БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики Кафедра вычислительной математики

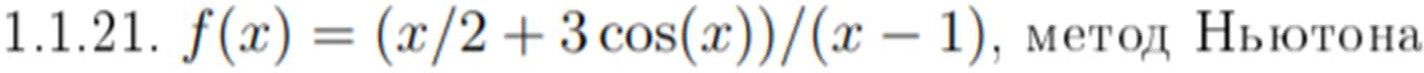
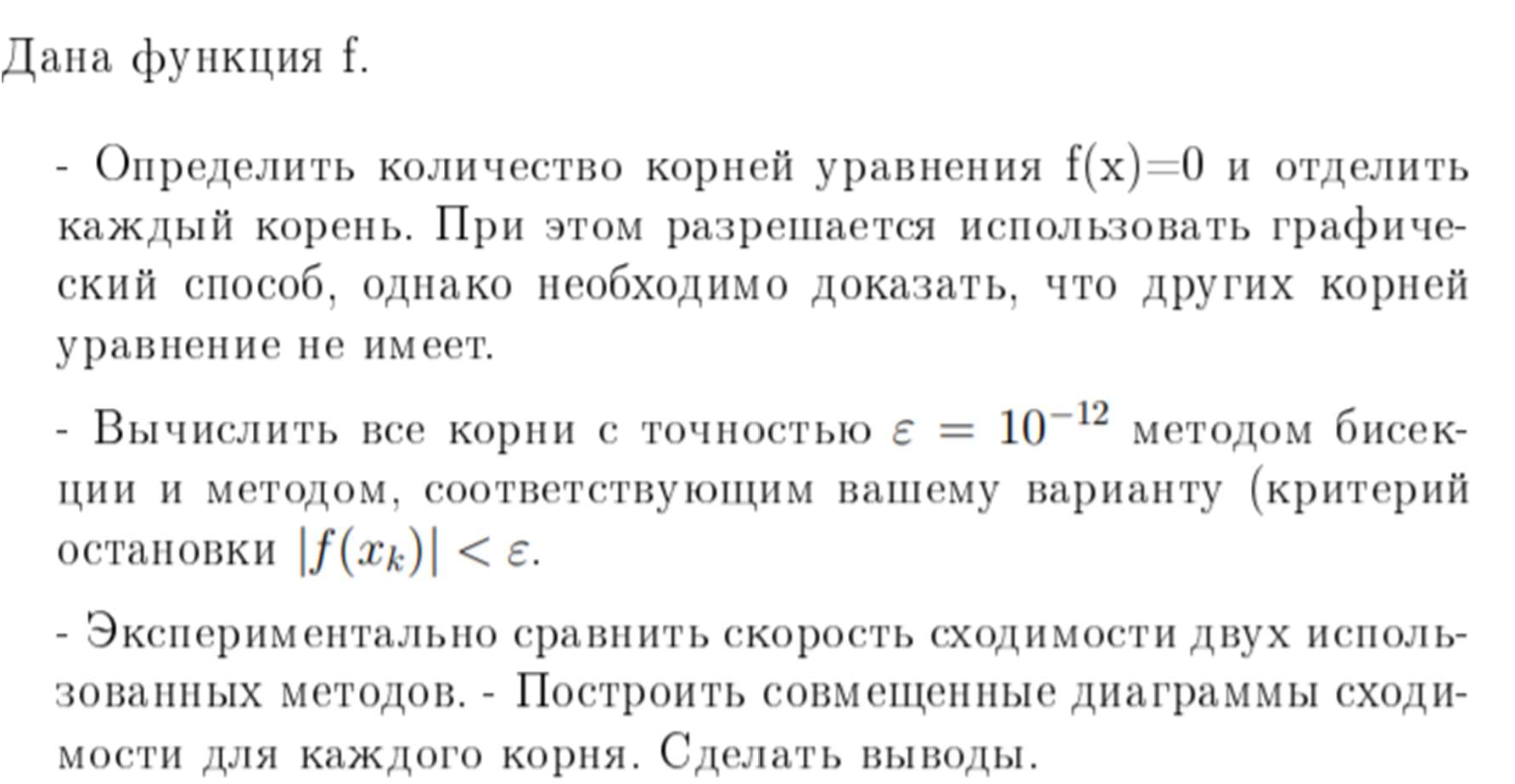
Никончик Даниил Викторович ОТЧЕТ ПО МЕТОДАМ ВЫЧИСЛЕНИЙ

студента 2 курса 12 группы Лабораторная работа №3

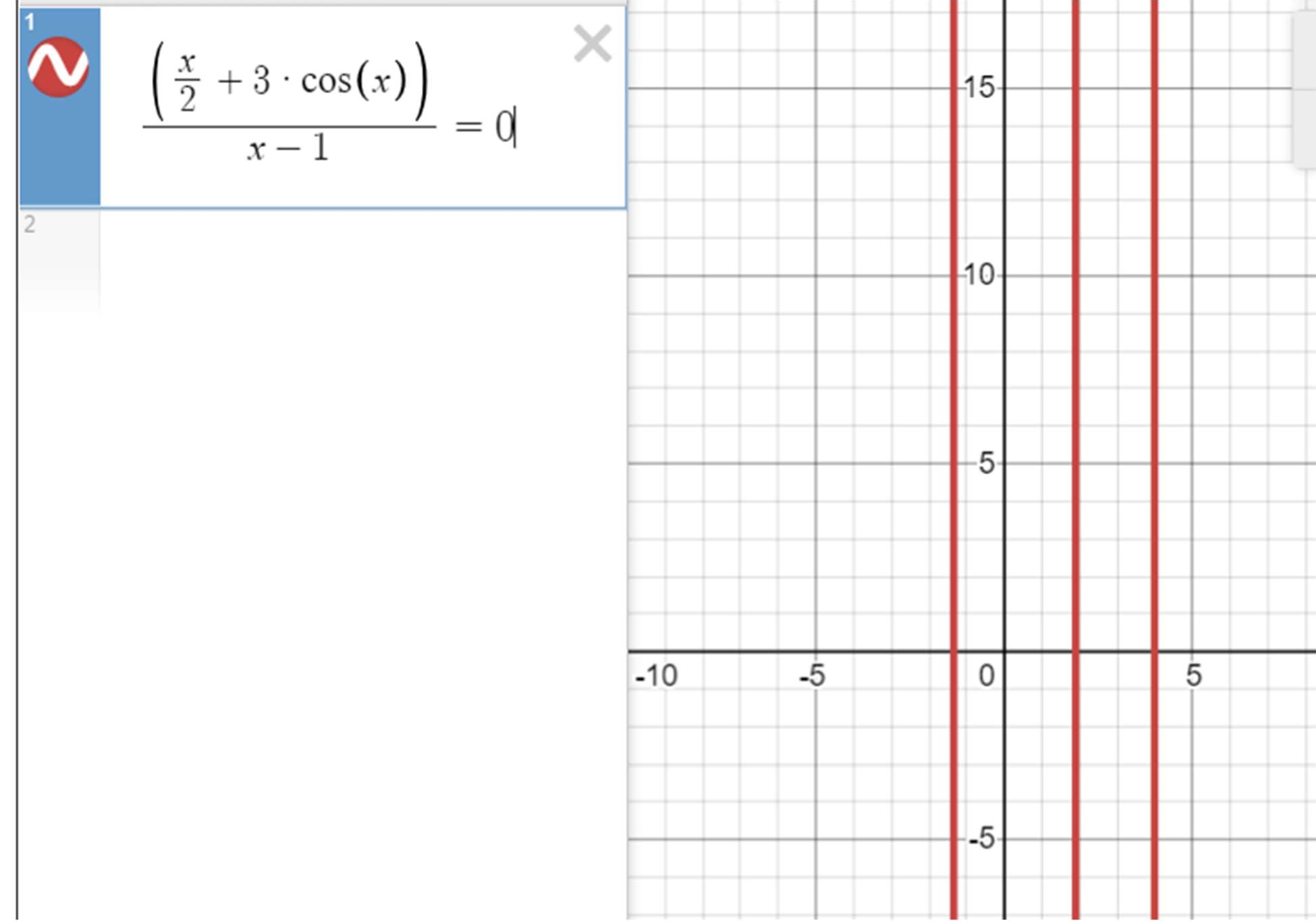
Преподаватель Бондарь И.В.

Минск 2020

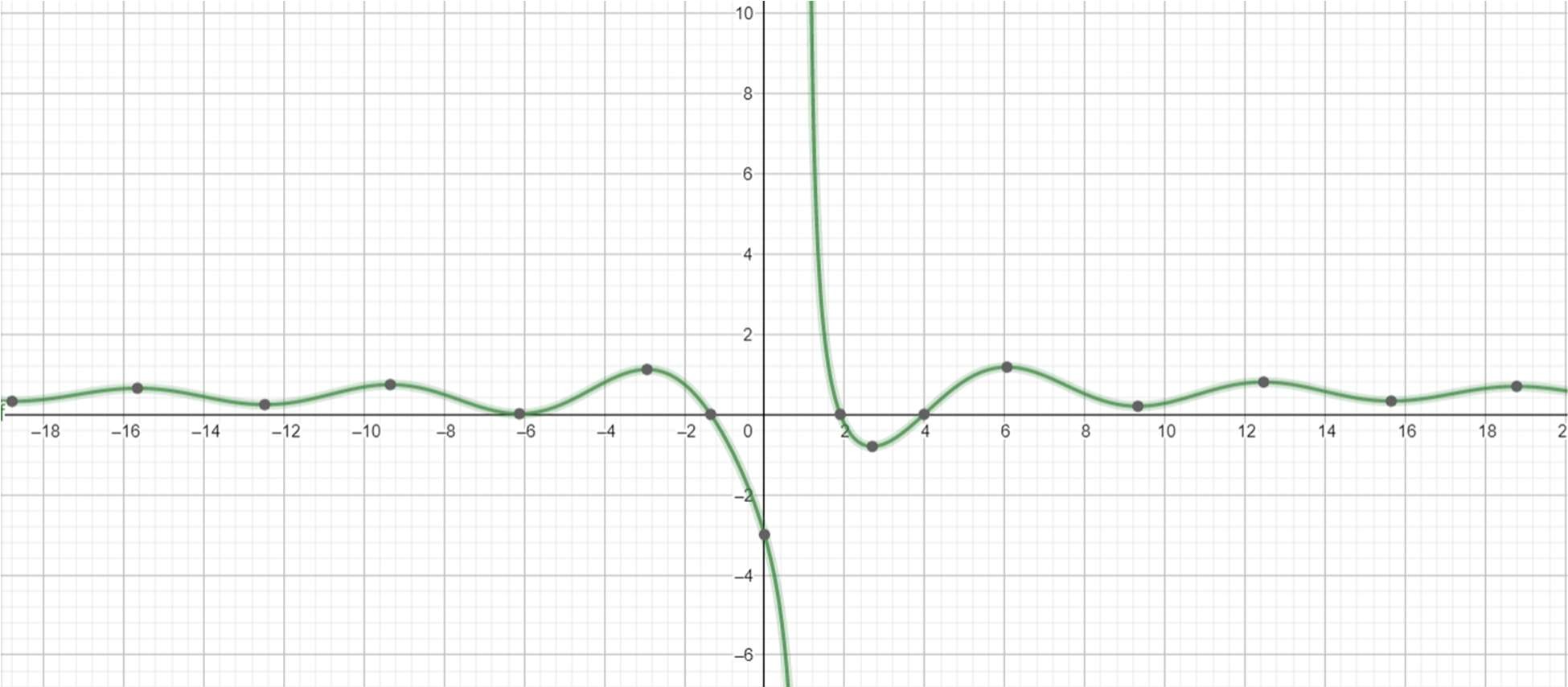
# Задание 1. Методы Якоби и Гаусса-Зейделя. Вариант 1.1.21



Для начала найдем решение уравнения графическим методом:



Или, если рассмотреть исключительно график кривой:

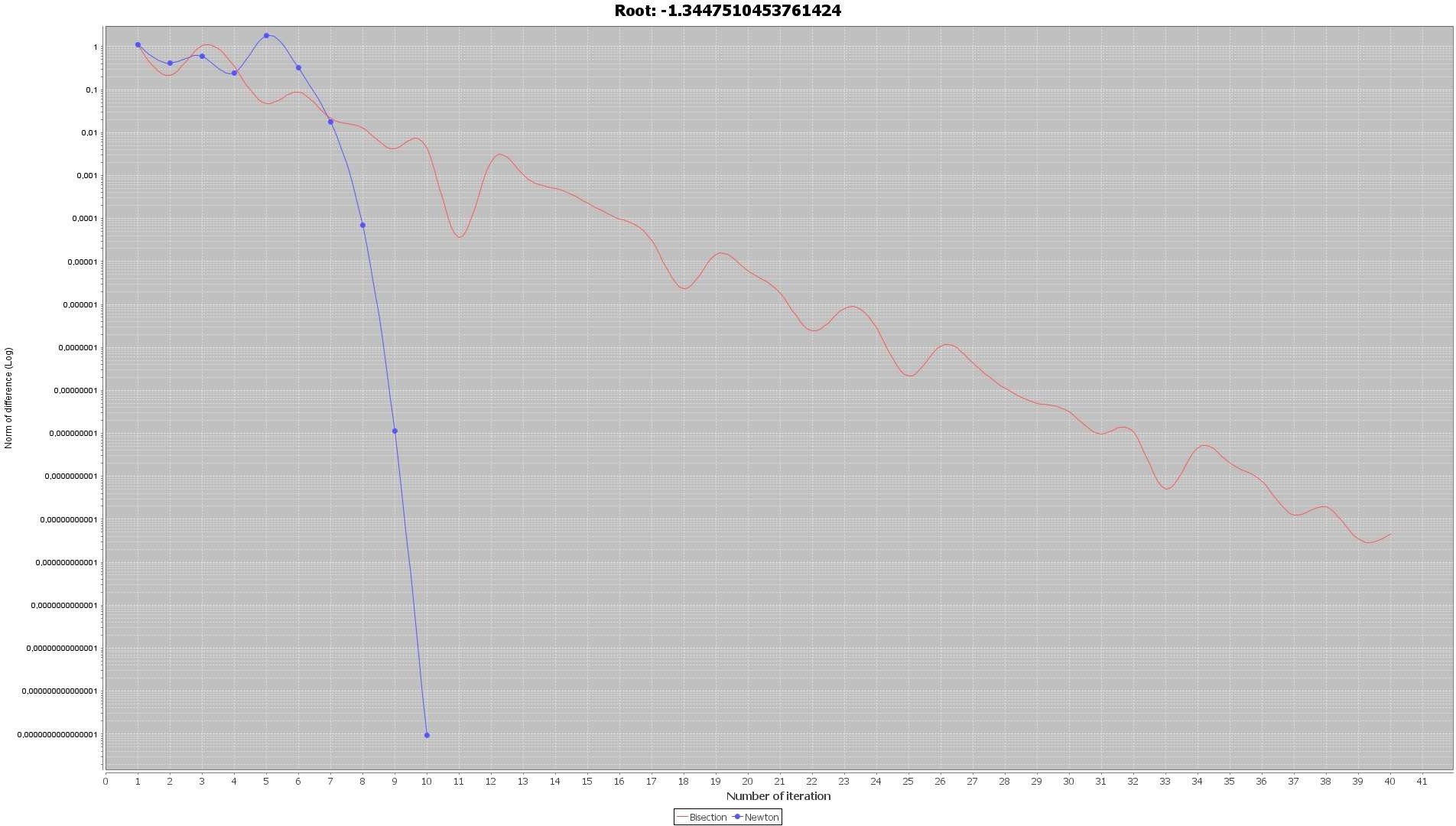


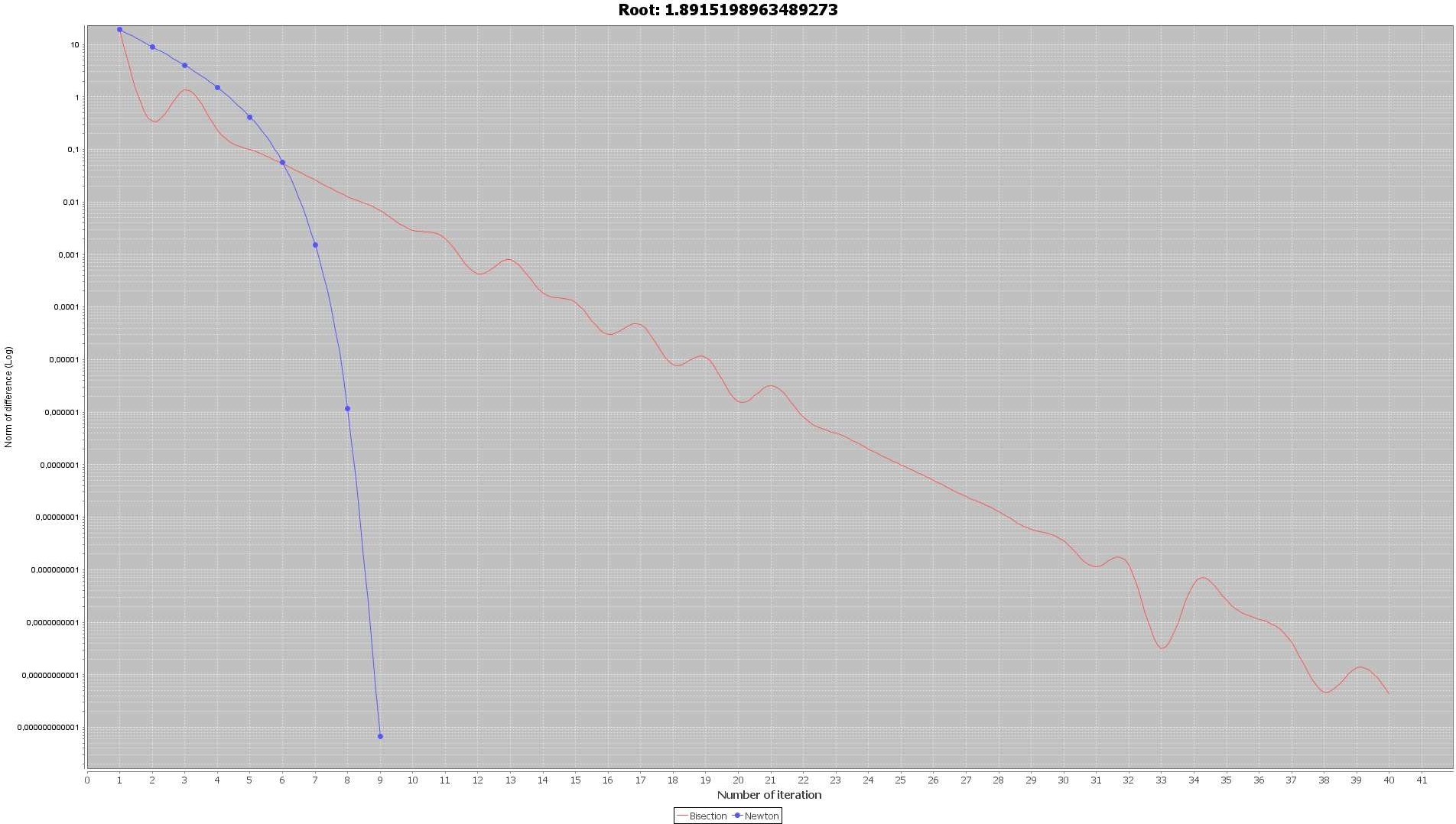
При этом, при приближении графика в районе x = -6, становится понятно, что данное приближение к оси ОХ не является корнем. Оценим поведение функции на промежутках x<-6, x>10.Преобразуем вид функции и оценим её значения:

Так как , то для

Таким образом, корней нет на отрезке х<-6. Для x>10 доказательство аналогично.

Диаграммы сходимости для каждого корня:







По графику видно, что метод ньютона сходится явно быстрее метода бисекций.

## Код подсчета значения и производной функции:

**public** **class** **MyFunction** **implements** Function{

**@Override**

**public** **double** **getFunctionResult**(**double** x) **throws** Exception{ **if**(x == **1**)

**throw** **new** **Exception**("incorrect x"); **return** ((x/**2**)+**3**\*Math.cos(x))/(x-**1**);

}

**@Override**

**public** **double** **getDerivativeResult**(**double** x) **throws** Exception {

**return** ((**0.5**-**3**\*Math.sin(x))\*(x-**1**)-(x/**2**+**3**\*Math.cos(x)))/Math.pow((x-**1**),**2**);

}

}

## Код итерации метода бисекций:

**public** **class** **BisectionCalculator** **extends** Calculator { **public** BisectionCalculator(Function function){

**super**(function);

}

**@Override**

**protected** **double** **getNextApproximation**(**double** x) **throws** Exception {

**if**(function.getFunctionResult(a)\*function.getFunctionResult(x)<**0**) b = x;

**else**

a = x;

**return** (a+b)/**2**;

}

}

## Код итерации метода Ньютона:

**public** **class** **NewtonCalculator** **extends** Calculator { **public** NewtonCalculator(Function function) {

**super**(function);

}

**@Override**

**protected** **double** **getNextApproximation**(**double** x) **throws** Exception { **return** x-function.getFunctionResult(x)/function.getDerivativeResult(x);

}

}

## Общий код вычисления по методам:

**public** **abstract** **class** **Calculator** {

**protected** **final** **double** PRECISION = **1**E-**12**; **protected** ArrayList<Double> statistics; **protected** Function function;

**protected** **double** a,b;

**public** **Calculator**(Function function){ **this**.function = function;

}

**public** **double** **calculate**(Interval interval) **throws** Exception { a = interval.a();

b = interval.b();

statistics = **new** ArrayList<>(); **double** x = interval.a();

**double** functionFromA = function.getFunctionResult(a); **double** functionFromB = function.getFunctionResult(b); **if**(functionFromA\*functionFromB>=**0**)

**throw** **new** **Exception**("incorrect interval"); **int** iteration = **1**;

**double** functionFromX = function.getFunctionResult(x); **while**(Math.abs(functionFromX)>PRECISION && iteration<**100**){

statistics.add(Math.abs(functionFromX)); iteration++;

x = getNextApproximation(x);

functionFromX = function.getFunctionResult(x);

}

statistics.add(Math.abs(functionFromX)); **return** x;

}

**protected** **abstract** **double** **getNextApproximation**(**double** x) **throws** Exception;

**public** Double[] **getStatistic**(){

**return** statistics.toArray(Double[]::**new**);

}

}

## Интервалы вычислений:

**static** Interval[] bisectionIntervals = **new** Interval[]{ **new** Interval(-**3**,**0**),

**new** **Interval**(**1.1**,**3**), **new** Interval(**3**,**5**),

};

## Класс, производящий работу:

**class** **CalculationsProcessor**{ **double**[]results;

Double[][]statistics; Calculator calculator;

**public** **CalculationsProcessor**(Calculator calculator){ **this**.calculator = calculator;

}

**public** **void** **calculate**(Interval[]intervals) **throws** Exception { results = **new** **double**[intervals.length];

statistics = **new** Double[results.length][**0**]; **for**(**int** i = **0**;i< results.length;i++){

results[i] = calculator.calculate(intervals[i]); var localStatistic = calculator.getStatistic();

statistics[i] = Arrays.copyOf(localStatistic,localStatistic.length);

}

}

**public** **double**[] **getPrecisionsWithIndex**(**int** i){ **double**[]result = **new** **double**[statistics[i].length]; **for**(**int** j = **0**;j<statistics[i].length;j++)

result[j] = statistics[i][j]; **return** result;

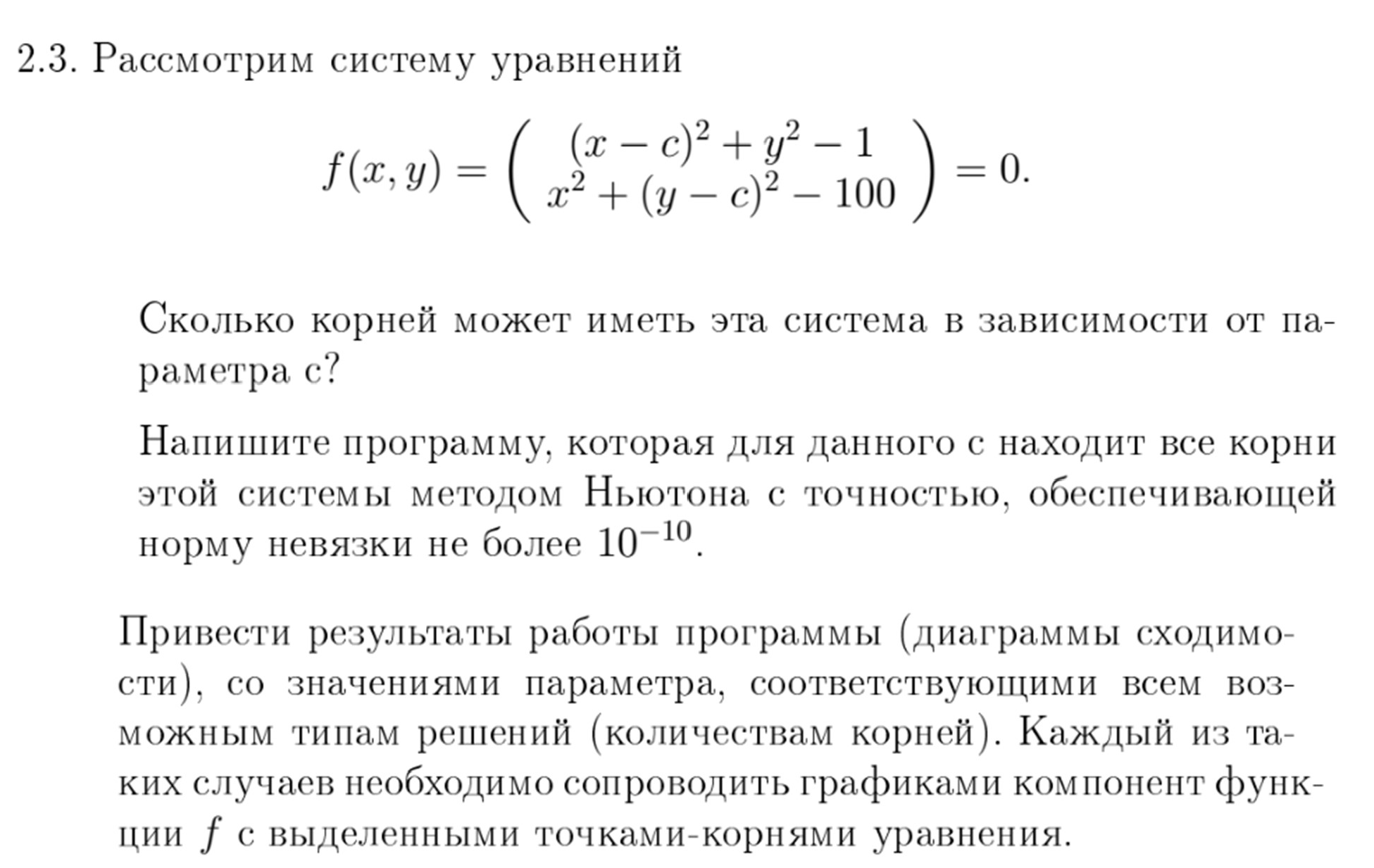
}

**public** **int** **getIterationsNumberWithIndex**(**int** i){ **return** statistics[i].length;

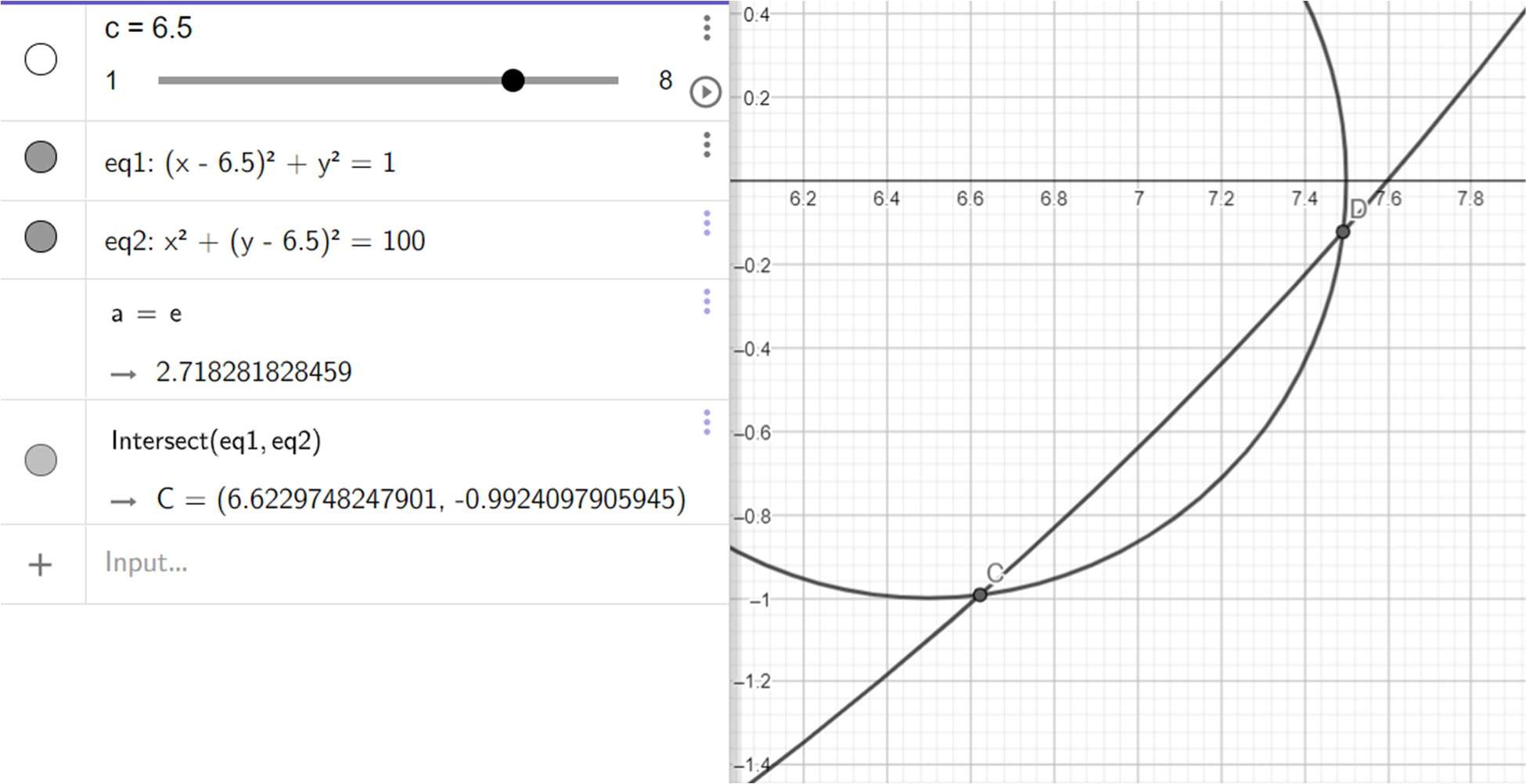
}

}

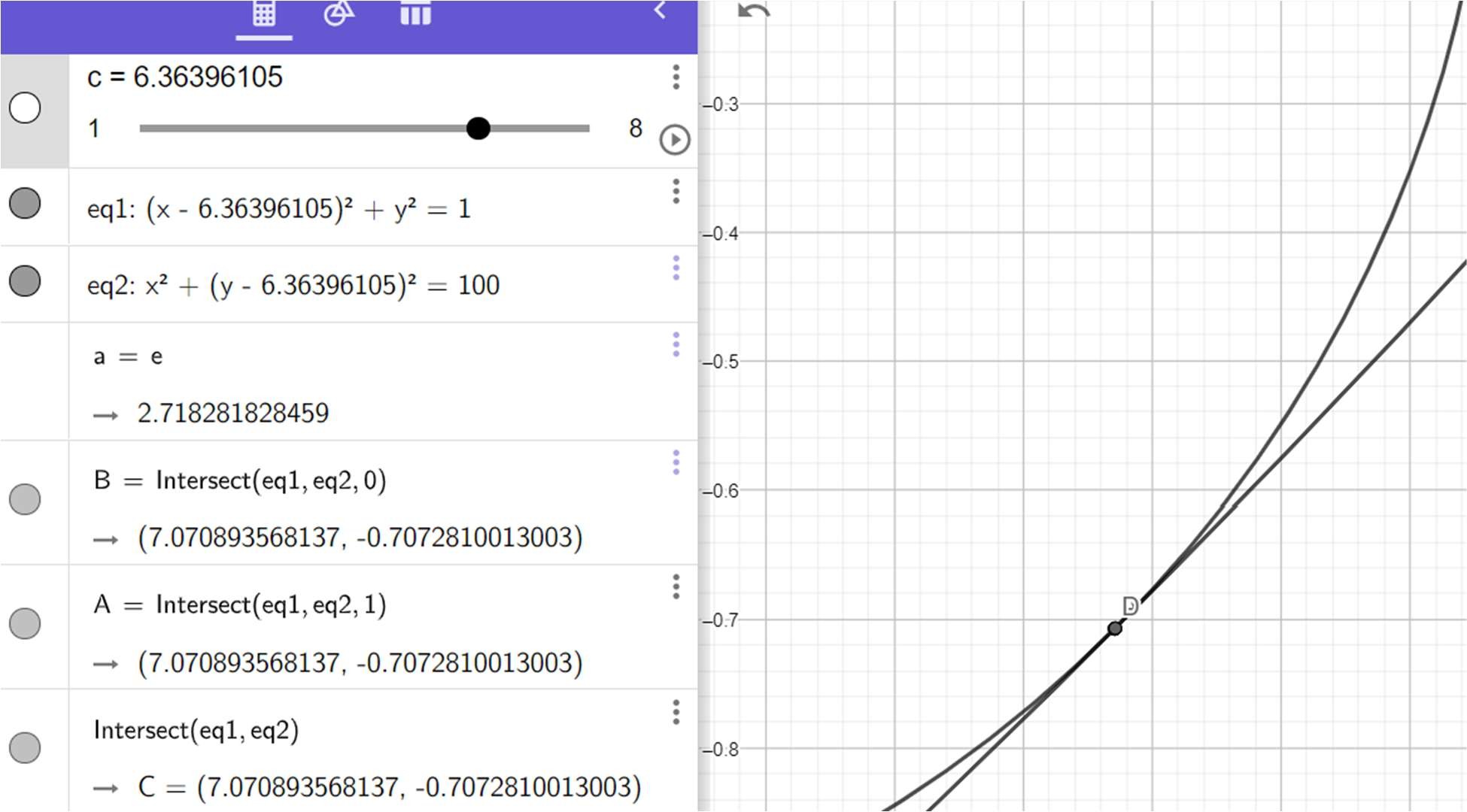
# Задание 2. Вариант 2.3



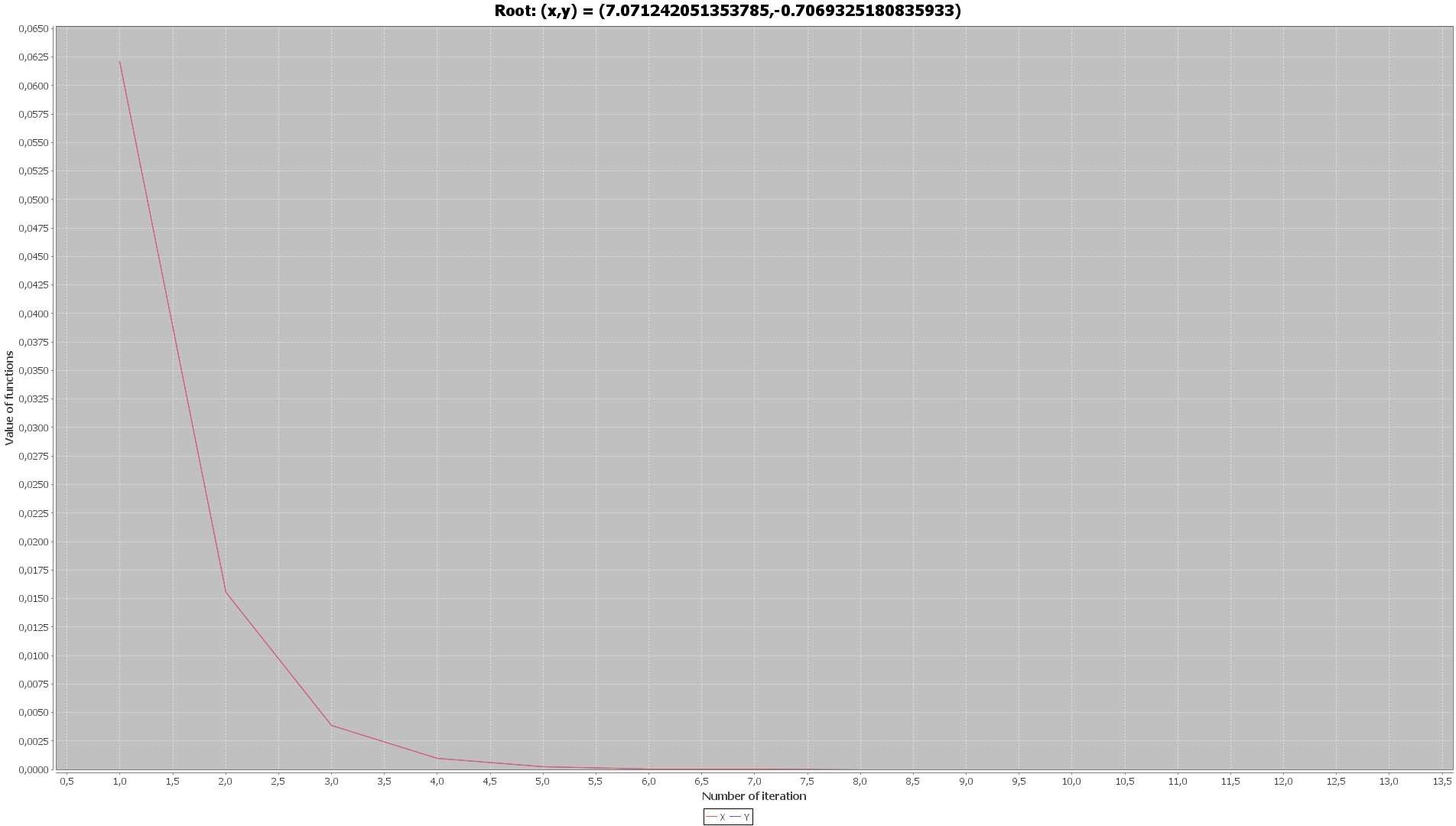
Так как данные два уравнения задают две окружности с центрами в точках (с,0) и (0,с) то в зависимости от расположение окружностей могут иметь место 1, 2 или ни одной точек пересечения, которые и являются решениями уравнений. Найдем графическим методом одни из значений параметра с, при которых это достигается:

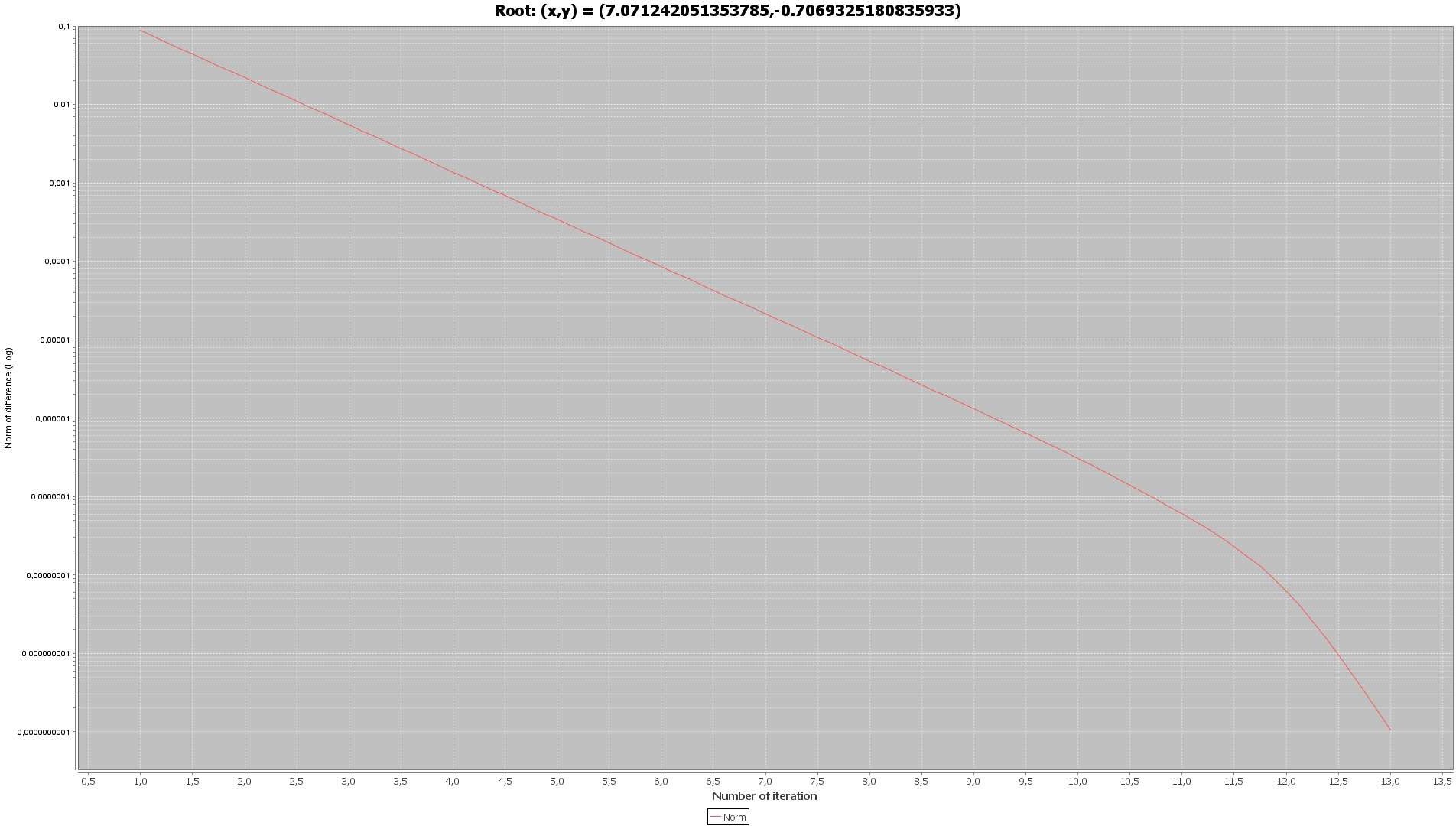


И, соответственно, для одного корня (точки касания):

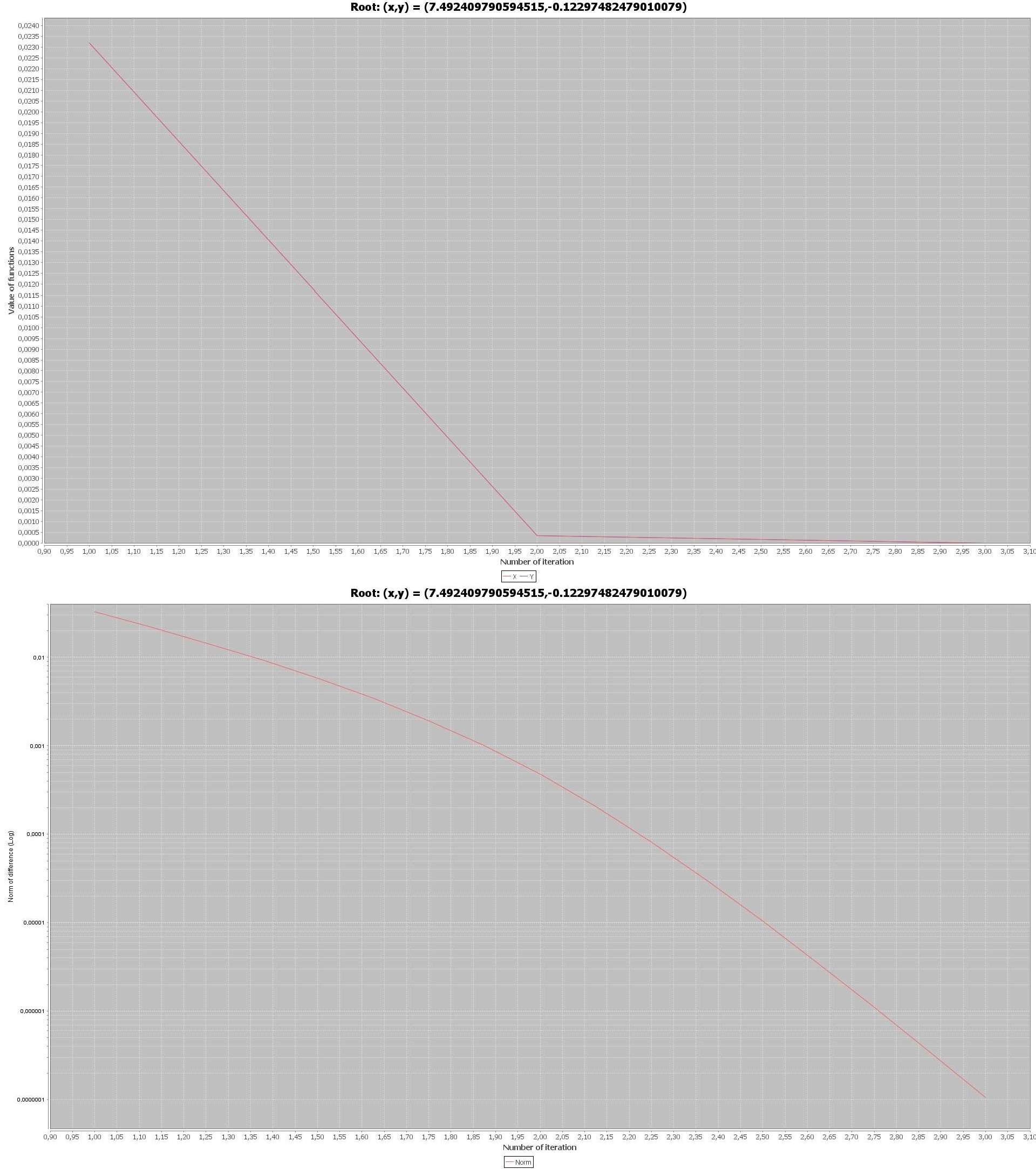


**Диаграмма сходимости и График компонент функции для первого корня:**

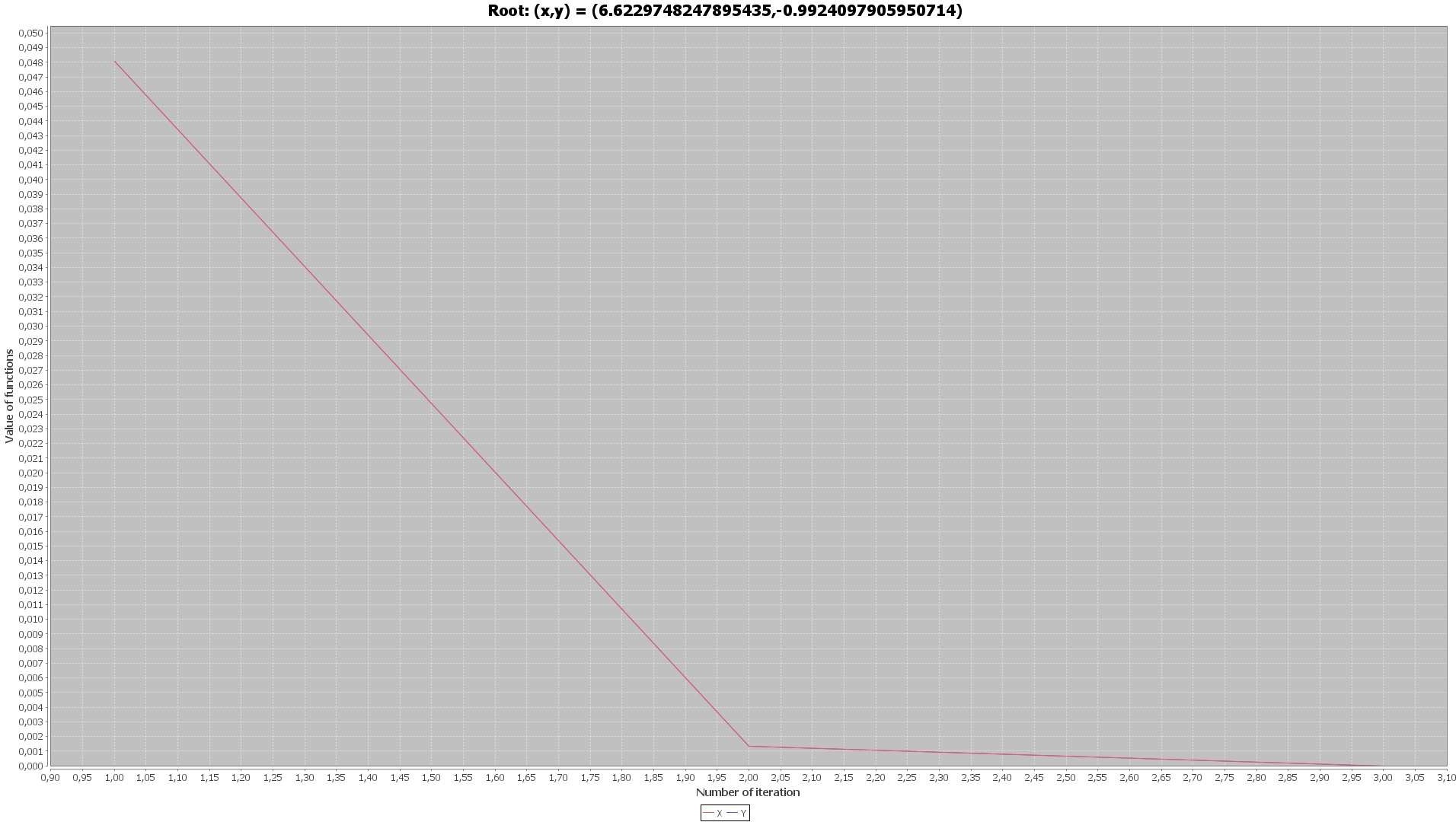


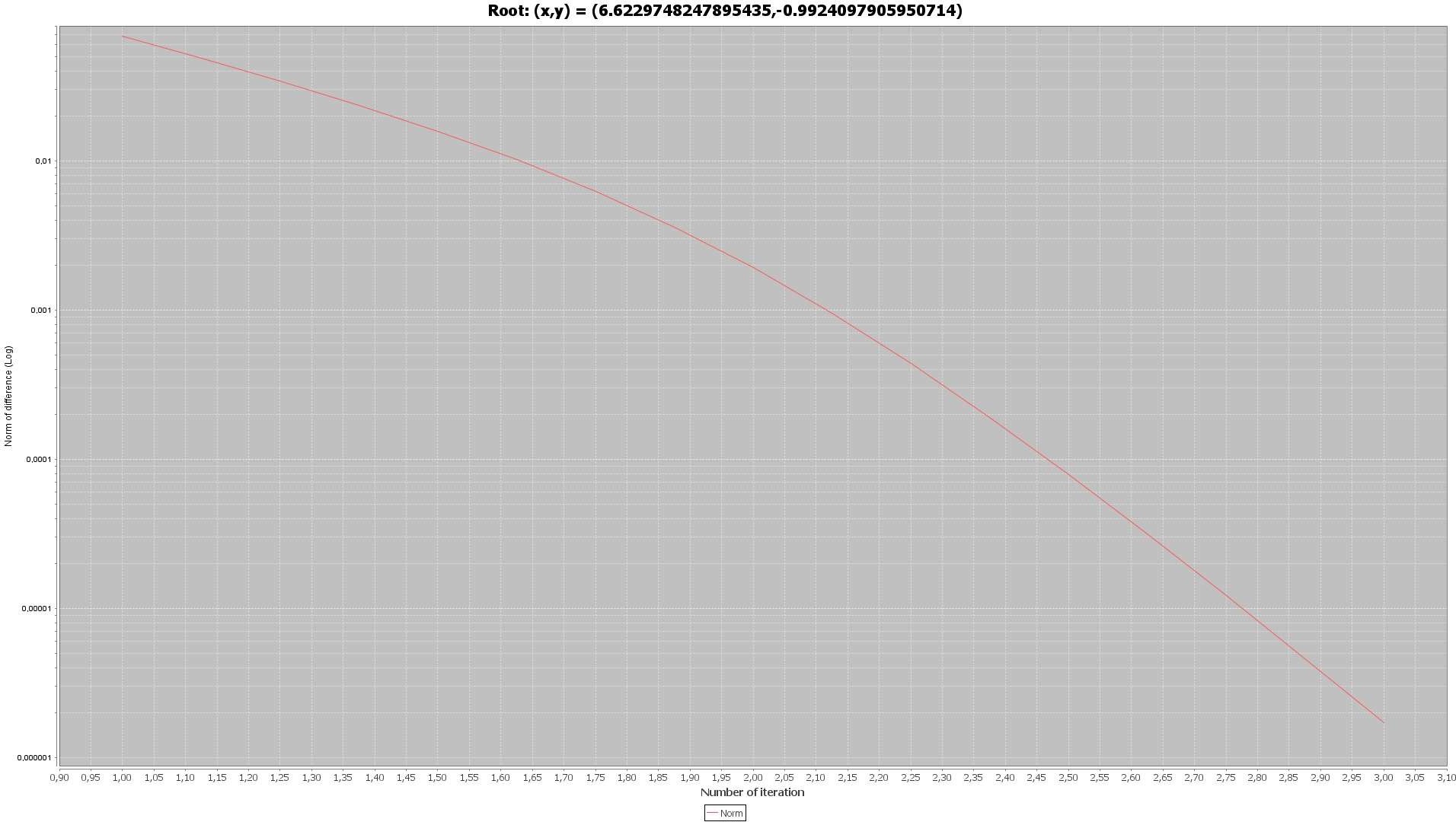


**Диаграмма сходимости и График компонент функции для первого корня из случай с двумя точками пересения:**



**Диаграмма сходимости и График компонент функции для второго корня для случая двумя точками пересечения:**





**Базовый код подсчета:**

**public** **class** **NewtonSystemCalculator** {

**protected** **final** **double** PRECISION = **1**E-**10**; **protected** ArrayList<Double> statistics; ArrayList<Double>functionXValuesStatistic; ArrayList<Double>functionYValuesStatistic; **protected** MultiFunction function;

**public** **NewtonSystemCalculator**(MultiFunction function) {

**this**.function = function;

}

**public** **double**[] **calculate**(DoubleInterval interval) { statistics = **new** ArrayList<>(); functionXValuesStatistic = **new** ArrayList<>(); functionYValuesStatistic = **new** ArrayList<>();

**double**[] point = **new** **double**[]{interval.x1(), interval.y1()}; **double** max;

**int** iteration = **1**; **do** {

**double**[] functionValues = function.getFunctionResult(point); **for** (**int** i = **0**; i < functionValues.length; i++)

functionValues[i] = -functionValues[i];

**double**[][] derivativeValues = function.getDerivativeFunctionResult(point); **double**[] differences = getSolvesOfSystem(derivativeValues, functionValues); **for** (**int** i = **0**; i < differences.length; i++)

point[i] += differences[i];

**double**[] residual = function.getFunctionResult(point); functionXValuesStatistic.add(residual[**0**]); functionYValuesStatistic.add(residual[**1**]);

max = Math.sqrt(DoubleStream.of(residual).map(a->Math.pow(a,**2**)).sum()); **if**(max!=**0**)

statistics.add(max); iteration++;

} **while** (max > PRECISION && iteration<**102**); **return** point;

}

**public** ArrayList<Double> **getStatistics**() { **return** statistics;

}

**public** **double**[] **getFunctionXValuesStatistic**() {

**return** functionXValuesStatistic.stream().mapToDouble(**Double:**:doubleValue).toArray();

}

**public** **double**[] **getFunctionYValuesStatistic**() {

**return** functionYValuesStatistic.stream().mapToDouble(**Double:**:doubleValue).toArray();

}

**private** **double**[] **getSolvesOfSystem**(**double**[][] A, **double**[] b) { **double**[] result = **new** **double**[b.length];

**for** (**int** k = **0**; k < A.length; k++) {

**for** (**int** i = k + **1**; i < A[**0**].length; i++) { **double** l = A[i][k] / A[k][k];

**for** (**int** j = **0**; j < A[**0**].length; j++) { A[i][j] -= l \* A[k][j];

}

b[i] -= l \* b[k];

}

}

**for** (**int** i = result.length - **1**; i >= **0**; i--) { **double** sum = **0**;

**for** (**int** j = i + **1**; j < result.length; j++) sum += result[j] \* A[i][j];

result[i] = (b[i] - sum) / A[i][i];

}

**return** result;

}

}

## Вычисление значений функции и производных:

**public** **class** **MultiFunction** { **double** c;

**public** **MultiFunction**(**double** parameter){ **this**.c = parameter;

}

**public** **double**[]getFunctionResult(**double**[]points){ **double**[]result = **new** **double**[points.length];

result[**0**] = Math.pow(points[**0**]-c,**2**)+Math.pow(points[**1**],**2**)-**1**; result[**1**] = Math.pow(points[**0**],**2**)+Math.pow(points[**1**]-c,**2**)-**100**; **return** result;

}

**public** **double**[][]getDerivativeFunctionResult(**double**[]points){ **double**[][]result = **new** **double**[points.length][points.length]; result[**0**][**0**] = **2**\*(points[**0**]-c);

result[**0**][**1**] = **2**\*points[**1**]; result[**1**][**0**] = **2**\*points[**0**]; result[**1**][**1**] = **2**\*(points[**1**]-c); **return** result;

}

}

## Код организации вычислений:

**public** **static** **void** **calculateSystem**(**double** parameter,DoubleInterval interval){

NewtonSystemCalculator calculator = **new** NewtonSystemCalculator(**new** MultiFunction(parameter)); var result = calculator.calculate(interval);

var functionXValues = calculator.getFunctionXValuesStatistic(); var functionYValues = calculator.getFunctionYValuesStatistic(); var statistic = calculator.getStatistics();

**int**[]iterations = IntStream.range(**1**,functionXValues.length).toArray(); Painter.drawSimpleGraphics(iterations, **new** **double**[][]{functionXValues,functionYValues},

**new** String[]{"X","Y"},

"Root: (x,y) = ("+result[**0**]+","+result[**1**]+")");

**double**[]curStatistic = statistic.stream().mapToDouble(**Double:**:doubleValue).toArray(); Painter.drawLogarithmicGraphics(iterations, **new** **double**[][]{curStatistic},**new** String[]{"Norm"},

"Root: (x,y) = ("+result[**0**]+","+result[**1**]+")");

}

## Интервалы вычислений:

**static** DoubleInterval[] systemIntervals = **new** DoubleInterval[]{

**new** DoubleInterval(**7**,**8**,-**0.5**,-**1**),

**new** **DoubleInterval**(**7.4**,**7.6**,**0**,-**0.2**), **new** DoubleInterval(**6.6**,**6.8**,-**0.8**,-**1**),

};

## Код всего приложения:

## Класс BisectionCalculator:

**package** com.company.calculators;

**import** **com.company.Function**;

**public** **class** **BisectionCalculator** **extends** Calculator {

**public** **BisectionCalculator**(Function function){

**super**(function);

}

**@Override**

**protected** **double** **getNextApproximation**(**double** x) **throws** Exception {

**if**(function.getFunctionResult(a)\*function.getFunctionResult(x)<**0**)

b = x;

**else**

a = x;

**return** (a+b)/**2**;

}

}

## Абстрактный класс Calculator:

**package** com.company.calculators;

**import** **com.company.Function**;

**import** **com.company.Interval**;

**import** **java.util.ArrayList**;

**public** **abstract** **class** **Calculator** {

**protected** **final** **double** PRECISION = **1**E-**12**;

**protected** ArrayList<Double> statistics;

**protected** Function function;

**protected** **double** a,b;

**public** **Calculator**(Function function){

**this**.function = function;

}

**public** **double** **calculate**(Interval interval) **throws** Exception {

a = interval.a();

b = interval.b();

statistics = **new** ArrayList<>();

**double** x = interval.a();

**double** functionFromA = function.getFunctionResult(a);

**double** functionFromB = function.getFunctionResult(b);

**if**(functionFromA\*functionFromB>=**0**)

**throw** **new** **Exception**("incorrect interval");

**int** iteration = **1**;

**double** functionFromX = function.getFunctionResult(x);

**while**(Math.abs(functionFromX)>PRECISION && iteration<**100**){

statistics.add(Math.abs(functionFromX));

iteration++;

x = getNextApproximation(x);

functionFromX = function.getFunctionResult(x);

}

statistics.add(Math.abs(functionFromX));

**return** x;

}

**protected** **abstract** **double** **getNextApproximation**(**double** x) **throws** Exception;

**public** Double[] **getStatistic**(){

**return** statistics.toArray(Double[]::**new**);

}

}

## Класс MultiFunction:

**package** com.company.calculators;

**public** **class** **MultiFunction** {

**double** c;

**public** **MultiFunction**(**double** parameter){

**this**.c = parameter;

}

**public** **double**[]getFunctionResult(**double**[]points){

**double**[]result = **new** **double**[points.length];

result[**0**] = Math.pow(points[**0**]-c,**2**)+Math.pow(points[**1**],**2**)-**1**;

result[**1**] = Math.pow(points[**0**],**2**)+Math.pow(points[**1**]-c,**2**)-**100**;

**return** result;

}

**public** **double**[][]getDerivativeFunctionResult(**double**[]points){

**double**[][]result = **new** **double**[points.length][points.length];

result[**0**][**0**] = **2**\*(points[**0**]-c);

result[**0**][**1**] = **2**\*points[**1**];

result[**1**][**0**] = **2**\*points[**0**];

result[**1**][**1**] = **2**\*(points[**1**]-c);

**return** result;

}

}

## Класс NewtonCalculator:

**package** com.company.calculators;

**import** **com.company.Function**;

**public** **class** **NewtonCalculator** **extends** Calculator {

**public** **NewtonCalculator**(Function function) {

**super**(function);

}

**@Override**

**protected** **double** **getNextApproximation**(**double** x) **throws** Exception {

**return** x-function.getFunctionResult(x)/function.getDerivativeResult(x);

}

}

## Интерфейс Function:

**package** com.company;

**public** **interface** **Function** {

**double** **getFunctionResult**(**double** point) **throws** Exception;

**double** **getDerivativeResult**(**double** point) **throws** Exception;

}

## Класс MyFunction:

**package** com.company;

**public** **class** **MyFunction** **implements** Function{

**@Override**

**public** **double** **getFunctionResult**(**double** x) **throws** Exception{

**if**(x == **1**)

**throw** **new** **Exception**("incorrect x");

**return** ((x/**2**)+**3**\*Math.cos(x))/(x-**1**);

}

**@Override**

**public** **double** **getDerivativeResult**(**double** x) **throws** Exception {

**return** ((**0.5**-**3**\*Math.sin(x))\*(x-**1**)-(x/**2**+**3**\*Math.cos(x)))/Math.pow((x-**1**),**2**);

}

}

## Класс NewtonSystemCalculators:

**package** com.company;

**import** **com.company.calculators.MultiFunction**;

**import** **java.util.ArrayList**;

**import** **java.util.stream.DoubleStream**;

**public** **class** **NewtonSystemCalculator** {

**protected** **final** **double** PRECISION = **1**E-**10**;

**protected** ArrayList<Double> statistics;

ArrayList<Double>functionXValuesStatistic;

ArrayList<Double>functionYValuesStatistic;

**protected** MultiFunction function;

**public** **NewtonSystemCalculator**(MultiFunction function) {

**this**.function = function;

}

**public** **double**[] **calculate**(DoubleInterval interval) {

statistics = **new** ArrayList<>();

functionXValuesStatistic = **new** ArrayList<>();

functionYValuesStatistic = **new** ArrayList<>();

**double**[] point = **new** **double**[]{interval.x1(), interval.y1()};

**double** max;

**int** iteration = **1**;

**do** {

**double**[] functionValues = function.getFunctionResult(point);

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

functionValues[i] = -functionValues[i];

**double**[][] derivativeValues = function.getDerivativeFunctionResult(point);

**double**[] differences = getSolvesOfSystem(derivativeValues, functionValues);

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

point[i] += differences[i];

**double**[] residual = function.getFunctionResult(point);

functionXValuesStatistic.add(residual[**0**]);

functionYValuesStatistic.add(residual[**1**]);

max = Math.sqrt(DoubleStream.of(residual).map(a->Math.pow(a,**2**)).sum());

**if**(max!=**0**)

statistics.add(max);

iteration++;

} **while** (max > PRECISION && iteration<**102**);

**return** point;

}

**public** ArrayList<Double> **getStatistics**() {

**return** statistics;

}

**public** **double**[] **getFunctionXValuesStatistic**() {

**return** functionXValuesStatistic.stream().mapToDouble(**Double:**:doubleValue).toArray();

}

**public** **double**[] **getFunctionYValuesStatistic**() {

**return** functionYValuesStatistic.stream().mapToDouble(**Double:**:doubleValue).toArray();

}

**private** **double**[] **getSolvesOfSystem**(**double**[][] A, **double**[] b) {

**double**[] result = **new** **double**[b.length];

**for** (**int** k = **0**; k < A.length; k++) {

**for** (**int** i = k + **1**; i < A[**0**].length; i++) {

**double** l = A[i][k] / A[k][k];

**for** (**int** j = **0**; j < A[**0**].length; j++) {

A[i][j] -= l \* A[k][j];

}

b[i] -= l \* b[k];

}

}

**for** (**int** i = result.length - **1**; i >= **0**; i--) {

**double** sum = **0**;

**for** (**int** j = i + **1**; j < result.length; j++)

sum += result[j] \* A[i][j];

result[i] = (b[i] - sum) / A[i][i];

}

**return** result;

}

}

## Класс Painter:

**package** com.company;

**import** **org.jfree.chart.ChartFactory**;

**import** **org.jfree.chart.JFreeChart**;

**import** **org.jfree.chart.axis.LogarithmicAxis**;

**import** **org.jfree.chart.axis.NumberAxis**;

**import** **org.jfree.chart.plot.PlotOrientation**;

**import** **org.jfree.chart.renderer.xy.XYSplineRenderer**;

**import** **org.jfree.data.xy.XYSeries**;

**import** **org.jfree.data.xy.XYSeriesCollection**;

**import** **java.io.File**;

**import** **java.io.IOException**;

**import** **static** org.jfree.chart.ChartUtilities.saveChartAsJPEG;

**class** **Painter** {

**static** **int** index = **34**;

**public** **static** **void** **drawLogarithmicGraphics**(**int**[] numbers, **double**[][]differences, String[] graphicNames,String title){

XYSeriesCollection dataSet = **new** XYSeriesCollection();

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

**final** XYSeries series = **new** XYSeries(graphicNames[i], **false**, **true**);

**for**(**int** j = **0**;j<numbers.length;j++){

**if**(differences[i].length == j)

**break**;

series.add(numbers[j], differences[i][j]);

}

dataSet.addSeries(series);

}

JFreeChart lineChart = ChartFactory.createXYLineChart(title, "Number of iteration",

"Norm of difference",

dataSet, PlotOrientation.VERTICAL, **true**, **true**, **false**);

var plot = lineChart.getXYPlot();

XYSplineRenderer r1 = **new** XYSplineRenderer();

r1.setPrecision(**8**);

r1.setSeriesShapesVisible(**0**, **false**);

**final** NumberAxis rangeAxis = **new** LogarithmicAxis("Norm of difference (Log)");

plot.setRangeAxis(rangeAxis);

plot.setRenderer(r1);

**int** width = **1920**; /\* Width of the image \*/

**int** height = **1080**; /\* Height of the image \*/

File jFreeChart = **new** File(String.format("LineChart%s.jpeg", index));

index++;

**try** {

saveChartAsJPEG(jFreeChart, lineChart, width, height);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** **static** **void** **drawSimpleGraphics**(**int**[] numbers, **double**[][]differences, String[] graphicNames,String title){

XYSeriesCollection dataSet = **new** XYSeriesCollection();

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

**final** XYSeries series = **new** XYSeries(graphicNames[i], **false**, **true**);

**for**(**int** j = **0**;j<numbers.length;j++){

**if**(differences[i].length == j)

**break**;

series.add(numbers[j], differences[i][j]);

}

dataSet.addSeries(series);

}

JFreeChart lineChart = ChartFactory.createXYLineChart(title, "Number of iteration",

"Value of functions",

dataSet, PlotOrientation.VERTICAL, **true**, **true**, **false**);

XYSplineRenderer r1 = **new** XYSplineRenderer();

r1.setPrecision(**8**);

r1.setSeriesShapesVisible(**0**, **false**);

**int** width = **1920**;

**int** height = **1080**;

File jFreeChart = **new** File(String.format("LineChart%s.jpeg", index));

index++;

**try** {

saveChartAsJPEG(jFreeChart, lineChart, width, height);

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

## Класс Main:

**package** com.company;

**import** **com.company.calculators.BisectionCalculator**;

**import** **com.company.calculators.Calculator**;

**import** **com.company.calculators.MultiFunction**;

**import** **com.company.calculators.NewtonCalculator**;

**import** **java.util.Arrays**;

**import** **java.util.stream.IntStream**;

**public** **static** **void** **main**(String[] args) {

CalculationsProcessor bisectionProcessor = **new** CalculationsProcessor(**new** BisectionCalculator(**new** MyFunction()));

bisectionProcessor.calculate(bisectionIntervals);

CalculationsProcessor newtonProcessor = **new** CalculationsProcessor(**new** NewtonCalculator(**new** MyFunction()));

newtonProcessor.calculate(bisectionIntervals);

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

**double**[][]precisions = **new** **double**[**2**][**0**];

precisions[**0**] = bisectionProcessor.getPrecisionsWithIndex(i);

precisions[**1**] = newtonProcessor.getPrecisionsWithIndex(i);

**int** bisectionIterationsNumber = bisectionProcessor.getIterationsNumberWithIndex(i);

**int** newtonIterationNumber = newtonProcessor.getIterationsNumberWithIndex(i);

**int** max = Integer.max(bisectionIterationsNumber,newtonIterationNumber);

**int**[]iterations = IntStream.range(**1**,max).toArray();

Painter.drawLogarithmicGraphics(iterations,precisions,**new** String[]{"Bisection","Newton"},

"Root: "+bisectionProcessor.results[i]);

}

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

calculateSystem(parameters[i],systemIntervals[i]);

calculateSystem(parameters[parameters.length-**1**],systemIntervals[systemIntervals.length-**1**]);

}

}

## Класс CalculationsProcessor:

**package** com.company;

**import** **com.company.calculators.BisectionCalculator**;

**import** **com.company.calculators.Calculator**;

**import** **com.company.calculators.MultiFunction**;

**import** **com.company.calculators.NewtonCalculator**;

**import** **java.util.Arrays**;

**import java.util.stream.IntStream;**

**class** **CalculationsProcessor**{

**double**[]results;

Double[][]statistics;

Calculator calculator;

**public** **CalculationsProcessor**(Calculator calculator){

**this**.calculator = calculator;

}

**public** **void** **calculate**(Interval[]intervals) **throws** Exception {

results = **new** **double**[intervals.length];

statistics = **new** Double[results.length][**0**];

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

results[i] = calculator.calculate(intervals[i]);

var localStatistic = calculator.getStatistic();

statistics[i] = Arrays.copyOf(localStatistic,localStatistic.length);

}

}

**public** **double**[] **getPrecisionsWithIndex**(**int** i){

**double**[]result = **new** **double**[statistics[i].length];

**for**(**int** j = **0**;j<statistics[i].length;j++)

result[j] = statistics[i][j];

**return** result;

}

**public** **int** **getIterationsNumberWithIndex**(**int** i){

**return** statistics[i].length;

}

}

**public** **class** **Main** {

**static** **double**[] parameters = **new** **double**[]{

**6.36396105**,

**6.5**,

};

**static** DoubleInterval[] systemIntervals = **new** DoubleInterval[]{

**new** **DoubleInterval**(**7**,**8**,-**0.5**,-**1**),

**new** **DoubleInterval**(**7.4**,**7.6**,**0**,-**0.2**),

**new** **DoubleInterval**(**6.6**,**6.8**,-**0.8**,-**1**),

};

**public** **static** **void** **calculateSystem**(**double** parameter,DoubleInterval interval){

NewtonSystemCalculator calculator = **new** NewtonSystemCalculator(**new** MultiFunction(parameter));

var result = calculator.calculate(interval);

var functionXValues = calculator.getFunctionXValuesStatistic();

var functionYValues = calculator.getFunctionYValuesStatistic();

var statistic = calculator.getStatistics();

**int**[]iterations = IntStream.range(**1**,functionXValues.length).toArray();

Painter.drawSimpleGraphics(iterations, **new** **double**[][]{functionXValues,functionYValues},

**new** String[]{"X","Y"},

"Root: (x,y) = ("+result[**0**]+","+result[**1**]+")");

**double**[]curStatistic = statistic.stream().mapToDouble(**Double:**:doubleValue).toArray();

Painter.drawLogarithmicGraphics(iterations, **new** **double**[][]{curStatistic},**new** String[]{"Norm"},

"Root: (x,y) = ("+result[**0**]+","+result[**1**]+")");

}