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

Практическое задание № 3

по дисциплине «Уравнение математической физики»

РЕШЕНИЕ ГАРМОНИЧЕСКИХ ЗАДАЧ

Группа ПМ-12	БЕРЕЗЕНЦЕВ ТИМОФЕЙ
Вариант 9	ПАВЛОВ ДМИТРИЙ

Преподаватели	ЗАДОРОЖНЫЙ АЛЕКСАНДР ГЕНАДЬЕВИЧ
	ЛЕОНОВИЧ ДАРЬЯНА АЛЕКСАНДРОВНА

Новосибирск, 2024

1 **Задание**

Разработать программу решения гармонической задачи методом конечных элементов. Провести сравнение прямого и итерационного методов решения получаемой в результате конечноэлементной аппроксимации СЛАУ.

2 Теоретическая часть

Определение нелинейной задачи

Постановка задачи

Рассмотрим задачу для уравнения

$$\chi \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{t}^2} + \sigma \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{t}} - \operatorname{div}(\lambda \operatorname{grad} \boldsymbol{u}) = \boldsymbol{f}\,. \qquad (3.1)$$

в котором правая часть $\boldsymbol{f}$ представима в виде

$$\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{t}) = \boldsymbol{f}^s(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\sin\omega\boldsymbol{t} + \boldsymbol{f}^c(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\cos\omega\boldsymbol{t}\,. \qquad (3.2)$$

Если остальные параметры рассматриваемого уравнения (3.1) не зависят от времени, то тогда и его решение $\boldsymbol{u}$ может быть представлено в виде

$$\boldsymbol{u}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{t}) = \boldsymbol{u}^s(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\sin\omega\boldsymbol{t} + \boldsymbol{u}^c(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\cos\omega\boldsymbol{t}\,, \qquad (3.3)$$

где $\boldsymbol{u}^s$ и $\boldsymbol{u}^c$ – две зависящие только от пространственных координат функции, удовлетворяющие системе уравнений

$$\begin{cases} -\operatorname{div}(\lambda \operatorname{grad} \boldsymbol{u}^s) - \omega\sigma\boldsymbol{u}^c - \omega^2\chi\boldsymbol{u}^s = \boldsymbol{f}^s, \\ -\operatorname{div}(\lambda \operatorname{grad} \boldsymbol{u}^c) + \omega\sigma\boldsymbol{u}^s - \omega^2\chi\boldsymbol{u}^c = \boldsymbol{f}^c. \end{cases} \qquad (3.4)$$

Обозначим

$$\boldsymbol{p}_{ij} = \int_{\Omega} \left(\lambda \operatorname{grad}\psi_i \operatorname{grad}\psi_j - \omega^2\chi\psi_i\psi_j\right) \boldsymbol{d}\Omega + \int_{\boldsymbol{S}_3} \beta\psi_i\psi_j \boldsymbol{dS}\,,$$

$$\boldsymbol{c}_{ij} = \omega \int_{\Omega} \sigma\psi_i\psi_j \boldsymbol{d}\Omega\,.$$

Тогда матрица конечноэлементной СЛАУ будет иметь следующую структуру:

$$\mathbf{A} = \begin{pmatrix} \boldsymbol{p}_{11} & -\boldsymbol{c}_{11} & \boldsymbol{p}_{12} & -\boldsymbol{c}_{12} & & & \boldsymbol{p}_{1n} & -\boldsymbol{c}_{1n} \\ \boldsymbol{c}_{11} & \boldsymbol{p}_{11} & \boldsymbol{c}_{12} & \boldsymbol{p}_{12} & & & \boldsymbol{c}_{1n} & \boldsymbol{p}_{1n} \\ \boldsymbol{p}_{21} & -\boldsymbol{c}_{21} & \boldsymbol{p}_{22} & -\boldsymbol{c}_{22} & & & \boldsymbol{p}_{2n} & -\boldsymbol{c}_{2n} \\ \boldsymbol{c}_{21} & \boldsymbol{p}_{21} & \boldsymbol{c}_{22} & \boldsymbol{p}_{22} & & & \boldsymbol{c}_{2n} & \boldsymbol{p}_{2n} \\ & & & & \ldots & & & \\ & & & & & \ldots & & \\ \boldsymbol{p}_{n1} & -\boldsymbol{c}_{n1} & \boldsymbol{p}_{n2} & -\boldsymbol{c}_{n2} & & & \boldsymbol{p}_{nn} & -\boldsymbol{c}_{nn} \\ \boldsymbol{c}_{n1} & \boldsymbol{p}_{n1} & \boldsymbol{c}_{n2} & \boldsymbol{p}_{n2} & & & \boldsymbol{c}_{nn} & \boldsymbol{p}_{nn} \end{pmatrix}.$$

Тестирование программы

Равномерная сетка, $0 \leq x \leq 3, 0 \leq y \leq 3, 0 \leq z \leq 3$, первые краевые условия на всех границах, начальные параметры $\lambda = 1, \omega = 1, \sigma = 1,\ \chi = 1$

$$u^s=2x+y+5z$$

$$u^c=z-3x-y$$

Размерность 686

λ	LU(t), c	LU(погр.)	LOS(t), c	LOS(погр.)
1E+02	0,082	3,37E-16	0,034	2,37E-12
1E+04	0,118	3,08E-16	0,078	1,19E-14
1E+05	0,114	3,41E-16	0,033	9,79E-15
8E+05	0,085	3,86E-16	0,024	4,59E-16

ω	LU(t), c	LU(погр.)	LOS(t), c	LOS(погр.)
1E-04	0,081	3,40E-16	0,018	7,63E-11
1E-01	0,105	5,05E-16	0,027	7,50E-10
1E+00	0,229	3,03E-16	0,108	1,86E-09
1E+01	0,129	6,95E-15	1,171	0,008717
1E+03	0,1	1,31E-15	0,055	5,40E-11
1E+05	0,097	6,54E-16	0,058	1,80E-06
1E+07	0,117	5,78E-16	0,074	0,016931
1E+09	0,94	7,22E-16	0,071	92,90198

σ	LU(t), c	LU(погр.)	LOS(t), c	LOS(погр.)
0E+00	0,116	2,51E-16	0,017	6,54E-11
1E+01	0,094	4,51E-16	0,41	5,11E-10
1E+02	0,079	1,45E-15	1,062	0,022311
1E+04	0,08	4,30E-14	0,909	0,219979
1E+06	0,089	3,51E-12	1,015	18,9771
1E+08	0,075	2,42E-10	0,843	180,835

χ	LU(t), c	LU(погр.)	LOS(t), c	LOS(погр.)
8,81E-12	0,101	3,24E-16	0,043	5,58E-11
1,00E-11	0,124	3,10E-16	0,067	5,58E-11
1,00E-10	0,096	4,27E-16	0,039	5,58E-11

Размерность 8192

λ	LU(t), c	LU(погр.)	LOS(t), c	LOS(погр.)
1E+02	23,497	7,59E-16	0,621	7,15E-12
1E+04	24,107	7,90E-16	0,761	5,21E-14
1E+05	25,201	4,33E-15	0,769	3,05E-14
8E+05	23,906	1,39E-15	0,777	1,33E-15

ω	LU(t), с	LU(погр.)	LOS(t), с	LOS(погр.)
1E-04	24,041	1,66E-15	0,582	2,79E-10
1E-01	23,619	1,68E-15	0,601	1,14E-09
1E+00	23,583	1,77E-15	3,027	5,38E-09
1E+01	23,387	3,25E-15	14,444	0,199376
1E+03	24,319	1,93E-15	1,361	1,18E-11
1E+05	23,774	1,18E-15	2,862	1,26E-07
1E+07	23,61	8,07E-16	3,485	0,001702
1E+09	30,63	7,84E-16	4,14	13,51949

σ	LU(t), с	LU(погр.)	LOS(t), с	LOS(погр.)
0E+00	35,895	1,89E-15	0,699	3,09E-10
1E+01	38,785	8,76E-16	1,857	1,36E-09
1E+02	41,909	1,60E-15	23,80	0,056977
1E+04	39,406	2,35E-14	22,594	0,067575
1E+06	40,206	1,57E-12	23,307	6,544783
1E+08	45,07	9,25E-11	23,714	827,693

χ	LU(t), с	LU(погр.)	LOS(t), с	LOS(погр.)
8,81E-12	38,724	1,20E-15	3,495	3,73E-09
1,00E-11	38,301	2,96E-15	2,725	3,73E-09
1,00E-10	35,688	7,55E-16	2,108	3,73E-09

Вывод:

При небольшой размерности СЛАУ прямой метод и ЛОС решают примерно с одинаковой скоростью, при увеличении размерности ЛОС становится явно быстрее прямого метода.

Однако при увеличении параметров, дающих вклад в побочные диагонали, ω и σ, ЛОС сильно теряет в точности или вовсе не находит решение, так как матрица теряет диагональное преобладание. На точность прямого метода изменение этих параметров практически не влияет.

Листинг программы

```
using EMP_PR3;
using System.Globalization;

CultureInfo.CurrentCulture = new CultureInfo("en-US");

const string file1 = "C:\\Users\\1\\downloads\\UMF_3\\Area\\AreaDescription.txt";

var mesh = new Mesh3D();

mesh.Input(file1);
mesh.BuildMesh();

double[] lambdas = { 1e2, 1e4, 1e5, 8 * 1e5 };
double[] omegas = { 1e-4, 1e-2, 1e0, 1e1, 1e3, 1e5, 1e7, 1e9 };
double[] sigmas = { 0, 1e1, 1e2, 1e4, 1e6, 1e8 };
double[] chis = { 8.81 * 1e-12, 1e-11, 1e-10 };

var sw = new StreamWriter($"C:/Users/1/downloads/UMF_3/Re-
sult/lambda_{mesh.Nodes.Length}.txt");
for (int i = 0; i < lambdas.Length; i++)
{
    mesh.Omega = 1;
    mesh.Sigma = 1;
    mesh.Chi = 1;
    mesh.Lambda = lambdas[i];
    sw.WriteLine($"Lambda = {lambdas[i]}");
    FEM fem = new(mesh);
    fem.SetTest(new Test1(mesh));
    //sw.WriteLine($"By LU");
    //fem.SetSolver(new LU());
    //fem.Compute();
    //fem.PrintSolution(sw);
    sw.WriteLine($"By LOSLU");
    fem.SetSolver(new LOSWithLU());
    fem.Compute();
    fem.PrintSolution(sw);

}
sw.Close();
sw = new StreamWriter($"C:/Users/1/downloads/UMF_3/Re-
sult/omega_{mesh.Nodes.Length}.txt");
for (int i = 0; i < omegas.Length; i++)
{
    mesh.Lambda = 1;
    mesh.Sigma = 1;
    mesh.Chi = 1;
    mesh.Omega = omegas[i];
    sw.WriteLine($"Omega = {omegas[i]}");
    FEM fem = new(mesh);
    fem.SetTest(new Test1(mesh));
    //sw.WriteLine($"By LU");
    //fem.SetSolver(new LU());
    //fem.Compute();
    //fem.PrintSolution(sw);
    sw.WriteLine($"By LOS");
    fem.SetSolver(new LOSWithLU());
    fem.Compute();
    fem.PrintSolution(sw);

}
sw.Close();
sw = new StreamWriter($"C:/Users/1/downloads/UMF_3/Re-
sult/sigma_{mesh.Nodes.Length}.txt");
```

```
for (int i = 0; i < sigmas.Length; i++)
{
    mesh.Lambda = 1;
    mesh.Omega = 1;
    mesh.Chi = 1;
    mesh.Sigma = sigmas[i];
    sw.WriteLine($"sigma = {sigmas[i]}");
    FEM fem = new(mesh);
    fem.SetTest(new Test1(mesh));
    //sw.WriteLine($"By LU");
    //fem.SetSolver(new LU());
    //fem.Compute();
    //fem.PrintSolution(sw);
    sw.WriteLine($"By LOS");
    fem.SetSolver(new LOSWithLU());
    fem.Compute();
    fem.PrintSolution(sw);

}
sw.Close();
sw = new StreamWriter($"C:/Users/1/downloads/UMF_3/Result/chi_{mesh.Nodes.Length}.txt");
for (int i = 0; i < chis.Length; i++)
{
    mesh.Lambda = 1;
    mesh.Omega = 1;
    mesh.Sigma = 1;
    mesh.Chi = chis[i];
    sw.WriteLine($"chi = {chis[i]}");
    FEM fem = new(mesh);
    fem.SetTest(new Test1(mesh));
    //sw.WriteLine($"By LU");
    //fem.SetSolver(new LU());
    //fem.Compute();
    //fem.PrintSolution(sw);
    sw.WriteLine($"By LOS");
    fem.SetSolver(new LOSWithLU());
    fem.Compute();
    fem.PrintSolution(sw);

}
sw.Close();
```

Fem.cs

```
namespace EMP_PR3;
public class FEM
{
    public delegate double Basis(Point3D point);
    private Mesh3D _mesh;
    private SparseMatrix _globalMatrix;
    private Vector _globalVector;
    private Vector _solution;
    private Solver _slae;
    private Vector _localVector, _localVector1, _localVector2;
    private Matrix _stiffnessMatrix;
    private Matrix _massMatrix;
    public static int NodesPerElement => 8;
    public Test TestCase;
    public FEM(Mesh3D mesh)
    {
        _mesh = mesh;
        _stiffnessMatrix = new(NodesPerElement);
        _massMatrix = new(NodesPerElement);
        _localVector = new(2 * NodesPerElement);
        _localVector1 = new(NodesPerElement);
        _localVector2 = new(NodesPerElement);
        _globalMatrix = new SparseMatrix(0, 0);
        _globalVector = new(0);
        _solution = new(0);
        _slae = new LOSWithLU();
    }
    public void SetTest(Test test)
    => TestCase = test;
    public void SetSolver(Solver slae)
    => _slae = slae;
    public void Compute()
    {
        BuildPortrait();
        AssemblySlae();
        AccountFirstConditions();
        _slae.SetSLAE(_globalVector, _globalMatrix);
        _solution = _slae.Solve();
    }
    public void BuildPortrait()
    {
        var list = new HashSet<int>[2 * _mesh.Nodes.Length].Select(_ => new
       HashSet<int>()).ToList();
        foreach (var element in _mesh.Elements)
        {
            foreach (var position in element)
            {
                foreach (var node in element)
                {
                    if (position == node)
                    {
                        list[2 * position + 1].Add(2 * position);
                    }
                    else if (position > node)
                    {
                        list[2 * position].Add(2 * node);
                        list[2 * position].Add(2 * node + 1);
                        list[2 * position + 1].Add(2 * node);
                        list[2 * position + 1].Add(2 * node + 1);
                    }
                }
            }
        }
        list = list.Select(childlist => childlist.Order().ToHashSet()).ToList();
        int offDiagonalElementsCount = list.Sum(childList => childList.Count);
        _globalMatrix = new(2 * _mesh.Nodes.Length, offDiagonalElementsCount);
        _globalVector = new(2 * _mesh.Nodes.Length);
        _globalMatrix._ia[0] = 0;
        for (int i = 0; i < list.Count; i++)
            _globalMatrix._ia[i + 1] = _globalMatrix._ia[i] + list[i].Count;
        int k = 0;
        foreach (var childList in list)
        {
            foreach (var value in childList)
            {
                _globalMatrix._ja[k++] = value;
            }
        }
    }
    private void AssemblySlae()
    {
        _globalVector.Fill(0);
        _globalMatrix.Clear();
        for (int ielem = 0; ielem < _mesh.Elements.Length; ielem++)
        {
            AssemblyLocalSLAE(ielem);
            for (int i = 0; i < NodesPerElement; i++)
                for (int j = 0; j < NodesPerElement; j++)
                {
                    AddElement(2 * _mesh.Elements[ielem][i], 2 *
                   _mesh.Elements[ielem][j], _stiffnessMatrix[i, j]);
                    AddElement(2 * _mesh.Elements[ielem][i] + 1, 2 *
                   _mesh.Elements[ielem][j] + 1, _stiffnessMatrix[i, j]);
                    AddElement(2 * _mesh.Elements[ielem][i], 2 *
                   _mesh.Elements[ielem][j] + 1, -_massMatrix[i, j]);
                    AddElement(2 * _mesh.Elements[ielem][i] + 1, 2 *
                   _mesh.Elements[ielem][j], _massMatrix[i, j]);
                }
            AddElementToVector(ielem);
            _stiffnessMatrix.Clear();
            _massMatrix.Clear();
            _localVector1.Fill(0);
            _localVector2.Fill(0);
        }
    }
    private void AssemblyLocalSLAE(int ielem)
    {
        int Mu(int i) => i % 2;
        int Nu(int i) => i / 2 % 2;
        int Theta(int i) => i / 4;
        double hx = Math.Abs(_mesh.Nodes[_mesh.Elements[ielem][7]].X -
       _mesh.Nodes[_mesh.Elements[ielem][0]].X);
        double hy = Math.Abs(_mesh.Nodes[_mesh.Elements[ielem][7]].Y -
       _mesh.Nodes[_mesh.Elements[ielem][0]].Y);
        double hz = Math.Abs(_mesh.Nodes[_mesh.Elements[ielem][7]].Z -
       _mesh.Nodes[_mesh.Elements[ielem][0]].Z);
        double[,] matrixG =
        {
            { 1.0, -1.0 },
            { -1.0, 1.0 }
        };
        double[,] matrixM =
        {
            { 2.0 / 6.0, 1.0 / 6.0 },
            { 1.0 / 6.0, 2.0 / 6.0 }
        };
        for (int i = 0; i < NodesPerElement; i++)
        {
            for (int j = 0; j < NodesPerElement; j++)
            {
                _stiffnessMatrix[i, j] =
                matrixG[Mu(i), Mu(j)] / hx * matrixM[Nu(i), Nu(j)] * hy *
               matrixM[Theta(i), Theta(j)] * hz +
                matrixM[Mu(i), Mu(j)] * hx * matrixG[Nu(i), Nu(j)] / hy *
               matrixM[Theta(i), Theta(j)] * hz +
                matrixM[Mu(i), Mu(j)] * hx * matrixM[Nu(i), Nu(j)] * hy *
               matrixG[Theta(i), Theta(j)] / hz;
                _massMatrix[i, j] = matrixM[Mu(i), Mu(j)] * hx * matrixM[Nu(i),
               Nu(j)] * hy * matrixM[Theta(i), Theta(j)] * hz;
            }
        }
        for (int i = 0; i < NodesPerElement; i++)
        {
            _localVector1[i] = TestCase.Fs(_mesh.Nodes[_mesh.Elements[ielem][i]]);
            _localVector2[i] = TestCase.Fc(_mesh.Nodes[_mesh.Elements[ielem][i]]);
        }
        _localVector1 = _massMatrix * _localVector1;
        _localVector2 = _massMatrix * _localVector2;
        for (int i = 0; i < NodesPerElement; i++)
        {
            _localVector[2 * i] = _localVector1[i];
            _localVector[2 * i + 1] = _localVector2[i];
        }
        _stiffnessMatrix = _mesh.Lambda * _stiffnessMatrix + (-_mesh.Omega) *
       _mesh.Omega * _mesh.Chi * _massMatrix;
        _massMatrix = _mesh.Omega * _mesh.Sigma * _massMatrix;
    }
    private void AddElement(int i, int j, double value)
    {
        if (i == j)
        {
            _globalMatrix._di[i] += value;
            return;
        }
        if (i > j)
            for (int icol = _globalMatrix._ia[i]; icol < _globalMatrix._ia[i + 1];
icol++)
            {
                if (_globalMatrix._ja[icol] == j)
                {
                    _globalMatrix._al[icol] += value;
                    return;
                }
            }
        else
            for (int icol = _globalMatrix._ia[j]; icol < _globalMatrix._ia[j + 1];
icol++)
            {
                if (_globalMatrix._ja[icol] == i)
                {
                    _globalMatrix._au[icol] += value;
                    return;
                }
            }
    }
    private void AddElementToVector(int ielem)
    {
        for (int i = 0; i < NodesPerElement; i++)
        {
            _globalVector[2 * _mesh.Elements[ielem][i]] += _localVector[2 * i];
            _globalVector[2 * _mesh.Elements[ielem][i] + 1] += _localVector[2 * i +
           1];
        }
    }
    public void AccountFirstConditions()
    {
        foreach (var node in _mesh.Boundaries)
        {
            int row = 2 * node;
            _globalMatrix._di[row] = 1;
            _globalVector[row] = TestCase.Us(_mesh.Nodes[node]);
            for (int i = _globalMatrix._ia[row]; i < _globalMatrix._ia[row + 1];
           i++)
            {
                _globalMatrix._al[i] = 0;
            }
            for (int col = row + 1; col < _globalMatrix.Size; col++)
            {
                for (int j = _globalMatrix._ia[col]; j < _globalMatrix._ia[col + 1];
               j++)
                    if (_globalMatrix._ja[j] == row)
                    {
                        _globalMatrix._au[j] = 0;
                        break;
                    }
            }
            row = 2 * node + 1;
            _globalMatrix._di[row] = 1;
            _globalVector[row] = TestCase.Uc(_mesh.Nodes[node]);
            for (int i = _globalMatrix._ia[row]; i < _globalMatrix._ia[row + 1]; i++)
            {
                _globalMatrix._al[i] = 0;
            }
            for (int col = row + 1; col < _globalMatrix.Size; col++)
            {
                for (int j = _globalMatrix._ia[col]; j < _globalMatrix._ia[col + 1];
j++)
                    if (_globalMatrix._ja[j] == row)
                    {
                        _globalMatrix._au[j] = 0;
                        break;
                    }
            }
        }
    }
    public void PrintSolution(StreamWriter sw)
    {
        Vector exactSolution = new(2 * _mesh.Nodes.Length);
        for (int i = 0; i < exactSolution.Size / 2; i++)
        {
            exactSolution[2 * i] = TestCase.Us(_mesh.Nodes[i]);
            exactSolution[2 * i + 1] = TestCase.Uc(_mesh.Nodes[i]);
        }
        Vector inaccuracySin = new(_mesh.Nodes.Length);
        Vector inaccuracyCos = new(_mesh.Nodes.Length);
        Vector inaccuracy = new(2 * _mesh.Nodes.Length);
        for (int i = 0; i < _mesh.Nodes.Length; i++)
        {
            if (!_mesh.Boundaries.Contains(i / 2))
            {
                inaccuracySin[i] = exactSolution[2 * i] - _solution[2 * i];
                inaccuracyCos[i] = exactSolution[2 * i + 1] - _solution[2 * i + 1];
            }
        }
        for (int i = 0; i < inaccuracy.Size; i++)
        {
            if (!_mesh.Boundaries.Contains(i / 2))
                inaccuracy[i] = exactSolution[i] - _solution[i];

        }

        sw.Write($"{_slae.SolveTime / 1000.0} ");
        sw.Write($"{inaccuracy.Norm() / exactSolution.Norm()} ");
        sw.Write($"{inaccuracySin.Norm() / exactSolution.Norm()} ");
        sw.Write($"{inaccuracyCos.Norm() / exactSolution.Norm()} ");
        sw.WriteLine();

        Console.WriteLine($"{_slae.SolveTime / 1000.0} сек.");
    }
}
```

Solver.cs

```
using System.Diagnostics;


namespace EMP_PR3;
```

```
public abstract class Solver
{
    protected double _eps = 1e-14;
    protected int _maxIters = 1000;
    protected SparseMatrix _matrix;
    protected Vector _vector;
    protected Vector _solution;
    public double SolveTime { get; protected set; }
    public Solver()
    {
        _matrix = new SparseMatrix(0, 0);
        _vector = new Vector(0);
        _solution = new Vector(0);
    }
    public void SetSLAE(Vector vector, SparseMatrix matrix)
    {
        _vector = vector;
        _matrix = matrix;
    }
    public abstract Vector Solve();
    protected void LU()
    {
        for (int i = 0; i < _matrix.Size; i++)
        {
            for (int j = _matrix._ia[i]; j < _matrix._ia[i + 1]; j++)
            {
                int jCol = _matrix._ja[j];
                int jk = _matrix._ia[jCol];
                int k = _matrix._ia[i];
                int sdvig = _matrix._ja[_matrix._ia[i]] -
               _matrix._ja[_matrix._ia[jCol]];
                if (sdvig > 0)
                    jk += sdvig;
                else
                    k -= sdvig;
                double sumL = 0.0;
                double sumU = 0.0;
                for (; k < j && jk < _matrix._ia[jCol + 1]; k++, jk++)
                {
                    sumL += _matrix._al[k] * _matrix._au[jk];
                    sumU += _matrix._au[k] * _matrix._al[jk];
                }
                _matrix._al[j] -= sumL;
                _matrix._au[j] -= sumU;
                _matrix._au[j] /= _matrix._di[jCol];
            }
            double sumD = 0.0;
            for (int j = _matrix._ia[i]; j < _matrix._ia[i + 1]; j++)
                sumD += _matrix._al[j] * _matrix._au[j];
            _matrix._di[i] -= sumD;
        }
    }
    protected void ForwardElimination()
    {
        for (int i = 0; i < _matrix.Size; i++)
        {
            for (int j = _matrix._ia[i]; j < _matrix._ia[i + 1]; j++)
                {
                    _solution[i] -= _matrix._al[j] * _solution[_matrix._ja[j]];
                }
                _solution[i] /= _matrix._di[i];
        }
    }
    protected void BackwardSubstitution()
    {
        for (int i = _matrix.Size - 1; i >= 0; i--)
        {
            for (int j = _matrix._ia[i + 1] - 1; j >= _matrix._ia[i]; j--)
            {
                _solution[_matrix._ja[j]] -= _matrix._au[j] * _solution[i];
            }
        }
    }
    public void PrintSolution()
    {
        for (int i = 0; i < _solution.Size; i++)
        {
            Console.WriteLine(_solution[i]);
        }
    }
}
public class LOS : Solver
{
    public override Vector Solve()
    {
        _solution = new(_vector.Size);
        Vector.Copy(_vector, _solution);
        Vector r = _vector - _matrix * _solution;
        Vector z = 1 * r;
        Vector p = _matrix * z;
        Vector tmp;
        double alpha;
        double beta;
        Stopwatch sw = Stopwatch.StartNew();
        double discrepancy = r * r;
        for (int i = 1; i <= _maxIters && discrepancy > _eps; i++)
        {
            alpha = (p * r) / (p * p);
            _solution += alpha * z;
            r -= alpha * p;
            tmp = _matrix * r;
            beta = -(p * tmp) / (p * p);
            z = r + beta * z;
            p = tmp + beta * p;
            discrepancy = r * r;
        }
        sw.Stop();

        SolveTime = sw.ElapsedMilliseconds;
        return _solution;
    }
}
public class LU : Solver
{
    public override Vector Solve()
    {
        _solution = new(_vector.Size);
        Vector.Copy(_vector, _solution);
        _matrix = _matrix.ConvertToProfile();
        Stopwatch sw = Stopwatch.StartNew();
        LU();
        ForwardElimination();
        BackwardSubstitution();
        sw.Stop();
        SolveTime = sw.ElapsedMilliseconds;
        return _solution;
    }
}
public class LOSWithLU : Solver
{
    public override Vector Solve()
    {
        _solution = new(_vector.Size);
        Vector.Copy(_vector, _solution);
        SparseMatrix matrixLU = new(_matrix.Size, _matrix._ja.Length);
        SparseMatrix.Copy(_matrix, matrixLU);
        //matrixLU = matrixLU.ConvertToProfile();
        PartitialLU(matrixLU);
        Vector r = Forward(matrixLU, _vector - _matrix * _solution);
        Vector z = Backward(matrixLU, r);
        Vector p = Forward(matrixLU, _matrix * z);
        Vector tmp;
        double alpha;
        double beta;
        Stopwatch sw = Stopwatch.StartNew();
        double discrepancy = r * r;
        for (int i = 1; i <= _maxIters && discrepancy > _eps; i++)
        {
            alpha = (p * r) / (p * p);
            _solution += alpha * z;
            r -= alpha * p;
            tmp = Forward(matrixLU, _matrix * Backward(matrixLU, r));
            beta = -(p * tmp) / (p * p);
            z = Backward(matrixLU, r) + beta * z;
            p = tmp + beta * p;
            discrepancy = r * r;
        }
        sw.Stop();
        SolveTime = sw.ElapsedMilliseconds;
        return _solution;
    }
    protected static void PartitialLU(SparseMatrix Matrix)
    {
        for (int i = 0; i < Matrix.Size; i++)
        {
            for (int j = Matrix._ia[i]; j < Matrix._ia[i + 1]; j++)
            {
                int jCol = Matrix._ja[j];
                int jk = Matrix._ia[jCol];
                int k = Matrix._ia[i];
                int sdvig = Matrix._ja[Matrix._ia[i]] -
               Matrix._ja[Matrix._ia[jCol]];
                if (sdvig > 0)
                    jk += sdvig;
                else
                    k -= sdvig;
                double sumL = 0.0;
                double sumU = 0.0;
                for (; k < j && jk < Matrix._ia[jCol + 1]; k++, jk++)
                {
                    sumL += Matrix._al[k] * Matrix._au[jk];
                    sumU += Matrix._au[k] * Matrix._al[jk];
                }
                Matrix._al[j] -= sumL;
                Matrix._au[j] -= sumU;
                Matrix._au[j] /= Matrix._di[jCol];
            }
            double sumD = 0.0;
            for (int j = Matrix._ia[i]; j < Matrix._ia[i + 1]; j++)
                sumD += Matrix._al[j] * Matrix._au[j];
            Matrix._di[i] -= sumD;
        }
    }
    protected static Vector Forward(SparseMatrix Matrix, Vector b)
    {
        var result = new Vector(b.Size);
        Vector.Copy(b, result);
        for (int i = 0; i < Matrix.Size; i++)
        {
            for (int j = Matrix._ia[i]; j < Matrix._ia[i + 1]; j++)
            {
                result[i] -= Matrix._al[j] * result[Matrix._ja[j]];
            }
            result[i] /= Matrix._di[i];
        }
        return result;
    }
    protected static Vector Backward(SparseMatrix Matrix, Vector b)
    {
        var result = new Vector(b.Size);
        Vector.Copy(b, result);
        for (int i = Matrix.Size - 1; i >= 0; i--)
        {
            for (int j = Matrix._ia[i + 1] - 1; j >= Matrix._ia[i]; j--)
            {
                result[Matrix._ja[j]] -= Matrix._au[j] * result[i];
            }
        }
        return result;
    }
}
```