Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №3
Дисциплина «Вычислительная математика»

Решение нелинейных уравнений

Вариант:
аг1

Выполнил:
Студент группы Р3212
Анищенко Анатолий Алексеевич

Преподаватель:
Перл Ольга Вячеславовна

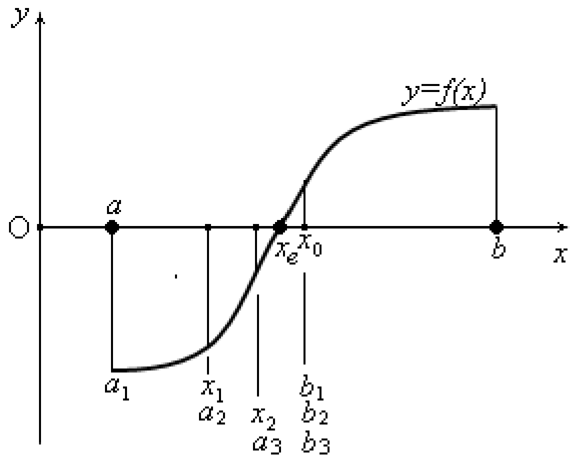
г. Санкт-Петербург
2020 г.

Цель работы:

Реализовать метод деления пополам и метод простой итераций для решения нелинейных уравнений и реализовать решение систем линейных уравнений методом простой итерации.

Описание использованных методов:

Метод деления пополам:



Суть метода заключается в том, что на каждой итерации отрезок на котором мы ищем решение уменьшается в 2 раза.

Рабочая формула метода: $x_i = \frac{a_i+b_i}{2}$

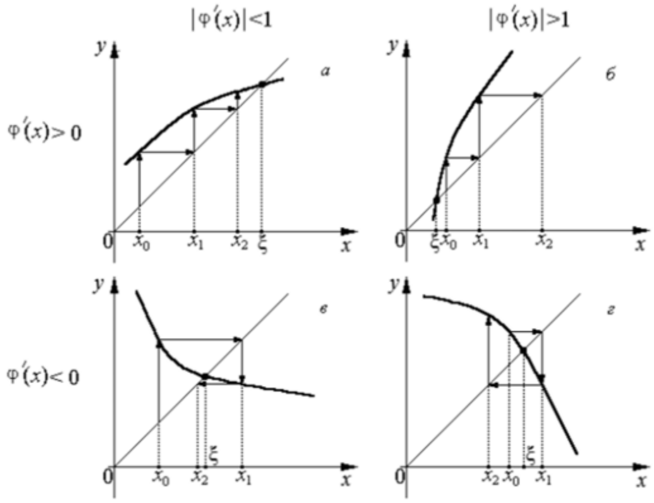
Для нахождения корня уравнения. Будем последовательно сужать отрезок поиска, и когда заданная точность будет достигнута результат будет найден.

Метод половинного деления всегда сходится. Скорость сходимости линейна.

Критерий окончания итерационного процесса: $|b_n - a_n| \leq \varepsilon$ или $|f(x_n)| \leq \ \varepsilon$.

Приближенное значение корня: $x^* = \frac{a_n+b_n}{2}$ или $x^* = \ a_n$ или $x^* = \ b_n$

Метод простой итерации:



Суть метода заключается в том, что уравнение $f(x) = 0$ с помощью некоторых преобразований необходимо переписать в виде $x \ = \ \varphi(x)$.

Рабочая формула метода: $x_{i+1} \ = \ \varphi(x_i)$.

Уравнение $f(x) = 0$ эквивалентно уравнению $x \ = \ x \ + \ \lambda * f(x)$ для любой $\lambda \ \neq \ 0$.

Для нахождения корня уравнения $x \ = \ \varphi(x)$ выберем некоторое начальное значение x_0, которое должно находиться как можно ближе к корню уравнения. Дальше с помощью итерационной формулы $x_{n+1} \ = \ \varphi(x_n)$ будем находить каждое следующее приближение корня уравнения.

Условия сходимости метода простой итерации определяются теоремой:

Если в некоторой $\sigma -$окрестности корня x^* уравнения $f(x) = 0$ функция $x \ = \ \varphi(x)$ дифференцируема и удовлетворяет неравенству $|\varphi'(x)| < q$, где $0 \ \leq \ q \ < \ 1$ постоянная, то независимо от выбора начального приближения x_0 из указанной $\sigma -$окрестности итерационная последовательность x_n не выходит из этой окрестности, метод сходится со скоростью геометрической прогрессии.

Достаточное условие сходимости метода: $|\varphi'(x)| \ \leq \ q \ < \ 1$, где q – некоторая константа.

Критерий окончания итерационного процесса: $|x_n - x_{n-1}| \ \leq \ \varepsilon$ или

Метод Ньютона для решения СНАУ:

Метод Ньютона решения систем нелинейных уравнений является обобщением метода Ньютона решения нелинейных уравнений, который основан на идее линеаризации.

Взяв некоторое x_0 в качестве начального приближения решения, мы можем построить линейную аппроксимацию $F(x)$ в окрестности x_0: $F(x_0 + h) \approx F(x_0) + F'(x_0) * h$ и решить получающееся линейное уравнение $F(x_0) + F'(x_0) * h = 0$.

$x^{(k+1)} = x^{(k)} - W^{-1}\left(x^{(k)}\right) * F\left(x^{(k)}\right), k = 0,1,2, \dots$, где

$$W(x) = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{pmatrix} - \text{матрица Якоби}$$

Так как процесс вычисления обратной матрицы является трудоемким, преобразуем наше уравнение следующим образом:

$\Delta x^{(k)} = -W^{-1}\left(x^{(k)}\right) * F\left(x^{(k)}\right), k = 0,1,2, \dots$
где $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ $-$ поправка к текущему приближению $x^{(k)}$

$W\left(x^{(k)}\right) * \Delta x^{(k)} = F\left(x^{(k)}\right), k = 0, 1, 2, \dots$

В результате получена система линейных алгебраических уравнений относительно поправки $\Delta x^{(k)}$. После ее определения вычисляется следующее приближение $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$

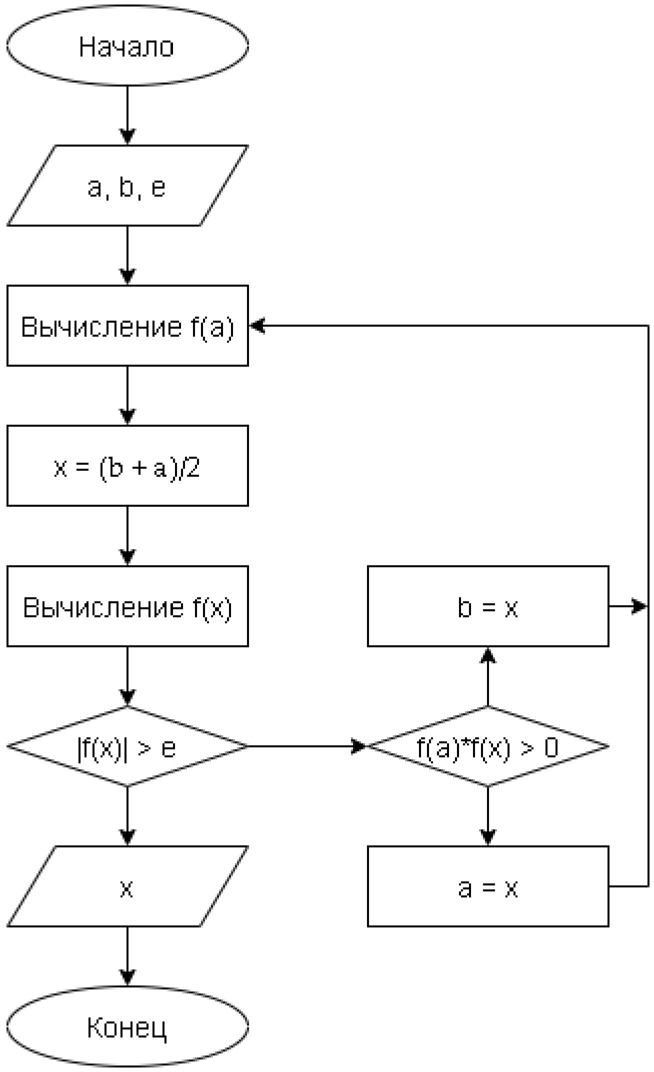
Существует теорема о достаточных условиях сходимости метода Ньютона.

Листинг:

Метод деления пополам:

```
static private NonlinearEquationSolutionResult bisectionMethodSolution(
        Function function,
        Bounds bounds,
        double accuracy
) throws
        NotAllowedScopeException,
        NoSolutionException,
        UnavailableCodeException {
    double left = bounds.getLeftBound();
    double right = bounds.getRightBound();

    for (long i = 0; i < N_MAX_VALUE; i++) {
        double x = (left + right) / 2.0d;

        double leftValue = function.getValue(left);
        double value = function.getValue(x);

        if (value * leftValue > 0) {
            left = x;
        } else {
            right = x;
        }

        if (Math.abs(right - left) < accuracy || Math.abs(value) < accuracy) {
            return new NonlinearEquationSolutionResult(x, value, i);
        }
```

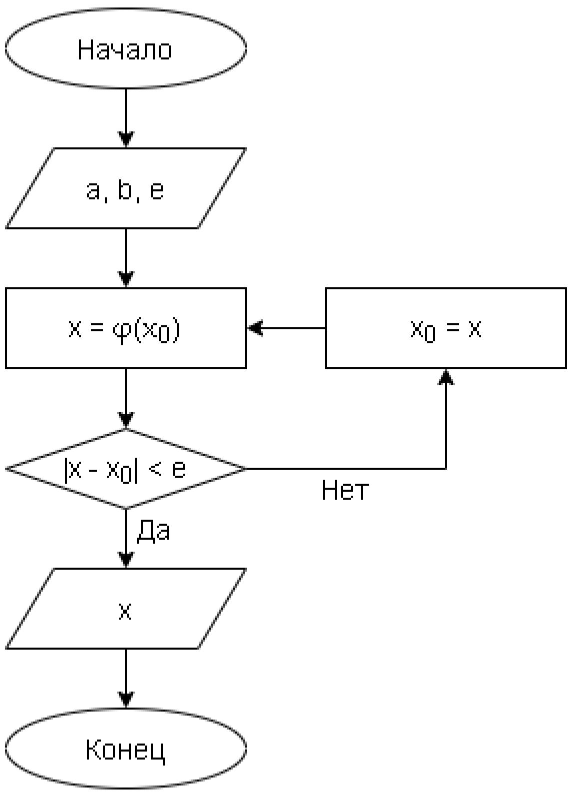
```
    }

    throw new NoSolutionException("count of iterations more than 10_000_000");
}
```



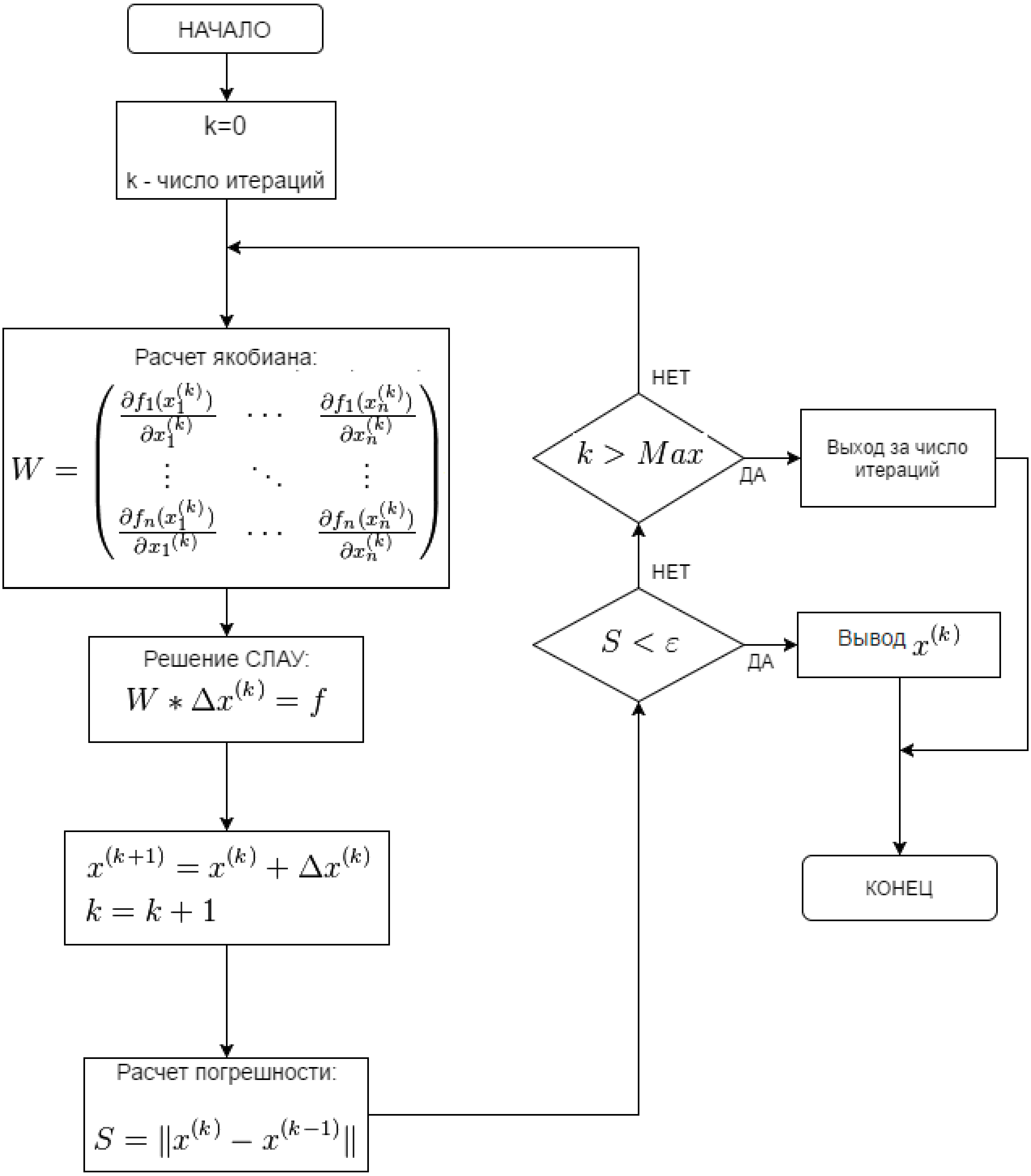
Метод простых итераций:

```
static private NonlinearEquationSolutionResult iterativeMethodSolution(
        Function function,
        Bounds bounds,
        double accuracy
) throws
        NotImplementedMethodException,
        NotAllowedScopeException,
        NoSolutionException,
        UnavailableCodeException {
    double lambda = -1 / function.getDerivative().getMaxValue(bounds);
    Function phi = new DerivativeFunc() {
        @Override
        public double get(double argument) throws UnavailableCodeException {
            try {
                return argument + lambda  * function.getValue(argument);
            } catch (Exception e) {
                throw new UnavailableCodeException();
            }
        }

        @Override
```

```java
        public Interval[] getNotAllowedScope() throws UnavailableCodeException {
            try {
                return function.getDerivative().getNotAllowedScope();
            } catch (NotImplementedMethodException e) {
                throw new UnavailableCodeException();
            }
        }
    };

    double prev_x = bounds.getLeftBound();
    double x;

    for (long i = 0; i < N_MAX_VALUE; i++) {
        x = phi.getValue(prev_x);

        if (Math.abs(x - prev_x) < accuracy || Math.abs(function.getValue(x)) < accuracy) {
            return new NonlinearEquationSolutionResult(x, function.getValue(x), i);
        }
    }

    prev_x = bounds.getRightBound();

    for (long i = 0; i < N_MAX_VALUE; i++) {
        x = phi.getValue(prev_x);

        if (Math.abs(x - prev_x) < accuracy || Math.abs(function.getValue(x)) < accuracy) {
            return new NonlinearEquationSolutionResult(x, function.getValue(x), i);
        }
    }

    throw new NoSolutionException("count of iterations more than 10_000_000");
}
```



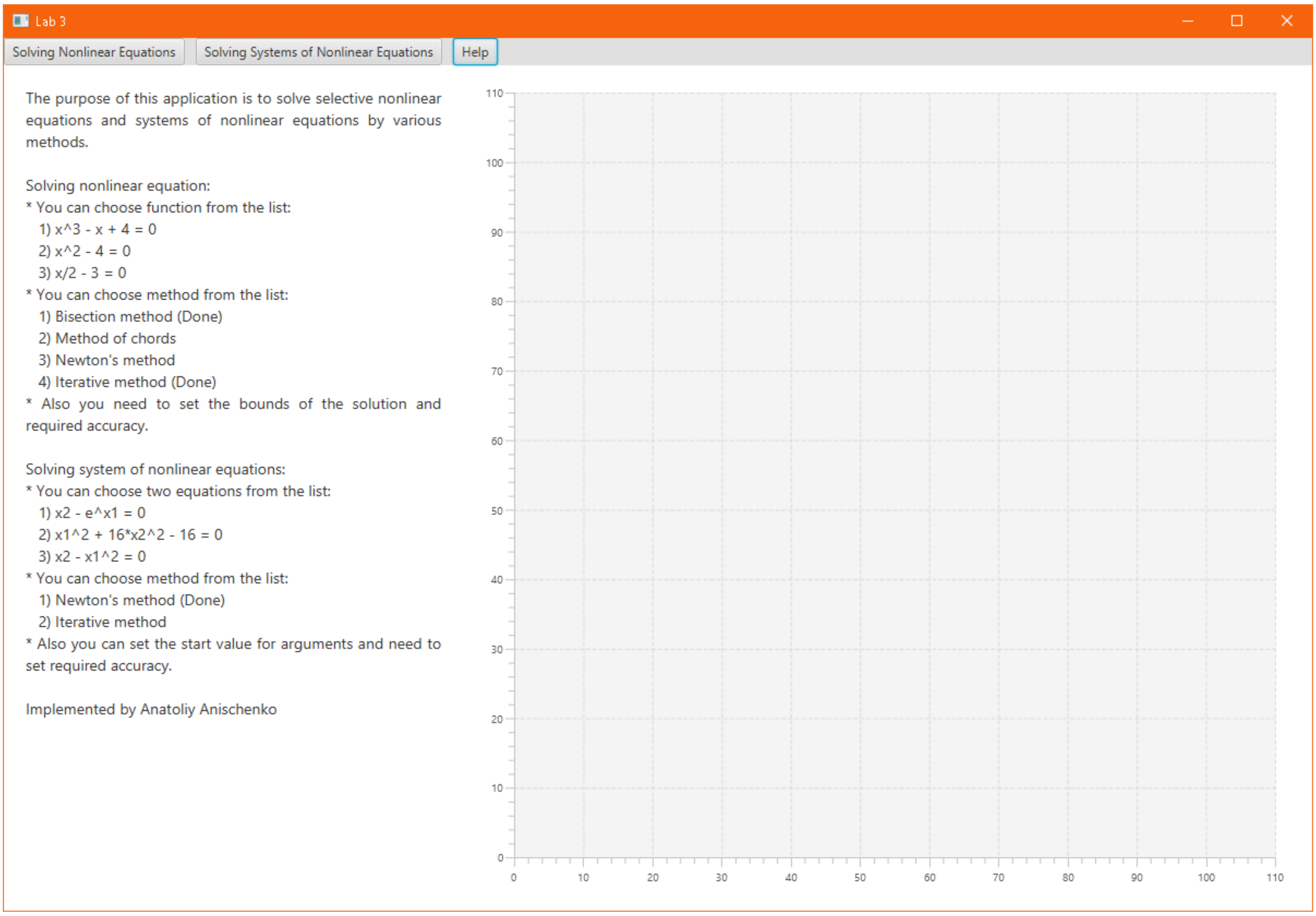
Метод Ньютона для СНАУ:

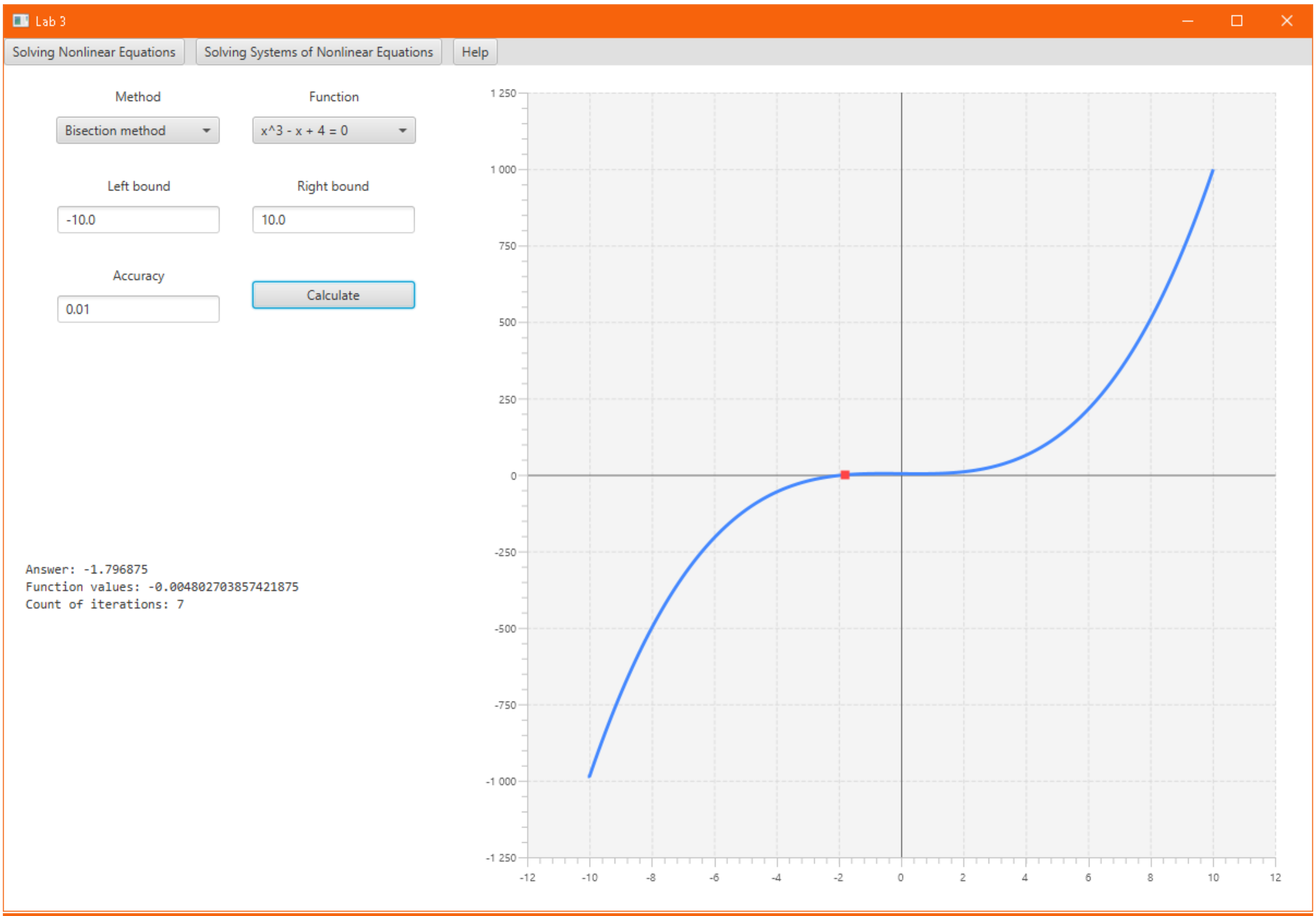


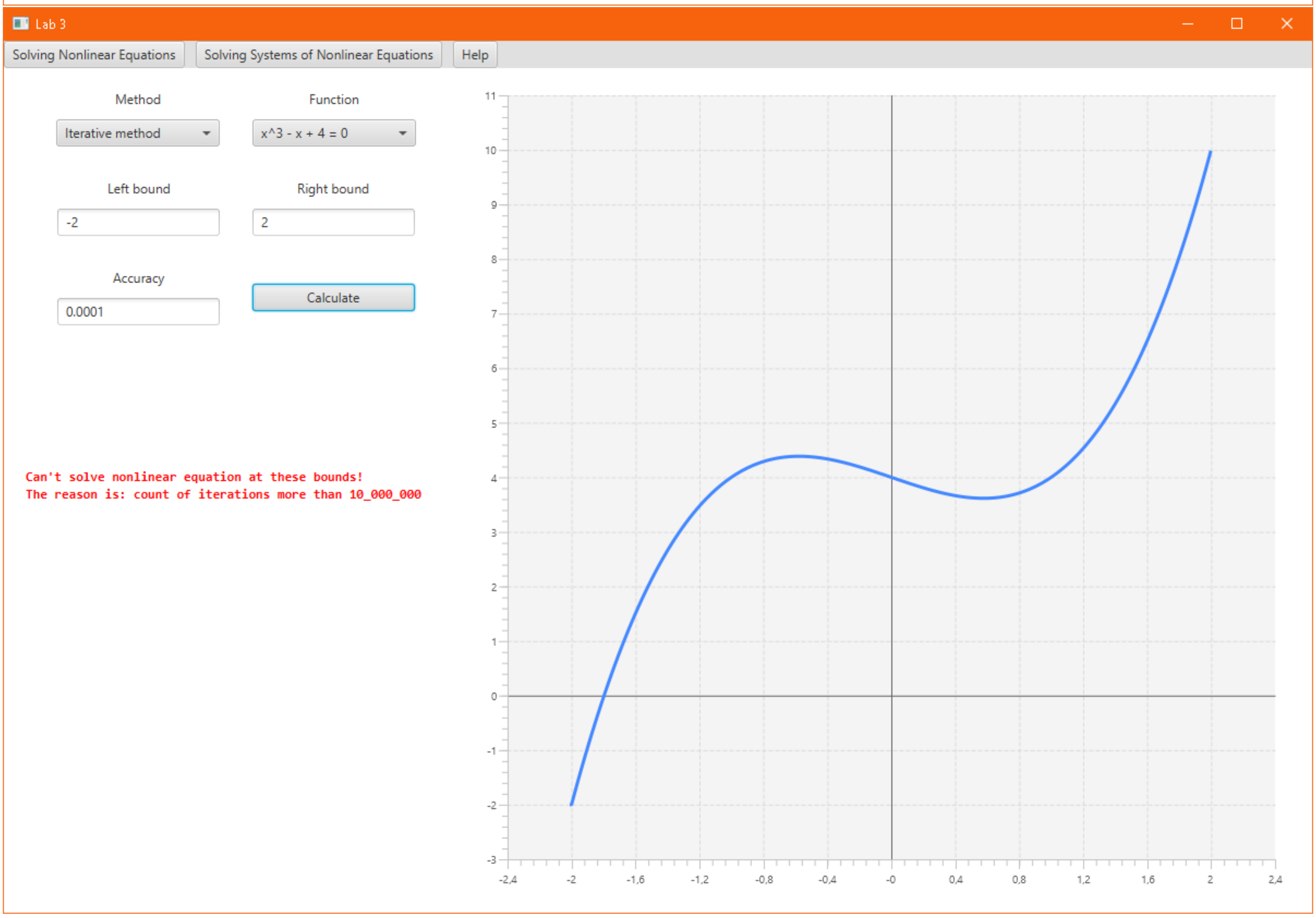

```java
static private SystemOfNonlinearEquationsSolutionResult newtonMethodSolution(
        SystemOfNonlinearEquations system,
        ArrayList<Double> startValue,
        double accuracy
) throws NoSolutionException, UnavailableCodeException {
    ArrayList<Double> x = new ArrayList<>(startValue);

    for (int i = 0; i < N_MAX_VALUE; i++) {
        ArrayList<Double> prev_x = new ArrayList<>(x);

        SystemOfLinearEquations<Double, LinearEquation> linearSystem = new
SystemOfLinearEquations<>();

        ArrayList<ArrayList<NonlinearEquation>> jacobianMatrix =
system.getJacobianMatrix();
        for (int j = 0; j < x.size(); j++) {
```

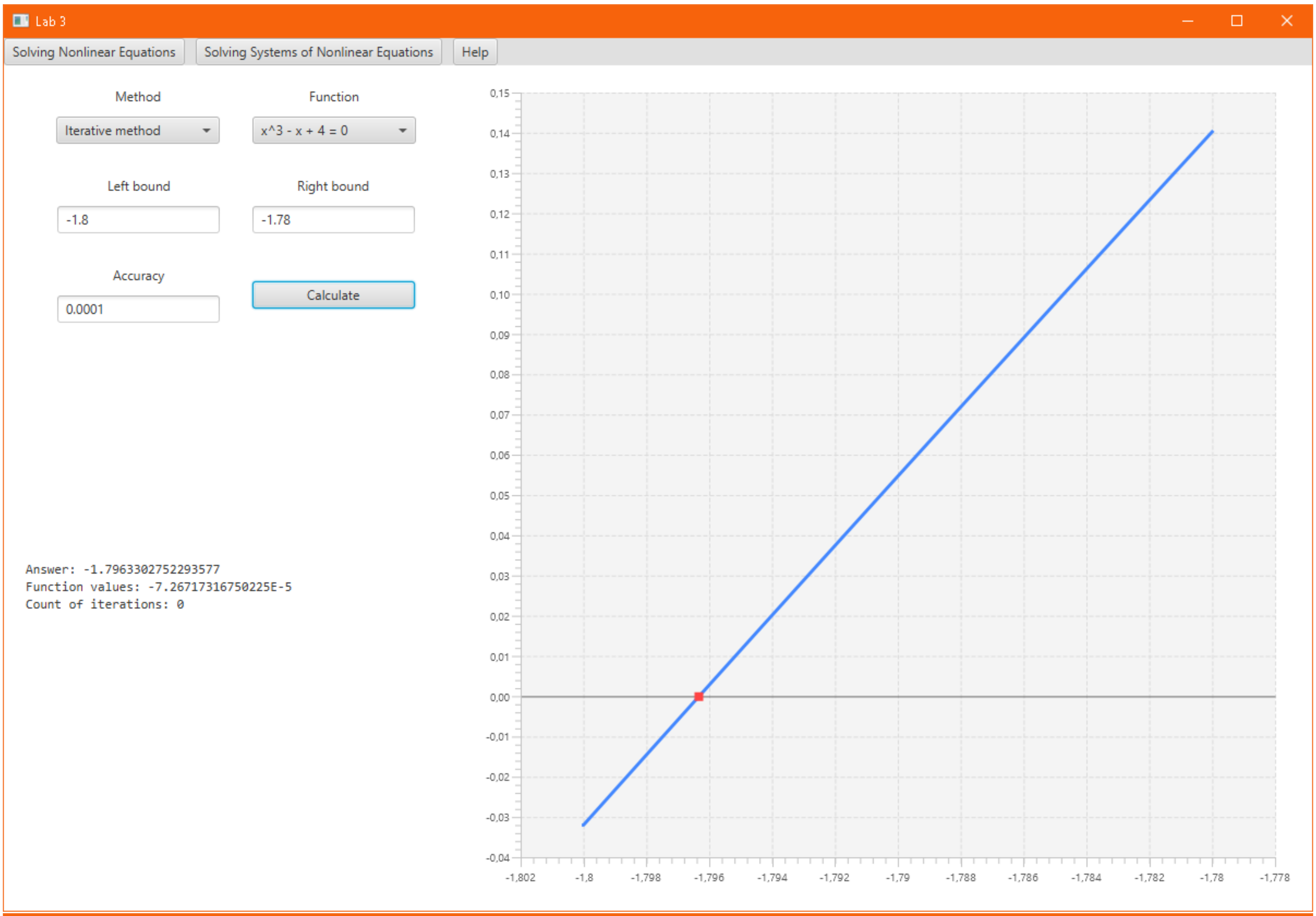
```
            ArrayList<Double> multipliers = new ArrayList<>();

            for (int k = 0; k < x.size(); k++) {
                multipliers.add(jacobianMatrix.get(j).get(k).getValue(prev_x));
            }

            LinearEquation curLinearEquation = new LinearEquation(multipliers, -system.get(j).getValue(prev_x));
            linearSystem.push(curLinearEquation);
        }

        ArrayList<Double> solution = new ArrayList<>(SystemOfLinearEquationsSolver.getSolution(linearSystem));

        for (int j = 0; j < solution.size(); j++) {
            x.set(j, prev_x.get(j) + solution.get(j));
        }

        if (getMeasuredError(x, prev_x) < accuracy) {
            return new SystemOfNonlinearEquationsSolutionResult(x, i);
        }
    }

    throw new NoSolutionException("count of iterations more than 10_000_000");
}
```

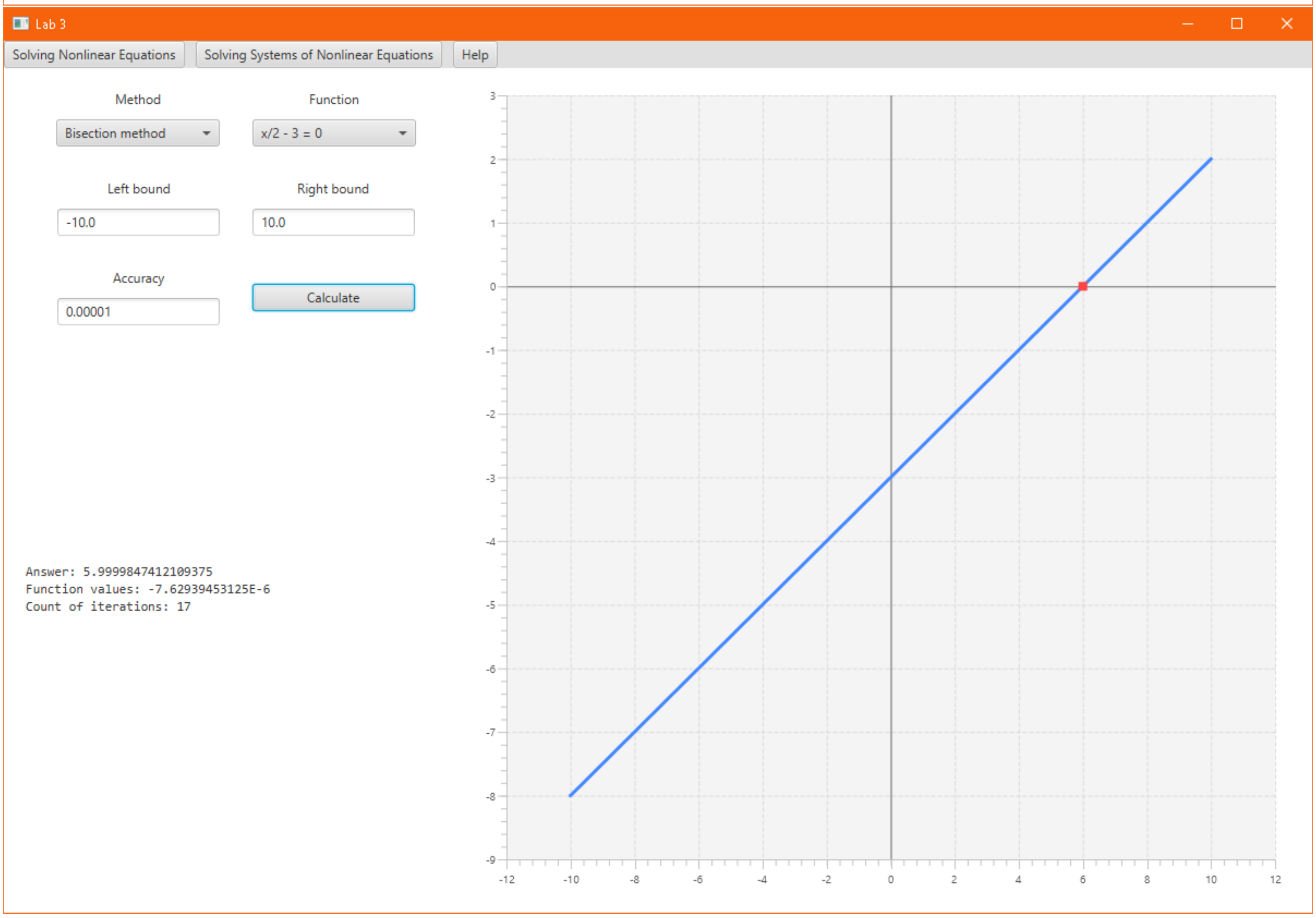
Результат работы:

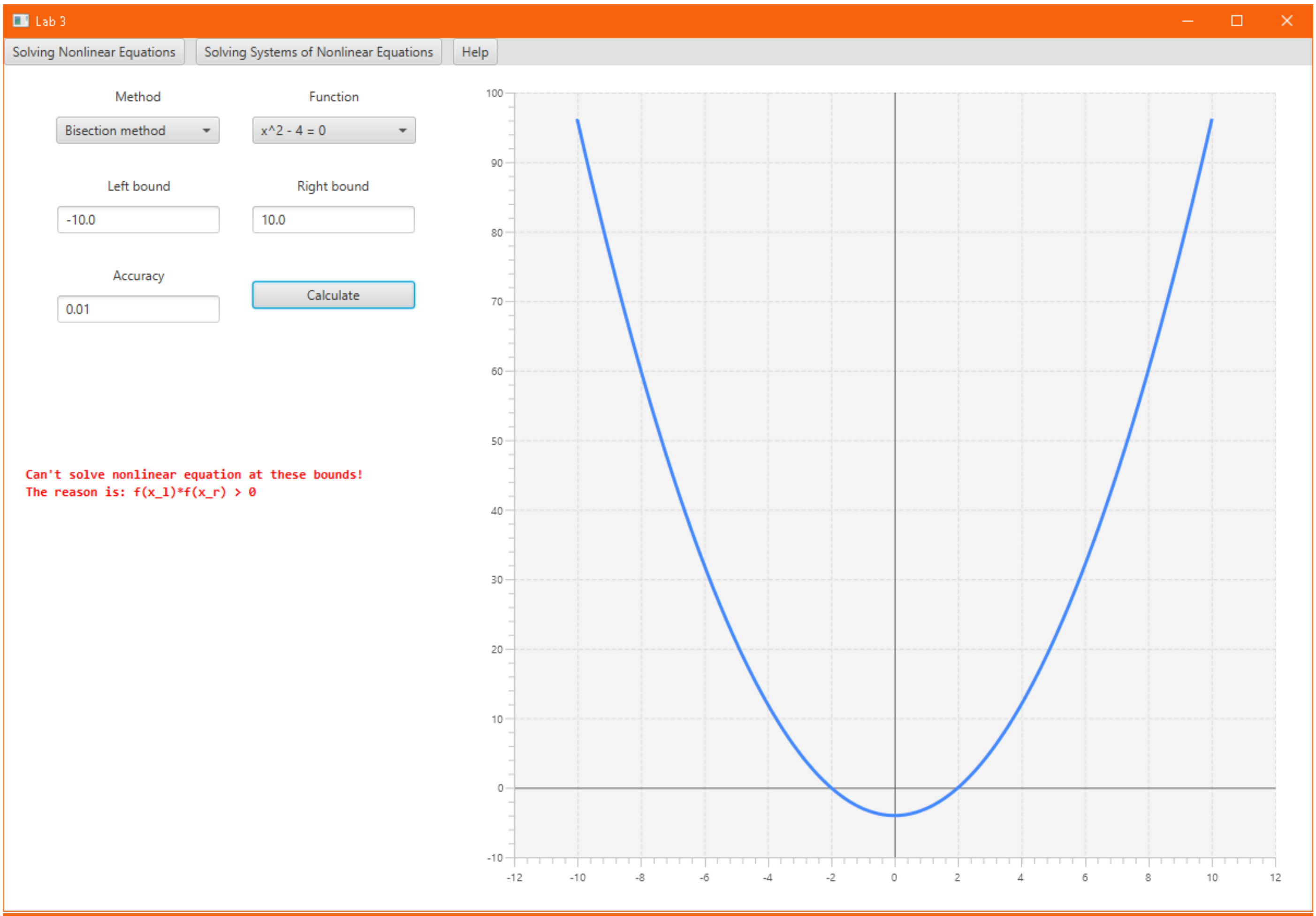


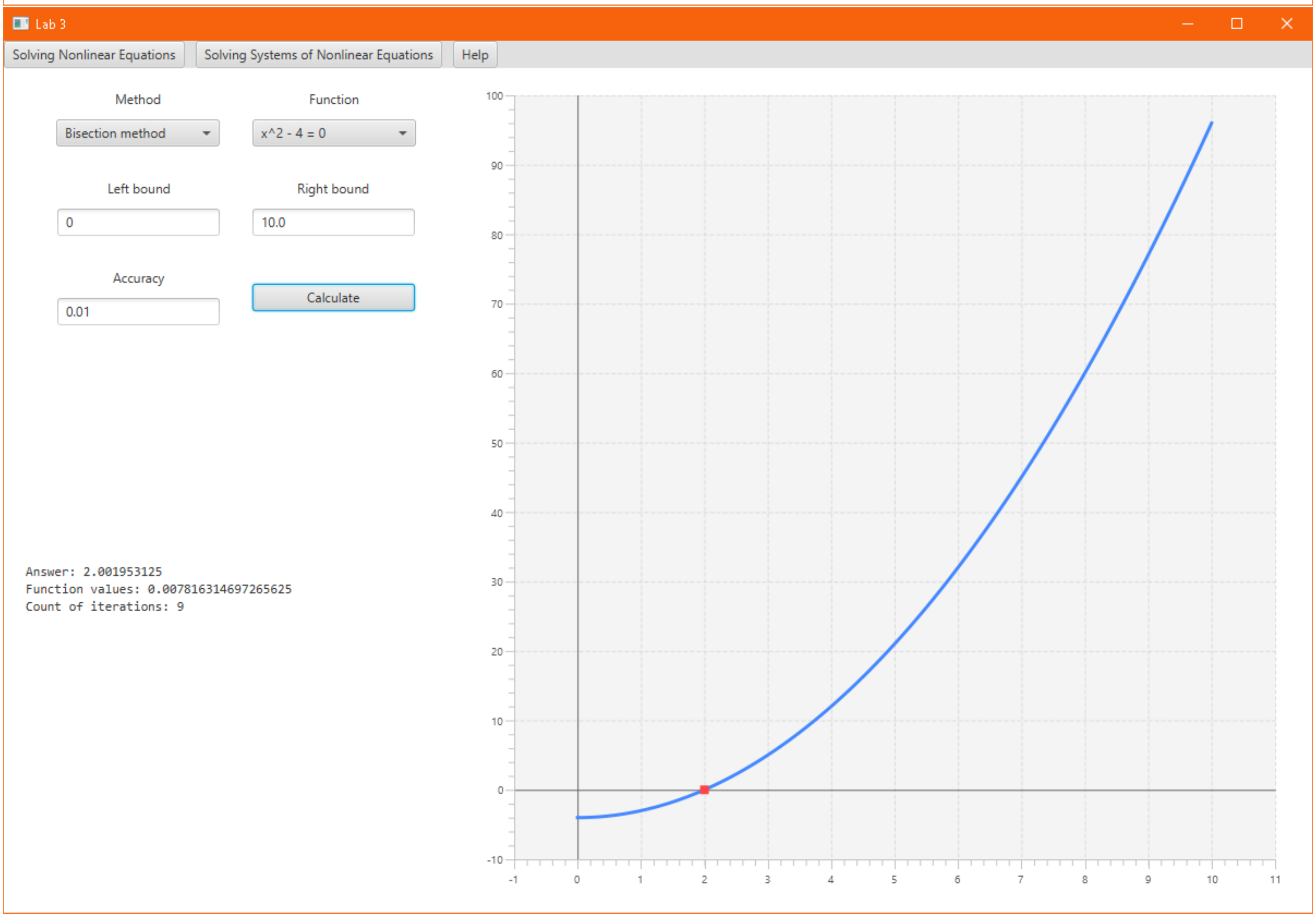

Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Bisection method
Function
x^3 - x + 4 = 0
Left bound
-10.0
Right bound
10.0
Accuracy
0.01
Calculate
Answer: -1.796875
Function values: -0.004802703857421875
Count of iterations: 7

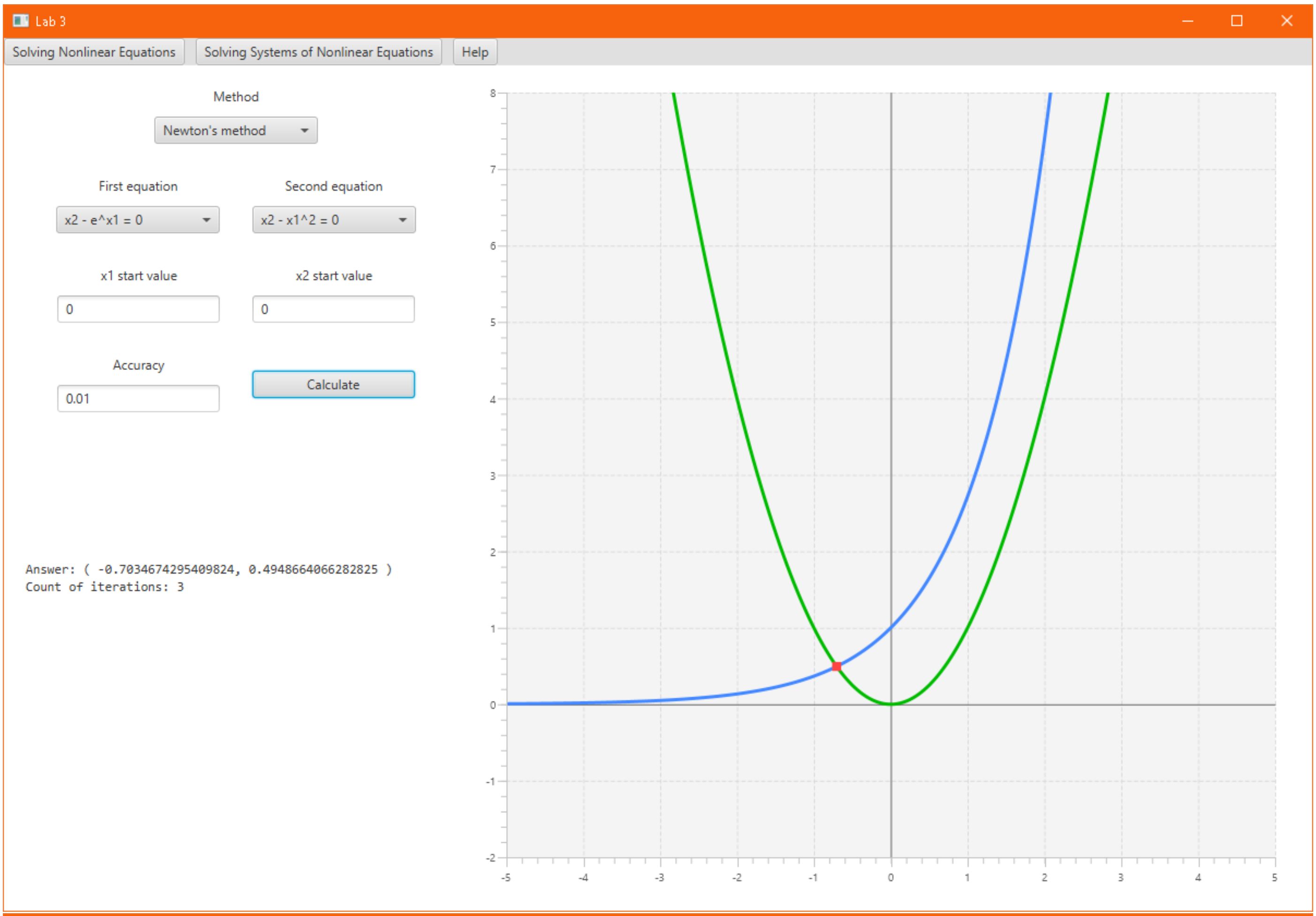

Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Iterative method
Function
x^3 - x + 4 = 0
Left bound
-2
Right bound
2
Accuracy
0.0001
Calculate
Can't solve nonlinear equation at these bounds!
The reason is: count of iterations more than 10_000_000

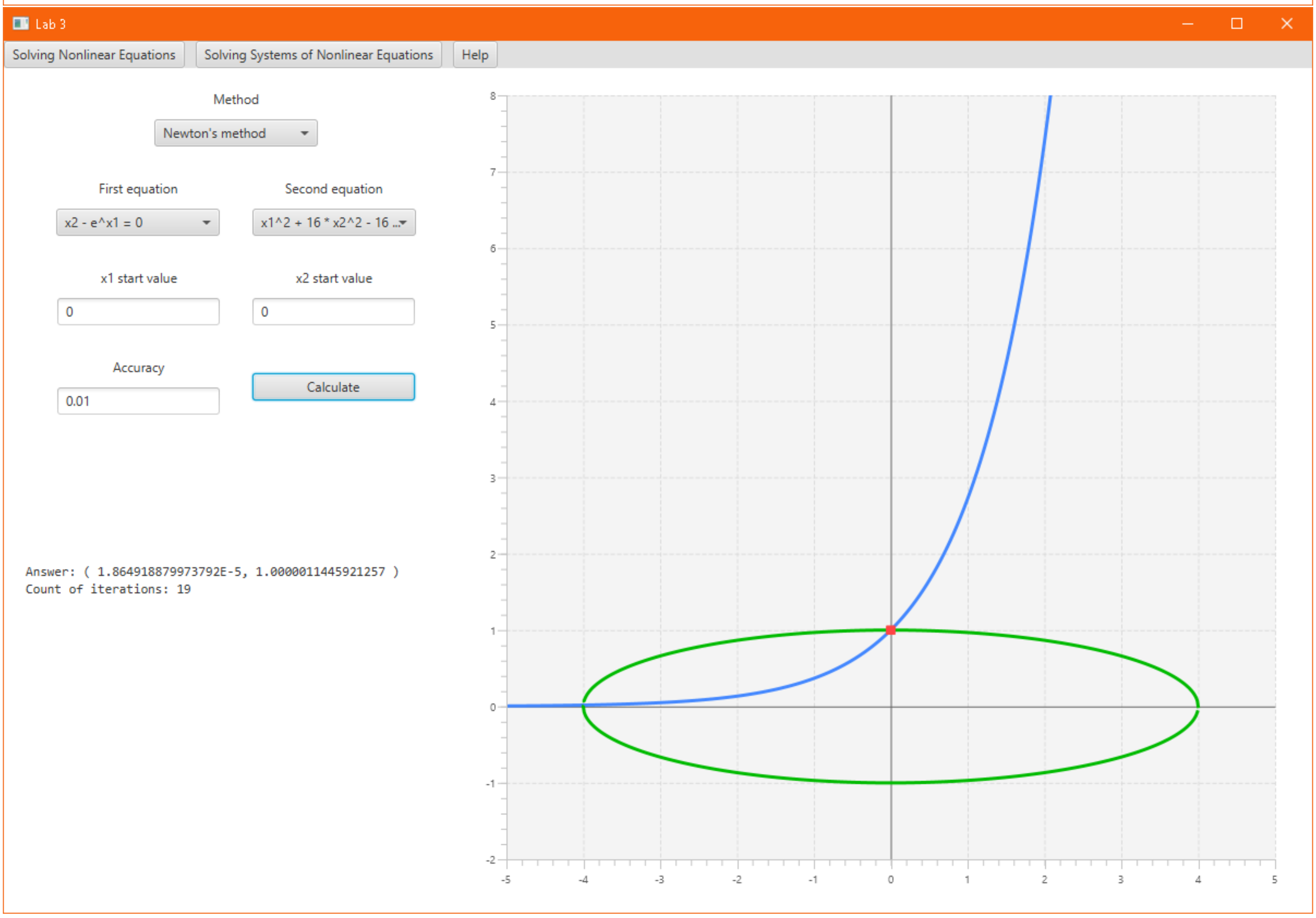

Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Iterative method
Function
x^3 - x + 4 = 0
Left bound
-1.8
Right bound
-1.78
Accuracy
0.0001
Calculate
Answer: -1.7963302752293577
Function values: -7.267173167502225E-5
Count of iterations: 0

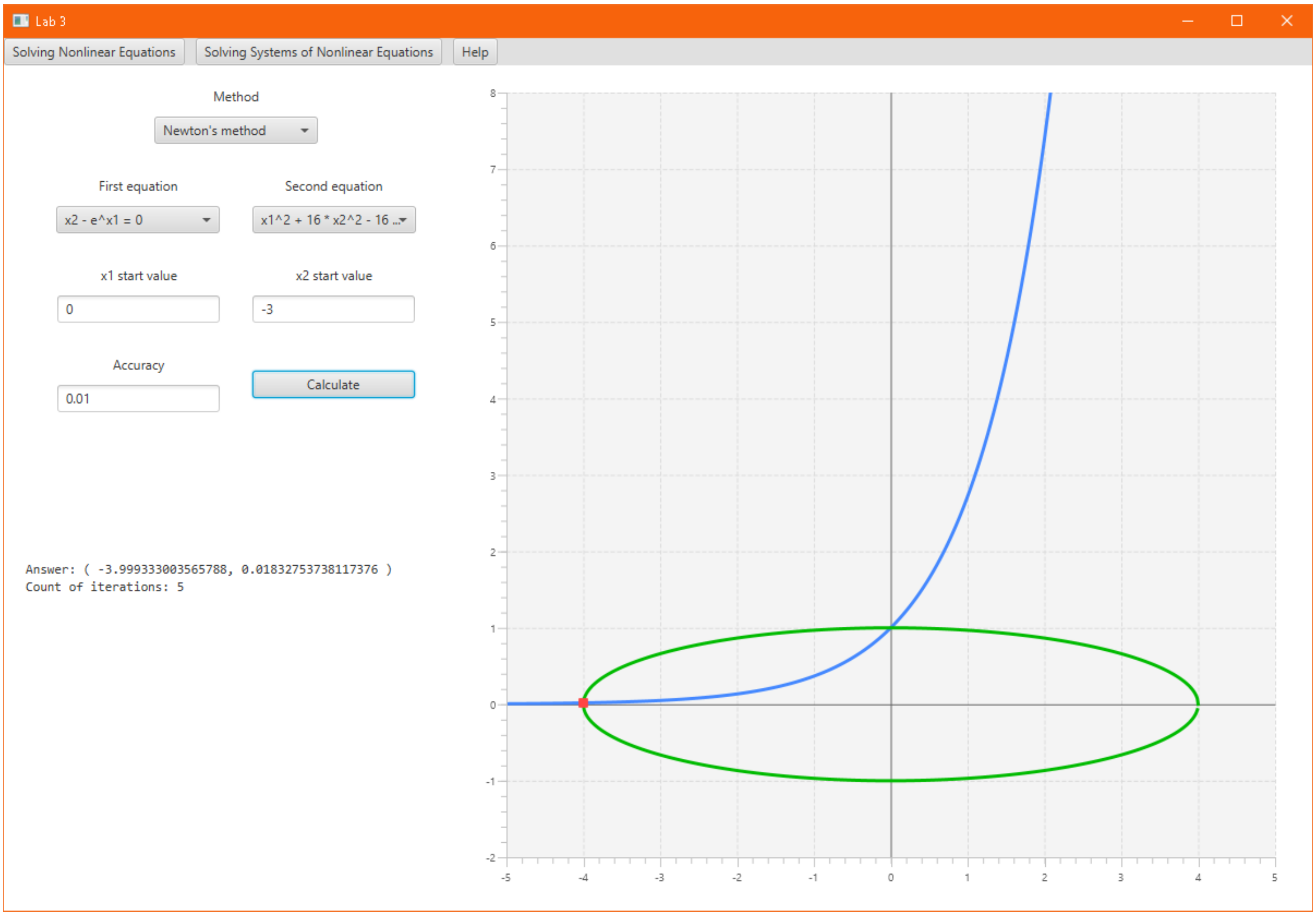

Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Bisection method
Function
x/2 - 3 = 0
Left bound
-10.0
Right bound
10.0
Accuracy
0.00001
Calculate
Answer: 5.9999847412109375
Function values: -7.62939453125E-6
Count of iterations: 17


Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Bisection method
Function
x^2 - 4 = 0
Left bound
-10.0
Right bound
10.0
Accuracy
0.01
Calculate
Can't solve nonlinear equation at these bounds!
The reason is: f(x_l)*f(x_r) > 0


Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Bisection method
Function
x^2 - 4 = 0
Left bound
0
Right bound
10.0
Accuracy
0.01
Calculate
Answer: 2.001953125
Function values: 0.007816314697265625
Count of iterations: 9


Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Newton's method
First equation
x2 - e^x1 = 0
Second equation
x2 - x1^2 = 0
x1 start value
0
x2 start value
0
Accuracy
0.01
Calculate
Answer: (-0.7034674295409824, 0.4948664066282825)
Count of iterations: 3


Lab 3
Solving Nonlinear Equations
Solving Systems of Nonlinear Equations
Help
Method
Newton's method
First equation
x2 - e^x1 = 0
Second equation
x1^2 + 16 * x2^2 - 16 ...
x1 start value
0
x2 start value
0
Accuracy
0.01
Calculate
Answer: (1.864918879973792E-5, 1.0000011445921257)
Count of iterations: 19



Выводы:

Методы решения нелинейных уравнений:

- Метод половинного деления:
    - Обладает абсолютной сходимостью (близость получаемого численного решения задачи к истинному решению)
    - Если интервал содержит несколько решений, неизвестно к какому сходимся.
    - Линейная сходимость
- Метод простой итерации:
    - Сходимость метода в малой окрестности корня и вытекающая отсюда необходимость выбора начального приближения к корню из этой малой окрестности
    - Требует подсчёта производной
    - Сходится со скоростью геометрической прогрессии
- Метод касательных (Ньютона)
    - Квадратичная скорость сходимости
    - Подсчёт производной
    - Условия сходимости (постоянный знак производных)
    - Выбор начального приближения
- Метод хорд
    - Скорость сходимости линейная (больше, чем у метода половинного деления)
    - Условия сходимости (постоянный знак производных)
    - Выбор начального приближения

Методы решения системы нелинейных уравнений:

- Метод простой итерации:
    - Проще реализация (не надо считать матрицу Якоби)

    - Скорость сходимости ниже
- Метод касательных (Ньютона)
    - Выше скорость сходимости
    - Реализация сложнее, на каждом шаге надо находить матрицу производных и решать систему линейных уравнений