



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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



**НГТУ
НЭТИ** | **Факультет прикладной
математики и информатики**

Кафедра прикладной математики

Лабораторная работа №3
по дисциплине «Уравнения математической физики»

РЕШЕНИЕ ГАРМОНИЧЕСКИХ ЗАДАЧ

Студенты БЕГИЧЕВ АЛЕКСАНДР

ШИШКИН НИКИТА

Группа ПМ-92

Преподаватели ЗАДОРОВ ЖЫН А. Г.

ПАТРУШЕВ И. И.

ПЕРСОВА М. Г.

Новосибирск, 2022

Цель работы

Разработать программу решения гармонической задачи методом конечных элементов. Провести сравнение прямого и итерационного методов решения получаемой в результате конечноэлементной аппроксимации СЛАУ.

Вариант: Решить трехмерную гармоническую задачу в декартовых координатах, базисные функции-трилинейные.

Теоретическая часть

Постановка задачи

Рассмотрим задачу для уравнения

$$\chi \frac{\partial u}{\partial t^2} + \sigma \frac{\partial u}{\partial t} - \operatorname{div}(\lambda \operatorname{grad} u) = f,$$

в котором правая часть f представима в виде:

$$f(x, y, z, t) = f^s(x, y, z) \sin \omega t + f^c(x, y, z) \cos \omega t.$$

Если остальные параметры рассматриваемого уравнения не зависят от времени, то тогда и его решения u может быть представлено в виде:

$$u(x, y, z, t) = u^s(x, y, z) \sin \omega t + u^c(x, y, z) \cos \omega t,$$

где u^s и u^c – две зависящие только от пространственных координат функции, удовлетворяющие системе уравнений:

$$\begin{cases} -\operatorname{div}(\lambda \operatorname{grad} u^s) - \omega \sigma u^c - \omega^2 \chi u^s = f^s, \\ -\operatorname{div}(\lambda \operatorname{grad} u^c) + \omega \sigma u^s - \omega^2 \chi u^c = f^c. \end{cases}$$

То же самое можно сказать и о решении краевой задачи, если не только f , но и параметры краевых условий являются гармонически изменяющимися по времени с одной и той же частотой ω :

$$u_g(x, y, z, t) = u_g^s(x, y, z) \sin \omega t + u_g^c(x, y, z) \cos \omega t,$$

$$\theta(x, y, z, t) = \theta^s(x, y, z) \sin \omega t + \theta^c(x, y, z) \cos \omega t,$$

$$u_\beta(x, y, z, t) = u_\beta^s(x, y, z) \sin \omega t + u_\beta^c(x, y, z) \cos \omega t.$$

В этом случае функции u^s и u^c должны удовлетворять краевым условиям:

$$u^s|_{S_1} = u_g^s, \quad u^c|_{S_1} = u_g^c,$$

$$\lambda \frac{\partial u^s}{\partial n} \Big|_{S_2} = \theta^s, \quad \lambda \frac{\partial u^c}{\partial n} \Big|_{S_2} = \theta^c$$

$$\lambda \frac{\partial u^s}{\partial n} \Big|_{S_2} + \beta (u^s|_{S_3} - u_\beta^s) = 0, \quad \lambda \frac{\partial u^c}{\partial n} \Big|_{S_2} + \beta (u^c|_{S_3} - u_\beta^c) = 0$$

Конечноэлементная аппроксимация

Выполним конечноэлементную аппроксимацию краевой задачи.

Сначала получим эквивалентную вариационную формулировку. Для этого умножим каждое из уравнений системы на пробную функцию v и применим формулу Грина (интегрирования по частям). В результате получим систему двух вариационных уравнений:

$$\begin{cases} \int_{\Omega} (\lambda \operatorname{grad} u^s \operatorname{grad} v - \omega \sigma u^c v - \omega^2 \chi u^s v) d\Omega + \int_{S_3} \beta u^s v dS = \\ = \int_{\Omega} f^s v d\Omega + \int_{S_2} \theta^s v dS + \int_{S_3} \beta u_{\beta}^s v dS \\ \int_{\Omega} (\lambda \operatorname{grad} u^c \operatorname{grad} v - \omega \sigma u^s v - \omega^2 \chi u^c v) d\Omega + \int_{S_3} \beta u^c v = \\ = \int_{\Omega} f^c v d\Omega + \int_{S_2} \theta^c v dS + \int_{S_3} \beta u_{\beta}^c v dS. \end{cases}$$

Уравнения должны выполняться для любой $v \in H^1$, удовлетворяющей однородному первому краевому условию на границе S_1 .

Построим конечноэлементную аппроксимацию на основе вариационной формулировки. Пусть $\{\psi_i\}$ – набор базисных функций. Чтобы получить конечноэлементную СЛАУ, заменим в каждую из искоемых функций u^s и u^c на функции $u^{s,h} = \sum_{i=1}^n q_i^s \psi_i$ и $u^{c,h} = \sum_{i=1}^n q_i^c \psi_i$, а вместо пробной функции подставим поочередно базисные функции ψ_i :

$$\begin{cases} \sum_{j=1}^n \left(\int_{\Omega} (\lambda \operatorname{grad} \psi_i \operatorname{grad} \psi_j - \omega^2 \chi \psi_i \psi_j) d\Omega + \int_{S_3} \beta \psi_i \psi_j \right) q_j^s - \omega \sum_{j=1}^n \left(\int_{\Omega} \sigma \psi_i \psi_j d\Omega \right) q_j^c = \\ = \int_{\Omega} f^s \psi_i d\Omega + \int_{S_2} \theta^s \psi_i dS + \int_{S_3} \beta u_{\beta}^s \psi_i dS, \quad i = 1 \dots n, \\ \sum_{j=1}^n \left(\int_{\Omega} (\lambda \operatorname{grad} \psi_i \operatorname{grad} \psi_j - \omega^2 \chi \psi_i \psi_j) d\Omega + \int_{S_3} \beta \psi_i \psi_j \right) q_j - \omega \sum_{j=1}^n \left(\int_{\Omega} \sigma \psi_i \psi_j d\Omega \right) q_j^s = \\ = \int_{\Omega} f^c \psi_i d\Omega + \int_{S_2} \theta^c \psi_i dS + \int_{S_3} \beta u_{\beta}^c \psi_i dS, \quad i = 1 \dots n. \end{cases}$$

В результате мы получили систему из $2n$ уравнений с $2n$ неизвестными q_j^s и q_j^c . Чтобы определить матрицу и вектор правой части полученной конечноэлементной СЛАУ, пронумеруем ее уравнения как $2i - 1$ и $2i$. Соответственно пронумеруем и неизвестные этой системы, которые обозначим q_j , $j = 1 \dots 2n$: $q_{2j-1} = q_j^s$, $q_{2j} = q_j^c$, $j = 1 \dots n$.

Обозначим

$$p_{ij} = \int_{\Omega} (\lambda \operatorname{grad} \psi_i \operatorname{grad} \psi_j - \omega^2 \chi \psi_i \psi_j) d\Omega + \int_{S_3} \beta \psi_i \psi_j,$$

$$c_{ij} = \omega \int_{\Omega} \sigma \psi_i \psi_j d\Omega.$$

Тогда матрица конечноэлементной СЛАУ будет иметь следующую структуру:

$$A = \begin{pmatrix} p_{11} & -c_{11} & p_{12} & -c_{12} & \dots & \dots & p_{1n} & -c_{1n} \\ c_{11} & p_{11} & c_{12} & p_{12} & \dots & \dots & c_{1n} & p_{1n} \\ p_{21} & -c_{21} & p_{22} & -c_{22} & \dots & \dots & p_{2n} & -c_{2n} \\ c_{21} & p_{21} & c_{22} & p_{22} & \dots & \dots & c_{2n} & p_{2n} \\ & & & & \ddots & & & \\ & & & & & \ddots & & \\ p_{n1} & -c_{n1} & p_{n2} & -c_{n2} & \dots & \dots & p_{nn} & -c_{nn} \\ c_{n1} & p_{n1} & c_{n2} & p_{n2} & \dots & \dots & c_{nn} & p_{nn} \end{pmatrix}.$$

Очевидно, что в этой матрице выделяются блоки размера 2 x 2 вида:

$$A^{ij} = \begin{pmatrix} p_{ij} & -c_{ij} \\ c_{ij} & p_{ij} \end{pmatrix},$$

для хранения которых достаточно только двух ячеек памяти. Таким образом, для хранения всей матрицы в блочном формате с учетом структуры блока A^{ij} требуется практически в два раза меньше памяти, чем при хранении ее покомпонентно. Поэтому при программной реализации в таких случаях, как правило, используются блочные форматы хранения данных, в том числе и с учетом разреженной структуры матрицы.

Практическая часть

1. Выполнить конечноэлементную аппроксимацию исходного уравнения в соответствии с заданием. Получить формулы для вычисления компонент матрицы A и вектора правой части b для метода простой итерации.
2. Реализовать программу для решения гармонической задачи.
3. Протестировать разработанную программу на полиномах первой степени.
4. Провести исследования реализованных методов для сеток с небольшим количеством узлов 500 - 1000 и большим количеством узлов - порядка 20000 - 50000 для различных значений параметров $10^{-4} \leq \omega \leq 10^9$, $10^2 \leq \lambda \leq 8 \cdot 10^5$, $0 \leq \sigma \leq 10^8$, $8.81 \cdot 10^{-12} \leq \chi \leq 10^{-10}$. Для всех решенных задач сравнить вычислительные затраты, требуемые для решения СЛАУ итерационным и прямым методом.

Описание программы

Программа состоит из нескольких модулей:

- Класс FEM, в котором происходят основные вычислительные операции.
- Класс SparseMatrix для представления матрицы в разреженном формате.
- Класс Matrix для представления матрицы в плотном формате.
- Класс Vector для представления вектора.
- Абстрактный класс Solver и его класс-наследник LOSLU для решения СЛАУ итерационным методом.
- Абстрактный класс Decomposer и его класс-наследник DecomposerLDU для решения СЛАУ прямым методом.
- Структура Interval для представления интервала.
- Статические классы Integration и Quadratures для интегрирования.
- Статический класс LinearBasis, в котором записаны базисные функции - трилинейные.
- Статический класс ArrayHelper для методов расширения одномерного массива (копирование, заполнение).
- Класс GridParameters для различных параметров сетки.
- Класс GridFactory для генерации сеток через фабричный метод.
- Абстрактный класс Grid и классы-наследники, которые его реализуют RegularGrid, IrregularGrid для представления сеток.
- Структура Point3D для представления трехмерной точки.
- Интерфейс ITest и классы, которые его реализуют для тестирования.

Тестирование

Исследования для сетки с небольшим количеством узлов

Функция:

- $u^s = x + y + z$
- $u^c = x - y - z.$

Правая часть:

- $f^s = -\omega\sigma(x + y + z) - \omega^2\chi(x - y - z)$
- $f^c = \omega\sigma(x - y - z) - \omega^2\chi(x + y + z).$

Заданы краевые условия первого рода.

Изменение параметра ω

Коэффициенты:

- $\sigma = 2$
- $\chi = 10^{-11}$
- $\lambda = 110.$

ω	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
$1 \cdot 10^{-4}$	16	46	$1.62 \cdot 10^{-11}$	$8.16 \cdot 10^{-12}$	$1.74 \cdot 10^{-16}$	$7.13 \cdot 10^{-17}$
$1 \cdot 10^{-3}$	5	43	$1.6 \cdot 10^{-11}$	$8.4 \cdot 10^{-12}$	$1.84 \cdot 10^{-16}$	$8.65 \cdot 10^{-17}$
$1 \cdot 10^{-2}$	5	43	$1.53 \cdot 10^{-11}$	$8.51 \cdot 10^{-12}$	$2.53 \cdot 10^{-16}$	$7.37 \cdot 10^{-17}$
0.1	5	42	$6.49 \cdot 10^{-11}$	$2.66 \cdot 10^{-11}$	$8.42 \cdot 10^{-16}$	$2.9 \cdot 10^{-16}$
1	7	51	$7.37 \cdot 10^{-11}$	$4.67 \cdot 10^{-11}$	$3.36 \cdot 10^{-16}$	$1.34 \cdot 10^{-16}$
10	6	51	$5.65 \cdot 10^{-11}$	$2.08 \cdot 10^{-10}$	$3.86 \cdot 10^{-16}$	$2.79 \cdot 10^{-16}$
100	4	42	$1.75 \cdot 10^{-11}$	$1.6 \cdot 10^{-11}$	$1.71 \cdot 10^{-16}$	$7.98 \cdot 10^{-17}$
1,000	4	49	$3.56 \cdot 10^{-11}$	$1.32 \cdot 10^{-11}$	$3.97 \cdot 10^{-16}$	$1.32 \cdot 10^{-16}$
10,000	7	50	$6.85 \cdot 10^{-12}$	$4.37 \cdot 10^{-12}$	$3.11 \cdot 10^{-15}$	$1.78 \cdot 10^{-16}$
$1 \cdot 10^5$	13	45	$5.22 \cdot 10^{-11}$	$3.33 \cdot 10^{-11}$	$3.46 \cdot 10^{-16}$	$1.5 \cdot 10^{-16}$
$1 \cdot 10^6$	4	41	$5.22 \cdot 10^{-11}$	$3.33 \cdot 10^{-11}$	$3.56 \cdot 10^{-16}$	$1.14 \cdot 10^{-16}$
$1 \cdot 10^7$	4	41	$5.22 \cdot 10^{-11}$	$3.33 \cdot 10^{-11}$	$3.63 \cdot 10^{-16}$	$1.04 \cdot 10^{-16}$
$1 \cdot 10^8$	4	41	$5.22 \cdot 10^{-11}$	$3.33 \cdot 10^{-11}$	$3.19 \cdot 10^{-16}$	$1.08 \cdot 10^{-16}$
$1 \cdot 10^9$	4	41	$5.22 \cdot 10^{-11}$	$3.33 \cdot 10^{-11}$	$3.41 \cdot 10^{-16}$	$1.21 \cdot 10^{-16}$

Изменение параметра λ

Коэффициенты:

- $\omega = 1$
- $\sigma = 2$
- $\chi = 10^{-11}$

λ	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
100	6	48	$7.89 \cdot 10^{-11}$	$6.77 \cdot 10^{-11}$	$6 \cdot 10^{-16}$	$2.05 \cdot 10^{-16}$
1,000	5	42	$7.59 \cdot 10^{-11}$	$3.26 \cdot 10^{-11}$	$3.9 \cdot 10^{-16}$	$1.29 \cdot 10^{-16}$
10,000	6	51	$1.53 \cdot 10^{-11}$	$8.46 \cdot 10^{-12}$	$1.79 \cdot 10^{-16}$	$7.49 \cdot 10^{-17}$
$1 \cdot 10^5$	5	42	$1.6 \cdot 10^{-11}$	$8.42 \cdot 10^{-12}$	$1.83 \cdot 10^{-16}$	$6.36 \cdot 10^{-17}$
$8 \cdot 10^5$	5	45	$1.62 \cdot 10^{-11}$	$8.17 \cdot 10^{-12}$	$1.21 \cdot 10^{-16}$	$5.19 \cdot 10^{-17}$

Изменение параметра σ

Коэффициенты:

- $\omega = 1$
- $\chi = 10^{-11}$
- $\lambda = 110$.

σ	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
0	2	42	0.5	0.23	NaN	NaN
10	6	47	$3.47 \cdot 10^{-10}$	$1.45 \cdot 10^{-10}$	$3.31 \cdot 10^{-16}$	$1.76 \cdot 10^{-16}$
100	5	42	$3.74 \cdot 10^{-11}$	$1.06 \cdot 10^{-10}$	$1.68 \cdot 10^{-16}$	$1.79 \cdot 10^{-16}$
1,000	4	45	$9.42 \cdot 10^{-11}$	$1.03 \cdot 10^{-10}$	$2.72 \cdot 10^{-16}$	$1.35 \cdot 10^{-16}$
10,000	5	46	$2.6 \cdot 10^{-11}$	$1.53 \cdot 10^{-11}$	$1.56 \cdot 10^{-15}$	$1.57 \cdot 10^{-16}$
$1 \cdot 10^5$	4	42	$6.3 \cdot 10^{-12}$	$1.79 \cdot 10^{-12}$	$1.38 \cdot 10^{-14}$	$1.68 \cdot 10^{-16}$
$1 \cdot 10^6$	5	42	$2.47 \cdot 10^{-13}$	$8.41 \cdot 10^{-14}$	$1.54 \cdot 10^{-13}$	$1.53 \cdot 10^{-16}$
$1 \cdot 10^7$	5	42	$3.34 \cdot 10^{-13}$	$3.33 \cdot 10^{-15}$	$1.39 \cdot 10^{-12}$	$1.39 \cdot 10^{-16}$
$1 \cdot 10^8$	7	51	$3.82 \cdot 10^{-12}$	$2.33 \cdot 10^{-16}$	$1.37 \cdot 10^{-11}$	$1.58 \cdot 10^{-16}$

Изменение параметра χ

Коэффициенты:

- $\omega = 1$
- $\sigma = 2$
- $\lambda = 110$.

χ	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
$1 \cdot 10^{-12}$	5	44	$1.52 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$1.58 \cdot 10^{-16}$	$8.3 \cdot 10^{-17}$
$1 \cdot 10^{-11}$	5	45	$1.52 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$3.9 \cdot 10^{-16}$	$1.28 \cdot 10^{-16}$
$1 \cdot 10^{-10}$	5	42	$1.52 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$3.92 \cdot 10^{-16}$	$1.6 \cdot 10^{-16}$

Исследования для сетки с большим количеством узлов

Функция:

- $u^s = 3x - 2y + z$
- $u^c = 2x + y - z.$

Правая часть:

- $f^s = -\omega\sigma(2x + y - z) - \omega^2\chi(3x - 2y + z)$
- $f^c = \omega\sigma(3x - 2y + z) - \omega^2\chi(2x + y - z).$

Заданы краевые условия первого рода.

Изменение параметра ω

Коэффициенты:

- $\sigma = 2$
- $\chi = 10^{-11}$
- $\lambda = 110.$

ω	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
$1 \cdot 10^{-4}$	69	324	$5.16 \cdot 10^{-11}$	$3.57 \cdot 10^{-11}$	$2.32 \cdot 10^{-16}$	$2.11 \cdot 10^{-16}$
$1 \cdot 10^{-3}$	59	310	$5.14 \cdot 10^{-11}$	$3.75 \cdot 10^{-11}$	$3.51 \cdot 10^{-16}$	$2.6 \cdot 10^{-16}$
$1 \cdot 10^{-2}$	60	312	$1.12 \cdot 10^{-10}$	$3.42 \cdot 10^{-11}$	$4.32 \cdot 10^{-16}$	$3.69 \cdot 10^{-16}$
0.1	64	301	$3.9 \cdot 10^{-11}$	$1.91 \cdot 10^{-11}$	$2 \cdot 10^{-15}$	$1.91 \cdot 10^{-15}$
1	82	296	$1.11 \cdot 10^{-10}$	$4.14 \cdot 10^{-11}$	$9.01 \cdot 10^{-16}$	$6.9 \cdot 10^{-16}$
10	113	281	$9.09 \cdot 10^{-11}$	$1.6 \cdot 10^{-11}$	$8.28 \cdot 10^{-16}$	$3.55 \cdot 10^{-16}$
100	41	321	$1.34 \cdot 10^{-11}$	$1.44 \cdot 10^{-11}$	$2.07 \cdot 10^{-16}$	$2.51 \cdot 10^{-16}$
1,000	39	279	$1.42 \cdot 10^{-11}$	$1.24 \cdot 10^{-11}$	$8.56 \cdot 10^{-16}$	$4.87 \cdot 10^{-16}$
10,000	38	272	$2.35 \cdot 10^{-12}$	$1.97 \cdot 10^{-12}$	$8.28 \cdot 10^{-15}$	$4.98 \cdot 10^{-16}$
$1 \cdot 10^5$	47	170	$1.99 \cdot 10^{-11}$	$1.13 \cdot 10^{-11}$	$4.67 \cdot 10^{-16}$	$3.68 \cdot 10^{-16}$
$1 \cdot 10^6$	43	248	$1.17 \cdot 10^{-12}$	$2.77 \cdot 10^{-12}$	$8.49 \cdot 10^{-13}$	$4.96 \cdot 10^{-16}$
$1 \cdot 10^7$	16	261	$3.68 \cdot 10^{-12}$	$3.02 \cdot 10^{-12}$	$4.61 \cdot 10^{-11}$	$9.76 \cdot 10^{-16}$
$1 \cdot 10^8$	20	819	$3.68 \cdot 10^{-12}$	$3.02 \cdot 10^{-12}$	$1.12 \cdot 10^{-12}$	$4.64 \cdot 10^{-16}$
$1 \cdot 10^9$	18	794	$3.68 \cdot 10^{-12}$	$3.02 \cdot 10^{-12}$	$1.08 \cdot 10^{-13}$	$4.7 \cdot 10^{-16}$

Изменение параметра λ

Коэффициенты:

- $\omega = 1$
- $\sigma = 2$
- $\chi = 10^{-11}$

λ	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
100	105	809	$1.49 \cdot 10^{-10}$	$7.88 \cdot 10^{-11}$	$1.51 \cdot 10^{-15}$	$1.08 \cdot 10^{-15}$
1,000	87	818	$5.33 \cdot 10^{-11}$	$1.76 \cdot 10^{-11}$	$9.35 \cdot 10^{-16}$	$8.48 \cdot 10^{-16}$
10,000	65	801	$1.17 \cdot 10^{-10}$	$4.01 \cdot 10^{-11}$	$3.44 \cdot 10^{-16}$	$3.14 \cdot 10^{-16}$
$1 \cdot 10^5$	66	184	$5.15 \cdot 10^{-11}$	$3.77 \cdot 10^{-11}$	$2.18 \cdot 10^{-16}$	$1.78 \cdot 10^{-16}$
$8 \cdot 10^5$	73	163	$5.15 \cdot 10^{-11}$	$3.57 \cdot 10^{-11}$	$4.86 \cdot 10^{-16}$	$3.94 \cdot 10^{-16}$

Изменение параметра σ

Коэффициенты:

- $\omega = 1$
- $\chi = 10^{-11}$
- $\lambda = 110$.

σ	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
0	18	577	0.37	0.3	NaN	NaN
10	127	997	$5.32 \cdot 10^{-11}$	$1.48 \cdot 10^{-10}$	$8.94 \cdot 10^{-16}$	$2.76 \cdot 10^{-16}$
100	58	140	$8.92 \cdot 10^{-12}$	$6.44 \cdot 10^{-12}$	$2.51 \cdot 10^{-16}$	$2.58 \cdot 10^{-16}$
1,000	40	368	$3.37 \cdot 10^{-12}$	$5.3 \cdot 10^{-12}$	$4.46 \cdot 10^{-16}$	$4.08 \cdot 10^{-16}$
10,000	39	407	$4.99 \cdot 10^{-12}$	$3.47 \cdot 10^{-12}$	$4.29 \cdot 10^{-15}$	$5.34 \cdot 10^{-16}$
$1 \cdot 10^5$	44	851	$1.6 \cdot 10^{-13}$	$5.46 \cdot 10^{-14}$	$4.26 \cdot 10^{-14}$	$5.02 \cdot 10^{-16}$
$1 \cdot 10^6$	46	189	$5.95 \cdot 10^{-14}$	$2.72 \cdot 10^{-15}$	$4.37 \cdot 10^{-13}$	$4.99 \cdot 10^{-16}$
$1 \cdot 10^7$	49	200	$5.84 \cdot 10^{-13}$	$2.81 \cdot 10^{-15}$	$4.28 \cdot 10^{-12}$	$5.09 \cdot 10^{-16}$
$1 \cdot 10^8$	53	196	$6.07 \cdot 10^{-12}$	$3.04 \cdot 10^{-16}$	$4.14 \cdot 10^{-11}$	$4.8 \cdot 10^{-16}$

Изменение параметра χ

Коэффициенты:

- $\omega = 1$
- $\sigma = 2$
- $\lambda = 110$.

χ	LOSLU (ms)	LDU (ms)	Погрешность u^s (LOS)	Погрешность u^c (LOS)	Погрешность u^s (LDU)	Погрешность u^c (LDU)
$1 \cdot 10^{-12}$	137	513	$9.99 \cdot 10^{-11}$	$2.83 \cdot 10^{-11}$	$3.98 \cdot 10^{-16}$	$3.36 \cdot 10^{-16}$
$1 \cdot 10^{-11}$	75	324	$9.99 \cdot 10^{-11}$	$2.83 \cdot 10^{-11}$	$8.8 \cdot 10^{-16}$	$7.43 \cdot 10^{-16}$
$1 \cdot 10^{-10}$	86	254	$9.99 \cdot 10^{-11}$	$2.83 \cdot 10^{-11}$	$1.16 \cdot 10^{-15}$	$9.05 \cdot 10^{-16}$

Проведенные исследования и выводы

В целом, результаты везде одинаковые. При $\sigma = 0$ мы не смогли получить решение. LDU работает в разы дольше, чем LOS с LU предобуславливанием.

Тексты основных модулей

Program.cs

```
1 using eMP_3;
2
3 GridFactory gridFactory = new();
4 var grid = gridFactory.CreateGrid(GridTypes.Regular,
5   ↪ GridParameters.ReadJson("grid.jsonc")!.Value);
6
7 #region Итерационный метод
8 FEM fem1 = FEM.CreateBuilder().SetSpaceGrid(grid).SetTest(new
9   ↪ Test1()).SetSolverSLAE(new LOSLU(1000, 1E-14));
10 fem1.Compute();
11 #endregion
12
13 #region Прямой метод
14 FEM fem2 = FEM.CreateBuilder().SetSpaceGrid(grid).SetTest(new
15   ↪ Test1()).SetDecomposer(new DecomposerLDU());
16 fem2.Compute();
17 #endregion
```

FEM.cs

```
1 namespace eMP_3;
2
3 public class FEM {
4     public class FEMBuilder {
5         private readonly FEM _fem = new();
6
7         public FEMBuilder SetTest(ITest test) {
8             _fem._test = test;
9             return this;
10        }
11
12        public FEMBuilder SetSpaceGrid(Grid grid) {
13            _fem._grid = grid;
14            return this;
15        }
16
17        public FEMBuilder SetSolverSLAE(Solver solver) {
18            _fem._solver = solver;
19            return this;
20        }
21
22        public FEMBuilder SetDecomposer(Decomposer decomposer) {
23            _fem._decomposer = decomposer;
24            return this;
25        }
26
27        public static implicit operator FEM(FEMBuilder builder)
28            => builder._fem;
```

```

29     }
30
31     // default! указывает на то, что данное поле не может принимать null
32     private delegate double Basis(Point3D point);
33     private Basis[] _basis = default!;
34     private ITest _test = default!;
35     private Grid _grid = default!;
36     private Solver? _solver;
37     private Decomposer? _decomposer;
38     private Matrix _massMatrix = default!; // матрица масс
39     private Matrix _stiffnessMatrix = default!; // матрица жесткости
40     private SparseMatrix _globalMatrix = default!;
41     private Vector<double> _localVector1 = default!;
42     private Vector<double> _localVector2 = default!;
43     private Vector<double> _vector = default!; // вектор правой части
44
45     public void Compute() {
46         try {
47             ArgumentNullException.ThrowIfNull(_test, $"{nameof(_test)} cannot be
48                 ↪ null, set the test");
49
50             if (_solver is null && _decomposer is null) {
51                 throw new ArgumentNullException(nameof(_solver), "Set the method of
52                     ↪ solving SLAE");
53             }
54
55             Init();
56             Solve();
57         } catch (Exception ex) {
58             Console.WriteLine($"{We had problem: {ex.Message}");
59         }
60     }
61
62     private void Init() {
63         _massMatrix = new(8);
64         _stiffnessMatrix = new(8);
65         _localVector1 = new(8);
66         _localVector2 = new(8);
67
68         _basis = new Basis[] { LinearBasis.Psi1, LinearBasis.Psi2, LinearBasis.Psi3,
69             LinearBasis.Psi4, LinearBasis.Psi5, LinearBasis.Psi6,
70             LinearBasis.Psi7, LinearBasis.Psi8};
71     }
72
73     private void InitSLAE(int sizeOffDiag) {
74         _globalMatrix = new(2 * _grid.Points.Length, sizeOffDiag); // resizing in
75             ↪ method ConstructPortrait()
76         _vector = new(2 * _grid.Points.Length);
77     }
78
79     private void Solve() {
80         ConstructPortrait();
81         AssemblySLAE();
82
83         AccountingDirichletBoundary();
84         _globalMatrix.PrintDense("matrix.txt");
85
86         if (_decomposer is not null) {
87             _globalMatrix.AsProfileMatrix();
88             _decomposer.SetMatrix(_globalMatrix);
89         }
90     }

```

```

86         _decomposer.SetVector(_vector);
87         _decomposer.Compute();
88         Console.WriteLine(_decomposer.RunningTime);
89         ErrForward();
90     } else {
91         _solver!.SetMatrix(_globalMatrix);
92         _solver.SetVector(_vector);
93         _solver.Compute();
94         Console.WriteLine(_solver.RunningTime);
95         ErrIter();
96     }
97 }
98
99 private void ConstructPortrait() {
100     List<int>[] list = new List<int>[_grid.Points.Length].Select(_ => new
        ↳ List<int>()).ToArray();
101
102     for (int ielem = 0; ielem < _grid.Elements.Length; ielem++) {
103         for (int i = 0; i < _grid.Elements[ielem].Length; i++) {
104             for (int j = i + 1; j < _grid.Elements[ielem].Length; j++) {
105                 int pos = _grid.Elements[ielem][j];
106                 int elem = _grid.Elements[ielem][i];
107
108                 if (!list[pos].Contains(elem)) {
109                     list[pos].Add(elem);
110                 }
111             }
112         }
113     }
114
115     list = list.Select(list => list.OrderBy(value => value).ToList()).ToArray();
116     int sizeOffDiag = list.Sum(childList => childList.Count);
117
118     InitSLAE(sizeOffDiag);
119
120     _globalMatrix.ig[2] = 1;
121
122     for (int i = 1; i < list.Length; i++) {
123         _globalMatrix.ig[(2 * i) + 1] = _globalMatrix.ig[2 * i] + (2 *
            ↳ list[i].Count);
124         _globalMatrix.ig[(2 * i) + 2] = _globalMatrix.ig[(2 * i) + 1] + (2 *
            ↳ list[i].Count) + 1;
125     }
126
127     _globalMatrix.jg = new int[_globalMatrix.ig[^1]];
128     _globalMatrix.ggl = new double[_globalMatrix.ig[^1]];
129     _globalMatrix.ggu = new double[_globalMatrix.ig[^1]];
130
131     int index = 1;
132
133     for (int i = 1; i < list.Length; i++) {
134         for (int j = 0; j < list[i].Count; j++) {
135             _globalMatrix.jg[index] = 2 * list[i][j];
136             _globalMatrix.jg[index + 1] = (2 * list[i][j]) + 1;
137             index += 2;
138         }
139
140         for (int k = 0; k < list[i].Count; k++) {
141             _globalMatrix.jg[index] = 2 * list[i][k];
142             _globalMatrix.jg[index + 1] = (2 * list[i][k]) + 1;

```

```

143         index += 2;
144     }
145
146     _globalMatrix.jg[index] = 2 * i;
147     index++;
148 }
149 }
150
151 private void AssemblySLAE() {
152     for (int ielem = 0; ielem < _grid.Elements.Length; ielem++) {
153         AssemblyLocalMatrices(ielem);
154         AssemblyLocalVector(ielem);
155
156         for (int i = 0; i < _grid.Elements[ielem].Length; i++) {
157             for (int j = 0; j < _grid.Elements[ielem].Length; j++) {
158                 AddElementToGlobalMatrix(2 * _grid.Elements[ielem][i], 2 *
                    ↪ _grid.Elements[ielem][j], _stiffnessMatrix[i, j]);
159                 AddElementToGlobalMatrix((2 * _grid.Elements[ielem][i]) + 1, (2 *
                    ↪ _grid.Elements[ielem][j]) + 1, _stiffnessMatrix[i, j]);
160                 AddElementToGlobalMatrix(2 * _grid.Elements[ielem][i], (2 *
                    ↪ _grid.Elements[ielem][j]) + 1, -_massMatrix[i, j]);
161                 AddElementToGlobalMatrix((2 * _grid.Elements[ielem][i]) + 1, 2 *
                    ↪ _grid.Elements[ielem][j], _massMatrix[i, j]);
162             }
163         }
164
165         for (int i = 0; i < _localVector1.Length; i++) {
166             _vector[2 * _grid.Elements[ielem][i]] += _localVector1[i];
167             _vector[(2 * _grid.Elements[ielem][i]) + 1] += _localVector2[i];
168         }
169
170         _localVector1.Fill(0);
171         _localVector2.Fill(0);
172     }
173 }
174
175 private void AddElementToGlobalMatrix(int i, int j, double value) {
176     if (i == j) {
177         _globalMatrix.di[i] += value;
178         return;
179     }
180
181     if (i < j) {
182         for (int index = _globalMatrix.ig[j]; index < _globalMatrix.ig[j + 1];
            ↪ index++) {
183             if (_globalMatrix.jg[index] == i) {
184                 _globalMatrix.ggu[index] += value;
185                 return;
186             }
187         }
188     } else {
189         for (int index = _globalMatrix.ig[i]; index < _globalMatrix.ig[i + 1];
            ↪ index++) {
190             if (_globalMatrix.jg[index] == j) {
191                 _globalMatrix.ggl[index] += value;
192                 return;
193             }
194         }
195     }
196 }

```

```

197 private void AssemblyLocalMatrices(int ielem) {
198     double hx = Math.Abs(_grid.Points[_grid.Elements[ielem][1]].X -
199         ↪ _grid.Points[_grid.Elements[ielem][0]].X);
200     double hy = Math.Abs(_grid.Points[_grid.Elements[ielem][2]].Y -
201         ↪ _grid.Points[_grid.Elements[ielem][0]].Y);
202     double hz = Math.Abs(_grid.Points[_grid.Elements[ielem][4]].Z -
203         ↪ _grid.Points[_grid.Elements[ielem][0]].Z);
204
205     for (int i = 0; i < _stiffnessMatrix.Size; i++) {
206         for (int j = 0; j < _stiffnessMatrix.Size; j++) {
207             _stiffnessMatrix[i, j] = (_grid.Lambda *
208                 ↪ (Integration.GaussSegmentGrad(_basis[i].Invoke, _basis[j].Invoke,
209                 ↪ new(0, 0, 0), new(1, 1, 1)) /
210                 (hx * hy * hz))) - (_grid.Omega * _grid.Chi *
211                 ↪ (Integration.GaussSegment(_basis[i].Invoke, _basis[j].Invoke,
212                 ↪ new(0, 0, 0), new(1, 1, 1)) * (hx * hy * hz)));
213         }
214     }
215
216     for (int i = 0; i < _massMatrix.Size; i++) {
217         for (int j = 0; j < _massMatrix.Size; j++) {
218             _massMatrix[i, j] = _grid.Omega * _grid.Sigma *
219                 ↪ Integration.GaussSegment(_basis[i].Invoke, _basis[j].Invoke,
220                 ↪ new(0, 0, 0), new(1, 1, 1)) * (hx * hy * hz);
221         }
222     }
223 }
224
225 private void AssemblyLocalVector(int ielem) {
226     for (int i = 0; i < _massMatrix.Size; i++) {
227         for (int j = 0; j < _massMatrix.Size; j++) {
228             _localVector1[i] += _test.Fs(_grid.Points[_grid.Elements[ielem][j]])
229                 ↪ * _massMatrix[i, j] / (_grid.Omega * _grid.Sigma);
230             _localVector2[i] += _test.Fc(_grid.Points[_grid.Elements[ielem][j]])
231                 ↪ * _massMatrix[i, j] / (_grid.Omega * _grid.Sigma);
232         }
233     }
234 }
235
236 private void AccountingDirichletBoundary() {
237     for (int iside = 0; iside < _grid.Sides.Length; iside++) {
238         for (int inode = 0; inode < _grid.Sides[iside].Length; inode++) {
239             _globalMatrix.di[2 * _grid.Sides[iside][inode]] = 1.0;
240             _globalMatrix.di[(2 * _grid.Sides[iside][inode]) + 1] = 1.0;
241             _vector[2 * _grid.Sides[iside][inode]] =
242                 ↪ _test.Us(_grid.Points[_grid.Sides[iside][inode]]);
243             _vector[(2 * _grid.Sides[iside][inode]) + 1] =
244                 ↪ _test.Uc(_grid.Points[_grid.Sides[iside][inode]]);
245
246             int diagonal = 2 * _grid.Sides[iside][inode];
247
248             for (int k = _globalMatrix.ig[diagonal]; k <
249                 ↪ _globalMatrix.ig[diagonal + 1]; k++) {
250                 _globalMatrix.ggl[k] = 0.0;
251             }
252
253             for (int i = diagonal + 1; i < _globalMatrix.Size; i++) {
254                 for (int k = _globalMatrix.ig[i]; k < _globalMatrix.ig[i + 1];
255                     ↪ k++) {

```



```

243         if (_globalMatrix.jg[k] == diagonal) {
244             _globalMatrix.ggu[k] = 0.0;
245             break;
246         }
247     }
248 }
249
250 diagonal = (2 * _grid.Sides[iside][inode]) + 1;
251
252 for (int k = _globalMatrix.ig[diagonal]; k <
    ↪ _globalMatrix.ig[diagonal + 1]; k++) {
253     _globalMatrix.ggl[k] = 0.0;
254 }
255
256 for (int i = diagonal + 1; i < _globalMatrix.Size; i++) {
257     for (int k = _globalMatrix.ig[i]; k < _globalMatrix.ig[i + 1];
    ↪ k++) {
258         if (_globalMatrix.jg[k] == diagonal) {
259             _globalMatrix.ggu[k] = 0.0;
260             break;
261         }
262     }
263 }
264 }
265 }
266 }
267
268 //for report
269 private void ErrIter() {
270     using var sw = new StreamWriter("results/errIter.txt");
271     for (int i = 0; i < _grid.Points.Length; i++) {
272         sw.WriteLine(Math.Abs(_solver!.Solution!.Value[2 * i] -
    ↪ _test.Us(_grid.Points[i])));
273         sw.WriteLine(Math.Abs(_solver.Solution.Value[(2 * i) + 1] -
    ↪ _test.Uc(_grid.Points[i])));
274     }
275 }
276
277 private void ErrForward() {
278     using var sw = new StreamWriter("results/errForward.txt");
279     for (int i = 0; i < _grid.Points.Length; i++) {
280         sw.WriteLine(Math.Abs(_decomposer!.Solution!.Value[2 * i] -
    ↪ _test.Us(_grid.Points[i])));
281         sw.WriteLine(Math.Abs(_decomposer.Solution.Value[(2 * i) + 1] -
    ↪ _test.Uc(_grid.Points[i])));
282     }
283 }
284
285 public static FEMBuilder CreateBuilder()
286     => new();
287 }

```

SparseMatrix.cs

```

1 namespace eMP_3;
2
3 public class SparseMatrix {
4     // public fields - its bad, but the readability is better
5     public int[] ig = default!;
6     public int[] jg = default!;

```

```

7 public double[] di = default!;
8 public double[] ggl = default!;
9 public double[] ggu = default!;
10 public int Size { get; init; }
11
12 public SparseMatrix(int size, int sizeOffDiag) {
13     Size = size;
14     ig = new int[size + 1];
15     jg = new int[sizeOffDiag];
16     ggl = new double[sizeOffDiag];
17     ggu = new double[sizeOffDiag];
18     di = new double[size];
19 }
20
21 public static Vector<double> operator *(SparseMatrix matrix, Vector<double>
↪ vector) {
22     Vector<double> product = new(vector.Length);
23
24     for (int i = 0; i < vector.Length; i++) {
25         product[i] = matrix.di[i] * vector[i];
26
27         for (int j = matrix.ig[i]; j < matrix.ig[i + 1]; j++) {
28             product[i] += matrix.ggl[j] * vector[matrix.jg[j]];
29             product[matrix.jg[j]] += matrix.ggu[j] * vector[i];
30         }
31     }
32
33     return product;
34 }
35
36 public void PrintDense(string path) {
37     double[,] A = new double[Size, Size];
38
39     for (int i = 0; i < Size; i++) {
40         A[i, i] = di[i];
41
42         for (int j = ig[i]; j < ig[i + 1]; j++) {
43             A[i, jg[j]] = ggl[j];
44             A[jg[j], i] = ggu[j];
45         }
46     }
47
48     using var sw = new StreamWriter(path);
49     for (int i = 0; i < Size; i++) {
50         for (int j = 0; j < Size; j++) {
51             sw.Write(A[i, j].ToString("0.0000") + "\t");
52         }
53
54         sw.WriteLine();
55     }
56 }
57
58 public void AsProfileMatrix() {
59     int[] ignew = ig.ToArray();
60
61     for (int i = 0; i < Size; i++) {
62         int i0 = ig[i];
63         int i1 = ig[i + 1];
64
65         int profile = i1 - i0;

```

```

66         if (profile > 0) {
67             int count = i - jg[i0];
68             ignew[i + 1] = ignew[i] + count;
69         } else {
70             ignew[i + 1] = ignew[i];
71         }
72     }
73 }
74
75 double[] gglnew = new double[ignew[^1]];
76 double[] ggunew = new double[ignew[^1]];
77
78 for (int i = 0; i < Size; i++) {
79     int i0 = ignew[i];
80     int i1 = ignew[i + 1];
81
82     int j = i - (i1 - i0);
83
84     int i0Old = ig[i];
85
86     for (int ik = i0; ik < i1; ik++, j++) {
87         if (j == jg[i0Old]) {
88             gglnew[ik] = ggl[i0Old];
89             ggunew[ik] = ggu[i0Old];
90             i0Old++;
91         } else {
92             gglnew[ik] = 0.0;
93             ggunew[ik] = 0.0;
94         }
95     }
96 }
97
98 ig = ignew;
99 ggl = gglnew;
100 ggu = ggunew;
101 }
102 }

```

Matrix.cs

```

1 namespace eMP_3;
2
3 public class Matrix {
4     private readonly double[,] storage;
5     public int Size { get; init; }
6
7     public double this[int i, int j] {
8         get => storage[i, j];
9         set => storage[i, j] = value;
10    }
11
12    public Matrix(int size) {
13        storage = new double[size, size];
14        Size = size;
15    }
16
17    public void Clear()
18        => Array.Clear(storage, 0, storage.Length);
19
20    public static Matrix operator +(Matrix fstMatrix, Matrix sndMatrix) {

```

```

21     Matrix resultMatrix = new(fstMatrix.Size);
22
23     for (int i = 0; i < resultMatrix.Size; i++) {
24         for (int j = 0; j < resultMatrix.Size; j++) {
25             resultMatrix[i, j] = fstMatrix[i, j] + sndMatrix[i, j];
26         }
27     }
28
29     return resultMatrix;
30 }
31 }

```

Vector.cs

```

1  namespace eMP_3;
2
3  public class Vector<T> where T : INumber<T> {
4      private readonly T[] vec;
5      public int Length { get; init; }
6
7      public T this[int index] {
8          get => vec[index];
9          set => vec[index] = value;
10     }
11
12     public Vector(int dim) {
13         vec = new T[dim];
14         Length = dim;
15     }
16
17     public static T operator *(Vector<T> firstVec, Vector<T> secondVec) {
18         T result = T.Zero;
19
20         for (int i = 0; i < firstVec.Length; i++) {
21             result += firstVec[i] * secondVec[i];
22         }
23
24         return result;
25     }
26
27     public static Vector<T> operator *(double constant, Vector<T> vector) {
28         Vector<T> result = new(vector.Length);
29
30         for (int i = 0; i < vector.Length; i++) {
31             result[i] = vector[i] * T.Create(constant);
32         }
33
34         return result;
35     }
36
37     public static Vector<T> operator +(Vector<T> firstVec, Vector<T> secondVec) {
38         Vector<T> result = new(firstVec.Length);
39
40         for (int i = 0; i < firstVec.Length; i++) {
41             result[i] = firstVec[i] + secondVec[i];
42         }
43
44         return result;
45     }
46 }

```

```

47     public static Vector<T> operator -(Vector<T> firstVec, Vector<T> secondVec) {
48         Vector<T> result = new<T>(firstVec.Length);
49
50         for (int i = 0; i < firstVec.Length; i++) {
51             result[i] = firstVec[i] - secondVec[i];
52         }
53
54         return result;
55     }
56
57     public static void Copy(Vector<T> source, Vector<T> destination) {
58         for (int i = 0; i < source.Length; i++) {
59             destination[i] = source[i];
60         }
61     }
62
63     public void Fill(double value) {
64         for (int i = 0; i < Length; i++) {
65             vec[i] = T.Create(value);
66         }
67     }
68
69     public double Norm() {
70         T result = T.Zero;
71
72         for (int i = 0; i < Length; i++) {
73             result += vec[i] * vec[i];
74         }
75
76         return Math.Sqrt(Convert.ToDouble(result));
77     }
78
79     public ImmutableArray<T> ToImmutableArray()
80         => ImmutableArray.Create(vec);
81 }

```

Solvers.cs

```

1  namespace eMP_3;
2
3  public abstract class Solver {
4      protected TimeSpan _runningTime;
5      protected SparseMatrix _matrix = default!;
6      protected Vector<double> _vector = default!;
7      protected Vector<double>? _solution;
8      public int MaxIters { get; init; }
9      public double Eps { get; init; }
10     public TimeSpan? RunningTime => _runningTime;
11     public ImmutableArray<double>? Solution => _solution?.ToImmutableArray();
12
13     protected Solver(int maxIters, double eps)
14         => (MaxIters, Eps) = (maxIters, eps);
15
16     public void SetMatrix(SparseMatrix matrix)
17         => _matrix = matrix;
18
19     public void SetVector(Vector<double> vector)
20         => _vector = vector;
21
22     public abstract void Compute();

```

```

23 }
24
25 public class LOS : Solver {
26     public LOS(int maxIters, double eps) : base(maxIters, eps) { }
27
28     public override void Compute() {
29         try {
30             ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
31             ↪ null, set the matrix");
32             ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
33             ↪ null, set the vector");
34
35             double alpha, beta;
36             double squareNorm;
37
38             _solution = new(_vector.Length);
39
40             Vector<double> r = new(_vector.Length);
41             Vector<double> z = new(_vector.Length);
42             Vector<double> p = new(_vector.Length);
43             Vector<double> tmp = new(_vector.Length);
44
45             Stopwatch sw = Stopwatch.StartNew();
46
47             r = _vector - (_matrix * _solution);
48
49             Vector<double>.Copy(r, z);
50
51             p = _matrix * z;
52
53             squareNorm = r * r;
54
55             for (int index = 0; index < MaxIters && squareNorm > Eps; index++) {
56                 alpha = p * r / (p * p);
57                 _solution += alpha * z;
58                 squareNorm = (r * r) - (alpha * alpha * (p * p));
59                 r -= alpha * p;
60
61                 tmp = _matrix * r;
62
63                 beta = -(p * tmp) / (p * p);
64                 z = r + (beta * z);
65                 p = tmp + (beta * p);
66             }
67
68             sw.Stop();
69
70             _runningTime = sw.Elapsed;
71         } catch (Exception ex) {
72             Console.WriteLine($"{We had problem: {ex.Message}");
73         }
74     }
75 }
76
77 public class LOSLU : Solver {
78     public LOSLU(int maxIters, double eps) : base(maxIters, eps) { }
79
80     public override void Compute() {
81         try {
82             ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
83             ↪ null, set the matrix");

```

```

81     ArgumentException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
      ↪ null, set the vector");
82
83     double alpha, beta;
84     double squareNorm;
85
86     _solution = new(_vector.Length);
87
88     double[] gglnew = new double[_matrix.ggl.Length];
89     double[] ggunew = new double[_matrix.ggu.Length];
90     double[] dinew = new double[_matrix.di.Length];
91
92     _matrix.ggl.Copy(gglnew);
93     _matrix.ggu.Copy(ggunew);
94     _matrix.di.Copy(dinew);
95
96     Vector<double> r = new(_vector.Length);
97     Vector<double> z = new(_vector.Length);
98     Vector<double> p = new(_vector.Length);
99     Vector<double> tmp = new(_vector.Length);
100
101     Stopwatch sw = Stopwatch.StartNew();
102
103     LU(gglnew, ggunew, dinew);
104
105     r = Direct(_vector - MultDi(_solution), gglnew, dinew);
106     z = Reverse(r, ggunew);
107     p = Direct(_matrix * z, gglnew, dinew);
108
109     squareNorm = r * r;
110
111     for (int iter = 0; iter < MaxIters && squareNorm > Eps; iter++) {
112         alpha = p * r / (p * p);
113         squareNorm = (r * r) - (alpha * alpha * (p * p));
114         _solution += alpha * z;
115         r -= alpha * p;
116
117         tmp = Direct(_matrix * Reverse(r, ggunew), gglnew, dinew);
118
119         beta = -(p * tmp) / (p * p);
120         z = Reverse(r, ggunew) + (beta * z);
121         p = tmp + (beta * p);
122     }
123
124     sw.Stop();
125
126     _runningTime = sw.Elapsed;
127 } catch (Exception ex) {
128     Console.WriteLine($"We had problem: {ex.Message}");
129 }
130 }
131
132 private Vector<double> Direct(Vector<double> vector, double[] gglnew, double[]
      ↪ dinew) {
133     Vector<double> y = new(vector.Length);
134     Vector<double>.Copy(vector, y);
135
136     double sum = 0.0;
137
138     for (int i = 0; i < _matrix.Size; i++) {

```

```

139         int i0 = _matrix.ig[i];
140         int i1 = _matrix.ig[i + 1];
141
142         for (int k = i0; k < i1; k++)
143             sum += gglnew[k] * y[_matrix.jg[k]];
144
145         y[i] = (y[i] - sum) / dinew[i];
146         sum = 0.0;
147     }
148
149     return y;
150 }
151
152 private Vector<double> Reverse(Vector<double> vector, double[] ggunew) {
153     Vector<double> result = new(vector.Length);
154     Vector<double>.Copy(vector, result);
155
156     for (int i = _matrix.Size - 1; i >= 0; i--) {
157         int i0 = _matrix.ig[i];
158         int i1 = _matrix.ig[i + 1];
159
160         for (int k = i0; k < i1; k++)
161             result[_matrix.jg[k]] -= ggunew[k] * result[i];
162     }
163
164     return result;
165 }
166
167 private void LU(double[] gglnew, double[] ggunew, double[] dinew) {
168     double suml = 0.0;
169     double sumu = 0.0;
170     double sumdi = 0.0;
171
172     for (int i = 0; i < _matrix.Size; i++) {
173         int i0 = _matrix.ig[i];
174         int i1 = _matrix.ig[i + 1];
175
176         for (int k = i0; k < i1; k++) {
177             int j = _matrix.jg[k];
178             int j0 = _matrix.ig[j];
179             int j1 = _matrix.ig[j + 1];
180             int ik = i0;
181             int kj = j0;
182
183             while (ik < k && kj < j1) {
184                 if (_matrix.jg[ik] == _matrix.jg[kj]) {
185                     suml += gglnew[ik] * ggunew[kj];
186                     sumu += ggunew[ik] * gglnew[kj];
187                     ik++;
188                     kj++;
189                 } else if (_matrix.jg[ik] > _matrix.jg[kj]) {
190                     kj++;
191                 } else {
192                     ik++;
193                 }
194             }
195
196             gglnew[k] -= suml;
197             ggunew[k] = (ggunew[k] - sumu) / dinew[j];
198             sumdi += gglnew[k] * ggunew[k];

```



```

199         suml = 0.0;
200         sumu = 0.0;
201     }
202
203     dinew[i] -= sumdi;
204     sumdi = 0.0;
205 }
206
207 private Vector<double> MultDi(Vector<double> vector) {
208     Vector<double> product = new(vector.Length);
209
210     for (int i = 0; i < _matrix.Size; i++) {
211         product[i] = 1 / Math.Sqrt(_matrix.di[i]) * vector[i];
212     }
213
214     return product;
215 }
216
217 }
218
219 public class BCGSTABLU : Solver {
220     public BCGSTABLU(int maxIters, double eps) : base(maxIters, eps) { }
221
222     public override void Compute() {
223         try {
224             ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
225                 ↳ null, set the matrix");
226             ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
227                 ↳ null, set the vector");
228
229             double alpha = 1.0;
230             double omega = 1.0;
231             double rho = 1.0;
232             double beta, temp;
233
234             double vectorNorm = _vector.Norm();
235
236             _solution = new(_vector.Length);
237
238             double[] gglnew = new double[_matrix.ggl.Length];
239             double[] ggunew = new double[_matrix.ggu.Length];
240             double[] dinew = new double[_matrix.di.Length];
241
242             _matrix.ggl.Copy(gglnew);
243             _matrix.ggu.Copy(ggunew);
244             _matrix.di.Copy(dinew);
245
246             Vector<double> r = new(_vector.Length);
247             Vector<double> r0 = new(_vector.Length);
248             Vector<double> z = new(_vector.Length);
249             Vector<double> p = new(_vector.Length);
250             Vector<double> v = new(_vector.Length);
251             Vector<double> s = new(_vector.Length);
252             Vector<double> t = new(_vector.Length);
253
254             Stopwatch sw = Stopwatch.StartNew();
255
256             LU(gglnew, ggunew, dinew);
257
258             r = Direct(_vector - (_matrix * _solution), gglnew, dinew);

```

```

257     Vector<double>.Copy(r, r0);
258
259
260     for (int iter = 0; iter < MaxIters && r.Norm() / vectorNorm >= Eps;
261         ↪ iter++) {
262         temp = rho;
263         rho = r0 * r;
264         beta = rho / temp * (alpha / omega);
265         p = r + (beta * (p - (omega * v)));
266         v = Direct(_matrix * Reverse(p, ggunew), gglnew, dinew);
267         alpha = rho / (r0 * v);
268         s = r - (alpha * v);
269         t = Direct(_matrix * Reverse(s, ggunew), gglnew, dinew);
270         omega = t * s / (t * t);
271         _solution += (omega * s) + (alpha * p);
272         r = s - (omega * t);
273     }
274
275     _solution = Reverse(_solution, ggunew);
276
277     sw.Stop();
278
279     _runningTime = sw.Elapsed;
280 } catch (Exception ex) {
281     Console.WriteLine($"We had problem: {ex.Message}");
282 }
283
284 private Vector<double> Direct(Vector<double> vector, double[] gglnew, double[]
285     ↪ dinew) {
286     Vector<double> y = new(vector.Length);
287     Vector<double>.Copy(vector, y);
288
289     double sum = 0.0;
290
291     for (int i = 0; i < _matrix.Size; i++) {
292         int i0 = _matrix.ig[i];
293         int i1 = _matrix.ig[i + 1];
294
295         for (int k = i0; k < i1; k++)
296             sum += gglnew[k] * y[_matrix.jg[k]];
297
298         y[i] = (y[i] - sum) / dinew[i];
299         sum = 0.0;
300     }
301
302     return y;
303 }
304
305 private Vector<double> Reverse(Vector<double> vector, double[] ggunew) {
306     Vector<double> result = new(vector.Length);
307     Vector<double>.Copy(vector, result);
308
309     for (int i = _matrix.Size - 1; i >= 0; i--) {
310         int i0 = _matrix.ig[i];
311         int i1 = _matrix.ig[i + 1];
312
313         for (int k = i0; k < i1; k++)
314             result[_matrix.jg[k]] -= ggunew[k] * result[i];
315     }

```

```

315         return result;
316     }
317
318     private void LU(double[] gglnew, double[] ggunew, double[] dinew) {
319         double suml = 0.0;
320         double sumu = 0.0;
321         double sumdi = 0.0;
322
323         for (int i = 0; i < _matrix.Size; i++) {
324             int i0 = _matrix.ig[i];
325             int i1 = _matrix.ig[i + 1];
326
327             for (int k = i0; k < i1; k++) {
328                 int j = _matrix.jg[k];
329                 int j0 = _matrix.ig[j];
330                 int j1 = _matrix.ig[j + 1];
331                 int ik = i0;
332                 int kj = j0;
333
334                 while (ik < k && kj < j1) {
335                     if (_matrix.jg[ik] == _matrix.jg[kj]) {
336                         suml += gglnew[ik] * ggunew[kj];
337                         sumu += ggunew[ik] * gglnew[kj];
338                         ik++;
339                         kj++;
340                     } else if (_matrix.jg[ik] > _matrix.jg[kj]) {
341                         kj++;
342                     } else {
343                         ik++;
344                     }
345                 }
346
347                 gglnew[k] -= suml;
348                 ggunew[k] = (ggunew[k] - sumu) / dinew[j];
349                 sumdi += gglnew[k] * ggunew[k];
350                 suml = 0.0;
351                 sumu = 0.0;
352             }
353
354             dinew[i] -= sumdi;
355             sumdi = 0.0;
356         }
357     }
358 }
359

```

Decomposers.cs

```

1  namespace eMP_3;
2
3  public abstract class Decomposer {
4      protected TimeSpan _runningTime;
5      protected SparseMatrix _matrix = default!;
6      protected Vector<double> _vector = default!;
7      protected Vector<double>? _solution;
8      public TimeSpan? RunningTime => _runningTime;
9      public ImmutableArray<double>? Solution => _solution?.ToImmutableArray();
10
11     public void SetMatrix(SparseMatrix matrix)
12         => _matrix = matrix;

```

```

13
14     public void SetVector(Vector<double> vector)
15         => _vector = vector;
16
17     public abstract void Compute();
18 }
19
20 public class DecomposerLDU : Decomposer {
21     public override void Compute() {
22         _solution = new(_matrix.Size);
23         Vector<double>.Copy(_vector, _solution);
24
25         Stopwatch sw = Stopwatch.StartNew();
26
27         LDU();
28         ForwardElimination();
29         DiagonalStroke();
30         BackSubstitution();
31
32         sw.Stop();
33
34         _runningTime = sw.Elapsed;
35     }
36
37     private void LDU() {
38         for (int i = 0; i < _matrix.Size; i++) {
39             double sumdi = 0.0;
40
41             int i0 = _matrix.ig[i];
42             int i1 = _matrix.ig[i + 1];
43
44             int j = i - (i1 - i0);
45
46             for (int ij = i0; ij < i1; ij++, j++) {
47                 double suml = 0.0;
48                 double sumu = 0.0;
49
50                 int j0 = _matrix.ig[j];
51                 int j1 = _matrix.ig[j + 1];
52
53                 int ik = i0;
54                 int jk = j0;
55
56                 int k = i - (i1 - i0);
57
58                 int ci = ij - i0;
59                 int cj = j1 - j0;
60
61                 int cij = ci - cj;
62
63                 if (cij > 0) {
64                     ik += cij;
65                     k += cij;
66                 } else {
67                     jk -= cij;
68                 }
69
70                 for (; ik < ij; ik++, jk++, k++) {
71                     suml += _matrix.ggl[ik] * _matrix.di[k] * _matrix.ggu[jk];
72                     sumu += _matrix.ggu[ik] * _matrix.di[k] * _matrix.ggl[jk];

```

```

73         }
74
75         _matrix.ggl[ij] = (_matrix.ggl[ij] - suml) / _matrix.di[j];
76         _matrix.ggu[ij] = (_matrix.ggu[ij] - sumu) / _matrix.di[j];
77
78         sumdi += _matrix.ggl[ij] * _matrix.ggu[ij] * _matrix.di[k];
79     }
80
81     _matrix.di[i] -= sumdi;
82 }
83 }
84
85 private void DiagonalStroke() {
86     for (int i = 0; i < _matrix.Size; i++) {
87         _solution![i] /= _matrix.di[i];
88     }
89 }
90
91 private void ForwardElimination() {
92     for (int i = 0; i < _matrix.Size; i++) {
93         double sum = 0.0;
94
95         int i0 = _matrix.ig[i];
96         int i1 = _matrix.ig[i + 1];
97
98         int j = i - (i1 - i0);
99
100        for (int ij = i0; ij < i1; ij++, j++) {
101            sum += _matrix.ggl[ij] * _solution![j];
102        }
103
104        _solution![i] -= sum;
105    }
106 }
107
108 private void BackSubstitution() {
109     for (int i = _matrix.Size - 1; i >= 0; i--) {
110         int i0 = _matrix.ig[i];
111         int i1 = _matrix.ig[i + 1];
112
113         for (int j = i - 1, ij = i1 - 1; ij >= i0; ij--, j--) {
114             _solution![j] -= _matrix.ggu[ij] * _solution[i];
115         }
116     }
117 }
118 }

```

Interval.cs

```

1 namespace eMP_3;
2
3 public readonly record struct Interval {
4     [JsonProperty("Left Border")]
5     public double LeftBorder { get; init; }
6
7     [JsonProperty("Right Border")]
8     public double RightBorder { get; init; }
9
10    [JsonIgnore]
11    public double Lenght { get; init; }

```

```

12
13 [JsonConstructor]
14 public Interval(double leftBorder, double rightBorder) {
15     LeftBorder = leftBorder;
16     RightBorder = rightBorder;
17     Lenght = Math.Abs(rightBorder - leftBorder);
18 }
19 }

```

Integration.cs

```

1 namespace eMP_3;
2
3 public static class Integration {
4     public static double GaussSegment(Func<Point3D, double> psiI, Func<Point3D,
5     ↪ double> psiJ, Point3D firstPoint, Point3D secondPoint) {
6         var quadratures = Quadratures.SegmentGaussOrder9();
7
8         double hx = secondPoint.X - firstPoint.X;
9         double hy = secondPoint.Y - firstPoint.Y;
10        double hz = secondPoint.Z - firstPoint.Z;
11        double result = 0.0;
12
13        foreach (var qi in quadratures) {
14            foreach (var qj in quadratures) {
15                foreach (var qk in quadratures) {
16                    Point3D point = new(((qi.Node * hx) + firstPoint.X +
17                    ↪ secondPoint.X) / 2.0,
18                    ((qj.Node * hy) + firstPoint.Y + secondPoint.Y)
19                    ↪ / 2.0,
20                    ((qk.Node * hz) + firstPoint.Z + secondPoint.Z)
21                    ↪ / 2.0);
22
23                    result += psiI(point) * psiJ(point) * qi.Weight * qj.Weight *
24                    ↪ qk.Weight;
25                }
26            }
27        }
28
29        return result * hx * hy * hz / 8.0;
30    }
31
32    public static double GaussSegmentGrad(Func<Point3D, double> psiI, Func<Point3D,
33    ↪ double> psiJ, Point3D firstPoint, Point3D secondPoint) {
34        var quadratures = Quadratures.SegmentGaussOrder9();
35
36        double hx = secondPoint.X - firstPoint.X;
37        double hy = secondPoint.Y - firstPoint.Y;
38        double hz = secondPoint.Z - firstPoint.Z;
39        double resultX = 0.0;
40        double resultY = 0.0;
41        double resultZ = 0.0;
42
43        foreach (var qi in quadratures) {
44            foreach (var qj in quadratures) {
45                foreach (var qk in quadratures) {
46                    Point3D point = new(((qi.Node * hx) + firstPoint.X +
47                    ↪ secondPoint.X) / 2.0,
48                    ((qj.Node * hy) + firstPoint.Y + secondPoint.Y)
49                    ↪ / 2.0,

```

```

42         ((qk.Node * hz) + firstPoint.Z + secondPoint.Z)
43         ↪ / 2.0);
44
45         resultX += DerivativeX(point, hx) * qi.Weight * qj.Weight *
46         ↪ qk.Weight;
47     }
48 }
49
50 resultX *= hx * hy * hz / 8.0;
51
52 foreach (var qi in quadratures) {
53     foreach (var qj in quadratures) {
54         foreach (var qk in quadratures) {
55             Point3D point = new(((qi.Node * hx) + firstPoint.X +
56             ↪ secondPoint.X) / 2.0,
57             ((qj.Node * hy) + firstPoint.Y + secondPoint.Y)
58             ↪ / 2.0,
59             ((qk.Node * hz) + firstPoint.Z + secondPoint.Z)
60             ↪ / 2.0);
61
62             resultY += DerivativeY(point, hy) * qi.Weight * qj.Weight *
63             ↪ qk.Weight;
64         }
65     }
66 }
67
68 resultY *= hx * hy * hz / 8.0;
69
70 foreach (var qi in quadratures) {
71     foreach (var qj in quadratures) {
72         foreach (var qk in quadratures) {
73             Point3D point = new(((qi.Node * hx) + firstPoint.X +
74             ↪ secondPoint.X) / 2.0,
75             ((qj.Node * hy) + firstPoint.Y + secondPoint.Y)
76             ↪ / 2.0,
77             ((qk.Node * hz) + firstPoint.Z + secondPoint.Z)
78             ↪ / 2.0);
79
80             resultZ += DerivativeZ(point, hz) * qi.Weight * qj.Weight *
81             ↪ qk.Weight;
82         }
83     }
84 }
85
86 resultZ *= hx * hy * hz / 8.0;
87
88 return resultX + resultY + resultZ;
89
90 double DerivativeX(Point3D point, double h)
91     => (psiI(point + (h, 0, 0)) - psiI(point - (h, 0, 0))) / (2 * h) *
92         ((psiJ(point + (h, 0, 0)) - psiJ(point - (h, 0, 0))) / (2 * h));
93
94 double DerivativeY(Point3D point, double h)
95     => (psiI(point + (0, h, 0)) - psiI(point - (0, h, 0))) / (2 * h) *
96         ((psiJ(point + (0, h, 0)) - psiJ(point - (0, h, 0))) / (2 * h));
97
98 double DerivativeZ(Point3D point, double h)
99     => (psiI(point + (0, 0, h)) - psiI(point - (0, 0, h))) / (2 * h) *
100         ((psiJ(point + (0, 0, h)) - psiJ(point - (0, 0, h))) / (2 * h));

```

```

92     }
93 }

```

Quadratures.cs

```

1  namespace eMP_3;
2
3  public class QuadratureNode<T> where T : notnull {
4      public T Node { get; init; }
5      public double Weight { get; init; }
6
7      public QuadratureNode(T node, double weight) {
8          Node = node;
9          Weight = weight;
10     }
11 }
12
13 public class Quadrature<T> where T : notnull {
14     private readonly QuadratureNode<T>[] _nodes = default!;
15     public ImmutableArray<QuadratureNode<T>> Nodes => _nodes.ToImmutableArray();
16
17     public Quadrature(QuadratureNode<T>[] nodes)
18         => _nodes = nodes;
19 }
20
21 public static class Quadratures {
22     public static IEnumerable<QuadratureNode<double>> SegmentGaussOrder9() {
23         const int n = 5;
24         double[] points = { -1.0 / 3.0 * Math.Sqrt(5 + (2 * Math.Sqrt(10.0 / 7.0))),
25                             1.0 / 3.0 * Math.Sqrt(5 - (2 * Math.Sqrt(10.0 / 7.0))),
26                             0.0,
27                             -1.0 / 3.0 * Math.Sqrt(5 - (2 * Math.Sqrt(10.0 / 7.0))),
28                             1.0 / 3.0 * Math.Sqrt(5 + (2 * Math.Sqrt(10.0 / 7.0))) };
29
30         double[] weights = { (322.0 - (13.0 * Math.Sqrt(70.0))) / 900.0, (322.0 +
31                               ↳ (13.0 * Math.Sqrt(70.0))) / 900.0,
32                               128.0 / 225.0,
33                               (322.0 + (13.0 * Math.Sqrt(70.0))) / 900.0,
34                               (322.0 - (13.0 * Math.Sqrt(70.0))) / 900.0 };
35
36         for (int i = 0; i < n; i++) {
37             yield return new QuadratureNode<double>(points[i], weights[i]);
38         }
39     }
40 }

```

LinearBasis.cs

```

1  namespace eMP_3;
2
3  public static class LinearBasis {
4      public static double Psi1(Point3D point)
5          => (1 - point.X) * (1 - point.Y) * (1 - point.Z);
6
7      public static double Psi2(Point3D point)
8          => point.X * (1 - point.Y) * (1 - point.Z);
9
10     public static double Psi3(Point3D point)
11         => point.Y * (1 - point.X) * (1 - point.Z);

```



```

12
13     public static double Psi4(Point3D point)
14         => point.Z * (1 - point.X) * (1 - point.Y);
15
16     public static double Psi5(Point3D point)
17         => (1 - point.X) * point.Y * point.Z;
18
19     public static double Psi6(Point3D point)
20         => point.X * point.Y * (1 - point.Z);
21
22     public static double Psi7(Point3D point)
23         => point.X * point.Z * (1 - point.Y);
24
25     public static double Psi8(Point3D point)
26         => point.X * point.Z * point.Y;
27 }

```

ArrayHelper.cs

```

1 namespace eMP_3;
2
3 public static class ArrayHelper {
4     public static T[] Copy<T>(this T[] source, T[] destination) {
5         for (int i = 0; i < source.Length; i++) {
6             destination[i] = source[i];
7         }
8
9         return destination;
10    }
11
12    public static void Fill<T>(this T[] array, T value) {
13        for (int i = 0; i < array.Length; i++) {
14            array[i] = value;
15        }
16    }
17 }

```

GridParameters.cs

```

1 namespace eMP_3;
2
3 public class GridParametersJsonConverter : JsonConverter {
4     public override void WriteJson(JsonWriter writer, object? value, JsonSerializer
5     ↪ serializer) {
6         if (value is null) {
7             writer.WriteNull();
8             return;
9         }
10
11         var gridParameters = (GridParameters)value;
12
13         writer.WriteStartObject();
14         writer.WritePropertyName("Initial area in X");
15         serializer.Serialize(writer, gridParameters.IntervalX);
16         writer.WritePropertyName("Splits by X");
17         writer.WriteValue(gridParameters.SplitsX);
18         writer.WriteWhitespace("\n");
19
20         writer.WritePropertyName("Initial area in Y");

```

```

20     serializer.Serialize(writer, gridParameters.IntervalY);
21     writer.WritePropertyName("Splits by Y");
22     writer.WriteValue(gridParameters.SplitsY);
23     writer.WriteWhitespace("\n");
24
25     writer.WritePropertyName("Initial area in Z");
26     serializer.Serialize(writer, gridParameters.IntervalZ);
27     writer.WritePropertyName("Splits by Z");
28     writer.WriteValue(gridParameters.SplitsZ);
29     writer.WriteWhitespace("\n");
30
31     writer.WriteComment("Коэффициент разрядки");
32     writer.WritePropertyName("Coef");
33     writer.WriteValue(gridParameters.K);
34
35     writer.WriteComment("Коэффициенты уравнения");
36     writer.WritePropertyName("Lambda");
37     writer.WriteValue(gridParameters.Lambda);
38     writer.WritePropertyName("Omega");
39     writer.WriteValue(gridParameters.Omega);
40     writer.WritePropertyName("Chi");
41     writer.WriteValue(gridParameters.Chi);
42     writer.WriteEndObject();
43 }
44
45 public override object? ReadJson(JsonReader reader, Type objectType, object?
↪ existingValue, JsonSerializer serializer) {
46     if (reader.TokenType == JsonToken.Null || reader.TokenType !=
↪     JsonToken.StartObject)
47         return null;
48
49     Interval intervalX;
50     Interval intervalY;
51     Interval intervalZ;
52     int splitsX;
53     int splitsY;
54     int splitsZ;
55     double? coef;
56     double lambda, omega, sigma, chi;
57
58     var maintoken = JObject.Load(reader);
59
60     var token = maintoken["Initial area in X"];
61     intervalX = serializer.Deserialize<Interval>(token!.CreateReader());
62     token = maintoken["Splits by X"];
63     splitsX = Convert.ToInt32(token);
64
65     token = maintoken["Initial area in Y"];
66     intervalY = serializer.Deserialize<Interval>(token!.CreateReader());
67     token = maintoken["Splits by Y"];
68     splitsY = Convert.ToInt32(token);
69
70     token = maintoken["Initial area in Z"];
71     intervalZ = serializer.Deserialize<Interval>(token!.CreateReader());
72     token = maintoken["Splits by Z"];
73     splitsZ = Convert.ToInt32(token);
74
75     token = maintoken["Coef"];
76     if (token is not null) {
77         coef = double.TryParse(token.ToString(), out double res) ? res : null;

```

```

78     } else {
79         coef = null;
80     }
81
82     token = maintoken["Lambda"];
83     lambda = Convert.ToDouble(token);
84
85     token = maintoken["Omega"];
86     omega = Convert.ToDouble(token);
87
88     token = maintoken["Sigma"];
89     sigma = Convert.ToDouble(token);
90
91     token = maintoken["Chi"];
92     chi = Convert.ToDouble(token);
93
94     return new GridParameters(intervalX, splitsX, intervalY, splitsY, intervalZ,
95         ↪ splitsZ, coef, lambda, omega, sigma, chi);
96 }
97
98 public override bool CanConvert(Type objectType)
99     => objectType == typeof(GridParameters);
100 }
101 [JsonConverter(typeof(GridParametersJsonConverter))]
102 public readonly record struct GridParameters {
103     public Interval IntervalX { get; init; }
104     public int SplitsX { get; init; }
105     public Interval IntervalY { get; init; }
106     public int SplitsY { get; init; }
107     public Interval IntervalZ { get; init; }
108     public int SplitsZ { get; init; }
109     public double? K { get; init; }
110     public double Lambda { get; init; }
111     public double Omega { get; init; }
112     public double Sigma { get; init; }
113     public double Chi { get; init; }
114
115     public GridParameters(Interval intervalX, int splitsX, Interval intervalY, int
116         ↪ splitsY, Interval intervalZ, int splitsZ,
117         double? k, double lambda, double omega, double sigma, double
118         ↪ chi) {
119         IntervalX = intervalX;
120         SplitsX = splitsX;
121         IntervalY = intervalY;
122         SplitsY = splitsY;
123         IntervalZ = intervalZ;
124         SplitsZ = splitsZ;
125         K = k;
126         Lambda = lambda;
127         Omega = omega;
128         Sigma = sigma;
129         Chi = chi;
130     }
131
132     public static GridParameters? ReadJson(string jsonPath) {
133         try {
134             if (!File.Exists(jsonPath))
135                 throw new Exception("File does not exist");
136         }

```

```

135         var sr = new StreamReader(jsonPath);
136         using (sr) {
137             return JsonConvert.DeserializeObject<GridParameters>(sr.ReadToEnd());
138         }
139     } catch (Exception ex) {
140         Console.WriteLine($"We had problem: {ex.Message}");
141         return null;
142     }
143 }
144 }

```

GridFactory.cs

```

1 namespace eMP_3;
2
3 public enum GridTypes {
4     Regular,
5     Irregular
6 }
7
8 public interface IFactory {
9     public Grid CreateGrid(GridTypes gridType, GridParameters gridParameters);
10 }
11
12 public class GridFactory : IFactory {
13     public Grid CreateGrid(GridTypes gridType, GridParameters gridParameters) {
14         return gridType switch {
15             GridTypes.Regular => new RegularGrid(gridParameters),
16             GridTypes.Irregular => new IrregularGrid(gridParameters),
17
18             _ => throw new InvalidEnumArgumentException($"This type of grid does not
19                 ↳ exist: {nameof(gridType)}");
20         };
21     }
22 }

```

Grid.cs

```

1 namespace eMP_3;
2
3 public abstract class Grid {
4     public abstract ImmutableArray<Point3D> Points { get; }
5     public abstract ImmutableArray<ImmutableArray<int>> Elements { get; }
6     public abstract ImmutableArray<ImmutableArray<int>> Sides { get; }
7     public double Lambda { get; init; }
8     public double Omega { get; init; }
9     public double Sigma { get; init; }
10    public double Chi { get; init; }
11
12    protected Grid(GridParameters gridParameters)
13        => (Lambda, Omega, Sigma, Chi) = (gridParameters.Lambda,
14        ↳ gridParameters.Omega, gridParameters.Sigma, gridParameters.Chi);
15 }

```

RegularGrid.cs

```

1 namespace eMP_3;
2
3 public class RegularGrid : Grid {

```

```

4 private readonly Point3D[] _points = default!;
5 private readonly int[][] _elements = default!;
6 private readonly int[][] _sides = default!;
7 public override ImmutableArray<Point3D> Points => _points.ToImmutableArray();
8 public override ImmutableArray<ImmutableArray<int>> Elements =>
  ↪ _elements.Select(item => item.ToImmutableArray()).ToImmutableArray();
9 public override ImmutableArray<ImmutableArray<int>> Sides => _sides.Select(item
  ↪ => item.ToImmutableArray()).ToImmutableArray();

10
11 public RegularGrid(GridParameters gridParameters) : base (gridParameters) {
12     _points = new Point3D[(gridParameters.SplitsX + 1) * (gridParameters.SplitsY
  ↪ + 1) * (gridParameters.SplitsZ + 1)];
13     _elements = new double[gridParameters.SplitsX * gridParameters.SplitsY *
  ↪ gridParameters.SplitsZ].Select(_ => new int[8]).ToArray();
14     _sides = new int[6][];
15     _sides[0] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsZ +
  ↪ 1)]; // front
16     _sides[1] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsZ +
  ↪ 1)]; // back
17     _sides[2] = new int[(gridParameters.SplitsY + 1) * (gridParameters.SplitsZ +
  ↪ 1)]; // left
18     _sides[3] = new int[(gridParameters.SplitsY + 1) * (gridParameters.SplitsZ +
  ↪ 1)]; // right
19     _sides[4] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsY +
  ↪ 1)]; // bottom
20     _sides[5] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsY +
  ↪ 1)]; // top
21     Build(gridParameters);
22 }

23
24 private void Build(GridParameters gridParameters) {
25     try {
26         if (gridParameters.SplitsX < 1 || gridParameters.SplitsY < 1 ||
  ↪ gridParameters.SplitsZ < 1) {
27             throw new Exception("The number of splits must be greater than or
  ↪ equal to 1");
28         }
29
30         double hx = gridParameters.IntervalX.Length / gridParameters.SplitsX;
31         double hy = gridParameters.IntervalY.Length / gridParameters.SplitsY;
32         double hz = gridParameters.IntervalZ.Length / gridParameters.SplitsZ;
33
34         double[] pointsX = new double[gridParameters.SplitsX + 1];
35         double[] pointsY = new double[gridParameters.SplitsY + 1];
36         double[] pointsZ = new double[gridParameters.SplitsZ + 1];
37
38         pointsX[0] = gridParameters.IntervalX.LeftBorder;
39         pointsY[0] = gridParameters.IntervalY.LeftBorder;
40         pointsZ[0] = gridParameters.IntervalZ.LeftBorder;
41
42         for (int i = 1; i < pointsX.Length; i++) {
43             pointsX[i] = pointsX[i - 1] + hx;
44         }
45
46         for (int i = 1; i < pointsY.Length; i++) {
47             pointsY[i] = pointsY[i - 1] + hy;
48         }
49
50         for (int i = 1; i < pointsZ.Length; i++) {
51             pointsZ[i] = pointsZ[i - 1] + hz;

```

```

52     }
53
54     int index = 0;
55
56     for (int k = 0; k < pointsZ.Length; k++) {
57         for (int j = 0; j < pointsY.Length; j++) {
58             for (int i = 0; i < pointsX.Length; i++) {
59                 _points[index++] = new(pointsX[i], pointsY[j], pointsZ[k]);
60             }
61         }
62     }
63
64     // k по z, j по y, i по x
65
66     int nx = pointsX.Length;
67     int ny = pointsY.Length;
68     int nz = pointsZ.Length;
69
70     int Nx = pointsX.Length - 1;
71     int Ny = pointsY.Length - 1;
72     int Nz = pointsZ.Length - 1;
73
74     index = 0;
75
76     for (int k = 0; k < Nz; k++) {
77         for (int j = 0; j < Ny; j++) {
78             for (int i = 0; i < Nx; i++) {
79                 _elements[index][0] = i + (j * nx) + (k * nx * ny);
80                 _elements[index][1] = i + 1 + (j * nx) + (k * nx * ny);
81                 _elements[index][2] = i + ((j + 1) * nx) + (k * nx * ny);
82                 _elements[index][3] = i + 1 + ((j + 1) * nx) + (k * nx * ny);
83                 _elements[index][4] = i + (j * nx) + ((k + 1) * nx * ny);
84                 _elements[index][5] = i + 1 + (j * nx) + ((k + 1) * nx * ny);
85                 _elements[index][6] = i + ((j + 1) * nx) + ((k + 1) * nx *
86                     ↪ ny);
87                 _elements[index++][7] = i + 1 + ((j + 1) * nx) + ((k + 1) *
88                     ↪ nx * ny);
89             }
90         }
91     }
92
93     // front and back
94     for (int k = 0; k < nz; k++) {
95         for (int i = 0; i < nx; i++) {
96             _sides[0][i + (k * nx)] = i + 0 + (k * nx * ny);
97             _sides[1][i + (k * nx)] = i + (nx * (ny - 1)) + (k * nx * ny);
98         }
99     }
100
101     // left and right
102     for (int k = 0; k < nz; k++) {
103         for (int j = 0; j < ny; j++) {
104             _sides[2][j + (k * ny)] = 0 + (j * nx) + (k * nx * ny);
105             _sides[3][j + (k * ny)] = nx - 1 + (j * nx) + (k * nx * ny);
106         }
107     }
108
109     // bottom and top
110     for (int j = 0; j < ny; j++) {
111         for (int i = 0; i < nx; i++) {

```

```

110         _sides[4][i + (j * nx)] = i + (j * nx) + 0;
111         _sides[5][i + (j * nx)] = i + (j * nx) + (nx * ny * (nz - 1));
112     }
113 }
114
115 // var res = _points.Except(_internalPoints);
116
117 WritePoints();
118 } catch (Exception ex) {
119     Console.WriteLine($"We had problem: {ex.Message}");
120 }
121 }
122
123 private void WritePoints() {
124     var sw = new StreamWriter("points.txt");
125     using (sw) {
126         for (int i = 0; i < _points.Length; i++) {
127             sw.WriteLine(_points[i]);
128         }
129     }
130 }
131 }

```

IrregularGrid.cs

```

1 namespace eMP_3;
2
3 public class IrregularGrid : Grid {
4     private readonly Point3D[] _points = default!;
5     private readonly int[][] _elements = default!;
6     private readonly int[][] _sides = default!;
7     public override ImmutableArray<Point3D> Points => _points.ToImmutableArray();
8     public override ImmutableArray<ImmutableArray<int>> Elements =>
9         ↪ _elements.Select(item => item.ToImmutableArray()).ToImmutableArray();
10     public override ImmutableArray<ImmutableArray<int>> Sides => _sides.Select(item
11         ↪ => item.ToImmutableArray()).ToImmutableArray();
12
13     public IrregularGrid(GridParameters gridParameters) : base(gridParameters) {
14         _points = new Point3D[(gridParameters.SplitsX + 1) * (gridParameters.SplitsY
15             ↪ + 1) * (gridParameters.SplitsZ + 1)];
16         _sides = new int[6][];
17         _sides[0] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsZ +
18             ↪ 1)]; // front
19         _sides[1] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsZ +
20             ↪ 1)]; // back
21         _sides[2] = new int[(gridParameters.SplitsY + 1) * (gridParameters.SplitsZ +
22             ↪ 1)]; // left
23         _sides[3] = new int[(gridParameters.SplitsY + 1) * (gridParameters.SplitsZ +
24             ↪ 1)]; // right
25         _sides[4] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsY +
26             ↪ 1)]; // bottom
27         _sides[5] = new int[(gridParameters.SplitsX + 1) * (gridParameters.SplitsY +
28             ↪ 1)]; // top
29         Build(gridParameters);
30     }
31
32     private void Build(GridParameters gridParameters) {
33         try {
34             if (gridParameters.SplitsX < 1 || gridParameters.SplitsY < 1 ||
35                 ↪ gridParameters.SplitsZ < 1) {

```

```

26         throw new Exception("The number of splits must be greater than or
27         ↪ equal to 1");
28     }
29     ArgumentNullException.ThrowIfNull(gridParameters.K,
30     ↪ $"{nameof(gridParameters.K)} cannot be null");
31     if (gridParameters.K <= 0) {
32         throw new Exception("The coefficient must be greater than 0");
33     }
34
35     double[] pointsX = new double[gridParameters.SplitsX + 1];
36     double[] pointsY = new double[gridParameters.SplitsY + 1];
37     double[] pointsZ = new double[gridParameters.SplitsZ + 1];
38
39     pointsX[0] = gridParameters.IntervalX.LeftBorder;
40     pointsY[0] = gridParameters.IntervalY.LeftBorder;
41     pointsZ[0] = gridParameters.IntervalZ.LeftBorder;
42
43     double hx, hy, hz;
44     double sum = 0.0;
45
46     for (int k = 0; k < gridParameters.SplitsX; k++) {
47         sum += Math.Pow(gridParameters.K.Value, k);
48     }
49
50     hx = gridParameters.IntervalX.Lenght / sum;
51     sum = 0.0;
52
53     for (int k = 0; k < gridParameters.SplitsY; k++) {
54         sum += Math.Pow(gridParameters.K.Value, k);
55     }
56
57     hy = gridParameters.IntervalY.Lenght / sum;
58     sum = 0.0;
59
60     for (int k = 0; k < gridParameters.SplitsZ; k++) {
61         sum += Math.Pow(gridParameters.K.Value, k);
62     }
63
64     hz = gridParameters.IntervalZ.Lenght / sum;
65
66     for (int i = 1; i < pointsX.Length; i++) {
67         pointsX[i] = pointsX[i - 1] + hx;
68     }
69
70     for (int i = 1; i < pointsY.Length; i++) {
71         pointsY[i] = pointsY[i - 1] + hy;
72     }
73
74     for (int i = 1; i < pointsZ.Length; i++) {
75         pointsZ[i] = pointsZ[i - 1] + hz;
76     }
77
78     int index = 0;
79
80     for (int i = 0; i < pointsZ.Length; i++) {
81         for (int j = 0; j < pointsY.Length; j++) {
82             for (int k = 0; k < pointsX.Length; k++) {
83                 _points[index++] = new(pointsX[k], pointsY[j], pointsZ[i]);

```



```

84         }
85     }
86 }
87
88 // k по z, j по y, i по x
89
90 int nx = pointsX.Length;
91 int ny = pointsY.Length;
92 int nz = pointsZ.Length;
93
94 int Nx = pointsX.Length - 1;
95 int Ny = pointsY.Length - 1;
96 int Nz = pointsZ.Length - 1;
97
98 index = 0;
99
100 for (int k = 0; k < Nz; k++) {
101     for (int j = 0; j < Ny; j++) {
102         for (int i = 0; i < Nx; i++) {
103             _elements[index][0] = i + (j * nx) + (k * nx * ny);
104             _elements[index][1] = i + 1 + (j * nx) + (k * nx * ny);
105             _elements[index][2] = i + ((j + 1) * nx) + (k * nx * ny);
106             _elements[index][3] = i + 1 + ((j + 1) * nx) + (k * nx * ny);
107             _elements[index][4] = i + (j * nx) + ((k + 1) * nx * ny);
108             _elements[index][5] = i + 1 + (j * nx) + ((k + 1) * nx * ny);
109             _elements[index][6] = i + ((j + 1) * nx) + ((k + 1) * nx *
110                 ↪ ny);
111             _elements[index++][7] = i + 1 + ((j + 1) * nx) + ((k + 1) *
112                 ↪ nx * ny);
113         }
114     }
115 }
116
117 // front and back
118 for (int k = 0; k < nz; k++) {
119     for (int i = 0; i < nx; i++) {
120         _sides[0][i + (k * nx)] = i + 0 + (k * nx * ny);
121         _sides[1][i + (k * nx)] = i + (nx * (ny - 1)) + (k * nx * ny);
122     }
123 }
124
125 // left and right
126 for (int k = 0; k < nz; k++) {
127     for (int j = 0; j < ny; j++) {
128         _sides[2][j + (k * ny)] = 0 + (j * nx) + (k * nx * ny);
129         _sides[3][j + (k * ny)] = nx - 1 + (j * nx) + (k * nx * ny);
130     }
131 }
132
133 // bottom and top
134 for (int j = 0; j < ny; j++) {
135     for (int i = 0; i < nx; i++) {
136         _sides[4][i + (j * nx)] = i + (j * nx) + 0;
137         _sides[5][i + (j * nx)] = i + (j * nx) + (nx * ny * (nz - 1));
138     }
139 }
140
141 WritePoints();
142 } catch (Exception ex) {
143     Console.WriteLine($"We had problem: {ex.Message}");

```

```

142     }
143 }
144
145 private void WritePoints() {
146     var sw = new StreamWriter("points.txt");
147     using (sw) {
148         for (int i = 0; i < _points.Length; i++) {
149             sw.WriteLine(_points[i]);
150         }
151     }
152 }
153 }

```

Point3D.cs

```

1 namespace eMP_3;
2
3 public readonly record struct Point3D(double X, double Y, double Z) {
4     public static Point3D operator +(Point3D point, (double, double, double) value)
5         => new(point.X + value.Item1, point.Y + value.Item2, point.Z + value.Item3);
6
7     public static Point3D operator -(Point3D point, (double, double, double) value)
8         => new(point.X - value.Item1, point.Y - value.Item2, point.Z - value.Item3);
9
10    public override string ToString()
11        => $"{X} {Y} {Z}";
12 }

```

Tests.cs

```

1 namespace eMP_3;
2
3 public interface ITest {
4     public double Us(Point3D point);
5
6     public double Fs(Point3D point);
7
8     public double Uc(Point3D point);
9
10    public double Fc(Point3D point);
11 }
12
13 public class Test1 : ITest { // все коэффициенты равны 1
14     public double Fc(Point3D point)
15         => -point.X + point.Y;
16
17     public double Fs(Point3D point)
18         => (-3 * point.X) - (3 * point.Y) - (2 * point.Z);
19
20     public double Us(Point3D point)
21         => point.X + (2 * point.Y) + point.Z;
22
23     public double Uc(Point3D point)
24         => (2 * point.X) + point.Y + point.Z;
25 }
26
27 public class Test2 : ITest { // λ =, ω =, χ =
28     public double Fc(Point3D point)
29         => 0;

```

```
30
31     public double Fs(Point3D point)
32         => 0;
33
34     public double Us(Point3D point)
35         => point.X * point.Y * point.Z;
36
37     public double Uc(Point3D point)
38         => 2 * point.X * 2 * point.Y * 2 * point.Z;
39 }
```