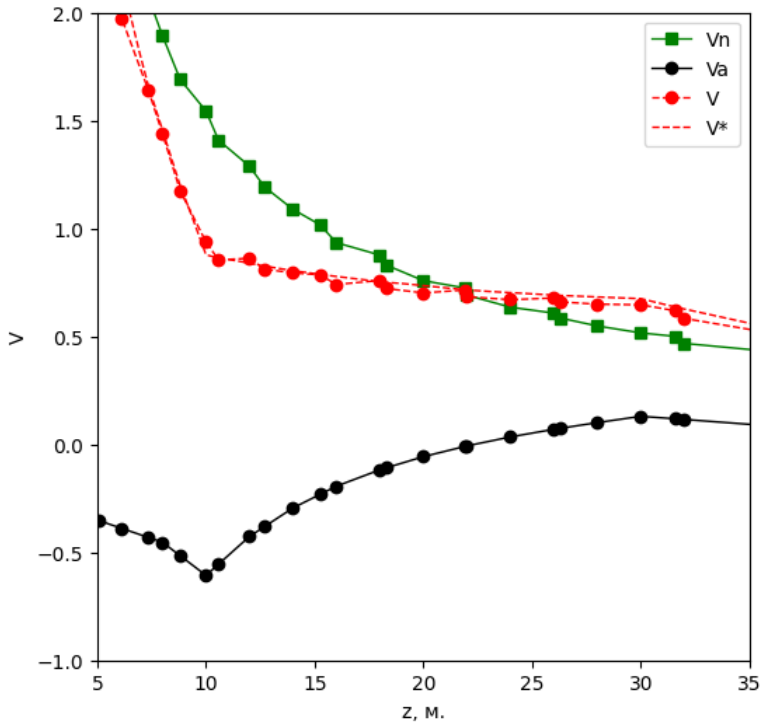


Рисунок 34 – График решения **первым** способом

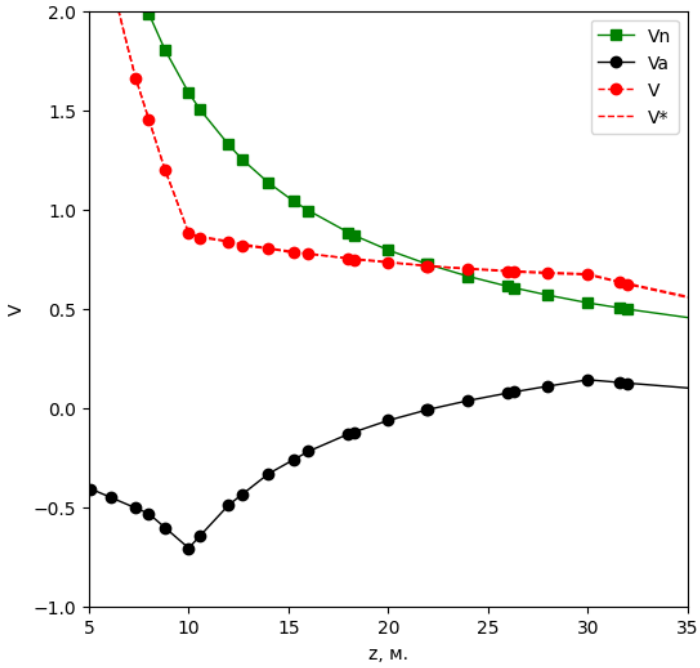


Рисунок 35 – График решения **вторым** способом

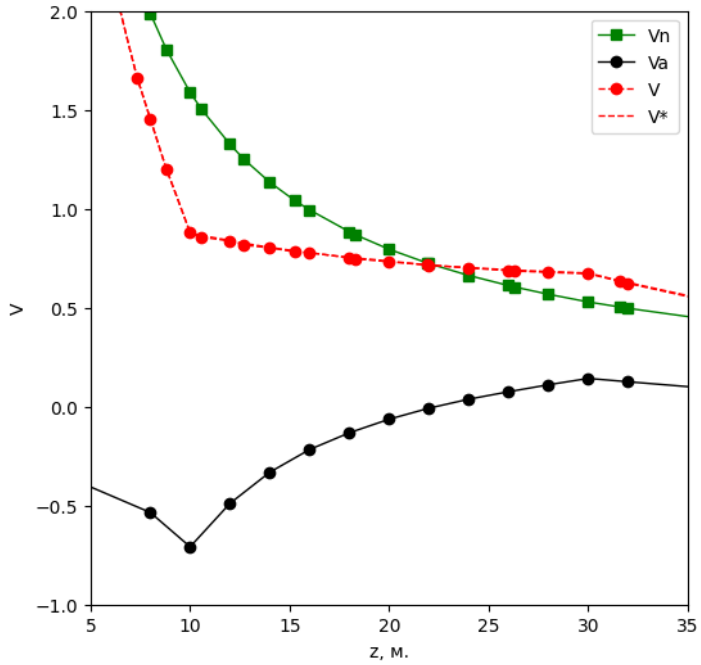


Рисунок 36 – График решения **третьим** способом

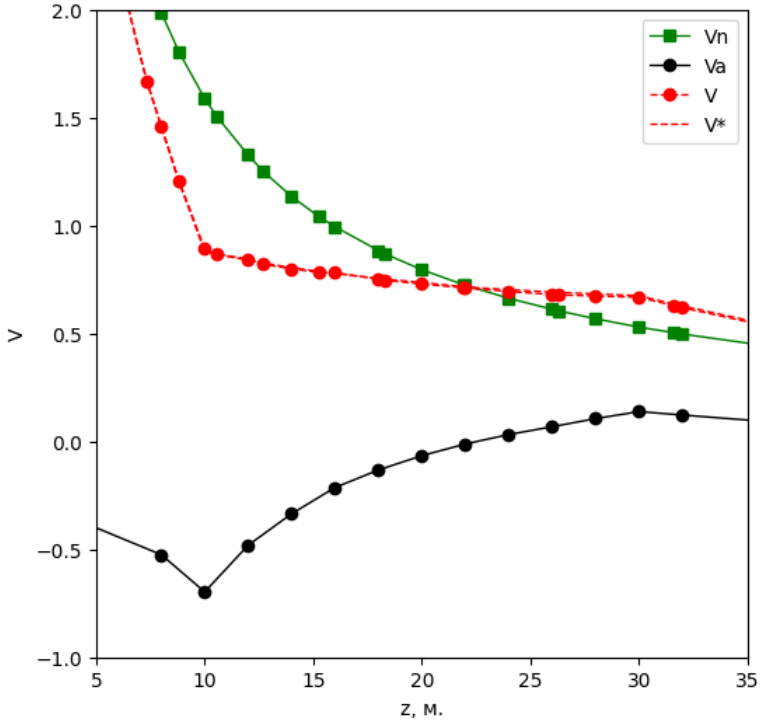


Рисунок 37 – График решения **четвертым** способом

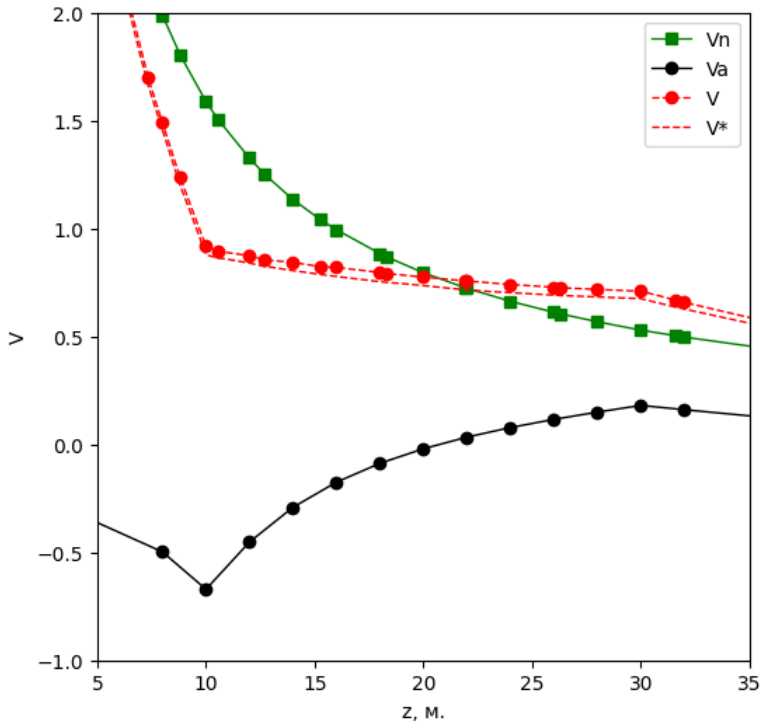


Рисунок 38 – График решения **пятым** способом

На рисунках 39 и 40: V1 – решение с первым способом выделения части поля (формирование правой части СЛАУ умножением матрицы жесткости на вектор значения нормального поля **в центре элемента**), V2 – второй способ (матрица жесткости умножается на вектор значений нормального поля **в узлах элемента**), V3 – третий способ (численное интегрирование методом **Гаусса**), V4 – четвёртый способ ($gradV^N$ для элемента считается в его центре), V5 – пятый способ ($gradV^N$ для элемента считается в его узлах), V* – аналитическое решение.

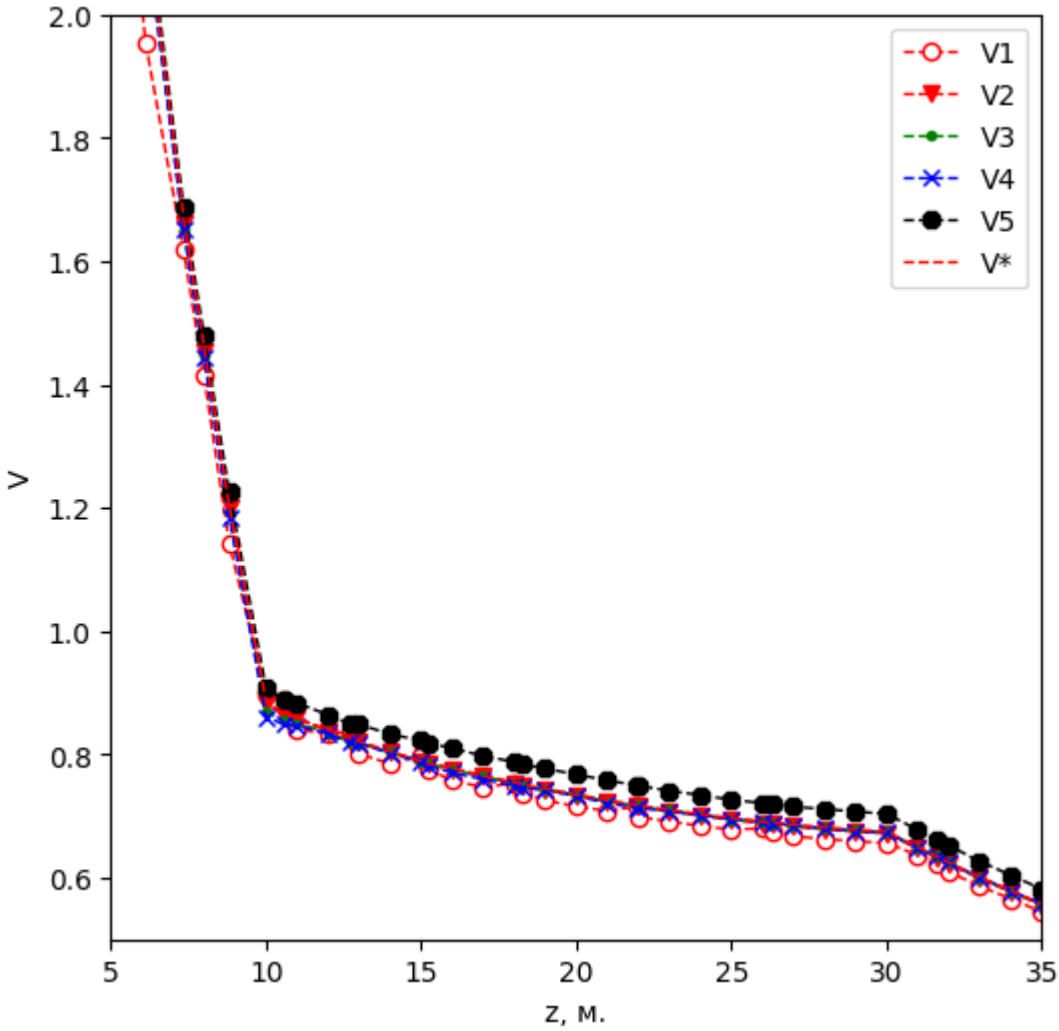


Рисунок 39 – Сравнение решений

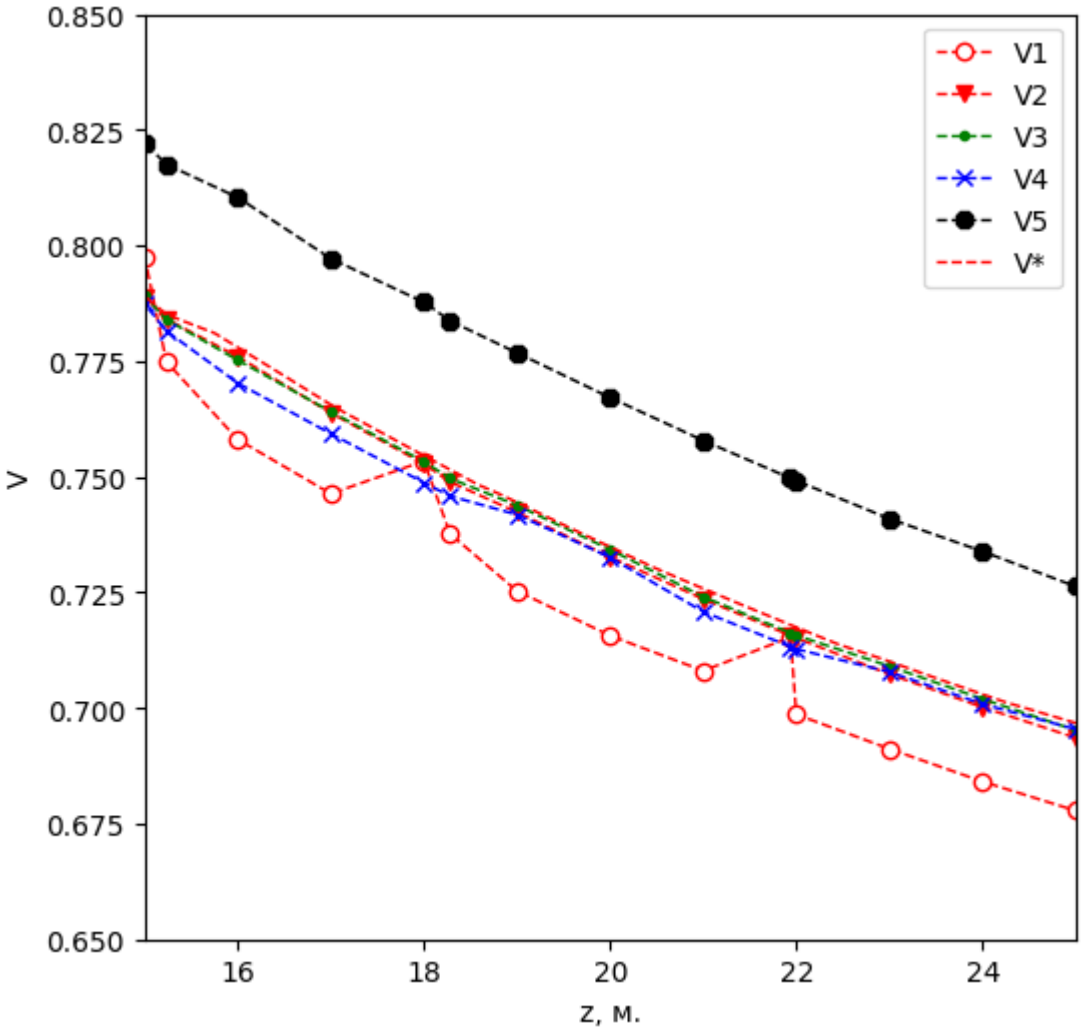


Рисунок 40 – Сравнение решений, крупный масштаб

Посчитаем относительную погрешность решения на поверхности. Будем рассматривать линию z = 0, y = 0 и x в интервале от 0 до 100 м (рисунок 41).

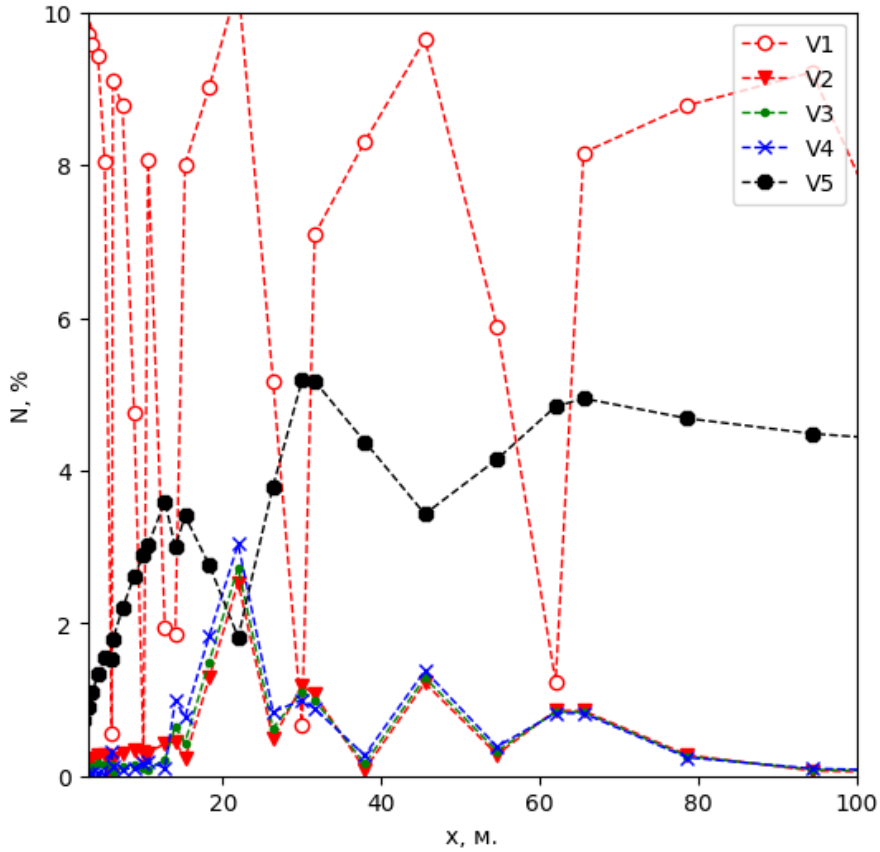


Рисунок 41 – Относительная погрешность решений

Посчитаем норму относительной погрешности для каждого из решений. На этот раз третий и четвертый методы не уступают Гауссу (таблица 10). Отличие с первым и пятым методом от Гаусса в 7 и 5 раз соответственно.

Таблица 10 – Норма относительной погрешности решений

	q в центре	q в узлах	$gradV^N$ в точках Гаусса	$gradV^N$ в центре	$gradV^N$ в узлах
Норма относительной погрешности, %	18.11	3.05	3.32	3.14	30.93

4.5. ИССЛЕДОВАНИЕ 5. ОБЪЕКТ В СРЕДЕ СО СЛОЕМ

Для пятого исследования построим область следующим образом. В среде с проводимостью $\sigma = 0.01$, разместим слой на глубине 30 м, толщиной 70 м и проводимостью $\sigma^{слой} = 0.001$. В этом слое будет находится объект на глубине 60 м,

с размером 20x20x20 м и проводимостью $\sigma^A = 0.1$. В качестве точного решения будет использовано соответствующее решение трёхмерной задачи на подробной (рисунок 43) сетке третьим способом. Относительно грубая сетка представлена на рисунке 42.

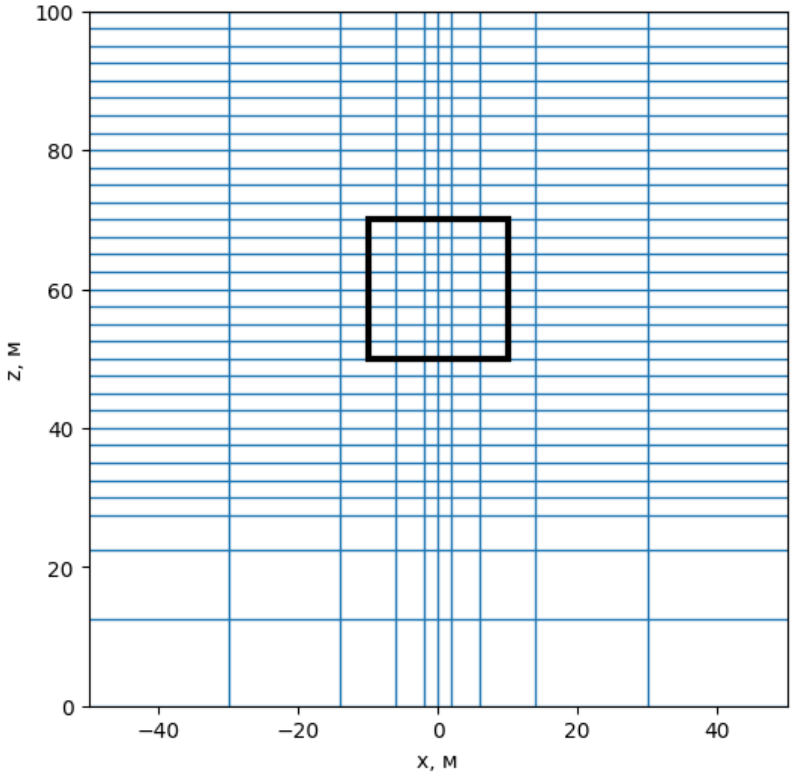


Рисунок 42 – Вид сетки с трехмерным объектом, срез среды $y = 0$, 54450 узлов

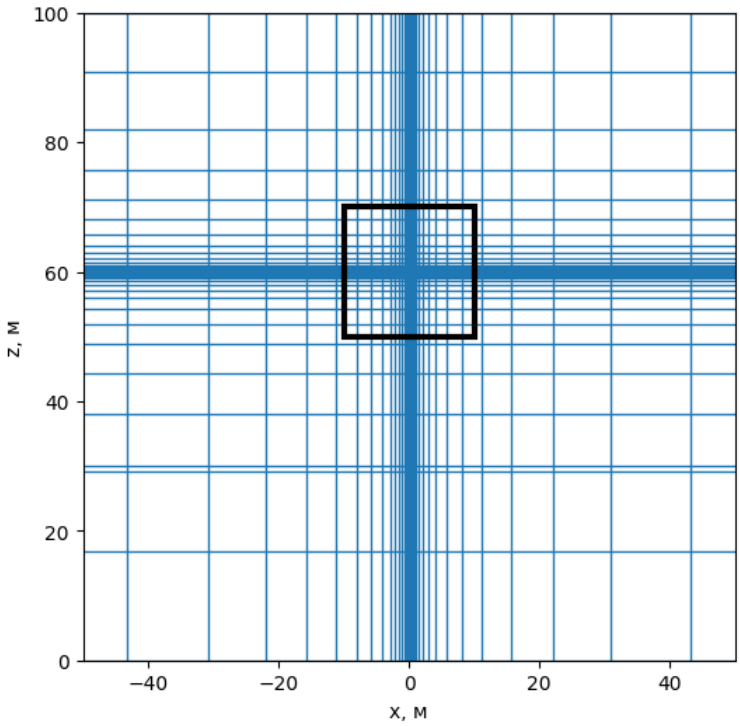


Рисунок 43 – Вид **подробной** сетки с трехмерным объектом, срез среды $y = 0$, 2203285 узлов

На рисунках 44-48: Vn – решение соответствующей двумерной задачи, Va – решение задачи на добавочное поле, V – полученное численное решение, V* – решение полученное третьим способом на подробной сетке.

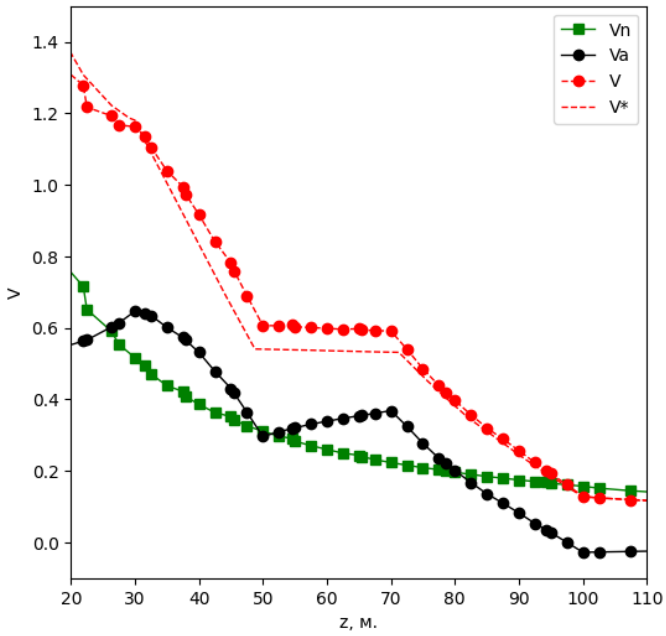


Рисунок 44 – График решения **первым** способом

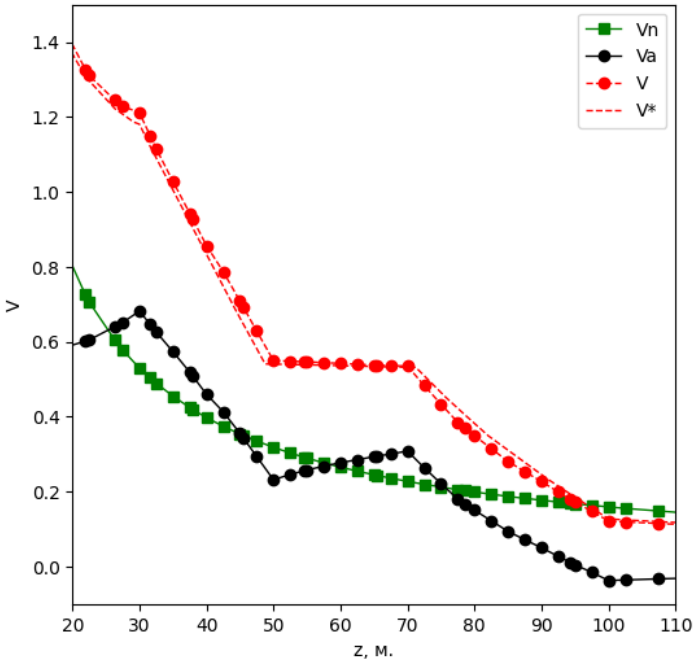


Рисунок 45 – График решения **вторым** способом

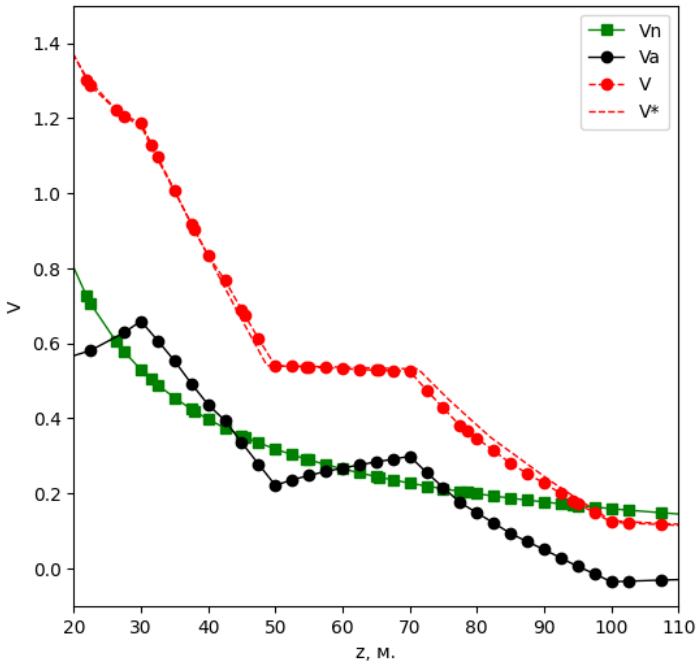


Рисунок 46 – График решения **третьим** способом

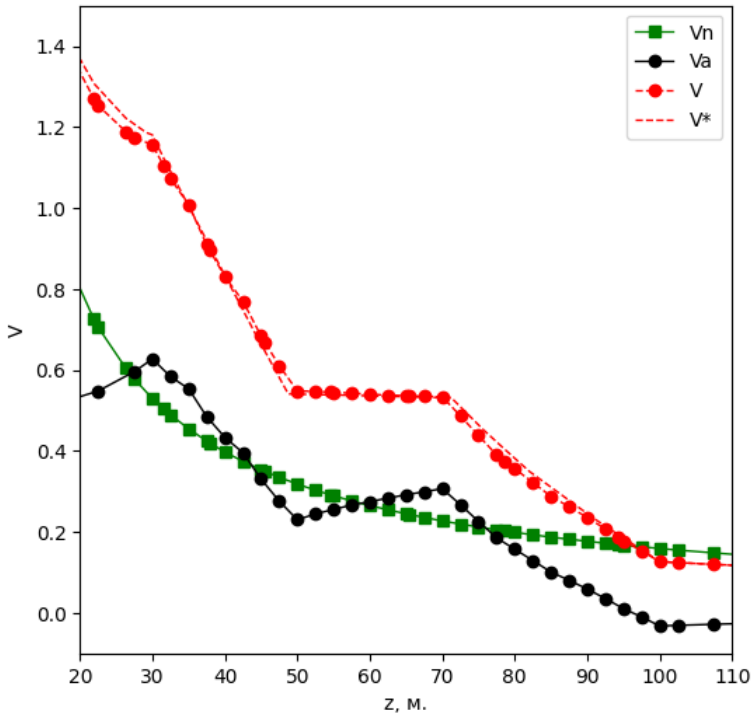


Рисунок 47 – График решения **четвертым** способом

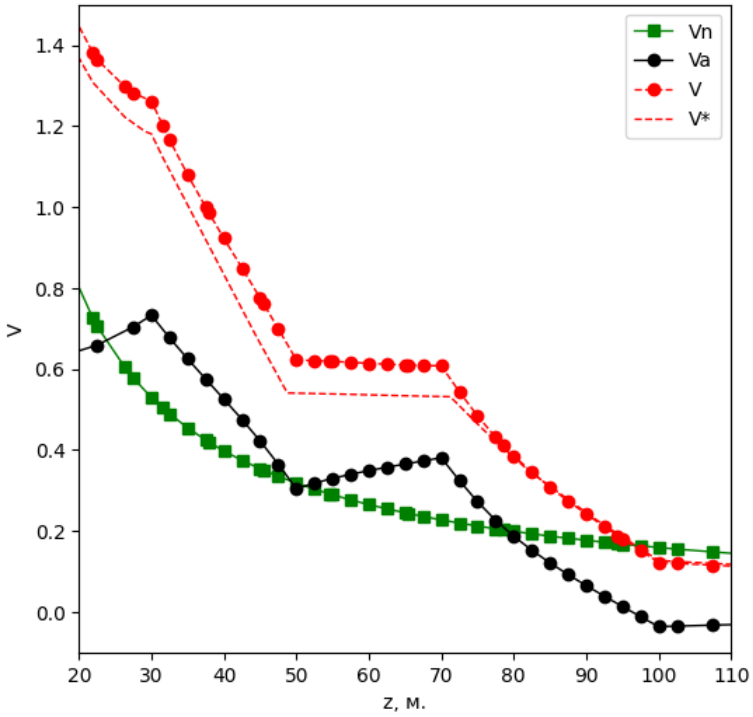


Рисунок 48 – График решения **пятым** способом

На рисунках 49 и 50: V1 – решение с первым способом выделения части поля (формирование правой части СЛАУ умножением матрицы жесткости на вектор значения нормального поля **в центре элемента**), V2 – второй способ

(матрица жесткости умножается на вектор значений нормального поля **в узлах элемента**), V3 – третий способ (численное интегрирование методом **Гаусса**), V4 – четвёртый способ ($gradV^N$ для элемента считается в его центре), V5 – пятый способ ($gradV^N$ для элемента считается в его узлах), V* – аналитическое решение.

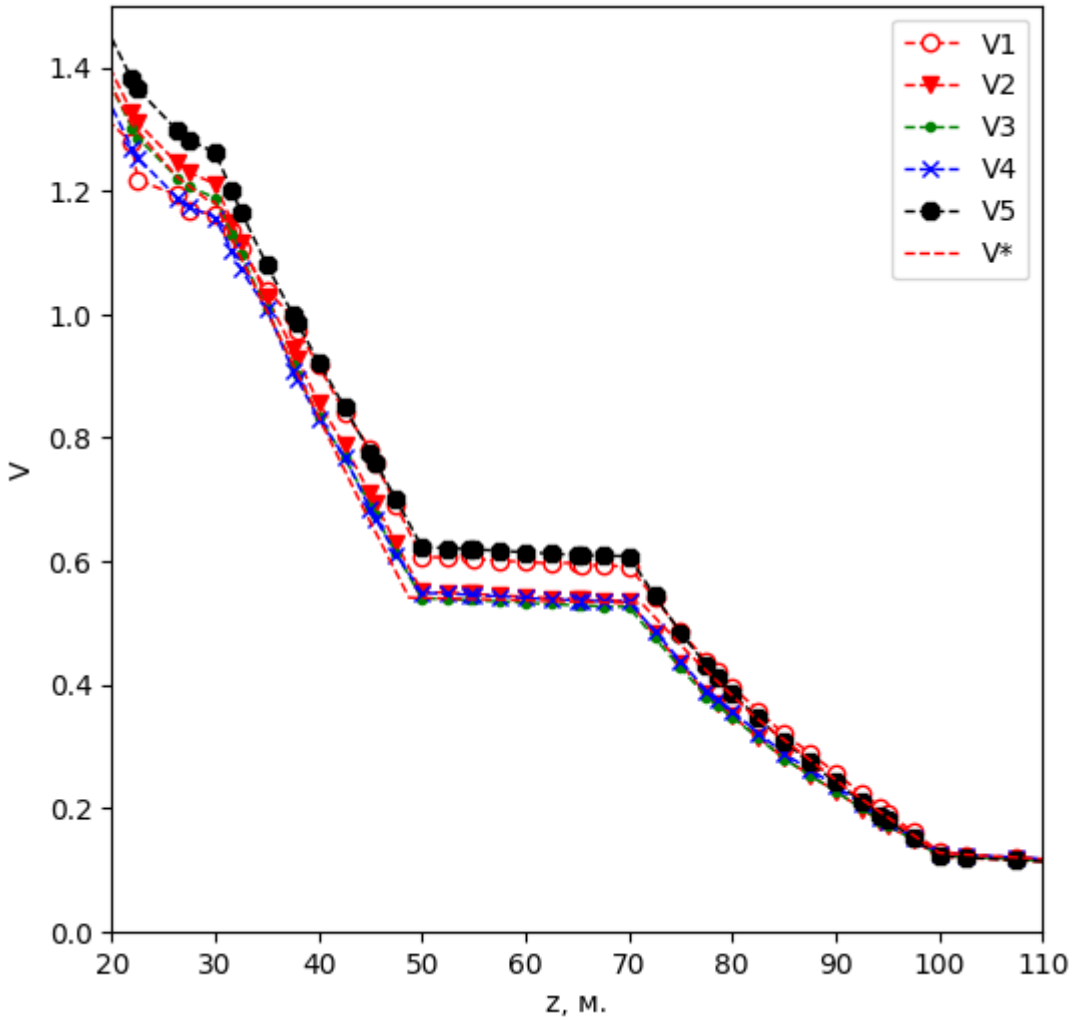


Рисунок 49 – Сравнение решений

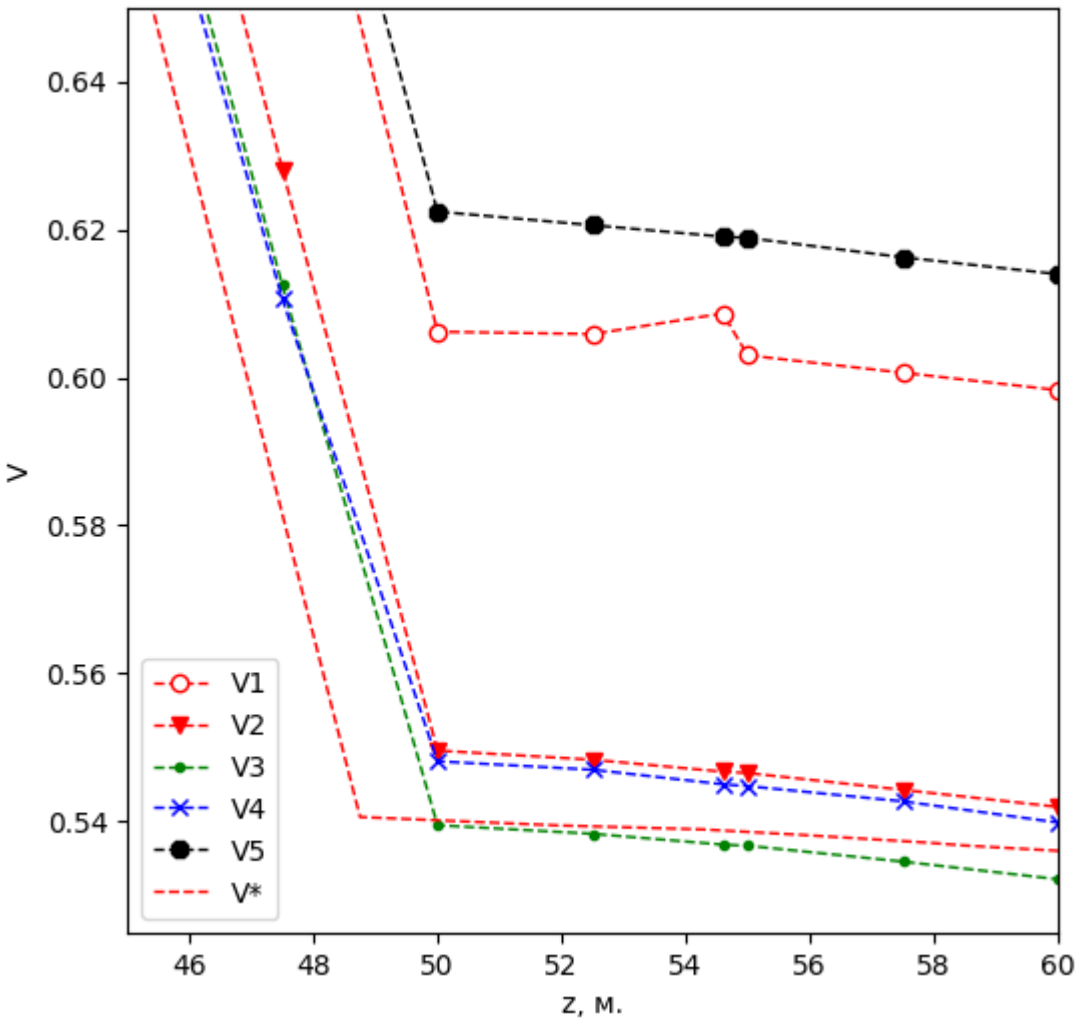


Рисунок 50 – Сравнение решений, крупный масштаб

Посчитаем относительную погрешность решения на поверхности. Будем рассматривать линию z = 0, y = 0 и x в интервале от 0 до 100 м (рисунок 51).

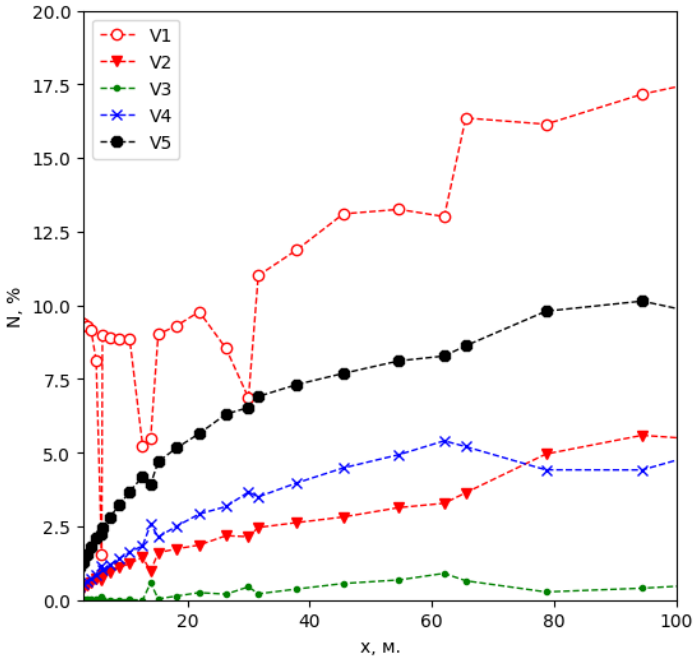


Рисунок 51 – Относительная погрешность решений

Посчитаем норму относительной погрешности для каждого из решений. Из таблицы 11 видим: вновь Гаусс лидирует, с разницей в 6 и 8.3 раза со вторым и третьим методом соответственно, и с разницей в 25.3 и 15.2 раза с первым и пятым методом соответственно.

Таблица 11 – Норма относительной погрешности решений

	q в центре	q в узлах	$gradV^N$ в точках Гаусса	$gradV^N$ в центре	$gradV^N$ в узлах
Норма относительной погрешности, %	43.66	10.18	1.72	14.35	26.19

ЗАКЛЮЧЕНИЕ

В данной работе были реализованы различные методы учета части аномального стационарного электрического поля в горизонтально слоистой среде.

Лучший результат показал метод с численным вычислением интеграла методом Гаусса (с 3 узлами по каждой координате). Решение этим методом дало наиболее гладкий и точный из всех результат, даже на относительно грубой сетке.

Одного порядка погрешность, с отличием от Гаусса в 4-5 раз, имели следующие методы учёта: формирование правой части умножение локальной матрицы жесткости на вектор значений нормального поля в *каждом узле* элемента и вычисление градиента *в центре* элемента.

Хуже всего погрешность в сравнении с методом Гаусса (отличие в 14-16 раз) имели следующие методы: формирование правой части умножение локальной матрицы жесткости на вектор значений нормального поля в *центре* элемента и вычисление градиента *в узлах* элемента.

Таким образом, среди реализованных методов, для использования в рассмотренной задаче отдаём предпочтения численному интегрированию методом Гаусса. Даже с учётом возрастающей вычислительной сложности, стоит учитывать, что решение СЛАУ в расчётах занимает бóльшую часть времени. Поэтому выигрыш в точности становится наиболее значимым результатом.

СПИСОК ЛИТЕРАТУРЫ

1. Соловейчик Ю. Г., Рояк М. Э., Персова М. Г. Метод конечных элементов для решения скалярных и векторных задач: Учеб. пос. Новосибирск: изд. НГТУ, 2007.

2. Вержибцкий В.М. Основы численных методов : учебник для вузов по направлению «Прикладная математика» / В.М. Вержибцкий. – М., 2005.

3. Рояк М.Э., Рояк С.Х. Программирование вычислений. – Новосибирск.: Изд-во НГТУ, 2012.

4. Численные методы в уравнениях математической физики: Учеб. Пос. / М.Г. Персова, Ю.Г. Соловейчик, Д.В. Вагин, П.А. Домников, Ю.И. Кошкина. – Новосибирск: Изд-во НГТУ, 2016.

5. М. Г. Персова, Ю. Г. Соловейчик, М. В. Абрамов, Конечноэлементное моделирование геоэлектромагнитных полей, возбуждаемых горизонтальной электрической линией, Сиб. журн. индустр. матем., 2009, том 12, номер 4, 106–119.

6. Соловейчик Ю.Г., Токарева М.Г., Персова М.Г. Решение трехмерных стационарных задач электроразведки на нерегулярных параллелепипеидальных сетках // Вестник ИрГТУ. Иркутск. 2004б. № 1. 45–60.

7. Спичак В.В. Магнитотеллурические поля в трехмерных моделях геоэлектрики. М.: Научный мир. 1999. 204 с.

8. Biro, O., and Preis, K., 1990, Finite element analysis of 3D eddy currents: IEEE Trans. Magn. 26, 418–423.

9. Mackie, R. L., Smith, J. T., and Madden, T. R., 1994, Three-dimensional electromagnetic modeling using finite difference equations: The magnetotelluric example: Radio Sci., 29, 923–935.

10. Xiong, Z., 1992, Electromagnetic modeling of three-dimensional structures by the method of system iterations using integral equations: Geophysics, 57, 1556–1561.

ПРИЛОЖЕНИЕ А. ТЕКСТ ПРОГРАММЫ

Файл main.c

```
#include <iostream>
#include <string.h>
#include "fem.h"
#include "fem_n.h"

#define MESH "./meshs/simple_mesh"
#define MESH "./meshs/test_segregation"
#define MESH "./meshs/mesh1"
#define MESH "./meshs/polinom/polinom_lambda_test_6"
#define MESH "./meshs/mesh_2x2"
#define MESH "./meshs/polinom/test_splitting_rz_z"
#define MESH "./meshs/rz_optim/h0"
#define MESH "./meshs/rz_optim/optim"
#define MESH "./meshs/polinom/polinom_lambda_1"

#define MAXITER 100
#define EPS 1e-30
#define RELAX 1.

#define SEG1 1
#define SEG2 1
#define SEG3 1
#define SEG4 1
#define SEG5 1


void test_rz_optim() {
    std::cout << "\nTEST RZ_OPTIM"<< endl;
    string filename = "./meshs/rz_optim/optim";
    FEM_N fem(filename);
    fem.solve_rz(2000, EPS, 1., (string)"./solutions/rz_optim/optim/q_rz");
    //fem.solution_rz_in_points("solution_in_point.txt");
}

void test_xyz_optim() {
    std::cout << "\nTEST XYZ_OPTIM" << endl;
    string filename = "./meshs/xyz_optim";
    FEM_N fem(filename);
    fem.solve(2000, EPS, 1., (string)"./solutions/xyz_optim/q_xyz");
    //fem.solution_rz_in_points("solution_in_point.txt");
}

void test_rz_optim_h0() {
    for (int i = 1; i < 6; i++) {
        std::cout << "\nTEST RZ_OPTIM H0 #" << i << endl;
        char buf[20];
        sprintf_s(buf, "/1e-%d", i);
        string filename = MESH +(string)buf;
        FEM_N fem(filename);
        fem.solve_rz(2000, EPS, 1.,
(string)"./solutions/rz_optim/h0"+(string)buf);
        //fem.solution_rz_in_points("solution_in_point.txt");
    }
}

void test_rz_optim_k() {
    for (int i = 1; i < 11; i++) {
        std::cout << "\nTEST RZ_OPTIM K #" << i << endl;
```

```
        char buf[20];
        sprintf_s(buf, "/1_%02d", i);
        string filename = "./meshs/rz_optim/k" + (string)buf;
        FEM_N fem(filename);
        fem.solve_rz(2000, EPS, 1., (string)"./solutions/rz_optim/k" +
(string)buf);
    }
}

void segregation_test() {
    string folder = "test";
    std::cout << "\nSEGREGATION TEST 1\n";
    FEM_N fem("./meshs/segregation/" + folder + "/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    fem.solution_rz_to_xyz2("./solutions/segregation/" + folder + "/rz");
    if (SEG1) {
        fem.solve_segregation1(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz1");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG2) {
        fem.solve_segregation2(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz2");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG3) {
        fem.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz3");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG4) {
        fem.solve_segregation4(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz4");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG5) {
        fem.solve_segregation5(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz5");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
}

void segregation_analitycal() {
    string folder = "analytical";
    std::cout << "\nSEGREGATION ANALYTICAL 1\n";
    FEM_N fem("./meshs/segregation/" + folder + "/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder  +
"/rz_for_xyz");
    fem.solution_rz_to_xyz2("./solutions/segregation/" + folder + "/rz");
    if (SEG1) {
        fem.solve_segregation1(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz1");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG2) {
        fem.solve_segregation2(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz2");
```

```
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG3) {
        fem.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz3");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG4) {
        fem.solve_segregation4(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz4");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG5) {
        fem.solve_segregation5(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz5");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
}

void i2() {
    string folder = "i2";
    std::cout << "\nTEST 2\n";
    FEM_N femrz("./meshs/segregation/" + folder + "/rz");
    femrz.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder + "/rz");
    FEM_N fem("./meshs/segregation/" + folder + "/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    if (SEG1) {
        fem.solve_segregation1(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz1");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG2) {
        fem.solve_segregation2(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz2");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG3) {
        fem.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz3");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG4) {
        fem.solve_segregation4(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz4");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG5) {
        fem.solve_segregation5(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz5");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
}

void i3() {
```

```
    string folder = "i3";
    std::cout << "\nTEST 3\n";
    FEM_N femrz("./meshs/segregation/" + folder + "/rz");
    femrz.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder + "/rz");
    FEM_N fem("./meshs/segregation/" + folder + "/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    if (SEG1) {
        fem.solve_segregation1(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz1");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG2) {
        fem.solve_segregation2(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz2");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG3) {
        fem.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz3");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG4) {
        fem.solve_segregation4(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz4");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG5) {
        fem.solve_segregation5(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz5");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
}

void i4() {
    string folder = "i4";
    std::cout << "\nTEST 4\n";
    FEM_N fem_big("./meshs/segregation/" + folder + "/xyz_big");
    fem_big.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    fem_big.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz_big");
    fem_big.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");

    FEM_N fem("./meshs/segregation/" + folder + "/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    if (SEG1) {
        fem.solve_segregation1(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz1");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG2) {
        fem.solve_segregation2(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz2");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
```

```
    }
    if (SEG3) {
        fem.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz3");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG4) {
        fem.solve_segregation4(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz4");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG5) {
        fem.solve_segregation5(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz5");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
}

void i5() {
    string folder = "i5";
    std::cout << "\nTEST 5\n";

    FEM_N fem_big("./meshs/segregation/" + folder + "/xyz_big");
    fem_big.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    fem_big.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz_big");
    fem_big.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");

    FEM_N fem("./meshs/segregation/" + folder + "/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/" + folder +
"/rz_for_xyz");
    if (SEG1) {
        fem.solve_segregation1(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz1");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG2) {
        fem.solve_segregation2(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz2");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG3) {
        fem.solve_segregation3(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz3");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG4) {
        fem.solve_segregation4(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz4");
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
    if (SEG5) {
        fem.solve_segregation5(2000, EPS, 1., (string)"./solutions/segregation/"
+ folder + "/xyz5");
```

```
        fem.solution_xyz_in_points("./meshs/segregation/" + folder +
"/xyz/solutions");
    }
}

void solution_trasnition_test() {
    std::cout << "\nSEGREGATION ANALYTICAL 1\n";
    FEM_N fem("./meshs/segregation/analytical/xyz");
    fem.solve_rz(2000, EPS, 1.,
"./solutions/segregation/analytical/rz_for_xyz");
    //fem.solution_rz_to_xyz1("./solutions/segregation/analytical/xyz1");
    fem.solve_segregation4(2000, EPS, 1.,
(string)"./solutions/segregation/analytical/xyz1");
}

void segregation_3_test() {
    std::cout << "\nSEGREGATION 3 TEST\n";
    FEM_N fem("./meshs/segregation/seg3test/xyz");
    fem.solve_rz(2000, EPS, 1., "./solutions/segregation/seg3test/rz_for_xyz");
    fem.solve_segregation3(2000, EPS, 1.,
(string)"./solutions/segregation/seg3test/xyz3");
    fem.solution_xyz_in_points("./meshs/segregation/seg3test/xyz/solutions");

}

int main()
{
    //test_polinome_rz_z_splitting();
    //test_rz_optim();
    //test_xyz_optim(); // Однородная среда
    //test_rz_optim_h0();
    //test_rz_optim_k();

    //segregation_test();
    //segregation_analitycal();
    //i2();
    //i3();
    //i4();
    //i5();
    //solution_trasnition_test();
    //segregation_3_test();
}
```

Файл fem_n.h

```
#ifndef FEM_RZ_H_
#define FEM_RZ_H_

#include <algorithm>
#include <time.h>

#include "sreda.h"
#include "STRCTRS.h"
#include "cgm.h"


// Количество узлов в элементе
#define FEM_XYZ_NODES_NUM 8
#define FEM_RZ_NODES_NUM 4

class FEM_N
{
      void print_matrix();
```

```
	int find_nel(double x, double y, double z, class MESH& mesh, class STRCTRS& ss,
std::vector<double>& q);
	SREDA sreda;
	MESH mesh;

	SREDA sreda_n;
	MESH mesh_n;

	SREDA_RZ sreda_rz;
	MESH_RZ mesh_rz;

	STRCTRS ssxyz, ssxyzn,ssrz;
	// СЛАУ
	// ggl - нижний треугольник
	// di - диагональные элементы
	std::vector<double>ggl, ggu, di, pr, qrz, q;
	std::vector<long> ia, ja;
	std::vector < std::set<long>> A; // Профиль ВСЕЙ матрицы, используется для применения
первых краевых
	std::vector<double> Vns; // Решение двумерной задачи в узлах общей сетки
	std::vector<double> Vnb; // Решение двумерной задачи в узлах sreda
	std::vector<double> Va;  // Решение задачи на аномальное поле в узлах sreda
	std::vector<double> Vas; // Решение задачи на аномальное поле в узлах общей сетки
	std::vector<double> Vs; // Сумма решений Vas и Vns

	int xi, yi, zi; // Переменные поиска элемента для функции xyz_to_xyz
	// Статические переменные не используется для корректной работы последовательного
расчета
public:
	// filename -
	FEM_N(string filename);
	void solve(int max_iter, double eps, double relax, std::string solution_filename);
	void solve_segregation1(int max_iter, double eps, double relax, std::string
solution_filename); // rz->xyz :Решение константа на элементе
	void solve_segregation2(int max_iter, double eps, double relax, std::string
solution_filename); // rz->xyz: Решение в узлах элемента
	void solve_segregation3(int max_iter, double eps, double relax, std::string
solution_filename); // rz->xyz: Точки Гауса?
	void solve_segregation4(int max_iter, double eps, double relax, std::string
solution_filename); // rz->xyz: Точки Гауса?
	void solve_segregation5(int max_iter, double eps, double relax, std::string
solution_filename); // rz->xyz: Точки Гауса?
	void solve_rz(int max_iter, double eps, double relax, std::string solution_filename);

	void solution_xyz_in_points(std::string filename);
	void solution_rz_in_points(std::string filename);


private:
	void build_matrix_profile();
	void build_matrix_profile_rz();
	void slae_init();

	void add_G(); // Строит матрицу жесткости для задачи xyz без выделения поля
	void add_b(); // Правая часть для задачи без выделения поля
	void add_G_sigma_diff(); // Строит матрицу с sigma = sigma(normal) - sigma
	void add_b_segregation_1(); // Считает правую часть для первого способа выделения
поля
	void add_b_segregation_2(); // Считает правую часть для второго способа выделения
поля
	void add_b_segregation_3(); // Считает правую часть для третьего способа выделения
поля
	void add_b_segregation_4(); // Считает правую часть для четвертого способа выделения
поля
	void add_b_segregation_5(); // Считает правую часть для пятого способа выделения поля
```

```
	void add_b_segregation_3(int NOMEL); // Считает правую часть для третьего способа выделения поля
	void add_b_segregation_4(int NOMEL); // Считает правую часть для четвертого способа выделения поля
	void add_b_segregation_5(int NOMEL); // Считает правую часть для пятого способа выделения поля

	void add_G_b_rz(); // Строит глобальные матрицу жесткости и вектор правой части для задачи rz
	void edge_cond_1(); // Первые краевые для задачи xyz
	void edge_cond_1_rz(); // Первые краевые для задачи rz

	// Выведет заполненность матрицы ненулевыми элементами
	void print_matrix_plenum();

	// Номер элемента в который попал узел
	int n_el_xyz(double x, double y, double z);

	// Перевести решение двумерной задачи в решение трехмерной
	// 1: Формирует вектор с длиной равной КОЛИЧЕСТВУ УЗЛОВ с решением двумерной задачи в КАЖДОМ ЭЛЕМЕНТЕ
	void q_rz_to_xyz1(vector<double>& ans, class MESH& mesh, class STRCTRS& ss);
	// 2: Формирует вектор с длиной равной КОЛИЧЕСТВУ УЗЛОВ с решением двумерной задачи в КАЖДОМ УЗЛЕ
	void q_rz_to_xyz2(vector<double>& ans, class MESH& mesh, class STRCTRS& ss);

	// Перевести решение трехмерной задачи с одной сетки в другую
	// q1 - исходный вектор решений
	// mesh1 - сетка исходного решения
	//
	// q2 - получаемый вектор решений
	// mesh2 - сетка для получаемого решения
	// ss2 - струкутры получаемого решения
	void q_xyz_to_xyz(vector<double>& q1, class MESH& mesh1,class STRCTRS& ss1, vector<double>& q2, class MESH& mesh2, class STRCTRS& ss2);

private:
	void add_to_sparse(std::vector<long>& ia, std::vector<long>& ja, std::vector<double>& ggl, long str, long col, double x);
	void replace_in_sparse(vector<long>& ia, vector<long>& ja, vector<double>& ggl, long str, long col, double x);
	double replace_in_sparse_r(vector<long>& ia, vector<long>& ja, vector<double>& ggl, long str, long col, double x);
	// Сложение двух векторов x = a + b
	void vec_vec_sum(std::vector<double>& a, std::vector<double>& b, std::vector<double>& x);
	// Умножит матрицу на вектор q, результат будет помещен в вектор x
	void matrix_vector_mul(std::vector<double>& q, std::vector<double>& x);

	void solve_cgm(vector<double>& q, int max_iter, double eps, double relax);


	void local_G(vector<vector<double>>& G, double h_x, double h_y, double h_z, double lambda);
	void local_G_rz(vector<vector<double>>& G, double h_r, double h_z, double lambda, int i);
	void local_G_rzn(vector<vector<double>>& G, double h_r, double h_z, double lambda, int i); //Удалить
	// -------Функции тестирования на полиномах---------
public:
	void test_polinome(int max_iter, double eps, double relax, std::string solution_filename);
	void test_polinome_rz(int max_iter, double eps, double relax, std::string solution_filename);
```

```
	double solution_xyz_in_point(double x, double y, double z); // Решение трехмерной
задачи в точке
	double solution_xyz_in_point(double x, double y, double z,
		std::vector<double>& q, class MESH& mesh, class STRCTRS& ss); // Решение
трехмерной задачи для сетки mesh с решением q
	double solution_rz_in_point(double r, double z); // Выдает решение двумерной задачи в
точке
	void grad_rz_in_point(double x, double y, double z, std::vector<double>& grad); //
Значение градиента по декартовым координатам для цилиндрической задачи
	void grad_rz_in_point2(double x, double y, double z, std::vector<double>& grad); //
Значение градиента по декартовым координатам для цилиндрической задачи
	void solution_export(class MESH& mesh, std::string solution_filename,
std::vector<double>& solution);
	void solution_rz_export(std::string filename);

	void solution_rz_to_xyz2(std::string filename); // Переведет точки решения двумерной
задачи в трехмерную
private:
	void add_G_b_p(); // Считает глобальную матрицу жесткости и правую часть
	void add_G_b_rz_p();
	void edge_cond_1_p();
	void edge_cond_1_rz_p();
	void print_solution_p_miss();
	void print_solution_rz_p_miss(); // Норма вектора относительной погрешности решения

	// u - искомая функция дифференциального уравнения
	// f - функция правой части дифференциального уравнения
	double u(double x, double y, double z);
	double f(double x, double y, double z);
	double u_rz(double r, double z);
	double f_rz(double r, double z);

	// Считает локальную правую часть
	void local_b_p(vector<double>& b, double h_x, double h_y, double h_z, int i);
	void local_b_rz_p(vector<double>& b, double h_r, double h_z, int i); // тест на
полином
	void local_b_rz_pn(vector<double>& b, double h_r, double h_z, int i); // удалить
	//-----------------------------------------------------
};

#endif // FEM_RZ_H_
```

Файл cgm.h

```
#ifndef CGM_H_
#define CGM_H_

#include <cmath>
#include <vector>
#include <iostream>
using namespace std;


double vec_vec(vector<double>& X, vector<double>& Y);
double nev(vector<double>& p, vector<double>& pr);
void matrix_vector_i0(vector<double>& ggl, vector<double>& ggu, vector<long>& ig,
vector<long>& jg, vector<double>& di, vector<double>& x, vector<double>& y);
void matrix_vector(vector<double>& ggl, vector<double>& ggu, vector<long>& ig, vector<long>&
jg, vector<double>& di, vector<double>& x, vector<double>& y);



class CGM
{
public:
```

```
    void init(vector< long>& gi_s, vector< long>& gj_s, vector< double>& di_s, vector<
double>& gg_s, vector< double>& rp_s);
    void solve(vector<double>& solution, int max_iter, double eps, double relax);
    void solve_msg(vector<double>& ggl, vector<long>& ia, vector<long>& ja, vector<double>&
di, vector<double>& pr,
        vector<double>& q, int max_iter, double eps, double relax);

private:
    void make_LLT_decomposition();
    void mul_matrix(vector<double>& f, vector<double>& x);
    void solve_L(vector<double>& f, vector<double>& x);
    void solve_LT(vector<double>& f, vector<double>& x);
    void solve_LLT(vector<double>& f, vector<double>& x);
    double dot_prod(vector<double>& a, vector<double>& b);

    int n;
    vector< long> ia, ja;
    vector<double> di, gg, pr, r, x0, z, p, s;
    vector<double> L_di, L_gg;
};

#endif // CGM_H_
```

Файл mesh.h

```
#ifndef MESH_H_
#define MESH_H_

#include <vector>
#include <list>
#include <set>

// MAT_N - номер материала
//
enum { X0_COORD, X1_COORD, Y0_COORD, Y1_COORD, Z0_COORD, Z1_COORD, MAT_N, ANOMAL };

// Описание массива источников
enum { SOURCE_X, SOURCE_Y, SOURCE_Z, SOURCE_POW, SOURCE_N_UZLA };

//Описание массива с первыми краевыми
enum { L1_X0, L1_X1, L1_Y0, L1_Y1, L1_Z0, L1_Z1 };

using namespace std;

int get_mat_from_sreda_direct(class SREDA& sreda, double x, double y, double z);

class MESH_RZ {
public:
    int kuzlov;
    int kel;
    std::vector<double> r;
    std::vector<double> z;
    void gen_mesh(class SREDA_RZ & sreda_rz, class SREDA & sreda);
};

struct MESH {
public:
    int kuzlov;
    int kel;
    std::vector < double>x;
    std::vector < double>y;
    std::vector < double>z;
    std::vector < double>r;
    std::vector < double>h;

    /*vector<vector<int>> nvtr;
```

```
    vector<int> nvkat2d;
    vector<vector<double>> rz;
    vector<int> l1;
    vector<double> sigma;*/

    void gen_mesh(class SREDA& sreda);
    void gen_structures(class FEM& fem, class SREDA& sreda);

    void gen_mesh_rh(class SREDA& sreda);
    void gen_structures_rh(class FEM& fem, class SREDA& sreda);

    void add_mesh(class MESH& mesh);

};

#endif MESH_H_
```

Файл sreda.h

```
#ifndef SREDA_H_
#define SREDA_H_

#include <vector>
#include <iostream>
#include <fstream>

#define SREDA_FILENAME "/sreda"
#define SREDA_N_FILENAME "/sreda_n"
#define EDGE_CONDITIONS_FILENAME "/edge_conditions"
#define CURRENT_SOURCES_FILENAME "/current_sources"
#define MATERIALS_FILENAME "/materials"
#define POINTS_FILENAME "/points"
#define POINTS_RZ_FILENAME "/points_rz"


using namespace std;
// радиус, начальный шаг, максимальный шаг, коэффициент разрядки
enum {END, H0, HMAX, K};

enum sigma_strucure { SIGMA, SIG_ANOMAL };

class SREDA_RZ {
public:
    // радиус, начальный шаг, максимальный шаг, коэффициент разрядки
    std::vector<double> r;
    std::vector<double> z;
    // Параметры дробления сетки по r и по z
    std::vector<double> splitting;

    //Файл points_rz
    std::vector<std::vector<double>> points;

    void read_points(string filename);

    // Прочтет информацию о двумерной задаче
    void read_sreda(string filename);
};

class SREDA
{
public:
    std::vector < std::vector<double>> elms;

    std::vector<double> x;
    std::vector<double> hx;
    std::vector<double> kx;
```

```
    std::vector<double> y;
    std::vector<double> hy;
    std::vector<double> ky;

    std::vector <double> z;
    std::vector<double> hz;
    std::vector<double> kz;
    std::vector<std::vector<int>> left_right;

    std::vector<int> splitting;

    //Файл edge_conditions
    vector<double> edge_conditions;

    //Файл current_sources
    vector<vector<double>> current_sources;

    //Файл materials
    vector<vector<double>> sigma;

    // Файл points
    vector<vector<double>> points;

    //Прочитает файл sreda
    void read_sreda(const char* filename);
    void read_sreda(string filename);

    //Прочитает файл edge_conditions
    void read_edge_conditions(const char* filename);
    void read_edge_conditions(string filename);

    //Прочитает файл current_sources
    void read_current_sources(const char* filename);
    void read_current_sources(string filename);

    //Прочитает файл materials
    void read_materials(const char* filename, class FEM& fem);
    void read_materials(string filename);

    // Прочитает файл points
    void read_points(string filename);

    // Прочтет все входные файлы для трехмерной задачи
    void read_problem(string meshname);
    void read_problem_n(string meshname); // файл sreda_n
};
#endif // SREDA_H_
```

Файл strctrs.h

```
#ifndef STRCTRS_H_
#define STRCTRS_H_

#include "mesh.h"
#include "sreda.h"

class STRCTRS
{
public:
    vector<vector<int>> nvtr; // mesh.kel записей номеров узлов элементов
    vector<int> nvkat2d;  // mesh.kel записей номеров материала элементов
    vector<int> nvkat2dr;  // mesh.kel записей номеров материалов элементов для нормального
поля
    vector<vector<double>> coord; // mesh.kuzlov координат точек
    vector<int> l1; // номера узлов с первыми краевыми условиями
```

```
public: // Вызов функций генерации структур
    void gen_structures(SREDA& sreda, MESH& mesh);
    void gen_structures_rz(SREDA& sreda, MESH_RZ& mesh);

    void gen_structures_p(SREDA& sreda, MESH& mesh);
    void gen_structures_rz_p(SREDA& sreda, MESH_RZ& mesh); // Тест на полиномах


private: // Функции генерации структур для трехмерной задачи
    void gen_nvtr(class MESH& mesh);
    void gen_coord(class MESH& mesh);
    void gen_nvkat2d(class SREDA& sreda, class MESH& mesh);
    void gen_nvkat2dr(class SREDA& sreda, class MESH& mesh); // Номера материалов для
нормальной задачи
    void gen_l1(class MESH& mesh, vector<double>& edge_conditions);
    void gen_l1_p(class MESH& mesh, vector<double>& edge_conditions);

private: // Функции генерации структур для двумерной задачи
    void gen_nvtr_rz(class MESH_RZ& mesh);
    void gen_coord_rz(class MESH_RZ& mesh);
    void gen_nvkat2d_rz(class MESH_RZ& mesh, class SREDA& sreda);
    void gen_l1_rz(class MESH_RZ& mesh, vector<double>& edge_conditions);
    void gen_l1_rz_p(class MESH_RZ& mesh, vector<double>& edge_conditions); // Тест на
полиномах


private:
    int get_mat_from_sreda_reverse(class SREDA& sreda, double x, double y, double z);
    int get_mat_from_sreda_direct(class SREDA& sreda, double x, double y, double z);
    int get_mat_from_sreda_rz(class SREDA& sreda, double z);
};

#endif // STRCTRS_H_
```

Файл fem_n.cpp

```
#include "fem_n.h"

#define I 1.
#define P_X 0
#define P_Y 0
#define P_Z 15.

double FEM_N::u(double x, double y, double z) {
    //return x * y + y * z + z * x;
    return x*y*z;
}
double FEM_N::f(double x, double y, double z) {
    return 0;
}
double FEM_N::u_rz(double r, double z) {
    return z*z;
}
double FEM_N::f_rz(double r, double z) {
    //if (r == 0) { return 0; }
    return -2.;
}

// Внесение элемента в разреженную матрицу
// ia, ja - профиль матрицы
// ggl - вектор куда вставляется значение
// str - номер строки начиная с 1
// col - номер столбца начиная с 1
// x - вносимое значение
```

```
void FEM_N::add_to_sparse(vector<long>& ia, vector<long>& ja, vector<double>&
ggl, long str, long col, double x) {
    long start = ia[str - 1] - 1, end = ia[str] - 1;
    for (long i = start; i < end; i++) {
        if (ja[i] == col) { ggl[i] += x; break; }
    }
}

// Замена значения в разреженной матрице
// ia, ja - профиль матрицы
// ggl - вектор куда вставляется значение
// str - номер строки начиная с 1
// col - номер столбца начиная с 1
// x - вносимое значение
double FEM_N::replace_in_sparse_r(vector<long>& ia, vector<long>& ja,
vector<double>& ggl, long str, long col, double x) {
    double el = 0;
    long start = ia[str - 1] - 1, end = ia[str] - 1;
    for (long i = start; i < end; i++) {
        if (ja[i] == col) { el = ggl[i];  ggl[i] = x; break; }
    }
    return el;
}
// Заменит значение в разреженной матрице и вернет его
// ia, ja - профиль матрицы
// ggl - вектор куда вставляется значение
// str - номер строки начиная с 1
// col - номер столбца начиная с 1
// x - вносимое значение
void FEM_N::replace_in_sparse(vector<long>& ia, vector<long>& ja,
vector<double>& ggl, long str, long col, double x) {
    long start = ia[str - 1] - 1, end = ia[str] - 1;
    for (long i = start; i < end; i++) {
        if (ja[i] == col) { ggl[i] = x; break; }
    }
}

void FEM_N::vec_vec_sum(std::vector<double>& a, std::vector<double>& b,
std::vector<double>& x) {
    x.resize(a.size());
    for (int i = 0; i < a.size(); i++) { x[i] = a[i] + b[i]; }
}
void FEM_N::matrix_vector_mul(std::vector<double>& q, std::vector<double>& x) {
    int n = q.size();
    x.resize(0);
    x.resize(n);
    for (int i = 0; i < n; i++)
    {
        x[i] = di[i] * q[i];
        for (unsigned int k = ia[i] - 1, k1 = ia[i + 1] - 1; k < k1; k++)
        {
            unsigned int j = ja[k] - 1;
            x[i] += ggl[k] * q[j];
            x[j] += ggl[k] * q[i];
        }
    }
}

// Посчитает и запишет в G[8][8] локальную матрицу жесткости
void FEM_N::local_G(vector<vector<double>>& G, double h_x, double h_y, double
h_z, double lambda) {
    double k1 = lambda * h_y * h_z / (h_x*36.), k2 = lambda * h_x * h_z /
(h_y*36.), k3 = lambda * h_x * h_y / (h_z*36.);
    G[0] = { +k1 * 4 + k2 * 4 + k3 * 4 };
```

```
    G[1] = { -k1 * 4 + k2 * 2 + k3 * 2, +k1 * 4 + k2 * 4 + k3 * 4 };
    G[2] = { +k1 * 2 - k2 * 4 + k3 * 2, -k1 * 2 - k2 * 2 + k3 * 1, +k1 * 4 + k2
* 4 + k3 * 4 };
    G[3] = { -k1 * 2 - k2 * 2 + k3 * 1, +k1 * 2 - k2 * 4 + k3 * 2, -k1 * 4 + k2
* 2 + k3 * 2, +k1 * 4 + k2 * 4 + k3 * 4 };
    G[4] = { +k1 * 2 + k2 * 2 - k3 * 4, -k1 * 2 + k2 * 1 - k3 * 2, +k1 * 1 - k2
* 2 - k3 * 2, -k1 * 1 - k2 * 1 - k3 * 1, +k1 * 4 + k2 * 4 + k3 * 4 };
    G[5] = { -k1 * 2 + k2 * 1 - k3 * 2, +k1 * 2 + k2 * 2 - k3 * 4, -k1 * 1 - k2
* 1 - k3 * 1, +k1 * 1 - k2 * 2 - k3 * 2, -k1 * 4 + k2 * 2 + k3 * 2, +k1 * 4 + k2
* 4 + k3 * 4 };
    G[6] = { +k1 * 1 - k2 * 2 - k3 * 2, -k1 * 1 - k2 * 1 - k3 * 1, +k1 * 2 + k2
* 2 - k3 * 4, -k1 * 2 + k2 * 1 - k3 * 2, +k1 * 2 - k2 * 4 + k3 * 2, -k1 * 2 - k2
* 2 + k3 * 1, +k1 * 4 + k2 * 4 + k3 * 4 };
    G[7] = { -k1 * 1 - k2 * 1 - k3 * 1, +k1 * 1 - k2 * 2 - k3 * 2, -k1 * 2 + k2
* 1 - k3 * 2, +k1 * 2 + k2 * 2 - k3 * 4, -k1 * 2 - k2 * 2 + k3 * 1, +k1 * 2 - k2
* 4 + k3 * 2, -k1 * 4 + k2 * 2 + k3 * 2, +k1 * 4 + k2 * 4 + k3 * 4 };
}
// Посчитает и запишет в G[4][4] локальную матрицу жесткости
void FEM_N::local_G_rzn(vector<vector<double>>& G, double h_r, double h_z,
double lambda, int i) {
    double k1 = h_z / h_r * lambda, k2 = h_r / h_z * lambda;
    G[0] = { 2. / 6. * k1 + 2. / 6. * k2,-2. / 6. * k1 + 1. / 6. * k2,1. / 6. *
k1 - 2. / 6. * k2,-1. / 6. * k1 - 1. / 6. * k2 };
    G[1] = { -2. / 6. * k1 + 1. / 6. * k2,2. / 6. * k1 + 2. / 6. * k2,-1. / 6. *
k1 - 1. / 6. * k2,1. / 6. * k1 - 2. / 6. * k2 };
    G[2] = { 1. / 6. * k1 - 2. / 6. * k2,-1. / 6. * k1 - 1. / 6. * k2,2. / 6. *
k1 + 2. / 6. * k2,-2. / 6. * k1 + 1. / 6. * k2 };
    G[3] = { -1. / 6. * k1 - 1. / 6. * k2,1. / 6. * k1 - 2. / 6. * k2,-2. / 6. *
k1 + 1. / 6. * k2,2. / 6. * k1 + 2. / 6. * k2 };
}
void FEM_N::local_G_rz(vector<vector<double>>& G, double hr, double hz, double
lambda, int i) {
    double k1 = lambda * hz / (hr * 12.), k2 = lambda * hr / (hz * 12.);
    double r0 = ssrz.coord[ssrz.nvtr[i][0] - 1][0];
    double G24 = k1 * (2 * hr + 4 * r0), G14 = k2 * (hr + 4 * r0);
    double G112 = k1 * (hr + 2 * r0), G212 = k2 * (hr + 2 * r0);
    G[0] = { G24 + G14,-G24 + G212,G112 - G14,-G112 - G212 };
    G[1] = { G[0][1],G[0][0],-G112 - G212,G112 - G14 };
    G[2] = { G[0][2],G[1][2],G[0][0],-G24+G212 };
    G[3] = { G[0][3],G[1][3],G[2][3],G[0][0] };
}
void FEM_N::local_b_p(vector<double>& b, double h_x, double h_y, double h_z, int
i) {
    double x0, x1, y0, y1, z0,z1;
    double f0, f1, f2, f3, f4, f5, f6, f7;
    double k = h_x*h_y*h_z / 216.;
    x0 = ssxyz.coord[ssxyz.nvtr[i][0] - 1][0];
    x1 = ssxyz.coord[ssxyz.nvtr[i][1] - 1][0];
    y0 = ssxyz.coord[ssxyz.nvtr[i][0] - 1][1];
    y1 = ssxyz.coord[ssxyz.nvtr[i][2] - 1][1];
    z0 = ssxyz.coord[ssxyz.nvtr[i][0] - 1][2];
    z1 = ssxyz.coord[ssxyz.nvtr[i][4] - 1][2];
    f0 = f(x0, y0, z0);
    f1 = f(x1, y0, z0);
    f2 = f(x0, y1, z0);
    f3 = f(x1, y1, z0);
    f4 = f(x0, y0, z1);
    f5 = f(x1, y0, z1);
    f6 = f(x0, y1, z1);
    f7 = f(x1, y1, z1);
    b[0] = k * (8 * f0 + 4 * f1 + 4 * f2 + 4 * f3 + 2 * f4 + 2 * f5 + 2 * f6 + 1
* f7);
    b[1] = k * (4 * f0 + 8 * f1 + 2 * f2 + 2 * f3 + 4 * f4 + 1 * f5 + 4 * f6 + 2
* f7);
```

```
    b[2] = k * (4 * f0 + 2 * f1 + 8 * f2 + 2 * f3 + 4 * f4 + 4 * f5 + 1 * f6 + 2
* f7);
    b[3] = k * (4 * f0 + 2 * f1 + 2 * f2 + 8 * f3 + 1 * f4 + 4 * f5 + 4 * f6 + 2
* f7);
    b[4] = k * (2 * f0 + 4 * f1 + 4 * f2 + 1 * f3 + 8 * f4 + 2 * f5 + 2 * f6 + 4
* f7);
    b[5] = k * (2 * f0 + 1 * f1 + 4 * f2 + 4 * f3 + 2 * f4 + 8 * f5 + 2 * f6 + 4
* f7);
    b[6] = k * (2 * f0 + 4 * f1 + 1 * f2 + 4 * f3 + 2 * f4 + 2 * f5 + 8 * f6 + 4
* f7);
    b[7] = k * (1 * f0 + 2 * f1 + 2 * f2 + 2 * f3 + 4 * f4 + 4 * f5 + 4 * f6 + 8
* f7);
}
void FEM_N::local_b_rz_pn(vector<double>& b, double h_r, double h_z, int i) {
    double k = h_r * h_z / 36.;
    double r0, r1, z0, z1;
    r0 = ssrz.coord[ssrz.nvtr[i][0]-1][0];
    r1 = ssrz.coord[ssrz.nvtr[i][1]-1][0];
    z0 = ssrz.coord[ssrz.nvtr[i][0]-1][1];
    z1 = ssrz.coord[ssrz.nvtr[i][2]-1][1];
    b[0] = k*(4 * f_rz(r0, z0) + 2 * f_rz(r1, z0) + 2 * f_rz(r0, z1) + 1 *
f_rz(r1, z1));
    b[1] = k*(2 * f_rz(r0, z0) + 4 * f_rz(r1, z0) + 1 * f_rz(r0, z1) + 2 *
f_rz(r1, z1));
    b[2] = k*(2 * f_rz(r0, z0) + 1 * f_rz(r1, z0) + 4 * f_rz(r0, z1) + 2 *
f_rz(r1, z1));
    b[3] = k*(1 * f_rz(r0, z0) + 2 * f_rz(r1, z0) + 2 * f_rz(r0, z1) + 4 *
f_rz(r1, z1));
}
void FEM_N::local_b_rz_p(vector<double>& b, double hr, double hz, int i) {
    double k = hr * hz / 72.;
    double r0, r1, z0, z1;
    r0 = ssrz.coord[ssrz.nvtr[i][0] - 1][0];
    r1 = ssrz.coord[ssrz.nvtr[i][1] - 1][0];
    z0 = ssrz.coord[ssrz.nvtr[i][0] - 1][1];
    z1 = ssrz.coord[ssrz.nvtr[i][2] - 1][1];
    b[0] = k * ((2 * hr + 8 * r0) * f_rz(r0, z0) + (2 * hr + 4 * r0) * f_rz(r1,
z0) + (1 * hr + 4 * r0) * f_rz(r0, z1) + (1 * hr + 2 * r0) * f_rz(r1, z1));
    b[1] = k * ((2 * hr + 4 * r0) * f_rz(r0, z0) + (6 * hr + 8 * r0) * f_rz(r1,
z0) + (1 * hr + 2 * r0) * f_rz(r0, z1) + (3 * hr + 4 * r0) * f_rz(r1, z1));
    b[2] = k * ((1 * hr + 4 * r0) * f_rz(r0, z0) + (1 * hr + 2 * r0) * f_rz(r1,
z0) + (2 * hr + 8 * r0) * f_rz(r0, z1) + (2 * hr + 4 * r0) * f_rz(r1, z1));
    b[3] = k * ((1 * hr + 2 * r0) * f_rz(r0, z0) + (3 * hr + 4 * r0) * f_rz(r1,
z0) + (2 * hr + 4 * r0) * f_rz(r0, z1) + (6 * hr + 8 * r0) * f_rz(r1, z1));

}



void FEM_N::build_matrix_profile() {
    A.resize(0);
    A.resize(mesh.kuzlov);

    // Пройдем по элементам и занесем узлы и их соседей в соответствующие
позиции
    for (int i = 0; i < mesh.kel; i++) {
        for (int j = 0; j < FEM_XYZ_NODES_NUM; j++) {
            A[ssxyz.nvtr[i][j] - 1].insert(ssxyz.nvtr[i].begin(),
ssxyz.nvtr[i].end());
        }
    }

    ia.resize(mesh.kuzlov + 1);
    ia[0] = 1;
```

```
    ia[1] = 1;
    list<long> ja_list;

    //Построим по списку узлов A профиль матрицы
    for (int i = 1; i < mesh.kuzlov; i++) {
        //Складываем в ja все элементы строки которые находятся под диагональю
        for (set<long>::iterator it = A[i].begin(); *it < i + 1; it++) {
            ja_list.push_back(*it);
        }
        //Добавляем в ia количество считанных элементов на момент текущей строки
        ia[i + 1] = ja_list.size() + 1;
    }

    //Скопируем значения с листа в вектор
    ja.resize(ja_list.size());
    int i = 0;
    for (list<long>::iterator it = ja_list.begin(); i < ja_list.size(); it++) {
        ja[i++] = *it;
    }
}
void FEM_N::build_matrix_profile_rz() {
    A.resize(0);
    A.resize(mesh_rz.kuzlov);

    // Пройдем по элементам и занесем узлы и их соседи в соответствующие позиции
    for (int i = 0; i < mesh_rz.kel; i++) {
        for (int j = 0; j < FEM_RZ_NODES_NUM; j++) {
            A[ssrz.nvtr[i][j] - 1].insert(ssrz.nvtr[i].begin(),
ssrz.nvtr[i].end());
        }
    }

    ia.resize(mesh_rz.kuzlov + 1);
    ia[0] = 1;
    ia[1] = 1;
    list<long> ja_list;

    //Построим по списку узлов A профиль матрицы
    for (int i = 1; i < mesh_rz.kuzlov; i++) {
        //Складываем в ja все элементы строки которые находятся под диагональю
        for (set<long>::iterator it = A[i].begin(); *it < i + 1; it++) {
            ja_list.push_back(*it);
        }
        //Добавляем в ia количество считанных элементов на момент текущей строки
        ia[i + 1] = ja_list.size() + 1;
    }

    //Скопируем значения с листа в вектор
    ja.resize(ja_list.size());
    int i = 0;
    for (list<long>::iterator it = ja_list.begin(); i < ja_list.size(); it++) {
        ja[i++] = *it;
    }
}

void FEM_N::slae_init() {
    ggl.resize(0);
    di.resize(0);
    pr.resize(0);
    q.resize(0);

    ggl.resize(ja.size());
    di.resize(ia.size() - 1);
    pr.resize(di.size());
```

```
    q.resize(di.size());
    // Зададим начальное решение единицами
    for (int i = 0; i < q.size(); i++) { q[i] = 1.; }
}

void FEM_N::add_G() {
    int ggls = ggl.size();
    ggl.resize(0);
    di.resize(0);
    di.resize(mesh.kuzlov);
    ggl.resize(ggls);
    // Инициализация локального элемента
    vector<vector<double>> local_g;
    local_g.resize(FEM_XYZ_NODES_NUM);
    for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
local_g[i].resize(FEM_XYZ_NODES_NUM); }

    // Основной цикл по элементам
    double h_x, h_y, h_z, lambda;
    vector<int> buff(FEM_XYZ_NODES_NUM);
    for (int i = 0; i < mesh.kel; i++) {
        //Сортируем номера локального элемента
        buff = ssxyz.nvtr[i];
        sort(buff.begin(), buff.end());

        //Посчитаем длину шага по x, y и z
        //Для этого получим номера нижнего левого, нижнего правого и верхнего
левого узлов и посчитаем разности их координат
        h_x = ssxyz.coord[buff[1] - 1][0] - ssxyz.coord[buff[0] - 1][0];
        h_y = ssxyz.coord[buff[2] - 1][1] - ssxyz.coord[buff[0] - 1][1];
        h_z = ssxyz.coord[buff[4] - 1][2] - ssxyz.coord[buff[0] - 1][2];
        lambda = sreda.sigma[ssxyz.nvkat2d[i] - 1][SIGMA];

        //Посчитаем соответствующую локальную матрицу
        local_G(local_g, h_x, h_y, h_z, lambda);

        //Если номер столбца элемента меньше номера строки (лежит ниже
диагонали), то вносим его в разреженню матрицу
        //Если номер столбца элемента равен номеру строки (лежит на диагонали),
то складываем значение в диагональ
        for (int l = 0; l < FEM_XYZ_NODES_NUM; l++) {
            for (int m = 0; m < FEM_XYZ_NODES_NUM; m++) {
                if (buff[m] < buff[l]) { add_to_sparse(ia, ja, ggl, buff[l],
buff[m], local_g[l][m]); }
                else if (buff[m] == buff[l]) {
                    di[buff[m] - 1] += local_g[m][m];
                }
            }
        }
    }
}
void FEM_N::add_b() {
    // Номер узла источника
    int n_uzl = -1;
    // Ищем узел совпадающий с координатами источника
    for (int i = 0; i < mesh.kuzlov; i++) {
        // Сравниваем координаты узлов и источника (подразумевается что источник
расположен в координате {0,0,0})
        if (ssxyz.coord[i][0] == 0
            && ssxyz.coord[i][1] == 0
            && ssxyz.coord[i][2] == 0) {
            n_uzl = i;
            break;
        }
```

```
    }
    if (n_uzl == -1) {
        printf("Source node not found\n");
    }
    else {
        pr[n_uzl] += sreda.current_sources[0][SOURCE_POW];
        printf("Source coord: %f %f %f, Node num = %d\n",
            ssxyz.coord[n_uzl][0], ssxyz.coord[n_uzl][1], ssxyz.coord[n_uzl][2],
n_uzl+1);
    }
   /* int n_el = n_el_xyz(0, 0, 0);
    for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
        pr[ssxyz.nvtr[n_el][i]-1]+=
sreda.current_sources[0][SOURCE_POW]/FEM_XYZ_NODES_NUM;
    }*/
}
void FEM_N::add_G_sigma_diff() {
    // Инициализация локального элемента
    vector<vector<double>> local_g;
    local_g.resize(FEM_XYZ_NODES_NUM);
    for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
local_g[i].resize(FEM_XYZ_NODES_NUM); }

    // Основной цикл по элементам
    double h_x, h_y, h_z, lambda;
    vector<int> buff(FEM_XYZ_NODES_NUM);
    for (int i = 0; i < mesh.kel; i++) {
        //Сортируем номера локального элемента
        buff = ssxyz.nvtr[i];
        sort(buff.begin(), buff.end());

        //Посчитаем длину шага по x, y и z
        //Для этого получим номера нижнего левого, нижнего правого и верхнего
левого узлов и посчитаем разности их координат
        h_x = ssxyz.coord[buff[1] - 1][0] - ssxyz.coord[buff[0] - 1][0];
        h_y = ssxyz.coord[buff[2] - 1][1] - ssxyz.coord[buff[0] - 1][1];
        h_z = ssxyz.coord[buff[4] - 1][2] - ssxyz.coord[buff[0] - 1][2];
        lambda = sreda.sigma[ssxyz.nvkat2dr[i] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[i]-1][SIGMA];

        if (lambda) {
            //Посчитаем соответствующую локальную матрицу
                local_G(local_g, h_x, h_y, h_z, lambda);
            //Если номер столбца элемента меньше номера строки (лежит ниже
диагонали), то вносим его в разреженню матрицу
            //Если номер столбца элемента равен номеру строки (лежит на
диагонали), то складываем значение в диагональ
            for (int l = 0; l < FEM_XYZ_NODES_NUM; l++) {
                for (int m = 0; m < FEM_XYZ_NODES_NUM; m++) {
                    if (buff[m] < buff[l]) { add_to_sparse(ia, ja, ggl, buff[l],
buff[m], local_g[l][m]); }
                    else if (buff[m] == buff[l]) {
                        di[buff[m] - 1] += local_g[m][m];
                    }
                }
            }
        }

    }
}
void FEM_N::add_b_segregation_1() {
    /*vector<double> rz_to_xyz_solution;
    q_rz_to_xyz1(rz_to_xyz_solution);
    V1 = rz_to_xyz_solution;
```

```
    matrix_vector_mul(rz_to_xyz_solution,pr);*/
}
void FEM_N::add_b_segregation_2() {
    /*vector<double> rz_to_xyz_solution;
    q_rz_to_xyz2(rz_to_xyz_solution);
    V2 = rz_to_xyz_solution;
    matrix_vector_mul(rz_to_xyz_solution, pr);*/
}

void FEM_N::add_b_segregation_3() {
    const double Xi[3] = { -sqrt(0.6), 0,sqrt(0.6) };
    const double Ci[3] = { 5. / 9., 8. / 9.,5. / 9. };
    std::vector<vector<vector<vector<double>>>> gradV(3);
    double sigma_diff, counter = 0;
    // Обьявление для производных в точках Гаусса трехмерного элемента
    for (int i = 0; i < 3; i++) {
        gradV[i].resize(3);
        for (int j = 0; j < 3; j++) {
            gradV[i][j].resize(3);
            for (int k = 0; k < 3; k++) {
                gradV[i][j][k].resize(3);
            }
        }
    }

    for (int nel = 0; nel < mesh.kel; nel++) {
        sigma_diff = sreda.sigma[ssxyz.nvkat2dr[nel] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[nel] - 1][SIGMA];
        if (sigma_diff) {
            counter++;
            double
                x1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][0], x2 =
ssxyz.coord[ssxyz.nvtr[nel][1] - 1][0],
                y1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][1], y2 =
ssxyz.coord[ssxyz.nvtr[nel][2] - 1][1],
                z1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][2], z2 =
ssxyz.coord[ssxyz.nvtr[nel][4] - 1][2];
            double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;
            double cx = (x2 + x1) / 2., cy = (y2 + y1) / 2., cz = (z2 + z1) /
2.;

            //double dotx, doty, dotz;
            //sigma_diff = 1.;

            // Считаем производные в точках Гаусса
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    for (int k = 0; k < 3; k++) {
                        //dotx = cx + Xi[i] * hx / 2.; doty = cy + Xi[j] * hy /
2.; dotz = cz + Xi[k] * hz / 2.;
                        grad_rz_in_point(cx + Xi[i] * hx / 2., cy + Xi[j] * hy /
2., cz + Xi[k] * hz / 2., gradV[i][j][k]);
                        //gradV[i][j][k] = { doty * dotz,dotx * dotz,dotx * doty
};
                    }
                }
            }

            double dpsidx, dpsidy, dpsidz, sx, sy, sz, Gaus, f;
            for (int nuzl = 0; nuzl < FEM_XYZ_NODES_NUM; nuzl++) {
                Gaus = 0;
                // Неконстантная часть частных производных по базисным функциям
                if (nuzl % 2 == 0) { dpsidx = -1. / hx; sx = -1.; }
                else { dpsidx = 1. / hx; sx = 1.; }
```

```
                if (nuzl / 2 % 2 == 0) { dpsidy = -1. / hy; sy = -1.; }
                else { dpsidy = 1. / hy; sy = 1.; }
                if (nuzl / 4 % 2 == 0) { dpsidz = -1. / hz; sz = -1.; }
                else { dpsidz = 1. / hz; sz = 1.; }

                for (int i = 0; i < 3; i++) {
                    for (int j = 0; j < 3; j++) {
                        for (int k = 0; k < 3; k++) {
                            f = gradV[i][j][k][0] * dpsidx * (1 + sy * Xi[j]) *
(1 + sz * Xi[k]) / 4. +
                                gradV[i][j][k][1] * dpsidy * (1 + sx * Xi[i]) *
(1 + sz * Xi[k]) / 4. +
                                gradV[i][j][k][2] * dpsidz * (1 + sx * Xi[i]) *
(1 + sy * Xi[j]) / 4.;
                            /*f = dpsidx * (1 + sy * Xi[j]) * (1 + sz * Xi[k]) /
4. +
                                dpsidy * (1 + sx * Xi[i]) * (1 + sz * Xi[k]) /
4. +
                                dpsidz * (1 + sx * Xi[i]) * (1 + sy * Xi[j]) /
4.;*/
                            Gaus += Ci[i] * Ci[j] * Ci[k] * f;
                        }
                    }
                }
                Gaus *= hx * hy * hz / 8.;
                pr[ssxyz.nvtr[nel][nuzl] - 1] += Gaus * sigma_diff;
                //pr[ssxyz.nvtr[nel][nuzl] - 1] += Gaus*3;
            }
        }
    }
    printf("non zero elements = %f\n", counter);
}
void FEM_N::add_b_segregation_4() {
    double sigma_diff, counter = 0;
    vector<double> gradV(3);
    for (int nel = 0; nel < mesh.kel; nel++) {
        sigma_diff = sreda.sigma[ssxyz.nvkat2dr[nel] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[nel] - 1][SIGMA];
        if (sigma_diff) {
            counter++;
            double
                x1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][0], x2 =
ssxyz.coord[ssxyz.nvtr[nel][1] - 1][0],
                y1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][1], y2 =
ssxyz.coord[ssxyz.nvtr[nel][2] - 1][1],
                z1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][2], z2 =
ssxyz.coord[ssxyz.nvtr[nel][4] - 1][2];
            double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;
            double cx = (x2 + x1) / 2., cy = (y2 + y1) / 2., cz = (z2 + z1) /
2.;

            // Считаем производную в центре элемента
            grad_rz_in_point(cx, cy, cz, gradV);
            //gradV = { 1.00100025,1.00100025,1.00100025};
            double sx, sy, sz, Integral;
            for (int nuzl = 0; nuzl < FEM_XYZ_NODES_NUM; nuzl++) {
                (nuzl % 2 == 0) ? sx = -1. : sx = 1.;
                (nuzl / 2 % 2 == 0) ? sy = -1. : sy = 1.;
                (nuzl / 4 % 2 == 0) ? sz = -1. : sz = 1;

                Integral = sigma_diff * (gradV[0] * sx * hy * hz / 4. + gradV[1]
* sy * hx * hz / 4. + gradV[2] * sz * hx * hy / 4.);
                pr[ssxyz.nvtr[nel][nuzl] - 1] += Integral;
            }
```

```
        }
    }
    printf("non zero elements = %f\n", counter);
}
void FEM_N::add_b_segregation_5() {
    double sigma_diff, counter = 0;
    vector<vector<double>> gradV(FEM_XYZ_NODES_NUM); // Вектор произвожных по
x,y и z в каждом узле элемента
    for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) { gradV[i].resize(3); }
    for (int nel = 0; nel < mesh.kel; nel++) {
        sigma_diff = sreda.sigma[ssxyz.nvkat2dr[nel] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[nel] - 1][SIGMA];
        if (sigma_diff) {
            counter++;
            double
                x1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][0], x2 =
ssxyz.coord[ssxyz.nvtr[nel][1] - 1][0],
                y1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][1], y2 =
ssxyz.coord[ssxyz.nvtr[nel][2] - 1][1],
                z1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][2], z2 =
ssxyz.coord[ssxyz.nvtr[nel][4] - 1][2];
            double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;
            double cx = (x2 + x1) / 2., cy = (y2 + y1) / 2., cz = (z2 + z1) /
2.;

            // Считаем производные
            grad_rz_in_point(x1, y1, z1, gradV[0]);
            grad_rz_in_point(x2, y1, z1, gradV[1]);
            grad_rz_in_point(x1, y2, z1, gradV[2]);
            grad_rz_in_point(x2, y2, z1, gradV[3]);
            grad_rz_in_point(x1, y1, z2, gradV[4]);
            grad_rz_in_point(x2, y1, z2, gradV[5]);
            grad_rz_in_point(x1, y2, z2, gradV[6]);
            grad_rz_in_point(x2, y2, z2, gradV[7]);

            /*gradV[0] = { 1,1,1 };
            gradV[1] = { 1,1.001,1.001 };
            gradV[2] = { 1.001,1.,1.001};
            gradV[3] = { 1.001,1.,1.002001};
            gradV[4] = { 1.001,1.001,1.};
            gradV[5] = { 1.001,1.002001,1.001};
            gradV[6] = { 1.002001,1.001,1.001};
            gradV[7] = { 1.002001,1.001,1.002001};*/
            double sx, sy, sz, Integral;
            for (int nuzl = 0; nuzl < FEM_XYZ_NODES_NUM; nuzl++) {
                (nuzl % 2 == 0) ? sx = -1. : sx = 1.;
                (nuzl / 2 % 2 == 0) ? sy = -1. : sy = 1.;
                (nuzl / 4 % 2 == 0) ? sz = -1. : sz = 1;
                Integral = sigma_diff * (
                    gradV[nuzl][0] * sx * hy * hz / 4. +
                    gradV[nuzl][1] * sy * hx * hz / 4. +
                    gradV[nuzl][2] * sz * hx * hy / 4.);
                if (Integral != Integral) { printf("Nan %f %f %f : %f %f %f\n",
x1, y1, z1,x2,y2,z2); }
                pr[ssxyz.nvtr[nel][nuzl] - 1] += Integral;
            }
        }
    }
    printf("non zero elements = %f\n", counter);
}

void FEM_N::add_b_segregation_3(int NOMEL) {
    const double Xi[3] = { -sqrt(0.6), 0,sqrt(0.6) };
    const double Ci[3] = { 5. / 9., 8. / 9.,5. / 9. };
```

```
    std::vector<vector<vector<vector<double>>>> gradV(3);
    double sigma_diff, counter = 0;
    // Обьявление для производных в точках Гаусса трехмерного элемента
    for (int i = 0; i < 3; i++) {
        gradV[i].resize(3);
        for (int j = 0; j < 3; j++) {
            gradV[i][j].resize(3);
            for (int k = 0; k < 3; k++) {
                gradV[i][j][k].resize(3);
            }
        }
    }

    for (int nel = 0; nel < mesh.kel; nel++) {
        sigma_diff = sreda.sigma[ssxyz.nvkat2dr[nel] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[nel] - 1][SIGMA];
        if (sigma_diff) {
            counter++;
            double
                x1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][0], x2 =
ssxyz.coord[ssxyz.nvtr[nel][1] - 1][0],
                y1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][1], y2 =
ssxyz.coord[ssxyz.nvtr[nel][2] - 1][1],
                z1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][2], z2 =
ssxyz.coord[ssxyz.nvtr[nel][4] - 1][2];
            double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;
            double cx = (x2 + x1) / 2., cy = (y2 + y1) / 2., cz = (z2 + z1) /
2.;

            // Считаем производные в точках Гаусса
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    for (int k = 0; k < 3; k++) {
                        grad_rz_in_point(cx + Xi[i] * hx / 2., cy + Xi[j] * hy /
2., cz + Xi[k] * hz / 2., gradV[i][j][k]);
                        if (nel == NOMEL) { printf("i - %d, j - %d, k - %d  ||
dx = %e | dy = %e | dz = %e\n", i,j,k,gradV[i][j][k][0], gradV[i][j][k][1],
gradV[i][j][k][2]); }
                    }
                }
            }

            double dpsidx, dpsidy, dpsidz, sx, sy, sz, Gaus, f;
            for (int nuzl = 0; nuzl < FEM_XYZ_NODES_NUM; nuzl++) {
                Gaus = 0;
                // Неконстантная часть частных производных по базисным функциям
                if (nuzl % 2 == 0) { dpsidx = -1. / hx; sx = -1.; }
                else { dpsidx = 1. / hx; sx = 1.; }
                if (nuzl / 2 % 2 == 0) { dpsidy = -1. / hy; sy = -1.; }
                else { dpsidy = 1. / hy; sy = 1.; }
                if (nuzl / 4 % 2 == 0) { dpsidz = -1. / hz; sz = -1.; }
                else { dpsidz = 1. / hz; sz = 1.; }

                for (int i = 0; i < 3; i++) {
                    for (int j = 0; j < 3; j++) {
                        for (int k = 0; k < 3; k++) {
                            f = gradV[i][j][k][0] * dpsidx * (1 + sy * Xi[j]) *
(1 + sz * Xi[k]) / 4. +
                                gradV[i][j][k][1] * dpsidy * (1 + sx * Xi[i]) *
(1 + sz * Xi[k]) / 4. +
                                gradV[i][j][k][2] * dpsidz * (1 + sx * Xi[i]) *
(1 + sy * Xi[j]) / 4.;
                            /*f = dpsidx * (1 + sy * Xi[j]) * (1 + sz * Xi[k]) /
4. +
```

```
                                dpsidy * (1 + sx * Xi[i]) * (1 + sz * Xi[k]) /
4. +
                                dpsidz * (1 + sx * Xi[i]) * (1 + sy * Xi[j]) /
4.;*/
                            Gaus += Ci[i] * Ci[j] * Ci[k] * f;
                        }
                    }
                }
                Gaus *= hx * hy * hz / 8.;
                pr[ssxyz.nvtr[nel][nuzl] - 1] += Gaus * sigma_diff;
                //pr[ssxyz.nvtr[nel][nuzl] - 1] += Gaus*3;
            }
        }
    }
    printf("non zero elements = %f\n", counter);
}
void FEM_N::add_b_segregation_4(int NOMEL) {
    double sigma_diff, counter = 0;
    vector<double> gradV(3);
    for (int nel = 0; nel < mesh.kel; nel++) {
        sigma_diff = sreda.sigma[ssxyz.nvkat2dr[nel] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[nel] - 1][SIGMA];
        if (sigma_diff) {
            counter++;
            double
                x1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][0], x2 =
ssxyz.coord[ssxyz.nvtr[nel][1] - 1][0],
                y1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][1], y2 =
ssxyz.coord[ssxyz.nvtr[nel][2] - 1][1],
                z1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][2], z2 =
ssxyz.coord[ssxyz.nvtr[nel][4] - 1][2];
            double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;
            double cx = (x2 + x1) / 2., cy = (y2 + y1) / 2., cz = (z2 + z1) /
2.;

            // Считаем производную в центре элемента
            grad_rz_in_point(cx, cy, cz, gradV);
            if (nel == NOMEL) { printf("dx = %e | dy = %e | dz = %e\n",
gradV[0], gradV[1], gradV[2]); }

            double sx, sy, sz, Integral;
            for (int nuzl = 0; nuzl < FEM_XYZ_NODES_NUM; nuzl++) {
                (nuzl % 2 == 0) ? sx = -1. : sx = 1.;
                (nuzl / 2 % 2 == 0) ? sy = -1. : sy = 1.;
                (nuzl / 4 % 2 == 0) ? sz = -1. : sz = 1;

                Integral = sigma_diff * (gradV[0] * sx * hy * hz / 4. + gradV[1]
* sy * hx * hz / 4. + gradV[2] * sz * hx * hy / 4.);
                if (nel == NOMEL) { printf("local b %d = %e\n", nuzl, Integral);
}
                pr[ssxyz.nvtr[nel][nuzl] - 1] += Integral;
            }
        }
    }
    printf("non zero elements = %f\n", counter);
}
void FEM_N::add_b_segregation_5(int NOMEL) {
    double sigma_diff, counter = 0;
    vector<vector<double>> gradV(FEM_XYZ_NODES_NUM); // Вектор произвожных по
x,y и z в каждом узле элемента
    for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) { gradV[i].resize(3); }
    for (int nel = 0; nel < mesh.kel; nel++) {
        sigma_diff = sreda.sigma[ssxyz.nvkat2dr[nel] - 1][SIGMA] -
sreda.sigma[ssxyz.nvkat2d[nel] - 1][SIGMA];
```

```
        if (sigma_diff) {
            counter++;
            double
                x1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][0], x2 =
ssxyz.coord[ssxyz.nvtr[nel][1] - 1][0],
                y1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][1], y2 =
ssxyz.coord[ssxyz.nvtr[nel][2] - 1][1],
                z1 = ssxyz.coord[ssxyz.nvtr[nel][0] - 1][2], z2 =
ssxyz.coord[ssxyz.nvtr[nel][4] - 1][2];
            double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;
            double cx = (x2 + x1) / 2., cy = (y2 + y1) / 2., cz = (z2 + z1) /
2.;

            // Считаем производные
            grad_rz_in_point(x1, y1, z1, gradV[0]);
            grad_rz_in_point(x2, y1, z1, gradV[1]);
            grad_rz_in_point(x1, y2, z1, gradV[2]);
            grad_rz_in_point(x2, y2, z1, gradV[3]);
            grad_rz_in_point(x1, y1, z2, gradV[4]);
            grad_rz_in_point(x2, y1, z2, gradV[5]);
            grad_rz_in_point(x1, y2, z2, gradV[6]);
            grad_rz_in_point(x2, y2, z2, gradV[7]);

            double sx, sy, sz, Integral;
            for (int nuzl = 0; nuzl < FEM_XYZ_NODES_NUM; nuzl++) {
                if (nel == NOMEL) { printf("%d -  || dx = %e | dy = %e | dz =
%e\n",nuzl, gradV[nuzl][0], gradV[nuzl][1], gradV[nuzl][2]); }

                (nuzl % 2 == 0) ? sx = -1. : sx = 1.;
                (nuzl / 2 % 2 == 0) ? sy = -1. : sy = 1.;
                (nuzl / 4 % 2 == 0) ? sz = -1. : sz = 1;
                Integral = sigma_diff * (
                    gradV[nuzl][0] * sx * hy * hz / 4. +
                    gradV[nuzl][1] * sy * hx * hz / 4. +
                    gradV[nuzl][2] * sz * hx * hy / 4.);
                if (Integral != Integral) { printf("Nan %f %f %f : %f %f %f\n",
x1, y1, z1, x2, y2, z2); }
                pr[ssxyz.nvtr[nel][nuzl] - 1] += Integral;
            }
        }
    }
    printf("non zero elements = %f\n", counter);
}

// Добавит матрцу жесткости и вектор правой части для двумерной задачи в
разреженном строчно-столбцовом формате
void FEM_N::add_G_b_rz()
{
    // Инициализация локального элемента
    vector<vector<double>> local_G;
    local_G.resize(FEM_RZ_NODES_NUM);
    for (int i = 0; i < FEM_RZ_NODES_NUM; i++) {
local_G[i].resize(FEM_RZ_NODES_NUM); }

    // Основной цикл по элементам
    double h_r, h_z, lambda;
    vector<int> buff(FEM_RZ_NODES_NUM);
    for (int i = 0; i < mesh_rz.kel; i++) {
        //Сортируем номера локального элемента
        buff = ssrz.nvtr[i];
        sort(buff.begin(), buff.end());

        //Посчитаем длину шага по x и y
```

```
        //Для этого получим номера нижнего левого, нижнего правого и верхнего
левого узлов и посчитаем разности их координат
        h_r = ssrz.coord[buff[1] - 1][0] - ssrz.coord[buff[0] - 1][0];
        h_z = ssrz.coord[buff[2] - 1][1] - ssrz.coord[buff[0] - 1][1];
        lambda = sreda.sigma[ssrz.nvkat2d[i] - 1][SIGMA];

        //Посчитаем соответствующую локальную матрицу
        local_G_rz(local_G, h_r, h_z, lambda,i);

        //Если номер столбца элемента меньше номера строки (лежит ниже
диагонали), то вносим его в разреженню матрицу
        //Если номер столбца элемента равен номеру строки (лежит на диагонали),
то складываем значение в диагональ
        for (int l = 0; l < FEM_RZ_NODES_NUM; l++) {
            for (int m = 0; m < FEM_RZ_NODES_NUM; m++) {
                if (buff[m] < buff[l]) { add_to_sparse(ia, ja, ggl, buff[l],
buff[m], local_G[l][m]); }
                else if (buff[m] == buff[l]) {
                    di[buff[m] - 1] += local_G[m][m];
                }
            }
        }

    }

    // Для учета правой части
    /*
    Для учета тока мы ставим ненулевое значение тока в первый узел

        ->  0-------------> r
            |         |
            |         |
            |         |
            |         |
            |----------
            |
            z
    Получим решение для I=1 затем, т.к. решение линейно по току, мы получим
решение для необходимой силы тока
    умножением решения на константу
    */
    pr[0] = I / (2. * 3.141592653589793l);
}


// Добавит правую часть в вектор FEM.pr тест на полином
void FEM_N::add_G_b_p()
{
    // Инициализация локального элемента
    vector<vector<double>> local_g;
    local_g.resize(FEM_XYZ_NODES_NUM);
    for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
local_g[i].resize(FEM_XYZ_NODES_NUM); }

    //Инициализация локальной правой части
    vector<double> local_b;
    local_b.resize(FEM_XYZ_NODES_NUM);

    // Основной цикл по элементам
    double h_x, h_y, h_z, lambda;
    vector<int> buff(FEM_XYZ_NODES_NUM);
    for (int i = 0; i < mesh.kel; i++) {
        //Сортируем номера локального элемента
        buff = ssxyz.nvtr[i];
```

```
        sort(buff.begin(), buff.end());

        //Посчитаем длину шага по x, y и z
        //Для этого получим номера нижнего левого, нижнего правого и верхнего
левого узлов и посчитаем разности их координат
        h_x = ssxyz.coord[buff[1] - 1][0] - ssxyz.coord[buff[0] - 1][0];
        h_y = ssxyz.coord[buff[2] - 1][1] - ssxyz.coord[buff[0] - 1][1];
        h_z = ssxyz.coord[buff[4] - 1][2] - ssxyz.coord[buff[0] - 1][2];
        lambda = sreda.sigma[ssxyz.nvkat2d[i] - 1][SIGMA];

        //Посчитаем соответствующую локальную матрицу
        local_G(local_g, h_x, h_y, h_z, lambda);

        //Если номер столбца элемента меньше номера строки (лежит ниже
диагонали), то вносим его в разреженню матрицу
        //Если номер столбца элемента равен номеру строки (лежит на диагонали),
то складываем значение в диагональ
        for (int l = 0; l < FEM_XYZ_NODES_NUM; l++) {
            for (int m = 0; m < FEM_XYZ_NODES_NUM; m++) {
                if (buff[m] < buff[l]) { add_to_sparse(ia, ja, ggl, buff[l],
buff[m], local_g[l][m]); }
                else if (buff[m] == buff[l]) {
                    di[buff[m] - 1] += local_g[m][m];
                }
            }
        }

        local_b_p(local_b, h_x, h_y, h_z, i);
        for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) { pr[buff[i] - 1] +=
local_b[i]; }
    }
}
void FEM_N::add_G_b_rz_p()
{
    // Инициализация локального элемента
    vector<vector<double>> local_G;
    local_G.resize(FEM_RZ_NODES_NUM);
    for (int i = 0; i < FEM_RZ_NODES_NUM; i++) { local_G[i].resize(4); }

    //Инициализация локальной правой части
    vector<double> local_b;
    local_b.resize(FEM_RZ_NODES_NUM);

    // Основной цикл по элементам
    double h_x, h_y, lambda;
    vector<int> buff(FEM_RZ_NODES_NUM);
    for (int i = 0; i < mesh_rz.kel; i++) {
        //Сортируем номера локального элемента
        buff = ssrz.nvtr[i];
        sort(buff.begin(), buff.end());

        //Посчитаем длину шага по x и y
        //Для этого получим номера нижнего левого, нижнего правого и верхнего
левого узлов и посчитаем разности их координат
        h_x = ssrz.coord[buff[1] - 1][0] - ssrz.coord[buff[0] - 1][0];
        h_y = ssrz.coord[buff[2] - 1][1] - ssrz.coord[buff[0] - 1][1];
        lambda = sreda.sigma[ssrz.nvkat2d[i] - 1][SIGMA];

        //Посчитаем соответствующую локальную матрицу
        local_G_rz(local_G, h_x, h_y, lambda,i);

        //Если номер столбца элемента меньше номера строки (лежит ниже
диагонали), то вносим его в разреженню матрицу
```

```
        //Если номер столбца элемента равен номеру строки (лежит на диагонали),
то складываем значение в диагональ
        for (int l = 0; l < FEM_RZ_NODES_NUM; l++) {
            for (int m = 0; m < FEM_RZ_NODES_NUM; m++) {
                if (buff[m] < buff[l]) { add_to_sparse(ia, ja, ggl, buff[l],
buff[m], local_G[l][m]); }
                else if (buff[m] == buff[l]) {
                    di[buff[m] - 1] += local_G[m][m];
                }
            }
        }

        // Учет правой части
        local_b_rz_p(local_b,h_x,h_y,i);
        for (int i = 0; i < FEM_RZ_NODES_NUM; i++) { pr[buff[i] - 1] +=
local_b[i]; }
    }
}

void FEM_N::edge_cond_1() {
    // Для сохранения симметричности матрицы будем также занулять столбцы вместе
со строками
    // при этом pr = pr - a*u, но т.к. u = 0 ( условие на удаленной границе), то
правая часть не изменится при занулении столбцов
    for (int i = 0; i < ssxyz.l1.size(); i++) {
        int node = ssxyz.l1[i] - 1;
        di[ssxyz.l1[i] - 1] = 1.;
        pr[ssxyz.l1[i] - 1] = 0;

        // Зануляем строку нижнего треугольника ( и столбец верхнего
треугольника т.к. матрица симметричная)
        long start = ia[ssxyz.l1[i] - 1] - 1, end = ia[ssxyz.l1[i]] - 1;
        for (long j = start; j < end; j++) {
            ggl[j] = 0;
        }

        // Зануляем столбец нижнего треугольника ( и строку верхнего
треугольника т.к. матрицы симметричная)
        /*for (int j = ssxyz.l1[i] + 1; j <= di.size(); j++) {
            replace_in_sparse(ia, ja, ggl, j, ssxyz.l1[i], 0);
        }*/
        for (std::set<long>::iterator it = A[node].begin(); it != A[node].end();
it++) {
            if (node + 1 < *it) { // Если узел лежит в верхнем треугольнике
                replace_in_sparse(ia, ja, ggl, *it, node + 1, 0);
            }
        }

    }
}
void FEM_N::edge_cond_1_rz() {
    // Для сохранения симметричности матрицы будем также занулять столбцы вместе
со строками
    // при этом pr = pr - a*u, но т.к. u = 0 ( условие на удаленной границе), то
правая часть не изменится при занулении столбцов
    for (int i = 0; i < ssrz.l1.size(); i++) {
        int node = ssrz.l1[i] - 1;
        di[ssrz.l1[i] - 1] = 1.;
        pr[ssrz.l1[i] - 1] = 0;

        // Зануляем строку нижнего треугольника ( и столбец верхнего
треугольника т.к. матрица симметричная)
        long start = ia[ssrz.l1[i] - 1] - 1, end = ia[ssrz.l1[i]] - 1;
        for (long j = start; j < end; j++) {
```

```
            ggl[j] = 0;
        }

        // Зануляем столбец нижнего треугольника ( и строку верхнего
треугольника т.к. матрицы симметричная)
        /*for (int j = ssrz.l1[i] + 1; j <= di.size(); j++) {
            replace_in_sparse(ia, ja, ggl, j, ssrz.l1[i], 0);
        }*/
        for (std::set<long>::iterator it = A[node].begin(); it != A[node].end();
it++) {
            if (node + 1 < *it) { // Если узел лежит в верхнем треугольнике
                replace_in_sparse(ia, ja, ggl, *it, node + 1, 0);
            }
        }
    }
}

void FEM_N::print_matrix_plenum() {
    int nonzero = 0, dis = di.size(), ggls=ggl.size();
    for (int i = 0; i < ggl.size(); i++) { if (ggl[i]) { nonzero++; } }
    double ggl_matrix, nonzero_matrix, nonzero_ggl;
    ggl_matrix = (double)ggls/(double)(dis*dis);
    nonzero_matrix = (double)nonzero / (double)(dis * dis);
    nonzero_ggl = (double)nonzero /(double)ggls;
    printf("GGL/MATRIX = %f%% | NONZERO/MATRIX = %f%% | NONZERO/GGL = %.2f%% |
ggl size = %d\n",
        ggl_matrix * 100., nonzero_matrix * 100., nonzero_ggl * 100.,ggls);
}

void FEM_N::edge_cond_1_p() {
    vector<double> coords;
    double reminder, u_f;
    int node,col;

    // Для сохранения симметричности матрицы будем также занулять столбцы вместе
со строками
    // при этом pr = pr - a*u
    for (int i = 0; i < ssxyz.l1.size(); i++) {
        node = ssxyz.l1[i] - 1;
        di[node] = 1.;
        coords = ssxyz.coord[node];
        u_f = u(coords[0], coords[1],coords[2]);

        // Зануляем строку нижнего треугольника ( и столбец верхнего
треугольника т.к. матрица симметричная)
        // учитываем замененное значение в правой части
        long start = ia[node] - 1, end = ia[node+1] - 1;
        for (long j = start; j < end; j++) {
            pr[ja[j] - 1] -= ggl[j] * u_f;
            ggl[j] = 0;
        }

        // Зануляем столбец нижнего треугольника ( и строку верхнего
треугольника т.к. матрицы симметричная)
        // учитываем замененное значение в правой части

        /*for (int j = ssxyz.l1[i] + 1; j <= di.size(); j++) {
            reminder = replace_in_sparse_r(ia, ja, ggl, j, ssxyz.l1[i], 0);
            pr[j - 1] -= reminder * u_f;
        }*/

        for (std::set<long>::iterator it = A[node].begin();
it!=A[node].end();it++){
            if (node + 1 < *it) { // Если узел лежит в верхнем треугольнике
```

```
                reminder = replace_in_sparse_r(ia, ja, ggl, *it, node + 1, 0);
                pr[*it-1] -= reminder * u_f;
            }
        }
    }

    // После того как мы учли зануленные столбцы в правой части
    // заменим правую часть первым краевым
    for (int i = 0; i < ssxyz.l1.size(); i++) {
        coords = ssxyz.coord[ssxyz.l1[i] - 1];
        pr[ssxyz.l1[i] - 1] = u(coords[0], coords[1],coords[2]);
    }
}
void FEM_N::edge_cond_1_rz_p() {
    vector<double> coords;
    double reminder, u;
    int node, col;

    // Для сохранения симметричности матрицы будем также занулять столбцы вместе
со строками
    // при этом pr = pr - a*u
    for (int i = 0; i < ssrz.l1.size(); i++) {
        node = ssrz.l1[i] - 1;
        di[node] = 1.;
        coords = ssrz.coord[node];
        u = u_rz(coords[0], coords[1]);

        // Зануляем строку нижнего треугольника ( и столбец верхнего
треугольника т.к. матрица симметричная)
        // учитываем замененное значение в правой части
        long start = ia[node] - 1, end = ia[node+1] - 1;
        for (long j = start; j < end; j++) {
            pr[ja[j]-1] -= ggl[j] * u;
            ggl[j] = 0;
        }

        // Зануляем столбец нижнего треугольника ( и строку верхнего
треугольника т.к. матрицы симметричная)
        // учитываем замененное значение в правой части
        /*for (int j = ssrz.l1[i] + 1; j <= di.size(); j++) {
            reminder = replace_in_sparse_r(ia, ja, ggl, j, ssrz.l1[i], 0);
            pr[j - 1] -= reminder * u;
        }*/
        for (std::set<long>::iterator it = A[node].begin(); it != A[node].end();
it++) {
            if (node + 1 < *it) { // Если узел лежит в верхнем треугольнике
                reminder = replace_in_sparse_r(ia, ja, ggl, *it, node + 1, 0);
                pr[*it - 1] -= reminder * u;
            }
        }
    }

    // После того как мы учли зануленные столбцы в правой части
    // заменим правую часть первым краевым
    for (int i = 0; i < ssrz.l1.size(); i++) {
        coords = ssrz.coord[ssrz.l1[i] - 1];
        pr[ssrz.l1[i] - 1] = u_rz(coords[0], coords[1]);
    }
}

FEM_N::FEM_N(string filename) {
      sreda.read_problem(filename);
      mesh.gen_mesh(sreda);
```

```
    sreda_n.read_problem_n(filename);
    mesh_n.gen_mesh(sreda_n);
    mesh_n.add_mesh(mesh); // Прибавим узлы

      sreda_rz.read_sreda(filename);
      mesh_rz.gen_mesh(sreda_rz, sreda);
    printf("Edge conditions: X -%f,%f | Y - %f,%f | Z0 - %d, Z1 - %d\n\n",
        sreda.edge_conditions[0], sreda.edge_conditions[1],
sreda.edge_conditions[2],
        sreda.edge_conditions[3], sreda.edge_conditions[4],
sreda.edge_conditions[5]);
}

void FEM_N::solve(int max_iter, double eps, double relax, std::string
solution_filename) {
    printf("\nSOLVE XYZ-------------------------\n");
    ssxyz.gen_structures(sreda, mesh);

    unsigned int start = clock();
    build_matrix_profile();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    add_G();
    add_b();
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    print_matrix_plenum();

    start = clock();
    edge_cond_1();
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    solve_cgm(q, max_iter, eps, relax);
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);

    solution_export(mesh,solution_filename,q);
    printf("END XYZ--------------------\n");
}

int FEM_N::find_nel(double x, double y, double z, class MESH& mesh, class
STRCTRS& ss, std::vector<double>& q) {
    int res = 0;
    //static int xi = 0, yi = 0, zi = 0; // Индексы нижних границ в сетке
    if (mesh.x[0] <= x && x <= mesh.x[mesh.x.size() - 1] &&
        mesh.y[0] <= y && y <= mesh.y[mesh.y.size() - 1] &&
        mesh.z[0] <= z && z <= mesh.z[mesh.z.size() - 1]) {// Если точка лежит
внутри
        while (!(mesh.x[xi] <= x && x <= mesh.x[xi + 1])) {
            if (x > mesh.x[xi + 1]) { xi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (x < mesh.x[xi]) { xi--; } // Если точка левее текущего отрезка,
то сместить отрезок влево
        }
        while (!(mesh.y[yi] <= y && y <= mesh.y[yi + 1])) {
            if (y > mesh.y[yi + 1]) { yi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
```

```
            if (y < mesh.y[yi]) { yi--; } // Если точка левее текущего отрезка,
то сместить отрезок влево
        }
        while (!(mesh.z[zi] <= z && z <= mesh.z[zi + 1])) {
            if (z > mesh.z[zi + 1]) { zi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (z < mesh.z[zi]) { zi--; } // Если точка левее текущего отрезка,
то сместить отрезок влево
        }
        int n_el = (mesh.x.size() - 1) * (mesh.y.size() - 1) * zi +
(mesh.x.size() - 1) * yi + xi; // Номер элемента в котором содержится точка
       /* double hx = mesh.x[xi + 1] - mesh.x[xi], hy = mesh.y[yi + 1] -
mesh.y[yi], hz = mesh.z[zi + 1] - mesh.z[zi];
        double X1 = (mesh.x[xi + 1] - x) / hx, X2 = (x - mesh.x[xi]) / hx;
        double Y1 = (mesh.y[yi + 1] - y) / hy, Y2 = (y - mesh.y[yi]) / hy;
        double Z1 = (mesh.z[zi + 1] - z) / hz, Z2 = (z - mesh.z[zi]) / hz;
        vector<double> psi = {
            X1 * Y1 * Z1,X2 * Y1 * Z1,
            X1 * Y2 * Z1,X2 * Y2 * Z1,
            X1 * Y1 * Z2,X2 * Y1 * Z2,
            X1 * Y2 * Z2,X2 * Y2 * Z2 };*/
        for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
            printf("Qi = %e\n", q[ss.nvtr[n_el][i] - 1]);
        }
        res = n_el;
    }
    else {
        res = -1;
    }
    return res;
}
void FEM_N::solve_segregation1(int max_iter, double eps, double relax,
std::string solution_filename) {
    printf("\nSOLVE XYZ: SEGREGATION 1-------------------------\n");
    unsigned int solve_segregation_start = clock();
    unsigned int start = clock();
    ssxyz.gen_structures(sreda, mesh);
    printf("gen_structures sreda -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    ssxyzn.gen_structures(sreda_n, mesh_n);
    printf("gen_structures sreda_n -     %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    build_matrix_profile();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    // Сперва построим матрицу для sigma = sigma(normal) - sigma, ее мы умножим
на вектор
    add_G_sigma_diff();
    q_rz_to_xyz1(Vnb,mesh,ssxyz); // Получим решение двумерной задачи для
трехмерной сетки из файла sreda
    matrix_vector_mul(Vnb, pr); // Умножим глобальную матрицу на вектор Vnb, это
будет правая часть для задачи с выделением
    /*int bel = find_nel(P_X, P_Y, P_Z, mesh, ssxyz, Vnb);
    for (int i = 0; i < 8; i++) { printf("bi = %e\n", pr[ssxyz.nvtr[bel][i] -
1]); }
```

```
    printf("b in point: %e\n", solution_xyz_in_point(P_X, P_Y, P_Z, pr, mesh,
ssxyz));
    xi = 0; yi = 0; zi = 0;*/
    //for (int i = 0; i < pr.size(); i++) { printf("pr%d = %e\n", i, pr[i]); }

    add_G(); // Матрица жесткости
    printf("global G and b -            %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    edge_cond_1(); // Применение первых нулевых краевых условий
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    solve_cgm(Va, max_iter, eps, relax); // Решение задачи на добавочное поле в
узлах sreda
    printf("SLAE solution -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    q_rz_to_xyz1(Vns, mesh_n, ssxyzn); // Получим решение двумерной задачи для
трехмерной сетки sreda+sreda_n
    q_xyz_to_xyz(Va,mesh, ssxyz,Vas, mesh_n,ssxyzn); // Решение задачи с
аномальным полем расширить до большей сетки
    vec_vec_sum(Vas, Vns, Vs); // Сложим решения q = Va + Vн

    solution_export(mesh_n, solution_filename + "vn", Vns); // Выведем Vn
    solution_export(mesh_n, solution_filename + "va", Vas); // Выведем Va
    solution_export(mesh_n,solution_filename, Vs); // Выведем суммарное решение
для общей сетки
    printf("SOLVE SEGREGATION -         %.2f sec\n", (double)(clock() -
solve_segregation_start) / 1000.);
    printf("END XYZ: SEGREGATION 1--------------------\n");
}

void FEM_N::print_matrix() {
    int jaind = 0;
    for (int i = 0; i < 8; i++) {
        int elms = ia[i + 1] - ia[i];
        for (int j = 0; j < 8; j++) {
            if (elms) {
                if (j == ja[jaind] - 1) {
                    printf("%.3e ", ggl[jaind]);
                    jaind++;
                }
                else {
                    printf("%.3e ", 0);
                }
                elms--;
            }
            else {
                printf("%.3e ", 0);
            }
        }
        printf("\n");
    }
}

void FEM_N::solve_segregation2(int max_iter, double eps, double relax,
std::string solution_filename) {
    printf("\nSOLVE XYZ: SEGREGATION 2-------------------------\n");
    unsigned int solve_segregation_start = clock();
    unsigned int start = clock();
    ssxyz.gen_structures(sreda, mesh);
```

```
    printf("gen_structures sreda -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    ssxyzn.gen_structures(sreda_n, mesh_n);
    printf("gen_structures sreda_n -     %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    build_matrix_profile();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    // Сперва построим матрицу для sigma = sigma(normal) - sigma, ее мы умножим
на вектор
    add_G_sigma_diff();
    q_rz_to_xyz2(Vnb, mesh, ssxyz); // Получим решение двумерной задачи для
трехмерной сетки из файла sreda
    //Vnb = { 1,1.001,1.001,1.002001,1.001,1.002001,1.002001,1.003003001};
    matrix_vector_mul(Vnb, pr); // Умножим глобальную матрицу на вектор Vnb, это
будет правая часть для задачи с выделением
   /* xi = 0; yi = 0; zi = 0;
    int bel = find_nel(P_X, P_Y, P_Z, mesh, ssxyz, Vnb);
    for (int i = 0; i < 8; i++) { printf("bi = %e\n", pr[ssxyz.nvtr[bel][i] -
1]); }
    printf("b in point: %e\n", solution_xyz_in_point(P_X, P_Y, P_Z, pr, mesh,
ssxyz));*/
    //for (int i = 0; i < pr.size(); i++) { printf("pr%d = %e\n", i, pr[i]); }

    add_G(); // Матрица жесткости
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    edge_cond_1(); // Применение первых нулевых краевых условий
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    solve_cgm(Va, max_iter, eps, relax); // Решение задачи на добавочное поле в
узлах sreda
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);

    q_rz_to_xyz2(Vns, mesh_n, ssxyzn); // Получим решение двумерной задачи для
трехмерной сетки sreda+sreda_n
    q_xyz_to_xyz(Va, mesh, ssxyz, Vas, mesh_n, ssxyzn); // Решение задачи с
аномальным полем расширить до большей сетки
    vec_vec_sum(Vas, Vns, Vs); // Сложим решения q = Va + Vн

    solution_export(mesh_n, solution_filename + "vn", Vns); // Выведем Vn
    solution_export(mesh_n, solution_filename + "va", Vas); // Выведем Va
    solution_export(mesh_n, solution_filename, Vs); // Выведем суммарное решение
для общей сетки
    printf("SOLVE SEGREGATION -          %.2f sec\n", (double)(clock() -
solve_segregation_start) / 1000.);
    printf("END XYZ: SEGREGATION 2--------------------\n");
}
void FEM_N::solve_segregation3(int max_iter, double eps, double relax,
std::string solution_filename) {
    printf("\nSOLVE XYZ: SEGREGATION 3-------------------------\n");
```

```
    unsigned int solve_segregation_start = clock();
    unsigned int start = clock();
    ssxyz.gen_structures(sreda, mesh);
    printf("gen_structures sreda -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    ssxyzn.gen_structures(sreda_n, mesh_n);
    printf("gen_structures sreda_n -     %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    build_matrix_profile();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    add_G(); // Матрица жесткости
    add_b_segregation_3();
    //for (int i = 0; i < pr.size(); i++) { printf("pr%d = %e\n", i, pr[i]); }

    /*xi = 0; yi = 0; zi = 0;
    int NOMEL = find_nel(P_X, P_Y, P_Z, mesh, ssxyz, q);
    add_b_segregation_3(NOMEL);
    for (int i = 0; i < 8; i++) { printf("bi = %e\n", pr[ssxyz.nvtr[NOMEL][i] -
1]); }
    printf("b in point: %e\n", solution_xyz_in_point(P_X, P_Y, P_Z, pr, mesh,
ssxyz));*/
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    edge_cond_1(); // Применение первых нулевых краевых условий
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    solve_cgm(Va, max_iter, eps, relax); // Решение задачи на добавочное поле в
узлах sreda
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);

    q_rz_to_xyz2(Vns, mesh_n, ssxyzn); // Получим решение двумерной задачи для
трехмерной сетки sreda+sreda_n
    q_xyz_to_xyz(Va, mesh, ssxyz, Vas, mesh_n, ssxyzn); // Решение задачи с
аномальным полем расширить до большей сетки
    vec_vec_sum(Vas, Vns, Vs); // Сложим решения q = Va + Vн

    solution_export(mesh_n, solution_filename + "vn", Vns); // Выведем Vn
    //solution_export(mesh_n, solution_filename + "va", Vas); // Выведем Va
    solution_export(mesh, solution_filename + "va", Va); // Выведем Va
    solution_export(mesh_n, solution_filename, Vs); // Выведем суммарное решение
для общей сетки
    printf("SOLVE SEGREGATION -          %.2f sec\n", (double)(clock() -
solve_segregation_start) / 1000.);
    printf("END XYZ: SEGREGATION 3--------------------\n");
}
void FEM_N::solve_segregation4(int max_iter, double eps, double relax,
std::string solution_filename) {
    printf("\nSOLVE XYZ: SEGREGATION 4-------------------------\n");
    unsigned int solve_segregation_start = clock();
    unsigned int start = clock();
```

```
    ssxyz.gen_structures(sreda, mesh);
    printf("gen_structures sreda -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    ssxyzn.gen_structures(sreda_n, mesh_n);
    printf("gen_structures sreda_n -     %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    build_matrix_profile();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    add_G(); // Матрица жесткости
    add_b_segregation_4();
    //for (int i = 0; i < pr.size(); i++) { printf("pr%d = %e\n", i, pr[i]); }
    /*xi = 0; yi = 0; zi = 0;
    int NOMEL = find_nel(P_X, P_Y, P_Z, mesh, ssxyz, q);
    add_b_segregation_4(NOMEL);
    printf("x1 = %f, x2 = %f\n y1 = %f, y2 = %f\n, z1 = %f, z2 = %f\n",
        ssxyz.coord[ssxyz.nvtr[NOMEL][0] - 1][0],
ssxyz.coord[ssxyz.nvtr[NOMEL][1] - 1][0],
        ssxyz.coord[ssxyz.nvtr[NOMEL][0] - 1][1],
ssxyz.coord[ssxyz.nvtr[NOMEL][2] - 1][1],
        ssxyz.coord[ssxyz.nvtr[NOMEL][0] - 1][2],
ssxyz.coord[ssxyz.nvtr[NOMEL][4] - 1][2]);
    xi = 0; yi = 0; zi = 0;
    for (int i = 0; i < 8; i++) { printf("bi = %e\n", pr[ssxyz.nvtr[NOMEL][i] -
1]); }
    printf("b in point: %e\n", solution_xyz_in_point(P_X, P_Y, P_Z, pr, mesh,
ssxyz));*/
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    edge_cond_1(); // Применение первых нулевых краевых условий
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    //CGM solver;
    //solver.solve_msg(ggl,ia,ja,di,pr,Va, max_iter, eps, relax); // Решение
задачи на добавочное поле в узлах sreda
    solve_cgm(Va, max_iter, eps, relax); // Решение задачи на добавочное поле в
узлах sreda
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);

    q_rz_to_xyz2(Vns, mesh_n, ssxyzn); // Получим решение двумерной задачи для
трехмерной сетки sreda+sreda_n
    q_xyz_to_xyz(Va, mesh, ssxyz, Vas, mesh_n, ssxyzn); // Решение задачи с
аномальным полем расширить до большей сетки
    vec_vec_sum(Vas, Vns, Vs); // Сложим решения q = Va + Vн

    solution_export(mesh_n, solution_filename + "vn", Vns); // Выведем Vn
    //solution_export(mesh_n, solution_filename + "va", Vas); // Выведем Va
    solution_export(mesh, solution_filename + "va", Va); // Выведем Va
    solution_export(mesh_n, solution_filename, Vs); // Выведем суммарное решение
для общей сетки
```

```
    printf("SOLVE SEGREGATION -          %.2f sec\n", (double)(clock() -
solve_segregation_start) / 1000.);
    printf("END XYZ: SEGREGATION 4--------------------\n");
}
void FEM_N::solve_segregation5(int max_iter, double eps, double relax,
std::string solution_filename) {
    printf("\nSOLVE XYZ: SEGREGATION 5-------------------------\n");
    unsigned int solve_segregation_start = clock();
    unsigned int start = clock();
    ssxyz.gen_structures(sreda, mesh);
    printf("gen_structures sreda -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    ssxyzn.gen_structures(sreda_n, mesh_n);
    printf("gen_structures sreda_n -     %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    build_matrix_profile();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    add_G(); // Матрица жесткости
    add_b_segregation_5();
    //for (int i = 0; i < pr.size(); i++) { printf("pr%d = %e\n", i, pr[i]); }
    /*xi = 0; yi = 0; zi = 0;
    int NOMEL = find_nel(P_X, P_Y, P_Z, mesh, ssxyz, q);
    add_b_segregation_5(NOMEL);
    xi = 0; yi = 0; zi = 0;
    for (int i = 0; i < 8; i++) { printf("bi = %e\n", pr[ssxyz.nvtr[NOMEL][i] -
1]); }
    printf("b in point: %e\n", solution_xyz_in_point(P_X, P_Y, P_Z, pr, mesh,
ssxyz));*/
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    edge_cond_1(); // Применение первых нулевых краевых условий
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    solve_cgm(Va, max_iter, eps, relax); // Решение задачи на добавочное поле в
узлах sreda
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);

    q_rz_to_xyz2(Vns, mesh_n, ssxyzn); // Получим решение двумерной задачи для
трехмерной сетки sreda+sreda_n
    q_xyz_to_xyz(Va, mesh, ssxyz, Vas, mesh_n, ssxyzn); // Решение задачи с
аномальным полем расширить до большей сетки
    vec_vec_sum(Vas, Vns, Vs); // Сложим решения q = Va + Vн

    solution_export(mesh_n, solution_filename + "vn", Vns); // Выведем Vn
    //solution_export(mesh_n, solution_filename + "va", Vas); // Выведем Va
    solution_export(mesh, solution_filename + "va", Va); // Выведем Va
    solution_export(mesh_n, solution_filename, Vs); // Выведем суммарное решение
для общей сетки
    printf("SOLVE SEGREGATION -          %.2f sec\n", (double)(clock() -
solve_segregation_start) / 1000.);
```

```
    printf("END XYZ: SEGREGATION 5--------------------\n");
}

void FEM_N::solve_rz(int max_iter, double eps, double relax, std::string
solution_filename) {
    printf("SOLVE RZ--------------------------\n");
    ssrz.gen_structures_rz(sreda, mesh_rz);

    unsigned int start = clock();
    build_matrix_profile_rz();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);

    slae_init();

    start = clock();
    add_G_b_rz();
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);

    //print_matrix_plenum();

    start = clock();
    edge_cond_1_rz();
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);

    start = clock();
    solve_cgm(q, max_iter, eps, relax);
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);

    qrz = q;
    solution_rz_export(solution_filename);
    printf("END RZ--------------------\n\n");
}


// Тесты на полином
void FEM_N::test_polinome(int max_iter, double eps, double relax, std::string
solution_filename) {
    printf("\nTEST POLINOME XYZ-------------------------\n");
    ssxyz.gen_structures_p(sreda,mesh);

    unsigned int start = clock();
    build_matrix_profile();
    printf("build matrix profile - %.2f sec\n", (double)(clock() -
start)/1000.);

    slae_init();

    start = clock();
    add_G_b_p();
    printf("global G and b - %.2f sec\n", (double)(clock() - start) / 1000.);

    //print_matrix_plenum();

    start = clock();
    edge_cond_1_p();
    printf("first edge condition apply - %.2f sec\n", (double)(clock()-start) /
1000.);

    start = clock();
    solve_cgm(q, max_iter, eps, relax);
```

```
    printf("SLAE solution - %.2f sec\n", (double)(clock() - start) / 1000.);

    print_solution_p_miss();

    solution_export(mesh,solution_filename, q);
    printf("END TEST POLINOME XYZ--------------------\n");
}
void FEM_N::test_polinome_rz(int max_iter, double eps, double relax, std::string
solution_filename) {
    printf("TEST POLINOME RZ-------------------------\n");
      ssrz.gen_structures_rz_p(sreda,mesh_rz);

    unsigned int start = clock();
    build_matrix_profile_rz();
    printf("build matrix profile -       %.2f sec\n", (double)(clock() - start)
/ 1000.);
    slae_init();
    start = clock();
    add_G_b_rz_p();
    printf("global G and b -             %.2f sec\n", (double)(clock() - start)
/ 1000.);
    print_matrix_plenum();
    start = clock();
    edge_cond_1_rz_p();
    printf("first edge condition apply - %.2f sec\n", (double)(clock() - start)
/ 1000.);
    start = clock();
    solve_cgm(q, max_iter, eps, relax);
    //solve_msg(q, max_iter, eps, relax);
    printf("SLAE solution -              %.2f sec\n", (double)(clock() - start)
/ 1000.);
    print_solution_rz_p_miss();
    qrz = q;
    solution_rz_export(solution_filename);
    printf("END TEST POLINOME RZ-------------------------\n\n");
}

void FEM_N::solve_cgm(vector<double>& q, int max_iter, double eps, double relax)
{
    CGM cgm;
    cgm.init(ia, ja, di, ggl, pr);
    cgm.solve(q, max_iter, eps, relax);
}

void FEM_N::q_rz_to_xyz1(vector<double>& ans, class MESH& mesh, class STRCTRS&
ss) {
    unsigned int start = clock();
    double x, y, z, r, sol;
    ans.resize(0);
    ans.resize(mesh.kuzlov);
    for (int i = 0; i < mesh.kel; i++) {
        x = (ss.coord[ss.nvtr[i][0] - 1][0]+ ss.coord[ss.nvtr[i][1] - 1][0])/2.;
        y = (ss.coord[ss.nvtr[i][0] - 1][1] + ss.coord[ss.nvtr[i][2] - 1][1]) /
2.;
        z = (ss.coord[ss.nvtr[i][0] - 1][2] + ss.coord[ss.nvtr[i][4] - 1][2]) /
2.;
        r = sqrtl(x * x + y * y);
        sol = solution_rz_in_point(r, z);
        for (int j = 0; j < FEM_XYZ_NODES_NUM; j++) {
            ans[ss.nvtr[i][j] - 1] = sol;
        }
    }
    printf("q_rz_to_xyz1   -             %.2f sec\n", (double)(clock() - start)
/ 1000.);
```

```
}

// ans - Вектор в который запишется решение
// mesh - Трехмерная сетка
// ss - Струткуры трехмерной сетки
void FEM_N::q_rz_to_xyz2(vector<double>& ans, class MESH& mesh, class STRCTRS&
ss) {
    unsigned int start = clock();
    double x, y, z, r; // Координаты узла
    ans.resize(0);
    ans.resize(mesh.kuzlov);
    for (int i = 0; i < mesh.kuzlov; i++) {
        x = ss.coord[i][0];
        y = ss.coord[i][1];
        z = ss.coord[i][2];
        r = sqrtl(x * x + y * y);

        ans[i] = solution_rz_in_point(r, z);
    }
    printf("q_rz_to_xyz2  -             %.2f sec\n", (double)(clock() - start)
/ 1000.);
}

void FEM_N::q_xyz_to_xyz(vector<double>& q1, class MESH& mesh1, class STRCTRS&
ss1, vector<double>& q2, class MESH& mesh2, class STRCTRS& ss2) {
    unsigned int start = clock();
    double x, y, z; // Координаты узла
    q2.resize(0);
    q2.resize(mesh2.kuzlov);
    xi = 0; yi = 0; zi = 0;
    for (int i = 0; i < mesh2.kuzlov; i++) {
        x = ss2.coord[i][0];
        y = ss2.coord[i][1];
        z = ss2.coord[i][2];
        //if (i % 10000 == 0)printf("%d %f %f %f ", i, x, y, z);
        q2[i] = solution_xyz_in_point(x,y,z,q1,mesh1,ss1);

    }
    printf("q_xyz_to_xyz  -             %.2f sec\n", (double)(clock() - start)
/ 1000.);
}

double norm(vector<double>& x) {
    double res = 0;
    for (int i = 0; i < x.size(); i++) { res += x[i] * x[i]; }
    return sqrtl(res);
}
void FEM_N::print_solution_p_miss() {
    /*vector<double> diff,solution;
    diff.resize(q.size());
    solution.resize(q.size());
    for (int i = 0; i < q.size(); i++) { solution[i] = u(ssxyz.coord[i][0],
ssxyz.coord[i][1], ssxyz.coord[i][2]); }
    for (int i = 0; i < q.size(); i++) { diff[i] =solution[i]-q[i]; }
    printf("Norm of error vector: %e\n", norm(diff) / norm(solution));*/
    vector<double> diff, solution, solution_a;
    double x,y, z;
    diff.resize(mesh.kel);
    solution.resize(mesh.kel);
    solution_a.resize(mesh.kel);
    for (int i = 0; i < mesh.kel; i++) {
        // Для каждого элемента получаем решение в центре
        for (int j = 0; j < FEM_XYZ_NODES_NUM; j++) {
            solution[i] += q[ssxyz.nvtr[i][j] - 1] / 8.;
```

```
        }
        x = (ssxyz.coord[ssxyz.nvtr[i][0] - 1][0] + ssxyz.coord[ssxyz.nvtr[i][1]
- 1][0]) / 2.;
        y = (ssxyz.coord[ssxyz.nvtr[i][0] - 1][1] + ssxyz.coord[ssxyz.nvtr[i][2]
- 1][1]) / 2.;
        z = (ssxyz.coord[ssxyz.nvtr[i][0] - 1][2] + ssxyz.coord[ssxyz.nvtr[i][4]
- 1][2]) / 2.;
        solution_a[i] = u(x,y, z);
    }
    for (int i = 0; i < solution.size(); i++) { diff[i] = solution[i] -
solution_a[i]; }
    printf("Norm of error vector: %e\n", norm(diff) / norm(solution));
}
void FEM_N::print_solution_rz_p_miss() {
    /*vector<double> diff, solution;
    diff.resize(q.size());
    solution.resize(q.size());
    for (int i = 0; i < q.size(); i++) { solution[i] = u_rz(ssrz.coord[i][0],
ssrz.coord[i][1]); }
    for (int i = 0; i < q.size(); i++) { diff[i] = solution[i] - q[i]; }
    printf("Norm of error vector: %e\n", norm(diff) / norm(solution));*/

    vector<double> diff, solution, solution_a;
    double r, z;
    diff.resize(mesh_rz.kel);
    solution.resize(mesh_rz.kel);
    solution_a.resize(mesh_rz.kel);
    for (int i = 0; i < mesh_rz.kel; i++) {
        // Для каждого элемента получаем решение в центре
        for (int j = 0; j < FEM_RZ_NODES_NUM; j++) {
            solution[i] += q[ssrz.nvtr[i][j] - 1] / 4.;
        }
        r = (ssrz.coord[ssrz.nvtr[i][0] - 1][0] + ssrz.coord[ssrz.nvtr[i][1] -
1][0]) / 2.;
        z = (ssrz.coord[ssrz.nvtr[i][0] - 1][1] + ssrz.coord[ssrz.nvtr[i][2] -
1][1]) / 2.;
        solution_a[i] = u_rz(r, z);
    }
    for (int i = 0; i < solution.size(); i++) { diff[i] = solution[i] -
solution_a[i]; }
    printf("Norm of error vector: %e\n", norm(diff) / norm(solution));
}

double FEM_N::solution_rz_in_point(double r, double z) {
    double res = 0;
    static int ri = 0, zi = 0; // Индексы нижних границ отрезков в одномерных
сетках
    if (mesh_rz.r[0] <= r && r <= mesh_rz.r[mesh_rz.r.size() - 1] && // Если
точка лежит внутри
        mesh_rz.z[0] <= z && z <= mesh_rz.z[mesh_rz.z.size() - 1]) {

        while (!(mesh_rz.r[ri] <= r && r <= mesh_rz.r[ri + 1])) {
            if (r > mesh_rz.r[ri + 1]) { ri++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (r < mesh_rz.r[ri]) { ri--; } // Если точка левее текущего
отрезка, то сместить отрезок влево
        }
        while (!(mesh_rz.z[zi] <= z && z <= mesh_rz.z[zi + 1])) {
            if (z > mesh_rz.z[zi + 1]) { zi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (z < mesh_rz.z[zi]) { zi--; } // Если точка левее текущего
отрезка, то сместить отрезок влево
        }
        int n_el = (mesh_rz.r.size() - 1) * zi + ri;
```

```
        double hr = mesh_rz.r[ri + 1] - mesh_rz.r[ri], hh = mesh_rz.z[zi + 1] -
mesh_rz.z[zi];

        double R1 = (mesh_rz.r[ri + 1] - r) / hr, R2 = (r - mesh_rz.r[ri]) / hr;
        double H1 = (mesh_rz.z[zi + 1] - z) / hh, H2 = (z - mesh_rz.z[zi]) / hh;
        vector<double> psi = { R1 * H1,R2 * H1,R1 * H2,R2 * H2 };
        for (int i = 0; i < FEM_RZ_NODES_NUM; i++) {
            res += psi[i] * qrz[ssrz.nvtr[n_el][i] - 1];
        }
    } else {
        res = NAN;
        printf("Error for point: r = %f, z = %f\n", r, z);
    }
    return res;
}

void FEM_N::grad_rz_in_point(double x, double y, double z, std::vector<double>&
grad) {
    static int ri = 0, zi = 0; // Индексы нижних границ отрезков в одномерных
сетках
    double r = sqrt(x*x+y*y);
    //if (r == 0) { r = 1e-30; }
    if (mesh_rz.r[0] <= r && r <= mesh_rz.r[mesh_rz.r.size() - 1] && // Если
точка лежит внутри
        mesh_rz.z[0] <= z && z <= mesh_rz.z[mesh_rz.z.size() - 1]) {
        while (!(mesh_rz.r[ri] <= r && r <= mesh_rz.r[ri + 1])) {
            if (r > mesh_rz.r[ri + 1]) { ri++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            else if (r < mesh_rz.r[ri]) { ri--; } // Если точка левее текущего
отрезка, то сместить отрезок влево
        }
        while (!(mesh_rz.z[zi] <= z && z <= mesh_rz.z[zi + 1])) {
            if (z > mesh_rz.z[zi + 1]) { zi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            else if (z < mesh_rz.z[zi]) { zi--; } // Если точка левее текущего
отрезка, то сместить отрезок влево
        }
        int n_el = (mesh_rz.r.size() - 1) * zi + ri;
        double hr = mesh_rz.r[ri + 1] - mesh_rz.r[ri], hh = mesh_rz.z[zi + 1] -
mesh_rz.z[zi];
        double R1 = (mesh_rz.r[ri + 1] - r) / hr, R2 = (r - mesh_rz.r[ri]) / hr;
        double H1 = (mesh_rz.z[zi + 1] - z) / hh, H2 = (z - mesh_rz.z[zi]) / hh;
        double dVdri[4] = { -H1 / hr ,H1 / hr,-H2 / hr,H2 / hr };
        double dVdzi[4] = { -R1 / hh ,-R2 / hh,R1 / hh,R2 / hh };
        //vector<double> psi = { R1 * H1,R2 * H1,R1 * H2,R2 * H2 };
        grad[0] = 0;
        grad[1] = 0;
        grad[2] = 0;
        for (int i = 0; i < FEM_RZ_NODES_NUM; i++) {
            grad[0] += dVdri[i] * qrz[ssrz.nvtr[n_el][i] - 1];
            grad[1] += dVdri[i] * qrz[ssrz.nvtr[n_el][i] - 1];
            grad[2] += dVdzi[i] * qrz[ssrz.nvtr[n_el][i] - 1];
        }
        if (r) {
            grad[0] *= x / r;
            grad[1] *= y / r;
        }
    }
    else {
        printf("Error for point: r = %f, z = %f\n", r, z);
    }
}
void FEM_N::grad_rz_in_point2(double x, double y, double z, std::vector<double>&
grad) {
```

```
    static int ri = 0, zi = 0; // Индексы нижних границ отрезков в одномерных
сетках
    double r = sqrt(x * x + y * y);
    //if (r == 0) { r = 1e-30; }
    if (mesh_rz.r[0] <= r && r <= mesh_rz.r[mesh_rz.r.size() - 1] && // Если
точка лежит внутри
        mesh_rz.z[0] <= z && z <= mesh_rz.z[mesh_rz.z.size() - 1]) {
        while (!(mesh_rz.r[ri] <= r && r <= mesh_rz.r[ri + 1])) {
            if (r > mesh_rz.r[ri + 1]) { ri++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            else if (r < mesh_rz.r[ri]) { ri--; } // Если точка левее текущего
отрезка, то сместить отрезок влево
        }
        while (!(mesh_rz.z[zi] <= z && z <= mesh_rz.z[zi + 1])) {
            if (z > mesh_rz.z[zi + 1]) { zi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            else if (z < mesh_rz.z[zi]) { zi--; } // Если точка левее текущего
отрезка, то сместить отрезок влево
        }
        int n_el = (mesh_rz.r.size() - 1) * zi + ri;
        vector<double> V = { qrz[ssrz.nvtr[n_el][0] - 1],qrz[ssrz.nvtr[n_el][1]
- 1],
            qrz[ssrz.nvtr[n_el][2] - 1],qrz[ssrz.nvtr[n_el][3] - 1] };
        double hr = mesh_rz.r[ri + 1] - mesh_rz.r[ri], hh = mesh_rz.z[zi + 1] -
mesh_rz.z[zi];
        double R1 = (mesh_rz.r[ri + 1] - r) / hr, R2 = (r - mesh_rz.r[ri]) / hr;
        double H1 = (mesh_rz.z[zi + 1] - z) / hh, H2 = (z - mesh_rz.z[zi]) / hh;
        grad[0] = (H1 * (V[1] - V[0]) + H2 * (V[3] - V[2])) / hr;
        grad[1] = grad[0];
        grad[2] = (R1*(V[2]-V[0])+R2*(V[3]-V[1]))/hh;
        if (r) {
            grad[0] *= x / r;
            grad[1] *= y / r;
        }
    }
    else {
        printf("Error for point: r = %f, z = %f\n", r, z);
    }
}


int FEM_N::n_el_xyz(double x, double y, double z) {
    int n_el = -1;
    if (mesh.x[0] <= x && x <= mesh.x[mesh.x.size() - 1] && // Если точка лежит
внутри
        mesh.y[0] <= y && y <= mesh.y[mesh.y.size() - 1] &&
        mesh.z[0] <= z && z <= mesh.z[mesh.z.size() - 1]) {
        int xi = 0, yi = 0, zi = 0; // Индексы нижних границ в сетке
        while (!(mesh.x[xi] <= x && x <= mesh.x[xi + 1])) { xi++; }
        while (!(mesh.y[yi] <= y && y <= mesh.y[yi + 1])) { yi++; }
        while (!(mesh.z[zi] <= z && z <= mesh.z[zi + 1])) { zi++; }
        n_el = (mesh.x.size() - 1) * (mesh.y.size() - 1) * zi + (mesh.x.size() -
1) * yi + xi;
    }
    return n_el;
}

double FEM_N::solution_xyz_in_point(double x, double y, double z) {
    double res = 0;
    if (mesh.x[0] <= x && x <= mesh.x[mesh.x.size() - 1] && // Если точка лежит
внутри
        mesh.y[0] <= y && y <= mesh.y[mesh.y.size() - 1] &&
        mesh.z[0] <= z && z <= mesh.z[mesh.z.size() - 1]) {
        int xi = 0, yi = 0, zi = 0; // Индексы нижних границ в сетке
```

```
        while (!(mesh.x[xi] <= x && x <= mesh.x[xi + 1])) { xi++; }
        while (!(mesh.y[yi] <= y && y <= mesh.y[yi + 1])) { yi++; }
        while (!(mesh.z[zi] <= z && z <= mesh.z[zi + 1])) { zi++; }
        int n_el = (mesh.x.size() - 1) * (mesh.y.size() - 1) * zi +
(mesh.x.size()-1) * yi + xi; // Номер элемента в котором содержится точка
        double hx = mesh.x[xi + 1] - mesh.x[xi], hy = mesh.y[yi + 1] -
mesh.y[yi], hz = mesh.z[zi + 1] - mesh.z[zi];
        double X1 = (mesh.x[xi + 1] - x) / hx, X2 = (x - mesh.x[xi]) / hx;
        double Y1 = (mesh.y[yi + 1] - y) / hy, Y2 = (y - mesh.y[yi]) / hy;
        double Z1 = (mesh.z[zi + 1] - z) / hz, Z2 = (z - mesh.z[zi]) / hz;
        vector<double> psi = {
            X1 * Y1 * Z1,X2 * Y1 * Z1,
            X1 * Y2 * Z1,X2 * Y2 * Z1,
            X1 * Y1 * Z2,X2 * Y1 * Z2,
            X1 * Y2 * Z2,X2 * Y2 * Z2 };
        for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
            res += psi[i] * q[ssxyz.nvtr[n_el][i] - 1];
        }
    } else {
        res = NAN;
    }
    return res;
}

double FEM_N::solution_xyz_in_point(double x, double y, double z,
    std::vector<double>& q, class MESH& mesh, class STRCTRS& ss) {
    double res = 0;
    //static int xi = 0, yi = 0, zi = 0; // Индексы нижних границ в сетке
    if (mesh.x[0] <= x && x <= mesh.x[mesh.x.size() - 1] &&
        mesh.y[0] <= y && y <= mesh.y[mesh.y.size() - 1] &&
        mesh.z[0] <= z && z <= mesh.z[mesh.z.size() - 1]) {// Если точка лежит
внутри
        while (!(mesh.x[xi] <= x && x <= mesh.x[xi + 1])) {
            if (x > mesh.x[xi + 1]) { xi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (x < mesh.x[xi]) { xi--; } // Если точка левее текущего отрезка,
то сместить отрезок влево
        }
        while (!(mesh.y[yi] <= y && y <= mesh.y[yi + 1])) {
            if (y > mesh.y[yi + 1]) { yi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (y < mesh.y[yi]) { yi--; } // Если точка левее текущего отрезка,
то сместить отрезок влево
        }
        while (!(mesh.z[zi] <= z && z <= mesh.z[zi + 1])) {
            if (z > mesh.z[zi + 1]) { zi++; } // Если точка правее текущего
отрезка, то сместить отрезок вправо
            if (z < mesh.z[zi]) { zi--; } // Если точка левее текущего отрезка,
то сместить отрезок влево
        }
        int n_el = (mesh.x.size() - 1) * (mesh.y.size() - 1) * zi +
(mesh.x.size() - 1) * yi + xi; // Номер элемента в котором содержится точка
        double hx = mesh.x[xi + 1] - mesh.x[xi], hy = mesh.y[yi + 1] -
mesh.y[yi], hz = mesh.z[zi + 1] - mesh.z[zi];
        double X1 = (mesh.x[xi + 1] - x) / hx, X2 = (x - mesh.x[xi]) / hx;
        double Y1 = (mesh.y[yi + 1] - y) / hy, Y2 = (y - mesh.y[yi]) / hy;
        double Z1 = (mesh.z[zi + 1] - z) / hz, Z2 = (z - mesh.z[zi]) / hz;
        vector<double> psi = {
            X1 * Y1 * Z1,X2 * Y1 * Z1,
            X1 * Y2 * Z1,X2 * Y2 * Z1,
            X1 * Y1 * Z2,X2 * Y1 * Z2,
            X1 * Y2 * Z2,X2 * Y2 * Z2 };
        for (int i = 0; i < FEM_XYZ_NODES_NUM; i++) {
            res += psi[i] * q[ss.nvtr[n_el][i] - 1];
```

```
        }
    }
    else {
        res = NAN;
    }
    return res;
}


void FEM_N::solution_xyz_in_points(std::string filename) {
    ofstream ifs;
    ifs.open(filename, ios::app);
    double x, y, z;
    ifs.setf(ios::scientific | ios::showpos);
    for (int i = 0; i < sreda.points.size(); i++) {
        x = sreda.points[i][0];
        y = sreda.points[i][1];
        z = sreda.points[i][2];
        ifs <<  x << " "<<  y<< " " << z  << " " << solution_xyz_in_point(x, y,
z,Vs,mesh_n,ssxyzn) << endl;
    }
    ifs << endl;
    ifs.close();
}
void FEM_N::solution_rz_in_points(std::string filename) {
    ofstream ifs;
    ifs.open(filename, ios::app);
    double r, z;
    ifs.setf(ios::scientific | ios::showpos);
    for (int i = 0; i < sreda_rz.points.size(); i++) {
        r = sreda_rz.points[i][0];
        z = sreda_rz.points[i][1];
        ifs << r << " " << z  << " " << solution_rz_in_point(r, z) << endl;
    }
    ifs << endl;
    ifs.close();
}

void FEM_N::solution_export(class MESH& mesh, std::string solution_filename,
std::vector<double>& solution) {
    ofstream ifs;
    ifs.open(solution_filename, ios::binary);
    __int32 size;
    // Запишем в файл размеры сеток по x, y и z
    size = mesh.x.size();
    ifs.write((char*)&size, sizeof(size));
    size = mesh.y.size();
    ifs.write((char*)&size, sizeof(size));
    size = mesh.z.size();
    ifs.write((char*)&size, sizeof(size));

    // Запишем координаты узлов
    // Для x
    for (int i = 0; i < mesh.x.size(); i++) {
        ifs.write((char*)&mesh.x[i], sizeof(mesh.x[0]));
    }
    // Для y
    for (int i = 0; i < mesh.y.size(); i++) {
        ifs.write((char*)&mesh.y[i], sizeof(mesh.x[0]));
    }
    // Для z
    for (int i = 0; i < mesh.z.size(); i++) {
        ifs.write((char*)&mesh.z[i], sizeof(mesh.x[0]));
    }
```

```
    //Вывод точек решения q
    for (int i = 0; i < solution.size(); i++) {
        ifs.write((char*)&solution[i], sizeof(solution[0]));
    }

    ifs.close();
}
void FEM_N::solution_rz_export(std::string filename) {
    ofstream ifs;
    ifs.open(filename, ios::binary);
    __int32 size;
    // Запишем в файл размеры сеток по r и z
    size = mesh_rz.r.size();
    ifs.write((char*)&size, sizeof(size));
    size = mesh_rz.z.size();
    ifs.write((char*)&size, sizeof(size));

    // Запишем координаты узлов
    // Для r
    for (int i = 0; i < mesh_rz.r.size(); i++) {
        ifs.write((char*)&mesh_rz.r[i], sizeof(mesh_rz.r[0]));
    }
    // Для z
    for (int i = 0; i < mesh_rz.z.size(); i++) {
        ifs.write((char*)&mesh_rz.z[i], sizeof(mesh_rz.r[0]));
    }

    //Вывод точек решения q; R*Z точек
    for (int i = 0; i < q.size(); i++) {
        ifs.write((char*)&q[i], sizeof(q[0]));
    }

    ifs.close();
}

void FEM_N::solution_rz_to_xyz2(std::string filename)
{
    ssxyzn.gen_structures(sreda_n, mesh_n);
    q_rz_to_xyz2(Vns, mesh_n, ssxyzn); // Получим решение двумерной задачи для
трехмерной сетки sreda+sreda_n
    solution_export(mesh_n,filename,Vns);
}
```

Файл cgm.cpp

```
#include "cgm.h"

// Скалярное умножение векторов
double vec_vec(vector<double>& X, vector<double>& Y)
{
    double result = 0;
    for (int i = 0; i < X.size(); i++)
        result += X[i] * Y[i];
    return result;
}

// Вычисление невязки
double nev(vector<double>& p, vector<double>& pr)
{
    double neva = sqrt(vec_vec(p, p)) / sqrt(vec_vec(pr, pr));
    return neva;
}

// Индексация с 0
```

```
void matrix_vector_i0(vector<double>& ggl, vector<double>& ggu, vector<long>& ig,
vector<long>& jg, vector<double>& di, vector<double>& x, vector<double>& y)
{
    vector<double> Z;
    Z = x;
    int n = x.size();
    for (int i = 0; i < n; i++)
        Z[i] = x[i] * di[i];
    for (int i = 0; i < n; i++)
        for (int j = ig[i]; j < ig[i + 1]; j++)
        {
            Z[i] += x[jg[j]] * ggl[j];
            Z[jg[j]] += x[i] * ggu[j];
        }
    y = Z;
}

// Индексация с 1
void matrix_vector(vector<double>& ggl, vector<double>& ggu, vector<long>& ig, vector<long>&
jg, vector<double>& di, vector<double>& x, vector<double>& y)
{
    vector<double> Z;
    Z = x;
    int n = x.size();
    for (int i = 0; i < n; i++)
        Z[i] = x[i] * di[i];
    for (int i = 0; i < n; i++)
        for (int j = ig[i] - 1; j < ig[i + 1] - 1; j++)
        {
            Z[i] += x[jg[j] - 1] * ggl[j];
            Z[jg[j] - 1] += x[i] * ggu[j];
        }
    y = Z;
}
void matrix_vector(vector<double>& ggl, vector<double>& ggu, vector<int>& ig, vector<int>&
jg, vector<double>& di, vector<double>& x, vector<double>& y)
{
    vector<double> Z;
    Z = x;
    int n = x.size();
    for (int i = 0; i < n; i++)
        Z[i] = x[i] * di[i];
    for (int i = 0; i < n; i++)
        for (int j = ig[i] - 1; j < ig[i + 1] - 1; j++)
        {
            Z[i] += x[jg[j] - 1] * ggl[j];
            Z[jg[j] - 1] += x[i] * ggu[j];
        }
    y = Z;
}


void CGM::init(vector<long>& gi_s, vector< long>& gj_s, vector< double>& di_s, vector<
double>& gg_s, vector< double>& rp_s)
{
    ia = gi_s;
    ja = gj_s;
    // Перейдем с нумерации с 1 на нумерацию с 0
    for (int i = 0; i < ia.size(); i++) {
        ia[i]--;
    }
    for (int i = 0; i < ja.size(); i++){
        ja[i]--;
    }
```

```
    di = di_s;
    gg = gg_s;
    pr = rp_s;
    n = di.size();

    unsigned int m = ia[n];
    r.resize(n);
    x0.resize(n);
    z.resize(n);
    p.resize(n);
    s.resize(n);

    L_di.resize(n);
    L_gg.resize(m);

    for (unsigned int i = 0; i < n; i++)
    {
        L_di[i] = di[i];
        x0[i] = 0; // Начальное приближение
    }

    for (unsigned int i = 0; i < m; i++)
        L_gg[i] = gg[i];
}

void CGM::make_LLT_decomposition()
{

    double sum_d, sum_l;

    for (unsigned int k = 0; k < n; k++)
    {

        sum_d = 0;
        unsigned int i_s = ia[k], i_e = ia[k + 1];

        for (unsigned int i = i_s; i < i_e; i++)
        {

            sum_l = 0;
            unsigned int j_s = ia[ja[i]], j_e = ia[ja[i] + 1];

            for (unsigned int m = i_s; m < i; m++)
            {
                for (unsigned int j = j_s; j < j_e; j++)
                {
                    if (ja[m] == ja[j])
                    {
                        sum_l += L_gg[m] * L_gg[j];
                        j_s++;
                    }
                }
            }
            L_gg[i] = (L_gg[i] - sum_l) / L_di[ja[i]];

            sum_d += L_gg[i] * L_gg[i];
        }
        L_di[k] = sqrt(L_di[k] - sum_d);

    }
}

double CGM::dot_prod(vector<double>& a, vector<double>& b)
{
    double d_p = 0;
```

```
    for (unsigned int i = 0; i < n; i++)
        d_p += a[i] * b[i];
    return d_p;
}

void CGM::mul_matrix(vector<double>& f, vector<double>& x)
{

    for (unsigned int i = 0; i < n; i++)
    {
        double v_el = f[i];
        x[i] = di[i] * v_el;
        for (unsigned int k = ia[i], k1 = ia[i + 1]; k < k1; k++)
        {
            unsigned int j = ja[k];
            x[i] += gg[k] * f[j];
            x[j] += gg[k] * v_el;
        }
    }
}

void CGM::solve_L(vector<double>& f, vector<double>& x)
{

    for (unsigned int k = 1, k1 = 0; k <= n; k++, k1++)
    {
        double sum = 0;

        for (unsigned int i = ia[k1]; i < ia[k]; i++)
            sum += L_gg[i] * x[ja[i]];

        x[k1] = (f[k1] - sum) / L_di[k1];
    }

}

void CGM::solve_LT(vector<double>& f, vector<double>& x)
{

    for (unsigned int k = n, k1 = n - 1; k > 0; k--, k1--)
    {

        x[k1] = f[k1] / L_di[k1];
        double v_el = x[k1];

        for (unsigned int i = ia[k1]; i < ia[k]; i++)
            f[ja[i]] -= L_gg[i] * v_el;
    }
}

void CGM::solve_LLT(vector<double>& f, vector<double>& x)
{
    solve_L(f, x);
    solve_LT(x, x);
}

void CGM::solve(vector<double>& solution, int max_iter, double eps, double relax)
{
            //int a=0;
    mul_matrix(x0, r);
    make_LLT_decomposition();

    for (unsigned int i = 0; i < n; i++)
    {
        r[i] = pr[i] - r[i];
```

```
   /* if (r[i] != 0) {
        a++;
    }*/
}

solve_LLT(r, z);
for (unsigned int i = 0; i < n; i++)
    p[i] = z[i];

double alpha, betta, prod_1, prod_2;
double discr, rp_norm;

rp_norm = sqrt(dot_prod(pr, pr));

prod_1 = dot_prod(p, r);

bool end = false;

int max_iters = 0;
double nevazka_0 = 1, nevazka_1 = 2;
vector<double> buff(di.size());
vector<double> x_past(x0.size());
for (unsigned int iter = 0; iter < max_iter && !end; iter++)
{
    max_iters++;
    discr = sqrt(dot_prod(r, r));

    if (discr != discr || rp_norm != rp_norm)
        cerr << "Error: NaN detected!" << endl;

    // Способ вычисления невязки который был в решателе
    /*nevazka_0 = nevazka_1;
    nevazka_1 = discr / rp_norm;*/


    if (nevazka_1 > eps && nevazka_0 != nevazka_1)
    {

        mul_matrix(z, s);

        alpha = prod_1 / dot_prod(s, z);

        x_past = x0;
        for (unsigned int i = 0; i < n; i++)
        {
            x0[i] += alpha * z[i];
            r[i] -= alpha * s[i];
        }
        ////Релаксация
        for (int i = 0; i < x0.size(); i++) {
            x0[i] = relax * x0[i] + (1. - relax) * x_past[i];
        }

        solve_LLT(r, p);
        prod_2 = dot_prod(p, r);

        betta = prod_2 / prod_1;

        prod_1 = prod_2;

        for (unsigned int i = 0; i < n; i++)
            z[i] = p[i] + betta * z[i];

        // Вычисление невязки
        // Используется файл solver.h
```

```
            matrix_vector_i0(gg, gg, ia, ja, di, x0, buff); //A*x
            for (int i = 0; i < di.size(); i++) { buff[i] -= pr[i]; }
            nevazka_0 = nevazka_1;
            nevazka_1 = nev(buff, pr);
        }
        else
            end = true;


        //printf("CGM %d : nev= %e    \n", iter, nevazka_1);

    }
    printf("CGM : nev= %e ; iterations= %d\n", nevazka_1, max_iters);

    solution.resize(x0.size());
    for (unsigned int i = 0; i < n; i++)
        solution[i] = x0[i];
}

void CGM::solve_msg(vector<double>& ggl, vector<long>& ia, vector<long>& ja, vector<double>&
di, vector<double>& pr,
    vector<double>& q, int max_iter, double eps, double relax) {
    int n = di.size();
    //Начальное приближение
    vector<double> x0(n), x1(n);
    for (int i = 0; i < n; x1[i++] = 1);
    x0 = x1;

    vector<double> buff(di.size());
    matrix_vector(ggl, ggl, ia, ja, di, x1, buff);
    vector<double> r(di.size());
    for (int i = 0; i < r.size(); i++) { r[i] = pr[i] - buff[i]; }
    vector<double> z = r;

    int iter = 0;
    double alpha, betta, nevazka_1 = 1., nevazka_0 = 0;
    while (iter < max_iter && nevazka_1>eps /*&& nevazka_0!=nevazka_1*/) {
        nevazka_0 = nevazka_1;
        matrix_vector(ggl, ggl, ia, ja, di, z, buff); //A*z
        alpha = vec_vec(r, r) / vec_vec(buff, z);
        for (int i = 0; i < n; i++) { x1[i] = x1[i] + alpha * z[i]; }
        betta = 1 / vec_vec(r, r);
        for (int i = 0; i < n; i++) { r[i] = r[i] - alpha * buff[i]; }
        betta *= vec_vec(r, r);
        for (int i = 0; i < n; i++) { z[i] = r[i] + betta * z[i]; }

        //релаксация
        /*for (int i = 0; i < n; i++) { x1[i] = w_relax * x1[i] + (1. - w_relax) * x0[i]; }
        x0 = x1;*/

        matrix_vector(ggl, ggl, ia, ja, di, x1, buff); //A*x
        for (int i = 0; i < n; i++) { buff[i] -= pr[i]; }

        nevazka_1 = nev(buff, pr);
        iter++;
        //printf("nev = %e, iterations = %d\n", nevazka_1, iter);
    }
    q = x1;
    printf("MSG : nev = %e, iterations = %d\n", nevazka_1, iter);
}
```

Файл mesh.cpp

```
#include "fem.h"
```

```
#define MESH_R_K 1.1
#define MESH_R_H0 1e-6
#define MESH_H_K 1.1
#define MESH_H_H0 1e-6

void gen_1d_mesh(vector<double>& sreda_x, vector<double>& hx, vector<double>&
kx, vector<int>& left_right, vector<double>& x) {
	list<double> buff;
	list<double> buff_inv;

	double next;
	double k, h0;
	for (int i = 0; i < sreda_x.size() - 1; i++) {
		buff.resize(0);
		//Если требуется разрядка от правой границы
		if (left_right[i]) {
			//Значение первого шага
			h0 = hx[i];
			//Вычислим положение следущего узла
			next = sreda_x[i + 1] - h0;
			//Коэффициент разрядки
			k = kx[i];
			//Пока мы не перешли за левую границу будем добавлять узлы
			while (next > sreda_x[i]) {
				//Добавляем узел
				buff.push_back(next);
				//Вычисляем следующий
				h0 *= k;
				next -= h0;
			}
			//В конец кладем левую границу
			buff.push_back(sreda_x[i]);

			//Обходим buff в обратном порядке и записываем значения в x
			x.insert(x.end(), buff.rbegin(), buff.rend());
		}
		//Если разрядка от левой границы
		else {
			//В начало кладем левую границу
			buff.push_back(sreda_x[i]);
			//Значение первого шага
			h0 = hx[i];
			//Вычислим положение следущего узла
			next = sreda_x[i] + h0;
			//Коэффициент разрядки
			k = kx[i];
			//Пока мы не перешли за правую границу, будем добавлять узлы
			while (next < sreda_x[i + 1]) {
				//Добавляем узел
				buff.push_back(next);
				//Вычисляем следующий
				h0 *= k;
				next += h0;
			}

			//Обходим buff и записываем значения в x
			x.insert(x.end(), buff.begin(), buff.end());
		}
	}
	x.push_back(sreda_x[sreda_x.size() - 1]);

}

// Генерация одномерной сетки r для двумерной задачи
```

```
void gen_1d_mesh_r(class SREDA& sreda, vector<double>& r) {
	list<double> buff;
	list<double> buff_inv;

	double next;
	double k = MESH_R_K, h0 = MESH_R_H0,hx,hy;
	double r0, r1;
	// Для выбора радуиса на котором будет решаться двумерная задача
выберается максимально возможное расстояние до границы от
	// источника: В случае квадрата его диагональ равна a*sqrt(2)
	hx = sreda.x[sreda.x.size() - 1] - sreda.x[0];
	hy = sreda.y[sreda.y.size() - 1] - sreda.y[0];

	r1 = max(hx, hy) * sqrt(2);
	buff.resize(0);
	next = h0;

	r.push_back(0); //Сперва занесем 0
	//Пока мы не перешли за правую границу, будем добавлять узлы
	while (next < r1) {
		//Добавляем узел
		buff.push_back(next);
		//Вычисляем следующий
		h0 *= k;
		next += h0;
	}

	//Обходим buff и записываем значения в x
	r.insert(r.end(), buff.begin(), buff.end());

	// Заносим последний узел
	r.push_back(r1);
}

// Генерация одномерной сетки h для двумерной задачи
void gen_1d_mesh_h(class SREDA& sreda, vector<double>& h) {
	list<double> buff;
	list<double> buff_inv;

	double next;
	double k = MESH_H_K, h0 = MESH_H_H0, hx, hy;
	double h_end;

	h_end = sreda.z[sreda.z.size()-1];
	buff.resize(0);
	next = h0;

	h.push_back(0); //Сперва занесем 0
	//Пока мы не перешли за правую границу, будем добавлять узлы
	while (next < h_end) {
		//Добавляем узел
		buff.push_back(next);
		//Вычисляем следующий
		h0 *= k;
		next += h0;
	}

	//Обходим buff и записываем значения в x
	h.insert(h.end(), buff.begin(), buff.end());

	// Заносим последний узел
	h.push_back(h_end);
}
```

```
//Подробит сетку
void splitting(vector<double>& x, int k) {
	if (k) {
		list<double> buff;
		double h;
		for (int i = 0; i < x.size() - 1; i++) {
			buff.push_back(x[i]);

			h = (x[i + 1] - x[i]) / pow(2, k);
			for (int j = 1; j < pow(2, k); j++) {
				buff.push_back(x[i] + (j * h));
			}
		}
		buff.push_back(x[x.size() - 1]);

		x.resize(0);
		x.insert(x.begin(), buff.begin(), buff.end());
	}
}

// Функция для объединения двух отсортированных векторов X[] и Y[]
std::vector<double> vector_merge(std::vector<double> const& X, std::vector <
double > const& Y)
{
	int k = 0, i = 0, j = 0;
	std::vector<double> aux(X.size() + Y.size());

	// пока есть элементы в первом и втором массивах
	while (i < X.size() && j < Y.size())
	{
		if (X[i] <= Y[j]) {
			aux[k++] = X[i++];
		}
		else {
			aux[k++] = Y[j++];
		}
	}

	// копируем оставшиеся элементы
	while (i < X.size()) {
		aux[k++] = X[i++];
	}

	while (j < Y.size()) {
		aux[k++] = Y[j++];
	}

	aux.resize(std::distance(aux.begin(), std::unique(aux.begin(),
aux.end())));

	return aux;
}

// Добавит узлы аномальных элементов в сетку по X
void add_elements_x_edges(class SREDA& sreda, std::vector<double>& x) {
	std::set<double> buff;
	for (int i = 0; i < sreda.elms.size(); i++) {
		if (sreda.elms[i][ANOMAL]) { // Если указано что элемент относится к
аномальному полю то включаем его границы
			if (sreda.x[0] <= sreda.elms[i][X0_COORD] &&
sreda.elms[i][X0_COORD] <= sreda.x[sreda.x.size() - 1]) {
				buff.insert(sreda.elms[i][X0_COORD]);
			}
```

```
                  if (sreda.x[0] <= sreda.elms[i][X1_COORD] &&
sreda.elms[i][X1_COORD] <= sreda.x[sreda.x.size() - 1]) {
                        buff.insert(sreda.elms[i][X1_COORD]);
                  }
            }
      }
      std::vector<double> nodes;
      nodes.insert(nodes.end(), buff.begin(), buff.end());

      // Добавим в сетку новые узлы и удалим повторяющиеся
      x = vector_merge(x, nodes);
      x.resize(std::distance(x.begin(), std::unique(x.begin(), x.end())));
}

// Добавит узлы аномальных элементов в сетку по y
void add_elements_y_edges(class SREDA& sreda, std::vector<double>& y) {
      std::set<double> buff;
      for (int i = 0; i < sreda.elms.size(); i++) {
            if (sreda.elms[i][ANOMAL]) { // Если указано что элемент относится к
аномальному полю то включаем его границы
                  if (sreda.y[0] <= sreda.elms[i][Y0_COORD] &&
sreda.elms[i][Y0_COORD] <= sreda.y[sreda.y.size() - 1]) {
                        buff.insert(sreda.elms[i][Y0_COORD]);
                  }
                  if (sreda.y[0] <= sreda.elms[i][Y1_COORD] &&
sreda.elms[i][Y1_COORD] <= sreda.y[sreda.y.size() - 1]) {
                        buff.insert(sreda.elms[i][Y1_COORD]);
                  }
            }
      }
      std::vector<double> nodes;
      nodes.insert(nodes.end(), buff.begin(), buff.end());

      // Добавим в сетку новые узлы и удалим повторяющиеся
      y = vector_merge(y, nodes);
      y.resize(std::distance(y.begin(), std::unique(y.begin(), y.end())));
}

// Добавит узлы аномальных элементов в сетку по z
void add_elements_z_edges(class SREDA& sreda, std::vector<double>& z) {
      std::set<double> buff;
      for (int i = 0; i < sreda.elms.size(); i++) {
            if (sreda.elms[i][ANOMAL]) { // Если указано что элемент относится к
аномальному полю то включаем его границы
                  if (sreda.z[0] <= sreda.elms[i][Z0_COORD] &&
sreda.elms[i][Z0_COORD] <= sreda.z[sreda.z.size() - 1]) {
                        buff.insert(sreda.elms[i][Z0_COORD]);
                  }
                  if (sreda.z[0] <= sreda.elms[i][Z1_COORD] &&
sreda.elms[i][Z1_COORD] <= sreda.z[sreda.z.size() - 1]) {
                        buff.insert(sreda.elms[i][Z1_COORD]);
                  }
            }
      }
      std::vector<double> nodes;
      nodes.insert(nodes.end(), buff.begin(), buff.end());

      // Добавим в сетку новые узлы и удалим повторяющиеся
      z = vector_merge(z, nodes);
}


void  MESH::gen_mesh(class SREDA& sreda) {
      //Генерация сетки по x
```

```
	gen_1d_mesh(sreda.x, sreda.hx, sreda.kx, sreda.left_right[0], x);
	splitting(x, sreda.splitting[0]);
	add_elements_x_edges(sreda, x);

	//Генерация сетки по y
	gen_1d_mesh(sreda.y, sreda.hy, sreda.ky, sreda.left_right[1], y);
	splitting(y, sreda.splitting[1]);
	add_elements_y_edges(sreda, y);

	//Генерация сетки по z
	gen_1d_mesh(sreda.z, sreda.hz, sreda.kz, sreda.left_right[2], z);
	splitting(z, sreda.splitting[2]);
	add_elements_z_edges(sreda, z);

	kuzlov = x.size() * y.size() * z.size();
	kel = (x.size() - 1) * (y.size() - 1) * (z.size() - 1);

	printf("3D: kuzlov - %d ; kel - %d; x - %d ; y - %d, z - %d\n",
		kuzlov, kel,x.size(),y.size(),z.size());
}

// Генерация двумерной сетки
void  MESH::gen_mesh_rh(class SREDA& sreda) {
	//Генерация сетки по r
	gen_1d_mesh_r(sreda, r);

	//Генерация сетки по h
	gen_1d_mesh_h(sreda, h);

	kuzlov = r.size() * h.size();
	kel = (r.size() - 1) * (h.size() - 1);

	printf("2D: kuzlov - %d ; kel - %d; r - %d, z - %d\n", kuzlov, kel,
r.size(), h.size());
}

// Генерирует номера узлов элементов для трехмерной сетки
void gen_nvtr(class MESH* mesh,
	vector<vector<int>>& nvtr)
{
	double kel = mesh->kel;
	nvtr.resize(mesh->kel);
	for (int i = 0; i < kel; i++) { nvtr[i].resize(8); }

	int x, y, z;
	// Количество элементов в слое x
	int kx = (mesh->x.size() - 1);
	// Количество элементов в слое xy
	int kxy = (mesh->x.size() - 1) * (mesh->y.size() - 1);
	for (int i = 0; i < kel; i++) {
		x = i % kx + 1;
		y = (i % kxy) / (kx)+1;
		z = i / kxy + 1;

		nvtr[i][0] = x + (y - 1) * (kx + 1) + (z - 1) * mesh->x.size() *
mesh->y.size();
		nvtr[i][1] = nvtr[i][0] + 1;
		nvtr[i][2] = nvtr[i][0] + mesh->x.size();
		nvtr[i][3] = nvtr[i][2] + 1;
		nvtr[i][4] = nvtr[i][0] + mesh->x.size() * mesh->y.size();
		nvtr[i][5] = nvtr[i][4] + 1;
		nvtr[i][6] = nvtr[i][4] + mesh->x.size();
		nvtr[i][7] = nvtr[i][6] + 1;
	}
```

```
}

// Генерирует номера узлов элементов для двумерной сетки
void gen_nvtr_rh(class MESH* mesh,
	vector<vector<int>>& nvtr)
{
	double kel = mesh->kel;
	nvtr.resize(mesh->kel);
	for (int i = 0; i < kel; i++) { nvtr[i].resize(4); }

	int n;
	for (int i = 0; i < kel; i++) {
		n = i / (mesh->r.size()-1)*(mesh->r.size()) + i%(mesh->r.size()-1) +
1;

		nvtr[i][0] = n;
		nvtr[i][1] = n + 1;
		nvtr[i][2] = n + mesh->r.size();
		nvtr[i][3] = n + mesh->r.size() + 1;

	}
}


void gen_coord(class MESH* mesh,
	vector<vector<double>>& rz) {
	rz.resize(mesh->kuzlov);
	int xs = mesh->x.size();
	int ys = mesh->y.size();
	int zs = mesh->z.size();
	int x, y, z;
	for (int i = 0; i < mesh->kuzlov; i++) {
		x = i % xs;
		y = (i % (xs * ys)) / xs;
		z = i / (xs * ys);
		rz[i].resize(3);
		rz[i] = { mesh->x[x], mesh->y[y], mesh->z[z] };
	}

}

void gen_coord_rh(class MESH* mesh,
	vector<vector<double>>& coord) {
	coord.resize(mesh->kuzlov);
	int rs = mesh->r.size();
	int hs = mesh->h.size();
	int r, h;
	for (int i = 0; i < mesh->kuzlov; i++) {
		r = i % rs;
		h = i / rs;
		coord[i].resize(2);
		coord[i] = { mesh->r[r], mesh->h[h]};
	}

}

// По кооринатам определит номера материала
// Обход обратный: будет получен материал аномального поля, если он не
существует, то нормального
int get_mat_from_sreda_reverse(class SREDA& sreda, double x, double y, double z)
{
	// Будем обходить подобласти в обратном порядке (учитывая что в файле
среда подоблости заданны вложением по порядку)
	for (int i = sreda.elms.size() - 1; i >= 0; i--) {
```

```
            /*printf("%lf %lf %lf\n", sreda.elms[i][X0_COORD], x,
sreda.elms[i][X1_COORD]);
            printf("%lf %lf %lf\n", sreda.elms[i][Y0_COORD], y,
sreda.elms[i][Y1_COORD]);
            printf("%lf %lf %lf\n", sreda.elms[i][Z0_COORD], z,
sreda.elms[i][Z1_COORD]);*/
            if (sreda.elms[i][X0_COORD] < x && x < sreda.elms[i][X1_COORD] &&
                  sreda.elms[i][Y0_COORD] < y && y < sreda.elms[i][Y1_COORD] &&
                  sreda.elms[i][Z0_COORD] < z && z < sreda.elms[i][Z1_COORD]) {
                  return sreda.elms[i][MAT_N];
            }
      }
      return 0;
}

// По кооринатам определит номера материала
// Обход прямой: в первую очередь будет получен материал нормального поля
int get_mat_from_sreda_direct(class SREDA& sreda, double x, double y, double z)
{
      // Будем обходить подобласти в обратном порядке (учитывая что в файле
среда подоблости заданны вложением по порядку)
      for (int i = sreda.elms.size() - 1; i >= 0; i--) {
            /*printf("%lf %lf %lf\n", sreda.elms[i][X0_COORD], x,
sreda.elms[i][X1_COORD]);
            printf("%lf %lf %lf\n", sreda.elms[i][Y0_COORD], y,
sreda.elms[i][Y1_COORD]);
            printf("%lf %lf %lf\n", sreda.elms[i][Z0_COORD], z,
sreda.elms[i][Z1_COORD]);*/
            if (sreda.elms[i][X0_COORD] < x && x < sreda.elms[i][X1_COORD] &&
                  sreda.elms[i][Y0_COORD] < y && y < sreda.elms[i][Y1_COORD] &&
                  sreda.elms[i][Z0_COORD] < z && z < sreda.elms[i][Z1_COORD]) {
                  return sreda.elms[i][MAT_N];
            }
      }
      return 0;
}

// По кооринатам определит номера материала для двумерной задачи
int get_mat_from_sreda_rz(class SREDA& sreda, double r, double h) {
      // Будем обходить подобласти в прямом порядке (учитывая что в файле среда
подоблости заданны вложением по порядку)
      // Считается, что для двумерной задачи элементы, которые не располагаются
от начала до конца границы являются аномальными
      for (int i = sreda.elms.size() - 1; i >= 0; i--) {
            if (sreda.elms[i][Z0_COORD] < h && h < sreda.elms[i][Z1_COORD]) {
                  return sreda.elms[i][MAT_N];
            }
      }
      return 0;
}

void gen_nvkat2d(class SREDA& sreda, class MESH* mesh, vector<int>& nvkat2d,
vector<vector<int>>& nvtr,
      vector<vector<double>>& rz) {
      nvkat2d.resize(mesh->kel);
      double x, y, z;
      for (int i = 0; i < mesh->kel; i++) {
            // Координаты центров элементов
            x = (rz[nvtr[i][0] - 1][0] + rz[nvtr[i][1] - 1][0]) / 2.;
            y = (rz[nvtr[i][0] - 1][1] + rz[nvtr[i][2] - 1][1]) / 2.;
            z = (rz[nvtr[i][0] - 1][2] + rz[nvtr[i][4] - 1][2]) / 2.;
            nvkat2d[i] = get_mat_from_sreda_reverse(sreda, x, y, z);
      }
}
```

```
void gen_nvkat2d_rh(class SREDA& sreda, class MESH* mesh, vector<int>& nvkat2d,
vector<vector<int>>& nvtr,
      vector<vector<double>>& rz) {
      nvkat2d.resize(mesh->kel);
      double r, z;
      for (int i = 0; i < mesh->kel; i++) {
            // Координаты центров элементов
            r = (rz[nvtr[i][0] - 1][0] + rz[nvtr[i][1] - 1][0]) / 2.;
            z = (rz[nvtr[i][0] - 1][1] + rz[nvtr[i][2] - 1][1]) / 2.;
            nvkat2d[i] = get_mat_from_sreda_rz(sreda, r, z);
      }
}

void gen_l1(class MESH* mesh, vector<double>& edge_conditions, vector<int>& l1,
vector<vector<double>>& rz) {
      l1.resize(0);
      set<int> buff;
      for (int i = 0; i < mesh->kuzlov; i++) {
            // Если указано, что на грани x = x0 заданы первые краевые,
            // то если координата текущей точки совпадает с x0, тогда добавить
ее номер.
            // С остальными гранями аналогично
            if (edge_conditions[L1_X0]) { if (rz[i][0] == mesh->x[0]) {
buff.insert(i + 1); } }
            if (edge_conditions[L1_X1]) { if (rz[i][0] == mesh->x[mesh->x.size()
- 1]) { buff.insert(i + 1); } }

            if (edge_conditions[L1_Y0]) { if (rz[i][1] == mesh->y[0]) {
buff.insert(i + 1); } }
            if (edge_conditions[L1_Y1]) { if (rz[i][1] == mesh->y[mesh->y.size()
- 1]) { buff.insert(i + 1); } }

            if (edge_conditions[L1_Z0]) { if (rz[i][2] == mesh->z[0]) {
buff.insert(i + 1); } }
            if (edge_conditions[L1_Z1]) { if (rz[i][2] == mesh->z[mesh->z.size()
- 1]) { buff.insert(i + 1); } }
      }
      l1.insert(l1.begin(), buff.begin(), buff.end());
}

void gen_l1_rh(class MESH* mesh, vector<double>& edge_conditions, vector<int>&
l1, vector<vector<double>>& coord) {
      l1.resize(0);
      set<int> buff;
      /* Краевые условия в двумерной задаче заданы следующим образом

         S2=0
       ----------
       |        |
s2=0 |        | S1 = 0
       |        |
       ----------
          S1 = 0
       */
      for (int i = 0; i < mesh->kuzlov; i++) {
            if (coord[i][0] == mesh->r[mesh->r.size()-1]) { buff.insert(i + 1);
}
            else if (coord[i][1] == mesh->h[mesh->h.size()-1]) { buff.insert(i +
1); }
      }
      l1.insert(l1.begin(), buff.begin(), buff.end());
}
```

```
// nvtr : 4 * int номера вершин прямоугольников
// nvkat2d : 1 * int номера матреиала прямоугольников
// coord : 3 * double (r, z) координаты вершин
// l1 : 1 * int (номера вершин с первым нулевым краевым условием)
// sigma : 1 * double значение параметра сигма по номеру материала
void MESH::gen_structures(class FEM& fem, class SREDA& sreda)
{
	gen_nvtr(this, fem.nvtr);
	gen_coord(this, fem.coord);
	gen_nvkat2d(sreda, this, fem.nvkat2d, fem.nvtr, fem.coord);
	gen_l1(this, sreda.edge_conditions, fem.l1, fem.coord);
}

// nvtr : 8 * int номера вершин прямоугольников
// nvkat2d : int nvkat2d номера матреиала прямоугольников
// rz : 3 * double (x, y, z) координаты вершин
// l1 : 1 * int (номера вершин с первым нулевым краевым условием)
// sigma : 1 * double значение параметра сигма по номеру материала
void MESH::gen_structures_rh(class FEM& fem, class SREDA& sreda)
{
	gen_nvtr_rh(this, fem.nvtr);
	gen_coord_rh(this, fem.coord);
	gen_nvkat2d_rh(sreda, this, fem.nvkat2d, fem.nvtr, fem.coord);
	fem.nvtr_rh = fem.nvtr; // Сохраним структуру для дальнейшего пользования
	gen_l1_rh(this, sreda.edge_conditions, fem.l1, fem.coord);
}

// Генерация двумерной сетки
void  MESH_RZ::gen_mesh(class SREDA_RZ & sreda_rz, class SREDA & sreda) {
	////Генерация сетки по r
	//gen_1d_mesh_r(sreda, r);

	////Генерация сетки по h
	//gen_1d_mesh_h(sreda, z);

	//kuzlov = r.size() * z.size();
	//kel = (r.size() - 1) * (z.size() - 1);

	vector<double> hx = { sreda_rz.r[H0] };
	vector<double> kx = { sreda_rz.r[K] };
	vector<int> left_right = { 0 };
	vector < double> x = { 0,sreda_rz.r[END] };

		//Генерация сетки по r
	gen_1d_mesh(x, hx, kx, left_right, r);
	splitting(r, sreda_rz.splitting[0]);


	//Генерация сетки по h
	hx[0] = sreda_rz.z[H0];
	kx[0] = sreda_rz.z[K];
	x[1] = sreda_rz.z[END];
	gen_1d_mesh(x, hx, kx, left_right, z);
	splitting(z, sreda_rz.splitting[1]);
	add_elements_z_edges(sreda, z);


	kuzlov = r.size() * z.size();
	kel = (r.size() - 1) * (z.size() - 1);

	printf("2D: kuzlov - %d ; kel - %d; r - %d, z - %d\n", kuzlov, kel,
r.size(), z.size());
}
```

```
void MESH::add_mesh(class MESH& mesh) {
	x = vector_merge(x, mesh.x);
	y = vector_merge(y, mesh.y);
	z = vector_merge(z, mesh.z);

	kuzlov = x.size() * y.size() * z.size();
	kel = (x.size() - 1) * (y.size() - 1) * (z.size() - 1);

	printf("3D SUM NORMAL: kuzlov - %d ; kel - %d; x - %d ; y - %d, z - %d\n",
		kuzlov, kel, x.size(), y.size(), z.size());

}
```

Файл sreda.cpp

```
#include "fem.h"

//Считает вектор из потока
void read_vector(ifstream* ifs, vector<double>& x, int n) {
	x.resize(n);
	for (int i = 0; i < n; i++) { *ifs >> x[i]; }
}

void read_vector(ifstream* ifs, vector<int>& x, int n) {
	x.resize(n);
	for (int i = 0; i < n; i++) { *ifs >> x[i]; }
}

void SREDA::read_sreda(const char* filename) {
	cout << "Reading file sreda\n";

	ifstream ifs;
	ifs.open(filename, ios::in);

	int num_areas;
	ifs >> num_areas;
	elms.resize(num_areas);
	for (int i = 0; i < num_areas; i++) { read_vector(&ifs, elms[i], 8); }

	int x_num, y_num, z_num;
	left_right.resize(3);
	ifs >> x_num;
	read_vector(&ifs, x, x_num);
	read_vector(&ifs, hx, x_num - 1);
	read_vector(&ifs, kx, x_num - 1);
	read_vector(&ifs, left_right[0], x_num - 1);

	ifs >> y_num;
	read_vector(&ifs, y, y_num);
	read_vector(&ifs, hy, y_num - 1);
	read_vector(&ifs, ky, y_num - 1);
	read_vector(&ifs, left_right[1], y_num - 1);

	ifs >> z_num;
	read_vector(&ifs, z, z_num);
	read_vector(&ifs, hz, z_num - 1);
	read_vector(&ifs, kz, z_num - 1);
	read_vector(&ifs, left_right[2], z_num - 1);

	vector<int> split(3);
	read_vector(&ifs, split, 3);
	splitting = split;
}
void SREDA::read_sreda(string filename) {
```

```
	ifstream ifs;
	ifs.open(filename, ios::in);
	if (ifs.is_open()) {
		cout << "Reading file sreda: " << filename << endl;
		int num_areas;
		ifs >> num_areas;
		elms.resize(num_areas);
		for (int i = 0; i < num_areas; i++) { read_vector(&ifs, elms[i], 8); }

		int x_num, y_num, z_num;
		left_right.resize(3);
		ifs >> x_num;
		read_vector(&ifs, x, x_num);
		read_vector(&ifs, hx, x_num - 1);
		read_vector(&ifs, kx, x_num - 1);
		read_vector(&ifs, left_right[0], x_num - 1);

		ifs >> y_num;
		read_vector(&ifs, y, y_num);
		read_vector(&ifs, hy, y_num - 1);
		read_vector(&ifs, ky, y_num - 1);
		read_vector(&ifs, left_right[1], y_num - 1);

		ifs >> z_num;
		read_vector(&ifs, z, z_num);
		read_vector(&ifs, hz, z_num - 1);
		read_vector(&ifs, kz, z_num - 1);
		read_vector(&ifs, left_right[2], z_num - 1);

		vector<int> split(3);
		read_vector(&ifs, split, 3);
		splitting = split;
	}
	else {
		cout << "Error opening file sreda: " << filename << endl;
	}

}

void SREDA::read_edge_conditions(const char* filename) {
	ifstream ifs;
	ifs.open(filename, ios::in);

	edge_conditions.resize(6);
	for (int i = 0; i < 6; i++) { ifs >> edge_conditions[i]; }
}
void SREDA::read_edge_conditions(string filename) {
	ifstream ifs;
	ifs.open(filename, ios::in);

	edge_conditions.resize(6);
	for (int i = 0; i < 6; i++) { ifs >> edge_conditions[i]; }
}

void SREDA::read_current_sources(const char* filename) {
	ifstream ifs;
	ifs.exceptions(std::ios::failbit);   // throw if failbit get set
	try {
		ifs.open(filename, ios::in);
	}
	catch (const std::exception& ex) {
		std::cerr << "Could not open current sourcess: "
			<< ex.what();
	}
```

```
	int n;
	ifs >> n;

	current_sources.resize(n);
	for (int i = 0; i < n; i++) {
		current_sources[i].resize(5);
		ifs >> current_sources[i][SOURCE_X] >> current_sources[i][SOURCE_Y] >>
current_sources[i][SOURCE_Z] >> current_sources[i][SOURCE_POW];
	}
}
void SREDA::read_current_sources(string filename) {
	ifstream ifs;
	ifs.exceptions(std::ios::failbit);   // throw if failbit get set
	try {
		ifs.open(filename, ios::in);
	}
	catch (const std::exception& ex) {
		std::cerr << "Could not open current sourcess: "
			<< ex.what();
	}


	int n;
	ifs >> n;

	current_sources.resize(n);
	for (int i = 0; i < n; i++) {
		current_sources[i].resize(5);
		ifs >> current_sources[i][SOURCE_X] >> current_sources[i][SOURCE_Y] >>
current_sources[i][SOURCE_Z] >> current_sources[i][SOURCE_POW];
	}
}

void SREDA::read_materials(const char* filename, class FEM& fem) {
	ifstream ifs;
	ifs.exceptions(std::ios::failbit);   // throw if failbit get set
	try {
		ifs.open(filename, ios::in);
	}
	catch (const std::exception& ex) {
		std::cerr << "Could not open materials: "
			<< ex.what();
	}
	int n;
	ifs >> n;

	fem.sigma.resize(n);
	for (int i = 0; i < n; i++) {
		fem.sigma[i].resize(2);
		ifs >> fem.sigma[i][0];
		ifs >> fem.sigma[i][1];
	}
}
void SREDA::read_materials(string filename) {
	ifstream ifs;
	ifs.exceptions(std::ios::failbit);   // throw if failbit get set
	try {
		ifs.open(filename, ios::in);
	}
	catch (const std::exception& ex) {
		std::cerr << "Could not open materials: "
			<< ex.what();
	}
```

```
	int n;
	ifs >> n;

	sigma.resize(n);
	for (int i = 0; i < n; i++) {
		sigma[i].resize(2);
		ifs >> sigma[i][0];
		ifs >> sigma[i][1];
	}
}

void SREDA::read_points(string filename) {
	ifstream ifs;
	ifs.open(filename, ios::in);
	std::vector<double> buf;
	while (!ifs.eof()) {
		read_vector(&ifs, buf, 3);
		points.push_back(buf);
	}
}


void SREDA::read_problem(string meshname) {
	read_sreda(meshname + SREDA_FILENAME);
	read_edge_conditions(meshname + EDGE_CONDITIONS_FILENAME);
	read_current_sources(meshname + CURRENT_SOURCES_FILENAME);
	read_materials(meshname + MATERIALS_FILENAME);
	read_points(meshname + POINTS_FILENAME);

}

void SREDA::read_problem_n(string meshname) {
	read_sreda(meshname + SREDA_N_FILENAME);
	read_edge_conditions(meshname + EDGE_CONDITIONS_FILENAME);
	read_current_sources(meshname + CURRENT_SOURCES_FILENAME);
	read_materials(meshname + MATERIALS_FILENAME);
	read_points(meshname + POINTS_FILENAME);
}

void SREDA_RZ::read_points(string filename) {
	ifstream ifs;
	ifs.open(filename, ios::in);
	std::vector<double> buf;
	while (!ifs.eof()) {
		read_vector(&ifs, buf, 2);
		points.push_back(buf);
	}
}

// Считает файл с описанием среды для двумерной задачи
void SREDA_RZ::read_sreda(string filename) {
	cout << "Reading file sreda_rz\n";

	ifstream ifs;
	ifs.open(filename + "/sreda_rz", ios::in);

	read_vector(&ifs, r, 4);
	read_vector(&ifs, z, 4);
	read_vector(&ifs, splitting, 2);
	read_points(filename + POINTS_RZ_FILENAME);
}
```

Файл strctrs.cpp

```
#include "STRCTRS.h"
```

```
// Генерирует номера узлов элементов для трехмерной сетки
void STRCTRS::gen_nvtr(class MESH& mesh)
{
	double kel = mesh.kel;
	nvtr.resize(mesh.kel);
	for (int i = 0; i < kel; i++) { nvtr[i].resize(8); }

	int x, y, z;
	// Количество элементов в слое x
	int kx = (mesh.x.size() - 1);
	// Количество элементов в слое xy
	int kxy = (mesh.x.size() - 1) * (mesh.y.size() - 1);
	for (int i = 0; i < kel; i++) {
		x = i % kx + 1;
		y = (i % kxy) / (kx)+1;
		z = i / kxy + 1;

		nvtr[i][0] = x + (y - 1) * (kx + 1) + (z - 1) * mesh.x.size() * mesh.y.size();
		nvtr[i][1] = nvtr[i][0] + 1;
		nvtr[i][2] = nvtr[i][0] + mesh.x.size();
		nvtr[i][3] = nvtr[i][2] + 1;
		nvtr[i][4] = nvtr[i][0] + mesh.x.size() * mesh.y.size();
		nvtr[i][5] = nvtr[i][4] + 1;
		nvtr[i][6] = nvtr[i][4] + mesh.x.size();
		nvtr[i][7] = nvtr[i][6] + 1;
	}
}

// Генерирует номера узлов элементов для двумерной сетки
void STRCTRS::gen_nvtr_rz(class MESH_RZ& mesh)
{
	double kel = mesh.kel;
	nvtr.resize(mesh.kel);
	for (int i = 0; i < kel; i++) { nvtr[i].resize(4); }

	int n;
	for (int i = 0; i < kel; i++) {
		n = i / (mesh.r.size() - 1) * (mesh.r.size()) + i % (mesh.r.size() - 1) + 1;

		nvtr[i][0] = n;
		nvtr[i][1] = n + 1;
		nvtr[i][2] = n + mesh.r.size();
		nvtr[i][3] = n + mesh.r.size() + 1;

	}
}

// Генерирует координаты узлов трехмерной сетки
void STRCTRS::gen_coord(class MESH& mesh){
	coord.resize(mesh.kuzlov);
	int xs = mesh.x.size();
	int ys = mesh.y.size();
	int zs = mesh.z.size();
	int x, y, z;
	for (int i = 0; i < mesh.kuzlov; i++) {
		x = i % xs;
		y = (i % (xs * ys)) / xs;
		z = i / (xs * ys);
		coord[i].resize(3);
		coord[i] = { mesh.x[x], mesh.y[y], mesh.z[z] };
	}

}

// Генерирует кооридната узлов двумерной сетки
```

```
void STRCTRS::gen_coord_rz(class MESH_RZ& mesh){
	coord.resize(mesh.kuzlov);
	int rs = mesh.r.size();
	int zs = mesh.z.size();
	int r, h;
	for (int i = 0; i < mesh.kuzlov; i++) {
		r = i % rs;
		h = i / rs;
		coord[i].resize(2);
		coord[i] = { mesh.r[r], mesh.z[h] };
	}

}

// По коорinатам определит номера материала
// Обход обратный: будет получен материал аномального поля, если он не существует, то
нормального
int STRCTRS::get_mat_from_sreda_reverse(class SREDA& sreda, double x, double y, double z) {
	// Будем обходить подобласти в обратном порядке (учитывая что в файле среда
подоблости заданны вложением по порядку)
	for (int i = sreda.elms.size() - 1; i >= 0; i--) {
		/*printf("%lf %lf %lf\n", sreda.elms[i][X0_COORD], x,
sreda.elms[i][X1_COORD]);
		printf("%lf %lf %lf\n", sreda.elms[i][Y0_COORD], y, sreda.elms[i][Y1_COORD]);
		printf("%lf %lf %lf\n", sreda.elms[i][Z0_COORD], z,
sreda.elms[i][Z1_COORD]);*/
		if (sreda.elms[i][X0_COORD] < x && x < sreda.elms[i][X1_COORD] &&
			sreda.elms[i][Y0_COORD] < y && y < sreda.elms[i][Y1_COORD] &&
			sreda.elms[i][Z0_COORD] < z && z < sreda.elms[i][Z1_COORD]) {
			return sreda.elms[i][MAT_N];
		}
	}
	return 0;
}

// По коорinатам определит номера материала
// Обход прямой: в первую очередь будет получен материал нормального поля
int STRCTRS::get_mat_from_sreda_direct(class SREDA& sreda, double x, double y, double z) {
	// Будем обходить подобласти в обратном порядке (учитывая что в файле среда
подоблости заданны вложением по порядку)
	for (int i = 0; i< sreda.elms.size(); i++) {
		/*printf("%lf %lf %lf\n", sreda.elms[i][X0_COORD], x,
sreda.elms[i][X1_COORD]);
		printf("%lf %lf %lf\n", sreda.elms[i][Y0_COORD], y, sreda.elms[i][Y1_COORD]);
		printf("%lf %lf %lf\n", sreda.elms[i][Z0_COORD], z,
sreda.elms[i][Z1_COORD]);*/
		if (sreda.elms[i][X0_COORD] < x && x < sreda.elms[i][X1_COORD] &&
			sreda.elms[i][Y0_COORD] < y && y < sreda.elms[i][Y1_COORD] &&
			sreda.elms[i][Z0_COORD] < z && z < sreda.elms[i][Z1_COORD]) {
			return sreda.elms[i][MAT_N];
		}
	}
	return 0;
}

// По коорinатам определит номер материала для двумерной задачи
int STRCTRS::get_mat_from_sreda_rz(class SREDA& sreda, double z) {
	// Будем обходить подобласти в прямом порядке (учитывая что в файле среда подоблости
заданны вложением по порядку)
	// Считается, что для двумерной задачи элементы, которые не располагаются от начала
до конца границы являются аномальными
	for (int i = 0; i < sreda.elms.size(); i++) {
		if (sreda.elms[i][Z0_COORD] < z && z < sreda.elms[i][Z1_COORD]) {
			return sreda.elms[i][MAT_N];
		}
```

```
	}
	return 0;
}

// Генерирует номера материалов параллелепипедов для трехмерной задачи
void STRCTRS::gen_nvkat2d(class SREDA& sreda, class MESH& mesh) {
	nvkat2d.resize(mesh.kel);
	double x, y, z;
	for (int i = 0; i < mesh.kel; i++) {
		// Координаты центров элементов
		x = (coord[nvtr[i][0] - 1][0] + coord[nvtr[i][1] - 1][0]) / 2.;
		y = (coord[nvtr[i][0] - 1][1] + coord[nvtr[i][2] - 1][1]) / 2.;
		z = (coord[nvtr[i][0] - 1][2] + coord[nvtr[i][4] - 1][2]) / 2.;
		nvkat2d[i] = get_mat_from_sreda_reverse(sreda, x, y, z);
	}
}

void STRCTRS::gen_nvkat2dr(class SREDA& sreda, class MESH& mesh) {
	nvkat2dr.resize(mesh.kel);
	double x, y, z;
	for (int i = 0; i < mesh.kel; i++) {
		// Координаты центров элементов
		x = (coord[nvtr[i][0] - 1][0] + coord[nvtr[i][1] - 1][0]) / 2.;
		y = (coord[nvtr[i][0] - 1][1] + coord[nvtr[i][2] - 1][1]) / 2.;
		z = (coord[nvtr[i][0] - 1][2] + coord[nvtr[i][4] - 1][2]) / 2.;
		nvkat2dr[i] = get_mat_from_sreda_direct(sreda, x, y, z);
	}
}

// Генерирует номера материалов прямоугольников для двумерной задачи
void STRCTRS::gen_nvkat2d_rz(class MESH_RZ& mesh, class SREDA& sreda) {
	nvkat2d.resize(mesh.kel);
	double r, z;
	for (int i = 0; i < mesh.kel; i++) {
		// Координаты центров элементов
		//r = (coord[nvtr[i][0] - 1][0] + coord[nvtr[i][1] - 1][0]) / 2.;
		z = (coord[nvtr[i][0] - 1][1] + coord[nvtr[i][2] - 1][1]) / 2.;
		nvkat2d[i] = get_mat_from_sreda_rz(sreda, z);
	}
}

// Генерирует номера узлов с первыми нулевыми краевыми для трехмерной задачи
void STRCTRS::gen_l1(class MESH& mesh, vector<double>& edge_conditions) {
	l1.resize(0);
	set<int> buff;
	for (int i = 0; i < mesh.kuzlov; i++) {
		// Если указано, что на грани x = x0 заданы первые краевые,
		// то если координата текущей точки совпадает с x0, тогда добавить ее номер.
		// С остальными гранями аналогично
		if (edge_conditions[L1_X0]) { if (coord[i][0] == mesh.x[0]) { buff.insert(i +
1); } }
		if (edge_conditions[L1_X1]) { if (coord[i][0] == mesh.x[mesh.x.size() - 1]) {
buff.insert(i + 1); } }

		if (edge_conditions[L1_Y0]) { if (coord[i][1] == mesh.y[0]) { buff.insert(i +
1); } }
		if (edge_conditions[L1_Y1]) { if (coord[i][1] == mesh.y[mesh.y.size() - 1]) {
buff.insert(i + 1); } }

		if (edge_conditions[L1_Z0]) { if (coord[i][2] == mesh.z[0]) { buff.insert(i +
1); } }
		if (edge_conditions[L1_Z1]) { if (coord[i][2] == mesh.z[mesh.z.size() - 1]) {
buff.insert(i + 1); } }
	}
	l1.insert(l1.begin(), buff.begin(), buff.end());
```

```
}
// Генерирует номера узлов с первыми нулевыми краевыми для двумерной задачи
void STRCTRS::gen_l1_rz(class MESH_RZ& mesh, vector<double>& edge_conditions) {
	l1.resize(0);
	set<int> buff;
	/* Краевые условия в двумерной задаче заданы следующим образом

	  S2=0
	 ----------
	 |        |
s2=0 |        | S1 = 0
	 |        |
	 ----------
	      S1 = 0
	 */
	for (int i = 0; i < mesh.kuzlov; i++) {
		if (coord[i][0] == mesh.r[mesh.r.size() - 1]) { buff.insert(i + 1); }
		else if (coord[i][1] == mesh.z[mesh.z.size() - 1]) { buff.insert(i + 1); }
	}
	l1.insert(l1.begin(), buff.begin(), buff.end());
}

void STRCTRS::gen_l1_p(class MESH& mesh, vector<double>& edge_conditions) {
	l1.resize(0);
	set<int> buff;
	for (int i = 0; i < mesh.kuzlov; i++) {
		// Для теста на полином зададим первые краевые на всех границах
		if (coord[i][0] == mesh.x[0]) { buff.insert(i + 1); }
		else if (coord[i][0] == mesh.x[mesh.x.size() - 1]) { buff.insert(i + 1); }
		else if (coord[i][1] == mesh.y[0]) { buff.insert(i + 1); }
		else if (coord[i][1] == mesh.y[mesh.y.size() - 1]) { buff.insert(i + 1); }
		else if (coord[i][2] == mesh.z[0]) { buff.insert(i + 1); }
		else if (coord[i][2] == mesh.z[mesh.z.size() - 1]) { buff.insert(i + 1); }
	}
	l1.insert(l1.begin(), buff.begin(), buff.end());
}
void STRCTRS::gen_l1_rz_p(class MESH_RZ& mesh, vector<double>& edge_conditions) {
	l1.resize(0);
	set<int> buff;
	/* Для теста на полином зададим первые краевые на всех границах

	    S1
	 ----------
	 |        |
  S1 |        | S1
	 |        |
	 ----------
	       S1
	 */
	for (int i = 0; i < mesh.kuzlov; i++) {
		if (coord[i][0] == mesh.r[mesh.r.size() - 1]) { buff.insert(i + 1); }
		else if (coord[i][0] == mesh.r[0]) { buff.insert(i + 1); }
		else if (coord[i][1] == mesh.z[mesh.z.size() - 1]) { buff.insert(i + 1); }
		else if (coord[i][1] == mesh.z[0]) { buff.insert(i + 1); }
	}
	l1.insert(l1.begin(), buff.begin(), buff.end());
}


// nvtr : 8 * int номера вершин прямоугольников
// nvkat2d : 1 * int номера матреиала прямоугольников
// coord : 3 * double координаты вершин
// l1 : 1 * int (номера вершин с первым нулевым краевым условием)
void STRCTRS::gen_structures(SREDA& sreda, MESH& mesh) {
	gen_nvtr(mesh);
```

```
	gen_coord(mesh);
	gen_nvkat2d(sreda,mesh);
	gen_nvkat2dr(sreda, mesh);
	gen_l1(mesh, sreda.edge_conditions);
}
// nvtr : 4 * int номера вершин прямоугольников
// nvkat2d : 1 * int номера матреиала прямоугольников
// coord : 2 * double координаты вершин
// l1 : 1 * int (номера вершин с первым нулевым краевым условием)
void STRCTRS::gen_structures_rz(SREDA& sreda, MESH_RZ& mesh) {
	gen_nvtr_rz(mesh);
	gen_coord_rz(mesh);
	gen_nvkat2d_rz(mesh, sreda);
	gen_l1_rz(mesh, sreda.edge_conditions);
}

// Тест на полиномы
void STRCTRS::gen_structures_p(SREDA& sreda, MESH& mesh) {
	gen_nvtr(mesh);
	gen_coord(mesh);
	gen_nvkat2d(sreda, mesh);
	gen_l1_p(mesh, sreda.edge_conditions);
}
void STRCTRS::gen_structures_rz_p(SREDA& sreda, MESH_RZ& mesh) {
	gen_nvtr_rz(mesh);
	gen_coord_rz(mesh);
	gen_nvkat2d_rz(mesh, sreda);
	gen_l1_rz_p(mesh, sreda.edge_conditions);
}
```