Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A picture containing logo

Description automatically generated

Лабораторная работа №2 по дисциплине  
«Вычислительная математика»

" Численное решение нелинейных уравнений и систем"

Выполнил: Дау Конг Туан Ань

Группа: P32151

Преподаватель: Машина Е.А

г. Санкт-Петербург

2023

Цель работы: изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов

1. Вычислительная реализация задачи:

Исследуемое уравнение, согласно варианту:

1. Задание:
   1. Отделить корни заданного нелинейного уравнения графически (вид уравнения представлен в табл. 6

Chart, line chart

Description automatically generated

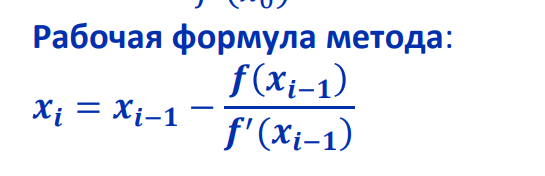
* 1. Определить интервалы изоляции корней

Уравнение имеет три корня, выберем начальные приближения для корней как интервалы:

* Для крайнего левого корня =>
* Для центрального корня =>
* Для крайнего правого корня =>
  1. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью ε=10-2
  2. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.
  3. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.
     1. Для метода Ньютона заполнить таблицу 3:

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |  |
| 0 | -1.5 | -2.69825 | 12.625 | -1.286 | 0.214 |
| 1 | -1.286 | -0.3359 | 9.49 | -1.251 | 0.035 |
| 2 | -1.251 | -0.012 | 9.01 | -1.249 | **0.002** |

* + 1. Для метода половинного деления заполнить таблицу 1

Timeline

Description automatically generated

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № итерации | a | b | x | F(a) | F(b) | F(x) | |a – b| |
| 0 | 0.3 | 0.7 | 0.5 | 0.154 | -1.18 | 0.052 | 0.4 |
| 1 | 0.5 | 0.7 | 0.6 | 0.052 | -1.18 | -0.051 | 0.2 |
| 2 | 0.5 | 0.6 | 0.55 | 0.052 | -0.051 | -0.246 | 0.1 |
| 3 | 0.5 | 0.55 | 0.525 | 0.052 | -0.246 | -0.096 | 0.05 |
| 4 | 0.5 | 0.525 | 0.5125 | 0.052 | -0.096 | -0.022 | 0.025 |
| 5 | 0.5 | 0.5125 | 0.506 | 0.052 | -0.022 | 0.016 | 0.0125 |
| 6 | 0.506 | 0.5125 | 0.509 | 0.016 | -0.022 | **-0.001** | 0.065 |

Правый корень уравнения: 0.509(точность 0.01)

* + 1. Для метода простой итерации заполнить таблицу 5

Timeline

Description automatically generated

λ =

Chart, line chart

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № итерации |  |  |  | || |
| 0 | 3.5 | 3.766 | -1.63 | 0.266 |
| 1 | 3.766 | 3.849 | -0.287 | 0.083 |
| 2 | 3.849 | 3.863 | -0.049 | 0.014 |
| 3 | 3.863 | 3.865 | -0.01 | **0.002** |

Левый корень уравнения: 3.865 (точность 0.01)

1. Программная реализация задачи:
2. Метод простой итерации

public void execute(LinearEquation equation)  
 throws ImplementRangeException {  
 if(equation.resultAt(leftPoint) \* equation.resultAt(rightPoint) >= 0)  
 throw new ImplementRangeException();  
  
 lambda = Math.*max*(Math.*abs*(equation.firstDerivative(leftPoint)),  
 Math.*abs*(equation.firstDerivative(rightPoint)));  
  
 if(lambda == 0  
 || 1 - 1/lambda\* equation.firstDerivative(leftPoint) > 1  
 || 1 - 1/lambda\* equation.firstDerivative(rightPoint) > 1)  
 throw new ImplementRangeException();  
  
 double x, x\_1 = leftPoint, f\_x\_1;  
 do{  
 x = x\_1;  
 x\_1 = x - 1/lambda \* equation.resultAt(x);  
 f\_x\_1 = equation.resultAt(x\_1);  
  
 result.setRoot(x\_1);  
 result.setError(Math.*abs*(x\_1 - x));  
 result.increaseStep();  
 } while (Math.*abs*(x\_1 - x) >= *ERROR* );  
}

1. Метод хорд

public void execute(LinearEquation equation)  
 throws ImplementRangeException {  
 *// check if implement range is valid* if(equation.resultAt(leftPoint) \* equation.resultAt(rightPoint) > 0)  
 throw new ImplementRangeException();  
  
 double x, f\_x;  
 do {  
 x = leftPoint *// implement result* - (rightPoint - leftPoint)  
 / (equation.resultAt(rightPoint) - equation.resultAt(leftPoint))  
 \* equation.resultAt(leftPoint);  
  
 f\_x = equation.resultAt(x);  
  
 *// save result of current step* result.setRoot(x);  
 result.setError(Math.*abs*(f\_x));  
 result.increaseStep();  
  
 *// change range for next step* if(equation.resultAt(leftPoint)\* equation.resultAt(x) < 0) {  
 setRange(leftPoint, x);  
 } else {  
 setRange(x, rightPoint);  
 }  
 } while(Math.*abs*(f\_x) >= *ERROR*);  
}

1. Метод Ньютона(linear)

public void execute(LinearEquation equation)  
 throws ImplementRangeException {  
 if(equation.resultAt(leftPoint) \* equation.secondDerivative(leftPoint) <= 0  
 && equation.resultAt(rightPoint )\* equation.secondDerivative(rightPoint) <= 0)  
 throw new ImplementRangeException();  
  
 double x, f\_x, f\_1\_x, x\_1 = (equation.secondDerivative(leftPoint)  
 \* equation.secondDerivative(leftPoint)) > 0 ? leftPoint : rightPoint;  
  
 do {  
 x = x\_1;  
 f\_x = equation.resultAt(x);  
 f\_1\_x = equation.firstDerivative(x);  
 x\_1 = x - f\_x / f\_1\_x;  
  
 result.setRoot(x\_1);  
 result.setError(Math.*abs*(x - x\_1));  
 result.increaseStep();  
 } while(Math.*abs*(x - x\_1) >= *ERROR*);  
  
}

1. Метод Ньютона(System)

public void execute(SystemEquation equation)  
 throws ImplementFirstValuesException {  
 double f\_x, f\_y, g\_x, g\_y, delta\_x, delta\_y, f, g;  
  
 do{  
 f\_x = equation.derivativeFxForX(x, y);  
 f\_y = equation.derivativeFxForY(x, y);  
 g\_x = equation.derivativeGxForX(x, y);  
 g\_y = equation.derivativeGxForY(x, y);  
  
 f = -equation.FxAt(x, y);  
 g = -equation.GxAt(x, y);  
  
 *// check if equation have root with given value or not  
 // throw exception if we cannot find root with given root* if(f\_x \* g\_y - f\_y \* g\_x == 0)  
 throw new ImplementFirstValuesException();  
 delta\_x = (f \* g\_y - g \* f\_y) / (f\_x \* g\_y - f\_y \* g\_x);  
 delta\_y = (f\_x \* g - g\_x \* f) / (f\_x \* g\_y - f\_y \* g\_x);  
  
 *// set value of x and y for next loop step* x += delta\_x;  
 y += delta\_y;  
  
*// Printer.println(delta\_x + " " + delta\_y + " " + f\_x + " " + f\_y + " " +g\_x + " " +g\_y + " " + f + " " + g +"\n");  
  
 // save result of current step* result.setRoot(x, y);  
 result.setError(Math.*abs*(delta\_x), Math.*abs*(delta\_y));  
 result.increaseStep();  
 } while(Math.*abs*(delta\_x) >= *ERROR* || Math.*abs*(delta\_y) >= *ERROR*);  
 }

1. Пример входных данных

Do you want to type data from console(1) or get data from file(2) ?

1

Do you want to solve linear equation(1) or system equation(2) ?

1

List of linear algorithm:

Algorithm 0 : Fixed Point

Algorithm 1 : Newton(linear)

Algorithm 2 : Secant

Please choose your algorithm:

1

List of linear equation:

Equation 0 : x^3 - 3.125x^2 - 3.5x + 2.458

Equation 1 : x^3 - x + 4

Equation 2 : 2x + sin(x)

Please choose your Equation:

0

Type left point of range: 3.5

Type right point of range: 3.9

Dow you wanna write result to file ?

2

Result of program:

Root: 3.8658934876489637

Error: 0.003825473956743153

Number of step: 3

1. Выводы:

В ходе этой лабораторной работы я познакомился с несколькими методами, позволяющими решать нелинейные уравнения и системы нелинейных уравнений. Все методы довольно легко программируются и дают высокую точность и быструю сходимость при удачном выборе начального приближения.