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 «Метод конечных элементов»

Применение МКЭ для решения задач в областях с криволинейными границами

Студент ШИШКИН НИКИТА

Группа ПМ-92

Преподаватель РОЯК МИХАИЛ ЭММАНУИЛОВИЧ

Новосибирск, 2023

Постановка задачи

Криволинейные элементы второго порядка с Лагранжевыми функциями формы для двумерного уравнения Пуассона:

$$-\operatorname{div}\left(\lambda \operatorname{grad} u\right) = f \tag{1}$$

Теоретическая часть

Предисловие

Четырхеугольные элементы применяются для решения задач в областях с криволинейными границами. Эти элементы сочетают в себе преимущества прямоугольных элементов в точности аппроксимации решениями при описании расчетных областей со сложными границами.

Четырехугольные элементы высоких порядков

Для построения четырехугольных конечных элементов второго порядка воспользуемся подходом, который предполагает, что при построении отображения шаблонного элемента в координатах (ξ, η) в конечных элемент в координатах (x, y) используются биквадратичные функции.

Отображение единичного квадрата в четырехугольник с криволинейными границами может записано в виде:

$$\begin{aligned} x &= \sum_{i=1}^{9} \varphi_i(\xi, \eta)\hat{x}_i, \\ y &= \sum_{i=1}^{9} \varphi_i(\xi, \eta)\hat{y}_i, \end{aligned} \tag{2}$$

где φ_i – стандартные биквадратичные функции, а $(\hat{x}_i, \hat{y}_i)$ – координаты девяти узлов элемента.

Базисные функции

Вид базисных функций:

$$\begin{aligned} \varphi_1(\xi, \eta) &= W_1(\xi)W_1(\eta), & \varphi_2(\xi, \eta) &= W_2(\xi)W_1(\eta), \\ \varphi_3(\xi, \eta) &= W_3(\xi)W_1(\eta), & \varphi_4(\xi, \eta) &= W_1(\xi)W_2(\eta), \\ \varphi_5(\xi, \eta) &= W_2(\xi)W_2(\eta), & \varphi_6(\xi, \eta) &= W_3(\xi)W_2(\eta), \\ \varphi_7(\xi, \eta) &= W_1(\xi)W_3(\eta), & \varphi_8(\xi, \eta) &= W_2(\xi)W_3(\eta), \\ \varphi_9(\xi, \eta) &= W_3(\xi)W_3(\eta), \end{aligned} \tag{3}$$

где

$$\begin{aligned} W_1(\zeta) &= 2\left(\zeta - \frac{1}{2}\right)(\zeta - 1), \\ W_2(\zeta) &= -4\zeta(\zeta - 1), \\ W_3(\zeta) &= 2\zeta\left(\zeta - \frac{1}{2}\right). \end{aligned} \tag{4}$$

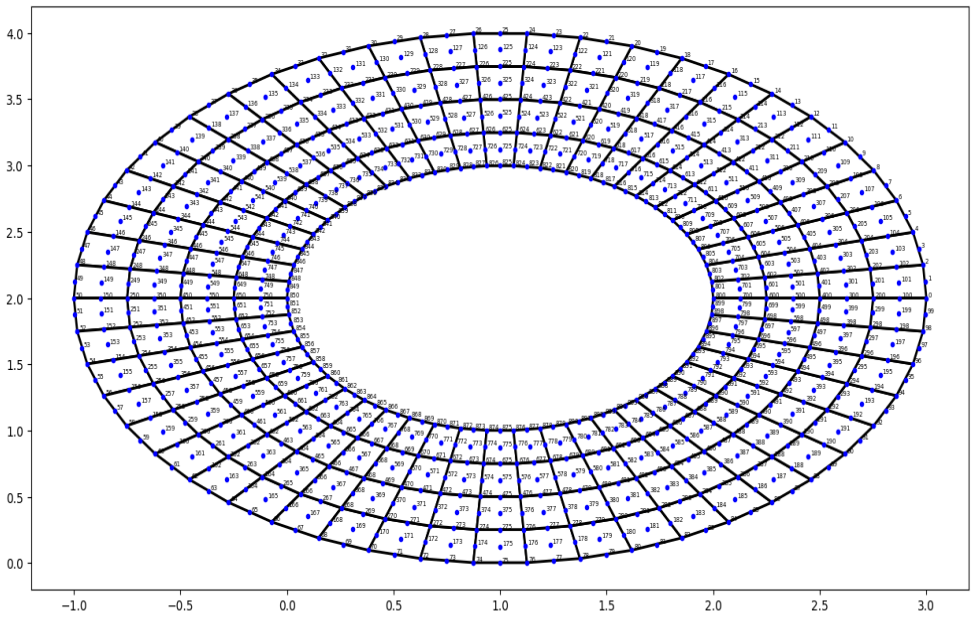
Тестирование программы

Первый тест

Функция: $x + y$
Правая часть: 0
Коэффициенты : $\lambda = 1$
Сетка:

- Внутренний радиус = 1,
- Внешний радиус = 2,
- Центр кольца = $(1, 2)$
- $h = \dfrac{2\pi}{50}$

Заданы краевые условия 1-го рода на внешней и внутренней границах кольца.



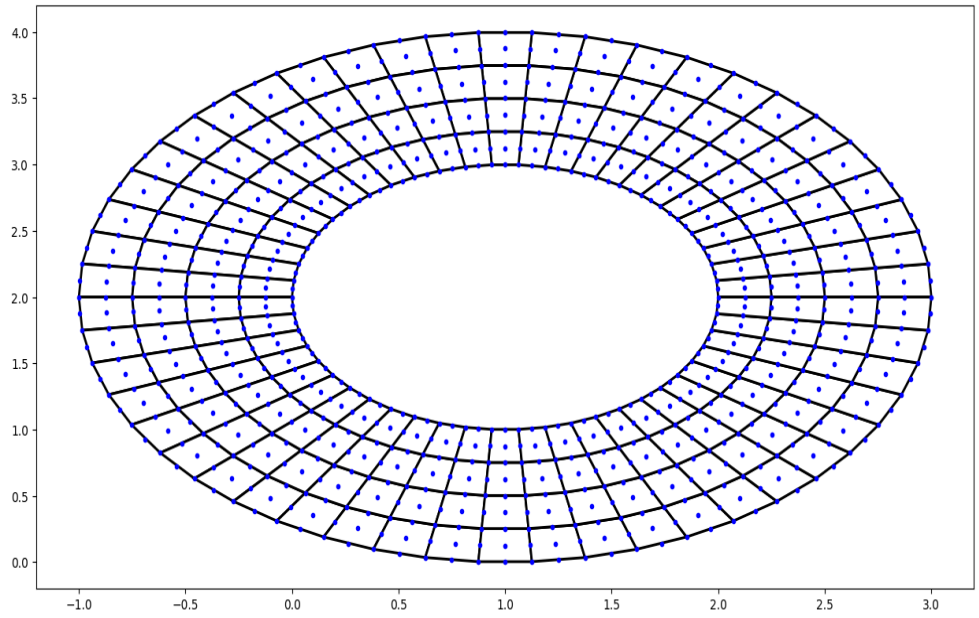
i	u_i^*	u_i	$\lvert u^* - u \rvert$	Погрешность
0	5.00000000000	5.00000000000	0	$3.66 \cdot 10^{-15}$
1	5.12163449592	5.12163449592	$8.88 \cdot 10^{-16}$	...
2	5.23489586976	5.23489586976	$2.66 \cdot 10^{-15}$	...
3	5.33933713063	5.33933713063	$1.78 \cdot 10^{-15}$	...
4	5.43454609659	5.43454609659	$8.88 \cdot 10^{-16}$	...
5	5.52014702134	5.52014702134	$8.88 \cdot 10^{-16}$	...
6	5.59580207715	5.59580207715	0	...
7	5.66121268806	5.66121268806	$8.88 \cdot 10^{-16}$	...
8	5.71612070829	5.71612070829	$8.88 \cdot 10^{-16}$	...
9	5.76030944096	5.76030944096	$1.78 \cdot 10^{-15}$	...
10	5.79360449333	5.79360449333	$2.66 \cdot 10^{-15}$	...
11	5.81587446505	5.81587446505	$8.88 \cdot 10^{-16}$	...
12	5.82703146670	5.82703146670	$1.78 \cdot 10^{-15}$	...
13	5.82703146670	5.82703146670	$1.78 \cdot 10^{-15}$	...
14	5.81587446505	5.81587446505	$8.88 \cdot 10^{-16}$	...
15	5.79360449333	5.79360449333	$2.66 \cdot 10^{-15}$	...
16	5.76030944096	5.76030944096	$1.78 \cdot 10^{-15}$	...
17	5.71612070829	5.71612070829	$8.88 \cdot 10^{-16}$	...
18	5.66121268806	5.66121268806	$8.88 \cdot 10^{-16}$	...
19	5.59580207715	5.59580207715	0	...
...	...	...	...	...
880	2.35796047808	2.35796047808	$8.88 \cdot 10^{-16}$	...
881	2.43834806680	2.43834806680	$1.78 \cdot 10^{-15}$	...
882	2.52095223910	2.52095223910	$4.44 \cdot 10^{-16}$	...
883	2.60544699406	2.60544699406	$4.44 \cdot 10^{-16}$	...
884	2.69149886948	2.69149886948	$4.44 \cdot 10^{-16}$	...
885	2.77876825792	2.77876825792	$1.78 \cdot 10^{-15}$	...
886	2.86691074697	2.86691074697	$4.44 \cdot 10^{-16}$	...
887	2.95557847851	2.95557847851	$4.44 \cdot 10^{-16}$	...
888	3.04442152149	3.04442152149	$8.88 \cdot 10^{-16}$	...
889	3.13308925303	3.13308925303	$8.88 \cdot 10^{-16}$	...
890	3.22123174208	3.22123174208	$8.88 \cdot 10^{-16}$	...
891	3.30850113052	3.30850113052	0	...
892	3.39455300594	3.39455300594	$1.33 \cdot 10^{-15}$	...
893	3.47904776090	3.47904776090	$4.44 \cdot 10^{-16}$	...
894	3.56165193320	3.56165193320	$4.44 \cdot 10^{-16}$	...
895	3.64203952192	3.64203952192	0	...
896	3.71989327396	3.71989327396	$4.44 \cdot 10^{-16}$	...
897	3.79490593614	3.79490593614	$4.44 \cdot 10^{-16}$	...
898	3.86678146775	3.86678146775	$4.44 \cdot 10^{-16}$	...
899	3.93523620890	3.93523620890	$4.44 \cdot 10^{-16}$	...

Второй тест

Функция: $x^2 + y^2$
Правая часть: -4
Коэффициенты : $\lambda = 1$
Сетка:

- Внутренний радиус = 1,
- Внешний радиус = 2,
- Центр кольца = $(1, 2)$
- $h = \frac{2\pi}{50}$

Заданы краевые условия 1-го рода на внешней и внутренней границах кольца.



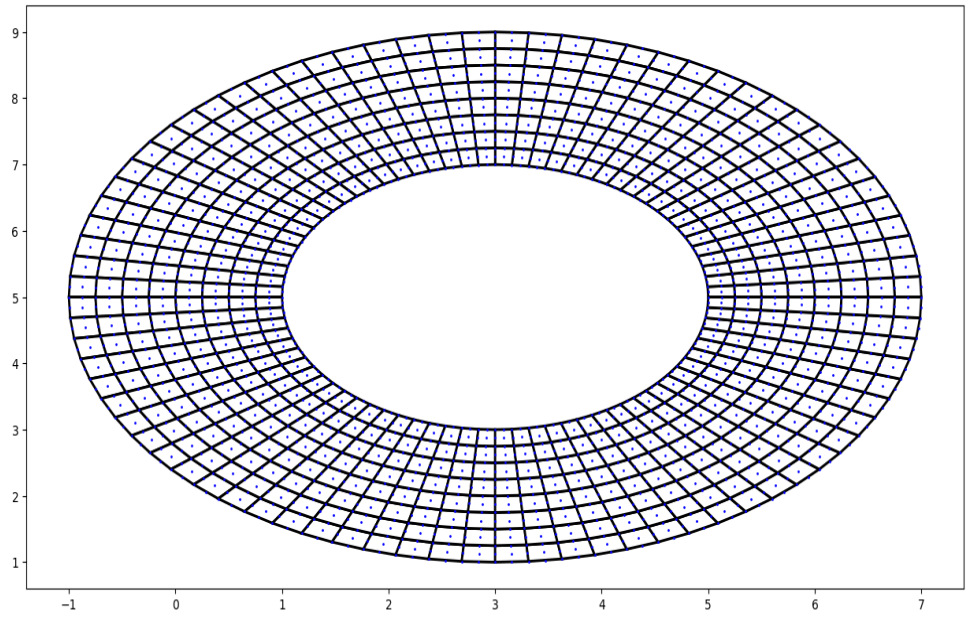
i	u_i^*	u_i	$\lvert u^* - u \rvert$	Погрешность
0	13.00000000000	13.00000000000	$1.78 \cdot 10^{-15}$	$9.39 \cdot 10^{-7}$
1	13.49443106995	13.49443106995	$1.78 \cdot 10^{-15}$	$\cdots$
2	13.97112467377	13.97112467377	$1.78 \cdot 10^{-15}$	$\cdots$
3	14.42819951960	14.42819951960	$1.78 \cdot 10^{-15}$	$\cdots$
4	14.86385174183	14.86385174183	$7.11 \cdot 10^{-15}$	$\cdots$
5	15.27636202018	15.27636202018	$3.55 \cdot 10^{-15}$	$\cdots$
6	15.66410236503	15.66410236503	$1.78 \cdot 10^{-15}$	$\cdots$
7	16.02554254238	16.02554254238	$3.55 \cdot 10^{-15}$	$\cdots$
8	16.35925611299	16.35925611299	$3.55 \cdot 10^{-15}$	$\cdots$
9	16.66392606184	16.66392606184	0	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
460	1.29623750312	1.29623891002	$1.41 \cdot 10^{-6}$	$\cdots$
461	1.11391633318	1.11391602267	$3.11 \cdot 10^{-7}$	$\cdots$
462	0.95581148216	0.95581288907	$1.41 \cdot 10^{-6}$	$\cdots$
463	0.82254691769	0.82254660718	$3.11 \cdot 10^{-7}$	$\cdots$
464	0.71464857410	0.71464998100	$1.41 \cdot 10^{-6}$	$\cdots$
465	0.63254227687	0.63254196636	$3.11 \cdot 10^{-7}$	$\cdots$
466	0.57655206205	0.57655346895	$1.41 \cdot 10^{-6}$	$\cdots$
467	0.54689889743	0.54689858692	$3.11 \cdot 10^{-7}$	$\cdots$
468	0.54369981051	0.54370121741	$1.41 \cdot 10^{-6}$	$\cdots$
469	0.56696742662	0.56696711611	$3.11 \cdot 10^{-7}$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
660	1.60103125260	1.60103219941	$9.47 \cdot 10^{-7}$	$\cdots$
661	1.44909694432	1.44909678484	$1.59 \cdot 10^{-7}$	$\cdots$
662	1.31734290180	1.31734384862	$9.47 \cdot 10^{-7}$	$\cdots$
663	1.20628909807	1.20628893860	$1.59 \cdot 10^{-7}$	$\cdots$
664	1.11637381175	1.11637475856	$9.47 \cdot 10^{-7}$	$\cdots$
665	1.04795189739	1.04795173792	$1.59 \cdot 10^{-7}$	$\cdots$
666	1.00129338504	1.00129433185	$9.47 \cdot 10^{-7}$	$\cdots$
667	0.97658241453	0.97658225505	$1.59 \cdot 10^{-7}$	$\cdots$
668	0.97391650876	0.97391745557	$9.47 \cdot 10^{-7}$	$\cdots$
669	0.99330618885	0.99330602937	$1.59 \cdot 10^{-7}$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
890	5.26689297958	5.26689297958	$1.78 \cdot 10^{-15}$	$\cdots$
891	5.54534867109	5.54534867109	$1.78 \cdot 10^{-15}$	$\cdots$
892	5.82559866368	5.82559866368	$1.78 \cdot 10^{-15}$	$\cdots$
893	6.10653693867	6.10653693867	$8.88 \cdot 10^{-16}$	$\cdots$
894	6.38705476104	6.38705476104	0	$\cdots$
895	6.66604505509	6.66604505509	$1.78 \cdot 10^{-15}$	$\cdots$
896	6.94240677360	6.94240677360	$8.88 \cdot 10^{-16}$	$\cdots$
897	7.21504924311	7.21504924311	$1.78 \cdot 10^{-15}$	$\cdots$
898	7.48289646837	7.48289646837	$2.66 \cdot 10^{-15}$	$\cdots$
899	7.74489137874	7.74489137874	$8.88 \cdot 10^{-16}$	$\cdots$

Третий тест

Функция: $x^3 \cdot y^3$
Правая часть: $-6x \cdot y^3 - 6y \cdot x^3$
Коэффициенты : $\lambda = 1$
Сетка:

- Внутренний радиус = 2,
- Внешний радиус = 4,
- Центр кольца = $(3, 5)$
- $h = \dfrac{2\pi}{80}$

Заданы краевые условия 1-го рода на внешней и внутренней границах кольца.



i	u_i^*	u_i	$\lvert u^* - u \rvert$	Погрешность
0	42,875.00000000000	42,875.00000000002	$1.46 \cdot 10^{-11}$	$1.42 \cdot 10^{-2}$
1	46,980.89879833817	46,980.89879833817	$7.28 \cdot 10^{-12}$	...
2	51,194.29544041086	51,194.29544041085	$7.28 \cdot 10^{-12}$	...
3	55,478.01509702865	55,478.01509702864	$7.28 \cdot 10^{-12}$	...
4	59,790.76504834089	59,790.76504834089	0	...
5	64,087.67327360821	64,087.67327360820	$7.28 \cdot 10^{-12}$	...
6	68,320.95240412107	68,320.95240412105	$1.46 \cdot 10^{-11}$	...
7	72,440.67809564878	72,440.67809564873	$4.37 \cdot 10^{-11}$	...
8	76,395.66661234669	76,395.66661234670	$1.46 \cdot 10^{-11}$	...
9	80,134.43245801852	80,134.43245801848	$4.37 \cdot 10^{-11}$	...
...	...	...	...	...
300	2,182.96836100430	2,182.97141659192	$3.06 \cdot 10^{-3}$	...
301	2,657.76121554626	2,657.75388594694	$7.33 \cdot 10^{-3}$	...
302	3,220.58266196660	3,220.58491948397	$2.26 \cdot 10^{-3}$	...
303	3,883.35016034093	3,883.34336092709	$6.8 \cdot 10^{-3}$	...
304	4,658.58984009952	4,658.59103373537	$1.19 \cdot 10^{-3}$	...
305	5,559.25401121558	5,559.24877181502	$5.24 \cdot 10^{-3}$	...
306	6,598.49371738247	6,598.49359533571	$1.22 \cdot 10^{-4}$	...
307	7,789.38663063864	7,789.38398276999	$2.65 \cdot 10^{-3}$	...
308	9,144.62231337760	9,144.62064419816	$1.67 \cdot 10^{-3}$	...
309	10,676.14874535303	10,676.14952294823	$7.78 \cdot 10^{-4}$	...
...	...	...	...	...
2,060	3,659.76379106055	3,659.76218407333	$1.61 \cdot 10^{-3}$	...
2,061	4,075.03805608397	4,075.03533278897	$2.72 \cdot 10^{-3}$	...
2,062	4,531.18697025020	4,531.18544439029	$1.53 \cdot 10^{-3}$	...
2,063	5,030.62550767472	5,030.62353489701	$1.97 \cdot 10^{-3}$	...
2,064	5,575.61927766995	5,575.61793415184	$1.34 \cdot 10^{-3}$	...
2,065	6,168.22677989021	6,168.22580594234	$9.74 \cdot 10^{-4}$	...
2,066	6,810.23744188638	6,810.23637563378	$1.07 \cdot 10^{-3}$	...
2,067	7,503.10634725356	7,503.10656409170	$2.17 \cdot 10^{-4}$	...
2,068	8,247.88677943052	8,247.88607114792	$7.08 \cdot 10^{-4}$	...
2,069	9,045.16191494182	9,045.16342683109	$1.51 \cdot 10^{-3}$	...
...	...	...	...	...
2,710	8,651.08254373486	8,651.08254373486	$3.64 \cdot 10^{-12}$	...
2,711	9,269.35715187108	9,269.35715187108	$5.46 \cdot 10^{-12}$	...
2,712	9,911.90148207159	9,911.90148207159	$1.82 \cdot 10^{-12}$	...
2,713	10,576.88391608979	10,576.88391608979	0	...
2,714	11,262.08769136371	11,262.08769136371	0	...
2,715	11,964.90452480258	11,964.90452480258	0	...
2,716	12,682.33421004564	12,682.33421004564	$1.82 \cdot 10^{-12}$	...
2,717	13,410.99073608564	13,410.99073608564	0	...
2,718	14,147.11536878494	14,147.11536878494	$1.82 \cdot 10^{-12}$	...
2,719	14,886.59700960653	14,886.59700960653	$3.64 \cdot 10^{-12}$	...

Исследование на определение порядка сходимости

Функция: e^{x+y}
Правая часть: $-2e^{x+y}$
Коэффициенты : $\lambda = 1$
Заданы краевые условия 1-го рода на внешней и внутренней границах кольца.

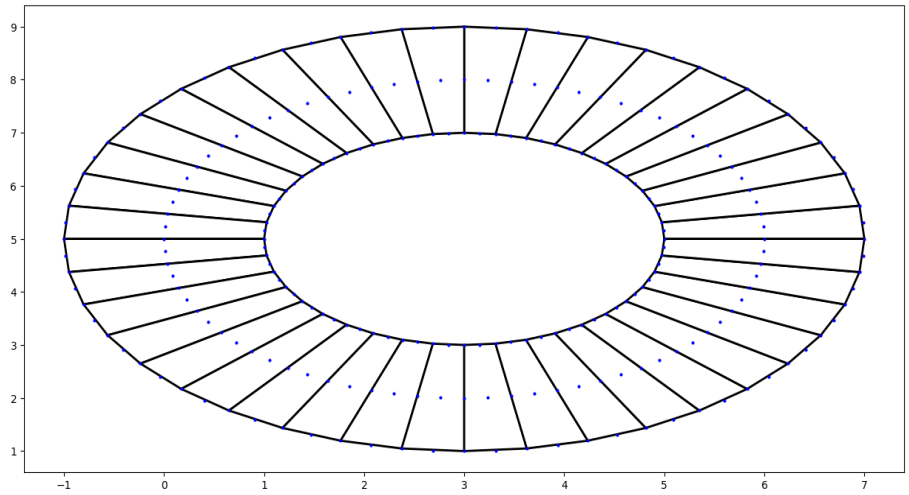


Рис. 1: Сетка с шагом h

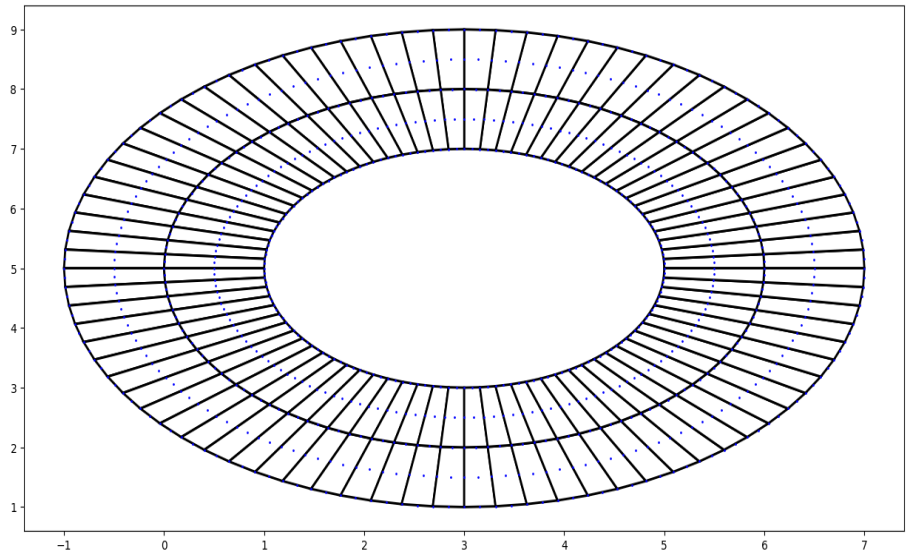


Рис. 2: Сетка с шагом $h/2$

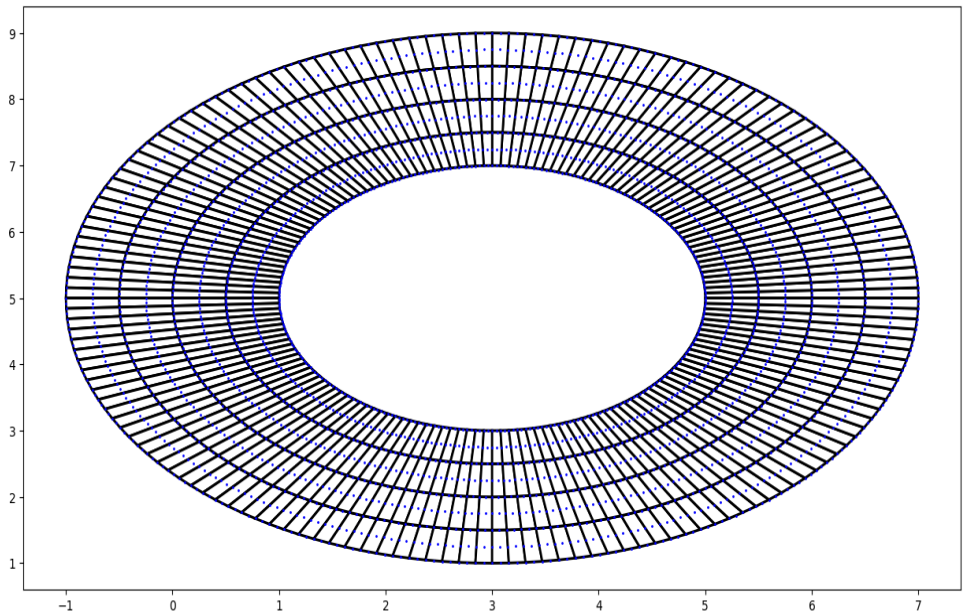


Рис. 3: Сетка с шагом $h/4$

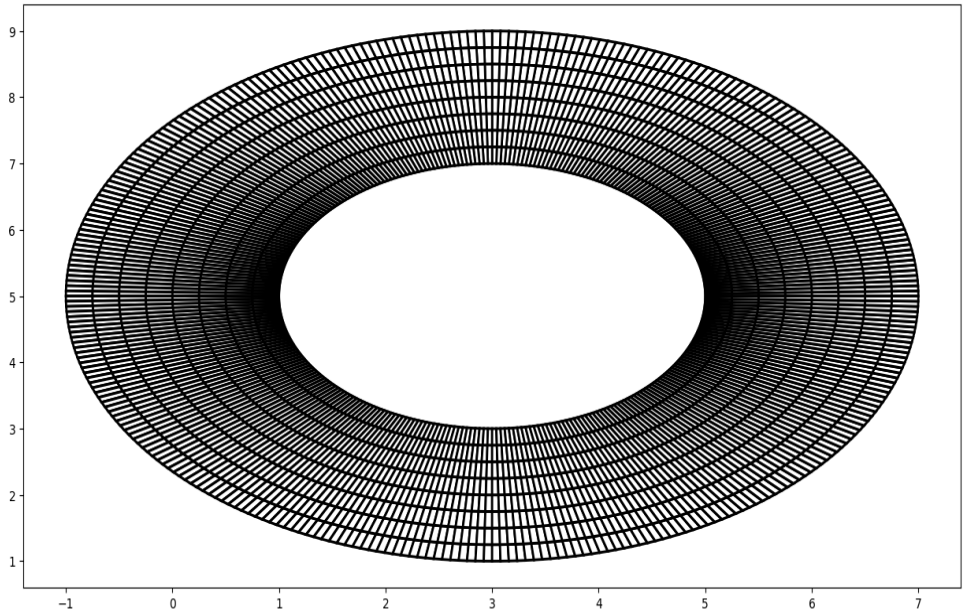


Рис. 4: Сетка с шагом $h/8$

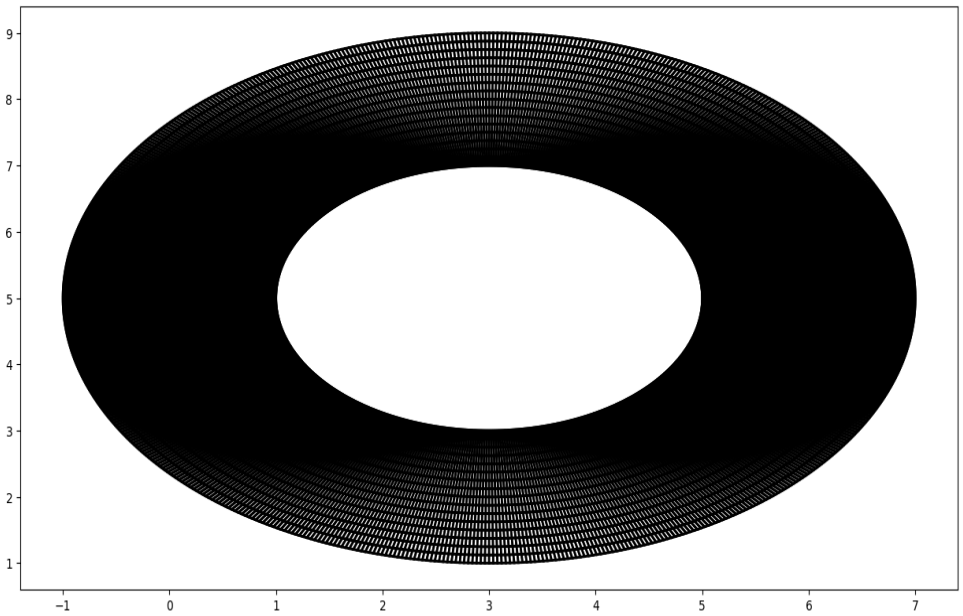


Рис. 5: Сетка с шагом $h/16$

Сетка	Погрешность	Порядок сходимости
h	5068.776478250752	–
$h/2$	328.98613863392546	3.9455
$h/4$	21.61675051625887	3.9278
$h/8$	1.3903468293788837	3.9586
$h/16$	0.08820816810481347	3.9784

Проведенные исследования и выводы

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

Тексты основных модулей

Program.cs

```
var meshParameters = CurveMeshParameters.ReadJson("input/curveMeshParameters.json");
// var meshParameters = MeshParameters.ReadJson("input/meshParameters.json");
var boundariesParameters =
↪  BoundaryParameters.ReadJson("input/boundaryParameters.json");
var boundaryHandler = new CurveQuadraticBoundaryHandler(boundariesParameters,
↪  meshParameters);
var meshCreator = new RegularMeshCreator();
var mesh = meshCreator.CreateMesh(meshParameters, new CurveQuadraticMeshBuilder());
SolverFem problem = SolverFem.CreateBuilder()
    .SetMesh(mesh)
    .SetTest(new Test6())
    .SetSolverSlae(new CGMCholesky(1000, 1E-16))
    .SetAssembler(new CurveMatrixAssembler(new QuadraticBasis(),
↪  new(Quadratures.SegmentGaussOrder5()), mesh))
    .SetBoundaries(boundaryHandler.Process());

problem.Compute();
```

FEM.cs

```
namespace Project;

public class SolverFem
{
    public class SolverFemBuilder
    {
        private readonly SolverFem _solverFem = new();

        public SolverFemBuilder SetTest(ITest test)
        {
            _solverFem._test = test;
            return this;
        }

        public SolverFemBuilder SetMesh(IBaseMesh mesh)
        {
            _solverFem._mesh = mesh;
            return this;
        }

        public SolverFemBuilder SetSolverSlae(IterativeSolver iterativeSolver)
        {
            _solverFem._iterativeSolver = iterativeSolver;
            return this;
        }

        public SolverFemBuilder SetBoundaries(IEnumerable<IBoundary> boundaries)
        {
            _solverFem._boundaries = boundaries.DistinctBy(b => b.Node);
            return this;
        }

        public SolverFemBuilder SetAssembler(BaseMatrixAssembler matrixAssembler)
        {
            _solverFem._matrixAssembler = matrixAssembler;
            return this;
        }
```

```
38
39          public static implicit operator SolverFem(SolverFemBuilder builder)
40              => builder._solverFem;
41      }
42
43      private IBaseMesh _mesh = default!;
44      private ITest _test = default!;
45      private IterativeSolver _iterativeSolver = default!;
46      private IEnumerable<IBoundary> _boundaries = default!;
47      private Vector<double> _localVector = default!;
48      private Vector<double> _globalVector = default!;
49      private BaseMatrixAssembler _matrixAssembler = default!;
50
51      public void Compute()
52      {
53          Initialize();
54          AssemblySystem();
55          _matrixAssembler.GlobalMatrix.PrintDense("output/matrixBefore.txt");
56          AccountingDirichletBoundary();
57
58          _matrixAssembler.GlobalMatrix.PrintDense("output/matrixAfter.txt");
59
60          _iterativeSolver.SetMatrix(_matrixAssembler.GlobalMatrix!);
61          _iterativeSolver.SetVector(_globalVector);
62          _iterativeSolver.Compute();
63
64          var exact = new double[_mesh.Points.Count];
65
66          for (int i = 0; i < exact.Length; i++)
67          {
68              exact[i] = _test.U(_mesh.Points[i]);
69          }
70
71          var result = exact.Zip(_iterativeSolver.Solution!.Value, (v1, v2) => (v2, v1));
72
73          foreach (var (v1, v2) in result)
74          {
75              Console.WriteLine($"{v1} ---------- {v2} ");
76          }
77
78          Console.WriteLine("---------------------------");
79
80          CalculateError();
81      }
82
83      private void Initialize()
84      {
85          PortraitBuilder.Build(_mesh, out var ig, out var jg);
86          _matrixAssembler.GlobalMatrix = new(ig.Length - 1, jg.Length)
87          {
88              Ig = ig,
89              Jg = jg
90          };
91
92          _globalVector = new(ig.Length - 1);
93          _localVector = new(_matrixAssembler.BasisSize);
94      }
95
96      private void AssemblySystem()
97      {
```

```
        for (int ielem = 0; ielem < _mesh.Elements.Count; ielem++)
        {
            var element = _mesh.Elements[ielem];

            _matrixAssembler.BuildLocalMatrices(ielem);
            BuildLocalVector(ielem);

            for (int i = 0; i < _matrixAssembler.BasisSize; i++)
            {
                _globalVector[element[i]] += _localVector[i];

                for (int j = 0; j < _matrixAssembler.BasisSize; j++)
                {
                    _matrixAssembler.FillGlobalMatrix(element[i], element[j],
↪   _matrixAssembler.StiffnessMatrix[i, j]);
                }
            }
        }
    }

    private void BuildLocalVector(int ielem)
    {
        _localVector.Fill(0.0);

        for (int i = 0; i < _matrixAssembler.BasisSize; i++)
        {
            for (int j = 0; j < _matrixAssembler.BasisSize; j++)
            {
                _localVector[i] += _matrixAssembler.MassMatrix[i, j] *
↪   _test.F(_mesh.Points[_mesh.Elements[ielem][j]]);
            }
        }
    }

    private void AccountingDirichletBoundary()
    {
        int[] checkBc = new int[_mesh.Points.Count];

        checkBc.Fill(-1);
        var boundariesArray = _boundaries.ToArray();

        for (int i = 0; i < boundariesArray.Length; i++)
        {
            boundariesArray[i].Value = _test.U(_mesh.Points[boundariesArray[i].Node]);
            checkBc[boundariesArray[i].Node] = i;
        }

        // for (int i = 0; i < arrayBoundaries.Length; i++)
        // {
        //     _matrixAssembler.GlobalMatrix.Di[arrayBoundaries[i].Node] = 1E+32;
        //     _globalVector[arrayBoundaries[i].Node] = 1E+32 *
↪   arrayBoundaries[i].Value;
        // }

        for (int i = 0; i < _mesh.Points.Count; i++)
        {
            int index;
            if (checkBc[i] != -1)
            {
                _matrixAssembler.GlobalMatrix!.Di[i] = 1.0;
```

```
155                 _globalVector[i] = boundariesArray[checkBc[i]].Value;
156
157                 for (int k = _matrixAssembler.GlobalMatrix.Ig[i]; k <
    ↪  _matrixAssembler.GlobalMatrix.Ig[i + 1]; k++)
158                 {
159                     index = _matrixAssembler.GlobalMatrix.Jg[k];
160
161                     if (checkBc[index] == -1)
162                     {
163                         _globalVector[index] -= _matrixAssembler.GlobalMatrix.Gg[k] *
    ↪  _globalVector[i];
164                     }
165
166                     _matrixAssembler.GlobalMatrix.Gg[k] = 0.0;
167                 }
168             }
169             else
170             {
171                 for (int k = _matrixAssembler.GlobalMatrix!.Ig[i]; k <
    ↪  _matrixAssembler.GlobalMatrix.Ig[i + 1]; k++)
172                 {
173                     index = _matrixAssembler.GlobalMatrix.Jg[k];
174
175                     if (checkBc[index] == -1) continue;
176                     _globalVector[i] -= _matrixAssembler.GlobalMatrix.Gg[k] *
    ↪  _globalVector[index];
177                     _matrixAssembler.GlobalMatrix.Gg[k] = 0.0;
178                 }
179             }
180         }
181     }
182
183     private void CalculateError()
184     {
185         var error = new double[_mesh.Points.Count];
186
187         for (int i = 0; i < error.Length; i++)
188         {
189             error[i] = Math.Abs(_iterativeSolver.Solution!.Value[i] -
    ↪  _test.U(_mesh.Points[i]));
190         }
191
192         Array.ForEach(error, Console.WriteLine);
193
194         var sum = error.Sum(t => t * t);
195
196         sum = Math.Sqrt(sum / _mesh.Points.Count);
197
198         Console.WriteLine($"rms = {sum}");
199
200         // using var sw = new StreamWriter("output/3.csv");
201         //
202         // for (int i = 0; i < error.Length; i++)
203         // {
204         //     if (i == 0)
205         //     {
206         //         sw.WriteLine("$i$, $u_i^*$, $u_i$, $|u^* - u|$, Погрешность");
207         //         sw.WriteLine(
208         //             $"{i}, {_test.U(_mesh.Points[i])},
    ↪  {_iterativeSolver.Solution!.Value[i]}, {error[i]}, {sum}");
```

```
        //         continue;
        //     }
        //
        //     sw.WriteLine($"{i}, {_test.U(_mesh.Points[i])},
↪   {_iterativeSolver.Solution!.Value[i]}, {error[i]},");
        // }
    }

    public static SolverFemBuilder CreateBuilder() => new();
}
```

Assembler.cs

```
namespace Project;

public abstract class BaseMatrixAssembler
{
    protected readonly IBasis _basis;
    protected readonly IBaseMesh _mesh;
    protected readonly Integration _integrator;
    protected Matrix[]? _baseStiffnessMatrix;
    protected Matrix? _baseMassMatrix;

    public SparseMatrix? GlobalMatrix { get; set; } // need initialize with portrait
↪   builder
    public Matrix StiffnessMatrix { get; }
    public Matrix MassMatrix { get; }
    public int BasisSize => _basis.Size;

    protected BaseMatrixAssembler(IBasis basis, Integration integrator, IBaseMesh mesh)
    {
        _basis = basis;
        _integrator = integrator;
        _mesh = mesh;
        StiffnessMatrix = new(_basis.Size);
        MassMatrix = new(_basis.Size);
    }

    public abstract void BuildLocalMatrices(int ielem);

    public void FillGlobalMatrix(int i, int j, double value)
    {
        if (GlobalMatrix is null)
        {
            throw new("Initialize the global matrix (use portrait builder)!");
        }

        if (i == j)
        {
            GlobalMatrix.Di[i] += value;
            return;
        }

        if (i <= j) return;
        for (int ind = GlobalMatrix.Ig[i]; ind < GlobalMatrix.Ig[i + 1]; ind++)
        {
            if (GlobalMatrix.Jg[ind] != j) continue;
            GlobalMatrix.Gg[ind] += value;
            return;
        }
    }
```

```
}

public class BiMatrixAssembler : BaseMatrixAssembler
{
    public BiMatrixAssembler(IBasis basis, Integration integrator, IBaseMesh mesh) :
↪   base(basis, integrator, mesh)
    {
    }

    public override void BuildLocalMatrices(int ielem)
    {
        var element = _mesh.Elements[ielem];
        var bPoint = _mesh.Points[element[0]];
        var ePoint = _mesh.Points[element[^1]];

        double hx = ePoint.X - bPoint.X;
        double hy = ePoint.Y - bPoint.Y;

        if (_baseStiffnessMatrix is null)
        {
            _baseStiffnessMatrix = new Matrix[] { new(_basis.Size), new(_basis.Size) };
            _baseMassMatrix = new(_basis.Size);
            var templateElement = new Rectangle(new(0.0, 0.0), new(1.0, 1.0));

            for (int i = 0; i < _basis.Size; i++)
            {
                for (int j = 0; j <= i; j++)
                {
                    Func<Point2D, double> function;

                    for (int k = 0; k < 2; k++)
                    {
                        var ik = i;
                        var jk = j;
                        var k1 = k;
                        function = p =>
                        {
                            var dFi1 = _basis.GetDPsi(ik, k1, p);
                            var dFi2 = _basis.GetDPsi(jk, k1, p);

                            return dFi1 * dFi2;
                        };

                        _baseStiffnessMatrix[k][i, j] = _baseStiffnessMatrix[k][j, i] =
                            _integrator.Gauss2D(function, templateElement);
                    }

                    var i1 = i;
                    var j1 = j;
                    function = p =>
                    {
                        var fi1 = _basis.GetPsi(i1, p);
                        var fi2 = _basis.GetPsi(j1, p);

                        return fi1 * fi2;
                    };
                    _baseMassMatrix[i, j] = _baseMassMatrix[j, i] =
↪   _integrator.Gauss2D(function, templateElement);
                }
            }
```

```
106         }
107
108         for (int i = 0; i < _basis.Size; i++)
109         {
110             for (int j = 0; j <= i; j++)
111             {
112                 StiffnessMatrix[i, j] = StiffnessMatrix[j, i] =
113                     hy / hx * _baseStiffnessMatrix[0][i, j] + hx / hy *
    ↪  _baseStiffnessMatrix[1][i, j];
114             }
115         }
116
117         for (int i = 0; i < _basis.Size; i++)
118         {
119             for (int j = 0; j <= i; j++)
120             {
121                 MassMatrix[i, j] = MassMatrix[j, i] = hx * hy * _baseMassMatrix![i, j];
122             }
123         }
124     }
125 }
126
127 public class CurveMatrixAssembler : BaseMatrixAssembler // maybe rename the class
128 {
129     public CurveMatrixAssembler(IBasis basis, Integration integrator, IBaseMesh mesh)
    ↪  : base(basis, integrator, mesh)
130     {
131         _baseStiffnessMatrix = new Matrix[] { new(_basis.Size), new(_basis.Size) };
132         _baseMassMatrix = new(_basis.Size);
133     }
134
135     public override void BuildLocalMatrices(int ielem)
136     {
137         var templateElement = new Rectangle(new(0.0, 0.0), new(1.0, 1.0));
138
139         for (int i = 0; i < _basis.Size; i++)
140         {
141             for (int j = 0; j <= i; j++)
142             {
143                 var i1 = i;
144                 var j1 = j;
145                 Func<Point2D, double> function = p =>
146                 {
147                     var dxFi1 = _basis.GetDPsi(i1, 0, p);
148                     var dxFi2 = _basis.GetDPsi(j1, 0, p);
149                     var dyFi1 = _basis.GetDPsi(i1, 1, p);
150                     var dyFi2 = _basis.GetDPsi(j1, 1, p);
151                     var calculates = CalculateJacobian(ielem, p);
152
153                     return ((calculates.Reverse[0, 0] * dxFi1 + calculates.Reverse[0,
    ↪  1] * dyFi1) *
154                             (calculates.Reverse[0, 0] * dxFi2 + calculates.Reverse[0,
    ↪  1] * dyFi2) +
155                             (calculates.Reverse[1, 0] * dxFi1 + calculates.Reverse[1,
    ↪  1] * dyFi1) *
156                             (calculates.Reverse[1, 0] * dxFi2 + calculates.Reverse[1,
    ↪  1] * dyFi2)) *
157                            Math.Abs(calculates.Determinant);
158                 };
159
```

```
                _baseStiffnessMatrix![0][i, j] =
                    _baseStiffnessMatrix[0][j, i] = _integrator.Gauss2D(function,
↪   templateElement);

                function = p =>
                {
                    var fi1 = _basis.GetPsi(i1, p);
                    var fi2 = _basis.GetPsi(j1, p);
                    var calculates = CalculateJacobian(ielem, p);

                    return fi1 * fi2 * Math.Abs(calculates.Determinant);
                };
                _baseMassMatrix![i, j] = _baseMassMatrix[j, i] =
                    _integrator.Gauss2D(function, templateElement);
            }
        }

        for (int i = 0; i < _basis.Size; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                StiffnessMatrix[i, j] = StiffnessMatrix[j, i] =
↪   _baseStiffnessMatrix![0][i, j];
            }
        }

        for (int i = 0; i < _basis.Size; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                MassMatrix[i, j] = MassMatrix[j, i] = _baseMassMatrix![i, j];
            }
        }
    }

    private (double Determinant, Matrix Reverse) CalculateJacobian(int ielem, Point2D
↪   point)
    {
        var dx = new double[2];
        var dy = new double[2];

        var element = _mesh.Elements[ielem];

        for (int i = 0; i < _basis.Size; i++)
        {
            for (int k = 0; k < 2; k++)
            {
                dx[k] += _basis.GetDPsi(i, k, point) * _mesh.Points[element[i]].X;
                dy[k] += _basis.GetDPsi(i, k, point) * _mesh.Points[element[i]].Y;
            }
        }

        var jacobian = dx[0] * dy[1] - dx[1] * dy[0];

        var reverse = new Matrix(2)
        {
            [0, 0] = dy[1],
            [0, 1] = -dy[0],
            [1, 0] = -dx[1],
            [1, 1] = dx[0]
```

```
        };

        return (jacobian, 1.0 / jacobian * reverse);
    }
}
```

PortraitBuilder.cs

```
namespace Project;

public static class PortraitBuilder
{
    public static void Build(IBaseMesh mesh, out int[] ig, out int[] jg)
    {
        var connectivityList = new List<HashSet<int>>();

        for (int i = 0; i < mesh.Points.Count; i++)
        {
            connectivityList.Add(new());
        }

        int localSize = mesh.Elements[0].Count;

        foreach (var element in mesh.Elements.Select(list => list.OrderBy(node =>
↪   node).ToArray()))
        {
            for (int i = 0; i < localSize - 1; i++)
            {
                int nodeToInsert = element[i];

                for (int j = i + 1; j < localSize; j++)
                {
                    int posToInsert = element[j];

                    connectivityList[posToInsert].Add(nodeToInsert);
                }
            }
        }

        var orderedList = connectivityList.Select(list => list.OrderBy(val =>
↪   val)).ToList();

        ig = new int[connectivityList.Count + 1];

        ig[0] = 0;
        ig[1] = 0;

        for (int i = 1; i < connectivityList.Count; i++)
        {
            ig[i + 1] = ig[i] + connectivityList[i].Count;
        }

        jg = new int[ig[^1]];

        for (int i = 1, j = 0; i < connectivityList.Count; i++)
        {
            foreach (var it in orderedList[i])
            {
                jg[j++] = it;
            }
        }
```

```
    }
}
```

Matrices.cs

```
namespace Project;

public class SparseMatrix
{
    public int[] Ig { get; init; }
    public int[] Jg { get; init; }
    public double[] Di { get; }
    public double[] Gg { get; }
    public int Size { get; }

    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;
            }
        }
    }
}

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

    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;
    }

    public static Matrix operator *(double value, Matrix matrix)
    {
        Matrix resultMatrix = new(matrix.Size);

        for (int i = 0; i < resultMatrix.Size; i++)
        {
            for (int j = 0; j < resultMatrix.Size; j++)
            {
                resultMatrix[i, j] = value * matrix[i, j];
            }
        }

        return resultMatrix;
    }
}
```

Vector.cs

```
namespace Project;

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

    public T this[int idx]
    {
        get => _storage[idx];
        set => _storage[idx] = value;
    }

    public Vector(int length)
        => (Length, _storage) = (length, new T[length]);

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

        for (int i = 0; i < a.Length; i++)
        {
            result += a[i] * b[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[i] = vector[i] * T.CreateChecked(constant);
        }
```

```
37
38          return result;
39      }
40
41      public static Vector<T> operator +(Vector<T> a, Vector<T> b)
42      {
43          Vector<T> result = new(a.Length);
44
45          for (int i = 0; i < a.Length; i++)
46          {
47              result[i] = a[i] + b[i];
48          }
49
50          return result;
51      }
52
53      public static Vector<T> operator -(Vector<T> a, Vector<T> b)
54      {
55          Vector<T> result = new(a.Length);
56
57          for (int i = 0; i < a.Length; i++)
58          {
59              result[i] = a[i] - b[i];
60          }
61
62          return result;
63      }
64
65      public static void Copy(Vector<T> source, Vector<T> destination)
66      {
67          for (int i = 0; i < source.Length; i++)
68          {
69              destination[i] = source[i];
70          }
71      }
72
73      public static Vector<T> Copy(Vector<T> otherVector)
74      {
75          Vector<T> newVector = new(otherVector.Length);
76
77          Array.Copy(otherVector._storage, newVector._storage, otherVector.Length);
78
79          return newVector;
80      }
81
82      public void Fill(double value)
83      {
84          for (int i = 0; i < Length; i++)
85          {
86              _storage[i] = T.CreateChecked(value);
87          }
88      }
89
90      public double Norm()
91      {
92          T result = T.Zero;
93
94          for (int i = 0; i < Length; i++)
95          {
96              result += _storage[i] * _storage[i];
```

```
        }

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

    public ImmutableArray<T> ToImmutableArray()
        => ImmutableArray.Create(_storage);

    public IEnumerator<T> GetEnumerator()
    {
        foreach (T value in _storage)
        {
            yield return value;
        }
    }

    IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

    public void Add(IEnumerable<T> collection)
    {
        var enumerable = collection as T[] ?? collection.ToArray();

        if (Length != enumerable.Length)
        {
            throw new ArgumentOutOfRangeException(nameof(collection), "Sizes of vector
↪   and collection not equal");
        }

        for (int i = 0; i < Length; i++)
        {
            _storage[i] = enumerable[i];
        }
    }
}
```

Solvers.cs

```
namespace Project;

public abstract class IterativeSolver
{
    protected TimeSpan? _runningTime;
    protected SparseMatrix _matrix = default!;
    protected Vector<double> _vector = default!;
    protected Vector<double>? _solution;

    public int MaxIters { get; }
    public double Eps { get; }
    public TimeSpan? RunningTime => _runningTime;
    public ImmutableArray<double>? Solution => _solution?.ToImmutableArray();

    protected IterativeSolver(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();
```

```
25
26      protected void Cholesky(double[] ggnew, double[] dinew)
27      {
28          double suml = 0.0;
29          double sumdi = 0.0;
30
31          for (int i = 0; i < _matrix.Size; i++)
32          {
33              int i0 = _matrix.Ig[i];
34              int i1 = _matrix.Ig[i + 1];
35
36              for (int k = i0; k < i1; k++)
37              {
38                  int j = _matrix.Jg[k];
39                  int j0 = _matrix.Ig[j];
40                  int j1 = _matrix.Ig[j + 1];
41                  int ik = i0;
42                  int kj = j0;
43
44                  while (ik < k && kj < j1)
45                  {
46                      if (_matrix.Jg[ik] == _matrix.Jg[kj])
47                      {
48                          suml += ggnew[ik] * ggnew[kj];
49                          ik++;
50                          kj++;
51                      }
52                      else
53                      {
54                          if (_matrix.Jg[ik] > _matrix.Jg[kj])
55                              kj++;
56                          else
57                              ik++;
58                      }
59                  }
60
61                  ggnew[k] = (ggnew[k] - suml) / dinew[j];
62                  sumdi += ggnew[k] * ggnew[k];
63                  suml = 0.0;
64              }
65
66              dinew[i] = Math.Sqrt(dinew[i] - sumdi);
67              sumdi = 0.0;
68          }
69      }
70
71      protected Vector<double> MoveForCholesky(Vector<double> vector, double[] ggnew,
    ↪  double[] dinew)
72      {
73          Vector<double> y = new(vector.Length);
74          Vector<double> x = new(vector.Length);
75          Vector<double>.Copy(vector, y);
76
77          double sum = 0.0;
78
79          for (int i = 0; i < _matrix.Size; i++) // Прямой ход
80          {
81              int i0 = _matrix.Ig[i];
82              int i1 = _matrix.Ig[i + 1];
83
```

```
            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;
    }
}

public class CGM : IterativeSolver
{
    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 vectorNorm = _vector.Norm();

            _solution = new(_vector.Length);

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

            Stopwatch sw = Stopwatch.StartNew();

            var r = _vector - (_matrix * _solution);

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

            for (int iter = 0; iter < MaxIters && r.Norm() / vectorNorm >= Eps; iter++)
            {
                var tmp = _matrix * z;
                var alpha = r * r / (tmp * z);
                _solution += alpha * z;
                var squareNorm = r * r;
                r -= alpha * tmp;
                var beta = r * r / squareNorm;
                z = r + beta * z;
            }
```

```
142
143             sw.Stop();
144
145             _runningTime = sw.Elapsed;
146         }
147         catch (ArgumentNullException ex)
148         {
149             Console.WriteLine($"We had problem: {ex.Message}");
150             throw;
151         }
152         catch (Exception ex)
153         {
154             Console.WriteLine($"We had problem: {ex.Message}");
155         }
156     }
157 }
158
159 public class CGMCholesky : IterativeSolver
160 {
161     public CGMCholesky(int maxIters, double eps) : base(maxIters, eps)
162     {
163     }
164
165     public override void Compute()
166     {
167         try
168         {
169             ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be
    ↪  null, set the matrix");
170             ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be
    ↪  null, set the vector");
171
172             double vectorNorm = _vector.Norm();
173
174             _solution = new(_vector.Length);
175
176             double[] ggnew = new double[_matrix.Gg.Length];
177             double[] dinew = new double[_matrix.Di.Length];
178
179             _matrix.Gg.Copy(ggnew);
180             _matrix.Di.Copy(dinew);
181
182             Stopwatch sw = Stopwatch.StartNew();
183
184             Cholesky(ggnew, dinew);
185
186             var r = _vector - _matrix * _solution;
187             var z = MoveForCholesky(r, ggnew, dinew);
188
189             for (int iter = 0; iter < MaxIters && r.Norm() / vectorNorm >= Eps; iter++)
190             {
191                 var tmp = MoveForCholesky(r, ggnew, dinew) * r;
192                 var sndTemp = _matrix * z;
193                 var alpha = tmp / (sndTemp * z);
194                 _solution += alpha * z;
195                 r -= alpha * sndTemp;
196                 var fstTemp = MoveForCholesky(r, ggnew, dinew);
197                 var beta = fstTemp * r / tmp;
198                 z = fstTemp + beta * z;
199             }
```

```
200
201             sw.Stop();
202
203             _runningTime = sw.Elapsed;
204         }
205         catch (ArgumentNullException ex)
206         {
207             Console.WriteLine($"We had problem: {ex.Message}");
208             throw;
209         }
210         catch (Exception ex)
211         {
212             Console.WriteLine($"We had problem: {ex.Message}");
213         }
214     }
215 }
```

Integration.cs

```
1  namespace Project;
2
3  public class Integration
4  {
5      private readonly IEnumerable<QuadratureNode<double>> _quadratures;
6
7      public Integration(IEnumerable<QuadratureNode<double>> quadratures) =>
   ↪  _quadratures = quadratures;
8
9      public double Gauss2D(Func<Point2D, double> psi, Rectangle element)
10     {
11         double hx = element.RightTop.X - element.LeftTop.X;
12         double hy = element.RightTop.Y - element.RightBottom.Y;
13
14         var result = (from qi in _quadratures
15             from qj in _quadratures
16             let point = new Point2D((qi.Node * hx + element.LeftBottom.X +
   ↪  element.RightBottom.X) / 2.0,
17                 (qj.Node * hy + element.RightBottom.Y + element.RightTop.Y) / 2.0)
18             select psi(point) * qi.Weight * qj.Weight).Sum();
19
20         return result * hx * hy / 4.0;
21     }
22 }
```

Quadratures.cs

```
1  namespace Project;
2
3  public class QuadratureNode<T> where T : notnull
4  {
5      public T Node { get; }
6      public double Weight { get; }
7
8      public QuadratureNode(T node, double weight)
9      {
10         Node = node;
11         Weight = weight;
12     }
13 }
14
15 public static class Quadratures
```

```
{
    public static IEnumerable<QuadratureNode<double>> SegmentGaussOrder5()
    {
        const int n = 3;
        double[] points =
        {
            0,
            -Math.Sqrt(3.0 / 5.0),
            Math.Sqrt(3.0 / 5.0)
        };
        double[] weights =
        {
            8.0 / 9.0,
            5.0 / 9.0,
            5.0 / 9.0
        };

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

    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(points[i], weights[i]);
        }
    }
}
```

Basis.cs

```
namespace Project;

public interface IBasis
{
    int Size { get; }

    public double GetPsi(int number, Point2D point);

```

```
    public double GetDPsi(int number, int varNumber, Point2D point);
}

public readonly record struct LinearBasis : IBasis
{
    public int Size => 4;

    public double GetPsi(int number, Point2D point)
        => number switch
        {
            0 => (1.0 - point.X) * (1.0 - point.Y),
            1 => point.X * (1.0 - point.Y),
            2 => (1.0 - point.X) * point.Y,
            3 => point.X * point.Y,
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not
↪   expected function number!")
        };

    public double GetDPsi(int number, int varNumber, Point2D point)
        => varNumber switch
        {
            0 => number switch
            {
                0 => point.Y - 1.0,
                1 => 1.0 - point.Y,
                2 => -point.Y,
                3 => point.Y,
                _ => throw new ArgumentOutOfRangeException(nameof(number), number,
↪   "Not expected function number!")
            },
            1 => number switch
            {
                0 => point.X - 1.0,
                1 => -point.X,
                2 => 1.0 - point.X,
                3 => point.X,
                _ => throw new ArgumentOutOfRangeException(nameof(number), number,
↪   "Not expected function number!")
            },
            _ => throw new ArgumentOutOfRangeException(nameof(varNumber), varNumber,
↪   "Not expected var number!")
        };
}

public readonly record struct QuadraticBasis : IBasis
{
    public int Size => 9;

    public double GetPsi(int number, Point2D point)
        => number switch
        {
            0 => 4.0 * (point.X - 0.5) * (point.X - 1.0) * (point.Y - 0.5) * (point.Y
↪   - 1.0),
            1 => -8.0 * point.X * (point.X - 1.0) * (point.Y - 0.5) * (point.Y - 1.0),
            2 => 4.0 * point.X * (point.X - 0.5) * (point.Y - 0.5) * (point.Y - 1.0),
            3 => -8.0 * (point.X - 0.5) * (point.X - 1.0) * point.Y * (point.Y - 1.0),
            4 => 16.0 * point.X * point.Y * (point.X - 1.0) * (point.Y - 1.0),
            5 => -8.0 * point.X * point.Y * (point.X - 0.5) * (point.Y - 1.0),
            6 => 4.0 * point.Y * (point.X - 0.5) * (point.X - 1.0) * (point.Y - 0.5),
            7 => -8.0 * point.X * point.Y * (point.X - 1.0) * (point.Y - 0.5),
```

```
            8 => 4.0 * point.X * point.Y * (point.X - 0.5) * (point.Y - 0.5),
            _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not expected function number!")
        };

    public double GetDPsi(int number, int varNumber, Point2D point)
        => varNumber switch
        {
            0 => number switch
            {
                0 => 4.0 * (point.X - 1.0 + (point.X - 0.5)) * (point.Y - 0.5) * (point.Y - 1.0),
                1 => -8.0 * (point.X - 1.0 + point.X) * (point.Y - 0.5) * (point.Y - 1.0),
                2 => 4.0 * (point.X - 0.5 + point.X) * (point.Y - 0.5) * (point.Y - 1.0),
                3 => -8.0 * (point.X - 1.0 + (point.X - 0.5)) * point.Y * (point.Y - 1.0),
                4 => 16.0 * (point.X - 1.0 + point.X) * point.Y * (point.Y - 1.0),
                5 => -8.0 * (point.X - 0.5 + point.X) * point.Y * (point.Y - 1.0),
                6 => 4.0 * (point.X - 1.0 + (point.X - 0.5)) * point.Y * (point.Y - 0.5),
                7 => -8.0 * (point.X - 1.0 + point.X) * point.Y * (point.Y - 0.5),
                8 => 4.0 * (point.X - 0.5 + point.X) * point.Y * (point.Y - 0.5),
                _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not expected function number!")
            },
            1 => number switch
            {
                0 => 4.0 * (point.X - 0.5) * (point.X - 1.0) * (point.Y - 1.0 + (point.Y - 0.5)),
                1 => -8.0 * point.X * (point.X - 1.0) * (point.Y - 1.0 + (point.Y - 0.5)),
                2 => 4.0 * point.X * (point.X - 0.5) * (point.Y - 1.0 + (point.Y - 0.5)),
                3 => -8.0 * (point.X - 0.5) * (point.X - 1.0) * (point.Y - 1.0 + point.Y),
                4 => 16.0 * point.X * (point.X - 1.0) * (point.Y - 1.0 + point.Y),
                5 => -8.0 * point.X * (point.X - 0.5) * (point.Y - 1.0 + point.Y),
                6 => 4.0 * (point.X - 0.5) * (point.X - 1.0) * (point.Y - 0.5 + point.Y),
                7 => -8.0 * point.X * (point.X - 1.0) * (point.Y - 0.5 + point.Y),
                8 => 4.0 * point.X * (point.X - 0.5) * (point.Y - 0.5 + point.Y),
                _ => throw new ArgumentOutOfRangeException(nameof(number), number, "Not expected function number!")
            },
            _ => throw new ArgumentOutOfRangeException(nameof(varNumber), varNumber, "Not expected var number!")
        };
}
```

Boundaries.cs

```
namespace Project;

public interface IBoundary
{
    int Node { get; }
    double Value { get; set; }
}

```

```
public class DirichletBoundary : IBoundary
{
    public int Node { get; }
    public double Value { get; set; }

    public DirichletBoundary(int node, double value) => (Node, Value) = (node, value);
}

public readonly record struct BoundaryParameters
{
    public required byte LeftBorder { get; init; }
    public required byte RightBorder { get; init; }
    public required byte BottomBorder { get; init; }
    public required byte TopBorder { get; init; }

    public static BoundaryParameters ReadJson(string jsonPath)
    {
        if (!File.Exists(jsonPath))
        {
            throw new("File does not exist");
        }

        using var sr = new StreamReader(jsonPath);
        return JsonConvert.DeserializeObject<BoundaryParameters>(sr.ReadToEnd());
    }
}

public interface IBoundaryHandler
{
    IEnumerable<IBoundary> Process();
}

public class LinearBoundaryHandler : IBoundaryHandler
{
    private readonly BoundaryParameters? _parameters;
    private readonly MeshParameters? _meshParameters;

    public LinearBoundaryHandler(BoundaryParameters? parameters, IParameters?
↪   meshParameters)
        => (_parameters, _meshParameters) = (parameters,
            (MeshParameters)(meshParameters ?? throw new
↪   ArgumentNullException(nameof(meshParameters))));

    public IEnumerable<IBoundary> Process() // for now only Dirichlet
    {
        if (_parameters!.Value.TopBorder == 1)
        {
            int startingNode = (_meshParameters!.Value.SplitsX + 1) *
↪   _meshParameters.Value.SplitsY;

            for (int i = 0; i < _meshParameters.Value.SplitsX + 1; i++)
            {
                yield return new DirichletBoundary(startingNode + i, 0.0);
            }
        }

        if (_parameters.Value.BottomBorder == 1)
        {
            for (int i = 0; i < _meshParameters!.Value.SplitsX + 1; i++)
            {
```

```
                yield return new DirichletBoundary(i, 0.0);
            }
        }

        if (_parameters.Value.LeftBorder == 1)
        {
            for (int i = 0; i < _meshParameters!.Value.SplitsY + 1; i++)
            {
                yield return new DirichletBoundary(i * (_meshParameters.Value.SplitsX
↪   + 1), 0.0);
            }
        }

        if (_parameters.Value.RightBorder != 1) yield break;
        {
            for (int i = 0; i < _meshParameters!.Value.SplitsY + 1; i++)
            {
                yield return new DirichletBoundary(
                    i * _meshParameters.Value.SplitsX + _meshParameters.Value.SplitsX
↪   + i, 0.0);
            }
        }
    }
}

public class QuadraticBoundaryHandler : IBoundaryHandler
{
    private readonly BoundaryParameters? _parameters;
    private readonly MeshParameters? _meshParameters;

    public QuadraticBoundaryHandler(BoundaryParameters? parameters, IParameters
↪   meshParameters)
        => (_parameters, _meshParameters) = (parameters,
↪   (MeshParameters)meshParameters);

    public IEnumerable<IBoundary> Process() // for now only Dirichlet
    {
        if (_parameters!.Value.TopBorder == 1)
        {
            int startingNode = (2 * _meshParameters!.Value.SplitsX + 1) * 2 *
↪   _meshParameters.Value.SplitsY;

            for (int i = 0; i < 2 * _meshParameters.Value.SplitsX + 1; i++)
            {
                yield return new DirichletBoundary(startingNode + i, 0.0);
            }
        }

        if (_parameters.Value.BottomBorder == 1)
        {
            for (int i = 0; i < 2 * _meshParameters!.Value.SplitsX + 1; i++)
            {
                yield return new DirichletBoundary(i, 0.0);
            }
        }

        if (_parameters.Value.LeftBorder == 1)
        {
            for (int i = 0; i < 2 * _meshParameters!.Value.SplitsY + 1; i++)
            {
```

```
                yield return new DirichletBoundary(i * (2 *
↪   _meshParameters.Value.SplitsX + 1), 0.0);
            }
        }

        if (_parameters.Value.RightBorder != 1) yield break;
        {
            for (int i = 0; i < 2 * _meshParameters!.Value.SplitsY + 1; i++)
            {
                yield return new DirichletBoundary(
                    i * 2 * _meshParameters.Value.SplitsX + 2 *
↪   _meshParameters.Value.SplitsX + i, 0.0);
            }
        }
    }
}

public class CurveLinearBoundaryHandler : IBoundaryHandler
{
    private readonly BoundaryParameters? _parameters;
    private readonly CurveMeshParameters? _meshParameters;

    public CurveLinearBoundaryHandler(BoundaryParameters? parameters, IParameters?
↪   meshParameters)
        => (_parameters, _meshParameters) = (parameters,
            (CurveMeshParameters)(meshParameters ?? throw new
↪   ArgumentNullException(nameof(meshParameters))));

    public IEnumerable<IBoundary> Process() // for now only Dirichlet
    {
        var array = new DirichletBoundary[2 * _meshParameters!.Steps];

        for (int i = 0,
             k = _meshParameters!.Steps,
             j = (_meshParameters.RadiiCounts!.Value - 1) * _meshParameters.Steps;
             i < array.Length / 2;
             i++, j++, k++)
        {
            array[i] = new(i, 0.0);
            array[k] = new(j, 0.0);
        }

        return array;
    }
}

public class CurveQuadraticBoundaryHandler : IBoundaryHandler
{
    private readonly BoundaryParameters? _parameters;
    private readonly CurveMeshParameters? _meshParameters;

    public CurveQuadraticBoundaryHandler(BoundaryParameters? parameters, IParameters?
↪   meshParameters)
        => (_parameters, _meshParameters) = (parameters,
            (CurveMeshParameters)(meshParameters ?? throw new
↪   ArgumentNullException(nameof(meshParameters))));

    public IEnumerable<IBoundary> Process() // for now only Dirichlet
    {
        var array = new DirichletBoundary[4 * _meshParameters!.Steps];
```

```
175
176         for (int i = 0,
177              k = 2 * _meshParameters!.Steps,
178              j = 2 * (_meshParameters.RadiiCounts!.Value - 1) * _meshParameters.Steps;
179              i < array.Length / 2;
180              i++, j++, k++)
181         {
182             array[i] = new(i, 0.0);
183             array[k] = new(j, 0.0);
184         }
185
186         return array;
187     }
188 }
```

Tests.cs

```
1  namespace Project.Tests;
2
3  public interface ITest
4  {
5      double U(Point2D point);
6
7      double F(Point2D point);
8  }
9
10 public class Test1 : ITest
11 {
12     public double U(Point2D point) => point.X + point.Y;
13
14     public double F(Point2D point) => 0.0;
15 }
16
17 public class Test2 : ITest
18 {
19     public double U(Point2D point) => point.X * point.X + point.Y * point.Y;
20
21     public double F(Point2D point) => -4.0;
22 }
23
24 public class Test3 : ITest
25 {
26     public double U(Point2D point) => point.X * point.X * point.X * point.Y * point.Y
   ↪  * point.Y;
27
28     public double F(Point2D point) =>
29         -6.0 * point.X * point.Y * point.Y * point.Y - 6.0 * point.Y * point.X *
   ↪  point.X * point.X;
30 }
31
32 public class Test4 : ITest
33 {
34     public double U(Point2D point) => point.X * point.X * point.X * point.X + point.Y
   ↪  * point.Y * point.Y * point.Y;
35
36     public double F(Point2D point) => -12.0 * point.X * point.X - 12.0 * point.Y *
   ↪  point.Y;
37 }
38
39 public class Test5 : ITest
40 {
```

```
41      public double U(Point2D point) => Math.Sin(point.X);
42
43      public double F(Point2D point) => Math.Sin(point.X);
44  }
45
46  public class Test6 : ITest
47  {
48      public double U(Point2D point) => Math.Exp(point.X + point.Y);
49
50      public double F(Point2D point) => -2.0 * Math.Exp(point.X + point.Y);
51  }
52
53  public class Test7 : ITest
54  {
55      public double U(Point2D point) => (point.X + 1) * (point.X + 1) * (point.X + 1) *
    ↪  point.Y * point.Y * point.Y;
56
57      public double F(Point2D point) => -6.0 * (point.X + 1) * point.Y * point.Y *
    ↪  point.Y -
58                                        6.0 * point.Y * (point.X + 1) * (point.X + 1) *
    ↪  (point.X + 1);
59  }
```

Meshes.cs

```
1   namespace Project.Meshes;
2
3   public interface IMeshCreator
4   {
5       IBaseMesh CreateMesh(IParameters meshParameters, MeshBuilder? meshBuilder = null);
6   }
7
8   public class RegularMeshCreator : IMeshCreator
9   {
10      public IBaseMesh CreateMesh(IParameters meshParameters, MeshBuilder? meshBuilder =
    ↪  null) =>
11          new RegularMesh(meshParameters, meshBuilder ?? new LinearMeshBuilder());
12  }
13
14  // public class IrregularMesh : BaseMesh TODO maybe
15
16  public abstract class MeshBuilder
17  {
18      protected abstract int SizeElement { get; }
19
20      public abstract (List<Point2D>, int[][]) Build(IParameters meshParameters);
21
22      protected (List<Point2D>, int[][]) BaseBuild(IParameters meshParameters)
23      {
24          if (meshParameters is not MeshParameters parameters)
25          {
26              throw new ArgumentNullException(nameof(parameters), "Parameters mesh is
    ↪  null!");
27          }
28
29          if (parameters.SplitsX is < 1 or < 1)
30          {
31              throw new("The number of splits must be greater than or equal to 1");
32          }
33
34          var result = new
```

```
        {
            Points = new List<Point2D>(),
            Elements = new int[parameters.SplitsX * parameters.SplitsY][].Select(_ =>
    new int[SizeElement])
                .ToArray(),
        };

        double hx = parameters.IntervalX.Length / parameters.SplitsX;
        double hy = parameters.IntervalY.Length / parameters.SplitsY;

        double[] pointsX = new double[parameters.SplitsX + 1];
        double[] pointsY = new double[parameters.SplitsY + 1];

        pointsX[0] = parameters.IntervalX.LeftBorder;
        pointsY[0] = parameters.IntervalY.LeftBorder;

        for (int i = 1; i < parameters.SplitsX + 1; i++)
        {
            pointsX[i] = pointsX[i - 1] + hx;
        }

        for (int i = 1; i < parameters.SplitsY + 1; i++)
        {
            pointsY[i] = pointsY[i - 1] + hy;
        }

        for (int j = 0; j < parameters.SplitsY + 1; j++)
        {
            for (int i = 0; i < parameters.SplitsX + 1; i++)
            {
                result.Points.Add(new(pointsX[i], pointsY[j]));
            }
        }

        int nx = parameters.SplitsX + 1;
        int index = 0;

        for (int j = 0; j < parameters.SplitsY; j++)
        {
            for (int i = 0; i < parameters.SplitsX; i++)
            {
                result.Elements[index][0] = i + j * nx;
                result.Elements[index][1] = i + 1 + j * nx;
                result.Elements[index][2] = i + (j + 1) * nx;
                result.Elements[index++][3] = i + 1 + (j + 1) * nx;
            }
        }

        return (result.Points, result.Elements);
    }
}

public class LinearMeshBuilder : MeshBuilder
{
    protected override int SizeElement => 4;

    public override (List<Point2D>, int[][]) Build(IParameters meshParameters)
    {
        if (meshParameters is not MeshParameters parameters)
        {
```

```
94              throw new ArgumentNullException(nameof(parameters), "Parameters mesh is
    ↪  null!");
95          }
96
97          var result = BaseBuild(meshParameters);
98
99          return (result.Item1, result.Item2);
100     }
101 }
102
103 public class QuadraticMeshBuilder : MeshBuilder
104 {
105     protected override int SizeElement => 9;
106
107     public override (List<Point2D>, int[][]) Build(IParameters meshParameters)
108     {
109         if (meshParameters is not MeshParameters parameters)
110         {
111             throw new ArgumentNullException(nameof(parameters), "Parameters mesh is
    ↪  null!");
112         }
113
114         (List<Point2D> Points, int[][] Elements) result = BaseBuild(meshParameters);
115         var pointsX = new double[2 * parameters.SplitsX + 1];
116         var pointsY = new double[2 * parameters.SplitsY + 1];
117         var vertices = new Point2D[9];
118
119         pointsX.Fill(int.MinValue);
120         pointsY.Fill(int.MinValue);
121
122         foreach (var ielem in result.Elements)
123         {
124             var v1 = result.Points[ielem[0]];
125             var v2 = result.Points[ielem[1]];
126             var v3 = result.Points[ielem[2]];
127             var v4 = result.Points[ielem[3]];
128
129             RecalculatePoints(v1, v2, v3, v4);
130
131             pointsX = pointsX.Concat(vertices.Select(p => p.X)).ToArray();
132             pointsY = pointsY.Concat(vertices.Select(p => p.Y)).ToArray();
133         }
134
135         pointsX = pointsX.OrderBy(v => v).Distinct().ToArray();
136         pointsY = pointsY.OrderBy(v => v).Distinct().ToArray();
137         result.Points.Clear();
138
139         foreach (var pointY in pointsY.Skip(1))
140         {
141             foreach (var pointX in pointsX.Skip(1))
142             {
143                 result.Points.Add(new(pointX, pointY));
144             }
145         }
146
147         var nx = 2 * parameters.SplitsX + 1;
148         var index = 0;
149
150         for (int j = 0; j < parameters.SplitsY; j++)
151         {
```

```
            for (int i = 0; i < parameters.SplitsX; i++)
            {
                result.Elements[index][0] = i + j * 2 * nx + i;
                result.Elements[index][1] = i + 1 + 2 * j * nx + i;
                result.Elements[index][2] = i + 2 + 2 * j * nx + i;
                result.Elements[index][3] = i + nx + 2 * j * nx + i;
                result.Elements[index][4] = i + nx + 1 + 2 * j * nx + i;
                result.Elements[index][5] = i + nx + 2 + 2 * j * nx + i;
                result.Elements[index][6] = i + 2 * nx + 2 * j * nx + i;
                result.Elements[index][7] = i + 2 * nx + 1 + 2 * j * nx + i;
                result.Elements[index++][8] = i + 2 * nx + 2 + 2 * j * nx + i;
            }
        }

        return (result.Points, result.Elements);

        void RecalculatePoints(Point2D v1, Point2D v2, Point2D v3, Point2D v4)
        {
            vertices[0] = v1;
            vertices[1] = v2;
            vertices[2] = v3;
            vertices[3] = v4;
            vertices[4] = (v1 + v2) / 2.0;
            vertices[5] = (v3 + v4) / 2.0;
            vertices[6] = (v1 + v3) / 2.0;
            vertices[7] = (v4 + v2) / 2.0;
            vertices[8] = (vertices[4] + vertices[5]) / 2.0;
        }
    }
}

public class CurveLinearMeshBuilder : MeshBuilder
{
    protected override int SizeElement => 4;

    public override (List<Point2D>, int[][]) Build(IParameters meshParameters)
    {
        if (meshParameters is not CurveMeshParameters parameters)
        {
            throw new ArgumentNullException(nameof(parameters), "Parameters mesh is
↪   null!");
        }

        var radiiList = new List<double> { parameters.Radius2, parameters.Radius1 };

        int count;
        for (int k = 0; k < parameters.Splits; k++)
        {
            count = radiiList.Count;

            for (int i = 0; i < count - 1; i++)
            {
                radiiList.Add((radiiList[i] + radiiList[i + 1]) / 2.0);
            }

            radiiList = radiiList.OrderByDescending(v => v).ToList();
        }

        var result = new
        {
```

```
            Points = new List<Point2D>(),
            Elements = new int[parameters.Steps * (radiiList.Count - 1)][].Select(_ =>
↪   new int[SizeElement])
                .ToArray()
        };

        foreach (var radius in radiiList)
        {
            for (int i = 0; i < parameters.Steps; i++)
            {
                double newX = radius * Math.Cos(parameters.Angle * i) +
↪   parameters.Center.X;
                double newY = radius * Math.Sin(parameters.Angle * i) +
↪   parameters.Center.Y;

                result.Points.Add(new(newX, newY));
            }
        }

        parameters.RadiiCounts = radiiList.Count; // TODO

        var idx = 0;
        var pass = false;
        var step = 0;
        count = 0;

        for (int i = 0; i < (radiiList.Count - 1) * parameters.Steps; i++)
        {
            if (!pass)
            {
                result.Elements[idx][0] = i;
                result.Elements[idx][1] = i + 1;
                result.Elements[idx][2] = result.Elements[idx][0] + parameters.Steps;
                result.Elements[idx][3] = result.Elements[idx++][1] + parameters.Steps;
                step++;

                if (step != parameters.Steps - 1) continue;
                pass = true;
                step = 0;
            }
            else
            {
                result.Elements[idx][0] = count * parameters.Steps;
                result.Elements[idx][1] = result.Elements[idx - 1][1];
                result.Elements[idx][2] = result.Elements[idx - 1][1] + 1;
                count++;
                result.Elements[idx++][3] = (count + 1) * parameters.Steps - 1;
                pass = false;
            }
        }

        using StreamWriter sw1 = new("output/linearPoints.txt"),
            sw2 = new("output/points.txt"),
            sw3 = new("output/elements.txt");

        foreach (var point in result.Points)
        {
            sw1.WriteLine($"{point.X} {point.Y}");
        }

```

```
268         foreach (var element in result.Elements)
269         {
270             foreach (var node in element)
271             {
272                 sw3.Write(node + " ");
273             }
274 
275             sw3.WriteLine();
276         }
277 
278         return (result.Points, result.Elements.ToArray());
279     }
280 }
281 
282 public class CurveQuadraticMeshBuilder : MeshBuilder
283 {
284     protected override int SizeElement => 9;
285 
286     public override (List<Point2D>, int[][]) Build(IParameters meshParameters)
287     {
288         if (meshParameters is not CurveMeshParameters parameters)
289         {
290             throw new ArgumentNullException(nameof(parameters), "Parameters mesh is
    ↪  null!");
291         }
292 
293         var radiiList = new List<double>
294             { parameters.Radius2, (parameters.Radius1 + parameters.Radius2) / 2.0,
    ↪  parameters.Radius1 };
295 
296         int count;
297 
298         for (int k = 0; k < parameters.Splits; k++)
299         {
300             count = radiiList.Count;
301 
302             for (int i = 0; i < count - 1; i++)
303             {
304                 radiiList.Add((radiiList[i] + radiiList[i + 1]) / 2.0);
305             }
306 
307             radiiList = radiiList.OrderByDescending(v => v).ToList();
308         }
309 
310         var result = new
311         {
312             Points = new List<Point2D>(),
313             Elements = new int[parameters.Steps * (radiiList.Count / 2)][]
314                 .Select(_ => new int[SizeElement])
315                 .ToArray()
316         };
317 
318         int newSteps = parameters.Steps * 2;
319         double newAngle = 2.0 * Math.PI / newSteps;
320 
321         foreach (var radius in radiiList)
322         {
323             for (int i = 0; i < newSteps; i++)
324             {
325                 double newX = radius * Math.Cos(newAngle * i) + parameters.Center.X;
```

```
                double newY = radius * Math.Sin(newAngle * i) + parameters.Center.Y;

                result.Points.Add(new(newX, newY));
            }
        }

        parameters.RadiiCounts = radiiList.Count; // TODO

        var idx = 0;
        var pass = false;
        var step = 0;
        count = 0;

        for (int i = 0, k = 0; i < parameters.Steps * (radiiList.Count / 2); i++, k += 2)
        {
            if (!pass)
            {
                result.Elements[idx][0] = k;
                result.Elements[idx][1] = k + 1;
                result.Elements[idx][2] = k + 2;
                result.Elements[idx][3] = result.Elements[idx][0] + newSteps;
                result.Elements[idx][4] = result.Elements[idx][1] + newSteps;
                result.Elements[idx][5] = result.Elements[idx][2] + newSteps;
                result.Elements[idx][6] = result.Elements[idx][0] + 2 * newSteps;
                result.Elements[idx][7] = result.Elements[idx][1] + 2 * newSteps;
                result.Elements[idx][8] = result.Elements[idx++][2] + 2 * newSteps;
                step++;

                if (step != parameters.Steps - 1) continue;
                pass = true;
                step = 0;
            }
            else
            {
                result.Elements[idx][0] = result.Elements[idx - 1][2];
                result.Elements[idx][1] = result.Elements[idx][0] + 1;
                result.Elements[idx][2] = count * newSteps;
                count += 2;
                result.Elements[idx][3] = result.Elements[idx - 1][5];
                result.Elements[idx][4] = result.Elements[idx - 1][5] + 1;
                result.Elements[idx][5] = result.Elements[idx][2] + newSteps;
                result.Elements[idx][6] = result.Elements[idx - 1][8];
                result.Elements[idx][7] = result.Elements[idx - 1][8] + 1;
                result.Elements[idx][8] = result.Elements[idx][5] + newSteps;
                k = result.Elements[idx++][8] - 2;
                pass = false;
            }
        }

        using StreamWriter sw = new("output/points.txt");

        foreach (var point in result.Points)
        {
            sw.WriteLine($"{point.X} {point.Y}");
        }

        return (result.Points, result.Elements.ToArray());
    }
}
```

```
385
386 public interface IParameters
387 {
388     public static abstract IParameters ReadJson(string jsonPath);
389 }
390
391 public readonly record struct MeshParameters
392     (Interval IntervalX, int SplitsX, Interval IntervalY, int SplitsY) : IParameters
393 {
394     public static IParameters ReadJson(string jsonPath)
395     {
396         if (!File.Exists(jsonPath))
397         {
398             throw new("File does not exist");
399         }
400
401         using var sr = new StreamReader(jsonPath);
402         return JsonConvert.DeserializeObject<MeshParameters>(sr.ReadToEnd());
403     }
404 }
405
406 public class CurveMeshParameters : IParameters
407 {
408     [JsonIgnore] public double Angle { get; }
409     public Point2D Center { get; }
410     public double Radius1 { get; }
411     public double Radius2 { get; }
412     public int Steps { get; }
413     public int Splits { get; }
414     public int? RadiiCounts { get; set; }
415
416     [JsonConstructor]
417     public CurveMeshParameters(Point2D center, double radius1, double radius2, int
    ↪  steps, int splits)
418     {
419         Center = center;
420         Radius1 = radius1;
421         Radius2 = radius2;
422         Steps = steps;
423         Angle = 2.0 * Math.PI / Steps;
424         Splits = splits;
425
426         if (radius1 <= 0 || radius2 <= 0 || Math.Abs(radius1 - radius2) < 1E-07)
427         {
428             throw new ArgumentException("Incorrect data in mesh parameters");
429         }
430     }
431
432     public static IParameters ReadJson(string jsonPath)
433     {
434         if (!File.Exists(jsonPath))
435         {
436             throw new("File does not exist");
437         }
438
439         using var sr = new StreamReader(jsonPath);
440         return JsonConvert.DeserializeObject<CurveMeshParameters>(sr.ReadToEnd()) ??
441                throw new NullReferenceException("Incorrect mesh parameters!");
442     }
443 }
```

```

public interface IBaseMesh
{
    IReadOnlyList<Point2D> Points { get; }
    IReadOnlyList<IReadOnlyList<int>> Elements { get; }
}

public class RegularMesh : IBaseMesh
{
    private readonly List<Point2D> _points;
    private readonly int[][] _elements;

    public IReadOnlyList<Point2D> Points => _points;
    public IReadOnlyList<IReadOnlyList<int>> Elements => _elements;

    public RegularMesh(IParameters meshParameters, MeshBuilder meshBuilder)
        => (_points, _elements) = meshBuilder.Build(meshParameters);
}
```

Geometry.cs

```
namespace Project;

public class Point2DJsonConverter : JsonConverter
{
    public override bool CanConvert(Type objectType) => typeof(Point2D) == objectType;

    public override object ReadJson(JsonReader reader, Type objectType, object?
    ↪  existingValue,
        JsonSerializer serializer)
    {
        if (reader.TokenType == JsonToken.StartArray)
        {
            var array = JArray.Load(reader);
            if (array.Count == 2) return new Point2D(array[0].Value<double>(),
    ↪  array[1].Value<double>());
            throw new FormatException($"Wrong vector length({array.Count})!");
        }

        if (Point2D.TryParse((string?)reader.Value ?? "", out var point)) return point;
        throw new FormatException($"Can't parse({(string?)reader.Value}) as Point2D!");
    }

    public override void WriteJson(JsonWriter writer, object? value, JsonSerializer
    ↪  serializer)
    {
        value ??= new Point2D();
        var p = (Point2D)value;
        writer.WriteRawValue($"[{p.X}, {p.Y}]");
        // [[0, 0],[0, 0]] // runtime exception if use method WriteRaw()
        // [[0, 0][0, 0]]
    }
}

[JsonConverter(typeof(Point2DJsonConverter))]
public readonly record struct Point2D(double X, double Y)
{
    public static bool TryParse(string line, out Point2D point)
    {
        var words = line.Split(new[] { ' ', ',', '(', ')' },
    ↪  StringSplitOptions.RemoveEmptyEntries);
```

```
        if (words.Length != 3 || !float.TryParse(words[1], out var x) ||
↪   !float.TryParse(words[2], out var y))
        {
            point = default;
            return false;
        }

        point = new(x, y);
        return true;
    }

    public static Point2D operator +(Point2D a, Point2D b) => new(a.X + b.X, a.Y +
↪   b.Y);

    public static Point2D operator -(Point2D a, Point2D b) => new(a.X - b.X, a.Y -
↪   b.Y);

    public static Point2D operator *(Point2D p, double value) => new(p.X * value, p.Y
↪   * value);

    public static Point2D operator /(Point2D p, double value) => new(p.X / value, p.Y
↪   / value);
}

public class IntervalJsonConverter : JsonConverter
{
    public override bool CanConvert(Type objectType) => typeof(Interval) == objectType;

    public override object ReadJson(JsonReader reader, Type objectType, object?
↪   existingValue,
        JsonSerializer serializer)
    {
        if (reader.TokenType == JsonToken.StartArray)
        {
            var array = JArray.Load(reader);
            if (array.Count == 2) return new Interval(array[0].Value<double>(),
↪   array[1].Value<double>());
            throw new FormatException($"Wrong vector length({array.Count})!");
        }

        if (Interval.TryParse((string?)reader.Value ?? "", out var interval)) return
↪   interval;
        throw new FormatException($"Can't parse({(string?)reader.Value}) as
↪   Interval!");
    }

    public override void WriteJson(JsonWriter writer, object? value, JsonSerializer
↪   serializer)
    {
        value ??= new Interval();
        var interval = (Interval)value;
        serializer.Serialize(writer, interval);
    }
}

[JsonConverter(typeof(IntervalJsonConverter))]
public readonly record struct Interval(
    [property: JsonProperty("Left border")]
    double LeftBorder,
    [property: JsonProperty("Right border")]
```

```
    double RightBorder)
{
    [JsonIgnore] public double Center => (LeftBorder + RightBorder) / 2.0;
    [JsonIgnore] public double Length => Math.Abs(RightBorder - LeftBorder);

    public static bool TryParse(string line, out Interval interval)
    {
        var words = line.Split(new[] { ' ', ',', '[', ']' },
↪   StringSplitOptions.RemoveEmptyEntries);
        if (words.Length != 2 || !float.TryParse(words[0], out var x) ||
↪   !float.TryParse(words[1], out var y))
        {
            interval = default;
            return false;
        }

        interval = new(x, y);
        return true;
    }
}

public readonly record struct Rectangle(Point2D LeftBottom, Point2D RightTop)
{
    public Point2D LeftTop { get; } = new(LeftBottom.X, RightTop.Y);
    public Point2D RightBottom { get; } = new(RightTop.X, LeftBottom.Y);
}
```

ArrayHelper.cs

```
namespace Project;

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

        return destination;
    }

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