Содержание

Формулировка задания

Решить гиперболическую задачу с помощью векторного метода конечных элементов (далее МКЭ).

Векторный МКЭ для трёхмерной краевой задачи, уравнение гиперболического типа в декартовой системе координат. Четырёхслойная неявная схема по времени. Линейные базисные-вектор функции на параллелепипедах. Краевые условия первого рода.

Постановка нестационарной задачи

Начально-краевая задача определяется уравнением:

$$\operatorname{rot}\left(\frac{1}{\mu}\ \operatorname{rot}\ \vec{A}\right)+\sigma\frac{\partial\vec{A}}{\partial t}+\varepsilon\frac{\partial^2\vec{A}}{\partial t^2}=\vec{J^{\text{ст}}}$$

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

$$\vec{A}_{t=t_0}=\vec{A}_0.$$

Аппроксимация по времени

Обозначим через $\vec{A}=\vec{A}(x,y,z,t_j)$ значение вектор-потенциала на текущем временном слое, а через $\vec{A^{\Leftarrow 1}}=\vec{A}(x,y,z,t_{j-1})$, $\vec{A^{\Leftarrow 2}}=\vec{A}(x,y,z,t_{j-2})$ и $\vec{A^{\Leftarrow 3}}=\vec{A}(x,y,z,t_{j-3})$ - значение вектор-потенциала на трёх предыдущих временных слоях. Тогда в результате аппроксимации по времени получим векторное уравнение:

$$\operatorname{rot}\left(\frac{1}{\mu}\ \operatorname{rot}\vec{A}\right)+\gamma\vec{A}=\vec{F}$$

где коэффициент γ и вектор-функция $\vec{F}$ определяются схемой аппроксимации по времени. В этом случае используется четырёхслойная неявная схема. Сделаем замену обозначения временных промежутков для более удобной записи:

$$\begin{aligned}
t_{01}&=t_j-t_{j-1}, \quad & t_{12}&=t_{j-1}-t_{j-2},\\
t_{02}&=t_j-t_{j-2}, \quad & t_{13}&=t_{j-1}-t_{j-3},\\
t_{03}&=t_j-t_{j-3}, \quad & t_{23}&=t_{j-2}-t_{j-3}.
\end{aligned}$$

Коэффициент γ и вектор-функция $\vec{F}$ имеют вид:

$$\gamma=\frac{\sigma(t_{01}t_{02}+t_{01}t_{03}+t_{02}t_{03})+2\varepsilon(t_{01}+t_{02}+t_{03})}{t_{01}t_{02}t_{03}},$$

$$\vec{F}=\vec{J^{\text{ст}}}+\frac{\sigma t_{02}t_{03}+2\varepsilon(t_{02}+t_{03})}{t_{01}t_{12}t_{13}}\vec{A^{\Leftarrow 1}}-\frac{\sigma t_{01}t_{03}+2\varepsilon(t_{01}+t_{03})}{t_{02}t_{12}t_{23}}\vec{A^{\Leftarrow 2}}+\frac{\sigma t_{01}t_{02}+2\varepsilon(t_{01}+t_{02})}{t_{03}t_{13}t_{23}}\vec{A^{\Leftarrow 3}}.$$

Локальные матрицы и вектор

В одномерном случае линейные базисные функции имеют вид:

$$\psi_1 = \mathrm{N}_1(\nu) = \frac{\nu_{r+1} - \nu}{h_\nu}, \psi_2 = \mathrm{N}_2(\nu) = \frac{\nu - \nu_r}{h_\nu}.$$

Перейдём в трехмерный случай и базисные функции получат вид:

$$\vec{\psi}_1 = \begin{pmatrix} Y_1(y)\cdot Z_1(z) \\ 0 \\ 0 \end{pmatrix}, \quad \vec{\psi}_2 = \begin{pmatrix} Y_2(y)\cdot Z_1(z) \\ 0 \\ 0 \end{pmatrix}, \quad \vec{\psi}_3 = \begin{pmatrix} 0 \\ X_1(x)\cdot Z_1(z) \\ 0 \end{pmatrix}, \quad \vec{\psi}_4 = \begin{pmatrix} 0 \\ X_2(x)\cdot Z_1(z) \\ 0 \end{pmatrix},$$

$$\vec{\psi}_5 = \begin{pmatrix} 0 \\ 0 \\ X_1(x)\cdot Y_1(y) \end{pmatrix}, \quad \vec{\psi}_6 = \begin{pmatrix} 0 \\ 0 \\ X_2(x)\cdot Y_1(y) \end{pmatrix}, \quad \vec{\psi}_7 = \begin{pmatrix} 0 \\ 0 \\ X_1(x)\cdot Y_2(y) \end{pmatrix}, \quad \vec{\psi}_8 = \begin{pmatrix} 0 \\ 0 \\ X_2(x)\cdot Y_2(y) \end{pmatrix},$$

$$\vec{\psi}_9 = \begin{pmatrix} Y_1(y)\cdot Z_2(z) \\ 0 \\ 0 \end{pmatrix}, \quad \vec{\psi}_{10} = \begin{pmatrix} Y_2(y)\cdot Z_2(z) \\ 0 \\ 0 \end{pmatrix}, \quad \vec{\psi}_{11} = \begin{pmatrix} 0 \\ X_1(x)\cdot Z_2(z) \\ 0 \end{pmatrix}, \quad \vec{\psi}_{12} = \begin{pmatrix} 0 \\ X_2(x)\cdot Z_2(z) \\ 0 \end{pmatrix}.$$

Локальные матрицы жёсткости и масс собираются по формулам:

$$G_{ij} = \int\limits_\Omega \mathrm{rot}\ \vec{\psi}_i \cdot \mathrm{rot}\ \vec{\psi}_j d\Omega, \quad M_{ij} = \int\limits_\Omega \gamma \vec{\psi}_i \cdot \vec{\psi}_j d\Omega.$$

Компоненты вектора b правой части определяются соотношением:

$$b_i = \int\limits_\Omega \vec{F} \cdot \vec{\psi}_i d\Omega$$

В программе используется численное интегрирование для вычисления локальных матриц. А вектор правой части вычисляется посредством умножения матрицы масс на вектор $\vec{F}$.

Входные файлы

GridParameters - файл для задания пространственной сетки. Файл состоит из 10 + n-строк (взависимости от количества заданных зон для коэффициента σ), каждая строка состоит из значений, где через пробел разделяются значения (в примере будет показано с использованием символа | для более понятного представления читателю):

1. *начало по x | конец по x | кол-во шагов по x | коэф. разрядки по x*
2. *разбиения слоёв по x*
3. *начало по y | конец по y | кол-во шагов по y | коэф. разрядки по y*
4. *разбиения слоёв по y*
5. *начало по z | конец по z | кол-во шагов по z | коэф. разрядки по z*
6. *разбиения слоёв по z*
7. *левая граница | правая граница | нижняя граница | верхняя граница | задняя граница | передняя граница*
8. *μ | ε*
9. *количество разрывов области n*

10 - 10 + n. значение σ | индекс начала зоны по x | индекс конца зоны по x | индекс начала зоны по y | индекс конца зоны по y | индекс начала зоны по z | индекс конца зоны по z

Пример задания области из будущего теста:

1. 0 5 4 1

2. 0 1 4 5
3. 0 5 4 1
4. 0 1 4 5
5. 0 5 4 1
6. 0 1 4 5
7. 1 1 1 1 1 1
8. 1 1
9. 4
10. 3 0 1 0 2 0 3
11. 1 0 1 2 3 0 3
12. 5 1 3 0 3 0 1
13. 3 1 3 0 3 1 3

TimeGridParameters - файл для задания временной сетки. Во временной сетке нужно всего 4 параметра:

1. начало по t | конец по t | кол-во шагов по t | коэф. разрядки по t

Пример:

1. 0 10 20 1.1

Описание и возможности программы

1. В программе можно выбрать как будут собираться временные слои: физически (собирая из начальных условий двухслойную неявную схему, далее трёхслойную и потом четырёхслойную) или сгенерировать точные значения на всех трёх слоях.
2. СЛАУ можно решать с помощью BCGStab-LU или решение методом LU-разложения.

Немного про классы в программе:

1. Основным классом является **FEM**, в котором происходит генерация портрета глобальной матрицы, сборка локальных матриц и векторов, учёт краевых условий.
2. Генерация пространственной сетки происходит в классе **Grid**, а временной в **TimeGrid**
3. Для реализации численного интегрирования были созданы 2 record struct-а: **SegmentGaussOrder9**, который является методом Гаусса-9, и **TriLinearVectorBasis** для удобного вычисления базисных вектор-функций.
4. Само численное интегрирование реализовано в классе **Integration**, в который передаются все квадратуры.
5. Оставшиеся классы реализуют структуры для удобной работы с основными классами. Например есть 2 класса **SparseMatrix** и **Vector**, которые используются для хранения глобальной матрицы и вектора соответственно.

Пример работы программы

· $\vec{A} = \begin{pmatrix} y^2 \\ x \\ z \end{pmatrix}$, $\vec{F} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$,

· Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$

· $\sigma_1 = 3, x \in [0, 1], y \in [0, 4], z \in [0, 5],$

· $\sigma_2 = 1, x \in [0, 1], y \in [4, 5], z \in [0, 5],$

· $\sigma_3 = 5, x \in [1, 5], y \in [0, 5], z \in [0, 1],$

· $\sigma_4 = 3, x \in [1, 5], y \in [0, 5], z \in [1, 5],$

· $\mu = 1, \qquad \varepsilon = 1,$

· Временная сетка: $t \in [0, 10], 20$ разбиений, $k = 1.1$

· Краевые условия первого рода заданы на всех границах

Полученный результат

t_i	Error
0.577913579971265	4.82489932000256E-14
0.8103011856938493	1.3940693470754484E-13
1.065927551988692	2.460891289198625E-13
1.3471165549130188	3.394249738126983E-13
1.6564244581297785	3.965816301698403E-13
1.9966631516682143	4.1775712889988677E-13
2.3709257145604936	4.167877265697349E-13
2.782614533742001	4.1079847720710554E-13
3.235472234841659	4.036306665353112E-13
3.733615706051283	3.959716120830492E-13
4.281573524381869	3.8966748523007065E-13
4.884327124545514	3.852655050578968E-13
5.547356084725524	3.8364013246706505E-13
6.2766879409235345	3.8273637662769855E-13
7.078952982741346	3.8463942052263713E-13
7.961444528740939	3.8717551802067576E-13
8.932185229340492	3.8647441249621437E-13
10	3.850171665089873E-13

Исследования

Порядок аппроксимации по пространству

Кубическая функция

· $\vec{A} = \begin{pmatrix} y^3 \\ x^3 \\ z^3 \end{pmatrix}$, $\vec{F} = \begin{pmatrix} -6y \\ -6x \\ 0 \end{pmatrix}$,

· Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$

· $\mu = 1$,

· Краевые условия первого рода заданы на всех границах

Полученный результат

Edge number	Approx value	Real value	Error
...	...	...	...
145	1.9531249999999951	1.953125	4.884981308350689E-15
146	15.625000000000009	15.625	-8.881784197001252E-15
147	52.734375000000014	52.734375	-1.4210854715202004E-14
148	124.99999999999999	125	1.4210854715202004E-14
149	15.624999999999993	15.625	7.105427357601002E-15
150	15.625000000000012	15.625	-1.2434497875801753E-14
151	15.624999999999972	15.625	2.842170943040401E-14
152	15.625000000000048	15.625	-4.796163466380676E-14
153	0	0	0
154	1.953125000000012	1.953125	-1.199040866595169E-14
155	15.624999999999972	15.625	2.842170943040401E-14
156	52.73437500000004	52.734375	-4.263256414560601E-14
157	124.99999999999999	125	1.4210854715202004E-14
158	52.734375000000036	52.734375	-3.552713678800501E-14
159	52.73437499999996	52.734375	4.263256414560601E-14
160	52.73437500000001	52.734375	-7.105427357601002E-15
161	52.73437499999997	52.734375	2.842170943040401E-14
162	0	0	0
163	1.9531249999999818	1.953125	1.8207657603852567E-14
164	15.624999999999998	15.625	1.7763568394002505E-15
165	52.73437500000006	52.734375	-5.684341886080802E-14
...	...	...	...

Полином 4-ой степени

· $\vec{A} = \begin{pmatrix} y^4 \\ x^3 \\ z^3 \end{pmatrix}$, $\vec{F} = \begin{pmatrix} -12y^2 \\ -6x \\ 0 \end{pmatrix}$,

· Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$,

· $\mu = 1$,

· Краевые условия первого рода заданы на всех границах

Полученный результат

Edge number	Approx value	Real value	Error
...	...	...	...
145	1.9535943868618157	1.953125	-0.00046938686181574596
146	15.624894556143284	15.625	0.00010544385671629186
147	52.734397299353276	52.734375	-2.2299353275911926E-05
148	125	125	0
149	38.92792438479348	39.0625	0.13457561520652206
150	38.91961306626871	39.0625	0.14288693373129036
151	38.9198804449208	39.0625	0.1426195550792002
152	38.91982890311402	39.0625	0.14267109688597657
153	0	0	0
154	1.9506819254358199	1.953125	0.0024430745641801366
155	15.625484562740365	15.625	-0.00048456274036468017
156	52.734282475815164	52.734375	9.252418483640668E-05
157	125	125	0
158	197.52904361364068	197.75390625	0.2248626363593189
159	197.57225233249338	197.75390625	0.18165391750662252
160	197.57198015227647	197.75390625	0.18192609772353308
161	197.57202986463363	197.75390625	0.1818763853663654
162	0	0	0
163	1.9556449007374526	1.953125	-0.002519900737452563
164	15.624515065902793	15.625	0.0004849340972068461
165	52.73446505373405	52.734375	-9.00537340484675E-05
...	...	...	...

Порядок аппроксимации = 3.

Порядок сходимости по пространству

· $\vec{A} = \begin{pmatrix} y^4 \\ x^3 \\ z^3 \end{pmatrix}$, $\vec{F} = \begin{pmatrix} -12y^2 \\ -6x \\ 0 \end{pmatrix}$,

· Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$,

· $\mu = 1$,

· Краевые условия первого рода заданы на всех границах

Полученный результат

Шаг	Погрешность	Порядок сходимости
h	0.07208790544253746	-
$\frac{h}{2}$	0.01921482055319743	1.9
$\frac{h}{4}$	0.004815259951729759	1.99

Порядок сходимости по пространству ≈ 2

Порядок аппроксимации по времени

Кубическая функция

$\cdot \vec{A} = \begin{pmatrix} t^2 \\ t \\ t^3 \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} 2\sigma t + 2\varepsilon \\ \sigma \\ 3\sigma t^2 + 6\varepsilon t \end{pmatrix},$

$\cdot$ Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$

$\cdot \sigma_1 = 3, x \in [0, 1], y \in [0, 4], z \in [0, 5],$

$\cdot \sigma_2 = 1, x \in [0, 1], y \in [4, 5], z \in [0, 5],$

$\cdot \sigma_3 = 5, x \in [1, 5], y \in [0, 5], z \in [0, 1],$

$\cdot \sigma_4 = 3, x \in [1, 5], y \in [0, 5], z \in [1, 5],$

$\cdot \mu = 1, \qquad \varepsilon = 1,$

$\cdot$ Временная сетка: $t \in [0, 10], 20$ разбиений, $h = 0.5$

$\cdot$ Краевые условия первого рода заданы на всех границах

Полученный результат

t_i	Error
1.5	1.9991414559945356E-15
2	6.37761687193886E-15
2.5	1.2639728638768716E-14
3	1.7825134592834246E-14
3.5	2.7429378944515652E-14
4	3.806478395271707E-14
4.5	5.5385384806962116E-14
5	7.392997489907755E-14
5.5	1.0414302595484017E-13
6	1.3811905512052903E-13
6.5	1.7544248823602672E-13
7	2.554108558836447E-13
7.5	3.3168480528180676E-13
8	4.3292679791742183E-13
8.5	5.323086164616099E-13
9	6.982133725139537E-13
9.5	8.716220214389105E-13
10	1.0333877247861136E-12

Полином 4-ой степени

$\cdot \vec{A} = \begin{pmatrix} t^2 \\ t \\ t^4 \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} 2\sigma t + 2\varepsilon \\ \sigma \\ 3\sigma t^2 + 6\varepsilon t \end{pmatrix},$

$\cdot$ Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$

$\cdot \sigma_1 = 3, x \in [0, 1], y \in [0, 4], z \in [0, 5],$

$\cdot \sigma_2 = 1, x \in [0, 1], y \in [4, 5], z \in [0, 5],$

$\cdot \sigma_3 = 5, x \in [1, 5], y \in [0, 5], z \in [0, 1],$

$\cdot \sigma_4 = 3, x \in [1, 5], y \in [0, 5], z \in [1, 5],$

$\cdot \mu = 1, \qquad \varepsilon = 1,$

$\cdot$ Временная сетка: $t \in [0, 10], 20$ разбиений, $h = 0.5$

$\cdot$ Краевые условия первого рода заданы на всех границах

Полученный результат

t_i	Error
1.5	0.17562942397686412
2	0.5029113632739743
2.5	0.902490618979112
3	1.3073070370088904
3.5	1.6794836618216342
4	2.004470006394667
4.5	2.2809878098552385
5	2.513779014084634
5.5	2.709514111022524
6	2.8747012085723154
6.5	3.01482306605559
7	3.1342172274543323
7.5	3.2362820367769056
8	3.32372331242698
8.5	3.398736590796777
9	3.4631287289165638
9.5	3.5184050664893567
10	3.565836385631419

Порядок аппроксимации = 4.

Порядок сходимости по времени

· $\vec{A} = \begin{pmatrix} t^2 \\ t \\ t^4 \end{pmatrix}$, $\vec{F} = \begin{pmatrix} 2\sigma t + 2\varepsilon \\ \sigma \\ 3\sigma t^2 + 6\varepsilon t \end{pmatrix}$,

· Пространственная сетка: $x, y, z \in [0, 5], h_{x,y,z} = 1.25$

· $\sigma_1 = 3, x \in [0, 1], y \in [0, 4], z \in [0, 5]$,

· $\sigma_2 = 1, x \in [0, 1], y \in [4, 5], z \in [0, 5]$,

· $\sigma_3 = 5, x \in [1, 5], y \in [0, 5], z \in [0, 1]$,

· $\sigma_4 = 3, x \in [1, 5], y \in [0, 5], z \in [1, 5]$,

· $\mu = 1, \qquad \varepsilon = 1$,

· Временная сетка: $t \in [0, 10], 20$ разбиений, $h = 0.5$

· Краевые условия первого рода заданы на всех границах

Полученный результат

Шаг	Погрешность	Порядок сходимости
h	0.17562942397686412	-
$\frac{h}{2}$	0.013906410305277344	3.65
$\frac{h}{4}$	0.001020206923985269	3.76

Порядок сходимости по времени ≈ 4.

Листинг

FEM.cs

```
namespace VectorFEM3D;

public class FEM
{
    private SparseMatrix? _globalMatrix;
    private Vector? _globalVector;
    private Vector[]? _layers;
    private Vector? _solution;
    private Vector? _localVector;
    private Matrix? _stiffnessMatrix;
    private Matrix? _massMatrix;
    private Grid? _grid;
    private TimeGrid _timeGrid;
    private Test? _test;
    private IBasis3D? _basis;
    private Integration? _integration;
    private SLAE? _slae;
    private Scheme _scheme;

    public FEM(Grid grid, TimeGrid timeGrid)
    {
        _grid = grid;
        _timeGrid = timeGrid;
        _basis = new TriLinearVectorBasis();
        _integration = new Integration(new SegmentGaussOrder9());
        _stiffnessMatrix = new(_basis.Size);
        _massMatrix = new(_basis.Size);
        _localVector = new(_basis.Size);
    }

    public void SetTest(Test test)
    {
        _test = test;
    }

    public void SetSolver(SLAE slae)
    {
        _slae = slae;
    }

    public void SetScheme(Scheme scheme)
    {
        _scheme = scheme;
    }

    public void Compute()
    {
        BuildPortrait();
        PrepareLayers();

        int itime = 0;

        switch (_scheme)
```

```
        {
            case Scheme.Three_layer_Implicit:
                itime = 2;
                break;

            case Scheme.Four_layer_Implicit:
                itime = 3;
                break;
        }

        for ( ; itime < _timeGrid.TGrid.Length; itime++)
        {
            AssemblySLAE(itime);
            AccountDirichletBoundaries(itime);

            _slae.SetSLAE(_globalVector, _globalMatrix);
            _solution = _slae.Solve();

            switch (_scheme)
            {
                case Scheme.Three_layer_Implicit:
                    Vector.Copy(_layers[1], _layers[0]);
                    Vector.Copy(_solution, _layers[1]);
                    break;

                case Scheme.Four_layer_Implicit:
                    Vector.Copy(_layers[1], _layers[0]);
                    Vector.Copy(_layers[2], _layers[1]);
                    Vector.Copy(_solution, _layers[2]);
                    break;
            }

            PrintError(itime);
        }
    }

    private void AssemblySLAE(int itime)
    {
        _globalVector.Fill(0);
        _globalMatrix.Clear();

        for (int ielem = 0; ielem < _grid.Elements.Length; ielem++)
        {
            AssemblyLocalElement(ielem, itime);

            _stiffnessMatrix += SchemeUsage(ielem, itime, _scheme, 0) * _massMatrix;

            for (int i = 0; i < _basis.Size; i++)
            {
                for (int j = 0; j < _basis.Size; j++)
                {
                    AddElement(_grid.Elements[ielem][i], _grid.Elements[ielem][j],
_stiffnessMatrix[i, j]);
                }
            }
```

```
            AssemblyGlobalVector(ielem, itime);

            _stiffnessMatrix.Clear();
            _massMatrix.Clear();
            _localVector.Fill(0);
        }
    }

    private void AssemblyGlobalVector(int ielem, int itime)
    {
        double[] qj3 = new double[_basis.Size];
        double[] qj2 = new double[_basis.Size];
        double[] qj1 = new double[_basis.Size];

        switch (_scheme)
        {
            case Scheme.Three_layer_Implicit:

                for (int i = 0; i < _basis.Size; i++)
                {
                    for (int j = 0; j < _basis.Size; j++)
                    {
                        qj2[i] += _massMatrix[i, j] * _layers[1]
[_grid.Elements[ielem][j]];
                        qj1[i] += _massMatrix[i, j] * _layers[0]
[_grid.Elements[ielem][j]];
                    }
                }

                for (int i = 0; i < _basis.Size; i++)
                {
                    _localVector[i] += SchemeUsage(ielem, itime, _scheme, 1) *
qj2[i];
                    _localVector[i] += SchemeUsage(ielem, itime, _scheme, 2) *
qj1[i];

                    _globalVector[_grid.Elements[ielem][i]] += _localVector[i];
                }

                break;

            case Scheme.Four_layer_Implicit:

                for (int i = 0; i < _basis.Size; i++)
                {
                    for (int j = 0; j < _basis.Size; j++)
                    {
                        qj3[i] += _massMatrix[i, j] * _layers[2]
[_grid.Elements[ielem][j]];
                        qj2[i] += _massMatrix[i, j] * _layers[1]
[_grid.Elements[ielem][j]];
                        qj1[i] += _massMatrix[i, j] * _layers[0]
[_grid.Elements[ielem][j]];
                    }
                }
```

```
                for (int i = 0; i < _basis.Size; i++)
                {
                    _localVector[i] += SchemeUsage(ielem, itime, _scheme, 1) *
qj3[i];
                    _localVector[i] += SchemeUsage(ielem, itime, _scheme, 2) *
qj2[i];
                    _localVector[i] += SchemeUsage(ielem, itime, _scheme, 3) *
qj1[i];

                    _globalVector[_grid.Elements[ielem][i]] += _localVector[i];
                }

                break;
            }
        }

        private void AddElement(int i, int j, double value)
        {
            if (i == j)
            {
                _globalMatrix.Di[i] += value;
                return;
            }

            if (i > j)
            {
                for (int icol = _globalMatrix.Ig[i]; icol < _globalMatrix.Ig[i + 1];
icol++)
                {
                    if (_globalMatrix.Jg[icol] == j)
                    {
                        _globalMatrix.Ggl[icol] += value;
                        return;
                    }
                }
            }

            else
            {
                for (int icol = _globalMatrix.Ig[j]; icol < _globalMatrix.Ig[j + 1];
icol++)
                {
                    if (_globalMatrix.Jg[icol] == i)
                    {
                        _globalMatrix.Ggu[icol] += value;
                        return;
                    }
                }
            }
        }

        private void AssemblyLocalElement(int ielem, int itime)
        {
            double hx = _grid.Edges[_grid.Elements[ielem][0]].Length;
            double hy = _grid.Edges[_grid.Elements[ielem][2]].Length;
            double hz = _grid.Edges[_grid.Elements[ielem][4]].Length;
```

```
        for (int i = 0; i < _basis.Size; i++)
        {
            for (int j = 0; j < _basis.Size; j++)
            {
                Func<Point3D, double> kek;
                Vector3D psi1 = new(0, 0, 0);
                Vector3D psi2 = new(0, 0, 0);
                Vector3D dPsi1 = new(0, 0, 0);
                Vector3D dPsi2 = new(0, 0, 0);

                int ik = i;
                int jk = j;
                kek = point =>
                {
                    psi1.Copy(_basis.GetPsi(ik, point));
                    psi2.Copy(_basis.GetPsi(jk, point));

                    return psi1 * psi2;
                };

                _massMatrix[i, j] += hx * hy * hz * _integration.Gauss3D(kek);

                kek = point =>
                {
                    dPsi1.Copy(_basis.GetDPsi(ik, point));
                    dPsi2.Copy(_basis.GetDPsi(jk, point));

                    return Vector3D.DotProductJacob(dPsi1, dPsi2, hx, hy, hz);
                };

                _stiffnessMatrix[i, j] += 1 / _grid.Mu * _integration.Gauss3D(kek);
            }

            _localVector[i] = _test.F(_grid.Edges[_grid.Elements[ielem][i]].Point,
_timeGrid[itime], i, _grid.GetSigma(
                new Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                    _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                    _grid.Edges[_grid.Elements[ielem][11]].Point.Z)));
        }

        _localVector = _massMatrix * _localVector;
    }

    private void AccountDirichletBoundaries(int itime)
    {
        foreach (var edge in _grid.DirichletBoundaries)
        {

            _globalMatrix.Di[edge] = 1;
            _globalVector[edge] = _test.UValue(_grid.Edges[edge].Point,
_timeGrid[itime], _grid.Edges[edge].GetAxis());

            for (int i = _globalMatrix.Ig[edge]; i < _globalMatrix.Ig[edge + 1]; i++)
                _globalMatrix.Ggl[i] = 0;
```

```
            for (int col = edge + 1; col < _globalMatrix.Size; col++)
                for (int j = _globalMatrix.Ig[col]; j < _globalMatrix.Ig[col + 1]; j+
+)
                    if (_globalMatrix.Jg[j] == edge)
                    {
                        _globalMatrix.Ggu[j] = 0;
                        break;
                    }
        }
    }

    private double SchemeUsage(int ielem, int itime, Scheme scheme, int i)
    {
       double t01 = _timeGrid[itime] - _timeGrid[itime - 1];
       double t02 = _timeGrid[itime] - _timeGrid[itime - 2];
       double t12 = _timeGrid[itime - 1] - _timeGrid[itime - 2];

        switch (scheme)
        {
            case Scheme.Three_layer_Implicit:
                switch (i)
                {
                    case 0:
                        return (_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                    _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                    _grid.Edges[_grid.Elements[ielem][11]].Point.Z)) *
(t01 + t02) + 2 * _grid.Epsilon) /
                                (t01 * t02);

                    case 1:
                        return (_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                _grid.Edges[_grid.Elements[ielem][11]].Point.Z)) * t02 +
2 * _grid.Epsilon) / (t01 * t12);

                    case 2:
                        return -(_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                _grid.Edges[_grid.Elements[ielem][11]].Point.Z)) * t01 +
2 * _grid.Epsilon) / (t02 * t12);

                }

                break;

            case Scheme.Four_layer_Implicit:
                double t03 = _timeGrid[itime] - _timeGrid[itime - 3];
                double t13 = _timeGrid[itime - 1] - _timeGrid[itime - 3];
                double t23 = _timeGrid[itime - 2] - _timeGrid[itime - 3];

                switch (i)
                {
                    case 0:
```

```
                        return (_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                               _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                               _grid.Edges[_grid.Elements[ielem]
[11]].Point.Z))
                                           * (t01 * t02 + t01 * t03 + t02 * t03) + 2 *
_grid.Epsilon * (t01 + t02 + t03)) /
                                       (t01 * t02 * t03);

                    case 1:
                        return (_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                           _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                           _grid.Edges[_grid.Elements[ielem][11]].Point.Z))
* t02 * t03 +
                                       2 * _grid.Epsilon * (t02 + t03)) / (t01 * t12 * t13);

                    case 2:
                        return -(_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                            _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                            _grid.Edges[_grid.Elements[ielem][11]].Point.Z))
* t01 * t03 +
                                        2 * _grid.Epsilon * (t01 + t03)) / (t02 * t12 *
t23);

                    case 3:
                        return (_grid.GetSigma(new
Point3D(_grid.Edges[_grid.Elements[ielem][0]].Point.X,
                                           _grid.Edges[_grid.Elements[ielem][3]].Point.Y,
                                           _grid.Edges[_grid.Elements[ielem][11]].Point.Z))
* t01 * t02 +
                                       2 * _grid.Epsilon * (t01 + t02)) / (t03 * t13 * t23);
                }

                return 0;

            default:
                throw new Exception("Undefined scheme");
        }

        throw new Exception("SchemeUsage can't return a value");
    }

    private void BuildPortrait()
    {
        HashSet<int>[] list = new HashSet<int>[_grid.Edges.Length].Select(_ => new
HashSet<int>()).ToArray();
        foreach (var element in _grid.Elements)
            foreach (var pos in element)
                foreach (var node in element)
                    if (pos > node)
                        list[pos].Add(node);

        list = list.Select(childlist => childlist.Order().ToHashSet()).ToArray();
        int count = list.Sum(childlist => childlist.Count);
```

```
        _globalMatrix = new(_grid.Edges.Length, count);
        _globalVector = new(_grid.Edges.Length);
        _layers = new Vector[3].Select(_ => new
Vector(_grid.Edges.Length)).ToArray();

        _globalMatrix.Ig[0] = 0;

        for (int i = 0; i < list.Length; i++)
            _globalMatrix.Ig[i + 1] = _globalMatrix.Ig[i] + list[i].Count;

        int k = 0;

        foreach (var childlist in list)
            foreach (var value in childlist)
                _globalMatrix.Jg[k++] = value;
    }

    private void PrepareLayers()
    {
        switch (_scheme)
        {
            case Scheme.Three_layer_Implicit:
                for (int i = 0; i < _grid.Edges.Length; i++)
                {
                    _layers[0][i] = _test.UValue(_grid.Edges[i].Point, _timeGrid[0],
_grid.Edges[i].GetAxis());
                    _layers[1][i] = _test.UValue(_grid.Edges[i].Point, _timeGrid[1],
_grid.Edges[i].GetAxis());
                }

                break;

            case Scheme.Four_layer_Implicit:
                for (int i = 0; i < _grid.Edges.Length; i++)
                {
                    _layers[0][i] = _test.UValue(_grid.Edges[i].Point, _timeGrid[0],
_grid.Edges[i].GetAxis());
                    _layers[1][i] = _test.UValue(_grid.Edges[i].Point, _timeGrid[1],
_grid.Edges[i].GetAxis());
                    _layers[2][i] = _test.UValue(_grid.Edges[i].Point, _timeGrid[2],
_grid.Edges[i].GetAxis());
                }

                break;
        }
    }

    private void PrintError(int itime)
    {
        double error = 0;
        for (int i = 0; i < _grid.Edges.Length; i++)
        {
            error += Math.Pow(
                _test.UValue(_grid.Edges[i].Point, _timeGrid[itime],
_grid.Edges[i].GetAxis()) - _solution[i], 2);
```

```
        }
        Console.WriteLine($"Layer error {itime} = {Math.Sqrt(error /
_grid.Edges.Length)}");
    }
}
```

SLAE.cs

```
namespace VectorFEM3D;
public abstract class SLAE
{
   protected SparseMatrix matrix = default!;
   protected Vector vector = default!;
   public Vector solution = default!;
   public double time;
   protected double eps;
   protected int maxIters;
   public int lastIter;

   public SLAE()
   {
      eps = 1e-16;
      maxIters = 2000;
   }

   public SLAE(double eps, int maxIters)
   {
      this.eps = eps;
      this.maxIters = maxIters;
   }

   public void SetSLAE(Vector vector, SparseMatrix matrix)
   {
      this.vector = vector;
      this.matrix = matrix;
   }

   public abstract Vector Solve();


   protected void LU()
   {
      for (int i = 0; i < matrix.Size; i++)
      {

         for (int j = matrix.Ig[i]; j < matrix.Ig[i + 1]; j++)
         {
            int jCol = matrix.Jg[j];
            int jk = matrix.Ig[jCol];
            int k = matrix.Ig[i];

            int sdvig = matrix.Jg[matrix.Ig[i]] - matrix.Jg[matrix.Ig[jCol]];

            if (sdvig > 0)
               jk += sdvig;
            else
               k -= sdvig;
```

```
            double sumL = 0.0;
            double sumU = 0.0;

            for (; k < j && jk < matrix.Ig[jCol + 1]; k++, jk++)
            {
               sumL += matrix.Ggl[k] * matrix.Ggu[jk];
               sumU += matrix.Ggu[k] * matrix.Ggl[jk];
            }

            matrix.Ggl[j] -= sumL;
            matrix.Ggu[j] -= sumU;
            matrix.Ggu[j] /= matrix.Di[jCol];
         }

         double sumD = 0.0;
         for (int j = matrix.Ig[i]; j < matrix.Ig[i + 1]; j++)
            sumD += matrix.Ggl[j] * matrix.Ggu[j];

         matrix.Di[i] -= sumD;
      }
   }

   protected void ForwardElimination()
   {
      for (int i = 0; i < matrix.Size; i++)
      {
         for (int j = matrix.Ig[i]; j < matrix.Ig[i + 1]; j++)
         {
            solution[i] -= matrix.Ggl[j] * solution[matrix.Jg[j]];
         }

         solution[i] /= matrix.Di[i];
      }
   }

   protected void BackwardSubstitution()
   {
      for (int i = matrix.Size - 1; i >= 0; i--)
      {
         for (int j = matrix.Ig[i + 1] - 1; j >= matrix.Ig[i]; j--)
         {
            solution[matrix.Jg[j]] -= matrix.Ggu[j] * solution[i];
         }
      }
   }

   public void PrintSolution()
   {
      for(int i = 0; i < solution.Length; i++)
      {
         Console.WriteLine(solution[i]);
      }
   }
}
```

```
public class BCGSTABLUSolver : SLAE
{
   public BCGSTABLUSolver(double eps, int maxIters) : base(eps, maxIters) { }
   public override Vector Solve()
   {
      solution = new(vector.Length);

      double vecNorm = vector.Norm();

      SparseMatrix matrixLU = new(matrix.Size, matrix.Jg.Length);
      SparseMatrix.Copy(matrix, matrixLU);

      Vector r = new(vector.Length);
      Vector p = new(vector.Length);
      Vector s;
      Vector t;
      Vector v = new(vector.Length);

      double alpha = 1.0;
      double beta;
      double omega = 1.0;
      double rho = 1.0;
      double rhoPrev;

      int i;

      LU(matrixLU);

      Vector r0 = DirElim(matrixLU, vector - matrix * solution);
      Vector.Copy(r0, r);

      for (i = 1; i <= maxIters && r.Norm() / vecNorm > eps; i++)
      {
         rhoPrev = rho;
         rho = (r0 * r);

         beta = rho / rhoPrev * alpha / omega;

         p = r + beta * (p - omega * v);

         v = DirElim(matrixLU, matrix * BackSub(matrixLU, p));

         alpha = rho / (r0 * v);

         s = r - alpha * v;

         t = DirElim(matrixLU, matrix * BackSub(matrixLU, s));

         omega = (t * s) / (t * t);

         solution = solution + omega * s + alpha * p;

         r = s - omega * t;
      }
```

```
   solution = BackSub(matrixLU, solution);

   return solution;
}

protected static void LU(SparseMatrix Matrix)
{
   for (int i = 0; i < Matrix.Size; i++)
   {

      for (int j = Matrix.Ig[i]; j < Matrix.Ig[i + 1]; j++)
      {
         int jCol = Matrix.Jg[j];
         int jk = Matrix.Ig[jCol];
         int k = Matrix.Ig[i];

         int sdvig = Matrix.Jg[Matrix.Ig[i]] - Matrix.Jg[Matrix.Ig[jCol]];

         if (sdvig > 0)
            jk += sdvig;
         else
            k -= sdvig;

         double sumL = 0.0;
         double sumU = 0.0;

         for (; k < j && jk < Matrix.Ig[jCol + 1]; k++, jk++)
         {
            sumL += Matrix.Ggl[k] * Matrix.Ggu[jk];
            sumU += Matrix.Ggu[k] * Matrix.Ggl[jk];
         }

         Matrix.Ggl[j] -= sumL;
         Matrix.Ggu[j] -= sumU;
         Matrix.Ggu[j] /= Matrix.Di[jCol];
      }

      double sumD = 0.0;
      for (int j = Matrix.Ig[i]; j < Matrix.Ig[i + 1]; j++)
         sumD += Matrix.Ggl[j] * Matrix.Ggu[j];

      Matrix.Di[i] -= sumD;
   }
}

protected static Vector DirElim(SparseMatrix Matrix, Vector b)
{
   Vector result = new Vector(b.Length);
   Vector.Copy(b, result);

   for (int i = 0; i < Matrix.Size; i++)
   {
      for (int j = Matrix.Ig[i]; j < Matrix.Ig[i + 1]; j++)
      {
         result[i] -= Matrix.Ggl[j] * result[Matrix.Jg[j]];
      }
```

```
            result[i] /= Matrix.Di[i];
        }

        return result;
    }

    protected static Vector BackSub(SparseMatrix Matrix, Vector b)
    {
        Vector result = new Vector(b.Length);
        Vector.Copy(b, result);

        for (int i = Matrix.Size - 1; i >= 0; i--)
        {
            for (int j = Matrix.Ig[i + 1] - 1; j >= Matrix.Ig[i]; j--)
            {
                result[Matrix.Jg[j]] -= Matrix.Ggu[j] * result[i];
            }
        }
        return result;
    }
}

public class LUSolver : SLAE
{
    public override Vector Solve()
    {
        solution = new(vector.Length);
        Vector.Copy(vector, solution);
        matrix = matrix.ConvertToProfile();

        LU();
        ForwardElimination();
        BackwardSubstitution();

        return solution;
    }
}
```

Grid.cs

```
namespace VectorFEM3D;

public class Grid
{
    private readonly double _xStart;
    private readonly double _xEnd;
    private readonly int _xSteps;
    private readonly double _xRaz;
    private readonly double _yStart;
    private readonly double _yEnd;
    private readonly int _ySteps;
    private readonly double _yRaz;
    private readonly double _zStart;
    private readonly double _zEnd;
    private readonly int _zSteps;
    private readonly double _zRaz;
```

```
    private readonly int[] _boundaries;

    private readonly double[] _xZones;
    private readonly double[] _yZones;
    private readonly double[] _zZones;
    private readonly int[][] _zones;
    private readonly double[] _sigmaValues;

    public Point3D[] Nodes { get; private set; }
    public Edge3D[] Edges { get; private set; }
    public HashSet<int> DirichletBoundaries { get; private set; }
    public List<(HashSet<(int, int)>, ElementSide)> NewmanBoundaries { get; private
set; }
    public int[][] Elements { get; private set; }
    public double Mu { get; set; }
    public double Sigma { get; set; }
    public double Epsilon { get; set; }

    public Grid(string path)
    {
        using (var sr = new StreamReader(path))
        {
            string[] data;
            data = sr.ReadLine()!.Split(" ").ToArray();
            _xStart = Convert.ToDouble(data[0]);
            _xEnd = Convert.ToDouble(data[1]);
            _xSteps = Convert.ToInt32(data[2]);
            _xRaz = Convert.ToDouble(data[3]);

            _xZones = sr.ReadLine()!.Split(" ").Select(x =>
Convert.ToDouble(x)).ToArray();

            data = sr.ReadLine()!.Split(" ").ToArray();
            _yStart = Convert.ToDouble(data[0]);
            _yEnd = Convert.ToDouble(data[1]);
            _ySteps = Convert.ToInt32(data[2]);
            _yRaz = Convert.ToDouble(data[3]);

            _yZones = sr.ReadLine()!.Split(" ").Select(x =>
Convert.ToDouble(x)).ToArray();

            data = sr.ReadLine()!.Split(" ").ToArray();
            _zStart = Convert.ToDouble(data[0]);
            _zEnd = Convert.ToDouble(data[1]);
            _zSteps = Convert.ToInt32(data[2]);
            _zRaz = Convert.ToDouble(data[3]);

            _zZones = sr.ReadLine()!.Split(" ").Select(x =>
Convert.ToDouble(x)).ToArray();

            data = sr.ReadLine()!.Split(" ").ToArray();
            _boundaries = new int[6];
            _boundaries[0] = Convert.ToInt32(data[0]);
            _boundaries[1] = Convert.ToInt32(data[1]);
            _boundaries[2] = Convert.ToInt32(data[2]);
            _boundaries[3] = Convert.ToInt32(data[3]);
```

```
            _boundaries[4] = Convert.ToInt32(data[4]);
            _boundaries[5] = Convert.ToInt32(data[5]);

            data = sr.ReadLine()!.Split(" ").ToArray();
            Mu = Convert.ToDouble(data[0]);
            //Sigma = Convert.ToDouble(data[1]);
            Epsilon = Convert.ToDouble(data[1]);

            int kek = Convert.ToInt32(sr.ReadLine());
            _zones = new int[kek].Select(_ => new int[6]).ToArray();
            _sigmaValues = new double[kek];

            for (int i = 0; i < kek; i++)
            {
                data = sr.ReadLine()!.Split(" ").ToArray();
                _sigmaValues[i] = Convert.ToDouble(data[0]);
                _zones[i][0] = Convert.ToInt32(data[1]);
                _zones[i][1] = Convert.ToInt32(data[2]);
                _zones[i][2] = Convert.ToInt32(data[3]);
                _zones[i][3] = Convert.ToInt32(data[4]);
                _zones[i][4] = Convert.ToInt32(data[5]);
                _zones[i][5] = Convert.ToInt32(data[6]);
            }
        }
    }

    public void BuildGrid()
    {
        Elements = new int[_xSteps * _ySteps * _zSteps].Select(_ => new
int[12]).ToArray();
        Nodes = new Point3D[(_xSteps + 1) * (_ySteps + 1) * (_zSteps + 1)];

        double sumRazX = 0, sumRazY = 0, sumRazZ = 0;
        for (int i = 0; i < _xSteps; i++)
            sumRazX += Math.Pow(_xRaz, i);

        for (int i = 0; i < _ySteps; i++)
            sumRazY += Math.Pow(_yRaz, i);

        for (int i = 0; i < _zSteps; i++)
            sumRazZ += Math.Pow(_zRaz, i);

        int nodesInRow = _xSteps + 1;
        int nodesInSlice = nodesInRow * (_ySteps + 1);

        int zEdges = _zSteps * nodesInSlice;
        int xEdges = _xSteps;
        int yEdges = 1 + _xSteps;
        int edgesInSlice = xEdges * (1 + _ySteps) + yEdges * _ySteps;

        Edges = new Edge3D[edgesInSlice * (_zSteps + 1) + zEdges];

        double x = _xStart, y = _yStart, z = _zStart;
        double xStep = (_xEnd - _xStart) / sumRazX;
        double yStep = (_yEnd - _yStart) / sumRazY;
        double zStep = (_zEnd - _zStart) / sumRazZ;
```

```
        DirichletBoundaries = new();
        NewmanBoundaries = new();

        for (int j = 0; j < _xSteps; j++)
        {
            Nodes[j] = new(x, y, z);
            x += xStep;
            xStep *= _xRaz;
        }

        Nodes[_xSteps] = new(_xEnd, y, z);

        for (int i = 1; i <= _ySteps; i++)
        {
            y += yStep;
            yStep *= _yRaz;
            for (int j = 0; j < _xSteps + 1; j++)
            {
                Nodes[i * nodesInRow + j] = new(Nodes[j].X, y, z);
            }
        }

        for (int i = 1; i <= _zSteps; i++)
        {
            z += zStep;
            zStep *= _zRaz;
            for (int j = 0; j < _ySteps + 1; j++)
            {
                for (int k = 0; k < _xSteps + 1; k++)
                    Nodes[i * nodesInSlice + j * nodesInRow + k] = new(Nodes[k].X,
Nodes[j * nodesInRow].Y, z);
            }
        }

        int index = 0;

        for (int j = 0; j < _zSteps + 1; j++)
        {
            int xLocal = 0;
            int yLocal = 0;
            int zLocal = 0;


            for (int i = 0; i < nodesInSlice / nodesInRow; i++)
            {
                for (int k = 0; k < nodesInRow - 1; k++)
                {
                    Edges[index++] = new Edge3D(Nodes[xLocal + nodesInSlice * j],
Nodes[xLocal + nodesInSlice * j + 1]);
                    xLocal++;
                }

                xLocal++;

                if (i != nodesInSlice / nodesInRow - 1)
```

```
                {
                    for (int k = 0; k < nodesInRow; k++)
                    {
                        Edges[index++] = new Edge3D(Nodes[yLocal + nodesInSlice * j],
                            Nodes[yLocal + nodesInSlice * j + nodesInRow]);
                        yLocal++;
                    }
                }
            }

            if (j != _zSteps)
            {
                for (int k = 0; k < nodesInSlice; k++)
                {
                    Edges[index++] = new Edge3D(Nodes[zLocal + nodesInSlice * j],
                        Nodes[zLocal + nodesInSlice * j + nodesInSlice]);
                    zLocal++;
                }
            }
        }

        index = 0;

        for (int k = 0; k < _zSteps; k++)
        {
            for (int i = 0; i < _ySteps; i++)
            {
                for (int j = 0; j < _xSteps; j++)
                {
                    Elements[index][0] = j + (nodesInSlice + edgesInSlice) * k +
(xEdges + yEdges) * i;
                    Elements[index][1] = j + (nodesInSlice + edgesInSlice) * k +
(xEdges + yEdges) * (i + 1);
                    Elements[index][2] = j + (nodesInSlice + edgesInSlice) * k +
(xEdges + yEdges) * i + xEdges;
                    Elements[index][3] = j + (nodesInSlice + edgesInSlice) * k +
(xEdges + yEdges) * i + xEdges + 1;

                    Elements[index][4] =
                        j + (nodesInSlice + edgesInSlice) * k + (xEdges + yEdges) * i
+ edgesInSlice - _xSteps * i;
                    Elements[index][5] =
                        j + (nodesInSlice + edgesInSlice) * k + (xEdges + yEdges) * i
+ edgesInSlice + 1 - _xSteps * i;
                    Elements[index][6] =
                        j + (nodesInSlice + edgesInSlice) * k + (xEdges + yEdges) * i
+ edgesInSlice + nodesInRow -
                        _xSteps * i;
                    Elements[index][7] =
                        j + (nodesInSlice + edgesInSlice) * k + (xEdges + yEdges) * i
+ edgesInSlice + 1 +
                        nodesInRow - _xSteps * i;

                    Elements[index][8] = j + (nodesInSlice + edgesInSlice) * (k + 1)
+ (xEdges + yEdges) * i;
                    Elements[index][9] = j + (nodesInSlice + edgesInSlice) * (k + 1)
```

```
+ (xEdges + yEdges) * (i + 1);
                    Elements[index][10] = j + (nodesInSlice + edgesInSlice) * (k + 1)
+ (xEdges + yEdges) * i + xEdges;
                    Elements[index++][11] = j + (nodesInSlice + edgesInSlice) * (k +
1) + (xEdges + yEdges) * i + xEdges + 1;
                }
            }
        }
    }

    public double GetSigma(Point3D point)
    {
        for (int i = 0; i < _zones.Length; i++)
        {
            if (point.X <= _xZones[_zones[i][1]] && point.Y <= _yZones[_zones[i][3]]
&& point.Z <= _zZones[_zones[i][5]] &&
                point.X > _xZones[_zones[i][0]] && point.Y > _yZones[_zones[i][2]] &&
point.Z > _zZones[_zones[i][4]])
            {
                return _sigmaValues[i];
            }
        }

        throw new Exception("Can't find eligible zone for sigma");
    }

    public void AccountBoundaryConditions()
    {
        for (int ielem = 0; ielem < Elements.Length; ielem++)
        {
            if (ielem < _xSteps * _ySteps)
            {
                if (_boundaries[2] == 1) DirichletBoundary(ElementSide.Bottom,
ielem);
            }

            if (ielem >= _xSteps * _ySteps * _zSteps - _xSteps * _ySteps || _zSteps
== 1)
            {
                if (_boundaries[3] == 1) DirichletBoundary(ElementSide.Upper, ielem);
            }

            if (ielem % _xSteps == 0)
            {
                if (_boundaries[0] == 1) DirichletBoundary(ElementSide.Left, ielem);
            }

            if ((ielem + 1) % _xSteps == 0)
            {
                if (_boundaries[1] == 1) DirichletBoundary(ElementSide.Right, ielem);
            }

            if (ielem % (_xSteps * _ySteps) < _xSteps)
            {
                if (_boundaries[5] == 1) DirichletBoundary(ElementSide.Front, ielem);
            }
```

```
            if (ielem % (_xSteps * _ySteps) >= _xSteps * _ySteps - _xSteps)
            {
                if (_boundaries[4] == 1) DirichletBoundary(ElementSide.Rear, ielem);
            }
        }
    }

    private void DirichletBoundary(ElementSide elementSide, int ielem)
    {
        switch (elementSide)
        {
            case ElementSide.Bottom:
                DirichletBoundaries.Add(Elements[ielem][0]);
                DirichletBoundaries.Add(Elements[ielem][1]);
                DirichletBoundaries.Add(Elements[ielem][2]);
                DirichletBoundaries.Add(Elements[ielem][3]);
                break;

            case ElementSide.Upper:
                DirichletBoundaries.Add(Elements[ielem][8]);
                DirichletBoundaries.Add(Elements[ielem][9]);
                DirichletBoundaries.Add(Elements[ielem][10]);
                DirichletBoundaries.Add(Elements[ielem][11]);
                break;

            case ElementSide.Left:
                DirichletBoundaries.Add(Elements[ielem][2]);
                DirichletBoundaries.Add(Elements[ielem][4]);
                DirichletBoundaries.Add(Elements[ielem][6]);
                DirichletBoundaries.Add(Elements[ielem][10]);
                break;

            case ElementSide.Right:
                DirichletBoundaries.Add(Elements[ielem][3]);
                DirichletBoundaries.Add(Elements[ielem][5]);
                DirichletBoundaries.Add(Elements[ielem][7]);
                DirichletBoundaries.Add(Elements[ielem][11]);
                break;

            case ElementSide.Front:
                DirichletBoundaries.Add(Elements[ielem][0]);
                DirichletBoundaries.Add(Elements[ielem][4]);
                DirichletBoundaries.Add(Elements[ielem][5]);
                DirichletBoundaries.Add(Elements[ielem][8]);
                break;

            case ElementSide.Rear:
                DirichletBoundaries.Add(Elements[ielem][1]);
                DirichletBoundaries.Add(Elements[ielem][6]);
                DirichletBoundaries.Add(Elements[ielem][7]);
                DirichletBoundaries.Add(Elements[ielem][9]);
                break;
        }
    }
}
```

```
public class TimeGrid
{
    private readonly double _tStart;
    private readonly double _tEnd;
    private readonly int _tSteps;
    private readonly double _tRaz;
    public double[] TGrid { get; set;  }

    public TimeGrid(string path)
    {
        using (var sr = new StreamReader(path))
        {
            string[] data;
            data = sr.ReadLine()!.Split(" ").ToArray();
            _tStart = Convert.ToDouble(data[0]);
            _tEnd = Convert.ToDouble(data[1]);
            _tSteps = Convert.ToInt32(data[2]);
            _tRaz = Convert.ToDouble(data[3]);
            TGrid = new double[_tSteps + 1];
        }
    }

    public double this[int index]
    {
        get => TGrid[index];
        set => TGrid[index] = value;
    }

    public void BuildTimeGrid()
    {
        double sumRaz = 0;
        for (int i = 0; i < _tSteps; i++)
            sumRaz += Math.Pow(_tRaz, i);

        double t = _tStart;
        double tStep = (_tEnd - _tStart) / sumRaz;

        for (int i = 0; i < _tSteps; i++)
        {
            TGrid[i] = t;
            t += tStep;
            tStep *= _tRaz;
        }

        TGrid[_tSteps] = _tEnd;
    }
}
```

Basis.cs

```
namespace VectorFEM3D;

public interface IBasis3D
{
    int Size { get; }
    Vector3D GetPsi(int number, Point3D point);
```

```
    Vector3D GetDPsi(int number, Point3D point);
}

public readonly record struct TriLinearVectorBasis : IBasis3D
{
    public int Size => 12;
    private readonly Vector3D _vector = new Vector3D(0, 0, 0);

    public TriLinearVectorBasis() { }

    public Vector3D GetPsi(int number, Point3D point)
        => number switch
        {
            0 => _vector.UpdateVector(GetXi(0, point.Y) * GetXi(0, point.Z), 0, 0),
            1 => _vector.UpdateVector(GetXi(1, point.Y) * GetXi(0, point.Z), 0, 0),
            2 => _vector.UpdateVector(0, GetXi(0, point.X) * GetXi(0, point.Z), 0),
            3 => _vector.UpdateVector(0, GetXi(1, point.X) * GetXi(0, point.Z), 0),
            4 => _vector.UpdateVector(0, 0, GetXi(0, point.X) * GetXi(0, point.Y)),
            5 => _vector.UpdateVector(0, 0, GetXi(1, point.X) * GetXi(0, point.Y)),
            6 => _vector.UpdateVector(0, 0, GetXi(0, point.X) * GetXi(1, point.Y)),
            7 => _vector.UpdateVector(0, 0, GetXi(1, point.X) * GetXi(1, point.Y)),
            8 => _vector.UpdateVector(GetXi(0, point.Y) * GetXi(1, point.Z), 0, 0),
            9 => _vector.UpdateVector(GetXi(1, point.Y) * GetXi(1, point.Z), 0, 0),
            10 => _vector.UpdateVector(0, GetXi(0, point.X) * GetXi(1, point.Z), 0),
            11 => _vector.UpdateVector(0, GetXi(1, point.X) * GetXi(1, point.Z), 0),
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not
expected function number")
        };

    public Vector3D GetDPsi(int number, Point3D point)
        => number switch
        {
            0 => _vector.UpdateVector(0, -GetXi(0, point.Y), GetXi(0, point.Z)),
            1 => _vector.UpdateVector(0, -GetXi(1, point.Y), -GetXi(0, point.Z)),
            2 => _vector.UpdateVector(GetXi(0, point.X), 0, -GetXi(0, point.Z)),
            3 => _vector.UpdateVector(GetXi(1, point.X), 0, GetXi(0, point.Z)),
            4 => _vector.UpdateVector(-GetXi(0, point.X), GetXi(0, point.Y), 0),
            5 => _vector.UpdateVector(-GetXi(1, point.X), -GetXi(0, point.Y), 0),
            6 => _vector.UpdateVector(GetXi(0, point.X), GetXi(1, point.Y), 0),
            7 => _vector.UpdateVector(GetXi(1, point.X), -GetXi(1, point.Y), 0),
            8 => _vector.UpdateVector(0,  GetXi(0, point.Y), GetXi(1, point.Z)),
            9 => _vector.UpdateVector(0, GetXi(1, point.Y), -GetXi(1, point.Z)),
            10 => _vector.UpdateVector(-GetXi(0, point.X), 0, -GetXi(1, point.Z)),
            11 => _vector.UpdateVector(-GetXi(1, point.X), 0, GetXi(1, point.Z)),
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not
expected function number")
        };

    private double GetXi(int number, double value)
        => number switch
        {
            0 => 1 - value,
            1 => value,
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not
expected Xi member")
```

```
        };
}
```

Integration.cs

```
namespace VectorFEM3D;

public class Integration
{
    private readonly SegmentGaussOrder9 _quadratures;

    public Integration(SegmentGaussOrder9 quadratures)
    {
        _quadratures = quadratures;
    }

    public double Gauss3D(Func<Point3D, double> psi)
    {
        double result = 0;
        Point3D point = new(0, 0, 0);

        for (int i = 0; i < _quadratures.Size; i++)
        {
            point.X = (_quadratures.GetPoint(i) + 1) / 2.0;

            for (int j = 0; j < _quadratures.Size; j++)
            {
                point.Y = (_quadratures.GetPoint(j) + 1) / 2.0;

                for (int k = 0; k < _quadratures.Size; k++)
                {
                    point.Z = (_quadratures.GetPoint(k) + 1) / 2.0;

                    result += psi(point) * _quadratures.GetWeight(i) *
_quadratures.GetWeight(j) *
                              _quadratures.GetWeight(k);
                }
            }
        }

        return result / 8.0;
    }
}
```

QuadratureNode.cs

```
namespace VectorFEM3D;

public interface IQuadrature
{
    int Size { get; }
    double GetPoint(int number);
    double GetWeight(int number);
}

public readonly record struct SegmentGaussOrder9 : IQuadrature
{
    public int Size => 5;
```

```
    public SegmentGaussOrder9() { }

    public double GetPoint(int number)
        => number switch
        {
            0 => 0.0,
            1 => 1.0 / 3.0 * Math.Sqrt(5 - 2 * Math.Sqrt(10.0 / 7.0)),
            2 => -1.0 / 3.0 * Math.Sqrt(5 - 2 * Math.Sqrt(10.0 / 7.0)),
            3 => 1.0 / 3.0 * Math.Sqrt(5 + 2 * Math.Sqrt(10.0 / 7.0)),
            4 => -1.0 / 3.0 * Math.Sqrt(5 + 2 * Math.Sqrt(10.0 / 7.0)),
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not
expected point number")
        };

    public double GetWeight(int number)
        => number switch
        {
            0 => 128.0 / 225.0,
            1 => (322.0 + 13.0 * Math.Sqrt(70.0)) / 900.0,
            2 => (322.0 + 13.0 * Math.Sqrt(70.0)) / 900.0,
            3 => (322.0 - 13.0 * Math.Sqrt(70.0)) / 900.0,
            4 => (322.0 - 13.0 * Math.Sqrt(70.0)) / 900.0,
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not
expected weight number")
        };
}
```