

Федеральное государственное бюджетное образовательное учреждение высшего образования «Новосибирский государственный технический университет»



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.

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

Схема

Функция u в четырёхслойной схеме представляется следующим образом:

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

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 $\eta^j_\nu(t)$ являются кубическими полиномами t, причем η^j_3 равна единице при $t=t_{j-3}$ и нулю при $t=t_{j-2}$, $t=t_{j-1}$ и $t=t_j$, функция η^j_2 равна единице при $t=t_{j-2}$ и нулю при $t=t_{j-3}$, $t=t_{j-1}$ и $t=t_j$, функция η^j_1 равна единице при $t=t_{j-1}$ и нулю при $t=t_{j-3}$, $t=t_{j-2}$ и $t=t_j$, а функция η^j_0 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 $\eta^j_
u(t)$ – это базисные кубические полиномы Лагранжа, которые выглядят следующим образом:

$$\eta_{3}^{j}(t) = \frac{(t - t_{j-2})(t - t_{j-1})(t - t_{j})}{(t_{j-3} - t_{j-2})(t_{j-3} - t_{j-1})(t_{j-3} - t_{j})} \quad \eta_{2}^{j}(t) = \frac{(t - t_{j-3})(t - t_{j-1})(t - t_{j})}{(t_{j-2} - t_{j-3})(t_{j-2} - t_{j-1})(t_{j-2} - t_{j})}$$

$$\eta_{1}^{j}(t) = \frac{(t - t_{j-3})(t - t_{j-2})(t - t_{j})}{(t_{j-1} - t_{j-3})(t_{j-1} - t_{j-1})(t_{j-1} - t_{j})} \quad \eta_{0}^{j}(t) = \frac{(t - t_{j-3})(t - t_{j-2})(t - t_{j-1})}{(t_{j-2} - t_{j-1})(t_{j-1} - t_{j-1})}$$
(3

Упростим выражения для более простой записи производных с помощью соотношений:

$$\Delta t_{03} = t_j - t_{j-3} \qquad \Delta t_{02} = t_j - t_{j-2}$$

$$\Delta t_{01} = t_j - t_{j-1} \qquad \Delta t_{13} = t_{j-1} - t_{j-3}$$

$$\Delta t_{12} = t_{j-1} - t_{j-2} \qquad \Delta t_{23} = t_{j-2} - t_{j-3}$$

$$(4)$$

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

$$\frac{\partial \eta_3^j(t)}{\partial t} \bigg|_{t=t_j} = -\frac{\Delta t_{02} \Delta t_{01}}{\Delta t_{23} \Delta t_{13} \Delta t_{03}}$$

$$\frac{\partial \eta_2^j(t)}{\partial t} \bigg|_{t=t_j} = \frac{\Delta t_{03} \Delta t_{01}}{\Delta t_{23} \Delta t_{12} \Delta t_{02}}$$

$$\frac{\partial \eta_1^j(t)}{\partial t} \bigg|_{t=t_j} = -\frac{\Delta t_{03} \Delta t_{02}}{\Delta t_{13} \Delta t_{12} \Delta t_{01}}$$

$$\frac{\partial \eta_0^j(t)}{\partial t} \bigg|_{t=t_j} = \frac{1}{\Delta t_{03}} + \frac{1}{\Delta t_{02}} + \frac{1}{\Delta t_{01}}$$
(5)

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

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

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

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

$$(7)$$

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

$$\left(\left(\frac{1}{\Delta t_{03}} + \frac{1}{\Delta t_{02}} + \frac{1}{\Delta t_{01}} \right) \mathbf{M} + \mathbf{G} \right) q^{j} =$$

$$= b^{j} + \frac{\Delta t_{02} \Delta t_{01}}{\Delta t_{23} \Delta t_{13} \Delta t_{03}} \mathbf{M} q^{j-3} - \frac{\Delta t_{03} \Delta t_{01}}{\Delta t_{23} \Delta t_{12} \Delta t_{02}} \mathbf{M} q^{j-2} + \frac{\Delta t_{03} \Delta t_{02}}{\Delta t_{13} \Delta t_{12} \Delta t_{01}} \mathbf{M} q^{j-1} \tag{8}$$

где ${\bf M}$ – матрица массы, ${\bf G}$ – 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

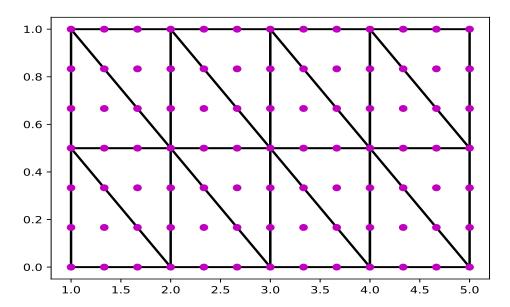
Первый тест

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	$1.93 \cdot 10^{-14}$

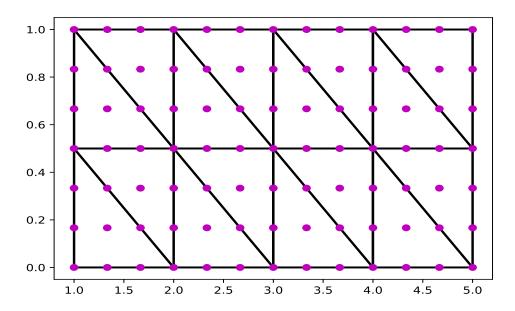
Второй тест

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.



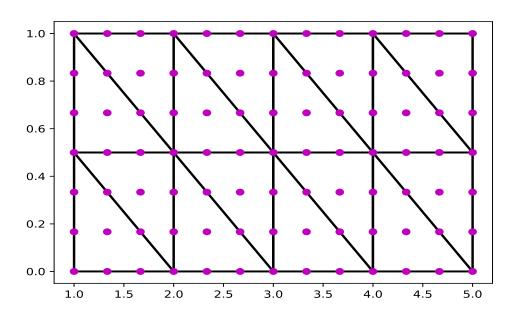
t_i	Погрешность
0.1	$2.19 \cdot 10^{-3}$
0.2	$7.49 \cdot 10^{-4}$
0.3	$1.62 \cdot 10^{-4}$
0.4	$1.46 \cdot 10^{-4}$
0.5	$9.75 \cdot 10^{-6}$
0.6	$3.14 \cdot 10^{-5}$
0.7	$5.76 \cdot 10^{-6}$
0.8	$4.56 \cdot 10^{-6}$
0.9	$1.96 \cdot 10^{-6}$
1	$4.79 \cdot 10^{-7}$

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.



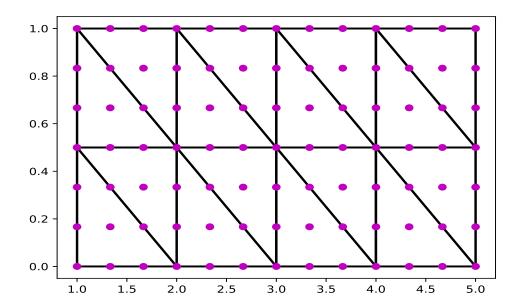
t_i	Погрешность
$1 \cdot 10^{-2}$	$1.05 \cdot 10^{-3}$
$2 \cdot 10^{-2}$	$1.51 \cdot 10^{-3}$
$3 \cdot 10^{-2}$	$1.6 \cdot 10^{-3}$
$4 \cdot 10^{-2}$	$1.56 \cdot 10^{-3}$
$5 \cdot 10^{-2}$	$1.51 \cdot 10^{-3}$
$6 \cdot 10^{-2}$	$1.47 \cdot 10^{-3}$
$7 \cdot 10^{-2}$	$1.45 \cdot 10^{-3}$
$8 \cdot 10^{-2}$	$1.43 \cdot 10^{-3}$
$9 \cdot 10^{-2}$	$1.42 \cdot 10^{-3}$
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

 $\dot{\kappa}$ Коэффициенты : $\lambda=1$, $\sigma=0$

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

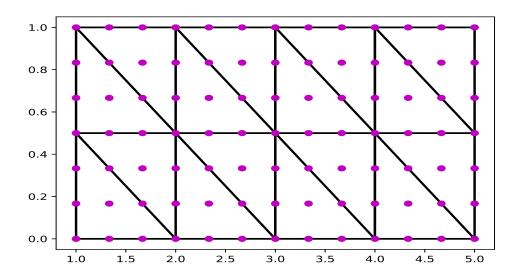


Рис. 1: Сетка с шагом h

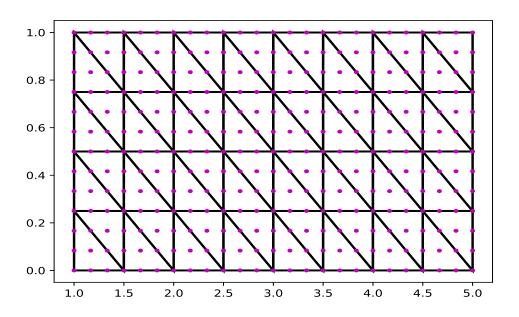


Рис. 2: Сетка с шагом $\hbar/2$

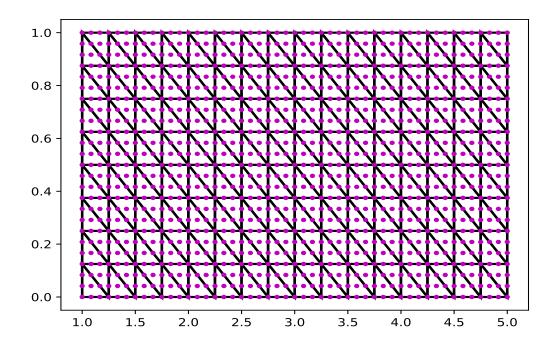


Рис. 3: Сетка с шагом h/4

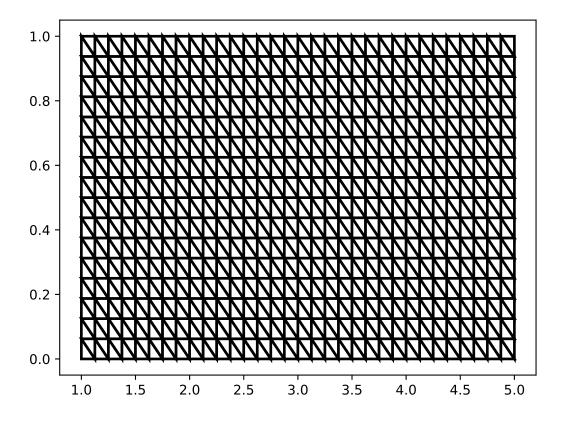


Рис. 4: Сетка с шагом h/8

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

```
using courseProject;
    SpaceGridFactory spaceGridFactory = new();
3
   TimeGridFactory timeGridFactory = new();
4
6
   // FEM fem = FEM.CreateBuilder().SetTest(new Test1())
   // .SetSpaceGrid(spaceGridFactory.CreateGrid(GridTypes.SpaceRegular,
    → SpaceGridParameters.ReadJson("input/spaceGrid.jsonc")!.Value))
    // .SetTimeGrid(timeGridFactory.CreateGrid(GridTypes.TimeRegular,
8
    → TimeGridParameters.ReadJson("input/timeGrid.json")!.Value))
   // .SetSolverSLAE(new CGMCholesky(1000, 1E-14))
    // .SetDiriclhetBoundaries(DirichletBoundary.ReadJson("input/DirichletBoundaries.json|
10

→ ")!)
    // .SetNeumannBoundaries(NeumannBoundary.ReadJson("input/NeumannBoundaries.json")!)
11
   // .IsPhysical(false);
12
13
   FEM fem = FEM.CreateBuilder().SetTest(new Test7())
14
    .SetSpaceGrid(spaceGridFactory.CreateGrid(GridTypes.SpaceRegular,
15
    → SpaceGridParameters.ReadJson("input/spaceGrid.jsonc")!.Value))
    . \\ Set \\ Time \\ Grid \\ Time \\ Regular, \\
16
    → TimeGridParameters.ReadJson("input/timeGrid.json")!.Value))
    .SetSolverSLAE(new CGMCholesky(1000, 1E-14))
17
    .SetDiriclhetBoundaries(DirichletBoundary.ReadJson("input/DirichletBoundaries.json")!)
18
    .IsPhysical(false);
20
    fem.Compute();
21
   fem.WriteToFile("results/q.txt");
22
23
   // var valuesAtPoints = fem.ValueAtPoint(new Point2D[] { new(1.5, 0.5), new (1, 0) });
24
   // Array.ForEach(valuesAtPoints, Console.WriteLine);
```

FEM.cs

```
namespace courseProject;
1
2
    public class FEM {
3
        public class FEMBuilder {
4
            private readonly FEM _fem = new();
5
6
            public FEMBuilder SetTest(ITest test) {
                 _fem._test = test;
8
                 return this;
9
10
11
            public FEMBuilder SetSpaceGrid(ISpaceGrid spaceGrid) {
                 _fem._spaceGrid = spaceGrid;
13
                 return this;
14
            public FEMBuilder SetTimeGrid(ITimeGrid grid) {
17
                 _fem._timeGrid = grid;
18
                 return this;
21
            public FEMBuilder SetSolverSLAE(Solver solver) {
22
                 _fem._solver = solver;
23
```

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

```
Console.WriteLine($\"We had problem: \{ex.Message\}");
80
             }
81
         }
82
83
         private void Init() {
84
             _basis = new Basis[] { CubicBasis.Psi1, CubicBasis.Psi2, CubicBasis.Psi3,
85

→ CubicBasis.Psi4,

             CubicBasis.Psi5, CubicBasis.Psi6, CubicBasis.Psi7, CubicBasis.Psi8,
86

→ CubicBasis.Psi9, CubicBasis.Psi10};

87
             _stiffnessMatrix = new(10);
88
             _{\text{massMatrix}} = \text{new}(10);
89
             _massMatrixCopy = new(10);
             _{alphas} = new(3);
91
92
             _vertices = new Point2D[3];
93
94
             _layers = new double[3].Select(_ => new
95
             → double[_spaceGrid.Points.Count]).ToArray();
96
             _{M} = new double[10].Select(_ => new double[10].ToArray().
97
                               Select(_ => new double[3]).ToArray()).ToArray();
98
99
             _{G} = new double[10].Select(_ => new double[10].ToArray().
100
                              Select(_ => new double[6].ToArray().
101
                              Select(_ => new double[3]).ToArray()).ToArray()).ToArray();
102
103
             using StreamReader sr1 = new("input/Grz.txt"), sr2 = new("input/Mrz.txt");
             string[] vars;
105
106
             for (int i = 0; i < 600; i++) {
107
                 vars = sr1.ReadLine()!.Split(" ").ToArray();
108
                 _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][0] =
110

→ double.Parse(vars[3]);

                 _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][1] =
111
                  → double.Parse(vars[4]);
                 G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][2] =
112
                  → double.Parse(vars[5]);
             }
114
             for (int i = 0; i < 100; i++) {
115
                 vars = sr2.ReadLine()!.Split(" ").ToArray();
116
117
                 _M[int.Parse(vars[0])][int.Parse(vars[1])][0] = double.Parse(vars[2]);
118
                 _M[int.Parse(vars[0])][int.Parse(vars[1])][1] = double.Parse(vars[3]);
110
                 _M[int.Parse(vars[0])][int.Parse(vars[1])][2] = double.Parse(vars[4]);
             }
121
122
123
         private void Prepare() {
124
125
             if (!IsPhysical) {
                 for (int i = 0; i < _spaceGrid.Points.Count; i++) {</pre>
126
                     _layers[0][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[0]);
127
                      _layers[1][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[1]);
128
                     _layers[2][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[2]);
129
130
             } else {
131
                 for (int i = 0; i < _spaceGrid.Points.Count; i++) {</pre>
132
                      _layers[0][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[0]);
133
```

```
134
135
                 double t01 = _timeGrid.Points[1] - _timeGrid.Points[0];
136
137
                 AssemblyGlobalMatrix(1, t01: t01);
138
139
                 if (_neumannBoundaries is not null) {
140
                      AccountingNeumannBoundaries();
141
                  }
143
                 AccountingDirichletBoundaries(1);
144
145
                 _solver.SetMatrix(_globalMatrix);
146
                 _solver.SetVector(_vector);
147
                  _solver.Compute();
148
                 Err(1);
149
150
                  _solver.Solution!.Value.CopyTo(0, _layers[1], 0, _layers[1].Length);
151
152
153
                 _globalMatrix.Clear();
                 _vector.Fill(∅);
154
155
                 double t02 = _timeGrid.Points[2] - _timeGrid.Points[0];
156
                 double t12 = _timeGrid.Points[1] - _timeGrid.Points[0];
                 t01 = _timeGrid.Points[2] - _timeGrid.Points[1];
158
159
                 AssemblyGlobalMatrix(2, t02: t02, t01: t01, t12: t12);
160
161
                  if (_neumannBoundaries is not null) {
162
                      AccountingNeumannBoundaries();
163
164
165
                 AccountingDirichletBoundaries(2);
167
                 _solver.SetMatrix(_globalMatrix);
168
                 _solver.SetVector(_vector);
169
                  _solver.Compute();
170
                 Err(2);
171
172
                  _solver.Solution!.Value.CopyTo(0, _layers[2], 0, _layers[2].Length);
173
174
                 _globalMatrix.Clear();
175
                 _vector.Fill(0);
176
             }
177
         }
178
170
         private void Solve() {
180
             for (int itime = 3; itime < _timeGrid.Points.Length; itime++) {</pre>
181
                  double t03 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 3];
182
                 double t02 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 2];
183
                 double t01 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 1];
184
185
                 double t13 = _timeGrid.Points[itime - 1] - _timeGrid.Points[itime - 3];
                 double t12 = _timeGrid.Points[itime - 1] - _timeGrid.Points[itime - 2];
186
                 double t23 = _timeGrid.Points[itime - 2] - _timeGrid.Points[itime - 3];
187
188
                 AssemblyGlobalMatrix(itime, t03, t02, t01, t13, t12, t23);
189
190
                 if (_neumannBoundaries is not null) {
191
                      AccountingNeumannBoundaries();
192
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
                  _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
200
         }
211
         private void ConstructPortrait() {
212
              List<<mark>int</mark>>[] list = new List<<mark>int</mark>>[_spaceGrid.Points.Count].Select(_ => new
213

→ List(int)()).ToArray();
214
              foreach (var element in _spaceGrid.Elements.Select(array =>
215
                  array.OrderBy(value => value).ToArray()).ToArray()) {
                   for (int i = 0; i < element.Length; <math>i++) {
                       for (int j = i + 1; j < element.Length; <math>j++) {
217
                            int pos = element[j];
218
                           int elem = element[i];
220
                            if (!list[pos].Contains(elem)) {
221
                                list[pos].Add(elem);
222
                            }
223
                       }
                  }
225
226
227
              list = list.Select(list => list.OrderBy(value => value).ToList()).ToArray();
              int count = list.Sum(childList => childList.Count);
229
230
              InitSLAE(count);
231
232
              _globalMatrix.ig[0] = 0;
233
234
              for (int i = 0; i < list.Length; i++)</pre>
235
                  _globalMatrix.ig[i + 1] = _globalMatrix.ig[i] + list[i].Count;
236
237
              int k = 0;
238
239
              for (int i = 0; i < list.Length; i++) {
240
                  for (int j = 0; j < list[i].Count; j++) {</pre>
241
                       _globalMatrix.jg[k] = list[i][j];
242
243
                       k++;
                  }
244
              }
245
         }
246
247
         private void InitSLAE(int count) {
248
              _globalMatrix = new(_spaceGrid.Points.Count, count);
249
              _vector = new(_spaceGrid.Points.Count);
250
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 \rightarrow j) {
259
                  for (int index = _globalMatrix.ig[i]; index < _globalMatrix.ig[i + 1];</pre>
                     index++) {
                      if (_globalMatrix.jg[index] == j) {
261
                           _globalMatrix.gg[index] += value;
262
263
                          return;
                      }
                  }
265
             }
266
268
         private void AssemblyGlobalMatrix(int itime = 1, double t03 = 1, double t02 = 1,
269
         \rightarrow double t01 = 1, double t13 = 1, double t12 = 1, double t23 = 1) {
             for (int ielem = 0; ielem < _spaceGrid.Elements.Length; ielem++) {</pre>
                  _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
271
                  _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
272
                  _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];
274
                  AssemblyLocalMatrices(itime, t03, t02, t01);
275
276
                  _stiffnessMatrix += _massMatrix;
278
                  for (int i = 0; i < 10; i++) {
279
                      for (int j = 0; j < 10; j++) {
280
                          AddElement(_spaceGrid.Elements[ielem][i],
281

    _spaceGrid.Elements[ielem][j], _stiffnessMatrix[i, j]);

282
                  }
283
284
                  AssemblyGlobalVector(itime, ielem, t03, t02, t01, t13, t12, t23);
285
286
                  _stiffnessMatrix.Clear();
287
                  _massMatrix.Clear();
288
             }
289
         }
290
291
         private void AssemblyGlobalVector(int itime = 1, int ielem = 1, double t03 = 1,
292
         \rightarrow double t02 = 1, double t01 = 1, double t13 = 1, double t12 = 1, double t23 =
             1) {
             double[] qj1 = new double[10];
             double[] qj2 = new double[10];
294
             double[] qj3 = new double[10];
295
296
             for (int i = 0; i < 10; i++) {
297
298
                  for (int j = 0; j < 10; j++) {
                      if (itime == 1) {
299
                          qj1[i] += _massMatrixCopy[i, j] *
300
                           → _layers[0][_spaceGrid.Elements[ielem][j]];
                      } else if (itime == 2) {
301
                          qj2[i] += _massMatrixCopy[i, j] *
302
                           → _layers[0][_spaceGrid.Elements[ielem][j]];
                          qj1[i] += _massMatrixCopy[i, j] *
                           → _layers[1][_spaceGrid.Elements[ielem][j]];
```

```
} else {
304
                           qj3[i] += _massMatrixCopy[i, j] *
305
                            → _layers[0][_spaceGrid.Elements[ielem][j]];
                           qj2[i] += _massMatrixCopy[i, j] *
306
                            → _layers[1][_spaceGrid.Elements[ielem][j]];
                           qj1[i] += _massMatrixCopy[i, j] *
307
                            → _layers[2][_spaceGrid.Elements[ielem][j]];
                       }
308
                  }
              }
310
311
              for (int j = 0; j < 10; j++) {
312
                  if (itime == 1) {
                       _vector[_spaceGrid.Elements[ielem][j]] +=
314

    Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
                          _timeGrid.Points[2]) -
                                                                  (-1.0 / t01 * qj1[j]);
315
                   } else if (itime == 2) {
316
                       _vector[_spaceGrid.Elements[ielem][j]] +=
317

→ Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
                          _timeGrid.Points[2]) -
                                                    (t01 / (t02 * t12) * qj2[j]) + (t02 / (t12)
318
                                                     → * t01) * qj1[j]);
                  } else {
319
                       _vector[_spaceGrid.Elements[ielem][j]] +=

→ Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,

    _timeGrid.Points[itime]) +

                                                                      (t02 * t01 / (t23 * t13 *
                                                                       \rightarrow t03) * qj3[j]) -
                                                                       (t03 * t01 / (t23 * t12 *
322
                                                                       \rightarrow t02) * qj2[j]) +
                                                                       (t03 * t02 / (t13 * t12 *
323
                                                                       \rightarrow t01) * qj1[j]);
                  }
324
325
326
327
         private double DeterminantD()
328
              \Rightarrow ((_vertices[1].R - _vertices[0].R) * (_vertices[2].Z - _vertices[0].Z)) -
                 ((\_vertices[2].R - \_vertices[0].R) * (\_vertices[1].Z - \_vertices[0].Z));
330
331
         private void CalcAlphas() {
332
              double dD = DeterminantD();
333
334
              _{alphas}[0, 0] = ((_{vertices}[1].R * _{vertices}[2].Z) - (_{vertices}[2].R *
335
              \rightarrow _vertices[1].Z)) / dD;
              _{alphas}[0, 1] = (_{vertices}[1].Z - _{vertices}[2].Z) / dD;
336
              _{alphas}[0, 2] = (_{vertices}[2].R - _{vertices}[1].R) / dD;
337
338
              _{alphas[1, 0]} = ((_{vertices[2].R} * _{vertices[0].Z}) - (_{vertices[0].R} * _{vertices[0].R})
339
              → _vertices[2].Z)) / dD;
340
              _{alphas[1, 1]} = (_{vertices[2].Z} - _{vertices[0].Z}) / dD;
              _{alphas}[1, 2] = (_{vertices}[0].R - _{vertices}[2].R) / dD;
341
342
              _{alphas[2, \emptyset]} = ((_{vertices[\emptyset]}.R * _{vertices[1].Z}) - (_{vertices[1]}.R *
343
              → _vertices[0].Z)) / dD;
              _{alphas[2, 1]} = (_{vertices[0].Z} - _{vertices[1].Z}) / dD;
344
              _{alphas[2, 2]} = (_{vertices[1].R} - _{vertices[0].R}) / dD;
345
         }
346
347
```

```
private void AssemblyLocalMatrices(int itime, double t03, double t02, double t01)
348
              double dD = Math.Abs(DeterminantD());
349
             CalcAlphas();
350
351
             //rs=[d11^2 d11d12 d11d13 d12^2 d12d13 d13d13]
352
             //каждая из 6 пар дифференциалов состоит из 3 составляющих (т.к. вектор
353
              \rightarrow r=r1L1+r2L2+r3L3)
              for (int i = 0; i < 10; i++) {
354
                  for (int j = 0; j < 10; j++) {
355
                       \_stiffnessMatrix[i, j] += \_G[i][j][0][0] * \_vertices[0].R *
356
                       \rightarrow _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) +
                       \rightarrow (_alphas[0, 2] * _alphas[0, 2])) * dD;
                       _{\text{stiffnessMatrix}[i, j] += _G[i][j][0][1] * _{\text{vertices}[1].R} *
357
                          \_spaceGrid.Lambda * ((\_alphas[0, 1] * \_alphas[0, 1]) +
                       \rightarrow (_alphas[0, 2] * _alphas[0, 2])) * dD;
                       _{stiffnessMatrix[i, j] += _G[i][j][0][2] * _vertices[2].R *}
358
                       \rightarrow _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) +
                           (alphas[0, 2] * alphas[0, 2])) * dD;
359
                       _{\text{stiffnessMatrix}[i, j] += _G[i][j][1][0] * _{\text{vertices}[0].R} *
360
                       \rightarrow _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) +
                          (_alphas[0, 2] * _alphas[1, 2])) * dD;
                       \_stiffnessMatrix[i, j] += \_G[i][j][1][1] * \_vertices[1].R *
361
                       \rightarrow _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) +
                           (alphas[0, 2] * alphas[1, 2])) * dD;
                       _{\text{stiffnessMatrix}[i, j] += _G[i][j][1][2] * _{\text{vertices}[2].R} *
362
                       \rightarrow _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) +
                          (alphas[0, 2] * alphas[1, 2])) * dD;
363
                       _{\text{stiffnessMatrix[i, j]}} += _{G[i][j][2][0]} * _{\text{vertices[0]}.R} *
364
                       \rightarrow _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[2, 1]) +
                          (alphas[0, 2] * alphas[2, 2])) * dD;
                       \_stiffnessMatrix[i, j] += \_G[i][j][2][1] * \_vertices[1].R *
365
                           \_spaceGrid.Lambda * ((\_alphas[0, 1] * \_alphas[2, 1]) +
                          (_alphas[0, 2] * _alphas[2, 2])) * dD;
                       _{\text{stiffnessMatrix}[i, j] += _G[i][j][2][2] * _{\text{vertices}[2].R} *
366
                           \_spaceGrid.Lambda * ((\_alphas[0, 1] * \_alphas[2, 1]) +
                           (alphas[0, 2] * alphas[2, 2])) * dD;
                       _{stiffnessMatrix[i, j]} += _{G[i][j][3][0]} * _{vertices[0].R} *
368
                           \_spaceGrid.Lambda * ((\_alphas[1, 1] * \_alphas[1, 1]) +
                          (alphas[1, 2] * alphas[1, 2])) * dD;
                       _{stiffnessMatrix[i, j]} += _{G[i][j][3][1]} * _{vertices[1].R} *
                       \rightarrow _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) +
                       \rightarrow (_alphas[1, 2] * _alphas[1, 2])) * dD;
                       \_stiffnessMatrix[i, j] += \_G[i][j][3][2] * \_vertices[2].R *
                       \rightarrow _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) +
                           (_alphas[1, 2] * _alphas[1, 2])) * dD;
371
                       _{\text{stiffnessMatrix}[i, j] += _G[i][j][4][0] * _{\text{vertices}[0].R} *
372
                       \rightarrow _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) +
                          (_alphas[1, 2] * _alphas[2, 2])) * dD;
                       \_stiffnessMatrix[i, j] += \_G[i][j][4][1] * \_vertices[1].R *
373
                       \rightarrow _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) +
                           (_alphas[1, 2] * _alphas[2, 2])) * dD;
                       \_stiffnessMatrix[i, j] += \_G[i][j][4][2] * \_vertices[2].R *
374
                       \rightarrow _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) +
                          (alphas[1, 2] * alphas[2, 2])) * dD;
375
```

```
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;
                       _stiffnessMatrix[i, j] += _G[i][j][5][1] * _vertices[1].R *
377
                          \_spaceGrid.Lambda * ((\_alphas[2, 1] * \_alphas[2, 1]) +
                          (alphas[2, 2] * alphas[2, 2])) * dD;
                       _stiffnessMatrix[i, j] += _G[i][j][5][2] * _vertices[2].R *
378
                          \_spaceGrid.Lambda * ((\_alphas[2, 1] * \_alphas[2, 1]) +
                          (_alphas[2, 2] * _alphas[2, 2])) * dD;
                  }
379
              }
380
381
              for (int i = 0; i < 10; i++) {
382
                  for (int j = 0; j < 10; j++) {
383
                      for (int k = 0; k < 3; k++) {
384
                           _{\mathsf{massMatrix}[i, j] += _{\mathsf{M}[i][j][k] * _{\mathsf{vertices}[k].R *}}
385

→ _spaceGrid.Sigma * dD;

386
                  }
387
388
389
             _massMatrix.Copy(_massMatrixCopy);
390
391
             if (itime == 1) {
                  for (int i = 0; i < 10; i++) {
393
                      for (int j = 0; j < 10; j++) {
394
                           _massMatrix[i, j] *= 1.0 / t01;
395
396
397
              } else if (itime == 2) {
398
                  for (int i = 0; i < 10; i++) {
399
                      for (int j = 0; j < 10; j++) {
                           _massMatrix[i, j] *= (t02 + t01) / (t02 * t01);
401
402
403
              } else {
                  for (int i = 0; i < 10; i++) {
405
                      for (int j = 0; j < 10; j++) {
406
                           _massMatrix[i, j] *= (1.0 / t03) + (1.0 / t02) + (1.0 / t01);
                  }
409
              }
410
411
         private void AccountingDirichletBoundaries(int itime) {
413
              (int Node, double Value)[] boundaries = new (int, double)[4 *
414

→ _dirichletBoundaries.Length];
              int[] checkBC = new int[_spaceGrid.Points.Count];
415
416
              int index = 0;
417
418
              for (int i = 0; i < \_dirichletBoundaries.Length; <math>i++) {
419
                  int ielem = _dirichletBoundaries[i].Element;
420
                  int edge = _dirichletBoundaries[i].Edge;
421
                  for (int j = 0; j < 4; j++) {
423
                      boundaries[index++] = (_spaceGrid.Edges[ielem][edge][j],
424
                                               _test.U(_spaceGrid.Points[_spaceGrid.Edges[iele_
425
                                               \rightarrow m][edge][j]],
                                                   _timeGrid.Points[itime]));
```

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

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

    _vertices[0].Z)) -
```

```
((_vertices[1].R - _vertices[0].R) * (_vertices[0].Z - point.Z));
539
                  double b = ((_vertices[1].R - point.R) * (_vertices[2].Z -
541

    _vertices[1].Z)) -

                            (_vertices[2].R - (_vertices[1].R * (_vertices[1].Z - point.Z)));
542
543
                  double c = ((_vertices[2].R - point.R) * (_vertices[0].Z -
544
                  \rightarrow _vertices[2].Z)) -
                            ((\_vertices[0].R - \_vertices[2].R) * (\_vertices[2].Z - point.Z));
545
546
                  if ((a >= 0 && b >= 0 && c >= 0) || (a <= 0 && b <= 0 && c <= 0))
547
                      return ielem;
548
             }
550
             throw new Exception("The point does not belong to the grid");
551
         }
552
553
         public void WriteToFile(string path) {
554
             using var sw = new StreamWriter(path);
555
             for (int i = 0; i < _solver.Solution!.Value.Length; i++) {</pre>
556
                  sw.WriteLine(i + ") " + _solver.Solution!.Value[i]);
557
558
         }
559
560
         // for report
         private void Err(int itime) {
562
             double[] err = new double[_spaceGrid.Points.Count];
563
             double[] exact = new double[_spaceGrid.Points.Count];
564
             double sum = 0.0;
565
566
             using StreamWriter sw1 = new("results/err.txt"),
567
568
                                  sw2 = new("results/Exact.txt"),
                                  sw3 = new("results/rms.txt"),
569
                                  sw4 = new("csv2/2.csv", true);
570
571
             for (int i = 0; i < _spaceGrid.Points.Count; i++) {</pre>
572
                  err[i] = Math.Abs(_solver.Solution!.Value[i] -
573

    _test.U(_spaceGrid.Points[i], _timeGrid.Points[itime]));

                  sw1.WriteLine(err[i]);
574
             }
575
576
             for (int i = 0; i < _spaceGrid.Points.Count; i++) {</pre>
577
                  exact[i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[itime]);
578
                  sw2.WriteLine(exact[i]);
579
580
581
             for (int i = 0; i < _spaceGrid.Points.Count; i++) {</pre>
582
                  sum += err[i] * err[i];
583
584
585
             // sum = Math.Sqrt(sum); // относительная погрешность
586
587
             // sum /= exact.Norm();
588
             sum = Math.Sqrt(sum / _spaceGrid.Points.Count); // среднеквадратичное
589
              ⊶ отклонение
             sw3.Write(sum);
590
591
             // sw4.WriteLine("$r $z, Точное, Численное, Вектор погрешности, Погрешность");
592
593
             // for (int i = 0; i < _spaceGrid.Points.Count; i++) {</pre>
594
```

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

SparseMatrix.cs

```
namespace courseProject;
1
2
    public class SparseMatrix {
3
        // public fields - its bad, but the readability is better
4
        public int[] ig = default!;
5
6
        public int[] jg = default!;
        public double[] di = default!;
        public double[] gg = default!;
8
        public int Size { get; init; }
9
        public SparseMatrix(int size, int sizeOffDiag) {
11
            Size = size;
12
            ig = new int[size + 1];
13
            jg = new int[sizeOffDiag];
            gg = new double[sizeOffDiag];
15
            di = new double[size];
16
17
18
        public static Vector<double> operator *(SparseMatrix matrix, Vector<double>
19
         → vector) {
            Vector (double) product = new(vector.Length);
21
            for (int i = 0; i < vector.Length; i++) {</pre>
22
                 product[i] = matrix.di[i] * vector[i];
                 for (int j = matrix.ig[i]; j < matrix.ig[i + 1]; j++) {</pre>
25
                     product[i] += matrix.gg[j] * vector[matrix.jg[j]];
26
                     product[matrix.jg[j]] += matrix.gg[j] * vector[i];
27
                 }
28
29
30
            return product;
31
```

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

Matrix.cs

```
namespace courseProject;
2
    public class Matrix {
3
        private readonly double[,] storage;
4
        public int Size { get; init; }
5
6
        public double this[int i, int j] {
7
            get => storage[i, j];
8
            set => storage[i, j] = value;
9
10
11
        public Matrix(int size) {
12
             storage = new double[size, size];
13
            Size = size;
14
        }
15
        public void Clear()
17
            ⇒ Array.Clear(storage, ∅, storage.Length);
18
19
        public void Copy(Matrix destination) {
20
             for (int i = 0; i < destination.Size; i++) {</pre>
21
                 for (int j = 0; j < destination.Size; j++) {</pre>
22
                     destination[i, j] = storage[i, j];
23
```

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

Vector.cs

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

```
42
43
             return result;
44
45
46
        public static Vector<T> operator -(Vector<T> firstVec, Vector<T> secondVec) {
47
            Vector<T> result = new(firstVec.Length);
48
             for (int i = 0; i < firstVec.Length; i++) {</pre>
                 result.vec[i] = firstVec[i] - secondVec[i];
51
52
53
            return result;
        }
55
56
        public static void Copy(Vector(T) source, Vector(T) destination) {
57
             for (int i = 0; i < source.Length; i++) {
                 destination[i] = source[i];
59
60
        }
61
62
        public void Fill(double value) {
63
             for (int i = 0; i < Length; i++) {
64
                 vec[i] = T.Create(value);
65
        }
67
68
        public double Norm() {
69
            T result = T.Zero;
70
71
             for (int i = \emptyset; i < Length; i++) {
72
                 result += vec[i] * vec[i];
73
74
75
            return Math.Sqrt(Convert.ToDouble(result));
76
77
78
        public ImmutableArray(T> ToImmutableArray()
79
            => ImmutableArray.Create(vec);
80
81
```

Solvers.cs

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

```
=> _vector = vector;
18
19
        public abstract void Compute();
20
21
22
    public class LOS : Solver {
23
        public LOS(int maxIters, double eps) : base(maxIters, eps) { }
24
25
        public override void Compute() {
26
             try {
27
                 ArgumentNullException.ThrowIfNull(_matrix, $\"\nameof(_matrix)\} cannot be
28
                  → null, set the matrix");
                 ArgumentNullException.ThrowIfNull(_vector, $\bar{\star} \nameof(_vector)\} cannot be
                  → null, set the vector");
30
                 double alpha, beta;
31
                 double squareNorm;
33
                 _solution = new(_vector.Length);
34
35
                 Vector (double) r = new(_vector.Length);
36
                 Vector < double > z = new(_vector.Length);
37
                 Vector < double > p = new(_vector.Length);
38
                 Vector (double) tmp = new(_vector.Length);
                 r = _vector - (_matrix * _solution);
41
42
                 Vector < double > . Copy(r, z);
44
                 p = _matrix * z;
45
46
                 squareNorm = r * r;
                 for (int iter = 0; iter < MaxIters && squareNorm > Eps; iter++) {
49
                     alpha = p * r / (p * p);
50
                     \_solution += alpha * z;
51
                     squareNorm = (r * r) - (alpha * alpha * (p * p));
52
                     r = alpha * p;
53
                     tmp = _matrix * r;
56
                     beta = -(p * tmp) / (p * p);
57
                     z = r + (beta * z);
58
                     p = tmp + (beta * p);
59
60
             } catch (Exception ex) {
61
                 Console.WriteLine($\big|"We had problem: \{ex.Message\}");
62
63
64
65
66
67
    public class CGM : Solver {
68
        public CGM(int maxIters, double eps) : base(maxIters, eps) { }
69
        public override void Compute() {
             try {
71
                 ArgumentNullException.ThrowIfNull(_matrix, $\"\nameof(_matrix)\} cannot be
72
                  → null, set the matrix");
                 ArgumentNullException.ThrowIfNull(_vector, $\\[ \frac{1}{3} \] \"\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) >=
                  \rightarrow Eps; iter++) {
                      tmp = _matrix * z;
91
                      alpha = r * r / (tmp * z);
92
                      \_solution += alpha * z;
93
                      squareNorm = r * r;
94
                      r = alpha * tmp;
95
                      beta = r * r / squareNorm;
96
                      z = r + (beta * z);
97
                  }
98
             } catch (Exception ex) {
99
                  Console.WriteLine($\"We had problem: \{ex.Message\}\");
100
101
         }
102
103
104
    public class CGMCholesky : Solver {
105
         public CGMCholesky(int maxIters, double eps) : base(maxIters, eps) { }
106
107
108
         public override void Compute() {
             try {
109
                  double alpha, beta;
110
                  double tmp;
111
                  double vectorNorm = _vector.Norm();
113
114
                  _solution = new(_vector.Length);
115
116
                  double[] ggnew = new double[_matrix.gg.Length];
117
                  double[] dinew = new double[_matrix.di.Length];
118
119
                  _matrix.gg.Copy(ggnew);
120
                  _matrix.di.Copy(dinew);
121
122
                  Vector < double > r = new(\_vector.Length);
123
                  Vector (double) z = new(_vector.Length);
                  Vector<double> fstTemp = new(_vector.Length);
125
                  Vector < double > sndTemp = new(_vector.Length);
126
127
                  Cholesky(ggnew, dinew);
129
                  r = _vector - (_matrix * _solution);
130
                  z = MoveForCholesky(r, ggnew, dinew);
131
132
```

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

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

Integration.cs

```
namespace courseProject;
1
2
    public static class Integration {
3
        public static double GaussSegment(Func<Point2D, double> psi, Point2D firstPoint,
4
        → Point2D secondPoint) {
            var quadratures = Quadratures.SegmentGaussOrder9();
5
6
            double result = 0.0;
7
            double lenghtEdge = Math.Sqrt(((firstPoint.R - secondPoint.R) * (firstPoint.R
8
             → - secondPoint.R)) +
                                  ((firstPoint.Z - secondPoint.Z) * (firstPoint.Z -
9

    secondPoint.Z)));
10
            foreach (var q in quadratures) {
11
                 double qi = q.Weight;
12
                double pi = (1 + q.Node) / 2.0;
13
                var parameterized = Parameterization(pi);
15
16
                result += qi * psi(parameterized) * parameterized.R;
17
            }
18
19
            return result * lenghtEdge / 2.0;
20
21
            Point2D Parameterization(double t)
22
                => ((secondPoint- firstPoint) * t) + firstPoint;
23
        }
24
25
```

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

Quadratures.cs

```
namespace courseProject;
    public class QuadratureNode<T> where T : notnull {
3
        public T Node { get; init; }
4
        public double Weight { get; init; }
5
6
        public QuadratureNode(T node, double weight) {
            Node = node;
8
            Weight = weight;
        }
10
    }
11
12
    public class Quadrature <T> where T : notnull {
13
        private readonly QuadratureNode<T>[] _nodes = default!;
14
        public ImmutableArray<QuadratureNode<T>> Nodes => _nodes.ToImmutableArray();
15
16
        public Quadrature(QuadratureNode<T>[] nodes) {
17
            _nodes = nodes;
18
        }
19
20
21
    public static class Quadratures {
22
        public static IEnumerable<QuadratureNode<double>>> SegmentGaussOrder9() {
23
            const int n = 5;
            double[] points = { 0.0,
25
                                 1.0 / 3.0 * Math.Sqrt(5 - (2 * Math.Sqrt(10.0 / 7.0))),
26
                                 -1.0 / 3.0 * Math.Sqrt(5 - (2 * Math.Sqrt(10.0 / 7.0))),
                                 1.0 / 3.0 * Math.Sqrt(5 + (2 * Math.Sqrt(10.0 / 7.0))),
                                 -1.0 / 3.0 * Math.Sqrt(5 + (2 * Math.Sqrt(10.0 / 7.0))));
29
30
            double[] weights = { 128.0 / 225.0,
31
                                 (322.0 + (13.0 * Math.Sqrt(70.0))) / 900.0,
                                 (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
```

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

CubicBasis.cs

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

```
31
32
        public static double Psi4(Point2D point) {
33
             double 11 = L1(point);
34
             double 12 = L2(point);
35
            return 4.5 * 11 * 12 * ((3 * 11) - 1);
36
        }
37
38
        public static double Psi5(Point2D point) {
39
             double 11 = L1(point);
40
             double 12 = L2(point);
41
            return 4.5 * 11 * 12 * ((3 * 12) - 1);
42
43
44
        public static double Psi6(Point2D point) {
45
             double 12 = L2(point);
46
             double 13 = L3(point);
47
            return 4.5 * 12 * 13 * ((3 * 12) - 1);
48
49
50
        public static double Psi7(Point2D point) {
51
             double 12 = L2(point);
52
             double 13 = L3(point);
53
            return 4.5 * 12 * 13 * ((3 * 13) - 1);
        }
55
56
        public static double Psi8(Point2D point) {
57
             double 11 = L1(point);
58
             double 13 = L3(point);
59
            return 4.5 * 11 * 13 * ((3 * 13) - 1);
60
61
        }
62
        public static double Psi9(Point2D point) {
63
             double 11 = L1(point);
64
             double 13 = L3(point);
65
            return 4.5 * 11 * 13 * ((3 * 11) - 1);
66
        }
67
68
        public static double Psi10(Point2D point) {
69
             double 11 = L1(point);
             double 12 = L2(point);
71
             double 13 = L3(point);
72
            return 27 * 11 * 12 * 13;
73
        }
74
75
```

Boundaries.cs

```
namespace courseProject;
   1
   2
                             public readonly record struct DirichletBoundary(int Element, int Edge) {
   3
                                                             public static DirichletBoundary[]? ReadJson(string jsonPath) {
   4
                                                                                            try {
   5
                                                                                                                             if (!File.Exists(jsonPath))
   6
                                                                                                                                                            throw new Exception("File does not exist");
   8
                                                                                                                            var sr = new StreamReader(jsonPath);
                                                                                                                             using (sr) {
10
                                                                                                                                                           \textbf{return} \  \, \textbf{JsonConvert.DeserializeObject} \land \textbf{DirichletBoundary[]} \land (\texttt{sr.ReadToEn} \ | \ \textbf{St.ReadToEn} 
 11
                                                                                                                                                                \rightarrow d());
```

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

Tests.cs

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

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

GridsParameters.cs

```
namespace courseProject;
1
2
    public class SpaceGridParametersJsonConverter : JsonConverter {
3
        public override void WriteJson(JsonWriter writer, object? value, JsonSerializer
4

    serializer) {
            if (value is null) {
5
                writer.WriteNull();
6
                return;
            }
8
            var gridParameters = (SpaceGridParameters)value;
10
11
            writer.WriteStartObject();
12
            writer.WritePropertyName("Initial area in R");
            serializer.Serialize(writer, gridParameters.IntervalR);
14
            writer.WritePropertyName("Splits by R");
15
            writer.WriteValue(gridParameters.SplitsR);
16
            writer.WriteWhitespace("\n");
17
18
            writer.WritePropertyName("Initial area in Z");
10
            serializer.Serialize(writer, gridParameters.IntervalZ);
20
            writer.WritePropertyName("Splits by Z");
21
            writer.WriteValue(gridParameters.SplitsZ);
22
23
            writer.WriteComment("Коэффициент разрядки");
24
```

```
writer.WritePropertyName("Coef");
25
            writer.WriteValue(gridParameters.K);
26
            writer.WriteWhitespace("\n");
27
            writer.WriteComment("Uniform area");
28
            writer.WritePropertyName("Lambda");
29
            writer.WriteValue(gridParameters.Lambda);
30
            writer.WritePropertyName("Sigma");
31
            writer.WriteValue(gridParameters.Sigma);
32
33
            writer.WriteEndObject();
        }
34
35
        public override object? ReadJson(JsonReader reader, Type objectType, object?
36

→ existingValue, JsonSerializer serializer) {
            if (reader.TokenType == JsonToken.Null || reader.TokenType !=
37
             → JsonToken.StartObject)
                 return null;
38
39
            Interval intervalR;
40
            Interval intervalZ;
41
            int splitsR;
42
            int splitsZ;
43
            double? coef;
44
            double lambda, sigma;
45
46
            var maintoken = JObject.Load(reader);
47
48
            var token = maintoken["Initial area in R"];
49
            intervalR = serializer.Deserialize<Interval>(token!.CreateReader());
            token = maintoken["Splits by R"];
51
            splitsR = Convert.ToInt32(token);
52
53
            token = maintoken["Initial area in Z"];
54
            intervalZ = serializer.Deserialize<Interval>(token!.CreateReader());
55
            token = maintoken["Splits by Z"];
56
            splitsZ = Convert.ToInt32(token);
57
58
            token = maintoken["Coef"];
59
60
            if (token is not null) {
61
                coef = double.TryParse(token.ToString(), out double res) ? res : null;
63
            else {
64
                 coef = null;
65
            }
66
67
            token = maintoken["Lambda"];
68
            lambda = Convert.ToDouble(token);
69
70
            token = maintoken["Sigma"];
71
72
            if (token is not null) {
73
74
                 sigma = double.TryParse(token.ToString(), out double res) ? res : 0.0;
75
            else {
76
                 sigma = 0.0;
78
79
            return new SpaceGridParameters(intervalR, splitsR, intervalZ, splitsZ, coef,
80
             → lambda, sigma);
```

```
82
         public override bool CanConvert(Type objectType)
83
             => objectType == typeof(SpaceGridParameters);
84
85
86
    [JsonConverter(typeof(SpaceGridParametersJsonConverter))]
87
    public readonly record struct SpaceGridParameters {
88
         public Interval IntervalR { get; init; }
80
         public int SplitsR { get; init; }
         public Interval IntervalZ { get; init; }
91
         public int SplitsZ { get; init; }
92
         public double? K { get; init; }
93
         // Uniform area
95
         public double Lambda { get; init; }
96
         public double Sigma { get; init; }
97
         public SpaceGridParameters(Interval intervalR, int splitsR, Interval intervalZ,
99
            int splitsZ, double? k, double lambda, double sigma) {
             IntervalR = intervalR;
100
             SplitsR = splitsR;
101
             IntervalZ = intervalZ;
102
             SplitsZ = splitsZ;
103
             K = k;
             Lambda = lambda;
105
             Sigma = sigma;
106
107
108
         public static SpaceGridParameters? ReadJson(string jsonPath) {
109
110
             try {
                 if (!File.Exists(jsonPath))
111
                      throw new Exception("File does not exist");
112
113
                 var sr = new StreamReader(jsonPath);
114
                 using (sr) {
115
                     return JsonConvert.DeserializeObject<SpaceGridParameters>(sr.ReadToEn |
116
                 }
117
             } catch (Exception ex) {
118
                 Console.WriteLine($\"We had problem: \{ex.Message\}");
                 return null;
120
             }
121
         }
122
    }
123
124
    public readonly record struct TimeGridParameters {
125
         [JsonProperty("Initial area")]
126
         public Interval Interval { get; init; }
127
128
         [JsonProperty("Number of splits")]
129
         public int Splits { get; init; }
130
131
         [JsonProperty("Coef")]
132
         public double? K { get; init; } // коэффициент разрядки
133
         public TimeGridParameters(Interval interval, int splits, double? k) {
135
             Interval = interval;
136
             Splits = splits;
137
             K = k;
138
139
```

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

IGrids.cs

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

GridFactories.cs

```
namespace courseProject;
1
2
    public enum GridTypes {
3
        SpaceRegular,
4
        TimeRegular,
5
        TimeIrregular
6
7
8
    public interface ISpaceFactory {
9
        public ISpaceGrid CreateGrid(GridTypes spaceGridType, SpaceGridParameters

    gridParameters);
11
12
    public interface ITimeFactory {
13
        public ITimeGrid CreateGrid(GridTypes timeGridType, TimeGridParameters
14

    gridParameters);
15
    }
16
    public class SpaceGridFactory : ISpaceFactory {
17
        public ISpaceGrid CreateGrid(GridTypes spaceGridType, SpaceGridParameters
18
            gridParameters) {
            return spaceGridType switch {
19
                GridTypes.SpaceRegular => new SpaceRegularGrid(gridParameters),
20
```

```
21
            type of grid does not exist: {spaceGridType}"),
         };
23
      }
24
25
26
   public class TimeGridFactory : ITimeFactory {
27
      public ITimeGrid CreateGrid(GridTypes timeGridType, TimeGridParameters
28
         gridParameters) {
         return timeGridType switch {
29
            GridTypes.TimeRegular => new TimeRegularGrid(gridParameters),
30
31
            GridTypes.TimeIrregular => new TimeIrregularGrid(gridParameters),
32
33
            34

→ type of grid does not exist: {timeGridType}"),
         };
35
36
   }
37
```

SpaceGrids.cs

```
namespace courseProject;
1
2
    public class SpaceRegularGrid : ISpaceGrid {
3
        private readonly List<Point2D> _points = default!;
4
        private readonly int[][] _elements = default!;
5
        private readonly int[][][] _edges = default!;
6
        public ImmutableList<Point2D> Points => _points.ToImmutableList();
        public ImmutableArray<ImmutableArray<int>>> Elements => _elements.Select(item =>
        → item.ToImmutableArray()).ToImmutableArray();
        public ImmutableArray<ImmutableArray<ImmutableArray<int >>> Edges =>
9

    _edges.Select(item ⇒ item.ToImmutableArray().Select(item ⇒ )

           item.ToImmutableArray()).ToImmutableArray()).ToImmutableArray();
        public double Lambda { get; init; }
10
        public double Sigma { get; init; }
11
        public SpaceRegularGrid(SpaceGridParameters gridParameters) {
13
            Lambda = gridParameters.Lambda;
14
            Sigma = gridParameters.Sigma;
15
            _points = new();
            _elements = new int[2 * gridParameters.SplitsR *
17

¬ gridParameters.SplitsZ].Select(_ ⇒ new int[10]).ToArray();
            _edges = new int[2 * gridParameters.SplitsR *
18

¬ gridParameters.SplitsZ].Select(_ => new int[3].ToArray().Select(_ => new
                int[4]).ToArray()).ToArray();
            Build(gridParameters);
19
20
21
        private void Build(SpaceGridParameters gridParameters) {
22
            try {
                if (gridParameters.SplitsR < 1 || gridParameters.SplitsZ < 1) {</pre>
                    throw new Exception("The number of splits must be greater than or
25
                     \rightarrow equal to 1");
26
27
                double hr = gridParameters.IntervalR.Lenght / gridParameters.SplitsR;
28
                double hz = gridParameters.IntervalZ.Lenght / gridParameters.SplitsZ;
29
30
```

```
double[] pointsR = new double[(3 * gridParameters.SplitsR) + 1];
31
                 double[] pointsZ = new double[(3 * gridParameters.SplitsZ) + 1];
32
33
                 pointsR[0] = gridParameters.IntervalR.LeftBorder;
34
                 pointsZ[0] = gridParameters.IntervalZ.LeftBorder;
35
36
                 for (int i = 1; i < gridParameters.SplitsR + 1; i++) {</pre>
37
                     pointsR[i] = pointsR[i - 1] + hr;
38
                 }
39
40
                 for (int i = 1; i < gridParameters.SplitsZ + 1; i++) {</pre>
41
                     pointsZ[i] = pointsZ[i - 1] + hz;
42
43
44
                 for (int j = 0; j < gridParameters.SplitsZ + 1; j++) {</pre>
45
                      for (int i = 0; i < gridParameters.SplitsR + 1; i++) {</pre>
46
                          _points.Add(new(pointsR[i], pointsZ[j]));
47
48
                 }
49
50
                 Point2D[] allNodes = new Point2D[10];
51
                 Point2D[] vertices = new Point2D[3];
52
53
                 const int dummySplits = 2;
                 int nr = gridParameters.SplitsR + 1;
55
                 int index = 0;
56
57
                 for (int j = 0; j < gridParameters.SplitsZ; j++) {</pre>
58
                      for (int i = 0; i < gridParameters.SplitsR; i++) {</pre>
59
                          _elements[index][\emptyset] = i + (j * nr);
60
                          _elements[index][1] = i + 1 + (j * nr);
61
                          _elements[index++][2] = i + ((j + 1) * nr);
62
63
                          _elements[index][\emptyset] = i + 1 + (j * nr);
64
                          _elements[index][1] = i + nr + (j * nr);
65
                          _elements[index++][2] = i + 1 + nr + (j * nr);
66
                      }
67
                 }
68
69
                 WritePoints("forGraphics/pointsLinear.txt");
                 WriteElements("forGraphics/elements.txt");
71
                 pointsR.Fill(-1.0);
72
                 pointsZ.Fill(-1.0);
73
74
                 for (int ielem = 0; ielem < _elements.Length; ielem++) {</pre>
75
                     vertices[0] = _points[_elements[ielem][0]];
76
                     vertices[1] = _points[_elements[ielem][1]];
                     vertices[2] = _points[_elements[ielem][2]];
78
79
                     GetNodes(allNodes, vertices);
80
81
82
                     pointsR = pointsR.Concat(allNodes.Select(point => point.R)).ToArray();
                     pointsR = pointsR.Distinct().OrderBy(value => value).ToArray();
83
84
                     pointsZ = pointsZ.Concat(allNodes.Select(point => point.Z)).ToArray();
85
                     pointsZ = pointsZ.Distinct().OrderBy(value => value).ToArray();
87
88
                 _points.Clear();
89
90
```

```
foreach (var pointZ in pointsZ.Skip(1)) {
  91
                                                                                foreach (var pointR in pointsR.Skip(1)) {
  92
                                                                                               _points.Add(new(pointR, pointZ));
 93
 94
                                                                }
  95
 96
                                                               nr = pointsR.Length - 1;
 97
                                                                index = 0;
 98
                                                                for (int j = 0; j < gridParameters.SplitsZ; j++) {</pre>
100
                                                                                for (int i = 0; i < gridParameters.SplitsR; i++) {</pre>
101
                                                                                               _elements[index][\emptyset] = i + (3 * j * nr) + (i * dummySplits); // \emptyset
102
                                                                                              _elements[index][1] = i + (3 * nr) + (3 * j * nr) + (i * j * nr) + (i * nr)
103

    dummySplits); // 12

                                                                                               _elements[index][2] = i + 3 + (3 * j * nr) + (i * dummySplits);
104
                                                                                                → // 3
                                                                                               _elements[index][3] = i + nr + (3 * j * nr) + (i * dummySplits);
                                                                                                \hookrightarrow // 4
                                                                                               _elements[index][4] = i + (2 * nr) + (3 * j * nr) + (i * nr)
106
                                                                                                \rightarrow dummySplits); // 8
                                                                                              _elements[index][5] = i + 1 + (2 * nr) + (3 * j * nr) + (i * nr)

    dummySplits); //9

                                                                                               elements[index][6] = i + nr + 2 + (3 * j * nr) + (i * j * nr) + (i * nr) + 
108

    dummySplits); // 6

                                                                                               _elements[index][7] = i + 2 + (3 * j * nr) + (i * dummySplits);
109
                                                                                                            // 2
                                                                                               _elements[index][8] = i + 1 + (3 * j * nr) + (i * dummySplits);
110
                                                                                                \rightarrow // 1
                                                                                               elements[index][9] = i + nr + 1 + (3 * j * nr) + (i * )
 111

    dummySplits); // 5

112
                                                                                               _{edges[index][0][0] = _{elements[index][0];}
                                                                                               _edges[index][0][1] = _elements[index][8];
114
                                                                                               _{edges[index][0][2] = _{elements[index][7];}
115
                                                                                               _edges[index][0][3] = _elements[index][2];
116
117
                                                                                              _edges[index][1][0] = _elements[index][0];
118
                                                                                              _edges[index][1][1] = _elements[index][3];
119
                                                                                               _edges[index][1][2] = _elements[index][4];
120
                                                                                               _edges[index][1][3] = _elements[index][1];
122
                                                                                               _{\text{edges}[index][2][0]} = _{\text{elements}[index][2]};
123
                                                                                              _edges[index][2][1] = _elements[index][6];
124
                                                                                              _edges[index][2][2] = _elements[index][5];
125
                                                                                              _{edges[index][2][3]} = _{elements[index++][1]};
126
127
                                                                                               _elements[index][0] = i + (3 * nr) + (3 * j * nr) + (i *
128

    dummySplits); // 12

                                                                                               _elements[index][1] = i + 3 + (3 * j * nr) + (i * dummySplits);
129
                                                                                                → //3
                                                                                              _elements[index][2] = i + 3 + (3 * nr) + (3 * j * nr) + (i * j * nr) + (i *
130

    dummySplits); // 15

                                                                                               _elements[index][3] = i + 1 + (2 * nr) + (3 * j * nr) + (i * nr)
131

    dummySplits); // 9

                                                                                               132

→ dummySplits); // 6

                                                                                               _elements[index][5] = i + nr + 3 + (3 * j * nr) + (i *
133
                                                                                                \rightarrow dummySplits); // 7
                                                                                              _elements[index][6] = i + 3 + (2 * nr) + (3 * j * nr) + (i * nr)
134
                                                                                                            dummySplits); // 11
```

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

TimeGrids.cs

```
namespace courseProject;
2
    public class TimeRegularGrid : ITimeGrid {
3
        private double[] _points = default!;
4
        public ImmutableArray(double> Points => _points.ToImmutableArray();
5
6
        public TimeRegularGrid(TimeGridParameters gridParameters) {
             Build(gridParameters);
8
        }
9
10
        private void Build(TimeGridParameters gridParameters) {
11
            try {
                 if (gridParameters.Interval.LeftBorder < 0)</pre>
13
                     throw new Exception("The beginning of the time segment cannot be less
14
                      \rightarrow than 0");
                 if (gridParameters.Splits < 1)</pre>
16
                     throw new Exception("The number of splits must be greater than or
17

    equal to 1");

18
                 _points = new double[gridParameters.Splits + 1];
19
20
                 double h = gridParameters.Interval.Lenght / gridParameters.Splits;
21
                 _points[0] = gridParameters.Interval.LeftBorder;
23
                 _points[^1] = gridParameters.Interval.RightBorder;
24
                 for (int i = 1; i < points.Length - 1; i++) {
                     _{points[i]} = _{points[i - 1]} + h;
27
28
             } catch (Exception ex) {
                 Console.WriteLine($\"We had problem: \{ex.Message\}\");
30
             }
31
        }
32
33
34
    public class TimeIrregularGrid : ITimeGrid {
35
        private double[] _points = default!;
36
        public ImmutableArray<double> Points => _points.ToImmutableArray();
37
38
        public TimeIrregularGrid(TimeGridParameters gridParameters) {
30
             Build(gridParameters);
40
42
        private void Build(TimeGridParameters gridParameters) {
43
            try {
                 if (gridParameters.Interval.LeftBorder < 0)</pre>
45
                     throw new Exception("The beginning of the time segment cannot be less
46

→ than 0");

                 if (gridParameters.Splits < 1)</pre>
                     throw new Exception("The number of splits must be greater than or
49
                      → equal to 1");
50
                 ArgumentNullException.ThrowIfNull(gridParameters.K,
51
                 → $\[ \{\text{nameof(gridParameters.K)}\} \] cannot be null");
52
                 _points = new double[gridParameters.Splits + 1];
```

```
double h;
55
               double sum = 0.0;
56
57
               for (int k = 0; k < gridParameters.Splits; k++)</pre>
58
                   sum += Math.Pow(gridParameters.K.Value, k);
59
60
               h = gridParameters.Interval.Lenght / sum;
61
               _points[0] = gridParameters.Interval.LeftBorder;
               _points[^1] = gridParameters.Interval.RightBorder;
63
64
               for (int i = 1; i \leftarrow points.Length - 1; i++) {
65
                   _{points[i]} = _{points[i - 1]} + h;
66
                   h *= gridParameters.K.Value;
67
68
           } catch (Exception ex) {
69
               71
72
73
```

Interval.cs

```
namespace courseProject;
1
   public readonly record struct Interval {
3
        [JsonProperty("Left Border")]
4
        public double LeftBorder { get; init; }
5
6
        [JsonProperty("Right Border")]
        public double RightBorder { get; init; }
8
C
        [JsonIgnore]
        public double Lenght { get; init; }
11
12
        [JsonConstructor]
13
        public Interval(double leftBorder, double rightBorder) {
            LeftBorder = leftBorder;
15
            RightBorder = rightBorder;
16
            Lenght = Math.Abs(rightBorder - leftBorder);
17
18
19
```

Point2D.cs

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

```
public static Point2D operator /(Point2D point, double constant)
public static Point2D operator /(Point2D point, double constant)
public override string ToString()
public override string ToString()
public override string ToString()
public override string ToString()
```

ArrayHelper.cs

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

GMrz.py

```
from sympy import expand, Symbol, diff
2
    11 = Symbol(['|11]')
3
    12 = Symbol('12')
4
    13 = Symbol('13')
5
    dl1 = Symbol('d1l')
6
    d12 = Symbol('d21')
    d13 = Symbol('d31')
9
    #функции
10
    # psi1 = 0.5*l1*(3*l1-1)*(3*l1-2)
11
    # psi2 = 0.5*12*(3*12-1)*(3*12-2)
12
    # psi3 = 0.5*13*(3*13-1)*(3*13-2)
13
    \# psi4 = 4.5*11*12*(3*11-1)
14
    # psi5 = 4.5*l1*l2*(3*l2-1)
15
    # psi6 = 4.5*12*13*(3*12-1)
    \# psi7 = 4.5*12*13*(3*13-1)
17
    \# psi8 = 4.5*13*11*(3*13-1)
18
    \# psi9 = 4.5*13*11*(3*11-1)
19
      psi10 = 27*11*12*13
20
21
```

```
psi1 = 0.5*11*(3*11-1)*(3*11-2)
22
    psi2 = 4.5*13*11*(3*11-1)
23
    psi3 = 4.5*13*11*(3*13-1)
24
    psi4 = 0.5*13*(3*13-1)*(3*13-2)
25
    psi5 = 4.5*11*12*(3*11-1)
26
    psi6 = 27*11*12*13
27
    psi7 = 4.5*12*13*(3*13-1)
28
    psi8 = 4.5*11*12*(3*12-1)
29
30
    psi9 = 4.5*12*13*(3*12-1)
    psi10 = 0.5*12*(3*12-1)*(3*12-2)
31
32
    psi = [psi1, psi2, psi3, psi4, psi5, psi6, psi7, psi8, psi9, psi10]
33
34
    #производные
35
    p1 = d11*diff(psi1, l1)+dl2*diff(psi1, l2)+dl3*diff(psi1, l3)
36
    p2 = d11*diff(psi2, 11)+d12*diff(psi2, 12)+d13*diff(psi2, 13)
37
    p3 = dl1*diff(psi3, l1)+dl2*diff(psi3, l2)+dl3*diff(psi3, l3)
    p4 = dl1*diff(psi4, l1)+dl2*diff(psi4, l2)+dl3*diff(psi4, l3)
39
    p5 = dl1*diff(psi5, l1)+dl2*diff(psi5, l2)+dl3*diff(psi5, l3)
40
41
    p6 = dl1*diff(psi6, l1)+dl2*diff(psi6, l2)+dl3*diff(psi6, l3)
    p7 = dl1*diff(psi7, l1)+dl2*diff(psi7, l2)+dl3*diff(psi7, l3)
    p8 = dl1*diff(psi8, l1)+dl2*diff(psi8, l2)+dl3*diff(psi8, l3)
43
    p9 = d11*diff(psi9, 11)+d12*diff(psi9, 12)+d13*diff(psi9, 13)
44
    p10 = d11*diff(psi10, 11)+d12*diff(psi10, 12)+d13*diff(psi10, 13)
45
46
    p=[p1, p2, p3, p4, p5, p6, p7, p8, p9, p10]
47
48
    def fac(n):
49
        if n == 0:
50
            return 1
51
        return fac(n-1) * n
52
53
    def Integrate(func):
54
        res = [0, 0, 0]
55
        func = func.replace("- 2", "-2")
56
        func = func.replace("-3", "-3")
57
        func = func.replace("- 4", "-4")
58
        func = func.replace("- 1", "-1")
59
        func = func.replace("- 5", "-5")
60
        func = func.replace("- 6", "-6")
61
        func = func.replace("- 7")
62
        func = func.replace("- 8", "-8")
63
        func = func.replace("- 9",
64
        func = func.replace([+ [],
65
        func = func.replace("**", "^")
66
        func = func.split()
67
        func = [i.split("*", 1) for i in func]
68
        for iter in range(len(func)):
69
             a = float(func[iter][0])
70
             f = func[iter][1].rfind(||11^{||})
71
            if f>=0:
72
                 i = int(func[iter][1][func[iter][1].rfind(<mark>|</mark>|11<mark>^|</mark>)+3])
73
74
                 f = func[iter][1].rfind(| l1 | )
75
                 if f>=∅:
76
                     i = 1
77
                 else:
78
                     i = 0
79
             f = func[iter][1].rfind([12^{1}])
80
             if f>=0:
81
```

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

```
elif func[iter][1].rfind("d11*d21")>=0: flag = 1
141
              elif func[iter][1].rfind("d11*d31")>=0: flag = 2
              elif func[iter][1].rfind("d21^2")>=0: flag = 3
143
              elif func[iter][1].rfind("d21*d31")>=0: flag = 4
144
              elif func[iter][1].rfind("d31^2")>=0: flag = 5
145
              a = float(func[iter][0])
146
              f = func[iter][1].rfind(| l1^|)
147
              if f>=0:
148
                  i = int(func[iter][1][func[iter][1].rfind([11^{-}])+3])
              else:
150
                   f = func[iter][1].rfind([11])
151
                  if f>=0:
152
                       i = 1
153
                  else:
154
                       i = 0
155
              f = func[iter][1].rfind([12^{-1}])
156
              if f>=0:
157
                   j = <mark>int</mark>(func[iter][1][func[iter][1].rfind(<mark>|</mark>12^<mark>|</mark>)+3])
158
              else:
159
                   f = func[iter][1].rfind(|12|)
160
                  if f>=0:
161
                       j = 1
162
                  else:
163
                       i = 0
              f = func[iter][1].rfind([13^{"}])
165
              if f>=0:
166
                  k = int(func[iter][1][func[iter][1].rfind([13^{1}])+3])
167
              else:
168
                   f = func[iter][1].rfind("13")
169
                  if f>=0:
170
                       k = 1
171
                  else:
172
173
              res[flag][0]+=a * fac(i+1)*fac(j)*fac(k)/fac(i+j+k+3)
174
              res[flag][1]+=a * fac(i)*fac(j+1)*fac(k)/fac(i+j+k+3)
175
              res[flag][2]+=a * fac(i)*fac(j)*fac(k+1)/fac(i+j+k+3)
176
177
         return res
178
179
     #L(x,y)=a0i+a1ix+a2iy
180
     #rs=[dl1^2 dl1dl2 dl1dl3 dl2^2 dl2dl3 dl3dl3]
181
     def makeG():
182
         f2 = open('Grz.txt'),
183
         rs = []
         for i in range(10):
185
              rs.append([])
186
              for j in range(10):
187
                  rs[i].append([])
188
189
         for i in range(10):
190
              for j in range(10):
191
192
                  r = str(expand(p[i]*p[j]))
                  rs[i][j]=IntegrateGrad(r)
193
         i=0
194
         j = ∅
         k = 0
196
         for row in rs:
197
              for x in row:
198
                  for y in x:
199
                       f2.write(str(i) + "" + str(j) + "" + str(k) + "" + "
200
                           '.join(["{0:.20f}".format(float(elem)) for elem in y]) + "\n")
```

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

graphics.py

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