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

Практическая работа №2
по дисциплине «Цифровые модели и оценивание параметров»

Нелинейные обратные задачи

Студенты	БЕГИЧЕВ АЛЕКСАНДР
	ГРОСС АЛЕКСЕЙ
	ШИШКИН НИКИТА
Группа	ПМ-92
Преподаватели	ВАГИН Д.В.

Новосибирск, 2022

Цель работы

Разработать программу решения нелинейной обратной задачи (задачи электроразведки).

Вариант 9: Два слоя: $h_1 = 20$ м, $h_2 = \infty$. Определить значения удельной электрической проводимости σ_1, σ_2 при известном положении приемников и источника.

Положение источника:

- $A(0, 0, 0)$, $B(100, 0, 0)$.

Положение приемников:

- $M1(200, 0, 0)$, $N1(300, 0, 0)$;
- $M2(500, 0, 0)$, $N2(600, 0, 0)$;
- $M3(1000, 0, 0)$, $N3(1100, 0, 0)$.

Постановка задачи

Задача электроразведки. Два слоя. Источник поля – заземленная электрическая линия с постоянным значение тока $I = 1$ А. Измеряется разность потенциалов в приёмных линиях. Неизвестными являются значения удельной электрической проводимости на каждом слое.

Прямая задача

Краевая задача в цилиндрической системе описывается уравнением:

$$-\operatorname{div}(\sigma \operatorname{grad} V) = J$$

с граничными условиями:

- на левой и верхней границе $\sigma \dfrac{\partial V}{\partial n}\Big|_S = 0$;
- на правой и нижней границе $V\Big|_S = 0$,

где V – распределение поля потенциалов, J – внешний источник тока.

Обратная задача

Целью обратной задачи является нахождение неизвестных значений электрической проводимости σ на каждом слое. Разность потенциалов в линиях приемника по формуле:

$$V_{AB}^{MN} = V_A^M + V_B^M - \left(V_A^N + V_B^N\right).$$

Сборка системы

$$A_{qs} = \sum_{i}^{N^r} (w_i)^2 \frac{\partial \left(\delta\varepsilon_i \left(p^n\right)\right)}{\partial p_q} \frac{\partial \left(\delta\varepsilon_i \left(p^n\right)\right)}{\partial p_s},$$

$$b_q = \sum_{i}^{N^r} (w_i)^2 \delta\varepsilon_i \left(p^n\right) \frac{\partial \left(\delta\varepsilon_i \left(p^n\right)\right)}{\partial p_q},$$

где $\delta\varepsilon_i(p) = (\varepsilon_i(p) - \widetilde{\varepsilon_i})$, N^r – число приемников, ε_i – результаты измерений в приемниках, p^n – начальный вектор значений параметров, w – весовые коэффициенты, которые равны обратным значениям синтетических значений в приемниках.

Регуляризация Тихонова

Если матрица вырождена, то решений обратной задачи будет неединственным. Для преодоления неединственности решения нужно внести соответствующие регуляризирующие добавки. При использовании классической регуляризации Тихонова получаем:

$$\sum_{i}^{N^r} \left(w_i \delta\varepsilon_i(p)\right)^2 + \alpha \sum_{j}^{m} \left(p_j^n - \overline{p}_j\right)^2 \to \min_p .$$

Сумма при α налагает штрафы на отклонения абсолютных значений искомых параметров от некоторых известных $\overline{p}_j$, что приводит к невырожденности матрицы системы:

$$\left(A + \alpha I\right) \Delta p = b - \alpha \left(p^n - \overline{p}\right),$$

где I – это единичная матрица. Значение α может быть подобрано постепенным увеличением от некоторого достаточно малого значения или, если первое невозможно, то “на глаз”.

Тестирование программы

Положение источников:

- $A(0,0,0)$, $B(100,0,0)$;

Положение приемников:

- $M1(200,0,0)$, $N1(300,0,0)$;
- $M2(500,0,0)$, $N2(600,0,0)$;
- $M3(1000,0,0)$, $N3(1100,0,0)$.

Задана область:

- $r \in (0, 5000)$;
- $h_1 = 20$;
- $h_2 = 9980$.

Первый тест

Шумы отсутствуют. σ^* = {0.1, 0.1}, σ^0 = {0.1, 0.09}.

iter	functional	σ_1	σ_2
0	$3.23 \cdot 10^{-2}$	0.1	0.09
1	$2.14 \cdot 10^{-13}$	0.0999999	0.1

Второй тест

Шумы отсутствуют. σ^* = {0.1, 0.2}, σ^0 = {0.01, 0.11}.

iter	functional	σ_1	σ_2
0	2.3	0.01	0.11
1	0.18	0.0320714	0.1640676
2	$3.08 \cdot 10^{-3}$	0.0728316	0.1954338
3	$1.09 \cdot 10^{-6}$	0.0984845	0.2001356
4	$3.05 \cdot 10^{-9}$	0.1000114	0.1999929

Третий тест

Шумы отсутствуют. σ^* = {0.1, 0.2}, σ^0 = {0.01, 0.1}.

iter	functional	σ_1	σ_2
0	3.37	0.01	0.1
1	0.3	0.0309889	0.1550462
2	$7.41 \cdot 10^{-3}$	0.0693632	0.1922174
3	$2.99 \cdot 10^{-6}$	0.0973102	0.200101
4	$1.66 \cdot 10^{-9}$	0.1000138	0.1999945

Четвертый тест

Шумы – +10% на каждом приемнике. σ^* = {0.1, 0.2}, σ^0 = {0.05, 0.05}.

iter	functional	σ_1	σ_2
0	19.27	0.05	0.05
1	2.51	0.0739364	0.0934515
2	0.2	0.0924716	0.1437272
3	$2.96 \cdot 10^{-3}$	0.0942358	0.1759423
4	$1.77 \cdot 10^{-6}$	0.0910312	0.1819549
5	$4.73 \cdot 10^{-9}$	0.0909118	0.1818106

Пятый тест

Поменяли область:

- $h_1 = 200$;
- $h_2 = 9800$.

Шумы отсутствуют. σ^* = {0.1, 0.2}, σ^0 = {0.01, 0.01}.

iter	functional	σ_1	σ_2
0	417.96	0.01	0.01
1	8.4	0.0252998	0.004811
2	0.49	0.049383	0.0062081
3	$8.45 \cdot 10^{-2}$	0.0750818	0.0195379
4	$3.77 \cdot 10^{-2}$	0.0885625	0.042162
5	3.59	0.0838135	0.0424434
6	1.45	0.0895512	0.0669382
7	0.14	0.101083	0.1287385
8	$4.84 \cdot 10^{-3}$	0.1001565	0.1828269
9	$3.52 \cdot 10^{-6}$	0.0999894	0.199522
10	$5.13 \cdot 10^{-9}$	0.1000009	0.2000176

Вывод

Из тестирования видно, что начальное приближение влияет на точность получаемого решения. Появление шумов в приемниках влияет на изменение значения параметров – из четвертого теста видно, что увеличение шумов приводит к уменьшению проводимости.

Если глубина первого слоя h_1 слишком маленькая, то маленькое влияние оказывает σ_1 на получаемое решение, а потому оно находится неточно. С ростом глубины первого слоя h_1 увеличивается влияние σ_1 на функционал, поэтому оно находится точнее.

Листинг

```
// Mesh

MeshGenerator meshGenerator = new(new MeshBuilder(MeshParameters.ReadJson("MeshParameters.jsonc")));
var mesh = meshGenerator.CreateMesh();
mesh.Save("Mesh.json");

// FEM
double Field(double r, double z) => r * r + z;
double Source(double r, double z) => 0.0;

FEMBuilder.FEM fem = FEMBuilder.FEM
    .CreateBuilder()
    .SetMesh(mesh)
    .SetBasis(new LinearBasis())
    .SetSolver(new LOSLU(1000, 1e-13))
    .SetTest(Source);

fem.Solve();
Console.WriteLine($"Residual: {fem.Residual}");

#region Для Python

System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");

// Выводим все значения функции с 1-ого по Z слоя (для отрисовки графика)
using (var sw = new StreamWriter("../../../Python/function.txt"))
{
    for (int i = 0; i < mesh.Elements[0].Nodes[2]; i++)
    {
        double rPoint = mesh.Points[i].R;
        double value = fem.Solution!.Value[i];

        sw.WriteLine($"{string.Format("{0:f14}", rPoint)}\t {string.Format("{0:f14}", value)}");
    }
}

#endregion

// Electro Exploration
ElectroParameters electroParameters = ElectroParameters.ReadJson("ElectroParameters.json");
ElectroExplorationBuilder explorationBuilder = new();
ElectroExplorationBuilder.ElectroExploration electroExploration =
    explorationBuilder.SetParameters(electroParameters).SetMesh(mesh).SetFEM(fem).SetSolver(new Gauss());

double functional = electroExploration.Solve();
Console.WriteLine(
    $"Sigma1: {electroExploration.Sigma[0]}, Sigma2:  {electroExploration.Sigma[1]}, Functional: {functional}");
```

```
namespace problem_2.Source.FEM;

public class FEMBuilder
{
    #region Класс МКЭ

    public class FEM
    {
        private readonly Mesh _mesh;
        private readonly IBasis _basis;
        private Matrix<double>[]? _precalcLocalGR;
        private Matrix<double>[]? _precalcLocalGZ;
        private Matrix<double>[]? _precalcLocalM;
        private readonly Matrix<double> _stiffnessMatrix;
        private readonly Matrix<double> _massMatrix;
        private readonly Vector<double> _localB;
        private readonly SparseMatrix _globalMatrix;
        private readonly Vector<double> _globalVector;
        private readonly IterativeSolver _solver;
        private readonly Integration _gauss;
        private readonly Func<double, double, double> _source;
        private readonly Func<double, double, double>? _field;

        public ImmutableArray<double>? Solution => _solver.Solution;
        public double? Residual { get; private set; }

        public FEM(
            Mesh mesh, IBasis basis, IterativeSolver solver,
            Func<double, double, double> source,
            Func<double, double, double>? field)
        {
            _source = source;
            _field = field;

            _mesh = mesh;
            _basis = basis;
            _solver = solver;

            _stiffnessMatrix = new(_basis.Size);
            _massMatrix = new(_basis.Size);
            _localB = new(_basis.Size);

            PortraitBuilder.Build(_mesh, out int[] ig, out int[] jg);
            _globalMatrix = new(ig.Length - 1, jg.Length)
            {
                Ig = ig,
                Jg = jg
            };
            _globalVector = new(ig.Length - 1);

            _gauss = new(Quadratures.GaussOrder3());
        }

        private void BuildLocalMatrices(int ielem)
        {
            var elem = _mesh.Elements[ielem];

            var bPoint = _mesh.Points[elem.Nodes[0]];
            var ePoint = _mesh.Points[elem.Nodes[_basis.Size - 1]];

```

```
61              double hr = ePoint.R - bPoint.R;
62              double hz = ePoint.Z - bPoint.Z;
63
64              if (_precalcLocalGR is null)
65              {
66                  _precalcLocalGR = new Matrix<double>[] { new(_basis.Size),
    ↪  new(_basis.Size) };
67                  _precalcLocalGZ = new Matrix<double>[] { new(_basis.Size),
    ↪  new(_basis.Size) };
68                  _precalcLocalM = new Matrix<double>[] { new(_basis.Size),
    ↪  new(_basis.Size) };
69
70                  Rectangle rect = new(new(0, 0), new(1, 1));
71
72                  for (int i = 0; i < _basis.Size; i++)
73                  {
74                      for (int j = 0; j <= i; j++)
75                      {
76                          Func<double, double, double> function;
77                          for (int k = 0; k < 2; k++)
78                          {
79                              var i1 = i;
80                              var j1 = j;
81                              var k1 = k;
82                              function = (ksi, etta) =>
83                              {
84                                  Point2D point = new(ksi, etta);
85
86                                  double dphiiR = _basis.DPsi(i1, 0, point);
87                                  double dphijR = _basis.DPsi(j1, 0, point);
88
89                                  return k1 == 0 ? dphiiR * dphijR : dphiiR * dphijR *
    ↪  ksi;
90                              };
91
92                              _precalcLocalGR[k][i, j] = _precalcLocalGR[k][j, i] =
    ↪  _gauss.Integrate2D(function, rect);
93                          }
94
95                          for (int k = 0; k < 2; k++)
96                          {
97                              var k1 = k;
98                              var j1 = j;
99                              var i1 = i;
100                             function = (ksi, etta) =>
101                             {
102                                 Point2D point = new(ksi, etta);
103
104                                 double dphiiZ = _basis.DPsi(i1, 1, point);
105                                 double dphijZ = _basis.DPsi(j1, 1, point);
106
107                                 return k1 == 0 ? dphiiZ * dphijZ : dphiiZ * dphijZ *
    ↪  ksi;
108                             };
109
110                             _precalcLocalGZ[k][i, j] = _precalcLocalGZ[k][j, i] =
    ↪  _gauss.Integrate2D(function, rect);
111                         }
112
113                         for (int k = 0; k < 2; k++)
```

```
                        {
                            var k1 = k;
                            var i1 = i;
                            var j1 = j;
                            function = (ksi, etta) =>
                            {
                                Point2D point = new(ksi, etta);

                                double phiI = _basis.Psi(i1, point);
                                double phiJ = _basis.Psi(j1, point);

                                return k1 == 0 ? phiI * phiJ : phiI * phiJ * ksi;
                            };

                            _precalcLocalM[k][i, j] = _precalcLocalM[k][j, i] =
    ↪ _gauss.Integrate2D(function, rect);
                        }
                    }
                }
            }

            for (int i = 0; i < _basis.Size; i++)
            {
                for (int j = 0; j <= i; j++)
                {
                    _stiffnessMatrix[i, j] = _stiffnessMatrix[j, i] =
                        hz / hr * bPoint.R * _precalcLocalGR![0][i, j] +
                        hz * _precalcLocalGR![1][i, j] +
                        hr / hz * bPoint.R * _precalcLocalGZ![0][i, j] +
                        hr * hr / hz * _precalcLocalGZ![1][i, j];
                }
            }

            for (int i = 0; i < _basis.Size; i++)
            {
                for (int j = 0; j <= i; j++)
                {
                    _massMatrix[i, j] = _massMatrix[j, i] =
                        hr * bPoint.R * hz * _precalcLocalM![0][i, j] +
                        hr * hr * hz * _precalcLocalM![1][i, j];
                }
            }
        }

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

            var elem = _mesh.Elements[ielem];

            double[] f = new double[_basis.Size];

            for (int i = 0; i < _basis.Size; i++)
            {
                var point = _mesh.Points[elem.Nodes[i]];

                f[i] = _source(point.R, point.Z);
            }

            for (int i = 0; i < _basis.Size; i++)
```

```
            {
                for (int j = 0; j < _basis.Size; j++)
                {
                    _localB[i] += _massMatrix[i, j] * f[j];
                }
            }
        }

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

            if (i < j)
            {
                for (int ind = _globalMatrix.Ig[j]; ind < _globalMatrix.Ig[j + 1];
↪   ind++)
                {
                    if (_globalMatrix.Jg[ind] == i)
                    {
                        _globalMatrix.GGu[ind] += value;
                        return;
                    }
                }
            }
            else
            {
                for (int ind = _globalMatrix.Ig[i]; ind < _globalMatrix.Ig[i + 1];
↪   ind++)
                {
                    if (_globalMatrix.Jg[ind] == j)
                    {
                        _globalMatrix.GGl[ind] += value;
                        return;
                    }
                }
            }
        }

        private void AssemblySLAE()
        {
            _globalMatrix.Clear();
            _globalVector.Fill(0.0);

            for (int ielem = 0; ielem < _mesh.Elements.Length; ielem++)
            {
                var elem = _mesh.Elements[ielem];
                double coef = _mesh.AreaProperty[elem.AreaNumber];

                BuildLocalMatrices(ielem);
                BuildLocalVector(ielem);

                for (int i = 0; i < _basis.Size; i++)
                {
                    _globalVector[elem.Nodes[i]] += _localB[i];

                    for (int j = 0; j < _basis.Size; j++)
```

```
                    {
                        AddToGlobal(elem.Nodes[i], elem.Nodes[j], coef *
↪  _stiffnessMatrix[i, j]);
                    }
                }
            }
        }

        private void AddDirichlet()
        {
            foreach (var (node, value) in _mesh.Dirichlet)
            {
                var point = _mesh.Points[node];

                _globalMatrix.Di[node] = 1E+32;

                if (_field is not null)
                {
                    _globalVector[node] = _field(point.R, point.Z) * 1E+32;
                }
                else
                {
                    _globalVector[node] = value * 1E+32;
                }
            }

            if (_field is null)
            {
                _globalVector[0] = 1.0;
            }
        }

        private double Error()
        {
            if (_field is not null)
            {
                double[] exact = new double[_solver.Solution!.Value.Length];

                for (int i = 0; i < _mesh.Points.Length; i++)
                {
                    var point = _mesh.Points[i];

                    exact[i] = _field(point.R, point.Z);
                }

                double exactNorm = exact.Norm();

                for (int i = 0; i < _mesh.Points.Length; i++)
                {
                    exact[i] -= _solver.Solution!.Value[i];
                }

                return exact.Norm() / exactNorm;
            }

            return 0.0;
        }

        public void Solve()
        {
```

```
            AssemblySLAE();

            AddDirichlet();

            _solver.SetSystem(_globalMatrix, _globalVector);
            _solver.Compute();

            Residual = Error();
        }

        private int FindElem(Point2D point)
        {
            for (int i = 0; i < _mesh.Elements.Length; i++)
            {
                var nodes = _mesh.Elements[i].Nodes;

                var leftBottom = _mesh.Points[nodes[0]];
                var rightTop = _mesh.Points[nodes[_basis.Size - 1]];

                if (leftBottom.R <= point.R && point.R <= rightTop.R &&
                    leftBottom.Z <= point.Z && point.Z <= rightTop.Z)
                {
                    return i;
                }
            }

            return -1;
        }

        public void UpdateMesh(double[] newSigma) => _mesh.UpdateProperties(newSigma);

        public double ValueAtPoint(Point2D point)
        {
            double value = 0.0;

            try
            {
                int ielem = FindElem(point);

                if (ielem == -1)
                    throw new ArgumentException(nameof(point), $"Not expected point
↪  value: {point}");

                var nodes = _mesh.Elements[ielem].Nodes;
                var leftBottom = _mesh.Points[nodes[0]];
                var rightTop = _mesh.Points[nodes[_basis.Size - 1]];

                double ksi = (point.R - leftBottom.R) / (rightTop.R - leftBottom.R);
                double eta = (point.Z - leftBottom.Z) / (rightTop.Z - leftBottom.Z);

                for (int i = 0; i < _basis.Size; i++)
                {
                    value += _solver.Solution!.Value[nodes[i]] * _basis.Psi(i,
↪  new(ksi, eta));
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Exception: {ex.Message}");
            }
```

```
348
349             return value;
350         }
351
352         public static FEMBuilder CreateBuilder()
353             => new();
354     }
355
356     #endregion
357
358     #region Содержимое класса FEMBuilder
359
360     private Mesh _mesh = default!;
361     private IBasis _basis = default!;
362     private IterativeSolver _solver = default!;
363     private Func<double, double, double>? _field;
364     private Func<double, double, double> _source = default!;
365
366     public FEMBuilder SetMesh(Mesh mesh)
367     {
368         _mesh = mesh;
369         return this;
370     }
371
372     public FEMBuilder SetBasis(IBasis basis)
373     {
374         _basis = basis;
375         return this;
376     }
377
378     public FEMBuilder SetSolver(IterativeSolver solver)
379     {
380         _solver = solver;
381         return this;
382     }
383
384     public FEMBuilder SetTest(Func<double, double, double> source, Func<double,
    ↪  double, double>? field = null)
385     {
386         _source = source;
387         _field = field;
388         return this;
389     }
390
391     public static implicit operator FEM(FEMBuilder fB)
392         => new(fB._mesh, fB._basis, fB._solver, fB._source, fB._field);
393
394     #endregion
395 }
```

```
1  namespace problem_2.Source.ElectroExploration;
2
3  public class ElectroExplorationBuilder
4  {
5      #region ElectroExploration
6
7      private enum ElectrodType
8      {
9          ElectrodA = -1,
```

```
        ElectrodB = 1
    }

    public class ElectroExploration
    {
        private readonly ElectroParameters _parameters;
        private readonly DirectSolver _solver;
        private readonly Vector<double> _potentials;
        private readonly Vector<double> _currentPotentials;
        private readonly Matrix<double> _potentialsDiffs;
        private readonly Matrix<double> _matrix;
        private readonly Vector<double> _vector;
        private readonly double[] _sigma;
        private readonly Mesh _mesh;
        private readonly FEMBuilder.FEM _fem;
        private readonly double _current;

        private const double DeltaSigma = 1E-2;
        private double _alphaRegulator = 1E-12;

        public ImmutableArray<double> Sigma => _sigma.ToImmutableArray();

        public ElectroExploration(ElectroParameters parameters, Mesh mesh,
↪   FEMBuilder.FEM fem, DirectSolver solver)
        {
            _current = 1.0;

            _parameters = parameters;
            _solver = solver;
            _fem = fem;
            _mesh = mesh;

            _sigma = _parameters.PrimarySigma!;

            _matrix = new(_sigma.Length);
            _vector = new(_sigma.Length);

            _potentials = new(_parameters.PowerReceivers!.Length);
            _currentPotentials = new(_parameters.PowerReceivers.Length);
            _potentialsDiffs = new(_sigma.Length, _parameters.PowerReceivers.Length);
        }

        private double Potential(int ireciever)
        {
            var source = _parameters.PowerSources![0];

            var receiver = _parameters.PowerReceivers![ireciever];

            double rAM = Point2D.Distance(source.A, receiver.M);
            double rBM = Point2D.Distance(source.B, receiver.M);
            double VrAM = (int)ElectrodType.ElectrodA * _current *
↪   _fem.ValueAtPoint(new(rAM, _mesh.Points[0].Z));
            double VrBM = (int)ElectrodType.ElectrodB * _current *
↪   _fem.ValueAtPoint(new(rBM, _mesh.Points[0].Z));

            double rAN = Point2D.Distance(source.A, receiver.N);
            double rBN = Point2D.Distance(source.B, receiver.N);
            double VrAN = (int)ElectrodType.ElectrodA * _current *
↪   _fem.ValueAtPoint(new(rAN, _mesh.Points[0].Z));
            double VrBN = (int)ElectrodType.ElectrodB * _current *
↪   _fem.ValueAtPoint(new(rBN, _mesh.Points[0].Z));
```

```
66
67              return VrAM + VrBM - (VrAN + VrBN);
68          }
69
70          private void CalcDiffs()
71          {
72              for (int i = 0; i < _sigma.Length; i++)
73              {
74                  for (int j = 0; j < _parameters.PowerReceivers!.Length; j++)
75                  {
76                      _sigma[i] += DeltaSigma;
77
78                      _fem.UpdateMesh(_sigma);
79                      _fem.Solve();
80
81                      _sigma[i] -= DeltaSigma;
82
83                      _potentialsDiffs[i, j] = (Potential(j) - _currentPotentials[j]) /
    ↪  DeltaSigma;
84                  }
85              }
86          }
87
88          private void AssemblySystem()
89          {
90              CalcDiffs();
91
92              _matrix.Clear();
93              _vector.Fill(0.0);
94
95              for (int q = 0; q < _sigma.Length; q++)
96              {
97                  for (int s = 0; s < _sigma.Length; s++)
98                  {
99                      for (int i = 0; i < _parameters.PowerReceivers!.Length; i++)
100                     {
101                         double diffQ = _potentialsDiffs[q, i];
102                         double diffS = _potentialsDiffs[s, i];
103                         double w = 1.0 / _potentials[i];
104
105                         _matrix[q, s] += w * w * diffQ * diffS;
106                     }
107                 }
108
109                 for (int i = 0; i < _parameters.PowerReceivers!.Length; i++)
110                 {
111                     double w = 1.0 / _potentials[i];
112
113                     _vector[q] -= w * w * _potentialsDiffs[q, i] *
    ↪  (_currentPotentials[i] - _potentials[i]);
114                 }
115             }
116         }
117
118         private void DirectProblem()
119         {
120             for (int i = 0; i < _parameters.PowerReceivers!.Length; i++)
121             {
122                 _potentials[i] = Potential(i);
123             }
```

```
124
125             // Добавляем шумы
126             for (int i = 0; i < _potentials.Length; i++)
127             {
128                 _potentials[i] += _parameters.Noises![i] * _potentials[i];
129             }
130         }
131
132         private double InverseProblem()
133         {
134             const double eps = 1E-7;
135
136             _fem.UpdateMesh(_sigma);
137             _fem.Solve();
138
139             for (int i = 0; i < _currentPotentials.Length; i++)
140             {
141                 _currentPotentials[i] = Potential(i);
142             }
143
144             double functional = Functional(_currentPotentials.ToArray());
145
146             int iters = 0;
147
148             while (functional >= eps && iters < 500)
149             {
150                 // for report
151                 var sw1 = new StreamWriter("../../../CSV/6.csv", true);
152                 using (sw1)
153                 {
154                     if (iters == 0)
155                     {
156                         sw1.WriteLine($"Iter,Functional,sigma1,sigma2");
157                     }
158
159                     sw1.WriteLine($"{iters},{functional},{_sigma[0]},{_sigma[1]}");
160                 }
161
162
163                 Console.WriteLine($"Iter: {iters},  Functional: {functional}, Sigmas:
    {_sigma[0]}, {_sigma[1]}");
164
165                 iters++;
166
167                 AssemblySystem();
168
169                 _solver.SetMatrix(_matrix);
170                 _solver.SetVector(_vector);
171                 _solver.Compute();
172
173                 Regularization();
174
175                 for (int i = 0; i < _sigma.Length; i++)
176                 {
177                     _sigma[i] += _solver.Solution!.Value[i];
178                 }
179
180                 _fem.UpdateMesh(_sigma);
181                 _fem.Solve();
182
```

```
                for (int i = 0; i < _currentPotentials.Length; i++)
                {
                    _currentPotentials[i] = Potential(i);
                }

                functional = Functional(_currentPotentials.ToArray());
            }

            // for report
            var sw = new StreamWriter("../../../CSV/6.csv", true);
            using (sw)
            {
                if (iters == 0)
                {
                    sw.WriteLine($"Iter,Functional,sigma1,sigma2");
                }

                sw.WriteLine($"{iters},{functional},{_sigma[0]},{_sigma[1]}");
            }

            return functional;
        }

        private double Functional(double[] currentPotentials)
        {
            double functional = 0.0;

            for (int i = 0; i < _parameters.PowerReceivers!.Length; i++)
            {
                double error = 1.0 / _potentials[i] * (_potentials[i] -
↪   currentPotentials[i]);
                functional += error * error;
            }

            return functional;
        }

        private void Regularization()
        {
            double prevAlpha = 0.0;

            while (!_solver.IsSolved())
            {
                for (int i = 0; i < _matrix.Rows; i++)
                {
                    _matrix[i, i] -= prevAlpha;
                    _matrix[i, i] += _alphaRegulator;

                    _vector[i] += prevAlpha * (_potentials[i] -
↪   _currentPotentials[i]);
                    _vector[i] -= _alphaRegulator * (_potentials[i] -
↪   _currentPotentials[i]);

                    prevAlpha = _alphaRegulator;
                    _alphaRegulator *= 10.0;
                }

                _solver.SetMatrix(_matrix);
                _solver.SetVector(_vector);
                _solver.Compute();
```

```
            }
        }

        public double Solve()
        {
            DirectProblem();
            double functional = InverseProblem();
            return functional;
        }
    }

    #endregion

    #region ElectroExplorationBuilder

    private ElectroParameters _parameters = default!;
    private DirectSolver _solver = default!;
    private FEMBuilder.FEM _fem = default!;
    private Mesh _mesh = default!;

    public ElectroExplorationBuilder SetParameters(ElectroParameters parameters)
    {
        _parameters = parameters;
        return this;
    }

    public ElectroExplorationBuilder SetSolver(DirectSolver solver)
    {
        _solver = solver;
        return this;
    }

    public ElectroExplorationBuilder SetMesh(Mesh mesh)
    {
        _mesh = mesh;
        return this;
    }

    public ElectroExplorationBuilder SetFEM(FEMBuilder.FEM fem)
    {
        _fem = fem;
        return this;
    }

    public static implicit operator ElectroExploration(ElectroExplorationBuilder
↪   builder)
        => new(builder._parameters, builder._mesh, builder._fem, builder._solver);

    #endregion
}
```

```
using problem_2.Source.FEM;

namespace problem_2.Source.ElectroExploration;

public readonly record struct PowerSource(Point2D A, Point2D B);

public readonly record struct PowerReceiver(Point2D M, Point2D N);

```

```
public class ElectroParameters
{
    [JsonProperty("Power sources", Required = Required.Always)]
    public PowerSource[]? PowerSources { get; init; }

    [JsonProperty("Power receivers", Required = Required.Always)]
    public PowerReceiver[]? PowerReceivers { get; init; }

    [JsonProperty("Primary sigma", Required = Required.Always)]
    public double[]? PrimarySigma { get; init; }

    [JsonProperty("Noises", Required = Required.Always)]
    public double[]? Noises { get; init; }

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

            using var sr = new StreamReader(jsonPath);
            return JsonConvert.DeserializeObject<ElectroParameters>(sr.ReadToEnd()) ??
                   throw new NullReferenceException("Fill in the parameter data
↪ correctly");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception: {ex.Message}");
            throw;
        }
    }
}
```

```
namespace problem_2.Source.FEM;

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

    public double Psi(int ifunc, Point2D point) =>
        ifunc switch
        {
            0 => (1 - point.R) * (1 - point.Z),
            1 => point.R * (1 - point.Z),
            2 => (1 - point.R) * point.Z,
            3 => point.R * point.Z,
            _ => throw new ArgumentOutOfRangeException(nameof(ifunc), $"Not expected
↪ ifunc value: {ifunc}")
        };

    public double DPsi(int ifunc, int ivar, Point2D point) =>
        ivar switch
        {
            0 => ifunc switch
            {
                0 => point.Z - 1,
```

```
                1 => 1 - point.Z,
                2 => -point.Z,
                3 => point.Z,
                _ => throw new ArgumentOutOfRangeException(nameof(ifunc), $"Not expected ifunc value: {ifunc}")
            },
            1 => ifunc switch
            {
                0 => point.R - 1,
                1 => -point.R,
                2 => 1 - point.R,
                3 => point.R,
                _ => throw new ArgumentOutOfRangeException(nameof(ifunc), $"Not expected ifunc value: {ifunc}")
            },
            _ => throw new ArgumentOutOfRangeException(nameof(ivar), $"Not expected ivar value: {ivar}")
        };
}
```

```
namespace problem_2.Source.FEM;

public readonly record struct DirichletBoundary(int Node, double Value);
```

```
namespace problem_2.Source.FEM;

public static class EnumerableExtensions
{
    public static double Norm<T>(this IEnumerable<T> collection) where T : INumber<T>
    {
        T scalar = T.Zero;

        foreach (var item in collection)
        {
            scalar += item * item;
        }

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

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

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

```
namespace problem_2.Source.FEM;

public class FiniteElement
{
    public ImmutableArray<int> Nodes { get; }
    public int AreaNumber { get; }

    public FiniteElement(int[] nodes, int areaNumber)
    {
        Nodes = nodes.ToImmutableArray();
        AreaNumber = areaNumber;
    }
}
```

```
namespace problem_2.Source.FEM;

public readonly record struct Point2D(double R, double Z)
{
    public override string ToString() => $"R: {R}, Z: {Z}";

    public static double Distance(Point2D a, Point2D b) =>
        Math.Sqrt((b.R - a.R) * (b.R - a.R) + (b.Z - a.Z) * (b.Z - a.Z));
}

public readonly record struct Interval
{
    [JsonProperty("Left border")] public double LeftBorder { get; init; }
    [JsonProperty("Right border")] public double RightBorder { get; init; }
    [JsonIgnore] public double Length => Math.Abs(RightBorder - LeftBorder);

    [JsonConstructor]
    public Interval(double leftBorder, double rightBorder) =>
        (LeftBorder, RightBorder) = (leftBorder, rightBorder);
}

public readonly record struct Rectangle
{
    [JsonProperty("Left bottom")] public Point2D LeftBottom { get; init; }
    [JsonProperty("Right top")] public Point2D RightTop { get; init; }

    [JsonIgnore] public double Square => (RightTop.R - LeftBottom.R) * (RightTop.Z -
↪   LeftBottom.Z);

    [JsonConstructor]
    public Rectangle(Point2D leftBottom, Point2D rightTop) =>
        (LeftBottom, RightTop) = (leftBottom, rightTop);
}

public readonly record struct Layer(
    [property: JsonProperty("Height")] double Height,
    [property: JsonProperty("Sigma")] double Sigma);
```

```
using problem_2.Source.FEM;

namespace problem_2.Interfaces;

public interface IBasis
{
```

```
    int Size { get; }

    double Psi(int ifunc, Point2D point);

    double DPsi(int ifunc, int ivar, Point2D point);
}
```

```
using problem_2.Source.FEM;

namespace problem_2.Interfaces;

public interface IMeshBuilder
{
    IEnumerable<Point2D> CreatePoints();
    IEnumerable<FiniteElement> CreateElements();
    IEnumerable<double> CreateMaterials();
    IEnumerable<DirichletBoundary> CreateDirichlet();
}
```

```
namespace problem_2.Source.FEM;

public class Integration
{
    private readonly IEnumerable<QuadratureNode<double>> _quadratures;

    public Integration(IEnumerable<QuadratureNode<double>> quadratures) =>
        _quadratures = quadratures;

    public double Integrate1D(Func<double, double> f, Interval interval)
    {
        double a = interval.LeftBorder;
        double b = interval.RightBorder;
        double h = interval.Length;

        double sum = 0.0;

        foreach (var quad in _quadratures)
        {
            double qi = quad.Weight;
            double pi = (a + b + quad.Node * h) / 2.0;

            sum += qi * f(pi);
        }

        return sum * h / 2.0;
    }

    public double Integrate2D(Func<double, double, double> f, Rectangle rectangle)
    {
        var leftBottom = rectangle.LeftBottom;
        var rightTop = rectangle.RightTop;

        double hr = rightTop.R - leftBottom.R;
        double hz = rightTop.Z - leftBottom.Z;

        double sum = 0.0;

        foreach (var iquad in _quadratures)
```

```
        {
            double qi = iquad.Weight;
            double pi = (leftBottom.R + rightTop.R + iquad.Node * hr) / 2.0;

            foreach (var jquad in _quadratures)
            {
                double qj = jquad.Weight;
                double pj = (leftBottom.Z + rightTop.Z + jquad.Node * hz) / 2.0;

                sum += qi * qj * f(pi, pj);
            }
        }

        return sum * hr * hz / 4.0;
    }
}
```

```
namespace problem_2.Source.FEM;

public class Matrix<T> where T : INumber<T>
{
    private T[][] _storage;
    public int Rows { get; }
    public int Columns { get; }

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

    public Matrix(int size)
    {
        Rows = size;
        Columns = size;
        _storage = new T[size].Select(_ => new T[size]).ToArray();
    }

    public Matrix(int rows, int columns)
    {
        Rows = rows;
        Columns = columns;
        _storage = new T[rows].Select(_ => new T[columns]).ToArray();
    }

    public static Matrix<T> Copy(Matrix<T> otherMatrix)
    {
        Matrix<T> newMatrix = new(otherMatrix.Rows, otherMatrix.Columns);

        for (int i = 0; i < otherMatrix.Rows; i++)
        {
            for (int j = 0; j < otherMatrix.Columns; j++)
            {
                newMatrix[i, j] = otherMatrix[i, j];
            }
        }

        return newMatrix;
    }
```

```
43
44      public static IEnumerable<T> operator *(Matrix<T> matrix, T[] vector)
45      {
46          if (matrix.Columns != vector.Length)
47          {
48              throw new Exception("Numbers of columns not equal to size of vector");
49          }
50
51          var product = new Vector<T>(vector.Length);
52
53          for (int i = 0; i < matrix.Rows; i++)
54          {
55              for (int j = 0; j < matrix.Columns; j++)
56              {
57                  product[i] += matrix[i, j] * vector[j];
58              }
59          }
60
61          return product;
62      }
63
64      public void Clear()
65          => _storage = _storage.Select(row => row.Select(_ =>
    ↪ T.Zero).ToArray()).ToArray();
66  }
67
68  public class SparseMatrix
69  {
70      public int[] Ig { get; init; }
71      public int[] Jg { get; init; }
72      public double[] Di { get; }
73      public double[] GGl { get; }
74      public double[] GGu { get; }
75      public int Size { get; }
76
77      public SparseMatrix(int size, int sizeOffDiag)
78      {
79          Size = size;
80          Ig = new int[size + 1];
81          Jg = new int[sizeOffDiag];
82          GGl = new double[sizeOffDiag];
83          GGu = new double[sizeOffDiag];
84          Di = new double[size];
85      }
86
87      public static Vector<double> operator *(SparseMatrix matrix, Vector<double>
    ↪ vector)
88      {
89          Vector<double> product = new(vector.Length);
90
91          for (int i = 0; i < vector.Length; i++)
92          {
93              product[i] = matrix.Di[i] * vector[i];
94
95              for (int j = matrix.Ig[i]; j < matrix.Ig[i + 1]; j++)
96              {
97                  product[i] += matrix.GGl[j] * vector[matrix.Jg[j]];
98                  product[matrix.Jg[j]] += matrix.GGu[j] * vector[i];
99              }
100         }
```

```
        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]] = GGl[j];
                a[Jg[j], i] = GGu[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("0.0000") + "\t\t");
            }

            sw.WriteLine();
        }
    }

    public void Clear()
    {
        Di.Fill(0.0);
        GGl.Fill(0.0);
        GGu.Fill(0.0);
    }
}
```

```
namespace problem_2.Source.FEM;

public class Mesh
{
    public ImmutableArray<Point2D> Points { get; }
    public ImmutableArray<FiniteElement> Elements { get; }

    [JsonIgnore] private double[] _areaProperty;
    [JsonIgnore] public ImmutableArray<double> AreaProperty =>
↪   _areaProperty.ToImmutableArray();
    [JsonIgnore] public ImmutableArray<DirichletBoundary> Dirichlet { get; }

    public Mesh(
        IEnumerable<Point2D> points,
        IEnumerable<FiniteElement> elements,
        IEnumerable<double> properties,
        IEnumerable<DirichletBoundary> dirichlet
    )
    {
        Points = points.ToImmutableArray();
```

```
        Elements = elements.ToImmutableArray();
        _areaProperty = properties.ToArray();
        Dirichlet = dirichlet.ToImmutableArray();
    }

    public void UpdateProperties(double[] newProperties)
        => _areaProperty = newProperties;


    public void Save(string path)
    {
        using var sw = new StreamWriter(path);
        sw.Write(JsonConvert.SerializeObject(this));
    }
}
```

```
namespace problem_2.Source.FEM;

public class MeshBuilder : IMeshBuilder
{
    private readonly MeshParameters _params;
    private Point2D[] _points = default!;
    private FiniteElement[] _elements = default!;
    private double[] _materials = default!;
    private DirichletBoundary[] _dirichlet = default!;

    public MeshBuilder(MeshParameters parameters) => _params = parameters;

    public IEnumerable<Point2D> CreatePoints()
    {
        double[] pointsR = new double[_params.SplitsR + 1];
        double[] pointsZ = new double[_params.SplitsZ.Sum() + 1];

        _points = new Point2D[pointsR.Length * pointsZ.Length];

        // Точки по оси R
        double rPoint = _params.IntervalR.LeftBorder;

        double hr = _params.KR == 1.0
            ? (_params.IntervalR.Length) / _params.SplitsR
            : (_params.IntervalR.Length) * (1.0 - _params.KR) / (1.0 -
↪   Math.Pow(_params.KR, _params.SplitsR));

        for (int i = 0; i < _params.SplitsR + 1; i++)
        {
            pointsR[i] = rPoint;
            rPoint += hr;
            hr *= _params.KR;
        }

        // Точки по оси Z
        double zPoint = 0.0;

        for (int ilayer = 0, ipoint = 0; ilayer < _params.SplitsZ.Count; ilayer++)
        {
            var layer = _params.Layers[ilayer];
            var splitsZ = _params.SplitsZ[ilayer];
            var kz = _params.KZ[ilayer];

```

```
            double hz = kz == 1.0
                ? layer.Height / splitsZ
                : (layer.Height) * (1.0 - kz) / (1.0 - Math.Pow(kz, splitsZ));

            for (int i = 0; i < splitsZ + 1; i++)
            {
                pointsZ[ipoint++] = zPoint;
                zPoint += hz;
                hz *= kz;
            }

            zPoint = layer.Height;
            ipoint--;
        }

        for (int i = 0, ipoint = 0; i < pointsZ.Length; i++)
        {
            for (int j = 0; j < pointsR.Length; j++)
            {
                _points[ipoint++] = new Point2D(pointsR[j], pointsZ[i]);
            }
        }

        return _points;
    }

    public IEnumerable<FiniteElement> CreateElements()
    {
        _elements = new FiniteElement[_params.SplitsR * _params.SplitsZ.Sum()];

        int[] nodes = new int[4];

        int layerStartIdx = 0;

        for (int ilayer = 0, ielem = 0; ilayer < _params.Layers.Count; ilayer++)
        {
            for (int i = 0; i < _params.SplitsZ[ilayer]; i++)
            {
                for (int j = 0; j < _params.SplitsR; j++)
                {
                    nodes[0] = ilayer * layerStartIdx + j + (_params.SplitsR + 1) * i;
                    nodes[1] = ilayer * layerStartIdx + j + (_params.SplitsR + 1) * i
↪   + 1;
                    nodes[2] = ilayer * layerStartIdx + j + (_params.SplitsR + 1) * i
↪   + _params.SplitsR + 1;
                    nodes[3] = ilayer * layerStartIdx + j + (_params.SplitsR + 1) * i
↪   + _params.SplitsR + 2;

                    _elements[ielem++] = new(nodes, ilayer);
                }
            }

            layerStartIdx += _params.SplitsZ[ilayer] * (_params.SplitsR + 1);
        }

        return _elements;
    }

    public IEnumerable<double> CreateMaterials()
    {
```

```
        _materials = new double[_params.Layers.Count];

        for (int i = 0; i < _materials.Length; i++)
        {
            _materials[i] = _params.Layers[i].Sigma;
        }

        return _materials;
    }

    public IEnumerable<DirichletBoundary> CreateDirichlet()
    {
        HashSet<int> dirichletNodes = new();

        if (_params.TopBorder == 1)
        {
            int startNode = (_params.SplitsR + 1) * _params.SplitsZ.Sum();

            for (int i = 0; i < _params.SplitsR + 1; i++)
            {
                dirichletNodes.Add(startNode + i);
            }
        }

        if (_params.BottomBorder == 1)
        {
            for (int i = 0; i < _params.SplitsR + 1; i++)
            {
                dirichletNodes.Add(i);
            }
        }

        if (_params.LeftBorder == 1)
        {
            for (int i = 0; i < _params.SplitsZ.Sum() + 1; i++)
            {
                dirichletNodes.Add(i * _params.SplitsR + i);
            }
        }

        if (_params.RightBorder == 1)
        {
            for (int i = 0; i < _params.SplitsZ.Sum() + 1; i++)
            {
                dirichletNodes.Add(_params.SplitsR + i * (_params.SplitsR + 1));
            }
        }

        var array = dirichletNodes.OrderBy(x => x).ToArray();

        _dirichlet = new DirichletBoundary[dirichletNodes.Count];

        for (int i = 0; i < _dirichlet.Length; i++)
        {
            _dirichlet[i] = new(array[i], 0.0);
        }

        return _dirichlet;
    }
}
```

```
namespace problem_2.Source.FEM;

public class MeshGenerator
{
    private readonly IMeshBuilder _builder;

    public MeshGenerator(IMeshBuilder builder) => _builder = builder;

    public Mesh CreateMesh() => new(
        _builder.CreatePoints(),
        _builder.CreateElements(),
        _builder.CreateMaterials(),
        _builder.CreateDirichlet()
    );
}
```

```
namespace problem_2.Source.FEM;

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

        var meshParameters = (MeshParameters)value;

        writer.WriteStartObject();
        writer.WritePropertyName("Interval R");
        serializer.Serialize(writer, meshParameters.IntervalR);

        writer.WritePropertyName("Splits R");
        writer.WriteValue(meshParameters.SplitsR);

        writer.WritePropertyName("Coefficient R");
        writer.WriteValue(meshParameters.KR);

        writer.WriteWhitespace("\n");

        writer.WritePropertyName("Layers");
        serializer.Serialize(writer, meshParameters.Layers);

        writer.WriteWhitespace("\n");

        writer.WriteComment("Разбиения для каждого слоя");
        writer.WritePropertyName("Splits Z");
        serializer.Serialize(writer, meshParameters.SplitsZ);

        writer.WriteWhitespace("\n");

        writer.WriteComment("Коэффициенты разрядки для каждого слоя");
        writer.WritePropertyName("Coefficients Z");
        serializer.Serialize(writer, meshParameters.KZ);
```

```
41
42          writer.WriteWhitespace("\n");
43
44          writer.WriteComment("Граница и тип краевого на ней");
45          writer.WritePropertyName("Left border");
46          writer.WriteValue(meshParameters.LeftBorder);
47          writer.WritePropertyName("Right border");
48          writer.WriteValue(meshParameters.RightBorder);
49          writer.WritePropertyName("Top border");
50          writer.WriteValue(meshParameters.TopBorder);
51          writer.WritePropertyName("Bottom border");
52          writer.WriteValue(meshParameters.BottomBorder);
53      }
54
55      public override object? ReadJson(JsonReader reader, Type objectType, object?
    ↪ existingValue,
56          JsonSerializer serializer)
57      {
58          if (reader.TokenType is JsonToken.Null or not JsonToken.StartObject) return
    ↪ null;
59
60          List<Layer> layers = new();
61          List<int> splitsZ = new();
62          List<double> kz = new();
63
64          var data = JObject.Load(reader);
65
66          // Интервал по R и его разбиение
67          var token = data["Interval R"];
68          var intervalR = serializer.Deserialize<Interval>(token!.CreateReader());
69
70          token = data["Splits R"];
71          var splitsR = Convert.ToInt32(token);
72
73          token = data["Coefficient R"];
74          var kr = Convert.ToDouble(token);
75
76          // Слои по Z и их разбиение
77          token = data["Layers"];
78
79          foreach (var child in token!)
80          {
81              layers.Add(serializer.Deserialize<Layer>(child.CreateReader()));
82          }
83
84          token = data["Splits Z"];
85
86          foreach (var child in token!)
87          {
88              splitsZ.Add(serializer.Deserialize<int>(child.CreateReader()));
89          }
90
91          token = data["Coefficients Z"];
92
93          foreach (var child in token!)
94          {
95              kz.Add(serializer.Deserialize<double>(child.CreateReader()));
96          }
97
98          // Границы и типы краевых на них
```

```
        var leftBorder = Convert.ToByte(data["Left border"]);
        var rightBorder = Convert.ToByte(data["Right border"]);
        var bottomBorder = Convert.ToByte(data["Bottom border"]);
        var topBorder = Convert.ToByte(data["Top border"]);

        return new MeshParameters(intervalR, splitsR, kr, layers, splitsZ, kz,
↪   leftBorder, rightBorder, bottomBorder,
            topBorder);
    }

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

[JsonConverter(typeof(MeshParametersJsonConverter))]
public class MeshParameters
{
    public Interval IntervalR { get; }
    public int SplitsR { get; }
    public double KR { get; }
    public ImmutableList<Layer> Layers { get; }
    public ImmutableList<int> SplitsZ { get; }
    public ImmutableList<double> KZ { get; }

    public byte LeftBorder { get; }
    public byte RightBorder { get; }
    public byte BottomBorder { get; }
    public byte TopBorder { get; }

    public MeshParameters(
        Interval intervalR, int splitsR, double kr,
        List<Layer> layers, List<int> splitsZ, List<double> kz,
        byte leftBorder, byte rightBorder,
        byte bottomBorder, byte topBorder)
    {
        IntervalR = intervalR;
        SplitsR = splitsR;
        KR = kr;
        Layers = layers.ToImmutableList();
        SplitsZ = splitsZ.ToImmutableList();
        KZ = kz.ToImmutableList();
        LeftBorder = leftBorder;
        RightBorder = rightBorder;
        BottomBorder = bottomBorder;
        TopBorder = topBorder;
    }

    public static MeshParameters ReadJson(string jsonPath)
    {
        try
        {
            if (!File.Exists(jsonPath))
            {
                throw new Exception("File doesn't exist");
            }

            using var sr = new StreamReader(jsonPath);
            return JsonConvert.DeserializeObject<MeshParameters>(sr.ReadToEnd())
                   ?? throw new NullReferenceException("Fill in the parameter data
↪   correctly");
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception: {ex.Message}");
            throw;
        }
    }
}
```

```
namespace problem_2.Source.FEM;

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

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

        int localSize = mesh.Elements[0].Nodes.Length;

        foreach (var element in mesh.Elements)
        {
            for (int i = 0; i < localSize - 1; i++)
            {
                int nodeToInsert = element.Nodes[i];

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

```
            }
        }
    }
}
```

```
namespace problem_2.Source.FEM;

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

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

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

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

public static class Quadratures
{
    public static IEnumerable<QuadratureNode<double>> GaussOrder3()
    {
        const int n = 3;

        double[] points = { 0.0, 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 QuadratureNode<double>(points[i], weights[i]);
        }
    }

    public static IEnumerable<QuadratureNode<double>> GaussOrder4()
    {
        const int n = 4;

        double[] points =
        {
            Math.Sqrt(3.0 / 7.0 - 2.0 / 7.0 * Math.Sqrt(6.0 / 5.0)),
            -Math.Sqrt(3.0 / 7.0 - 2.0 / 7.0 * Math.Sqrt(6.0 / 5.0)),
            Math.Sqrt(3.0 / 7.0 + 2.0 / 7.0 * Math.Sqrt(6.0 / 5.0)),
            -Math.Sqrt(3.0 / 7.0 + 2.0 / 7.0 * Math.Sqrt(6.0 / 5.0))
        };

        double[] weights =
```

```
        {
            18.0 + Math.Sqrt(30.0) / 36.0,
            18.0 + Math.Sqrt(30.0) / 36.0,
            18.0 - Math.Sqrt(30.0) / 36.0,
            18.0 - Math.Sqrt(30.0) / 36.0,
        };

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

    public static IEnumerable<QuadratureNode<double>> GaussOrder5()
    {
        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]);
        }
    }
}
```

```
namespace problem_2.Source.FEM;

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 SetSystem(SparseMatrix matrix, Vector<double> vector)
```

```
        => (_matrix, _vector) = (matrix, vector);

    public abstract void Compute();

    protected Vector<double> Direct(Vector<double> vector, double[] gglnew, double[]
↪   dinew)
    {
        Vector<double> y = new(vector.Length);
        Vector<double>.Copy(vector, y);

        double sum = 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++)
                sum += gglnew[k] * y[_matrix.Jg[k]];

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

        return y;
    }

    protected Vector<double> Reverse(Vector<double> vector, double[] ggunew)
    {
        Vector<double> result = new(vector.Length);
        Vector<double>.Copy(vector, result);

        for (int i = _matrix.Size - 1; i >= 0; i--)
        {
            int i0 = _matrix.Ig[i];
            int i1 = _matrix.Ig[i + 1];

            for (int k = i0; k < i1; k++)
                result[_matrix.Jg[k]] -= ggunew[k] * result[i];
        }

        return result;
    }

    protected void LU(double[] gglnew, double[] ggunew, double[] dinew)
    {
        double suml = 0.0;
        double sumu = 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 += gglnew[ik] * ggunew[kj];
                        sumu += ggunew[ik] * gglnew[kj];
                        ik++;
                        kj++;
                    }
                    else if (_matrix.Jg[ik] > _matrix.Jg[kj])
                    {
                        kj++;
                    }
                    else
                    {
                        ik++;
                    }
                }

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

            dinew[i] -= sumdi;
            sumdi = 0.0;
        }
    }
}

public abstract class DirectSolver
{
    protected Vector<double>? _solution;
    protected Vector<double> _vector = default!;
    protected Matrix<double> _matrix = default!;

    public ImmutableArray<double>? Solution => _solution?.ToImmutableArray();

    public void SetVector(Vector<double> vector)
        => _vector = Vector<double>.Copy(vector);

    public void SetMatrix(Matrix<double> matrix)
        => _matrix = Matrix<double>.Copy(matrix);

    protected DirectSolver(Matrix<double> matrix, Vector<double> vector)
        => (_matrix, _vector) = (Matrix<double>.Copy(matrix),
↪   Vector<double>.Copy(vector));

    protected DirectSolver()
    {
    }

    public abstract void Compute();

    public bool IsSolved() => !(Solution is null);
```

```
}

public class Gauss : DirectSolver
{
    public Gauss(Matrix<double> matrix, Vector<double> vector) : base(matrix, vector)
    {
    }

    public Gauss()
    {
    }

    public override void Compute()
    {
        _solution = null;

        try
        {
            ArgumentNullException.ThrowIfNull(_matrix, $"{nameof(_matrix)} cannot be null, set the Matrix");
            ArgumentNullException.ThrowIfNull(_vector, $"{nameof(_vector)} cannot be null, set the Vector");

            if (_matrix.Rows != _matrix.Columns)
            {
                throw new NotSupportedException("The Gaussian method will not be able to solve this system");
            }

            double eps = 1E-15;

            for (int k = 0; k < _matrix.Rows; k++)
            {
                var max = Math.Abs(_matrix[k, k]);
                int index = k;

                for (int i = k + 1; i < _matrix.Rows; i++)
                {
                    if (Math.Abs(_matrix[i, k]) > max)
                    {
                        max = Math.Abs(_matrix[i, k]);
                        index = i;
                    }
                }

                for (int j = 0; j < _matrix.Rows; j++)
                {
                    (_matrix[k, j], _matrix[index, j]) =
                        (_matrix[index, j], _matrix[k, j]);
                }

                (_vector[k], _vector[index]) = (_vector[index], _vector[k]);

                for (int i = k; i < _matrix.Rows; i++)
                {
                    double temp = _matrix[i, k];

                    if (Math.Abs(temp) < eps)
                    {
                        throw new Exception("Zero element of the column");
```

```
                    }

                    for (int j = 0; j < _matrix.Rows; j++)
                    {
                        _matrix[i, j] /= temp;
                    }

                    _vector[i] /= temp;

                    if (i != k)
                    {
                        for (int j = 0; j < _matrix.Rows; j++)
                        {
                            _matrix[i, j] -= _matrix[k, j];
                        }

                        _vector[i] -= _vector[k];
                    }
                }
            }

            _solution = new(_vector.Length);

            for (int k = _matrix.Rows - 1; k >= 0; k--)
            {
                _solution![k] = _vector[k];

                for (int i = 0; i < k; i++)
                {
                    _vector[i] -= _matrix[i, k] * _solution[k];
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}

public class LOSLU : IterativeSolver
{
    public LOSLU(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");

            _solution = new(_vector.Length);

            double[] gglnew = new double[_matrix.GGl.Length];
            double[] ggunew = new double[_matrix.GGu.Length];
            double[] dinew = new double[_matrix.Di.Length];
```

```
252
253             _matrix.GGl.Copy(gglnew);
254             _matrix.GGu.Copy(ggunew);
255             _matrix.Di.Copy(dinew);
256
257             Stopwatch sw = Stopwatch.StartNew();
258
259             LU(gglnew, ggunew, dinew);
260
261             var r = Direct(_vector - (_matrix * _solution), gglnew, dinew);
262             var z = Reverse(r, ggunew);
263             var p = Direct(_matrix * z, gglnew, dinew);
264
265             var squareNorm = r * r;
266
267             for (int iter = 0; iter < MaxIters && squareNorm > Eps; iter++)
268             {
269                 var alpha = p * r / (p * p);
270                 squareNorm = (r * r) - (alpha * alpha * (p * p));
271                 _solution += alpha * z;
272                 r -= alpha * p;
273
274                 var tmp = Direct(_matrix * Reverse(r, ggunew), gglnew, dinew);
275
276                 var beta = -(p * tmp) / (p * p);
277                 z = Reverse(r, ggunew) + (beta * z);
278                 p = tmp + (beta * p);
279             }
280
281             sw.Stop();
282
283             _runningTime = sw.Elapsed;
284         }
285         catch (Exception ex)
286         {
287             Console.WriteLine($"Exception: {ex.Message}");
288         }
289     }
290 }
```

```
1  namespace problem_2.Source.FEM;
2
3  public class Vector<T> : IEnumerable<T> where T : INumber<T>
4  {
5      private readonly T[] _storage;
6      public int Length { get; }
7
8      public T this[int idx]
9      {
10         get => _storage[idx];
11         set => _storage[idx] = value;
12     }
13
14     public Vector(int length)
15         => (Length, _storage) = (length, new T[length]);
16
17     public static T operator *(Vector<T> a, Vector<T> b)
18     {
19         T result = T.Zero;
```

```
20
21          for (int i = 0; i < a.Length; i++)
22          {
23              result += a[i] * b[i];
24          }
25
26          return result;
27      }
28
29      public static Vector<T> operator *(double constant, Vector<T> vector)
30      {
31          Vector<T> result = new(vector.Length);
32
33          for (int i = 0; i < vector.Length; i++)
34          {
35              result[i] = vector[i] * T.Create(constant);
36          }
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;
```

```
    }

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

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

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