МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Соловейчик Ю.Г.
(фамилия, имя, отчество)

(подпись)

«11» марта 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Янковского Егора Вадимовича

(фамилия, имя, отчество студента – автора работы)

Расчет трехмерного нестационарного поля индукционного источника с помощью многоэтапной схемы разделения поля

(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 01.03.02. Прикладная математика и информатика
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ

Соловейчик Ю.Г.
(фамилия, имя, отчество)

д.т.н., профессор
(ученая степень, ученое звание)

(подпись, дата)

Автор выпускной квалификационной работы

Янковский Е.В.
(фамилия, И.О.)

ФПМИ, ПМ-02
(факультет, группа)

(подпись, дата)

Новосибирск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Соловейчик Ю.Г.
(фамилия, имя, отчество)

«11» марта 2024 г.

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Янковскому Егору Вадимовичу***
(фамилия, имя, отчество студента)

Направление подготовки *01.03.02. Прикладная математика и информатика*

Факультет Прикладной математики и информатики

Тема ***Расчет трехмерного нестационарного поля индукционного источника с помощью многоэтапной схемы разделения поля***

Исходные данные (или цель работы):

Разработка программы для моделирования трёхмерного нестационарного поля индукционного источника с помощью многоэтапной схемы разделения поля.

Структурные части работы:

Изучение теоретических материалов
Реализация программного модуля для решения осесимметричной задачи
Реализация программного модуля для решения трехмерной векторной задачи
Тестирование разработанной программы

Проведение исследования на многоэтапную схему разделения поля

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Соловейчик Ю.Г.	*Янковский Е.В.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
д.т.н., профессор	*ФПМИ, ПМ-02*
(ученая степень, ученое звание)	(факультет, группа)
11.03.2024 г.	*11.03.2024 г.*
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № *1308/2* от «*11*» *марта* *2024* г.

ВКР сдана в ГЭК №_____, тема сверена с данными приказа

(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчёт 91 с., 4 ч., 24 рис., 26 табл., 13 источников, 1 прил.

ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ ЭЛЕКТРОМАГНИТНОГО ПОЛЯ, МЕТОД КОНЕЧНЫХ ЭЛЕМЕНТОВ, МНОГОЭТАПНАЯ СХЕМА РАЗДЕЛЕНИЯ ПОЛЕЙ

Цель работы: разработка программы для численного моделирования нестационарного электромагнитного поля в трёхмерной среде, создаваемого индукционным источником тока, при помощи многоэтапной схемы разделения полей.

В процессе работы был разработан и протестирован программный модуль численного моделирования электромагнитного поля с помощью многоэтапной схемы разделения полей.

С помощью программы проводилось исследование поведения поля в приповерхностных слоях земной коры с различными значениями удельной электропроводности горизонтально-слоистой среды и аномальных объектов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С увеличением потребности в природных ресурсах с конца XIX-ого – начала XX-ого века развивались методы поиска и исследования земных пород и руд. Одним из наиболее распространенных методов является электроразведка. Конкретно в электроразведке сейчас насчитывается свыше пятидесяти различных методов и модификаций, предназначенных как для глубинных исследований, так и для изучения верхней части разреза земной коры. Одним из наиболее распространённым в наши дня является индукционный метод. Его принцип заключается в следующем: под влиянием переменного электрического или магнитного поля в земле за счет феномена магнитной индукции возникает электромагнитное поле. Зная точно параметры источника поля, можно измерять различные электрические и магнитные компоненты индуцированного поля, восстанавливая по ним значения параметров среды.

Помимо возможности нахождения параметров среды, что является обратной задачей по определению, можно изучать и поведение самого индукционного поля. Зная значения силы тока на источнике и физические характеристики верхних слоёв земной коры можно смоделировать электромагнитное поле и изучать характер его поведения в зависимости, например, от количества неоднородностей в земной среде или параметров горизонтально-слоистой среды Земли [1].

Однако на тот момент аналитических методов для моделирования электромагнитных полей в трёхмерном пространстве не существовало. Достаточно быстро решать такие задачи стало возможно лишь с развитием ЭВМ – в 30-х – 70-х годах XX–го века. Использование компьютеров позволило численно моделировать сложные поля, для которых было необходимо преобразовывать их таким образом, чтобы разделить аномалии в зависимости от глубины расположения источников поля и обособлять такие изменения поля, которые соответствуют аномалиям тел простейших форм [2].

В это же время для решения сложных уравнений в частных производных, описывающих большинство всех физический процессов в природе, были разработаны такие методы как: метод конечных элементов (МКЭ), метод конечных объемов (МКО) и метод конечных разностей (МКР). МКР известен своей простотой в реализации и используется при решении очень большого класса задач математической физики. Однако данный метод имеет ряд существенных недостатков, например, невозможность решения задачи на объектах, имеющих сложную геометрию. Куда более универсальными численными методами решения задач математической физики являются МКО и МКЭ. Оба метода применяются для решения задач аэрогидродинамики, электростатики, упругости и прочности материалов. Данные методы позволяют решать задачи для предметов любой геометрической сложности, а также находить значение искомой функции в любом месте расчетной области без дополнительного применения интерполяции полученного результата. Сегодня есть очень много профессионального коммерческого программного обеспечения для анализа проблем методом конечных элементов или контрольных объёмов, например американская ANSYS или российская Штуцер-МКЭ.

При численном моделировании электромагнитных полей, большее предпочтение отдаётся МКЭ нежели МКО, поскольку при решении задачи методом конечных элементов можно воспользоваться модификацией данного метода - векторным МКЭ. Данная модификация позволяет куда лучше найти решение задачи при наличии, например, разрывности напряженности электрического поля и удельной электрической проводимости среды.

В данной работе будет рассмотренна возможность расчёта нестационарного электромагнитного поля, создаваемого индукционным источником при использовании как скалярного, так и векторного МКЭ, а также с применением многоэтапной технологии разделения полей.

Исследования будем проводить на области размером $\Omega \in [-55000; 55000]_x \times [-55000; 55000]_y \times [-25000; 25000]_z$. Данная область включает в себя несколько разных слоев земных пород, имеющих различные удельные значения физических величин. Также рассмотрим процесс добавления нескольких аномальных пород, характерных различным земным рудам.

При написании программы для выпускной квалификационной работы использовались следующие языки программирования: для математических расчётов – C# 12 на платформе .NET 8.0, для визуализации полученных результатов – Python 3.12.2 с пакетами matplotlib версии 3.8.2 и numpy версии 1.26.4.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. АППАРАТ МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ

Математическая модель, описывающая поведение электромагнитного поля в пространстве, известна в наши дни, как система уравнений Максвелла. Она позволяет описывать взаимосвязь сразу нескольких физических величин: напряжённости электрического $\overrightarrow{\mathbf{E}}$ и магнитного $\overrightarrow{\mathbf{H}}$ полей, а также индукцию магнитного поля $\overrightarrow{\mathbf{B}}$. Большинство вычислительных задач электромагнетизма базируются на дифференциальной форме системы уравнений Максвелла:

$$\mathrm{rot}\overrightarrow{\mathbf{H}} = \overrightarrow{\mathbf{J}^{\text{ст}}} + \sigma\overrightarrow{\mathbf{E}} + \frac{\partial\left(\varepsilon\overrightarrow{\mathbf{E}}\right)}{\partial t}, \tag{1.1}$$

$$\mathrm{rot}\,\overrightarrow{\mathbf{E}} = -\frac{\partial\overrightarrow{\mathbf{B}}}{\partial t}, \tag{1.2}$$

$$\mathrm{div}\overrightarrow{\mathbf{B}} = 0, \tag{1.3}$$

$$\mathrm{div}\varepsilon\overrightarrow{\mathbf{E}} = \rho, \tag{1.4}$$

где $\overrightarrow{\mathbf{J}^{\text{ст}}}$ – вектор плотностей сторонних токов, σ – удельная электрическая проводимость среды, ε – диэлектрическая проницаемость среды, а ρ – объёмная плотность стороннего электрического заряда.

Основное преимущество использования системы уравнений (1.1) – (1.4) в дифференциальной форме, заключается в возможности учитывать нелинейность, анизотропию и другие нетривиальные аспекты среды [3].

Пусть электромагнитное поле возбуждается индукционным источником. В таком случае, при отсутствии аномальных объектов, будем решать задачу в цилиндрических координатах. Источник поля в таком случае описывается точкой, расположенной на некотором расстоянии, достаточно далёком от границы расчётной области. Тогда при условии однородности среды по магнитной проницаемости и отстутствия токов смещения электромагнитное поле полностью описывается одной компонентой $A_\varphi = A_\varphi(r, z, t)$ вектор-потенциала $\overrightarrow{\mathbf{A}}$. Функция $A_\varphi(r, z, t)$ может быть найдена из решения двумерного уравнения (1.5):

$$-\frac{1}{\mu_0}\Delta A_\varphi + \frac{A_\varphi}{\mu_0 r^2} + \sigma\frac{\partial A_\varphi}{\partial t} = J_\varphi, \qquad (1.5)$$

где $\mu_0 = 4 \cdot \pi \cdot 10^{-7} = 1.25663753 \cdot 10^{-6}$ Гн/м – магнитная постоянная, J_φ - источник стороннего тока, описываемый дельта-функцией, равной 1 в одной из подобласти, описывающей источник поля, и 0 во всех остальных [4]. Удельную электропроводность σ представим в виде кусочно-постоянной функции, описывающей физические характеристики горизонтально-слоистой среды. Потребуем, чтобы на всех границах было главное краевое условие $A_\varphi(r, z, t)|_s = 0$. Тогда решение задачи (1.5) с главными однородными условиями на границах будем называть первичным или нормальным полем.

Решением задачи на оценку влияния аномальных объектов в горизонтально-слоистой среде будем называть вторичным (добавочным) полем. Также, как и в (1.5) потребуем на всех границах главное однородное краевое условие $\overrightarrow{\mathbf{A}} \times \overrightarrow{\mathbf{n}}|_s = 0$. Тогда, нестационарный процесс, возникающий после выключения источника тока в круглой обмотке, описывается следствием из уравнения (1.1) [5]:

$$\mathrm{rot}\left(\frac{1}{\mu_o}\mathrm{rot}\,\overrightarrow{\mathbf{A}}^+\right)+\sigma\frac{\partial\overrightarrow{\mathbf{A}}^+}{\partial t}=(\sigma-\sigma_n)\overrightarrow{\mathbf{E}}^n, \qquad (1.6)$$

где σ_n – значение удельной электрической проводимости среды на нормальном слое, $\overrightarrow{\mathbf{E}}^n$ – напряжённость первичного электрического поля, $\overrightarrow{\mathbf{A}}^+$ – значение вектор-потенциала на добавочном поле.

1.2. ОПИСАНИЕ РАСЧЁТНОЙ ОБЛАСТИ

Пусть у нас имеется расчётная область, геометрически представленная в виде параллелепипеда: $\Omega \in [-55000, 55000]_x \times [-55000, 55000]_y \times [-25000, 25000]_z$. Внутри неё имеются слои воздуха, и некоторых пород верхних слоёв земной коры. Тогда половина продольного диагонального среза горизонтально-слоистой среды изображена на рисунке 1.1. Будем её использовать в качестве расчётной области для двумерной задачи. В среде, обозначенной коричневым цветом задано значение $\sigma_1 = 0.01$ См/м, в бледной $\sigma_2 = 0.005$ См/м и в зелёной $\sigma_3 = 0.001$ См/м. Поскольку воздух является диэлектриком, значение удельной электропроводности для него $\sigma_{\text{возд.}} = 0$ См/м.

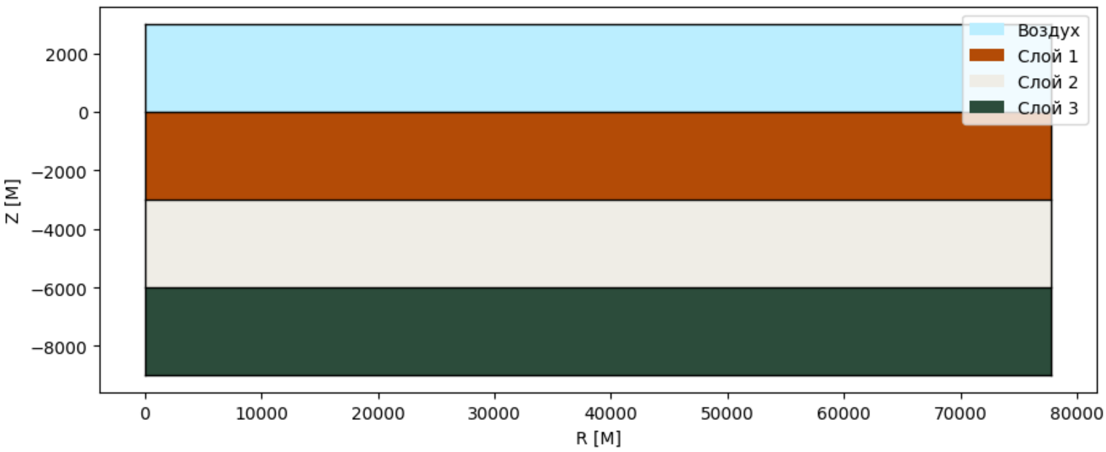

Рисунок 1.1 – Срез горизонтально-слоистой среды

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. МНОГОЭТАПНАЯ СХЕМА РАЗДЕЛЕНИЯ ПОЛЯ

Предлагаемый в работе подход к численному моделированию основан на технологии разделения полей, позволяющей существенно сократить вычислительные затраты. В рассматриваемой задаче под неоднородностями (аномалиями) будем понимать трёхмерные геологические объекты, существенно отличные от сопротивления вмещающей горизонтально-слоистой среды [6]. Сначала решение ищется на поле, создаваемом в среде, максимально упрощённой относительно исходной. Её решение, которое может быть меньшей размерности, берётся в качестве основного поля первого уровня. На основе этого поля решается задача на добавочное поле, в которую включается часть неоднородностей исходной задачи, дающих максимальный вклад в искомое решение. Используемая для нахождения этого добавочного поля сетка строится так, чтобы максимально учесть влияние источников, порождённых включёнными в на этом этапе неоднородностями [7].

Далее в качестве основного поля будет учитываться сумма основного и добавочного на предыдущем этапе выделения. Новое добавочное поле будет формироваться из учёта следующих по влиянию на решение исходной задачи. Процесс можно продолжать до тех пор, пока не будут учтены все неоднородности среды.

2.2. ВАРИАЦИОННАЯ ПОСТАНОВКА ДВУМЕРНОЙ ЗАДАЧИ

Перед тем, как приступить к решению задачи на нормальное поле, необходимо перевести уравнение (1.5) в вариационную форму. В основе использования МКЭ лежит вариационная постановка, в которой решение краевой задачи заменяется минимизацией функционала невязки. Областью определения этого функционала обычно является Гильбертово пространство функций H^m, содержащее в качестве одного из своих элементов решение данной краевой задачи. Потребуем, чтобы невязка $R(A_\varphi) = -\frac{1}{\mu_0}\Delta A_\varphi + \frac{A_\varphi}{\mu_0 r^2} + \sigma\frac{\partial A_\varphi}{\partial t} - J_\varphi$ дифференциального уравнения (1.5) была ортогональна в смысле скалярного произведения пространства $L^2(\Omega) \equiv H_0$ некоторому пространству Φ функций v, которое называется пространством пробных функций, т.е.:

$$\int\limits_\Omega \left(-\frac{1}{\mu_0}\Delta A_\varphi + \frac{A_\varphi}{\mu_0 r^2} + \sigma\frac{\partial A_\varphi}{\partial t} - J_\varphi\right) v d\Omega = 0, \tag{2.1}$$

$$\int\limits_\Omega \left(-\frac{1}{\mu_0}\Delta A_\varphi + \frac{A_\varphi}{\mu_0 r^2} + \sigma\frac{\partial A_\varphi}{\partial t}\right) v d\Omega = \int\limits_\Omega J_\varphi v d\Omega. \tag{2.2}$$

Используя формулу Грина, получим:

$$\begin{gathered}\int\limits_\Omega \frac{1}{\mu_0}\mathrm{grad} A_\varphi \cdot \mathrm{grad} v d\Omega - \int\limits_S \frac{1}{\mu_0}\frac{\partial A_\varphi}{\partial n} v dS + \int\limits_\Omega \frac{A_\varphi}{\mu_0 r^2} d\Omega + \\ + \int\limits_\Omega \sigma\frac{\partial A_\varphi}{\partial t} v d\Omega = \int\limits_\Omega J_\varphi v d\Omega.\end{gathered} \tag{2.3}$$

Так как пространство пробных функций H_0^1 имеет след 0, то слагаемое $\int\limits_S \frac{1}{\mu_0}\frac{\partial A_\varphi}{\partial n} v dS$ не оказывает никакого вклада в (2.3). Таким образом для (1.5) получим уравнение в слабой форме:

$$\int\limits_\Omega \frac{1}{\mu_0}\mathrm{grad} A_\varphi \cdot \mathrm{grad} v d\Omega + \int\limits_\Omega \frac{A_\varphi}{\mu_0 r^2} v d\Omega + \int\limits_\Omega \sigma\frac{\partial A_\varphi}{\partial t} v d\Omega = \int\limits_\Omega J_\varphi v d\Omega. \tag{2.4}$$

2.3. КОНЕЧНОЭЛЕМЕНТНАЯ ДИСКРЕТИЗАЦИЯ ДВУМЕРНОЙ ЗАДАЧИ

Сетку для решения задачи будем строить используя прямоугольные элементы. При использовании многоэтапной схемы разделения поля для первичного слоя допускается, что сетка может быть равномерной, не сгущающейся к неоднородностям среды. Поэтому рассмотрим сетку на 26878 узлов, имеющей 302 узла по оси r и 89 узлов по оси z.

Для решения задачи будем использовать билинейные базисные функции, задаваемые линейными функциями на $\Omega_{ps} = [r_p, r_{p+1}] \times [z_s, z_{s+1}]$ следующего вида:

$$\begin{cases} \hat{\psi}_1(r,z) = R_1(r)Z_1(z), \\ \hat{\psi}_2(r,z) = R_2(r)Z_1(z), \\ \hat{\psi}_3(r,z) = R_1(r)Z_2(z), \\ \hat{\psi}_4(r,z) = R_2(r)Z_2(z). \end{cases} \tag{2.5}$$

где:

$$\begin{cases} R_1(r) = \frac{r_{p+1}-r}{r_{p+1}-r_p}, \\ R_2(r) = \frac{r-r_p}{r_{p+1}-r_p}, \\ Z_1(z) = \frac{z_{p+1}-z}{z_{p+1}-z_p}, \\ Z_2(z) = \frac{z-z_p}{z_{p+1}-z_p}. \end{cases} \tag{2.6}$$

2.4. ПОСТРОЕНИЕ МАТРИЦ МАСС И ЖЁСТКОСТИ ДЛЯ ДВУМЕРНОЙ ЗАДАЧИ

Будем считать, что функции A_φ и v в вариационном уравнении (2.4) являются компонентами конечноэлементного функционального пространства, натянутого на базисные функции $\hat{\psi}_j$, где $j = \overline{1,n}$, т.е.

$$\begin{cases} A_\varphi = \sum\limits_{j=1}^{n} q_j^{A_\varphi} \hat{\psi}_j, \\ v = \sum\limits_{j=1}^{n} q_j^{v} \hat{\psi}_j, \end{cases} \tag{2.7}$$

где $q_j^{A_\varphi}$ – веса в разложении функции u по базисным функциями $\hat{\psi}_j$, а q_j^v – веса в разложении функции v по тем же базисным функциями $\hat{\psi}_j$. Нетрудно убедиться, что с учётом разложения (2.7) вариационное уравнение (2.4) эквивалентно системе уравнений

$$\begin{gathered} \sum_{j=1}^{n} \left(\int\limits_{\Omega} -\frac{1}{\mu_0} \mathrm{grad}\hat{\psi}_i \cdot \mathrm{grad}\hat{\psi}_j d\Omega + \int\limits_{\Omega} \frac{\hat{\psi}_i \hat{\psi}_j}{\mu_0 r^2} d\Omega \right) + \\ + \sum_{j=1}^{n} \int\limits_{\Omega} \sigma \frac{\partial \hat{\psi}_i}{\partial t} \hat{\psi}_j d\Omega \cdot q_j^u = \int\limits_{\Omega} J_\varphi \hat{\psi}_i d\Omega. \end{gathered} \tag{2.8}$$

Так как задача решается в цилиндрических координатах, то в уравнении (2.8) $d\Omega = r dr dz$, где r - якобиан перехода от декартовых к цилиндрическим координатам.

$$\begin{gathered} \sum_{j=1}^{n} \left(\int\limits_{r} \int\limits_{z} r \frac{1}{\mu_0} \mathrm{grad}\hat{\psi}_i \cdot \mathrm{grad}\hat{\psi}_j dr dz + \int\limits_{r} \int\limits_{z} r \frac{\hat{\psi}_i}{\mu_0 r^2} \hat{\psi}_j dr dz \right) \cdot q_j^u + \\ + \sum_{j=1}^{n} \int\limits_{r} \int\limits_{z} r\sigma \frac{\partial \hat{\psi}_i}{\partial t} \hat{\psi}_j dr dz \cdot q_j^u = \int\limits_{r} \int\limits_{z} r J_\varphi \hat{\psi}_i dr dz. \end{gathered} \tag{2.9}$$

Рассмотрим аппроксимацию по пространству (r, z) в уравнении (2.9) для произвольных i и j

$$\left(\int\limits_r \int\limits_z r\frac{1}{\mu_0}\mathrm{grad}\hat{\psi}_i \cdot \mathrm{grad}\hat{\psi}_j drdz + \int\limits_r \int\limits_z r\frac{\hat{\psi}_i}{\mu_0 r^2}\hat{\psi}_j drdz\right) \cdot q_j^u = \\ = \int\limits_r \int\limits_z rJ_\varphi \hat{\psi}_i drdz. \tag{2.10}$$

Пусть $\hat{\psi}_i = R_i(r) \cdot Z_i(z)$, а $\hat{\psi}_j = R_j(r) \cdot Z_j(z)$, тогда

$$\left(\int\limits_r \int\limits_z r\frac{1}{\mu_0}\mathrm{grad}(R_i Z_i) \cdot \mathrm{grad}(R_j Z_j) drdz + \int\limits_r \int\limits_z r\frac{R_i Z_i}{\mu_0 r^2} R_j Z_j drdz\right) \cdot q_j^u = \\ = \int\limits_r \int\limits_z rJ_\varphi R_i Z_i drdz. \tag{2.11}$$

Преобразовав (2.11) получим следующее

$$\left(\frac{1}{\mu_0}\left(\int\limits_r r\frac{dR_i}{dr}\frac{dR_j}{dr}dr \cdot \int\limits_z Z_i Z_j dz + \int\limits_r rR_i R_j dr \cdot \int\limits_z \frac{dZ_i}{dz}\frac{dZ_j}{dz}dz\right)\right) q_j^u + \\ + \left(\frac{1}{\mu_0}\int\limits_r \frac{1}{r}R_i R_j dr \int\limits_z Z_i Z_j dz\right) q_j^u = \int\limits_r \int\limits_z rJ_\varphi R_i Z_i drdz. \tag{2.12}$$

Получим локальные матрицы масс $\hat{\mathbf{M}}$ и жёсткости $\hat{\mathbf{G}}$, а также локальный вектор правой части $\hat{\mathbf{b}}$. Поскольку на одном элементе для аппроксимации билинейными базисными функциями необходимо 4 узла, то локальные матрицы будут иметь размерность 4×4, а векторы 4×1. Заметим, что каждый интеграл в уравнении (2.12) является компонентой интеграла, лежащего в основе построения локальных матриц масс и жёсткости для одномерных задач (2.13) – (2.17).

$$\hat{\mathbf{G}}_r^{1D} = \frac{r_k + \frac{h_k}{2}}{h_k} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{2.13}$$

$$\hat{\mathbf{M}}_r^{1D} = \frac{\hat{\gamma} h_k}{6} \left(r_k \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} + \frac{h_k}{2} \begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix} \right), \tag{2.14}$$

$$\begin{gathered} \hat{\mathbf{M}}_{rr}^{1D} = \ln\left(1 + \frac{1}{d}\right) \begin{pmatrix} (1+d)^2 & -d(1+d) \\ -d(1+d) & d^2 \end{pmatrix} - d \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \\ + \frac{1}{2} \begin{pmatrix} -3 & 1 \\ 1 & 1 \end{pmatrix}, \end{gathered} \tag{2.15}$$

где $d = \frac{r_k}{h_k}$.

$$\hat{\mathbf{G}}_z^{1D} = \frac{\hat{\lambda}}{h_k} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{2.16}$$

$$\hat{\mathbf{M}}_z^{1D} = \frac{\hat{\gamma} h_k}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{2.17}$$

Тогда элементы верхнего треугольника матрицы жесткости для двумерных задач, можем представить в виде:

$$\begin{aligned}
\hat{\mathbf{G}}_{11} &= \left(\hat{\mathbf{G}}_{r11}^{1D}\hat{\mathbf{M}}_{z11}^{1D} + \hat{\mathbf{M}}_{r11}^{1D}\hat{\mathbf{G}}_{z11}^{1D}\right), & \hat{\mathbf{G}}_{12} &= \left(\hat{\mathbf{G}}_{r12}^{1D}\hat{\mathbf{M}}_{z11}^{1D} + \hat{\mathbf{M}}_{r12}^{1D}\hat{\mathbf{G}}_{z11}^{1D}\right), \\
\hat{\mathbf{G}}_{13} &= \left(\hat{\mathbf{G}}_{r11}^{1D}\hat{\mathbf{M}}_{z12}^{1D} + \hat{\mathbf{M}}_{r11}^{1D}\hat{\mathbf{G}}_{z12}^{1D}\right), & \hat{\mathbf{G}}_{14} &= \left(\hat{\mathbf{G}}_{r12}^{1D}\hat{\mathbf{M}}_{z12}^{1D} + \hat{\mathbf{M}}_{r12}^{1D}\hat{\mathbf{G}}_{z12}^{1D}\right), \\
\hat{\mathbf{G}}_{22} &= \left(\hat{\mathbf{G}}_{r22}^{1D}\hat{\mathbf{M}}_{z11}^{1D} + \hat{\mathbf{M}}_{r22}^{1D}\hat{\mathbf{G}}_{z11}^{1D}\right), & \hat{\mathbf{G}}_{23} &= \left(\hat{\mathbf{G}}_{r21}^{1D}\hat{\mathbf{M}}_{z12}^{1D} + \hat{\mathbf{M}}_{r21}^{1D}\hat{\mathbf{G}}_{z12}^{1D}\right), \\
\hat{\mathbf{G}}_{24} &= \left(\hat{\mathbf{G}}_{r22}^{1D}\hat{\mathbf{M}}_{z12}^{1D} + \hat{\mathbf{M}}_{r22}^{1D}\hat{\mathbf{G}}_{z12}^{1D}\right), & \hat{\mathbf{G}}_{33} &= \left(\hat{\mathbf{G}}_{r11}^{1D}\hat{\mathbf{M}}_{z22}^{1D} + \hat{\mathbf{M}}_{r11}^{1D}\hat{\mathbf{G}}_{z22}^{1D}\right), \\
\hat{\mathbf{G}}_{34} &= \left(\hat{\mathbf{G}}_{r12}^{1D}\hat{\mathbf{M}}_{z22}^{1D} + \hat{\mathbf{M}}_{r12}^{1D}\hat{\mathbf{G}}_{z22}^{1D}\right), & \hat{\mathbf{G}}_{44} &= \left(\hat{\mathbf{G}}_{r22}^{1D}\hat{\mathbf{M}}_{z22}^{1D} + \hat{\mathbf{M}}_{r22}^{1D}\hat{\mathbf{G}}_{z22}^{1D}\right).
\end{aligned}$$

Верхний треугольник элементов матрицы масс, для слагаемого с коэффициентом $\frac{1}{r^2}$ может быть представлен в виде:

$$\begin{aligned}
\hat{\mathbf{M}}_{11} &= \hat{\mathbf{M}}^{1D}_{rr11}\hat{\mathbf{M}}^{1D}_{z11}, & \hat{\mathbf{M}}_{12} &= \hat{\mathbf{M}}^{1D}_{rr12}\hat{\mathbf{M}}^{1D}_{z11},\\
\hat{\mathbf{M}}_{13} &= \hat{\mathbf{M}}^{1D}_{rr11}\hat{\mathbf{M}}^{1D}_{z12}, & \hat{\mathbf{M}}_{14} &= \hat{\mathbf{M}}^{1D}_{rr12}\hat{\mathbf{M}}^{1D}_{z12},\\
\hat{\mathbf{M}}_{22} &= \hat{\mathbf{M}}^{1D}_{rr22}\hat{\mathbf{M}}^{1D}_{z11}, & \hat{\mathbf{M}}_{23} &= \hat{\mathbf{M}}^{1D}_{rr21}\hat{\mathbf{M}}^{1D}_{z12},\\
\hat{\mathbf{M}}_{24} &= \hat{\mathbf{M}}^{1D}_{rr22}\hat{\mathbf{M}}^{1D}_{z12}, & \hat{\mathbf{M}}_{33} &= \hat{\mathbf{M}}^{1D}_{rr11}\hat{\mathbf{M}}^{1D}_{z22},\\
\hat{\mathbf{M}}_{34} &= \hat{\mathbf{M}}^{1D}_{rr12}\hat{\mathbf{M}}^{1D}_{z22}, & \hat{\mathbf{M}}_{44} &= \hat{\mathbf{M}}^{1D}_{rr22}\hat{\mathbf{M}}^{1D}_{z22}.
\end{aligned}$$

Верхние треугольники элементов матрицы масс, для слагаемых с коэффициентом σ могут быть представлены в виде:

$$\begin{aligned}
\hat{\mathbf{C}}_{11} &= \hat{\mathbf{M}}^{1D}_{r11}\hat{\mathbf{M}}^{1D}_{z11}, & \hat{\mathbf{C}}_{12} &= \hat{\mathbf{M}}^{1D}_{r12}\hat{\mathbf{M}}^{1D}_{z11},\\
\hat{\mathbf{C}}_{13} &= \hat{\mathbf{M}}^{1D}_{r11}\hat{\mathbf{M}}^{1D}_{z12}, & \hat{\mathbf{C}}_{14} &= \hat{\mathbf{M}}^{1D}_{r12}\hat{\mathbf{M}}^{1D}_{z12},\\
\hat{\mathbf{C}}_{22} &= \hat{\mathbf{M}}^{1D}_{r22}\hat{\mathbf{M}}^{1D}_{z11}, & \hat{\mathbf{C}}_{23} &= \hat{\mathbf{M}}^{1D}_{r21}\hat{\mathbf{M}}^{1D}_{z12},\\
\hat{\mathbf{C}}_{24} &= \hat{\mathbf{M}}^{1D}_{r22}\hat{\mathbf{M}}^{1D}_{z12}, & \hat{\mathbf{C}}_{33} &= \hat{\mathbf{M}}^{1D}_{r11}\hat{\mathbf{M}}^{1D}_{z22},\\
\hat{\mathbf{C}}_{34} &= \hat{\mathbf{M}}^{1D}_{r12}\hat{\mathbf{M}}^{1D}_{z22}, & \hat{\mathbf{C}}_{44} &= \hat{\mathbf{M}}^{1D}_{r22}\hat{\mathbf{M}}^{1D}_{z22}.
\end{aligned}$$

Так как мы имеем сосредоточенный в точке источник J_φ, то получим следующее:

$$\int\limits_{\Omega} J_\phi \delta d\Omega = \begin{cases} 1, p \in \Omega_\epsilon \\ 0, p \notin \Omega_\epsilon \end{cases} \tag{2.18}$$

Преобразуем теперь нестационарную составляющую в уравнении (2.4) в матричную форму. Для аппроксимации задачи по времени будем использовать трёхслойную неявную схему. Тогда искомое решение A_φ на интервале (t_{j-2}, t_j) представим в следующем виде:

$$A_\varphi(r,z,t) \approx A_\varphi^{j-2}(r,z)\eta_2(t)^j + A_\varphi^{j-1}(r,z)\eta_1(t)^j + A_\varphi^{j}(r,z)\eta_0(t)^j, \tag{2.19}$$

где $\eta_2(t)^j$, $\eta_1(t)^j$, $\eta_0(t)^j$ – базисные квадратичные полиномы Лагранжа, которые записываются в виде (2.20).

$$\begin{cases} \eta_2(t)^j = \frac{(t-t_{j-1})(t-t_j)}{(t_{j-1}-t_{j-2})(t_j-t_{j-2})}, \\ \eta_1(t)^j = -\frac{(t-t_{j-2})(t-t_j)}{(t_{j-1}-t_{j-2})(t_j-t_{j-1})}, \\ \eta_0(t)^j = \frac{(t-t_{j-2})(t-t_{j-1})}{(t_j-t_{j-2})(t_j-t_{j-1})}. \end{cases} \tag{2.20}$$

Подставим выражение (2.19) в нестационарное слагаемое уравнения (2.4) на временном слое $t = t_j$:

$$\begin{gathered} \frac{\partial}{\partial t}\left(A_\varphi^{j-2}(r,z)\eta_2^j(t) + A_\varphi^{j-1}(r,z)\eta_1^j(t) + A_\varphi^{j}(r,z)\eta_0^j(t)\right)|_{t=t_j} = \\ = \tau_2 A_\varphi^{j-2}(r,z) + \tau_1 A_\varphi^{j-1}(r,z) + \tau_0 A_\varphi^{j}(r,z), \end{gathered} \tag{2.21}$$

где

$$\begin{cases} \tau_2 = \left.\frac{\partial \eta_2^j(t)}{\partial t}\right|_{t=t_j} = \frac{t_j-t_{j-1}}{(t_{j-1}-t_{j-2})(t_j-t_{j-2})}, \\ \tau_1 = \left.\frac{\partial \eta_1^j(t)}{\partial t}\right|_{t=t_j} = -\frac{t_j-t_{j-2}}{(t_{j-1}-t_{j-2})(t_j-t_{j-1})}, \\ \tau_0 = \left.\frac{\partial \eta_0^j(t)}{\partial t}\right|_{t=t_j} = \frac{(t_j-t_{j-2})+(t_j-t_{j-1})}{(t_j-t_{j-2})(t_j-t_{j-1})}. \end{cases} \tag{2.22}$$

Выполняя конечноэлементную аппроксимацию краевой задачи для уравнения (2.4), получим СЛАУ следующего вида:

$$\left(\tau_0 \mathbf{C} + \mathbf{G} + \mathbf{M}\right)\mathbf{q}^j = \mathbf{b} - \tau_2 \mathbf{C}\mathbf{q}^{j-2} + \tau_1 \mathbf{C}\mathbf{q}^{j-1}. \tag{2.23}$$

2.5. СВЯЗЬ КОМПОНЕНТ ЭЛЕКТРОМАГНИТНОГО ПОЛЯ В ДЕКАРТОВОЙ И ЦИЛИНДРИЧЕСКОЙ СИСТЕМЕ КООРДИНАТ

Как мы уже выяснили, решение задачи в (r, z) координатах можно использовать в качестве нормального поля. Перед тем, как перейти к трёхмерной постановке задачи, найдем значение E^0_φ, через следующее преобразование:

$$E^0_\varphi(r, z, t) = -\frac{\partial A^0_\varphi(r, z, t)}{\partial t}. \tag{2.24}$$

Для перевода полученных результатов в трёхмерную задачу нужно перевести полученные значения вектора-потенциала A^0_φ и напряженности электрического поля E^0_φ из цилиндрической в декартову систему координат. По формулам преобразования векторов (2.25) – (2.26) найдем составляющие компоненты базисных вектор-функций уже для трёхмерной постановки задачи.

$$\begin{cases} E^0_x(x, y, z, t) = -E^0_\varphi(\sqrt{x^2+y^2}, z, t)\frac{y}{\sqrt{x^2+y^2}}, \\ E^0_y(x, y, z, t) = E^0_\varphi(\sqrt{x^2+y^2}, z, t)\frac{x}{\sqrt{x^2+y^2}}, \\ E^0_z(x, y, z, t) = 0. \end{cases} \tag{2.25}$$

$$\begin{cases} A^0_x(x, y, z, t) = -A^0_\varphi(\sqrt{x^2+y^2}, z, t)\frac{y}{\sqrt{x^2+y^2}}, \\ A^0_y(x, y, z, t) = A^0_\varphi(\sqrt{x^2+y^2}, z, t)\frac{x}{\sqrt{x^2+y^2}}, \\ A^0_z(x, y, z, t) = 0. \end{cases} \tag{2.26}$$

2.6. ВАРИАЦИОННАЯ ПОСТАНОВКА ТРЁХМЕРНОЙ ЗАДАЧИ

Принцип построения вариационного уравнения для трёхмерной задачи в целом похож на принцип построения вариационного уравнения для двумерной задачи [8], поэтому сразу получим слабую форму для (1.6): домножим обе части на пробную вектор-функцию $\overrightarrow{\Psi}$ и проинтегрируем по всей области Ω.

$$\int\limits_{\Omega} \frac{1}{\mu_0} \mathrm{rot}\left(\mathrm{rot}\overrightarrow{\mathbf{A}}^{+}\right)\overrightarrow{\Psi}\,d\Omega + \int\limits_{\Omega} \sigma \frac{\partial \overrightarrow{\mathbf{A}}^{+}}{\partial t}\overrightarrow{\Psi}\,d\Omega = \int\limits_{\Omega} (\sigma - \sigma_n)\overrightarrow{\mathbf{E}}^{n}\overrightarrow{\Psi}\,d\Omega. \tag{2.27}$$

Применяя формулу Грина и учитывая главные краевые условия для (1.6), в итоге получим:

$$\int\limits_{\Omega} \frac{1}{\mu_0} \mathrm{rot}\,\overrightarrow{\mathbf{A}}^{+}\mathrm{rot}\,\overrightarrow{\Psi}\,d\Omega + \int\limits_{\Omega} \sigma \frac{\partial \overrightarrow{\mathbf{A}}^{+}}{\partial t}\overrightarrow{\Psi}\,d\Omega = \int\limits_{\Omega} (\sigma - \sigma_n)\,\overrightarrow{\mathbf{E}}^{n}\overrightarrow{\Psi}\,d\Omega. \tag{2.28}$$

2.7. КОНЕЧНОЭЛЕМЕНТНАЯ ДИСКРЕТИЗАЦИЯ ТРЁХМЕРНОЙ ЗАДАЧИ

Сетку для решения задач на добавочное поле будем строить с помощью прямоугольных параллелепипедов. Будем сгущать сетку к аномальным элементам в расчётной области, чтобы не создавать лишних вычислительных затрат.

Для решения задачи будем использовать билинейные базисные вектор-функции, которые задаются на параллелепипеде $\Omega_{rsp} = [x_p, x_{p+1}] \times [y_s, y_{s+1}] \times [z_p, z_{p+1}]$ следующим образом:

$$\overrightarrow{\psi}_1 = \begin{pmatrix} Y_1 \cdot Z_1 \\ 0 \\ 0 \end{pmatrix}, \qquad \overrightarrow{\psi}_2 = \begin{pmatrix} Y_2 \cdot Z_1 \\ 0 \\ 0 \end{pmatrix}, \qquad \overrightarrow{\psi}_3 = \begin{pmatrix} Y_1 \cdot Z_2 \\ 0 \\ 0 \end{pmatrix},$$

$$\overrightarrow{\psi}_4 = \begin{pmatrix} Y_2 \cdot Z_2 \\ 0 \\ 0 \end{pmatrix}, \qquad \overrightarrow{\psi}_5 = \begin{pmatrix} 0 \\ X_1 \cdot Z_1 \\ 0 \end{pmatrix}, \qquad \overrightarrow{\psi}_6 = \begin{pmatrix} 0 \\ X_2 \cdot Z_1 \\ 0 \end{pmatrix},$$

$$\overrightarrow{\psi}_7 = \begin{pmatrix} 0 \\ X_1 \cdot Z_2 \\ 0 \end{pmatrix}, \qquad \overrightarrow{\psi}_8 = \begin{pmatrix} 0 \\ X_2 \cdot Z_2 \\ 0 \end{pmatrix}, \qquad \overrightarrow{\psi}_9 = \begin{pmatrix} 0 \\ 0 \\ X_1 \cdot Y_1 \end{pmatrix},$$

$$\overrightarrow{\psi}_{10} = \begin{pmatrix} 0 \\ 0 \\ X_2 \cdot Y_1 \end{pmatrix}, \qquad \overrightarrow{\psi}_{11} = \begin{pmatrix} 0 \\ 0 \\ X_1 \cdot Y_2 \end{pmatrix}, \qquad \overrightarrow{\psi}_{12} = \begin{pmatrix} 0 \\ 0 \\ X_2 \cdot Y_2 \end{pmatrix},$$

где

$$X_1(x) = \frac{x_{r+1} - x}{x_{r+1} - x_r}, \qquad X_2(x) = \frac{x - x_r}{x_{r+1} - x_r},$$

$$Y_1(y) = \frac{y_{s+1} - y}{y_{s+1} - y_s}, \qquad Y_2(y) = \frac{y - y_s}{y_{s+1} - y_s},$$

$$Z_1(z) = \frac{z_{p+1} - z}{z_{p+1} - z_p}, \qquad Z_2(z) = \frac{z - z_p}{z_{p+1} - z_p}.$$

2.8. ПОСТРОЕНИЕ МАТРИЦ МАСС И ЖЁСТКОСТИ ДЛЯ ТРЁХМЕРНОЙ ЗАДАЧИ

Формулы для вычисления глобальных матриц жёсткости $\mathbf{G}$ и масс $\mathbf{M}$ конечноэлементной СЛАУ имеют вид:

$$\hat{\mathbf{G}}_{ij} = \int\limits_{\Omega} \frac{1}{\mu_0} \mathrm{rot} \overrightarrow{\Psi_i} \cdot \mathrm{rot} \overrightarrow{\Psi_j} d\Omega, \qquad \hat{\mathbf{M}}_{ij} = \int\limits_{\Omega} \sigma \overrightarrow{\Psi_i} \cdot \overrightarrow{\Psi_j} d\Omega.$$

Компоненты глобального вектора $\mathbf{b}$ конечноэлементной СЛАУ определяются соотношением:

$$\mathbf{b}_i = \int\limits_{\Omega} \overrightarrow{\mathbf{F}} \cdot \overrightarrow{\Psi_i} d\Omega.$$

Локальная матрица жёсткости $\hat{\mathbf{G}}$ на параллелепипеде при $\overline{\mu} = const$ принимает вид:

$$\hat{\mathbf{G}} = \frac{1}{\overline{\mu}} \begin{pmatrix} \frac{h_x h_y}{6h_z}\mathbf{G}_1 + \frac{h_x h_z}{6h_y}\mathbf{G}_2 & -\frac{h_z}{6}\mathbf{G}_2 & \frac{h_y}{6}\mathbf{G}_3 \\ -\frac{h_z}{6}\mathbf{G}_2 & \frac{h_x h_y}{6h_z}\mathbf{G}_1 + \frac{h_y h_z}{6h_x}\mathbf{G}_2 & -\frac{h_x}{6}\mathbf{G}_1 \\ \frac{h_y}{6}\mathbf{G}_3^{\mathrm{T}} & -\frac{h_x}{6}\mathbf{G}_1 & \frac{h_x h_z}{6h_y}\mathbf{G}_1 + \frac{h_y h_z}{6h_x}\mathbf{G}_2 \end{pmatrix},$$

где

$$\mathbf{G}_1 = \begin{pmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & -2 & 1 & 2 \end{pmatrix}, \qquad \mathbf{G}_2 = \begin{pmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 2 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{pmatrix},$$

$$\mathbf{G}_3 = \begin{pmatrix} -2 & 2 & -1 & 1 \\ -1 & 1 & -2 & 2 \\ 2 & -2 & 1 & -1 \\ 1 & -1 & 2 & -2 \end{pmatrix}.$$

Матрицу $\hat{\mathbf{M}}$ можно представить в виде:

$$\mathbf{M} = \begin{pmatrix} \mathbf{D} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{D} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{D} \end{pmatrix},$$

где $\mathbf{O}$ - матрица 4×4, состоящая из нулей подматрица, а $\mathbf{D}$ определяется следующим образом:

$$\mathbf{D} = \begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix}.$$

Локальный вектор правой части определяется в виде:

$$\hat{\mathbf{b}}_i = \hat{\mathbf{M}} \cdot \hat{\mathbf{f}}_i, \tag{2.29}$$

где $\hat{\mathbf{f}}_i = \overrightarrow{\mathbf{F}}(\hat{x}_c, \hat{y}_c, \hat{z}_c) \cdot \overrightarrow{\mathbf{v}}/l$, для которого $\hat{x}_c, \hat{y}_c, \hat{z}_c$ – координаты центра ребра, $\overrightarrow{\mathbf{v}}$ – вектор, направленный вдоль ребра Γ и сонаправленный с базисной вектор-функцией $\overrightarrow{\Psi_i}$, l – длина вектора $\overrightarrow{\mathbf{F}}$.

Применяя трехслойную неявную схему аппроксимации по времени, получим итоговое СЛАУ для вариационного уравнения (2.28):

$$\left(\mathbf{G} + \tau_0 \mathbf{M}^{\sigma}\right) \mathbf{q} = \mathbf{M}^{\sigma - \sigma_n} \cdot \mathbf{E} + \tau_1 \mathbf{M}^{\sigma} \mathbf{q}^{\Leftarrow 1} - \tau_2 \mathbf{M}^{\sigma} \mathbf{q}^{\Leftarrow 2}. \tag{2.30}$$

3. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1. ФОРМАТ ВХОДНЫХ ДАННЫХ

Входные данные содержатся в папке "Data/Input/". Файл "WholeMesh.txt"содержит данные о трёхмерной сетке, из которой автоматически строится сетка для решения двумерной задачи на нормальном поле. В файле содержится информация о границах расчётной области по x, y, z, количество необходимых разбиений для каждой оси, коэффициенты разрядки, количество областей с разными значениями удельной электропроводности и информация о границах расчётной области. Полностью формат изображен на рисунке 3.1.

```
[x0] ... [xn]
[n_x_0] [k_x_0] ... [n_x_n-1] [k_x_n-1]
[y0] ... [ym]
[n_y_0] [k_y_0] ... [n_y_m-1] [k_y_m-1]
[z0] ... [zk]
[n_z_0] [k_z_0] ... [n_z_k-1] [k_z_k-1]
[r0] ... [rl]
[n_r_0] [k_r_0] ... [n_r_l-1] [k_r_l-1]
[radius_of_the_induction_current_source]
[areas_amount]
[area_num_i] [x0_i] [x1_i] [y0_i] [y1_i] [z0_i] [z1_i] [sigma_i]
[borders_amount]
[border_i_type] [formula_num] [x0_i] [x1_i] [y0_i] [y1_i] [z0_i] [z1_i]
```

Рисунок 3.1 – Входной формат сетки по пространству

Входные данные для учёта поля влияния хранятся в папке "Data/Input/Anomalies/". Каждый файл, находящийся в этой папке, содержит примерно похожий формат хранения, как и для основной сетки. Задаются границы по осям x, y, z, количество необходимых разбиений для каждой оси, коэффициенты разрядки, значения удельной электропроводности на аномальной области и границы этой области. Полностью формат изображен на рисунке 3.2.

```
[x0] ... [xn]
[n_x_0] [k_x_0] ... [n_x_n-1] [k_x_n-1]
[y0] ... [ym]
[n_y_0] [k_y_0] ... [n_y_m-1] [k_y_m-1]
[z0] ... [zk]
[n_z_0] [k_z_0] ... [n_z_k-1] [k_z_k-1]
[areas_amount]
[area_num_i] [x0_i] [x1_i] [y0_i] [y1_i] [z0_i] [z1_i] [sigma_i]
[anomaly_borders]
```

Рисунок 3.2 – Входной формат сетки по пространству для аномалии

Входные данные для сетки по времени, содержатся в файле "Time.txt в папке "Data/Input/"и содержат четыре значения: время начала и конца, количество разбиений и коэффициент разрядки.

3.2. СБОРКА ГЛОБАЛЬНОЙ МАТРИЦЫ И ГЛОБАЛЬНОГО ВЕКТОРА ПРАВОЙ ЧАСТИ

При формировании матрицы **A** для решения СЛАУ необходимо учитывать соответствие локальной к глобальной нумерации каждого узла. Глобальная нумерация узлов сетки однозначно определяет вклад локальной матрицы в соответствующие строчки и столбцы матрицы **A**. Поэтому, зная глобальную нумерацию узлов конечного элемента, можно определить какие элементы глобальной матрицы изменятся при добавлении в нее локальной. Аналогичным образом определяется вклад локального вектора правой части в глобальный.

3.3. УЧЁТ КРАЕВЫХ УСЛОВИЙ

Поскольку в решаемой задаче у нас на всех границах задаётся однородное краевое условие первого рода, технически необходимо в соответствующей строчке матрицы обнулить вне диагональные элементы, на диагонали поставить значение 1, а в соответствующую строчку вектора правой части поставить значение краевого условия на этой границе, т.е. в нашем случае тоже обнулить.

3.4. РЕШЕНИЕ СЛАУ

Для решения СЛАУ мы будем использовать локально-оптимальную схему с ILU-предобусловливанием [9]. Это хороший и быстрый метод решения систем уравнений для несимметричных матриц. Перед решением СЛАУ задаются параметры для досрочного выхода из итерационного процесса, а именно: выход по максимальному количеству совершённых итераций и минимальному значению нормы вектора невязки.

В результате решения СЛАУ мы получим вектор q весов базисных функций, на которые раскладывается функция A_{φ}^{0} или вектор-функция $\overrightarrow{\mathbf{A}}^{+}$. Учитывая построение базисных функций, компонентами этого вектора будут значения функции в соответствующих узлах сетки.

3.5. ОПРЕДЕЛЕНИЕ ЗНАЧЕНИЯ ВЕКТОР-ПОТЕНЦИАЛА И НАПРЯЖЁННОСТИ ЭЛЕКТРИЧЕСКОГО ПОЛЯ

После решения СЛАУ вида (2.23) или (2.30), необходимо найти напряжённость электрического поля по формуле (2.24). Пользуясь аналитическим представлением из (2.21), программная реализация будет выглядеть следующим образом.

Поскольку в качестве конечных элементов использовались прямоугольники для двумерной и прямые параллелепипеды для трёхмерной задач, то можно упростить алгоритм нахождения значения функции на элементе. Можно не перебирать каждый элемент отдельно и проверять значение интересующей точки на принадлежность ему, а последовательно сравнивать координаты точки со значениями на разбиениях по осям координат. Тогда сложность алгоритма будет не $O(n^2)$ для двумерной или $O(n^3)$ для трёхмерной задач, а $O(k \cdot n)$, где n – количество отрезков, на которые разбиваются оси координат.

3.6. ТЕСТИРОВАНИЕ ДВУМЕРНОЙ ЗАДАЧИ НА ПОЛИНОМАХ

Проведем сначала тестирование программы на работоспособность для уравнения (3.1). В таблицах 3.1 – 3.9 представлен результат тестирования на полиномиальных функциях. Образец расчетной области изображен на рисунке 3.3. Это область $\Omega = [1.0, 2.0]_r \times [1.0, 2.0]_z$, она содержит 16 узлов, а на всех границах будем задавать первые краевые условия.

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) - \frac{\partial^2 u}{\partial z^2} + \frac{u}{r^2} + \sigma\frac{\partial u}{\partial t} = f, \tag{3.1}$$

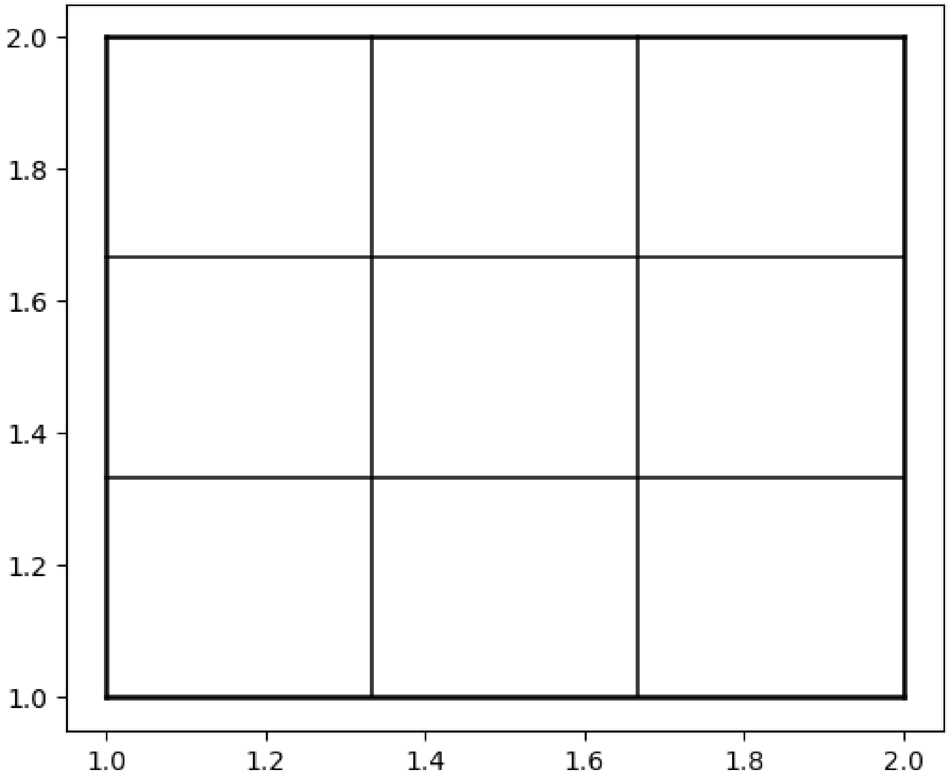

Рисунок 3.3 – Расчетная область

Таблица 3.1 – Тестирование при $u = 2$, $f = \frac{2}{r^2}$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$({}^4/_3; {}^4/_3)$	2.00226896E+000	2.26896083E-003	1.13448042E-003
$({}^5/_3; {}^4/_3)$	2.00130487E+000	1.30486533E-003	6.52432666E-004
$({}^4/_3; {}^5/_3)$	2.00226896E+000	2.26896083E-003	1.13448042E-003
$({}^5/_3; {}^5/_3)$	2.00130487E+000	1.30486533E-003	6.52432666E-004

Таблица 3.2 – Тестирование при $u = r$, $f = 0$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$({}^4/_3; {}^4/_3)$	1.33333333E+000	1.33226763E-015	9.99200722E-016
$({}^5/_3; {}^4/_3)$	1.66666667E+000	6.66133815E-016	3.99680289E-016
$({}^4/_3; {}^5/_3)$	1.33333333E+000	1.77635684E-015	1.33226763E-015
$({}^5/_3; {}^5/_3)$	1.66666667E+000	6.66133815E-016	3.99680289E-016

Таблица 3.3 – Тестирование при $u = z$, $f = \frac{z}{r^2}$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$(^4/_3;\, ^4/_3)$	1.33491362E+000	1.58028263E-003	1.18521198E-003
$(^5/_3;\, ^4/_3)$	1.33426439E+000	9.31054340E-004	6.98290755E-004
$(^4/_3;\, ^5/_3)$	1.66848983E+000	1.82315862E-003	1.09389517E-003
$(^5/_3;\, ^5/_3)$	1.66769291E+000	1.02624366E-003	6.15746195E-004

Таблица 3.4 – Тестирование при $u = r + z$, $f = \frac{z}{r^2}$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$(^4/_3;\, ^4/_3)$	2.66824695E+000	1.58028263E-003	5.92605988E-004
$(^5/_3;\, ^4/_3)$	3.00093105E+000	9.31054340E-004	3.10351447E-004
$(^4/_3;\, ^5/_3)$	3.00182316E+000	1.82315862E-003	6.07719539E-004
$(^5/_3;\, ^5/_3)$	3.33435958E+000	1.02624366E-003	3.07873097E-004

Таблица 3.5 – Тестирование при $u = rz$, $f = 0$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$(^4/_3;\, ^4/_3)$	1.77777778E+000	1.11022302E-015	6.24500451E-016
$(^5/_3;\, ^4/_3)$	2.22222222E+000	3.10862447E-015	1.39888101E-015
$(^4/_3;\, ^5/_3)$	2.22222222E+000	8.88178420E-016	3.99680289E-016
$(^5/_3;\, ^5/_3)$	2.77777778E+000	4.88498131E-015	1.75859327E-015

Таблица 3.6 – Тестирование при $u = r^2 + z^2$, $f = \frac{z^2}{r^2} - 5$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$(^4/_3;\, ^4/_3)$	3.55717205E+000	1.61649660E-003	4.54639669E-004
$(^5/_3;\, ^4/_3)$	4.55644336E+000	8.87803368E-004	1.94883666E-004
$(^4/_3;\, ^5/_3)$	4.55790068E+000	2.34512455E-003	5.14783438E-004
$(^5/_3;\, ^5/_3)$	5.55672893E+000	1.17337132E-003	2.11206838E-004

Таблица 3.7 – Тестирование при $u = r^2z^2$, $f = -3z^2 - 2r^2$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$({}^4/_3;\ {}^4/_3)$	3.15919390E+000	1.29993140E-003	4.11306418E-004
$({}^5/_3;\ {}^4/_3)$	4.93728492E+000	9.86688136E-004	1.99804348E-004
$({}^4/_3;\ {}^5/_3)$	4.93658403E+000	1.68757555E-003	3.41734049E-004
$({}^5/_3;\ {}^5/_3)$	7.71481536E+000	1.23402231E-003	1.59929291E-004

Таблица 3.8 – Тестирование при $u = r^3 + z^3$, $f = -8r - 6z + \frac{z^3}{r^2}$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$({}^4/_3;\ {}^4/_3)$	4.73874864E+000	1.99210278E-003	4.20209180E-004
$({}^5/_3;\ {}^4/_3)$	6.99757994E+000	2.42006018E-003	3.45722883E-004
$({}^4/_3;\ {}^5/_3)$	6.99968104E+000	3.18957115E-004	4.55653022E-005
$({}^5/_3;\ {}^5/_3)$	9.25749495E+000	1.76431155E-003	1.90545647E-004

Таблица 3.9 – Тестирование при $u = r^3z^3$, $f = -8rz^3 - 6r^3z$, $\sigma = 0$

Узел	Значение	Абсолютная погрешность	Относительная погрешность
$({}^4/_3;\ {}^4/_3)$	5.60268110E+000	1.59745896E-002	2.84313374E-003
$({}^5/_3;\ {}^4/_3)$	1.09603120E+001	1.36249123E-002	1.24157014E-003
$({}^4/_3;\ {}^5/_3)$	1.09509327E+001	2.30042146E-002	2.09625906E-003
$({}^5/_3;\ {}^5/_3)$	2.14142095E+001	1.92609735E-002	8.98639979E-004

Исходя из полученных данных, можно сказать, что программа верно находит численное решение задачи.

Рассмотрим решение функции $u = e^{r \cdot z}$, последовательно разбивая сетку в 2 раза. Результаты тестирования приведены в таблице 3.10.

Таблица 3.10 – Тестирование при $u = e^{r \cdot z}$, $\sigma = 0$

Количество разбиений	Средняя погрешность	$\log_2\left(\frac{\sigma_{i-1}}{\sigma_i}\right)$
2	1.7116567E-004	-
4	5.2066366E-005	1.716969754
8	1.4089198E-005	1.885762274
16	3.6602112E-006	1.94459064
32	9.3301457E-007	1.971955381

Порядок сходимости стремится к 2.

Теперь проведём тестирование уравнения (3.1) на порядок аппроксимации и сходимости для аппроксимации по времени. Для чистоты исследования мы не будем учитывать слагаемое с ${}^1/_{r^2}$. Сетка по времени равномерная $t \in [0, 1]$, $h_t = 0.2$. При тестировании на порядок сходимости будем рассматривать функцию $u = e^t$ и $f = e^t$. Результат тестирования представлен в таблице 3.11.

Таблица 3.11 – Тестирование при $u = e^t$, $f = e^t$

Количество разбиений	Средняя погрешность	$\log_2\left(\frac{\sigma_{i-1}}{\sigma_i}\right)$
4	8.4940866E-003	-
8	2.4848144E-003	1.77332095
16	6.6533732E-004	1.90098023
32	1.7191414E-004	1.95239775

Порядок сходимости стремится к 2.

Поскольку мы использовали трёхслойную неявную схему, то для тестирования на порядок аппроксимации рассмотрим функцию $u = t^2$, и $f = 2 \cdot t$. Результат тестирования представлен в таблице 3.12.

Таблица 3.12 – Тестирование при $u = t^2$, $f = 2 \cdot t$, $\sigma = 1$

Временной слой	Абсолютная погрешность	Относительная погрешность
0.0	0.0000000E+000	-
	0.0000000E+000	-
	0.0000000E+000	-
	0.0000000E+000	-
0.2	0.0000000E+000	0.0000000E+000
	0.0000000E+000	0.0000000E+000
	0.0000000E+000	0.0000000E+000
	0.0000000E+000	0.0000000E+000
0.4	8.3266727E-017	5.2041704E-016
	2.7755576E-017	1.7347235E-016
	2.7755576E-017	1.7347235E-016
	8.3266727E-017	5.2041704E-016
0.6	5.5511151E-017	1.5419764E-016
	5.5511151E-017	1.5419764E-016
	5.5511151E-017	1.5419764E-016
	5.5511151E-017	1.5419764E-016
0.8	5.5511151E-016	8.6736174E-016
	0.0000000E+000	0.0000000E+000
	1.1102230E-016	1.7347235E-016
	2.2204460E-016	3.4694470E-016
1.0	4.4408921E-016	4.4408921E-016
	0.0000000E+000	0.0000000E+000
	0.0000000E+000	0.0000000E+000
	4.4408921E-016	4.4408921E-016

Как и предполагалось, квадратичная функция по времени находится без численной погрешности.

3.7. ТЕСТИРОВАНИЕ ТРЁХМЕРНОЙ ЗАДАЧИ НА ПОЛИНОМИАЛЬНЫХ ВЕКТОР-ФУНКЦИЯХ

Проведем сначала тестирование разработанной программы по векторному МКЭ на работоспособность. Образец расчетной области изображен на рисунке 3.4. Это область $\Omega = [0.0, 3.0]_x \times [0.0, 3.0]_y \times [0.0, 3.0]_z$, она содержит 144 ребра, на всех границах будем задавать первые краевые условия.

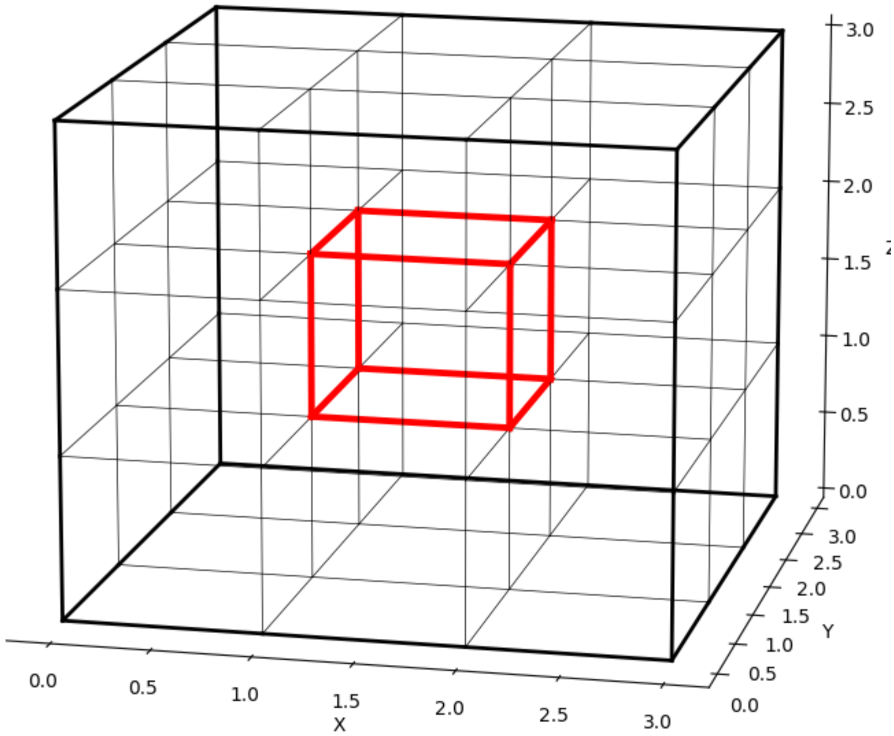

Рисунок 3.4 – Расчетная область

Тестирование будем проводить дифференциального уравнения (3.2):

$$\operatorname{rot}\left(\frac{1}{\mu}\operatorname{rot}\overrightarrow{\mathbf{A}}\right) + \gamma\overrightarrow{\mathbf{A}} + \sigma\frac{\partial\overrightarrow{\mathbf{A}}}{\partial t} = \overrightarrow{\mathbf{F}}. \tag{3.2}$$

В таблицах 3.13 – 3.21 приведено тестирование на работоспособность программы. Для искомых $\overrightarrow{\mathbf{A}}$ будем выводить значения функции в центрах рёбер сетки, отмеченных красным цветом на рисунке 3.4.

Таблица 3.13 – Тестирование при $\overrightarrow{\mathbf{A}} = (1.0, 1.0, 1.0)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (1.0, 1.0, 1.0)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.14 – Тестирование при $\overrightarrow{\mathbf{A}} = (y, z, x)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (y, z, x)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	2.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.15 – Тестирование при $\overrightarrow{\mathbf{A}} = (1 + y + x; 1 + x + z; 1 + x + y)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (1 + y + x; 1 + x + z; 1 + x + y)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	3.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	3.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	3.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	5.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.16 – Тестирование при $\overrightarrow{\mathbf{A}} = (y - z; x - z; x - y)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (y - x; x - z; x - y)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	2.35132600E-016	2.35132600E-016	0.00000000E+000
$(x; 2.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	-1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	-5.55111512E-016	-5.55111512E-016	0.00000000E+000
$(1.0; y; 1.0)$	-3.97378607E-016	-3.97378607E-016	0.00000000E+000
$(2.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	-1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	-1.94289029E-016	-1.94289029E-016	0.00000000E+000
$(1.0; 1.0; z)$	-2.74847895E-016	-2.74847895E-016	0.00000000E+000
$(2.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	-1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	4.27842044E-016	4.27842044E-016	0.00000000E+000

Таблица 3.17 – Тестирование при $\overrightarrow{\mathbf{A}} = (y \cdot z; x \cdot z; x \cdot y)^{\mathrm{T}}$,

$\overrightarrow{\mathbf{F}} = (y \cdot z; x \cdot z; x \cdot y)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.18 – Тестирование при $\overrightarrow{\mathbf{A}} = (y^2; z^2; x^2)^{\mathrm{T}}$,

$\overrightarrow{\mathbf{F}} = (y^2 - 2; z^2 - 2; x^2 - 2)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.19 – Тестирование при $\overrightarrow{\mathbf{A}} = (y^2 + z^2; x^2 + z^2; x^2 + y^2)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (y^2 + z^2 - 4; x^2 + z^2 - 4; x^2 + y^2 - 4)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	8.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	8.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	2.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	5.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	8.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.20 – Тестирование при $\overrightarrow{\mathbf{A}} = (y^3; 0; 0)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (y^3 - 6y; 0; 0)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	8.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	8.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	0.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	0.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.21 – Тестирование при $\overrightarrow{\mathbf{A}} = (y^2 \cdot z^2; x^2 \cdot z^2; x^2 \cdot y^2)^{\mathrm{T}}$,
$\overrightarrow{\mathbf{F}} = (y^2 \cdot z^2 - 2(y^2 + z^2); x^2 \cdot z^2 - 2(x^2 + z^2); x^2 \cdot y^2 - 2(x^2 + y^2))^{\mathrm{T}}$,
$\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(x; 1.0; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 1.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 1.0; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(x; 2.0; 2.0)$	1.60000000E+001	0.00000000E+000	0.00000000E+000
$(1.0; y; 1.0)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 1.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; y; 2.0)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; y; 2.0)$	1.60000000E+001	0.00000000E+000	0.00000000E+000
$(1.0; 1.0; z)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 1.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.0; 2.0; z)$	4.00000000E+000	0.00000000E+000	0.00000000E+000
$(2.0; 2.0; z)$	1.60000000E+001	0.00000000E+000	0.00000000E+000

Проведём тестирование на порядок аппроксимации. Для оценки будем брать значения вектор-функции в центрах параллелепипедов. Сетка по пространству для данных тестов изображена на рисунке 3.5.

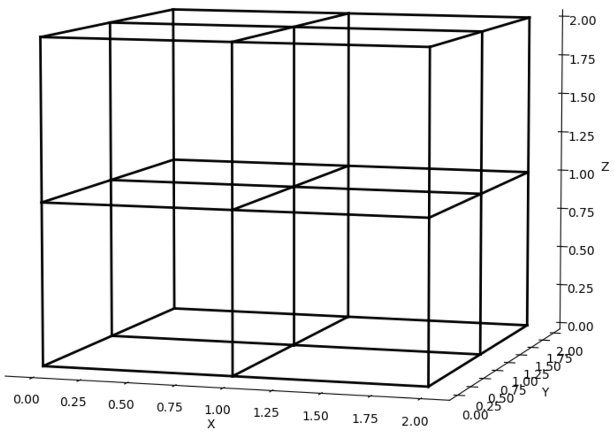

Рисунок 3.5 – Расчетная область

В таблицах 3.22 – 3.23 представлены результаты тестирования для постоянной и линейной вектор-функциях.

Таблица 3.22 – Тестирование при $\overrightarrow{\mathbf{A}} = (1.0; 1.0; 1.0)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (1.0; 1.0; 1.0)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(0.5; 0.5; 0.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.5; 0.5; 0.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(0.5; 1.5; 0.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.5; 1.5; 0.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(0.5; 0.5; 1.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.5; 0.5; 1.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(0.5; 1.5; 1.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
$(1.5; 1.5; 1.5)$	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000
	1.00000000E+000	0.00000000E+000	0.00000000E+000

Таблица 3.23 – Тестирование при $\overrightarrow{\mathbf{A}} = (y; z; x)^{\text{T}}$, $\overrightarrow{\mathbf{F}} = (y; z; x)^{\text{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Ребро	Значение	Абсолютная погрешность	Относительная погрешность
$(0.5; 0.5; 0.5)$	5.00000000E-001	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
$(1.5; 0.5; 0.5)$	5.00000000E-001	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000
$(0.5; 1.5; 0.5)$	1.50000000E+000	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
$(1.5; 1.5; 0.5)$	1.50000000E+000	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000
$(0.5; 0.5; 1.5)$	5.00000000E-001	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
$(1.5; 0.5; 1.5)$	5.00000000E-001	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
$(0.5; 1.5; 1.5)$	1.50000000E+000	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000
	5.00000000E-001	0.00000000E+000	0.00000000E+000
$(1.5; 1.5; 1.5)$	1.50000000E+000	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000
	1.50000000E+000	0.00000000E+000	0.00000000E+000

Как и предполагали, при использовании билинейных вектор-функций точное решение находится вплоть до линейной вектор-функции без численной погрешности.

Проведём теперь тестирование на порядок сходимости на сетке изображённой на рисунке 3.4. Для этого последовательно будем разбивать сетку в 2 раза сначала по оси x, потом по y и затем по z. Результаты тестирования приведены в таблицах 3.24 – 3.26.

Таблица 3.24 – Тестирование при $\overrightarrow{\mathbf{A}} = (0; 0; e^x)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (0; 0; 0)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Шаг по оси x	Средняя погрешность	$\log_2\left(\frac{\sigma_{i-1}}{\sigma_i}\right)$
h	4.1223218E-001	-
$^h/_2$	6.9015889E-002	2.57845668
$^h/_4$	1.4360912E-002	2.26478117
$^h/_8$	3.28952607E-003	2.1261957

Таблица 3.25 – Тестирование при $\overrightarrow{\mathbf{A}} = (e^y; 0; 0)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (0; 0; 0)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Шаг по оси y	Средняя погрешность	$\log_2\left(\frac{\sigma_{i-1}}{\sigma_i}\right)$
h	4.1223218E-001	-
$^h/_2$	6.9015889E-002	2.57845668
$^h/_4$	1.4360912E-002	2.26478117
$^h/_8$	3.28952607E-003	2.1261957

Таблица 3.26 – Тестирование при $\overrightarrow{\mathbf{A}} = (0; e^z; 0)^{\mathrm{T}}$, $\overrightarrow{\mathbf{F}} = (0; 0; 0)^{\mathrm{T}}$, $\mu = 1$, $\gamma = 1$, $\sigma = 0$

Шаг по оси z	Средняя погрешность	$\log_2\left(\frac{\sigma_{i-1}}{\sigma_i}\right)$
h	4.1223218E-001	-
$^h/_2$	6.9015889E-002	2.57845668
$^h/_4$	1.4360912E-002	2.26478117
$^h/_8$	3.28952607E-003	2.1261957

Во всех трёх случая порядок сходимости стремится к 2. Исходя из полученных данных, можно сказать, что программа верно находит численное решение эллиптической задачи.

3.8. ПРОВЕРКА ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Проверку полученных результатов решения СЛАУ будем из закона индукции Фарадея (1.2) и теоремы о циркуляции магнитного поля (1.1). Учитывая (2.23) – (2.25), получим выражение для $\overrightarrow{\mathbf{B}}$, которое будем использовать для проверки:

$$\overrightarrow{\mathbf{B}} = \operatorname{rot}\overrightarrow{\mathbf{A}} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ A_x & A_y & 0 \end{vmatrix} = -\frac{\partial A_y}{\partial z}\mathbf{i} + \frac{\partial A_x}{\partial z}\mathbf{j} + \left(\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}\right)\mathbf{k}. \tag{3.3}$$

Исходя из теории конечно-разностных схем [10], численно определим значения для частных первых производных в выражении (3.3):

$$\frac{\partial A_{x_i}}{\partial x_k} + o\left(h_{x_k}^3\right) = \frac{A_{x_i}^{j+1} - A_{x_i}^{j-1}}{2h_{x_k}}, \tag{3.4}$$

где x_k – переменная по которой проводится дифференцирование, x_i – соответствующая компонента вектора-потенциала $\overrightarrow{\mathbf{A}}$, $A_{x_i}^{j+1} = A_{x_i}\left(x_{0i} + h_{x_k}\right)$, $A_{x_i}^{j-1} = A_{x_i}\left(x_{0i} - h_{x_k}\right)$, h_{x_k} – шаг от точки, в которой необходимо найти значение производной, равный 10^{-10}.

4. ИССЛЕДОВАНИЯ

4.1. ОПИСАНИЕ ИССЛЕДОВАНИЙ И РАСПОЛОЖЕНИЕ ПРИЁМНИКОВ

Для начала будем проводить исследования в горизонтально-слоистой среде без каких-либо аномалий. Напомню, что на рисунке 1.1 изображен срез горизонтально-слоистой среды, который мы используем в качестве решения нормального поля на первом этапе разделения полей. Далее рассмотрим горизонтально-слоистую среду с двумя аномалиями вместе, т.е для решения задачи на добавочное поле будем использовать сетку, учитывающую сразу две аномалии. После этого последовательно рассмотрим добавление в среду сначала первой аномалии потом второй, а затем сначала второй и после первой. На заключительном этапе рассмотрим временные затраты на решение задачи при использовании многоэтапной схемы разделения поля и при разбиении поля на нормальное и добавочное.

Пусть источник индукционного поля имеет радиус $R = 100$ м. от оси симметрии и имееет силу тока, равную $J_\varphi = 1.0$ А. Также условимся, что источник работал достаточно долго, чтобы создать стабильное электромагнитное поле. Сетка по времени: $t \in [0.0; 1.0]$ на 100 временных слоёв с начальным шагом $h_t = 10^{-5}$ с. и коэффициентом разрядки $t_k = 1.1$. После отключим наш источник, т.е. $J_\varphi = 0.0$ А при $t > 0.0$. Возьмём 4 приемника и расположим их вдоль линии $x = 0$ между аномалиями. Пусть они будут располагаться на расстояниях 101 м., 1000 м., 2000 м., 3000 м. от центра симметрии. На рисунке 4.1 тёмно-бирюзовой кривой нарисован индукционный источник тока, контурными линиями нарисованы положение аномальных объектов в среде, точками – расположение приёмников.

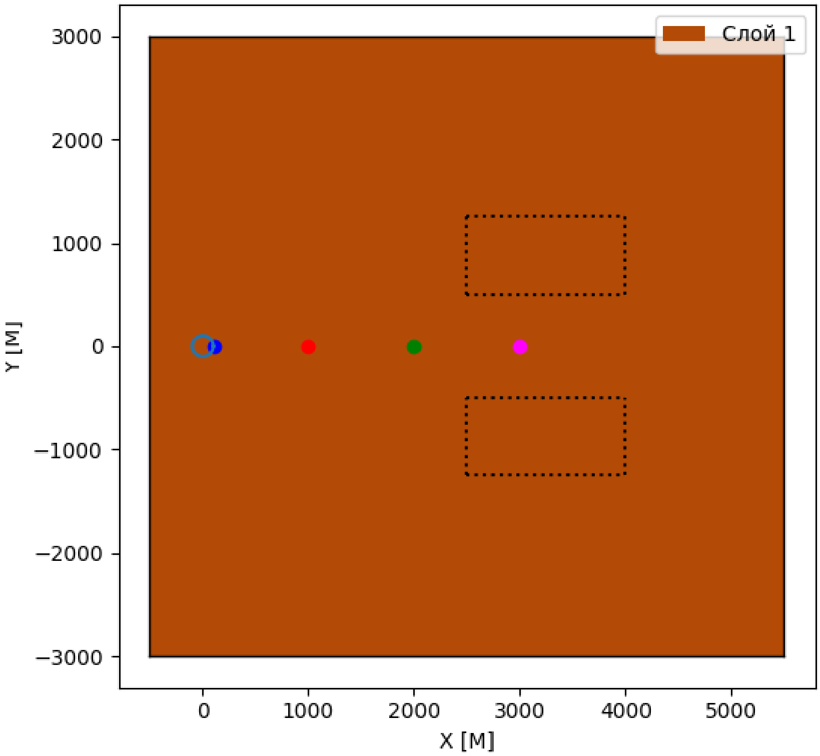

Рисунок 4.1 – Расположение приёмников

4.2. ИССЛЕДОВАНИЕ НОРМАЛЬНОГО ПОЛЯ БЕЗ АНОМАЛИЙ

На рисунках 4.2 – 4.5 представлено распространение этого поля в среде в начальный, промежуточные и последний моменты времени. Также, проведём замеры значения напряжённости электрического поля в них. Полученные зависимости E^0 от времени изображены на рисунке 4.6.

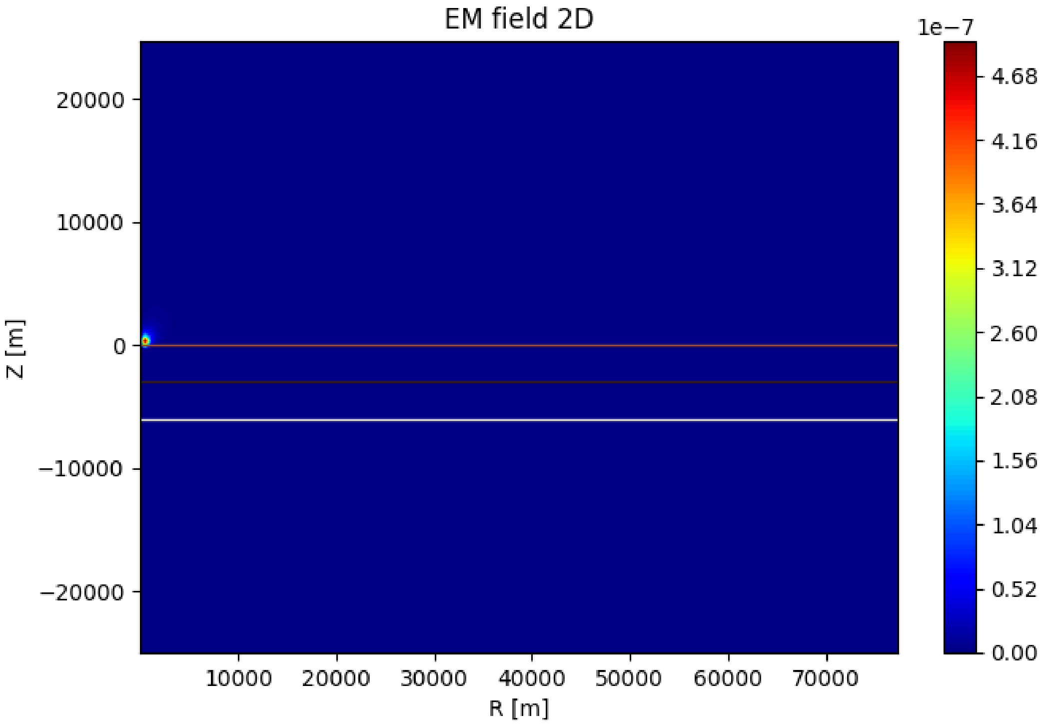

Рисунок 4.2 – Решение E_φ при $t = 10^{-5}$с

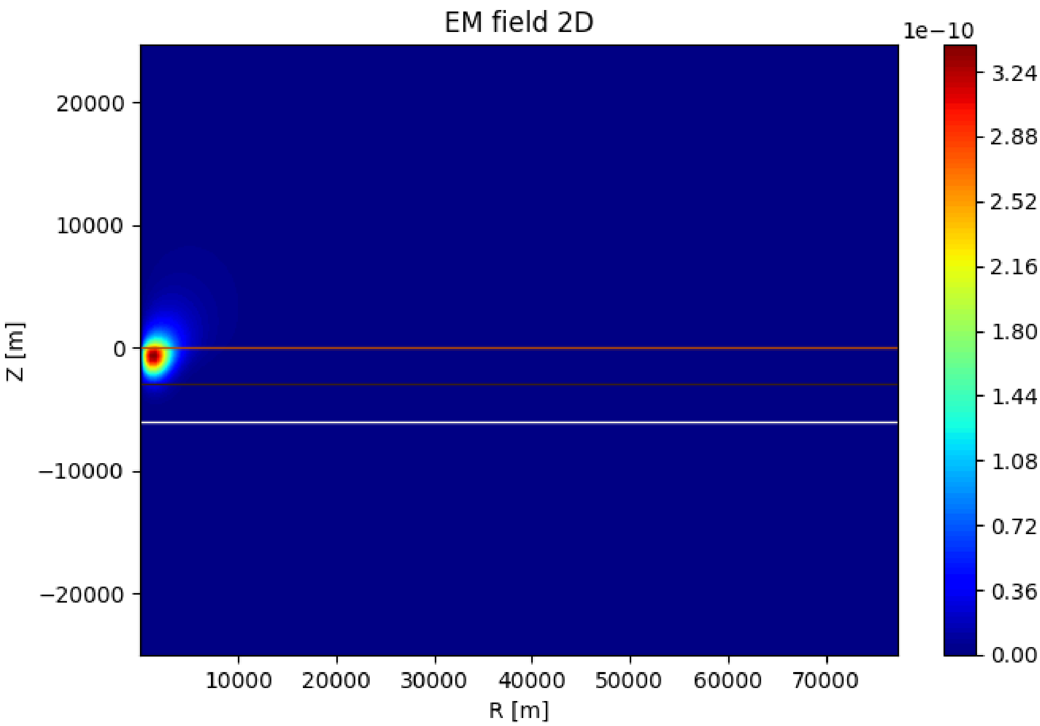

Рисунок 4.3 – Решение E_φ при $t = 0.01$с

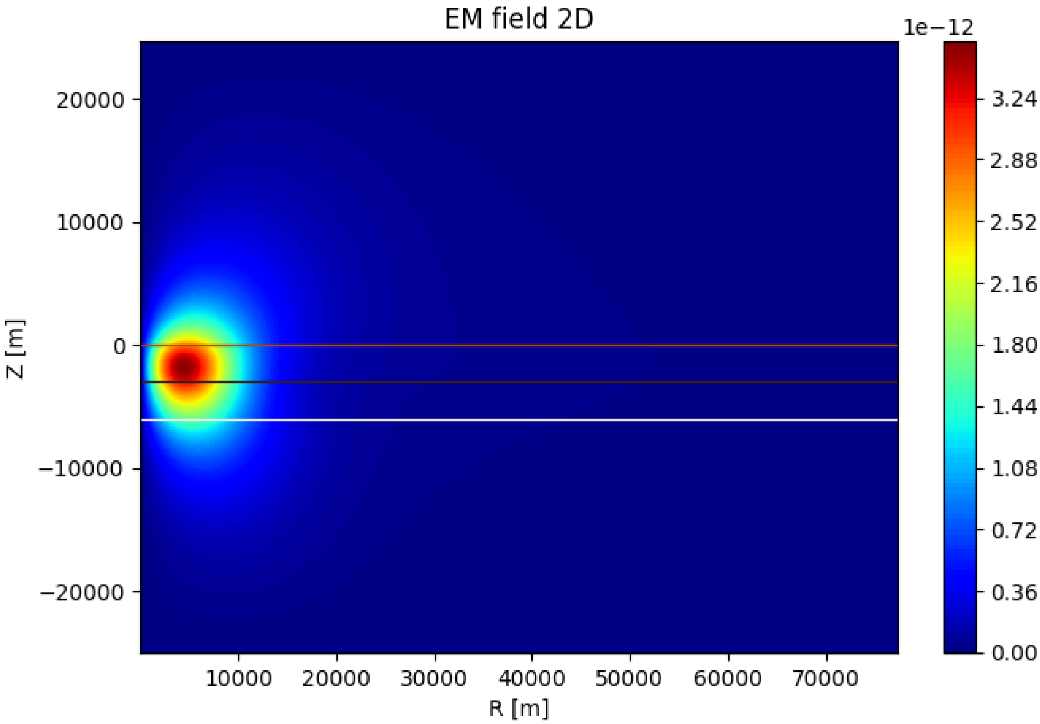

Рисунок 4.4 – Решение E_φ при $t = 0.1$с

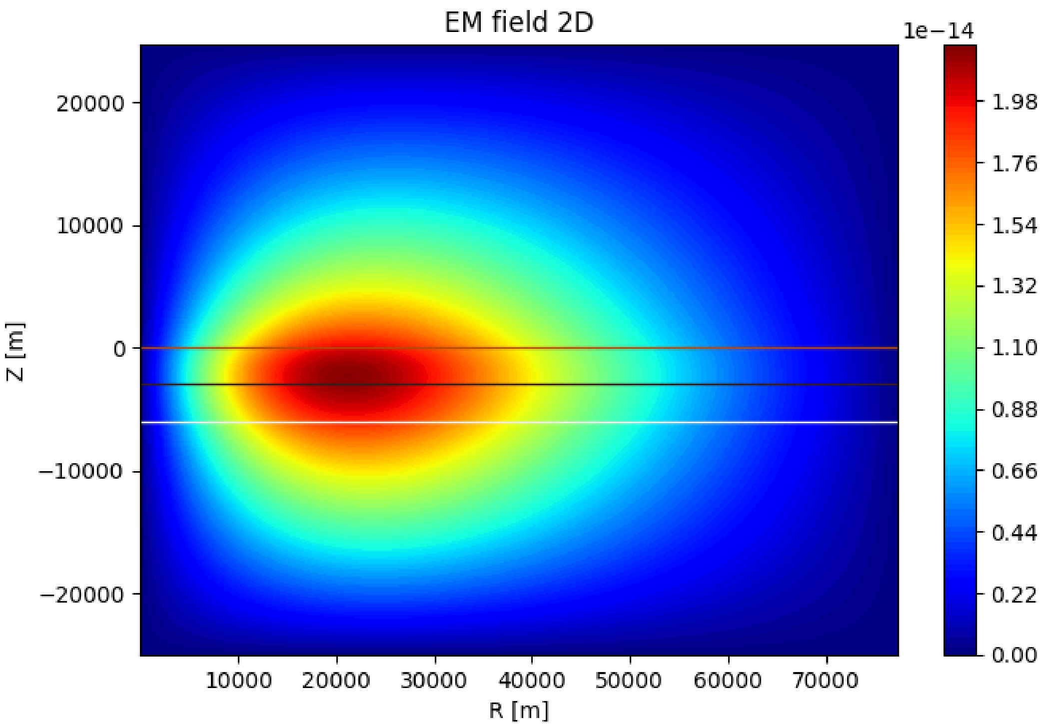

Рисунок 4.5 – Решение E_φ при $t = 1$с

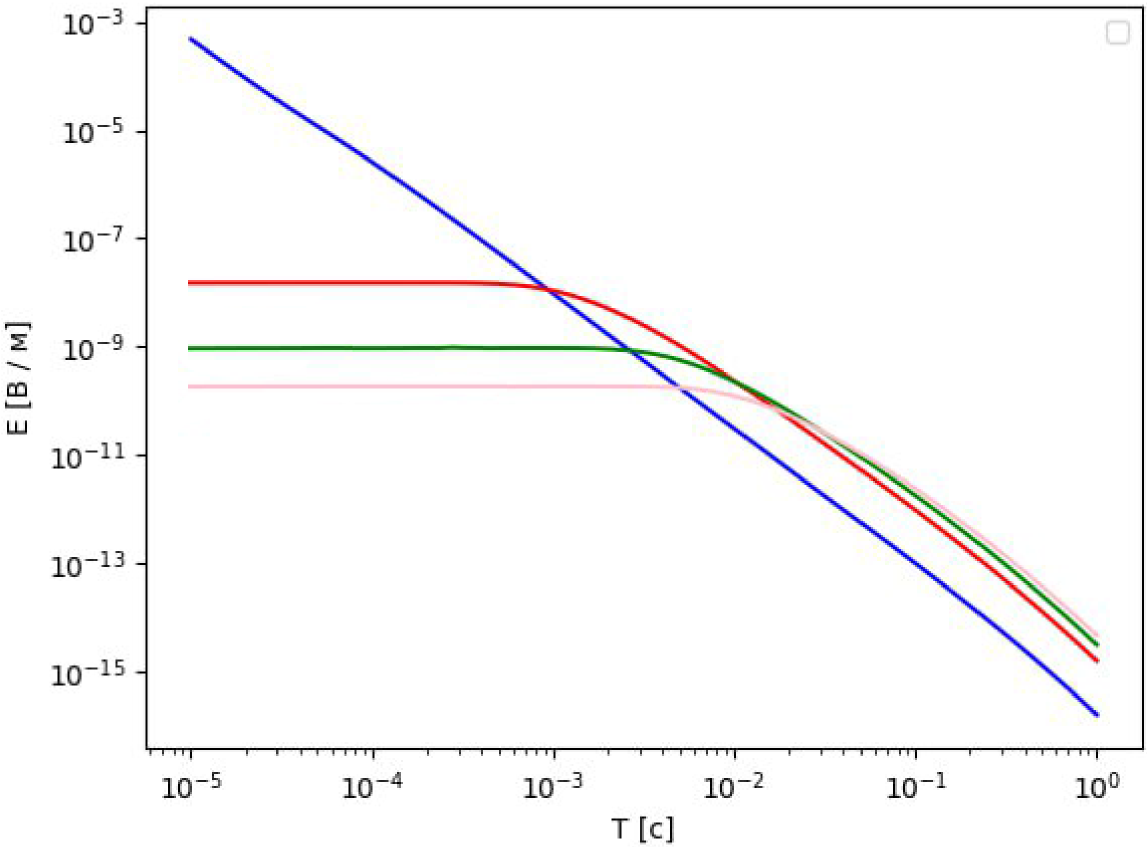

Рисунок 4.6 – Зависимость значения модуля E^0 от времени в разных приёмниках

Как видим, значения электрической напряжённости поля на приёмниках не имеют каких-либо резких колебаний и постепенно снижаются. Из этого можно заключить, что, как и предполагалось, никаких аномальных зон в исследуемой области нет.

4.3. ИССЛЕДОВАНИЕ В СРЕДЕ С ДВУМЯ АНОМАЛИЯМИ

Теперь расположим в расчётной области два аномальных объекта. Пусть первый объект имеет удельную электропроводность $\sigma_1 = 5.0$ См/м и находится в $\Omega_1 \in [2500, 4000]_x \times [500, 1250]_y \times [-2000, -750]_z$, а второй $\sigma_2 = 10.0$

См/м и находится в $\Omega_2 \in [2500, 4000]_x \times [-1250, -500]_y \times [-2000, -750]_z$. Их расположение в сечениях изображено на рисунках 4.7 – 4.8.

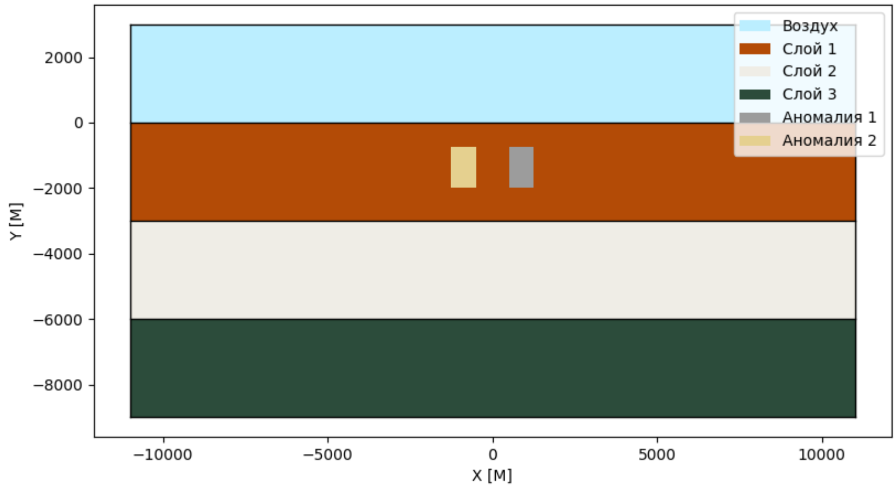

Рисунок 4.7 – Срез среды при $x = 2500$м

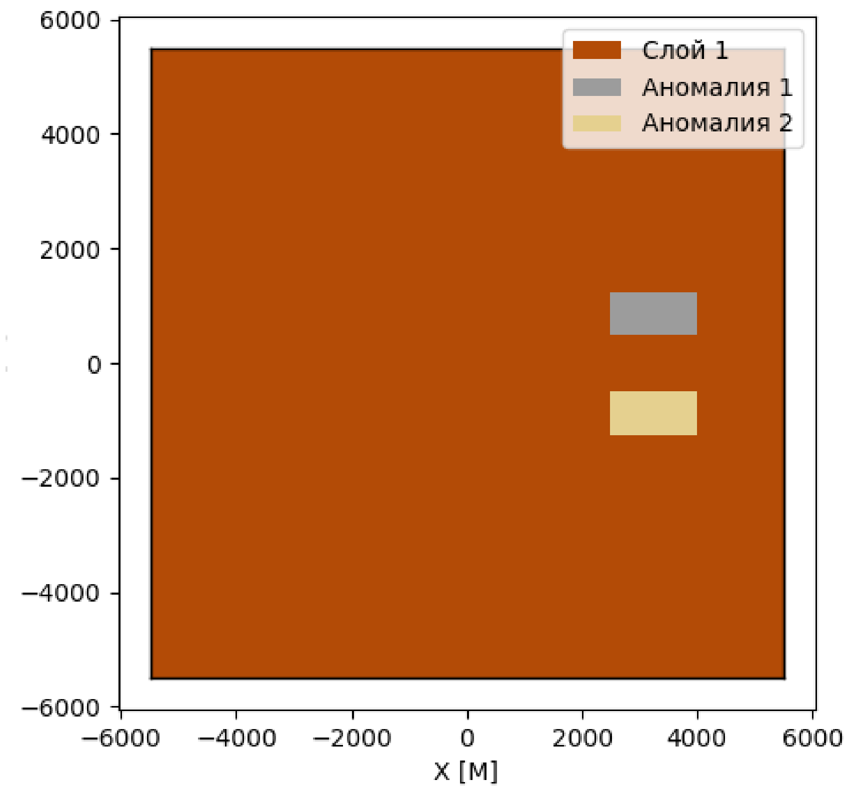

Рисунок 4.8 – Срез среды при $z = -1000$м

Тогда графики зависимости значения напряжённости электрического поля E от времени на выбранных нами приёмниках представлены на рисунке 4.9. Процесс рассматривался с 0.002 секунды.

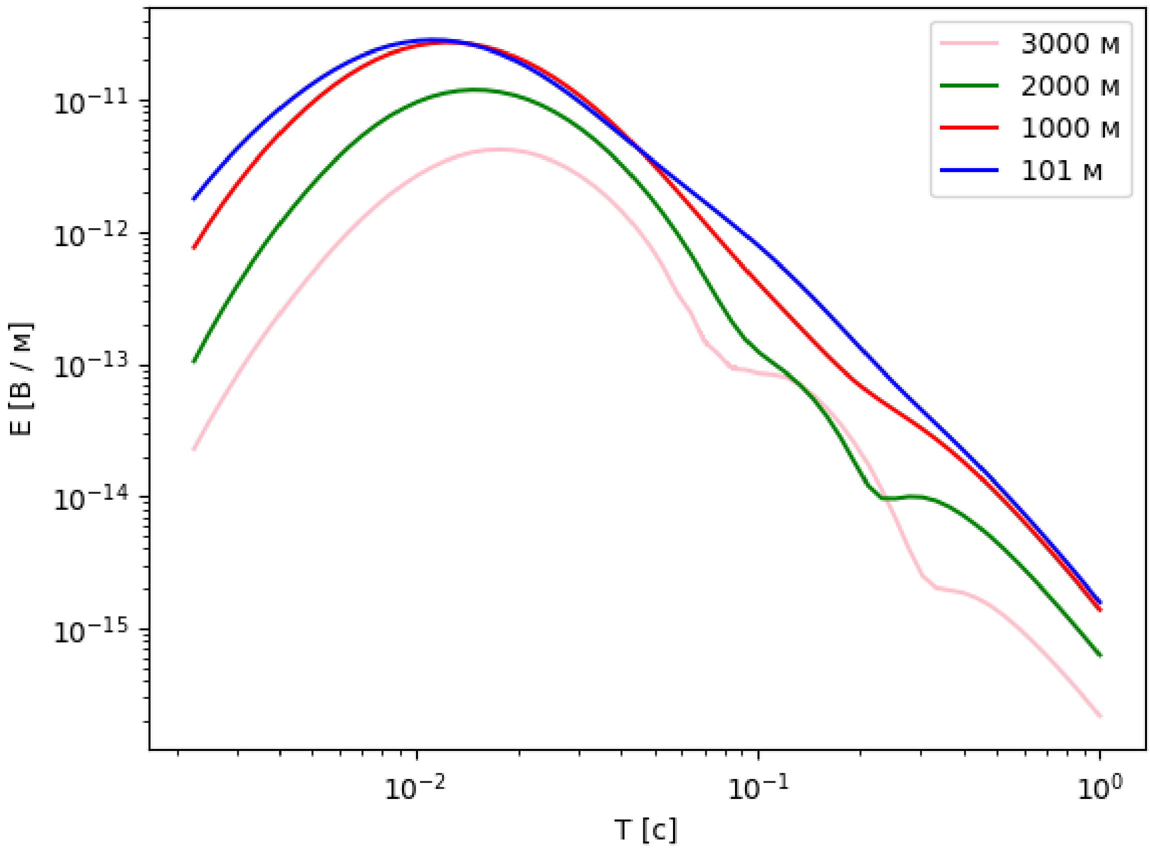

Рисунок 4.9 – Зависимость значения модуля E^{+} от времени в приёмниках при одновременном разделении аномалий

Как можно заметить, со временем, значение модуля напряжённости электрического поля добавочных объектов сначала увеличивается, а затем уменьшается. Объяснить это можно тем, что рассеиваемое поле горизонтально-слоистой среды со временем начинает оказывать влияние на аномальный по электропроводности объект. Из-за этого модуль значения напряжённости электромагнитного поля сначала растёт, а затем плавно также рассеивается вместе с полем горизонтально-слоистой среды.

4.4. ИССЛЕДОВАНИЕ МНОГОЭТАПНОГО РАЗДЕЛЕНИЯ ПОЛЕЙ

Теперь рассмотрим постепенное добавление аномальных зон в расчётную область. Пусть на первом этапе, в качестве добавочного поля будем считать поле, создаваемое объектом с удельной электропроводностью $\sigma = 5.0$ См/м. На втором этапе в качестве нормального поля будем рассматривать поле, полученное на предыдущем этапе, а в качестве добавочного, поле создаваемое объектом с электропроводностью $\sigma = 10.0$ См/м. Графики изменения значения напряжённости электрического поля от времени после добавления первой аномалии, а затем второй изображены на рисунке 4.10.

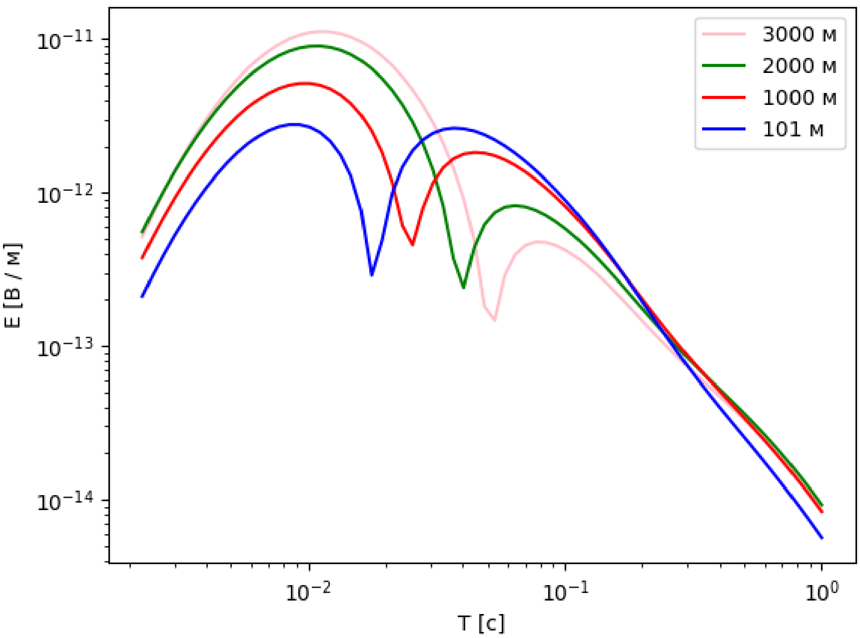

Рисунок 4.10 – Зависимость значения модуля E^+ от времени в разных приёмниках при разделении сначала первой, а затем второй аномалии

По началу значения напряжённости поля повышаются. Резкий скачок вниз можно объяснить инверсией электромагнитного поля – направление распространения может измениться на противоположное по знаку значение.

Теперь рассмотрим постепенное добавление аномальных зон в расчётную область наоборот. На первом этапе, в качестве добавочного поля будем считать поле, создаваемое объектом с удельной электропроводностью $\sigma = 10.0$ См/м. На втором этапе в качестве нормального поля будем рассматривать поле, полученное на предыдущем этапе, а в качестве добавочного, поле создаваемое объектом с электропроводностью $\sigma = 5.0$ См/м. График изменения значения напряжённости электрического поля от времени после добавления первой аномалии, а затем второй изображён на рисунке 4.11.

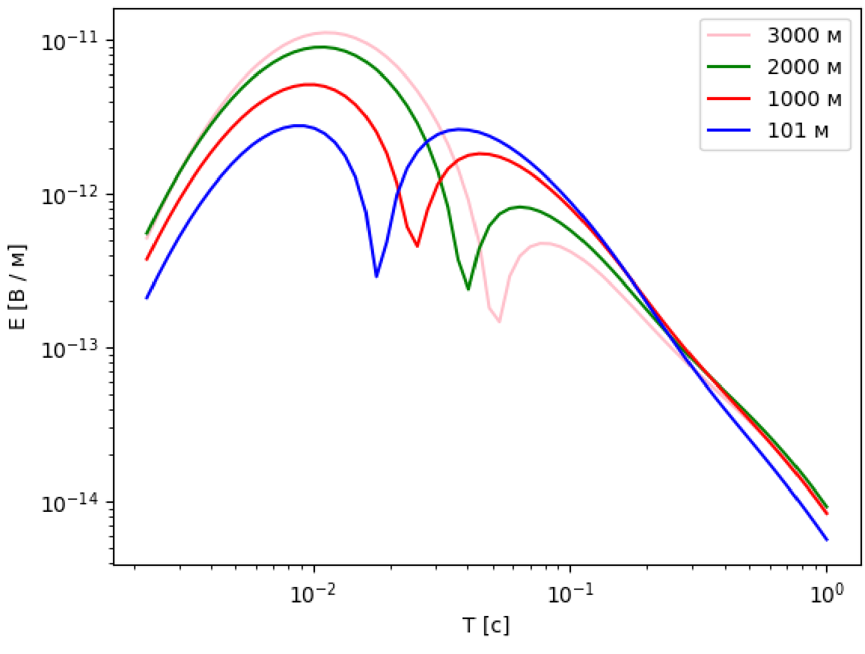

Рисунок 4.11 – Зависимость значения модуля E^{+} от времени в разных приёмниках при разделении сначала второй, а затем первой аномалии

На рисунке 4.12 изображено сравнение результатов первого и второго подхода к многоэтапному разделению полей. Цветными линиями показано значение напряжённости электрического поля при разделении сначала второй, а затем первой, чёрным пунктиром значение напряжённости при разделении сначала первой, а затем второй аномалий.

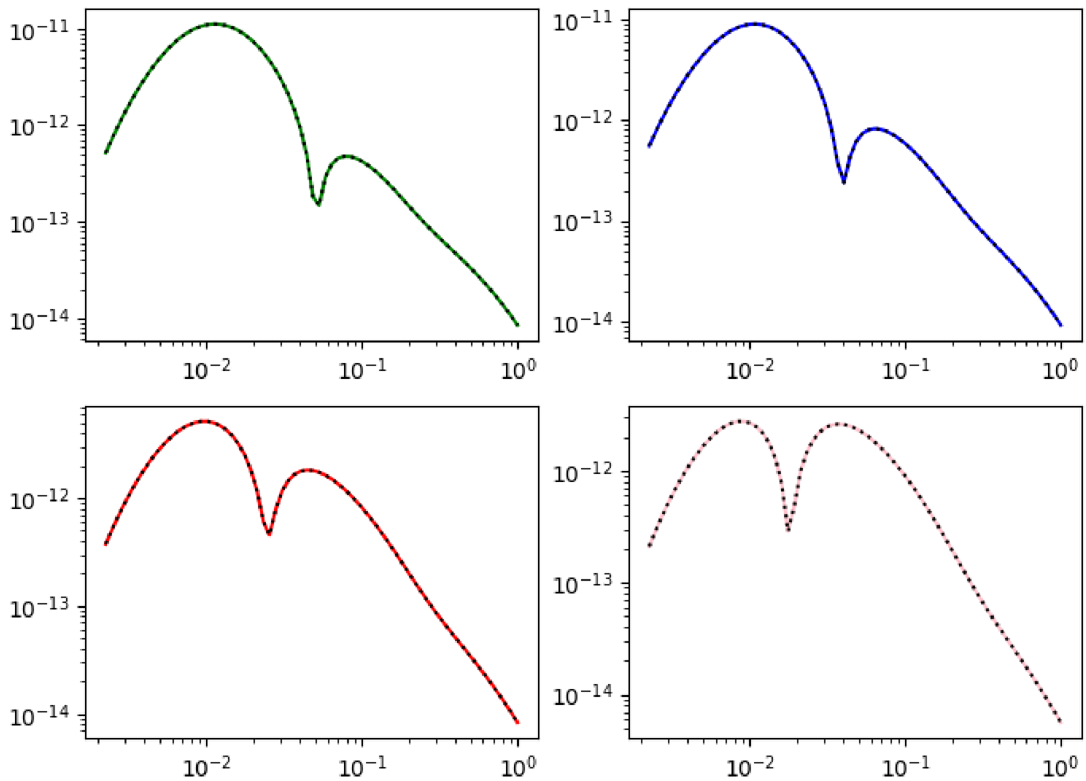

Рисунок 4.12 – Сравнение значений модуля E^{+} от времени на разных приёмниках при разной очерёдности разделении аномалий

Как выяснилось, очерёдность разделения полей не дала видимых различий, следовательно результат не зависит от очерёдности добавления аномалий в расчётную область.

Поскольку очерёдность добавления аномальных объектов в расчётную область не даёт очевидной разницы, то рассмотрим на рисунке 4.13 сравнение значений E^{+} при одновременном разделении сразу двух объектов и при использовании многоэтапной схемы, сначала выделяя первый, а затем второй объект. Цветными линиями показаны значения при использовании многоэтапной схемы разделения полей, пунктиром при одновременном разделении сразу двух аномалий.

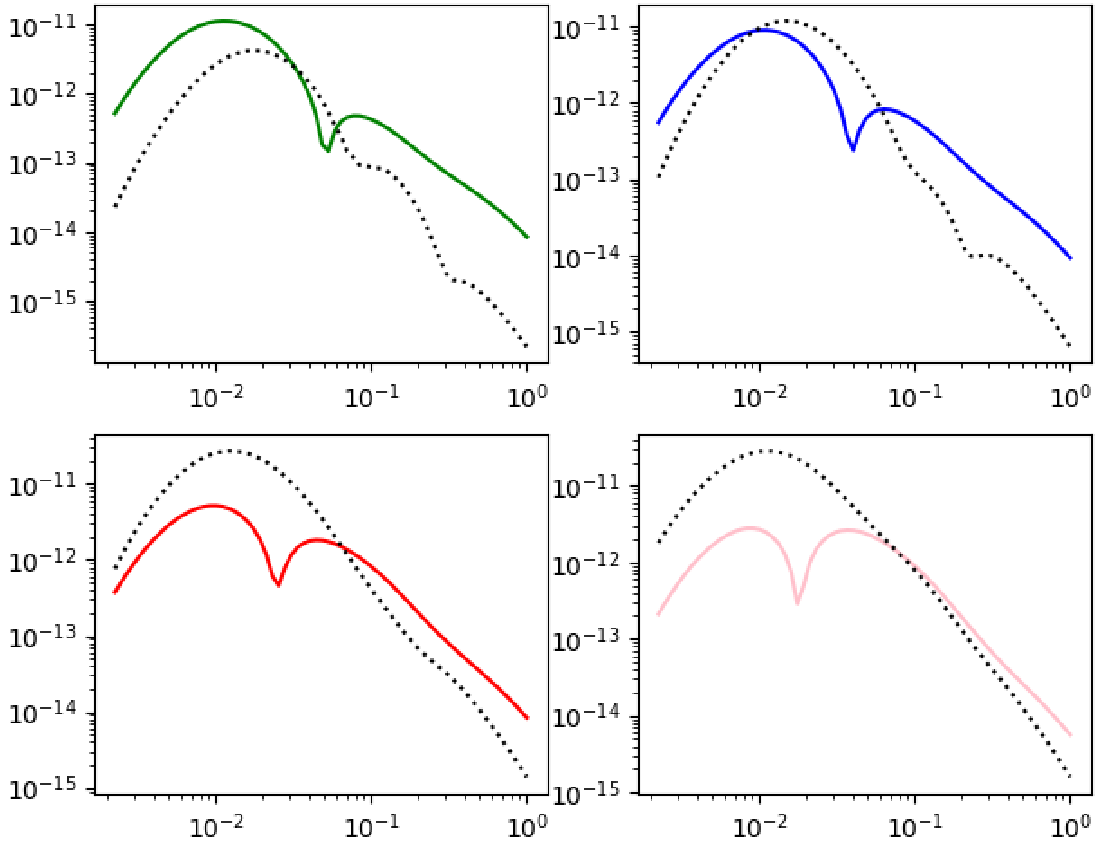

Рисунок 4.13 – Сравнение значений модуля E^{+} от времени в разных приёмниках при одноэтапном и многоэтапном разделении

4.5. ИССЛЕДОВАНИЕ ЯВЛЕНИЯ ВЗАИМОИНДУКЦИИ

Теперь предположим, что при использовании многоэтапной схемы разделения полей для обоих наших объектов можно найти, не учитывая влияния другого объекта, а учитывая лишь влияние поля в горизонтально-слоистой среде. Иными словами, значение $\overrightarrow{\mathbf{E}}^{n}$ при учёте второй по очереди аномалии в уравнении (1.6) будет рассматриваться не суммой $E^0 + E^a$ как мы делали это прежде, а просто через E^0, где E^0 – напряжённость электрического поля от влияния горизонтально-слоистой среды, а E^a – напряжённость электрического поля от влияния первой аномалии. На рисунках 4.14 – 4.17 приведены сравнения значений для разных приёмников. Сплошной линией отображена зависимость при учёте напряжённости электрического поля только горизонтально-слоистой среды (то есть просто E^0), а пунктиром с учётом ещё и влияния другой аномалии (то есть $E^0 + E^a$).

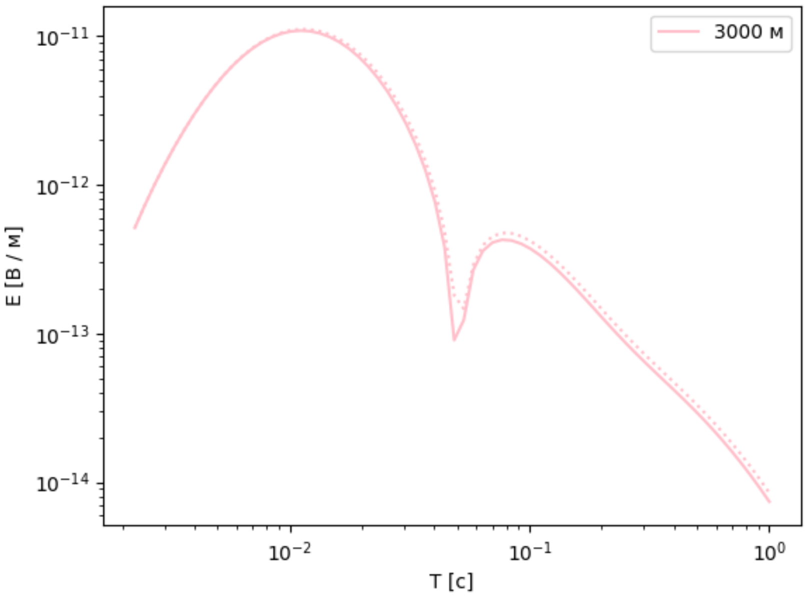

Рисунок 4.14 – Сравнение аномалии от двух объектов с суммой аномалий от одиночных объектов на приёмнике удалённом на 3000 м

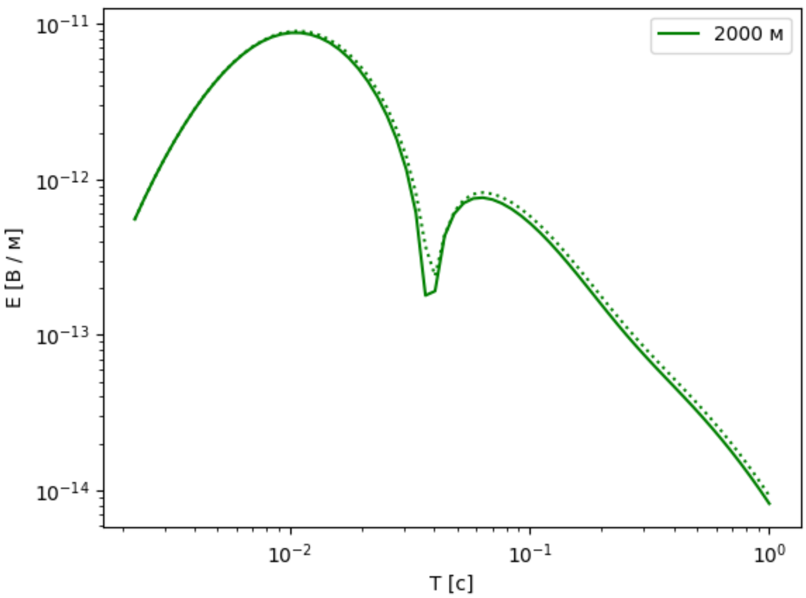

Рисунок 4.15 – Сравнение аномалии от двух объектов с суммой аномалий от одиночных объектов на приёмнике удалённом на 2000 м

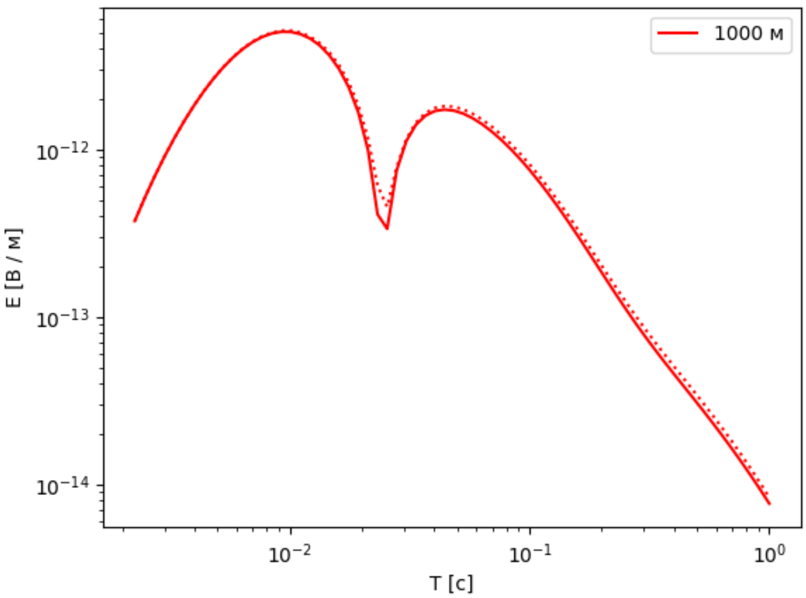

Рисунок 4.16 – Сравнение аномалии от двух объектов с суммой аномалий от одиночных объектов на приёмнике удалённом на 1000 м

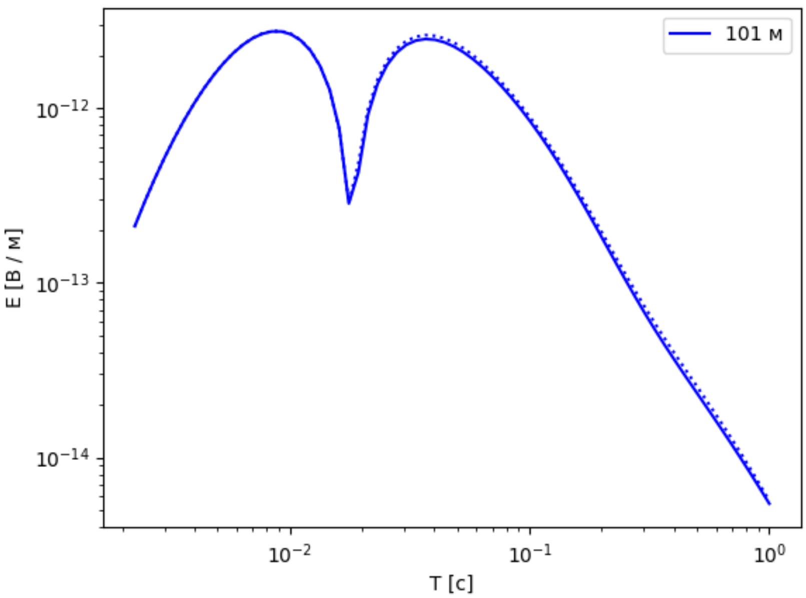

Рисунок 4.17 – Сравнение аномалии от двух объектов с суммой аномалий от одиночных объектов на приёмнике удалённом на 101 м

На рисунках 4.14 – 4.16 отчётливо видно, что пунктирная прямая, начиная примерно с 0.01 секунды, несколько выше, чем сплошная. Такую разницу можно объяснить явлением взаимоиндукции электромагнитных полей. Соответственно, такой учёт аномальных объектов при многоэтапном разделении полей не стоит применять если аномальные объекты находятся достаточно близко друг к другу.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе была разработанная программа для расчёта электромагнитного поля в трёхмерном пространстве.

Для проверки корректности работы программы была проведена ее верификация на полиномиальных функциях и вектор-функциях. В процессе тестирования осесимметричной задачи было получено, что на полиномах первой степени задача решается без погрешности, однако начиная с полинома второй степени появлялась погрешность, которая уменьшалась при дроблении сетки. Был рассчитан порядок сходимости метода решения, который, как и предполагалось, оказался равен порядку сходимости билинейных базисных функций. В процессе тестирования трёхмерных задач векторным методом конечных элементов результат оказался аналогичный результату осесимметричной задачи.

Было проведено исследование на поведение электромагнитного поля, при добавлении аномалий в разные места горизонтально-слоистой среды многоэтапной схемой разделения полей. По итогам исследования была проведена оценка поведения поля при различном использовании схемы разделения. Порядок добавления аномалий в область не дал никакого влияния, т.е. порядок добавления объектов не имеет разницы при разделении полей. Также было выяснено, что при достаточно близком расположении аномальных объектов друг к другу может возникать явление взаимоиндукции двух тел. Соответственно, при использовании многоэтапной схемы разделения полей не рекомендуется пренебрегать учётом влияния других аномальных тел, расположенных на достаточно близком друг к другу расстоянии.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. М.С. Жданов Электроразведка. - М.: Недра, 1986. - 316 с.

2. А.А. Логачев, В.П. Захаров Магниторазведка. - 5 изд. - Ленинград: Недра, 1979. - 350 с.

3. Pavel Solin Partial Differential Equations and the Finite Element Method. - Hoboken, New Jersey: A JOHN WILEY & SONS, INC., 2006.

4. А.Н. Тихонов, А.А. Самарский Уравнения математической физики: Учеб.пособие. / А.Н. Тихонов, А.А. Самарский — 6-е изд., — М: Изд-во МГУ, 1999 — 799 с.

5. М.Г. Персова, Ю.Г. Соловейчик, М.Г. Токарева, М.В. Абрамов 3D-моделирование процессов индукционной вызванной поляризации при возбуждении токовой петлей и проблема эквивалентности // Научный вестник НГТУ. - 2013. - №2(51). - С. 53 - 61.

6. М. Г. Персова, Ю. Г. Соловейчик, Г. М. Тригубович, М. В. Абрамов, А. А. Заборцева О вычислении трёхмерного нестационарного поля вертикальной электрической линии в удалённой обсаженной скважине // Сибирский журнал индустриальной математики. - 2007. - №3(31). - С. 114 - 127.

7. Ю.Г. Соловейчик, М.Э. Рояк, М.Г. Персова Метод конечных элементов для скалярных и векторных задач Учеб. пособие. — Новосибирск: Изд-во НГТУ, 2007 — 896 с.

8. М.Ю.Баландин, Э.П.Шурина Векторный метод конечных элементов: Учеб. пособие. - Новосибирск: Изд-во НГТУ, 2001 — 69 с.

9. М.Ю.Баландин, Э.П.Шурина Методы решения СЛАУ большой размерности: Учеб. пособие. - Новосибирск: Изд-во НГТУ, 2000 — 70 с.

10. М.Г. Персова, Ю.Г. Соловейчик, Д.В. Вагин, П.А. Домников, Ю.И. Кошкина Численные методы в уравнениях математической физики. - Новосибирск: Изд-во НГТУ, 2016 — 60 с.

11. Вагин Денис Владимирович Разработка методов конечноэлементного моделирования трехмерных электромагнитных полей на неструктурированных сетках: автореф. дис. канд. техн. наук: 05.13.18. - Новосибирск, 2012.

12. Тракимус Юрий Викторович Разработка и применение схем конечноэлементного моделирования электромагнитных полей в задачах электроразведки с использованием скважин: автореф. дис. канд. техн. наук: 05.13.18. - Новосибирск, 2007.

13. П.А. Домников Решение систем конечноэлементных уравнений при моделировании гармонических геоэлектромагнитных полей в трехмерных задачах морской электроразведки // Доклады Академии наук высшей школы Российской Федерации. - 2013. - №1 (20).

ПРИЛОЖЕНИЕ 3. ТЕКСТ ПРОГРАММЫ

Program.cs

```
using Project;
using System.Globalization;
using Solver;
using Processor;
using Grid;
using static Grid.MeshReader;
using static Grid.MeshGenerator;
using static Manager.FolderManager;
using DataStructs;

CultureInfo.CurrentCulture = CultureInfo.InvariantCulture;

string InputDirectory = Path.GetFullPath("../../../../Data/Input/");
string SubtotalsDirectory = Path.GetFullPath("../../../../Data/Subtotals/");
string OutputDirectory = Path.GetFullPath("..\\..\\..\\..\\Data\\Output\\");
string PicturesDirectory = Path.GetFullPath("../../../../Drawer/Pictures/");
bool isSolving2DimTask = Checker(OutputDirectory);

// Pre-processor. Clearing output folders.
if (isSolving2DimTask)
    ClearFolders(new List<string> {SubtotalsDirectory + "/2_dim/",
                                   PicturesDirectory + "/E_phi/",
                                   PicturesDirectory + "/A_phi/",
                                   OutputDirectory});

// Reading mesh.
ReadMesh(InputDirectory + "WholeMesh.txt");
ReadTimeMesh(InputDirectory + "Time.txt");

// Set recivers
List<Point3D> recivers = [new(681.9, -681.9, 0.0), new(1331.4, -1331.4, 0.0),
                          new(1980.8, -1980.8, 0.0), new(2630.3, -2630.3, 0.0)];

Mesh3Dim mesh3D = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
                      NodesZ, InfoAboutZ, Elems, Borders);
Mesh2Dim mesh2D = new(NodesR, InfoAboutR, NodesZ, InfoAboutZ,
                      Elems, Math.Sqrt(Math.Pow(mesh3D.nodesX[^1], 2) +
                      ↪ Math.Pow(mesh3D.nodesY[^1], 2)));
mesh2D.SetBorders(mesh3D.borders);
var timeMesh = GenerateTimeMesh(Time.Item1, Time.Item2, tn, tk);

// Main process of 2-dim task.
ConstructMesh(ref mesh2D);
FEM2D myFEM2D = new(mesh2D, timeMesh);
if (isSolving2DimTask)
{
    myFEM2D.SetSolver(new LU_LOS());
    myFEM2D.Solve();
    myFEM2D.GenerateVectorEphi();
    myFEM2D.WriteData(OutputDirectory);
    myFEM2D.WritePointsToDraw(OutputDirectory + "ToDraw\\2_dim\\Aphi\\",
                              OutputDirectory + "ToDraw\\2_dim\\Ephi\\");
    myFEM2D.MeasureValuesOnReceivers(recivers, OutputDirectory +
    ↪ "ToDraw\\2_dim\\Receivers\\");
}
else
{
    myFEM2D.ReadAnswer(OutputDirectory);
    Console.WriteLine("2D answer read");
}
myFEM2D.MeasureValuesOnReceivers(recivers, OutputDirectory +
↪ "ToDraw\\2_dim\\Receivers\\");
ConstructMesh(ref mesh3D);
Console.WriteLine("3D mesh constructed");
```

```
FEM3D myFEM3D = new(mesh3D, timeMesh);
myFEM3D.ConvertResultTo3Dim(myFEM2D);
myFEM3D.GenerateVectorB();
Console.WriteLine("2D answer converted to 3D");

// Solving first layer: groundwater.
ReadAnomaly(InputDirectory + "Anomalies\\Anomaly1.txt");
Console.WriteLine("Anomaly read");

Mesh3Dim mesh3D_a1 = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
                         NodesZ, InfoAboutZ, Elems, Borders);
mesh3D_a1.CommitAnomalyBorders(FieldBorders);
ConstructMeshAnomaly(ref mesh3D_a1, SubtotalsDirectory + "3_dim\\Anomaly0\\");
Console.WriteLine("Anomaly mesh built");
FEM3D fem3D_a1 = new(mesh3D_a1, timeMesh, myFEM3D, 0);
fem3D_a1.SetSolver(new LU_LOS(15_000, 1e-15));
Console.WriteLine("Solving begun");
fem3D_a1.Solve();
fem3D_a1.WriteData(OutputDirectory + $"A_phi\\Answer3D\\AfterField1\\");
Console.WriteLine("Solved");
fem3D_a1.GenerateVectorE();
Console.WriteLine("E generated");
myFEM3D.AddSolution(fem3D_a1);
myFEM3D.MeasureValuesOnReceivers(recivers, OutputDirectory +
↪ "ToDraw\\3_dim\\Receivers\\");

// Solving second layer: groundwater.
ReadAnomaly(InputDirectory + "Anomalies\\Anomaly2.txt");
Console.WriteLine("Anomaly read");
Mesh3Dim mesh3D_a2 = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
                         NodesZ, InfoAboutZ, Elems, Borders);
mesh3D_a2.CommitAnomalyBorders(FieldBorders);
ConstructMeshAnomaly(ref mesh3D_a2, SubtotalsDirectory + "3_dim\\Anomaly1\\");
Console.WriteLine("Anomaly mesh built");
FEM3D fem3D_a2 = new(mesh3D_a2, timeMesh, myFEM3D, 1);
fem3D_a2.SetSolver(new LU_LOS(15_000, 1e-15));
Console.WriteLine("Solving begun");
fem3D_a2.Solve();
fem3D_a2.WriteData(OutputDirectory + $"A_phi\\Answer3D\\AfterField2\\");
Console.WriteLine("Solved");
fem3D_a2.GenerateVectorE();
Console.WriteLine("E generated");
myFEM3D.AddSolution(fem3D_a2);
myFEM3D.WriteDrawingForSecond(OutputDirectory + "ToDraw\\3_dim\\");
myFEM3D.MeasureValuesOnReceivers(recivers, OutputDirectory +
↪ "ToDraw\\3_dim\\Receivers\\");
Console.WriteLine($"{e}");

// Solving both anomalies.
ReadBothAnomalies(InputDirectory + "Anomalies\\AnomalyBoth.txt");
Console.WriteLine("Anomalies read");
Mesh3Dim mesh3D_ab = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
                         NodesZ, InfoAboutZ, Elems, Borders);
mesh3D_ab.CommitAnomalyBorders(FieldBorders);
mesh3D_ab.CommitSecondAnomalyBorders(FieldBorders1);
ConstructMeshAnomaly(ref mesh3D_ab, SubtotalsDirectory + "3_dim\\Anomaly2\\");
Console.WriteLine("Anomalies mesh built");
FEM3D fem3D_ab = new(mesh3D_ab, timeMesh, myFEM3D, 2);
fem3D_ab.SetSolver(new LU_LOS(15_000, 1e-15));
Console.WriteLine("Solving begun");
fem3D_ab.Solve();
fem3D_ab.WriteData(OutputDirectory + $"A_phi\\Answer3D\\AfterFieldBoth\\");
Console.WriteLine("Solved");
fem3D_ab.GenerateVectorE();
Console.WriteLine("E generated");
myFEM3D.AddSolution(fem3D_ab);
myFEM3D.WriteDataToDraw2DimSolution(OutputDirectory + "ToDraw\\3_dim\\");
myFEM3D.MeasureValuesOnReceivers(recivers, OutputDirectory +
↪ "ToDraw\\3_dim\\Receivers\\");
return 0;

static bool Checker(string Answer)
{
    if (CountFilesAmount(Answer + "A_phi/Answer/") == CountFilesAmount(Answer +
    ↪ "E_phi/Answer/"))
```

```
        while (true)
        {
            string? ans;
            Console.WriteLine("Detected answer for 2-dim task. Would you like to solve
            ↪  2-dim task again? [Y/n]");
            ans = Console.ReadLine()?.ToLower();
            if (ans == "y" || ans == "yes" || ans == "да" || ans == "д")
                return true;
            else if (ans == "n" || ans == "no" || ans == "нет" || ans == "н")
                return false;
            Console.WriteLine("Unexpected answer!");
        }
    return true;
}
```

LocalMatrix.cs

```
using DataStructs;
using static Functions.BasisFunctions2D;
using Solution;

namespace MathObjects;


public class LocalMatrix : Matrix
{
    private TypeOfMatrixM _typeOfMatrixM;
    private readonly double _lambda;
    private readonly double _gamma;
    private readonly double _rk;
    private readonly double _hr;
    private readonly double _hz;

    public override double this[int i, int j]
    {
        get
        {
            if (i > 3 || j > 3) throw new IndexOutOfRangeException("Local matrix error.");
            return _typeOfMatrixM switch
            {
                TypeOfMatrixM.Mr =>  _gamma * (_Mr[i % 2, j % 2] * _Mz[i / 2, j / 2]),
                TypeOfMatrixM.Mrr => _lambda * (_Gr[i % 2, j % 2] * _Mz[i / 2, j / 2] +
                ↪  _Mr[i % 2, j % 2] * _Gz[i / 2, j / 2]) +
                                     _lambda * (_Mrr[i % 2, j % 2] * _Mz[i / 2, j / 2]),
                _ => throw new Exception("Unexpected matrix"),
            };
        }
        set{}
    }

    private readonly double[,] _G =  {{ 1.0, -1.0},
                                      {-1.0,  1.0}};
    private readonly double[,] _Mz = {{2.0, 1.0},
                                      {1.0, 2.0}};
    private readonly double[,] _Mr1;
    private readonly double[,] _M1r = {{2.0, 1.0},
                                       {1.0, 2.0}};
    private readonly double[,] _M2r = {{1.0, 1.0},
                                       {1.0, 3.0}};
    private readonly double[,] _Mr2 = {{-3.0D, 1.0D},
                                       { 1.0D, 1.0D}};
    private readonly double[,] _Gr = new double[2, 2];
    private readonly double[,] _Mr = new double[2, 2];
    private readonly double[,] _Gz = new double[2, 2];
    private readonly double[,] _Mrr = new double[2, 2];
    double[,] matr = new double[4, 4];

    public LocalMatrix(double lambda, double rk, double hz, double hr)
    {
        _rk = rk;
        _hr = hr;
        _hz = hz;
        double _d = _rk / _hr;
```

```
        _lambda = lambda;
        _gamma = _lambda;
        _Mr1 = new double[2,2] {{ (1 + _d) * (1 + _d), -1.0 * _d * (1 + _d)},
                                {-1.0 * _d * (1 + _d),                _d * _d}};
        _typeOfMatrixM = TypeOfMatrixM.Mrr;
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                _Gr[i, j] = ((_rk + _hr / 2.0D) / _hr) * _G[i, j];
                _Mr[i, j] = (_hr / 6.0D) * (_rk * _M1r[i, j] + (_hr / 2.0D) * _M2r[i, j]);
                _Mrr[i, j] = Math.Log(1.0D + 1.0D / _d) * _Mr1[i, j] - _d * _G[i, j] + 0.5 * _Mr2[i, j];
                _Gz[i, j] = _G[i, j] / _hz;
                _Mz[i, j] = (_hz / 6.0D) * _Mz[i, j];
            }
        }
    }

    public LocalMatrix(List<int> elem, ArrayOfPoints2D arrPt, TypeOfMatrixM typeOfMatrixM, double lambda = 0.0D, double gamma = 0.0D)
    {
        _typeOfMatrixM = typeOfMatrixM;
        _rk = arrPt[elem[0]].R;
        _hr = arrPt[elem[1]].R - arrPt[elem[0]].R;
        _hz = arrPt[elem[2]].Z - arrPt[elem[0]].Z;
        double _d = _rk / _hr;
        _lambda = 1.0D / lambda;
        _gamma = gamma;
        _Mr1 = new double[2,2] {{ (1 + _d) * (1 + _d), -1.0 * _d * (1 + _d)},
                                {-1.0 * _d * (1 + _d),                _d * _d}};

        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                _Gr[i, j] = ((_rk + _hr / 2.0D) / _hr) * _G[i, j];
                _Mr[i, j] = (_hr / 6.0D) * (_rk * _M1r[i, j] + (_hr / 2.0D) * _M2r[i, j]);
                _Mrr[i, j] = Math.Log(1.0D + 1.0D / _d) * _Mr1[i, j] - _d * _G[i, j] + 0.5 * _Mr2[i, j];
                _Gz[i, j] = _G[i, j] / _hz;
                _Mz[i, j] = (_hz / 6.0D) * _Mz[i, j];
            }
        }
    }
}
```

LocalVector.cs

```
using System.Drawing;
using DataStructs;
using Functions;

namespace MathObjects;

public class LocalVector : Vector
{

    private readonly double _r0;
    private readonly double _r1;
    private readonly double _z0;
    private readonly double _z1;
    private readonly double _hr;
    private readonly double _hz;
    private readonly double _t;
    public override int Size => 4;

    private readonly double[,] _M2R = {{1.0D, 1.0D},
                                       {1.0D, 3.0D}};
    private readonly double[,] _Mz = {{2.0D, 1.0D},
                                      {1.0D, 2.0D}};
    private readonly double[,] _M1R = {{2.0D, 1.0D},
                                       {1.0D, 2.0D}};
```

```
    public override double this[int i] => i switch
    {
        0 => _Mr[0, 0] * _Mz[0, 0] * Function.F(_r0, _z0, _t) +
             _Mr[0, 1] * _Mz[0, 0] * Function.F(_r1, _z0, _t) +
             _Mr[0, 0] * _Mz[0, 1] * Function.F(_r0, _z1, _t) +
             _Mr[0, 1] * _Mz[0, 1] * Function.F(_r1, _z1, _t),

        1 => _Mr[1, 0] * _Mz[0, 0] * Function.F(_r0, _z0, _t) +
             _Mr[1, 1] * _Mz[0, 0] * Function.F(_r1, _z0, _t) +
             _Mr[1, 0] * _Mz[0, 1] * Function.F(_r0, _z1, _t) +
             _Mr[1, 1] * _Mz[0, 1] * Function.F(_r1, _z1, _t),

        2 => _Mr[0, 0] * _Mz[1, 0] * Function.F(_r0, _z0, _t) +
             _Mr[0, 1] * _Mz[1, 0] * Function.F(_r1, _z0, _t) +
             _Mr[0, 0] * _Mz[1, 1] * Function.F(_r0, _z1, _t) +
             _Mr[0, 1] * _Mz[1, 1] * Function.F(_r1, _z1, _t),

        3 => _Mr[1, 0] * _Mz[1, 0] * Function.F(_r0, _z0, _t) +
             _Mr[1, 1] * _Mz[1, 0] * Function.F(_r1, _z0, _t) +
             _Mr[1, 0] * _Mz[1, 1] * Function.F(_r0, _z1, _t) +
             _Mr[1, 1] * _Mz[1, 1] * Function.F(_r1, _z1, _t),

        _ => throw new IndexOutOfRangeException("Vector out of index"),
    };

    private double[,] _Mr = new double[2, 2];

    public LocalVector(List<int> elem, ArrayOfPoints2D arrPt, double t)
    {
        _r0 = arrPt[elem[0]].R;
        _r1 = arrPt[elem[1]].R;
        _t = t;

        _z0 = arrPt[elem[0]].Z;
        _z1 = arrPt[elem[2]].Z;
        _hr = _r1 - _r0;
        _hz = _z1 - _z0;

        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                _Mr[i, j] = (_hr / 6.0) * (_r0 * _M1R[i, j] + (_hr / 2.0) * _M2R[i, j]);
                _Mz[i, j] = (_hz / 6.0) * _Mz[i, j];
            }
        }
    }

    private void WriteVector()
    {
        for (int i = 0; i < 4; i++)
            Console.WriteLine($"{this[i]:E5}");
    }

    public LocalVector(double r0, double r1, double z0, double z1)
    {
        _r0 = r0;
        _r1 = r1;
        _z0 = z0;
        _z1 = z1;
        _hr = _r1 - _r0;
        _hz = _z1 - _z0;
    }
}
```

FEM.cs

```
namespace Project;
using System.Collections.Immutable;
using System.Numerics;
using MathObjects;
using Solver;
```

```
using Grid;
using DataStructs;
using System.Diagnostics;
using Functions;
using System.ComponentModel.DataAnnotations;
using System.Timers;

public enum EquationType
{
    Elliptic,
    Parabolic
}

public abstract class FEM(TimeMesh time)
{
    protected static string _3dValuesPath = Path.GetFullPath("../../../../Data/Subtotals/3_dim/");
    protected static string _elemspath2D = Path.GetFullPath("../../../../Data/Subtotals/2_dim/Elems.poly");
    protected static string _pointspath2D = Path.GetFullPath("../../../../Data/Subtotals/2_dim/Points.poly");
    protected static string _borderspath2D = Path.GetFullPath("../../../../Data/Subtotals/2_dim/Borders.poly");
    protected static string _elemspath3D = Path.GetFullPath("../../../../Data/Subtotals/3_dim/Elems.poly");
    protected static string _pointspath3D = Path.GetFullPath("../../../../Data/Subtotals/3_dim/Points.poly");
    protected static string _borderspath3D = Path.GetFullPath("../../../../Data/Subtotals/3_dim/Borders.poly");

    public TimeMesh Time = time;
    protected internal EquationType equationType;
    public ISolver? solver;
    public ArrayOfElems elemsArr;
    public ArrayOfBorders bordersArr;
    public GlobalMatrix? Matrix;
    public GlobalVector? Vector;
    public GlobalVector? Answer;
    public GlobalVector[] Solutions = new GlobalVector[time.Count];
    public GlobalVector[] Discrepancy = new GlobalVector[time.Count];

    public void SetSolver(ISolver solver)
    {
        this.solver = solver;
        Debug.WriteLine("Solvet set");
    }
}
```

FEM2D.cs

```
using MathObjects;
using DataStructs;
using Grid;
using System.Diagnostics;
using Functions;

namespace Project;

public class FEM2D : FEM
{
    public ArrayOfPoints2D pointsArr = new(_pointspath2D);

    public FEM2D(Mesh2Dim mesh, TimeMesh timeMesh) : base(timeMesh)
    {
        mesh2Dim = mesh;
        if (timeMesh[0] == timeMesh[^1])
            equationType = EquationType.Elliptic;
        else
            equationType = EquationType.Parabolic;

        elemsArr = new(_elemspath2D);
        bordersArr = new(_borderspath2D);

        A_phi = new GlobalVector[Time.Count];
```

```
        E_phi = new GlobalVector[Time.Count];
        Debug.WriteLine("Generated data submited");
    }

    private readonly Mesh2Dim mesh2Dim;
    public GlobalVector[] A_phi;
    public GlobalVector[] E_phi;

    public void Solve()
    {
        if (solver is null) throw new ArgumentNullException("solver is null !");
        if (Time is null) throw new ArgumentNullException("Time is null!");

        Stopwatch solutionStopwatch = new();
        solutionStopwatch.Start();

        Debug.WriteLine($"\nTime layer: before BC");
        Thread.Sleep(1500);

        Matrix = new GlobalMatrix(pointsArr.GetLength());
        Generator.BuildPortait(ref Matrix, pointsArr.GetLength(), elemsArr);
        Generator.FillMatrix(ref Matrix, pointsArr, elemsArr, TypeOfMatrixM.Mrr);

        Vector = new GlobalVector(pointsArr.GetLength());
        Generator.FillVector(ref Vector, pointsArr, elemsArr, Time[0]);

        Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, pointsArr, bordersArr, Time[0]);
        (Solutions[0], Discrepancy[0]) = solver.Solve(Matrix, Vector);

        if (Time.Count > 1)
        {
            (Solutions[1], Discrepancy[1]) = (Solutions[0], Discrepancy[0]);
            if (Time.Count > 2)
            for (int i = 2; i < Time.Count; i++)
            {
                Console.WriteLine($"\n {i} / {Time.Count - 1}. Time layer: {Time[i]}");
                //Thread.Sleep(1500);

                double deltT = Time[i] - Time[i - 2];
                double deltT0 = Time[i] - Time[i - 1];
                double deltT1 = Time[i - 1] - Time[i - 2];
                double tau0 = (deltT + deltT0) / (deltT * deltT0);
                double tau1 = deltT / (deltT1 * deltT0);
                double tau2 = deltT0 / (deltT * deltT1);
                double deltT = Time[i] - Time[i - 1];
                double tau = 1.0D / deltT;

                var matrix1 = new GlobalMatrix(pointsArr.GetLength());
                Generator.BuildPortait(ref matrix1, pointsArr.GetLength(), elemsArr);
                Generator.FillMatrix(ref matrix1, pointsArr, elemsArr, TypeOfMatrixM.Mrr);
                var M = new GlobalMatrix(pointsArr.GetLength()); // ???
                Generator.BuildPortait(ref M, pointsArr.GetLength(), elemsArr);
                Generator.FillMatrix(ref M, pointsArr, elemsArr, TypeOfMatrixM.Mr);

                var bi = new GlobalVector(pointsArr.GetLength());
                Matrix = (tau0 * M) + matrix1;
                Vector = bi + (tau1 * (M * Solutions[i - 1])) - (tau2 * (M * Solutions[i - 2]));
                Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, pointsArr, bordersArr, Time[i]);
                (Solutions[i], Discrepancy[i]) = solver.Solve(Matrix, Vector);
            }
        }
        A_phi = Solutions;
        solutionStopwatch.Stop();
        var milseconds = solutionStopwatch.ElapsedMilliseconds;
        Console.WriteLine($"Lin eq solved for {milseconds / 60000} min {(milseconds % 60000) / 1000} sec");
    }

    public void WriteData()
    {
        if (Answer is null)
            throw new Exception("Vector _answer is null");
```

```
        for (int i = 0; i < Answer.Size; i++)
            Console.WriteLine($"{Answer[i]:E15}");
    }

    public void WriteData(string _path)
    {
        if (A_phi is null) throw new ArgumentNullException();
        if (E_phi is null) throw new ArgumentNullException();

        if (Time.Count != 1)
        {
            for (int i = 0; i < Time.Count; i++)
            {
                using var sw = new StreamWriter($"{_path}\\A_phi\\Answer\\Answer_Aphi_time={Time[i]}.dat");
                for (int j = 0;   j < A_phi[i].Size; j++)
                    sw.WriteLine($"{A_phi[i][j]:E8}");
                sw.Close();
            }
            for (int i = 0; i < Time.Count; i++)
            {
                using var sw = new StreamWriter($"{_path}\\E_phi\\Answer\\Answer_Ephi_time={Time[i]}.dat");
                for (int j = 0;   j < E_phi[i].Size; j++)
                    sw.WriteLine($"{E_phi[i][j]:E8}");
                sw.Close();
            }
        }
        else
        {
            using var sw = new StreamWriter($"{_path}\\A_phi\\Answer\\Answer.dat");
            for (int j = 0; j < A_phi[0].Size; j++)
                sw.WriteLine($"{A_phi[0][j]:E8}");
            sw.Close();
            using var sw1 = new StreamWriter($"{_path}\\E_phi\\Answer\\Answer.dat");
            for (int j = 0; j < E_phi[0].Size; j++)
                sw1.WriteLine($"{E_phi[0][j]:E8}");
            sw1.Close();
        }
    }

    public void WriteDiscrepancy(string _path)
    {
        if (A_phi is null) throw new ArgumentNullException();
        if (E_phi is null) throw new ArgumentNullException();

        if (Time.Count != 1)
        {
            for (int i = 0; i < Time.Count; i++)
            {
                using var sw_d = new StreamWriter($"{_path}\\A_phi\\Discrepancy\\Discrepancy_Aphi_time={Time[i]}.dat");

                int NotNaNamount = 0;
                double maxDisc = 0.0;
                double avgDisc = 0.0;
                double sumU = 0.0D;
                double sumD = 0.0D;

                List<double> TheorAnswer = [];
                foreach (var Z in mesh2Dim.nodesZ)
                    foreach (var R in mesh2Dim.nodesR)
                        TheorAnswer.Add(Function.U(R, Z, Time[i]));

                for (int j = 0; j < A_phi[i].Size; j++)
                {
                    double absDiff = Math.Abs(A_phi[i][j] - TheorAnswer[j]);
                    double currDisc = Math.Abs((A_phi[i][j] - TheorAnswer[j]) / TheorAnswer[j]);

                    if (Math.Abs(maxDisc) < Math.Abs(currDisc))
                        maxDisc = currDisc;

                    if (!double.IsNaN(currDisc) && currDisc > 1E-14)
                    {
```

```
                        avgDisc += currDisc;
                        NotNaNamount++;
                        sumU += absDiff * absDiff;
                        sumD += TheorAnswer[j] * TheorAnswer[j];
                    }
                    sw_d.WriteLine($"{absDiff:E8} {currDisc:E8}");
                }
                avgDisc = Math.Sqrt(sumU) / Math.Sqrt(sumD);
                sw_d.WriteLine($"Средняя невязка: {avgDisc:E15}");
                sw_d.WriteLine($"Максимальная невязка: {maxDisc:E15}");
                sw_d.WriteLine($"C: {avgDisc:E7}");
                sw_d.WriteLine($"M: {maxDisc:E7}");
                sw_d.Close();
            }
        }
        else
        {
            using var sw_d = new StreamWriter($"{_path}\\A_phi\\Discrepancy\\Discrepancy_Aphi.dat");

            int NotNaNamount = 0;
                double maxDisc = 0.0;
                double avgDisc = 0.0;
                double sumU = 0.0D;
                double sumD = 0.0D;
                List<double> TheorAnswer = [];
                foreach (var Z in mesh2Dim.nodesZ)
                    foreach (var R in mesh2Dim.nodesR)
                        TheorAnswer.Add(Function.U(R, Z, 0.0D));
                for (int j = 0; j < A_phi[0].Size; j++)
                {
                    double absDiff = Math.Abs(A_phi[0][j] - TheorAnswer[j]);
                    double currDisc = Math.Abs((A_phi[0][j] - TheorAnswer[j]) / TheorAnswer[j]);

                    if (Math.Abs(maxDisc) < Math.Abs(currDisc))
                        maxDisc = currDisc;

                    if (!double.IsNaN(currDisc) && currDisc > 1E-14)
                    {
                        avgDisc += currDisc;
                        NotNaNamount++;
                        sumU += absDiff * absDiff;
                        sumD += TheorAnswer[j] * TheorAnswer[j];
                    }
                    sw_d.WriteLine($"{absDiff:E8} {currDisc:E8}");
                }
                avgDisc = Math.Sqrt(sumU) / Math.Sqrt(sumD);
                sw_d.WriteLine($"Средняя невязка: {avgDisc:E15}");
                sw_d.WriteLine($"Максимальная невязка: {maxDisc:E15}");
                sw_d.WriteLine($"C: {avgDisc:E7}");
                sw_d.WriteLine($"M: {maxDisc:E7}");
                sw_d.Close();
        }
    }

    public void GenerateVectorEphi()
    {
        E_phi = new GlobalVector[A_phi.Length];
        for (int i = 0; i < E_phi.Length; i++)
        {
            if (i == 0)
                E_phi[i] = new GlobalVector(A_phi[i].Size);
            else
                E_phi[i] = -1.0D / (Time[i] - Time[i - 1]) * (A_phi[i] - A_phi[i - 1]);
        }
    }

    internal List<int>? GetElem(double r, double z)
    {
        if (r < mesh2Dim.nodesR[0] || mesh2Dim.nodesR[^1] < r || z < mesh2Dim.nodesZ[0] || mesh2Dim.nodesZ[^1] < z)
            return null;
        int i = 0;
        for (; i < mesh2Dim.nodesR.Count - 1 && r >= 0.001; i++)
            if (mesh2Dim.nodesR[i] <= r && r <= mesh2Dim.nodesR[i + 1])
```

```
                break;
        int j = 0;
        for (; j < mesh2Dim.nodesZ.Count - 1; j++)
            if (mesh2Dim.nodesZ[j] <= z && z <= mesh2Dim.nodesZ[j + 1])
                break;
        return elemsArr[j * (mesh2Dim.nodesR.Count - 1) + i].Arr;
    }

    public double GetA_phiAt(double r, double z, double t)
    {
        for (int tt = 0; tt < Time.Count; tt++)
        {
            if (Time[tt] == t)
            {
                var elem = GetElem(r, z);
                if (elem is null) return 0.0D;
                double[] q = new double[4];
                for (int i = 0; i < 4; i++)
                    q[i] = A_phi[tt][elem[i]];
                double r0 = pointsArr[elem[0]].R;
                double r1 = pointsArr[elem[3]].R;
                double z0 = pointsArr[elem[0]].Z;
                double z1 = pointsArr[elem[3]].Z;
                return BasisFunctions2D.GetValue(q[0], q[1], q[2], q[3], r0, r1, z0, z1,
                ↪  r, z);
            }
        }
        throw new Exception("Out of mesh borders");
    }

    public double GetE_phiAt(double r, double z, double t)
    {
        for (int tt = 0; tt < Time.Count; tt++)
        {
            if (Time[tt] == t)
            {
                var elem = GetElem(r, z);
                if (elem is null) return 0.0D;
                double[] q = new double[4];
                for (int i = 0; i < 4; i++)
                    q[i] = E_phi[tt][elem[i]];
                double r0 = pointsArr[elem[0]].R;
                double r1 = pointsArr[elem[3]].R;
                double z0 = pointsArr[elem[0]].Z;
                double z1 = pointsArr[elem[3]].Z;
                return BasisFunctions2D.GetValue(q[0], q[1], q[2], q[3], r0, r1, z0, z1,
                ↪  r, z);
            }
        }
        throw new Exception("Out of mesh borders");
    }

    public void ReadAnswer(string AnswerPath)
    {
        string file = string.Empty;
        if (equationType == EquationType.Elliptic)
        {
            file = "Answer.dat";
            var fileData = File.ReadAllText(AnswerPath + "A_phi/Answer/" +
            ↪  file).Split("\n");
            A_phi[0] = new GlobalVector(fileData.Length - 1);
            for (int i = 0; i < fileData.Length - 1; i++)
                A_phi[0][i] = double.Parse(fileData[i]);
            fileData = File.ReadAllText(AnswerPath + "E_phi/Answer/" + file).Split("\n");
            E_phi[0] = new GlobalVector(fileData.Length - 1);
            for (int i = 0; i < fileData.Length - 1; i++)
                E_phi[0][i] = double.Parse(fileData[i]);
        }
        else
        {
            for (int t = 0; t < Time.Count; t++)
            {
                file = $"Answer_Aphi_time={Time[t]}.dat";
                var fileData = File.ReadAllText(AnswerPath + "A_phi/Answer/" +
                ↪  file).Split("\n");
```

```
                A_phi[t] = new GlobalVector(fileData.Length - 1);
                for (int i = 0; i < fileData.Length - 1; i++)
                    A_phi[t][i] = double.Parse(fileData[i]);
                file = $"Answer_Ephi_time={Time[t]}.dat";
                fileData = File.ReadAllText(AnswerPath + "E_phi/Answer/" + file).Split("\n");
                E_phi[t] = new GlobalVector(fileData.Length - 1);
                for (int i = 0; i < fileData.Length - 1; i++)
                    E_phi[t][i] = double.Parse(fileData[i]);
            }
        }
    }

    public void MeasureValuesOnReceivers(List<Point3D> recivers, string OutputPath)
    {
        using var sw_a = new StreamWriter(OutputPath + "A.txt");
        using var sw_e = new StreamWriter(OutputPath + "E.txt");
        List<(double, double)> pnt2D = [];
        foreach (var reciver in recivers)
            pnt2D.Add((Math.Sqrt(reciver.X * reciver.X + reciver.Y * reciver.Y), reciver.Z));
        for (int t = 0; t < Time.Count; t++)
        {
            var a_a = GetA_phiAt(pnt2D[0].Item1, pnt2D[0].Item2, Time[t]);
            var b_a = GetA_phiAt(pnt2D[1].Item1, pnt2D[1].Item2, Time[t]);
            var c_a = GetA_phiAt(pnt2D[2].Item1, pnt2D[2].Item2, Time[t]);
            var d_a = GetA_phiAt(pnt2D[3].Item1, pnt2D[3].Item2, Time[t]);
            var a_e = GetE_phiAt(pnt2D[0].Item1, pnt2D[0].Item2, Time[t]);
            var b_e = GetE_phiAt(pnt2D[1].Item1, pnt2D[1].Item2, Time[t]);
            var c_e = GetE_phiAt(pnt2D[2].Item1, pnt2D[2].Item2, Time[t]);
            var d_e = GetE_phiAt(pnt2D[3].Item1, pnt2D[3].Item2, Time[t]);
            sw_a.WriteLine($"{Time[t]} {a_a} {b_a} {c_a} {d_a}");
            sw_e.WriteLine($"{Time[t]} {a_e} {b_e} {c_e} {d_e}");
        }
        sw_a.Close();
        sw_e.Close();
    }

    public void WritePointsToDraw(string pathA, string pathE)
    {
        double hr = (mesh2Dim.nodesR[^1] - mesh2Dim.nodesR[0]) / 150;
        double hz = (mesh2Dim.nodesZ[^1] - mesh2Dim.nodesZ[0]) / 150;
        for (int t = 0; t < Time.Count; t++)
        {
            using var swa = new StreamWriter(pathA + $"Answer_A_time_layer_{t}.txt");
            for (int j = 0; j < 150; j++)
            {
                for (int i = 0; i < 150; i++)
                {
                    double rCurr = mesh2Dim.nodesR[0] + i * hr;
                    double zCurr = mesh2Dim.nodesZ[0] + j * hz;
                    swa.WriteLine($"{rCurr:E15} {zCurr:E15} {GetA_phiAt(rCurr, zCurr, Time[t]):E15}");
                }
            }
            swa.Close();
        }
        for (int t = 0; t < Time.Count; t++)
        {
            using var swe = new StreamWriter(pathE + $"Answer_E_time_layer_{t}.txt");
            for (int j = 0; j < 150; j++)
            {
                for (int i = 0; i < 150; i++)
                {
                    double rCurr = mesh2Dim.nodesR[0] + i * hr;
                    double zCurr = mesh2Dim.nodesZ[0] + j * hz;
                    swe.WriteLine($"{rCurr:E15} {zCurr:E15} {GetE_phiAt(rCurr, zCurr, Time[t]):E15}");
                }
            }
            swe.Close();
        }
    }
}
```

FEM3D.cs

```
using MathObjects;
using Grid;
using DataStructs;
using Functions;
using System.Diagnostics;



namespace Project;

public class FEM3D : FEM
{
    private static readonly double mu0 = 4.0D * Math.PI * Math.Pow(10.0D, -7);
    public ArrayOfPoints3D pointsArr;
    public List<GlobalVector> A;
    public List<GlobalVector> E;
    public List<GlobalVector> B;
    public List<GlobalVector> H;
    public ArrayOfRibs? ribsArr;
    private readonly Mesh3Dim mesh3Dim;
    private GlobalMatrix? G;
    List<FEM3D> additionalFields = [];
    private readonly FEM3D? _originalFEM;
    private List<GlobalVector>? _originalE;
    private GlobalMatrix? M;
    internal static List<int> ConvertGlobalToLocalNumeration(List<int> global) =>
            [global[0], global[3], global[8], global[11],
             global[1], global[2], global[9], global[10],
             global[4], global[5], global[6], global[7]];

    public FEM3D(Mesh3Dim mesh, TimeMesh timeMesh) : base(timeMesh)
    {
        string pointsPath = _3dValuesPath + $"AfterConvertation\\Points.poly";
        string elemsPath = _3dValuesPath + $"AfterConvertation\\Elems.poly";
        string bordersPath = _3dValuesPath + $"AfterConvertation\\Borders.poly";
        pointsArr = new ArrayOfPoints3D(pointsPath);
        elemsArr = new(elemsPath, 3);
        bordersArr = new (bordersPath, 3);
        ribsArr = mesh.arrayOfRibs;
        equationType = timeMesh[0] == timeMesh[^1] ? EquationType.Elliptic : EquationType.Parabolic;
        A = [];
        E = [];
        B = [];
        H = [];
        mesh3Dim = mesh;
    }

    public FEM3D(Mesh3Dim mesh, TimeMesh timeMesh, FEM3D originalFEM, int Num) : base(timeMesh)
    {
        string pointsPath = _3dValuesPath + $"Anomaly{Num}\\Points.poly";
        string elemsPath = _3dValuesPath + $"Anomaly{Num}\\Elems.poly";
        string bordersPath = _3dValuesPath + $"Anomaly{Num}\\Borders.poly";
        pointsArr = new(pointsPath);
        elemsArr = new(elemsPath, 3);
        bordersArr = new(bordersPath, 3);
        ribsArr = mesh.arrayOfRibs;
        _originalFEM = originalFEM;
        _originalE = [];
        equationType = timeMesh[0] == timeMesh[^1] ? EquationType.Elliptic : EquationType.Parabolic;
        A = [];
        E = [];
        B = [];
        H = [];
        mesh3Dim = mesh;
        ConstructMatrixes();
    }

    public (double, double, double) GetAAt(double x, double y, double z, double t)
    {
        if (ribsArr is null) throw new ArgumentNullException("Array of ribs not generated");
```

```
        for (int tt = 0; tt < Time.Count; tt++)
            if (Time[tt] == t)
            {
                var arr = GetElem(x, y, z);
                if (arr is null) return (0.0D, 0.0D, 0.0D);
                var elem = ConvertGlobalToLocalNumeration(arr);
                double[] q = new double[12];
                for (int i = 0; i < 12; i++)
                    q[i] = A[tt][elem[i]];
                double x0 = ribsArr[elem[0]].a.X;
                double x1 = ribsArr[elem[0]].b.X;
                double y0 = ribsArr[elem[4]].a.Y;
                double y1 = ribsArr[elem[4]].b.Y;
                double z0 = ribsArr[elem[8]].a.Z;
                double z1 = ribsArr[elem[8]].b.Z;
                double eps = (x - x0) / (x1 - x0);
                double nu =  (y - y0) / (y1 - y0);
                double khi = (z - z0) / (z1 - z0);
                var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
                foreach (var solution in additionalFields)
                {
                    var arrCurr = solution.GetElem(x, y, z);
                    if (arrCurr is null) continue;
                    var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
                    double[] qCurr = new double[12];
                    for (int ii = 0; ii < 12; ii++)
                        qCurr[ii] = solution.A[tt][elemCurr[ii]];
                    double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
                    double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
                    double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
                    double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
                    double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
                    double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
                    double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
                    double nuCurr =  (y - y0Curr) / (y1Curr - y0Curr);
                    double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
                    var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr, qCurr);
                    ans.Item1 += ansCurr.Item1;
                    ans.Item2 += ansCurr.Item2;
                    ans.Item3 += ansCurr.Item3;
                }
                return ans;
            }
        throw new Exception("Out of mesh borders");
    }

    public (double, double, double) AdditioanalGetEAt(double x, double y, double z, double t)
    {
        if (ribsArr is null) throw new ArgumentNullException("Array of ribs not generated");
        for (int tt = 0; tt < Time.Count; tt++)
            if (Time[tt] == t)
            {
                var ans = (0.0D, 0.0D, 0.0D);
                foreach (var solution in additionalFields)
                {
                    var arrCurr = solution.GetElem(x, y, z);
                    if (arrCurr is null) continue;
                    var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
                    double[] qCurr = new double[12];
                    for (int ii = 0; ii < 12; ii++)
                        qCurr[ii] = solution.E[tt][elemCurr[ii]];
                    double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
                    double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
                    double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
                    double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
                    double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
                    double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
                    double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
                    double nuCurr =  (y - y0Curr) / (y1Curr - y0Curr);
                    double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
                    var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr, qCurr);
```

```
                    ans.Item1 += ansCurr.Item1;
                    ans.Item2 += ansCurr.Item2;
                    ans.Item3 += ansCurr.Item3;
                }
                return ans;
            }
        throw new Exception("Out of mesh borders");
    }

    public (double, double, double) GetEAt(double x, double y, double z, double t)
    {
        if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
        ↪  generated");
        for (int tt = 0; tt < Time.Count; tt++)
            if (Time[tt] == t)
            {
                var arr = GetElem(x, y, z);
                if (arr is null) return (0.0D, 0.0D, 0.0D);
                var elem = ConvertGlobalToLocalNumeration(arr);
                double[] q = new double[12];
                for (int i = 0; i < 12; i++)
                    q[i] = E[tt][elem[i]];
                double x0 = ribsArr[elem[0]].a.X;
                double x1 = ribsArr[elem[0]].b.X;
                double y0 = ribsArr[elem[4]].a.Y;
                double y1 = ribsArr[elem[4]].b.Y;
                double z0 = ribsArr[elem[8]].a.Z;
                double z1 = ribsArr[elem[8]].b.Z;
                double eps = (x - x0) / (x1 - x0);
                double nu =  (y - y0) / (y1 - y0);
                double khi = (z - z0) / (z1 - z0);
                var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
                foreach (var solution in additionalFields)
                {
                    var arrCurr = solution.GetElem(x, y, z);
                    if (arrCurr is null) continue;
                    var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
                    double[] qCurr = new double[12];
                    for (int ii = 0; ii < 12; ii++)
                        qCurr[ii] = solution.E[tt][elemCurr[ii]];
                    double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
                    double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
                    double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
                    double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
                    double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
                    double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
                    double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
                    double nuCurr =  (y - y0Curr) / (y1Curr - y0Curr);
                    double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
                    var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
                    ↪  qCurr);
                    ans.Item1 += ansCurr.Item1;
                    ans.Item2 += ansCurr.Item2;
                    ans.Item3 += ansCurr.Item3;
                }
                return ans;
            }
        throw new Exception("Out of mesh borders");
    }

    public (double, double, double) GetBAt(double x, double y, double z, double t)
    {
        if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
        ↪  generated");
        for (int tt = 0; tt < Time.Count; tt++)
            if (Time[tt] == t)
            {
                var arr = GetElem(x, y, z);
                if (arr is null) return (0.0D, 0.0D, 0.0D);
                var elem = ConvertGlobalToLocalNumeration(arr);
                double[] q = new double[12];
                for (int i = 0; i < 12; i++)
                    q[i] = B[tt][elem[i]];
                double x0 = ribsArr[elem[0]].a.X;
                double x1 = ribsArr[elem[0]].b.X;
```

```
                double y0 = ribsArr[elem[4]].a.Y;
                double y1 = ribsArr[elem[4]].b.Y;
                double z0 = ribsArr[elem[8]].a.Z;
                double z1 = ribsArr[elem[8]].b.Z;
                double eps = (x - x0) / (x1 - x0);
                double nu =  (y - y0) / (y1 - y0);
                double khi = (z - z0) / (z1 - z0);
                var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
                foreach (var solution in additionalFields)
                {
                    var arrCurr = solution.GetElem(x, y, z);
                    if (arrCurr is null) continue;
                    var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
                    double[] qCurr = new double[12];
                    for (int ii = 0; ii < 12; ii++)
                        qCurr[ii] = solution.B[tt][elemCurr[ii]];
                    double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
                    double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
                    double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
                    double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
                    double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
                    double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
                    double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
                    double nuCurr =  (y - y0Curr) / (y1Curr - y0Curr);
                    double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
                    var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
                    ↪  qCurr);
                    ans.Item1 += ansCurr.Item1;
                    ans.Item2 += ansCurr.Item2;
                    ans.Item3 += ansCurr.Item3;
                }
                return ans;
            }
        throw new Exception("Out of mesh borders");
    }

    public (double, double, double) GetHAt(double x, double y, double z, double t)
    {
        if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
        ↪  generated");
        for (int tt = 0; tt < Time.Count; tt++)
            if (Time[tt] == t)
            {
                var arr = GetElem(x, y, z);
                if (arr is null) return (0.0D, 0.0D, 0.0D);
                var elem = ConvertGlobalToLocalNumeration(arr);
                double[] q = new double[12];
                for (int i = 0; i < 12; i++)
                    q[i] = H[tt][elem[i]];
                double x0 = ribsArr[elem[0]].a.X;
                double x1 = ribsArr[elem[0]].b.X;
                double y0 = ribsArr[elem[4]].a.Y;
                double y1 = ribsArr[elem[4]].b.Y;
                double z0 = ribsArr[elem[8]].a.Z;
                double z1 = ribsArr[elem[8]].b.Z;
                double eps = (x - x0) / (x1 - x0);
                double nu =  (y - y0) / (y1 - y0);
                double khi = (z - z0) / (z1 - z0);
                var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
                foreach (var solution in additionalFields)
                {
                    var arrCurr = solution.GetElem(x, y, z);
                    if (arrCurr is null) continue;
                    var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
                    double[] qCurr = new double[12];
                    for (int ii = 0; ii < 12; ii++)
                        qCurr[ii] = solution.H[tt][elemCurr[ii]];
                    double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
                    double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
                    double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
                    double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
                    double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
                    double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
                    double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
                    double nuCurr =  (y - y0Curr) / (y1Curr - y0Curr);
```

```
                    double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
                    var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
                    ↪  qCurr);
                    ans.Item1 += ansCurr.Item1;
                    ans.Item2 += ansCurr.Item2;
                    ans.Item3 += ansCurr.Item3;
                }
                return ans;
            }
        throw new Exception("Out of mesh borders");
    }

    // B = rot A
    public void GenerateVectorB()
    {
        if (A is []) throw new Exception("Vector B isn't generated");
        if (ribsArr is null) throw new Exception("Array of ribs isn't generated");
        B = new(A.Count);
        for (int t = 0; t < Time.Count; t++)
        {
            B.Add(new GlobalVector(ribsArr.Count));
            for (int i = 0; i < A[t].Size; i++)
            {
                double ht = Math.Pow(10, -10);
                var pnt = ribsArr[i].GetMiddlePoint();
                var tan = ribsArr[i].GetTangent();
                var vec1 = GetAAt(pnt.X, pnt.Y, pnt.Z + ht, Time[t]);
                var vec2 = GetAAt(pnt.X, pnt.Y, pnt.Z - ht, Time[t]);
                var Bx = -1.0D * (vec1.Item2 - vec2.Item2) / (2.0D * ht);
                vec1 = GetAAt(pnt.X, pnt.Y, pnt.Z + ht, Time[t]);
                vec2 = GetAAt(pnt.X, pnt.Y, pnt.Z - ht, Time[t]);
                var By = (vec1.Item1 - vec2.Item1) / (2.0D * ht);
                vec1 = GetAAt(pnt.X + ht, pnt.Y, pnt.Z, Time[t]);
                vec2 = GetAAt(pnt.X - ht, pnt.Y, pnt.Z, Time[t]);
                var Bz1 = (vec1.Item2 - vec2.Item2) / (2.0D * ht);
                vec1 = GetAAt(pnt.X, pnt.Y + ht, pnt.Z, Time[t]);
                vec2 = GetAAt(pnt.X, pnt.Y - ht, pnt.Z, Time[t]);
                var Bz2 = (vec1.Item1 - vec2.Item1) / (2.0D * ht);
                B[t][i] = Bx * tan.Item1 + By * tan.Item2 + (Bz1 - Bz2) * tan.Item3;
            }
        }
    }

    // H = B / mu_0
    public void GenerateVectorH()
    {
        if (B is []) throw new Exception("Vector B isn't generated");
        H = new(B.Count);
        for (int t = 0; t < B.Count; t++)
        {
            H.Add(new GlobalVector(B[t].Size));
            for (int i = 0; i < B[i].Size; i++)
                H[t][i] = B[t][i] / mu0;
        }
    }

    // E = - dA / dt
    public void GenerateVectorE()
    {
        E = new(A.Count);
        for (int i = 0; i < A.Count; i++)
        {
            if (i == 0)
                E.Add(new GlobalVector(A[1].Size));
            else
            {
                if (A[i - 1].Size == 0) A[i - 1] = new GlobalVector(ribsArr.Count);
                if (A[i].Size == 0) A[i] = new GlobalVector(ribsArr.Count);
                E.Add(-1.0D / (Time[i] - Time[i - 1]) * (A[i] - A[i - 1]));
            }
        }
    }

    public void AddSolution(FEM3D fem) => additionalFields.Add(fem);
```

```
    public List<int>? GetElem(double x, double y, double z)
    {
        if (x < mesh3Dim.nodesX[0] || mesh3Dim.nodesX[^1] < x ||
            y < mesh3Dim.nodesY[0] || mesh3Dim.nodesY[^1] < y ||
            z < mesh3Dim.nodesZ[0] || mesh3Dim.nodesZ[^1] < z)
            return null;
        int i = 0;
        for (; i < mesh3Dim.nodesX.Count - 1; i++)
            if (mesh3Dim.nodesX[i] <= x && x <= mesh3Dim.nodesX[i + 1])
                break;
        int j = 0;
        for (; j < mesh3Dim.nodesY.Count - 1; j++)
            if (mesh3Dim.nodesY[j] <= y && y <= mesh3Dim.nodesY[j + 1])
                break;
        int k = 0;
        for (; k < mesh3Dim.nodesZ.Count - 1; k++)
            if (mesh3Dim.nodesZ[k] <= z && z <= mesh3Dim.nodesZ[k + 1])
                break;
        return elemsArr[k * (mesh3Dim.nodesX.Count - 1) * (mesh3Dim.nodesY.Count - 1) + j * (mesh3Dim.nodesX.Count - 1) + i].Arr;
    }

    public void ConvertResultTo3Dim(FEM2D fem2d)
    {
        if (fem2d.pointsArr is null) throw new ArgumentNullException();
        if (ribsArr is null) throw new ArgumentNullException();
        for (int i = 0; i < Time.Count; i++)
        {
            A.Add(new GlobalVector(ribsArr.Count));
            E.Add(new GlobalVector(ribsArr.Count));
            for (int j = 0; j < ribsArr.Count; j++)
            {
                var X = 0.5D * (ribsArr[j].a.X + ribsArr[j].b.X);
                var Y = 0.5D * (ribsArr[j].a.Y + ribsArr[j].b.Y);
                var Z = 0.5D * (ribsArr[j].a.Z + ribsArr[j].b.Z);
                var antinormal = ((ribsArr[j].b.X - ribsArr[j].a.X) / ribsArr[j].Length,
                                  (ribsArr[j].b.Y - ribsArr[j].a.Y) / ribsArr[j].Length,
                                  (ribsArr[j].b.Z - ribsArr[j].a.Z) / ribsArr[j].Length);
                double fa = 0.0;
                double fe = 0.0;
                var elem = fem2d.GetElem(Math.Sqrt(X * X + Y * Y), Z);
                if (elem is not null)
                {
                    fa = BasisFunctions2D.GetValue(
                                    fem2d.A_phi[i][elem[0]], fem2d.A_phi[i][elem[1]],
                                    fem2d.A_phi[i][elem[2]], fem2d.A_phi[i][elem[3]],
                                    fem2d.pointsArr[elem[0]].R, fem2d.pointsArr[elem[1]].R,
                                    fem2d.pointsArr[elem[0]].Z, fem2d.pointsArr[elem[3]].Z,
                                    Math.Sqrt(X * X + Y * Y), Z);
                    fe = BasisFunctions2D.GetValue(
                                    fem2d.E_phi[i][elem[0]], fem2d.E_phi[i][elem[1]],
                                    fem2d.E_phi[i][elem[2]], fem2d.E_phi[i][elem[3]],
                                    fem2d.pointsArr[elem[0]].R, fem2d.pointsArr[elem[1]].R,
                                    fem2d.pointsArr[elem[0]].Z, fem2d.pointsArr[elem[3]].Z,
                                    Math.Sqrt(X * X + Y * Y), Z);
                }
                var Ax = X == 0 && Y == 0 ? 0.0D : -1.0D * (Y / Math.Sqrt(X * X + Y * Y)) * fa;
                var Ay = X == 0 && Y == 0 ? 0.0D : X / Math.Sqrt(X * X + Y * Y) * fa;
                var Az = 0.0D;
                var Ex = X == 0 && Y == 0 ? 0.0D :  -1.0D * (Y / Math.Sqrt(X * X + Y * Y)) * fe;
                var Ey = X == 0 && Y == 0 ? 0.0D :  X / Math.Sqrt(X * X + Y * Y) * fe;
                var Ez = 0.0D;
                A[i][j] = Ax * antinormal.Item1 + Ay * antinormal.Item2 + Az * antinormal.Item3;
                E[i][j] = Ex * antinormal.Item1 + Ey * antinormal.Item2 + Ez * antinormal.Item3;
            }
        }
    }
```

```csharp

    public void CheckSolution(List<Point3D> recivers)
    {
        using var sw = new StreamWriter("C:\\Users\\USER\\Desktop\\Test.txt");
        GenerateVectorB();
        GenerateVectorH();
        for (int t = 0; t < Time.Count - 1; t++)
        {
            sw.WriteLine($"Time: {Time[t]}:E15");
            foreach (var reciver in recivers)
            {
                var BAt = GetBAt(reciver.X, reciver.Y, reciver.Z, Time[t]);
                var BAt_1 = GetBAt(reciver.X, reciver.Y, reciver.Z, Time[t + 1]);
                var dBdt = ((BAt_1.Item1 - BAt.Item1) / (Time[t + 1] - Time[t]),
                            (BAt_1.Item2 - BAt.Item2) / (Time[t + 1] - Time[t]),
                            (BAt_1.Item3 - BAt.Item3) / (Time[t + 1] - Time[t]));
                var Ex_1 = GetEAt(reciver.X - 1E-10, reciver.Y, reciver.Z, Time[t]);
                var Ex_2 = GetEAt(reciver.X + 1E-10, reciver.Y, reciver.Z, Time[t]);
                var Ey_1 = GetEAt(reciver.X, reciver.Y - 1E-10, reciver.Z, Time[t]);
                var Ey_2 = GetEAt(reciver.X, reciver.Y + 1E-10, reciver.Z, Time[t]);
                var Ez_1 = GetEAt(reciver.X, reciver.Y, reciver.Z - 1E-10, Time[t]);
                var Ez_2 = GetEAt(reciver.X, reciver.Y, reciver.Z + 1E-10, Time[t]);
                var rotE = ((Ey_2.Item3 - Ey_1.Item3) / (2.0 * 1E-10) - (Ez_2.Item2 - Ez_1.Item2) / (2.0 * 1E-10),
                   -1.0D * ((Ex_2.Item3 - Ex_1.Item3) / (2.0 * 1E-10) - (Ez_2.Item1 - Ez_1.Item1) / (2.0 * 1E-10)),
                            (Ex_2.Item2 - Ex_1.Item2) / (2.0 * 1E-10) - (Ey_2.Item1 - Ey_1.Item1) / (2.0 * 1E-10));
                sw.WriteLine($"reciver: ({reciver.X:E15}, {reciver.Y:E15}, {reciver.Z:E15})");
                sw.WriteLine($"  rot E: ({rotE.Item1:E15}, {rotE.Item2:E15}, {rotE.Item3:E15})");
                sw.WriteLine($"   dBdt: ({dBdt.Item1:E15}, {dBdt.Item2:E15}, {dBdt.Item3:E15})");
                sw.WriteLine($"discep.: ({Math.Abs(rotE.Item1 - dBdt.Item1):E15}, {Math.Abs(rotE.Item2 - dBdt.Item2):E15}, {Math.Abs(rotE.Item3 - dBdt.Item3):E15})");
            }
            sw.WriteLine();
        }
        sw.Close();
    }

    public void ConstructMatrixes()
    {
        if (ribsArr is null) throw new ArgumentNullException("ribsArr is null");
        var sparceMatrix = new GlobalMatrix(ribsArr.Count);
        Generator.BuildPortait(ref sparceMatrix, ribsArr.Count, elemsArr, true);
        G = new GlobalMatrix(sparceMatrix);
        Generator.FillMatrixG(ref G, ribsArr, elemsArr);
        M = new GlobalMatrix(sparceMatrix);
        Generator.FillMatrixM(ref M, ribsArr, elemsArr, mesh3Dim, _originalFEM.mesh3Dim);
    }

    public void Solve()
    {
        if (solver is null) throw new ArgumentNullException("Solver is null");
        if (ribsArr is null) throw new ArgumentNullException("ribs array is null");
        if (_originalFEM is null) throw new ArgumentNullException("original E is null");
        if (G is null) throw new ArgumentNullException();
        if (M is null) throw new ArgumentNullException();
        Stopwatch solutionStopwatch = new();
        solutionStopwatch.Start();
        Solutions = new GlobalVector[Time.Count];
        Discrepancy = new GlobalVector[Time.Count];
        (Solutions[0], Discrepancy[0]) = (new GlobalVector(ribsArr.Count), new GlobalVector(ribsArr.Count));
        if (Time.Count > 1)
        {
            (Solutions[1], Discrepancy[1]) = (Solutions[0], Discrepancy[0]);
            for (int i = 1; i < Time.Count; i++)
            {
                Console.WriteLine($"\n {i} / {Time.Count - 1}. Time layer: {Time[i]}");
```

```
                Thread.Sleep(1500);

                double deltT = Time[i] - Time[i - 2];
                double deltT0 = Time[i] - Time[i - 1];
                double deltT1 = Time[i - 1] - Time[i - 2];
                double tau0 = (deltT + deltT0) / (deltT * deltT0);
                double tau1 = deltT / (deltT1 * deltT0);
                double tau2 = deltT0 / (deltT * deltT1);
                double deltT = Time[i] - Time[i - 1];
                double tau = 1.0D / deltT;
                Matrix = G + tau0 * M;
                var b = new GlobalVector(ribsArr.Count);
                Generator.FillVector3D(ref b, _originalFEM.GetEAt, ribsArr, elemsArr, mesh3Dim, _originalFEM.mesh3Dim, Time[i]);
                Vector = b + (tau1 * (M * Solutions[i - 1])) - (tau2 * (M * Solutions[i - 2]));
                Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, ribsArr, bordersArr, Time[i]);
                var localStopWatch = Stopwatch.StartNew();
                (Solutions[i], Discrepancy[i]) = solver.Solve(Matrix, Vector);
                localStopWatch.Stop();
                var localMS = localStopWatch.ElapsedMilliseconds;
                Console.WriteLine($"Current iteration for {localMS / 60000} min {(localMS % 60000) / 1000} sec");
            }
        }
        A = [.. Solutions];
        solutionStopwatch.Stop();
        var milseconds = solutionStopwatch.ElapsedMilliseconds;
        Console.WriteLine($"Lin eq solved for {milseconds / 60000} min {(milseconds % 60000) / 1000} sec");
    }

    public void WriteData(string path)
    {
        if (Solutions is null) throw new ArgumentNullException("No solutions");
        for (int t = 0; t < Time.Count; t++)
        {
            using var sw = new StreamWriter(path + $"Answer_{Time[t]}.txt");
            for (int i = 0; i < A[t].Size; i++)
                sw.WriteLine($"{i} {A[t][i]:E8}");
            sw.Close();
        }
    }

    public void WriteDrawingForFirst(string path)
    {
        double hx = (50000.0D) / 300.0D;
        double hy = (25000.0D) / 300.0D;
        double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 300.0D;

        for (int t = 0; t < Time.Count; t++)
        {
            using var swe = new StreamWriter(path + $"E\\Answer_E_time_layer_{t}.txt");
            for (int k = 0; k < 300; k++)
            {
                for (int i = 0; i < 300; i++)
                {
                    double zCurr = mesh3Dim.nodesZ[0] + k * hz;
                    double xCurr = 0.0D + i * hx;
                    double yCurr = 0.0D + i * hy;
                    var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
                    var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2);
                    var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
                    swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");
                }
            }
            swe.Close();
        }
    }

    public void WriteDrawingForSecond(string path)
    {
        double hx = (50000.0D) / 300.0D;
        double hy = -1.0D * (12500.0D) / 300.0D;
```

```
        double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 300.0D;
        for (int t = 0; t < Time.Count; t++)
        {
            using var swe = new StreamWriter(path + $"E\\Answer_E_time_layer_{t}.txt");
            for (int k = 0; k < 300; k++)
            {
                for (int i = 0; i < 300; i++)
                {
                    double zCurr = mesh3Dim.nodesZ[0] + k * hz;
                    double xCurr = 0.0D + i * hx;
                    double yCurr = 0.0D + i * hy;
                    var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
                    var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2 +
                    ↪  vec.Item3 * vec.Item3);
                    var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
                    swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");                }
            }
            swe.Close();
        }
    }

    public void WriteDataToDraw2DimSolution(string path)
    {
        double hx = (25500.0D) / 300.0D;
        double hy = (25500.0D) / 300.0D;
        double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 300.0D;
        for (int t = 0; t < Time.Count; t++)
        {
            using var swe = new StreamWriter(path + $"E\\Answer_E_time_layer_{t}.txt");
            for (int k = 0; k < 300; k++)
            {
                for (int i = 0; i < 300; i++)
                {
                    double zCurr = mesh3Dim.nodesZ[0] + k * hz;
                    double xCurr = 0.0D + i * hx;
                    double yCurr = 0.0D + i * hy;
                    var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
                    var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2 +
                    ↪  vec.Item3 * vec.Item3);
                    var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
                    swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");
                }
            }
            swe.Close();
        }
    }

    public void WriteDataAtLine(string path)
    {
        double x0 = -1177.5;
        double hy = (mesh3Dim.nodesY[^1] - mesh3Dim.nodesY[0]) / 300.0D;
        double z0 = -1050.0D;
        List<double> times = [Time[0], Time[Time.Count / 2], Time[^1]];
        for (int t = 0; t < times.Count; t++)
        {
            using var swa = new StreamWriter(path + $"A_With_anomaly_{times[t]}.txt");

            for (int i = 0; i < 300; i++)
            {
                double xCurr = x0;
                double yCurr = mesh3Dim.nodesY[0] + i * hy;
                double zCurr = z0;
                var vec = GetAAt(xCurr, yCurr, zCurr, times[t]);
                var ans = vec.Item1;
                swa.WriteLine($"{yCurr:E15} {ans:E15}");
            }
            swa.Close();
        }
        for (int t = 0; t < times.Count; t++)
        {
            using var swe = new StreamWriter(path + $"E_With_anomaly_{times[t]}.txt");
            for (int i = 0; i < 300; i++)
            {
                double xCurr = x0;
                double yCurr = mesh3Dim.nodesY[0] + i * hy;
                double zCurr = z0;
```

```
                var vec = GetEAt(xCurr, yCurr, zCurr, times[t]);
                var ans = vec.Item1;
                swe.WriteLine($"{yCurr:E15} {ans:E15}");
            }
            swe.Close();
        }
    }

    public void WriteDataToDraw(string path)
    {
        double hx = (mesh3Dim.nodesX[^1] - mesh3Dim.nodesX[0]) / 10.0D;
        double hy = (mesh3Dim.nodesY[^1] - mesh3Dim.nodesY[0]) / 10.0D;
        double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 10.0D;
        double x0 = mesh3Dim.nodesX[0];
        double y0 = mesh3Dim.nodesY[0];
        double z0 = mesh3Dim.nodesZ[0];
        List<Point3D> points = [];
        int i = 0;
        int j = 0;
        int k = 0;
        while (z0 + hz * k <= mesh3Dim.nodesZ[^1])
        {
            j = 0;
            while (y0 + hy * j <= mesh3Dim.nodesY[^1])
            {
                i = 0;
                while (x0 + hx * i <= mesh3Dim.nodesX[^1])
                {
                    points.Add(new Point3D(x0 + i * hx, y0 + j * hy, z0 + k * hz));
                    i++;
                }
                j++;
            }
            k++;
        }
        for (int t = 0; t < Time.Count; t++)
        {
            using var sw = new StreamWriter(path + $"Answer{t}.txt");
            sw.WriteLine($"{mesh3Dim.nodesX[0]:E15} {mesh3Dim.nodesX[^1]:E15} {mesh3Dim.nodesY[0]:E15} {mesh3Dim.nodesY[^1]:E15} {mesh3Dim.nodesZ[0]:E15} {mesh3Dim.nodesZ[^1]:E15}");
            foreach (var point in points)
            {
                var ans = GetEAt(point.X, point.Y, point.Z, Time[t]);
                sw.WriteLine($"{point.X:E15} {point.Y:E15} {point.Z:E15} {ans.Item1:E15} {ans.Item2:E15} {ans.Item3:E15}");
            }
            sw.Close();
        }
    }

    public void ReadData(string AnswerPath)
    {
        A = new(Time.Count);
        string file = string.Empty;
        for (int t = 0; t < Time.Count; t++)
        {
            file = $"Answer_{Time[t]}.txt";
            var fileData = File.ReadAllText(AnswerPath + file).Split("\n");
            A.Add(new GlobalVector(fileData.Length - 1));
            for (int i = 0; i < fileData.Length - 1; i++)
            {
                var val = fileData[i].Split(" ")[1];
                A[t][i] = double.Parse(val);
            }
        }
    }

    public void TestOutput(string path)
    {
        using var sw = new StreamWriter(path + "/A_phi/Answer3D/Answer_Test.txt");
        var absDiscX = 0.0D;
        var absDiscY = 0.0D;
        var absDiscZ = 0.0D;
        var absDivX = 0.0D;
        var absDivY = 0.0D;
```

```
        var absDivZ = 0.0D;
        var relDiscX = 0.0D;
        var relDiscY = 0.0D;
        var relDiscZ = 0.0D;
        var relDivX = 0.0D;
        var relDivY = 0.0D;
        var relDivZ = 0.0D;
        int iter = 0;
        foreach (Elem elem in elemsArr)
        {
            int[] elem_local = [elem[0], elem[3], elem[8], elem[11],
                                elem[1], elem[2], elem[9], elem[10],
                                elem[4], elem[5], elem[6], elem[7]];
            var x = 0.5D * (ribsArr[elem_local[0]].a.X + ribsArr[elem_local[0]].b.X);
            var y = 0.5D * (ribsArr[elem_local[4]].a.Y + ribsArr[elem_local[4]].b.Y);
            var z = 0.5D * (ribsArr[elem_local[8]].a.Z + ribsArr[elem_local[8]].b.Z);
            sw.WriteLine($"Points {x:E15} {y:E15} {z:E15}");
            var eps = (x - ribsArr[elem_local[0]].a.X) / (ribsArr[elem_local[0]].b.X - ribsArr[elem_local[0]].a.X);
            var nu =  (y - ribsArr[elem_local[4]].a.Y) / (ribsArr[elem_local[4]].b.Y - ribsArr[elem_local[4]].a.Y);
            var khi = (z - ribsArr[elem_local[8]].a.Z) / (ribsArr[elem_local[8]].b.Z - ribsArr[elem_local[8]].a.Z);
            double[] q = [Solutions[0][elem_local[0]], Solutions[0][elem_local[1]], Solutions[0][elem_local[2]], Solutions[0][elem_local[3]],
                          Solutions[0][elem_local[4]], Solutions[0][elem_local[5]], Solutions[0][elem_local[6]], Solutions[0][elem_local[7]],
                          Solutions[0][elem_local[8]], Solutions[0][elem_local[9]], Solutions[0][elem_local[10]], Solutions[0][elem_local[11]]];
            var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
            var theorValue = Function.A(x, y, z, 0.0D);
            sw.WriteLine($"FEM A  {ans.Item1:E15} {ans.Item2:E15} {ans.Item3:E15}");
            sw.WriteLine($"Theor  {theorValue.Item1:E15} {theorValue.Item2:E15} {theorValue.Item3:E15}");
            var currAbsDiscX = Math.Abs(ans.Item1 - theorValue.Item1);
            var currAbsDiscY = Math.Abs(ans.Item2 - theorValue.Item2);
            var currAbsDiscZ = Math.Abs(ans.Item3 - theorValue.Item3);
            var currRelDiscX = currAbsDiscX / Math.Abs(theorValue.Item1);
            var currRelDiscY = currAbsDiscY / Math.Abs(theorValue.Item2);
            var currRelDiscZ = currAbsDiscZ / Math.Abs(theorValue.Item3);
            sw.WriteLine($"CurrAD {currAbsDiscX:E15} {currAbsDiscY:E15} {currAbsDiscZ:E15}");
            sw.WriteLine($"CurrRD {currRelDiscX:E15} {currRelDiscY:E15} {currRelDiscZ:E15}\n");
            absDiscX += currAbsDiscX;
            absDiscY += currAbsDiscY;
            absDiscZ += currAbsDiscZ;
            absDivX += theorValue.Item1;
            absDivY += theorValue.Item2;
            absDivZ += theorValue.Item3;
            relDiscX += currRelDiscX * currRelDiscX;
            relDiscY += currRelDiscY * currRelDiscY;
            relDiscZ += currRelDiscZ * currRelDiscZ;
            relDivX += theorValue.Item1 * theorValue.Item1;
            relDivY += theorValue.Item2 * theorValue.Item2;
            relDivZ += theorValue.Item3 * theorValue.Item3;
            iter++;
        }
        sw.WriteLine($"Avg disc: {absDiscX / iter:E15} {absDiscY / iter:E15} {absDiscZ / iter:E15}");
        sw.WriteLine($"Rel disc: {Math.Sqrt(relDiscX / relDivX):E15} {Math.Sqrt(relDiscY / relDivY):E15} {Math.Sqrt(relDiscZ / relDivZ):E15}");
        sw.Close();
    }

    public void MeasureValuesOnReceiversWithoutNormal(List<Point3D> recivers, string OutputPath)
    {
        using var sw_e = new StreamWriter(OutputPath + "E.txt");
        for (int t = 0; t < Time.Count; t++)
        {
            var a_e = AdditioanalGetEAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
            var b_e = AdditioanalGetEAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
```

```
            var c_e = AdditioanalGetEAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
            var d_e = AdditioanalGetEAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
            var f_a_e = Math.Sqrt(a_e.Item1 * a_e.Item1 + a_e.Item2 * a_e.Item2 + a_e.Item3 * a_e.Item3);
            var f_b_e = Math.Sqrt(b_e.Item1 * b_e.Item1 + b_e.Item2 * b_e.Item2 + b_e.Item3 * b_e.Item3);
            var f_c_e = Math.Sqrt(c_e.Item1 * c_e.Item1 + c_e.Item2 * c_e.Item2 + c_e.Item3 * c_e.Item3);
            var f_d_e = Math.Sqrt(d_e.Item1 * d_e.Item1 + d_e.Item2 * d_e.Item2 + d_e.Item3 * d_e.Item3);
            sw_e.WriteLine($"{Time[t]} {f_a_e} {f_b_e} {f_c_e} {f_d_e}");
        }
        sw_e.Close();
    }

    public void MeasureValuesOnReceivers(List<Point3D> recivers, string OutputPath)
    {
        using var sw_a = new StreamWriter(OutputPath + "A.txt");
        using var sw_e = new StreamWriter(OutputPath + "E.txt");
        for (int t = 0; t < Time.Count; t++)
        {
            var a_a = GetAAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
            var b_a = GetAAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
            var c_a = GetAAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
            var d_a = GetAAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
            var f_a_a = Math.Sqrt(a_a.Item1 * a_a.Item1 + a_a.Item2 * a_a.Item2 + a_a.Item3 * a_a.Item3);
            var f_b_a = Math.Sqrt(b_a.Item1 * b_a.Item1 + b_a.Item2 * b_a.Item2 + b_a.Item3 * b_a.Item3);
            var f_c_a = Math.Sqrt(c_a.Item1 * c_a.Item1 + c_a.Item2 * c_a.Item2 + c_a.Item3 * c_a.Item3);
            var f_d_a = Math.Sqrt(d_a.Item1 * d_a.Item1 + d_a.Item2 * d_a.Item2 + d_a.Item3 * d_a.Item3);
            var a_e = GetEAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
            var b_e = GetEAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
            var c_e = GetEAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
            var d_e = GetEAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
            var f_a_e = Math.Sqrt(a_e.Item1 * a_e.Item1 + a_e.Item2 * a_e.Item2 + a_e.Item3 * a_e.Item3);
            var f_b_e = Math.Sqrt(b_e.Item1 * b_e.Item1 + b_e.Item2 * b_e.Item2 + b_e.Item3 * b_e.Item3);
            var f_c_e = Math.Sqrt(c_e.Item1 * c_e.Item1 + c_e.Item2 * c_e.Item2 + c_e.Item3 * c_e.Item3);
            var f_d_e = Math.Sqrt(d_e.Item1 * d_e.Item1 + d_e.Item2 * d_e.Item2 + d_e.Item3 * d_e.Item3);
            sw_a.WriteLine($"{Time[t]} {f_a_a} {f_b_a} {f_c_a} {f_d_a}");
            sw_e.WriteLine($"{Time[t]} {f_a_e} {f_b_e} {f_c_e} {f_d_e}");
        }
        sw_a.Close();
        sw_e.Close();
    }
}
```

Mesh.cs

```
using DataStructs;

namespace Grid;

public abstract class Mesh(List<Elem> elems) : ICloneable
{
    public abstract int ElemsAmount { get; set; }
    internal int bordersAmount;
    public List<Elem> Elems = elems;
    public abstract int NodesAmountTotal { get; }
    public abstract object Clone();
}
```

Mesh2Dim.cs

```csharp
using System.Collections.Immutable;
using DataStructs;

namespace Grid;

public class Mesh2Dim(List<double> nodesR, string infoAboutR,
                      List<double> nodesZ, string infoAboutZ,
                      List<Elem> elems, double lastR) : Mesh(elems)
{
    public override int NodesAmountTotal => NodesAmountR * NodesAmountZ;
    public List<double> nodesR = [.. nodesR, lastR];
    public List<double> nodesZ = nodesZ;
    public override int ElemsAmount
    {
        get => (NodesAmountR - 1) * (NodesAmountZ - 1);
        set => ElemsAmount = value;
    }
    public int NodesAmountR => nodesR.Count;
    public int NodesAmountZ => nodesZ.Count;
    public List<int> NodesR_Refs = [0];
    public List<int> NodesZRefs = [0];
    internal ImmutableArray<double> NodesRWithoutFragmentation { get; set; } = [.. nodesR, lastR];
    public ImmutableArray<double> NodesZWithoutFragmentation { get; set; } = [.. nodesZ];
    internal string infoAboutR = infoAboutR;
    internal string infoAboutZ = infoAboutZ;
    internal List<Border2D> borders = [];
    public void SetLastR(double val)
    {
        int i = 1;
        if (val <= nodesR[^i])
        {
            while (val < nodesR[^i]) i++;
            nodesR[^i] = val;
            nodesR = nodesR.Take(i + 1).ToList();
        }
        else
            nodesR.Add(val);
    }

    public void SetBorders(List<Border3D> borders3D)
    {
        // Set lower border.
        borders.Add(new Border2D(borders3D[0].BorderType, borders3D[0].BorderFormula,
                                 0, NodesRWithoutFragmentation.Length - 1, 0, 0));
        // Set left border.
        borders.Add(new Border2D(borders3D[1].BorderType, borders3D[1].BorderFormula,
                                 0, 0, 0, NodesZWithoutFragmentation.Length - 1));
        // Set right border.
        borders.Add(new Border2D(borders3D[3].BorderType, borders3D[3].BorderFormula,
                                 NodesRWithoutFragmentation.Length - 1,
                                 NodesRWithoutFragmentation.Length - 1,
                                 0, NodesZWithoutFragmentation.Length - 1));
        // Set upper border.
        borders.Add(new Border2D(borders3D[^1].BorderType, borders3D[^1].BorderFormula,
                                 0, NodesRWithoutFragmentation.Length - 1,
                                 NodesZWithoutFragmentation.Length - 1,
                                 NodesZWithoutFragmentation.Length - 1));
    }

    public override Mesh2Dim Clone() => (Mesh2Dim)MemberwiseClone();
}
```

Mesh3Dim.cs

```csharp
using System.Collections.Immutable;
using DataStructs;

namespace Grid;

```

```
public class Mesh3Dim(List<double> nodesX, string infoAboutX,
                      List<double> nodesY, string infoAboutY,
                      List<double> nodesZ, string infoAboutZ,
                      List<Elem> elems, List<Border3D> borders) : Mesh(elems)
{
    public override int NodesAmountTotal
    {
        get => NodesAmountX * NodesAmountY * NodesAmountZ;
    }

    public override int ElemsAmount
    {
        get => (NodesAmountX - 1) * (NodesAmountY - 1) * (NodesAmountZ - 1);
        set => ElemsAmount = value;
    }

    public int NodesAmountX => nodesX.Count;
    public int NodesAmountY => nodesY.Count;
    public int NodesAmountZ => nodesZ.Count;
    public List<double> nodesX = nodesX;
    public List<double> nodesY = nodesY;
    public List<double> nodesZ = nodesZ;
    public void CommitFieldBorders(List<int> ints) => FieldBorders = ints;
    public void CommitAnomalyBorders(List<int> ints) => AnomalyBorders = ints;
    public void CommitSecondAnomalyBorders(List<int> ints) => SecondAnomaly = ints;
    public List<int> FieldBorders = [];
    public List<int> AnomalyBorders = [];
    public List<int> SecondAnomaly = [];
    internal ImmutableArray<double> NodesXWithoutFragmentation { get; set; } = [..
     nodesX];
    internal ImmutableArray<double> NodesYWithoutFragmentation { get; set; } = [..
     nodesY];
    public ImmutableArray<double> NodesZWithoutFragmentation { get; set; } = [.. nodesZ];
    internal string infoAboutX = infoAboutX;
    internal string infoAboutY = infoAboutY;
    internal string infoAboutZ = infoAboutZ;
    public List<int> NodesXRefs = [0];
    public List<int> NodesYRefs = [0];
    public List<int> NodesZRefs = [0];
    public List<Border3D> borders = borders;
    public ArrayOfRibs? arrayOfRibs;
    public override Mesh3Dim Clone() => (Mesh3Dim)MemberwiseClone();
}
```

LocalMatrix3D.cs

```
namespace MathObjects;

public class LocalMatrixG3D (double mu, double hx, double hy, double hz) : Matrix
{
    private readonly double _mu = mu;
    private readonly double _hx = hx;
    private readonly double _hy = hy;
    private readonly double _hz = hz;
    private readonly double[,] G1 = {{ 2.0D,  1.0D, -2.0D, -1.0D},
                                     { 1.0D,  2.0D, -1.0D, -2.0D},
                                     {-2.0D, -1.0D,  2.0D,  1.0D},
                                     {-1.0D, -2.0D,  1.0D,  2.0D}};
    private readonly double[,] G2 = {{ 2.0D, -2.0D,  1.0D, -1.0D},
                                     {-2.0D,  2.0D, -1.0D,  1.0D},
                                     { 1.0D, -1.0D,  2.0D, -2.0D},
                                     {-1.0D,  1.0D, -2.0D,  2.0D}};
    private readonly double[,] G3 = {{-2.0D,  2.0D, -1.0D,  1.0D},
                                     {-1.0D,  1.0D, -2.0D,  2.0D},
                                     { 2.0D, -2.0D,  1.0D, -1.0D},
                                     { 1.0D, -1.0D,  2.0D, -2.0D}};

    public override double this[int i, int j]
    {
        get
        {
            return (i / 4, j / 4)
            switch
```

```
            {
                (0, 0) => (_hx * _hy / (6.0D * _hz) * G1[i % 4, j % 4] + _hx * _hz / (6.0D * _hy) * G2[i % 4, j % 4]) / _mu,
                (0, 1) or (1, 0) => -1.0D * (_hz / 6.0D) * G2[i % 4, j % 4] / _mu,
                (0, 2) => _hy / 6.0D * G3[i % 4, j % 4] / _mu,
                (1, 1) => (_hx * _hy / (6.0D * _hz) * G1[i % 4, j % 4] + _hy * _hz / (6.0D * _hx) * G2[i % 4, j % 4]) / _mu,
                (1, 2) or (2, 1) => -1.0D * _hx / 6.0D * G1[i % 4, j % 4] / _mu,
                (2, 0) => _hy / 6.0D * G3[j % 4, i % 4] / _mu,
                (2, 2) => (_hx * _hz / (6.0D * _hy) * G1[i % 4, j % 4] + _hy * _hz / (6.0D * _hx) * G2[i % 4, j % 4]) / _mu,
                _ => throw new ArgumentOutOfRangeException("Out of local matrix 3d range"),
            };
        }
        set{}
    }
}

public class LocalMatrixM3D(double gamma, double hx, double hy, double hz) : Matrix
{
    private readonly double _gamma = gamma;
    private readonly double _hx = hx;
    private readonly double _hy = hy;
    private readonly double _hz = hz;
    private readonly double[,] D = {{4.0D, 2.0D, 2.0D, 1.0D},
                                    {2.0D, 4.0D, 1.0D, 2.0D},
                                    {2.0D, 1.0D, 4.0D, 2.0D},
                                    {1.0D, 2.0D, 2.0D, 4.0D}};

    public override double this[int i, int j]
    {
        get
        {
            return (i / 4, j / 4)
            switch
            {
                (0, 0) or (1, 1) or (2, 2) => _gamma * _hx * _hy * _hz / 36.0D * D[i % 4, j % 4],
                (0, 1) or (0, 2) or (1, 0) or (1, 2) or (2, 0) or (2, 1) => 0.0D,
                _ => throw new ArgumentOutOfRangeException("Out of local matrix 3d range"),
            };
        }
        set
        {}
    }
}
```

LocalVector3D.cs

```
using static Functions.Function;

namespace MathObjects;


public class LocalVector3D(Egetter egetter, double x0, double x1, double y0, double y1, double z0, double z1, double t) : Vector
{
    private readonly double x0 = x0;
    private readonly double x1 = x1;
    private readonly double y0 = y0;
    private readonly double y1 = y1;
    private readonly double z0 = z0;
    private readonly double z1 = z1;
    private readonly double xm = 0.5D * (x1 + x0);
    private readonly double ym = 0.5D * (y1 + y0);
    private readonly double zm = 0.5D * (z1 + z0);
    private readonly double t = t;

    private readonly LocalMatrixM3D M = new(1.0, x1 - x0, y1 - y0, z1 - z0);
    private readonly Egetter _egetter = egetter;

```

```
    public override double this[int i]
    {
        get
        {
            static double ScalarMult((double, double, double) a, (double, double, double)
            ↪ b)
            => a.Item1 * b.Item1 + a.Item2 * b.Item2 + a.Item3 * b.Item3;
            List<double> vect = [
                ScalarMult(_egetter(xm, y0, z0, t), (1.0D, 0.0D, 0.0D)),
                ScalarMult(_egetter(xm, y1, z0, t), (1.0D, 0.0D, 0.0D)),
                ScalarMult(_egetter(xm, y0, z1, t), (1.0D, 0.0D, 0.0D)),
                ScalarMult(_egetter(xm, y1, z1, t), (1.0D, 0.0D, 0.0D)),
                ScalarMult(_egetter(x0, ym, z0, t), (0.0D, 1.0D, 0.0D)),
                ScalarMult(_egetter(x1, ym, z0, t), (0.0D, 1.0D, 0.0D)),
                ScalarMult(_egetter(x0, ym, z1, t), (0.0D, 1.0D, 0.0D)),
                ScalarMult(_egetter(x1, ym, z1, t), (0.0D, 1.0D, 0.0D)),
                ScalarMult(_egetter(x0, y0, zm, t), (0.0D, 0.0D, 1.0D)),
                ScalarMult(_egetter(x1, y0, zm, t), (0.0D, 0.0D, 1.0D)),
                ScalarMult(_egetter(x0, y1, zm, t), (0.0D, 0.0D, 1.0D)),
                ScalarMult(_egetter(x1, y1, zm, t), (0.0D, 0.0D, 1.0D))
            ];
            double ans = 0.0D;
            for (int j = 0; j < vect.Count; j++)
                ans += M[i, j] * vect[j];
            return ans;
        }
    }
}
```

MathOperations.cs

```
using System.Threading.Tasks;

namespace MathObjects;

public static class MathOperations
{
    public static GlobalVector Diff(GlobalVector gv1, GlobalVector gv2)
    {
        if (gv1.Size != gv2.Size)
            throw new Exception("Найти разность векторов не возможно, т.к. они имеют
            ↪ разные размеры.");

        GlobalVector gv = new(gv1.Size);
        for (int i = 0; i < gv.Size; i++)
            gv[i] = gv1[i] - gv2[i];

        return gv;
    }

    public static GlobalVector DiffPar(GlobalVector gv1, GlobalVector gv2)
    {
        if (gv1.Size != gv2.Size)
            throw new Exception("Найти разность векторов не возможно, т.к. они имеют
            ↪ разные размеры.");
        GlobalVector gv = new(gv1.Size);
        Parallel.For(0, gv1.Size, i => {
            gv[i] = gv1[i] - gv2[i];
        });
        return gv;
    }

    public static GlobalVector Sum(GlobalVector gv1, GlobalVector gv2)
    {
        if (gv1.Size != gv2.Size)
            throw new Exception("Найти разность векторов не возможно, т.к. они имеют
            ↪ разные размеры.");

        GlobalVector gv = new(gv1.Size);
        for (int i = 0; i < gv.Size; i++)
            gv[i] = gv1[i] + gv2[i];

```

```
        return gv;
    }

    public static GlobalVector SumPar(GlobalVector gv1, GlobalVector gv2)
    {
        if (gv1.Size != gv2.Size)
            throw new Exception("Найти разность векторов не возможно, т.к. они имеют разные размеры.");

        GlobalVector gv = new(gv1.Size);
        Parallel.For(0, gv1.Size, i => {
            gv[i] = gv1[i] + gv2[i];
        });
        return gv;
    }

    public static GlobalVector Multiply(double a, GlobalVector gv)
    {
        GlobalVector _gv = new(gv.Size);
        for (int i = 0; i < gv.Size; i++)
            _gv[i] = a * gv[i];
        return _gv;
    }

    public static GlobalVector MultiplyPar(double a, GlobalVector gv)
    {
        GlobalVector _gv = new(gv.Size);
        Parallel.For(0, gv.Size, i => {
            _gv[i] = a * gv[i];
        });
        return _gv;
    }

    public static double Multiply(GlobalVector gv1, GlobalVector gv2)
    {
        if (gv1.Size != gv2.Size) throw new Exception("Невозможно найти скалярное умножение векторов, из-за разности в размерах.");
        double ans = 0.0D;
        for (int i = 0; i < gv1.Size; i++)
            ans += gv1[i] * gv2[i];
        return ans;
    }

    public static double MultiplyPar(GlobalVector gv1, GlobalVector gv2)
    {
        if (gv1.Size != gv2.Size) throw new Exception("Невозможно найти скалярное умножение векторов, из-за разности в размерах.");
        double ans = 0.0D;
        Parallel.For(0, gv1.Size, i => {
            ans = gv1[i] * gv2[i];
        });
        return ans;
    }


    public static GlobalVector Multiply(GlobalMatrix _gm, GlobalVector _gv)
    {
        if (_gm.Size != _gv.Size)
            throw new Exception("Невозможно перемножить матрицу на вектор.");
        GlobalVector ans = new(_gv.Size);

        for (int i = 0; i < _gv.Size; i++)
        {
            for (int j = 0; j < _gm._ig[i + 1] - _gm._ig[i]; j++)
            {
                ans[i] += _gm._al[_gm._ig[i] + j] * _gv[_gm._jg[_gm._ig[i] + j]];
                ans[_gm._jg[_gm._ig[i] + j]] += _gm._au[_gm._ig[i] + j] * _gv[i];
            }
            ans[i] += _gm._diag[i] * _gv[i];
        }
        return ans;
    }

    public static GlobalVector CustomMultiply(GlobalMatrix _gm, GlobalVector _gv)
    {
        if (_gm.Size != _gv.Size)
```

```
            throw new Exception("Невозможно перемножить матрицу на вектор.");
        GlobalVector ans = new(_gv.Size);

        for (int i = 0; i < _gv.Size; i++)
        {
            for (int j = _gm._ig[i]; j < _gm._ig[i + 1]; j++)
            {
                ans[i] += _gm._al[j] * _gv[_gm._jg[j]];
                ans[_gm._jg[j]] += _gm._au[j] * _gv[i];
            }
            ans[i] += _gm._diag[i] * _gv[i];
        }
        return ans;
    }


    public static GlobalVector MultiplyPar(GlobalMatrix _gm, GlobalVector _gv)
    {
        if (_gm.Size != _gv.Size)
            throw new Exception("Невозможно перемножить матрицу на вектор.");
        GlobalVector ans = new(_gv.Size);

        for (int i = 0; i < _gv.Size; i++)
        {
            Parallel.For(0, _gm._ig[i + 1] - _gm._ig[i], j => {
                ans[i] += _gm._al[_gm._ig[i] + j] * _gv[_gm._jg[_gm._ig[i] + j]];
            });
            Parallel.For(0, _gm._ig[i + 1] - _gm._ig[i], j => {
                ans[_gm._jg[_gm._ig[i] + j]] += _gm._au[_gm._ig[i] + j] * _gv[i];
            });
            ans[i] += _gm._diag[i] * _gv[i];
        }
        return ans;
    }


    public static GlobalMatrix Multiply(double a, GlobalMatrix gm)
    {
        var ans = new GlobalMatrix(gm);
        for (int i = 0; i < ans.Size; i++)
        {
            for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
            {
                ans._al[ans._ig[i] + j] *= a;
                ans._au[ans._ig[i] + j] *= a;
            }
            ans._diag[i] *= a;
        }
        return ans;
    }


    public static GlobalMatrix MultiplyPar(double a, GlobalMatrix gm)
    {
        var ans = new GlobalMatrix(gm);
        Parallel.For(0, ans.Size, i => {
            for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
            {
                ans._al[ans._ig[i] + j] *= a;
                ans._au[ans._ig[i] + j] *= a;
            }
            ans._diag[i] *= a;
        });
        return ans;
    }


    public static GlobalMatrix Sum(GlobalMatrix gm1, GlobalMatrix gm2)
    {
        if (!gm1.CheckPortrait(gm2)) throw new ArgumentException("Different matrixes
         ↪ portrait!");
        GlobalMatrix ans = new(gm1);
        for (int i = 0; i < ans.Size; i++)
        {
            for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
            {
```

```
                ans._al[ans._ig[i] + j] += gm2._al[gm2._ig[i] + j];
                ans._au[ans._ig[i] + j] += gm2._au[gm2._ig[i] + j];
            }
            ans._diag[i] += gm2._diag[i];
        }
        return ans;
    }


    public static GlobalMatrix SumPar(GlobalMatrix gm1, GlobalMatrix gm2)
    {
        if (!gm1.CheckPortrait(gm2)) throw new ArgumentException("Different matrixes
        ↪  portrait!");
        GlobalMatrix ans = new(gm1);

        Parallel.For(0, ans.Size, i => {
            for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
            {
                ans._al[ans._ig[i] + j] += gm2._al[gm2._ig[i] + j];
                ans._au[ans._ig[i] + j] += gm2._au[gm2._ig[i] + j];
            }
            ans._diag[i] += gm2._diag[i];
        });
        return ans;
    }
}
```

LULOS.cs

```
public class LU_LOS(int maxIter = 100_000, double eps = 1E-15) : ISolver
{
    private int _maxIter = maxIter;
    private double _eps = eps;

    private static void PartitialLU(GlobalMatrix A)
    {
        for (int i = 0; i < A.Size; i++)
        {
            for (int j = A._ig[i]; j < A._ig[i + 1]; j++)
            {
                int jCol = A._jg[j];
                int jk = A._ig[jCol];
                int k = A._ig[i];
                int sdvig = A._jg[A._ig[i]] - A._jg[A._ig[jCol]];
                if (sdvig > 0) jk += sdvig;
                else k -= sdvig;
                double sumL = 0.0;
                double sumU = 0.0;
                for (; k < j && jk < A._ig[jCol + 1]; k++, jk++)
                {
                    sumL += A._al[k] * A._au[jk];
                    sumU += A._au[k] * A._al[jk];
                }
                A._al[j] -= sumL;
                A._au[j] -= sumU;
                A._au[j] /= A._diag[jCol];
            }
            double sumD = 0.0;
            for (int j = A._ig[i]; j < A._ig[i + 1]; j++)
                sumD += A._al[j] * A._au[j];
            A._diag[i] -= sumD;
        }
    }

    private static GlobalVector Forward(GlobalMatrix Matrix, GlobalVector b)
    {
        var result = new GlobalVector(b);
        for (int i = 0; i < Matrix.Size; i++)
        {
            for (int j = Matrix._ig[i]; j < Matrix._ig[i + 1]; j++)
                result[i] -= Matrix._al[j] * result[Matrix._jg[j]];
            result[i] /= Matrix._diag[i];
```

```
        }
        return result;
    }

    private static GlobalVector Backward(GlobalMatrix A, GlobalVector b)
    {
        var result = new GlobalVector(b);
        for (int i = A.Size - 1; i >= 0; i--)
            for (int j = A._ig[i + 1] - 1; j >= A._ig[i]; j--)
                result[A._jg[j]] -= A._au[j] * result[i];
        return result;
    }

    public (GlobalVector, GlobalVector) Solve(GlobalMatrix A, GlobalVector b)
    {
        GlobalVector x = new(b.Size);
        GlobalVector x_ = new(b.Size);
        GlobalMatrix LU = new(A);
        PartitialLU(LU);
        GlobalVector r = Forward(LU, b - A * x);
        var r0 = new GlobalVector(r);
        GlobalVector z = Backward(LU, r);
        GlobalVector p = Forward(LU, A * z);
        GlobalVector tmp = new(b.Size);
        GlobalVector r_ = new(b.Size);
        GlobalVector z_ = new(b.Size);
        GlobalVector p_ = new(b.Size);
        double alph = 0.0D;
        double beta = 0.0D;
        int iter = 0;
        do
        {
            x_ = x;
            z_ = z;
            r_ = r;
            p_ = p;
            alph = (p_ * r_) / (p_ * p_);
            x = x_ + alph * z_;
            r = r_ - alph * p_;
            tmp = Forward(LU, A * Backward(LU, r));
            beta = -1.0D * (p_ * tmp) / (p_ * p_);
            z = Backward(LU, r) + beta * z_;
            p = tmp + beta * p_;
            iter++;
            if (iter % 10 == 0)
                Console.WriteLine($"{(r.Norma() * r.Norma()) / (r0.Norma() * r0.Norma()):E15}");
        } while (iter < _maxIter && (r.Norma() * r.Norma()) / (r0.Norma() * r0.Norma()) >= _eps * _eps);
        Console.WriteLine(
        $@"Computing finished!
Total iterations: {iter}
Relative residuality: {r.Norma() / b.Norma():E15}");
        return (x, x_);
    }
}
```