Университет ИТМО, факультет ПИиКТ

Лабораторная работа №3

Дисциплина: Вычислительная математика

Вариант 17

Выполнил: Чайка Алексей

Группа: Р3214

Преподаватель: Малышева Т. А.

Санкт-Петербург, 2020 год

**Текст задания:**

**Лабораторная работа 3. «Численное решение нелинейных уравнений»**

**Исходные данные:**

1. Пользователь выбирает функцию, интеграл которой требуется вычислить (3-5 функций), из тех, которые предлагает программа.

2. Пределы интегрирования задаются пользователем.

3. Точность вычисления задаются пользователем.

4. Начальное значение числа разбиения интервала интегрирования: n=4.

5. Ввод исходных данных осуществляется с клавиатуры.

**Программная реализация задачи:**

1. Все численные методы (см. табл. 5) должны быть реализованы в виде отдельных подпрограмм или классов.

2 Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.

3 Выполнить верификацию исходных данных. Для метода половинного деления (метода хорд) анализировать наличие корня на введенном интервале. Для метода Ньютона (метода секущих) – выбор начального приближения (а или b). Для метода простой итерации – достаточное условие сходимости метода. Программа должна реагировать на некорректные введенные данные.

4 Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.

5 Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

**Вычислительная реализация задачи:**

1. Сформировать таблицы, соответственно варианту

**Цель работы:** найти приближенное решение нелинейного уравнения с требуемой точностью различными численными методами.

**Ход работы:**

**Метод половинного деления:**

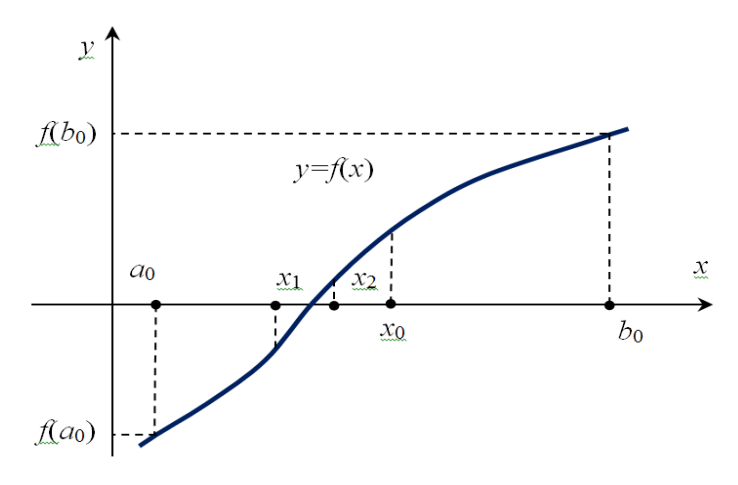
*Идея метода:* начальный интервал изоляции корня делим пополам, получаем начальное приближение к корню:

Вычисляем В качестве нового интервала выбираем ту половину отрезка, на концах которого функция имеет разные знаки: либо Другую половину отрезка , на которой функция знак не меняет, отбрасываем. Новый интервал вновь делим пополам, получаем очередное приближение к корню: . и т.д.

***Рабочая формула метода:***

*Критерий окончания итерационного процесса:* или

*Приближенное значение корня:* , или , или .

*Визуализация метода:* 

**Метод секущих:**

*Идея метода:* упростим метод Ньютона, заменив разностным приближением:

Метод секущих является двухшаговым, т.е. новое приближение определяется двумя предыдущими итерациями и . Выбор определяется, как и в методе Ньютона,

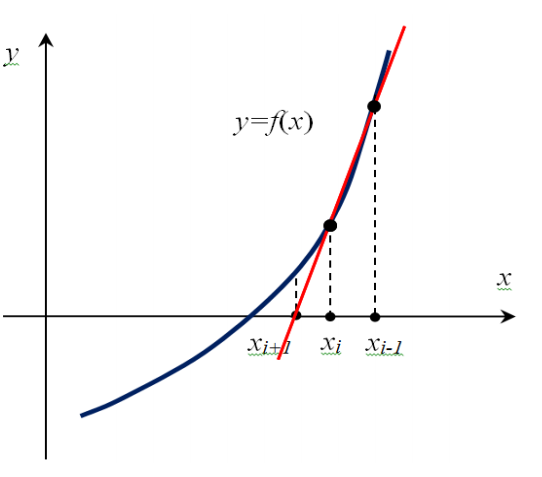
- выбирается рядом с начальным самостоятельно.

***Рабочая формула метода:***

*Критерий окончания итерационного процесса:* или

*Приближенное значение корня:* .

*Визуализация метода:*



**Метод простых итераций:**

*Идея метода:* уравнение приведем к эквивалентному виду:

Выбор начального приближения:

***Рабочая формула метода:***

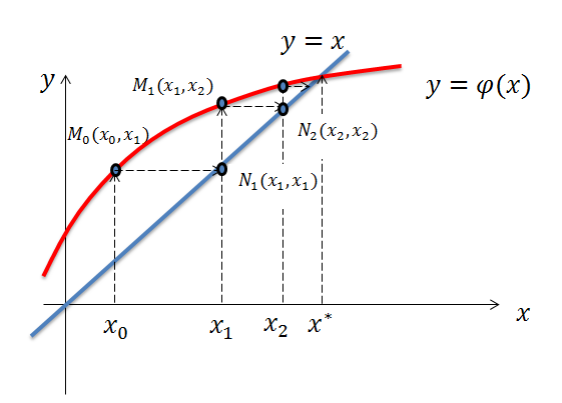
Условия сходимости метода простой итерации определяются следующей теоремой. ***Теорема.*** Если в некоторой 𝜎 - окрестности корня уравнения функция дифференцируема и удовлетворяет неравенству , где - постоянная, то независимо от выбора начального приближения из указанной 𝜎 - окрестности итерационная последовательность не выходит из этой окрестности, метод сходится со скоростью геометрической прогрессии.

*Достаточное условие сходимости метода:* , где 𝑞 – некоторая константа

*Критерий окончания итерационного процесса:*

*Приближенное значение корня:* .

*Визуализация метода:*



**Листинг:**

Main.java

**package** pepe.lmao;

**import** pepe.lmao.chart.XYChart;  
**import** pepe.lmao.method.Function;  
  
**import** java.io.\*;  
**import** java.util.Random;  
**import** java.util.Scanner;  
  
**public class** Main {  
  
 **public static void** main(String[] args) **throws** IOException {  
  
 **int** inputChosenOption;  
 **int** outputChosenOption;  
  
 **double** lower = 0;  
 **double** upper = 0;  
 **double** eps = 0;  
  
 Scanner scanner = **new** Scanner(System.***in***);  
 System.***out***.println(**"Input data via:\n 1. File\n 2. Console"**);  
 inputChosenOption = scanner.nextInt();  
  
 System.***out***.println(**"Output result via:\n 1. File\n 2. Console"**);  
 outputChosenOption = scanner.nextInt();  
  
 Function function = **new** Function();  
 **switch** (inputChosenOption) {  
 **case** 1:  
 String row;  
 String[] data;  
 BufferedReader reader = **new** BufferedReader(**new** FileReader(**"input.txt"**));  
 **while** ((row = reader.readLine()) != **null**) {  
 data = row.split(**","**);  
 lower = Double.*parseDouble*(data[0]);  
 upper = Double.*parseDouble*(data[1]);  
 eps = Double.*parseDouble*(data[2]);  
 }  
 **break**;  
 **case** 2:  
 System.***out***.println(**"Set lower bound: "**);  
 lower = scanner.nextDouble();  
 System.***out***.println(**"Set upper bound: "**);  
 upper = scanner.nextDouble();  
 System.***out***.println(**"Set accuracy: "**);  
 eps = scanner.nextDouble();  
 **break**;  
 **default**:  
 System.***out***.println(**"Unexpected value: "** + inputChosenOption);  
 **break**;  
 }  
 **switch** (outputChosenOption) {  
 **case** 1:  
 *writeUsingBufferedWriter*(function.solveToFile(lower, upper, eps));  
 System.***out***.println(**"-->Done! Check the file: "** + *filename*);  
 **break**;  
 **case** 2:  
 function.solve(lower, upper, eps);  
 **break**;  
 **default**:  
 System.***out***.println(**"Unexpected value: "** + inputChosenOption);  
 **break**;  
 }  
  
 XYChart xyChart = **new** XYChart();  
 xyChart.plot(lower, upper);  
 xyChart.pack();  
 xyChart.setVisible(**true**);  
 }  
  
 **private static** String *filename* = **""**;  
 **private static void** writeUsingBufferedWriter(String data)  
 **throws** IOException {  
 Random random = **new** Random();  
 *filename* = **"output\_"** + Math.*abs*(random.nextInt()) + **".txt"**;  
 BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(*filename*));  
 writer.write(data);  
 writer.close();  
 }  
}

Function.java

**package** pepe.lmao.method;

**public class** Function {  
  
  
 **public void** solve(**double** lower, **double** upper, **double** eps) {  
  
 **if** (f(lower) \* f(upper) <= 0) {  
 HalfSeparation halfSeparation = **new** HalfSeparation();  
 halfSeparation.result(lower, upper, eps, **true**);  
 } **else** {  
 System.***out***.println(**"-->Half Separation Method Exception:\n Invalid bounds"**);  
 }  
  
 **if** (f(lower) \* f(upper) <= 0) {  
 Secant secant = **new** Secant();  
 secant.result(lower, upper, eps, **true**);  
 } **else** {  
 System.***out***.println(**"-->Secant Method Exception:\n Invalid bounds"**);  
 }  
  
 **if** (Math.*abs*(Math.*max*(derivative(**"'"**, lower), derivative(**"'"**, upper))) >= 1) {  
 SimpleIteration simpleIteration = **new** SimpleIteration();  
 simpleIteration.result(lower, upper, eps, **true**);  
 } **else** {  
 System.***out***.println(**"-->Simple Iteration Method Exception:\n Invalid bounds"**);  
 }  
 }  
  
 **public** String solveToFile(**double** lower, **double** upper, **double** eps) {  
 StringBuilder stringBuilder = **new** StringBuilder();  
 **if** (f(lower) \* f(upper) <= 0) {  
 HalfSeparation halfSeparation = **new** HalfSeparation();  
 stringBuilder.append(halfSeparation.result(lower, upper, eps, **false**));  
  
 } **else** {  
 stringBuilder.append(**"-->Half Separation Method Exception:\n Invalid bounds"**);  
 }  
  
 **if** (f(lower) \* f(upper) <= 0) {  
 Secant secant = **new** Secant();  
 stringBuilder.append(secant.result(lower, upper, eps, **false**));  
  
 } **else** {  
 stringBuilder.append(**"-->Secant Method Exception:\n Invalid bounds"**);  
 }  
  
 **if** (Math.*abs*(Math.*max*(derivative(**"'"**, lower), derivative(**"'"**, upper))) >= 1) {  
 SimpleIteration simpleIteration = **new** SimpleIteration();  
 stringBuilder.append(simpleIteration.result(lower, upper, eps, **false**));  
 } **else** {  
 stringBuilder.append(**"-->Simple Iteration Method Exception:\n Invalid bounds"**);  
 }  
 **return** stringBuilder.toString();  
 }  
  
 **public double** f(**double** x) {  
 **return** Math.*pow*(x, 3) - 3.78 \* Math.*pow*(x, 2) + 1.25 \* x + 3.49;  
 }  
  
 **public double** derivative(String n, **double** x) {  
 **double** res;  
 **switch** (n) {  
 **case "'"**:  
 res = 3 \* Math.*pow*(x, 2) - 7.56 \* x + 1.25;  
 **case "''"**:  
 res = 6 \* x - 7.56;  
 **break**;  
 **default**:  
 **throw new** IllegalStateException(**"Unexpected value: "** + n);  
 }  
 **return** res;  
 }  
}

HalfSeparation.java

**package** pepe.lmao.method;  
  
**public class** HalfSeparation {  
 Function **function** = **new** Function();  
 **double x** = 0;  
 **int n** = 0;  
 StringBuilder **stringBuilder** = **new** StringBuilder();  
  
 **public** String result(**double** lower, **double** upper, **double** eps, **boolean** way) {  
  
 **while** (Math.*abs*(lower - upper) > eps) {  
 **x** = (upper + lower) / 2;  
 **if** (**function**.f(lower) \* **function**.f(**x**) <= 0) {  
 upper = **x**;  
 } **else** {  
 lower = **x**;  
 **x** = (upper + lower) / 2;  
 }  
 ++**n**;  
 }  
 **if** (way) {  
 System.***out***.println(**"-->Half Separation Method: "**);  
 System.***out***.println(**" x = "** + **x** + **"\n f(x) = "** + **function**.f(**x**) + **"\n Iterations: "** + **n** + **"\n"**);  
 **return "Done"**;  
 } **else** {  
 **stringBuilder**.append(**"-->Half Separation Method: \n"**);  
 **stringBuilder**.append(**" x = "**).append(**x**).append(**"\n f(x) = "**).append(**function**.f(**x**)).append(**"\n Iterations: "**).append(**n**).append(**"\n"**);  
 **return stringBuilder**.toString();  
 }  
 }  
}

Secant.java

**package** pepe.lmao.method;  
  
**import** java.util.ArrayList;  
  
**public class** Secant {  
 Function **function** = **new** Function();  
 ArrayList<Double> **x** = **new** ArrayList<>();  
 **int n** = 0;  
 StringBuilder **stringBuilder** = **new** StringBuilder();  
  
 **public** String result(**double** lower, **double** upper, **double** eps, **boolean** way) {  
 **if** (**function**.derivative(**"''"**, lower) \* **function**.f(lower) > 0)  
 **x**.add(0, lower);  
 **else  
 x**.add(0, upper);  
 **x**.add(1, **x**.get(0) - **x**.get(0) / 10);  
 **int** i = 1;  
 **while** (Math.*abs*(**x**.get(i - 1) - **x**.get(i)) > eps) {  
 **x**.add(i + 1, **x**.get(i) - (**x**.get(i) - **x**.get(i - 1)) / (**function**.f(**x**.get(i)) - **function**.f(**x**.get(i - 1))) \* **function**.f(**x**.get(i)));  
 i++;  
 **n**++;  
 }  
 **if** (way) {  
 System.***out***.println(**"-->Secant Method: "**);  
 System.***out***.println(**" x = "** + **x**.get(**x**.size() - 1) + **"\n f(x) = "** + **function**.f(**x**.get(**x**.size() - 1)) + **"\n Iterations: "** + **n** + **"\n"**);  
 **return "Done"**;  
 } **else** {  
 **stringBuilder**.append(**"\n-->Secant Method: \n"**);  
 **stringBuilder**.append(**" x = "**).append(**x**.get(**x**.size() - 1)).append(**"\n f(x) = "**).append(**function**.f(**x**.get(**x**.size() - 1))).append(**"\n Iterations: "**).append(**n**).append(**"\n"**);  
 **return stringBuilder**.toString();  
 }  
  
 }  
  
  
}

SimpleIteration.java

**package** pepe.lmao.method;  
  
**import** java.util.ArrayList;  
  
**public class** SimpleIteration {  
 Function **function** = **new** Function();  
 ArrayList<Double> **x** = **new** ArrayList<>();  
 **int n** = 0;  
  
 StringBuilder **stringBuilder** = **new** StringBuilder();  
  
 **public** String result(**double** lower, **double** upper, **double** eps, **boolean** way) {  
 **double** lambda = -1 / Math.*max*(**function**.derivative(**"'"**, lower), **function**.derivative(**"'"**, upper));  
 **if** (**function**.derivative(**"'"**, lower) > **function**.derivative(**"'"**, upper))  
 **x**.add(0, lower);  
 **else  
 x**.add(0, upper);  
  
 **x**.add(1, **x**.get(0) + lambda \* (**function**.f(**x**.get(0))));  
 **int** i = 1;  
 **while** (Math.*abs*(**x**.get(i) - **x**.get(i - 1)) > eps) {  
 **x**.add(i + 1, **x**.get(i) + (lambda \* **function**.f(**x**.get(i))));  
 **n**++;  
 i++;  
 }  
 **if** (way) {  
 System.***out***.println(**"-->Simple Iteration Method: "**);  
 System.***out***.println(**" x = "** + **x**.get(**x**.size() - 1) + **"\n f(x) = "** + **function**.f(**x**.get(**x**.size() - 1)) + **"\n Iterations: "** + **n** + **"\n"**);  
 **return "Done"**;  
 } **else** {  
 **stringBuilder**.append(**"\n-->Simple Iteration Method: \n"**);  
 **stringBuilder**.append(**" x = "**).append(**x**.get(**x**.size() - 1)).append(**"\n f(x) = "**).append(**function**.f(**x**.get(**x**.size() - 1))).append(**"\n Iterations: "**).append(**n**).append(**"\n"**);  
 **return stringBuilder**.toString();  
 }  
  
 }  
}

XYChart.java

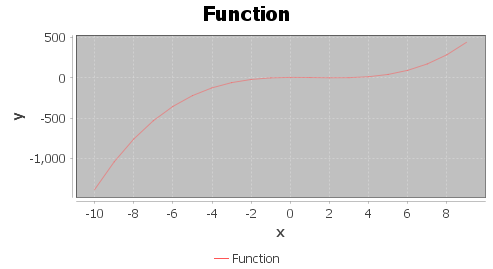
**package** pepe.lmao.chart;  
  
**import** org.jfree.chart.ChartFactory;  
**import** org.jfree.chart.ChartPanel;  
**import** org.jfree.chart.JFreeChart;  
**import** org.jfree.chart.plot.XYPlot;  
**import** org.jfree.data.xy.XYSeries;  
**import** org.jfree.data.xy.XYSeriesCollection;  
**import** pepe.lmao.method.Function;  
  
**import** javax.swing.\*;  
  
**public class** XYChart **extends** JFrame {  
 Function **function** = **new** Function();  
 **final** XYSeries **series** = **new** XYSeries(**"Function"**);  
 **final** XYSeriesCollection **dataset** = **new** XYSeriesCollection();  
  
 **public** XYPlot plot(**double** lower, **double** upper) {  
 **for** (**int** i = (**int**) lower; i < upper; i++) {  
 **series**.add(i, **function**.f(i));  
 }  
  
 **dataset**.addSeries(**series**);  
 JFreeChart chart = ChartFactory.*createXYLineChart*(**"Function"**, **"x"**, **"y"**, **dataset**);  
 ChartPanel chartPanel = **new** ChartPanel(chart);  
 chartPanel.setPreferredSize(**new** java.awt.Dimension(500, 270));  
 setContentPane(chartPanel);  
 XYPlot plot = chart.getXYPlot();  
 plot.setForegroundAlpha(0.5f);  
 **return** plot;  
 }  
}

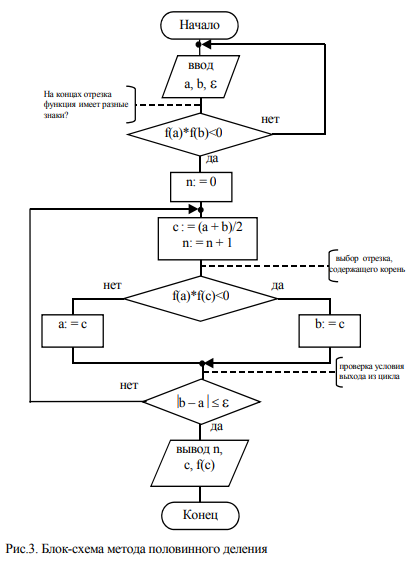
**Примеры работы программы:**

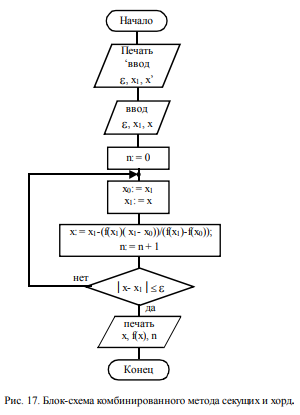
|  |  |
| --- | --- |
| Input data via:  1. File  2. Console  1  Output result via:  1. File  2. Console  2  -->Half Separation Method:  x = -0.7501220703125  f(x) = 0.003324189929335475  Iterations: 15  -->Secant Method:  x = -0.7505081385585306  f(x) = -3.5974866730015265E-7  Iterations: 11  -->Simple Iteration Method:  x = -0.7550765081348058  f(x) = -0.03947657333380228  Iterations: 27 | Input data via:  1. File  2. Console  2  Output result via:  1. File  2. Console  2  Set lower bound:  -10  Set upper bound:  10  Set accuracy:  0.0001  -->Half Separation Method:  x = -0.7505035400390625  f(x) = 3.925006409311038E-5  Iterations: 18  -->Secant Method:  x = -0.7505081385585306  f(x) = -3.5974866730015265E-7  Iterations: 11  -->Simple Iteration Method:  x = -0.7509498832172986  f(x) = -0.0038065613870594817  Iterations: 40 |
| Input data via:  1. File  2. Console  2  Output result via:  1. File  2. Console  1  Set lower bound:  1  Set upper bound:  4  Set accuracy:  0.000001  -->Done! Check the file: output\_151207941.txt | Input data via:  1. File  2. Console  1  Output result via:  1. File  2. Console  1  -->Done! Check the file: output\_1552421819.txt |
|  |  |

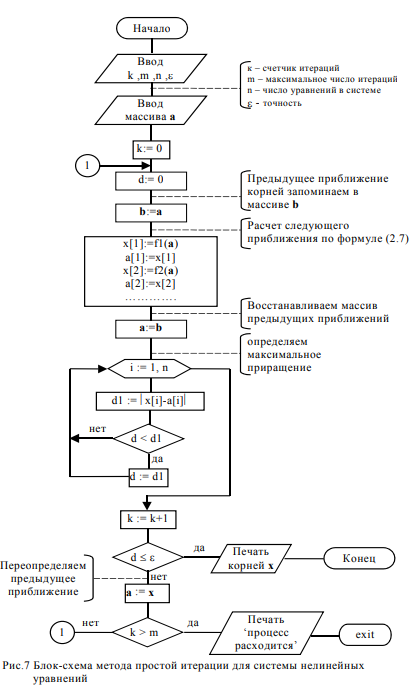
**Блок-схема:**

**Пример графика:**



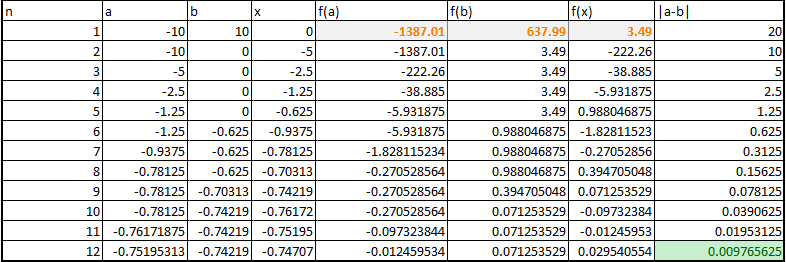
**Блок-схемы:**



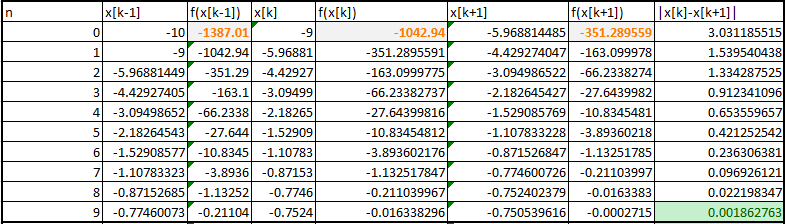


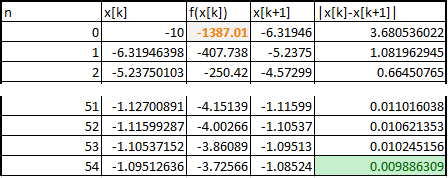
**Таблицы :**

*Уточнение корня уравнения методом половинного деления*



*Уточнение корня уравнения методом секущих*



*Уточнение корня уравнения методом простых итераций*

**Вывод:** в результате выполнения работы я познакомился с методами решения нелинейных уравнений путем реализации Java-программы. Освоил череду методов для реализации задач.