

# СОДЕРЖАНИЕ

|   |           |
|---|-----------|
| <b>Задание . . . . .</b>  | <b>3</b>  |
| <b>1. Теоретическая часть . . . . .</b>   | <b>4</b>  |
| 1.1. Векторные дифференциальные уравнения второго порядка с<br>разрывными решениями . . . . . | 4         |
| 1.2. Вариационная постановка . . . . .  | 5         |
| 1.3. Конечноэлементная дискретизация . . . . .  | 6         |
| 1.4. Построение матриц масс и жёсткости для трёхмерной задачи .                               | 7         |
| <b>2. Практическая часть. . . . .</b>   | <b>8</b>  |
| 2.1. Генерация трёхмерной сетки с ячейками в виде шестигранников                              | 8         |
| 2.2. Численное интегрирование. . . . .  | 12        |
| 2.3. Решение СЛАУ. . . . .  | 14        |
| 2.4. Тестирование трёхмерной задачи на полиномиальных вектор-<br>функциях. . . . .            | 14        |
| <b>3. Исследования . . . . .</b>  | <b>23</b> |
| <b>Заключение . . . . .</b>   | <b>24</b> |
| <b>Список используемых источников. . . . .</b>  | <b>25</b> |
| <b>Приложение 3. Текст программы . . . . .</b>  | <b>27</b> |

# 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):

$$\operatorname{rot} \left( \frac{1}{\mu} \operatorname{rot} \vec{A} \right) + \sigma \frac{\partial \vec{A}}{\partial t} + \epsilon \frac{\partial^2 \vec{A}}{\partial t^2} = \vec{J}^{\text{св}}. \quad (1.1)$$

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):

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

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

$$\operatorname{rot} \left( \frac{1}{\mu_0} \operatorname{rot} \vec{\mathbf{A}}^+ \right) + \sigma \frac{\partial \vec{\mathbf{A}}^+}{\partial t} = (\sigma - \sigma_n) \vec{\mathbf{E}}_n. \quad (1.3)$$

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

$$\operatorname{rot} \left( \frac{1}{\mu_0} \operatorname{rot} \vec{\mathbf{A}} \right) + \gamma \vec{\mathbf{A}} + \sigma \frac{\partial \vec{\mathbf{A}}}{\partial t} = \vec{\mathbf{F}}. \quad (1.4)$$

## 1.2. ВАРИАЦИОННАЯ ПОСТАНОВКА

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

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

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

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:

$$\begin{aligned} \int_{\Omega} \frac{1}{\mu_0} \operatorname{rot} \vec{\mathbf{A}} \cdot \operatorname{rot} \vec{\Psi} \, d\Omega + \int_{\Omega} \gamma \vec{\mathbf{A}} \cdot \vec{\Psi} \, d\Omega &= \int_{\Omega} \vec{\mathbf{F}} \cdot \vec{\Psi} \, d\Omega + \\ &+ \int_{S_2} \left( \vec{\mathbf{H}}^\Theta \times \vec{\mathbf{n}} \right) \cdot \vec{\Psi} \, dS \quad \forall \vec{\Psi} \in H_0^{rot}. \end{aligned} \quad (1.7)$$

### 1.3. КОНЕЧНОЭЛЕМЕНТНАЯ ДИСКРЕТИЗАЦИЯ

На шестиграннике базисные вектор-функции удобнее 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  $\Omega_m$  с вершинами  $(\hat{x}_i, \hat{y}_i, \hat{z}_i), i = 1 \dots 8$ . Тогда отображение шаблонного кубика  $\Omega^E$  в шестигранник  $\Omega_m$  будет задаваться соотношениями:

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

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

Отображение базисных вектор-функций  $\hat{\varphi}_i(\xi, \eta, \zeta)$  шаблонного элемента  $\Omega^E$  на шестигранник  $\Omega_m$  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  $\Omega^E$  в шестигранник  $\Omega_m$ .

## 1.4. ПОСТРОЕНИЕ МАТРИЦ МАСС И ЖЁСТКОСТИ ДЛЯ ТРЁХМЕРНОЙ ЗАДАЧИ

Матрица жёсткости:

$$\begin{aligned}\hat{G}_{ij} &= \int_{\Omega_e} \frac{1}{\mu_0} \text{rot} \hat{\psi}_i \cdot \text{rot} \hat{\psi}_j d\Omega = \\ &= \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \frac{1}{\mu_0} \frac{1}{|J|} (\mathbf{J}^T \text{rot} \hat{\varphi}_i) \cdot (\mathbf{J}^T \text{rot} \hat{\varphi}_j) d\xi d\eta d\zeta\end{aligned}\quad (1.11)$$

Матрица масс:

$$\hat{M}_{ij} = \int_{\Omega_e} \gamma \hat{\psi}_i \cdot \hat{\psi}_j d\Omega = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \gamma (\mathbf{J}^{-1} \hat{\varphi}_i) \cdot (\mathbf{J}^{-1} \hat{\varphi}_j) |J| d\xi d\eta d\zeta \quad (1.12)$$

## 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 перечислены 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  $\mu$  и  $\sigma$ . 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  $\mu$  и  $\sigma$  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).



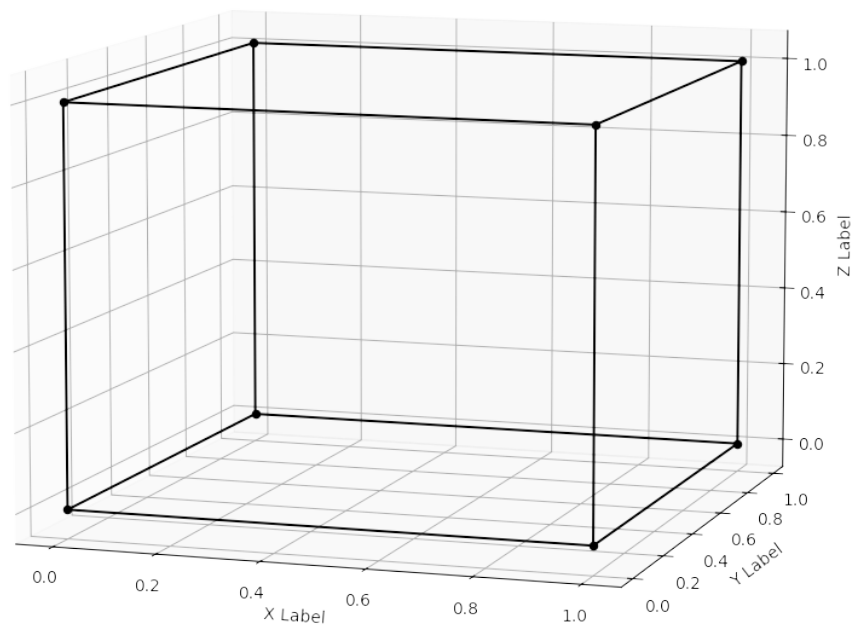


Рисунок 2.1 – Расчетная область для кубика

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

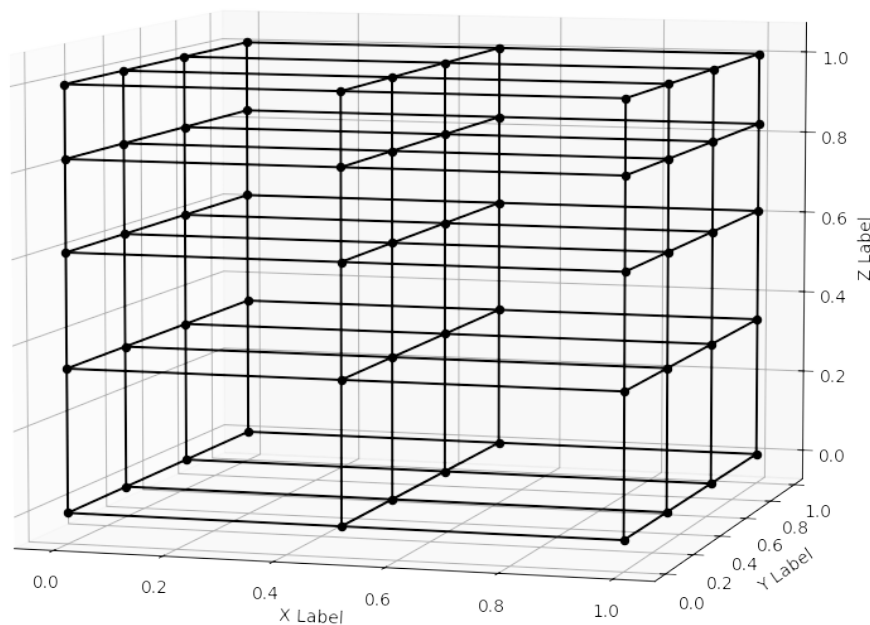
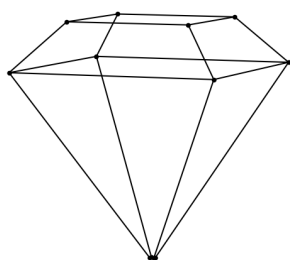
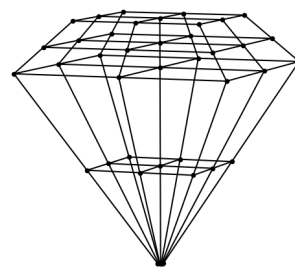


Рисунок 2.2 – Секта для кубика

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

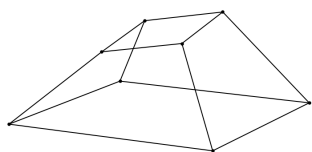


а)

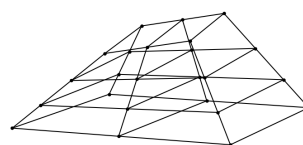


б)

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

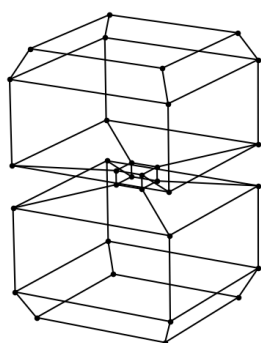


а)

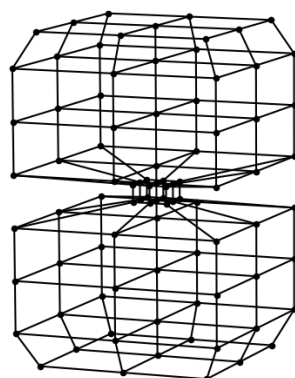


б)

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

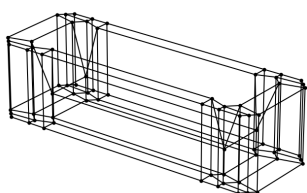


а)

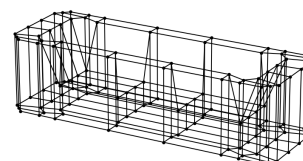


б)

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

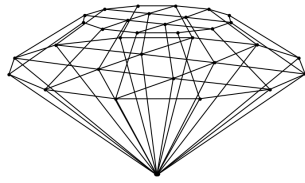


а)

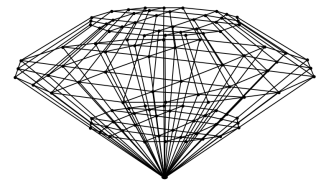


б)

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<br>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<br>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<br>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<br>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<br>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<br>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<br>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):

$$\text{rot} \left( \frac{1}{\mu} \text{rot} \vec{\mathbf{A}} \right) + \gamma \vec{\mathbf{A}} + \sigma \frac{\partial \vec{\mathbf{A}}}{\partial t} = \vec{\mathbf{F}}. \quad (2.1)$$

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

Таблица 2.8 – Тестирование при  $\vec{A} = (1.0, 1.0, 1.0)^T$ ,  $\vec{F} = (1.0, 1.0, 1.0)^T$ ,  
 $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{A} = (y, z, x)^T$ ,  $\vec{F} = (y, z, x)^T$ ,  $\mu = 1$ ,  $\gamma = 1$ ,  
 $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{\mathbf{A}} = (1 + y + x; 1 + x + z; 1 + x + y)^T$ ,  
 $\vec{\mathbf{F}} = (1 + y + x; 1 + x + z; 1 + x + y)^T$ ,  $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{\mathbf{A}} = (y - z; x - z; x - y)^T$ ,  
 $\vec{\mathbf{F}} = (y - x; x - z; x - y)^T$ ,  $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение         | Абсолютная<br>погрешность | Относительная<br>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  $\vec{\mathbf{A}} = (y \cdot z; x \cdot z; x \cdot y)^T$ ,  
 $\vec{\mathbf{F}} = (y \cdot z; x \cdot z; x \cdot y)^T$ ,  $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{\mathbf{A}} = (y^2; z^2; x^2)^T$ ,  
 $\vec{\mathbf{F}} = (y^2 - 2; z^2 - 2; x^2 - 2)^T$ ,  $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{A} = (y^2 + z^2; x^2 + z^2; x^2 + y^2)^T$ ,  
 $\vec{F} = (y^2 + z^2 - 4; x^2 + z^2 - 4; x^2 + y^2 - 4)^T$ ,  $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{A} = (y^3; 0; 0)^T$ ,  $\vec{F} = (y^3 - 6y; 0; 0)^T$ ,  $\mu = 1$ ,  
 $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{A} = (y^2 \cdot z^2; x^2 \cdot z^2; x^2 \cdot y^2)^T$ ,  
 $\vec{F} = (y^2 \cdot z^2 - 2(y^2 + z^2); x^2 \cdot z^2 - 2(x^2 + z^2); x^2 \cdot y^2 - 2(x^2 + y^2))^T$ ,  
 $\mu = 1, \gamma = 1, \sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{\mathbf{A}} = (1.0; 1.0; 1.0)^T$ ,  $\vec{\mathbf{F}} = (1.0; 1.0; 1.0)^T$ ,  
 $\mu = 1$ ,  $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>погрешность |
|-----------------|-----------------|---------------------------|------------------------------|
| (0.5; 0.5; 0.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (1.5; 0.5; 0.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (0.5; 1.5; 0.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (1.5; 1.5; 0.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (0.5; 0.5; 1.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (1.5; 0.5; 1.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (0.5; 1.5; 1.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
| (1.5; 1.5; 1.5) | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |
|                 | 1.00000000E+000 | 0.00000000E+000           | 0.00000000E+000              |

Таблица 2.18 – Тестирование при  $\vec{\mathbf{A}} = (y; z; x)^T$ ,  $\vec{\mathbf{F}} = (y; z; x)^T$ ,  $\mu = 1$ ,  
 $\gamma = 1$ ,  $\sigma = 0$

| Ребро           | Значение        | Абсолютная<br>погрешность | Относительная<br>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  $\vec{A} = (0; 0; e^x)^T$ ,  $\vec{F} = (0; 0; 0)^T$ ,  $\mu = 1$ ,  
 $\gamma = 1$ ,  $\sigma = 0$

| 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  $\vec{A} = (e^y; 0; 0)^T$ ,  $\vec{F} = (0; 0; 0)^T$ ,  $\mu = 1$ ,  
 $\gamma = 1$ ,  $\sigma = 0$

| 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  $\vec{A} = (0; e^z; 0)^T$ ,  $\vec{F} = (0; 0; 0)^T$ ,  $\mu = 1$ ,  
 $\gamma = 1$ ,  $\sigma = 0$

| 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 случаях 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

```
1 using Project;
2 using System.Globalization;
3 using Solver;
4 using Processor;
5 using Grid;
6 using static Grid.MeshReader;
7 using static Grid.MeshGenerator;
8 using static Manager.FolderManager;
9 using DataStructs;
10
11 CultureInfo.CurrentCulture = CultureInfo.InvariantCulture;
12
13 string InputDirectory = Path.GetFullPath("../.../Data/Input/");
14 string SubtotalsDirectory = Path.GetFullPath("../.../Data/Subtotals/");
15 string OutputDirectory = Path.GetFullPath("../.../Data/Output/");
16 string PicturesDirectory = Path.GetFullPath("../.../Drawer/Pictures/");
17 bool isSolving2DimTask = Checker(OutputDirectory);
18
19 // Pre-processor. Clearing output folders.
20 if (isSolving2DimTask)
21     ClearFolders(new List<string> {SubtotalsDirectory + "/2_dim/",
22                                     PicturesDirectory + "/E_phi/",
23                                     PicturesDirectory + "/A_phi/",
24                                     OutputDirectory});
25
26 // Reading mesh.
27 ReadMesh(InputDirectory + "WholeMesh.txt");
28 ReadTimeMesh(InputDirectory + "Time.txt");
29
30 // Set receivers
31 List<Point3D> receivers = [new(681.9, -681.9, 0.0), new(1331.4, -1331.4, 0.0),
32                             new(1980.8, -1980.8, 0.0), new(2630.3, -2630.3, 0.0)];
33
34 Mesh3Dim mesh3D = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
35                       NodesZ, InfoAboutZ, Elms, Borders);
36 Mesh2Dim mesh2D = new(NodesR, InfoAboutR, NodesZ, InfoAboutZ,
37                       Elms, Math.Sqrt(Math.Pow(mesh3D.nodesX[^1], 2) +
38                                             Math.Pow(mesh3D.nodesY[^1], 2)));
39 mesh2D.SetBorders(mesh3D.borders);
40 var timeMesh = GenerateTimeMesh(Time.Item1, Time.Item2, tn, tk);
41
42 // Main process of 2-dim task.
43 ConstructMesh(ref mesh2D);
44 FEM2D myFEM2D = new(mesh2D, timeMesh);
45 if (isSolving2DimTask)
46 {
47     myFEM2D.SetSolver(new LU_LOS());
48     myFEM2D.Solve();
49     myFEM2D.GenerateVectorEphi();
50     myFEM2D.WriteData(OutputDirectory);
51     myFEM2D.WritePointsToDraw(OutputDirectory + "ToDraw\\2_dim\\Aphi\\",
52                               OutputDirectory + "ToDraw\\2_dim\\Ephi\\");
53     myFEM2D.MeasureValuesOnReceivers(receivers, OutputDirectory +
54                                     "ToDraw\\2_dim\\Receivers\\");
55 }
56 else
57 {
58     myFEM2D.ReadAnswer(OutputDirectory);
59     Console.WriteLine("2D answer read");
60 }
61 myFEM2D.MeasureValuesOnReceivers(receivers, OutputDirectory +
62                                 "ToDraw\\2_dim\\Receivers\\");
63 ConstructMesh(ref mesh3D);
64 Console.WriteLine("3D mesh constructed");
```

```

62 FEM3D myFEM3D = new(mesh3D, timeMesh);
63 myFEM3D.ConvertResultTo3Dim(myFEM2D);
64 myFEM3D.GenerateVectorB();
65 Console.WriteLine("2D answer converted to 3D");
66
67 // Solving first layer: groundwater.
68 ReadAnomaly(InputDirectory + "Anomalies\\Anomaly1.txt");
69 Console.WriteLine("Anomaly read");
70
71 Mesh3Dim mesh3D_a1 = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
72     NodesZ, InfoAboutZ, Elems, Borders);
73 mesh3D_a1.CommitAnomalyBorders(FieldBorders);
74 ConstructMeshAnomaly(ref mesh3D_a1, SubtotalsDirectory + "3_dim\\Anomaly0\\");
75 Console.WriteLine("Anomaly mesh built");
76 FEM3D fem3D_a1 = new(mesh3D_a1, timeMesh, myFEM3D, 0);
77 fem3D_a1.SetSolver(new LU_LOS(15_000, 1e-15));
78 Console.WriteLine("Solving begun");
79 fem3D_a1.Solve();
80 fem3D_a1.WriteData(OutputDirectory + $"A_phi\\Answer3D\\AfterField1\\");
81 Console.WriteLine("Solved");
82 fem3D_a1.GenerateVectorE();
83 Console.WriteLine("E generated");
84 myFEM3D.AddSolution(fem3D_a1);
85 myFEM3D.MeasureValuesOnReceivers(recivers, OutputDirectory +
    ↪ "ToDraw\\3_dim\\Receivers\\");
86
87 // Solving second layer: groundwater.
88 ReadAnomaly(InputDirectory + "Anomalies\\Anomaly2.txt");
89 Console.WriteLine("Anomaly read");
90 Mesh3Dim mesh3D_a2 = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
91     NodesZ, InfoAboutZ, Elems, Borders);
92 mesh3D_a2.CommitAnomalyBorders(FieldBorders);
93 ConstructMeshAnomaly(ref mesh3D_a2, SubtotalsDirectory + "3_dim\\Anomaly1\\");
94 Console.WriteLine("Anomaly mesh built");
95 FEM3D fem3D_a2 = new(mesh3D_a2, timeMesh, myFEM3D, 1);
96 fem3D_a2.SetSolver(new LU_LOS(15_000, 1e-15));
97 Console.WriteLine("Solving begun");
98 fem3D_a2.Solve();
99 fem3D_a2.WriteData(OutputDirectory + $"A_phi\\Answer3D\\AfterField2\\");
100 Console.WriteLine("Solved");
101 fem3D_a2.GenerateVectorE();
102 Console.WriteLine("E generated");
103 myFEM3D.AddSolution(fem3D_a2);
104 myFEM3D.WriteDrawingForSecond(OutputDirectory + "ToDraw\\3_dim\\");
105 myFEM3D.MeasureValuesOnReceivers(recivers, OutputDirectory +
    ↪ "ToDraw\\3_dim\\Receivers\\");
106 Console.WriteLine($"{e}");
107
108 // Solving both anomalies.
109 ReadBothAnomalies(InputDirectory + "Anomalies\\AnomalyBoth.txt");
110 Console.WriteLine("Anomalies read");
111 Mesh3Dim mesh3D_ab = new(NodesX, InfoAboutX, NodesY, InfoAboutY,
112     NodesZ, InfoAboutZ, Elems, Borders);
113 mesh3D_ab.CommitAnomalyBorders(FieldBorders);
114 mesh3D_ab.CommitSecondAnomalyBorders(FieldBorders1);
115 ConstructMeshAnomaly(ref mesh3D_ab, SubtotalsDirectory + "3_dim\\Anomaly2\\");
116 Console.WriteLine("Anomalies mesh built");
117 FEM3D fem3D_ab = new(mesh3D_ab, timeMesh, myFEM3D, 2);
118 fem3D_ab.SetSolver(new LU_LOS(15_000, 1e-15));
119 Console.WriteLine("Solving begun");
120 fem3D_ab.Solve();
121 fem3D_ab.WriteData(OutputDirectory + $"A_phi\\Answer3D\\AfterFieldBoth\\");
122 Console.WriteLine("Solved");
123 fem3D_ab.GenerateVectorE();
124 Console.WriteLine("E generated");
125 myFEM3D.AddSolution(fem3D_ab);
126 myFEM3D.WriteDataToDraw2DimSolution(OutputDirectory + "ToDraw\\3_dim\\");
127 myFEM3D.MeasureValuesOnReceivers(recivers, OutputDirectory +
    ↪ "ToDraw\\3_dim\\Receivers\\");
128 return 0;
129
130 static bool Checker(string Answer)
131 {
132     if (CountFilesAmount(Answer + "A_phi/Answer/") == CountFilesAmount(Answer +
        ↪ "E_phi/Answer/"))

```

```

133     while (true)
134     {
135         string? ans;
136         Console.WriteLine("Detected answer for 2-dim task. Would you like to solve
            ↳ 2-dim task again? [Y/n]");
137         ans = Console.ReadLine()?.ToLower();
138         if (ans == "y" || ans == "yes" || ans == "да" || ans == "д")
139             return true;
140         else if (ans == "n" || ans == "no" || ans == "нет" || ans == "н")
141             return false;
142         Console.WriteLine("Unexpected answer!");
143     }
144     return true;
145 }

```

## LocalMatrix.cs

```

1  using DataStructs;
2  using static Functions.BasisFunctions2D;
3  using Solution;
4
5  namespace MathObjects;
6
7
8  public class LocalMatrix : Matrix
9  {
10     private TypeOfMatrixM _typeOfMatrixM;
11     private readonly double _lambda;
12     private readonly double _gamma;
13     private readonly double _rk;
14     private readonly double _hr;
15     private readonly double _hz;
16
17     public override double this[int i, int j]
18     {
19         get
20         {
21             if (i > 3 || j > 3) throw new IndexOutOfRangeException("Local matrix error.");
22             return _typeOfMatrixM switch
23             {
24                 TypeOfMatrixM.Mr => _gamma * (_Mr[i % 2, j % 2] * _Mz[i / 2, j / 2]),
25                 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]),
26                 _ => throw new Exception("Unexpected matrix"),
27             };
28         }
29         set{}
30     }
31 }
32
33 private readonly double[,] _G = {{ 1.0, -1.0},
34                                   {-1.0, 1.0}};
35 private readonly double[,] _Mz = {{2.0, 1.0},
36                                   {1.0, 2.0}};
37 private readonly double[,] _Mr1;
38 private readonly double[,] _Mir = {{2.0, 1.0},
39                                   {1.0, 2.0}};
40 private readonly double[,] _M2r = {{1.0, 1.0},
41                                   {1.0, 3.0}};
42 private readonly double[,] _Mr2 = {{-3.0D, 1.0D},
43                                   { 1.0D, 1.0D}};
44 private readonly double[,] _Gr = new double[2, 2];
45 private readonly double[,] _Mr = new double[2, 2];
46 private readonly double[,] _Gz = new double[2, 2];
47 private readonly double[,] _Mrr = new double[2, 2];
48 double[,] matr = new double[4, 4];
49
50 public LocalMatrix(double lambda, double rk, double hz, double hr)
51 {
52     _rk = rk;
53     _hr = hr;
54     _hz = hz;
55     double _d = _rk / _hr;

```

```

56     _lambda = lambda;
57     _gamma = _lambda;
58     _Mr1 = new double[2,2] {{ (1 + _d) * (1 + _d), -1.0 * _d * (1 + _d)},
59                             {-1.0 * _d * (1 + _d), _d * _d}};
60     _typeOfMatrixM = TypeOfMatrixM.Mrr;
61     for (int i = 0; i < 2; i++)
62     {
63         for (int j = 0; j < 2; j++)
64         {
65             _Gr[i, j] = ((_rk + _hr / 2.0D) / _hr) * _G[i, j];
66             _Mr[i, j] = (_hr / 6.0D) * (_rk * _M1r[i, j] + (_hr / 2.0D) * _M2r[i, j]);
67             _Mrr[i, j] = Math.Log(1.0D + 1.0D / _d) * _Mr1[i, j] - _d * _G[i, j] +
68                 ↪ 0.5 * _Mr2[i, j];
69             _Gz[i, j] = _G[i, j] / _hz;
70             _Mz[i, j] = (_hz / 6.0D) * _Mz[i, j];
71         }
72     }
73
74     public LocalMatrix(List<int> elem, ArrayOfPoints2D arrPt, TypeOfMatrixM
75     ↪ typeOfMatrixM, double lambda = 0.0D, double gamma = 0.0D)
76     {
77         _typeOfMatrixM = typeOfMatrixM;
78         _rk = arrPt[elem[0]].R;
79         _hr = arrPt[elem[1]].R - arrPt[elem[0]].R;
80         _hz = arrPt[elem[2]].Z - arrPt[elem[0]].Z;
81         double _d = _rk / _hr;
82         _lambda = 1.0D / lambda;
83         _gamma = gamma;
84         _Mr1 = new double[2,2] {{ (1 + _d) * (1 + _d), -1.0 * _d * (1 + _d)},
85                                 {-1.0 * _d * (1 + _d), _d * _d}};
86
87         for (int i = 0; i < 2; i++)
88         {
89             for (int j = 0; j < 2; j++)
90             {
91                 _Gr[i, j] = ((_rk + _hr / 2.0D) / _hr) * _G[i, j];
92                 _Mr[i, j] = (_hr / 6.0D) * (_rk * _M1r[i, j] + (_hr / 2.0D) * _M2r[i, j]);
93                 _Mrr[i, j] = Math.Log(1.0D + 1.0D / _d) * _Mr1[i, j] - _d * _G[i, j] +
94                     ↪ 0.5 * _Mr2[i, j];
95                 _Gz[i, j] = _G[i, j] / _hz;
96                 _Mz[i, j] = (_hz / 6.0D) * _Mz[i, j];
97             }
98         }
99     }

```

## LocalVector.cs

```

1  using System.Drawing;
2  using DataStructs;
3  using Functions;
4
5  namespace MathObjects;
6
7  public class LocalVector : Vector
8  {
9
10     private readonly double _r0;
11     private readonly double _r1;
12     private readonly double _z0;
13     private readonly double _z1;
14     private readonly double _hr;
15     private readonly double _hz;
16     private readonly double _t;
17     public override int Size => 4;
18
19     private readonly double[,] _M2R = {{1.0D, 1.0D},
20                                         {1.0D, 3.0D}};
21     private readonly double[,] _Mz = {{2.0D, 1.0D},
22                                         {1.0D, 2.0D}};
23     private readonly double[,] _M1R = {{2.0D, 1.0D},
24                                         {1.0D, 2.0D}};

```

```

25
26 public override double this[int i] => i switch
27 {
28     0 => _Mr[0, 0] * _Mz[0, 0] * Function.F(_r0, _z0, _t) +
29         _Mr[0, 1] * _Mz[0, 0] * Function.F(_r1, _z0, _t) +
30         _Mr[0, 0] * _Mz[0, 1] * Function.F(_r0, _z1, _t) +
31         _Mr[0, 1] * _Mz[0, 1] * Function.F(_r1, _z1, _t),
32
33     1 => _Mr[1, 0] * _Mz[0, 0] * Function.F(_r0, _z0, _t) +
34         _Mr[1, 1] * _Mz[0, 0] * Function.F(_r1, _z0, _t) +
35         _Mr[1, 0] * _Mz[0, 1] * Function.F(_r0, _z1, _t) +
36         _Mr[1, 1] * _Mz[0, 1] * Function.F(_r1, _z1, _t),
37
38     2 => _Mr[0, 0] * _Mz[1, 0] * Function.F(_r0, _z0, _t) +
39         _Mr[0, 1] * _Mz[1, 0] * Function.F(_r1, _z0, _t) +
40         _Mr[0, 0] * _Mz[1, 1] * Function.F(_r0, _z1, _t) +
41         _Mr[0, 1] * _Mz[1, 1] * Function.F(_r1, _z1, _t),
42
43     3 => _Mr[1, 0] * _Mz[1, 0] * Function.F(_r0, _z0, _t) +
44         _Mr[1, 1] * _Mz[1, 0] * Function.F(_r1, _z0, _t) +
45         _Mr[1, 0] * _Mz[1, 1] * Function.F(_r0, _z1, _t) +
46         _Mr[1, 1] * _Mz[1, 1] * Function.F(_r1, _z1, _t),
47
48     _ => throw new IndexOutOfRangeException("Vector out of index"),
49 };
50
51 private double[,] _Mr = new double[2, 2];
52
53 public LocalVector(List<int> elem, ArrayOfPoints2D arrPt, double t)
54 {
55     _r0 = arrPt[elem[0]].R;
56     _r1 = arrPt[elem[1]].R;
57     _t = t;
58
59     _z0 = arrPt[elem[0]].Z;
60     _z1 = arrPt[elem[2]].Z;
61     _hr = _r1 - _r0;
62     _hz = _z1 - _z0;
63
64     for (int i = 0; i < 2; i++)
65     {
66         for (int j = 0; j < 2; j++)
67         {
68             _Mr[i, j] = (_hr / 6.0) * (_r0 * _M1R[i, j] + (_hr / 2.0) * _M2R[i, j]);
69             _Mz[i, j] = (_hz / 6.0) * _Mz[i, j];
70         }
71     }
72 }
73
74 private void WriteVector()
75 {
76     for (int i = 0; i < 4; i++)
77         Console.WriteLine($"{this[i]:E5}");
78 }
79
80 public LocalVector(double r0, double r1, double z0, double z1)
81 {
82     _r0 = r0;
83     _r1 = r1;
84     _z0 = z0;
85     _z1 = z1;
86     _hr = _r1 - _r0;
87     _hz = _z1 - _z0;
88 }
89 }

```

## FEM.cs

```

1 namespace Project;
2 using System.Collections.Immutable;
3 using System.Numerics;
4 using MathObjects;
5 using Solver;

```

```

6  using Grid;
7  using DataStructs;
8  using System.Diagnostics;
9  using Functions;
10 using System.ComponentModel.DataAnnotations;
11 using System.Timers;
12
13 public enum EquationType
14 {
15     Elliptic,
16     Parabolic
17 }
18
19 public abstract class FEM(TimeMesh time)
20 {
21     protected static string _3dValuesPath =
22         ↪ Path.GetFullPath("../Data/Subtotals/3_dim/");
23     protected static string _elemspath2D =
24         ↪ Path.GetFullPath("../Data/Subtotals/2_dim/Elms.poly");
25     protected static string _pointspath2D =
26         ↪ Path.GetFullPath("../Data/Subtotals/2_dim/Points.poly");
27     protected static string _borderspath2D =
28         ↪ Path.GetFullPath("../Data/Subtotals/2_dim/Borders.poly");
29     protected static string _elemspath3D =
30         ↪ Path.GetFullPath("../Data/Subtotals/3_dim/Elms.poly");
31     protected static string _pointspath3D =
32         ↪ Path.GetFullPath("../Data/Subtotals/3_dim/Points.poly");
33     protected static string _borderspath3D =
34         ↪ Path.GetFullPath("../Data/Subtotals/3_dim/Borders.poly");
35
36     public TimeMesh Time = time;
37     protected internal EquationType equationType;
38     public ISolver? solver;
39     public ArrayOfElms elemsArr;
40     public ArrayOfBorders bordersArr;
41     public GlobalMatrix? Matrix;
42     public GlobalVector? Vector;
43     public GlobalVector? Answer;
44     public GlobalVector[] Solutions = new GlobalVector[time.Count];
45     public GlobalVector[] Discrepancy = new GlobalVector[time.Count];
46
47     public void SetSolver(ISolver solver)
48     {
49         this.solver = solver;
50         Debug.WriteLine("Solvset set");
51     }
52 }

```

## FEM2D.cs

```

1  using MathObjects;
2  using DataStructs;
3  using Grid;
4  using System.Diagnostics;
5  using Functions;
6
7  namespace Project;
8
9  public class FEM2D : FEM
10 {
11     public ArrayOfPoints2D pointsArr = new(_pointspath2D);
12
13     public FEM2D(Mesh2Dim mesh, TimeMesh timeMesh) : base(timeMesh)
14     {
15         mesh2Dim = mesh;
16         if (timeMesh[0] == timeMesh[~1])
17             equationType = EquationType.Elliptic;
18         else
19             equationType = EquationType.Parabolic;
20
21         elemsArr = new(_elemspath2D);
22         bordersArr = new(_borderspath2D);
23
24         A_phi = new GlobalVector[Time.Count];

```

```

25     E_phi = new GlobalVector[Time.Count];
26     Debug.WriteLine("Generated data submitted");
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($"Time layer: before BC");
42     Thread.Sleep(1500);
43
44     Matrix = new GlobalMatrix(pointsArr.GetLength());
45     Generator.BuildPortait(ref Matrix, pointsArr.GetLength(), elemsArr);
46     Generator.FillMatrix(ref Matrix, pointsArr, elemsArr, TypeOfMatrixM.Mrr);
47
48     Vector = new GlobalVector(pointsArr.GetLength());
49     Generator.FillVector(ref Vector, pointsArr, elemsArr, Time[0]);
50
51     Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, pointsArr,
52     ↪ bordersArr, Time[0]);
53     (Solutions[0], Discrepancy[0]) = solver.Solve(Matrix, Vector);
54
55     if (Time.Count > 1)
56     {
57         (Solutions[1], Discrepancy[1]) = (Solutions[0], Discrepancy[0]);
58         if (Time.Count > 2)
59         for (int i = 2; i < Time.Count; i++)
60         {
61             Console.WriteLine($"Time layer: {Time[i]}");
62             //Thread.Sleep(1500);
63
64             double delT = Time[i] - Time[i - 2];
65             double delT0 = Time[i] - Time[i - 1];
66             double delT1 = Time[i - 1] - Time[i - 2];
67             double tau0 = (delT + delT0) / (delT * delT0);
68             double tau1 = delT / (delT1 * delT0);
69             double tau2 = delT0 / (delT * delT1);
70             double delT = Time[i] - Time[i - 1];
71             double tau = 1.0D / delT;
72
73             var matrix1 = new GlobalMatrix(pointsArr.GetLength());
74             Generator.BuildPortait(ref matrix1, pointsArr.GetLength(), elemsArr);
75             Generator.FillMatrix(ref matrix1, pointsArr, elemsArr, TypeOfMatrixM.Mrr);
76             var M = new GlobalMatrix(pointsArr.GetLength()); // ???
77             Generator.BuildPortait(ref M, pointsArr.GetLength(), elemsArr);
78             Generator.FillMatrix(ref M, pointsArr, elemsArr, TypeOfMatrixM.Mr);
79
80             var bi = new GlobalVector(pointsArr.GetLength());
81             Matrix = (tau0 * M) + matrix1;
82             Vector = bi + (tau1 * (M * Solutions[i - 1])) - (tau2 * (M * Solutions[i
83             ↪ - 2]));
84             Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, pointsArr,
85             ↪ bordersArr, Time[i]);
86             (Solutions[i], Discrepancy[i]) = solver.Solve(Matrix, Vector);
87         }
88     }
89     A_phi = Solutions;
90     solutionStopwatch.Stop();
91     var milseconds = solutionStopwatch.ElapsedMilliseconds;
92     Console.WriteLine($"Lin eq solved for {milseconds / 60000} min {(milseconds %
93     ↪ 60000) / 1000} sec");
94 }
95
96 public void WriteData()
97 {
98     if (Answer is null)
99         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_
110                     ↪ e={Time[i]}.dat");
111                 for (int j = 0; j < A_phi[i].Size; j++)
112                     sw.WriteLine($"{A_phi[i][j]:E8}");
113                 sw.Close();
114             }
115             for (int i = 0; i < Time.Count; i++)
116             {
117                 using var sw = new StreamWriter($"{_path}\\E_phi\\Answer\\Answer_Ephi_tim_
118                     ↪ e={Time[i]}.dat");
119                 for (int j = 0; j < E_phi[i].Size; j++)
120                     sw.WriteLine($"{E_phi[i][j]:E8}");
121                 sw.Close();
122             }
123         }
124         else
125         {
126             using var sw = new StreamWriter($"{_path}\\A_phi\\Answer\\Answer.dat");
127             for (int j = 0; j < A_phi[0].Size; j++)
128                 sw.WriteLine($"{A_phi[0][j]:E8}");
129             sw.Close();
130             using var sw1 = new StreamWriter($"{_path}\\E_phi\\Answer\\Answer.dat");
131             for (int j = 0; j < E_phi[0].Size; j++)
132                 sw1.WriteLine($"{E_phi[0][j]:E8}");
133             sw1.Close();
134         }
135     }
136
137     public void WriteDiscrepancy(string _path)
138     {
139         if (A_phi is null) throw new ArgumentNullException();
140         if (E_phi is null) throw new ArgumentNullException();
141
142         if (Time.Count != 1)
143         {
144             for (int i = 0; i < Time.Count; i++)
145             {
146                 using var sw_d = new StreamWriter($"{_path}\\A_phi\\Discrepancy\\Discrepa_
147                     ↪ ncy_Aphi_time={Time[i]}.dat");
148
149                 int NotNaNamount = 0;
150                 double maxDisc = 0.0;
151                 double avgDisc = 0.0;
152                 double sumU = 0.0D;
153                 double sumD = 0.0D;
154
155                 List<double> TheorAnswer = [];
156                 foreach (var Z in mesh2Dim.nodesZ)
157                     foreach (var R in mesh2Dim.nodesR)
158                         TheorAnswer.Add(Function.U(R, Z, Time[i]));
159
160                 for (int j = 0; j < A_phi[i].Size; j++)
161                 {
162                     double absDiff = Math.Abs(A_phi[i][j] - TheorAnswer[j]);
163                     double currDisc = Math.Abs((A_phi[i][j] - TheorAnswer[j]) /
164                         ↪ TheorAnswer[j]);
165
166                     if (Math.Abs(maxDisc) < Math.Abs(currDisc))
167                         maxDisc = currDisc;
168
169                     if (!double.IsNaN(currDisc) && currDisc > 1E-14)
170                     {

```

```

167         avgDisc += currDisc;
168         NotNaNamount++;
169         sumU += absDiff * absDiff;
170         sumD += TheorAnswer[j] * TheorAnswer[j];
171     }
172     sw_d.WriteLine($"{absDiff:E8} {currDisc:E8}");
173 }
174 avgDisc = Math.Sqrt(sumU) / Math.Sqrt(sumD);
175 sw_d.WriteLine($"Средняя невязка: {avgDisc:E15}");
176 sw_d.WriteLine($"Максимальная невязка: {maxDisc:E15}");
177 sw_d.WriteLine($"C: {avgDisc:E7}");
178 sw_d.WriteLine($"M: {maxDisc:E7}");
179 sw_d.Close();
180 }
181 }
182 else
183 {
184     using var sw_d = new
185         ↳ StreamWriter($"{_path}\\A_phi\\Discrepancy\\Discrepancy_Aphi.dat");
186     int NotNaNamount = 0;
187     double maxDisc = 0.0;
188     double avgDisc = 0.0;
189     double sumU = 0.0D;
190     double sumD = 0.0D;
191     List<double> TheorAnswer = [];
192     foreach (var Z in mesh2Dim.nodesZ)
193         foreach (var R in mesh2Dim.nodesR)
194             TheorAnswer.Add(Function.U(R, Z, 0.0D));
195     for (int j = 0; j < A_phi[0].Size; j++)
196     {
197         double absDiff = Math.Abs(A_phi[0][j] - TheorAnswer[j]);
198         double currDisc = Math.Abs((A_phi[0][j] - TheorAnswer[j]) /
199             ↳ TheorAnswer[j]);
200
201         if (Math.Abs(maxDisc) < Math.Abs(currDisc))
202             maxDisc = currDisc;
203
204         if (!double.IsNaN(currDisc) && currDisc > 1E-14)
205         {
206             avgDisc += currDisc;
207             NotNaNamount++;
208             sumU += absDiff * absDiff;
209             sumD += TheorAnswer[j] * TheorAnswer[j];
210         }
211         sw_d.WriteLine($"{absDiff:E8} {currDisc:E8}");
212     }
213     avgDisc = Math.Sqrt(sumU) / Math.Sqrt(sumD);
214     sw_d.WriteLine($"Средняя невязка: {avgDisc:E15}");
215     sw_d.WriteLine($"Максимальная невязка: {maxDisc:E15}");
216     sw_d.WriteLine($"C: {avgDisc:E7}");
217     sw_d.WriteLine($"M: {maxDisc:E7}");
218     sw_d.Close();
219 }
220 }
221 public void GenerateVectorEphi()
222 {
223     E_phi = new GlobalVector[A_phi.Length];
224     for (int i = 0; i < E_phi.Length; i++)
225     {
226         if (i == 0)
227             E_phi[i] = new GlobalVector(A_phi[i].Size);
228         else
229             E_phi[i] = -1.0D / (Time[i] - Time[i - 1]) * (A_phi[i] - A_phi[i - 1]);
230     }
231 }
232
233 internal List<int>? GetElem(double r, double z)
234 {
235     if (r < mesh2Dim.nodesR[0] || mesh2Dim.nodesR[^1] < r || z < mesh2Dim.nodesZ[0]
236         ↳ || mesh2Dim.nodesZ[^1] < z)
237         return null;
238     int i = 0;
239     for (; i < mesh2Dim.nodesR.Count - 1 && r >= 0.001; i++)
240         if (mesh2Dim.nodesR[i] <= r && r <= mesh2Dim.nodesR[i + 1])

```

```

240         break;
241     int j = 0;
242     for (; j < mesh2Dim.nodesZ.Count - 1; j++)
243         if (mesh2Dim.nodesZ[j] <= z && z <= mesh2Dim.nodesZ[j + 1])
244             break;
245     return elemsArr[j * (mesh2Dim.nodesR.Count - 1) + i].Arr;
246 }
247
248 public double GetA_phiAt(double r, double z, double t)
249 {
250     for (int tt = 0; tt < Time.Count; tt++)
251     {
252         if (Time[tt] == t)
253         {
254             var elem = GetElem(r, z);
255             if (elem is null) return 0.0D;
256             double[] q = new double[4];
257             for (int i = 0; i < 4; i++)
258                 q[i] = A_phi[tt][elem[i]];
259             double r0 = pointsArr[elem[0]].R;
260             double r1 = pointsArr[elem[3]].R;
261             double z0 = pointsArr[elem[0]].Z;
262             double z1 = pointsArr[elem[3]].Z;
263             return BasisFunctions2D.GetValue(q[0], q[1], q[2], q[3], r0, r1, z0, z1,
264                 ↪ r, z);
265         }
266     }
267     throw new Exception("Out of mesh borders");
268 }
269
270 public double GetE_phiAt(double r, double z, double t)
271 {
272     for (int tt = 0; tt < Time.Count; tt++)
273     {
274         if (Time[tt] == t)
275         {
276             var elem = GetElem(r, z);
277             if (elem is null) return 0.0D;
278             double[] q = new double[4];
279             for (int i = 0; i < 4; i++)
280                 q[i] = E_phi[tt][elem[i]];
281             double r0 = pointsArr[elem[0]].R;
282             double r1 = pointsArr[elem[3]].R;
283             double z0 = pointsArr[elem[0]].Z;
284             double z1 = pointsArr[elem[3]].Z;
285             return BasisFunctions2D.GetValue(q[0], q[1], q[2], q[3], r0, r1, z0, z1,
286                 ↪ r, z);
287         }
288     }
289     throw new Exception("Out of mesh borders");
290 }
291
292 public void ReadAnswer(string AnswerPath)
293 {
294     string file = string.Empty;
295     if (equationType == EquationType.Elliptic)
296     {
297         file = "Answer.dat";
298         var fileData = File.ReadAllText(AnswerPath + "A_phi/Answer/" +
299             ↪ file).Split("\n");
300         A_phi[0] = new GlobalVector(fileData.Length - 1);
301         for (int i = 0; i < fileData.Length - 1; i++)
302             A_phi[0][i] = double.Parse(fileData[i]);
303         fileData = File.ReadAllText(AnswerPath + "E_phi/Answer/" + file).Split("\n");
304         E_phi[0] = new GlobalVector(fileData.Length - 1);
305         for (int i = 0; i < fileData.Length - 1; i++)
306             E_phi[0][i] = double.Parse(fileData[i]);
307     }
308     else
309     {
310         for (int t = 0; t < Time.Count; t++)
311         {
312             file = $"Answer_Aphi_time={Time[t]}.dat";
313             var fileData = File.ReadAllText(AnswerPath + "A_phi/Answer/" +
314                 ↪ 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 = $"Answer_Ephi_time={Time[t]}.dat";
315         fileData = File.ReadAllText(AnswerPath + "E_phi/Answer/" +
            ↪ file).Split("\n");
316         E_phi[t] = new GlobalVector(fileData.Length - 1);
317         for (int i = 0; i < fileData.Length - 1; i++)
318             E_phi[t][i] = double.Parse(fileData[i]);
319     }
320 }
321 }
322
323 public void MeasureValuesOnReceivers(List<Point3D> receivers, string OutputPath)
324 {
325     using var sw_a = new StreamWriter(OutputPath + "A.txt");
326     using var sw_e = new StreamWriter(OutputPath + "E.txt");
327     List<(double, double)> pnt2D = [];
328     foreach (var receiver in receivers)
329     {
330         pnt2D.Add((Math.Sqrt(receiver.X * receiver.X + receiver.Y * receiver.Y),
            ↪ receiver.Z));
331     }
332     for (int t = 0; t < Time.Count; t++)
333     {
334         var a_a = GetA_phiAt(pnt2D[0].Item1, pnt2D[0].Item2, Time[t]);
335         var b_a = GetA_phiAt(pnt2D[1].Item1, pnt2D[1].Item2, Time[t]);
336         var c_a = GetA_phiAt(pnt2D[2].Item1, pnt2D[2].Item2, Time[t]);
337         var d_a = GetA_phiAt(pnt2D[3].Item1, pnt2D[3].Item2, Time[t]);
338         var a_e = GetE_phiAt(pnt2D[0].Item1, pnt2D[0].Item2, Time[t]);
339         var b_e = GetE_phiAt(pnt2D[1].Item1, pnt2D[1].Item2, Time[t]);
340         var c_e = GetE_phiAt(pnt2D[2].Item1, pnt2D[2].Item2, Time[t]);
341         var d_e = GetE_phiAt(pnt2D[3].Item1, pnt2D[3].Item2, Time[t]);
342         sw_a.WriteLine($"{Time[t]} {a_a} {b_a} {c_a} {d_a}");
343         sw_e.WriteLine($"{Time[t]} {a_e} {b_e} {c_e} {d_e}");
344     }
345     sw_a.Close();
346     sw_e.Close();
347 }
348
349 public void WritePointsToDraw(string pathA, string pathE)
350 {
351     double hr = (mesh2Dim.nodesR[^1] - mesh2Dim.nodesR[0]) / 150;
352     double hz = (mesh2Dim.nodesZ[^1] - mesh2Dim.nodesZ[0]) / 150;
353     for (int t = 0; t < Time.Count; t++)
354     {
355         using var swa = new StreamWriter(pathA + $"Answer_A_time_layer_{t}.txt");
356         for (int j = 0; j < 150; j++)
357         {
358             for (int i = 0; i < 150; i++)
359             {
360                 double rCurr = mesh2Dim.nodesR[0] + i * hr;
361                 double zCurr = mesh2Dim.nodesZ[0] + j * hz;
362                 swa.WriteLine($"{rCurr:E15} {zCurr:E15} {GetA_phiAt(rCurr, zCurr,
                    ↪ Time[t]):E15}");
363             }
364         }
365         swa.Close();
366     }
367     for (int t = 0; t < Time.Count; t++)
368     {
369         using var swe = new StreamWriter(pathE + $"Answer_E_time_layer_{t}.txt");
370         for (int j = 0; j < 150; j++)
371         {
372             for (int i = 0; i < 150; i++)
373             {
374                 double rCurr = mesh2Dim.nodesR[0] + i * hr;
375                 double zCurr = mesh2Dim.nodesZ[0] + j * hz;
376                 swe.WriteLine($"{rCurr:E15} {zCurr:E15} {GetE_phiAt(rCurr, zCurr,
                    ↪ Time[t]):E15}");
377             }
378         }
379         swe.Close();
380     }
381 }

```

## FEM3D.cs

```

1  using MathObjects;
2  using Grid;
3  using DataStructs;
4  using Functions;
5  using System.Diagnostics;
6
7
8
9  namespace Project;
10
11 public class FEM3D : FEM
12 {
13     private static readonly double mu0 = 4.0D * Math.PI * Math.Pow(10.0D, -7);
14     public ArrayOfPoints3D pointsArr;
15     public List<GlobalVector> A;
16     public List<GlobalVector> E;
17     public List<GlobalVector> B;
18     public List<GlobalVector> H;
19     public ArrayOfRibs? ribsArr;
20     private readonly Mesh3Dim mesh3Dim;
21     private GlobalMatrix? G;
22     List<FEM3D> additionalFields = [];
23     private readonly FEM3D? _originalFEM;
24     private List<GlobalVector>? _originalE;
25     private GlobalMatrix? M;
26     internal static List<int> ConvertGlobalToLocalNumeration(List<int> global) =>
27         [global[0], global[3], global[8], global[11],
28          global[1], global[2], global[9], global[10],
29          global[4], global[5], global[6], global[7]];
30
31     public FEM3D(Mesh3Dim mesh, TimeMesh timeMesh) : base(timeMesh)
32     {
33         string pointsPath = _3dValuesPath + $"AfterConvertation\\Points.poly";
34         string elemsPath = _3dValuesPath + $"AfterConvertation\\Elems.poly";
35         string bordersPath = _3dValuesPath + $"AfterConvertation\\Borders.poly";
36         pointsArr = new ArrayOfPoints3D(pointsPath);
37         elemsArr = new(elemsPath, 3);
38         bordersArr = new(bordersPath, 3);
39         ribsArr = mesh.arrayOfRibs;
40         equationType = timeMesh[0] == timeMesh[1] ? EquationType.Elliptic :
41             ↪ EquationType.Parabolic;
42         A = [];
43         E = [];
44         B = [];
45         H = [];
46         mesh3Dim = mesh;
47     }
48
49     public FEM3D(Mesh3Dim mesh, TimeMesh timeMesh, FEM3D originalFEM, int Num) :
50         ↪ base(timeMesh)
51     {
52         string pointsPath = _3dValuesPath + $"Anomaly{Num}\\Points.poly";
53         string elemsPath = _3dValuesPath + $"Anomaly{Num}\\Elems.poly";
54         string bordersPath = _3dValuesPath + $"Anomaly{Num}\\Borders.poly";
55         pointsArr = new(pointsPath);
56         elemsArr = new(elemsPath, 3);
57         bordersArr = new(bordersPath, 3);
58         ribsArr = mesh.arrayOfRibs;
59         _originalFEM = originalFEM;
60         _originalE = [];
61         equationType = timeMesh[0] == timeMesh[1] ? EquationType.Elliptic :
62             ↪ EquationType.Parabolic;
63         A = [];
64         E = [];
65         B = [];
66         H = [];
67         mesh3Dim = mesh;
68         ConstructMatrixes();
69     }
70
71     public (double, double, double) GetAAAt(double x, double y, double z, double t)
72     {
73         if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
74             ↪ generated");
75     }

```

```

71     for (int tt = 0; tt < Time.Count; tt++)
72     if (Time[tt] == t)
73     {
74         var arr = GetElem(x, y, z);
75         if (arr is null) return (0.0D, 0.0D, 0.0D);
76         var elem = ConvertGlobalToLocalNumeration(arr);
77         double[] q = new double[12];
78         for (int i = 0; i < 12; i++)
79             q[i] = A[tt][elem[i]];
80         double x0 = ribsArr[elem[0]].a.X;
81         double x1 = ribsArr[elem[0]].b.X;
82         double y0 = ribsArr[elem[4]].a.Y;
83         double y1 = ribsArr[elem[4]].b.Y;
84         double z0 = ribsArr[elem[8]].a.Z;
85         double z1 = ribsArr[elem[8]].b.Z;
86         double eps = (x - x0) / (x1 - x0);
87         double nu = (y - y0) / (y1 - y0);
88         double khi = (z - z0) / (z1 - z0);
89         var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
90         foreach (var solution in additionalFields)
91         {
92             var arrCurr = solution.GetElem(x, y, z);
93             if (arrCurr is null) continue;
94             var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
95             double[] qCurr = new double[12];
96             for (int ii = 0; ii < 12; ii++)
97                 qCurr[ii] = solution.A[tt][elemCurr[ii]];
98             double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
99             double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
100            double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
101            double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
102            double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
103            double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
104            double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
105            double nuCurr = (y - y0Curr) / (y1Curr - y0Curr);
106            double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
107            var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
108                ↪ qCurr);
109            ans.Item1 += ansCurr.Item1;
110            ans.Item2 += ansCurr.Item2;
111            ans.Item3 += ansCurr.Item3;
112        }
113        return ans;
114    }
115    throw new Exception("Out of mesh borders");
116 }
117 public (double, double, double) AdditioanalGetEAt(double x, double y, double z,
118     ↪ double t)
119 {
120     if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
121     ↪ generated");
122     for (int tt = 0; tt < Time.Count; tt++)
123     if (Time[tt] == t)
124     {
125         var ans = (0.0D, 0.0D, 0.0D);
126         foreach (var solution in additionalFields)
127         {
128             var arrCurr = solution.GetElem(x, y, z);
129             if (arrCurr is null) continue;
130             var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
131             double[] qCurr = new double[12];
132             for (int ii = 0; ii < 12; ii++)
133                 qCurr[ii] = solution.E[tt][elemCurr[ii]];
134             double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
135             double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
136             double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
137             double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
138             double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
139             double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
140             double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
141             double nuCurr = (y - y0Curr) / (y1Curr - y0Curr);
142             double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
143             var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
144                 ↪ qCurr);

```

```

142         ans.Item1 += ansCurr.Item1;
143         ans.Item2 += ansCurr.Item2;
144         ans.Item3 += ansCurr.Item3;
145     }
146     return ans;
147 }
148 throw new Exception("Out of mesh borders");
149 }
150
151 public (double, double, double) GetEAt(double x, double y, double z, double t)
152 {
153     if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
154     ↪ generated");
155     for (int tt = 0; tt < Time.Count; tt++)
156     {
157         if (Time[tt] == t)
158         {
159             var arr = GetElem(x, y, z);
160             if (arr is null) return (0.0D, 0.0D, 0.0D);
161             var elem = ConvertGlobalToLocalNumeration(arr);
162             double[] q = new double[12];
163             for (int i = 0; i < 12; i++)
164             {
165                 q[i] = E[tt][elem[i]];
166                 double x0 = ribsArr[elem[0]].a.X;
167                 double x1 = ribsArr[elem[0]].b.X;
168                 double y0 = ribsArr[elem[4]].a.Y;
169                 double y1 = ribsArr[elem[4]].b.Y;
170                 double z0 = ribsArr[elem[8]].a.Z;
171                 double z1 = ribsArr[elem[8]].b.Z;
172                 double eps = (x - x0) / (x1 - x0);
173                 double nu = (y - y0) / (y1 - y0);
174                 double khi = (z - z0) / (z1 - z0);
175                 var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
176                 foreach (var solution in additionalFields)
177                 {
178                     var arrCurr = solution.GetElem(x, y, z);
179                     if (arrCurr is null) continue;
180                     var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
181                     double[] qCurr = new double[12];
182                     for (int ii = 0; ii < 12; ii++)
183                     {
184                         qCurr[ii] = solution.E[tt][elemCurr[ii]];
185                         double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
186                         double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
187                         double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
188                         double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
189                         double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
190                         double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
191                         double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
192                         double nuCurr = (y - y0Curr) / (y1Curr - y0Curr);
193                         double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
194                         var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
195                         ↪ qCurr);
196                         ans.Item1 += ansCurr.Item1;
197                         ans.Item2 += ansCurr.Item2;
198                         ans.Item3 += ansCurr.Item3;
199                     }
200                 }
201             }
202             return ans;
203         }
204     }
205     throw new Exception("Out of mesh borders");
206 }
207
208 public (double, double, double) GetBAtd(double x, double y, double z, double t)
209 {
210     if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
211     ↪ generated");
212     for (int tt = 0; tt < Time.Count; tt++)
213     {
214         if (Time[tt] == t)
215         {
216             var arr = GetElem(x, y, z);
217             if (arr is null) return (0.0D, 0.0D, 0.0D);
218             var elem = ConvertGlobalToLocalNumeration(arr);
219             double[] q = new double[12];
220             for (int i = 0; i < 12; i++)
221             {
222                 q[i] = B[tt][elem[i]];
223                 double x0 = ribsArr[elem[0]].a.X;
224                 double x1 = ribsArr[elem[0]].b.X;

```

```

214         double y0 = ribsArr[elem[4]].a.Y;
215         double y1 = ribsArr[elem[4]].b.Y;
216         double z0 = ribsArr[elem[8]].a.Z;
217         double z1 = ribsArr[elem[8]].b.Z;
218         double eps = (x - x0) / (x1 - x0);
219         double nu = (y - y0) / (y1 - y0);
220         double khi = (z - z0) / (z1 - z0);
221         var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
222         foreach (var solution in additionalFields)
223         {
224             var arrCurr = solution.GetElem(x, y, z);
225             if (arrCurr is null) continue;
226             var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
227             double[] qCurr = new double[12];
228             for (int ii = 0; ii < 12; ii++)
229                 qCurr[ii] = solution.B[tt][elemCurr[ii]];
230             double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
231             double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
232             double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
233             double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
234             double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
235             double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
236             double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
237             double nuCurr = (y - y0Curr) / (y1Curr - y0Curr);
238             double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
239             var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
240                 ↪ qCurr);
241             ans.Item1 += ansCurr.Item1;
242             ans.Item2 += ansCurr.Item2;
243             ans.Item3 += ansCurr.Item3;
244         }
245         return ans;
246     }
247     throw new Exception("Out of mesh borders");
248 }
249 public (double, double, double) GetHAt(double x, double y, double z, double t)
250 {
251     if (ribsArr is null) throw new ArgumentNullException("Array of ribs not
252         ↪ generated");
253     for (int tt = 0; tt < Time.Count; tt++)
254         if (Time[tt] == t)
255         {
256             var arr = GetElem(x, y, z);
257             if (arr is null) return (0.0D, 0.0D, 0.0D);
258             var elem = ConvertGlobalToLocalNumeration(arr);
259             double[] q = new double[12];
260             for (int i = 0; i < 12; i++)
261                 q[i] = H[tt][elem[i]];
262             double x0 = ribsArr[elem[0]].a.X;
263             double x1 = ribsArr[elem[0]].b.X;
264             double y0 = ribsArr[elem[4]].a.Y;
265             double y1 = ribsArr[elem[4]].b.Y;
266             double z0 = ribsArr[elem[8]].a.Z;
267             double z1 = ribsArr[elem[8]].b.Z;
268             double eps = (x - x0) / (x1 - x0);
269             double nu = (y - y0) / (y1 - y0);
270             double khi = (z - z0) / (z1 - z0);
271             var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
272             foreach (var solution in additionalFields)
273             {
274                 var arrCurr = solution.GetElem(x, y, z);
275                 if (arrCurr is null) continue;
276                 var elemCurr = ConvertGlobalToLocalNumeration(arrCurr);
277                 double[] qCurr = new double[12];
278                 for (int ii = 0; ii < 12; ii++)
279                     qCurr[ii] = solution.H[tt][elemCurr[ii]];
280                 double x0Curr = solution.ribsArr[elemCurr[0]].a.X;
281                 double x1Curr = solution.ribsArr[elemCurr[0]].b.X;
282                 double y0Curr = solution.ribsArr[elemCurr[4]].a.Y;
283                 double y1Curr = solution.ribsArr[elemCurr[4]].b.Y;
284                 double z0Curr = solution.ribsArr[elemCurr[8]].a.Z;
285                 double z1Curr = solution.ribsArr[elemCurr[8]].b.Z;
286                 double epsCurr = (x - x0Curr) / (x1Curr - x0Curr);
287                 double nuCurr = (y - y0Curr) / (y1Curr - y0Curr);

```



```

287         double khiCurr = (z - z0Curr) / (z1Curr - z0Curr);
288         var ansCurr = BasisFunctions3DVec.GetValue(epsCurr, nuCurr, khiCurr,
289             ↪ qCurr);
289         ans.Item1 += ansCurr.Item1;
290         ans.Item2 += ansCurr.Item2;
291         ans.Item3 += ansCurr.Item3;
292     }
293     return ans;
294 }
295 throw new Exception("Out of mesh borders");
296 }
297
298 // B = rot A
299 public void GenerateVectorB()
300 {
301     if (A is []) throw new Exception("Vector B isn't generated");
302     if (ribsArr is null) throw new Exception("Array of ribs isn't generated");
303     B = new(A.Count);
304     for (int t = 0; t < Time.Count; t++)
305     {
306         B.Add(new GlobalVector(ribsArr.Count));
307         for (int i = 0; i < A[t].Size; i++)
308         {
309             double ht = Math.Pow(10, -10);
310             var pnt = ribsArr[i].GetMiddlePoint();
311             var tan = ribsArr[i].GetTangent();
312             var vec1 = GetAAAt(pnt.X, pnt.Y, pnt.Z + ht, Time[t]);
313             var vec2 = GetAAAt(pnt.X, pnt.Y, pnt.Z - ht, Time[t]);
314             var Bx = -1.0D * (vec1.Item2 - vec2.Item2) / (2.0D * ht);
315             vec1 = GetAAAt(pnt.X, pnt.Y, pnt.Z + ht, Time[t]);
316             vec2 = GetAAAt(pnt.X, pnt.Y, pnt.Z - ht, Time[t]);
317             var By = (vec1.Item1 - vec2.Item1) / (2.0D * ht);
318             vec1 = GetAAAt(pnt.X + ht, pnt.Y, pnt.Z, Time[t]);
319             vec2 = GetAAAt(pnt.X - ht, pnt.Y, pnt.Z, Time[t]);
320             var Bz1 = (vec1.Item2 - vec2.Item2) / (2.0D * ht);
321             vec1 = GetAAAt(pnt.X, pnt.Y + ht, pnt.Z, Time[t]);
322             vec2 = GetAAAt(pnt.X, pnt.Y - ht, pnt.Z, Time[t]);
323             var Bz2 = (vec1.Item1 - vec2.Item1) / (2.0D * ht);
324             B[t][i] = Bx * tan.Item1 + By * tan.Item2 + (Bz1 - Bz2) * tan.Item3;
325         }
326     }
327 }
328
329 // H = B / mu_0
330 public void GenerateVectorH()
331 {
332     if (B is []) throw new Exception("Vector B isn't generated");
333     H = new(B.Count);
334     for (int t = 0; t < B.Count; t++)
335     {
336         H.Add(new GlobalVector(B[t].Size));
337         for (int i = 0; i < B[i].Size; i++)
338             H[t][i] = B[t][i] / mu0;
339     }
340 }
341
342 // E = - dA / dt
343 public void GenerateVectorE()
344 {
345     E = new(A.Count);
346     for (int i = 0; i < A.Count; i++)
347     {
348         if (i == 0)
349             E.Add(new GlobalVector(A[1].Size));
350         else
351         {
352             if (A[i - 1].Size == 0) A[i - 1] = new GlobalVector(ribsArr.Count);
353             if (A[i].Size == 0) A[i] = new GlobalVector(ribsArr.Count);
354             E.Add(-1.0D / (Time[i] - Time[i - 1]) * (A[i] - A[i - 1]));
355         }
356     }
357 }
358
359 public void AddSolution(FEM3D fem) => additionalFields.Add(fem);
360

```

```

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,
407                     ↪ fem2d.pointsArr[elem[1]].R,
408                     ↪ fem2d.pointsArr[elem[0]].Z,
409                     ↪ fem2d.pointsArr[elem[3]].Z,
410                     ↪ Math.Sqrt(X * X + Y * Y), Z);
411                 fe = BasisFunctions2D.GetValue(
412                     fem2d.E_phi[i][elem[0]], fem2d.E_phi[i][elem[1]],
413                     fem2d.E_phi[i][elem[2]], fem2d.E_phi[i][elem[3]],
414                     ↪ fem2d.pointsArr[elem[0]].R,
415                     ↪ fem2d.pointsArr[elem[1]].R,
416                     ↪ fem2d.pointsArr[elem[0]].Z,
417                     ↪ fem2d.pointsArr[elem[3]].Z,
418                     ↪ Math.Sqrt(X * X + Y * Y), Z);
419             }
420             var Ax = X == 0 && Y == 0 ? 0.0D : -1.0D * (Y / Math.Sqrt(X * X + Y * Y))
421                 ↪ * fa;
422             var Ay = X == 0 && Y == 0 ? 0.0D : X / Math.Sqrt(X * X + Y * Y) * fa;
423             var Az = 0.0D;
424             var Ex = X == 0 && Y == 0 ? 0.0D : -1.0D * (Y / Math.Sqrt(X * X + Y *
425                 ↪ Y)) * fe;
426             var Ey = X == 0 && Y == 0 ? 0.0D : X / Math.Sqrt(X * X + Y * Y) * fe;
427             var Ez = 0.0D;
428             A[i][j] = Ax * antinormal.Item1 + Ay * antinormal.Item2 + Az *
429                 ↪ antinormal.Item3;
430             E[i][j] = Ex * antinormal.Item1 + Ey * antinormal.Item2 + Ez *
431                 ↪ antinormal.Item3;
432         }
433     }
434 }

```

```

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($"Time: {Time[t]}:E15");
436         foreach (var reciver in recivers)
437         {
438             var Bat = GetBAT(reciver.X, reciver.Y, reciver.Z, Time[t]);
439             var BAT_1 = GetBAT(reciver.X, reciver.Y, reciver.Z, Time[t + 1]);
440             var dBdt = ((BAT_1.Item1 - BAT.Item1) / (Time[t + 1] - Time[t]),
441                         (BAT_1.Item2 - BAT.Item2) / (Time[t + 1] - Time[t]),
442                         (BAT_1.Item3 - BAT.Item3) / (Time[t + 1] - Time[t]));
443
444             var Ex_1 = GetEAt(reciver.X - 1E-10, reciver.Y, reciver.Z, Time[t]);
445             var Ex_2 = GetEAt(reciver.X + 1E-10, reciver.Y, reciver.Z, Time[t]);
446             var Ey_1 = GetEAt(reciver.X, reciver.Y - 1E-10, reciver.Z, Time[t]);
447             var Ey_2 = GetEAt(reciver.X, reciver.Y + 1E-10, reciver.Z, Time[t]);
448             var Ez_1 = GetEAt(reciver.X, reciver.Y, reciver.Z - 1E-10, Time[t]);
449             var Ez_2 = GetEAt(reciver.X, reciver.Y, reciver.Z + 1E-10, Time[t]);
450             var rotE = ((Ey_2.Item3 - Ey_1.Item3) / (2.0 * 1E-10) - (Ez_2.Item2 -
451                         ↪ Ez_1.Item2) / (2.0 * 1E-10),
452                         -1.0D * ((Ex_2.Item3 - Ex_1.Item3) / (2.0 * 1E-10) - (Ez_2.Item1 -
453                         ↪ Ez_1.Item1) / (2.0 * 1E-10)),
454                         (Ex_2.Item2 - Ex_1.Item2) / (2.0 * 1E-10) - (Ey_2.Item1 -
455                         ↪ Ey_1.Item1) / (2.0 * 1E-10));
456             sw.WriteLine($"reciver: ({reciver.X:E15}, {reciver.Y:E15},
457                         ↪ {reciver.Z:E15})");
458             sw.WriteLine($"rot E: ({rotE.Item1:E15}, {rotE.Item2:E15},
459                         ↪ {rotE.Item3:E15})");
460             sw.WriteLine($"dBdt: ({dBdt.Item1:E15}, {dBdt.Item2:E15},
461                         ↪ {dBdt.Item3:E15})");
462             sw.WriteLine($"discep.: ({Math.Abs(rotE.Item1 - dBdt.Item1):E15},
463                         ↪ {Math.Abs(rotE.Item2 - dBdt.Item2):E15}, {Math.Abs(rotE.Item3 -
464                         ↪ dBdt.Item3):E15})");
465         }
466         sw.WriteLine();
467     }
468     sw.Close();
469 }
470
471 public void ConstructMatrixes()
472 {
473     if (ribsArr is null) throw new ArgumentNullException("ribsArr is null");
474     var sparseMatrix = new GlobalMatrix(ribsArr.Count);
475     Generator.BuildPortait(ref sparseMatrix, ribsArr.Count, elemsArr, true);
476     G = new GlobalMatrix(sparseMatrix);
477     Generator.FillMatrixG(ref G, ribsArr, elemsArr);
478     M = new GlobalMatrix(sparseMatrix);
479     Generator.FillMatrixM(ref M, ribsArr, elemsArr, mesh3Dim, _originalFEM.mesh3Dim);
480 }
481
482 public void Solve()
483 {
484     if (solver is null) throw new ArgumentNullException("Solver is null");
485     if (ribsArr is null) throw new ArgumentNullException("ribs array is null");
486     if (_originalFEM is null) throw new ArgumentNullException("original E is null");
487     if (G is null) throw new ArgumentNullException();
488     if (M is null) throw new ArgumentNullException();
489     Stopwatch solutionStopwatch = new();
490     solutionStopwatch.Start();
491     Solutions = new GlobalVector[Time.Count];
492     Discrepancy = new GlobalVector[Time.Count];
493     (Solutions[0], Discrepancy[0]) = (new GlobalVector(ribsArr.Count), new
494     ↪ GlobalVector(ribsArr.Count));
495     if (Time.Count > 1)
496     {
497         (Solutions[1], Discrepancy[1]) = (Solutions[0], Discrepancy[0]);
498         for (int i = 1; i < Time.Count; i++)
499         {
500             Console.WriteLine($"\\n {i} / {Time.Count - 1}. Time layer: {Time[i]}");
501         }
502     }
503 }

```

```

491         Thread.Sleep(1500);
492
493         double delT = Time[i] - Time[i - 2];
494         double delT0 = Time[i] - Time[i - 1];
495         double delT1 = Time[i - 1] - Time[i - 2];
496         double tau0 = (delT + delT0) / (delT * delT0);
497         double tau1 = delT / (delT1 * delT0);
498         double tau2 = delT0 / (delT * delT1);
499         double delT = Time[i] - Time[i - 1];
500         double tau = 1.0D / delT;
501         Matrix = G + tau0 * M;
502         var b = new GlobalVector(ribsArr.Count);
503         Generator.FillVector3D(ref b, _originalFEM.GetEAt, ribsArr, elemsArr,
504             ↪ mesh3Dim, _originalFEM.mesh3Dim, Time[i]);
505         Vector = b + (tau1 * (M * Solutions[i - 1])) - (tau2 * (M * Solutions[i -
506             ↪ 2]));
507         Generator.ConsiderBoundaryConditions(ref Matrix, ref Vector, ribsArr,
508             ↪ bordersArr, Time[i]);
509         var localStopWatch = Stopwatch.StartNew();
510         (Solutions[i], Discrepancy[i]) = solver.Solve(Matrix, Vector);
511         localStopWatch.Stop();
512         var localMS = localStopWatch.ElapsedMilliseconds;
513         Console.WriteLine($"Current iteration for {localMS / 60000} min {(localMS
514             ↪ % 60000) / 1000} sec");
515     }
516 }
517
518 A = [... Solutions];
519 solutionStopwatch.Stop();
520 var milseconds = solutionStopwatch.ElapsedMilliseconds;
521 Console.WriteLine($"Lin eq solved for {milseconds / 60000} min {(milseconds %
522     ↪ 60000) / 1000} sec");
523 }
524
525 public void WriteData(string path)
526 {
527     if (Solutions is null) throw new ArgumentNullException("No solutions");
528     for (int t = 0; t < Time.Count; t++)
529     {
530         using var sw = new StreamWriter(path + $"Answer_{Time[t]}.txt");
531         for (int i = 0; i < A[t].Size; i++)
532             sw.WriteLine($"{i} {A[t][i]:E8}");
533         sw.Close();
534     }
535 }
536
537 public void WriteDrawingForFirst(string path)
538 {
539     double hx = (50000.0D) / 300.0D;
540     double hy = (25000.0D) / 300.0D;
541     double hz = (mesh3Dim.nodesZ[1] - mesh3Dim.nodesZ[0]) / 300.0D;
542
543     for (int t = 0; t < Time.Count; t++)
544     {
545         using var swe = new StreamWriter(path + $"E\\Answer_E_time_layer_{t}.txt");
546         for (int k = 0; k < 300; k++)
547         {
548             for (int i = 0; i < 300; i++)
549             {
550                 double zCurr = mesh3Dim.nodesZ[0] + k * hz;
551                 double xCurr = 0.0D + i * hx;
552                 double yCurr = 0.0D + i * hy;
553                 var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
554                 var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2);
555                 var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
556                 swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");
557             }
558         }
559         swe.Close();
560     }
561 }
562
563 public void WriteDrawingForSecond(string path)
564 {
565     double hx = (50000.0D) / 300.0D;
566     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 + $"E\\Answer_E_time_layer_{t}.txt");
565         for (int k = 0; k < 300; k++)
566         {
567             for (int i = 0; i < 300; i++)
568             {
569                 double zCurr = mesh3Dim.nodesZ[0] + k * hz;
570                 double xCurr = 0.0D + i * hx;
571                 double yCurr = 0.0D + i * hy;
572                 var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
573                 var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2 +
574                                     ↪ vec.Item3 * vec.Item3);
575                 var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
576                 swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");
577             }
578             swe.Close();
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 + $"E\\Answer_E_time_layer_{t}.txt");
589             for (int k = 0; k < 300; k++)
590             {
591                 for (int i = 0; i < 300; i++)
592                 {
593                     double zCurr = mesh3Dim.nodesZ[0] + k * hz;
594                     double xCurr = 0.0D + i * hx;
595                     double yCurr = 0.0D + i * hy;
596                     var vec = GetEAt(xCurr, yCurr, zCurr, Time[t]);
597                     var ans = Math.Sqrt(vec.Item1 * vec.Item1 + vec.Item2 * vec.Item2 +
598                                     ↪ vec.Item3 * vec.Item3);
599                     var rCurr = Math.Sqrt(xCurr * xCurr + yCurr * yCurr);
600                     swe.WriteLine($"{rCurr:E15} {zCurr:E15} {ans:E15}");
601                 }
602             }
603             swe.Close();
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 + $"A_With_anomaly_{times[t]}.txt");
615             for (int i = 0; i < 300; i++)
616             {
617                 double xCurr = x0;
618                 double yCurr = mesh3Dim.nodesY[0] + i * hy;
619                 double zCurr = z0;
620                 var vec = GetAAt(xCurr, yCurr, zCurr, times[t]);
621                 var ans = vec.Item1;
622                 swa.WriteLine($"{yCurr:E15} {ans:E15}");
623             }
624             swa.Close();
625         }
626         for (int t = 0; t < times.Count; t++)
627         {
628             using var swe = new StreamWriter(path + $"E_With_anomaly_{times[t]}.txt");
629             for (int i = 0; i < 300; i++)
630             {
631                 double xCurr = x0;
632                 double yCurr = mesh3Dim.nodesY[0] + i * hy;
633                 double zCurr = z0;
634

```

```

635         var vec = GetEAt(xCurr, yCurr, zCurr, times[t]);
636         var ans = vec.Item1;
637         swe.WriteLine($"{yCurr:E15} {ans:E15}");
638     }
639     swe.Close();
640 }
641 }
642
643 public void WriteDataToDraw(string path)
644 {
645     double hx = (mesh3Dim.nodesX[^1] - mesh3Dim.nodesX[0]) / 10.0D;
646     double hy = (mesh3Dim.nodesY[^1] - mesh3Dim.nodesY[0]) / 10.0D;
647     double hz = (mesh3Dim.nodesZ[^1] - mesh3Dim.nodesZ[0]) / 10.0D;
648     double x0 = mesh3Dim.nodesX[0];
649     double y0 = mesh3Dim.nodesY[0];
650     double z0 = mesh3Dim.nodesZ[0];
651     List<Point3D> points = [];
652     int i = 0;
653     int j = 0;
654     int k = 0;
655     while (z0 + hz * k <= mesh3Dim.nodesZ[^1])
656     {
657         j = 0;
658         while (y0 + hy * j <= mesh3Dim.nodesY[^1])
659         {
660             i = 0;
661             while (x0 + hx * i <= mesh3Dim.nodesX[^1])
662             {
663                 points.Add(new Point3D(x0 + i * hx, y0 + j * hy, z0 + k * hz));
664                 i++;
665             }
666             j++;
667         }
668         k++;
669     }
670     for (int t = 0; t < Time.Count; t++)
671     {
672         using var sw = new StreamWriter(path + $"Answer{t}.txt");
673         sw.WriteLine($"{mesh3Dim.nodesX[0]:E15} {mesh3Dim.nodesX[^1]:E15}
        ↳ {mesh3Dim.nodesY[0]:E15} {mesh3Dim.nodesY[^1]:E15}
        ↳ {mesh3Dim.nodesZ[0]:E15} {mesh3Dim.nodesZ[^1]:E15}");
674         foreach (var point in points)
675         {
676             var ans = GetEAt(point.X, point.Y, point.Z, Time[t]);
677             sw.WriteLine($"{point.X:E15} {point.Y:E15} {point.Z:E15} {ans.Item1:E15}
        ↳ {ans.Item2:E15} {ans.Item3:E15}");
678         }
679         sw.Close();
680     }
681 }
682
683 public void ReadData(string AnswerPath)
684 {
685     A = new(Time.Count);
686     string file = string.Empty;
687     for (int t = 0; t < Time.Count; t++)
688     {
689         file = $"Answer_{Time[t]}.txt";
690         var fileData = File.ReadAllText(AnswerPath + file).Split("\n");
691         A.Add(new GlobalVector(fileData.Length - 1));
692         for (int i = 0; i < fileData.Length - 1; i++)
693         {
694             var val = fileData[i].Split(" ")[1];
695             A[t][i] = double.Parse(val);
696         }
697     }
698 }
699
700 public void TestOutput(string path)
701 {
702     using var sw = new StreamWriter(path + "/A_phi/Answer3D/Answer_Test.txt");
703     var absDiscX = 0.0D;
704     var absDiscY = 0.0D;
705     var absDiscZ = 0.0D;
706     var absDivX = 0.0D;
707     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($"Points {x:E15} {y:E15} {z:E15}");
725         var eps = (x - ribsArr[elem_local[0]].a.X) / (ribsArr[elem_local[0]].b.X -
726             ↪ ribsArr[elem_local[0]].a.X);
727         var nu = (y - ribsArr[elem_local[4]].a.Y) / (ribsArr[elem_local[4]].b.Y -
728             ↪ ribsArr[elem_local[4]].a.Y);
729         var khi = (z - ribsArr[elem_local[8]].a.Z) / (ribsArr[elem_local[8]].b.Z -
730             ↪ ribsArr[elem_local[8]].a.Z);
731         double[] q = [Solutions[0][elem_local[0]], Solutions[0][elem_local[1]],
732             ↪ Solutions[0][elem_local[2]], Solutions[0][elem_local[3]],
733             ↪ Solutions[0][elem_local[4]], Solutions[0][elem_local[5]],
734             ↪ Solutions[0][elem_local[6]], Solutions[0][elem_local[7]],
735             ↪ Solutions[0][elem_local[8]], Solutions[0][elem_local[9]],
736             ↪ Solutions[0][elem_local[10]], Solutions[0][elem_local[11]]];
737         var ans = BasisFunctions3DVec.GetValue(eps, nu, khi, q);
738         var theorValue = Function.A(x, y, z, 0.0D);
739         sw.WriteLine($"FEM A {ans.Item1:E15} {ans.Item2:E15} {ans.Item3:E15}");
740         sw.WriteLine($"Theor {theorValue.Item1:E15} {theorValue.Item2:E15}
741             ↪ {theorValue.Item3:E15}");
742         var currAbsDiscX = Math.Abs(ans.Item1 - theorValue.Item1);
743         var currAbsDiscY = Math.Abs(ans.Item2 - theorValue.Item2);
744         var currAbsDiscZ = Math.Abs(ans.Item3 - theorValue.Item3);
745         var currRelDiscX = currAbsDiscX / Math.Abs(theorValue.Item1);
746         var currRelDiscY = currAbsDiscY / Math.Abs(theorValue.Item2);
747         var currRelDiscZ = currAbsDiscZ / Math.Abs(theorValue.Item3);
748         sw.WriteLine($"CurrAD {currAbsDiscX:E15} {currAbsDiscY:E15}
749             ↪ {currAbsDiscZ:E15}");
750         sw.WriteLine($"CurrRD {currRelDiscX:E15} {currRelDiscY:E15}
751             ↪ {currRelDiscZ:E15}\n");
752         absDiscX += currAbsDiscX;
753         absDiscY += currAbsDiscY;
754         absDiscZ += currAbsDiscZ;
755         absDivX += theorValue.Item1;
756         absDivY += theorValue.Item2;
757         absDivZ += theorValue.Item3;
758         relDiscX += currRelDiscX * currRelDiscX;
759         relDiscY += currRelDiscY * currRelDiscY;
760         relDiscZ += currRelDiscZ * currRelDiscZ;
761         relDivX += theorValue.Item1 * theorValue.Item1;
762         relDivY += theorValue.Item2 * theorValue.Item2;
763         relDivZ += theorValue.Item3 * theorValue.Item3;
764         iter++;
765     }
766     sw.WriteLine($"Avg disc: {absDiscX / iter:E15} {absDiscY / iter:E15} {absDiscZ /
767         ↪ iter:E15}");
768     sw.WriteLine($"Rel disc: {Math.Sqrt(relDiscX / relDivX):E15} {Math.Sqrt(relDiscY /
769         ↪ relDivY):E15} {Math.Sqrt(relDiscZ / relDivZ):E15}");
770     sw.Close();
771 }
772
773 public void MeasureValuesOnReceiversWithoutNormal(List<Point3D> receivers, string
774     ↪ OutputPath)
775 {
776     using var sw_e = new StreamWriter(OutputPath + "E.txt");
777     for (int t = 0; t < Time.Count; t++)
778     {
779         var a_e = AdditioanalGetEAt(receivers[0].X, receivers[0].Y, receivers[0].Z,
780             ↪ Time[t]);
781         var b_e = AdditioanalGetEAt(receivers[1].X, receivers[1].Y, receivers[1].Z,
782             ↪ Time[t]);
783     }
784 }

```

```

769         var c_e = AdditioanalGetEAt(recivers[2].X, recivers[2].Y, recivers[2].Z,
770             ↪ Time[t]);
771         var d_e = AdditioanalGetEAt(recivers[3].X, recivers[3].Y, recivers[3].Z,
772             ↪ Time[t]);
773         var f_a_e = Math.Sqrt(a_e.Item1 * a_e.Item1 + a_e.Item2 * a_e.Item2 +
774             ↪ a_e.Item3 * a_e.Item3);
775         var f_b_e = Math.Sqrt(b_e.Item1 * b_e.Item1 + b_e.Item2 * b_e.Item2 +
776             ↪ b_e.Item3 * b_e.Item3);
777         var f_c_e = Math.Sqrt(c_e.Item1 * c_e.Item1 + c_e.Item2 * c_e.Item2 +
778             ↪ c_e.Item3 * c_e.Item3);
779         var f_d_e = Math.Sqrt(d_e.Item1 * d_e.Item1 + d_e.Item2 * d_e.Item2 +
780             ↪ d_e.Item3 * d_e.Item3);
781         sw_e.WriteLine($"{Time[t]} {f_a_e} {f_b_e} {f_c_e} {f_d_e}");
782     }
783     sw_e.Close();
784 }
785
786 public void MeasureValuesOnReceivers(List<Point3D> recivers, string OutputPath)
787 {
788     using var sw_a = new StreamWriter(OutputPath + "A.txt");
789     using var sw_e = new StreamWriter(OutputPath + "E.txt");
790     for (int t = 0; t < Time.Count; t++)
791     {
792         var a_a = GetAAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
793         var b_a = GetAAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
794         var c_a = GetAAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
795         var d_a = GetAAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
796         var f_a_a = Math.Sqrt(a_a.Item1 * a_a.Item1 + a_a.Item2 * a_a.Item2 +
797             ↪ a_a.Item3 * a_a.Item3);
798         var f_b_a = Math.Sqrt(b_a.Item1 * b_a.Item1 + b_a.Item2 * b_a.Item2 +
799             ↪ b_a.Item3 * b_a.Item3);
800         var f_c_a = Math.Sqrt(c_a.Item1 * c_a.Item1 + c_a.Item2 * c_a.Item2 +
801             ↪ c_a.Item3 * c_a.Item3);
802         var f_d_a = Math.Sqrt(d_a.Item1 * d_a.Item1 + d_a.Item2 * d_a.Item2 +
803             ↪ d_a.Item3 * d_a.Item3);
804         var a_e = GetEAt(recivers[0].X, recivers[0].Y, recivers[0].Z, Time[t]);
805         var b_e = GetEAt(recivers[1].X, recivers[1].Y, recivers[1].Z, Time[t]);
806         var c_e = GetEAt(recivers[2].X, recivers[2].Y, recivers[2].Z, Time[t]);
807         var d_e = GetEAt(recivers[3].X, recivers[3].Y, recivers[3].Z, Time[t]);
808         var f_a_e = Math.Sqrt(a_e.Item1 * a_e.Item1 + a_e.Item2 * a_e.Item2 +
809             ↪ a_e.Item3 * a_e.Item3);
810         var f_b_e = Math.Sqrt(b_e.Item1 * b_e.Item1 + b_e.Item2 * b_e.Item2 +
811             ↪ b_e.Item3 * b_e.Item3);
812         var f_c_e = Math.Sqrt(c_e.Item1 * c_e.Item1 + c_e.Item2 * c_e.Item2 +
813             ↪ c_e.Item3 * c_e.Item3);
814         var f_d_e = Math.Sqrt(d_e.Item1 * d_e.Item1 + d_e.Item2 * d_e.Item2 +
815             ↪ d_e.Item3 * d_e.Item3);
816         sw_a.WriteLine($"{Time[t]} {f_a_a} {f_b_a} {f_c_a} {f_d_a}");
817         sw_e.WriteLine($"{Time[t]} {f_a_e} {f_b_e} {f_c_e} {f_d_e}");
818     }
819     sw_a.Close();
820     sw_e.Close();
821 }
822 }

```

## 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

```
1 using System.Collections.Immutable;
2 using DataStructs;
3
4 namespace Grid;
5
6 public class Mesh2Dim(List<double> nodesR, string infoAboutR,
7                      List<double> nodesZ, string infoAboutZ,
8                      List<Elem> elems, double lastR) : Mesh(elems)
9 {
10     public override int NodesAmountTotal => NodesAmountR * NodesAmountZ;
11     public List<double> nodesR = [.. nodesR, lastR];
12     public List<double> nodesZ = nodesZ;
13     public override int ElemsAmount
14     {
15         get => (NodesAmountR - 1) * (NodesAmountZ - 1);
16         set => ElemsAmount = value;
17     }
18     public int NodesAmountR => nodesR.Count;
19     public int NodesAmountZ => nodesZ.Count;
20     public List<int> NodesR_Refs = [0];
21     public List<int> NodesZ_Refs = [0];
22     internal ImmutableArray<double> NodesRWithoutFragmentation { get; set; } = [..
23     ↪ nodesR, lastR];
24     public ImmutableArray<double> NodesZWithoutFragmentation { get; set; } = [.. nodesZ];
25     internal string infoAboutR = infoAboutR;
26     internal string infoAboutZ = infoAboutZ;
27     internal List<Border2D> borders = [];
28     public void SetLastR(double val)
29     {
30         int i = 1;
31         if (val <= nodesR[^i])
32         {
33             while (val < nodesR[^i]) i++;
34             nodesR[^i] = val;
35             nodesR = nodesR.Take(i + 1).ToList();
36         }
37         else
38             nodesR.Add(val);
39     }
40     public void SetBorders(List<Border3D> borders3D)
41     {
42         // Set lower border.
43         borders.Add(new Border2D(borders3D[0].BorderType, borders3D[0].BorderFormula,
44                                 0, NodesRWithoutFragmentation.Length - 1, 0, 0));
45         // Set left border.
46         borders.Add(new Border2D(borders3D[1].BorderType, borders3D[1].BorderFormula,
47                                 0, 0, 0, NodesZWithoutFragmentation.Length - 1));
48         // Set right border.
49         borders.Add(new Border2D(borders3D[3].BorderType, borders3D[3].BorderFormula,
50                                 NodesRWithoutFragmentation.Length - 1,
51                                 NodesRWithoutFragmentation.Length - 1,
52                                 0, NodesZWithoutFragmentation.Length - 1));
53         // Set upper border.
54         borders.Add(new Border2D(borders3D[^1].BorderType, borders3D[^1].BorderFormula,
55                                 0, NodesRWithoutFragmentation.Length - 1,
56                                 NodesZWithoutFragmentation.Length - 1,
57                                 NodesZWithoutFragmentation.Length - 1));
58     }
59     public override Mesh2Dim Clone() => (Mesh2Dim)MemberwiseClone();
60 }
61
```

## Mesh3Dim.cs

```
1 using System.Collections.Immutable;
2 using DataStructs;
3
4 namespace Grid;
5
```

```

6 public class Mesh3Dim(List<double> nodesX, string infoAboutX,
7                      List<double> nodesY, string infoAboutY,
8                      List<double> nodesZ, string infoAboutZ,
9                      List<Elem> elems, List<Border3D> borders) : Mesh(elems)
10 {
11     public override int NodesAmountTotal
12     {
13         get => NodesAmountX * NodesAmountY * NodesAmountZ;
14     }
15
16     public override int ElemsAmount
17     {
18         get => (NodesAmountX - 1) * (NodesAmountY - 1) * (NodesAmountZ - 1);
19         set => ElemsAmount = value;
20     }
21
22     public int NodesAmountX => nodesX.Count;
23     public int NodesAmountY => nodesY.Count;
24     public int NodesAmountZ => nodesZ.Count;
25     public List<double> nodesX = nodesX;
26     public List<double> nodesY = nodesY;
27     public List<double> nodesZ = nodesZ;
28     public void CommitFieldBorders(List<int> ints) => FieldBorders = ints;
29     public void CommitAnomalyBorders(List<int> ints) => AnomalyBorders = ints;
30     public void CommitSecondAnomalyBorders(List<int> ints) => SecondAnomaly = ints;
31     public List<int> FieldBorders = [];
32     public List<int> AnomalyBorders = [];
33     public List<int> SecondAnomaly = [];
34     internal ImmutableArray<double> NodesXWithoutFragmentation { get; set; } = [..
35         ↪ nodesX];
36     internal ImmutableArray<double> NodesYWithoutFragmentation { get; set; } = [..
37         ↪ nodesY];
38     public ImmutableArray<double> NodesZWithoutFragmentation { get; set; } = [.. nodesZ];
39     internal string infoAboutX = infoAboutX;
40     internal string infoAboutY = infoAboutY;
41     internal string infoAboutZ = infoAboutZ;
42     public List<int> NodesXRefs = [0];
43     public List<int> NodesYRefs = [0];
44     public List<int> NodesZRefs = [0];
45     public List<Border3D> borders = borders;
46     public ArrayOfRibs? arrayOfRibs;
47     public override Mesh3Dim Clone() => (Mesh3Dim)MemberwiseClone();
48 }

```

## LocalMatrix3D.cs

```

1 namespace MathObjects;
2
3 public class LocalMatrixG3D (double mu, double hx, double hy, double hz) : Matrix
4 {
5     private readonly double _mu = mu;
6     private readonly double _hx = hx;
7     private readonly double _hy = hy;
8     private readonly double _hz = hz;
9     private readonly double[,] G1 = {{ 2.0D, 1.0D, -2.0D, -1.0D},
10                                     { 1.0D, 2.0D, -1.0D, -2.0D},
11                                     {-2.0D, -1.0D, 2.0D, 1.0D},
12                                     {-1.0D, -2.0D, 1.0D, 2.0D}};
13     private readonly double[,] G2 = {{ 2.0D, -2.0D, 1.0D, -1.0D},
14                                     {-2.0D, 2.0D, -1.0D, 1.0D},
15                                     { 1.0D, -1.0D, 2.0D, -2.0D},
16                                     {-1.0D, 1.0D, -2.0D, 2.0D}};
17     private readonly double[,] G3 = {{-2.0D, 2.0D, -1.0D, 1.0D},
18                                     {-1.0D, 1.0D, -2.0D, 2.0D},
19                                     { 2.0D, -2.0D, 1.0D, -1.0D},
20                                     { 1.0D, -1.0D, 2.0D, -2.0D}};
21
22     public override double this[int i, int j]
23     {
24         get
25         {
26             return (i / 4, j / 4)
27             switch

```

```

28     {
29         (0, 0) => (_hx * _hy / (6.0D * _hz) * G1[i % 4, j % 4] + _hx * _hz /
30             ↪ (6.0D * _hy) * G2[i % 4, j % 4]) / _mu,
31         (0, 1) or (1, 0) => -1.0D * (_hz / 6.0D) * G2[i % 4, j % 4] / _mu,
32         (0, 2) => _hy / 6.0D * G3[i % 4, j % 4] / _mu,
33         (1, 1) => (_hx * _hy / (6.0D * _hz) * G1[i % 4, j % 4] + _hy * _hz /
34             ↪ (6.0D * _hx) * G2[i % 4, j % 4]) / _mu,
35         (1, 2) or (2, 1) => -1.0D * _hx / 6.0D * G1[i % 4, j % 4] / _mu,
36         (2, 0) => _hy / 6.0D * G3[j % 4, i % 4] / _mu,
37         (2, 2) => (_hx * _hz / (6.0D * _hy) * G1[i % 4, j % 4] + _hy * _hz /
38             ↪ (6.0D * _hx) * G2[i % 4, j % 4]) / _mu,
39         _ => throw new ArgumentOutOfRangeException("Out of local matrix 3d
40             ↪ range"),
41     };
42 }
43 set{}
44 }
45 }
46
47 public class LocalMatrixM3D(double gamma, double hx, double hy, double hz) : Matrix
48 {
49     private readonly double _gamma = gamma;
50     private readonly double _hx = hx;
51     private readonly double _hy = hy;
52     private readonly double _hz = hz;
53     private readonly double[,] D = {{4.0D, 2.0D, 2.0D, 1.0D},
54                                     {2.0D, 4.0D, 1.0D, 2.0D},
55                                     {2.0D, 1.0D, 4.0D, 2.0D},
56                                     {1.0D, 2.0D, 2.0D, 4.0D}};
57
58     public override double this[int i, int j]
59     {
60         get
61         {
62             return (i / 4, j / 4)
63             switch
64             {
65                 (0, 0) or (1, 1) or (2, 2) => _gamma * _hx * _hy * _hz / 36.0D * D[i % 4,
66                     ↪ j % 4],
67                 (0, 1) or (0, 2) or (1, 0) or (1, 2) or (2, 0) or (2, 1) => 0.0D,
68                 _ => throw new ArgumentOutOfRangeException("Out of local matrix 3d
69                     ↪ range"),
70             };
71         }
72         set
73         {}
74     }
75 }

```

## LocalVector3D.cs

```

1 using static Functions.Function;
2
3 namespace MathObjects;
4
5
6 public class LocalVector3D(Egetter egetter, double x0, double x1, double y0, double y1,
7     ↪ double z0, double z1, double t) : Vector
8 {
9     private readonly double x0 = x0;
10    private readonly double x1 = x1;
11    private readonly double y0 = y0;
12    private readonly double y1 = y1;
13    private readonly double z0 = z0;
14    private readonly double z1 = z1;
15    private readonly double xm = 0.5D * (x1 + x0);
16    private readonly double ym = 0.5D * (y1 + y0);
17    private readonly double zm = 0.5D * (z1 + z0);
18    private readonly double t = t;
19
20    private readonly LocalMatrixM3D M = new(1.0, x1 - x0, y1 - y0, z1 - z0);
21    private readonly Egetter _egetter = egetter;

```

```

22 public override double this[int i]
23 {
24     get
25     {
26         static double ScalarMult((double, double, double) a, (double, double, double)
           ↪ b)
27         => a.Item1 * b.Item1 + a.Item2 * b.Item2 + a.Item3 * b.Item3;
28         List<double> vect = [
29             ScalarMult(_egetter(xm, y0, z0, t), (1.0D, 0.0D, 0.0D)),
30             ScalarMult(_egetter(xm, y1, z0, t), (1.0D, 0.0D, 0.0D)),
31             ScalarMult(_egetter(xm, y0, z1, t), (1.0D, 0.0D, 0.0D)),
32             ScalarMult(_egetter(xm, y1, z1, t), (1.0D, 0.0D, 0.0D)),
33             ScalarMult(_egetter(x0, ym, z0, t), (0.0D, 1.0D, 0.0D)),
34             ScalarMult(_egetter(x1, ym, z0, t), (0.0D, 1.0D, 0.0D)),
35             ScalarMult(_egetter(x0, ym, z1, t), (0.0D, 1.0D, 0.0D)),
36             ScalarMult(_egetter(x1, ym, z1, t), (0.0D, 1.0D, 0.0D)),
37             ScalarMult(_egetter(x0, y0, zm, t), (0.0D, 0.0D, 1.0D)),
38             ScalarMult(_egetter(x1, y0, zm, t), (0.0D, 0.0D, 1.0D)),
39             ScalarMult(_egetter(x0, y1, zm, t), (0.0D, 0.0D, 1.0D)),
40             ScalarMult(_egetter(x1, y1, zm, t), (0.0D, 0.0D, 1.0D))
41         ];
42         double ans = 0.0D;
43         for (int j = 0; j < vect.Count; j++)
44             ans += M[i, j] * vect[j];
45         return ans;
46     }
47 }
48 }

```

## MathOperations.cs

```

1 using System.Threading.Tasks;
2
3 namespace MathObjects;
4
5 public static class MathOperations
6 {
7     public static GlobalVector Diff(GlobalVector gv1, GlobalVector gv2)
8     {
9         if (gv1.Size != gv2.Size)
10             throw new Exception("Найти разность векторов не возможно, т.к. они имеют
           ↪ разные размеры.");
11
12         GlobalVector gv = new(gv1.Size);
13         for (int i = 0; i < gv.Size; i++)
14             gv[i] = gv1[i] - gv2[i];
15
16         return gv;
17     }
18
19     public static GlobalVector DiffPar(GlobalVector gv1, GlobalVector gv2)
20     {
21         if (gv1.Size != gv2.Size)
22             throw new Exception("Найти разность векторов не возможно, т.к. они имеют
           ↪ разные размеры.");
23         GlobalVector gv = new(gv1.Size);
24         Parallel.For(0, gv1.Size, i => {
25             gv[i] = gv1[i] - gv2[i];
26         });
27         return gv;
28     }
29
30     public static GlobalVector Sum(GlobalVector gv1, GlobalVector gv2)
31     {
32         if (gv1.Size != gv2.Size)
33             throw new Exception("Найти разность векторов не возможно, т.к. они имеют
           ↪ разные размеры.");
34
35         GlobalVector gv = new(gv1.Size);
36         for (int i = 0; i < gv.Size; i++)
37             gv[i] = gv1[i] + gv2[i];
38

```

```

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
46             ↪ разные размеры.");
47
48     GlobalVector gv = new(gv1.Size);
49     Parallel.For(0, gv1.Size, i => {
50         gv[i] = gv1[i] + gv2[i];
51     });
52     return gv;
53 }
54
55 public static GlobalVector Multiply(double a, GlobalVector gv)
56 {
57     GlobalVector _gv = new(gv.Size);
58     for (int i = 0; i < gv.Size; i++)
59         _gv[i] = a * gv[i];
60     return _gv;
61 }
62
63 public static GlobalVector MultiplyPar(double a, GlobalVector gv)
64 {
65     GlobalVector _gv = new(gv.Size);
66     Parallel.For(0, gv.Size, i => {
67         _gv[i] = a * gv[i];
68     });
69     return _gv;
70 }
71
72 public static double Multiply(GlobalVector gv1, GlobalVector gv2)
73 {
74     if (gv1.Size != gv2.Size) throw new Exception("Невозможно найти скалярное
75         ↪ умножение векторов, из-за разности в размерах.");
76     double ans = 0.0D;
77     for (int i = 0; i < gv1.Size; i++)
78         ans += gv1[i] * gv2[i];
79     return ans;
80 }
81
82 public static double MultiplyPar(GlobalVector gv1, GlobalVector gv2)
83 {
84     if (gv1.Size != gv2.Size) throw new Exception("Невозможно найти скалярное
85         ↪ умножение векторов, из-за разности в размерах.");
86     double ans = 0.0D;
87     Parallel.For(0, gv1.Size, i => {
88         ans = gv1[i] * gv2[i];
89     });
90     return ans;
91 }
92
93 public static GlobalVector Multiply(GlobalMatrix _gm, GlobalVector _gv)
94 {
95     if (_gm.Size != _gv.Size)
96         throw new Exception("Невозможно перемножить матрицу на вектор.");
97     GlobalVector ans = new(_gv.Size);
98
99     for (int i = 0; i < _gv.Size; i++)
100     {
101         for (int j = 0; j < _gm._ig[i + 1] - _gm._ig[i]; j++)
102         {
103             ans[i] += _gm._al[_gm._ig[i] + j] * _gv[_gm._jg[_gm._ig[i] + j]];
104             ans[_gm._jg[_gm._ig[i] + j]] += _gm._au[_gm._ig[i] + j] * _gv[i];
105         }
106         ans[i] += _gm._diag[i] * _gv[i];
107     }
108     return ans;
109 }
110
111 public static GlobalVector CustomMultiply(GlobalMatrix _gm, GlobalVector _gv)
112 {
113     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
182             ⇨ portrait!");
183         GlobalMatrix ans = new(gm1);
184         for (int i = 0; i < ans.Size; i++)
185         {
186             for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)

```

```

187         ans._al[ans._ig[i] + j] += gm2._al[gm2._ig[i] + j];
188         ans._au[ans._ig[i] + j] += gm2._au[gm2._ig[i] + j];
189     }
190     ans._diag[i] += gm2._diag[i];
191 }
192 return ans;
193 }
194
195
196 public static GlobalMatrix SumPar(GlobalMatrix gm1, GlobalMatrix gm2)
197 {
198     if (!gm1.CheckPortrait(gm2)) throw new ArgumentException("Different matrixes
199     ↪ portrait!");
200     GlobalMatrix ans = new(gm1);
201     Parallel.For(0, ans.Size, i => {
202         for (int j = 0; j < ans._ig[i + 1] - ans._ig[i]; j++)
203         {
204             ans._al[ans._ig[i] + j] += gm2._al[gm2._ig[i] + j];
205             ans._au[ans._ig[i] + j] += gm2._au[gm2._ig[i] + j];
206         }
207         ans._diag[i] += gm2._diag[i];
208     });
209     return ans;
210 }
211 }

```

## LULOS.cs

```

1 public class LU_LOS(int maxIter = 100_000, double eps = 1E-15) : ISolver
2 {
3     private int _maxIter = maxIter;
4     private double _eps = eps;
5
6     private static void PartitialLU(GlobalMatrix A)
7     {
8         for (int i = 0; i < A.Size; i++)
9         {
10             for (int j = A._ig[i]; j < A._ig[i + 1]; j++)
11             {
12                 int jCol = A._jg[j];
13                 int jk = A._ig[jCol];
14                 int k = A._ig[i];
15                 int sdvig = A._jg[A._ig[i]] - A._jg[A._ig[jCol]];
16                 if (sdvig > 0) jk += sdvig;
17                 else k -= sdvig;
18                 double sumL = 0.0;
19                 double sumU = 0.0;
20                 for (; k < j && jk < A._ig[jCol + 1]; k++, jk++)
21                 {
22                     sumL += A._al[k] * A._au[jk];
23                     sumU += A._au[k] * A._al[jk];
24                 }
25                 A._al[j] -= sumL;
26                 A._au[j] -= sumU;
27                 A._au[j] /= A._diag[jCol];
28             }
29             double sumD = 0.0;
30             for (int j = A._ig[i]; j < A._ig[i + 1]; j++)
31                 sumD += A._al[j] * A._au[j];
32             A._diag[i] -= sumD;
33         }
34     }
35
36     private static GlobalVector Forward(GlobalMatrix Matrix, GlobalVector b)
37     {
38         var result = new GlobalVector(b);
39         for (int i = 0; i < Matrix.Size; i++)
40         {
41             for (int j = Matrix._ig[i]; j < Matrix._ig[i + 1]; j++)
42                 result[i] -= Matrix._al[j] * result[Matrix._jg[j]];
43             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     PartititalLU(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() *
90                 ↪ r0.Norma()):E15}");
91     } while (iter < _maxIter && (r.Norma() * r.Norma()) / (r0.Norma() * r0.Norma())
92         ↪ >= _eps * _eps);
93     Console.WriteLine(
94         $"@Computing finished!
95         Total iterations: {iter}
96         Relative residuality: {r.Norma() / b.Norma():E15}");
97     return (x, x_);
98 }

```