Численные методы
Лабораторная работа №4

Решение проблем собственных значений Метод Якоби(классический)

Номер 3.3.3(а)

Выполнил Сайков Константин Александрович

Группа: ПМ1801

Данная работа была выполнена на языке программирования java

код

```java
package project;

import java.util.ArrayList;

public class some_class {

    public static void main(String[] args) {
        double i[][] ={{5,1,2}, {1, 4, 1}, {2, 1, 3}};
        System.out.println("Исходная матрица");
        printmatrix(i);
        System.out.println("Допустимая погрешность 0.003\n");
        System.out.println("старт программы");
        methodIkobi(i,0.003);
    }
    public static double[][] transpose(double arr[][]){
        int m = arr.length;
        int n = arr[0].length;
        double ret[][] = new double[n][m];

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                ret[j][i] = arr[i][j];
            }
        }

        return ret;
    }
    public static double[][] multiplyByMatrix(double[][] m1, double[][] m2) {
        int m1ColLength = m1[0].length;
        int m2RowLength = m2.length;
        if(m1ColLength != m2RowLength) return null;
        int mRRowLength = m1.length;
        int mRColLength = m2[0].length;
        double[][] mResult = new double[mRRowLength][mRColLength];
        for(int i = 0; i < mRRowLength; i++) {
            for(int j = 0; j < mRColLength; j++) {
                for(int k = 0; k < m1ColLength; k++) {
                    mResult[i][j] += m1[i][k] * m2[k][j];
                }
            }
        }
        return mResult;
    }

    public static void printmatrix(double matr[][]) {
        for(int i = 0; i < matr.length;i++) {
            for(int j = 0; j < matr.length;j++) {
                System.out.print(matr[i][j]+" ");
            }
            System.out.println();
        }
    }
    public static int[] find_i_j(double matr[][]) {
        double first = matr[0][1];
        int i_j []= {0,1};
        for(int i = 0; i < matr.length; i++)
            for(int j = i+1; j<matr.length;j++ ) {
                if(first < matr[i][j]) {
                    i_j[0]=i;
                    i_j[1]=j;
                }
```

```
                }
            }

        return i_j;
    }
    public static double approximal(double matr[][]) {
        double res = 0;
        for(int i = 0; i < matr.length; i++)
            for(int j = i+1; j<matr.length;j++ ) res+=matr[i][j]*matr[i][j];
        return  (double)Math.round(Math.sqrt(res) * 10000d) / 10000d;
    }
    public static double[][] around(double matrix_res[][]) {
        for(int i = 0; i < matrix_res.length; i++)
            for(int j = 0 ; j < matrix_res.length;j++) matrix_res[i][j] = (double)Math.round(matrix_res[i][j] * 1000000d) / 1000000d;
                return matrix_res;
    }
    public static void fill_rotate_matrix(double rotate_matrix[][],int i_j[] ,double fi) {
        for(int i = 0; i < rotate_matrix.length;i++) rotate_matrix[i][i] = 1;
        rotate_matrix[i_j[0]][i_j[0]] =  (double)Math.round(Math.cos(fi) * 1000000d) / 1000000d;
        rotate_matrix[i_j[1]][i_j[1]] = (double)Math.round(Math.cos(fi) * 1000000d) / 1000000d;
        rotate_matrix[i_j[0]][i_j[1]] =  (double)Math.round(-Math.sin(fi) * 1000000d) / 1000000d;
        rotate_matrix[i_j[1]][i_j[0]] =  (double)Math.round(Math.sin(fi) * 1000000d) / 1000000d;

    }
    public static void methodIkobi(double matr[][],double eps) {
        ArrayList<double [][]> matrV = new ArrayList();
        double error=eps+1;
        for(int i = 0; error > eps ; i++) {
        System.out.println(i+" шаг");
        int found_i_j[] = find_i_j(matr);
        System.out.println("максимальный элемент: " + matr[found_i_j[0]][found_i_j[1]]);
        double rotate_matrix[][] = new double [matr.length][matr.length];
        double fi = 0.5*Math.atan(2*matr[found_i_j[0]][found_i_j[1]]/(matr[found_i_j[0]][found_i_j[0]]-matr[found_i_j[1]][found_i_j[1]]));
        fill_rotate_matrix(rotate_matrix,found_i_j,fi);
        matrV.add(rotate_matrix);
        System.out.println("матрица вращения: ");
        printmatrix(rotate_matrix);
        System.out.println();


        matr = around(multiplyByMatrix(multiplyByMatrix(transpose(rotate_matrix),matr),rotate_matrix));
        System.out.println("матрица после перемножения: ");
        printmatrix(matr);
        System.out.println("погрешность " + approximal(matr));
        error = approximal(matr);
        System.out.println();
        //}while(error < eps);
        }
        System.out.println("результат вычислений: ");
        printmatrix(matr);
        System.out.println("собственные значения: ");
        System.out.println("");
        for(int i = 0; i < matr.length;i++) System.out.println("собственное число матрицы лямба с индексом "+i+" = "+matr[i][i]);
        System.out.println();

        System.out.println("После того,как были найдены все значения находим собственные вектора путем перемножения всех матриц вращения, которые получились(сохраняются в список matrV)");
        double res_Vector[][] = new double[matr.length][matr.length];
        res_Vector = matrV.get(0);
        for(int i = 1; i < matrV.size();i++) res_Vector = multiplyByMatrix(res_Vector,matrV.get(i));
        printmatrix(res_Vector);
```

Результат работы программы:

```
Исходная матрица
5.0 1.0 2.0
1.0 4.0 1.0
2.0 1.0 3.0
Допустимая погрешность 0.003

старт программы
0 шаг
максимальный элемент: 2.0
матрица вращения:
0.850651 0.0 -0.525731
0.0 1.0 0.0
0.525731 0.0 0.850651

матрица после перемножения:
6.236069 1.376382 1.0E-6
1.376382 4.0 0.32492
1.0E-6 0.32492 1.763932
погрешность 1.4142

1 шаг
максимальный элемент: 1.376382
матрица вращения:
0.902912 -0.429826 0.0
0.429826 0.902912 0.0
0.0 0.0 1.0

матрица после перемножения:
6.891291 1.0E-6 0.13966
1.0E-6 3.344782 0.293374
0.13966 0.293374 1.763932
погрешность 0.3249

2 шаг
максимальный элемент: 0.293374
матрица вращения:
1.0 0.0 0.0
0.0 0.984253 -0.176766
0.0 0.176766 0.984253

матрица после перемножения:
6.891291 0.024688 0.137461
0.024688 3.397471 0.0
0.137461 0.0 1.711244
погрешность 0.1397

3 шаг
максимальный элемент: 0.137461
матрица вращения:
0.999649 0.0 -0.026509
0.0 1.0 0.0
0.026509 0.0 0.999649

матрица после перемножения:
6.894942 0.024679 -2.0E-6
0.024679 3.397471 -6.54E-4
-2.0E-6 -6.54E-4 1.7076
погрешность 0.0247
```

```
4 шаг
максимальный элемент: 0.024679
матрица вращения:
0.999975 -0.007056 0.0
0.007056 0.999975 0.0
0.0 0.0 1.0

матрица после перемножения:
6.895115 -1.0E-6 -7.0E-6
-1.0E-6 3.397296 -6.54E-4
-7.0E-6 -6.54E-4 1.7076
погрешность 7.0E-4

результат вычислений:
6.895115 -1.0E-6 -7.0E-6
-1.0E-6 3.397296 -6.54E-4
-7.0E-6 -6.54E-4 1.7076
собственные значения:

собственное число матрицы лямба с индексом 0 = 6.895115
собственное число матрицы лямба с индексом 1 = 3.397296
собственное число матрицы лямба с индексом 2 = 1.7076

После того,как были найдены все значения находим собственные вектора путем перемножения всех матриц вращения, которые получились(сохраняются в список matrV)
0.752575681518312 -0.45812720796528394 -0.4730226643254259
0.43170417252186444 0.8856696932215234 -0.17094237897195022
0.4972550936811955 -0.07555878924615386 0.8643086926021226
```

Результатом данной программы является нахождение собственных значений матрицы и собственных векторов (без нормализации)

Сравниваем результаты с функцией в вольфраме

```
In[21]:= N@Eigenvalues[{{5, 1, 2}, {1, 4, 1}, {2, 1, 3}}] // MatrixForm

Out[21]//MatrixForm=
```

$$\begin{pmatrix} 6.89511 \\ 3.3973 \\ 1.7076 \end{pmatrix}$$

Как можно заметить, собственные значения матрицы получились почти идентичными