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 №4
по дисциплине «Уравнения математической физики»

Решение несимметричных СЛАУ

Студенты	БЕГИЧЕВ АЛЕКСАНДР
	ШИШКИН НИКИТА
Группа	ПМ-92
Преподаватели	ЗАДОРОЖНЫЙ А. Г.
	ПАТРУШЕВ И. И.
	ПЕРСОВА М. Г.

Новосибирск, 2022

Цель работы

Изучить особенности реализации итерационных методов BCG, BCGSTAB, GMRES для СЛАУ с несимметричными разреженными матрицами. Исследовать влияние предобуславливания.

Вариант: Реализовать решение СЛАУ методом BCGSTAB с LU - предобуславливанием.

Практическая часть

1. Реализовать программу решения СЛАУ большой размерности в разреженном строчном формате в соответствии с заданием.
2. Протестировать разработанную программу на небольших матрицах.
3. Сравнить реализованный метод по вычислительным затратам с методами, используемыми в предыдущей лабораторной работе, на матрицах большой размерности, полученных в результате конечноэлементной аппроксимации в предыдущей лабораторной работе.

Тестирование

Тестирование метода на небольших матрицах

$$\begin{bmatrix} 6 & 2 & 0 & 5 & 8 \\ 2 & 7 & 3 & 0 & 12 \\ 0 & 1 & 9 & 6 & 13 \\ 4 & 0 & 2 & 10 & 15 \\ 7 & 9 & 10 & 14 & -5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 70 \\ 85 \\ 118 \\ 125 \\ 86 \end{bmatrix}$$

Метод	Время (мс)	Кол-во итераций	Невязка
LOS (LU)	4	10	$4.31E-16$
BCGSTAB (LU)	4	3	$4.27E-21$

$$\begin{bmatrix} 10 & -4 & 0 & 0 & 0 & 0 & -2 & -3 & 0 & 0 & 0 & 0 \\ -2 & 17 & -6 & 0 & 0 & 0 & 0 & -5 & -4 & 0 & 0 & 0 \\ 0 & -7 & 13 & -2 & 0 & 0 & 0 & 0 & -3 & -1 & 0 & 0 \\ 0 & 0 & -3 & 6 & -1 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & -4 & 10 & -2 & 0 & 0 & 0 & 0 & -1 & -3 \\ 0 & 0 & 0 & 0 & 0 & 8 & -4 & 0 & 0 & 0 & 0 & -4 \\ -5 & 0 & 0 & 0 & 0 & -3 & 10 & -2 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & -2 & 6 & -3 & 0 & 0 & 0 \\ 0 & -2 & -1 & 0 & 0 & 0 & 0 & -1 & 4 & 0 & 0 & 0 \\ 0 & 0 & -4 & -2 & 0 & 0 & 0 & 0 & -3 & 13 & -4 & 0 \\ 0 & 0 & 0 & -3 & -4 & 0 & 0 & 0 & 0 & -5 & 13 & -1 \\ 0 & 0 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & -2 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{bmatrix} = \begin{bmatrix} -36 \\ -62 \\ -20 \\ -10 \\ -25 \\ -28 \\ 31 \\ 6 \\ 21 \\ 39 \\ 49 \\ 22 \end{bmatrix}$$

Метод	Время (мс)	Кол-во итераций	Невязка
LOS (LU)	5	23	$4.39E-15$
BCGSTAB (LU)	5	8	$4.89E-15$

Тестирование метода на матрицах большой размерности

Возьмем матрицу, сгенерированную в прошлой лабораторной работе (количество узлов $\approx$ 500).

Метод	Время (мс)	Кол-во итераций	Невязка
LOS (LU)	20	16	$3.26E-15$
BCGSTAB (LU)	16	11	$3.53E-16$

Матрица, сгенерированная в прошлой лабораторной работе (количество узлов $\approx$ 1400).

Метод	Время (мс)	Кол-во итераций	Невязка
LOS (LU)	54	20	$2.44E-15$
BCGSTAB (LU)	51	14	$5.56E-16$

Проведенные исследования и выводы

Из результатов проделанной работы можно сделать вывод, что метод BCGSTAB сходится за меньшее количество итераций, чем LOS, при этом время работы методов близки (разница в несколько миллисекунд). Для решения СЛАУ больших размерностей рекомендуется использовать BCGSTAB.

Тексты основных модулей

Solvers.cs

```
namespace eMP_3;

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

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

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

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

    public abstract void Compute();

    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++)
```

Приведен единственный класс, который был реализован в предыдущей лабораторной работе.

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

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

            double alpha, beta;
            double squareNorm;
```

```
            _solution = new(_vector.Length);

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

            Stopwatch sw = Stopwatch.StartNew();

            r = _vector - (_matrix * _solution);

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

            p = _matrix * z;

            squareNorm = r * r;

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

                tmp = _matrix * r;

                beta = -(p * tmp) / (p * p);
                z = r + (beta * z);
                p = tmp + (beta * p);
            }

            sw.Stop();

            _runningTime = sw.Elapsed;
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}

public class LOSLU : Solver {
    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");

            double alpha, beta;
            double squareNorm;

            _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];

            _matrix.ggl.Copy(gglnew);
```

```
            _matrix.ggu.Copy(ggunew);
            _matrix.di.Copy(dinew);

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

            Stopwatch sw = Stopwatch.StartNew();

            LU(gglnew, ggunew, dinew);

            r = Direct(_vector - (_matrix * _solution), gglnew, dinew);
            z = Reverse(r, ggunew);
            p = Direct(_matrix * z, gglnew, dinew);

            squareNorm = r * r;

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

                tmp = Direct(_matrix * Reverse(r, ggunew), gglnew, dinew);

                beta = -(p * tmp) / (p * p);
                z = Reverse(r, ggunew) + (beta * z);
                p = tmp + (beta * p);
            }

            sw.Stop();

            _runningTime = sw.Elapsed;
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}

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

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

            double alpha = 1.0;
            double omega = 1.0;
            double rho = 1.0;
            double beta, temp;

            double vectorNorm = _vector.Norm();

            _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];

            _matrix.ggl.Copy(gglnew);
            _matrix.ggu.Copy(ggunew);
            _matrix.di.Copy(dinew);

            Vector<double> r = new(_vector.Length);
            Vector<double> r0 = new(_vector.Length);
            Vector<double> z = new(_vector.Length);
            Vector<double> p = new(_vector.Length);
            Vector<double> v = new(_vector.Length);
            Vector<double> s = new(_vector.Length);
            Vector<double> t = new(_vector.Length);

            Stopwatch sw = Stopwatch.StartNew();

            LU(gglnew, ggunew, dinew);

            r = Direct(_vector - (_matrix * _solution), gglnew, dinew);

            Vector<double>.Copy(r, r0);

            for (int iter = 0; iter < MaxIters && r.Norm() / vectorNorm >= Eps;
            ↪  iter++) {
                temp = rho;
                rho = r0 * r;
                beta = rho / temp * (alpha / omega);
                p = r + (beta * (p - (omega * v)));
                v = Direct(_matrix * Reverse(p, ggunew), gglnew, dinew);
                alpha = rho / (r0 * v);
                s = r - (alpha * v);
                t = Direct(_matrix * Reverse(s, ggunew), gglnew, dinew);
                omega = t * s / (t * t);
                _solution += (omega * s) + (alpha * p);
                r = s - (omega * t);
            }

            _solution = Reverse(_solution, ggunew);

            sw.Stop();

            _runningTime = sw.Elapsed;
        } catch (Exception ex) {
            Console.WriteLine($"We had problem: {ex.Message}");
        }
    }
}
```