**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по практической работе №3**

**по дисциплине «Вычислительная математика»**

**Тема: Сравнительный анализ применения методов Ньютона, простых итераций, хорд для решения задач**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Гайфутдинов А.Р. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

2025

## Цель работы.

Найти корень уравнения методами Ньютона, простых итераций, хорд. Исследовать скорости сходимости и обусловленности методов.

**Основные теоретические положения.**

**Метод Ньютона.** В случае, когда известно хорошее начальное приближение решения уравнения , эффективным методом повышения точности является метод *Ньютона*. Он состоит в построении итерационной последовательности (1)

(1)

где Последовательность сходится к корню уравнения . По теореме о сходимости метода *Ньютона* должен быть простым корнем уравнения в отсекающем промежутке этого корня функция – дважды непрерывно дифференцируема и

Для оценки погрешности *n*-го приближения корня предлагается пользоваться неравенством (2)

(2)

где - наибольшее значение модуля второй производной на отрезке [a,b]; - наименьшее значение модуля первой производной на отрезке [a,b]. Таким образом, если , то . Это означает, что при хорошем начальном приближении корня после каждой итерации число верных десятичных знаков в очередном приближении удваивается, т.е. процесс сходится очень быстро и имеет место квадратическая сходимость.

Если необходимо найти корень с точностью **, то итерационный процесс можно прекращать, когда выполняется неравенство (3)

(3)

Если на *(n-1)*-м шаге очередное приближение не удовлетворяет условию окончания процесса, то вычисляются величины и следующие приближение корня . При выполнении условия (3) величина принимается за приближенное значение корня *с*, вычисленное с точностью **.

**Метод простых итераций.**

Метод *простых итераций* решения уравнения заключается в замене исходного уравнения эквивалентным ему уравнением и построении последовательности , сходящейся при к точному решению

Корень уравнения является точкой пересечения двух графиков и . Сходимость метода зависит от вида функции . В зависимости от величины модуля первой производной метод может сходиться и расходиться.

Достаточные условия сходимости метода *простых итераций* формулируются следующей теоремой:

Если функция определена на отрезке , дифференцируема, то существует число, такое что на , и последовательность,сходится к единственному решению на уравнения при (1)

(1)

Если , то |, если , то |.

Рассмотрим один шаг итерационного процесса. Исходя из найденного на предыдущем шаге значения , вычисляется . Если , то полагается и выполняется очередная итерация. Если же , то вычисления заканчиваются и за приближенное значение корня принимается величина . Погрешность результата вычислений зависит от знака производной : при : погрешность определения корня составляет , а при , погрешность не превышает . Здесь - число, такое, что на отрезке [a,b]. Существование числа является условием сходимости метода в соответствии с отмеченной выше теоремой.

Для применения метода *простых итераций* определяющее значение имеет выбор функции , в уравнении , эквивалентном исходному. Функцию необходимо подбирать так, чтобы . Это обусловливается тем, что если , на отрезке [a,b], то последовательные приближения . будут колебаться около корня *c*, если же , то последовательные приближения будут сходиться к корню *c* монотонно.

Число обусловленности метода *простых итераций* (2)

**Метод хорд**

Пусть найден отрезок , на котором функция меняет знак. Для определенности положим >0, . В методе *хорд* процесс итераций состоит в том, что в качестве приближений к корню уравнения =0 принимаются значения . . . точек пересечения хорды с осью абсцисс, как это показано на рисунке 2.

с

Рисунок 2 – Построение хорд в используемом методе

Сначала находится уравнение хорды АВ (прямой, проходящей через 2 точки): . Для точки пересечения ее с осью абсцисс получается уравнение . Далее сравниваются знаки величин и и для рассматриваемого случая оказывается, что корень находится в интервале , так как . Отрезок отбрасывается. Следующая итерации состоит в определении нового приближения - точки пересечения хорды с осью абсцисс и т.д. Итерационный процесс продолжается до тех пор, пока значение не станет по модулю меньше заданного числа .

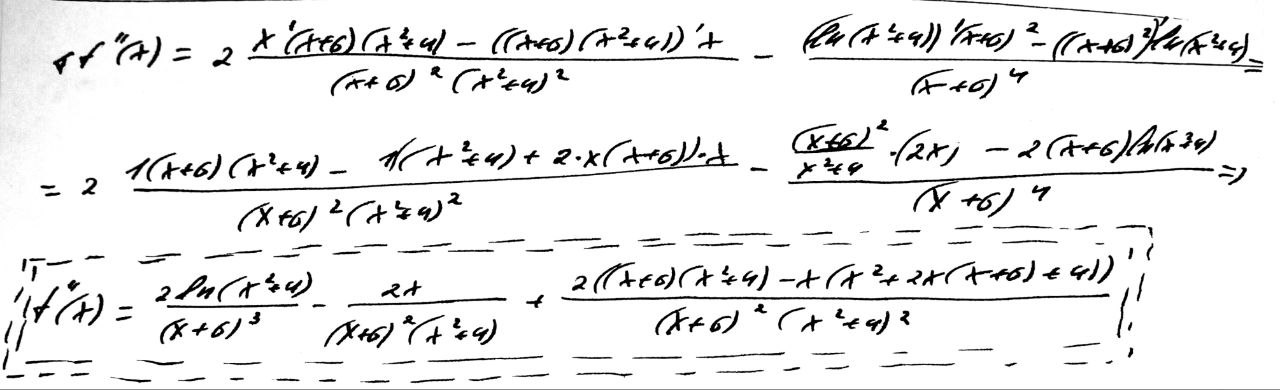
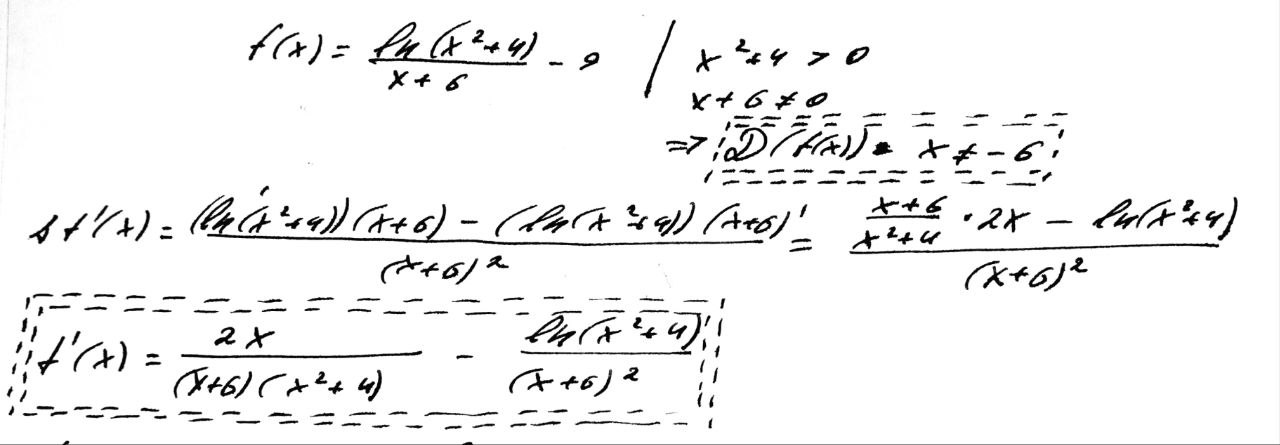
Алгоритмы методов *бисекции* и *хорд* похожи, однако метод хорд в ряде случаев дает более быструю сходимость итерационного процесса, причем успех его применения, как и метода *бисекции*, гарантирован. Метод обладает линейной сходимостью.

## Выполнение работы.

**Вариант 6.**

|  |  |
| --- | --- |
| 6 | [ln(x^(2)+4)/(x+6)]-9 |

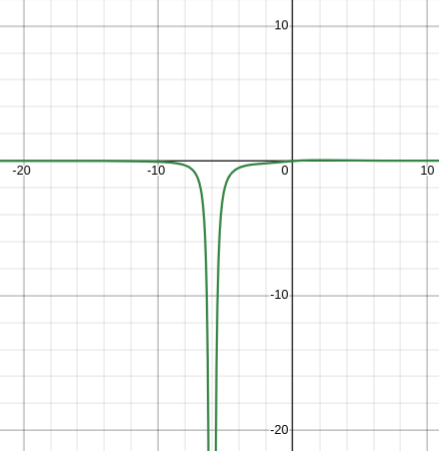
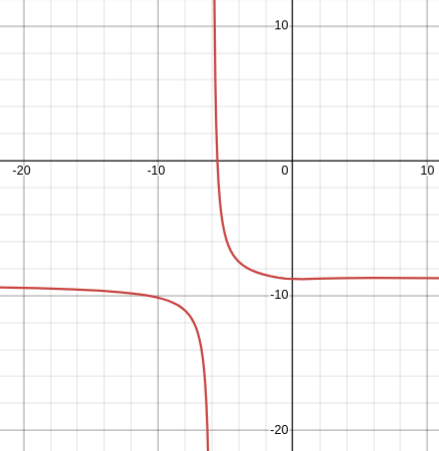
**1) Нахождение области определения функции, локализация корня уравнения 𝑓(𝑥) = 0, взятие первой и второй производных вручную, нахождение их минимума и максимума.**

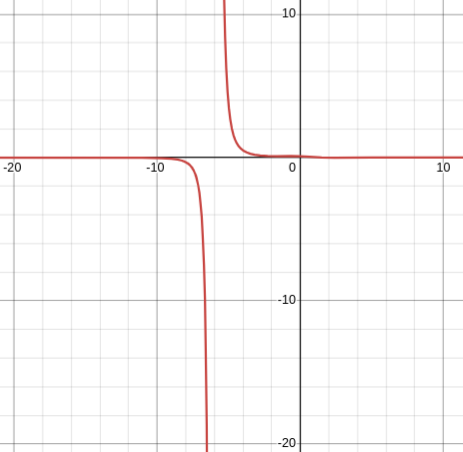


Область определения функции : x != 6.

Выберем отрезок локализации [-5.65, -5.55].

Графики функции, ее первой и второй производной:



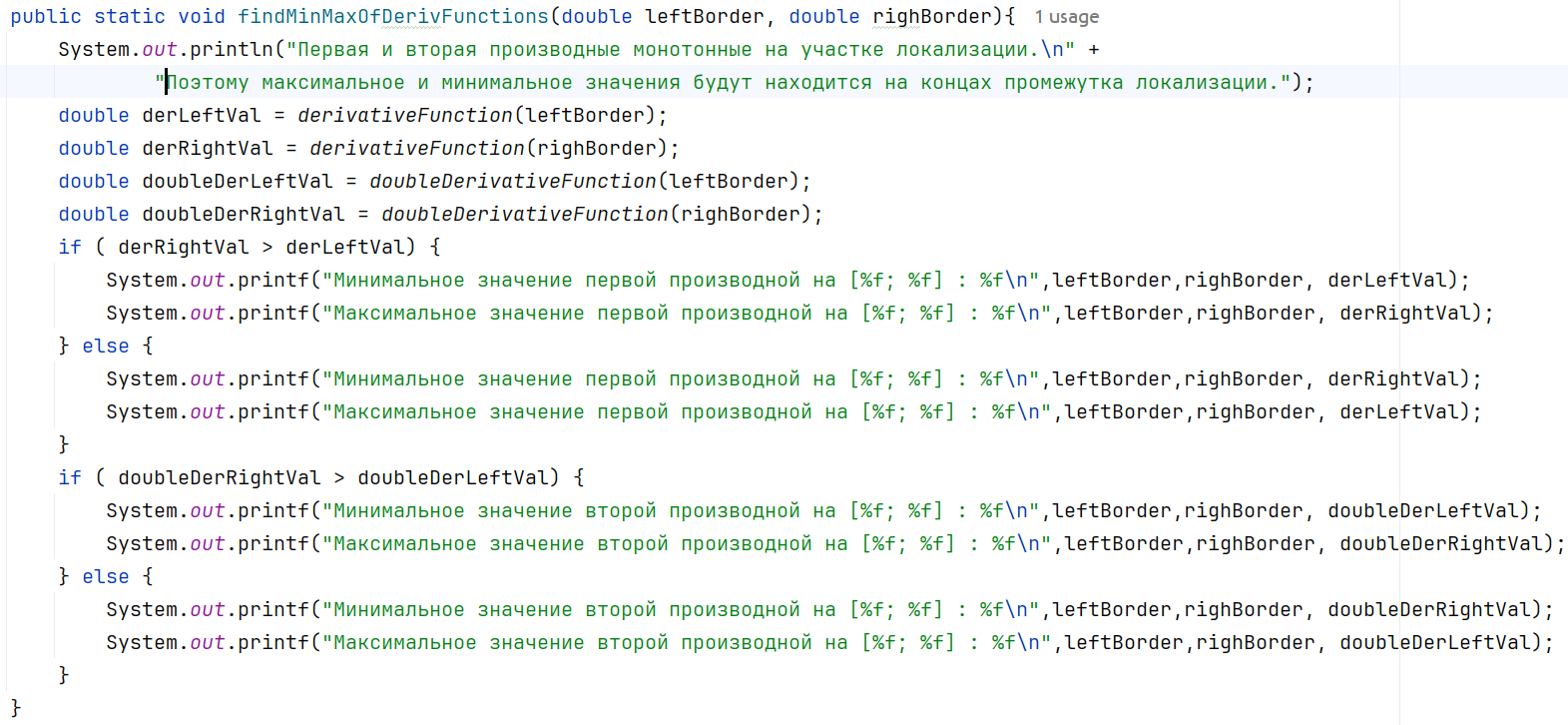
= (2\*x)/((x+6)\*(x^2+4))-(log(x^2+4))/(((x+6))^2)

= (2\*log(x^2+4))/(((x+6))^3)-(2\*x)/(((x+6))^2\*(x^2+4))+(2\*((x+6)\*(x^2+4)-x\*(x^2+2\*x\*(x+6)+4)))/(((x+6))^2\*((x^2+4))^2)

Функция и производные непрерывны на данном промежутке локализации.

Функция и вторая производная убывают на промежутке локализации.

Написана функция по вычислению и выводу на экран минимума и максимума первой и второй производной.



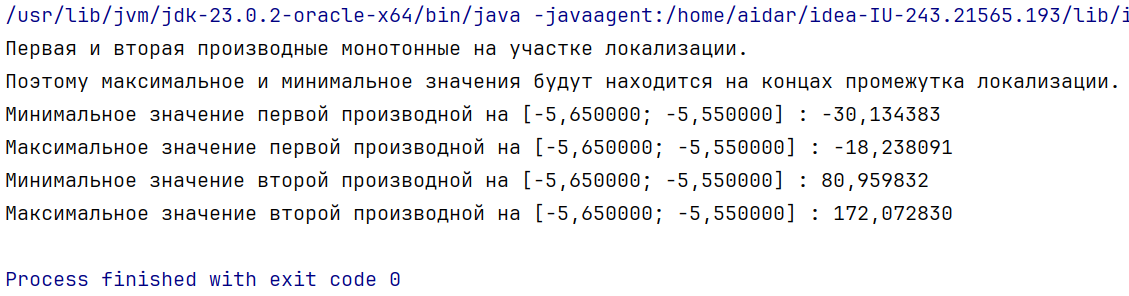
Первая производная возрастает на промежутке локализации.

Min() = -30,134383

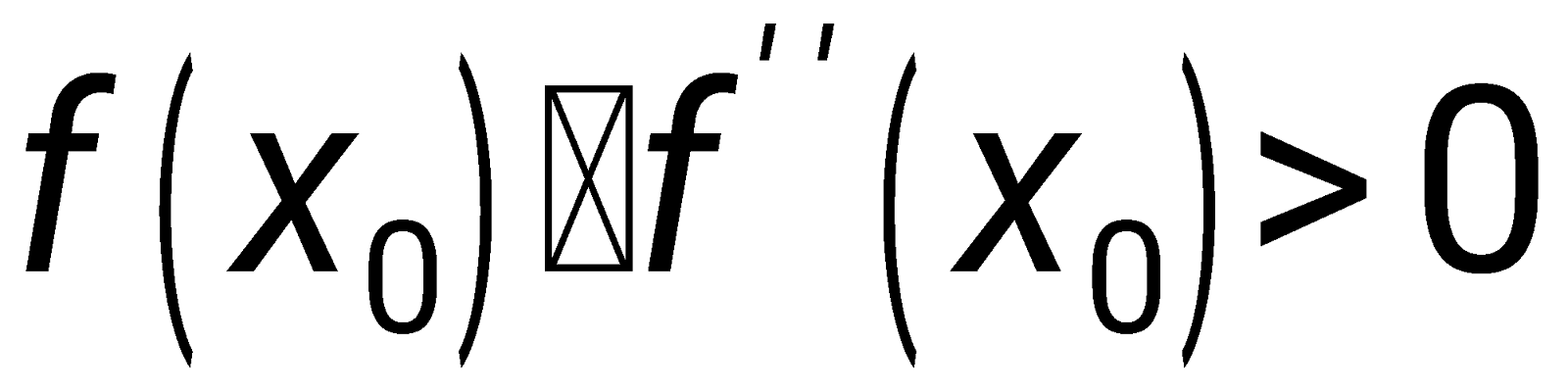
Max() = -18,238091

Min() = 80,959832

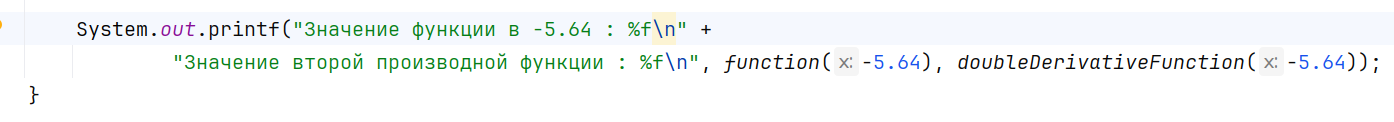
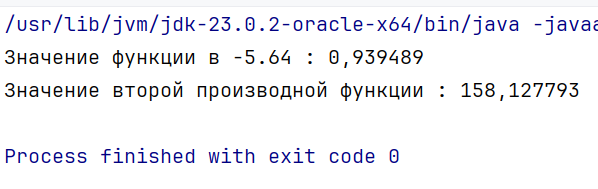
Max() = 172,072830

**2) Проверка данных для метода Ньютона:**

**a)** Функция выпукла вниз на промежутке локализации [-5.65; -5.55]

Рассмотрим следующее х в качестве начального приближения, соответствующее требованию .

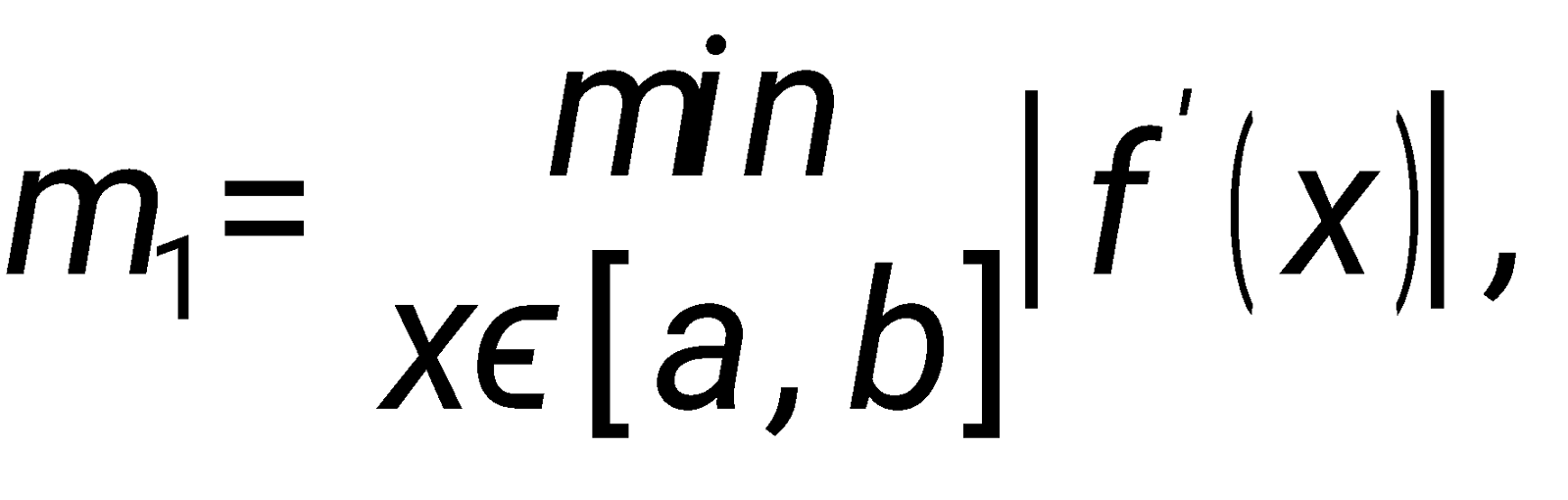
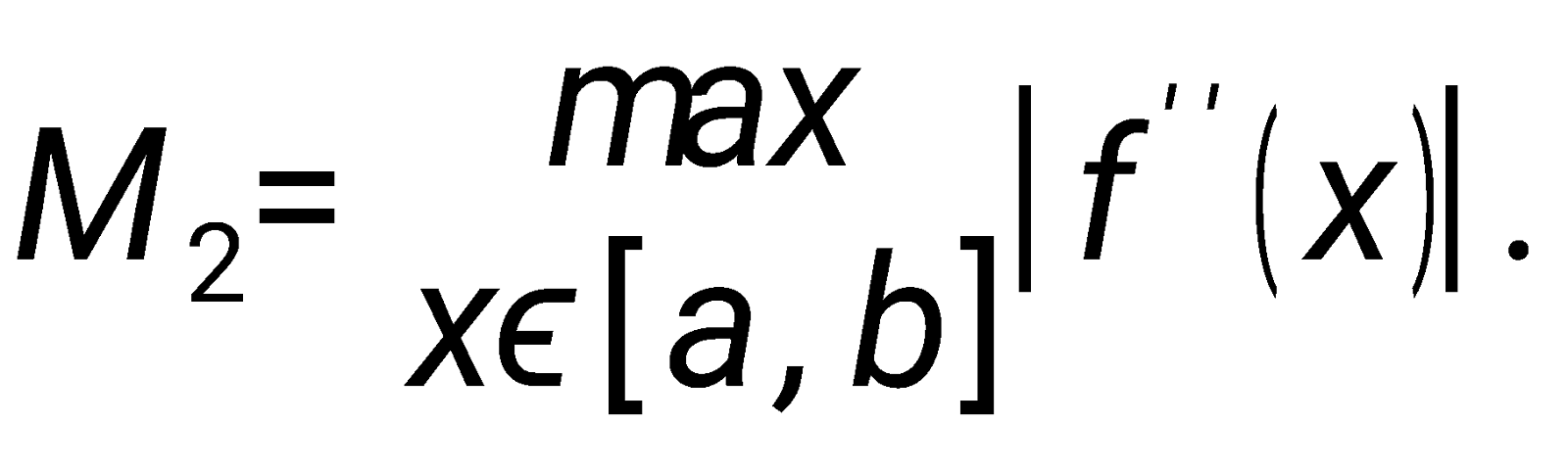
x = -5,64



f(-5,6) ≈ 0,939489

f’’(-5,6) ≈ 158,127793

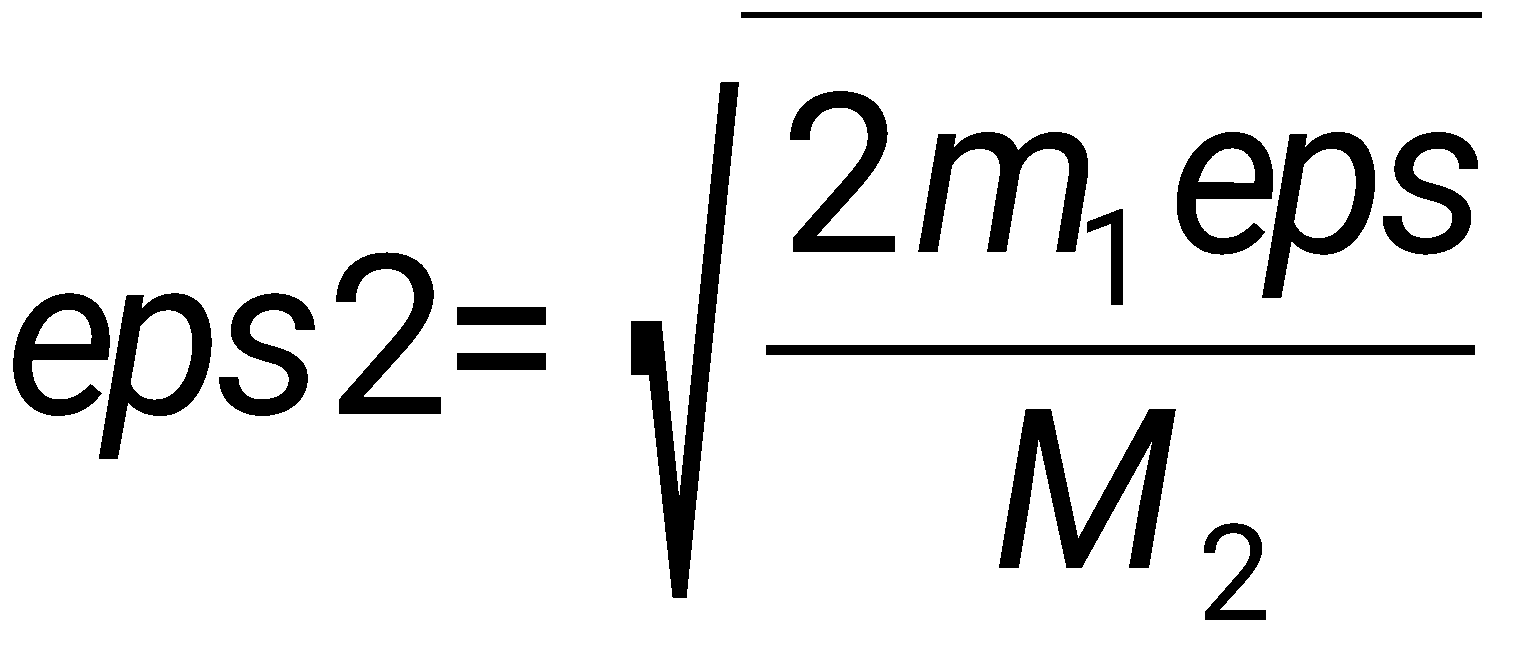
Эти данные абсолютно соответствуют полученным графикам.

**b)**  

m1 = -18,238091

M2 = 172,072830

**c)** Условие окончания итерационного процесса варьируется от заданного eps:

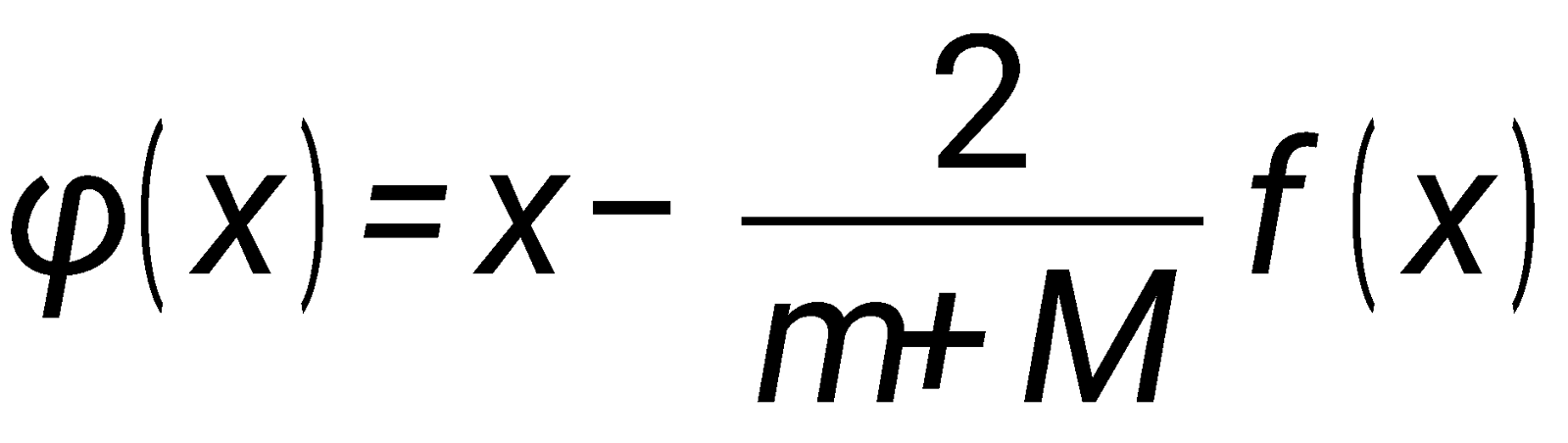
.

**3) Проверка данных для метода простых операций:**

a. m = Min() = -30,134383

M = Max() = -18,238091

b. Преобразование уравнения к виду, удобному для итераций:

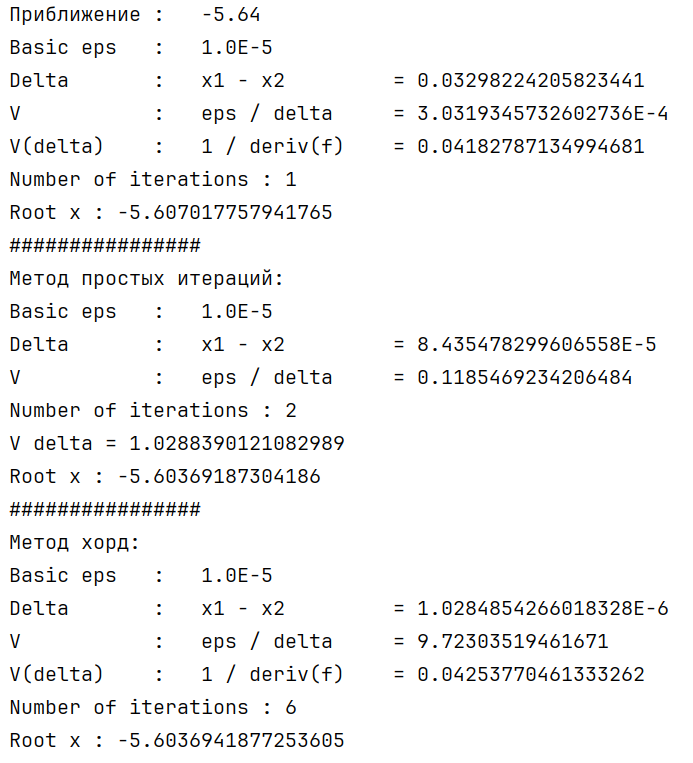


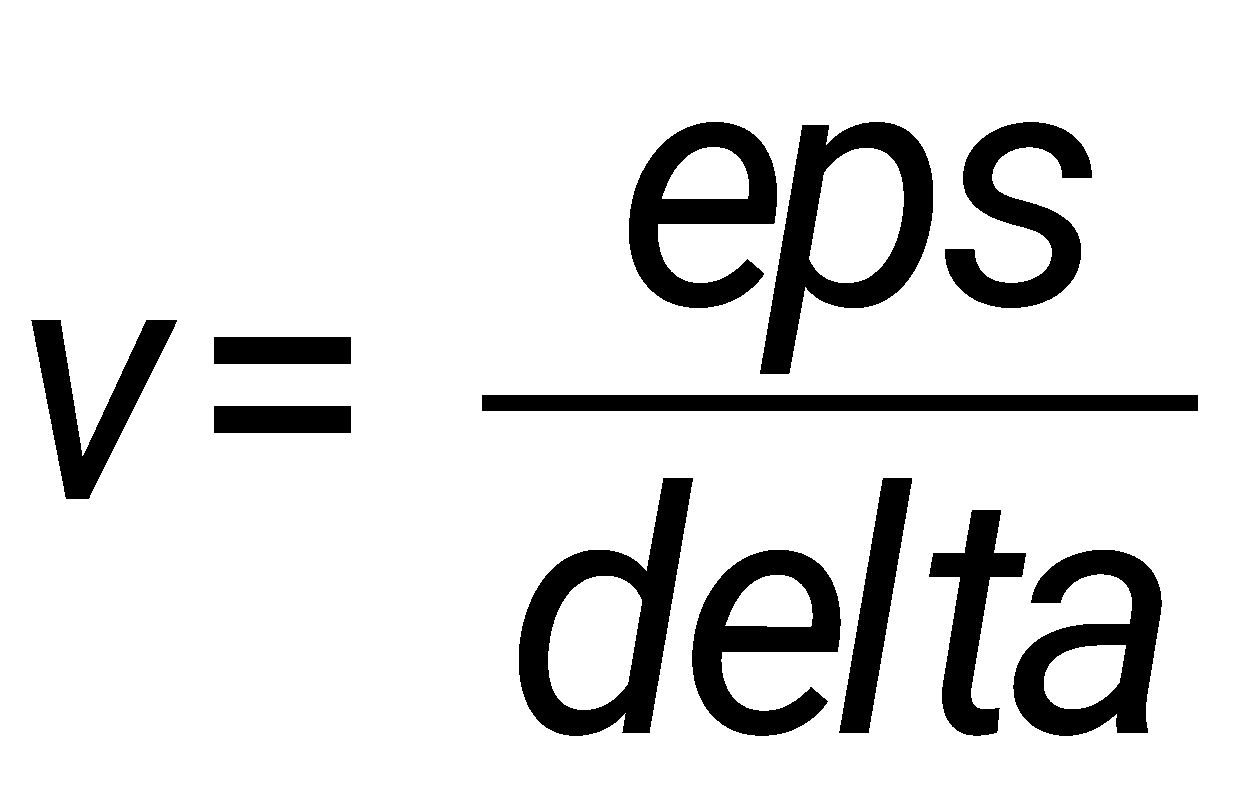
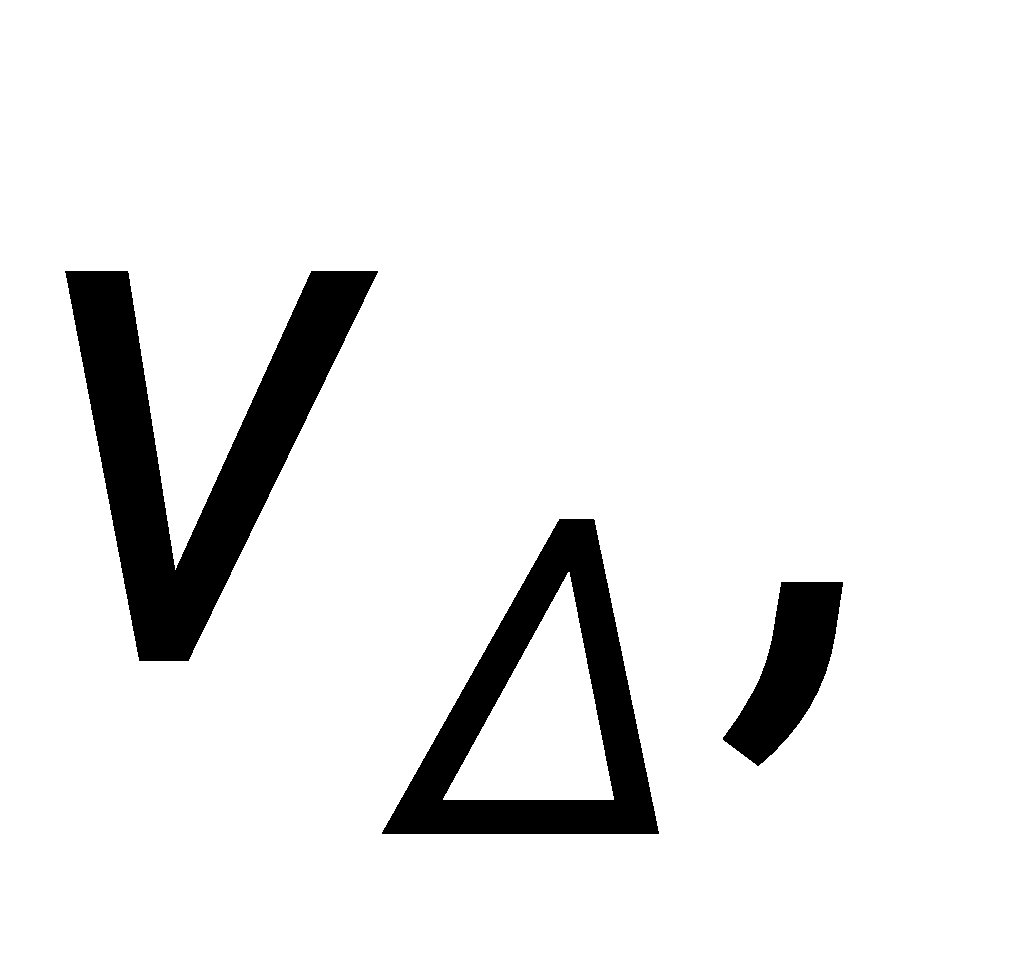
φ(x) = x - 2 / (-30,134383 -18,238091) \* ((ln(x^(2)+4)/(x+6))-9)

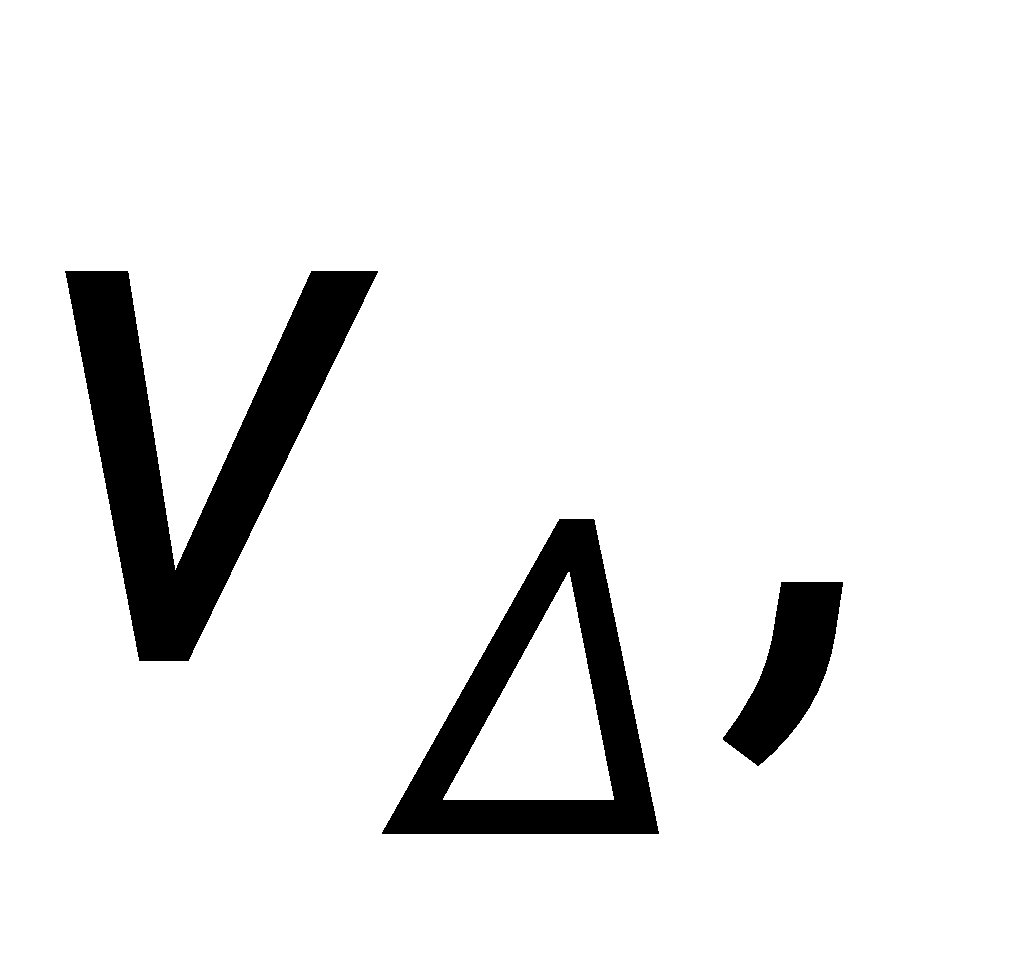
**4) Написание программы, включающей метод нахождения функции, производной на Java.**

Согласно заданию был реализован класс с необходимыми методами для нахождения корня уравнения. Код см. в приложениии.

Пример работы программы при eps = 0.00001:



**5) Вычисление** , **сравнение его с**  **получение вывода об обусловленности задачи.**

 = 1/|f’(x)|

Реализована функция по нахождению соответствующих v и vdelta. Результаты см в таблице к заданию 7.

**6) Поменять условие окончания итераций на N (большое значение) и поймать интервал неопределенности по правилу Гарвика.**

Реализованы методы-дубликаты методов Ньютона, простых итераций и хорд с условием окончания итерация на N (=100). Для каждого случая был найден интервал неопределенности (разболтки) по правилу Гарвика.

***Метод Ньютона:***

**Разболтка началась после 6 итераций.**

**Приближенное значение корня = -5,6036943**

***Метод Простых итераций:***

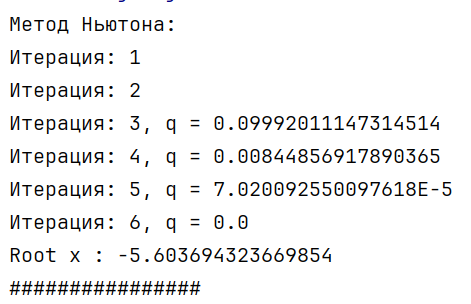
**Разболтка началась после 10 итераций.**

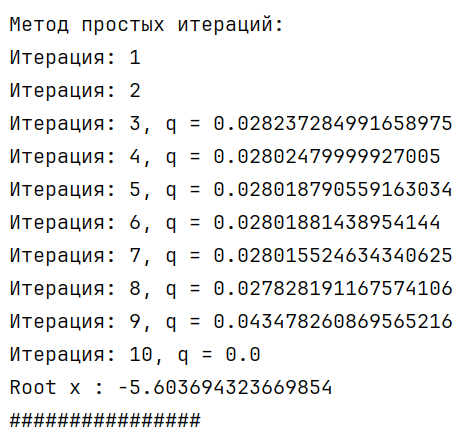
**Приближенное значение корня = -5,6036943**

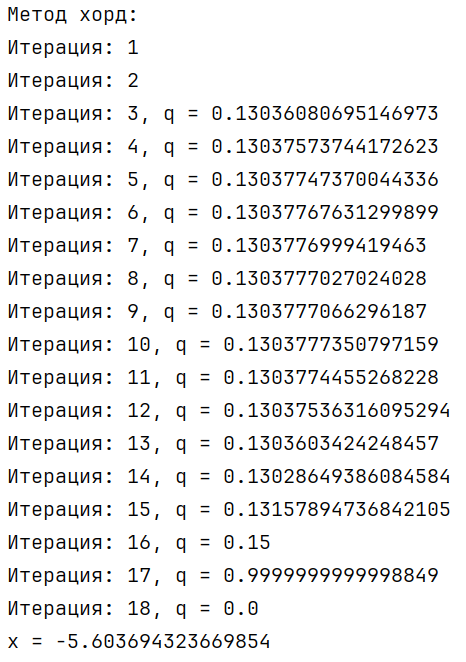
***Метод Хорд:***

**Разболтка началась после 18 итераций.**

**Приближенное значение корня = -5,6036943**



****

****

**7) Для каждого метода eps, который является условием остановки итераций меняем в диапазоне от 0.00001 и 0.1 и заполняем таблицу**

**Метод Ньютона:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *eps* | *delta* |  |  |  |  | Обусловленность |
| 0.00001 | 0.03298224205823441 | -5.607017757941765 | 1 | 0.04182787134994681 | 3.0319345732602736E-4 | Хорошая |
| 0.0001 | 0.03298224205823441 | -5.607017757941765 | 1 | 0.04182787134994681 | 0.003031934573260274 | Хорошая |
| 0.001 | 0.03298224205823441 | -5.607017757941765 | 1 | 0.04182787134994681 | 0.003031934573260274 | Хорошая |
| 0.01 | 0.03298224205823441 | -5.607017757941765 | 1 | 0.04182787134994681 | 0.003031934573260274 | Хорошая |
| 0.1 | 0.03298224205823441 | -5.607017757941765 | 1 | 0.04182787134994681 | 0.003031934573260274 | Хорошая |

**Метод простых итераций:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *eps* | *delta* |  |  |  |  | Обусловленность |
| 0.00001 | 8.435478299606558E-5 | -5.60369187304186 | 2 | 1.0288390121082989 | 0.1185469234206484 | Хорошая |
| 0.0001 | 0.03298224205823441 | -5.607017757941765 | 2 | 1.0288390121082989 | 1.185469234206484 | Хорошая |
| 0.001 | 0.002451466056728968 | -5.603607518258864 | 1 | 1.029276631025491 | 0.407919170349157 | Хорошая |
| 0.01 | 0.002451466056728968 | -5.603607518258864 | 1 | 1.029276631025491 | 4.07919170349157 | Хорошая |
| 0.1 | 0.002451466056728968 | -5.603607518258864 | 1 | 1.029276631025491 | 40.7919170349157 | Хорошая |

**Метод хорд:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *eps* | *delta* |  |  |  |  | Обусловленность |
| 0.00001 | 1.0284854266018328E-6 | -5.6036941877253605 | 6 | 0.04253770461333262 | 9.72303519461671 | Хорошая |
| 0.0001 | 8.80947214287886E-6 | -5.603693159239934 | 5 | 0.04253792519438549 | 11.351417925855529 | Хорошая |
| 0.001 | 7.5457377732846E-5 | -5.603684349767791 | 4 | 0.04253981460051375 | 13.252514598909896 | Хорошая |
| 0.01 | 6.463298258818639E-4 | -5.603608892390058 | 3 | 0.04255599998927359 | 15.471976689851536 | Хорошая |
| 0.1 | 0.042694761614882525 | -5.602962562564176 | 2 | 0.042694761614882525 | 18.062905076289052 | Хорошая |

**8)Сделать сравнительную таблицу, результаты Бисекции можно взять из предыдущей работы.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод  Характеристики | Бисекция | Ньютон | Простых итераций | Хорд |
| Многошаговый или одношаговый | Одношаговый | Одношаговый | Одношаговый | Одношаговый |
| Скорость сходимости | Линейнаяа | Квадратичная | Линейная | Линейная |
| Априорная погрешность | |b-a|  /(2^(n+1)|<epsilon | |x(n) - x(\*)| ≤ C|x(n-1)- x(\*)|^2 | |x(n-1) - x(\*)| ≤ M / 2 |x(n-1) - x(\*)|^2 | |x(n-1) — x(\*)| ≤ q^n|x(0)- x(\*)| |
| Апостериорная погрешность | |x(n)-x(n-1)|≤  2\*epsilon | |x(n)-x(n-1)|≤  √(2\*m1\*eps/M2) | |x(n)-x(\*)|≤  (1-q)\*epsilon/q | |f(x(n))| < eps |
| Номер итерации при начале разболтки | 53 (больший изначально взятый интервал) | 6 | 10 | 16 |

**ПРИЛОЖЕНИЕ А.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ.**

**Файл NewtonSimpleIterChordMethod.java**

package com.github.chelovekkrokant;public class NewtonSimpleIterChordMethod { static double function(double x){ return (Math.log(x\*x+4) / (x+6) - 9); } static double derivativeFunction(double x){ return ((2 \* x) / ((x + 6) \* (x \* x + 4))) - ((Math.log(x \* x + 4)) / ((x + 6) \* (x + 6))); } static double doubleDerivativeFunction(double x) { return (2 \* Math.log(x \* x + 4)) / Math.pow(x + 6, 3) - (2 \* x) / (Math.pow(x + 6, 2) \* (x \* x + 4)) + (2 \* ((x + 6) \* (x \* x + 4) - x \* (x \* x + 2 \* x \* (x + 6) + 4))) / (Math.pow(x + 6, 2) \* Math.pow(x \* x + 4, 2)); } static void getV (double x1, double x2, double eps) { double delta = Math.abs(x1 - x2); System.out.println("Basic eps : " + eps); System.out.println("Delta : x1 - x2 = " + delta); System.out.println("V : eps / delta = " + eps / delta); } static void getVDelta (double x) { System.out.println("V(delta) : 1 / deriv(f) = " + 1 / Math.abs(derivativeFunction(x))); } static double newtonMethod(double initialGuess, double precision) { System.out.println("Метод Ньютона:"); System.out.println("Приближение : " + initialGuess); *// Константы метода*final double LOWER\_M = -18.238091; final double UPPER\_M = 172.072830; *// Вычисление адаптированной точности*final double adaptedPrecision = Math.sqrt(2 \* LOWER\_M \* precision / UPPER\_M); double currentX = initialGuess; double previousX = 0; int iterationCount = 0; do { iterationCount++; double functionValue = function(currentX); *// Проверка на точное решение*if (functionValue == 0.0) { processResult(currentX, previousX, precision, iterationCount); return currentX; } double derivativeValue = derivativeFunction(currentX); *// Проверка на невозможность продолжения*if (derivativeValue == 0.0) { return Double.NaN; } *// Вычисление и применение шага Ньютона*double newtonStep = functionValue / derivativeValue; previousX = currentX; currentX -= newtonStep; } while (Math.abs(currentX - previousX) >= adaptedPrecision); processResult(currentX, previousX, precision, iterationCount); return currentX; } *// Вспомогательный метод для обработки результатов*private static void processResult(double currentX, double previousX, double precision, int iterations) { getV(currentX, previousX, precision); getVDelta(currentX); System.out.println("Number of iterations : " + iterations); } static double phi (double x, double m, double M) { return x - 2 \* function(x) / (m + M) ; } static double phiDerivative(double x, double m, double M) { return 1 - 2 / (m + M) \* derivativeFunction(x); } static void getSimpleIterationVDelta(double x, double m, double M) { System.out.println("V delta = " + 1 / (1 - Math.abs(phiDerivative(x, m, M)))); } static boolean checkPhi(double m, double M, double step) { for (double x = -5.65; x <= -5.55; x += step) { if (phiDerivative(x, m, M) >= 1) { return false; } } return true; } static double chordMethod(double leftBound, double rightBound, double precision) { System.out.println("Метод хорд:"); *// Вычисляем значения функции на границах интервала*double leftValue = function(leftBound); double rightValue = function(rightBound); *// Проверка корректности интервала*if (leftValue \* rightValue > 0.0) { System.out.println("Интервал задан неверно"); return Double.NaN; } *// Проверка на точное решение на границах*if (leftValue == 0.0) return leftBound; if (rightValue == 0.0) return rightBound; *// Инициализация переменных*double currentApproximation = 0; double previousApproximation = 0; int iterationCount = 0; final double convergenceConstant = 1.8; double currentValue; do { iterationCount++; *// Вычисление нового приближения*currentApproximation = leftBound - (rightBound - leftBound) \* leftValue / (rightValue - leftValue); currentValue = function(currentApproximation); *// Проверка на точное решение*if (currentValue == 0.0) { processResult(currentApproximation, previousApproximation, precision, iterationCount); return currentApproximation; } *// Обновление границ интервала*if (currentValue \* leftValue < 0.0) { previousApproximation = rightBound; rightBound = currentApproximation; rightValue = currentValue; } else { previousApproximation = leftBound; leftBound = currentApproximation; leftValue = currentValue; } } while (Math.abs(currentValue) >= precision); processResult(currentApproximation, previousApproximation, precision, iterationCount); return currentApproximation; } static double chordMethodGar(double leftBound, double rightBound, int n) { System.out.println("Метод хорд (Гарвик):"); *// Вычисляем значения функции на границах интервала*double leftValue = function(leftBound); double rightValue = function(rightBound); *// Проверка корректности интервала*if (leftValue \* rightValue > 0.0) { System.out.println("Интервал задан неверно"); return Double.NaN; } *// Проверка на точное решение на границах*if (leftValue == 0.0) return leftBound; if (rightValue == 0.0) return rightBound; *// Инициализация переменных*double currentApproximation = 0; double previousApproximation = 0; int iterationCount = 0; final double convergenceConstant = 1.8; double currentValue; do { iterationCount++; *// Вычисление нового приближения*currentApproximation = leftBound - (rightBound - leftBound) \* leftValue / (rightValue - leftValue); currentValue = function(currentApproximation); *// Проверка на точное решение*if (currentValue == 0.0) { processResult(currentApproximation, previousApproximation, n, iterationCount); return currentApproximation; } *// Обновление границ интервала*if (currentValue \* leftValue < 0.0) { previousApproximation = rightBound; rightBound = currentApproximation; rightValue = currentValue; } else { previousApproximation = leftBound; leftBound = currentApproximation; leftValue = currentValue; } } while (Math.abs(currentValue) >= n); processResult(currentApproximation, previousApproximation, n, iterationCount); return currentApproximation; } static double simpleIterationsMethod(double initialGuess, double precision) { System.out.println("Метод простых итераций:"); *// Параметры метода*final double UPPER\_M = -18.238091; final double LOWER\_M = -30.134383; *// Проверка условий сходимости*if (!checkConvergenceConditions(LOWER\_M, UPPER\_M, 0.001)) { return Double.NaN; } *// Инициализация приближений*double currentApproximation = phi(initialGuess, LOWER\_M, UPPER\_M); double nextApproximation = phi(currentApproximation, LOWER\_M, UPPER\_M); int iterationCount = 0; double convergenceRate = phiDerivative(currentApproximation, LOWER\_M, UPPER\_M); *// Основной итерационный процесс*while (Math.abs(currentApproximation - nextApproximation) > (1 - convergenceRate) / convergenceRate \* precision) { iterationCount++; *// Обновление приближений*double previousApproximation = currentApproximation; currentApproximation = nextApproximation; nextApproximation = phi(currentApproximation, LOWER\_M, UPPER\_M); *// Обновление коэффициента сходимости*convergenceRate = phiDerivative(nextApproximation, LOWER\_M, UPPER\_M); } *// Обработка и вывод результатов*processIterationResult(nextApproximation, currentApproximation, precision, iterationCount); getSimpleIterationVDelta(nextApproximation, LOWER\_M, UPPER\_M); return nextApproximation; } *// Вспомогательные методы*private static boolean checkConvergenceConditions(double lowerBound, double upperBound, double tolerance) { return checkPhi(lowerBound, upperBound, tolerance); } private static void processIterationResult(double current, double previous, double precision, int iterations) { getV(current, previous, precision); System.out.println("Number of iterations : " + (iterations + 1)); } static void launchAlgsEps(){ double left = -5.65, right = -5.55; double eps = 0.00001; double x0 = -5.64; double result = newtonMethod(x0, eps); System.out.println("Root x : " + result); System.out.println("################"); result = simpleIterationsMethod(x0, eps); System.out.println("Root x : " + result); System.out.println("################"); result = chordMethod(left, right, eps); System.out.println("Root x : " + result); } public static void main(String[] args) { launchAlgsEps(); } *///////////////////////////////////////////////*public static void findMinMaxOfDerivFunctions(double leftBorder, double righBorder){ System.out.println("Первая и вторая производные монотонные на участке локализации.\n" + "Поэтому максимальное и минимальное значения будут находится на концах промежутка локализации."); double derLeftVal = derivativeFunction(leftBorder); double derRightVal = derivativeFunction(righBorder); double doubleDerLeftVal = doubleDerivativeFunction(leftBorder); double doubleDerRightVal = doubleDerivativeFunction(righBorder); if ( derRightVal > derLeftVal) { System.out.printf("Минимальное значение первой производной на [%f; %f] : %f\n",leftBorder,righBorder, derLeftVal); System.out.printf("Максимальное значение первой производной на [%f; %f] : %f\n",leftBorder,righBorder, derRightVal); } else { System.out.printf("Минимальное значение первой производной на [%f; %f] : %f\n",leftBorder,righBorder, derRightVal); System.out.printf("Максимальное значение первой производной на [%f; %f] : %f\n",leftBorder,righBorder, derLeftVal); } if ( doubleDerRightVal > doubleDerLeftVal) { System.out.printf("Минимальное значение второй производной на [%f; %f] : %f\n",leftBorder,righBorder, doubleDerLeftVal); System.out.printf("Максимальное значение второй производной на [%f; %f] : %f\n",leftBorder,righBorder, doubleDerRightVal); } else { System.out.printf("Минимальное значение второй производной на [%f; %f] : %f\n",leftBorder,righBorder, doubleDerRightVal); System.out.printf("Максимальное значение второй производной на [%f; %f] : %f\n",leftBorder,righBorder, doubleDerLeftVal); } }}

**Файл GarwickNewtonSimpleIterChordMethod.java**

package com.github.chelovekkrokant;public class GarwickNewtonSimpleIterChordMethod { static double function(double x){ return (Math.*log*(x\*x+4) / (x+6) - 9); } static double derivativeFunction(double x){ return ((2 \* x) / ((x + 6) \* (x \* x + 4))) - ((Math.*log*(x \* x + 4)) / ((x + 6) \* (x + 6))); } static double phi (double x, double m, double M) { return x - 2 / (m + M) \* *function*(x); } static double chordMethod(double a, double b, int maxIterations) { System.*out*.println("Метод хорд:"); double leftValue = *function*(a); double rightValue = *function*(b); double currentX = 0, currentY, prevX = a, prevPrevX = b; int iterationCount = 0; if (leftValue \* rightValue > 0.0) { System.*out*.println("Интервал задан неверно"); return Double.*NaN*; } if (leftValue == 0.0) return leftValue; if (rightValue == 0.0) return rightValue; while (iterationCount < maxIterations) { currentX = a - (b - a) \* leftValue / (rightValue - leftValue); currentY = *function*(currentX); if (currentY == 0.0) { return currentX; } if (iterationCount < 2) { System.*out*.println("Итерация: " + (iterationCount + 1)); } if (iterationCount >= 2) { double q = Math.*abs*(currentX - prevX) / Math.*abs*(prevPrevX - prevX); System.*out*.println("Итерация: " + (iterationCount + 1) + ", q = " + q); if (q >= 1 || currentX == prevX) return prevX; } prevPrevX = prevX; if (currentY \* leftValue < 0.0) { prevX = b; b = currentX; rightValue = currentY; } else { prevX = a; a = currentX; leftValue = currentY; } iterationCount++; } return currentX; } static double newtonMethod(double initialGuess, int maxIterations) { System.*out*.println("Метод Ньютона:"); double functionValue, derivativeValue, delta; double prevX = 0, prevPrevX = 0; int iterationCount = 0; double x = initialGuess; while (iterationCount < maxIterations) { functionValue = *function*(x); if (functionValue == 0.0) { return x; } if (iterationCount < 2) { double q = Math.*abs*(x - prevX) / Math.*abs*(prevPrevX - prevX); System.*out*.println("Итерация: " + (iterationCount + 1)); } if (iterationCount >= 2) { double q = Math.*abs*(x - prevX) / Math.*abs*(prevPrevX - prevX); System.*out*.println("Итерация: " + (iterationCount + 1) + ", q = " + q); if (q >= 1 || x == prevX) return prevX; } derivativeValue = *derivativeFunction*(x); if (derivativeValue == 0.0) return Double.*NaN*; delta = functionValue / derivativeValue; prevPrevX = prevX; prevX = x; x -= delta; iterationCount++; } return x; } static double simpleIterationsMethod(double initialX, int maxIterations) { System.*out*.println("Метод простых итераций:"); double upperBound = -18.238091; double lowerBound = -30.134383; double x0 = initialX; double x1 = *phi*(x0, lowerBound, upperBound); double x2 = *phi*(x1, lowerBound, upperBound); int iterationCount = 0; while (iterationCount < maxIterations) { if (iterationCount < 2) { double q = Math.*abs*(x2 - x1) / Math.*abs*(x0 - x1); System.*out*.println("Итерация: " + (iterationCount + 1)); } if (iterationCount >= 2) { double q = Math.*abs*(x2 - x1) / Math.*abs*(x0 - x1); System.*out*.println("Итерация: " + (iterationCount + 1) + ", q = " + q); if (q >= 1 || x2 == x1) return x1; } x0 = x1; x1 = x2; x2 = *phi*(x1, lowerBound, upperBound); iterationCount++; } return x2; } static void launchAlgsGarwick(){ double left = -5.65, right = -5.55; int n = 100; double x0 = -5.64; double result = *newtonMethod*(x0, n); System.*out*.println("Root x : " + result); System.*out*.println("################"); result = *simpleIterationsMethod*(x0, n); System.*out*.println("Root x : " + result); System.*out*.println("################"); result = *chordMethod*(left, right, n); System.*out*.println("x = " + result); } public static void main(String[] args) { *launchAlgsGarwick*(); }}