СОДЕРЖАНИЕ

ЗАДАНИЕ

Разработка программы для моделирования трехмерного электромагнитного поля на шестигранниках с использованием векторного МКЭ.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. ВЕКТОРНЫЕ ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ ВТОРОГО ПОРЯДКА С РАЗРЫВНЫМИ РЕШЕНИЯМИ

Математическая модель, служащая для описания электромагнитного поля в средах с изменяющимся коэффициентом магнитной проницаемости и в ситуациях, когда нельзя пренебрегать влиянием токов смещения, выглядит следующим образом (1.1):

$$\mathrm{rot}\left(\frac{1}{\mu}\mathrm{rot}\overrightarrow{\mathbf{A}}\right)+\sigma\frac{\partial\overrightarrow{\mathbf{A}}}{\partial t}+\epsilon\frac{\partial^2\overrightarrow{\mathbf{A}}}{\partial t^2}=\overrightarrow{\mathbf{J}}^{\mathbf{ст}}. \tag{1.1}$$

Математическая модель электромагнитного поля на основе уравнения (1.1) позволяет решать самые сложные задачи электромагнетизма. Она корректно описывает электромагнитные поля в ситуациях, когда среда содержит любые неоднородности с измененными электрическими и магнитными свойствами.

При решении задач с использованием схемы разделения полей, для описания осесимметричной горизонтально-слоистой среды используется следующее уравнение (1.2):

$$-\frac{1}{\mu_0}\Delta A_\varphi+\frac{A_\varphi}{\mu_0 r^2}+\sigma\frac{\partial A_\varphi}{\partial t}=J_\varphi. \tag{1.2}$$

В свою очередь, учёт от объектов, имеющих неоднородные значения удельной электропроводности, осуществляется за счёт математической модели, описываемой уравнением (1.3)

$$\mathrm{rot}\left(\frac{1}{\mu_0}\mathrm{rot}\overrightarrow{\mathbf{A}}^{+}\right)+\sigma\frac{\partial\overrightarrow{\mathbf{A}}^{+}}{\partial t}=(\sigma-\sigma_{\mathrm{n}})\,\overrightarrow{\mathbf{E}}_{\mathrm{n}}. \tag{1.3}$$

Для тестирования на правильность решения дифференциального уравнения (1.3) будем использовать уравнение (1.4), правая часть которого представляется в виде вектор-функции $\overrightarrow{\mathbf{F}}$, а также будет иметь место быть слагаемое $\gamma\overrightarrow{\mathbf{A}}$ в левой части уравнения:

$$\mathrm{rot}\left(\frac{1}{\mu_0}\mathrm{rot}\overrightarrow{\mathbf{A}}\right)+\gamma\overrightarrow{\mathbf{A}}+\sigma\frac{\partial\overrightarrow{\mathbf{A}}}{\partial t}=\overrightarrow{\mathbf{F}}. \tag{1.4}$$

1.2. ВАРИАЦИОННАЯ ПОСТАНОВКА

Будем считать, что на границе $S=S_1\cup S_2$ расчётной области Ω, в которой определено уравнение (1.4), заданы краевые условия двух типов:

$$\left(\overrightarrow{\mathbf{A}}\times\overrightarrow{\mathbf{n}}\right)\Big|_{S_1}=\overrightarrow{\mathbf{A}}^{g}\times\overrightarrow{\mathbf{n}}, \tag{1.5}$$

$$\left(\frac{1}{\mu}\mathrm{rot}\overrightarrow{\mathbf{A}}\times\overrightarrow{\mathbf{n}}\right)\Big|_{S_1}=\overrightarrow{\mathbf{H}}^{\Theta}\times\overrightarrow{\mathbf{n}}. \tag{1.6}$$

Тогда эквивалентная вариационная формулировка в форме Галёркина для уравнения (1.4) без производной по времени, и с учётом краевых условий (1.5) - (1.6) имеет вид:

$$\begin{gathered}\int\limits_{\Omega}\frac{1}{\mu_0}\mathrm{rot}\overrightarrow{\mathbf{A}}\cdot\mathrm{rot}\overrightarrow{\Psi}\,d\Omega+\int\limits_{\Omega}\gamma\overrightarrow{\mathbf{A}}\cdot\overrightarrow{\Psi}\,d\Omega=\int\limits_{\Omega}\overrightarrow{\mathbf{F}}\cdot\overrightarrow{\Psi}\,d\Omega+\\+\int\limits_{S_2}\left(\overrightarrow{\mathbf{H}}^{\Theta}\times\overrightarrow{\mathbf{n}}\right)\cdot\overrightarrow{\Psi}\,dS\qquad\forall\overrightarrow{\Psi}\in H_0^{rot}.\end{gathered} \tag{1.7}$$

1.3. КОНЕЧНОЭЛЕМЕНТНАЯ ДИСКРЕТИЗАЦИЯ

На шестиграннике базисные вектор-функции удобней строить с помощью шаблонного элемента. Обычно в качестве такого берут кубик $[-1,1] \times [-1,1] \times [-1,1]$ при использовании базиса лагранжева или иерархического типа.

Пусть у нас имеется произвольный шестигранник Ω_m с вершинами $(\hat{x}_i, \hat{y}_i, \hat{z}_i)\,, i = 1...8$. Тогда отображение шаблонного кубика Ω^E в шестигранник Ω_m будет задаваться соотношениями:

$$x = \sum_{i=1}^{8} \hat{\varphi}_i(\xi,\eta,\zeta)\hat{x}_i, \qquad y = \sum_{i=1}^{8} \hat{\varphi}_i(\xi,\eta,\zeta)\hat{y}_i, \qquad z = \sum_{i=1}^{8} \hat{\varphi}_i(\xi,\eta,\zeta)\hat{z}_i, \quad (1.8)$$

где $\hat{\varphi}_i(\xi,\eta,\zeta)$ - стандартные скалярные трилинейные базисные функции, определённые на шаблонном элементе Ω^E.

Отображение базисных вектор-функций $\hat{\varphi}_i(\xi,\eta,\zeta)$ шаблонного элемента Ω^E на шестигранник Ω_m можно определить следующим образом:

$$\hat{\psi}_i(x,y,z) = \mathbf{J}^{-1}\hat{\varphi}_i(\xi(x,y,z),\eta(x,y,z),\zeta(x,y,z)), \quad (1.9)$$

где

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{pmatrix} \quad (1.10)$$

- функциональная матрица преобразования координат, переводящего кубик Ω^E в шестигранник Ω_m.

1.4. ПОСТРОЕНИЕ МАТРИЦ МАСС, ЖЁСТКОСТИ И ВЕКТОРА ПРАВОЙ ЧАСТИ НА ШЕСТИГРАННИКАХ

В силу сложной геометрии выпуклых шестигранников, расчёт локальных матриц удобнее проводить на отображении конечного элемента Ω_m в мастер-элемент Ω_E, представляющий из себя куб размером $\Omega_E = [-1,1]_x \times [-1,1]_y \times [-1,1]_z$. Тогда матрица жёсткости будет рассчитываться по формуле (1.11):

$$\begin{gathered}\hat{\mathrm{G}}_{ij} = \int\limits_{\Omega_e} \frac{1}{\mu_0} \mathrm{rot}\hat{\psi}_i \cdot \mathrm{rot}\hat{\psi}_j d\Omega = \\ = \int\limits_{-1}^{1}\int\limits_{-1}^{1}\int\limits_{-1}^{1} \frac{1}{\mu_0}\frac{1}{|J|} \left(\mathbf{J}^{\mathrm{T}}\mathrm{rot}\hat{\varphi}_i\right) \cdot \left(\mathbf{J}^{\mathrm{T}}\mathrm{rot}\hat{\varphi}_j\right) \, d\xi d\eta d\zeta\end{gathered} \tag{1.11}$$

матрица масс, в свою очередь, по формуле (1.12):

$$\hat{\mathrm{M}}_{ij} = \int\limits_{\Omega_e} \gamma\hat{\psi}_i \cdot \hat{\psi}_j d\Omega = \int\limits_{-1}^{1}\int\limits_{-1}^{1}\int\limits_{-1}^{1} \gamma \left(\mathbf{J}^{\text{-}1}\hat{\varphi}_i\right) \cdot \left(\mathbf{J}^{\text{-}1}\hat{\varphi}_j\right) |J| \, d\xi d\eta d\zeta \tag{1.12}$$

При расчёте локального вектора правой части будем использовать формулу:

$$\hat{\mathbf{b}}^{\mathbf{J^{CT}}} = \hat{\mathbf{M}}\,\hat{\mathbf{f}}. \tag{1.13}$$

она универсальна и применима для любой геометрии конечного элемента и любыми базисными функциями любого порядка на нём.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. ГЕНЕРАЦИЯ ТРЁХМЕРНОЙ СЕТКИ С ЯЧЕЙКАМИ В ВИДЕ ШЕСТИГРАННИКОВ

При написании программы был использован следующий подход к построению сетки на шестигранных элементах.

```
1. [lines amount x] 2 [lines amount y] 2 [lines amount z] 2
2. [field description of points]
3. 0.0 0.0 0.0     1.0 0.0 0.0
4. 0.0 1.0 0.0     1.0 1.0 0.0
5. 0.0 0.0 1.0     1.0 0.0 1.0
6. 0.0 1.0 1.0     1.0 1.0 1.0
7. [unique areas amount] 1
8. [unique areas description]
9. 1 0 1 0 1 0 1
10. [unique areas coefficients description]
11. 1 1.0 1.0
12. [delimiters above X description] 1 1.0
13. [delimiters above Y description] 3 1.1
14. [delimiters above Z description] 4 0.8
15. [borders amount] 6
16. [borders description]
17. 1 1 0 1 0 0 0 1
18. 1 1 0 1 1 1 0 1
19. 1 1 0 0 0 1 0 1
20. 1 1 1 1 0 1 0 1
```

```
21. 1 1 0 1 0 1 0 0
22. 1 1 0 1 0 1 1 1
```

В первой строке заданы количество опорных узлов N_x^W, N_y^W, N_z^W, базовой плоскости по осям X, Y, Z соответственно. С третьей по шестую строки перечисленны тройки чисел $(x_i,\ y_i,\ z_i)$ - как раз и определяющие эти опорные узлы.

В седьмой строке указано количество уникальных областей в расчётной области, которые имеют определённые уникальные значения физических параметров μ и σ. Начиная с девятой строки (в общем случае должен быть построчный перечень каждой области) описывается геометрическое расположение i - ой области. В одиннадцатой строке указаны уникальные значения параметров μ и σ для i - ой области.

В строках с двенадцатой по четырнадцатую описывается количество и характер необходимых разбиений для осей X, Y, Z соответственно.

В пятнадцатой строке целочисленным значением задаётся количество границ. Далее с семнадцатой по двадцать вторую строки описывается расположение и характер этих границ. Первым числом задаётся тип краевого условия (т.е. принимает значения 1 или 2), вторым числом задаётся номер формулы, третьим первая координатная линия по оси X, четвёртым вторая координатная линия по оси X, пятым и шестым аналогично по оси Y и седьмым и восьмым по оси Z.

Пример расчётной области этой фигуры изображён на рисунке (2.1).

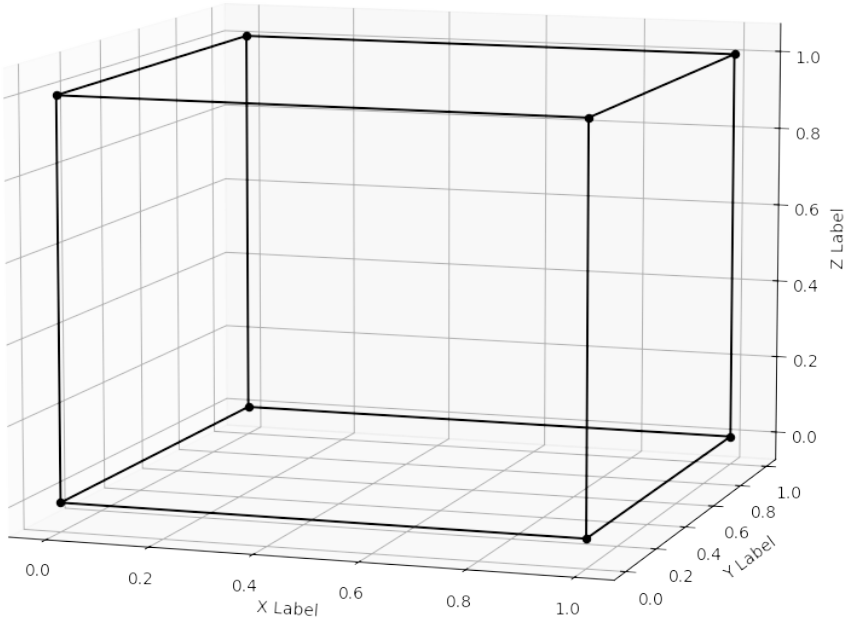


Рисунок 2.1 – Расчетная область для кубика

Попробуем подробить расчётную область (2.1) на несколько частей. Получим сетку изображённую на рисунке (2.2).

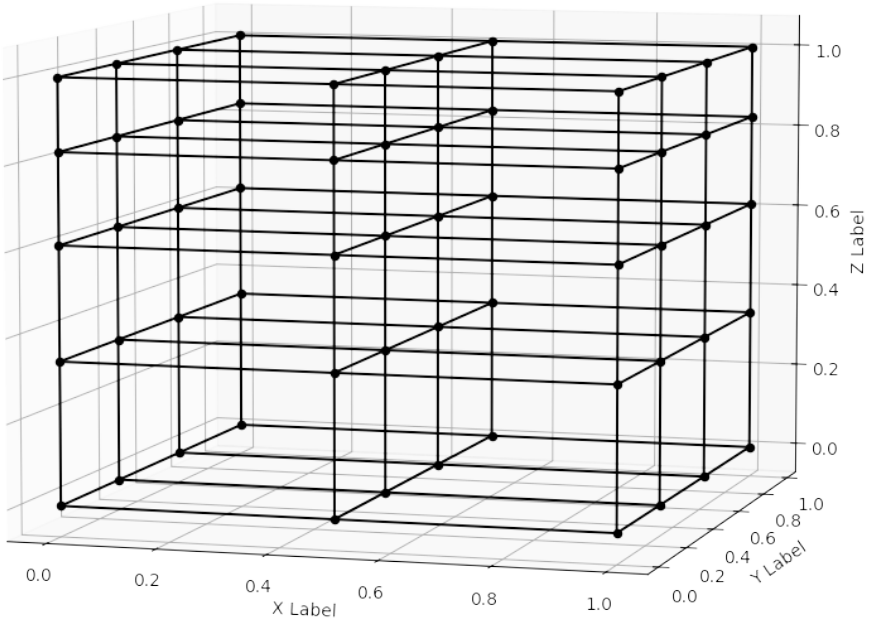
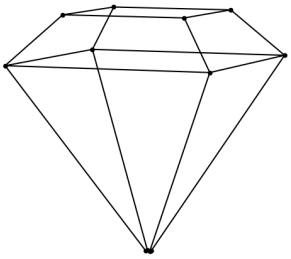
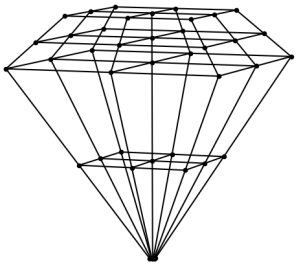


Рисунок 2.2 – Секта для кубика

Приведём ещё насколько примеров для построения сеток на шестигранниках, изображённых на рисунках 2.3 - 2.7.

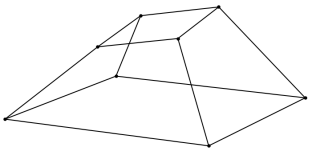


а)

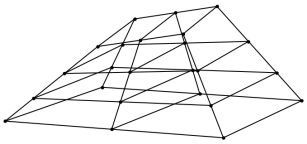


б)

Рисунок 2.3 – Расчётная область в форме изумруда (а) и сетка к ней (б).

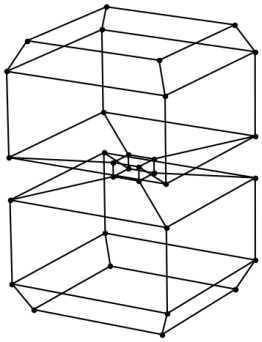


а)

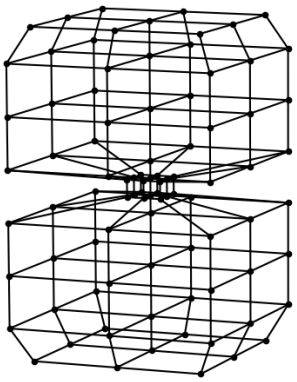


б)

Рисунок 2.4 – Расчётная область в форме скошенной пирамиды (а) и сетка к ней (б).

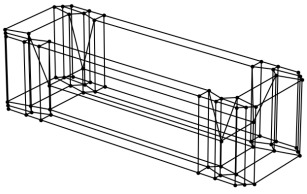


а)

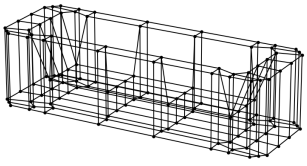


б)

Рисунок 2.5 – Расчётная область в форме песочных часов (а) и сетка к ней (б).

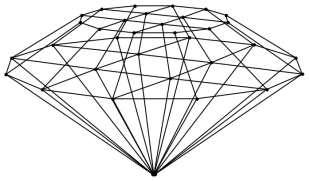


а)

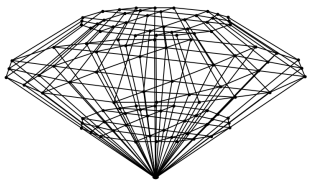


б)

Рисунок 2.6 – Расчётная область в форме ванной (а) и сетка к ней (б).



а)



б)

Рисунок 2.7 – Расчётная область в форме детализированного изумруда (а) и сетка к ней (б).

Таким образом, программа для построения сетки может строить достаточно геометрически сложные фигуры.

2.2. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

При расчёте элементов локальных матриц жёсткости (1.11) и масс (1.12) будем использовать численное интегрирование методами Гаусса разных порядков (2, 3, 4, 5). Результаты численного интегрирования на некоторых функциях приведены в таблицах 2.1 - 2.7. Область интегрирования для всех функций единый: $\Omega_E \in [-1;1]_x \times [-1;1]_y \times [-1;1]_z$.

Таблица 2.1 – Тестирование численного интегрирования на функции $u = 2$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
16.0	1.6000000e+01	1.6000000e+01	1.6000000e+01	1.6000000e+01

Таблица 2.2 – Тестирование численного интегрирования на функции $u = x + y + z$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
0.0	0.0000000e+00	-2.2204460e-16	5.6898930e-16	-6.5225603e-16

Таблица 2.3 – Тестирование численного интегрирования на функции $u = x^2 + y^2 + z^2$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
8.0	8.0000000e+00	8.0000000e+00	8.0000000e+00	8.0000000e+00

Таблица 2.4 – Тестирование численного интегрирования на функции $u = x \cdot y \cdot z$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
0.0	8.0000000e+00	0.0000000e+00	0.0000000e+00	8.6736174e-18

Таблица 2.5 – Тестирование численного интегрирования на функции $u = x^2 \cdot y^2 \cdot z^2$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
$\frac{8}{27} \approx 0.29630$	2.9629630e-01	2.9629630e-01	2.9629630e-01	2.9629630e-01

Таблица 2.6 – Тестирование численного интегрирования на функции $u = \cos(x + y + z)$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
4.7666...	4.7063579e+00	4.7671091e+00	4.7665835e+00	4.7665859e+00

Таблица 2.7 – Тестирование численного интегрирования на функции $u = e^{x+y+z}$.

Аналитический результат	Гаусс 2	Гаусс 3	Гаусс 4	Гаусс 5
12.9845...	1.2857243e+01	1.2983458e+01	1.2984538e+01	1.2984543e+01

2.3. РЕШЕНИЕ СЛАУ

Через LOS или Pardiso.

2.4. ТЕСТИРОВАНИЕ ТРЁХМЕРНОЙ ЗАДАЧИ НА ПОЛИНОМИАЛЬНЫХ ВЕКТОР-ФУНКЦИЯХ

Проведем сначала тестирование разработанной программы по векторному МКЭ на работоспособность. Образец расчетной области изображен на рисунке **??**. 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.

Тестирование будем проводить дифференциального уравнения (2.1):

$$\mathrm{rot}\left(\frac{1}{\mu}\mathrm{rot}\overrightarrow{\mathbf{A}}\right) + \gamma\overrightarrow{\mathbf{A}} + \sigma\frac{\partial\overrightarrow{\mathbf{A}}}{\partial t} = \overrightarrow{\mathbf{F}}. \tag{2.1}$$

В таблицах 2.8 – 2.16 приведено тестирование на работоспособность программы. Для искомых $\overrightarrow{\mathbf{A}}$ будем выводить значения функции в центрах рёбер сетки, отмеченных красным цветом на рисунке **??**.

Таблица 2.8 – 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 2.9 – 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 2.10 – 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 2.11 – 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 2.12 – 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 2.13 – 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 2.14 – 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 2.15 – 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 2.16 – 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 **??**.

В таблицах 2.17 – 2.18 представлены результаты тестирования для постоянной и линейной вектор-функциях.

Таблица 2.17 – 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 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(1.5; 0.5; 0.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(0.5; 1.5; 0.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(1.5; 1.5; 0.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(0.5; 0.5; 1.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(1.5; 0.5; 1.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(0.5; 1.5; 1.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000
$(1.5; 1.5; 1.5)$	1.00000000E+000 1.00000000E+000 1.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000	0.00000000E+000 0.00000000E+000 0.00000000E+000

Таблица 2.18 – 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
$(0.5; 0.5; 0.5)$	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 **??**. 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 2.19 – 2.21.

Таблица 2.19 – 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 2.20 – 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 2.21 – 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. ИССЛЕДОВАНИЯ

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];
```

```
25          E_phi = new GlobalVector[Time.Count];
26          Debug.WriteLine("Generated data submited");
27      }
28
29      private readonly Mesh2Dim mesh2Dim;
30      public GlobalVector[] A_phi;
31      public GlobalVector[] E_phi;
32
33      public void Solve()
34      {
35          if (solver is null) throw new ArgumentNullException("solver is null !");
36          if (Time is null) throw new ArgumentNullException("Time is null!");
37
38          Stopwatch solutionStopwatch = new();
39          solutionStopwatch.Start();
40
41          Debug.WriteLine($"\nTime layer: before BC");
42          Thread.Sleep(1500);
43
44          Matrix = new GlobalMatrix(pointsArr.GetLength());
45          Generator.BuildPortait(ref Matrix, pointsArr.GetLength(), elemsArr);
46          Generator.FillMatrix(ref Matrix, pointsArr, elemsArr, TypeOfMatrixM.Mrr);
47
48          Vector = new GlobalVector(pointsArr.GetLength());
49          Generator.FillVector(ref Vector, pointsArr, elemsArr, Time[0]);
50
51          Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, pointsArr,
            ↪  bordersArr, Time[0]);
52          (Solutions[0], Discrepancy[0]) = solver.Solve(Matrix, Vector);
53
54          if (Time.Count > 1)
55          {
56              (Solutions[1], Discrepancy[1]) = (Solutions[0], Discrepancy[0]);
57              if (Time.Count > 2)
58              for (int i = 2; i < Time.Count; i++)
59              {
60                  Console.WriteLine($"\n {i} / {Time.Count - 1}. Time layer: {Time[i]}");
61                  //Thread.Sleep(1500);
62
63                  double deltT = Time[i] - Time[i - 2];
64                  double deltT0 = Time[i] - Time[i - 1];
65                  double deltT1 = Time[i - 1] - Time[i - 2];
66                  double tau0 = (deltT + deltT0) / (deltT * deltT0);
67                  double tau1 = deltT / (deltT1 * deltT0);
68                  double tau2 = deltT0 / (deltT * deltT1);
69                  double deltT = Time[i] - Time[i - 1];
70                  double tau = 1.0D / deltT;
71
72                  var matrix1 = new GlobalMatrix(pointsArr.GetLength());
73                  Generator.BuildPortait(ref matrix1, pointsArr.GetLength(), elemsArr);
74                  Generator.FillMatrix(ref matrix1, pointsArr, elemsArr, TypeOfMatrixM.Mrr);
75                  var M = new GlobalMatrix(pointsArr.GetLength()); // ???
76                  Generator.BuildPortait(ref M, pointsArr.GetLength(), elemsArr);
77                  Generator.FillMatrix(ref M, pointsArr, elemsArr, TypeOfMatrixM.Mr);
78
79                  var bi = new GlobalVector(pointsArr.GetLength());
80                  Matrix = (tau0 * M) + matrix1;
81                  Vector = bi + (tau1 * (M * Solutions[i - 1])) - (tau2 * (M * Solutions[i
                    ↪  - 2]));
82                  Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, pointsArr,
                    ↪  bordersArr, Time[i]);
83                  (Solutions[i], Discrepancy[i]) = solver.Solve(Matrix, Vector);
84              }
85          }
86          A_phi = Solutions;
87          solutionStopwatch.Stop();
88          var milseconds = solutionStopwatch.ElapsedMilliseconds;
89          Console.WriteLine($"Lin eq solved for {milseconds / 60000} min {(milseconds %
            ↪  60000) / 1000} sec");
90      }
91
92      public void WriteData()
93      {
94          if (Answer is null)
95              throw new Exception("Vector _answer is null");
```

```
96          for (int i = 0; i < Answer.Size; i++)
97              Console.WriteLine($"{Answer[i]:E15}");
98      }
99
100     public void WriteData(string _path)
101     {
102         if (A_phi is null) throw new ArgumentNullException();
103         if (E_phi is null) throw new ArgumentNullException();
104
105         if (Time.Count != 1)
106         {
107             for (int i = 0; i < Time.Count; i++)
108             {
109                 using var sw = new StreamWriter($"{_path}\\A_phi\\Answer\\Answer_Aphi_tim
                    ↪  e={Time[i]}.dat");
110                 for (int j = 0;   j < A_phi[i].Size; j++)
111                     sw.WriteLine($"{A_phi[i][j]:E8}");
112                 sw.Close();
113             }
114             for (int i = 0; i < Time.Count; i++)
115             {
116                 using var sw = new StreamWriter($"{_path}\\E_phi\\Answer\\Answer_Ephi_tim
                    ↪  e={Time[i]}.dat");
117                 for (int j = 0;   j < E_phi[i].Size; j++)
118                     sw.WriteLine($"{E_phi[i][j]:E8}");
119                 sw.Close();
120             }
121         }
122         else
123         {
124             using var sw = new StreamWriter($"{_path}\\A_phi\\Answer\\Answer.dat");
125             for (int j = 0; j < A_phi[0].Size; j++)
126                 sw.WriteLine($"{A_phi[0][j]:E8}");
127             sw.Close();
128             using var sw1 = new StreamWriter($"{_path}\\E_phi\\Answer\\Answer.dat");
129             for (int j = 0; j < E_phi[0].Size; j++)
130                 sw1.WriteLine($"{E_phi[0][j]:E8}");
131             sw1.Close();
132         }
133     }
134
135     public void WriteDiscrepancy(string _path)
136     {
137         if (A_phi is null) throw new ArgumentNullException();
138         if (E_phi is null) throw new ArgumentNullException();
139
140         if (Time.Count != 1)
141         {
142             for (int i = 0; i < Time.Count; i++)
143             {
144                 using var sw_d = new StreamWriter($"{_path}\\A_phi\\Discrepancy\\Discrepa
                    ↪  ncy_Aphi_time={Time[i]}.dat");
145
146                 int NotNaNamount = 0;
147                 double maxDisc = 0.0;
148                 double avgDisc = 0.0;
149                 double sumU = 0.0D;
150                 double sumD = 0.0D;
151
152                 List<double> TheorAnswer = [];
153                 foreach (var Z in mesh2Dim.nodesZ)
154                     foreach (var R in mesh2Dim.nodesR)
155                         TheorAnswer.Add(Function.U(R, Z, Time[i]));
156
157                 for (int j = 0; j < A_phi[i].Size; j++)
158                 {
159                     double absDiff = Math.Abs(A_phi[i][j] - TheorAnswer[j]);
160                     double currDisc = Math.Abs((A_phi[i][j] - TheorAnswer[j]) /
                        ↪  TheorAnswer[j]);
161
162                     if (Math.Abs(maxDisc) < Math.Abs(currDisc))
163                         maxDisc = currDisc;
164
165                     if (!double.IsNaN(currDisc) && currDisc > 1E-14)
166                     {
```

```
                        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");
```

```
311                 A_phi[t] = new GlobalVector(fileData.Length - 1);
312                 for (int i = 0; i < fileData.Length - 1; i++)
313                     A_phi[t][i] = double.Parse(fileData[i]);
314                 file = $"Answer_Ephi_time={Time[t]}.dat";
315                 fileData = File.ReadAllText(AnswerPath + "E_phi/Answer/" +
                    ↪  file).Split("\n");
316                 E_phi[t] = new GlobalVector(fileData.Length - 1);
317                 for (int i = 0; i < fileData.Length - 1; i++)
318                     E_phi[t][i] = double.Parse(fileData[i]);
319             }
320         }
321     }
322
323     public void MeasureValuesOnReceivers(List<Point3D> recivers, string OutputPath)
324     {
325         using var sw_a = new StreamWriter(OutputPath + "A.txt");
326         using var sw_e = new StreamWriter(OutputPath + "E.txt");
327         List<(double, double)> pnt2D = [];
328         foreach (var reciver in recivers)
329             pnt2D.Add((Math.Sqrt(reciver.X * reciver.X + reciver.Y * reciver.Y),
                ↪  reciver.Z));
330         for (int t = 0; t < Time.Count; t++)
331         {
332             var a_a = GetA_phiAt(pnt2D[0].Item1, pnt2D[0].Item2, Time[t]);
333             var b_a = GetA_phiAt(pnt2D[1].Item1, pnt2D[1].Item2, Time[t]);
334             var c_a = GetA_phiAt(pnt2D[2].Item1, pnt2D[2].Item2, Time[t]);
335             var d_a = GetA_phiAt(pnt2D[3].Item1, pnt2D[3].Item2, Time[t]);
336             var a_e = GetE_phiAt(pnt2D[0].Item1, pnt2D[0].Item2, Time[t]);
337             var b_e = GetE_phiAt(pnt2D[1].Item1, pnt2D[1].Item2, Time[t]);
338             var c_e = GetE_phiAt(pnt2D[2].Item1, pnt2D[2].Item2, Time[t]);
339             var d_e = GetE_phiAt(pnt2D[3].Item1, pnt2D[3].Item2, Time[t]);
340             sw_a.WriteLine($"{Time[t]} {a_a} {b_a} {c_a} {d_a}");
341             sw_e.WriteLine($"{Time[t]} {a_e} {b_e} {c_e} {d_e}");
342         }
343         sw_a.Close();
344         sw_e.Close();
345     }
346
347     public void WritePointsToDraw(string pathA, string pathE)
348     {
349         double hr = (mesh2Dim.nodesR[^1] - mesh2Dim.nodesR[0]) / 150;
350         double hz = (mesh2Dim.nodesZ[^1] - mesh2Dim.nodesZ[0]) / 150;
351         for (int t = 0; t < Time.Count; t++)
352         {
353             using var swa = new StreamWriter(pathA + $"Answer_A_time_layer_{t}.txt");
354             for (int j = 0; j < 150; j++)
355             {
356                 for (int i = 0; i < 150; i++)
357                 {
358                     double rCurr = mesh2Dim.nodesR[0] + i * hr;
359                     double zCurr = mesh2Dim.nodesZ[0] + j * hz;
360                     swa.WriteLine($"{rCurr:E15} {zCurr:E15} {GetA_phiAt(rCurr, zCurr,
                        ↪  Time[t]):E15}");
361                 }
362             }
363             swa.Close();
364         }
365         for (int t = 0; t < Time.Count; t++)
366         {
367             using var swe = new StreamWriter(pathE + $"Answer_E_time_layer_{t}.txt");
368             for (int j = 0; j < 150; j++)
369             {
370                 for (int i = 0; i < 150; i++)
371                 {
372                     double rCurr = mesh2Dim.nodesR[0] + i * hr;
373                     double zCurr = mesh2Dim.nodesZ[0] + j * hz;
374                     swe.WriteLine($"{rCurr:E15} {zCurr:E15} {GetE_phiAt(rCurr, zCurr,
                        ↪  Time[t]):E15}");
375                 }
376             }
377             swe.Close();
378         }
379     }
380 }
```

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);

```

```
361     public List<int>? GetElem(double x, double y, double z)
362     {
363         if (x < mesh3Dim.nodesX[0] || mesh3Dim.nodesX[^1] < x ||
364             y < mesh3Dim.nodesY[0] || mesh3Dim.nodesY[^1] < y ||
365             z < mesh3Dim.nodesZ[0] || mesh3Dim.nodesZ[^1] < z)
366             return null;
367         int i = 0;
368         for (; i < mesh3Dim.nodesX.Count - 1; i++)
369             if (mesh3Dim.nodesX[i] <= x && x <= mesh3Dim.nodesX[i + 1])
370                 break;
371         int j = 0;
372         for (; j < mesh3Dim.nodesY.Count - 1; j++)
373             if (mesh3Dim.nodesY[j] <= y && y <= mesh3Dim.nodesY[j + 1])
374                 break;
375         int k = 0;
376         for (; k < mesh3Dim.nodesZ.Count - 1; k++)
377             if (mesh3Dim.nodesZ[k] <= z && z <= mesh3Dim.nodesZ[k + 1])
378                 break;
379         return elemsArr[k * (mesh3Dim.nodesX.Count - 1) * (mesh3Dim.nodesY.Count - 1) + j
    ↪  * (mesh3Dim.nodesX.Count - 1) + i].Arr;
380     }
381 
382     public void ConvertResultTo3Dim(FEM2D fem2d)
383     {
384         if (fem2d.pointsArr is null) throw new ArgumentNullException();
385         if (ribsArr is null) throw new ArgumentNullException();
386         for (int i = 0; i < Time.Count; i++)
387         {
388             A.Add(new GlobalVector(ribsArr.Count));
389             E.Add(new GlobalVector(ribsArr.Count));
390             for (int j = 0; j < ribsArr.Count; j++)
391             {
392                 var X = 0.5D * (ribsArr[j].a.X + ribsArr[j].b.X);
393                 var Y = 0.5D * (ribsArr[j].a.Y + ribsArr[j].b.Y);
394                 var Z = 0.5D * (ribsArr[j].a.Z + ribsArr[j].b.Z);
395                 var antinormal = ((ribsArr[j].b.X - ribsArr[j].a.X) / ribsArr[j].Length,
396                                   (ribsArr[j].b.Y - ribsArr[j].a.Y) / ribsArr[j].Length,
397                                   (ribsArr[j].b.Z - ribsArr[j].a.Z) / ribsArr[j].Length);
398                 double fa = 0.0;
399                 double fe = 0.0;
400                 var elem = fem2d.GetElem(Math.Sqrt(X * X + Y * Y), Z);
401                 if (elem is not null)
402                 {
403                     fa = BasisFunctions2D.GetValue(
404                                   fem2d.A_phi[i][elem[0]], fem2d.A_phi[i][elem[1]],
405                                   fem2d.A_phi[i][elem[2]], fem2d.A_phi[i][elem[3]],
406                                   fem2d.pointsArr[elem[0]].R,
                                      ↪  fem2d.pointsArr[elem[1]].R,
407                                   fem2d.pointsArr[elem[0]].Z,
                                      ↪  fem2d.pointsArr[elem[3]].Z,
408                                   Math.Sqrt(X * X + Y * Y), Z);
409                     fe = BasisFunctions2D.GetValue(
410                                   fem2d.E_phi[i][elem[0]], fem2d.E_phi[i][elem[1]],
411                                   fem2d.E_phi[i][elem[2]], fem2d.E_phi[i][elem[3]],
412                                   fem2d.pointsArr[elem[0]].R,
                                      ↪  fem2d.pointsArr[elem[1]].R,
413                                   fem2d.pointsArr[elem[0]].Z,
                                      ↪  fem2d.pointsArr[elem[3]].Z,
414                                   Math.Sqrt(X * X + Y * Y), Z);
415                 }
416                 var Ax = X == 0 && Y == 0 ? 0.0D : -1.0D * (Y / Math.Sqrt(X * X + Y * Y))
                    ↪  * fa;
417                 var Ay = X == 0 && Y == 0 ? 0.0D : X / Math.Sqrt(X * X + Y * Y) * fa;
418                 var Az = 0.0D;
419                 var Ex = X == 0 && Y == 0 ? 0.0D :  -1.0D * (Y / Math.Sqrt(X * X + Y *
                    ↪  Y)) * fe;
420                 var Ey = X == 0 && Y == 0 ? 0.0D :  X / Math.Sqrt(X * X + Y * Y) * fe;
421                 var Ez = 0.0D;
422                 A[i][j] = Ax * antinormal.Item1 + Ay * antinormal.Item2 + Az *
                    ↪  antinormal.Item3;
423                 E[i][j] = Ex * antinormal.Item1 + Ey * antinormal.Item2 + Ez *
                    ↪  antinormal.Item3;
424             }
425         }
426     }
```

```
427
428     public void CheckSolution(List<Point3D> recivers)
429     {
430         using var sw = new StreamWriter("C:\\Users\\USER\\Desktop\\Test.txt");
431         GenerateVectorB();
432         GenerateVectorH();
433         for (int t = 0; t < Time.Count - 1; t++)
434         {
435             sw.WriteLine($"Time: {Time[t]}:E15");
436             foreach (var reciver in recivers)
437             {
438                 var BAt = GetBAt(reciver.X, reciver.Y, reciver.Z, Time[t]);
439                 var BAt_1 = GetBAt(reciver.X, reciver.Y, reciver.Z, Time[t + 1]);
440                 var dBdt = ((BAt_1.Item1 - BAt.Item1) / (Time[t + 1] - Time[t]),
441                             (BAt_1.Item2 - BAt.Item2) / (Time[t + 1] - Time[t]),
442                             (BAt_1.Item3 - BAt.Item3) / (Time[t + 1] - Time[t]));
443                 var Ex_1 = GetEAt(reciver.X - 1E-10, reciver.Y, reciver.Z, Time[t]);
444                 var Ex_2 = GetEAt(reciver.X + 1E-10, reciver.Y, reciver.Z, Time[t]);
445                 var Ey_1 = GetEAt(reciver.X, reciver.Y - 1E-10, reciver.Z, Time[t]);
446                 var Ey_2 = GetEAt(reciver.X, reciver.Y + 1E-10, reciver.Z, Time[t]);
447                 var Ez_1 = GetEAt(reciver.X, reciver.Y, reciver.Z - 1E-10, Time[t]);
448                 var Ez_2 = GetEAt(reciver.X, reciver.Y, reciver.Z + 1E-10, Time[t]);
449                 var rotE = ((Ey_2.Item3 - Ey_1.Item3) / (2.0 * 1E-10) - (Ez_2.Item2 -
                    ↪  Ez_1.Item2) / (2.0 * 1E-10),
450                    -1.0D * ((Ex_2.Item3 - Ex_1.Item3) / (2.0 * 1E-10) - (Ez_2.Item1 -
                       ↪  Ez_1.Item1) / (2.0 * 1E-10)),
451                             (Ex_2.Item2 - Ex_1.Item2) / (2.0 * 1E-10) - (Ey_2.Item1 -
                             ↪  Ey_1.Item1) / (2.0 * 1E-10));
452                 sw.WriteLine($"reciver: ({reciver.X:E15}, {reciver.Y:E15},
                    ↪  {reciver.Z:E15})");
453                 sw.WriteLine($"  rot E: ({rotE.Item1:E15}, {rotE.Item2:E15},
                    ↪  {rotE.Item3:E15})");
454                 sw.WriteLine($"   dBdt: ({dBdt.Item1:E15}, {dBdt.Item2:E15},
                    ↪  {dBdt.Item3:E15})");
455                 sw.WriteLine($"discep.: ({Math.Abs(rotE.Item1 - dBdt.Item1):E15},
                    ↪  {Math.Abs(rotE.Item2 - dBdt.Item2):E15}, {Math.Abs(rotE.Item3 -
                    ↪  dBdt.Item3):E15})");
456             }
457             sw.WriteLine();
458         }
459         sw.Close();
460     }
461
462     public void ConstructMatrixes()
463     {
464         if (ribsArr is null) throw new ArgumentNullException("ribsArr is null");
465         var sparceMatrix = new GlobalMatrix(ribsArr.Count);
466         Generator.BuildPortait(ref sparceMatrix, ribsArr.Count, elemsArr, true);
467         G = new GlobalMatrix(sparceMatrix);
468         Generator.FillMatrixG(ref G, ribsArr, elemsArr);
469         M = new GlobalMatrix(sparceMatrix);
470         Generator.FillMatrixM(ref M, ribsArr, elemsArr, mesh3Dim, _originalFEM.mesh3Dim);
471     }
472
473     public void Solve()
474     {
475         if (solver is null) throw new ArgumentNullException("Solver is null");
476         if (ribsArr is null) throw new ArgumentNullException("ribs array is null");
477         if (_originalFEM is null) throw new ArgumentNullException("original E is null");
478         if (G is null) throw new ArgumentNullException();
479         if (M is null) throw new ArgumentNullException();
480         Stopwatch solutionStopwatch = new();
481         solutionStopwatch.Start();
482         Solutions = new GlobalVector[Time.Count];
483         Discrepancy = new GlobalVector[Time.Count];
484         (Solutions[0], Discrepancy[0]) = (new GlobalVector(ribsArr.Count), new
            ↪  GlobalVector(ribsArr.Count));
485         if (Time.Count > 1)
486         {
487             (Solutions[1], Discrepancy[1]) = (Solutions[0], Discrepancy[0]);
488             for (int i = 1; i < Time.Count; i++)
489             {
490                 Console.WriteLine($"\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;
```

```
561             double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 300.0D;
562             for (int t = 0; t < Time.Count; t++)
563             {
564                 using var swe = new StreamWriter(path + $"E\\Answer_E_time_layer_{t}.txt");
565                 for (int k = 0; k < 300; k++)
566                 {
567                     for (int i = 0; i < 300; i++)
568                     {
569                         double zCurr = mesh3Dim.nodesZ[0] + k * hz;
570                         double xCurr = 0.0D + i * hx;
571                         double yCurr = 0.0D + i * hy;
572                         var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
573                         var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2 +
                            ↪  vec.Item3 * vec.Item3);
574                         var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
575                         swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");                }
576                 }
577                 swe.Close();
578             }
579         }
580 
581         public void WriteDataToDraw2DimSolution(string path)
582         {
583             double hx = (25500.0D) / 300.0D;
584             double hy = (25500.0D) / 300.0D;
585             double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 300.0D;
586             for (int t = 0; t < Time.Count; t++)
587             {
588                 using var swe = new StreamWriter(path + $"E\\Answer_E_time_layer_{t}.txt");
589                 for (int k = 0; k < 300; k++)
590                 {
591                     for (int i = 0; i < 300; i++)
592                     {
593                         double zCurr = mesh3Dim.nodesZ[0] + k * hz;
594                         double xCurr = 0.0D + i * hx;
595                         double yCurr = 0.0D + i * hy;
596                         var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
597                         var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2 +
                            ↪  vec.Item3 * vec.Item3);
598                         var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
599                         swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");
600                     }
601                 }
602                 swe.Close();
603             }
604         }
605 
606         public void WriteDataAtLine(string path)
607         {
608             double x0 = -1177.5;
609             double hy = (mesh3Dim.nodesY[^1] - mesh3Dim.nodesY[0]) / 300.0D;
610             double z0 = -1050.0D;
611             List<double> times = [Time[0], Time[Time.Count / 2], Time[^1]];
612             for (int t = 0; t < times.Count; t++)
613             {
614                 using var swa = new StreamWriter(path + $"A_With_anomaly_{times[t]}.txt");
615 
616                 for (int i = 0; i < 300; i++)
617                 {
618                     double xCurr = x0;
619                     double yCurr = mesh3Dim.nodesY[0] + i * hy;
620                     double zCurr = z0;
621                     var vec = GetAAt(xCurr, yCurr, zCurr, times[t]);
622                     var ans = vec.Item1;
623                     swa.WriteLine($"{yCurr:E15} {ans:E15}");
624                 }
625                 swa.Close();
626             }
627             for (int t = 0; t < times.Count; t++)
628             {
629                 using var swe = new StreamWriter(path + $"E_With_anomaly_{times[t]}.txt");
630                 for (int i = 0; i < 300; i++)
631                 {
632                     double xCurr = x0;
633                     double yCurr = mesh3Dim.nodesY[0] + i * hy;
634                     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;
```

```
708             var absDivZ = 0.0D;
709             var relDiscX = 0.0D;
710             var relDiscY = 0.0D;
711             var relDiscZ = 0.0D;
712             var relDivX = 0.0D;
713             var relDivY = 0.0D;
714             var relDivZ = 0.0D;
715             int iter = 0;
716             foreach (Elem elem in elemsArr)
717             {
718                 int[] elem_local = [elem[0], elem[3], elem[8], elem[11],
719                                     elem[1], elem[2], elem[9], elem[10],
720                                     elem[4], elem[5], elem[6], elem[7]];
721                 var x = 0.5D * (ribsArr[elem_local[0]].a.X + ribsArr[elem_local[0]].b.X);
722                 var y = 0.5D * (ribsArr[elem_local[4]].a.Y + ribsArr[elem_local[4]].b.Y);
723                 var z = 0.5D * (ribsArr[elem_local[8]].a.Z + ribsArr[elem_local[8]].b.Z);
724                 sw.WriteLine($"Points {x:E15} {y:E15} {z:E15}");
725                 var eps = (x - ribsArr[elem_local[0]].a.X) / (ribsArr[elem_local[0]].b.X -
                    ↪  ribsArr[elem_local[0]].a.X);
726                 var nu =  (y - ribsArr[elem_local[4]].a.Y) / (ribsArr[elem_local[4]].b.Y -
                    ↪  ribsArr[elem_local[4]].a.Y);
727                 var khi = (z - ribsArr[elem_local[8]].a.Z) / (ribsArr[elem_local[8]].b.Z -
                    ↪  ribsArr[elem_local[8]].a.Z);
728                 double[] q = [Solutions[0][elem_local[0]], Solutions[0][elem_local[1]],
                    ↪  Solutions[0][elem_local[2]], Solutions[0][elem_local[3]],
729                               Solutions[0][elem_local[4]], Solutions[0][elem_local[5]],
                                  ↪  Solutions[0][elem_local[6]], Solutions[0][elem_local[7]],
730                               Solutions[0][elem_local[8]], Solutions[0][elem_local[9]],
                                  ↪  Solutions[0][elem_local[10]], Solutions[0][elem_local[11]]];
731                 var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
732                 var theorValue = Function.A(x, y, z, 0.0D);
733                 sw.WriteLine($"FEM A  {ans.Item1:E15} {ans.Item2:E15} {ans.Item3:E15}");
734                 sw.WriteLine($"Theor  {theorValue.Item1:E15} {theorValue.Item2:E15}
                    ↪  {theorValue.Item3:E15}");
735                 var currAbsDiscX = Math.Abs(ans.Item1 - theorValue.Item1);
736                 var currAbsDiscY = Math.Abs(ans.Item2 - theorValue.Item2);
737                 var currAbsDiscZ = Math.Abs(ans.Item3 - theorValue.Item3);
738                 var currRelDiscX = currAbsDiscX / Math.Abs(theorValue.Item1);
739                 var currRelDiscY = currAbsDiscY / Math.Abs(theorValue.Item2);
740                 var currRelDiscZ = currAbsDiscZ / Math.Abs(theorValue.Item3);
741                 sw.WriteLine($"CurrAD {currAbsDiscX:E15} {currAbsDiscY:E15}
                    ↪  {currAbsDiscZ:E15}");
742                 sw.WriteLine($"CurrRD {currRelDiscX:E15} {currRelDiscY:E15}
                    ↪  {currRelDiscZ:E15}\n");
743                 absDiscX += currAbsDiscX;
744                 absDiscY += currAbsDiscY;
745                 absDiscZ += currAbsDiscZ;
746                 absDivX += theorValue.Item1;
747                 absDivY += theorValue.Item2;
748                 absDivZ += theorValue.Item3;
749                 relDiscX += currRelDiscX * currRelDiscX;
750                 relDiscY += currRelDiscY * currRelDiscY;
751                 relDiscZ += currRelDiscZ * currRelDiscZ;
752                 relDivX += theorValue.Item1 * theorValue.Item1;
753                 relDivY += theorValue.Item2 * theorValue.Item2;
754                 relDivZ += theorValue.Item3 * theorValue.Item3;
755                 iter++;
756             }
757             sw.WriteLine($"Avg disc: {absDiscX / iter:E15} {absDiscY / iter:E15} {absDiscZ /
                ↪  iter:E15}");
758             sw.WriteLine($"Rel disc: {Math.Sqrt(relDiscX / relDivX):E15} {Math.Sqrt(relDiscY /
                ↪  relDivY):E15} {Math.Sqrt(relDiscZ / relDivZ):E15}");
759             sw.Close();
760         }
761
762         public void MeasureValuesOnReceiversWithoutNormal(List<Point3D> recivers, string
            ↪  OutputPath)
763         {
764             using var sw_e = new StreamWriter(OutputPath + "E.txt");
765             for (int t = 0; t < Time.Count; t++)
766             {
767                 var a_e = AdditioanalGetEAt(recivers[0].X, recivers[0].Y, recivers[0].Z,
                    ↪  Time[t]);
768                 var b_e = AdditioanalGetEAt(recivers[1].X, recivers[1].Y, recivers[1].Z,
                    ↪  Time[t]);
```

```
769             var c_e = AdditioanalGetEAt(recivers[2].X, recivers[2].Y, recivers[2].Z,
                ↪  Time[t]);
770             var d_e = AdditioanalGetEAt(recivers[3].X, recivers[3].Y, recivers[3].Z,
                ↪  Time[t]);
771             var f_a_e = Math.Sqrt(a_e.Item1 * a_e.Item1 + a_e.Item2 * a_e.Item2 +
                ↪  a_e.Item3 * a_e.Item3);
772             var f_b_e = Math.Sqrt(b_e.Item1 * b_e.Item1 + b_e.Item2 * b_e.Item2 +
                ↪  b_e.Item3 * b_e.Item3);
773             var f_c_e = Math.Sqrt(c_e.Item1 * c_e.Item1 + c_e.Item2 * c_e.Item2 +
                ↪  c_e.Item3 * c_e.Item3);
774             var f_d_e = Math.Sqrt(d_e.Item1 * d_e.Item1 + d_e.Item2 * d_e.Item2 +
                ↪  d_e.Item3 * d_e.Item3);
775             sw_e.WriteLine($"{Time[t]} {f_a_e} {f_b_e} {f_c_e} {f_d_e}");
776         }
777         sw_e.Close();
778     }
779
780     public void MeasureValuesOnReceivers(List<Point3D> recivers, string OutputPath)
781     {
782         using var sw_a = new StreamWriter(OutputPath + "A.txt");
783         using var sw_e = new StreamWriter(OutputPath + "E.txt");
784         for (int t = 0; t < Time.Count; t++)
785         {
786             var a_a = GetAAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
787             var b_a = GetAAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
788             var c_a = GetAAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
789             var d_a = GetAAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
790             var f_a_a = Math.Sqrt(a_a.Item1 * a_a.Item1 + a_a.Item2 * a_a.Item2 +
                ↪  a_a.Item3 * a_a.Item3);
791             var f_b_a = Math.Sqrt(b_a.Item1 * b_a.Item1 + b_a.Item2 * b_a.Item2 +
                ↪  b_a.Item3 * b_a.Item3);
792             var f_c_a = Math.Sqrt(c_a.Item1 * c_a.Item1 + c_a.Item2 * c_a.Item2 +
                ↪  c_a.Item3 * c_a.Item3);
793             var f_d_a = Math.Sqrt(d_a.Item1 * d_a.Item1 + d_a.Item2 * d_a.Item2 +
                ↪  d_a.Item3 * d_a.Item3);
794             var a_e = GetEAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
795             var b_e = GetEAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
796             var c_e = GetEAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
797             var d_e = GetEAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
798             var f_a_e = Math.Sqrt(a_e.Item1 * a_e.Item1 + a_e.Item2 * a_e.Item2 +
                ↪  a_e.Item3 * a_e.Item3);
799             var f_b_e = Math.Sqrt(b_e.Item1 * b_e.Item1 + b_e.Item2 * b_e.Item2 +
                ↪  b_e.Item3 * b_e.Item3);
800             var f_c_e = Math.Sqrt(c_e.Item1 * c_e.Item1 + c_e.Item2 * c_e.Item2 +
                ↪  c_e.Item3 * c_e.Item3);
801             var f_d_e = Math.Sqrt(d_e.Item1 * d_e.Item1 + d_e.Item2 * d_e.Item2 +
                ↪  d_e.Item3 * d_e.Item3);
802             sw_a.WriteLine($"{Time[t]} {f_a_a} {f_b_a} {f_c_a} {f_d_a}");
803             sw_e.WriteLine($"{Time[t]} {f_a_e} {f_b_e} {f_c_e} {f_d_e}");
804         }
805         sw_a.Close();
806         sw_e.Close();
807     }
808 }
```

Mesh.cs

```
1  using DataStructs;
2
3  namespace Grid;
4
5  public abstract class Mesh(List<Elem> elems) : ICloneable
6  {
7      public abstract int ElemsAmount { get; set; }
8      internal int bordersAmount;
9      public List<Elem> Elems = elems;
10     public abstract int NodesAmountTotal { get; }
11     public abstract object Clone();
12 }
```

Mesh2Dim.cs

```
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

```
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];

```

```
39          return gv;
40      }
41
42      public static GlobalVector SumPar(GlobalVector gv1, GlobalVector gv2)
43      {
44          if (gv1.Size != gv2.Size)
45              throw new Exception("Найти разность векторов не возможно, т.к. они имеют
                ↪  разные размеры.");
46
47          GlobalVector gv = new(gv1.Size);
48          Parallel.For(0, gv1.Size, i => {
49              gv[i] = gv1[i] + gv2[i];
50          });
51          return gv;
52      }
53
54      public static GlobalVector Multiply(double a, GlobalVector gv)
55      {
56          GlobalVector _gv = new(gv.Size);
57          for (int i = 0; i < gv.Size; i++)
58              _gv[i] = a * gv[i];
59          return _gv;
60      }
61
62      public static GlobalVector MultiplyPar(double a, GlobalVector gv)
63      {
64          GlobalVector _gv = new(gv.Size);
65          Parallel.For(0, gv.Size, i => {
66              _gv[i] = a * gv[i];
67          });
68          return _gv;
69      }
70
71      public static double Multiply(GlobalVector gv1, GlobalVector gv2)
72      {
73          if (gv1.Size != gv2.Size) throw new Exception("Невозможно найти скалярное
            ↪  умножение векторов, из-за разности в размерах.");
74          double ans = 0.0D;
75          for (int i = 0; i < gv1.Size; i++)
76              ans += gv1[i] * gv2[i];
77          return ans;
78      }
79
80      public static double MultiplyPar(GlobalVector gv1, GlobalVector gv2)
81      {
82          if (gv1.Size != gv2.Size) throw new Exception("Невозможно найти скалярное
            ↪  умножение векторов, из-за разности в размерах.");
83          double ans = 0.0D;
84          Parallel.For(0, gv1.Size, i => {
85              ans = gv1[i] * gv2[i];
86          });
87          return ans;
88      }
89
90
91      public static GlobalVector Multiply(GlobalMatrix _gm, GlobalVector _gv)
92      {
93          if (_gm.Size != _gv.Size)
94              throw new Exception("Невозможно перемножить матрицу на вектор.");
95          GlobalVector ans = new(_gv.Size);
96
97          for (int i = 0; i < _gv.Size; i++)
98          {
99              for (int j = 0; j < _gm._ig[i + 1] - _gm._ig[i]; j++)
100             {
101                 ans[i] += _gm._al[_gm._ig[i] + j] * _gv[_gm._jg[_gm._ig[i] + j]];
102                 ans[_gm._jg[_gm._ig[i] + j]] += _gm._au[_gm._ig[i] + j] * _gv[i];
103             }
104             ans[i] += _gm._diag[i] * _gv[i];
105         }
106         return ans;
107     }
108
109     public static GlobalVector CustomMultiply(GlobalMatrix _gm, GlobalVector _gv)
110     {
111         if (_gm.Size != _gv.Size)
```

```
112             throw new Exception("Невозможно перемножить матрицу на вектор.");
113         GlobalVector ans = new(_gv.Size);
114 
115         for (int i = 0; i < _gv.Size; i++)
116         {
117             for (int j = _gm._ig[i]; j < _gm._ig[i + 1]; j++)
118             {
119                 ans[i] += _gm._al[j] * _gv[_gm._jg[j]];
120                 ans[_gm._jg[j]] += _gm._au[j] * _gv[i];
121             }
122             ans[i] += _gm._diag[i] * _gv[i];
123         }
124         return ans;
125     }
126 
127 
128     public static GlobalVector MultiplyPar(GlobalMatrix _gm, GlobalVector _gv)
129     {
130         if (_gm.Size != _gv.Size)
131             throw new Exception("Невозможно перемножить матрицу на вектор.");
132         GlobalVector ans = new(_gv.Size);
133 
134         for (int i = 0; i < _gv.Size; i++)
135         {
136             Parallel.For(0, _gm._ig[i + 1] - _gm._ig[i], j => {
137                 ans[i] += _gm._al[_gm._ig[i] + j] * _gv[_gm._jg[_gm._ig[i] + j]];
138             });
139             Parallel.For(0, _gm._ig[i + 1] - _gm._ig[i], j => {
140                 ans[_gm._jg[_gm._ig[i] + j]] += _gm._au[_gm._ig[i] + j] * _gv[i];
141             });
142             ans[i] += _gm._diag[i] * _gv[i];
143         }
144         return ans;
145     }
146 
147 
148     public static GlobalMatrix Multiply(double a, GlobalMatrix gm)
149     {
150         var ans = new GlobalMatrix(gm);
151         for (int i = 0; i < ans.Size; i++)
152         {
153             for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
154             {
155                 ans._al[ans._ig[i] + j] *= a;
156                 ans._au[ans._ig[i] + j] *= a;
157             }
158             ans._diag[i] *= a;
159         }
160         return ans;
161     }
162 
163 
164     public static GlobalMatrix MultiplyPar(double a, GlobalMatrix gm)
165     {
166         var ans = new GlobalMatrix(gm);
167         Parallel.For(0, ans.Size, i => {
168             for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
169             {
170                 ans._al[ans._ig[i] + j] *= a;
171                 ans._au[ans._ig[i] + j] *= a;
172             }
173             ans._diag[i] *= a;
174         });
175         return ans;
176     }
177 
178 
179     public static GlobalMatrix Sum(GlobalMatrix gm1, GlobalMatrix gm2)
180     {
181         if (!gm1.CheckPortrait(gm2)) throw new ArgumentException("Different matrixes
            ↪  portrait!");
182         GlobalMatrix ans = new(gm1);
183         for (int i = 0; i < ans.Size; i++)
184         {
185             for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
186             {
```

```
                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];
```

```
44          }
45          return result;
46      }
47
48      private static GlobalVector Backward(GlobalMatrix A, GlobalVector b)
49      {
50          var result = new GlobalVector(b);
51          for (int i = A.Size - 1; i >= 0; i--)
52              for (int j = A._ig[i + 1] - 1; j >= A._ig[i]; j--)
53                  result[A._jg[j]] -= A._au[j] * result[i];
54          return result;
55      }
56
57      public (GlobalVector, GlobalVector) Solve(GlobalMatrix A, GlobalVector b)
58      {
59          GlobalVector x = new(b.Size);
60          GlobalVector x_ = new(b.Size);
61          GlobalMatrix LU = new(A);
62          PartitialLU(LU);
63          GlobalVector r = Forward(LU, b - A * x);
64          var r0 = new GlobalVector(r);
65          GlobalVector z = Backward(LU, r);
66          GlobalVector p = Forward(LU, A * z);
67          GlobalVector tmp = new(b.Size);
68          GlobalVector r_ = new(b.Size);
69          GlobalVector z_ = new(b.Size);
70          GlobalVector p_ = new(b.Size);
71          double alph = 0.0D;
72          double beta = 0.0D;
73          int iter = 0;
74          do
75          {
76              x_ = x;
77              z_ = z;
78              r_ = r;
79              p_ = p;
80              alph = (p_ * r_) / (p_ * p_);
81              x = x_ + alph * z_;
82              r = r_ - alph * p_;
83              tmp = Forward(LU, A * Backward(LU, r));
84              beta = -1.0D * (p_ * tmp) / (p_ * p_);
85              z = Backward(LU, r) + beta * z_;
86              p = tmp + beta * p_;
87              iter++;
88              if (iter % 10 == 0)
89                  Console.WriteLine($"{(r.Norma() * r.Norma()) / (r0.Norma() *
                    ↪  r0.Norma()):E15}");
90          } while (iter < _maxIter && (r.Norma() * r.Norma()) / (r0.Norma() * r0.Norma())
            ↪  >= _eps * _eps);
91          Console.WriteLine(
92          $@"Computing finished!
93  Total iterations: {iter}
94  Relative residuality: {r.Norma() / b.Norma():E15}");
95          return (x, x_);
96      }
97  }
```