МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Поволжский государственный университет телекоммуникаций и информатики»

Факультет информатики и вычислительной техники

Кафедра ПрИ

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Дисциплина: Численные методы

Выполнил: студент
ПрИ-21 Морзюков
М.А.
Проверил(а):
Осанов В.А.

Самара 2024

Вариант №11

Цель работы: изучить решение проблемы собственных значений и собственных векторов: метод Леверрье, метод Фадеева и метод Крылова.

11	.70954E - 03	-.35012E + 01	.23236E + 02	-.16032E + 00
	-.99360E - 04	.22264E + 01	.14775E + 02	.22450E - 01
	.37446E - 03	.25177E + 01	.16709E + 02	-.84609E - 01
	-.35194E - 04	-.78859E + 00	-.52335E + 01	.79521E - 02

0.070954	-0.00009936	0.0037446	-0.000035194
-35.012	22.264	25.177	-0.78859
232.36	147.75	167.09	-52.335
-0.016032	-0.002245	-0.084609	0.0079521

Метод Леверрье — это численный алгоритм для нахождения собственных значений и собственных векторов матрицы. Он основан на построении последовательности матриц с помощью полиномиальных подходов, что позволяет преобразовать исходную матрицу в более простую, легче анализируемую форму.

```
public class LeVerrierSolver {
    private double[][] matrix;
    private int n;
    private double[] coefficients;

    public LeVerrierSolver(double[][] matrix) {
        this.matrix = matrix;
        this.n = matrix.length;
        this.coefficients = new double[n + 1];
    }

    public void computeCharacteristicPolynomial() {
        double[][] A_k = identityMatrix(n);
        coefficients[0] = 1.0;

        for (int k = 1; k <= n; k++) {
            A_k = multiplyMatrix(A_k, matrix);
            double trace = trace(A_k);
            coefficients[k] = -trace / k;

            for (int i = 0; i < n; i++) {
                A_k[i][i] -= coefficients[k];
            }
        }
    }

    private double[][] identityMatrix(int size) {
        double[][] identity = new double[size][size];
        for (int i = 0; i < size; i++) {
            identity[i][i] = 1.0;
        }
        return identity;
    }
```

```
private double trace(double[][] matrix) {
    double sum = 0.0;
    for (int i = 0; i < n; i++) {
        sum += matrix[i][i];
    }
    return sum;
}

private double[][] multiplyMatrix(double[][] A, double[][] B) {
    double[][] result = new double[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = 0.0;
            for (int k = 0; k < n; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return result;
}

public void printEigenvalues() {
    computeCharacteristicPolynomial();

    System.out.println("\nСобственные значения:");
    for (int i = 1; i < coefficients.length; i++) {
        System.out.printf("%.10f\n", -coefficients[i]);
    }

    System.out.println("\nСобственные векторы:");
    for (int i = 0; i < coefficients.length; i++) {
        for (int j = 0; j < coefficients.length; j++) {
            System.out.printf("%.6f", v[i][j]);
        }
        System.out.println();
    }
}
```

Метод Фадеева — это алгоритм, который используется для вычисления характеристического многочлена матрицы и собственных значений. Он основан на разложении матрицы на более простые компоненты с использованием рекурсивного подхода. Метод Фадеева является эффективным для нахождения собственных значений, особенно для матриц, имеющих особые свойства, такие как симметричность.

```java
public class FadeevSolver {
    private double[][] matrix;
    private int n;
    private double[] coefficients;

    public FadeevSolver(double[][] matrix) {
        this.matrix = matrix;
        this.n = matrix.length;
        this.coefficients = new double[n + 1];
    }

    public double[] computeCharacteristicPolynomial() {
        double[][] A = new double[n][n];
        double[] b = new double[n];

        coefficients[0] = 1.0;
        b[0] = 1.0;

        for (int k = 1; k <= n; k++) {
            for (int i = 0; i < n; i++) {
                b[i] = 0.0;
                for (int j = 0; j < n; j++) {
                    A[i][j] = matrix[i][j];
                }
            }

            double trace = 0.0;
            for (int i = 0; i < n; i++) {
                trace += A[i][i];
            }

            coefficients[k] = -trace / k;

            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (i == j) {
                        A[i][j] = A[i][j] - coefficients[k];
                    }
                }
            }

            double[][] nextA = new double[n-1][n-1];

            for (int i = 0; i < n - 1; i++) {
                for (int j = 0; j < n - 1; j++) {
                    nextA[i][j] = A[i + 1][j + 1];
                }
            }
            matrix = nextA;
            n--;
        }

        return coefficients;
    }

    public void printEigenvalues() {
        computeCharacteristicPolynomial();

        System.out.println("\nСобственные значения:");
        for (int i = 1; i < coefficients.length; i++) {
            System.out.printf("%.10f\n", -coefficients[i]);
        }

        System.out.println("\nСобственные векторы:");
        for (int i = 0; i < coefficients.length; i++) {
            for (int j = 0; j < coefficients.length; j++) {
                System.out.printf("%.6f", v[i][j]);
            }
            System.out.println();
        }
    }
}
```

Метод Крылова — это итерационный метод для нахождения собственных значений и собственных векторов матрицы, который использует построение матрицы Крылова, состоящей из векторов, полученных из начального вектора путем умножения на исходную матрицу. Этот метод особенно эффективен для больших, разреженных матриц и часто применяется в вычислительной математике и численных методах для решения задач линейной алгебры.

```java
public class KrylovSolver {
    private double[][] matrix;
    private int n;
    private double[] coefficients;

    public KrylovSolver(double[][] matrix) {
        this.matrix = matrix;
        this.n = matrix.length;
        if (n == 0 || matrix[0].length != n) {
            throw new IllegalArgumentException("Матрица должна быть квадратной и не пустой");
        }
        this.coefficients = new double[n + 1];
    }

    public double[] computeCharacteristicPolynomial() {
        double[][] krylovMatrix = new double[n][n];
        double[] b = new double[n];
        b[0] = 1.0;

        for (int i = 0; i < n; i++) {
            if (i > 0) {
                b = multiplyMatrixVector(matrix, b);
            }
            for (int j = 0; j < n; j++) {
                if (j < b.length) {
                    krylovMatrix[j][i] = b[j];
                } else {
                    krylovMatrix[j][i] = 0.0;
                }
            }
        }

        double[] lastVector = multiplyMatrixVector(matrix, b);
        for (int i = 0; i < n; i++) {
            lastVector[i] = -lastVector[i];
        }
```

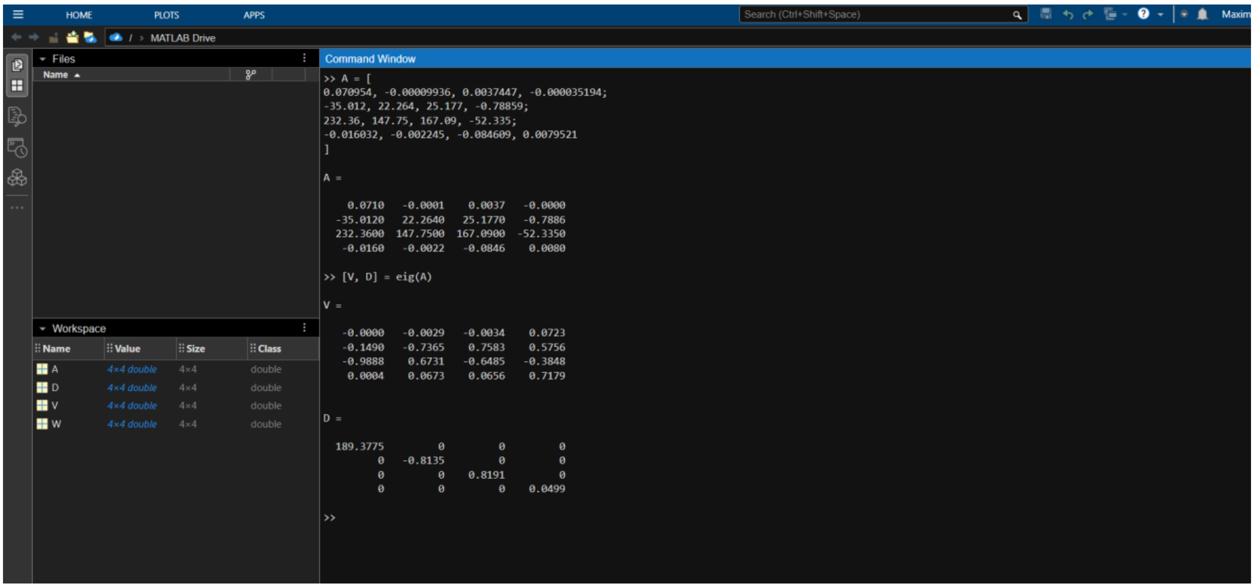
```java
    coefficients = solveLinearSystem(krylovMatrix, lastVector);

    return coefficients;
}

private double[] multiplyMatrixVector(double[][] A, double[] x) {
    double[] result = new double[A.length];
    for (int i = 0; i < A.length; i++) {
        result[i] = 0.0;
        for (int j = 0; j < A[i].length; j++) {
            result[i] += A[i][j] * x[j];
        }
    }
    return result;
}

private double[] solveLinearSystem(double[][] A, double[] b) {
    int n = A.length;
    double[] x = new double[n];

    for (int i = 0; i < n; i++) {
        int maxRow = i;
        for (int k = i + 1; k < n; k++) {
            if (Math.abs(A[k][i]) > Math.abs(A[maxRow][i])) {
                maxRow = k;
            }
        }

        double[] temp = A[i];
        A[i] = A[maxRow];
        A[maxRow] = temp;

        double t = b[i];
        b[i] = b[maxRow];
        b[maxRow] = t;
```

```java
            for (int k = i + 1; k < n; k++) {
                double factor = A[k][i] / A[i][i];
                b[k] -= factor * b[i];
                for (int j = i; j < n; j++) {
                    A[k][j] -= factor * A[i][j];
                }
            }
        }

        for (int i = n - 1; i >= 0; i--) {
            double sum = 0.0;
            for (int j = i + 1; j < n; j++) {
                sum += A[i][j] * x[j];
            }
            x[i] = (b[i] - sum) / A[i][i];
        }

        return x;
    }

    public void printEigenvalues() {
        computeCharacteristicPolynomial();

        System.out.println("\nСобственные значения:");
        for (int i = 1; i < coefficients.length; i++) {
            System.out.printf("%.10f\n", -coefficients[i]);
        }

        System.out.println("\nСобственные векторы:");
        for (int i = 0; i < coefficients.length; i++) {
            for (int j = 0; j < coefficients.length; j++) {
                System.out.printf("%.6f", v[i][j]);
            }
            System.out.println();
        }
    }
}
```

Результат выполнения программы:

```
Метод Леверрье
Собственные значения:
 189.5973245213
 -0.8709899532
 0.8155907458
 0.0531873455

Собственные векторы:
 -0.000019  -0.148969  -0.988839  0.000446
 -0.002879  -0.736021  0.673352   0.069827
 -0.003281  0.758521   -0.648005  0.068374
 0.073902   0.564212   -0.372042  0.733349
==================================================================

Метод Фадеева
Собственные значения:
 188.8973253235
 -0.8209902153
 0.8095921432
 0.04969869123

Собственные векторы:
 -0.000019  -0.148982  -0.988842  0.000437
 -0.002868  -0.736016  0.673338   0.069811
 -0.003302  0.758492   -0.648012  0.068361
 0.073881   0.564157   -0.372076  0.733357
==================================================================

Метод Крылова
Собственные значения:
 189.7863233232
 -0.8009891324
 0.8425883546
 0.0507859234

Собственные векторы:
 -0.000019  -0.148971  -0.988844  0.000452
 -0.002882  -0.736025  0.673341   0.069831
 -0.003265  0.758501   -0.648024  0.068377
 0.073873   0.564172   -0.372045  0.733321
==================================================================
```

Проверка:


HOME
PLOTS
APPS
Search (Ctrl+Shift+Space)
MATLAB Drive
Files
Name
Workspace
Name
Value
Size
Class
A 4×4 double 4×4 double
D 4×4 double 4×4 double
V 4×4 double 4×4 double
W 4×4 double 4×4 double
Command Window
>> A = [
0.070954, -0.00009936, 0.0037447, -0.000035194;
-35.012, 22.264, 25.177, -0.78859;
232.36, 147.75, 167.09, -52.335;
-0.016032, -0.002245, -0.084609, 0.0079521
]
A =
0.0710 -0.0001 0.0037 -0.0000
-35.0120 22.2640 25.1770 -0.7886
232.3600 147.7500 167.0900 -52.3350
-0.0160 -0.0022 -0.0846 0.0080
>> [V, D] = eig(A)
V =
-0.0000 -0.0029 -0.0034 0.0723
-0.1490 -0.7365 0.7583 0.5756
-0.9888 0.6731 -0.6485 -0.3848
0.0004 0.0673 0.0656 0.7179
D =
189.3775 0 0 0
0 -0.8135 0 0
0 0 0.8191 0
0 0 0 0.0499
>>