



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

Курсовая работа  
по дисциплине «Уравнения математической физики»

**ПРИМЕНЕНИЕ МКЭ ДЛЯ РЕШЕНИЯ НЕСТАЦИОНАРНЫХ ЗАДАЧ**

Студент                      ШИШКИН НИКИТА

Группа ПМ-92

Преподаватель    РОЯК МИХАИЛ ЭММАНУИЛОВИЧ

Новосибирск, 2022

## Постановка задачи

МКЭ для двумерной краевой задачи для уравнения параболического типа в цилиндрической системе координат (1). Четырёхслойная неявная схема по времени. Базисные функции кубические лагранжевого типа на треугольниках.

$$\sigma \frac{\partial u}{\partial t} - \operatorname{div} (\lambda \operatorname{grad} u) = f \quad (1)$$

## Схема

Функция  $u$  в четырёхслойной схеме представляется следующим образом:

$$u(x, y, t) = u^{j-3}(x, y) \eta_3^j(t) + u^{j-2}(x, y) \eta_2^j(t) + u^{j-1}(x, y) \eta_1^j(t) + u^j(x, y) \eta_0^j(t) \quad (2)$$

В аппроксимации (2) функции пространственных координат  $u^{j-3}$ ,  $u^{j-2}$ ,  $u^{j-1}$  и  $u^j$  являются значениями искомой  $u$  при  $t = t_{j-3}$ ,  $t = t_{j-2}$ ,  $t = t_{j-1}$  и  $t = t_j$  соответственно. Зависящие только от времени функции  $\eta_\nu^j(t)$  являются кубическими полиномами  $t$ , причем  $\eta_3^j$  равна единице при  $t = t_{j-3}$  и нулю при  $t = t_{j-2}$ ,  $t = t_{j-1}$  и  $t = t_j$ , функция  $\eta_2^j$  равна единице при  $t = t_{j-2}$  и нулю при  $t = t_{j-3}$ ,  $t = t_{j-1}$  и  $t = t_j$ , функция  $\eta_1^j$  равна единице при  $t = t_{j-1}$  и нулю при  $t = t_{j-3}$ ,  $t = t_{j-2}$  и  $t = t_j$ , а функция  $\eta_0^j$  равна единице при  $t = t_j$  и нулю при  $t = t_{j-3}$ ,  $t = t_{j-2}$  и  $t = t_{j-1}$ . Таким образом, соотношение (2) определяет аппроксимацию функции  $u$  по времени как кубический интерполянт ее значений на временных слоях  $t = t_{j-3}$ ,  $t = t_{j-2}$ ,  $t = t_{j-1}$ ,  $t = t_j$ .

Функции  $\eta_\nu^j(t)$  – это базисные кубические полиномы Лагранжа, которые выглядят следующим образом:

$$\begin{aligned} \eta_3^j(t) &= \frac{(t - t_{j-2})(t - t_{j-1})(t - t_j)}{(t_{j-3} - t_{j-2})(t_{j-3} - t_{j-1})(t_{j-3} - t_j)} & \eta_2^j(t) &= \frac{(t - t_{j-3})(t - t_{j-1})(t - t_j)}{(t_{j-2} - t_{j-3})(t_{j-2} - t_{j-1})(t_{j-2} - t_j)} \\ \eta_1^j(t) &= \frac{(t - t_{j-3})(t - t_{j-2})(t - t_j)}{(t_{j-1} - t_{j-3})(t_{j-1} - t_{j-2})(t_{j-1} - t_j)} & \eta_0^j(t) &= \frac{(t - t_{j-3})(t - t_{j-2})(t - t_{j-1})}{(t_j - t_{j-3})(t_j - t_{j-2})(t_j - t_{j-1})} \end{aligned} \quad (3)$$

Упростим выражения для более простой записи производных с помощью соотношений:

$$\begin{aligned} \Delta t_{03} &= t_j - t_{j-3} & \Delta t_{02} &= t_j - t_{j-2} \\ \Delta t_{01} &= t_j - t_{j-1} & \Delta t_{13} &= t_{j-1} - t_{j-3} \\ \Delta t_{12} &= t_{j-1} - t_{j-2} & \Delta t_{23} &= t_{j-2} - t_{j-3} \end{aligned} \quad (4)$$

Получим первые производные по  $t$  при  $t = t_j$  и используем соотношения (4):

$$\begin{aligned} \left. \frac{\partial \eta_3^j(t)}{\partial t} \right|_{t=t_j} &= -\frac{\Delta t_{02} \Delta t_{01}}{\Delta t_{23} \Delta t_{13} \Delta t_{03}} \\ \left. \frac{\partial \eta_2^j(t)}{\partial t} \right|_{t=t_j} &= \frac{\Delta t_{03} \Delta t_{01}}{\Delta t_{23} \Delta t_{12} \Delta t_{02}} \\ \left. \frac{\partial \eta_1^j(t)}{\partial t} \right|_{t=t_j} &= -\frac{\Delta t_{03} \Delta t_{02}}{\Delta t_{13} \Delta t_{12} \Delta t_{01}} \\ \left. \frac{\partial \eta_0^j(t)}{\partial t} \right|_{t=t_j} &= \frac{1}{\Delta t_{03}} + \frac{1}{\Delta t_{02}} + \frac{1}{\Delta t_{01}} \end{aligned} \quad (5)$$

Применим представление (2) для аппроксимации производной по времени параболического уравнения (1) на временном слое  $t = t_j$ :

$$\left. \frac{\partial}{\partial t} (u^{j-3}(x, y) \eta_3^j(t) + u^{j-2}(x, y) \eta_2^j(t) + u^{j-1}(x, y) \eta_1^j(t) + u^j(x, y) \eta_0^j(t)) \right|_{t=t_j} - \operatorname{div} (\lambda \operatorname{grad} u^j) = f^j \quad (6)$$

С учетом (5) уравнение (6) может переписано в виде:

$$-\frac{\Delta t_{02} \Delta t_{01}}{\Delta t_{23} \Delta t_{13} \Delta t_{03}} u^{j-3} + \frac{\Delta t_{03} \Delta t_{01}}{\Delta t_{23} \Delta t_{12} \Delta t_{02}} u^{j-2} - \frac{\Delta t_{03} \Delta t_{02}}{\Delta t_{13} \Delta t_{12} \Delta t_{01}} u^{j-1} + \left( \frac{1}{\Delta t_{03}} + \frac{1}{\Delta t_{02}} + \frac{1}{\Delta t_{01}} \right) u^j - \operatorname{div} (\lambda \operatorname{grad} u^j) = f^j \quad (7)$$

Выполняя конечноэлементную аппроксимацию краевой задачи для уравнения (7), получим СЛАУ следующего вида:

$$\begin{aligned} & \left( \left( \frac{1}{\Delta t_{03}} + \frac{1}{\Delta t_{02}} + \frac{1}{\Delta t_{01}} \right) \mathbf{M} + \mathbf{G} \right) q^j = \\ & = b^j + \frac{\Delta t_{02} \Delta t_{01}}{\Delta t_{23} \Delta t_{13} \Delta t_{03}} \mathbf{M} q^{j-3} - \frac{\Delta t_{03} \Delta t_{01}}{\Delta t_{23} \Delta t_{12} \Delta t_{02}} \mathbf{M} q^{j-2} + \frac{\Delta t_{03} \Delta t_{02}}{\Delta t_{13} \Delta t_{12} \Delta t_{01}} \mathbf{M} q^{j-1} \end{aligned} \quad (8)$$

где  $\mathbf{M}$  – матрица массы,  $\mathbf{G}$  – матрица жесткости,  $b^j$  – вектор правой части, построенный по значениям функции  $f$  на текущем ( $j$  - м) временном слое,  $q^{j-3}$ ,  $q^{j-2}$  и  $q^{j-1}$  – решение на трех предыдущих слоях по времени,  $q^j$  – искомое решение на текущем слое по времени.

# Тестирование программы

## Первый тест

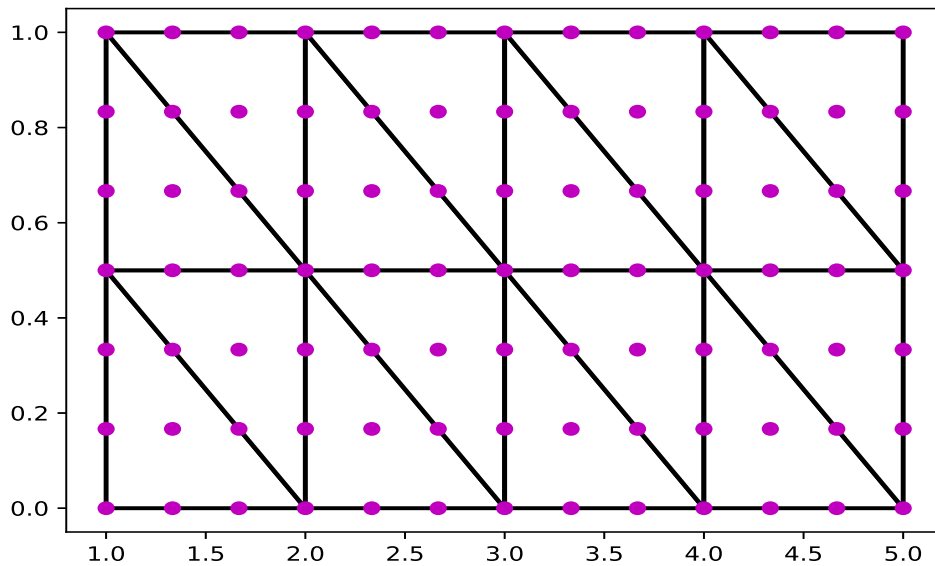
Функция:  $r^2 + z + t$

Правая часть:  $-3$

Коэффициенты :  $\lambda = 1, \sigma = 1$

Сетка по времени: равномерная  $[0,1], h_t = 0.1$

Заданы краевые условия 1-го рода.



$t_i$	Погрешность
0.1	$9.02 \cdot 10^{-14}$
0.2	$5.52 \cdot 10^{-14}$
0.3	$2.62 \cdot 10^{-14}$
0.4	$1.75 \cdot 10^{-14}$
0.5	$1.47 \cdot 10^{-14}$
0.6	$1.58 \cdot 10^{-14}$
0.7	$1.82 \cdot 10^{-14}$
0.8	$2.03 \cdot 10^{-14}$
0.9	$1.99 \cdot 10^{-14}$
1	$1.93 \cdot 10^{-14}$

## Второй тест

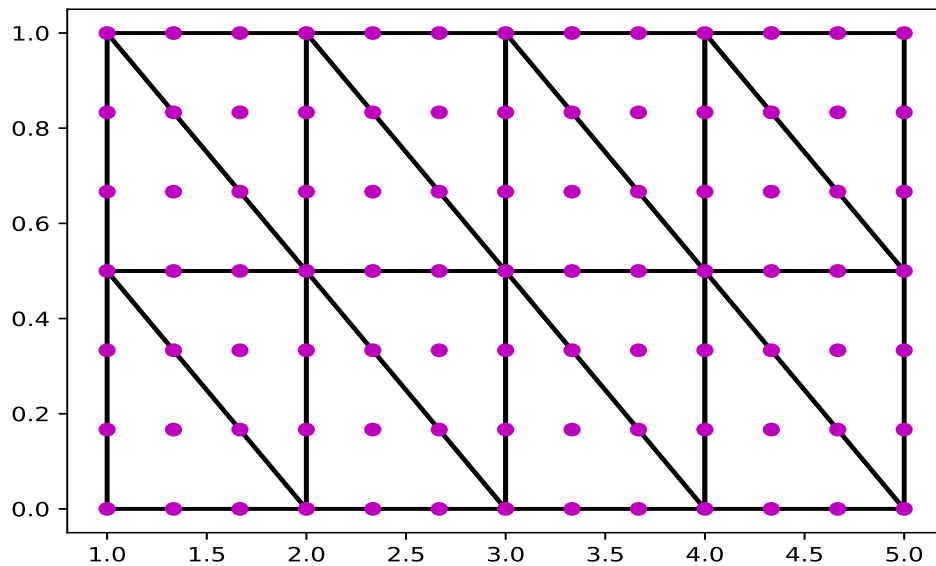
Функция:  $r^2 - zt^2$

Правая часть:  $-8 - zt$

Коэффициенты:  $\lambda = 2, \sigma = 0.5$

Сетка по времени: равномерная  $[0,1], h_t = 0.1$

Заданы краевые условия 1-го рода.



$t_i$	Погрешность
0.1	$2.19 \cdot 10^{-3}$
0.2	$7.49 \cdot 10^{-4}$
0.3	$1.62 \cdot 10^{-4}$
0.4	$1.46 \cdot 10^{-4}$
0.5	$9.75 \cdot 10^{-6}$
0.6	$3.14 \cdot 10^{-5}$
0.7	$5.76 \cdot 10^{-6}$
0.8	$4.56 \cdot 10^{-6}$
0.9	$1.96 \cdot 10^{-6}$
1	$4.79 \cdot 10^{-7}$

Возьмем сетку с меньшим шагом ( $h_t = 0.01$ ): погрешность =  $1.33E - 13$ .

### Третий тест

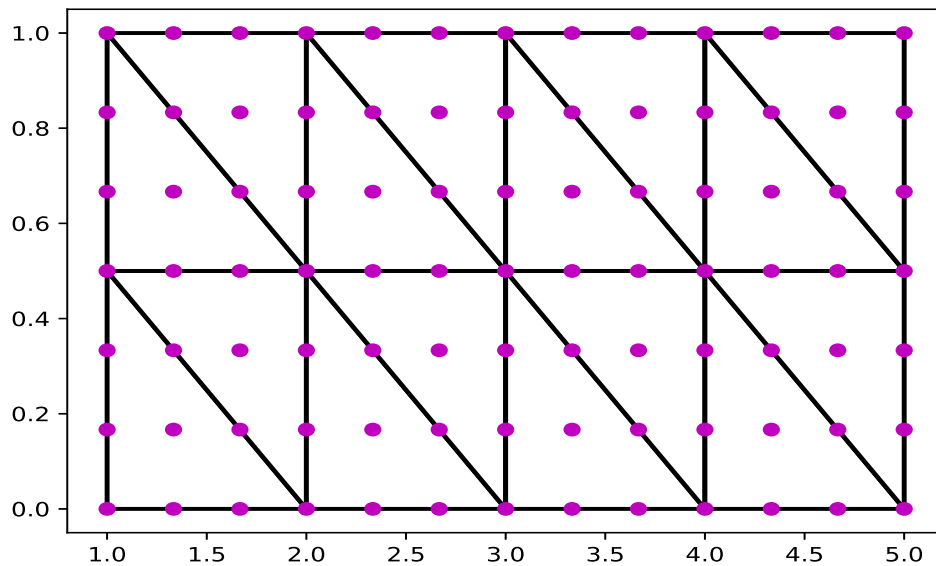
Функция:  $2r^3t^3$

Правая часть:  $-9rt^3 + 12r^3t^2$

Коэффициенты :  $\lambda = 0.5, \sigma = 2$

Сетка по времени: равномерная  $[0,1], h_t = 0.01$

Заданы краевые условия 1-го рода на левой и правой границе.



$t_i$	Погрешность
$1 \cdot 10^{-2}$	$1.05 \cdot 10^{-3}$
$2 \cdot 10^{-2}$	$1.51 \cdot 10^{-3}$
$3 \cdot 10^{-2}$	$1.6 \cdot 10^{-3}$
$4 \cdot 10^{-2}$	$1.56 \cdot 10^{-3}$
$5 \cdot 10^{-2}$	$1.51 \cdot 10^{-3}$
$6 \cdot 10^{-2}$	$1.47 \cdot 10^{-3}$
$7 \cdot 10^{-2}$	$1.45 \cdot 10^{-3}$
$8 \cdot 10^{-2}$	$1.43 \cdot 10^{-3}$
$9 \cdot 10^{-2}$	$1.42 \cdot 10^{-3}$
0.1	$1.41 \cdot 10^{-3}$
...	...
0.9	$9.99 \cdot 10^{-4}$
0.91	$9.96 \cdot 10^{-4}$
0.92	$9.94 \cdot 10^{-4}$
0.93	$9.91 \cdot 10^{-4}$
0.94	$9.88 \cdot 10^{-4}$
0.95	$9.86 \cdot 10^{-4}$
0.96	$9.83 \cdot 10^{-4}$
0.97	$9.81 \cdot 10^{-4}$
0.98	$9.78 \cdot 10^{-4}$
0.99	$9.76 \cdot 10^{-4}$
1	$9.73 \cdot 10^{-4}$

#### Четвертый тест

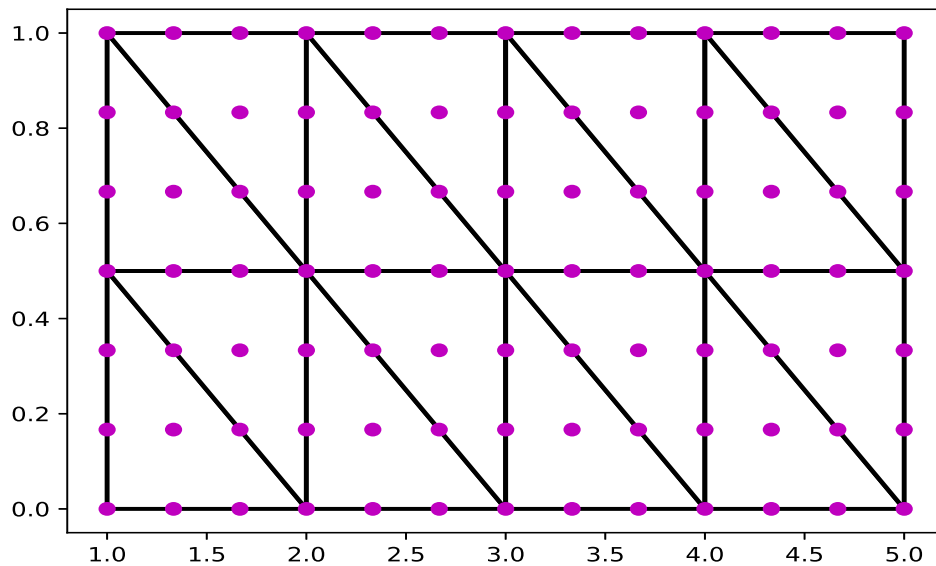
Функция:  $r^4 + z^4 + t^4$

Правая часть:  $-16r^2 - 12z^2 + 4t^3$

Коэффициенты :  $\lambda = 1, \sigma = 1$

Сетка по времени: равномерная  $[0,1], h_t = 0.001$

Заданы краевые условия 1-го рода.



$t_i$	Погрешность
0.001	$3.93 \cdot 10^{-4}$
0.002	$6.9 \cdot 10^{-4}$
0.003	$8.95 \cdot 10^{-4}$
0.004	$1.03 \cdot 10^{-3}$
0.005	$1.13 \cdot 10^{-3}$
0.006	$1.21 \cdot 10^{-3}$
0.007	$1.28 \cdot 10^{-3}$
0.008	$1.34 \cdot 10^{-3}$
0.009	$1.39 \cdot 10^{-3}$
0.010	$1.43 \cdot 10^{-3}$
...	...
0.990	$2.01 \cdot 10^{-3}$
0.991	$2.01 \cdot 10^{-3}$
0.992	$2.01 \cdot 10^{-3}$
0.993	$2.01 \cdot 10^{-3}$
0.994	$2.01 \cdot 10^{-3}$
0.995	$2.01 \cdot 10^{-3}$
0.996	$2.01 \cdot 10^{-3}$
0.997	$2.01 \cdot 10^{-3}$
0.998	$2.01 \cdot 10^{-3}$
0.999	$2.01 \cdot 10^{-3}$
1.000	$2.01 \cdot 10^{-3}$

## Исследование на определение порядка сходимости по пространству

Функция:  $\ln r$

Правая часть: 0

Коэффициенты:  $\lambda = 1, \sigma = 0$

Заданы краевые условия 1-го рода на левой и правой границе.

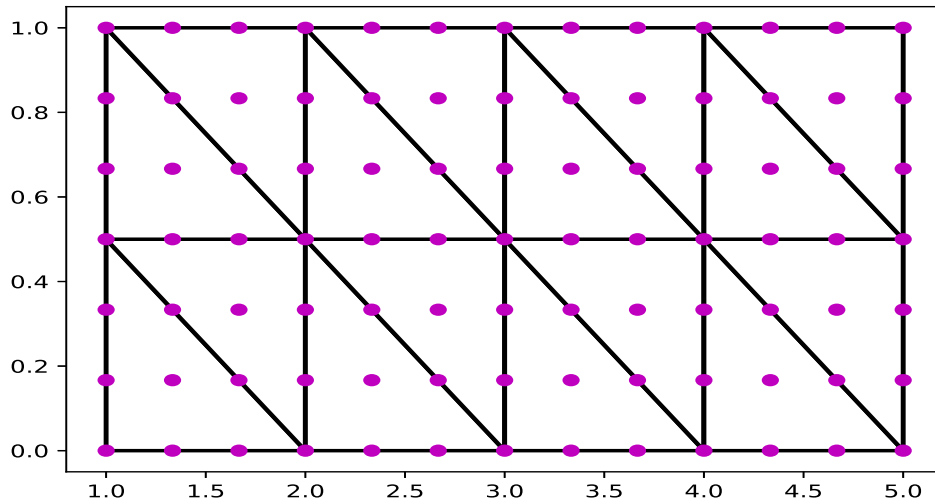


Рис. 1: Сетка с шагом  $h$

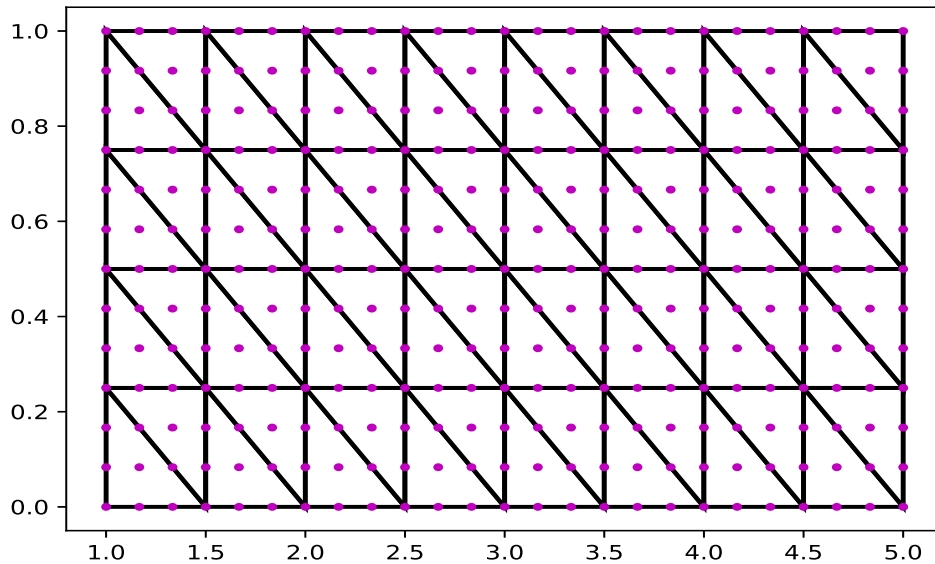


Рис. 2: Сетка с шагом  $h/2$



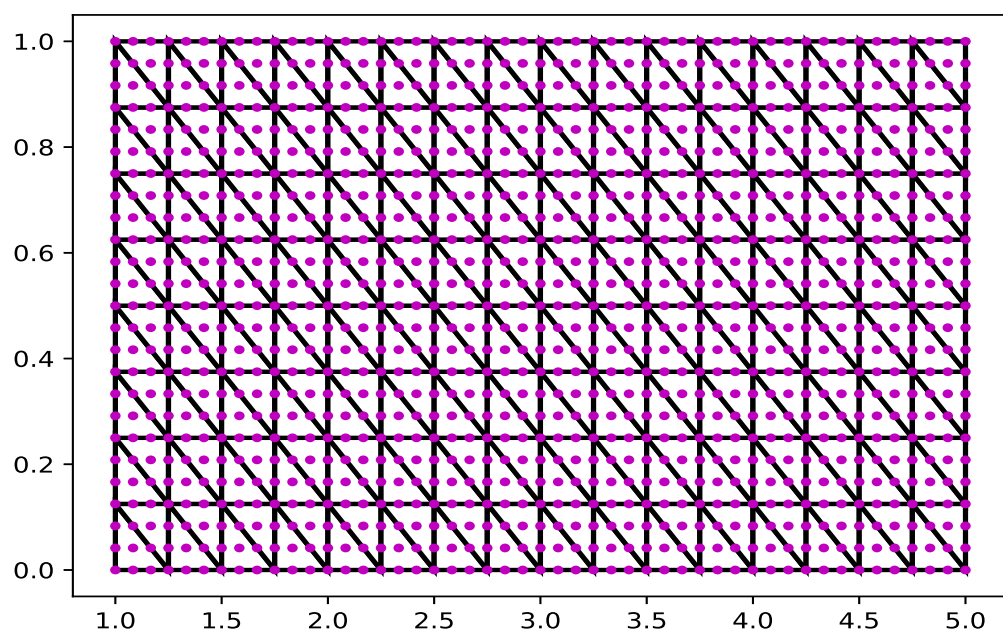


Рис. 3: Сетка с шагом  $h/4$

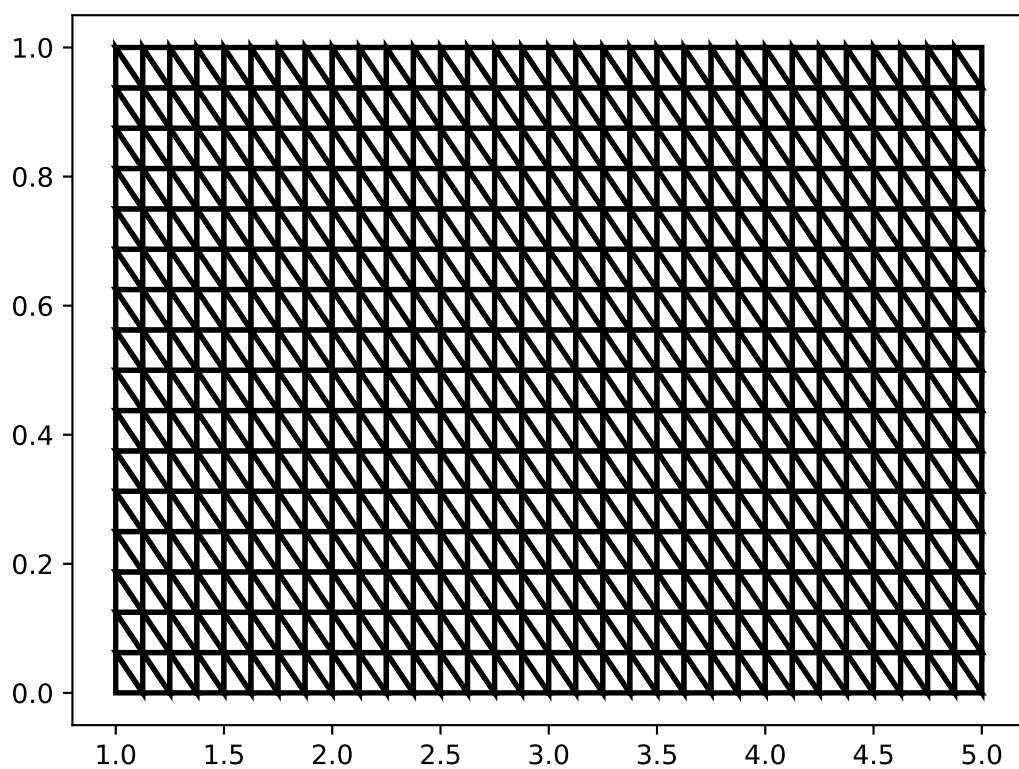


Рис. 4: Сетка с шагом  $h/8$

Сетка	Погрешность	Порядок сходимости
$h$	0.00010219151814123853	–
$h/2$	$9.3181384140705E - 06$	3.4551
$h/4$	$6.703524240893644E - 07$	3.7970
$h/8$	$4.3510098558766005E - 08$	3.9454

### Исследование на определение порядка сходимости схемы по времени

Функция:  $\cos t$

Правая часть:  $-\sin t$

Коэффициенты :  $\lambda = 1, \sigma = 1$

Сетка по времени: равномерная  $[0,1], h_t = 0.01$

Заданы краевые условия 1-го рода.

Сетка	Погрешность	Порядок сходимости
$h$	$1.1073473236551418E - 08$	–
$h/2$	$1.373830304097657E - 09$	3.01

Функция:  $t^4$

Правая часть:  $4t^3$

Коэффициенты :  $\lambda = 1, \sigma = 1$

Сетка по времени: равномерная  $[0,1], h_t = 0.1$

Заданы краевые условия 1-го рода.

Сетка	Погрешность	Порядок сходимости
$h$	0.0004249362540631516	–
$h/2$	$5.3306642166155186E - 05$	2.99

## Проведенные исследования и выводы

Теоретический порядок сходимости для кубического базиса равен 4, значит погрешность должна с дроблением падать в 16 раз. Из результатов исследования на определение порядка сходимости по пространству видно, что при дроблении сетки падение погрешности стремится к теоретическому значению.

Теоретический порядок сходимости для четырехслойной неявной схемы по времени равен 3. Из результатов исследования мы видим, что полученный результат совпал с теоретическим.

---

Сетка по пространству для исследования порядка сходимости схемы по времени взята с первого теста.

# Тексты основных модулей

## Program.cs

```
1 using courseProject;
2
3 SpaceGridFactory spaceGridFactory = new();
4 TimeGridFactory timeGridFactory = new();
5
6 // FEM fem = FEM.CreateBuilder().SetTest(new Test1())
7 // .SetSpaceGrid(spaceGridFactory.CreateGrid(GridTypes.SpaceRegular,
8 //   ↳ SpaceGridParameters.ReadJson("input/spaceGrid.jsonc")!.Value))
9 // .SetTimeGrid(timeGridFactory.CreateGrid(GridTypes.TimeRegular,
10 //   ↳ TimeGridParameters.ReadJson("input/timeGrid.json")!.Value))
11 // .SetSolverSLAE(new CGMCholesky(1000, 1E-14))
12 // .SetDirichletBoundaries(DirichletBoundary.ReadJson("input/DirichletBoundaries.json",
13 //   ↳ ")!))
14 // .SetNeumannBoundaries(NeumannBoundary.ReadJson("input/NeumannBoundaries.json")!)
15 // .IsPhysical(false);
16
17 FEM fem = FEM.CreateBuilder().SetTest(new Test7())
18 .SetSpaceGrid(spaceGridFactory.CreateGrid(GridTypes.SpaceRegular,
19   ↳ SpaceGridParameters.ReadJson("input/spaceGrid.jsonc")!.Value))
20 .SetTimeGrid(timeGridFactory.CreateGrid(GridTypes.TimeRegular,
21   ↳ TimeGridParameters.ReadJson("input/timeGrid.json")!.Value))
22 .SetSolverSLAE(new CGMCholesky(1000, 1E-14))
23 .SetDirichletBoundaries(DirichletBoundary.ReadJson("input/DirichletBoundaries.json")!)
24 .IsPhysical(false);
25
26 fem.Compute();
27 fem.WriteToFile("results/q.txt");
28
29 // var valuesAtPoints = fem.ValueAtPoint(new Point2D[] { new(1.5, 0.5), new(1, 0) });
30 // Array.ForEach(valuesAtPoints, Console.WriteLine);
```

## FEM.cs

```
1 namespace courseProject;
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(ISpaceGrid spaceGrid) {
13            _fem._spaceGrid = spaceGrid;
14            return this;
15        }
16
17        public FEMBuilder SetTimeGrid(ITimeGrid grid) {
18            _fem._timeGrid = grid;
19            return this;
20        }
21
22        public FEMBuilder SetSolverSLAE(Solver solver) {
23            _fem._solver = solver;
```

```

24         return this;
25     }
26
27     public FEMBuilder SetDirichletBoundaries(DirichletBoundary[] boundaries) {
28         _fem._dirichletBoundaries = boundaries;
29         return this;
30     }
31
32     public FEMBuilder SetNeumannBoundaries(NeumannBoundary[] boundaries) {
33         _fem._neumannBoundaries = boundaries;
34         return this;
35     }
36
37     public FEMBuilder IsPhysical(bool flag) {
38         _fem.IsPhysical = flag;
39         return this;
40     }
41
42     public static implicit operator FEM(FEMBuilder builder)
43         => builder._fem;
44 }
45
46 // default ~ cannot be null
47 private delegate double Basis(Point2D point);
48
49 private Basis[] _basis = default!;
50 private ISpaceGrid _spaceGrid = default!;
51 private ITimeGrid _timeGrid = default!;
52 private Point2D[] _vertices = default!;
53 private ITest _test = default!;
54 private Solver _solver = default!;
55 private DirichletBoundary[] _dirichletBoundaries = default!;
56 private NeumannBoundary[]? _neumannBoundaries;
57 private Matrix _stiffnessMatrix = default!;
58 private Matrix _massMatrix = default!;
59 private SparseMatrix _globalMatrix = default!;
60 private Vector<double> _vector = default!;
61 private static Matrix _alphas = default!; // коэффициенты альфа, костыль :(
62 private double[][][] _M = default!;
63 private double[][][] _G = default!;
64 private double[][] _layers = default!;
65 private Matrix _massMatrixCopy = default!;
66 public bool IsPhysical { get; private set; }
67
68 public void Compute() {
69     try {
70         ArgumentNullException.ThrowIfNull(_test, $"{nameof(_test)} cannot be
71         ↳ null, set the test");
72         ArgumentNullException.ThrowIfNull(_solver, $"{nameof(_solver)} cannot be
73         ↳ null, set the method of solving SLAE");
74         ArgumentNullException.ThrowIfNull(_dirichletBoundaries,
75         ↳ $"{nameof(_dirichletBoundaries)} cannot be null, set the Dirichlet
76         ↳ boundaries");
77
78         Init();
79         ConstructPortrait();
80         Prepare();
81         Solve();
82         // Err();
83     } catch (Exception ex) {

```

```

80         Console.WriteLine($"We had problem: {ex.Message}");
81     }
82 }
83
84 private void Init() {
85     _basis = new Basis[] { CubicBasis.Psi1, CubicBasis.Psi2, CubicBasis.Psi3,
86         ↪ CubicBasis.Psi4,
87         ↪ CubicBasis.Psi5, CubicBasis.Psi6, CubicBasis.Psi7, CubicBasis.Psi8,
88         ↪ CubicBasis.Psi9, CubicBasis.Psi10};
89
90     _stiffnessMatrix = new(10);
91     _massMatrix = new(10);
92     _massMatrixCopy = new(10);
93     _alphas = new(3);
94
95     _vertices = new Point2D[3];
96
97     _layers = new double[3].Select(_ => new
98         ↪ double[_spaceGrid.Points.Count]).ToArray();
99
100     _M = new double[10].Select(_ => new double[10].ToArray().
101         ↪ Select(_ => new double[3]).ToArray()).ToArray();
102
103     _G = new double[10].Select(_ => new double[10].ToArray().
104         ↪ Select(_ => new double[6].ToArray().
105         ↪ Select(_ => new double[3]).ToArray()).ToArray()).ToArray();
106
107     using StreamReader sr1 = new("input/Grz.txt"), sr2 = new("input/Mrz.txt");
108     string[] vars;
109
110     for (int i = 0; i < 600; i++) {
111         vars = sr1.ReadLine()!.Split(" ").ToArray();
112
113         _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][0] =
114             ↪ double.Parse(vars[3]);
115         _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][1] =
116             ↪ double.Parse(vars[4]);
117         _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][2] =
118             ↪ double.Parse(vars[5]);
119     }
120
121     for (int i = 0; i < 100; i++) {
122         vars = sr2.ReadLine()!.Split(" ").ToArray();
123
124         _M[int.Parse(vars[0])][int.Parse(vars[1])][0] = double.Parse(vars[2]);
125         _M[int.Parse(vars[0])][int.Parse(vars[1])][1] = double.Parse(vars[3]);
126         _M[int.Parse(vars[0])][int.Parse(vars[1])][2] = double.Parse(vars[4]);
127     }
128 }
129
130 private void Prepare() {
131     if (!IsPhysical) {
132         for (int i = 0; i < _spaceGrid.Points.Count; i++) {
133             _layers[0][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[0]);
134             _layers[1][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[1]);
135             _layers[2][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[2]);
136         }
137     } else {
138         for (int i = 0; i < _spaceGrid.Points.Count; i++) {
139             _layers[0][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[0]);

```

```

134     }
135
136     double t01 = _timeGrid.Points[1] - _timeGrid.Points[0];
137
138     AssemblyGlobalMatrix(1, t01: t01);
139
140     if (_neumannBoundaries is not null) {
141         AccountingNeumannBoundaries();
142     }
143
144     AccountingDirichletBoundaries(1);
145
146     _solver.SetMatrix(_globalMatrix);
147     _solver.SetVector(_vector);
148     _solver.Compute();
149     Err(1);
150
151     _solver.Solution!.Value.CopyTo(0, _layers[1], 0, _layers[1].Length);
152
153     _globalMatrix.Clear();
154     _vector.Fill(0);
155
156     double t02 = _timeGrid.Points[2] - _timeGrid.Points[0];
157     double t12 = _timeGrid.Points[1] - _timeGrid.Points[0];
158     t01 = _timeGrid.Points[2] - _timeGrid.Points[1];
159
160     AssemblyGlobalMatrix(2, t02: t02, t01: t01, t12: t12);
161
162     if (_neumannBoundaries is not null) {
163         AccountingNeumannBoundaries();
164     }
165
166     AccountingDirichletBoundaries(2);
167
168     _solver.SetMatrix(_globalMatrix);
169     _solver.SetVector(_vector);
170     _solver.Compute();
171     Err(2);
172
173     _solver.Solution!.Value.CopyTo(0, _layers[2], 0, _layers[2].Length);
174
175     _globalMatrix.Clear();
176     _vector.Fill(0);
177 }
178
179
180 private void Solve() {
181     for (int itime = 3; itime < _timeGrid.Points.Length; itime++) {
182         double t03 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 3];
183         double t02 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 2];
184         double t01 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 1];
185         double t13 = _timeGrid.Points[itime - 1] - _timeGrid.Points[itime - 3];
186         double t12 = _timeGrid.Points[itime - 1] - _timeGrid.Points[itime - 2];
187         double t23 = _timeGrid.Points[itime - 2] - _timeGrid.Points[itime - 3];
188
189         AssemblyGlobalMatrix(itime, t03, t02, t01, t13, t12, t23);
190
191         if (_neumannBoundaries is not null) {
192             AccountingNeumannBoundaries();
193         }

```

```

194     AccountingDirichletBoundaries(itime);
195     _globalMatrix.PrintDense("results/matrix.txt");
196
197     _solver.SetMatrix(_globalMatrix);
198     _solver.SetVector(_vector);
199     _solver.Compute();
200     Err(itime);
201
202     _layers[1].Copy(_layers[0]);
203     _layers[2].Copy(_layers[1]);
204     _solver.Solution!.Value.CopyTo(0, _layers[2], 0, _layers[2].Length);
205
206     _vector.Fill(0);
207     _globalMatrix.Clear();
208 }
209 }
210
211 private void ConstructPortrait() {
212     List<int>[] list = new List<int>[_spaceGrid.Points.Count].Select(_ => new
213     ↪ List<int>()).ToArray();
214
215     foreach (var element in _spaceGrid.Elements.Select(array =>
216     ↪ array.OrderBy(value => value).ToArray()).ToArray()) {
217         for (int i = 0; i < element.Length; i++) {
218             for (int j = i + 1; j < element.Length; j++) {
219                 int pos = element[j];
220                 int elem = element[i];
221
222                 if (!list[pos].Contains(elem)) {
223                     list[pos].Add(elem);
224                 }
225             }
226         }
227
228         list = list.Select(list => list.OrderBy(value => value).ToList()).ToArray();
229         int count = list.Sum(childList => childList.Count);
230
231         InitSLAE(count);
232
233         _globalMatrix.ig[0] = 0;
234
235         for (int i = 0; i < list.Length; i++)
236             _globalMatrix.ig[i + 1] = _globalMatrix.ig[i] + list[i].Count;
237
238         int k = 0;
239
240         for (int i = 0; i < list.Length; i++) {
241             for (int j = 0; j < list[i].Count; j++) {
242                 _globalMatrix.jg[k] = list[i][j];
243                 k++;
244             }
245         }
246     }
247
248     private void InitSLAE(int count) {
249         _globalMatrix = new(_spaceGrid.Points.Count, count);
250         _vector = new(_spaceGrid.Points.Count);
251     }

```

```

252 private void AddElement(int i, int j, double value) {
253     if (i == j) {
254         _globalMatrix.di[i] += value;
255         return;
256     }
257
258     if (i > j) {
259         for (int index = _globalMatrix.ig[i]; index < _globalMatrix.ig[i + 1];
260             ↪ index++) {
261             if (_globalMatrix.jg[index] == j) {
262                 _globalMatrix.gg[index] += value;
263                 return;
264             }
265         }
266     }
267 }
268
269 private void AssemblyGlobalMatrix(int itime = 1, double t03 = 1, double t02 = 1,
270     ↪ double t01 = 1, double t13 = 1, double t12 = 1, double t23 = 1) {
271     for (int ielem = 0; ielem < _spaceGrid.Elements.Length; ielem++) {
272         _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
273         _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
274         _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];
275
276         AssemblyLocalMatrices(itime, t03, t02, t01);
277
278         _stiffnessMatrix += _massMatrix;
279
280         for (int i = 0; i < 10; i++) {
281             for (int j = 0; j < 10; j++) {
282                 AddElement(_spaceGrid.Elements[ielem][i],
283                     ↪ _spaceGrid.Elements[ielem][j], _stiffnessMatrix[i, j]);
284             }
285         }
286
287         AssemblyGlobalVector(itime, ielem, t03, t02, t01, t13, t12, t23);
288
289         _stiffnessMatrix.Clear();
290         _massMatrix.Clear();
291     }
292 }
293
294 private void AssemblyGlobalVector(int itime = 1, int ielem = 1, double t03 = 1,
295     ↪ double t02 = 1, double t01 = 1, double t13 = 1, double t12 = 1, double t23 =
296     ↪ 1) {
297     double[] qj1 = new double[10];
298     double[] qj2 = new double[10];
299     double[] qj3 = new double[10];
300
301     for (int i = 0; i < 10; i++) {
302         for (int j = 0; j < 10; j++) {
303             if (itime == 1) {
304                 qj1[i] += _massMatrixCopy[i, j] *
305                     ↪ _layers[0][_spaceGrid.Elements[ielem][j]];
306             } else if (itime == 2) {
307                 qj2[i] += _massMatrixCopy[i, j] *
308                     ↪ _layers[0][_spaceGrid.Elements[ielem][j]];
309                 qj1[i] += _massMatrixCopy[i, j] *
310                     ↪ _layers[1][_spaceGrid.Elements[ielem][j]];
311             }
312         }
313     }

```



```

304         } else {
305             qj3[i] += _massMatrixCopy[i, j] *
306                 ↪ _layers[0][_spaceGrid.Elements[ielem][j]];
307             qj2[i] += _massMatrixCopy[i, j] *
308                 ↪ _layers[1][_spaceGrid.Elements[ielem][j]];
309             qj1[i] += _massMatrixCopy[i, j] *
310                 ↪ _layers[2][_spaceGrid.Elements[ielem][j]];
311         }
312     }
313 }
314
315 for (int j = 0; j < 10; j++) {
316     if (itime == 1) {
317         _vector[_spaceGrid.Elements[ielem][j]] +=
318             ↪ Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
319             ↪ _timeGrid.Points[2]) -
320                 (-1.0 / t01 * qj1[j]);
321     } else if (itime == 2) {
322         _vector[_spaceGrid.Elements[ielem][j]] +=
323             ↪ Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
324             ↪ _timeGrid.Points[2]) -
325                 (t01 / (t02 * t12) * qj2[j]) + (t02 / (t12
326                 ↪ * t01) * qj1[j]);
327     } else {
328         _vector[_spaceGrid.Elements[ielem][j]] +=
329             ↪ Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
330             ↪ _timeGrid.Points[itime]) +
331                 (t02 * t01 / (t23 * t13 *
332                 ↪ t03) * qj3[j]) -
333                 (t03 * t01 / (t23 * t12 *
334                 ↪ t02) * qj2[j]) +
335                 (t03 * t02 / (t13 * t12 *
336                 ↪ t01) * qj1[j]);
337     }
338 }
339
340 private double DeterminantD()
341 => ((_vertices[1].R - _vertices[0].R) * (_vertices[2].Z - _vertices[0].Z)) -
342     ((_vertices[2].R - _vertices[0].R) * (_vertices[1].Z - _vertices[0].Z));
343
344 private void CalcAlphas() {
345     double dD = DeterminantD();
346
347     _alphas[0, 0] = ((_vertices[1].R * _vertices[2].Z) - (_vertices[2].R *
348     ↪ _vertices[1].Z)) / dD;
349     _alphas[0, 1] = (_vertices[1].Z - _vertices[2].Z) / dD;
350     _alphas[0, 2] = (_vertices[2].R - _vertices[1].R) / dD;
351
352     _alphas[1, 0] = ((_vertices[2].R * _vertices[0].Z) - (_vertices[0].R *
353     ↪ _vertices[2].Z)) / dD;
354     _alphas[1, 1] = (_vertices[2].Z - _vertices[0].Z) / dD;
355     _alphas[1, 2] = (_vertices[0].R - _vertices[2].R) / dD;
356
357     _alphas[2, 0] = ((_vertices[0].R * _vertices[1].Z) - (_vertices[1].R *
358     ↪ _vertices[0].Z)) / dD;
359     _alphas[2, 1] = (_vertices[0].Z - _vertices[1].Z) / dD;
360     _alphas[2, 2] = (_vertices[1].R - _vertices[0].R) / dD;
361 }

```

```

348 private void AssemblyLocalMatrices(int itime, double t03, double t02, double t01)
349     {
350         double dD = Math.Abs(DeterminantD());
351         CalcAlphas();
352
353         //rs=[d11^2 d11d12 d11d13 d12^2 d12d13 d13d13]
354         //каждая из 6 пар дифференциалов состоит из 3 составляющих (т.к. вектор
355         //→ r=r1L1+r2L2+r3L3)
356         for (int i = 0; i < 10; i++) {
357             for (int j = 0; j < 10; j++) {
358                 _stiffnessMatrix[i, j] += _G[i][j][0][0] * _vertices[0].R *
359                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) +
360                 //→ (_alphas[0, 2] * _alphas[0, 2])) * dD;
361                 _stiffnessMatrix[i, j] += _G[i][j][0][1] * _vertices[1].R *
362                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) +
363                 //→ (_alphas[0, 2] * _alphas[0, 2])) * dD;
364                 _stiffnessMatrix[i, j] += _G[i][j][0][2] * _vertices[2].R *
365                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) +
366                 //→ (_alphas[0, 2] * _alphas[0, 2])) * dD;
367
368                 _stiffnessMatrix[i, j] += _G[i][j][1][0] * _vertices[0].R *
369                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) +
370                 //→ (_alphas[0, 2] * _alphas[1, 2])) * dD;
371                 _stiffnessMatrix[i, j] += _G[i][j][1][1] * _vertices[1].R *
372                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) +
373                 //→ (_alphas[0, 2] * _alphas[1, 2])) * dD;
374                 _stiffnessMatrix[i, j] += _G[i][j][1][2] * _vertices[2].R *
375                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) +
376                 //→ (_alphas[0, 2] * _alphas[1, 2])) * dD;
377
378                 _stiffnessMatrix[i, j] += _G[i][j][2][0] * _vertices[0].R *
379                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[2, 1]) +
380                 //→ (_alphas[0, 2] * _alphas[2, 2])) * dD;
381                 _stiffnessMatrix[i, j] += _G[i][j][2][1] * _vertices[1].R *
382                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[2, 1]) +
383                 //→ (_alphas[0, 2] * _alphas[2, 2])) * dD;
384                 _stiffnessMatrix[i, j] += _G[i][j][2][2] * _vertices[2].R *
385                 //→ _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[2, 1]) +
386                 //→ (_alphas[0, 2] * _alphas[2, 2])) * dD;
387
388                 _stiffnessMatrix[i, j] += _G[i][j][3][0] * _vertices[0].R *
389                 //→ _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) +
390                 //→ (_alphas[1, 2] * _alphas[1, 2])) * dD;
391                 _stiffnessMatrix[i, j] += _G[i][j][3][1] * _vertices[1].R *
392                 //→ _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) +
393                 //→ (_alphas[1, 2] * _alphas[1, 2])) * dD;
394                 _stiffnessMatrix[i, j] += _G[i][j][3][2] * _vertices[2].R *
395                 //→ _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) +
396                 //→ (_alphas[1, 2] * _alphas[1, 2])) * dD;
397
398                 _stiffnessMatrix[i, j] += _G[i][j][4][0] * _vertices[0].R *
399                 //→ _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) +
400                 //→ (_alphas[1, 2] * _alphas[2, 2])) * dD;
401                 _stiffnessMatrix[i, j] += _G[i][j][4][1] * _vertices[1].R *
402                 //→ _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) +
403                 //→ (_alphas[1, 2] * _alphas[2, 2])) * dD;
404                 _stiffnessMatrix[i, j] += _G[i][j][4][2] * _vertices[2].R *
405                 //→ _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) +
406                 //→ (_alphas[1, 2] * _alphas[2, 2])) * dD;

```

```

376         _stiffnessMatrix[i, j] += _G[i][j][5][0] * _vertices[0].R *
        ↪ _spaceGrid.Lambda * ((_alphas[2, 1] * _alphas[2, 1]) +
        ↪ (_alphas[2, 2] * _alphas[2, 2])) * dD;
377         _stiffnessMatrix[i, j] += _G[i][j][5][1] * _vertices[1].R *
        ↪ _spaceGrid.Lambda * ((_alphas[2, 1] * _alphas[2, 1]) +
        ↪ (_alphas[2, 2] * _alphas[2, 2])) * dD;
378         _stiffnessMatrix[i, j] += _G[i][j][5][2] * _vertices[2].R *
        ↪ _spaceGrid.Lambda * ((_alphas[2, 1] * _alphas[2, 1]) +
        ↪ (_alphas[2, 2] * _alphas[2, 2])) * dD;
379     }
380 }
381
382 for (int i = 0; i < 10; i++) {
383     for (int j = 0; j < 10; j++) {
384         for (int k = 0; k < 3; k++) {
385             _massMatrix[i, j] += _M[i][j][k] * _vertices[k].R *
            ↪ _spaceGrid.Sigma * dD;
386         }
387     }
388 }
389
390 _massMatrix.Copy(_massMatrixCopy);
391
392 if (itime == 1) {
393     for (int i = 0; i < 10; i++) {
394         for (int j = 0; j < 10; j++) {
395             _massMatrix[i, j] *= 1.0 / t01;
396         }
397     }
398 } else if (itime == 2) {
399     for (int i = 0; i < 10; i++) {
400         for (int j = 0; j < 10; j++) {
401             _massMatrix[i, j] *= (t02 + t01) / (t02 * t01);
402         }
403     }
404 } else {
405     for (int i = 0; i < 10; i++) {
406         for (int j = 0; j < 10; j++) {
407             _massMatrix[i, j] *= (1.0 / t03) + (1.0 / t02) + (1.0 / t01);
408         }
409     }
410 }
411 }
412
413 private void AccountingDirichletBoundaries(int itime) {
414     (int Node, double Value)[] boundaries = new (int, double)[4 *
        ↪ _dirichletBoundaries.Length];
415     int[] checkBC = new int[_spaceGrid.Points.Count];
416
417     int index = 0;
418
419     for (int i = 0; i < _dirichletBoundaries.Length; i++) {
420         int ielem = _dirichletBoundaries[i].Element;
421         int edge = _dirichletBoundaries[i].Edge;
422
423         for (int j = 0; j < 4; j++) {
424             boundaries[index++] = (_spaceGrid.Edges[ielem][edge][j],
            ↪ _test.U(_spaceGrid.Points[_spaceGrid.Edges[ielem]
            ↪ m][edge][j])),
            ↪ _timeGrid.Points[itime]));
425         }
426     }
427 }

```

```

426     }
427 }
428
429 boundaries = boundaries.Distinct().OrderBy(boundary =>
    ↳ boundary.Node).ToArray();
430 checkBC.Fill(-1);
431
432 for (int i = 0; i < boundaries.Length; i++)
433     checkBC[boundaries[i].Node] = i;
434
435 for (int i = 0; i < _spaceGrid.Points.Count; i++) {
436     if (checkBC[i] != -1) {
437         _globalMatrix.di[i] = 1;
438         _vector[i] = boundaries[checkBC[i]].Value;
439
440         for (int k = _globalMatrix.ig[i]; k < _globalMatrix.ig[i + 1]; k++) {
441             index = _globalMatrix.jg[k];
442
443             if (checkBC[index] == -1)
444                 _vector[index] -= _globalMatrix.gg[k] * _vector[i];
445
446             _globalMatrix.gg[k] = 0;
447         }
448     } else {
449         for (int k = _globalMatrix.ig[i]; k < _globalMatrix.ig[i + 1]; k++) {
450             index = _globalMatrix.jg[k];
451
452             if (checkBC[index] != -1) {
453                 _vector[i] -= _globalMatrix.gg[k] * _vector[index];
454                 _globalMatrix.gg[k] = 0;
455             }
456         }
457     }
458 }
459 }
460
461 private void AccountingNeumannBoundaries() {
462     Vector<double> localVector = new(10);
463     int[] localEdge = new int[4];
464
465     for (int i = 0; i < _neumannBoundaries!.Length; i++) {
466         int ielem = _neumannBoundaries[i].Element;
467
468         _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
469         _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
470         _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];
471
472         CalcAlphas();
473
474         // second else if you want to enter a single element
475
476         if (_neumannBoundaries[i].Edge == 0) {
477             localEdge[0] = 0;
478             localEdge[1] = 8;
479             localEdge[2] = 7;
480             localEdge[3] = 2;
481         } else if (_neumannBoundaries[i].Edge == 1) {
482             localEdge[0] = 0;
483             localEdge[1] = 3;
484             localEdge[2] = 4;

```

```

485         localEdge[3] = 1;
486     } else {
487         localEdge[0] = 1;
488         localEdge[1] = 5;
489         localEdge[2] = 6;
490         localEdge[3] = 2;
491     }
492
493     for (int j = 0; j < 4; j++) {
494         localVector[localEdge[j]] = _spaceGrid.Lambda *
495             ↪ _neumannBoundaries[i].Value *
496                 Integration.GaussSegment(_basis[localEdge[j]].Invok
497                     ↪ e,
498                         _spaceGrid.Points[_spaceGrid.Edges[ielem][_neumannB
499                             ↪ oundaries[i].Edge][0]],
500                         _spaceGrid.Points[_spaceGrid.Edges[ielem][_neumannB
501                             ↪ oundaries[i].Edge][3]]);
502     }
503
504     for (int j = 0; j < 10; j++) {
505         _vector[_spaceGrid.Elements[ielem][j]] += localVector[j];
506     }
507
508     localVector.Fill(0);
509 }
510
511 public double[] ValueAtPoint(Point2D[] points) {
512     try {
513         ArgumentNullException.ThrowIfNull(_solver.Solution,
514             ↪ $"{nameof(_solver.Solution)} cannot be null, solve problem first");
515
516         double[] result = new double[points.Length];
517
518         for (int ipoint = 0; ipoint < points.Length; ipoint++) {
519             int ielem = FindElement(points[ipoint]);
520             double determinant = DeterminantD();
521             CalcAlphas();
522
523             for (int i = 0; i < 10; i++) {
524                 result[ipoint] +=
525                     ↪ _solver.Solution!.Value[_spaceGrid.Elements[ielem][i]] *
526                         _basis[i](points[ipoint]);
527             }
528         }
529
530         return result;
531     } catch (Exception ex) {
532         Console.WriteLine($"{We had problem: {ex.Message}");
533         return Array.Empty<double>();
534     }
535 }
536
537 private int FindElement(Point2D point) {
538     for (int ielem = 0; ielem < _spaceGrid.Elements.Length; ielem++) {
539         _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
540         _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
541         _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];
542
543         double a = ((_vertices[0].R - point.R) * (_vertices[1].Z -
544             ↪ _vertices[0].Z)) -

```

```

539         ((_vertices[1].R - _vertices[0].R) * (_vertices[0].Z - point.Z));
540
541         double b = ((_vertices[1].R - point.R) * (_vertices[2].Z -
542             ↪ _vertices[1].Z)) -
543             (_vertices[2].R - (_vertices[1].R * (_vertices[1].Z - point.Z)));
544
545         double c = ((_vertices[2].R - point.R) * (_vertices[0].Z -
546             ↪ _vertices[2].Z)) -
547             ((_vertices[0].R - _vertices[2].R) * (_vertices[2].Z - point.Z));
548
549         if ((a >= 0 && b >= 0 && c >= 0) || (a <= 0 && b <= 0 && c <= 0))
550             return ielem;
551     }
552
553     throw new Exception("The point does not belong to the grid");
554 }
555
556 public void WriteToFile(string path) {
557     using var sw = new StreamWriter(path);
558     for (int i = 0; i < _solver.Solution!.Value.Length; i++) {
559         sw.WriteLine(i + " " + _solver.Solution!.Value[i]);
560     }
561 }
562
563 // for report
564 private void Err(int itime) {
565     double[] err = new double[_spaceGrid.Points.Count];
566     double[] exact = new double[_spaceGrid.Points.Count];
567     double sum = 0.0;
568
569     using StreamWriter sw1 = new("results/err.txt"),
570         sw2 = new("results/Exact.txt"),
571         sw3 = new("results/rms.txt"),
572         sw4 = new("csv2/2.csv", true);
573
574     for (int i = 0; i < _spaceGrid.Points.Count; i++) {
575         err[i] = Math.Abs(_solver.Solution!.Value[i] -
576             ↪ _test.U(_spaceGrid.Points[i], _timeGrid.Points[itime]));
577         sw1.WriteLine(err[i]);
578     }
579
580     for (int i = 0; i < _spaceGrid.Points.Count; i++) {
581         exact[i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[itime]);
582         sw2.WriteLine(exact[i]);
583     }
584
585     for (int i = 0; i < _spaceGrid.Points.Count; i++) {
586         sum += err[i] * err[i];
587     }
588
589     // sum = Math.Sqrt(sum); // относительная погрешность
590     // sum /= exact.Norm();
591
592     sum = Math.Sqrt(sum / _spaceGrid.Points.Count); // среднеквадратичное
593     ↪ ОТКЛОНЕНИЕ
594     sw3.Write(sum);
595
596     // sw4.WriteLine($"{i} $z, Точное, Численное, Вектор погрешности, Погрешность");
597
598     // for (int i = 0; i < _spaceGrid.Points.Count; i++) {

```

```

595 //      if (i == 0) {
596 //          sw4.WriteLine($"{_spaceGrid.Points[i]}, {exact[i]}, {_solver.Solutio
    ↪ n!.Value[i]}, {err[i].ToString(err[i] == 0 ? "0" :
    ↪ "0.00E+0")}, {sum:0.00E+0}");
597 //      }
598
599 //      sw4.WriteLine($"{_spaceGrid.Points[i]}, {exact[i]}, {_solver.Solution!.V
    ↪ alue[i]}, {err[i].ToString(err[i] == 0 ? "0" :
    ↪ "0.00E+0")}, ");
600 // }
601
602 if (itime == 1) {
603     sw4.WriteLine($"t_{i}$, Погрешность");
604 }
605
606 sw4.WriteLine($"{_timeGrid.Points[itime]:0.0000}, {sum:0.00E+0}");
607 }
608
609 public static FEMBuilder CreateBuilder()
610     => new();
611
612 // КОСТЫЛЬ
613 public static Matrix GetAlphas()
614     => _alphas;
615 }

```

## SparseMatrix.cs

```

1 namespace courseProject;
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[] gg = default!;
9     public int Size { get; init; }
10
11     public SparseMatrix(int size, int sizeOffDiag) {
12         Size = size;
13         ig = new int[size + 1];
14         jg = new int[sizeOffDiag];
15         gg = new double[sizeOffDiag];
16         di = new double[size];
17     }
18
19     public static Vector<double> operator *(SparseMatrix matrix, Vector<double>
    ↪ vector) {
20         Vector<double> product = new(vector.Length);
21
22         for (int i = 0; i < vector.Length; i++) {
23             product[i] = matrix.di[i] * vector[i];
24
25             for (int j = matrix.ig[i]; j < matrix.ig[i + 1]; j++) {
26                 product[i] += matrix.gg[j] * vector[matrix.jg[j]];
27                 product[matrix.jg[j]] += matrix.gg[j] * vector[i];
28             }
29         }
30
31         return product;

```

```

32     }
33
34     public void PrintDense(string path) {
35         double[,] A = new double[Size, Size];
36
37         for (int i = 0; i < Size; i++) {
38             A[i, i] = di[i];
39
40             for (int j = ig[i]; j < ig[i + 1]; j++) {
41                 A[i, jg[j]] = gg[j];
42                 A[jg[j], i] = gg[j];
43             }
44         }
45
46         using var sw = new StreamWriter(path);
47         for (int i = 0; i < Size; i++) {
48             for (int j = 0; j < Size; j++) {
49                 sw.Write(A[i, j].ToString("0.00") + "\t");
50             }
51
52             sw.WriteLine();
53         }
54     }
55
56     public void Clear() {
57         for (int i = 0; i < Size; i++) {
58             di[i] = 0.0;
59
60             for (int k = ig[i]; k < ig[i + 1]; k++) {
61                 gg[k] = 0.0;
62             }
63         }
64     }
65 }

```

## Matrix.cs

```

1  namespace courseProject;
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 void Copy(Matrix destination) {
21         for (int i = 0; i < destination.Size; i++) {
22             for (int j = 0; j < destination.Size; j++) {
23                 destination[i, j] = storage[i, j];

```



```

24     }
25     }
26 }
27
28 public static Matrix operator +(Matrix fstMatrix, Matrix sndMatrix) {
29     Matrix resultMatrix = new(fstMatrix.Size);
30
31     for (int i = 0; i < resultMatrix.Size; i++) {
32         for (int j = 0; j < resultMatrix.Size; j++) {
33             resultMatrix[i, j] = fstMatrix[i, j] + sndMatrix[i, j];
34         }
35     }
36
37     return resultMatrix;
38 }
39 }

```

## Vector.cs

```

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

```

```

18     => _vector = vector;
19
20     public abstract void Compute();
21 }
22
23 public class LOS : Solver {
24     public LOS(int maxIters, double eps) : base(maxIters, eps) { }
25
26     public override void Compute() {
27         try {
28             ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
                ↳ null, set the matrix");
29             ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
                ↳ null, set the vector");
30
31             double alpha, beta;
32             double squareNorm;
33
34             _solution = new(_vector.Length);
35
36             Vector<double> r = new(_vector.Length);
37             Vector<double> z = new(_vector.Length);
38             Vector<double> p = new(_vector.Length);
39             Vector<double> tmp = new(_vector.Length);
40
41             r = _vector - (_matrix * _solution);
42
43             Vector<double>.Copy(r, z);
44
45             p = _matrix * z;
46
47             squareNorm = r * r;
48
49             for (int iter = 0; iter < MaxIters && squareNorm > Eps; iter++) {
50                 alpha = p * r / (p * p);
51                 _solution += alpha * z;
52                 squareNorm = (r * r) - (alpha * alpha * (p * p));
53                 r -= alpha * p;
54
55                 tmp = _matrix * r;
56
57                 beta = -(p * tmp) / (p * p);
58                 z = r + (beta * z);
59                 p = tmp + (beta * p);
60             }
61         } catch (Exception ex) {
62             Console.WriteLine($"We had problem: {ex.Message}");
63         }
64     }
65 }
66
67 public class CGM : Solver {
68     public CGM(int maxIters, double eps) : base(maxIters, eps) { }
69
70     public override void Compute() {
71         try {
72             ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
                ↳ null, set the matrix");
73             ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
                ↳ null, set the vector");

```

```

74         double alpha, beta;
75         double norm, squareNorm;
76         double vectorNorm = _vector.Norm();
77
78         _solution = new(_vector.Length);
79
80         Vector<double> r = new(_vector.Length);
81         Vector<double> z = new(_vector.Length);
82         Vector<double> p = new(_vector.Length);
83         Vector<double> tmp = new(_vector.Length);
84
85         r = _vector - (_matrix * _solution);
86
87         Vector<double>.Copy(r, z);
88
89         for (int iter = 0; iter < MaxIters && (norm = r.Norm() / vectorNorm) >=
90             ↪ Eps; iter++) {
91             tmp = _matrix * z;
92             alpha = r * r / (tmp * z);
93             _solution += alpha * z;
94             squareNorm = r * r;
95             r -= alpha * tmp;
96             beta = r * r / squareNorm;
97             z = r + (beta * z);
98         }
99     } catch (Exception ex) {
100         Console.WriteLine($"We had problem: {ex.Message}");
101     }
102 }
103 }
104
105 public class CGMCholesky : Solver {
106     public CGMCholesky(int maxIters, double eps) : base(maxIters, eps) { }
107
108     public override void Compute() {
109         try {
110             double alpha, beta;
111             double tmp;
112
113             double vectorNorm = _vector.Norm();
114
115             _solution = new(_vector.Length);
116
117             double[] ggnew = new double[_matrix.gg.Length];
118             double[] dinew = new double[_matrix.di.Length];
119
120             _matrix.gg.Copy(ggnew);
121             _matrix.di.Copy(dinew);
122
123             Vector<double> r = new(_vector.Length);
124             Vector<double> z = new(_vector.Length);
125             Vector<double> fstTemp = new(_vector.Length);
126             Vector<double> sndTemp = new(_vector.Length);
127
128             Cholesky(ggnew, dinew);
129
130             r = _vector - (_matrix * _solution);
131             z = MoveForCholesky(r, ggnew, dinew);
132

```

```

133     for (int iter = 0; iter < MaxIters && r.Norm() / vectorNorm >= Eps;
134           ↪ iter++) {
135         tmp = MoveForCholesky(r, ggnew, dinew) * r;
136         sndTemp = _matrix * z;
137         alpha = tmp / (sndTemp * z);
138         _solution += alpha * z;
139         r -= alpha * sndTemp;
140         fstTemp = MoveForCholesky(r, ggnew, dinew);
141         beta = fstTemp * r / tmp;
142         z = fstTemp + (beta * z);
143     }
144     } catch (Exception ex) {
145         Console.WriteLine($"We had problem: {ex.Message}");
146     }
147 }
148
149 private void Cholesky(double[] ggnew, double[] dinew) {
150     double suml = 0.0;
151     double sumdi = 0.0;
152
153     for (int i = 0; i < _matrix.Size; i++) {
154         int i0 = _matrix.ig[i];
155         int i1 = _matrix.ig[i + 1];
156
157         for (int k = i0; k < i1; k++) {
158             int j = _matrix.jg[k];
159             int j0 = _matrix.ig[j];
160             int j1 = _matrix.ig[j + 1];
161             int ik = i0;
162             int kj = j0;
163
164             while (ik < k && kj < j1) {
165                 if (_matrix.jg[ik] == _matrix.jg[kj]) {
166                     suml += ggnew[ik] * ggnew[kj];
167                     ik++;
168                     kj++;
169                 } else {
170                     if (_matrix.jg[ik] > _matrix.jg[kj])
171                         kj++;
172                     else
173                         ik++;
174                 }
175             }
176
177             ggnew[k] = (ggnew[k] - suml) / dinew[j];
178             sumdi += ggnew[k] * ggnew[k];
179             suml = 0.0;
180         }
181
182         dinew[i] = Math.Sqrt(dinew[i] - sumdi);
183         sumdi = 0.0;
184     }
185 }
186
187 private Vector<double> MoveForCholesky(Vector<double> vector, double[] ggnew,
188 ↪ double[] dinew) {
189     Vector<double> y = new(vector.Length);
190     Vector<double> x = new(vector.Length);
191     Vector<double>.Copy(vector, y);

```

```

191     double sum = 0.0;
192
193     for (int i = 0; i < _matrix.Size; i++) // Прямой ход
194     {
195         int i0 = _matrix.ig[i];
196         int i1 = _matrix.ig[i + 1];
197
198         for (int k = i0; k < i1; k++)
199             sum += ggnew[k] * y[_matrix.jg[k]];
200
201         y[i] = (y[i] - sum) / dinew[i];
202         sum = 0.0;
203     }
204
205     Vector<double>.Copy(y, x);
206
207     for (int i = _matrix.Size - 1; i >= 0; i--) // Обратный ход
208     {
209         int i0 = _matrix.ig[i];
210         int i1 = _matrix.ig[i + 1];
211         x[i] = y[i] / dinew[i];
212
213         for (int k = i0; k < i1; k++)
214             y[_matrix.jg[k]] -= ggnew[k] * x[i];
215     }
216
217     return x;
218 }
219 }

```

## Integration.cs

```

1 namespace courseProject;
2
3 public static class Integration {
4     public static double GaussSegment(Func<Point2D, double> psi, Point2D firstPoint,
5     ↪ Point2D secondPoint) {
6         var quadratures = Quadratures.SegmentGaussOrder9();
7
8         double result = 0.0;
9         double lenghtEdge = Math.Sqrt(((firstPoint.R - secondPoint.R) * (firstPoint.R
10     ↪ - secondPoint.R)) +
11     ((firstPoint.Z - secondPoint.Z) * (firstPoint.Z -
12     ↪ secondPoint.Z)));
13
14         foreach (var q in quadratures) {
15             double qi = q.Weight;
16             double pi = (1 + q.Node) / 2.0;
17
18             var parameterized = Parameterization(pi);
19
20             result += qi * psi(parameterized) * parameterized.R;
21         }
22
23         return result * lenghtEdge / 2.0;
24
25         Point2D Parameterization(double t)
26             => ((secondPoint - firstPoint) * t) + firstPoint;
27     }
28 }

```

```

26 public static double Triangle(Func<Point2D, double, double> f, Func<Point2D,
    ↪ double> psi, Point2D[] vertices, double t) {
27     var quadratures = Quadratures.TriangleOrder6();
28
29     double result = 0.0;
30     double determinant = Math.Abs(Determinant());
31
32     foreach (var q in quadratures) {
33         var point = ((1 - q.Node.R - q.Node.Z) * vertices[0]) + (q.Node.R *
    ↪ vertices[1]) + (q.Node.Z * vertices[2]);
34
35         result += f(point, t) * psi(point) * q.Weight * determinant * 0.5 *
    ↪ point.R;
36     }
37
38     return result;
39
40     double Determinant()
41     => ((vertices[1].R - vertices[0].R) * (vertices[2].Z - vertices[0].Z)) -
42         ((vertices[2].R - vertices[0].R) * (vertices[1].Z - vertices[0].Z));
43 }
44 }

```

## Quadratures.cs

```

1 namespace courseProject;
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
22 public static class Quadratures {
23     public static IEnumerable<QuadratureNode<double>> SegmentGaussOrder9() {
24         const int n = 5;
25         double[] points = { 0.0,
26                             1.0 / 3.0 * Math.Sqrt(5 - (2 * Math.Sqrt(10.0 / 7.0))),
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                             -1.0 / 3.0 * Math.Sqrt(5 + (2 * Math.Sqrt(10.0 / 7.0))) };
30
31         double[] weights = { 128.0 / 225.0,
32                             (322.0 + (13.0 * Math.Sqrt(70.0))) / 900.0,
33                             (322.0 + (13.0 * Math.Sqrt(70.0))) / 900.0,
34                             (322.0 - (13.0 * Math.Sqrt(70.0))) / 900.0,
35                             (322.0 - (13.0 * Math.Sqrt(70.0))) / 900.0 };

```

```

36         for (int i = 0; i < n; i++) {
37             yield return new QuadratureNode<double>(points[i], weights[i]);
38         }
39     }
40
41
42     public static IEnumerable<QuadratureNode<Point2D>> TriangleOrder6() {
43         const double x1a = 0.873821971016996;
44         const double x1b = 0.063089014491502;
45         const double x2a = 0.501426509658179;
46         const double x2b = 0.249286745170910;
47         const double x3a = 0.636502499121399;
48         const double x3b = 0.310352451033785;
49         const double x3c = 0.053145049844816;
50         const double w1 = 0.050844906370207;
51         const double w2 = 0.116786275726379;
52         const double w3 = 0.082851075618374;
53
54         double[] p1 = { x1a, x1b, x1b, x2a, x2b, x2b, x3a, x3b, x3a, x3c, x3b, x3c };
55         double[] p2 = { x1b, x1a, x1b, x2b, x2a, x2b, x3b, x3a, x3c, x3a, x3c, x3b };
56         double[] w = { w1, w1, w1, w2, w2, w2, w3, w3, w3, w3, w3, w3 };
57
58         for (int i = 0; i < w.Length; i++) {
59             yield return new QuadratureNode<Point2D>(new(p1[i], p2[i]), w[i]);
60         }
61     }
62 }

```

## CubicBasis.cs

```

1 namespace courseProject;
2
3 public static class CubicBasis {
4     private static readonly Matrix _alphas;
5
6     static CubicBasis() {
7         _alphas = FEM.GetAlphas();
8     }
9
10    private static double L1(Point2D point)
11        => _alphas[0, 0] + (_alphas[0, 1] * point.R) + (_alphas[0, 2] * point.Z);
12
13    private static double L2(Point2D point)
14        => _alphas[1, 0] + (_alphas[1, 1] * point.R) + (_alphas[1, 2] * point.Z);
15
16    private static double L3(Point2D point)
17        => _alphas[2, 0] + (_alphas[2, 1] * point.R) + (_alphas[2, 2] * point.Z);
18
19    public static double Psi1(Point2D point) {
20        double l1 = L1(point);
21        return 0.5 * l1 * ((3 * l1) - 1) * ((3 * l1) - 2);
22    }
23    public static double Psi2(Point2D point) {
24        double l2 = L2(point);
25        return 0.5 * l2 * ((3 * l2) - 1) * ((3 * l2) - 2);
26    }
27
28    public static double Psi3(Point2D point) {
29        double l3 = L3(point);
30        return 0.5 * l3 * ((3 * l3) - 1) * ((3 * l3) - 2);

```



```

31     }
32
33     public static double Psi4(Point2D point) {
34         double l1 = L1(point);
35         double l2 = L2(point);
36         return 4.5 * l1 * l2 * ((3 * l1) - 1);
37     }
38
39     public static double Psi5(Point2D point) {
40         double l1 = L1(point);
41         double l2 = L2(point);
42         return 4.5 * l1 * l2 * ((3 * l2) - 1);
43     }
44
45     public static double Psi6(Point2D point) {
46         double l2 = L2(point);
47         double l3 = L3(point);
48         return 4.5 * l2 * l3 * ((3 * l2) - 1);
49     }
50
51     public static double Psi7(Point2D point) {
52         double l2 = L2(point);
53         double l3 = L3(point);
54         return 4.5 * l2 * l3 * ((3 * l3) - 1);
55     }
56
57     public static double Psi8(Point2D point) {
58         double l1 = L1(point);
59         double l3 = L3(point);
60         return 4.5 * l1 * l3 * ((3 * l3) - 1);
61     }
62
63     public static double Psi9(Point2D point) {
64         double l1 = L1(point);
65         double l3 = L3(point);
66         return 4.5 * l1 * l3 * ((3 * l1) - 1);
67     }
68
69     public static double Psi10(Point2D point) {
70         double l1 = L1(point);
71         double l2 = L2(point);
72         double l3 = L3(point);
73         return 27 * l1 * l2 * l3;
74     }
75 }

```

## Boundaries.cs

```

1 namespace courseProject;
2
3 public readonly record struct DirichletBoundary(int Element, int Edge) {
4     public static DirichletBoundary[]? ReadJson(string jsonPath) {
5         try {
6             if (!File.Exists(jsonPath))
7                 throw new Exception("File does not exist");
8
9             var sr = new StreamReader(jsonPath);
10            using (sr) {
11                return JsonConvert.DeserializeObject<DirichletBoundary[]>(sr.ReadToEnd())
12                    ↪ d();
13            }
14        }
15        catch { }
16    }
17 }

```

```

12     }
13     } catch (Exception ex) {
14         Console.WriteLine($"We had problem: {ex.Message}");
15         return null;
16     }
17 }
18 }
19
20 public readonly record struct NeumannBoundary(int Element, int Edge, double Value) {
21     public static NeumannBoundary[]? ReadJson(string jsonPath) {
22         try {
23             if (!File.Exists(jsonPath))
24                 throw new Exception("File does not exist");
25
26             var sr = new StreamReader(jsonPath);
27             using (sr) {
28                 return
29                     ↪ JsonConvert.DeserializeObject<NeumannBoundary[]>(sr.ReadToEnd());
30             }
31         } catch (Exception ex) {
32             Console.WriteLine($"We had problem: {ex.Message}");
33             return null;
34         }
35     }
36 }

```

## Tests.cs

```

1 namespace courseProject;
2
3 public interface ITest {
4     public double U(Point2D point, double t);
5
6     public double F(Point2D point, double t);
7 }
8
9 public class Test1 : ITest {
10     public double U(Point2D point, double t)
11         => (point.R * point.R) + point.Z + t;
12
13     public double F(Point2D point, double t)
14         => -3;
15 }
16
17 public class Test2 : ITest {
18     public double U(Point2D point, double t)
19         => (point.R * point.R) - (point.Z * t * t);
20
21     public double F(Point2D point, double t)
22         => -8 - (t * point.Z);
23 }
24
25 public class Test3 : ITest {
26     public double U(Point2D point, double t)
27         => 2 * point.R * point.R * point.R * t * t * t;
28
29     public double F(Point2D point, double t)
30         => (-9 * point.R * t * t * t) + (12 * point.R * point.R * point.R * t * t);
31 }
32

```

```

33 public class Test4 : ITest {
34     public double U(Point2D point, double t)
35         => (point.R * point.R * point.R * point.R) + (point.Z * point.Z * point.Z *
36             ↪ point.Z) + (t * t * t * t);
37
38     public double F(Point2D point, double t)
39         => (-16 * point.R * point.R) - (12 * point.Z * point.Z) + (4 * t * t * t);
40 }
41
42 public class Test5 : ITest {
43     public double U(Point2D point, double t)
44         => Math.Log(point.R);
45
46     public double F(Point2D point, double t)
47         => 0;
48 }
49
50 public class Test6 : ITest {
51     public double U(Point2D point, double t)
52         => Math.Cos(t);
53
54     public double F(Point2D point, double t)
55         => -Math.Sin(t);
56 }
57
58 public class Test7 : ITest {
59     public double U(Point2D point, double t)
60         => t * t * t * t;
61
62     public double F(Point2D point, double t)
63         => 4 * t * t * t;
64 }

```

## GridsParameters.cs

```

1 namespace courseProject;
2
3 public class SpaceGridParametersJsonConverter : 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 = (SpaceGridParameters)value;
12
13         writer.WriteStartObject();
14         writer.WritePropertyName("Initial area in R");
15         serializer.Serialize(writer, gridParameters.IntervalR);
16         writer.WritePropertyName("Splits by R");
17         writer.WriteValue(gridParameters.SplitsR);
18         writer.WriteWhitespace("\n");
19
20         writer.WritePropertyName("Initial area in Z");
21         serializer.Serialize(writer, gridParameters.IntervalZ);
22         writer.WritePropertyName("Splits by Z");
23         writer.WriteValue(gridParameters.SplitsZ);
24
25         writer.WriteComment("Коэффициент разрядки");
26     }
27 }

```

```

25     writer.WritePropertyName("Coef");
26     writer.WriteValue(gridParameters.K);
27     writer.WriteWhitespace("\n");
28     writer.WriteComment("Uniform area");
29     writer.WritePropertyName("Lambda");
30     writer.WriteValue(gridParameters.Lambda);
31     writer.WritePropertyName("Sigma");
32     writer.WriteValue(gridParameters.Sigma);
33     writer.WriteEndObject();
34 }
35
36 public override object? ReadJson(JsonReader reader, Type objectType, object?
↪ existingValue, JsonSerializer serializer) {
37     if (reader.TokenType == JsonToken.Null || reader.TokenType !=
↪     JsonToken.StartObject)
38         return null;
39
40     Interval intervalR;
41     Interval intervalZ;
42     int splitsR;
43     int splitsZ;
44     double? coef;
45     double lambda, sigma;
46
47     var maintoken = JObject.Load(reader);
48
49     var token = maintoken["Initial area in R"];
50     intervalR = serializer.Deserialize<Interval>(token!.CreateReader());
51     token = maintoken["Splits by R"];
52     splitsR = Convert.ToInt32(token);
53
54     token = maintoken["Initial area in Z"];
55     intervalZ = serializer.Deserialize<Interval>(token!.CreateReader());
56     token = maintoken["Splits by Z"];
57     splitsZ = Convert.ToInt32(token);
58
59     token = maintoken["Coef"];
60
61     if (token is not null) {
62         coef = double.TryParse(token.ToString(), out double res) ? res : null;
63     }
64     else {
65         coef = null;
66     }
67
68     token = maintoken["Lambda"];
69     lambda = Convert.ToDouble(token);
70
71     token = maintoken["Sigma"];
72
73     if (token is not null) {
74         sigma = double.TryParse(token.ToString(), out double res) ? res : 0.0;
75     }
76     else {
77         sigma = 0.0;
78     }
79
80     return new SpaceGridParameters(intervalR, splitsR, intervalZ, splitsZ, coef,
↪     lambda, sigma);
81 }

```

```

82
83     public override bool CanConvert(Type objectType)
84         => objectType == typeof(SpaceGridParameters);
85 }
86
87 [JsonConverter(typeof(SpaceGridParametersJsonConverter))]
88 public readonly record struct SpaceGridParameters {
89     public Interval IntervalR { get; init; }
90     public int SplitsR { get; init; }
91     public Interval IntervalZ { get; init; }
92     public int SplitsZ { get; init; }
93     public double? K { get; init; }
94
95     // Uniform area
96     public double Lambda { get; init; }
97     public double Sigma { get; init; }
98
99     public SpaceGridParameters(Interval intervalR, int splitsR, Interval intervalZ,
100         ↪ int splitsZ, double? k, double lambda, double sigma) {
101         IntervalR = intervalR;
102         SplitsR = splitsR;
103         IntervalZ = intervalZ;
104         SplitsZ = splitsZ;
105         K = k;
106         Lambda = lambda;
107         Sigma = sigma;
108     }
109
110     public static SpaceGridParameters? ReadJson(string jsonPath) {
111         try {
112             if (!File.Exists(jsonPath))
113                 throw new Exception("File does not exist");
114
115             var sr = new StreamReader(jsonPath);
116             using (sr) {
117                 return JsonConvert.DeserializeObject<SpaceGridParameters>(sr.ReadToEnd)
118                     ↪ d();
119             }
120         } catch (Exception ex) {
121             Console.WriteLine($"We had problem: {ex.Message}");
122             return null;
123         }
124     }
125
126     public readonly record struct TimeGridParameters {
127         [JsonProperty("Initial area")]
128         public Interval Interval { get; init; }
129
130         [JsonProperty("Number of splits")]
131         public int Splits { get; init; }
132
133         [JsonProperty("Coef")]
134         public double? K { get; init; } // коэффициент разрядки
135
136         public TimeGridParameters(Interval interval, int splits, double? k) {
137             Interval = interval;
138             Splits = splits;
139             K = k;
140         }
141     }

```

```

140
141     public static TimeGridParameters? ReadJson(string jsonPath) {
142         try {
143             if (!File.Exists(jsonPath))
144                 throw new Exception("File does not exist");
145
146             var sr = new StreamReader(jsonPath);
147             using (sr) {
148                 return
149                     ↪ JsonConvert.DeserializeObject<TimeGridParameters>(sr.ReadToEnd());
150             }
151         } catch (Exception ex) {
152             Console.WriteLine($"We had problem: {ex.Message}");
153             return null;
154         }
155     }

```

## IGrids.cs

```

1 namespace courseProject;
2
3 public interface ISpaceGrid {
4     public double Lambda { get; init; }
5     public double Sigma { get; init; }
6     public ImmutableList<Point2D> Points { get; }
7     public ImmutableArray<ImmutableArray<int>> Elements { get; }
8     public ImmutableArray<ImmutableArray<ImmutableArray<int>>> Edges { get; }
9 }
10
11 public interface ITimeGrid {
12     public ImmutableArray<double> Points { get; }
13 }

```

## GridFactories.cs

```

1 namespace courseProject;
2
3 public enum GridTypes {
4     SpaceRegular,
5     TimeRegular,
6     TimeIrregular
7 }
8
9 public interface ISpaceFactory {
10     public ISpaceGrid CreateGrid(GridTypes spaceGridType, SpaceGridParameters
11         ↪ gridParameters);
12 }
13
14 public interface ITimeFactory {
15     public ITimeGrid CreateGrid(GridTypes timeGridType, TimeGridParameters
16         ↪ gridParameters);
17 }
18
19 public class SpaceGridFactory : ISpaceFactory {
20     public ISpaceGrid CreateGrid(GridTypes spaceGridType, SpaceGridParameters
21         ↪ gridParameters) {
22         return spaceGridType switch {
23             GridTypes.SpaceRegular => new SpaceRegularGrid(gridParameters),

```

```

21         _ => throw new ArgumentOutOfRangeException(nameof(spaceGridType), $"This
22         ↪ type of grid does not exist: {spaceGridType}"),
23     };
24 }
25 }
26
27 public class TimeGridFactory : ITimeFactory {
28     public ITimeGrid CreateGrid(GridTypes timeGridType, TimeGridParameters
29     ↪ gridParameters) {
30         return timeGridType switch {
31             GridTypes.TimeRegular => new TimeRegularGrid(gridParameters),
32             GridTypes.TimeIrregular => new TimeIrregularGrid(gridParameters),
33
34             _ => throw new ArgumentOutOfRangeException(nameof(timeGridType), $"This
35             ↪ type of grid does not exist: {timeGridType}"),
36         };
37     }
38 }

```

## SpaceGrids.cs

```

1 namespace courseProject;
2
3 public class SpaceRegularGrid : ISpaceGrid {
4     private readonly List<Point2D> _points = default!;
5     private readonly int[][] _elements = default!;
6     private readonly int[][][] _edges = default!;
7     public ImmutableList<Point2D> Points => _points.ToImmutableList();
8     public ImmutableArray<ImmutableArray<int>> Elements => _elements.Select(item =>
9     ↪ item.ToImmutableArray()).ToImmutableArray();
10    public ImmutableArray<ImmutableArray<ImmutableArray<int>>> Edges =>
11    ↪ _edges.Select(item => item.ToImmutableArray().Select(item =>
12    ↪ item.ToImmutableArray()).ToImmutableArray()).ToImmutableArray();
13    public double Lambda { get; init; }
14    public double Sigma { get; init; }
15
16    public SpaceRegularGrid(SpaceGridParameters gridParameters) {
17        Lambda = gridParameters.Lambda;
18        Sigma = gridParameters.Sigma;
19        _points = new();
20        _elements = new int[2 * gridParameters.SplitsR *
21        ↪ gridParameters.SplitsZ].Select(_ => new int[10]).ToArray();
22        _edges = new int[2 * gridParameters.SplitsR *
23        ↪ gridParameters.SplitsZ].Select(_ => new int[3]).ToArray().Select(_ => new
24        ↪ int[4]).ToArray().ToArray();
25        Build(gridParameters);
26    }
27
28    private void Build(SpaceGridParameters gridParameters) {
29        try {
30            if (gridParameters.SplitsR < 1 || gridParameters.SplitsZ < 1) {
31                throw new Exception("The number of splits must be greater than or
32                ↪ equal to 1");
33            }
34
35            double hr = gridParameters.IntervalR.Length / gridParameters.SplitsR;
36            double hz = gridParameters.IntervalZ.Length / gridParameters.SplitsZ;
37
38        }
39    }

```

```

31 double[] pointsR = new double[(3 * gridParameters.SplitsR) + 1];
32 double[] pointsZ = new double[(3 * gridParameters.SplitsZ) + 1];
33
34 pointsR[0] = gridParameters.IntervalR.LeftBorder;
35 pointsZ[0] = gridParameters.IntervalZ.LeftBorder;
36
37 for (int i = 1; i < gridParameters.SplitsR + 1; i++) {
38     pointsR[i] = pointsR[i - 1] + hr;
39 }
40
41 for (int i = 1; i < gridParameters.SplitsZ + 1; i++) {
42     pointsZ[i] = pointsZ[i - 1] + hz;
43 }
44
45 for (int j = 0; j < gridParameters.SplitsZ + 1; j++) {
46     for (int i = 0; i < gridParameters.SplitsR + 1; i++) {
47         _points.Add(new Point2D(pointsR[i], pointsZ[j]));
48     }
49 }
50
51 Point2D[] allNodes = new Point2D[10];
52 Point2D[] vertices = new Point2D[3];
53
54 const int dummySplits = 2;
55 int nr = gridParameters.SplitsR + 1;
56 int index = 0;
57
58 for (int j = 0; j < gridParameters.SplitsZ; j++) {
59     for (int i = 0; i < gridParameters.SplitsR; i++) {
60         _elements[index][0] = i + (j * nr);
61         _elements[index][1] = i + 1 + (j * nr);
62         _elements[index++][2] = i + ((j + 1) * nr);
63
64         _elements[index][0] = i + 1 + (j * nr);
65         _elements[index][1] = i + nr + (j * nr);
66         _elements[index++][2] = i + 1 + nr + (j * nr);
67     }
68 }
69
70 WritePoints("forGraphics/pointsLinear.txt");
71 WriteElements("forGraphics/elements.txt");
72 pointsR.Fill(-1.0);
73 pointsZ.Fill(-1.0);
74
75 for (int ielem = 0; ielem < _elements.Length; ielem++) {
76     vertices[0] = _points[_elements[ielem][0]];
77     vertices[1] = _points[_elements[ielem][1]];
78     vertices[2] = _points[_elements[ielem][2]];
79
80     GetNodes(allNodes, vertices);
81
82     pointsR = pointsR.Concat(allNodes.Select(point => point.R)).ToArray();
83     pointsR = pointsR.Distinct().OrderBy(value => value).ToArray();
84
85     pointsZ = pointsZ.Concat(allNodes.Select(point => point.Z)).ToArray();
86     pointsZ = pointsZ.Distinct().OrderBy(value => value).ToArray();
87 }
88
89 _points.Clear();

```



```

91     foreach (var pointZ in pointsZ.Skip(1)) {
92         foreach (var pointR in pointsR.Skip(1)) {
93             _points.Add(new(pointR, pointZ));
94         }
95     }
96
97     nr = pointsR.Length - 1;
98     index = 0;
99
100    for (int j = 0; j < gridParameters.SplitsZ; j++) {
101        for (int i = 0; i < gridParameters.SplitsR; i++) {
102            _elements[index][0] = i + (3 * j * nr) + (i * dummySplits); // 0
103            _elements[index][1] = i + (3 * nr) + (3 * j * nr) + (i *
104                ↪ dummySplits); // 12
105            _elements[index][2] = i + 3 + (3 * j * nr) + (i * dummySplits);
106            ↪ // 3
107            _elements[index][3] = i + nr + (3 * j * nr) + (i * dummySplits);
108            ↪ // 4
109            _elements[index][4] = i + (2 * nr) + (3 * j * nr) + (i *
110                ↪ dummySplits); // 8
111            _elements[index][5] = i + 1 + (2 * nr) + (3 * j * nr) + (i *
112                ↪ dummySplits); // 9
113            _elements[index][6] = i + nr + 2 + (3 * j * nr) + (i *
114                ↪ dummySplits); // 6
115            _elements[index][7] = i + 2 + (3 * j * nr) + (i * dummySplits);
116            ↪ // 2
117            _elements[index][8] = i + 1 + (3 * j * nr) + (i * dummySplits);
118            ↪ // 1
119            _elements[index][9] = i + nr + 1 + (3 * j * nr) + (i *
120                ↪ dummySplits); // 5
121
122            _edges[index][0][0] = _elements[index][0];
123            _edges[index][0][1] = _elements[index][8];
124            _edges[index][0][2] = _elements[index][7];
125            _edges[index][0][3] = _elements[index][2];
126
127            _edges[index][1][0] = _elements[index][0];
128            _edges[index][1][1] = _elements[index][3];
129            _edges[index][1][2] = _elements[index][4];
130            _edges[index][1][3] = _elements[index][1];
131
132            _edges[index][2][0] = _elements[index][2];
133            _edges[index][2][1] = _elements[index][6];
134            _edges[index][2][2] = _elements[index][5];
135            _edges[index][2][3] = _elements[index][1];
136
137            _elements[index][0] = i + (3 * nr) + (3 * j * nr) + (i *
138                ↪ dummySplits); // 12
139            _elements[index][1] = i + 3 + (3 * j * nr) + (i * dummySplits);
140            ↪ // 3
141            _elements[index][2] = i + 3 + (3 * nr) + (3 * j * nr) + (i *
142                ↪ dummySplits); // 15
143            _elements[index][3] = i + 1 + (2 * nr) + (3 * j * nr) + (i *
144                ↪ dummySplits); // 9
145            _elements[index][4] = i + nr + 2 + (3 * j * nr) + (i *
146                ↪ dummySplits); // 6
147            _elements[index][5] = i + nr + 3 + (3 * j * nr) + (i *
148                ↪ dummySplits); // 7
149            _elements[index][6] = i + 3 + (2 * nr) + (3 * j * nr) + (i *
150                ↪ dummySplits); // 11

```

```

135         _elements[index][7] = i + 2 + (3 * nr) + (3 * j * nr) + (i *
            ↪ dummySplits); //14
136         _elements[index][8] = i + 1 + (3 * nr) + (3 * j * nr) + (i *
            ↪ dummySplits); // 13
137         _elements[index][9] = i + 2 + (2 * nr) + (3 * j * nr) + (i *
            ↪ dummySplits); // 10
138
139         _edges[index][0][0] = _elements[index][0];
140         _edges[index][0][1] = _elements[index][8];
141         _edges[index][0][2] = _elements[index][7];
142         _edges[index][0][3] = _elements[index][2];
143
144         _edges[index][1][0] = _elements[index][1];
145         _edges[index][1][1] = _elements[index][4];
146         _edges[index][1][2] = _elements[index][3];
147         _edges[index][1][3] = _elements[index][0];
148
149         _edges[index][2][0] = _elements[index][1];
150         _edges[index][2][1] = _elements[index][5];
151         _edges[index][2][2] = _elements[index][6];
152         _edges[index][2][3] = _elements[index++][2];
153     }
154 }
155
156     WritePoints("forGraphics/allPoints.txt");
157 } catch (Exception ex) {
158     Console.WriteLine($"We had problem: {ex.Message}");
159 }
160 }
161
162 private static void GetNodes(Point2D[] allNodes, Point2D[] vertices) {
163     allNodes[0] = vertices[0];
164     allNodes[1] = vertices[1];
165     allNodes[2] = vertices[2];
166     allNodes[3] = ((2 * vertices[0]) + vertices[1]) / 3;
167     allNodes[4] = ((2 * vertices[1]) + vertices[0]) / 3;
168     allNodes[5] = ((2 * vertices[1]) + vertices[2]) / 3;
169     allNodes[6] = ((2 * vertices[2]) + vertices[1]) / 3;
170     allNodes[7] = ((2 * vertices[2]) + vertices[0]) / 3;
171     allNodes[8] = ((2 * vertices[0]) + vertices[2]) / 3;
172     allNodes[9] = (vertices[0] + vertices[1] + vertices[2]) / 3;
173 }
174
175 private void WritePoints(string path) {
176     using var sw = new StreamWriter(path);
177     for (int i = 0; i < _points.Count; i++) {
178         sw.WriteLine(_points[i]);
179     }
180 }
181
182 private void WriteElements(string path) {
183     using var sw = new StreamWriter(path);
184     for (int i = 0; i < _elements.Length; i++) {
185         sw.WriteLine($"{_elements[i][0]} {_elements[i][1]} {_elements[i][2]}");
186     }
187 }
188 }

```

## TimeGrids.cs

```
1 namespace courseProject;
2
3 public class TimeRegularGrid : ITimeGrid {
4     private double[] _points = default!;
5     public ImmutableArray<double> Points => _points.ToImmutableArray();
6
7     public TimeRegularGrid(TimeGridParameters gridParameters) {
8         Build(gridParameters);
9     }
10
11     private void Build(TimeGridParameters gridParameters) {
12         try {
13             if (gridParameters.Interval.LeftBorder < 0)
14                 throw new Exception("The beginning of the time segment cannot be less
15                                     ↪ than 0");
16
17             if (gridParameters.Splits < 1)
18                 throw new Exception("The number of splits must be greater than or
19                                     ↪ equal to 1");
20
21             _points = new double[gridParameters.Splits + 1];
22
23             double h = gridParameters.Interval.Lenght / gridParameters.Splits;
24
25             _points[0] = gridParameters.Interval.LeftBorder;
26             _points[^1] = gridParameters.Interval.RightBorder;
27
28             for (int i = 1; i < _points.Length - 1; i++) {
29                 _points[i] = _points[i - 1] + h;
30             }
31         } catch (Exception ex) {
32             Console.WriteLine($"We had problem: {ex.Message}");
33         }
34     }
35
36 public class TimeIrregularGrid : ITimeGrid {
37     private double[] _points = default!;
38     public ImmutableArray<double> Points => _points.ToImmutableArray();
39
40     public TimeIrregularGrid(TimeGridParameters gridParameters) {
41         Build(gridParameters);
42     }
43
44     private void Build(TimeGridParameters gridParameters) {
45         try {
46             if (gridParameters.Interval.LeftBorder < 0)
47                 throw new Exception("The beginning of the time segment cannot be less
48                                     ↪ than 0");
49
50             if (gridParameters.Splits < 1)
51                 throw new Exception("The number of splits must be greater than or
52                                     ↪ equal to 1");
53
54             ArgumentNullException.ThrowIfNull(gridParameters.K,
55                 ↪ $"{nameof(gridParameters.K)} cannot be null");
56
57             _points = new double[gridParameters.Splits + 1];
```

```

54         double h;
55         double sum = 0.0;
56
57         for (int k = 0; k < gridParameters.Splits; k++)
58             sum += Math.Pow(gridParameters.K.Value, k);
59
60         h = gridParameters.Interval.Lenght / sum;
61         _points[0] = gridParameters.Interval.LeftBorder;
62         _points[^1] = gridParameters.Interval.RightBorder;
63
64         for (int i = 1; i < _points.Length - 1; i++) {
65             _points[i] = _points[i - 1] + h;
66             h *= gridParameters.K.Value;
67         }
68     } catch (Exception ex) {
69         Console.WriteLine($"We had problem: {ex.Message}");
70     }
71 }
72
73 }

```

## Interval.cs

```

1 namespace courseProject;
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 }

```

## Point2D.cs

```

1 namespace courseProject;
2
3 public readonly record struct Point2D(double R, double Z) {
4     public static Point2D operator *(double constant, Point2D point)
5         => new(constant * point.R, constant * point.Z);
6
7     public static Point2D operator *(Point2D point, double constant)
8         => new(point.R * constant, point.Z * constant);
9
10    public static Point2D operator +(Point2D firstPoint, Point2D secondPoint)
11        => new(firstPoint.R + secondPoint.R, firstPoint.Z + secondPoint.Z);
12
13    public static Point2D operator -(Point2D firstPoint, Point2D secondPoint)
14        => new(firstPoint.R - secondPoint.R, firstPoint.Z - secondPoint.Z);

```

```

15
16     public static Point2D operator /(Point2D point, double constant)
17         => new(point.R / constant, point.Z / constant);
18
19     public override string ToString()
20         => $"{R} {Z}";
21 }

```

## ArrayHelper.cs

```

1 namespace courseProject;
2
3 public static class ArrayHelper {
4     public static void Fill<T>(this T[] array, T value) {
5         for (int i = 0; i < array.Length; i++) {
6             array[i] = value;
7         }
8     }
9
10    public static double Norm(this double[] array) {
11        double result = 0.0;
12
13        for (int i = 0; i < array.Length; i++) {
14            result += array[i] * array[i];
15        }
16
17        return Math.Sqrt(result);
18    }
19
20    public static void Copy<T>(this T[] source, T[] destination) {
21        for (int i = 0; i < source.Length; i++) {
22            destination[i] = source[i];
23        }
24    }
25 }

```

## GMrz.py

```

1 from sympy import expand, Symbol, diff
2
3 l1 = Symbol('l1')
4 l2 = Symbol('l2')
5 l3 = Symbol('l3')
6 d11 = Symbol('d11')
7 d12 = Symbol('d21')
8 d13 = Symbol('d31')
9
10 #функции
11 # psi1 = 0.5*l1*(3*l1-1)*(3*l1-2)
12 # psi2 = 0.5*l2*(3*l2-1)*(3*l2-2)
13 # psi3 = 0.5*l3*(3*l3-1)*(3*l3-2)
14 # psi4 = 4.5*l1*l2*(3*l1-1)
15 # psi5 = 4.5*l1*l2*(3*l2-1)
16 # psi6 = 4.5*l2*l3*(3*l2-1)
17 # psi7 = 4.5*l2*l3*(3*l3-1)
18 # psi8 = 4.5*l3*l1*(3*l3-1)
19 # psi9 = 4.5*l3*l1*(3*l1-1)
20 # psi10 = 27*l1*l2*l3
21

```

```

22 psi1 = 0.5*l1*(3*l1-1)*(3*l1-2)
23 psi2 = 4.5*l3*l1*(3*l1-1)
24 psi3 = 4.5*l3*l1*(3*l3-1)
25 psi4 = 0.5*l3*(3*l3-1)*(3*l3-2)
26 psi5 = 4.5*l1*l2*(3*l1-1)
27 psi6 = 27*l1*l2*l3
28 psi7 = 4.5*l2*l3*(3*l3-1)
29 psi8 = 4.5*l1*l2*(3*l2-1)
30 psi9 = 4.5*l2*l3*(3*l2-1)
31 psi10 = 0.5*l2*(3*l2-1)*(3*l2-2)
32
33 psi = [psi1, psi2, psi3, psi4, psi5, psi6, psi7, psi8, psi9, psi10]
34
35 #производные
36 p1 = dl1*diff(psi1, l1)+dl2*diff(psi1, l2)+dl3*diff(psi1, l3)
37 p2 = dl1*diff(psi2, l1)+dl2*diff(psi2, l2)+dl3*diff(psi2, l3)
38 p3 = dl1*diff(psi3, l1)+dl2*diff(psi3, l2)+dl3*diff(psi3, l3)
39 p4 = dl1*diff(psi4, l1)+dl2*diff(psi4, l2)+dl3*diff(psi4, l3)
40 p5 = dl1*diff(psi5, l1)+dl2*diff(psi5, l2)+dl3*diff(psi5, l3)
41 p6 = dl1*diff(psi6, l1)+dl2*diff(psi6, l2)+dl3*diff(psi6, l3)
42 p7 = dl1*diff(psi7, l1)+dl2*diff(psi7, l2)+dl3*diff(psi7, l3)
43 p8 = dl1*diff(psi8, l1)+dl2*diff(psi8, l2)+dl3*diff(psi8, l3)
44 p9 = dl1*diff(psi9, l1)+dl2*diff(psi9, l2)+dl3*diff(psi9, l3)
45 p10 = dl1*diff(psi10, l1)+dl2*diff(psi10, l2)+dl3*diff(psi10, l3)
46
47 p=[p1, p2, p3, p4, p5, p6, p7, p8, p9, p10]
48
49 def fac(n):
50     if n == 0:
51         return 1
52     return fac(n-1) * n
53
54 def Integrate(func):
55     res = [0, 0, 0]
56     func = func.replace("- 2", "-2")
57     func = func.replace("- 3", "-3")
58     func = func.replace("- 4", "-4")
59     func = func.replace("- 1", "-1")
60     func = func.replace("- 5", "-5")
61     func = func.replace("- 6", "-6")
62     func = func.replace("- 7", "-7")
63     func = func.replace("- 8", "-8")
64     func = func.replace("- 9", "-9")
65     func = func.replace("+ ", "+")
66     func = func.replace("**", "^")
67     func = func.split()
68     func = [i.split("*", 1) for i in func]
69     for iter in range(len(func)):
70         a = float(func[iter][0])
71         f = func[iter][1].rfind("l1^")
72         if f>=0:
73             i = int(func[iter][1][func[iter][1].rfind("l1")+3])
74         else:
75             f = func[iter][1].rfind("l1")
76             if f>=0:
77                 i = 1
78             else:
79                 i = 0
80         f = func[iter][1].rfind("l2^")
81         if f>=0:

```

```

82         j = int(func[iter][1][func[iter][1].rfind('12')+3])
83     else:
84         f = func[iter][1].rfind('12')
85         if f>=0:
86             j = 1
87         else:
88             j = 0
89     f = func[iter][1].rfind('13')
90     if f>=0:
91         k = int(func[iter][1][func[iter][1].rfind('13')+3])
92     else:
93         f = func[iter][1].rfind('13')
94         if f>=0:
95             k = 1
96         else:
97             k = 0
98
99     res[0]+=a * fac(i+1)*fac(j)*fac(k)/fac(i+j+k+3)
100    res[1]+=a * fac(i)*fac(j+1)*fac(k)/fac(i+j+k+3)
101    res[2]+=a * fac(i)*fac(j)*fac(k+1)/fac(i+j+k+3)
102
103    return res
104
105    def makeM():
106        f1 = open('Mrz.txt', 'w')
107        mf = []
108        rs = [[], [], [], [], [], [], [], [], [], []]
109        for i in range(10):
110            for j in range(10):
111                r = str(expand(psi[i]*psi[j]))
112                mf.append(r)
113                rs[i].append(Integrate(r))
114        i, j = 0, 0
115        for row in rs:
116            for x in row:
117                f1.write(str(i) + " " + str(j) + " " + '
118                ↪ '.join(["{0:.20f}".format(float(elem)) for elem in x]) + "\n")
119                j+=1
120            i+=1
121            j=0
122        f1.close()
123        return rs
124
125    def IntegrateGrad(func):
126        res = [[0,0,0], [0,0,0], [0,0,0], [0,0,0], [0,0,0], [0,0,0]]
127        func = func.replace("- 2", "-2")
128        func = func.replace("- 3", "-3")
129        func = func.replace("- 4", "-4")
130        func = func.replace("- 1", "-1")
131        func = func.replace("- 5", "-5")
132        func = func.replace("- 6", "-6")
133        func = func.replace("- 7", "-7")
134        func = func.replace("- 8", "-8")
135        func = func.replace("- 9", "-9")
136        func = func.replace('+ ', '+')
137        func = func.replace("**", "^")
138        func = func.split()
139        func = [i.split("*", 1) for i in func]
140        for iter in range(len(func)):
            if func[iter][1].rfind("d11^2")>=0: flag = 0

```

```

141 elif func[iter][1].rfind("d11*d21")>=0: flag = 1
142 elif func[iter][1].rfind("d11*d31")>=0: flag = 2
143 elif func[iter][1].rfind("d21^2")>=0: flag = 3
144 elif func[iter][1].rfind("d21*d31")>=0: flag = 4
145 elif func[iter][1].rfind("d31^2")>=0: flag = 5
146 a = float(func[iter][0])
147 f = func[iter][1].rfind('11^')
148 if f>=0:
149     i = int(func[iter][1][func[iter][1].rfind('11^')+3])
150 else:
151     f = func[iter][1].rfind('11')
152     if f>=0:
153         i = 1
154     else:
155         i = 0
156 f = func[iter][1].rfind('12^')
157 if f>=0:
158     j = int(func[iter][1][func[iter][1].rfind('12^')+3])
159 else:
160     f = func[iter][1].rfind('12')
161     if f>=0:
162         j = 1
163     else:
164         j = 0
165 f = func[iter][1].rfind('13^')
166 if f>=0:
167     k = int(func[iter][1][func[iter][1].rfind('13^')+3])
168 else:
169     f = func[iter][1].rfind('13')
170     if f>=0:
171         k = 1
172     else:
173         k = 0
174 res[flag][0]+=a * fac(i+1)*fac(j)*fac(k)/fac(i+j+k+3)
175 res[flag][1]+=a * fac(i)*fac(j+1)*fac(k)/fac(i+j+k+3)
176 res[flag][2]+=a * fac(i)*fac(j)*fac(k+1)/fac(i+j+k+3)
177
178 return res
179
180 #L(x,y)=a0i+a1ix+a2iy
181 #rs=[d11^2 d11d12 d11d13 d12^2 d12d13 d13d13]
182 def makeG():
183     f2 = open('Grz.txt', 'w')
184     rs = []
185     for i in range(10):
186         rs.append([])
187         for j in range(10):
188             rs[i].append([])
189
190     for i in range(10):
191         for j in range(10):
192             r = str(expand(p[i]*p[j]))
193             rs[i][j]=IntegrateGrad(r)
194
195     i=0
196     j = 0
197     k = 0
198     for row in rs:
199         for x in row:
200             for y in x:
201                 f2.write(str(i) + " " + str(j) + " " + str(k) + " " + '
↵ '.join(["{0:.20f}".format(float(elem)) for elem in y]) + "\n")

```



```

201         k+=1
202         k=0
203         j+=1
204         i+=1
205         j=0
206         f2.close()
207         return rs
208
209 M = makeM()
210 G=makeG()

```

## graphics.py

```

1  import matplotlib.pyplot as plt
2  import matplotlib.patches as patches
3  import numpy as np
4
5  xL, yL, xAll, yAll = [], [], [], []
6  elements = []
7
8  fig, ax = plt.subplots()
9
10 with open("forGraphics/pointsLinear.txt") as file:
11     for line in file:
12         x, y = line.split()
13         xL.append(float(x))
14         yL.append(float(y))
15
16 with open("forGraphics/allPoints.txt") as file:
17     for line in file:
18         x, y = line.split()
19         xAll.append(float(x))
20         yAll.append(float(y))
21
22 with open("forGraphics/elements.txt") as file:
23     for line in file:
24         vert1, vert2, vert3 = map(int, line.split())
25         elements.append([vert1, vert2, vert3])
26
27 for elem in elements:
28     triangle = patches.Polygon([
29         [xL[elem[0]], yL[elem[0]]],
30         [xL[elem[1]], yL[elem[1]]],
31         [xL[elem[2]], yL[elem[2]]]
32     ],
33         edgecolor='black', facecolor='white', linewidth=2)
34     ax.add_patch(triangle)
35
36 # for i in range(len(xAll)):
37 #     ax.annotate(i, xy = (xAll[i] + 0.01, yAll[i] + 0.01))
38
39 plt.plot(xL, yL, " ")
40 # plt.show()
41 plt.savefig("images/4.eps")

```