Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 2

з дисципліни «Чисельні методи»

Тема: «Розв’язання систем лінійних рівнянь методом простих ітерацій»

Виконала:

студентка 3 курсу

групи КС-32

Дібцева Анна Миколаївна

Перевірила: доцент

Чуб Ольга Ігорівна

Харків – 2020

# ХОД РАБОТЫ

Целью данной работы является исследование алгоритма метода простых для решения систем линейных уравнений и его программная реализация.

Входными данные: невырожденная система линейных алгебраических уравнений, удовлетворяющая условиям сходимости метода и представленные в форме матрицы коэффициентов при переменных, вектора правых частей системы, вектора начального приближения и точности вычислений. Решение (ответ) представляется в виде вектора составленного из координат-значений переменных, удовлетворяющих системе.

Условием существования решения является ненулевой определитель матрицы, который находится при помощи метода разложения матрицы на алгебраические дополнения. Условием сходимости решения является условие преобладания диагональных элементов матрицы (для каждой строки модуль элемента на главной диагонали больше суммы модулей не диагональных элементов).

Метод решения системы заключается в последовательном приближении значений переменных к искомой точности вычислений. На каждом шаге строится матрица из коэффициентов предыдущего приближения, на главной диагонали которой элементы рассчитываются по следующей формуле:

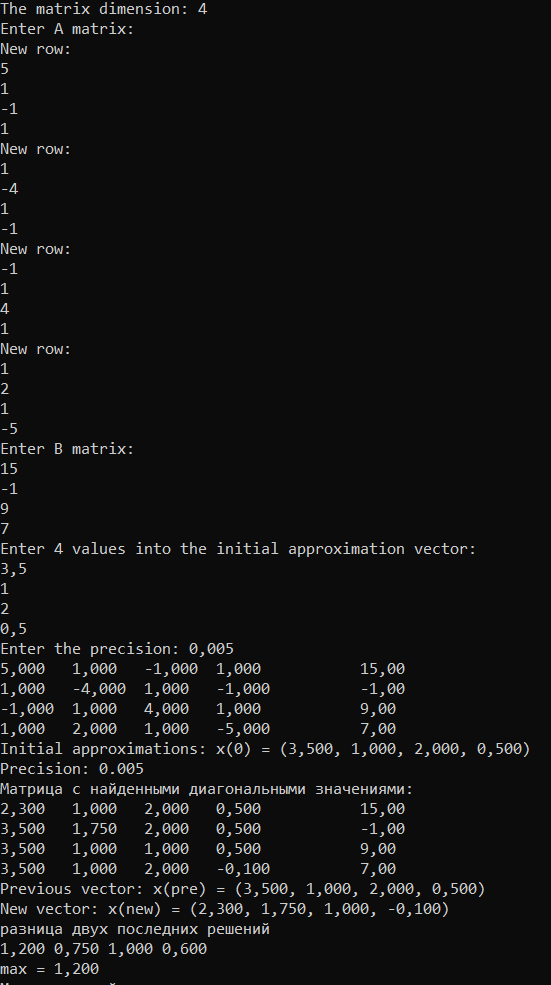


Элементы главной диагонали формируют новое приближение, и алгоритм повторяется заново.

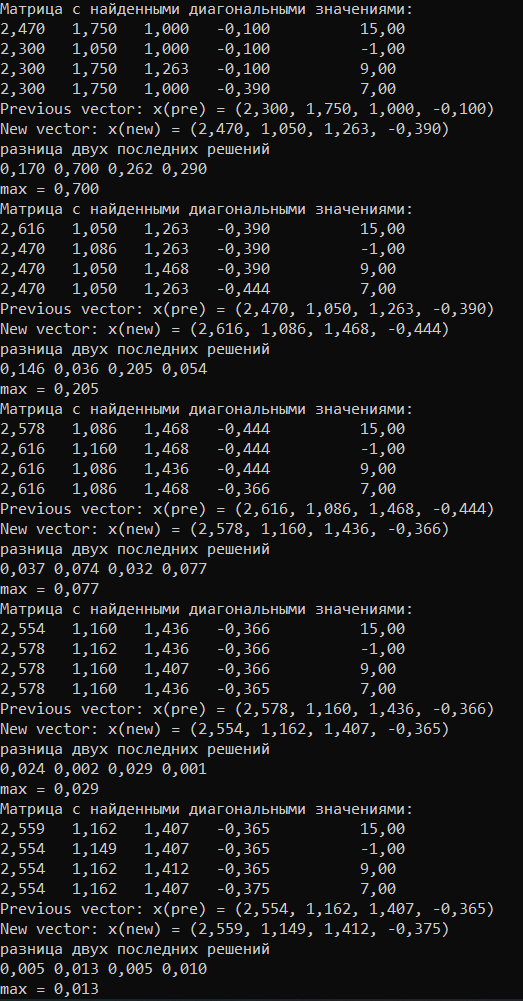
Для остановки алгоритма используется задание точности вычислений. Желаемая точность считается достигнутой, когда максимальный элемент модуля разности двух последних решений не превышает заданного числа.

Программа скомпилирована в исполняемый файл SimpleIterationMethod.exe.

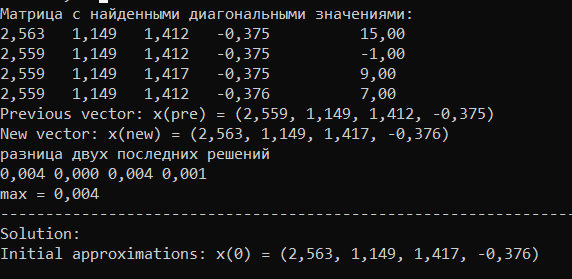
Результат работы программы приведен ниже.



Риcунок 1.1 – Результат запуска SimpleIterationMethod.exe



Риcунок 1.2 – Результат запуска SimpleIterationMethod.exe



Риcунок 1.3 – Результат запуска SimpleIterationMethod.exe

# ВЫВОДЫ

Таким образом, в ходе выполнения данной лабораторной работы был рассмотрен алгоритм простых итераций для решения систем линейных алгебраических уравнений и написана программа, которая позволяет решать невырожденные квадратные системы уравнений используя данный метод. Листинг программы приведен далее.

# ЛИСТИНГ ПРОГРАММЫ

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws InterruptedException {

Scanner scan = new Scanner(System.in);

System.out.print("The matrix dimension: ");

int dimension;

while(true){ //ввод размера

try {

dimension = scan.nextInt();

if (dimension < 2) continue;

break;

} catch (Exception e) {

scan.next();

System.out.print("Input error! Try again: ");

}

}

//создание и заполнение матриц

double[][] AMatrix = new double[dimension][dimension];

System.out.println("Enter A matrix:");

for(int i = 0; i < dimension; i++) {

System.out.println("New row:");

inputRow(AMatrix[i], scan);

}

double[] BMatrix = new double[dimension];

System.out.println("Enter B matrix:");

inputRow(BMatrix, scan);

double[] approximationVector = new double[AMatrix.length];

approximationVector = simpleIterationMentod(dimension, AMatrix, BMatrix, approximationVector, scan);

System.out.println("---------------------------------------------------------------------------------");

System.out.println("Solution: ");

printAprVector(approximationVector);

scan.close();

Thread.sleep(100000);

}

private static void inputRow(double row[], Scanner scan) {

for(int i = 0; i < row.length; i++) {

row[i] = inputValue(scan);

}

}

private static double inputValue(Scanner scan) {

double value;

while(true){

try {

value = scan.nextDouble();

break;

} catch (Exception e) {

scan.next();

System.out.print("Input error! Try again: ");

}

}

return value;

}

private static void matrixPrint(int dimension, double AMatrix[][], double BMatrix[]) {

for(int i = 0; i < dimension; i++) {

for(int j = 0; j < dimension; j++) {

System.out.printf("%4.3f", AMatrix[i][j]);

System.out.print("\t");

}

System.out.print("\t");

System.out.printf("%4.2f", BMatrix[i]);

System.out.println();

}

}

private static double[] simpleIterationMentod(int dimension, double AMatrix[][], double BMatrix[], double approximationVector[], Scanner scan) {

//проверка, что определитель матрицы не равен 0 и на сходимость

if(verification(dimension, AMatrix) == -1)

System.exit(-1);

//задаем вектор значений начального приближения

System.out.println("Enter " + AMatrix.length + " values into the initial approximation vector:");

inputRow(approximationVector, scan);

//задаем точность

System.out.print("Enter the precision: ");

double precision;

while(true){

precision = inputValue(scan);

if(precision > 0)

break;

}

//вывод матриц, начальных приближений и точности

matrixPrint(dimension, AMatrix, BMatrix);

printAprVector(approximationVector);

System.out.println("Precision: " + precision);

return iterations(AMatrix, BMatrix, approximationVector, precision);

}

/\*\*непосредственно функция итераций

\* AMatrix - матрица A

\* BMatrix - матрица B

\* previousVector - предыдущий вектор приближенных значений

\* precision - точность

\* @throws InterruptedException

\* \*/

private static double[] iterations(double AMatrix[][], double BMatrix[], double previousVector[], double precision) {

//матрица, с которой будут проходить вычисления

double[][] mainMatrix = new double[AMatrix.length][AMatrix.length];

do {

//заполнение матрицы значениями предыдущего вектора

mainMatrixFilling(mainMatrix, previousVector);

//нахождение диагональных значений

findingDiagonalValues(AMatrix, BMatrix, mainMatrix);

double[] newVector = new double[AMatrix.length];

//заполнение нового вектора значениями диагонали

fillingNewVector(newVector, mainMatrix);

if(precisionCheck(newVector, previousVector, precision) == 0) {

return previousVector = newVector.clone();

}

previousVector = newVector.clone();

}while(true);

}

//заполнение матрицы значениями предыдущего вектора

private static void mainMatrixFilling(double mainMatrix[][], double vector[]) {

for (int i = 0; i < mainMatrix.length; i++) {

for (int j = 0; j < mainMatrix[i].length; j++) {

mainMatrix[i][j] = vector[j];

}

}

}

//нахождение диагональных значений

private static void findingDiagonalValues(double AMatrix[][], double BMatrix[], double mainMatrix[][]) {

for (int i = 0; i < mainMatrix.length; i++) {

double val = BMatrix[i];

for (int j = 0; j < mainMatrix.length; j++) {

if(j != i)

val -= mainMatrix[i][j];

}

val /= AMatrix[i][i];

mainMatrix[i][i] = val;

}

System.out.println("Матрица с найденными диагональными значениями:");

matrixPrint(AMatrix.length, mainMatrix, BMatrix);

}

//заполнение нового вектора значениями диагонали

private static void fillingNewVector(double vector[], double mainMatrix[][]) {

for (int i = 0; i < mainMatrix.length; i++) {

vector[i] = mainMatrix[i][i];

}

}

//проверка точности - модуль разницы двух последних решений

private static int precisionCheck(double newVector[], double previousVector[], double precision) {

System.out.print("Previous vector: x(pre) = (");

for (int i = 0; i < previousVector.length; i++) {

System.out.printf("%4.3f", previousVector[i]);

if (i != previousVector.length-1)

System.out.print(", ");

}

System.out.println(")");

System.out.print("New vector: x(new) = (");

for (int i = 0; i < newVector.length; i++) {

System.out.printf("%4.3f", newVector[i]);

if (i != newVector.length-1)

System.out.print(", ");

}

System.out.println(")");

double[] differenceOfTwoSolutions = new double[newVector.length];

//разница двух последних решений

differenceOfTwoSolutions(newVector, previousVector, differenceOfTwoSolutions);

//нахождение максимума из разниц последних решений

double maxVal = maxInArr(differenceOfTwoSolutions);

if(maxVal <= precision)

return 0;

return 1;

}

//разница двух последних решений

private static void differenceOfTwoSolutions(double newVector[], double previousVector[], double differenceOfTwoSolutions[]) {

for (int i = 0; i < differenceOfTwoSolutions.length; i++) {

differenceOfTwoSolutions[i] = Math.abs(newVector[i] - previousVector[i]);

}

System.out.println("разница двух последних решений");

for (double d : differenceOfTwoSolutions) {

System.out.printf("%4.3f ", d);

}

System.out.println();

}

//нахождение максимума из разниц последних решений

private static double maxInArr(double arr[]) {

double max = arr[0];

for (int i = 1; i < arr.length; i++) {

if (max < arr[i])

max = arr[i];

}

System.out.printf("max = %4.3f\n", max);

return max;

}

private static void printAprVector(double arr[]) {

System.out.print("Initial approximations: x(0) = (");

for (int i = 0; i < arr.length; i++) {

System.out.printf("%3.3f",arr[i]);

if (i != arr.length-1)

System.out.print(", ");

}

System.out.println(")");

}

private static int verification (int dimension, double AMatrix[][]) {

//проверка, что определитель матрицы не равен 0

if (determinant(dimension, AMatrix) == 0) {

System.out.println("The determinant = 0");

return -1;

}

//проверка на сходимость

if (matrixСonvergence(AMatrix) == -1){

System.out.println("The matrix failed convergence test!");

return -1;

}

return 0;

}

//считает определитель матрицы

private static double determinant(int dimension, double matrix[][]) {

double matrixDeterminant = 0;

if(dimension > 2) {

for(int i = 0; i < dimension; i++) {

List<double[]> submatrix = new ArrayList<>();

for(int j = 0; j < dimension; j++) {

if(j != i) {

submatrix.add((double[])Arrays.copyOfRange(matrix[j], 1, matrix[j].length));

}

}

matrixDeterminant += Math.pow(-1, i) \* matrix[i][0] \*

determinant(dimension-1, submatrix.toArray(new double[dimension-1][dimension-1]));

}

return matrixDeterminant;

}

else {

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0];

}

}

//проверяет сходимость матрицы

private static int matrixСonvergence (double matrix[][]) {

for(int i = 0; i < matrix.length; i++) {

if(lineCheck(i, matrix[i]) == -1)

return -1;

}

return 0;

}

//проверяет, что сумма модулей элементов в строке, кроме элемента, входящего в диагональ матрицы,

//меньше, чем модуль элемента диагонали матрицы

private static int lineCheck(int index, double line[]) {

double sum = 0;

for(int i = 0; i < line.length; i++)

if(i != index)

sum += Math.abs(line[i]);

if(Math.abs(line[index]) < sum)

return -1;

return 0;

}

}