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

Лабораторная работа №1
по дисциплине «Цифровые модели и оценивание параметров»

Линейные обратные задачи

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

Новосибирск, 2022

Цель работы

Разработать программу решения линейной обратной задачи (задачи электроразведки).

Вариант: Однородное полупространство. Определить значение силы тока I в источнике при известном положении приемников и источников.

Положение приемников:

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

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.

Постановка задачи

Задача электроразведки. Однородное полупространство. Источники поля – заземленные электрические линии с известным значением удельной электрической проводимости σ.

Используемые формулы

$$V_{AB}^{MN} = \frac{I}{2\pi\sigma}\left(\left(\frac{1}{r_B^M} - \frac{1}{r_A^M}\right) - \left(\frac{1}{r_B^N} - \frac{1}{r_A^N}\right)\right)$$

– разность потенциалов в приемной линии MN,

$$\frac{\partial V_i}{\partial I} = \frac{1}{2\pi\sigma}\left(\left(\frac{1}{r_{B_i}^{M_i}} - \frac{1}{r_{A_i}^{M_i}}\right) - \left(\frac{1}{r_{B_i}^{N_i}} - \frac{1}{r_{A_i}^{N_i}}\right)\right)$$

– производная по искомому параметру.

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

Первый тест

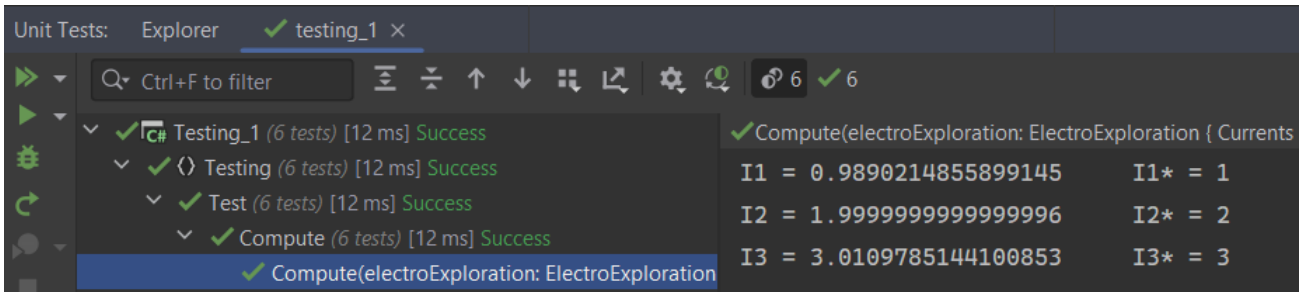
Приемники находятся на одной линии. $I_i^* = \{1.0, 2.0, 3.0\}$, $I_i^0 = \{0.1, 0.2, 0.3\}$

Положение приемников:

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

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.



Второй тест

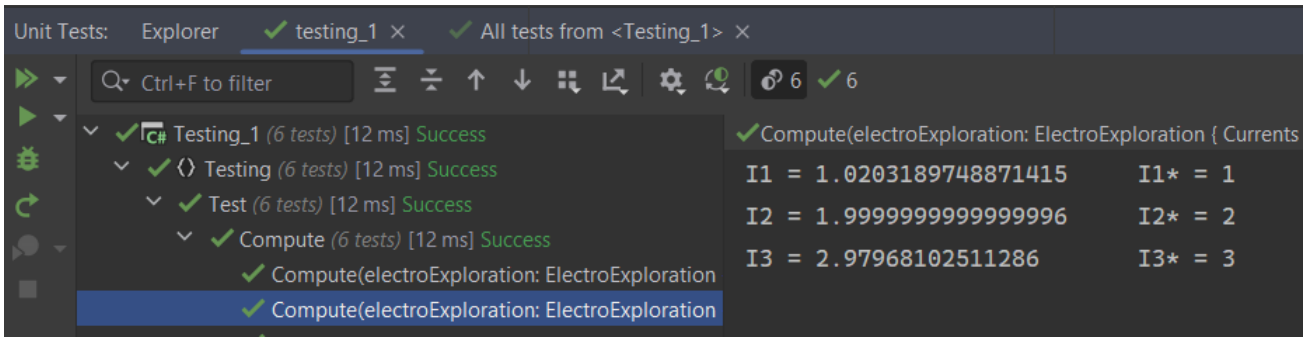
Приемники находятся на одной линии. $I_i^* = \{1.0, 2.0, 3.0\}$, $I_i^0 = \{0.01, 0.02, 0.03\}$

Положение приемников:

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

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.



Третий тест

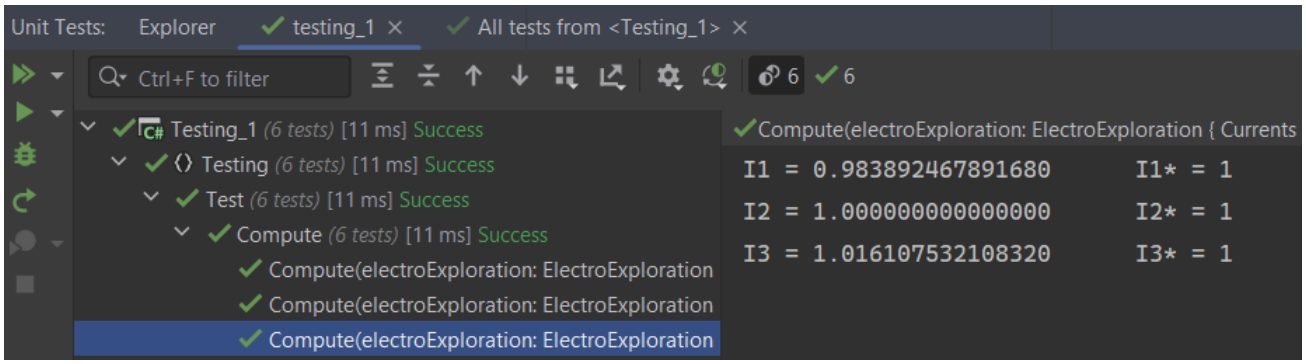
Приемники находятся на одной линии. $I_i^* = \{1.0, 1.0, 1.0\}$, $I_i^0 = \{0.1, 0.2, 0.3\}$

Положение приемников:

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

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.



Четвертый тест

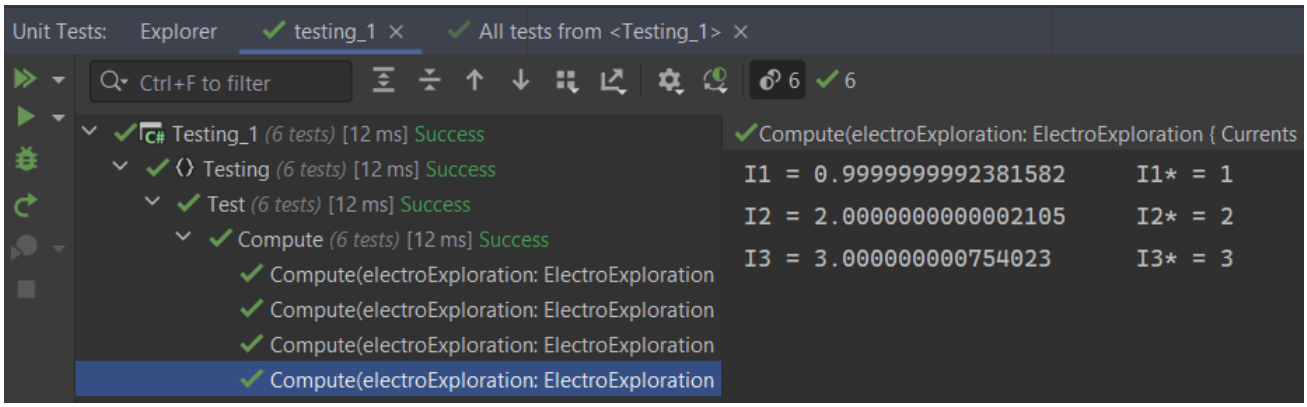
Приемники находятся на разных линиях. $I_i^* = \{1.0, 2.0, 3.0\}$, $I_i^0 = \{0.1, 0.2, 0.3\}$

Положение приемников:

- $M1(200, 1, 0)$, $N1(300, 1, 0)$;
- $M2(500, 0, 0)$, $N2(600, 0, 0)$;
- $M3(1000, 2, 0)$, $N3(1100, 2, 0)$.

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.



Пятый тест

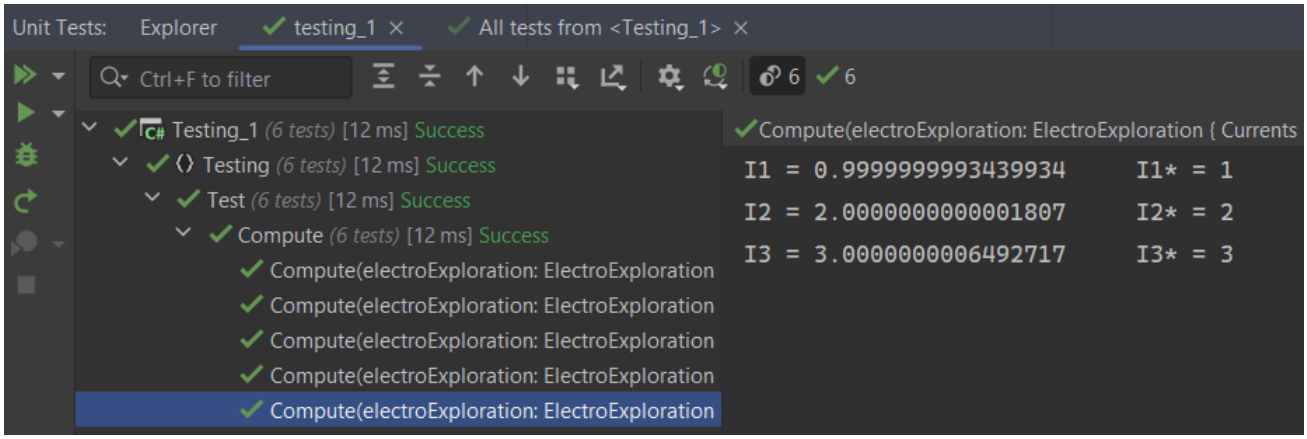
Приемники находятся на разных линиях. $I_i^* = \{1.0, 2.0, 3.0\}$, $I_i^0 = \{0.01, 0.02, 0.03\}$

Положение приемников:

- $M1(200, 1, 0)$, $N1(300, 1, 0)$;
- $M2(500, 0, 0)$, $N2(600, 0, 0)$;
- $M3(1000, 2, 0)$, $N3(1100, 2, 0)$.

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.



Шестой тест

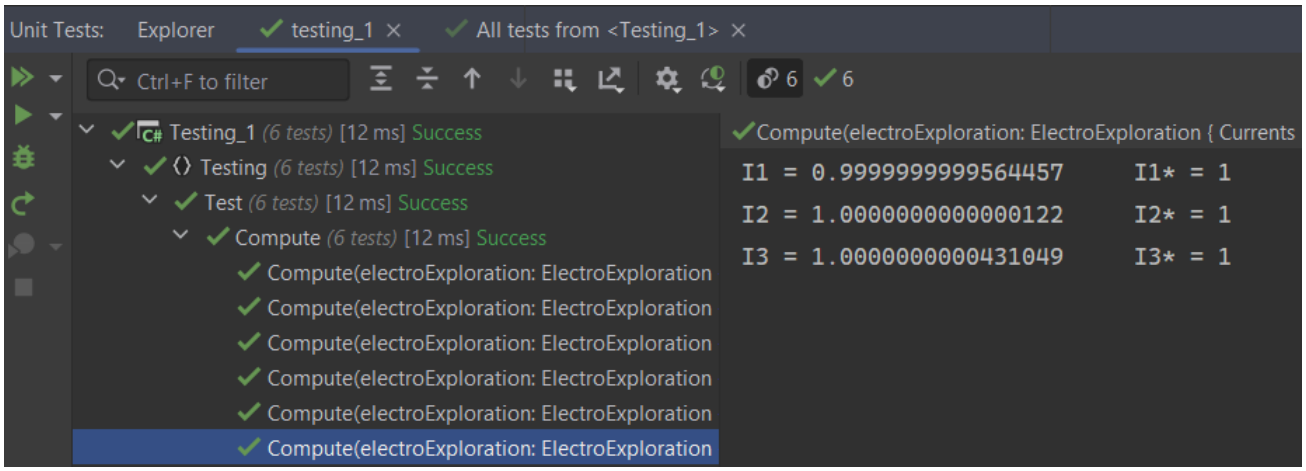
Приемники находятся на разных линиях. $I_i^* = \{1.0, 1.0, 1.0\}$, $I_i^0 = \{0.1, 0.2, 0.3\}$

Положение приемников:

- $M1(200, 1, 0)$, $N1(300, 1, 0)$;
- $M2(500, 0, 0)$, $N2(600, 0, 0)$;
- $M3(1000, 2, 0)$, $N3(1100, 2, 0)$.

Положение источников:

- $A1(0, -500, 0)$, $B1(100, -500, 0)$;
- $A2(0, 0, 0)$, $B2(100, 0, 0)$;
- $A3(0, 500, 0)$, $B3(100, 500, 0)$.



Вывод

Из результатов тестирования программы видно, что точность решения линейной обратной задачи зависит от положения приемников и начального приближения. Также в случае, если приемники находятся на одной линии, то после сборки системы матрица оказывается вырожденной, из-за чего приходится использовать регуляризацию Тихонова для устранения вырожденности матрицы.

Листинг

Program.cs

```
using problem_1;

ElectroExploration electroExploration = ElectroExploration.CreateBuilder()
    .SetParameters(Parameters.ReadJson("parameters.json")).SetSolver(new Gauss());

electroExploration.Compute();
```

Parameters.cs

```
namespace problem_1;

public readonly record struct Point3D(double X, double Y, double Z)
{
    public static double Distance(Point3D a, Point3D b) =>
        Math.Sqrt((b.X - a.X) * (b.X - a.X) + (b.Y - a.Y) * (b.Y - a.Y) + (b.Z - a.Z)
↪   * (b.Z - a.Z));
}
```

```

public readonly record struct PowerSource(Point3D A, Point3D B, [property: JsonProperty("Real current")]
    double RealCurrent, [property: JsonProperty("Primary current")]
    double PrimaryCurrent);

public readonly record struct PowerReceiver(Point3D M, Point3D N);

public class Parameters
{
    [JsonProperty("Power sources", Required = Required.Always)]
    public PowerSource[] PowerSources { get; init; } = Array.Empty<PowerSource>();

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

    [JsonRequired] public double Sigma { get; init; }

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

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

ElectroExploration.cs

```

namespace problem_1;

public class ElectroExploration
{
    public class ElectroExplorationBuilder
    {
        private readonly ElectroExploration _electroExploration = new();

        public ElectroExplorationBuilder SetParameters(Parameters parameters)
        {
            _electroExploration._parameters = parameters;
            return this;
        }

        public ElectroExplorationBuilder SetSolver(Solver solver)
        {
            _electroExploration._solver = solver;
```

```
            return this;
        }

        public static implicit operator ElectroExploration(ElectroExplorationBuilder
↪   builder)
            => builder._electroExploration;
    }

    private Parameters _parameters = default!;
    private Solver _solver = default!;
    private Matrix<double> _matrix = default!;
    private Matrix<double> _potentialsDiffs = default!;
    private Vector<double> _vector = default!;
    private Vector<double> _currents = default!;
    private Vector<double> _realPotentials = default!;
    private Vector<double> _primaryPotentials = default!;

    private double _alphaRegulator = 1e-15;

    private void Init()
    {
        _matrix = new(_parameters.PowerSources.Length);
        _vector = new(_parameters.PowerSources.Length);
        _currents = new(_parameters.PowerSources.Length);
        _potentialsDiffs = new(_parameters.PowerSources.Length,
↪   _parameters.PowerReceivers.Length);

        _realPotentials = new(_parameters.PowerReceivers.Length);
        _primaryPotentials = new(_parameters.PowerReceivers.Length);

        _currents.ApplyBy(_parameters.PowerSources, x => x.PrimaryCurrent);
    }

    public void Compute()
    {
        SetupSystem();
        SolveSystem();

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

        foreach (var (current, idx) in _currents.Select((current, idx) => (current,
↪   idx)))
        {
            Console.WriteLine($"I{idx + 1} = {current}");
        }
    }

    private void SetupSystem()
    {
        Init();
        DataGeneration();
        AssemblySystem();
    }

    private void DataGeneration()
    {
        for (int i = 0; i < _parameters.PowerSources.Length; i++)
```

```
        {
            for (int j = 0; j < _parameters.PowerReceivers.Length; j++)
            {
                _potentialsDiffs[i, j] =
                    1.0 / (2.0 * Math.PI * _parameters.Sigma) * (
                        1.0 / Point3D.Distance(_parameters.PowerSources[i].B,
↪ _parameters.PowerReceivers[j].M) -
                        1.0 / Point3D.Distance(_parameters.PowerSources[i].A,
↪ _parameters.PowerReceivers[j].M) -
                        1.0 / Point3D.Distance(_parameters.PowerSources[i].B,
↪ _parameters.PowerReceivers[j].N) +
                        1.0 / Point3D.Distance(_parameters.PowerSources[i].A,
↪ _parameters.PowerReceivers[j].N));

                _realPotentials[j] += _parameters.PowerSources[i].RealCurrent *
↪ _potentialsDiffs[i, j];
                _primaryPotentials[j] += _parameters.PowerSources[i].PrimaryCurrent *
↪ _potentialsDiffs[i, j];
            }
        }
    }

    private void AssemblySystem()
    {
        for (int q = 0; q < _parameters.PowerSources.Length; q++)
        {
            for (int s = 0; s < _parameters.PowerSources.Length; s++)
            {
                for (int i = 0; i < _parameters.PowerReceivers.Length; i++)
                {
                    double w = 1.0 / _realPotentials[i];

                    _matrix[q, s] += w * w * _potentialsDiffs[q, i] *
↪ _potentialsDiffs[s, i];
                }
            }

            for (int i = 0; i < _parameters.PowerReceivers.Length; i++)
            {
                double w = 1.0 / _realPotentials[i];

                _vector[q] -= w * w * _potentialsDiffs[q, i] * (_primaryPotentials[i]
↪ - _realPotentials[i]);
            }
        }
    }

    private void SolveSystem()
    {
        do
        {
            _solver.SetMatrix(_matrix);
            _solver.SetVector(_vector);

            _solver.Compute();

            Regularization();

            _alphaRegulator *= 2.0;
        } while (!_solver.IsSolved());
```

```
    }

    private void Regularization()
    {
        for (int i = 0; i < _matrix.Rows; i++)
        {
            _matrix[i, i] += _alphaRegulator;

            _vector[i] -= _alphaRegulator *
                          (_parameters.PowerSources[i].PrimaryCurrent -
↪   _parameters.PowerSources[i].RealCurrent);
        }
    }

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

Solvers.cs

```
namespace problem_1;

public abstract class Solver
{
    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 Solver(Matrix<double> matrix, Vector<double> vector)
        => (_matrix, _vector) = (Matrix<double>.Copy(matrix),
↪   Vector<double>.Copy(vector));

    protected Solver()
    {
    }

    public abstract void Compute();

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

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

    public Gauss()
    {
    }

    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");

            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);
        }
    }
}
```

Matrix.cs

```
namespace problem_1;

public class Matrix<T> where T : INumber<T>
{
    private readonly 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;
    }
}
```

Vector.cs

```
namespace problem_1;

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

    public static Vector<T> Copy(Vector<T> otherVector)
    {
        Vector<T> newVector = new(otherVector.Length);

        Array.Copy(otherVector._storage, newVector._storage, otherVector.Length);

        return newVector;
    }

    public void ApplyBy<TIn>(IEnumerable<TIn> from, Func<TIn, T> pullOutRule)
    {
        try
        {
            ApplyByLogic(from, pullOutRule);
        }
        catch (Exception ex)
        {
            throw new("Vector and IEnumerable sizes can't be different", ex);
        }
    }

    private void ApplyByLogic<TIn>(IEnumerable<TIn> from, Func<TIn, T> pullOutRule)
    {
        int index = 0;

        foreach (TIn item in from)
        {
            _storage[index] = pullOutRule(item);
            index++;
        }
```

```
50      }
51
52      public IEnumerator<T> GetEnumerator()
53      {
54          foreach (var value in _storage)
55          {
56              yield return value;
57          }
58      }
59
60      IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
61  }
```