Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 3

з дисципліни «Числові методи»

Тема: «Метод дихотомії»

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 дихотомии для решения нелинейных уравнений и его программная реализация.

Входными данными являются нелинейные уравнения, диапазон на котором у функции имеются корни, точность вычислений. Решение представляется в виде вектора корней уравнения.

Метод решения уравнений методом дихотомии можно разделить на две части: отделение корней (поиск интервалов, на которых функция является унимодальной, т.е., имеющий ровно один экстремум) и уточнение корней – последовательное увеличение точности до заданной с нахождением корня.

Желаемая точность считается достигнутой, когда модуль разности двух последовательных приближений корня меньше удвоенного значения заданной точности.

Программа для реализации данного метода была написана на языке Java. Был разработан абстрактный класс Equation, представляющий возможности для нахождения корней методом дихотомии для любого уравнения и оставляющий открытым для переопределения метод вычисления значения функции в точке. Его наследник – класс OrderEquation переопределяет этот метод для степенных уравнений любого порядка. Для более точных расчетов и округлений числа с плавающей точкой были заменены на «большие числа» BigDecimal с точностью в пять знаков после запятой и округлением половина-вниз (если > 5, округляем вверх, иначе вниз).

Для удобства пользования был создан графический интерфейс. По центру располагается график функции, под ним – кнопки его регуляции, настраивающие диапазон аргумента и шаг его изменения (влияет на точность вычисления). Вверху формы расположены поля для ввода порядка уравнения, точности и коэффициентов. По нажатию кнопки Solve в поле Result появится список корней уравнения. Стоит заметить, что для верности решения необходимо настраивать диапазон аргумента исходя из графика – программа будет искать корни только на тех значениях функции, которые попадают в область графика.

На все поля для ввода предусмотрена валидация: процесс решения не начнется если введены неверные данные или данные введены не полностью и будет выведено соответствующее сообщение.

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

Программа скомпилирована в исполняемый файл lab3\_DichotomyMethod.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:

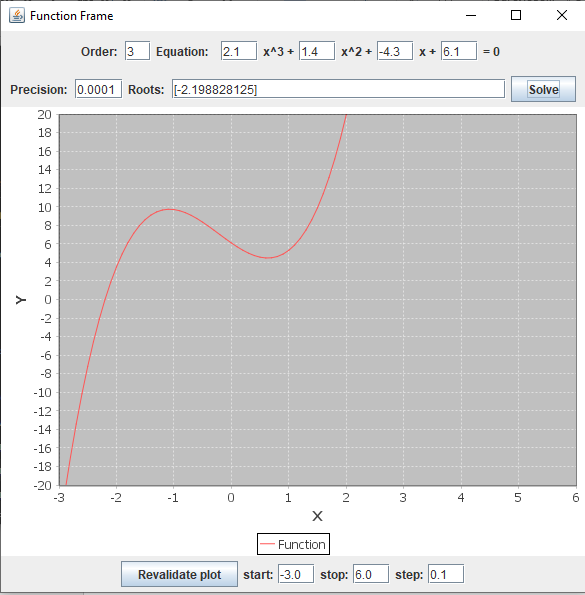


Рисунок 1 - пример работы программы

Как видно из графика, функция имеет всего одно пересечение с осью абсцисс, и, как следствие – всего один корень.

# ВЫВОДЫ

Таким образом, в ходе выполнения данной лабораторной работы был рассмотрен алгоритм дихотомии для решения нелинейных уравнений и написана программа, которая позволяет решать степенные уравнения используя метод. Листинг программы приведен далее.

# ЛИСТИНГ ПРОГРАММЫ

Класс Equation, инкапсулирующий любое абстрактное уравнение:

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**import** java.util.Arrays;

**public** **abstract** **class** Equation {

**protected** BigDecimal[] coeffs;

**protected** BigDecimal start;

**protected** BigDecimal stop;

**protected** BigDecimal step;

**protected** **int** scale = 5;

**protected** RoundingMode mode = RoundingMode.***HALF\_DOWN***;

{

start = **new** BigDecimal("-3.0");

start.setScale(scale, mode);

stop = **new** BigDecimal("6.0");

stop.setScale(scale, mode);

step = **new** BigDecimal("0.1");

step.setScale(scale, mode);

}

**public** Equation(**double**... row) {

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

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

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

coeffs[i].setScale(scale, mode);

}

}

**public** Equation(String... row) {

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

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

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

coeffs[i].setScale(scale, mode);

}

}

**public** Equation(BigDecimal... row) {

coeffs = Arrays.*copyOf*(row, row.length);

}

**public** **abstract** BigDecimal f(BigDecimal x);

**public** BigDecimal[][] getValuesDiapazone() {

java.util.List<BigDecimal[]> diapazone = **new** java.util.ArrayList<>();

BigDecimal I = **new** BigDecimal(start.toString());

I.setScale(scale, mode);

**for**(; I.compareTo(stop) != 1; I = I.add(step)) {

diapazone.add(**new** BigDecimal[] {I, f(I)});

}

**return** diapazone.toArray(**new** BigDecimal[diapazone.size()][2]);

}

**public** BigDecimal[] solve(BigDecimal precision) {

precision = precision.multiply(**new** BigDecimal("2.0"));

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

BigDecimal[][] diapazone = **this**.getValuesDiapazone();

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

**if**(diapazone[i][1].signum() != diapazone[i-1][1].signum()) {

//System.out.println("---------/"+diapazone[i-1][1]+" "+diapazone[i][1]);

BigDecimal middle = diapazone[i-1][0].add(diapazone[i][0]).divide(**new** BigDecimal("2.0"));

//System.out.println("<"+middle+">");

BigDecimal subResult = **this**.clarificateRoot(precision, diapazone[i-1][0], middle, diapazone[i][0]);

result.add(subResult);

}

}

**return** result.toArray(**new** BigDecimal[result.size()]);

}

**private** BigDecimal clarificateRoot(BigDecimal precision, BigDecimal... diapazone) {

//System.out.println(Arrays.toString(diapazone));

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

**if**(f(diapazone[i]).signum() != f(diapazone[i-1]).signum()) {

**if**(diapazone[i].subtract(diapazone[i-1]).abs().compareTo(precision) == -1)

**return** diapazone[i-1];

**else** {

BigDecimal middle = diapazone[i-1].add(diapazone[i]).divide(**new** BigDecimal("2.0"));

**return** clarificateRoot(precision, diapazone[i-1], middle, diapazone[i]);

}

}

}

**throw** **new** NoAxisIntersectionException(Arrays.*toString*(diapazone));

}

**public** **double** getStart() {

**return** start.doubleValue();

}

**public** **void** setStart(**double** start) {

**this**.start = **new** BigDecimal(start);

**this**.start.setScale(scale, mode);

}

**public** **void** setStart(String start) {

**this**.start = **new** BigDecimal(start);

**this**.start.setScale(scale, mode);

}

**public** **double** getStop() {

**return** stop.doubleValue();

}

**public** **void** setStop(**double** stop) {

**this**.stop = **new** BigDecimal(stop);

**this**.stop.setScale(scale, mode);

}

**public** **void** setStop(String stop) {

**this**.stop = **new** BigDecimal(stop);

**this**.stop.setScale(scale, mode);

}

**public** **double** getStep() {

**return** step.doubleValue();

}

**public** **void** setStep(**double** step) {

**this**.step = **new** BigDecimal(step);

**this**.step.setScale(scale, mode);

}

**public** **void** setStep(String step) {

**this**.step = **new** BigDecimal(step);

**this**.step.setScale(scale, mode);

}

}

Класс OrderEquation, инкапсулирующий конкретную реализацию своего абстрактного наследника – степенное уравнение.

**import** java.math.BigDecimal;

**public** **class** OrderEquation **extends** Equation{

**public** OrderEquation() {

**this**("1", "-2.8", "-6.2", "3.7");

//this("2.1", "1.4", "-4.3", "6.1");

}

**public** OrderEquation(**double**... row) {

**super**(row);

}

**public** OrderEquation(String... row) {

**super**(row);

}

**public** OrderEquation(BigDecimal... row) {

**super**(row);

}

**public** BigDecimal f(**double** x) {

BigDecimal newX = **new** BigDecimal(x);

newX.setScale(scale, mode);

**return** f(newX);

}

**public** BigDecimal f(BigDecimal x) {

x = **new** BigDecimal(x.toString());

BigDecimal result = **new** BigDecimal("0.0");

result.setScale(scale, mode);

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

result = result.add(coeffs[i].multiply(x.pow(coeffs.length-i-1)));

}

**return** result;

}

}

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.