**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Крагель Алина Олеговна

**Итерационный степенной метод**

Отчет по лабораторной работе №4

(«Вычислительные методы алгебры»)

Студента 2 курса 10 группы

Работа сдана \_\_\_декабря 2020 г. Преподаватель:

Зачтена \_\_\_\_\_\_\_\_\_ 2020 г. Горбачева Юлия Николаевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ассистент кафедры

(Подпись преподавателя) вычислительной математики

Минск 2020

**Постановка задачи**

Итерационным степенным методом и методом скалярных произведений с точностью найти наибольшее по модулю собственное значение матрицы *A* и соответствующий собственный вектор. Вычислительный процесс проводить с нормировкой векторов итерационной последовательности.

Матрица *A* определяется формулой , где k – номер варианта. В лабораторной решается вариант №9.

*B*=, *C*=, k = 9.

Исходная матрица *A*:

*A*=.

**Теория**

Итерационный степенной метод (называемый также степенным методом) предназначен для нахождения одного или нескольких собственных значений и соответствующих собственных векторов.

Пусть A – вещественная матрица порядка n. Мы рассмотрим степенной метод для случая диагонализируемых матриц (матриц простой структуры). Матрица заведомо диагонализируема в двух важных частных случаях: если она симметричная или если ее собственные значения различны. Диагонализируемая матрица имеет ровно n линейно независимых собственных векторов.

***Алгоритм степенного метода с пошаговой нормировкой векторов***:

1. Выбрать произвольный начальный ненулевой вектор . Вычислить и вектор . Положить .

2) Вычислить . Находим , . Вычисляем и .

3) Если , процесс завершить. За приближенное собственное значение можно взять значение при любом или среднее арифметическое этих значений. Вектор приближенно представляет собой нормированный собственный вектор , соответствующий собственному значению . Если , положить и перейти к п.2.

***Алгоритм метода скалярных произведений с пошаговой нормировкой векторов***:

1. Выбрать произвольный начальный ненулевой вектор . Вычислить и вектор . Положить .

2) Вычислить . Находим . Вычисляем и .

3) Если , процесс завершить. За приближенное собственное значение можно взять значение . Вектор приближенно представляет собой нормированный собственный вектор , соответствующий собственному значению . Если , положить и перейти к п.2.

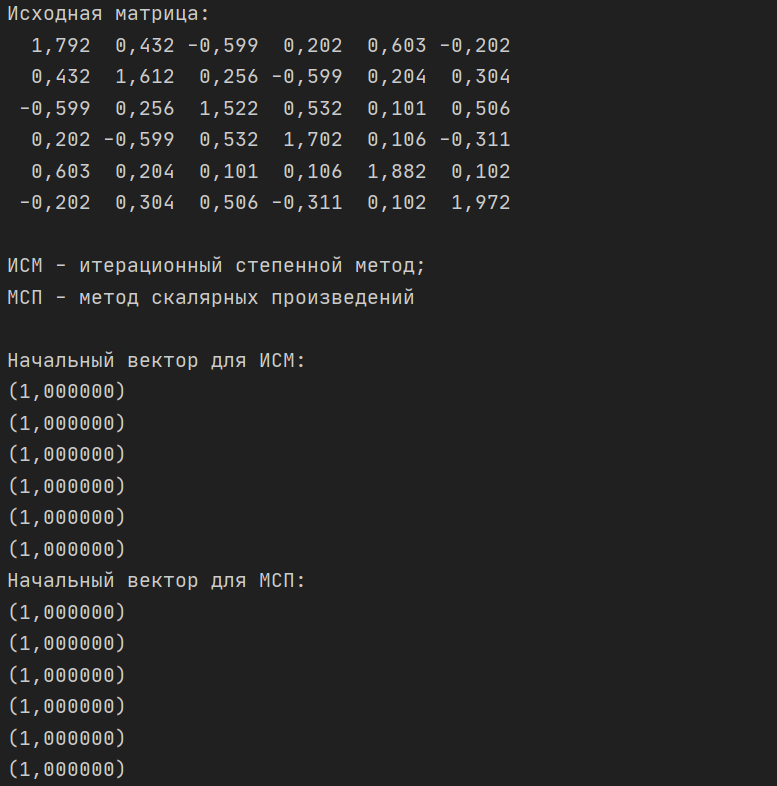
**Листинг**

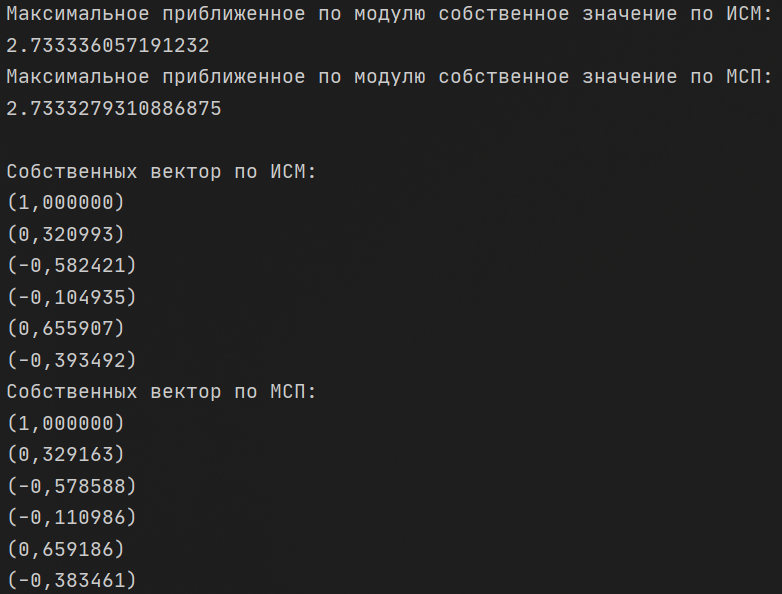
import java.util.\*;  
import java.lang.\*;  
  
public class iterativePowerMethod {  
 *//Максимальное значение вектора* private static double vectorNorm (double[] y0){  
 double yNorm = Math.abs(y0[0]);  
 for (int i = 1; i < y0.length;i++) {  
 if (yNorm < Math.abs(y0[i])) {  
 yNorm = Math.abs(y0[i]);  
 }  
 }  
 return yNorm;  
 }  
 *//Норма разности векторов* private static double vectorsDifferenceNorm (double[] y1, double[] y2){  
 double yNorm = Math.abs(y1[0] - y2[0]);  
 for (int i = 1; i < y1.length;i++) {  
 if (yNorm < Math.abs(y1[i] - y2[i]))  
 yNorm = Math.abs(y1[i] - y2[i]);  
 }  
 return yNorm;  
 }  
 *//Деление вектора на скаляр* private static double[] divisionVectorScalar (double [] y, double n){  
 double[] yN = new double[y.length];  
 for (int i = 0; i < y.length; i++)  
 yN[i] = y[i] / n;  
 return yN;  
 }  
 *//Умножение мтарицы на вектор* private static double[] multiplicationMatrixVector (double[][] A, double[] y){  
 double[] y1 = new double[y.length];  
 for (int i = 0; i < y1.length; i++){  
 for (int j = 0; j < y1.length; j++)  
 y1[i] += A[i][j] \* y[j];  
 }  
 return y1;  
 }  
 *//Максимальное значение вектора* private static double[] divisionVectors (double[] y1, double[] y2){  
 double[] y = new double[y1.length];  
 for (int i = 0; i < y1.length; i++)  
 y[i] = y1[i] / y2[i];  
 return y;  
 }  
 *//Среднее значение координат векторов* private static double averageEigenValue (double[] y){  
 double sum = 0;  
 for (int i = 0; i < y.length; i++)  
 sum += y[i];  
 return Math.abs(sum / y.length);  
 }  
 *//Максимальное собственное значение матрицы* public static double maxEigenValue(double[][] A, double eps, double[] y0){  
 double[] lBefore = new double[y0.length]; *//Вектор собственных значений на k-ой итерации* double[] lAfter = new double[y0.length]; *//Вектор собственных значений k+1 -ой итерации* double[] yBefore = Arrays.copyOf(y0, y0.length); *//Вектор значений на k-ой итерации* double[] yNormBefore = divisionVectorScalar(yBefore, vectorNorm(yBefore)); *//Вектор нормированных значений на k-ой итерации* double[] yAfter; *//Вектор значений на k+1 -ой итерации* double delta = 1.0; *//Условие выхода из цикла (норма разности векторов собственных значений на k-ой и k+1 -ой итерациях* while (delta > eps) { *//Пока не достигнута заданная точность* yAfter = multiplicationMatrixVector(A, yNormBefore);  
  
 lAfter = divisionVectors(yAfter, yNormBefore);  
  
 double norm = vectorNorm(yAfter); *//норма вектора значений на k+1 -ой итерации* yNormBefore = divisionVectorScalar(yAfter, norm);  
  
 delta = vectorsDifferenceNorm(lAfter, lBefore);  
  
 lBefore = Arrays.copyOf(lAfter, lAfter.length);  
 }  
  
 return averageEigenValue(lAfter);  
 }  
 *//Собственный вектор, соответствующий максимальному собственному значению  
 //Смысл переменных такой же, как в методе выше, отличие в возвращаемом значении* public static double[] vectorEigenValues(double[][] A, double eps, double[] y0){  
 double[] lBefore = new double[y0.length];  
 double[] lAfter = new double[y0.length];  
  
 double[] yBefore = Arrays.copyOf(y0, y0.length);  
 double[] yNormBefore = divisionVectorScalar(yBefore, vectorNorm(yBefore));  
  
 double[] yAfter;  
 double delta = 1.0;  
  
 while (delta > eps) {  
 yAfter = multiplicationMatrixVector(A, yNormBefore);  
  
 lAfter = divisionVectors(yAfter, yNormBefore);  
  
 double norm = vectorNorm(yAfter);  
 yNormBefore = divisionVectorScalar(yAfter, norm);  
  
 delta = vectorsDifferenceNorm(lAfter, lBefore);  
  
 lBefore = Arrays.copyOf(lAfter, lAfter.length);  
 }  
  
 return yNormBefore;  
 }  
 *//Количество итераций метода  
 //Смысл переменных такой же, как в методе выше, отличие в возвращаемом значении* public static int iterationsNum(double[][] A, double eps, double[] y0){  
 double[] lBefore = new double[y0.length];  
 double[] lAfter = new double[y0.length];  
  
 double[] yBefore = Arrays.copyOf(y0, y0.length);  
 double[] yNormBefore = divisionVectorScalar(yBefore, vectorNorm(yBefore));  
  
 double[] yAfter;  
 double delta = 1.0;  
  
 int k = 0;  
  
 while (delta > eps) {  
 yAfter = multiplicationMatrixVector(A, yNormBefore);  
  
 lAfter = divisionVectors(yAfter, yNormBefore);  
  
 double norm = vectorNorm(yAfter);  
 yNormBefore = divisionVectorScalar(yAfter, norm);  
  
 delta = vectorsDifferenceNorm(lAfter, lBefore);  
  
 lBefore = Arrays.copyOf(lAfter, lAfter.length);  
  
 k++;  
 }  
  
 return k;  
 }  
}

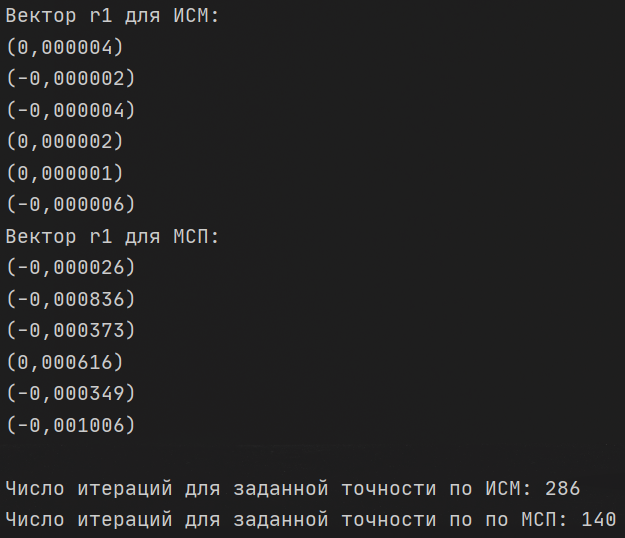
import java.util.\*;  
  
public class dotProductMethod {  
 *//Максимальное значение в векторе* private static double vectorNorm (double[] y0){  
 double yNorm = Math.abs(y0[0]);  
 for (int i = 0; i < y0.length;i++) {  
 if (yNorm < Math.abs(y0[i])) {  
 yNorm = Math.abs(y0[i]);  
 }  
 }  
 return yNorm;  
 }  
 *//Деление вектора на скаляр* private static double[] divisionVectorScalar (double [] y, double n){  
 double[] yN = new double[y.length];  
 for (int i = 0; i < y.length; i++)  
 yN[i] = y[i]/n;  
 return yN;  
 }  
 *//Умножение матрицы на вектор* private static double[] multiplicationMatrixVector (double[][] A, double[] y){  
 double[] y1 = new double[y.length];  
 for (int i = 0; i < y1.length; i++){  
 for (int j = 0; j < y1.length; j++)  
 y1[i] += A[i][j] \* y[j];  
 }  
 return y1;  
 }  
 *//Скалярное произведение векторов* private static double scalarMultiplicationVectors (double[] y1, double[] y2){  
 double y = 0;  
 for (int i = 0; i < y1.length; i++)  
 y += y1[i] \* y2[i];  
 return y;  
 }  
 *//Максимальное собственное значение* public static double maxEigenValue(double[][] A, double eps, double[] y0){  
 double lBefore = 0;  
 double lAfter = 0;  
  
 double[] yBefore = Arrays.copyOf(y0, y0.length);  
 double[] yNormBefore = divisionVectorScalar(yBefore, vectorNorm(yBefore));  
  
 double[] yAfter;  
 double delta = 1.0;  
  
 while (delta > eps) {  
 yAfter = multiplicationMatrixVector(A, yNormBefore);  
  
 lAfter = scalarMultiplicationVectors(yAfter, yNormBefore) / scalarMultiplicationVectors(yNormBefore, yNormBefore);  
  
 double norm = vectorNorm(yAfter);  
 yNormBefore = divisionVectorScalar(yAfter, norm);  
  
 delta = Math.abs(lAfter - lBefore);  
  
 lBefore = lAfter;  
 }  
  
 return Math.abs(lAfter);  
 }  
 *//Собственный вектор, соответствующий максимальному собственному значению  
 //Смысл переменных такой же, как в методе выше, отличие в возвращаемом значении* public static double[] vectorEigenValues(double[][] A, double eps, double[] y0){  
 double lBefore = 0;  
 double lAfter = 0;  
  
 double[] yBefore = Arrays.copyOf(y0, y0.length);  
 double[] yNormBefore = divisionVectorScalar(yBefore, vectorNorm(yBefore));  
  
 double[] yAfter;  
 double delta = 1.0;  
  
 while (delta > eps) {  
 yAfter = multiplicationMatrixVector(A, yNormBefore);  
  
 lAfter = scalarMultiplicationVectors(yAfter, yNormBefore) / scalarMultiplicationVectors(yNormBefore, yNormBefore);  
  
 double norm = vectorNorm(yAfter);  
 yNormBefore = divisionVectorScalar(yAfter, norm);  
  
 delta = Math.abs(lAfter - lBefore);  
  
 lBefore = lAfter;  
 }  
  
 return yNormBefore;  
 }  
 *//Количество итераций метода  
 //Смысл переменных такой же, как в методе выше, отличие в возвращаемом значении* public static int iterationsNum(double[][] A, double eps, double[] y0){  
 double lBefore = 0;  
 double lAfter = 0;  
  
 double[] yBefore = Arrays.copyOf(y0, y0.length);  
 double[] yNormBefore = divisionVectorScalar(yBefore, vectorNorm(yBefore));  
  
 double[] yAfter;  
 double delta = 1.0;  
  
 int k = 0;  
  
 while (delta > eps) {  
 yAfter = multiplicationMatrixVector(A, yNormBefore);  
  
 lAfter = scalarMultiplicationVectors(yAfter, yNormBefore) / scalarMultiplicationVectors(yNormBefore, yNormBefore);  
  
 double norm = vectorNorm(yAfter);  
 yNormBefore = divisionVectorScalar(yAfter, norm);  
  
 delta = Math.abs(lAfter - lBefore);  
  
 lBefore = lAfter;  
  
 k++;  
 }  
  
 return k;  
 }  
}

public class Test {  
  
 public static double eps = 1e-6;  
 public static double[][] B = {{1.342, 0.432, -0.599, 0.202, 0.603, -0.202},  
 {0.432, 1.342, 0.256, -0.599, 0.204, 0.304},  
 {-0.599, 0.256, 1.342, 0.532, 0.101, 0.506},  
 {0.202, -0.599, 0.532, 1.342, 0.106, -0.311},  
 {0.603, 0.204, 0.101, 0.106, 1.342, 0.102},  
 {-0.202, 0.304, 0.506, -0.311, 0.102, 1.342}};  
 public static int k = 9;  
 public static double[] C = {0.05, 0.03, 0.02, 0.04, 0.06, 0.07};  
 *//Получение матрицы А* public static double[][] getA (double[][] gB, int gK, double[] gC){  
 double[][] gA = new double[gB.length][gB.length];  
 for (int i = 0; i < gB.length; i++){  
 System.arraycopy(gB[i], 0, gA[i], 0, gB[i].length);  
 gA[i][i] = gB[i][i] + gK \* gC[i];  
 }  
 return gA;  
 }  
 *//Печать матрицы* public static void matrixPrinting(double[][] M, int m, int p) {  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < p; j++) {  
 System.out.printf("%7.3f", M[i][j]);  
 }  
 System.out.print("\n");  
 }  
 System.out.print("\n");  
 }  
 *//Печать вектора* public static void vectorPrinting(double[] M) {  
 for (int i = 0; i < M.length; i++) {  
 System.out.printf("(%f)\n", M[i]);  
 }  
 }  
 *//Получение вектора начальных значений* public static double[] getY0 (int n){  
 double y[] = new double[n];  
 for (int i = 0; i < n; i++)  
 y[i] = 1;  
 return y;  
 }  
 *//Умножение матрицы на вектор* public static double[] multiplicationMatrixVector (double[][] A, double[] y){  
 double[] y1 = new double[y.length];  
 for (int i = 0; i < y1.length; i++){  
 for (int j = 0; j < y1.length; j++)  
 y1[i] += A[i][j] \* y[j];  
 }  
 return y1;  
 }  
 *//Умножение вектора на скаляр* private static double[] multiplicationVectorScalar(double [] y, double n){  
 double[] yN = new double[y.length];  
 for (int i = 0; i < y.length; i++)  
 yN[i] = y[i] \* n;  
 return yN;  
 }  
 *//Вычисление невязки (вектора r1)* private static double[] vectorR1 (double[][] A, double l, double[] x){  
 double[] c = new double[x.length];  
 double[] ax = multiplicationMatrixVector(A, x);  
 double[] lx = multiplicationVectorScalar(x, l);  
 for (int i = 0; i < x.length; i++)  
 c[i] = ax[i] - lx[i];  
 return c;  
 }  
  
 public static void main(String[] args){  
 double[][] A = getA(B, k, C);  
 System.out.println("Исходная матрица: ");  
 matrixPrinting(A, A.length, A.length);  
  
 System.out.println("ИСМ - итерационный степенной метод;\nМСП - метод скалярных произведений\n");  
  
 double[] yIPM = getY0(A.length);  
 double[] yDPM = getY0(A.length);  
 System.out.println("Начальный вектор для ИСМ: ");  
 vectorPrinting(yIPM);  
 System.out.print("Начальный вектор для МСП: \n");  
 vectorPrinting(yDPM);  
  
 double maxEigenIPM = iterativePowerMethod.maxEigenValue(A, eps, yIPM);  
 double maxEigenDPM = dotProductMethod.maxEigenValue(A, eps, yDPM);  
 System.out.println("\nМаксимальное приближенное по модулю собственное значение по ИСМ: \n" + maxEigenIPM +  
 "\nМаксимальное приближенное по модулю собственное значение по МСП: \n" + maxEigenDPM + "\n");  
  
 double[] eigenvaluesIPM = iterativePowerMethod.vectorEigenValues(A, eps, yIPM);  
 double[] eigenvaluesDPM = dotProductMethod.vectorEigenValues(A, eps, yDPM);  
 System.out.println("Собственных вектор по ИСМ: ");  
 vectorPrinting(eigenvaluesIPM);  
 System.out.print("Собственных вектор по МСП: \n");  
 vectorPrinting(eigenvaluesDPM);  
  
 double[] r1IPM = vectorR1(A, maxEigenIPM, eigenvaluesIPM);  
 double[] r1DMP = vectorR1(A, maxEigenDPM, eigenvaluesDPM);  
 System.out.println("\nВектор r1 для ИСМ: ");  
 vectorPrinting(r1IPM);  
 System.out.print("Вектор r1 для МСП: \n");  
 vectorPrinting(r1DMP);  
  
 int iterationsIPM = iterativePowerMethod.iterationsNum(A, eps, yIPM);  
 int iterationsDPM = dotProductMethod.iterationsNum(A, eps, yDPM);  
 System.out.println("\nЧисло итераций для заданной точности по ИСМ: " + iterationsIPM +  
 "\nЧисло итераций для заданной точности по по МСП: " + iterationsDPM);  
 }  
}

**Результаты**

****

****

****

**Вывод**

Таким образом, в результате поставленного эксперимента удалось выяснить, что использованные итерационный степенной метод и метод скалярных произведений позволяют найти одно (или несколько) собственное(-ых) значение(-й) и соответствующих собственных векторов матрицы.