БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №3
Метод простых итераций. Метод Зейделя. Метод релаксации

Преподаватель: Полевиков Виктор Кузьмич
Студент: Шиляев Иван
2 курс 9 группа

Постановка задачи

Сгенерировать случайным образом вещественные элементы квадтарной матрицы A размера 9×9 с точностью до двух знаков после запятой. Задать вектор x и вычислить $f = Ax$. Далее:

1. Построить сходящийся алгоритм метода простых итераций, найти решение системы $Ax = f$ при $\epsilon = 10^{-7}$ и определить число итераций.

2. Найти решение, используя метод Зейделя (q=1).

3. Построить сходящийся алгоритм метода релаксации и вычислить решение при $0 < q < 2$. Найти q-оптимальное с точностью до $0,01$.

4. Изобразить график $N_{\epsilon(q)}$.

Краткая теория

Рассмотрим систему:

$$Ax = f, |A| \neq 0, A = \{a_{ij}\}_{n\times n}, x = \begin{pmatrix} x \\ \dots \\ x_n \end{pmatrix}, f = \begin{pmatrix} f_1 \\ \dots \\ f_n \end{pmatrix} \tag{1}$$

Для нахождения решения системы по методу простых итераций её необходимо привести к следующему каноническому виду:

$$x = Bx + g, B = \{a_{ij}\}_{n\times n}, g = \{g_i\} \tag{2}$$

По критерию сходимости (теорема 1) метод простых итераций сходится тогда и только тогда когда спектральный радиус матрицы B меньше 1. Поскольку матрица A выбирается произвольно, то необходимо сначала получить симметрическую и положительно определённую матрицу. Для этого (замечание 3) домножим обе части системы (1) слева на матрицу A^T. В результате получим следующую систему:

$$A^T Ax = A^T f, \widetilde{A}x = \widetilde{f}, \widetilde{A} = \widetilde{A^T} > 0 \tag{3}$$

Систему (3) можно привести к виду:

$$x = \left(E - \frac{\widetilde{A}}{\left\|\widetilde{A}\right\|}\right)x + \frac{f}{\left\|\widetilde{A}\right\|} \tag{4}$$

Выполняя замену $B = \left(E - \frac{\widetilde{A}}{\left\|\widetilde{A}\right\|}\right), \rho(B) < 1, g = \frac{f}{\left\|\widetilde{A}\right\|}$, получаем систему (2).

Далее выбираем вектор начального итерационного приближения: $x^{(0)} = g$. Строим итерационную последовательность по правилу: $x^{(k+1)} = Bx^{(k)} + g$, где $x^{(k)}$ - k-е итерационное приближение, а k - номер итерации.

Рассмотрим решение системы (1) по методу релаксации. Итерационная последовательность строится по следующему правилу:

$$x_i^{(k+1)} = (1-q)x_i^{(k)} + q\left(\sum_{j=1}^{i-1} b_{ij}x_j^{(k+1)} + \sum_{j=i}^{n} b_{ij}x_j^{(k)} + g_i\right), i = \overline{1,n} \tag{5}$$

По теореме при положительно определённой и симметрической матрице A при $0 < q < 2$ следующий метод релаксации сходится:

$$x_i^{(k+1)} = (1-q)x_i^{(k)} + q\left(-\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}}x_j^{(k+1)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}}x_j^{(k)} + \frac{f_i}{a_{ii}}\right), i = \overline{1,n} \tag{6}$$

Листинг программы на языке Java

```java
package com;

public class Solution {
    private static final int SIZE = 9;
    private static double[][] A = new double[SIZE][SIZE];
    private static double[][] E = new double[SIZE][SIZE];
    public static double[][] A_INVERSE = new double[SIZE][SIZE];
    private static double[][] A_TRANSPOSED = new double[SIZE][SIZE];
    private static double[] x = new double[SIZE];
    private static double[] myX = new double[SIZE];
    private static double[] f = new double[SIZE];
    private static boolean isDetZero = false;

    private static final double EPS = 0.0000001;

    public static double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();
        long factor = (long) Math.pow(10, places);
        value = value * factor;
        long tmp = Math.round(value);
        return (double) tmp / factor;
    }

    public static void printMatrix4(double[][] A) {
        for (int i=0; i<SIZE; ++i) {
            for (int j=0; j<SIZE; ++j) {
                System.out.printf("%13.4f ", A[i][j]);
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void printVector12(double[] x) {
        for (int i=0; i<SIZE; ++i) {
            System.out.printf("%17.12f\n", x[i]);
        }
        System.out.println("\n");
    }

    public static void generateMatrix(double[][] A) {
        for (int i=0; i<SIZE; ++i) {
            for (int j=i; j<SIZE; ++j) {
                A[i][j] = round( -99 + Math.random()*199, 2);
                A[j][i] = A[i][j];
            }
        }
    }

    public static void generateX(double[] x) {
        for (int i=0; i<SIZE; ++i) {
            x[i] = round(-99 + Math.random()*199, 10);
        }
    }

    public static void countF(double[][] A, double[] x) {
        for (int i=0; i<SIZE; ++i) {
            f[i]=0;
        }
        for (int i=0; i<SIZE; ++i) {
            for (int j=0; j<SIZE; ++j) {
                f[i]+=A[i][j]*x[j];
            }
        }
    }

    // решение по методу Гаусса без выбора главного элемента
    public static void solve(double[][] myA, double[] myX, double[] f) {
        double[][] A = new double[SIZE][SIZE];
        for (int i=0; i<SIZE; ++i) {
            System.arraycopy(myA[i], 0, A[i], 0, SIZE);
        }

        for (int k=1; k<SIZE; ++k) {
            for (int i=k; i<SIZE; ++i) {
                f[i] -= f[k-1]*A[i][k-1]/A[k-1][k-1];
```

```
            double tmp = A[i][k-1];
            for (int j = 0; j < SIZE; ++j) {
                A[i][j] -= A[k-1][j] * tmp / A[k-1][k-1];
            }
        }
    }

    myX[SIZE-1] = f[SIZE-1]/A[SIZE-1][SIZE-1];
    for (int i=SIZE-2; i>=0; --i) {
        double sum = 0;
        for (int j=i+1; j<SIZE; ++j)
            sum += A[i][j]*myX[j];
        myX[i] = (f[i]-sum)/A[i][i];
    }
}

public static double det(double[][] myA) {
    double det = 1;
    double[][] A = new double[SIZE][SIZE];
    for (int i=0; i<SIZE; ++i) {
        System.arraycopy(myA[i], 0, A[i], 0, SIZE);
    }
    for (int k=1; k<SIZE; ++k) {
        for (int i=k; i<SIZE; ++i) {
            double tmp = A[i][k-1];
            for (int j = 0; j < SIZE; ++j) {
                A[i][j] -= A[k-1][j] * tmp / A[k-1][k-1];
            }
        }
    }
    for (int i=0; i<SIZE; ++i) {
        det *= A[i][i];
    }
    return det;
}

public static void countInverseMatrix(double[][] A, double[][] A_INVERSE) {
    if (det(A) == 0) {
        isDetZero = true;
        return;
    }
    double[] myf = new double[SIZE];
    double[] myX = new double[SIZE];
    for (int l=0; l<SIZE; ++l) {
        for (int i=0; i<SIZE; ++i)
            myf[i] = 0;
        myf[l] = 1;
        solve(A, myX, myf);
        for (int i=0; i<SIZE; ++i) {
            A_INVERSE[i][l] = myX[i];
        }
    }
}

public static void countTransposedMatrix(double[][] A, double[][] A_TRANSPOSED) {
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            A_TRANSPOSED[i][j] = A[j][i];
        }
    }
}

// сложение матриц
public static double[][] addMatrix(double[][] A, double[][] B) {
    double[][] C = new double[SIZE][SIZE];
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
    return C;
}

// вычитание матриц
public static double[][] subtractMatrix(double[][] A, double[][] B) {
    double[][] C = new double[SIZE][SIZE];
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
```

```
    return C;
}

// умножение матриц
public static double[][] multiplyMatrix(double[][] A, double[][] B) {
    double[][] C = new double[SIZE][SIZE];
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            for (int k=0; k<SIZE; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}

// умножение матрицы на число
public static double[][] multiplyMatrixByNumber(double[][] A, double n) {
    double[][] C = new double[SIZE][SIZE];
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            C[i][j] = A[i][j] * n;
        }
    }
    return C;
}

// деление матрицы на число
public static double[][] divideMatrixByNumber(double[][] A, double n) {
    double[][] C = new double[SIZE][SIZE];
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            C[i][j] = A[i][j] / n;
        }
    }
    return C;
}

// деление вектора на число
public static double[] divideVectorByNumber(double[] f, double n) {
    double[] v = new double[SIZE];
    for (int i=0; i<SIZE; ++i) {
        v[i] = f[i] / n;
    }
    return v;
}

// умножение матрицы на вектор
public static double[] multipleMatrixByVector(double[][] A, double[] x) {
    double[] v = new double[SIZE];
    for (int i=0; i<SIZE; ++i) {
        for (int j=0; j<SIZE; ++j) {
            v[i] += A[i][j] * x[j];
        }
    }
    return v;
}

// сложение векторов
public static double[] addVectorToVector(double[] a, double[] b) {
    double[] v = new double[SIZE];
    for (int i=0; i<SIZE; ++i) {
        v[i] = a[i] + b[i];
    }
    return v;
}

// вычитание векторов
public static double[] subtractVectorFromVector(double[] a, double[] b) {
    double[] v = new double[SIZE];
    for (int i=0; i<SIZE; ++i) {
        v[i] = a[i] - b[i];
    }
    return v;
}

// норма вектора
public static double vectorNorm(double[] a) {
    double res = Math.abs(a[0]);
    for (int i=1; i<SIZE; ++i) {
        if (Math.abs(a[i]) > res)
```

```
                res = Math.abs(a[i]);
        }
        return res;
    }

    // сумма по столбцам
    public static double norm_1(double[][] A) {
        double max = 0;
        for (int j=0; j<SIZE; ++j) {
            double sum = 0;
            for (int i=0; i<SIZE; ++i) {
                sum += Math.abs(A[i][j]);
            }
            if (sum>max)
                max = sum;
        }
        return max;
    }

    // сумма по строкам
    public static double norm_2(double[][] A) {
        double max = 0;
        for (int i=0; i<SIZE; ++i) {
            double sum = 0;
            for (int j=0; j<SIZE; ++j) {
                sum += Math.abs(A[i][j]);
            }
            if (sum>max)
                max = sum;
        }
        return max;
    }

    // simple-iteration method
    public static int solveSIM(double[][] B, double[] g) {
        double[] x0 = g;
        double[] x1 = addVectorToVector(multipleMatrixByVector(B, x0), g);
        int iterationNumber = 0;
        while (vectorNorm(subtractVectorFromVector(x1, x0)) > EPS) {
            x0 = x1;
            x1 = addVectorToVector(multipleMatrixByVector(B, x0), g);
            ++iterationNumber;
        }
        System.out.println("Solution:");
        printVector10(x1);
        System.out.println("Number of iterations: " + iterationNumber + "\n");
        return iterationNumber;
    }

    // relaxation method
    public static int solveRelaxation(double[][] A1, double q) {
        // counting L, R, D
        double[][] L = new double[SIZE][SIZE];
        double[][] R = new double[SIZE][SIZE];
        double[][] D = new double[SIZE][SIZE];
        for (int i=1; i<SIZE; ++i) {
            System.arraycopy(A1[i], 0, L[i], 0, i);
        }
        for (int i=0; i<SIZE-1; ++i) {
            for (int j=i+1; j<SIZE; ++j) {
                R[i][j] = A1[i][j];
            }
        }
        for (int i=0; i<SIZE; ++i) {
            D[i][i] = A1[i][i];
        }

        // counting new B and g
        double[][] B1 = addMatrix(D, multiplyMatrixByNumber(L, q)); // B1 = D + q*L
        double[][] B = new double[SIZE][SIZE];
        countInverseMatrix(B1, B); // B = B1^(-1)
        double[] g = multipleMatrixByVector(multiplyMatrixByNumber(B, q), f);
        B = multiplyMatrix(B, subtractMatrix(D, multiplyMatrixByNumber(addMatrix(D, R),q))); // B =
(B1)^(-1) * (D - q(D + R))

        // finding solution
        double[] x0 = g;
        x = addVectorToVector(multipleMatrixByVector(B, x0), g);
        int iterationNumber = 0;
        while (vectorNorm(subtractVectorFromVector(x, x0)) / q > EPS) {
            x0 = x;
```

```
            x = addVectorToVector(multipleMatrixByVector(B, x0), g);
            ++iterationNumber;
        }
        return iterationNumber;
    }

    public static void main(String[] args) {
        // 1
        System.out.println("---#1 метод простых итераций---\n");
        generateMatrix(A);
        System.out.println("A:");
        printMatrix2(A);
        generateX(myX);
        countF(A, myX);
        System.out.println("f:");
        printVector10(f);

        countTransposedMatrix(A, A_TRANSPOSED);
        double[][] A1 = multiplyMatrix(A_TRANSPOSED, A);
        f = multipleMatrixByVector(A_TRANSPOSED, f);
        System.out.println("A1 = A^T * A:");
        printMatrix4(A1);
        System.out.println("f1:");
        printVector12(f);
        System.out.println("My solution:");
        printVector10(myX);

        generateE();
        double n = norm_1(A1);
        double[][] B = subtractMatrix(E, divideMatrixByNumber(A1, n)); // B = E - (A / ||A||)
        double[] g = divideVectorByNumber(f, n);
        int it1 = solveSIM(B, g);

        //2
        System.out.println("---#2 метод Зейделя---");
        int it2 = solveRelaxation(A1, 1);
        printVector10(x);
        System.out.println("Number of iterations: " + it2 + "\n");

        //3
        System.out.println("---#3 метод релаксации---");
        double qMin = 0.1;
        int itMin = solveRelaxation(A1, qMin);
        // [0,1; 1,9]
        double[] a = new double[19];
        double[] b = new double[19];
        int j=0;
        for (double q = 0.2; q<2; q+=0.1) {
            int tmp = solveRelaxation(A1, q);
            System.out.println("q = "+ round(qMin, 1));
            System.out.println("Number of iterations: " + tmp + "\n");
            if (tmp < itMin) {
                itMin = tmp;
                qMin = q;
            }
            a[j] = q;
            b[j++] = tmp;
        }
        // [qMin-0.1; qMin+0.1]
        double tmpMin = qMin;
        for (double q = tmpMin-0.1; q<tmpMin + 0.1; q+=0.01) {
            int tmp = solveRelaxation(A1, q);
            if (tmp < itMin) {
                itMin = tmp;
                qMin = q;
            }
        }
        System.out.println("q with least number of iterations:");
        int tmp = solveRelaxation(A1, qMin);
        System.out.println("q = "+ round(qMin, 2));
        System.out.println("Solution:");
        printVector10(x);
        System.out.println("Number of iterations: " + tmp + "\n");
    }
}
```

Результат

Сгенерированная матрица:

-80,61	77,97	-66,69	87,54	4,38	-15,83	-2,84	31,71	-41,89
77,97	25,01	71,52	22,25	62,38	-11,35	79,82	-72,41	-69,23
-66,69	71,52	-10,27	46,04	68,13	20,25	77,46	-24,99	-85,22
87,54	22,25	46,04	-43,14	67,79	50,55	62,58	-86,11	-29,80
4,38	62,38	68,13	67,79	47,00	41,43	-26,88	-1,25	-15,54
-15,83	-11,35	20,25	50,55	41,43	0,48	77,64	95,12	10,85
-2,84	79,82	77,46	62,58	-26,88	77,64	-41,09	-1,34	-39,12
31,71	-72,41	-24,99	-86,11	-1,25	95,12	-1,34	-5,21	81,45
-41,89	-69,23	-85,22	-29,80	-15,54	10,85	-39,12	81,45	11,50

Функция f:

928,7260337104
-7458,1370840479
-6818,9360593790
-4506,5182272031
-11560,1899245097
-16832,6118446709
-10488,1279403163
-3,0713516691
5002,8543047756

Изменённая матрица:

27726,2358	-6326,9061	18202,8416	-14331,8897	6139,1391	5980,8874	7131,1090	-19158,0057	3025,8442
-6326,9061	32742,3142	14791,0511	26663,2945	9765,0321	2192,2844	5674,7224	-9567,8509	-22664,6206
18202,8416	14791,0511	30726,5805	8475,3429	9905,1709	7907,9597	7909,1322	-16075,1102	-9539,1965
-14331,8897	26663,2945	8475,3429	31509,0704	6152,6204	-3709,3110	3206,1966	6390,1611	-18154,8531
6139,1391	9765,0321	9905,1709	6152,6204	18038,6632	3621,6928	18153,7446	-9259,1413	-11838,0676
5980,8874	2192,2844	7907,9597	-3709,3110	3621,6928	20254,9582	-947,5274	-4261,0340	2413,1922
7131,1090	5674,7224	7909,1322	3206,1966	18153,7446	-947,5274	26266,6681	-8899,8770	-11564,4692
-19158,0057	-9567,8509	-16075,1102	6390,1611	-9259,1413	-4261,0340	-8899,8770	30000,5835	9996,5565
3025,8442	-22664,6206	-9539,1965	-18154,8531	-11838,0676	2413,1922	-11564,4692	9996,5565	23353,9944

Изменённая функция f:

-560175,376236170600
-2415437,659340892500
-3099925,339732949600
-2643896,927407029600
-2267765,428470433700
-1543267,582105221200
-2169055,227975717300
-37160,119500064350
1657416,211776281200

Вектор решения:

```
  4,2818180642
  4,5585486928
-88,5718230505
-48,7145237012
-37,2786812262
-61,1504648965
-56,1069328204
-67,3960717622
-10,7214701046
```

Метод простых итераций:

```
  4,2818119089
  4,5585122537
-88,5718266715
-48,7144860080
-37,2786866631
-61,1504552808
-56,1069307322
-67,3960978532
-10,7214683696
```

N = 4455

Метод Зейделя:

```
  4,2818176458
  4,5585445458
-88,5718236433
-48,7145194478
-37,2786818501
-61,1504637842
-56,1069326785
-67,3960745417
-10,7214701818
```

N = 726

Метод релаксации:

q = 0.1 N = 5759
q = 0.2 N = 3716
q = 0.3 N = 2678
q = 0.4 N = 2046
q = 0.5 N = 1617
q = 0.6 N = 1306

q = 0.7 N = 1069
q = 0.8 N = 880
q = 0.9 N = 726
q = 1.0 N = 595
q = 1.1 N = 482
q = 1.2 N = 380
q = 1.3 N = 279
q = 1.4 N = 192
q = 1.5 N = 223
q = 1.6 N = 275
q = 1.7 N = 428
q = 1.8 N = 878
q = 1.9 N = 1275

q-оптимальное = 1.49 N = 172

Решение при q = 1.49:

```
  4,2818173301
  4,5585461514
-88,5718232028
-48,7145210103
-37,2786815982
-61,1504642563
-56,1069325788
-67,3960737951
-10,7214697576
```

График зависимости N от q: