МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Лабораторная работа № 7

Определение собственных векторов матрицы методом

Крылова

Вариант №15

ОТЧЕТ

по лабораторной работе

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Суркова А.С.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сапожников В.О.

19-ИВТ-3

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

**Содержание**

1. Цель……………………………………………………………………………..3
2. Постановка задачи……………………………………………………………..4
3. Теоретические сведения……………………………………………………….5
4. Расчётные данные…………………...………………………………………..12
5. Листинг разработанной программы…………………………………………14
6. Результаты работы программы………………………………………………24
7. Вывод………………………………………………………………………….27
8. **Цель**

Закрепление знаний и умений определения собственных числе и векторов матрицы методом Крылова.

1. **Постановка задачи**

Используя метод Крылова, найти собственные числа и собственные векторы матрицы. Собственные числа определить с четырьмя верными цифрами, а собственные векторы – с тремя десятичными знаками.

1. **Теоретические сведения**

Пусть

- характеристический полином (с точность до знака) матрицы A. Согласно тождеству Гамильтона-Кели, матрица А обращает в нуль свой характеристический полином; поэтому

Возьмем теперь произвольный ненулевой вектор

Умножая обе части равенства справа на , получим:

Положим:

Тогда равенство приобретает вид:

или

где

Следовательно, векторное равенство эквивалентно системе уравнений:

из которой, вообще говоря, можно определить неизвестный коэффициенты

Так как на основании формулы

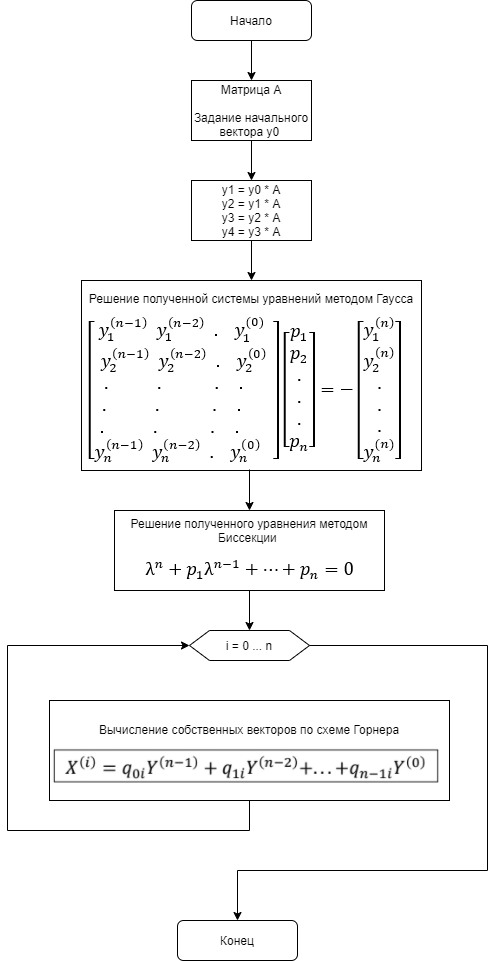
то координаты вектора последовательно вычисляются по формулам:

таким образом, определение коэффициентов характеристического полинома методом А.Н Крылова сводится к решению линейной системы уравнений, коэффициенты которой вычисляются по формулам, причем координаты начального вектора

произвольны. Если система имеет единственное решение, то ее корни являются коэффициентами характеристического полинома. Это решение может быть найдено, например методом Гаусса. Если система не имеет единственного решения, то задача усложняется. В этом случае рекомендуется изменить начальный вектор.

**Определение собственных векторов:**

Таким образом, если , то полученная линейная комбинация векторов дает собственный вектор с точностью до числового множителя. Коэффициенты могут быть легко определены по схеме *Горнера*



1. **Расчетные данные  
   Значения, полученные другим способом:**

Собственные числа

|  |  |
| --- | --- |
| λ1 | 4,50527 |
| λ2 | 1,40817 |
| λ3 | -1,39636 |
| λ4 | 0,582923 |

Собственные вектора

|  |  |
| --- | --- |
| u1 | (1.05279, 1.16478, 0.90948, 1) |
| u2 | (-0.941398, 1.34812, -1.73635, 1) |
| u3 | (-7.84415, 1.53217, 6.01838, 1) |
| u4 | (-0.0453245, -0.800903, -0.0213372, 1) |

**Значения, полученные в ходе лабораторной работы:**

Собственные числа

|  |  |
| --- | --- |
| λ1 | -1,3964 |
| λ2 | 0,5830 |
| λ3 | 1,4082 |
| λ4 | 4,5053 |

Собственные вектора

|  |  |
| --- | --- |
| v1 | (3.8944, -0.7607, -2.9888, -0.4965) ≈ u3 / -0.4965 |
| v2 | (0.1415, 2.4994, 0.0668, -3.1212) ≈ u4 / -3.1212 |
| v3 | (1.3541, -1.9391, 2.4975, -1.4383) ≈ u2 / -1.4383 |
| v4 | (20.4822, 22.6613, 17.6940, 19.4552) ≈ u1 / 19.4552 |

1. **Листинг разработанной программы**

**Main.java**

|  |
| --- |
| import equation.FourthDegreePolynomial; |
| import equation\_solution.BisectionSolution; |
| import equation\_system.SystemOfEquations; |
| import equation\_system.SystemOfFourEquations; |
| import equation\_system\_solution.GaussSolution; |
| import matrix.Matrix; |
| import matrix.Matrix4x4; |
|  |
| import java.util.List; |
|  |
| /\*\* |
| \* Класс, содержащий точку входа в программу - метод main. |
| \* Язык: java |
| \* |
| \* Реализация седьмой лабораторной работы |
| \* по дисциплине: Вычислительная математика |
| \* Вариант №15 |
| \* |
| \* Текст задания: |
| \* Используя метод Крылова, найти собственные числа и собственные |
| \* векторы матрицы. Собственные числа определить с четырьмя верными |
| \* цифрами, а собственные векторы – с тремя десятичными знаками. |
| \* |
| \* Исходная матрица: |
| \* 0.5 1.2 2 1 |
| \* 1.2 2 0.5 1.2 |
| \* 2 0.5 1 0.5 |
| \* 1 1.2 0.5 1.6 |
| \* |
| \* @release: 10.05.21 |
| \* @last\_update: 10.05.21 |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \*/ |
| public class Main |
| { |
| public static void main(String[] args) |
| { |
| double[] y0, y1, y2, y3, y4; |
|  |
| //задаем вектор y0 |
| y0 = new double[]{0.0, 1.0, 0.0, 0.0}; |
|  |
| //создаем объект матрица, передаем туда исходные данные |
| Matrix matrix2 = new Matrix4x4(new double[][] |
| {{0.5, 1.2, 2.0, 1.0}, |
| {1.2, 2.0, 0.5, 1.2}, |
| {2.0, 0.5, 1.0, 0.5}, |
| {1.0, 1.2, 0.5, 1.6}}); |
|  |
| System.out.println("Исходная матрица:"); |
| System.out.println(matrix2); |
| System.out.println(); |
|  |
|  |
| //вывод вектор y0 |
| System.out.println("Вектор y0:"); |
| printVector(y0); |
| System.out.println(); |
|  |
| //вычсиление вектор y1 и его вывод |
| y1 = matrix2.multiplyByVector(y0); |
| System.out.println("Вектор y1:"); |
| printVector(y1); |
| System.out.println(); |
|  |
| //вычсиление вектор y2 и его вывод |
| y2 = matrix2.multiplyByVector(y1); |
| System.out.println("Вектор y2:"); |
| printVector(y2); |
| System.out.println(); |
|  |
| //вычсиление вектор y3 и его вывод |
| y3 = matrix2.multiplyByVector(y2); |
| System.out.println("Вектор y3:"); |
| printVector(y3); |
| System.out.println(); |
|  |
| //вычсиление вектор y4 и его вывод |
| y4 = matrix2.multiplyByVector(y3); |
| System.out.println("Вектор y4:"); |
| printVector(y4); |
| System.out.println(); |
|  |
|  |
| //Создание объекта - система уравнений, на основе полученных векторов |
| System.out.println("Полученная система уравнений:"); |
| SystemOfEquations system = new SystemOfFourEquations( |
| new double[][]{{y3[0], y2[0], y1[0], y0[0]}, |
| {y3[1], y2[1], y1[1], y0[1]}, |
| {y3[2], y2[2], y1[2], y0[2]}, |
| {y3[3], y2[3], y1[3], y0[3]}}, |
|  |
| new double[]{-y4[0], -y4[1], -y4[2], -y4[3]} |
| ); |
| system.printSystem(); |
| System.out.println(); |
|  |
|  |
| //нахождение корней системы методом Гаусса |
| System.out.println("Решение системы уравнений при помощи метода Гаусса:"); |
| double[] p = new GaussSolution().getSolution(system); |
| printVector(p); |
|  |
|  |
| //Создаем объект уравнение на основе полученных |
| //решений системы |
| System.out.println(); |
| System.out.println("Полученное уравнение P(λ):"); |
| FourthDegreePolynomial equation = new FourthDegreePolynomial(); |
| equation.setCoefficients(p); |
| equation.printEquation(); |
|  |
| //Получение собственных чисел - решений полученного уравнения |
| System.out.println(); |
| System.out.println(); |
| List<Double> res = new BisectionSolution().getSolution(equation); |
| System.out.println("Решения уравнения: "); |
| System.out.printf("λ1 = %.4f\n", res.get(0)); |
| System.out.printf("λ2 = %.4f\n", res.get(1)); |
| System.out.printf("λ3 = %.4f\n", res.get(2)); |
| System.out.printf("λ4 = %.4f\n", res.get(3)); |
| System.out.println(); |
| System.out.println(); |
|  |
|  |
| //На основе полученных собственных чисел |
| //векторов p и векторов y0-y4 чеез схему Горнера |
| System.out.println("Нахождение собсвтенных векторов:"); |
| double[][] ownVectors = new double[4][4]; |
| for (int k = 0; k < 4; k++) |
| { |
| double[] q = new double[5]; |
| q[0] = 1.0; |
|  |
| //объединяем все значения y0-y4 в один массив |
| double[][] tempArr = new double[4][4]; |
| System.arraycopy(y0, 0, tempArr[0], 0, y0.length); |
| System.arraycopy(y1, 0, tempArr[1], 0, y1.length); |
| System.arraycopy(y2, 0, tempArr[2], 0, y2.length); |
| System.arraycopy(y3, 0, tempArr[3], 0, y3.length); |
|  |
| //вычисление значения qi |
| for (int j = 1, i = 0; i < p.length; i++, j++) |
| { |
| q[j] = res.get(k) \* q[j - 1] + p[i]; |
| } |
|  |
| //умножаем вектор Y3 на q0 |
| for (int j = 0; j < y3.length; j++) |
| { |
| tempArr[3][j] \*= q[0]; |
| } |
|  |
| //умножаем вектор Y2 на q1 |
| for (int j = 0; j < y3.length; j++) |
| { |
| tempArr[2][j] \*= q[1]; |
| } |
|  |
| //умножаем вектор Y1 на q2 |
| for (int j = 0; j < y3.length; j++) |
| { |
| tempArr[1][j] \*= q[2]; |
| } |
|  |
| //умножаем вектор Y0 на q3 |
| for (int j = 0; j < y3.length; j++) |
| { |
| tempArr[0][j] \*= q[3]; |
| } |
|  |
| //Получаем собтсвенный вектор путем сложения |
| //произвдеений qi на вектор Yj |
| ownVectors[k][0] = tempArr[0][0] + tempArr[1][0] + tempArr[2][0]  + tempArr[3][0]; |
| ownVectors[k][1] = tempArr[0][1] + tempArr[1][1] + tempArr[2][1]  + tempArr[3][1]; |
| ownVectors[k][2] = tempArr[0][2] + tempArr[1][2] + tempArr[2][2]  + tempArr[3][2]; |
| ownVectors[k][3] = tempArr[0][3] + tempArr[1][3] + tempArr[2][3]  + tempArr[3][3]; |
|  |
| System.out.println("Собственный вектор V" + (k + 1) + ": "); |
| printVector(ownVectors[k]); |
| System.out.println(); |
| } |
| } |
|  |
| /\*\* |
| \* Вспомогательный метод для вывода вектора. |
| \* |
| \* @param vector - вектор, который необходимо вывести |
| \* \*/ |
| public static void printVector(double[] vector) |
| { |
| for (double v : vector) |
| { |
| if (v < 0.0) |
| { |
| System.out.printf("%.4f\n", v); |
| } |
| else |
| { |
| System.out.printf(" %.4f\n", v); |
| } |
| } |
| } |
| } |

**equation/Equation.java**

|  |
| --- |
| package equation; |
|  |
| import java.util.List; |
|  |
| /\*\* |
| \* Интерфейс реализующий основные методы уравнений любого вида. |
| \* |
| \* Общий вид уравнения: |
| \* a\*x^n + b\*x^(n-1) + c\*x^(n-1) + ... + d\*x^0 = 0 |
| \* |
| \* Содержит 4 метода необходимых для данной лабораторной работы: |
| \* - задание коэффициентов уравнения |
| \* - получение значения функции в точке |
| \* - получение интервалов монотонности |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \* @see FourthDegreePolynomial |
| \* \*/ |
| public interface Equation |
| { |
| /\*\* |
| \* Метод для задания коэффициентов |
| \* |
| \* @param coefficients - массив коэффициентов при членах уравнения. |
| \* \*/ |
| void setCoefficients(double[] coefficients); |
|  |
| /\*\* |
| \* Метод для получения значения уравнения в заданной точке. |
| \* |
| \* @param x - точка, в которой необходимо получить значение функции. |
| \* @return значение функции в данной точке |
| \* \*/ |
| double getValueAtX(double x); |
|  |
| /\*\* |
| \* Метод для получения списка интервалов, где функция меняет знак. |
| \* |
| \* @return список из чисел, которые составляют отрезки монотонности |
| \* типа [i; i+1] |
| \* \*/ |
| List<List<Double>> getIntervals(); |
|  |
| /\*\* |
| \* Метод вывода уравнения в консоль |
| \* \*/ |
| void printEquation(); |
| } |

**equation/FourthDegreePolymonial.java**

|  |
| --- |
| package equation; |
|  |
| import java.util.\*; |
|  |
| /\*\* |
| \* Класс уравнений четвертой степени. |
| \* Реализует интерфейс Equation |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \* @see Equation |
| \* \*/ |
| public class FourthDegreePolynomial implements Equation |
| { |
| //переменная для хранения коэффициентов уравнения |
| private final double[] coefficients = new double[4]; |
|  |
| /\*\* |
| \* Метод для задания коэффициентов. |
| \* |
| \* @param coefficients - коэффициенты, которые необходимо задать |
| \* \*/ |
| @Override |
| public void setCoefficients(double[] coefficients) |
| { |
| System.arraycopy(coefficients, 0, this.coefficients, 0, coefficients.length); |
| } |
|  |
|  |
| /\*\* |
| \* Метод для получения значения функции в заданной точке |
| \* |
| \* @param x - точка, в которой необходимо получить значение функции |
| \* @return значение функции в заданной точке |
| \* \*/ |
| @Override |
| public double getValueAtX(double x) |
| { |
| return Math.pow(x, 4) + coefficients[0]\*Math.pow(x, 3) + coefficients[1]\*Math.pow(x, 2) + |
| coefficients[2]\*x + coefficients[3]; |
| } |
|  |
| /\*\* |
| \* Метод для получения интервалов смены знаков функции. |
| \* |
| \* @return список пар чисел - интервалов, в которых функция меняет знак. |
| \* \*/ |
| public List<List<Double>> getIntervals() |
| { |
| List<Double> xList = new LinkedList<>(); |
| List<Double> yList = new LinkedList<>(); |
|  |
| //Задаем интервал иксов от -100 до 100 |
| for (double i = -100.0; i <= 100; i++) |
| { |
| xList.add(i); |
| } |
|  |
| //Получаем значения функии в каждой точке интервала |
| //от -100 до 100 |
| for (Double aDouble : xList) |
| { |
| yList.add(getValueAtX(aDouble)); |
| } |
|  |
| //В коллекцию, которая может содержать только уникальные элементы |
| //вносим значения при которых функция меняет знак |
| List<Double> newList = new ArrayList<>(); |
| for (int i = 0; i < yList.size() - 1; i++) |
| { |
| if (yList.get(i) \* yList.get(i+1) < 0) |
| { |
| newList.add(xList.get(i)); |
| newList.add(xList.get(i+1)); |
| } |
| } |
|  |
| //Сортировка по возрастанию |
| Collections.sort(newList); |
|  |
| //Записываем по парам полученные значения |
| List<List<Double>> cordList = new ArrayList<>(); |
| for (int i = 0; i < newList.size(); i += 2) |
| { |
| List<Double> temp = new ArrayList<>(); |
|  |
| temp.add(newList.get(i)); |
| temp.add(newList.get(i + 1)); |
|  |
| cordList.add(temp); |
| } |
|  |
| return cordList; |
| } |
|  |
| /\*\* |
| \* Конуструктор без параметров |
| \* \*/ |
| public FourthDegreePolynomial() |
| { |
| } |
|  |
| /\*\* |
| \* Метод для вывода уравнения в консоль. |
| \* \*/ |
| @Override |
| public void printEquation() |
| { |
| System.out.printf("λ^4 %.4fλ^3 + %.4fλ^2 + %.4fλ %.4f = 0", coefficients[0],  coefficients[1], coefficients[2], coefficients[3]); |
| } |
| } |

**equation\_solution/SolutionStrategy.java**

|  |
| --- |
| package equation\_solution; |
| import equation.Equation; |
|  |
| import java.util.List; |
|  |
| /\*\* |
| \* Общий интерфейс всех стратегий решения. |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \* @see BisectionSolution |
| \* \*/ |
| public interface SolutionStrategy |
| { |
| /\*\* |
| \* Метод для вызова той или иной стратегии решения. |
| \* |
| \* @param equation - ур-ие, которое необходимо решить |
| \* @return список значений, которые являются решениями данного |
| \* ур-ия |
| \* \*/ |
| List<Double> getSolution(Equation equation); |
| } |

**equation\_solution/BisectionSolution.java**

|  |
| --- |
| package equation\_solution; |
| import equation.Equation; |
|  |
| import java.util.LinkedList; |
| import java.util.List; |
|  |
| /\*\* |
| \* Класс реализующий решение методом биссекций. |
| \* Реализует интерфейс SolutionStrategy |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \* @see SolutionStrategy |
| \* \*/ |
| public class BisectionSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения решений методом бисекции. |
| \* |
| \* @param equation - ур-ие, которое необходимо решить |
| \* @return список значений, являющимися решениями данного уравнения. |
| \* \*/ |
| @Override |
| public List<Double> getSolution(Equation equation) |
| { |
| //список, в который будут заноится ответы |
| List<Double> resList = new LinkedList<>(); |
|  |
| //Переменная для хранения значения Xi |
| double xI; |
|  |
| //Получение списка интервалов монотонности, которые содержат |
| //решения |
| List<List<Double>> intervals = equation.getIntervals(); |
|  |
| //Проход по каждому интервалу [a;b] |
| for (List<Double> interval: intervals) |
| { |
| //Изначально задается такое значение, которое не пройдет условие валидатора |
| xI = 0; |
|  |
| //Пока не сработает условие валидатора |
| //В предыдущее значение записываем текущее значение Xi |
| //Получаем новое значение Xi = (a+b)/2 |
| while (!(Math.abs(equation.getValueAtX(xI)) < 0.001)) |
| { |
| xI = ((interval.get(0)) + interval.get(1)) / 2.0; |
|  |
| //Если значение функии при Xi и при X=a имеет разные знаки, |
| //то меняем b из промежутка [a;b] на Xi |
| //иначе меняем a на Xi |
| if (equation.getValueAtX(xI) \* equation.getValueAtX(interval.get(0)) < 0) |
| { |
| interval.set(1, xI); |
| } |
| else |
| { |
| interval.set(0, xI); |
| } |
| } |
|  |
| //Записываем полученный ответ в результирующий список |
| resList.add(xI); |
| } |
| return resList; |
| } |
|  |
| /\*\* |
| \* Конструктор без параметров. |
| \* \*/ |
| public BisectionSolution() |
| { |
| } |
| } |

**equation\_system/SystemOfEquations.java**

|  |
| --- |
| package equation\_system; |
|  |
| /\*\* |
| \* Интерфейс системы уравнений. |
| \* |
| \* @see SystemOfFourEquations |
| \* \*/ |
| public interface SystemOfEquations |
| { |
| /\*\* |
| \* Получение двумерного массива коэфициентов системы. |
| \* |
| \* @return двумерный массив коэфифицентов системы. |
| \* \*/ |
| double[][] getCoefficients(); |
|  |
| /\*\* |
| \* Получение вектора B (вектор числе с правой стороны от '=') |
| \* |
| \* @return вектор B |
| \* \*/ |
| double[] getVectorB(); |
|  |
| /\*\* |
| \* Метод вывода системы уравненийи в консоль. |
| \* \*/ |
| void printSystem(); |
| } |

**equation\_system/SystemOfFourEquations.java**

|  |
| --- |
| package equation\_system; |
| /\*\* |
| \* Система из 4ех уравнений |
| \* |
| \* @see SystemOfEquations |
| \* \*/ |
| public class SystemOfFourEquations implements SystemOfEquations |
| { |
| //Переменные для хранения коэффициентов и вектора B |
| private final double[][] coefficients; |
| private final double[] vectorB ; |
|  |
| /\*\* |
| \* Конструктор с параметрами. |
| \* |
| \* @param coefficients - массив коэффициентов |
| \* @param vectorB - вектор B |
| \* \*/ |
| public SystemOfFourEquations(double[][] coefficients, double[] vectorB) |
| { |
| this.coefficients = coefficients; |
| this.vectorB = vectorB; |
| } |
|  |
| /\*\* |
| \* Метод получение двумерного массива коэффициентов |
| \* системы уравнений. |
| \* |
| \* @return двумерный массив коэффициентов системы уравнений |
| \* \*/ |
| public double[][] getCoefficients() |
| { |
| return coefficients; |
| } |
|  |
| /\*\* |
| \* Метод получение вектора B системы уравнений. |
| \* |
| \* @return вектор B |
| \* \*/ |
| public double[] getVectorB() |
| { |
| return vectorB; |
| } |
|  |
| /\*\* |
| \* Метод вывода системы уравнений в консоль. |
| \* \*/ |
| public void printSystem() |
| { |
| System.out.printf("%.4fp1 + %.4fp2 + %.4fp3 + %.4fp4 = %.4f\n", coefficients[0][0],  coefficients[0][1], coefficients[0][2], coefficients[0][3], vectorB[0]); |
| System.out.printf("%.4fp1 + %.4fp2 + %.4fp3 + %.4fp4 = %.4f\n", coefficients[1][0],  coefficients[1][1], coefficients[1][2], coefficients[1][3], vectorB[1]); |
| System.out.printf("%.4fp1 + %.4fp2 + %.4fp3 + %.4fp4 = %.4f\n", coefficients[2][0],  coefficients[2][1], coefficients[2][2], coefficients[2][3], vectorB[2]); |
| System.out.printf("%.4fp1 + %.4fp2 + %.4fp3 + %.4fp4 = %.4f\n", coefficients[3][0],  coefficients[3][1], coefficients[3][2], coefficients[3][3], vectorB[3]); |
| } |
| } |

**equation\_system\_solution/SolutionStrategy.java**

|  |
| --- |
| package equation\_system\_solution; |
|  |
| import equation\_system.SystemOfEquations; |
|  |
| /\*\* |
| \* Общий интерфейс всех стратегий решения. |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \* @see GaussSolution |
| \* \*/ |
| public interface SolutionStrategy |
| { |
| /\*\* |
| \* Метод для вызова той или иной стратегии решения. |
| \* |
| \* @param system - система, которую необходимо решить |
| \* @return список значений, являющимися решениями данной системы уравнений. |
| \* \*/ |
| double[] getSolution(SystemOfEquations system); |
| } |

**equation\_system\_solution/GaussSolution.java**

|  |
| --- |
| package equation\_system\_solution; |
|  |
| import equation\_system.SystemOfEquations; |
|  |
| /\*\* |
| \* Класс, реализующий решение методом Гаусса. |
| \* ВНИМАНИЕ!!! Данный метод подходит только для |
| \* системы уравнений данного варианта! |
| \* |
| \* @see SolutionStrategy |
| \* \*/ |
| public class GaussSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения решения системы уравнений |
| \* методом Гаусса |
| \* |
| \* @param system - система уравнений |
| \* @return массив решений данной системы. |
| \* \*/ |
| @Override |
| public double[] getSolution(SystemOfEquations system) |
| { |
| double[][] tempMatrix = new double[4][4]; |
| double[] tempVectorB = new double[4]; |
| double[] result = new double[4]; |
|  |
| for (int i = 0; i < 4; i++) |
| { |
| System.arraycopy(system.getCoefficients()[i], 0, tempMatrix[i], 0, system.getCoefficients()[i].length); |
| } |
| System.arraycopy(system.getVectorB(), 0, tempVectorB, 0, system.getVectorB().length); |
|  |
| //Умножим первую строку на (-1.144565): |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[0][i] \*= -1.145; |
| } |
| tempVectorB[0] \*= -1.144565; |
|  |
| //Добавим 2-ю строку к 1ой: |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[0][i] += tempMatrix[1][i]; |
| } |
| tempVectorB[0] += tempVectorB[1]; |
| // 0 1.178 0.627 1 | 1.933 |
| // 0 -0.63 -0.939 -0.719 | -6.103 |
| // 21.35 4.5 0.5 0 | -100.658 |
| // 25.238 5.77 1.2 0 | -112.596 |
|  |
|  |
| //Умножим 2-ю строку на -0,719485 |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[1][i] \*= -0.719485; |
| } |
| tempVectorB[1] \*= -0.719485; |
|  |
| //Добавим 3-ю строку ко второй |
| for (int i =0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[1][i] += tempMatrix[2][i]; |
| } |
| tempVectorB[1] += tempVectorB[2]; |
| // 0 1.78 0.627 1 |1.933 |
| // 0 -0.63 -0.939 -0.719 |-6.103 |
| // 21.35 4.5 0.5 0 | -100.658 |
| // 25.238 5.77 1.2 0 | -112.596 |
|  |
|  |
| //Умножим 3-ю строку на -1,182108 |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[2][i] \*= -1.182108; |
| } |
| tempVectorB[2] \*= -1.182108; |
|  |
| //Добавим 4-ю строку к 3-й |
| for (int i =0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[2][i] += tempMatrix[3][i]; |
| } |
| tempVectorB[2] += tempVectorB[3]; |
| // 0 1.78 0.627 1 |1.933 |
| // 0 -0.63 -0.939 -0.719 |-6.103 |
| // 0 0,451 0,609 0 | 6,393 |
| // 25.238 5.77 1.2 0 | -112.596 |
|  |
|  |
| //Умножим 1-ю строку на 0,534626 |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[0][i] \*= 0.534626; |
| } |
| tempVectorB[0] \*= 0.534626; |
|  |
| //Добавим 2-ю строку к 1-й |
| for (int i =0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[0][i] += tempMatrix[1][i]; |
| } |
| tempVectorB[0] += tempVectorB[1]; |
| // 0 1.78 0.627 1 |1.933 |
| // 0 -0.63 -0.939 -0.719 |-6.103 |
| // 0 0,451 0,609 0 | 6,393 |
| // 25.238 5.77 1.2 0 | -112.596 |
|  |
|  |
| //Умножим 2-ю строку на 0,715185 |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[1][i] \*= 0.715185; |
| } |
| tempVectorB[1] \*= 0.715185; |
|  |
| //Добавим 3-ю строку к 2-й |
| for (int i =0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[1][i] += tempMatrix[2][i]; |
| } |
| tempVectorB[1] += tempVectorB[2]; |
| // 0 0 -0.604 -0,185 |-5,07 |
| // 0 0 -0.0626 -0,515 |2,028 |
| // 0 0,451 0,609 0 | 6,393 |
| // 25.238 5.77 1.2 0 | -112.596 |
|  |
|  |
| //Умножим 1-ю строку на -0,103625 |
| for (int i = 0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[0][i] \*= -0.103625; |
| } |
| tempVectorB[0] \*= -0.103625; |
|  |
| //Добавим 2-ю строку к 1-й |
| for (int i =0; i < tempMatrix.length; i++) |
| { |
| tempMatrix[0][i] += tempMatrix[1][i]; |
| } |
| tempVectorB[0] += tempVectorB[1]; |
| // 0 0 0 0.495 |2.553 |
| // 0 0 -0.0626 -0,515 |2,028 |
| // 0 0,451 0,609 0 | 6,393 |
| // 25.238 5.77 1.2 0 | -112.596 |
|  |
| //x4 = 2.553/(-0.495) |
| result[3] = tempVectorB[0]/tempMatrix[0][3]; |
|  |
| //x3 = [2.028 - (-0.515x4)]/0.451 |
| result[2] = (tempVectorB[1] - (tempMatrix[1][3] \* result[3])) / tempMatrix[1][2]; |
|  |
| //x2 = [6.393 - (0.609x3)]/0.451 |
| result[1] = (tempVectorB[2] - (tempMatrix[2][2] \* result[2])) / tempMatrix[2][1]; |
|  |
| //x1 = [-112.596-(5.77\*x2 + 1.2x3)]/25.238 |
| result[0] = (tempVectorB[3] - (tempMatrix[3][1]\*result[1] +  tempMatrix[3][2]\*result[2]))/tempMatrix[3][0]; |
|  |
| return result; |
| } |
| } |

Начало формы

Конец формы

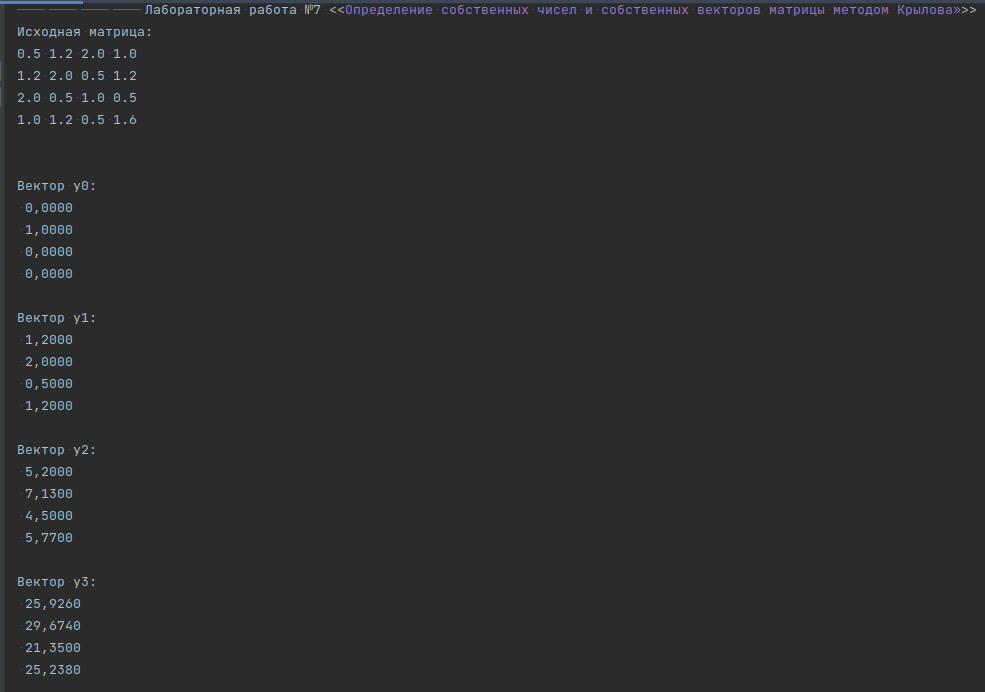
**matrix/Matrix.java**

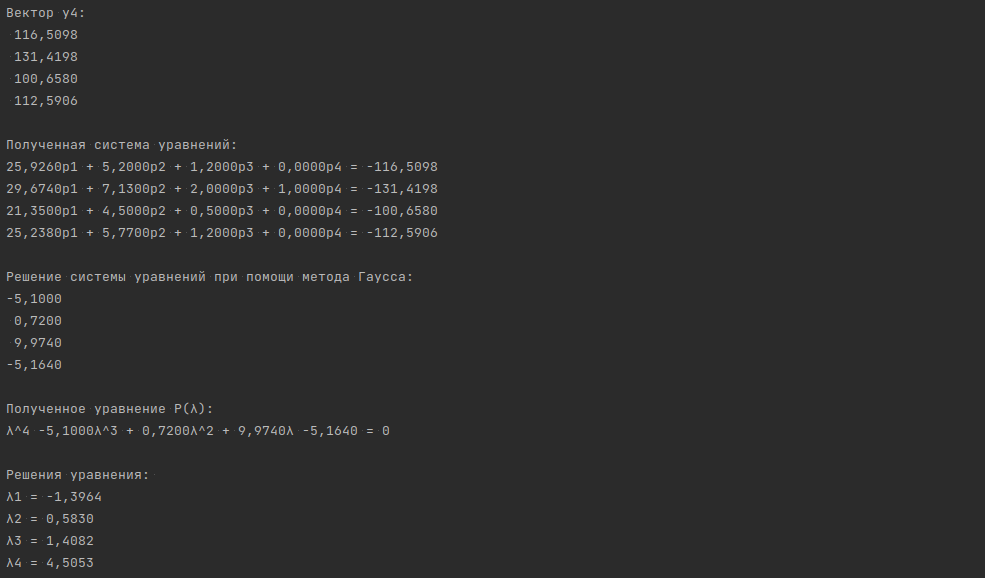
|  |
| --- |
| package matrix; |
|  |
| /\*\* |
| \* Интерфейс матриц |
| \* |
| \* @see Matrix4x4 |
| \* \*/ |
| public interface Matrix |
| { |
| /\*\* |
| \* Умножение матрицы на вектор. |
| \* |
| \* @param vector - вектор, на который необходимо умножить матрицу |
| \* @return полученный вектор |
| \* \*/ |
| double[] multiplyByVector(double[] vector); |
| } |

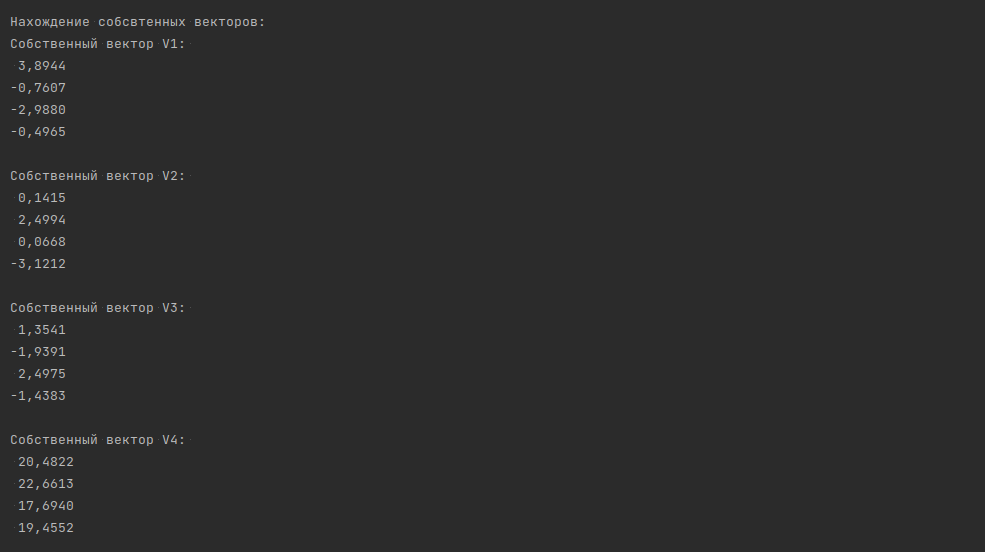
**matrix/Matrix4x4.java**

|  |
| --- |
| package matrix; |
|  |
| /\*\* |
| \* Класс матриц размером 4х4 |
| \* |
| \* @see Matrix |
| \* \*/ |
| public class Matrix4x4 implements Matrix |
| { |
| //поле для хранений значений матрицы |
| private final double[][] matrix = new double[4][4]; |
|  |
| /\*\* |
| \* Конструктор с параметрами. |
| \* |
| \* @param matrix - массив значений матрицы |
| \* \*/ |
| public Matrix4x4(double[][] matrix) |
| { |
| for (int i = 0; i < matrix.length; i++) |
| { |
| System.arraycopy(matrix[i], 0, this.matrix[i], 0, matrix[i].length); |
| } |
| } |
|  |
| /\*\* |
| \* Умножение матрицы на вектор. |
| \* |
| \* @param vector - вектор, на который необходимо умножить матрицу |
| \* @return полученный вектор |
| \* \*/ |
| public double[] multiplyByVector(double[] vector) |
| { |
| double[] result = new double[vector.length]; |
|  |
| for (int i = 0; i < matrix.length; i++) |
| { |
| for (int j = 0; j < matrix[i].length; j++) |
| { |
| result[i] += matrix[i][j] \* vector[j]; |
| } |
| } |
| return result; |
| } |
|  |
| /\*\* |
| \* Перегруженный метод вывода матрицы в консоль. |
| \* \*/ |
| @Override |
| public String toString() |
| { |
| return matrix[0][0] + " " + matrix[0][1] + " " + matrix[0][2] + " "  + matrix[0][3] + "\n" |
| + matrix[1][0] + " " + matrix[1][1] + " " + matrix[1][2] + " "  + matrix[1][3] + "\n" |
| + matrix[2][0] + " " + matrix[2][1] + " " + matrix[2][2] + " "  + matrix[2][3] + "\n" |
| + matrix[3][0] + " " + matrix[3][1] + " " + matrix[3][2] + " "  + matrix[3][3] + "\n"; |
| } |
| } |

1. **Результаты работы программы**







1. **Вывод**

В ходе данной работы были закреплены знания и умения по нахождение собственных чисел и собственных векторов методом Крылова.