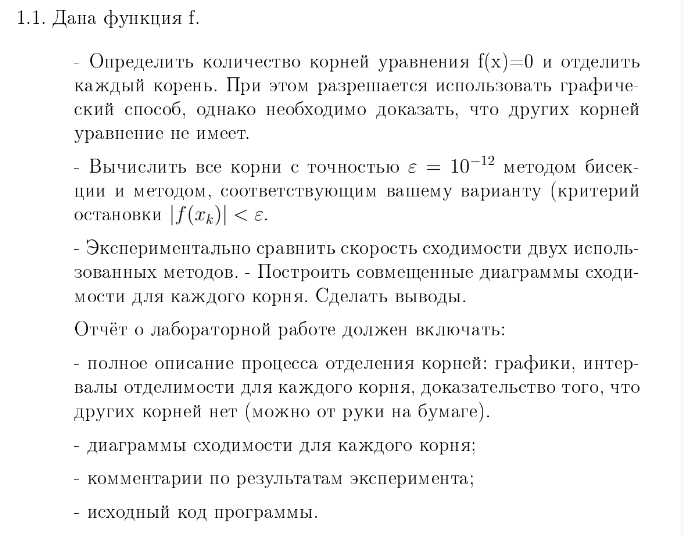
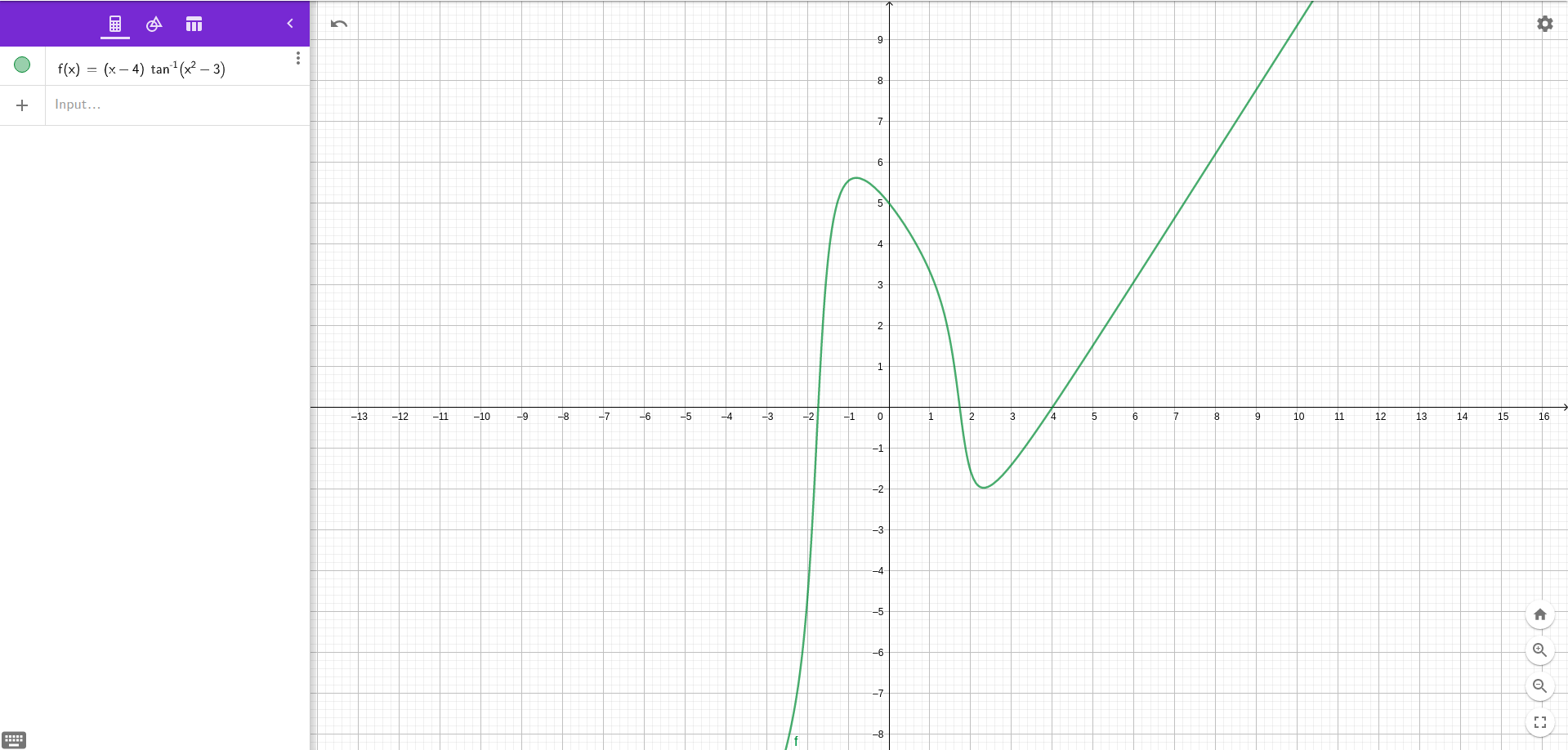
Вариант 2

(1.1.13 2.4)





Построим график на промежутке x=[-13;13]



На графике присутствуют три корня.

Докажем, что других корней не существует.

1. Рассмотрим промежуток при x < -3:

* (X – 4 ) < 0 для любого x из этого промежутка.
* арктангенс принимает значение большее нуля и при x→-∞, arctg →π/2 0 для любого x из этого промежутка.

Таким образом функция монотонна возрастает на промежутке [-∞, -3] и лежит ниже оси x;

1. Рассмотрим промежуток при x > 5:

* (X – 4 ) > 0 для любого x из этого промежутка.
* арктангенс принимает значение большее нуля и при x→+∞, arctg →π/2 0 для любого x из этого промежутка.

Таким образом функция монотонна возрастает на промежутке [-∞, -3] и лежит ниже оси x;

Из этого можно сделать вывод, что функция 1.1.13 имеет ровно 3 корня.

Уравнение имеет 3 корня:

-1.7320508075688852 на промежутке [-1.8;-1.7]

1.7320508075688852 на промежутке [1.7;1.8]

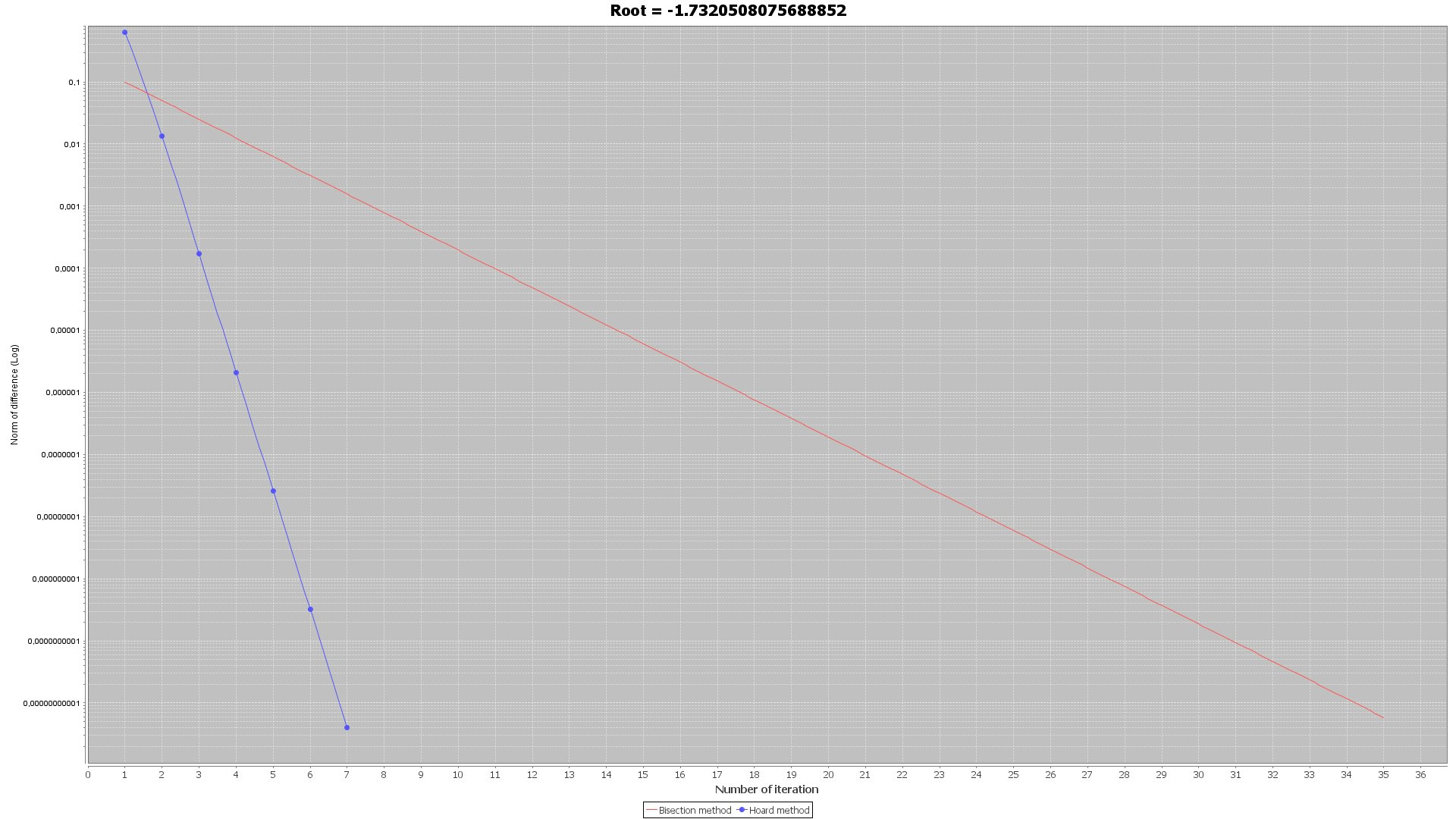
4.049999999998544 на промежутке [3.95; 4.05]

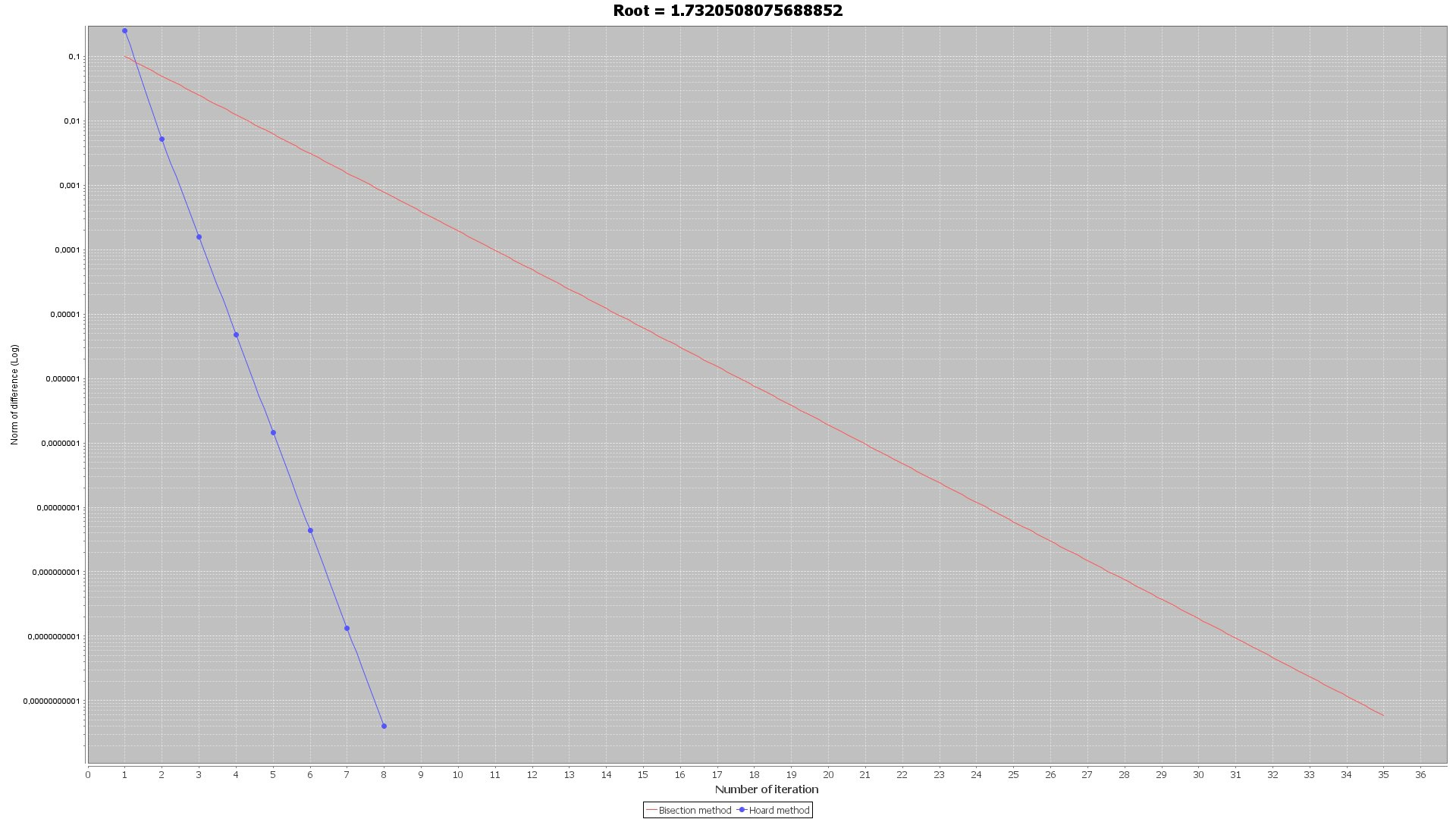
Данные корни были получены с точностью не ниже 10^-12

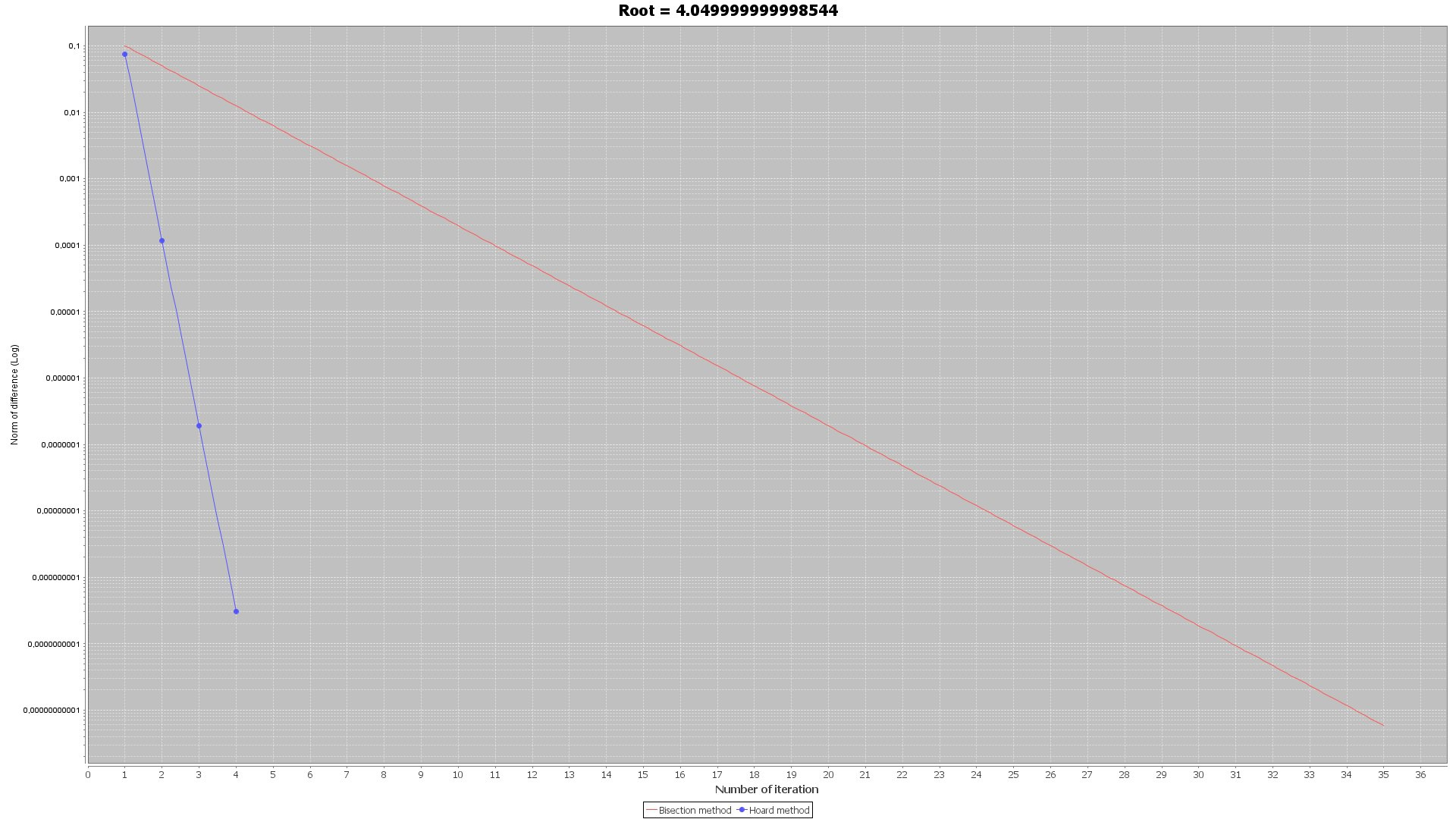
Ниже представлены диаграммы сходимости для каждого корня, где

Красный график получен методом бисекций

Синий график получен методом хорд







На основе графиков можно сделать вывод, что метод хорд сходиться быстрее

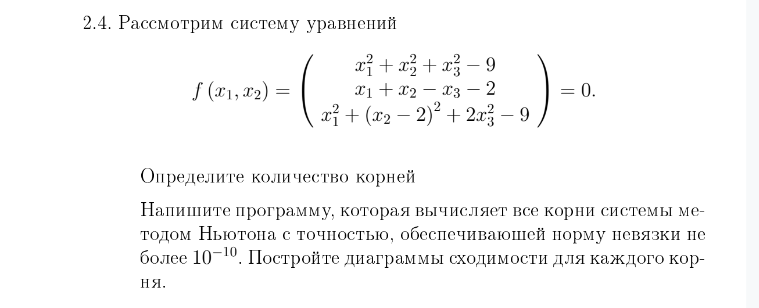
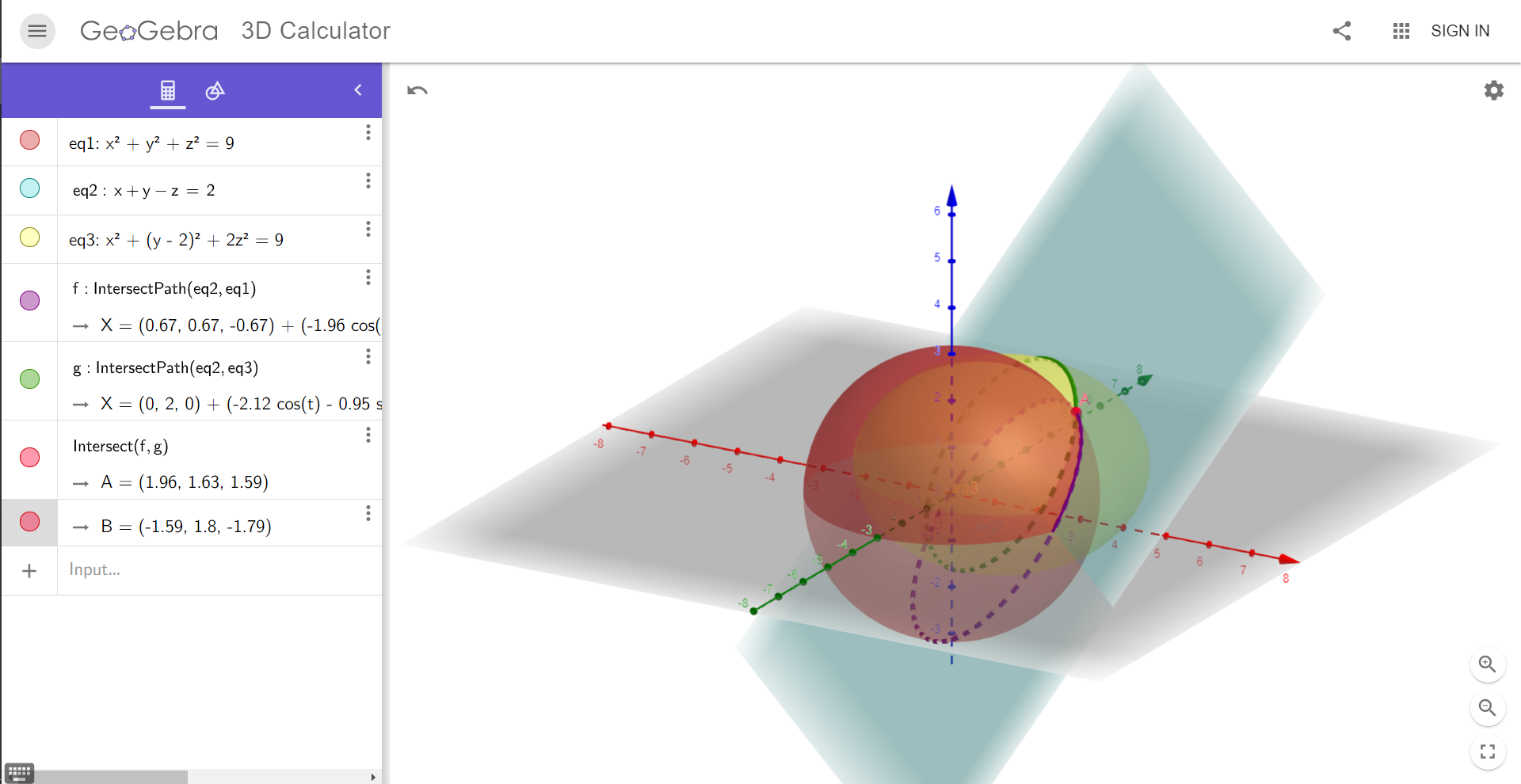


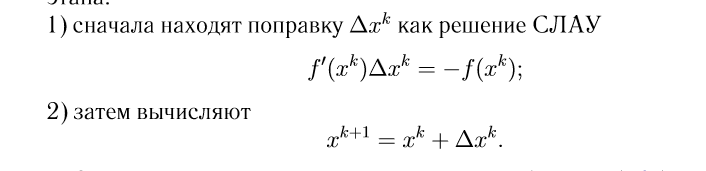
График системы уравнений имеет вид:



С помощью построений на графике нашел пересечения всех трех фигур, тем самым получил, что система имеет два решения (точки А и В на графике).

Мною с помощью метода ньютона вида 

При решении использовал алгоритм:



Где f′ =∂f /∂x— матрица Якоби

Получил корни системы с точностью 10^-10:

solve1 = [-1.594101307355829, 1.802478273690257, -1.7916230336655723]

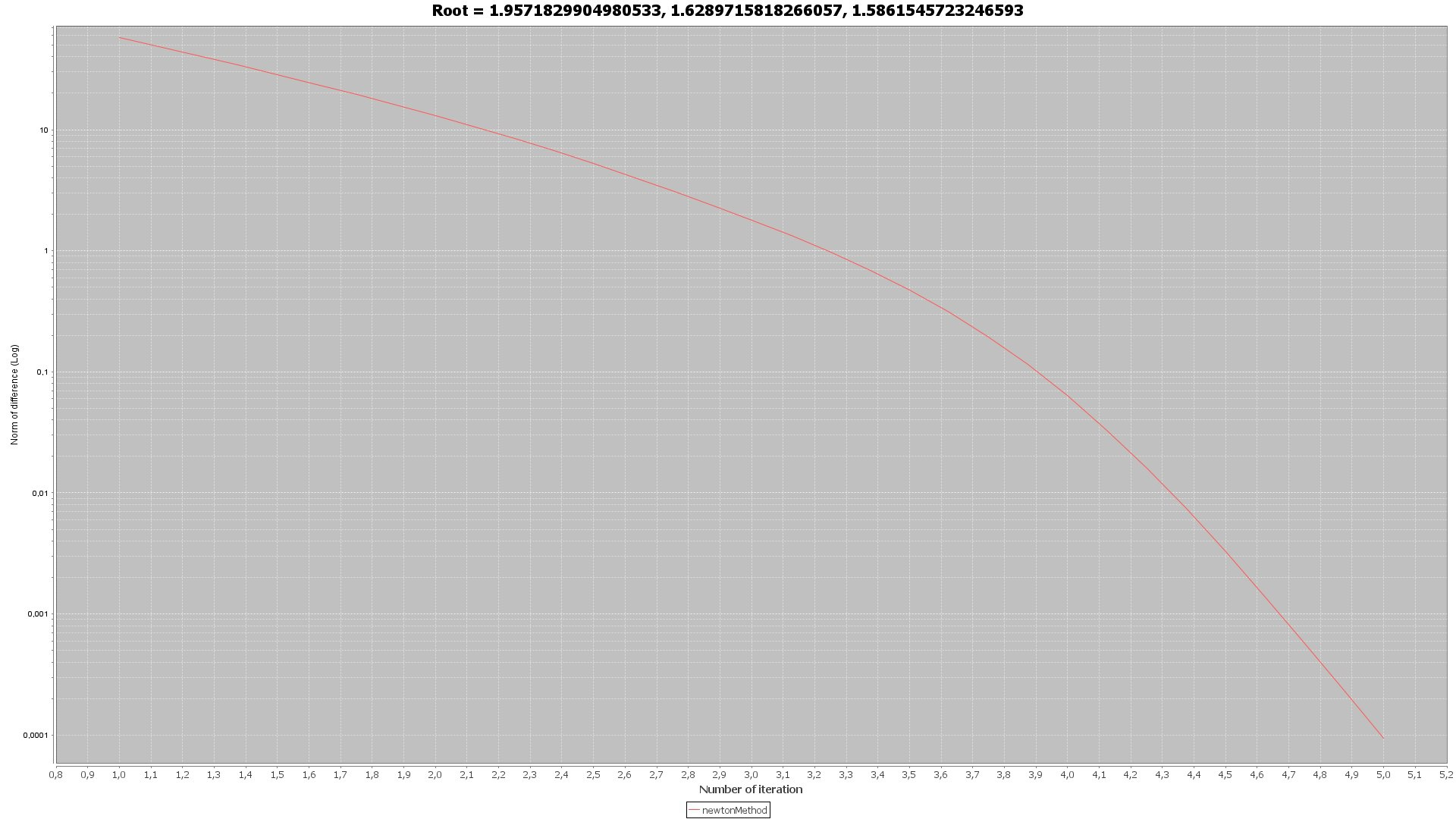
solve2 = [1.9571829904980533, 1.6289715818266057, 1.5861545723246593]

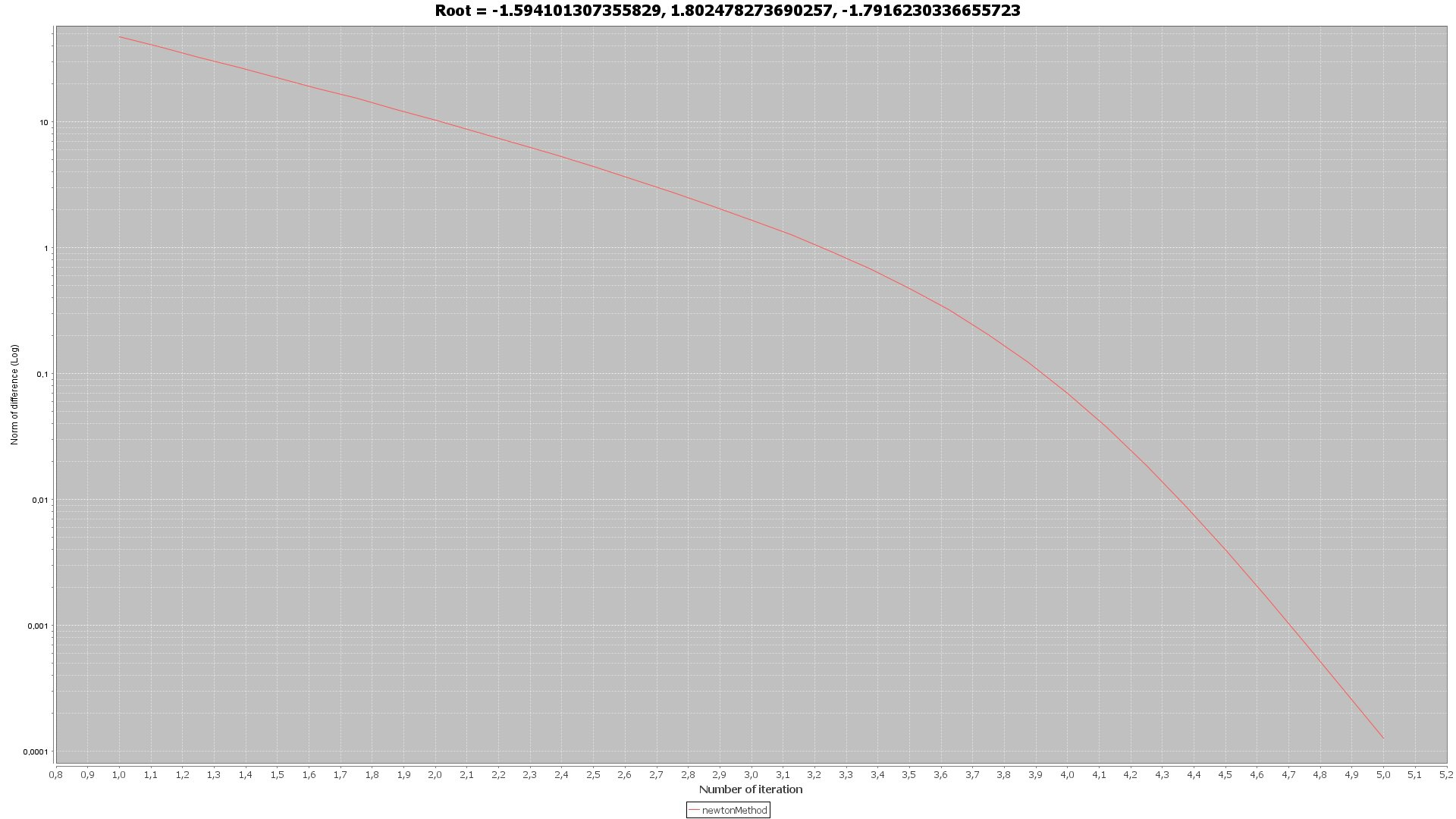
имея начальные приближения:

[-4, 4, -4] для solve1

[ 4, 2, 4] для solve2

Графики сходимости для каждого решения системы имеют вид:





Исходный код

Класс Main

package com.company;  
  
import com.company.task1.Task1;  
import com.company.task2.Task2;  
  
public class Main {  
 public static void main(String[] args) {  
 Task1 task1 = new Task1();  
 task1.solve();  
 Task2 task2 = new Task2();  
 task2.solve();  
 }  
}

Interface Function

package com.company;  
  
public interface Function {  
 double getValue(double x);  
}

Interface SolvingMethod

package com.company;  
  
public interface SolvingMethod {  
 double solve(double a, double b);  
}

Interface ThreeFunction

package com.company;  
  
public interface ThreeFunction {  
 double getValue(double x1, double x2, double x3);  
}

class StatisticCollector

package com.company;  
  
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
import java.util.Collections;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class StatisticCollector {  
 public static void collect(String fileName, List<Parameters> list){  
 var line = list.stream().map(Parameters::toString).collect(Collectors.*joining*("\n"));  
 try {  
 Files.*write*(Paths.*get*(fileName), Collections.*singleton*(line), StandardCharsets.*UTF\_8*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

class Pararmeters

package com.company;  
  
public class Parameters {  
 int iteration;  
 double discrepancy;  
  
 public Parameters(int iteration, double discrepancy) {  
 this.iteration = iteration;  
 this.discrepancy = discrepancy;  
 }  
  
 @Override  
 public String toString() {  
 return iteration + ";" + discrepancy;  
 }  
}

Для задания 1

Class Task1

package com.company.task1;  
  
import com.company.SolvingMethod;  
  
import java.util.Arrays;  
  
public class Task1 {  
  
 class Interval {  
 double a;  
 double b;  
  
 public Interval(double a, double b) {  
 this.a = a;  
 this.b = b;  
 }  
 }  
  
 final double exactness = Math.*pow*(10, -12);  
  
 double getFunction(double x) {  
 return (x - 4) \* Math.*atan*(x \* x - 3);  
 }  
  
 Interval[] intervals = {  
 new Interval(-1.8, -1.7),  
 new Interval(1.7, 1.8),  
 new Interval(3.95, 4.05)  
 };  
  
  
 public void solve() {  
 SolvingMethod bisectionMethod = new BisectionMethod(this::getFunction, exactness);  
 SolvingMethod chordMethod = new ChordMethod(this::getFunction, exactness);  
  
 double[] ans\_chord = new double[intervals.length];  
 double[] ans\_bis = new double[intervals.length];  
  
 for (int i = 0; i < intervals.length; i++) {  
 ans\_bis[i] = bisectionMethod.solve(intervals[i].a, intervals[i].b);  
 if(i==1)  
 ans\_chord[i] = chordMethod.solve(intervals[i].b, intervals[i].a);  
 else  
 ans\_chord[i] = chordMethod.solve(intervals[i].a, intervals[i].b);  
 }  
  
 System.*out*.println("bisection answers");  
 Arrays.*stream*(ans\_bis).forEach(System.*out*::println);  
 System.*out*.println("chord answers");  
 Arrays.*stream*(ans\_chord).forEach(System.*out*::println);  
  
 }  
  
}

class ChordMethod

package com.company.task1;  
  
import com.company.Function;  
import com.company.Parameters;  
import com.company.SolvingMethod;  
import com.company.StatisticCollector;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class ChordMethod implements SolvingMethod {  
 Function func;  
 double exactness;  
  
 public ChordMethod(Function func, double exactness) {  
 this.func = func;  
 this.exactness = exactness;  
 }  
  
 @Override  
 public double solve(double a, double b) {  
 var name = "chord\_a="+a+"\_b="+b;  
 List<Parameters> statist = new ArrayList<>();  
 double x = b;  
 double t;  
 for (int i = 0; (t = Math.*abs*(func.getValue(x))) > exactness; i++) {  
 statist.add(new Parameters(i, t));  
 x = x - func.getValue(x) \* (x - a)/ (func.getValue(x)- func.getValue(a));  
 }  
  
 StatisticCollector.*collect*(name, statist);  
 return x;  
 }  
}

class BisectionMethod

package com.company.task1;  
  
import com.company.Function;  
import com.company.Parameters;  
import com.company.SolvingMethod;  
import com.company.StatisticCollector;  
  
  
import java.lang.reflect.Parameter;  
import java.util.ArrayList;  
import java.util.List;  
  
public class BisectionMethod implements SolvingMethod {  
 Function func;  
 double exactness;  
  
 public BisectionMethod(Function func, double exactness) {  
 this.func = func;  
 this.exactness = exactness;  
 }  
  
 @Override  
 public double solve(double a, double b) {  
 var name = "bisection\_a="+a+"\_b="+b;  
 List<Parameters> statist = new ArrayList<>();  
 double x = a;  
 double t=0;  
 for (int i=0;(t = (b - a)) > exactness \* 2; i++) {  
 statist.add(new Parameters(i,t));  
 x = (a + b) / 2;  
 if ((func.getValue(a) \* func.getValue(x)) < 0) {  
 b = x;  
 } else {  
 a = x;  
 }  
 }  
 StatisticCollector.*collect*(name,statist);  
 return x;  
 }  
  
}

Для задания 2

Class Task2

package com.company.task2;  
  
import com.company.Function;  
import com.company.ThreeFunction;  
  
import java.util.Arrays;  
import java.util.List;  
  
public class Task2 {  
  
  
 ThreeFunction[] list = {  
 (x1, x2, x3) -> x1 \* x1 + x2 \* x2 + x3 \* x3 - 9,  
 (x1, x2, x3) -> x1 + x2 - x3 - 2,  
 (x1, x2, x3) -> x1 \* x1 + (x2 - 2) \* (x2 - 2) + 2 \* x3 \* x3 - 9  
 };  
  
 ThreeFunction[][] yakobi = {  
 {(x1, x2, x3) -> 2\* x1 ,  
 (x1, x2, x3) -> 2\* x2,  
 (x1, x2, x3) -> 2 \* x3 },  
 {(x1, x2, x3) -> 1 ,  
 (x1, x2, x3) -> 1,  
 (x1, x2, x3) -> - 1 },  
 {(x1, x2, x3) -> 2 \* x1 ,  
 (x1, x2, x3) -> 2 \* (x2) -4 ,  
 (x1, x2, x3) -> 4 \* x3 }  
 };  
  
  
 private final double exactness = Math.*pow*(10, -10);  
  
  
 public void solve(){  
 NewtonMethod newtonMethod = new NewtonMethod(list,yakobi,exactness, new GauseSolver());  
  
 double[] solve1 = newtonMethod.solve((new double[]{-4, 4, -4}));  
 System.*out*.println("solve1 = " + Arrays.*toString*(solve1));  
 double[] solve2 = newtonMethod.solve(new double[]{4, 2, 4});  
 System.*out*.println("solve2 = " + Arrays.*toString*(solve2));  
 }  
}

class NewtonMethod

package com.company.task2;  
  
import com.company.Parameters;  
import com.company.StatisticCollector;  
import com.company.ThreeFunction;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class NewtonMethod {  
  
 ThreeFunction[] func;  
 ThreeFunction[][] yakobi;  
 final double exactness;  
 SLAESolver solver;  
  
 public NewtonMethod(ThreeFunction[] func, ThreeFunction[][] yakobi, double exactness, SLAESolver solver) {  
 this.func = func;  
 this.yakobi = yakobi;  
 this.exactness = exactness;  
 this.solver = solver;  
 }  
  
 public double[] solve(double[] x0) {  
 var name = "newton\_x0="+ Arrays.*toString*(x0);  
 List<Parameters> statist = new ArrayList<>();  
 var ans =x0;  
 var prevDisp=1.;  
 var disp = 1.;  
 for(int i=0;(disp = getDiscrepancy(countVector(func, ans)))>exactness; i++){  
// if(disp>prevDisp){  
// throw new RuntimeException("bad discrepancy disc " + disp+" discr prec: "+ prevDisp+" sub "+ (disp-prevDisp));  
// }  
 statist.add(new Parameters(i, disp));  
 prevDisp = disp;  
  
 var a = countMatrix(yakobi, ans);  
 var b = changeSign(countVector(func, ans));  
 var diff=solver.solve(a,b);  
 ans = arraysSum(ans, diff);  
 }  
 StatisticCollector.*collect*(name, statist);  
 return ans;  
 }  
  
 private double[] changeSign(double[] ans){  
 return Arrays.*stream*(ans).map(e->-e).toArray();  
 }  
 private double[][] countMatrix(ThreeFunction[][] function, double[] variable){  
 var ans = new double[function.length][function[0].length];  
 for (int i = 0; i < function.length; i++) {  
 for (int j = 0; j < function.length; j++) {  
 ans[i][j] = function[i][j].getValue(variable[0], variable[1], variable[2]);  
 }  
 }  
 return ans;  
 }  
  
 private double[] countVector(ThreeFunction[] function, double[] variable){  
 var ans = new double[function.length];  
 for (int i = 0; i < function.length; i++) {  
 ans[i] = function[i].getValue(variable[0], variable[1], variable[2]);  
 }  
 return ans;  
 }  
  
 private double[] arraysSum(double[] a, double[] b){  
 var ans = new double[a.length];  
  
 for (int i = 0; i < a.length; i++) {  
 ans[i] = a[i] + b[i];  
 }  
 return ans;  
 }  
  
 private double getDiscrepancy(double[] ans) {  
 return Math.*sqrt*(Arrays.*stream*(ans).map((e)->e\*e).sum());  
 }  
  
}

Class GouseSolver

package com.company.task2;  
  
import java.util.Arrays;  
  
public class GauseSolver implements SLAESolver {  
  
  
 @Override  
 public double[] solve(double[][] a, double[] b) {  
 a = doForward(a,b);  
 return doBackward(a, b);  
 }  
  
 private double[][] doForward(double[][] a, double[] b) {  
  
 for (int k = 0; k < a[0].length - 1; k++) {  
 for (int i = k + 1; i < a.length; i++) {  
 var l = a[i][k] / a[k][k];  
 for (int j = 0; j < a.length; j++) {  
 a[i][j] -= l \* a[k][j];  
 }  
 b[i] -= l \* b[k];  
 }  
 }  
 return a;  
 }  
  
 private double[] doBackward(double[][] a, double[] b){  
 var ans = new double[b.length];  
  
 for (int i = b.length-1; i >= 0; i--) {  
 for (int j = i+1; j < b.length; j++) {  
 b[i] -= a[i][j] \* ans[j];  
 }  
 ans[i] = 1./a[i][i] \* b[i];  
 }  
  
 return ans;  
 }  
}

interface SLAESolver

package com.company.task2;  
  
public interface SLAESolver {  
 double[] solve(double[][] a, double[]b);  
}