Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 2

з дисципліни «Числові методи»

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

Виконав:

студент 3 курсу

групи КС-32

Безрук Юрій Русланович

Перевірила: доцент

Чуб Ольга Ігорівна

Харків – 2020

# ХОД РАБОТЫ

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

Входными данными являются невырожденные системы линейных алгебраических уравнений, удовлетворяющие условиям сходимости метода и представленные в форме матрицы коэффициентов при переменных, вектора правых частей системы, вектора начального приближения и точности вычислений. Решение 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 для реализации данного метода была написана на языке Java. В основе взяты наработки из прошлой лабораторной работы: классы Row и Matrix, дополненные новыми методами. Также, для более точных расчетов и округлений числа с плаввающей точкой были заменены на «большие числа» BigDecimal с точностью в три знака после запятой и округлением половина-вниз (если > 5, округляем вверх, иначе вниз).

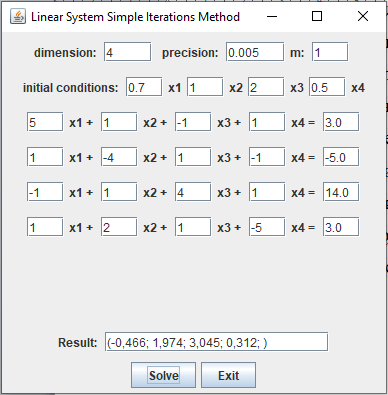
Для проверки сходимости метода у объекта матрицы следует вызвать метод isConverging(), а для получения решения итерационным способом – метод solveBySimpleIterarions(Row approach, double precision), где approach – вектор начального приближения, а precision – точность вычислений. При вычислении процесс вычисления логируется в файл countingProcess.txt.

Для удобства пользования графический интерфейс системы заимствован из прошлой работы, с некоторыми дополнениями – поле вывода результата смещено под поля ввода матрицы, а так же добавлены новые поля ввода: точности, вектора начального приближения и m – варианта студента (изменения в поле m отражаются на системе, данной по умолчанию при нажатии клавиши Enter).

Перед началом вычислений программа проверяет определитель матрицы и сходимость решения. В случае неподходящих данных об этом уведомит всплывающее сообщение. На все поля для ввода предусмотрена валидация: процесс решения не начнется если введены неверные данные или данные введены не полностью

В качестве значений по умолчанию в фрейме установлены коэффициенты предоставленные преподавателем.

Программа скомпилирована в исполняемый файл lab2\_Method.exe. В процессе его выполнения в директории где он находится будет обновлен/создан файл countingProcess.txt, содержащий подробный ход решения. Пример работы программы для варианта, заданного преподавателем:



# 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

Класс Row, инкапсулирующий строку матрицы:

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**import** java.util.Arrays;

**public** **class** Row **implements** Cloneable{

**private** BigDecimal[] row;

**public** Row(**double**... elems) {

row = **new** BigDecimal[elems.length];

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

row[i] = **new** BigDecimal(elems[i]);

row[i].setScale(3, RoundingMode.***HALF\_DOWN***);

}

}

**public** Row(BigDecimal... elems) {

row = **new** BigDecimal[elems.length];

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

row[i] = **new** BigDecimal(elems[i].doubleValue());

row[i].setScale(3, RoundingMode.***HALF\_DOWN***);

}

}

**public** **int** size() {

**return** row.length;

}

**public** BigDecimal max() {

**return** java.util.Collections.*max*(toList());

}

**public** BigDecimal min() {

**return** java.util.Collections.*min*(toList());

}

**public** Row add(Row otherRow) {

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

row[i] = row[i].add(otherRow.row[i]);

**return** **this**;

}

**public** Row subtract(Row otherRow) {

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

row[i] = row[i].subtract(otherRow.row[i]);

**return** **this**;

}

**public** Row multiply(**double** value) {

BigDecimal coeff = **new** BigDecimal(value);

coeff.setScale(3, RoundingMode.***HALF\_DOWN***);

**return** multiply(coeff);

}

**public** Row multiply(BigDecimal value) {

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

row[i] = row[i].multiply(value);

**return** **this**;

}

**public** Row divide(**double** value) {

BigDecimal coeff = **new** BigDecimal(value);

coeff.setScale(3, RoundingMode.***HALF\_DOWN***);

**return** divide(coeff);

}

**public** Row divide(BigDecimal value) {

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

row[i] = row[i].divide(value, 3, RoundingMode.***HALF\_DOWN***);

**return** **this**;

}

**public** BigDecimal elem(**int** index) {

**return** row[index];

}

**public** **void** set(**int** index, **double** value) {

BigDecimal elem = **new** BigDecimal(value);

elem.setScale(3, RoundingMode.***HALF\_DOWN***);

set(index, elem);

}

**public** **void** set(**int** index, BigDecimal value) {

row[index] = value;

}

**public** Row subRow(**int** from, **int** to) {

Row subRow = **new** Row(elem(from));

**for** (**int** i = from + 1; i < to; i++) {

subRow.appendElem(elem(i));

}

**return** subRow;

}

**public** Row appendElem(**double** elem) {

BigDecimal value = **new** BigDecimal(elem);

value.setScale(3, RoundingMode.***HALF\_DOWN***);

**return** appendElem(value);

}

**public** Row appendElem(BigDecimal elem) {

BigDecimal[] temp = Arrays.*copyOf*(row, row.length + 1);

temp[temp.length-1] = elem;

row = temp;

**return** **this**;

}

**public** java.util.List<BigDecimal> toList(){

java.util.List<BigDecimal> list = **new** java.util.ArrayList<>();

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

list.add(row[i]);

}

**return** list;

}

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

**return** **new** Row(row);

}

@Override

**public** String toString() {

String result = "";

**for** (BigDecimal elem : row) {

result += String.*format*((elem.compareTo(**new** BigDecimal(0)) == -1 ? "" : " ") + "%.3f ", elem);

}

**return** result + "\n";

}

}

Класс Matrix, инкапсулирующий возможности системы уравнений:

**import** java.io.IOException;

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Matrix {

**private** Row[] system;

**private** java.io.FileWriter writer;

**public** Matrix(Row... rows) {

system = **new** Row[rows.length];

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

**try** {

system[i] = (Row) rows[i].clone();

} **catch** (CloneNotSupportedException e) {

e.printStackTrace();

}

}

**try** {

writer = **new** java.io.FileWriter("countingProcess.txt");

}**catch**(java.io.IOException e) {

e.printStackTrace();

}

}

**public** **static** Matrix createFromRow(Row row) {

Matrix matrix = **new** Matrix(row);

**for** (**int** i = 0; i < row.size() - 1; i++) {

matrix.appendRow(row);

}

**return** matrix;

}

**public** **int** size() {

**return** system.length;

}

**public** Row row(**int** index) {

**return** system[index];

}

**public** Row rightPart() {

Row rightPart = **new** Row(row(0).elem(row(0).size() - 1));

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

rightPart.appendElem(row(i).elem(row(i).size() - 1));

}

**return** rightPart;

}

**public** Row diagonal() {

Row diagonal = **new** Row(row(0).elem(0));

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

diagonal.appendElem(row(i).elem(i));

}

**return** diagonal;

}

**public** Matrix appendRow(Row row) {

Row[] temp = Arrays.*copyOf*(system, system.length + 1);

**try** {

temp[temp.length-1] = (Row) row.clone();

} **catch** (CloneNotSupportedException e) {

e.printStackTrace();

}

system = temp;

**return** **this**;

}

**public** Matrix appendColumn(Row column) {

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

row(i).appendElem(column.elem(i));

}

**return** **this**;

}

**public** Matrix minor(**int** rowNumber) {

List<Row> rows = **new** ArrayList<>();

**for** (**int** j = 0; j < size(); j++) {

**if**(j != rowNumber) {

rows.add(row(j).subRow(1, row(j).size() - (row(j).size() > **this**.size() ? 1 : 0)));

}

}

**return** **new** Matrix(rows.toArray(**new** Row[rows.size()]));

}

**public** BigDecimal determinant() {

**if**(size() < 1) {

**return** **new** BigDecimal(0);

}

**else** **if**(size() == 1) {

**return** row(0).elem(0);

}

**else** **if**(size() == 2) {

**return** row(0).elem(0).multiply(row(1).elem(1)).subtract(row(0).elem(1).multiply(row(1).elem(0)));

}

**else** {

BigDecimal result = **new** BigDecimal(0);

result.setScale(3, RoundingMode.***HALF\_DOWN***);

**for**(**int** i = 0; i < size(); i++) {

BigDecimal sign = **new** BigDecimal(Math.*pow*(-1, i));

sign.setScale(3, RoundingMode.***HALF\_DOWN***);

result = result.add(sign.multiply(row(i).elem(0).multiply(minor(i).determinant())));

}

**return** result;

}

}

**public** Row solveByGauss() {

log();

**for**(**int** i = 0; i < size(); i++) {

row(i).divide(row(i).elem(i));

log();

**for** (**int** j = 0; j < size(); j++) {

**if**(i != j) {

Row next = **null**;

**try** {

next = (Row) row(i).clone();

} **catch** (CloneNotSupportedException e) {}

row(j).subtract(next.multiply(row(j).elem(i)));

log();

}

}

}

**try** {

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** rightPart();

}

**public** **boolean** isConverging() {

**for** (**int** i = 0; i < size(); i++) {

BigDecimal sum = **new** BigDecimal(0);

sum.setScale(3, RoundingMode.***HALF\_DOWN***);

**for** (**int** j = 0; j < size(); j++)

**if**(j != i)

sum = sum.add(row(i).elem(j).abs());

**if**(sum.compareTo(row(i).elem(i).abs()) == 1)

**return** **false**;

}

**return** **true**;

}

**public** Row solveBySimpleIterations(Row approach, **double** precision) {

BigDecimal bigPrecision = **new** BigDecimal(precision);

bigPrecision.setScale(3, RoundingMode.***HALF\_DOWN***);

Row difference = **new** Row(bigPrecision.add(**new** BigDecimal(1)));

**while**(difference.max().compareTo(bigPrecision) == 1) {

Matrix newSystem = Matrix.*createFromRow*(approach).appendColumn(**this**.rightPart());

**for** (**int** i = 0; i < newSystem.size(); i++) {

BigDecimal xi = newSystem.row(i).elem(row(i).size() - 1);

xi.setScale(3, RoundingMode.***HALF\_DOWN***);

**for** (**int** j = 0; j < newSystem.size(); j++) {

**if**(i != j) {

xi = xi.subtract(newSystem.row(i).elem(j));

}

}

newSystem.row(i).set(i, xi.divide(**this**.row(i).elem(i), 3, RoundingMode.***HALF\_DOWN***));

}

**this**.log(newSystem.toString());

difference = newSystem.diagonal();

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

difference.set(i, newSystem.diagonal().elem(i).subtract(approach.elem(i)).abs());

}

approach = newSystem.diagonal();

**this**.log(approach.toString());

}

**try** {

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** approach;

}

@Override

**public** String toString() {

String result = "";

**for** (Row row : system) {

result += row;

}

**return** result + "\n";

}

**private** **void** log() {

log(toString());

}

**private** **void** log(String what) {

**try** {

writer.write(what+"\n");

} **catch** (java.io.IOException e) {

e.printStackTrace();

}

}

}

Классы графического интерфейса реализованы на основе библиотеки swing и не обладают функционалом, необходимым к представлению в данной работе, поэтому в листинге не приводятся, но предоставляются вместе со всеми исходными и исполняемыми файлами преподавателю.