МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

НГТУ
НЭТИ

Факультет прикладной
математики и информатики

Кафедра прикладной математики

Курсовая работа
по дисциплине «Уравнения математической физики»

# Применение МКЭ для решения нестационарных задач

Студент ШИШКИН НИКИТА

Группа ПМ-92

Преподаватель РОЯК МИХАИЛ ЭММАНУИЛОВИЧ

Новосибирск, 2022

# Постановка задачи

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

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

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

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

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

Упростим выражения для более простой записи производных с помощью соотношений:

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

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

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

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

$$\frac{\partial}{\partial t}\left(u^{j-3}(x,y)\,\eta_3^j(t) + u^{j-2}(x,y)\,\eta_2^j(t) + u^{j-1}(x,y)\,\eta_1^j(t) + u^j(x,y)\,\eta_0^j(t)\right)\Bigg|_{t=t_j} -$$

$$-\operatorname{div}\left(\lambda \operatorname{grad} u^j\right) = f^j \qquad (6)$$

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

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

$$-\operatorname{div}\left(\lambda \operatorname{grad} u^j\right) = f^j \qquad (7)$$

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

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

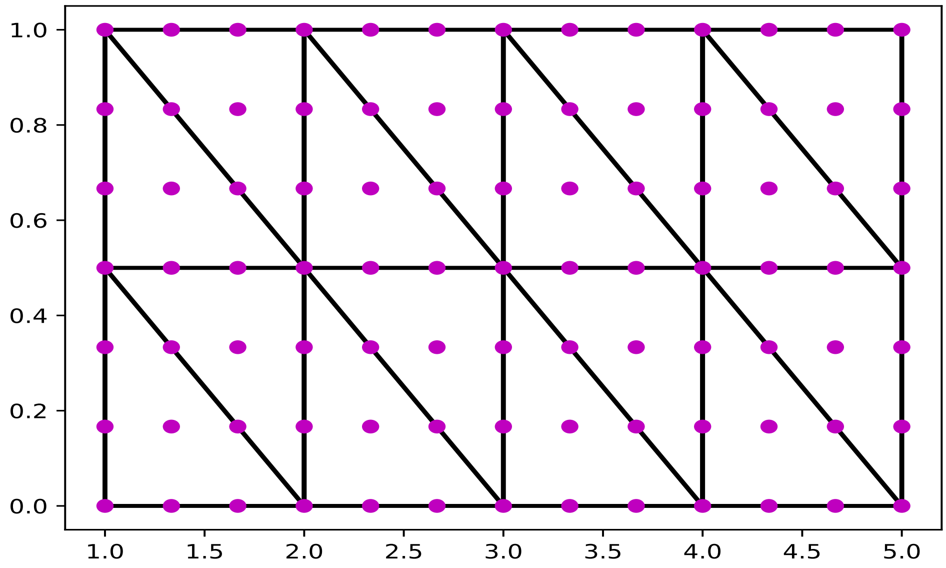
$$= b^j + \frac{\Delta t_{02}\Delta t_{01}}{\Delta t_{23}\Delta t_{13}\Delta t_{03}}\mathbf{M}\,q^{j-3} - \frac{\Delta t_{03}\Delta t_{01}}{\Delta t_{23}\Delta t_{12}\Delta t_{02}}\mathbf{M}\,q^{j-2} + \frac{\Delta t_{03}\Delta t_{02}}{\Delta t_{13}\Delta t_{12}\Delta t_{01}}\mathbf{M}\,q^{j-1} \qquad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

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

# Тестирование программы

## Первый тест

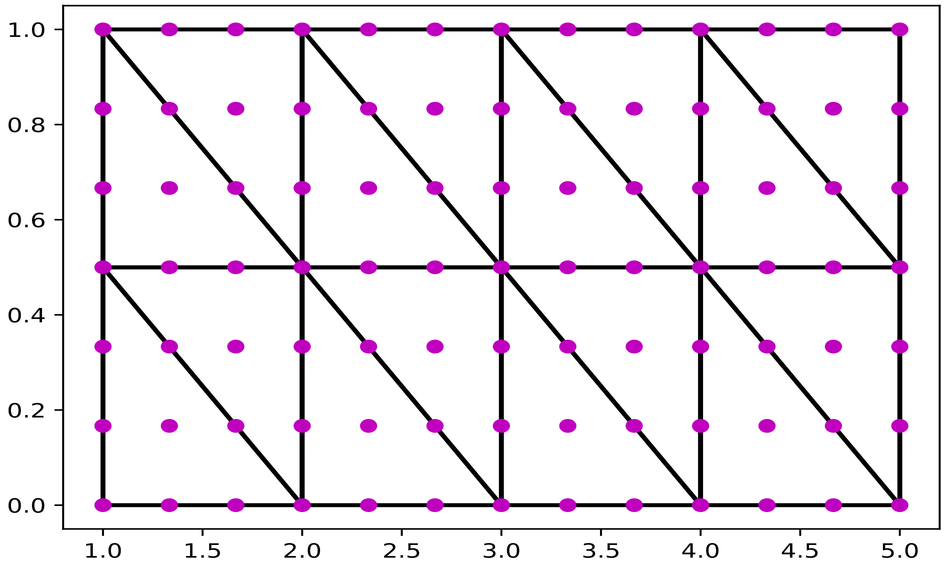
Функция: $r^2 + z + t$
Правая часть: $-3$
Коэффициенты : $\lambda = 1$, $\sigma = 1$
Сетка по времени: равномерная [0,1], $h_t = 0.1$
Заданы краевые условия 1-го рода.



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

### Второй тест

Функция: $r^2 - zt^2$
Правая часть: $-8 - zt$
Коэффициенты : $\lambda = 2$, $\sigma = 0.5$
Сетка по времени: равномерная [0,1], $h_t = 0.1$
Заданы краевые условия 1-го рода.

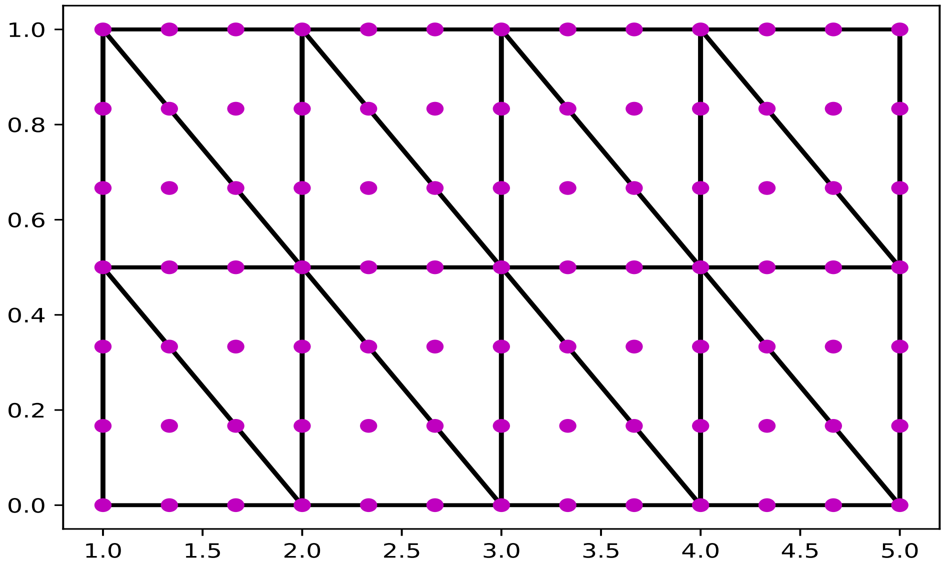


| $t_i$ | Погрешность |
|---|---|
| $0.1$ | $2.19 \cdot 10^{-3}$ |
| $0.2$ | $7.49 \cdot 10^{-4}$ |
| $0.3$ | $1.62 \cdot 10^{-4}$ |
| $0.4$ | $1.46 \cdot 10^{-4}$ |
| $0.5$ | $9.75 \cdot 10^{-6}$ |
| $0.6$ | $3.14 \cdot 10^{-5}$ |
| $0.7$ | $5.76 \cdot 10^{-6}$ |
| $0.8$ | $4.56 \cdot 10^{-6}$ |
| $0.9$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

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

**Третий тест**

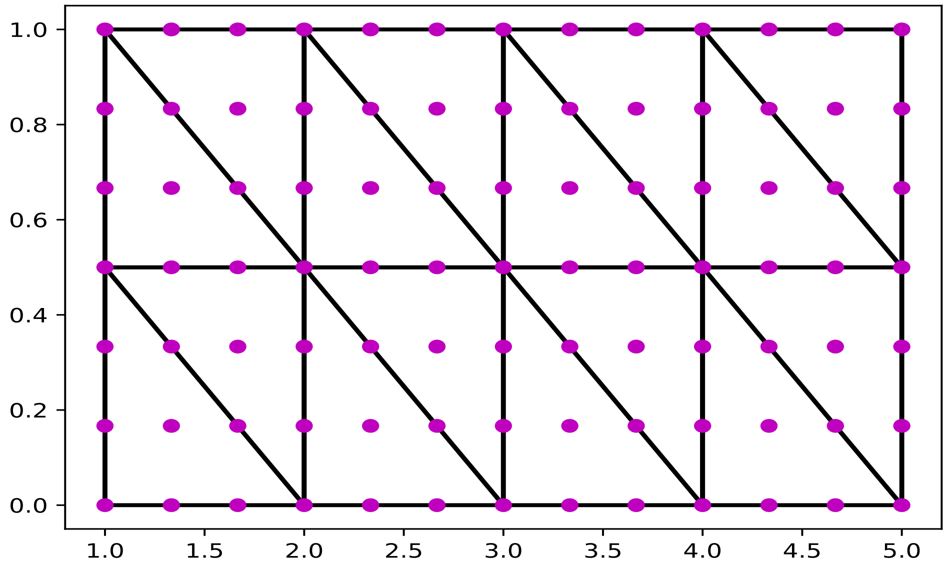
Функция: $2r^3t^3$
Правая часть: $-9rt^3 + 12r^3t^2$
Коэффициенты : $\lambda = 0.5$, $\sigma = 2$
Сетка по времени: равномерная [0,1], $h_t = 0.01$
Заданы краевые условия 1-го рода на левой и правой границе.



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

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

### Исследование на определение порядка сходимости по пространству

Функция: $\ln r$<br>Правая часть: $0$<br>Коэффициенты : $\lambda = 1$, $\sigma = 0$<br>Заданы краевые условия 1-го рода на левой и правой границе.

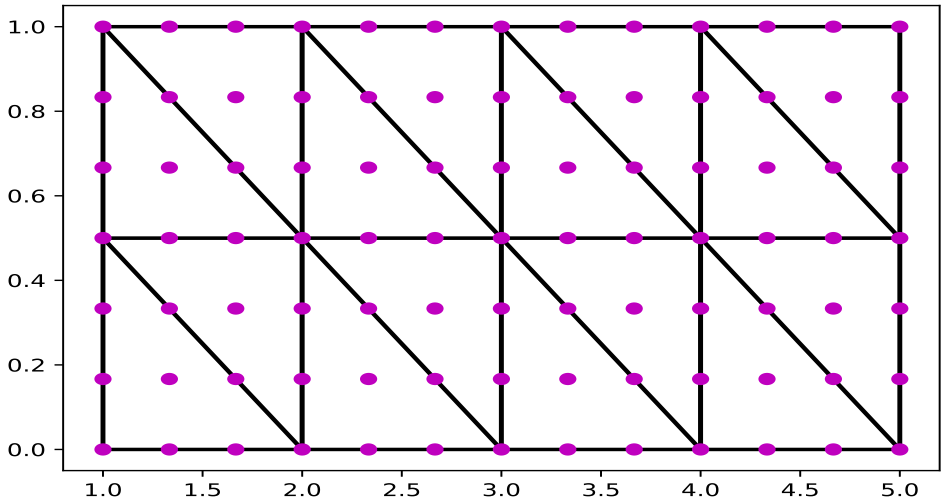


Рис. 1: Сетка с шагом $h$

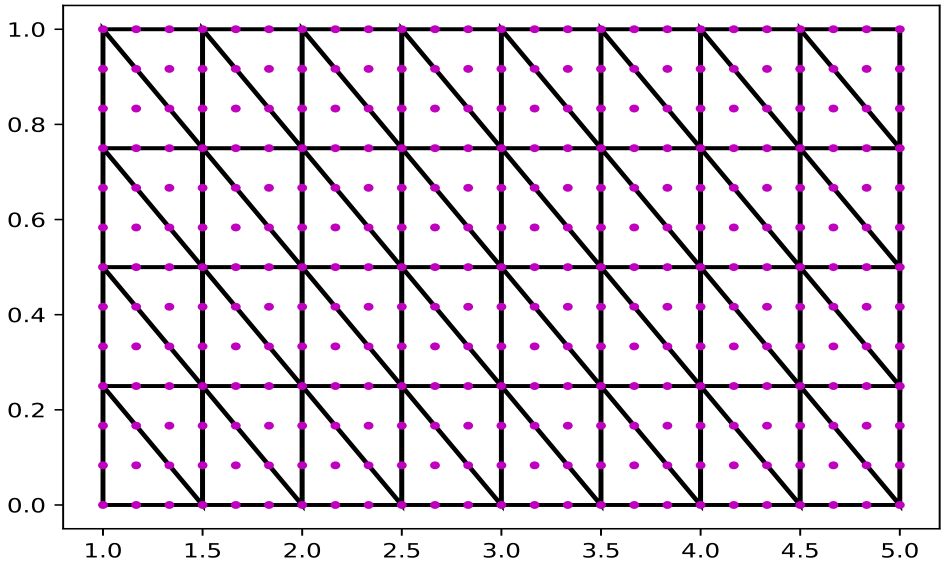


Рис. 2: Сетка с шагом $h/2$

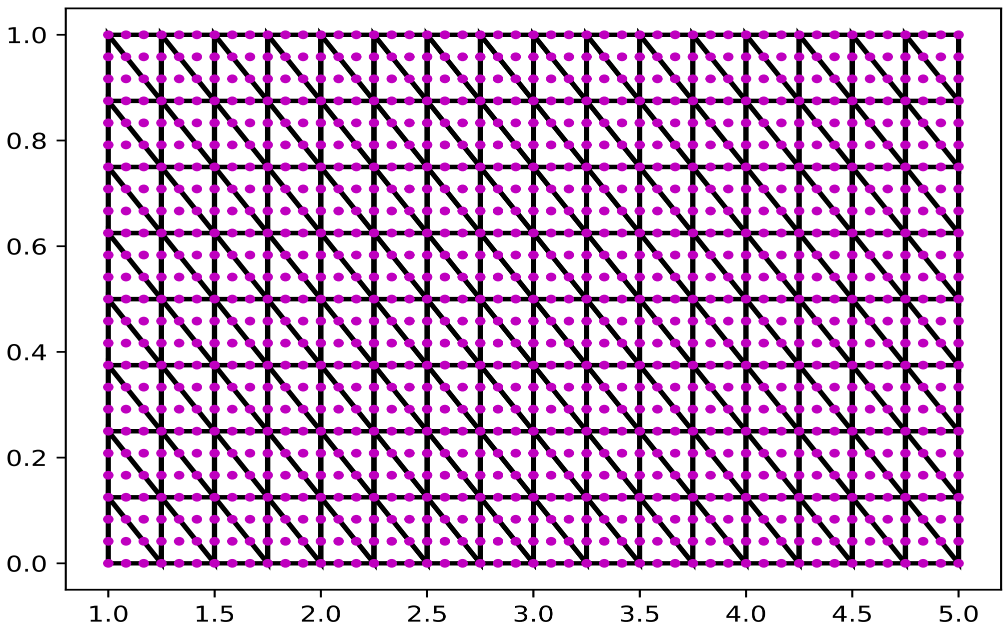


Рис. 3: Сетка с шагом $h/4$

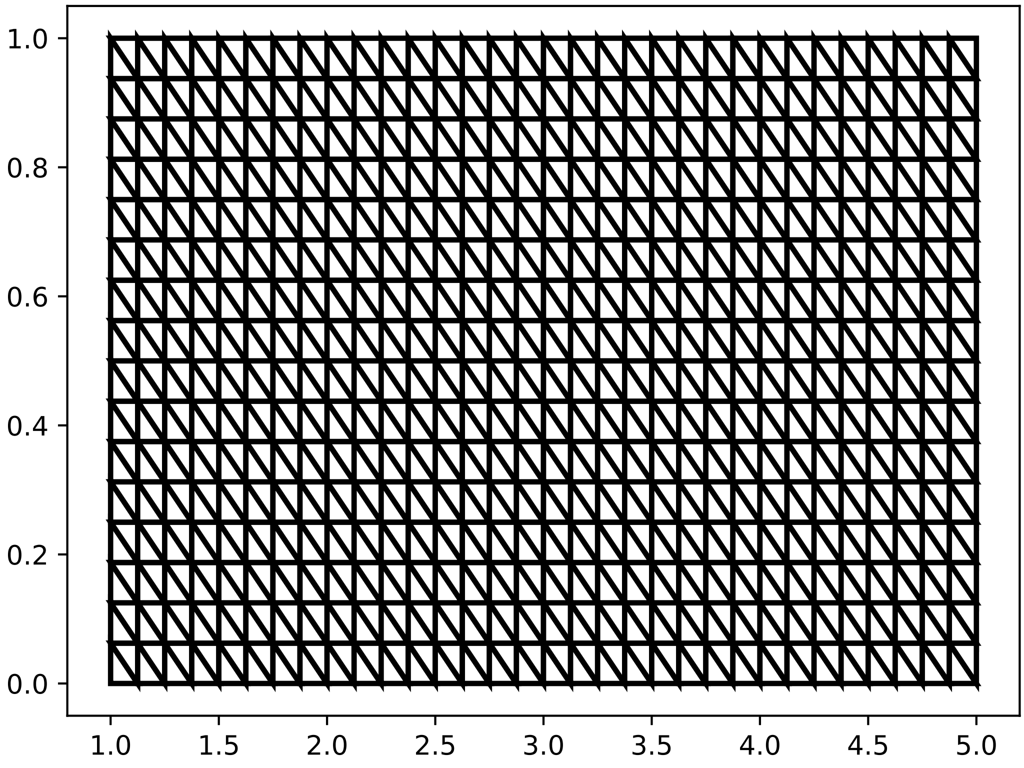


Рис. 4: Сетка с шагом $h/8$

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

### Исследование на определение порядка сходимости схемы по времени

Функция: $\cos t$<br>
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 |
|---|---|---|
| $h$ | $0.00042493625406315 16$ | – |
| $h/2$ | $5.3306642166155186E-05$ | $2.99$ |

# 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

```csharp
using courseProject;

SpaceGridFactory spaceGridFactory = new();
TimeGridFactory timeGridFactory = new();

// FEM fem = FEM.CreateBuilder().SetTest(new Test1())
// .SetSpaceGrid(spaceGridFactory.CreateGrid(GridTypes.SpaceRegular, SpaceGridParameters.ReadJson("input/spaceGrid.jsonc")!.Value))
// .SetTimeGrid(timeGridFactory.CreateGrid(GridTypes.TimeRegular, TimeGridParameters.ReadJson("input/timeGrid.json")!.Value))
// .SetSolverSLAE(new CGMCholesky(1000, 1E-14))
// .SetDiriclhetBoundaries(DirichletBoundary.ReadJson("input/DirichletBoundaries.json")!)
// .SetNeumannBoundaries(NeumannBoundary.ReadJson("input/NeumannBoundaries.json")!)
// .IsPhysical(false);

FEM fem = FEM.CreateBuilder().SetTest(new Test7())
.SetSpaceGrid(spaceGridFactory.CreateGrid(GridTypes.SpaceRegular, SpaceGridParameters.ReadJson("input/spaceGrid.jsonc")!.Value))
.SetTimeGrid(timeGridFactory.CreateGrid(GridTypes.TimeRegular, TimeGridParameters.ReadJson("input/timeGrid.json")!.Value))
.SetSolverSLAE(new CGMCholesky(1000, 1E-14))
.SetDiriclhetBoundaries(DirichletBoundary.ReadJson("input/DirichletBoundaries.json")!)
.IsPhysical(false);

fem.Compute();
fem.WriteToFile("results/q.txt");

// var valuesAtPoints = fem.ValueAtPoint(new Point2D[] { new(1.5, 0.5), new (1, 0) });
// Array.ForEach(valuesAtPoints, Console.WriteLine);
```

## FEM.cs

```csharp
namespace courseProject;

public class FEM {
    public class FEMBuilder {
        private readonly FEM _fem = new();

        public FEMBuilder SetTest(ITest test) {
            _fem._test = test;
            return this;
        }

        public FEMBuilder SetSpaceGrid(ISpaceGrid spaceGrid) {
            _fem._spaceGrid = spaceGrid;
            return this;
        }

        public FEMBuilder SetTimeGrid(ITimeGrid grid) {
            _fem._timeGrid = grid;
            return this;
        }

        public FEMBuilder SetSolverSLAE(Solver solver) {
            _fem._solver = solver;
            return this;
        }

        public FEMBuilder SetDiriclhetBoundaries(DirichletBoundary[] boundaries) {
            _fem._dirichletBoundaries = boundaries;
            return this;
        }

        public FEMBuilder SetNeumannBoundaries(NeumannBoundary[] boundaries) {
            _fem._neumannBoundaries = boundaries;
            return this;
        }

        public FEMBuilder IsPhysical(bool flag) {
            _fem.IsPhysical = flag;
            return this;
        }

        public static implicit operator FEM(FEMBuilder builder)
            => builder._fem;
    }

    // default ~ cannot be null
    private delegate double Basis(Point2D point);

    private Basis[] _basis = default!;
    private ISpaceGrid _spaceGrid = default!;
    private ITimeGrid _timeGrid = default!;
    private Point2D[] _vertices = default!;
    private ITest _test = default!;
    private Solver _solver = default!;
    private DirichletBoundary[] _dirichletBoundaries = default!;
    private NeumannBoundary[]? _neumannBoundaries;
    private Matrix _stiffnessMatrix = default!;
    private Matrix _massMatrix = default!;
    private SparseMatrix _globalMatrix = default!;
    private Vector<double> _vector = default!;
    private static Matrix _alphas = default!; // коэффициенты альфа, костыль :(
    private double[][][] _M = default!;
    private double[][][][] _G = default!;
    private double[][] _layers = default!;
    private Matrix _massMatrixCopy = default!;
    public bool IsPhysical { get; private set; }

    public void Compute() {
        try {
            ArgumentNullException.ThrowIfNull(_test, $"{nameof(_test)} cannot be
             ↪  null, set the test");
            ArgumentNullException.ThrowIfNull(_solver, $"{nameof(_solver)} cannot be
             ↪  null, set the method of solving SLAE");
            ArgumentNullException.ThrowIfNull(_dirichletBoundaries,
             ↪  $"{nameof(_dirichletBoundaries)} cannot be null, set the Dirichlet
             ↪  boundaries");

            Init();
            ConstructPortrait();
            Prepare();
            Solve();
            // Err();
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }

    private void Init() {
        _basis = new Basis[] { CubicBasis.Psi1, CubicBasis.Psi2, CubicBasis.Psi3, CubicBasis.Psi4,
        CubicBasis.Psi5, CubicBasis.Psi6, CubicBasis.Psi7, CubicBasis.Psi8, CubicBasis.Psi9, CubicBasis.Psi10};

        _stiffnessMatrix = new(10);
        _massMatrix = new(10);
        _massMatrixCopy = new(10);
        _alphas = new(3);

        _vertices = new Point2D[3];

        _layers = new double[3].Select(_ => new double[_spaceGrid.Points.Count]).ToArray();

        _M = new double[10].Select(_ => new double[10].ToArray().
                         Select(_ => new double[3]).ToArray()).ToArray();

        _G = new double[10].Select(_ => new double[10].ToArray().
                        Select(_ => new double[6].ToArray().
                        Select(_ => new double[3]).ToArray()).ToArray()).ToArray();

        using StreamReader sr1 = new("input/Grz.txt"), sr2 = new("input/Mrz.txt");
        string[] vars;

        for (int i = 0; i < 600; i++) {
            vars = sr1.ReadLine()!.Split(" ").ToArray();

            _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][0] = double.Parse(vars[3]);
            _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][1] = double.Parse(vars[4]);
            _G[int.Parse(vars[0])][int.Parse(vars[1])][int.Parse(vars[2])][2] = double.Parse(vars[5]);
        }

        for (int i = 0; i < 100; i++) {
            vars = sr2.ReadLine()!.Split(" ").ToArray();

            _M[int.Parse(vars[0])][int.Parse(vars[1])][0] = double.Parse(vars[2]);
            _M[int.Parse(vars[0])][int.Parse(vars[1])][1] = double.Parse(vars[3]);
            _M[int.Parse(vars[0])][int.Parse(vars[1])][2] = double.Parse(vars[4]);
        }
    }

    private void Prepare() {
        if (!IsPhysical) {
            for (int i = 0; i < _spaceGrid.Points.Count; i++) {
                _layers[0][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[0]);
                _layers[1][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[1]);
                _layers[2][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[2]);
            }
        } else {
            for (int i = 0; i < _spaceGrid.Points.Count; i++) {
                _layers[0][i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[0]);
            }

            double t01 = _timeGrid.Points[1] - _timeGrid.Points[0];

            AssemblyGlobalMatrix(1, t01: t01);

            if (_neumannBoundaries is not null) {
                AccountingNeumannBoundaries();
            }

            AccountingDirichletBoundaries(1);

            _solver.SetMatrix(_globalMatrix);
            _solver.SetVector(_vector);
            _solver.Compute();
            Err(1);

            _solver.Solution!.Value.CopyTo(0, _layers[1], 0, _layers[1].Length);

            _globalMatrix.Clear();
            _vector.Fill(0);

            double t02 = _timeGrid.Points[2] - _timeGrid.Points[0];
            double t12 = _timeGrid.Points[1] - _timeGrid.Points[0];
            t01 = _timeGrid.Points[2] - _timeGrid.Points[1];

            AssemblyGlobalMatrix(2, t02: t02, t01: t01, t12: t12);

            if (_neumannBoundaries is not null) {
                AccountingNeumannBoundaries();
            }

            AccountingDirichletBoundaries(2);

            _solver.SetMatrix(_globalMatrix);
            _solver.SetVector(_vector);
            _solver.Compute();
            Err(2);

            _solver.Solution!.Value.CopyTo(0, _layers[2], 0, _layers[2].Length);

            _globalMatrix.Clear();
            _vector.Fill(0);
        }
    }

    private void Solve() {
        for (int itime = 3; itime < _timeGrid.Points.Length; itime++) {
            double t03 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 3];
            double t02 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 2];
            double t01 = _timeGrid.Points[itime] - _timeGrid.Points[itime - 1];
            double t13 = _timeGrid.Points[itime - 1] - _timeGrid.Points[itime - 3];
            double t12 = _timeGrid.Points[itime - 1] - _timeGrid.Points[itime - 2];
            double t23 = _timeGrid.Points[itime - 2] - _timeGrid.Points[itime - 3];

            AssemblyGlobalMatrix(itime, t03, t02, t01, t13, t12, t23);

            if (_neumannBoundaries is not null) {
                AccountingNeumannBoundaries();
            }
            AccountingDirichletBoundaries(itime);
            _globalMatrix.PrintDense("results/matrix.txt");

            _solver.SetMatrix(_globalMatrix);
            _solver.SetVector(_vector);
            _solver.Compute();
            Err(itime);

            _layers[1].Copy(_layers[0]);
            _layers[2].Copy(_layers[1]);
            _solver.Solution!.Value.CopyTo(0, _layers[2], 0, _layers[2].Length);

            _vector.Fill(0);
            _globalMatrix.Clear();
        }
    }

    private void ConstructPortrait() {
        List<int>[] list = new List<int>[_spaceGrid.Points.Count].Select(_ => new
        ↪   List<int>()).ToArray();

        foreach (var element in _spaceGrid.Elements.Select(array =>
        ↪   array.OrderBy(value => value).ToArray()).ToArray()) {
            for (int i = 0; i < element.Length; i++) {
                for (int j = i + 1; j < element.Length; j++) {
                    int pos = element[j];
                    int elem = element[i];

                    if (!list[pos].Contains(elem)) {
                        list[pos].Add(elem);
                    }
                }
            }
        }

        list = list.Select(list => list.OrderBy(value => value).ToList()).ToArray();
        int count = list.Sum(childList => childList.Count);

        InitSLAE(count);

        _globalMatrix.ig[0] = 0;

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

        int k = 0;

        for (int i = 0; i < list.Length; i++) {
            for (int j = 0; j < list[i].Count; j++) {
                _globalMatrix.jg[k] = list[i][j];
                k++;
            }
        }
    }

    private void InitSLAE(int count) {
        _globalMatrix = new(_spaceGrid.Points.Count, count);
        _vector = new(_spaceGrid.Points.Count);
    }
    private void AddElement(int i, int j, double value) {
        if (i == j) {
            _globalMatrix.di[i] += value;
            return;
        }

        if (i > j) {
            for (int index = _globalMatrix.ig[i]; index < _globalMatrix.ig[i + 1]; index++) {
                if (_globalMatrix.jg[index] == j) {
                    _globalMatrix.gg[index] += value;
                    return;
                }
            }
        }
    }

    private void AssemblyGlobalMatrix(int itime = 1, double t03 = 1, double t02 = 1, double t01 = 1, double t13 = 1, double t12 = 1, double t23 = 1) {
        for (int ielem = 0; ielem < _spaceGrid.Elements.Length; ielem++) {
            _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
            _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
            _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];

            AssemblyLocalMatrices(itime, t03, t02, t01);

            _stiffnessMatrix += _massMatrix;

            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                    AddElement(_spaceGrid.Elements[ielem][i], _spaceGrid.Elements[ielem][j], _stiffnessMatrix[i, j]);
                }
            }

            AssemblyGlobalVector(itime, ielem, t03, t02, t01, t13, t12, t23);

            _stiffnessMatrix.Clear();
            _massMatrix.Clear();
        }
    }

    private void AssemblyGlobalVector(int itime = 1, int ielem = 1, double t03 = 1, 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];
        double[] qj3 = new double[10];

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                if (itime == 1) {
                    qj1[i] += _massMatrixCopy[i, j] * _layers[0][_spaceGrid.Elements[ielem][j]];
                } else if (itime == 2) {
                    qj2[i] += _massMatrixCopy[i, j] * _layers[0][_spaceGrid.Elements[ielem][j]];
                    qj1[i] += _massMatrixCopy[i, j] * _layers[1][_spaceGrid.Elements[ielem][j]];
                } else {
                    qj3[i] += _massMatrixCopy[i, j] *
                    ↪   _layers[0][_spaceGrid.Elements[ielem][j]];
                    qj2[i] += _massMatrixCopy[i, j] *
                    ↪   _layers[1][_spaceGrid.Elements[ielem][j]];
                    qj1[i] += _massMatrixCopy[i, j] *
                    ↪   _layers[2][_spaceGrid.Elements[ielem][j]];
                }
            }
        }

        for (int j = 0; j < 10; j++) {
            if (itime == 1) {
                _vector[_spaceGrid.Elements[ielem][j]] +=
                ↪   Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
                ↪   _timeGrid.Points[2]) -
                                                      (-1.0 / t01 * qj1[j]);
            } else if (itime == 2) {
                _vector[_spaceGrid.Elements[ielem][j]] +=
                ↪   Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
                ↪   _timeGrid.Points[2]) -
                                          (t01 / (t02 * t12) * qj2[j]) + (t02 / (t12
                                          ↪   * t01) * qj1[j]);
            } else {
                _vector[_spaceGrid.Elements[ielem][j]] +=
                ↪   Integration.Triangle(_test.F, _basis[j].Invoke, _vertices,
                ↪   _timeGrid.Points[itime]) +
                                                          (t02 * t01 / (t23 * t13 *
                                                          ↪   t03) * qj3[j]) -
                                                          (t03 * t01 / (t23 * t12 *
                                                          ↪   t02) * qj2[j]) +
                                                          (t03 * t02 / (t13 * t12 *
                                                          ↪   t01) * qj1[j]);
            }
        }
    }

    private double DeterminantD()
        => ((_vertices[1].R - _vertices[0].R) * (_vertices[2].Z - _vertices[0].Z)) -
           ((_vertices[2].R - _vertices[0].R) * (_vertices[1].Z - _vertices[0].Z));

    private void CalcAlphas() {
        double dD = DeterminantD();

        _alphas[0, 0] = ((_vertices[1].R * _vertices[2].Z) - (_vertices[2].R *
        ↪   _vertices[1].Z)) / dD;
        _alphas[0, 1] = (_vertices[1].Z - _vertices[2].Z) / dD;
        _alphas[0, 2] = (_vertices[2].R - _vertices[1].R) / dD;

        _alphas[1, 0] = ((_vertices[2].R * _vertices[0].Z) - (_vertices[0].R *
        ↪   _vertices[2].Z)) / dD;
        _alphas[1, 1] = (_vertices[2].Z - _vertices[0].Z) / dD;
        _alphas[1, 2] = (_vertices[0].R - _vertices[2].R) / dD;

        _alphas[2, 0] = ((_vertices[0].R * _vertices[1].Z) - (_vertices[1].R *
        ↪   _vertices[0].Z)) / dD;
        _alphas[2, 1] = (_vertices[0].Z - _vertices[1].Z) / dD;
        _alphas[2, 2] = (_vertices[1].R - _vertices[0].R) / dD;
    }

    private void AssemblyLocalMatrices(int itime, double t03, double t02, double t01) {
        double dD = Math.Abs(DeterminantD());
        CalcAlphas();

        //rs=[dl1^2 dl1dl2 dl1dl3 dl2^2 dl2dl3 dl3dl3]
        //каждая из 6 пар дифференциалов состоит из 3 составляющих (т.к. вектор r=r1L1+r2L2+r3L3)
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                _stiffnessMatrix[i, j] += _G[i][j][0][0] * _vertices[0].R * _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) + (_alphas[0, 2] * _alphas[0, 2])) * dD;
                _stiffnessMatrix[i, j] += _G[i][j][0][1] * _vertices[1].R * _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) + (_alphas[0, 2] * _alphas[0, 2])) * dD;
                _stiffnessMatrix[i, j] += _G[i][j][0][2] * _vertices[2].R * _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[0, 1]) + (_alphas[0, 2] * _alphas[0, 2])) * dD;

                _stiffnessMatrix[i, j] += _G[i][j][1][0] * _vertices[0].R * _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 * _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) + (_alphas[0, 2] * _alphas[1, 2])) * dD;
                _stiffnessMatrix[i, j] += _G[i][j][1][2] * _vertices[2].R * _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[1, 1]) + (_alphas[0, 2] * _alphas[1, 2])) * dD;

                _stiffnessMatrix[i, j] += _G[i][j][2][0] * _vertices[0].R * _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 * _spaceGrid.Lambda * ((_alphas[0, 1] * _alphas[2, 1]) + (_alphas[0, 2] * _alphas[2, 2])) * dD;
                _stiffnessMatrix[i, j] += _G[i][j][2][2] * _vertices[2].R * _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 * _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 * _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) + (_alphas[1, 2] * _alphas[1, 2])) * dD;
                _stiffnessMatrix[i, j] += _G[i][j][3][2] * _vertices[2].R * _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[1, 1]) + (_alphas[1, 2] * _alphas[1, 2])) * dD;

                _stiffnessMatrix[i, j] += _G[i][j][4][0] * _vertices[0].R * _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 * _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 * _spaceGrid.Lambda * ((_alphas[1, 1] * _alphas[2, 1]) + (_alphas[1, 2] * _alphas[2, 2])) * dD;

                _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 *
                ↪   _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 *
                ↪   _spaceGrid.Lambda * ((_alphas[2, 1] * _alphas[2, 1]) +
                ↪   (_alphas[2, 2] * _alphas[2, 2])) * dD;
            }
        }

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                for (int k = 0; k < 3; k++) {
                    _massMatrix[i, j] += _M[i][j][k] * _vertices[k].R *
                    ↪   _spaceGrid.Sigma * dD;
                }
            }
        }

        _massMatrix.Copy(_massMatrixCopy);

        if (itime == 1) {
            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                    _massMatrix[i, j] *= 1.0 / t01;
                }
            }
        } else if (itime == 2) {
            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                    _massMatrix[i, j] *= (t02 + t01) / (t02 * t01);
                }
            }
        } else {
            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                    _massMatrix[i, j] *= (1.0 / t03) + (1.0 / t02) + (1.0 / t01);
                }
            }
        }
    }

    private void AccountingDirichletBoundaries(int itime) {
        (int Node, double Value)[] boundaries = new (int, double)[4 *
        ↪   _dirichletBoundaries.Length];
        int[] checkBC = new int[_spaceGrid.Points.Count];

        int index = 0;

        for (int i = 0; i < _dirichletBoundaries.Length; i++) {
            int ielem = _dirichletBoundaries[i].Element;
            int edge = _dirichletBoundaries[i].Edge;

            for (int j = 0; j < 4; j++) {
                boundaries[index++] = (_spaceGrid.Edges[ielem][edge][j],
                                      _test.U(_spaceGrid.Points[_spaceGrid.Edges[iele
                                      ↪   m][edge][j]],
                                      ↪   _timeGrid.Points[itime]));
            }
        }

        boundaries = boundaries.Distinct().OrderBy(boundary =>
        ↪   boundary.Node).ToArray();
        checkBC.Fill(-1);

        for (int i = 0; i < boundaries.Length; i++)
            checkBC[boundaries[i].Node] = i;

        for (int i = 0; i < _spaceGrid.Points.Count; i++) {
            if (checkBC[i] != -1) {
                _globalMatrix.di[i] = 1;
                _vector[i] = boundaries[checkBC[i]].Value;

                for (int k = _globalMatrix.ig[i]; k < _globalMatrix.ig[i + 1]; k++) {
                    index = _globalMatrix.jg[k];

                    if (checkBC[index] == -1)
                        _vector[index] -= _globalMatrix.gg[k] * _vector[i];

                    _globalMatrix.gg[k] = 0;
                }
            } else {
                for (int k = _globalMatrix.ig[i]; k < _globalMatrix.ig[i + 1]; k++) {
                    index = _globalMatrix.jg[k];

                    if (checkBC[index] != -1) {
                        _vector[i] -= _globalMatrix.gg[k] * _vector[index];
                        _globalMatrix.gg[k] = 0;
                    }
                }
            }
        }
    }

    private void AccountingNeumannBoundaries() {
        Vector<double> localVector = new(10);
        int[] localEdge = new int[4];

        for (int i = 0; i < _neumannBoundaries!.Length; i++) {
            int ielem = _neumannBoundaries[i].Element;

            _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
            _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
            _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];

            CalcAlphas();

            // second else if you want to enter a single element

            if (_neumannBoundaries[i].Edge == 0) {
                localEdge[0] = 0;
                localEdge[1] = 8;
                localEdge[2] = 7;
                localEdge[3] = 2;
            } else if (_neumannBoundaries[i].Edge == 1) {
                localEdge[0] = 0;
                localEdge[1] = 3;
                localEdge[2] = 4;
                localEdge[3] = 1;
            } else {
                localEdge[0] = 1;
                localEdge[1] = 5;
                localEdge[2] = 6;
                localEdge[3] = 2;
            }

            for (int j = 0; j < 4; j++) {
                localVector[localEdge[j]] = _spaceGrid.Lambda * _neumannBoundaries[i].Value *
                                  Integration.GaussSegment(_basis[localEdge[j]].Invoke,
                                  _spaceGrid.Points[_spaceGrid.Edges[ielem][_neumannBoundaries[i].Edge][0]],
                                  _spaceGrid.Points[_spaceGrid.Edges[ielem][_neumannBoundaries[i].Edge][3]]);
            }

            for (int j = 0; j < 10; j++) {
                _vector[_spaceGrid.Elements[ielem][j]] += localVector[j];
            }

            localVector.Fill(0);
        }
    }

    public double[] ValueAtPoint(Point2D[] points) {
        try {
            ArgumentNullException.ThrowIfNull(_solver.Solution, $"{nameof(_solver.Solution)} cannot be null, solve problem first");

            double[] result = new double[points.Length];

            for (int ipoint = 0; ipoint < points.Length; ipoint++) {
                int ielem = FindElement(points[ipoint]);
                double determinant = DeterminantD();
                CalcAlphas();

                for (int i = 0; i < 10; i++) {
                    result[ipoint] += _solver.Solution!.Value[_spaceGrid.Elements[ielem][i]] *
                                      _basis[i](points[ipoint]);
                }
            }

            return result;
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
            return Array.Empty<double>();
        }
    }

    private int FindElement(Point2D point) {
        for (int ielem = 0; ielem < _spaceGrid.Elements.Length; ielem++) {
            _vertices[0] = _spaceGrid.Points[_spaceGrid.Elements[ielem][0]];
            _vertices[1] = _spaceGrid.Points[_spaceGrid.Elements[ielem][1]];
            _vertices[2] = _spaceGrid.Points[_spaceGrid.Elements[ielem][2]];

            double a = ((_vertices[0].R - point.R) * (_vertices[1].Z - _vertices[0].Z)) -
```

```
                     ((_vertices[1].R - _vertices[0].R) * (_vertices[0].Z - point.Z));

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

            double c = ((_vertices[2].R - point.R) * (_vertices[0].Z - _vertices[2].Z)) -
                     ((_vertices[0].R - _vertices[2].R) * (_vertices[2].Z - point.Z));

            if ((a >= 0 && b >= 0 && c >= 0) || (a <= 0 && b <= 0 && c <= 0))
                return ielem;
        }

        throw new Exception("The point does not belong to the grid");
    }

    public void WriteToFile(string path) {
        using var sw = new StreamWriter(path);
        for (int i = 0; i < _solver.Solution!.Value.Length; i++) {
            sw.WriteLine(i + ") " + _solver.Solution!.Value[i]);
        }
    }

    // for report
    private void Err(int itime) {
        double[] err = new double[_spaceGrid.Points.Count];
        double[] exact = new double[_spaceGrid.Points.Count];
        double sum = 0.0;

        using StreamWriter sw1 = new("results/err.txt"),
                           sw2 = new("results/Exact.txt"),
                           sw3 = new("results/rms.txt"),
                           sw4 = new("csv2/2.csv", true);

        for (int i = 0; i < _spaceGrid.Points.Count; i++) {
            err[i] = Math.Abs(_solver.Solution!.Value[i] - _test.U(_spaceGrid.Points[i], _timeGrid.Points[itime]));
            sw1.WriteLine(err[i]);
        }

        for (int i = 0; i < _spaceGrid.Points.Count; i++) {
            exact[i] = _test.U(_spaceGrid.Points[i], _timeGrid.Points[itime]);
            sw2.WriteLine(exact[i]);
        }

        for (int i = 0; i < _spaceGrid.Points.Count; i++) {
            sum += err[i] * err[i];
        }

        // sum = Math.Sqrt(sum); // относительная погрешность
        // sum /= exact.Norm();

        sum = Math.Sqrt(sum / _spaceGrid.Points.Count); // среднеквадратичное отклонение
        sw3.Write(sum);

        // sw4.WriteLine("$r $z, Точное, Численное, Вектор погрешности, Погрешность");

        // for (int i = 0; i < _spaceGrid.Points.Count; i++) {
```

```
        //     if (i == 0) {
        //         sw4.WriteLine($"{_spaceGrid.Points[i]},{exact[i]},{_solver.Solution!.Value[i]},{err[i].ToString(err[i] == 0 ? "0" : "0.00E+0")},{sum:0.00E+0}");
        //     }

        //     sw4.WriteLine($"{_spaceGrid.Points[i]},{exact[i]},{_solver.Solution!.Value[i]},{err[i].ToString(err[i] == 0 ? "0" : "0.00E+0")},");
        // }

        if (itime == 1) {
            sw4.WriteLine("$t_i$, Погрешность");
        }

        sw4.WriteLine($"{_timeGrid.Points[itime]:0.0000}, {sum:0.00E+0}");
    }

    public static FEMBuilder CreateBuilder()
            => new();

    // костыль
    public static Matrix GetAlphas()
        => _alphas;
}
```

### SparseMatrix.cs

```
namespace courseProject;

public class SparseMatrix {
    // public fields - its bad, but the readability is better
    public int[] ig = default!;
    public int[] jg = default!;
    public double[] di = default!;
    public double[] gg = default!;
    public int Size { get; init; }

    public SparseMatrix(int size, int sizeOffDiag) {
        Size = size;
        ig = new int[size + 1];
        jg = new int[sizeOffDiag];
        gg = new double[sizeOffDiag];
        di = new double[size];
    }

    public static Vector<double> operator *(SparseMatrix matrix, Vector<double> vector) {
        Vector<double> product = new(vector.Length);

        for (int i = 0; i < vector.Length; i++) {
            product[i] = matrix.di[i] * vector[i];

            for (int j = matrix.ig[i]; j < matrix.ig[i + 1]; j++) {
                product[i] += matrix.gg[j] * vector[matrix.jg[j]];
                product[matrix.jg[j]] += matrix.gg[j] * vector[i];
            }
        }

        return product;
    }

    public void PrintDense(string path) {
        double[,] A = new double[Size, Size];

        for (int i = 0; i < Size; i++) {
            A[i, i] = di[i];

            for (int j = ig[i]; j < ig[i + 1]; j++) {
                A[i, jg[j]] = gg[j];
                A[jg[j], i] = gg[j];
            }
        }

        using var sw = new StreamWriter(path);
        for (int i = 0; i < Size; i++) {
            for (int j = 0; j < Size; j++) {
                sw.Write(A[i, j].ToString("0.00") + "\t");
            }

            sw.WriteLine();
        }
    }

    public void Clear() {
        for (int i = 0; i < Size; i++) {
            di[i] = 0.0;

            for (int k = ig[i]; k < ig[i + 1]; k++) {
                gg[k] = 0.0;
            }
        }
    }
}
```

### Matrix.cs

```
namespace courseProject;

public class Matrix {
    private readonly double[,] storage;
    public int Size { get; init; }

    public double this[int i, int j] {
        get => storage[i, j];
        set => storage[i, j] = value;
    }

    public Matrix(int size) {
        storage = new double[size, size];
        Size = size;
    }

    public void Clear()
        => Array.Clear(storage, 0, storage.Length);

    public void Copy(Matrix destination) {
        for (int i = 0; i < destination.Size; i++) {
            for (int j = 0; j < destination.Size; j++) {
                destination[i, j] = storage[i, j];
            }
        }
    }

    public static Matrix operator +(Matrix fstMatrix, Matrix sndMatrix) {
        Matrix resultMatrix = new(fstMatrix.Size);

        for (int i = 0; i < resultMatrix.Size; i++) {
            for (int j = 0; j < resultMatrix.Size; j++) {
                resultMatrix[i, j] = fstMatrix[i, j] + sndMatrix[i, j];
            }
        }

        return resultMatrix;
    }
}
```

### Vector.cs

```
namespace courseProject;

public class Vector<T> where T : INumber<T> {
    private readonly T[] vec;
    public int Length { get; init; }

    public T this[int index] {
        get => vec[index];
        set => vec[index] = value;
    }

    public Vector(int dim) {
        vec = new T[dim];
        Length = dim;
    }

    public static T operator *(Vector<T> firstVec, Vector<T> secondVec) {
        T result = T.Zero;

        for (int i = 0; i < firstVec.Length; i++) {
            result += firstVec[i] * secondVec[i];
        }

        return result;
    }

    public static Vector<T> operator *(double constant, Vector<T> vector) {
        Vector<T> result = new(vector.Length);

        for (int i = 0; i < vector.Length; i++) {
            result.vec[i] = vector[i] * T.Create(constant);
        }

        return result;
    }

    public static Vector<T> operator +(Vector<T> firstVec, Vector<T> secondVec) {
        Vector<T> result = new(firstVec.Length);

        for (int i = 0; i < firstVec.Length; i++) {
            result.vec[i] = firstVec[i] + secondVec[i];
        }

        return result;
    }

    public static Vector<T> operator -(Vector<T> firstVec, Vector<T> secondVec) {
        Vector<T> result = new(firstVec.Length);

        for (int i = 0; i < firstVec.Length; i++) {
            result.vec[i] = firstVec[i] - secondVec[i];
        }

        return result;
    }

    public static void Copy(Vector<T> source, Vector<T> destination) {
        for (int i = 0; i < source.Length; i++) {
            destination[i] = source[i];
        }
    }

    public void Fill(double value) {
        for (int i = 0; i < Length; i++) {
            vec[i] = T.Create(value);
        }
    }

    public double Norm() {
        T result = T.Zero;

        for (int i = 0; i < Length; i++) {
            result += vec[i] * vec[i];
        }

        return Math.Sqrt(Convert.ToDouble(result));
    }

    public ImmutableArray<T> ToImmutableArray()
        => ImmutableArray.Create(vec);
}
```

## Solvers.cs

```
namespace courseProject;

public abstract class Solver {
    protected SparseMatrix _matrix = default!;
    protected Vector<double> _vector = default!;
    protected Vector<double>? _solution;
    public int MaxIters { get; init; }
    public double Eps { get; init; }
    public ImmutableArray<double>? Solution => _solution?.ToImmutableArray();

    protected Solver(int maxIters, double eps)
        => (MaxIters, Eps) = (maxIters, eps);

    public void SetMatrix(SparseMatrix matrix)
        => _matrix = matrix;

    public void SetVector(Vector<double> vector)
```

```
        => _vector = vector;

    public abstract void Compute();
}

public class LOS : Solver {
    public LOS(int maxIters, double eps) : base(maxIters, eps) { }

    public override void Compute() {
        try {
            ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
            ↪   null, set the matrix");
            ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
            ↪   null, set the vector");

            double alpha, beta;
            double squareNorm;

            _solution = new(_vector.Length);

            Vector<double> r = new(_vector.Length);
            Vector<double> z = new(_vector.Length);
            Vector<double> p = new(_vector.Length);
            Vector<double> tmp = new(_vector.Length);

            r = _vector - (_matrix * _solution);

            Vector<double>.Copy(r, z);

            p = _matrix * z;

            squareNorm = r * r;

            for (int iter = 0; iter < MaxIters && squareNorm > Eps; iter++) {
                alpha = p * r / (p * p);
                _solution += alpha * z;
                squareNorm = (r * r) - (alpha * alpha * (p * p));
                r -= alpha * p;

                tmp = _matrix * r;

                beta = -(p * tmp) / (p * p);
                z = r + (beta * z);
                p = tmp + (beta * p);
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}

public class CGM : Solver {
    public CGM(int maxIters, double eps) : base(maxIters, eps) { }

    public override void Compute() {
        try {
            ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
            ↪   null, set the matrix");
            ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
            ↪   null, set the vector");
            double alpha, beta;
            double norm, squareNorm;
            double vectorNorm = _vector.Norm();

            _solution = new(_vector.Length);

            Vector<double> r = new(_vector.Length);
            Vector<double> z = new(_vector.Length);
            Vector<double> p = new(_vector.Length);
            Vector<double> tmp = new(_vector.Length);

            r = _vector - (_matrix * _solution);

            Vector<double>.Copy(r, z);

            for (int iter = 0; iter < MaxIters && (norm = r.Norm() / vectorNorm) >= Eps; iter++) {
                tmp = _matrix * z;
                alpha = r * r / (tmp * z);
                _solution += alpha * z;
                squareNorm = r * r;
                r -= alpha * tmp;
                beta = r * r / squareNorm;
                z = r + (beta * z);
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}

public class CGMCholesky : Solver {
    public CGMCholesky(int maxIters, double eps) : base(maxIters, eps) { }

    public override void Compute() {
        try {
            double alpha, beta;
            double tmp;

            double vectorNorm = _vector.Norm();

            _solution = new(_vector.Length);

            double[] ggnew = new double[_matrix.gg.Length];
            double[] dinew = new double[_matrix.di.Length];

            _matrix.gg.Copy(ggnew);
            _matrix.di.Copy(dinew);

            Vector<double> r = new(_vector.Length);
            Vector<double> z = new(_vector.Length);
            Vector<double> fstTemp = new(_vector.Length);
            Vector<double> sndTemp = new(_vector.Length);

            Cholesky(ggnew, dinew);

            r = _vector - (_matrix * _solution);
            z = MoveForCholesky(r, ggnew, dinew);

            for (int iter = 0; iter < MaxIters && r.Norm() / vectorNorm >= Eps; iter++) {
                tmp = MoveForCholesky(r, ggnew, dinew) * r;
                sndTemp = _matrix * z;
                alpha = tmp / (sndTemp * z);
                _solution += alpha * z;
                r -= alpha * sndTemp;
                fstTemp = MoveForCholesky(r, ggnew, dinew);
                beta = fstTemp * r / tmp;
                z = fstTemp + (beta * z);
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }

    private void Cholesky(double[] ggnew, double[] dinew) {
        double suml = 0.0;
        double sumdi = 0.0;

        for (int i = 0; i < _matrix.Size; i++) {
            int i0 = _matrix.ig[i];
            int i1 = _matrix.ig[i + 1];

            for (int k = i0; k < i1; k++) {
                int j = _matrix.jg[k];
                int j0 = _matrix.ig[j];
                int j1 = _matrix.ig[j + 1];
                int ik = i0;
                int kj = j0;

                while (ik < k && kj < j1) {
                    if (_matrix.jg[ik] == _matrix.jg[kj]) {
                        suml += ggnew[ik] * ggnew[kj];
                        ik++;
                        kj++;
                    } else {
                        if (_matrix.jg[ik] > _matrix.jg[kj])
                            kj++;
                        else
                            ik++;
                    }
                }

                ggnew[k] = (ggnew[k] - suml) / dinew[j];
                sumdi += ggnew[k] * ggnew[k];
                suml = 0.0;
            }

            dinew[i] = Math.Sqrt(dinew[i] - sumdi);
            sumdi = 0.0;
        }
    }

    private Vector<double> MoveForCholesky(Vector<double> vector, double[] ggnew, double[] dinew) {
        Vector<double> y = new(vector.Length);
        Vector<double> x = new(vector.Length);
        Vector<double>.Copy(vector, y);

        double sum = 0.0;

        for (int i = 0; i < _matrix.Size; i++) // Прямой ход
        {
            int i0 = _matrix.ig[i];
            int i1 = _matrix.ig[i + 1];

            for (int k = i0; k < i1; k++)
                sum += ggnew[k] * y[_matrix.jg[k]];

            y[i] = (y[i] - sum) / dinew[i];
            sum = 0.0;
        }

        Vector<double>.Copy(y, x);

        for (int i = _matrix.Size - 1; i >= 0; i--) // Обратный ход
        {
            int i0 = _matrix.ig[i];
            int i1 = _matrix.ig[i + 1];
            x[i] = y[i] / dinew[i];

            for (int k = i0; k < i1; k++)
                y[_matrix.jg[k]] -= ggnew[k] * x[i];
        }

        return x;
    }
}
```

## Integration.cs

```
namespace courseProject;

public static class Integration {
    public static double GaussSegment(Func<Point2D, double> psi, Point2D firstPoint,
     ↪  Point2D secondPoint) {
        var quadratures = Quadratures.SegmentGaussOrder9();

        double result = 0.0;
        double lenghtEdge = Math.Sqrt(((firstPoint.R - secondPoint.R) * (firstPoint.R
         ↪  - secondPoint.R)) +
                            ((firstPoint.Z - secondPoint.Z) * (firstPoint.Z -
                             ↪  secondPoint.Z)));

        foreach (var q in quadratures) {
            double qi = q.Weight;
            double pi = (1 + q.Node) / 2.0;

            var parameterized = Parameterization(pi);

            result += qi * psi(parameterized) * parameterized.R;
        }

        return result * lenghtEdge / 2.0;

        Point2D Parameterization(double t)
            => ((secondPoint- firstPoint) * t) + firstPoint;
    }

    public static double Triangle(Func<Point2D, double, double> f, Func<Point2D,
     ↪  double> psi, Point2D[] vertices, double t) {
        var quadratures = Quadratures.TriangleOrder6();

        double result = 0.0;
        double determinant = Math.Abs(Determinant());

        foreach (var q in quadratures) {
            var point = ((1 - q.Node.R - q.Node.Z) * vertices[0]) + (q.Node.R *
             ↪  vertices[1]) + (q.Node.Z * vertices[2]);

            result += f(point, t) * psi(point) * q.Weight * determinant * 0.5 *
             ↪  point.R;
        }

        return result;

        double Determinant()
        => ((vertices[1].R - vertices[0].R) * (vertices[2].Z - vertices[0].Z)) -
           ((vertices[2].R - vertices[0].R) * (vertices[1].Z - vertices[0].Z));
    }
}
```

### Quadratures.cs

```
namespace courseProject;

public class QuadratureNode<T> where T : notnull {
    public T Node { get; init; }
    public double Weight { get; init; }

    public QuadratureNode(T node, double weight) {
        Node = node;
        Weight = weight;
    }
}

public class Quadrature<T> where T : notnull {
    private readonly QuadratureNode<T>[] _nodes = default!;
    public ImmutableArray<QuadratureNode<T>> Nodes => _nodes.ToImmutableArray();

    public Quadrature(QuadratureNode<T>[] nodes) {
        _nodes = nodes;
    }
}

public static class Quadratures {
    public static IEnumerable<QuadratureNode<double>> SegmentGaussOrder9() {
        const int n = 5;
        double[] points = { 0.0,
                            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))),
                            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)))};

        double[] weights = { 128.0 / 225.0,
                            (322.0 + (13.0 * Math.Sqrt(70.0))) / 900.0,
                            (322.0 + (13.0 * Math.Sqrt(70.0))) / 900.0,
                            (322.0 - (13.0 * Math.Sqrt(70.0))) / 900.0,
                            (322.0 - (13.0 * Math.Sqrt(70.0))) / 900.0 };

        for (int i = 0; i < n; i++) {
            yield return new QuadratureNode<double>(points[i], weights[i]);
        }
    }

    public static IEnumerable<QuadratureNode<Point2D>> TriangleOrder6() {
        const double x1a = 0.873821971016996;
        const double x1b = 0.063089014491502;
        const double x2a = 0.501426509658179;
        const double x2b = 0.249286745170910;
        const double x3a = 0.636502499121399;
        const double x3b = 0.310352451033785;
        const double x3c = 0.053145049844816;
        const double w1 = 0.050844906370207;
        const double w2 = 0.116786275726379;
        const double w3 = 0.082851075618374;

        double[] p1 = { x1a, x1b, x1b, x2a, x2b, x2b, x3a, x3b, x3a, x3c, x3b, x3c };
        double[] p2 = { x1b, x1a, x1b, x2b, x2a, x2b, x3b, x3a, x3c, x3a, x3c, x3b };
        double[] w = { w1, w1, w1, w2, w2, w2, w3, w3, w3, w3, w3, w3 };

        for (int i = 0; i < w.Length; i++) {
            yield return new QuadratureNode<Point2D>(new(p1[i], p2[i]), w[i]);
        }
    }
}
```

### CubicBasis.cs

```
namespace courseProject;

public static class CubicBasis {
    private static readonly Matrix _alphas;

    static CubicBasis() {
        _alphas = FEM.GetAlphas();
    }

    private static double L1(Point2D point)
        => _alphas[0, 0] + (_alphas[0, 1] * point.R) + (_alphas[0, 2] * point.Z);

    private static double L2(Point2D point)
        => _alphas[1, 0] + (_alphas[1, 1] * point.R) + (_alphas[1, 2] * point.Z);

    private static double L3(Point2D point)
        => _alphas[2, 0] + (_alphas[2, 1] * point.R) + (_alphas[2, 2] * point.Z);

    public static double Psi1(Point2D point) {
        double l1 = L1(point);
        return 0.5 * l1 * ((3 * l1) - 1) * ((3 * l1) - 2);
    }
    public static double Psi2(Point2D point) {
        double l2 = L2(point);
        return 0.5 * l2 * ((3 * l2) - 1) * ((3 * l2) - 2);
    }

    public static double Psi3(Point2D point) {
        double l3 = L3(point);
        return 0.5 * l3 * ((3 * l3) - 1) * ((3 * l3) - 2);
    }

    public static double Psi4(Point2D point) {
        double l1 = L1(point);
        double l2 = L2(point);
        return 4.5 * l1 * l2 * ((3 * l1) - 1);
    }

    public static double Psi5(Point2D point) {
        double l1 = L1(point);
        double l2 = L2(point);
        return 4.5 * l1 * l2 * ((3 * l2) - 1);
    }

    public static double Psi6(Point2D point) {
        double l2 = L2(point);
        double l3 = L3(point);
        return 4.5 * l2 * l3 * ((3 * l2) - 1);
    }

    public static double Psi7(Point2D point) {
        double l2 = L2(point);
        double l3 = L3(point);
        return 4.5 * l2 * l3 * ((3 * l3) - 1);
    }

    public static double Psi8(Point2D point) {
        double l1 = L1(point);
        double l3 = L3(point);
        return 4.5 * l1 * l3 * ((3 * l3) - 1);
    }

    public static double Psi9(Point2D point) {
        double l1 = L1(point);
        double l3 = L3(point);
        return 4.5 * l1 * l3 * ((3 * l1) - 1);
    }

    public static double Psi10(Point2D point) {
        double l1 = L1(point);
        double l2 = L2(point);
        double l3 = L3(point);
        return 27 * l1 * l2 * l3;
    }
}
```

### Boundaries.cs

```
namespace courseProject;

public readonly record struct DirichletBoundary(int Element, int Edge) {
    public static DirichletBoundary[]? ReadJson(string jsonPath) {
        try {
            if (!File.Exists(jsonPath))
                throw new Exception("File does not exist");

            var sr = new StreamReader(jsonPath);
            using (sr) {
                return JsonConvert.DeserializeObject<DirichletBoundary[]>(sr.ReadToEnd());
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
            return null;
        }
    }
}

public readonly record struct NeumannBoundary(int Element, int Edge, double Value) {
    public static NeumannBoundary[]? ReadJson(string jsonPath) {
        try {
            if (!File.Exists(jsonPath))
                throw new Exception("File does not exist");

            var sr = new StreamReader(jsonPath);
            using (sr) {
                return
                 ↪  JsonConvert.DeserializeObject<NeumannBoundary[]>(sr.ReadToEnd());
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
            return null;
        }
    }
}
```

## Tests.cs

```
namespace courseProject;

public interface ITest {
    public double U(Point2D point, double t);

    public double F(Point2D point, double t);
}

public class Test1 : ITest {
    public double U(Point2D point, double t)
        => (point.R * point.R) + point.Z + t;

    public double F(Point2D point, double t)
        => -3;
}

public class Test2 : ITest {
    public double U(Point2D point, double t)
        => (point.R * point.R) - (point.Z * t * t);

    public double F(Point2D point, double t)
        => -8 - (t * point.Z);
}

public class Test3 : ITest {
    public double U(Point2D point, double t)
        => 2 * point.R * point.R * point.R * t * t * t;

    public double F(Point2D point, double t)
        => (-9 * point.R * t * t * t) + (12 * point.R * point.R * point.R * t * t);
}

```

```
public class Test4 : ITest {
    public double U(Point2D point, double t)
       => (point.R * point.R * point.R * point.R) + (point.Z * point.Z * point.Z * point.Z) + (t * t * t * t);

    public double F(Point2D point, double t)
        => (-16 * point.R * point.R) - (12 * point.Z * point.Z) + (4 * t * t * t);
}

public class Test5 : ITest {
    public double U(Point2D point, double t)
        => Math.Log(point.R);

    public double F(Point2D point, double t)
        => 0;
}

public class Test6 : ITest {
    public double U(Point2D point, double t)
         => Math.Cos(t);

    public double F(Point2D point, double t)
        => -Math.Sin(t);
}

public class Test7 : ITest {
    public double U(Point2D point, double t)
        => t * t * t * t;

    public double F(Point2D point, double t)
        => 4 * t * t * t;
}
```

### GridsParameters.cs

```
namespace courseProject;

public class SpaceGridParametersJsonConverter : JsonConverter {
    public override void WriteJson(JsonWriter writer, object? value, JsonSerializer serializer) {
        if (value is null) {
            writer.WriteNull();
            return;
        }

        var gridParameters = (SpaceGridParameters)value;

        writer.WriteStartObject();
        writer.WritePropertyName("Initial area in R");
        serializer.Serialize(writer, gridParameters.IntervalR);
        writer.WritePropertyName("Splits by R");
        writer.WriteValue(gridParameters.SplitsR);
        writer.WriteWhitespace("\n");

        writer.WritePropertyName("Initial area in Z");
        serializer.Serialize(writer, gridParameters.IntervalZ);
        writer.WritePropertyName("Splits by Z");
        writer.WriteValue(gridParameters.SplitsZ);

        writer.WriteComment("Коэффициент разрядки");
        writer.WritePropertyName("Coef");
        writer.WriteValue(gridParameters.K);
        writer.WriteWhitespace("\n");
        writer.WriteComment("Uniform area");
        writer.WritePropertyName("Lambda");
        writer.WriteValue(gridParameters.Lambda);
        writer.WritePropertyName("Sigma");
        writer.WriteValue(gridParameters.Sigma);
        writer.WriteEndObject();
    }

    public override object? ReadJson(JsonReader reader, Type objectType, object? existingValue, JsonSerializer serializer) {
        if (reader.TokenType == JsonToken.Null || reader.TokenType != JsonToken.StartObject)
            return null;

        Interval intervalR;
        Interval intervalZ;
        int splitsR;
        int splitsZ;
        double? coef;
        double lambda, sigma;

        var maintoken = JObject.Load(reader);

        var token = maintoken["Initial area in R"];
        intervalR = serializer.Deserialize<Interval>(token!.CreateReader());
        token = maintoken["Splits by R"];
        splitsR = Convert.ToInt32(token);

        token = maintoken["Initial area in Z"];
        intervalZ = serializer.Deserialize<Interval>(token!.CreateReader());
        token = maintoken["Splits by Z"];
        splitsZ = Convert.ToInt32(token);

        token = maintoken["Coef"];

        if (token is not null) {
            coef = double.TryParse(token.ToString(), out double res) ? res : null;
        }
        else {
            coef = null;
        }

        token = maintoken["Lambda"];
        lambda = Convert.ToDouble(token);

        token = maintoken["Sigma"];

        if (token is not null) {
            sigma = double.TryParse(token.ToString(), out double res) ? res : 0.0;
        }
        else {
            sigma = 0.0;
        }

        return new SpaceGridParameters(intervalR, splitsR, intervalZ, splitsZ, coef, lambda, sigma);
    }

    public override bool CanConvert(Type objectType)
        => objectType == typeof(SpaceGridParameters);
}

[JsonConverter(typeof(SpaceGridParametersJsonConverter))]
public readonly record struct SpaceGridParameters {
    public Interval IntervalR { get; init; }
    public int SplitsR { get; init; }
    public Interval IntervalZ { get; init; }
    public int SplitsZ { get; init; }
    public double? K { get; init; }

    // Uniform area
    public double Lambda { get; init; }
    public double Sigma { get; init; }

    public SpaceGridParameters(Interval intervalR, int splitsR, Interval intervalZ, int splitsZ, double? k, double lambda, double sigma) {
        IntervalR = intervalR;
        SplitsR = splitsR;
        IntervalZ = intervalZ;
        SplitsZ = splitsZ;
        K = k;
        Lambda = lambda;
        Sigma = sigma;
    }

    public static SpaceGridParameters? ReadJson(string jsonPath) {
        try {
            if (!File.Exists(jsonPath))
                throw new Exception("File does not exist");

            var sr = new StreamReader(jsonPath);
            using (sr) {
                return JsonConvert.DeserializeObject<SpaceGridParameters>(sr.ReadToEnd());
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
            return null;
        }
    }
}

public readonly record struct TimeGridParameters {
    [JsonProperty("Initial area")]
    public Interval Interval { get; init; }

    [JsonProperty("Number of splits")]
    public int Splits { get; init; }

    [JsonProperty("Coef")]
    public double? K { get; init; } // коэффициент разрядки

    public TimeGridParameters(Interval interval, int splits, double? k) {
        Interval = interval;
        Splits = splits;
        K = k;
    }

    public static TimeGridParameters? ReadJson(string jsonPath) {
        try {
            if (!File.Exists(jsonPath))
                throw new Exception("File does not exist");

            var sr = new StreamReader(jsonPath);
            using (sr) {
                return
                 ↪  JsonConvert.DeserializeObject<TimeGridParameters>(sr.ReadToEnd());
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
            return null;
        }
    }
}
```

## IGrids.cs

```
namespace courseProject;

public interface ISpaceGrid {
    public double Lambda { get; init; }
    public double Sigma { get; init; }
    public ImmutableList<Point2D> Points { get; }
    public ImmutableArray<ImmutableArray<int>> Elements { get; }
    public ImmutableArray<ImmutableArray<ImmutableArray<int>>> Edges { get; }
}

public interface ITimeGrid {
    public ImmutableArray<double> Points { get; }
}
```

## GridFactories.cs

```
namespace courseProject;

public enum GridTypes {
    SpaceRegular,
    TimeRegular,
    TimeIrregular
}

public interface ISpaceFactory {
    public ISpaceGrid CreateGrid(GridTypes spaceGridType, SpaceGridParameters
     ↪  gridParameters);
}

public interface ITimeFactory {
    public ITimeGrid CreateGrid(GridTypes timeGridType, TimeGridParameters
     ↪  gridParameters);
}

public class SpaceGridFactory : ISpaceFactory {
    public ISpaceGrid CreateGrid(GridTypes spaceGridType, SpaceGridParameters
     ↪  gridParameters) {
        return spaceGridType switch {
            GridTypes.SpaceRegular => new SpaceRegularGrid(gridParameters),
            _ => throw new ArgumentOutOfRangeException(nameof(spaceGridType), $"This
            ↪  type of grid does not exist: {spaceGridType}"),
        };
    }
}

public class TimeGridFactory : ITimeFactory {
    public ITimeGrid CreateGrid(GridTypes timeGridType, TimeGridParameters
    ↪  gridParameters) {
        return timeGridType switch {
            GridTypes.TimeRegular => new TimeRegularGrid(gridParameters),

            GridTypes.TimeIrregular => new TimeIrregularGrid(gridParameters),

            _ => throw new ArgumentOutOfRangeException(nameof(timeGridType), $"This
            ↪  type of grid does not exist: {timeGridType}"),
        };
    }
}
```

### SpaceGrids.cs

```
namespace courseProject;

public class SpaceRegularGrid : ISpaceGrid {
    private readonly List<Point2D> _points = default!;
    private readonly int[][] _elements = default!;
    private readonly int[][][] _edges = default!;
    public ImmutableList<Point2D> Points => _points.ToImmutableList();
    public ImmutableArray<ImmutableArray<int>> Elements => _elements.Select(item =>
    ↪  item.ToImmutableArray()).ToImmutableArray();
    public ImmutableArray<ImmutableArray<ImmutableArray<int>>> Edges =>
    ↪  _edges.Select(item => item.ToImmutableArray().Select(item =>
    ↪  item.ToImmutableArray()).ToImmutableArray()).ToImmutableArray();
    public double Lambda { get; init; }
    public double Sigma { get; init; }

    public SpaceRegularGrid(SpaceGridParameters gridParameters) {
        Lambda = gridParameters.Lambda;
        Sigma = gridParameters.Sigma;
        _points = new();
        _elements = new int[2 * gridParameters.SplitsR *
        ↪  gridParameters.SplitsZ].Select(_ => new int[10]).ToArray();
        _edges = new int[2 * gridParameters.SplitsR *
        ↪  gridParameters.SplitsZ].Select(_ => new int[3].ToArray().Select(_ => new
        ↪  int[4]).ToArray()).ToArray();
        Build(gridParameters);
    }

    private void Build(SpaceGridParameters gridParameters) {
        try {
            if (gridParameters.SplitsR < 1 || gridParameters.SplitsZ < 1) {
                throw new Exception("The number of splits must be greater than or
                ↪  equal to 1");
            }

            double hr = gridParameters.IntervalR.Lenght / gridParameters.SplitsR;
            double hz = gridParameters.IntervalZ.Lenght / gridParameters.SplitsZ;

            double[] pointsR = new double[(3 * gridParameters.SplitsR) + 1];
            double[] pointsZ = new double[(3 * gridParameters.SplitsZ) + 1];

            pointsR[0] = gridParameters.IntervalR.LeftBorder;
            pointsZ[0] = gridParameters.IntervalZ.LeftBorder;

            for (int i = 1; i < gridParameters.SplitsR + 1; i++) {
                pointsR[i] = pointsR[i - 1] + hr;
            }

            for (int i = 1; i < gridParameters.SplitsZ + 1; i++) {
                pointsZ[i] = pointsZ[i - 1] + hz;
            }

            for (int j = 0; j < gridParameters.SplitsZ + 1; j++) {
                for (int i = 0; i < gridParameters.SplitsR + 1; i++) {
                    _points.Add(new(pointsR[i], pointsZ[j]));
                }
            }

            Point2D[] allNodes = new Point2D[10];
            Point2D[] vertices = new Point2D[3];

            const int dummySplits = 2;
            int nr = gridParameters.SplitsR + 1;
            int index = 0;

            for (int j = 0; j < gridParameters.SplitsZ; j++) {
                for (int i = 0; i < gridParameters.SplitsR; i++) {
                    _elements[index][0] = i + (j * nr);
                    _elements[index][1] = i + 1 + (j * nr);
                    _elements[index++][2] = i + ((j + 1) * nr);

                    _elements[index][0] = i + 1 + (j * nr);
                    _elements[index][1] = i + nr + (j * nr);
                    _elements[index++][2] = i + 1 + nr + (j * nr);
                }
            }

            WritePoints("forGraphics/pointsLinear.txt");
            WriteElements("forGraphics/elements.txt");
            pointsR.Fill(-1.0);
            pointsZ.Fill(-1.0);

            for (int ielem = 0; ielem < _elements.Length; ielem++) {
                vertices[0] = _points[_elements[ielem][0]];
                vertices[1] = _points[_elements[ielem][1]];
                vertices[2] = _points[_elements[ielem][2]];

                GetNodes(allNodes, vertices);

                pointsR = pointsR.Concat(allNodes.Select(point => point.R)).ToArray();
                pointsR = pointsR.Distinct().OrderBy(value => value).ToArray();

                pointsZ = pointsZ.Concat(allNodes.Select(point => point.Z)).ToArray();
                pointsZ = pointsZ.Distinct().OrderBy(value => value).ToArray();
            }

            _points.Clear();

            foreach (var pointZ in pointsZ.Skip(1)) {
                foreach (var pointR in pointsR.Skip(1)) {
                    _points.Add(new(pointR, pointZ));
                }
            }

            nr = pointsR.Length - 1;
            index = 0;

            for (int j = 0; j < gridParameters.SplitsZ; j++) {
                for (int i = 0; i < gridParameters.SplitsR; i++) {
                    _elements[index][0] = i + (3 * j * nr) + (i * dummySplits); // 0
                    _elements[index][1] = i + (3 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); // 12
                    _elements[index][2] = i + 3 + (3 * j * nr) + (i * dummySplits);
                    ↪  // 3
                    _elements[index][3] = i + nr + (3 * j * nr) + (i * dummySplits);
                    ↪  // 4
                    _elements[index][4] = i + (2 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); // 8
                    _elements[index][5] = i + 1 + (2 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); //9
                    _elements[index][6] = i + nr + 2 + (3 * j * nr) + (i *
                    ↪  dummySplits); // 6
                    _elements[index][7] = i + 2 + (3 * j * nr) + (i * dummySplits);
                    ↪  // 2
                    _elements[index][8] = i + 1 + (3 * j * nr) + (i * dummySplits);
                    ↪  // 1
                    _elements[index][9] = i + nr + 1 + (3 * j * nr) + (i *
                    ↪  dummySplits); // 5

                    _edges[index][0][0] = _elements[index][0];
                    _edges[index][0][1] = _elements[index][8];
                    _edges[index][0][2] = _elements[index][7];
                    _edges[index][0][3] = _elements[index][2];

                    _edges[index][1][0] = _elements[index][0];
                    _edges[index][1][1] = _elements[index][3];
                    _edges[index][1][2] = _elements[index][4];
                    _edges[index][1][3] = _elements[index][1];

                    _edges[index][2][0] = _elements[index][2];
                    _edges[index][2][1] = _elements[index][6];
                    _edges[index][2][2] = _elements[index][5];
                    _edges[index][2][3] = _elements[index++][1];

                    _elements[index][0] = i + (3 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); // 12
                    _elements[index][1] = i + 3 + (3 * j * nr) + (i * dummySplits);
                    ↪  //3
                    _elements[index][2] = i + 3 + (3 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); // 15
                    _elements[index][3] = i + 1 + (2 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); // 9
                    _elements[index][4] = i + nr + 2 + (3 * j * nr) + (i *
                    ↪  dummySplits); // 6
                    _elements[index][5] = i + nr + 3 + (3 * j * nr) + (i *
                    ↪  dummySplits); // 7
                    _elements[index][6] = i + 3 + (2 * nr) + (3 * j * nr) + (i *
                    ↪  dummySplits); // 11
                    _elements[index][7] = i + 2 + (3 * nr) + (3 * j * nr) + (i *
                    ↪   dummySplits); //14
                    _elements[index][8] = i + 1 + (3 * nr) + (3 * j * nr) + (i *
                    ↪   dummySplits); // 13
                    _elements[index][9] = i + 2 + (2 * nr) + (3 * j * nr) + (i *
                    ↪   dummySplits); // 10

                    _edges[index][0][0] = _elements[index][0];
                    _edges[index][0][1] = _elements[index][8];
                    _edges[index][0][2] = _elements[index][7];
                    _edges[index][0][3] = _elements[index][2];

                    _edges[index][1][0] = _elements[index][1];
                    _edges[index][1][1] = _elements[index][4];
                    _edges[index][1][2] = _elements[index][3];
                    _edges[index][1][3] = _elements[index][0];

                    _edges[index][2][0] = _elements[index][1];
                    _edges[index][2][1] = _elements[index][5];
                    _edges[index][2][2] = _elements[index][6];
                    _edges[index][2][3] = _elements[index++][2];
                }
            }

            WritePoints("forGraphics/allPoints.txt");
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }

    private static void GetNodes(Point2D[] allNodes, Point2D[] vertices) {
        allNodes[0] = vertices[0];
        allNodes[1] = vertices[1];
        allNodes[2] = vertices[2];
        allNodes[3] = ((2 * vertices[0]) + vertices[1]) / 3;
        allNodes[4] = ((2 * vertices[1]) + vertices[0]) / 3;
        allNodes[5] = ((2 * vertices[1]) + vertices[2]) / 3;
        allNodes[6] = ((2 * vertices[2]) + vertices[1]) / 3;
        allNodes[7] = ((2 * vertices[2]) + vertices[0]) / 3;
        allNodes[8] = ((2 * vertices[0]) + vertices[2]) / 3;
        allNodes[9] = (vertices[0] + vertices[1] + vertices[2]) / 3;
    }

    private void WritePoints(string path) {
        using var sw = new StreamWriter(path);
        for (int i = 0; i < _points.Count; i++) {
            sw.WriteLine(_points[i]);
        }
    }

    private void WriteElements(string path) {
        using var sw = new StreamWriter(path);
        for (int i = 0; i < _elements.Length; i++) {
            sw.WriteLine($"{_elements[i][0]} {_elements[i][1]} {_elements[i][2]}");
        }
    }
}
```

### TimeGrids.cs

```
namespace courseProject;

public class TimeRegularGrid : ITimeGrid {
    private double[] _points = default!;
    public ImmutableArray<double> Points => _points.ToImmutableArray();

    public TimeRegularGrid(TimeGridParameters gridParameters) {
        Build(gridParameters);
    }

    private void Build(TimeGridParameters gridParameters) {
        try {
            if (gridParameters.Interval.LeftBorder < 0)
                throw new Exception("The beginning of the time segment cannot be less
                ↪  than 0");

            if (gridParameters.Splits < 1)
                throw new Exception("The number of splits must be greater than or
                ↪  equal to 1");

            _points = new double[gridParameters.Splits + 1];

            double h = gridParameters.Interval.Lenght / gridParameters.Splits;

            _points[0] = gridParameters.Interval.LeftBorder;
            _points[^1] = gridParameters.Interval.RightBorder;

            for (int i = 1; i < _points.Length - 1; i++) {
                _points[i] = _points[i - 1] + h;
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}

public class TimeIrregularGrid : ITimeGrid {
    private double[] _points = default!;
    public ImmutableArray<double> Points => _points.ToImmutableArray();

    public TimeIrregularGrid(TimeGridParameters gridParameters) {
        Build(gridParameters);
    }

    private void Build(TimeGridParameters gridParameters) {
        try {
            if (gridParameters.Interval.LeftBorder < 0)
                throw new Exception("The beginning of the time segment cannot be less
                ↪  than 0");

            if (gridParameters.Splits < 1)
                throw new Exception("The number of splits must be greater than or
                ↪  equal to 1");

            ArgumentNullException.ThrowIfNull(gridParameters.K,
            ↪  $"{nameof(gridParameters.K)} cannot be null");

            _points = new double[gridParameters.Splits + 1];

            double h;
            double sum = 0.0;

            for (int k = 0; k < gridParameters.Splits; k++)
                sum += Math.Pow(gridParameters.K.Value, k);

            h = gridParameters.Interval.Lenght / sum;
            _points[0] = gridParameters.Interval.LeftBorder;
            _points[^1] = gridParameters.Interval.RightBorder;

            for (int i = 1; i < _points.Length - 1; i++) {
                _points[i] = _points[i - 1] + h;
                h *= gridParameters.K.Value;
            }
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}
```

### Interval.cs

```
namespace courseProject;

public readonly record struct Interval {
    [JsonProperty("Left Border")]
    public double LeftBorder { get; init; }

    [JsonProperty("Right Border")]
    public double RightBorder { get; init; }

    [JsonIgnore]
    public double Lenght { get; init; }

    [JsonConstructor]
    public Interval(double leftBorder, double rightBorder) {
        LeftBorder = leftBorder;
        RightBorder = rightBorder;
        Lenght = Math.Abs(rightBorder - leftBorder);
    }
}
```

### Point2D.cs

```
namespace courseProject;

public readonly record struct Point2D(double R, double Z) {
    public static Point2D operator *(double constant, Point2D point)
        => new(constant * point.R, constant * point.Z);

    public static Point2D operator *(Point2D point, double constant)
       => new(point.R * constant, point.Z * constant);

    public static Point2D operator +(Point2D firstPoint, Point2D secondPoint)
        => new(firstPoint.R + secondPoint.R, firstPoint.Z + secondPoint.Z);

    public static Point2D operator -(Point2D firstPoint, Point2D secondPoint)
        => new(firstPoint.R - secondPoint.R, firstPoint.Z - secondPoint.Z);

    public static Point2D operator /(Point2D point, double constant)
        => new(point.R / constant, point.Z / constant);

    public override string ToString()
        => $"{R} {Z}";
}
```

### ArrayHelper.cs

```
namespace courseProject;

public static class ArrayHelper {
    public static void Fill<T>(this T[] array, T value) {
        for (int i = 0; i < array.Length; i++) {
            array[i] = value;
        }
    }

    public static double Norm(this double[] array) {
        double result = 0.0;

        for (int i = 0; i < array.Length; i++) {
            result += array[i] * array[i];
        }

        return Math.Sqrt(result);
    }

    public static void Copy<T>(this T[] source, T[] destination) {
        for (int i = 0; i < source.Length; i++) {
            destination[i] = source[i];
        }
    }
}
```

### GMrz.py

```
from sympy import expand, Symbol, diff

l1 = Symbol('l1')
l2 = Symbol('l2')
l3 = Symbol('l3')
dl1 = Symbol('d1l')
dl2 = Symbol('d2l')
dl3 = Symbol('d3l')

#функции
# psi1 = 0.5*l1*(3*l1-1)*(3*l1-2)
# psi2 = 0.5*l2*(3*l2-1)*(3*l2-2)
# psi3 = 0.5*l3*(3*l3-1)*(3*l3-2)
# psi4 = 4.5*l1*l2*(3*l1-1)
# psi5 = 4.5*l1*l2*(3*l2-1)
# psi6 = 4.5*l2*l3*(3*l2-1)
# psi7 = 4.5*l2*l3*(3*l3-1)
# psi8 = 4.5*l3*l1*(3*l3-1)
# psi9 = 4.5*l3*l1*(3*l1-1)
# psi10 = 27*l1*l2*l3

```

```
psi1 = 0.5*l1*(3*l1-1)*(3*l1-2)
psi2 = 4.5*l3*l1*(3*l1-1)
psi3 = 4.5*l3*l1*(3*l3-1)
psi4 = 0.5*l3*(3*l3-1)*(3*l3-2)
psi5 = 4.5*l1*l2*(3*l1-1)
psi6 = 27*l1*l2*l3
psi7 = 4.5*l2*l3*(3*l3-1)
psi8 = 4.5*l1*l2*(3*l2-1)
psi9 = 4.5*l2*l3*(3*l2-1)
psi10 = 0.5*l2*(3*l2-1)*(3*l2-2)

psi = [psi1, psi2, psi3, psi4, psi5, psi6, psi7, psi8, psi9, psi10]

#производные
p1 = dl1*diff(psi1, l1)+dl2*diff(psi1, l2)+dl3*diff(psi1, l3)
p2 = dl1*diff(psi2, l1)+dl2*diff(psi2, l2)+dl3*diff(psi2, l3)
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)
p5 = dl1*diff(psi5, l1)+dl2*diff(psi5, l2)+dl3*diff(psi5, l3)
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)
p9 = dl1*diff(psi9, l1)+dl2*diff(psi9, l2)+dl3*diff(psi9, l3)
p10 = dl1*diff(psi10, l1)+dl2*diff(psi10, l2)+dl3*diff(psi10, l3)

p=[p1, p2, p3, p4, p5, p6, p7, p8, p9, p10]

def fac(n):
    if n == 0:
        return 1
    return fac(n-1) * n

def Integrate(func):
    res = [0, 0, 0]
    func = func.replace("- 2", "-2")
    func = func.replace("- 3", "-3")
    func = func.replace("- 4", "-4")
    func = func.replace("- 1", "-1")
    func = func.replace("- 5", "-5")
    func = func.replace("- 6", "-6")
    func = func.replace("- 7", "-7")
    func = func.replace("- 8", "-8")
    func = func.replace("- 9", "-9")
    func = func.replace('+ ', '')
    func = func.replace("**", "^")
    func = func.split()
    func = [i.split("*", 1) for i in func]
    for iter in range(len(func)):
        a = float(func[iter][0])
        f = func[iter][1].rfind('l1^')
        if f>=0:
            i = int(func[iter][1][func[iter][1].rfind('l1^')+3])
        else:
            f = func[iter][1].rfind('l1')
            if f>=0:
                i = 1
            else:
                i = 0
        f = func[iter][1].rfind('l2^')
        if f>=0:
            j = int(func[iter][1][func[iter][1].rfind('l2^')+3])
        else:
            f = func[iter][1].rfind('l2')
            if f>=0:
                j = 1
            else:
                j = 0
        f = func[iter][1].rfind('l3^')
        if f>=0:
            k = int(func[iter][1][func[iter][1].rfind('l3^')+3])
        else:
            f = func[iter][1].rfind('l3')
            if f>=0:
                k = 1
            else:
                k = 0

        res[0]+=a * fac(i+1)*fac(j)*fac(k)/fac(i+j+k+3)
        res[1]+=a * fac(i)*fac(j+1)*fac(k)/fac(i+j+k+3)
        res[2]+=a * fac(i)*fac(j)*fac(k+1)/fac(i+j+k+3)

    return res

def makeM():
    f1 = open('Mrz.txt', 'w')
    mf = []
    rs = [[], [], [], [], [], [], [], [], [], []]
    for i in range(10):
        for j in range(10):
            r = str(expand(psi[i]*psi[j]))
            mf.append(r)
            rs[i].append(Integrate(r))
    i, j = 0, 0
    for row in rs:
        for x in row:
            f1.write(str(i) +" "+ str(j) +" " + '
             ↪  '.join(["{0:.20f}".format(float(elem)) for elem in x]) + "\n")
            j+=1
        i+=1
        j=0
    f1.close()
    return rs

def IntegrateGrad(func):
    res = [[0,0,0], [0,0,0], [0,0,0], [0,0,0], [0,0,0], [0,0,0]]
    func = func.replace("- 2", "-2")
    func = func.replace("- 3", "-3")
    func = func.replace("- 4", "-4")
    func = func.replace("- 1", "-1")
    func = func.replace("- 5", "-5")
    func = func.replace("- 6", "-6")
    func = func.replace("- 7", "-7")
    func = func.replace("- 8", "-8")
    func = func.replace("- 9", "-9")
    func = func.replace('+ ', '')
    func = func.replace("**", "^")
    func = func.split()
    func = [i.split("*", 1) for i in func]
    for iter in range(len(func)):
        if func[iter][1].rfind("d1l^2")>=0: flag = 0
        elif func[iter][1].rfind("d1l*d2l")>=0: flag = 1
        elif func[iter][1].rfind("d1l*d3l")>=0: flag = 2
        elif func[iter][1].rfind("d2l^2")>=0: flag = 3
        elif func[iter][1].rfind("d2l*d3l")>=0: flag = 4
        elif func[iter][1].rfind("d3l^2")>=0: flag = 5
        a = float(func[iter][0])
        f = func[iter][1].rfind('l1^')
        if f>=0:
            i = int(func[iter][1][func[iter][1].rfind('l1^')+3])
        else:
            f = func[iter][1].rfind('l1')
            if f>=0:
                i = 1
            else:
                i = 0
        f = func[iter][1].rfind('l2^')
        if f>=0:
            j = int(func[iter][1][func[iter][1].rfind('l2^')+3])
        else:
            f = func[iter][1].rfind('l2')
            if f>=0:
                j = 1
            else:
                j = 0
        f = func[iter][1].rfind('l3^')
        if f>=0:
            k = int(func[iter][1][func[iter][1].rfind('l3^')+3])
        else:
            f = func[iter][1].rfind('l3')
            if f>=0:
                k = 1
            else:
                k = 0
        res[flag][0]+=a * fac(i+1)*fac(j)*fac(k)/fac(i+j+k+3)
        res[flag][1]+=a * fac(i)*fac(j+1)*fac(k)/fac(i+j+k+3)
        res[flag][2]+=a * fac(i)*fac(j)*fac(k+1)/fac(i+j+k+3)

    return res

#L(x,y)=a0i+a1ix+a2iy
#rs=[dl1^2 dl1dl2 dl1dl3 dl2^2 dl2dl3 dl3dl3]
def makeG():
    f2 = open('Grz.txt', 'w')
    rs = []
    for i in range(10):
        rs.append([])
        for j in range(10):
            rs[i].append([])

    for i in range(10):
        for j in range(10):
            r = str(expand(p[i]*p[j]))
            rs[i][j]=IntegrateGrad(r)
    i=0
    j = 0
    k = 0
    for row in rs:
        for x in row:
            for y in x:
                f2.write(str(i) +" "+ str(j) +" "+ str(k) +" "+ '
                ↪  '.join(["{0:.20f}".format(float(elem)) for elem in y]) + "\n")
                k+=1
            k=0
            j+=1
        i+=1
        j=0
    f2.close()
    return rs

M = makeM()
G=makeG()
```

## graphics.py

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np

xL, yL, xAll, yAll = [], [], [], []
elements = []

fig, ax = plt.subplots()

with open("forGraphics/pointsLinear.txt") as file:
    for line in file:
        x, y = line.split()
        xL.append(float(x))
        yL.append(float(y))

with open("forGraphics/allPoints.txt") as file:
    for line in file:
        x, y = line.split()
        xAll.append(float(x))
        yAll.append(float(y))

with open("forGraphics/elements.txt") as file:
    for line in file:
        vert1, vert2, vert3 = map(int, line.split())
        elements.append([vert1, vert2, vert3])

for elem in elements:
    triangle = patches.Polygon([
        [xL[elem[0]], yL[elem[0]]],
        [xL[elem[1]], yL[elem[1]]],
        [xL[elem[2]], yL[elem[2]]]
    ],
        edgecolor='black', facecolor='white', linewidth=2)
    ax.add_patch(triangle)

# for i in range(len(xAll)):
#     ax.annotate(i, xy = (xAll[i] + 0.01, yAll[i] + 0.01))

plt.plot(xL, yL, " ")
# plt.show()
plt.savefig("images/4.eps")
```