

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.

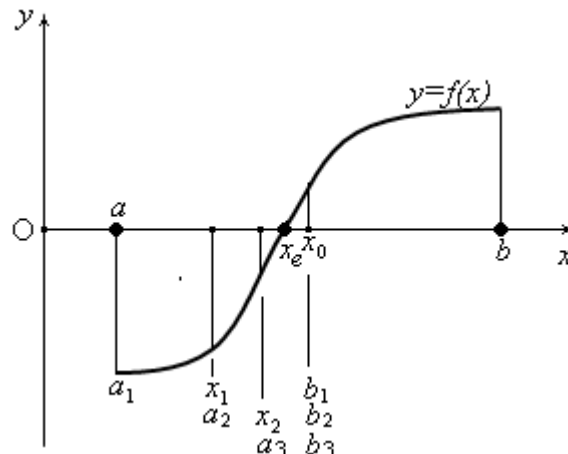
Рабочая формула метода: $x_i = \frac{a_i + b_i}{2}$

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

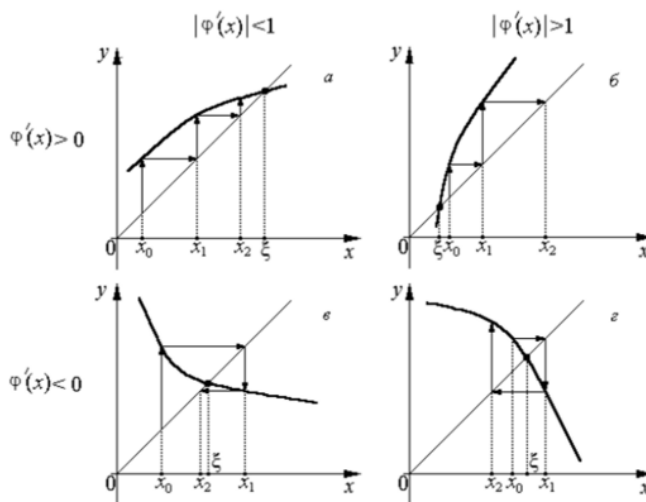
Уравнение $f(x) = 0$ эквивалентно уравнению $x = x + \lambda * f(x)$ для любой $\lambda \neq 0$.

Для нахождения корня уравнения $x = \varphi(x)$ выберем некоторое начальное значение x_0 , которое должно находиться как можно ближе к корню уравнения. Далее 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 σ – 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 σ – 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 $F(x_0) + F'(x_0) * h = 0$.

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

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

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

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

где $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ — поправка к текущему приближению $x^{(k)}$

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

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:

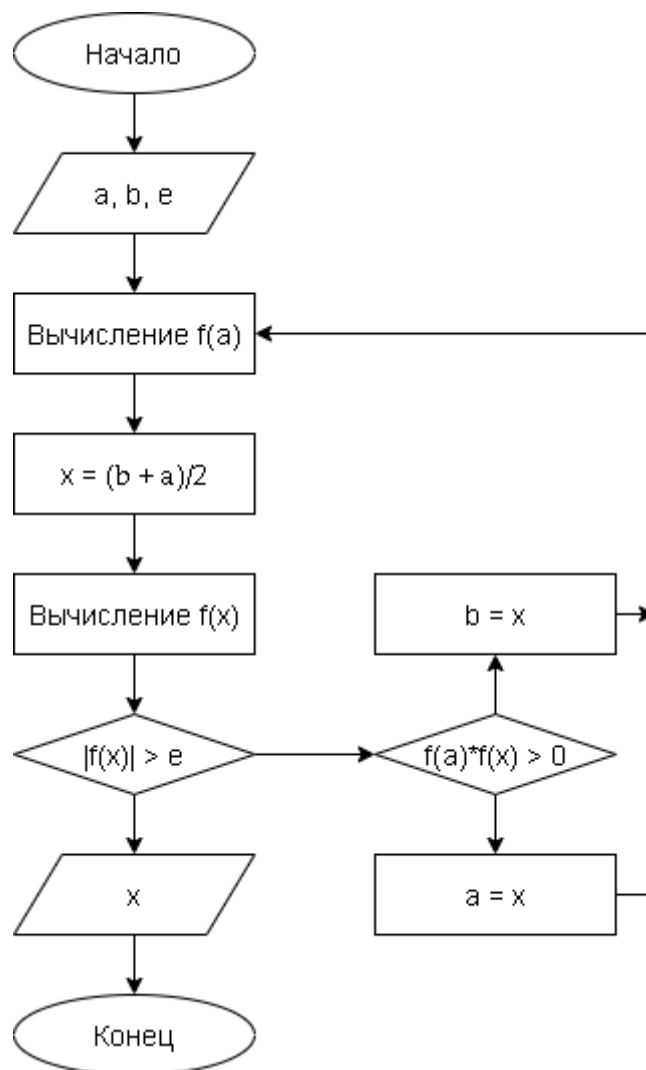
Метод деления пополам:

```
1 static private NonlinearEquationSolutionResult bisectionMethodSolution(  
2     Function function,  
3     Bounds bounds,  
4     double accuracy  
5 ) throws  
6     NotAllowedScopeException,  
7     NoSolutionException,  
8     UnavailableCodeException {  
9     double left = bounds.getLeftBound();  
10    double right = bounds.getRightBound();  
11  
12    for (long i = 0; i < N_MAX_VALUE; i++) {  
13        double x = (left + right) / 2.0d;  
14  
15        double leftValue = function.getValue(left);  
16        double value = function.getValue(x);  
17  
18        if (value * leftValue > 0) {  
19            left = x;  
20        } else {  
21            right = x;  
22        }  
23  
24        if (Math.abs(right - left) < accuracy || Math.abs(value) < accuracy) {
```

```

25         return new NonlinearEquationSolutionResult(x, value, i);
26     }
27 }
28
29 throw new NoSolutionException("count of iterations more than 10_000_000");
30 }

```



Метод простых итераций:

```

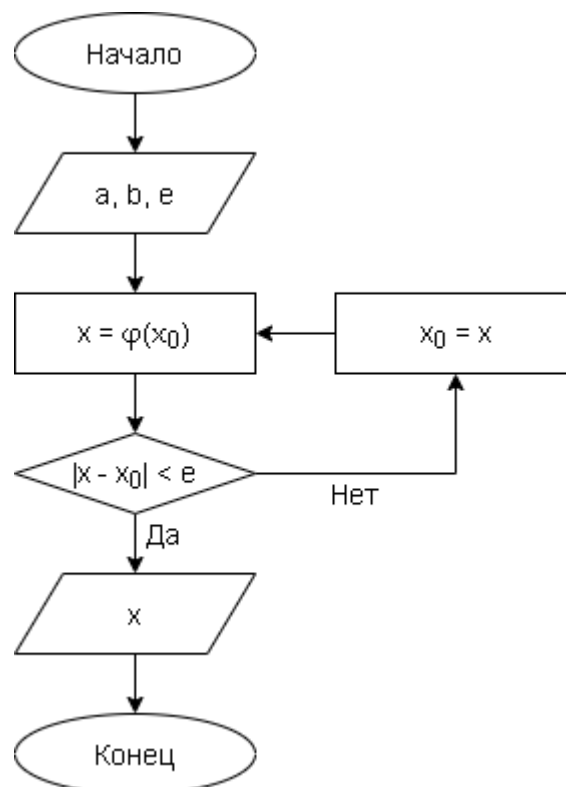
1 static private NonlinearEquationSolutionResult iterativeMethodSolution(
2     Function function,
3     Bounds bounds,
4     double accuracy
5 ) throws
6     NotImplementedException,
7     NotAllowedScopeException,
8     NoSolutionException,
9     UnavailableCodeException {
10     double lambda = -1 / function.getDerivative().getMaxValue(bounds);
11     Function phi = new DerivativeFunc() {
12         @Override
13         public double get(double argument) throws UnavailableCodeException {
14             try {
15                 return argument + lambda * function.getValue(argument);
16             } catch (Exception e) {
17                 throw new UnavailableCodeException();
18             }
19         }
20     };

```

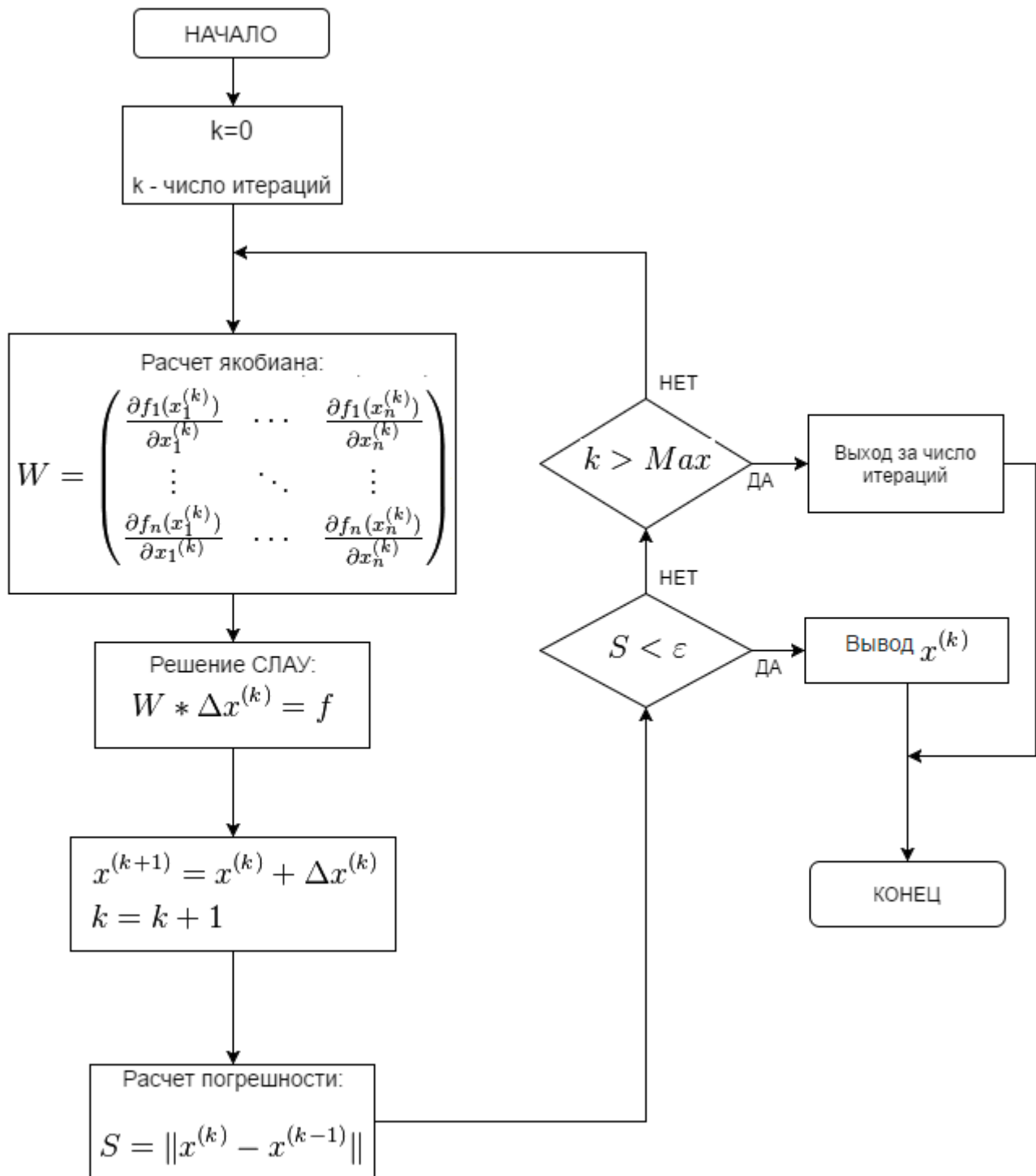
```

20
21     @Override
22     public Interval[] getNotAllowedScope() throws UnavailableCodeException {
23         try {
24             return function.getDerivative().getNotAllowedScope();
25         } catch (NotImplementedMethodException e) {
26             throw new UnavailableCodeException();
27         }
28     }
29 };
30
31 double prev_x = bounds.getLeftBound();
32 double x;
33
34 for (long i = 0; i < N_MAX_VALUE; i++) {
35     x = phi.getValue(prev_x);
36
37     if (Math.abs(x - prev_x) < accuracy || Math.abs(function.getValue(x)) <
accuracy) {
38         return new NonlinearEquationSolutionResult(x, function.getValue(x), i);
39     }
40 }
41
42 prev_x = bounds.getRightBound();
43
44 for (long i = 0; i < N_MAX_VALUE; i++) {
45     x = phi.getValue(prev_x);
46
47     if (Math.abs(x - prev_x) < accuracy || Math.abs(function.getValue(x)) <
accuracy) {
48         return new NonlinearEquationSolutionResult(x, function.getValue(x), i);
49     }
50 }
51
52 throw new NoSolutionException("count of iterations more than 10_000_000");
53 }

```



Метод Ньютона для СНАУ:



```

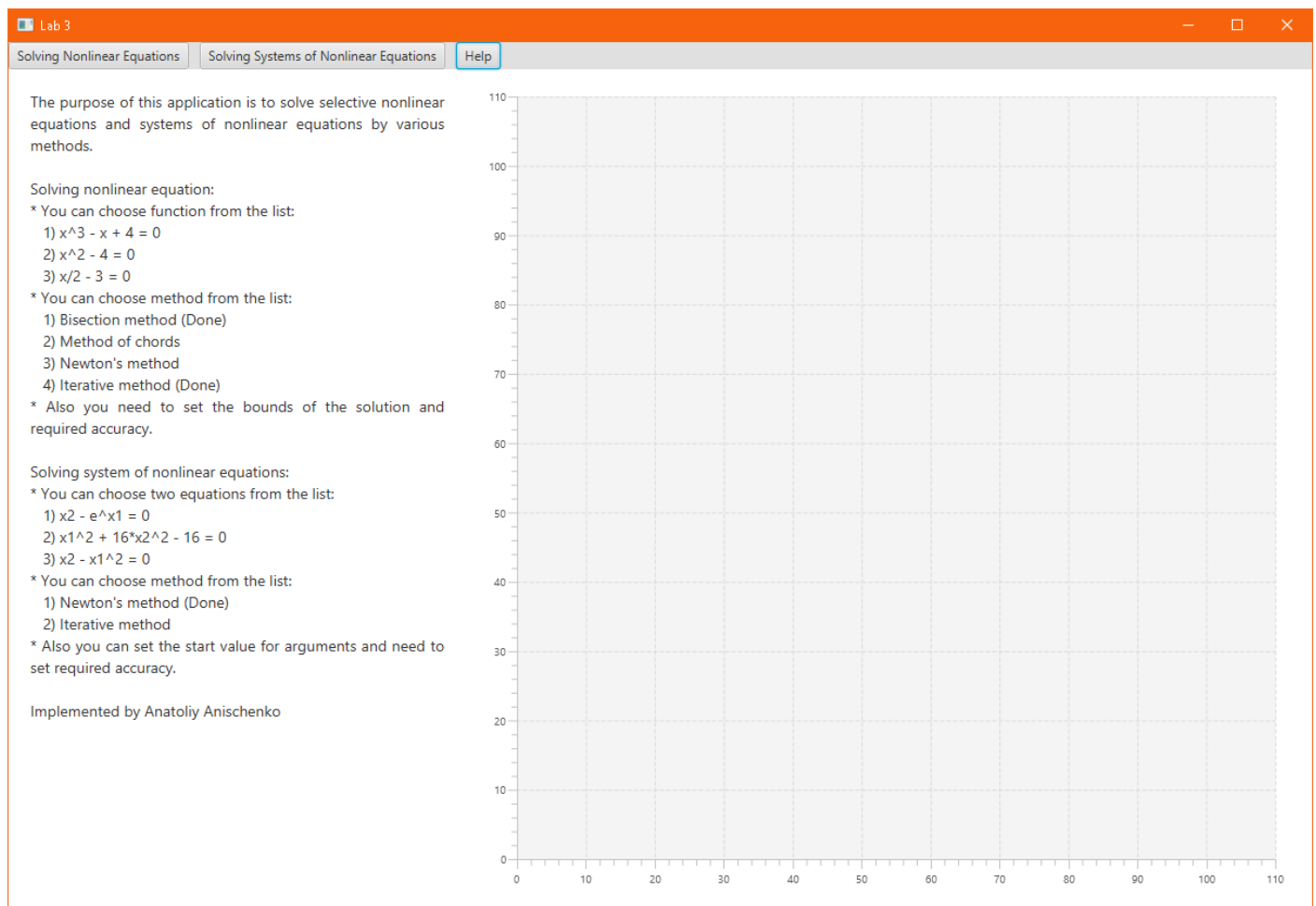
1 static private SystemOfNonlinearEquationsSolutionResult newtonMethodSolution (
2     SystemOfNonlinearEquations system,
3     ArrayList<Double> startValue,
4     double accuracy
5 ) throws NoSolutionException, UnavailableCodeException {
6     ArrayList<Double> x = new ArrayList<>(startValue);
7
8     for (int i = 0; i < N_MAX_VALUE; i++) {
9         ArrayList<Double> prev_x = new ArrayList<>(x);
10
11         SystemOfLinearEquations<Double, LinearEquation> linearSystem = new
12             SystemOfLinearEquations<>();
13
14         ArrayList<ArrayList<NonlinearEquation>> jacobianMatrix =
15             system.getJacobianMatrix();
16         for (int j = 0; j < x.size(); j++) {
17
18         }
19     }
20 }
  
```

```

15         ArrayList<Double> multipliers = new ArrayList<>();
16
17         for (int k = 0; k < x.size(); k++) {
18             multipliers.add(jacobianMatrix.get(j).get(k).getValue(prev_x));
19         }
20
21         LinearEquation curLinearEquation = new LinearEquation(multipliers, -
system.get(j).getValue(prev_x));
22         linearSystem.push(curLinearEquation);
23     }
24
25     ArrayList<Double> solution = new
ArrayList<>(SystemOfLinearEquationsSolver.getSolution(linearSystem));
26
27     for (int j = 0; j < solution.size(); j++) {
28         x.set(j, prev_x.get(j) + solution.get(j));
29     }
30
31     if (getMeasuredError(x, prev_x) < accuracy) {
32         return new SystemOfNonlinearEquationsSolutionResult(x, i);
33     }
34 }
35
36 throw new NoSolutionException("count of iterations more than 10_000_000");
37 }

```

Результат работы:



Method

Bisection method

Function

 $x^3 - x + 4 = 0$

Left bound

-10.0

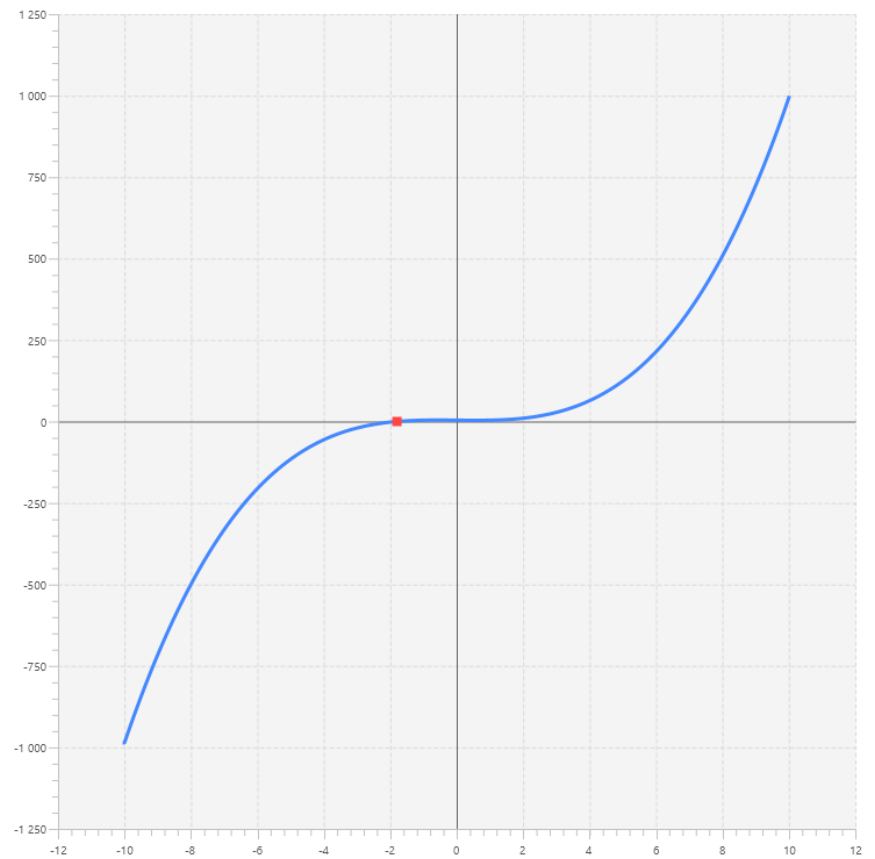
Right bound

10.0

Accuracy

0.01

Calculate



Answer: -1.796875

Function values: -0.004802703857421875

Count of iterations: 7

Method

Iterative method

Function

 $x^3 - x + 4 = 0$

Left bound

-2

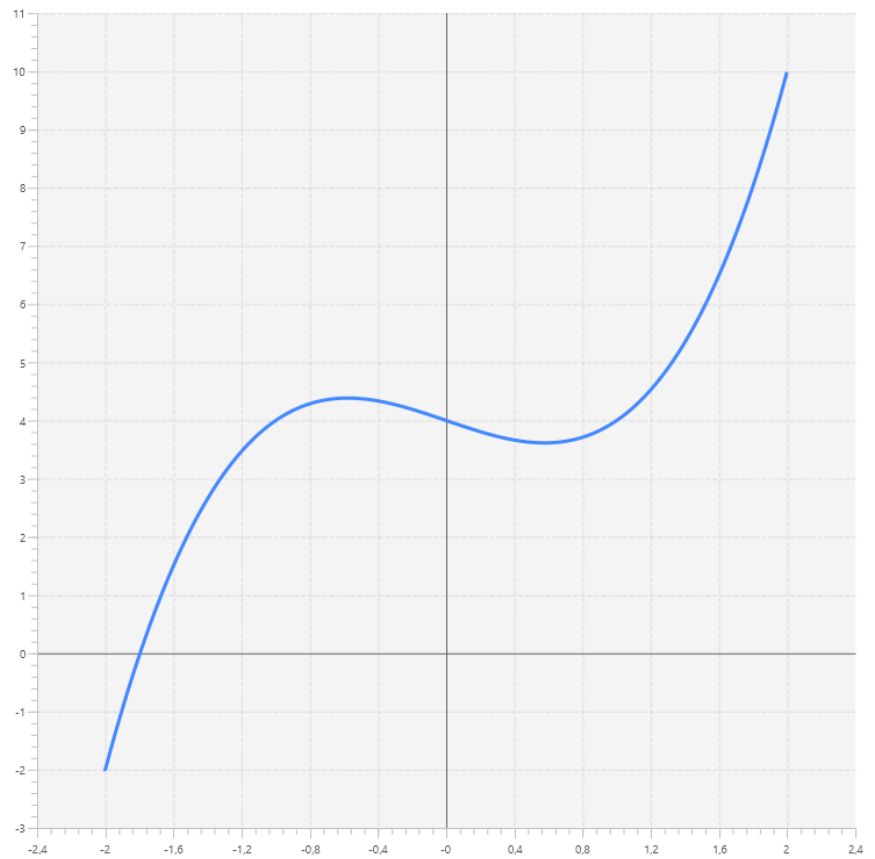
Right bound

2

Accuracy

0.0001

Calculate



Can't solve nonlinear equation at these bounds!

The reason is: count of iterations more than 10_000_000

Method

Iterative method

Function

 $x^3 - x + 4 = 0$

Left bound

-1.8

Right bound

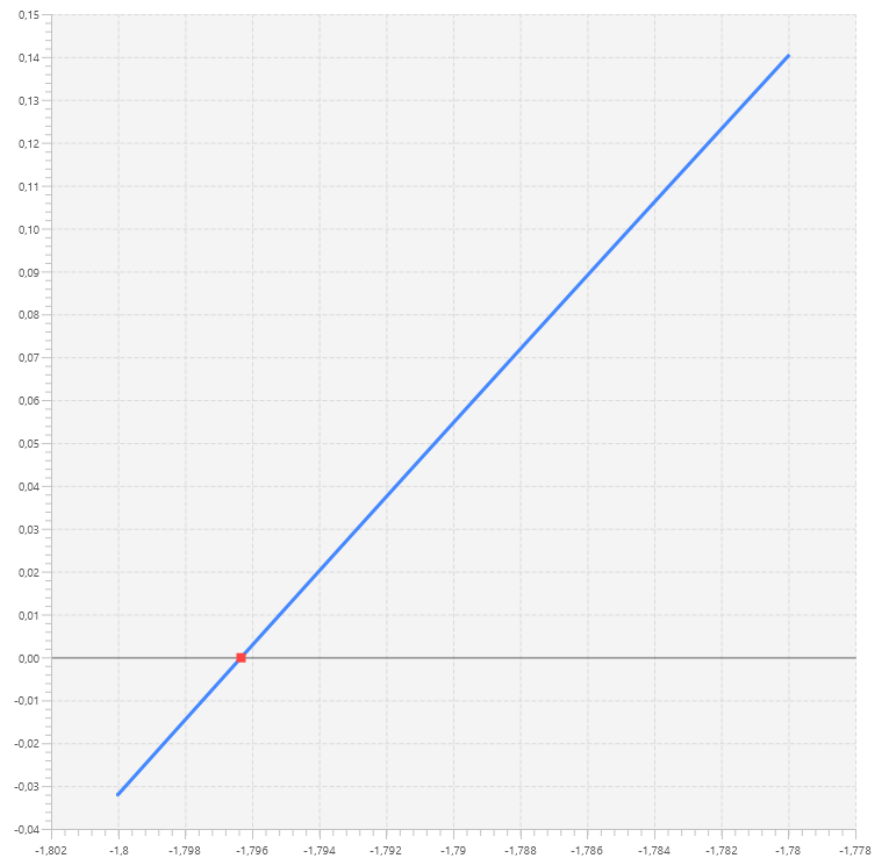
-1.78

Accuracy

0.0001

Calculate

Answer: -1.7963302752293577
Function values: -7.26717316750225E-5
Count of iterations: 0



Method

Bisection method

Function

 $x/2 - 3 = 0$

Left bound

-10.0

Right bound

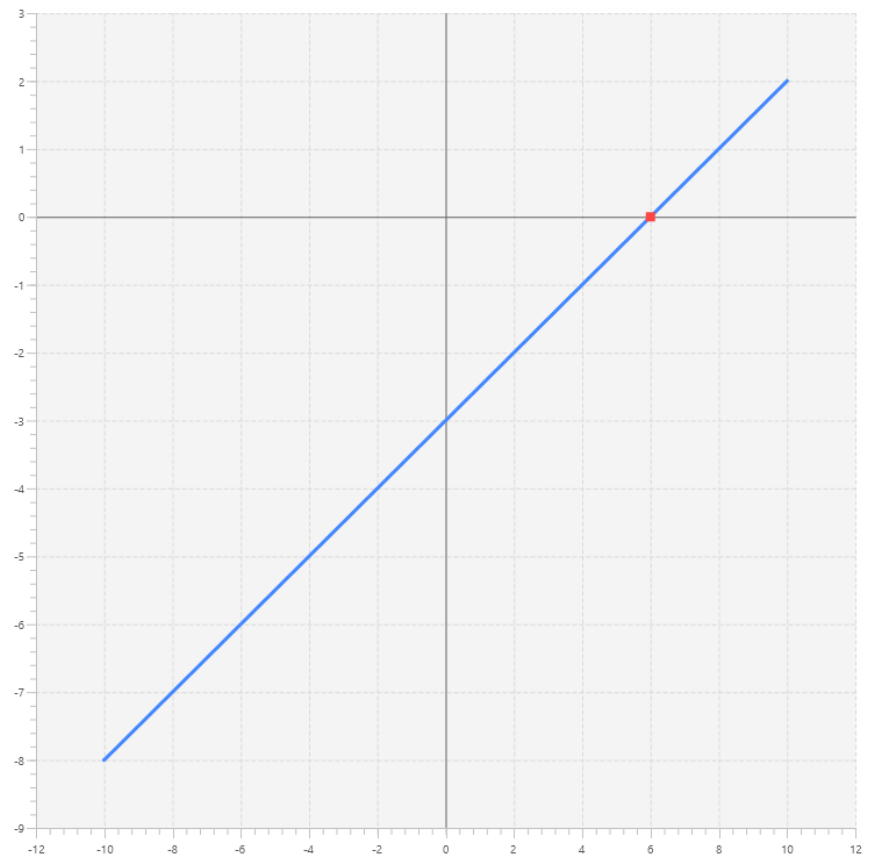
10.0

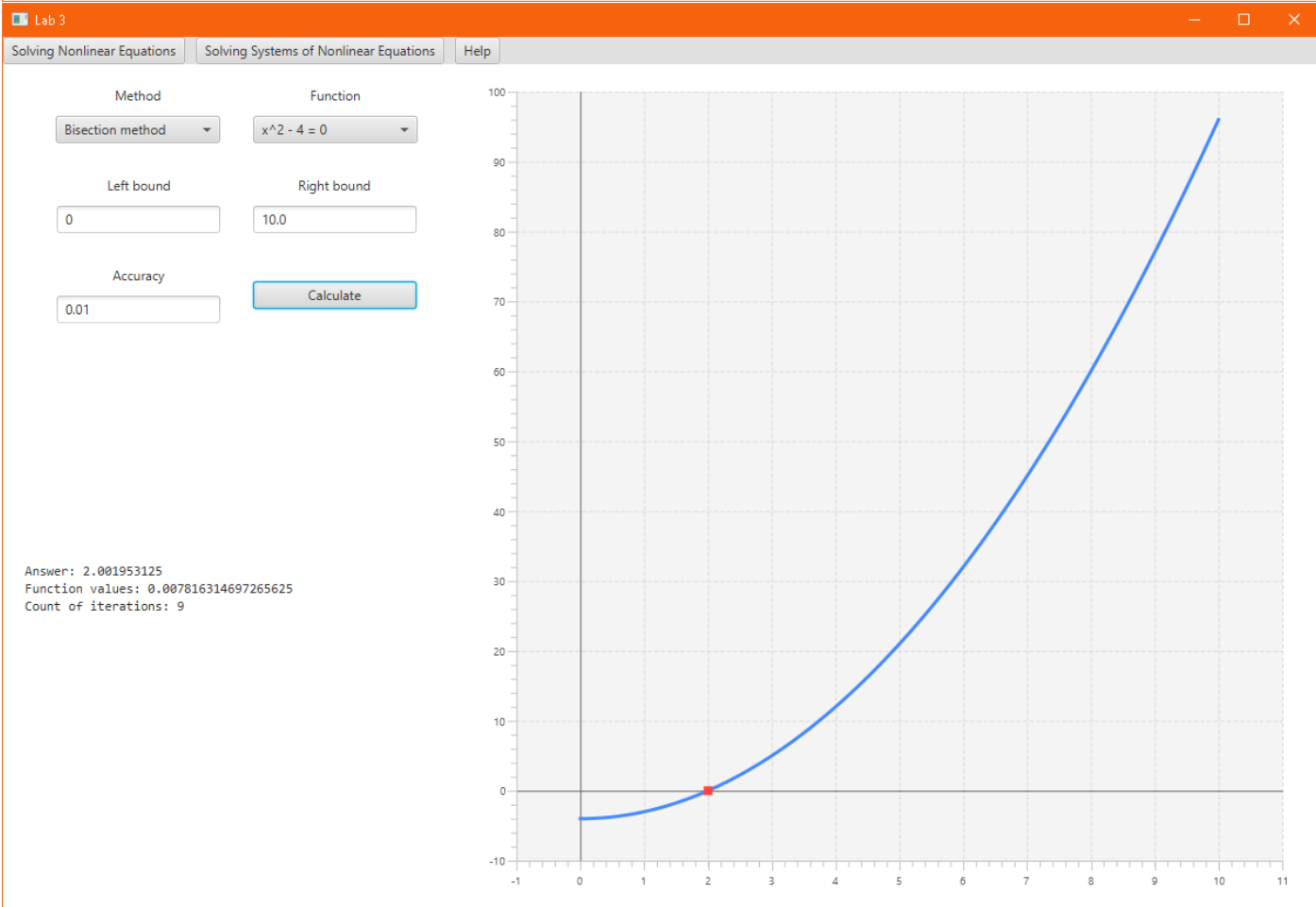
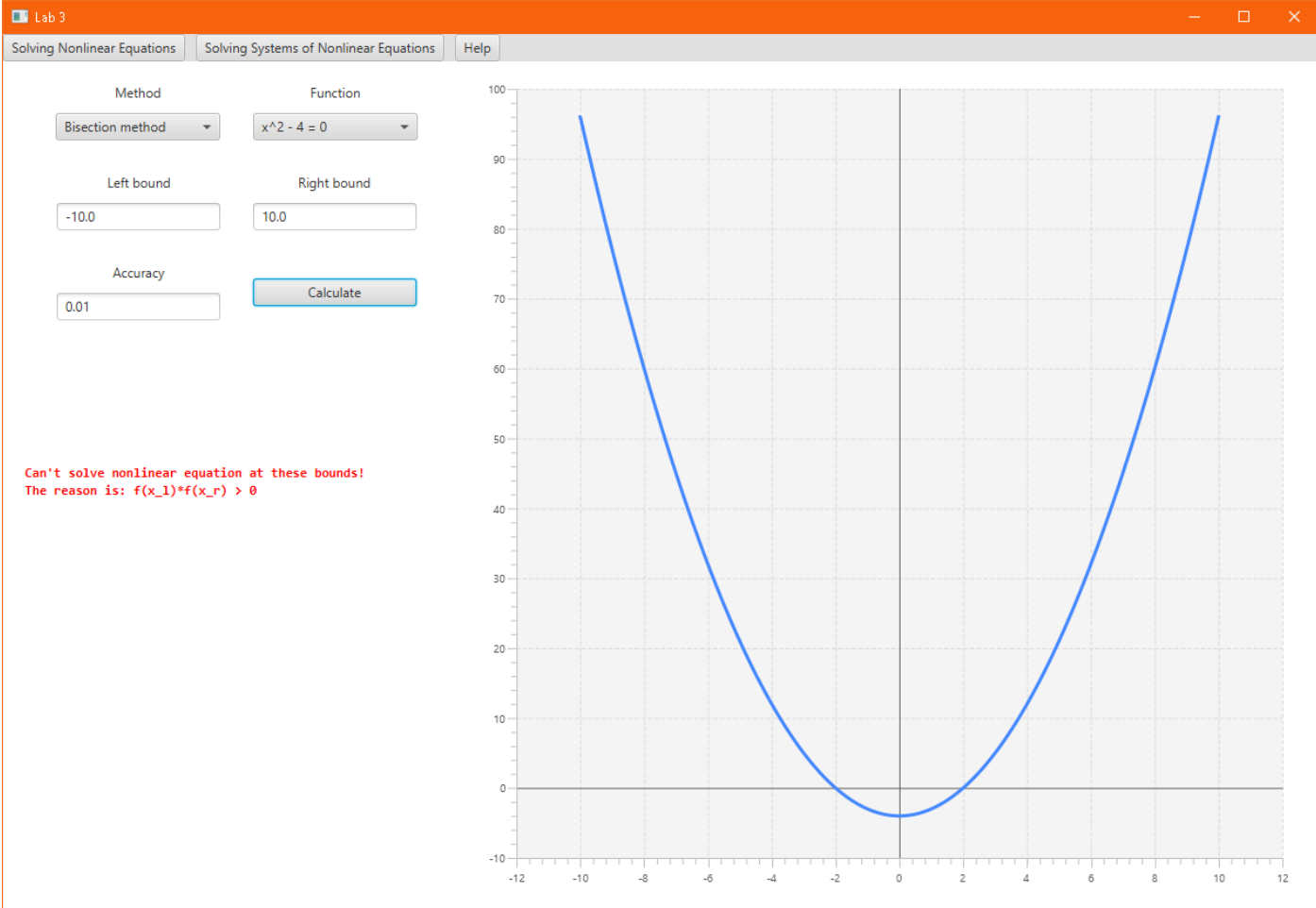
Accuracy

0.00001

Calculate

Answer: 5.9999847412109375
Function values: -7.62939453125E-6
Count of iterations: 17





Method

Newton's method

First equation

 $x_2 - e^{x_1} = 0$

Second equation

 $x_2 - x_1^2 = 0$

x1 start value

0

x2 start value

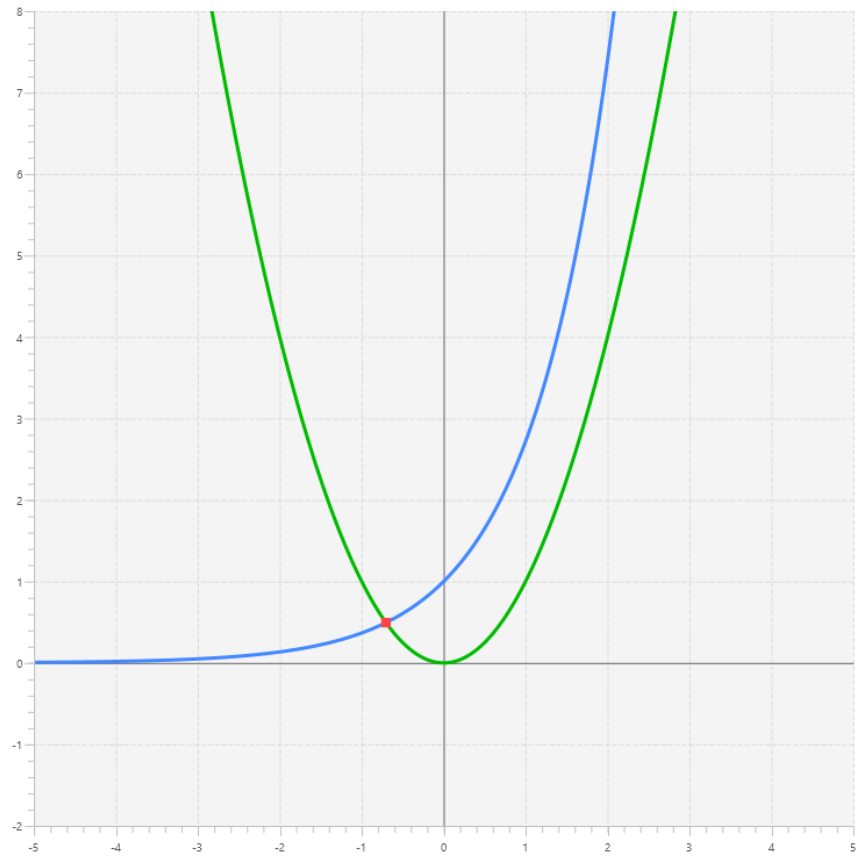
0

Accuracy

0.01

Calculate

Answer: (-0.7034674295409824, 0.4948664066282825)
Count of iterations: 3



Method

Newton's method

First equation

 $x_2 - e^{x_1} = 0$

Second equation

 $x_1^2 + 16 \cdot x_2^2 - 16 = 0$

x1 start value

0

x2 start value

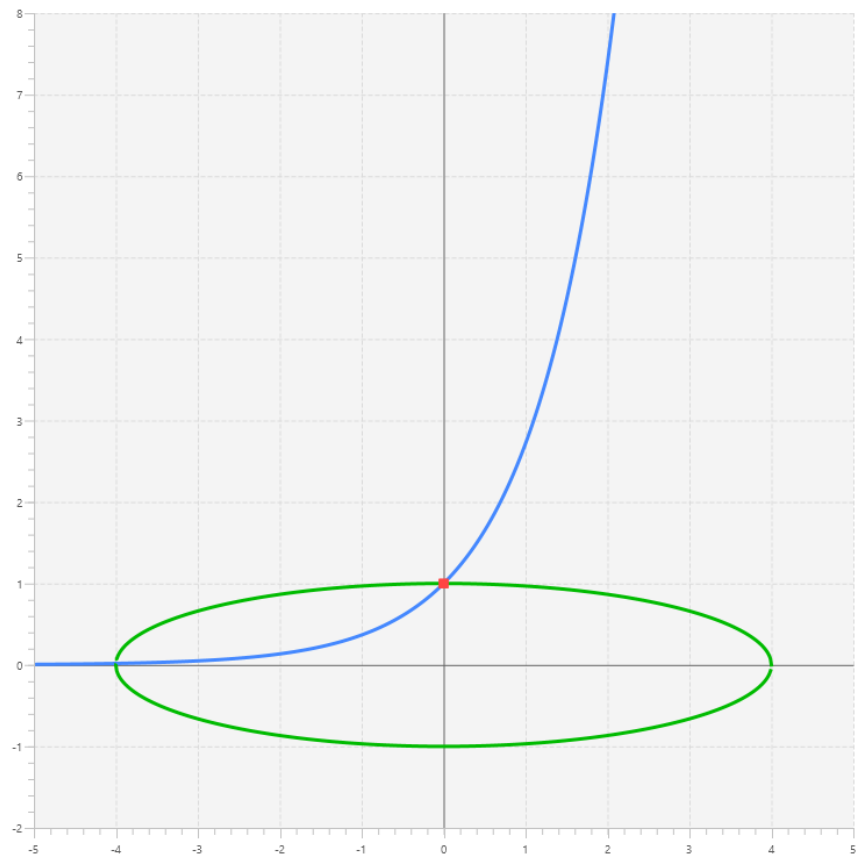
0

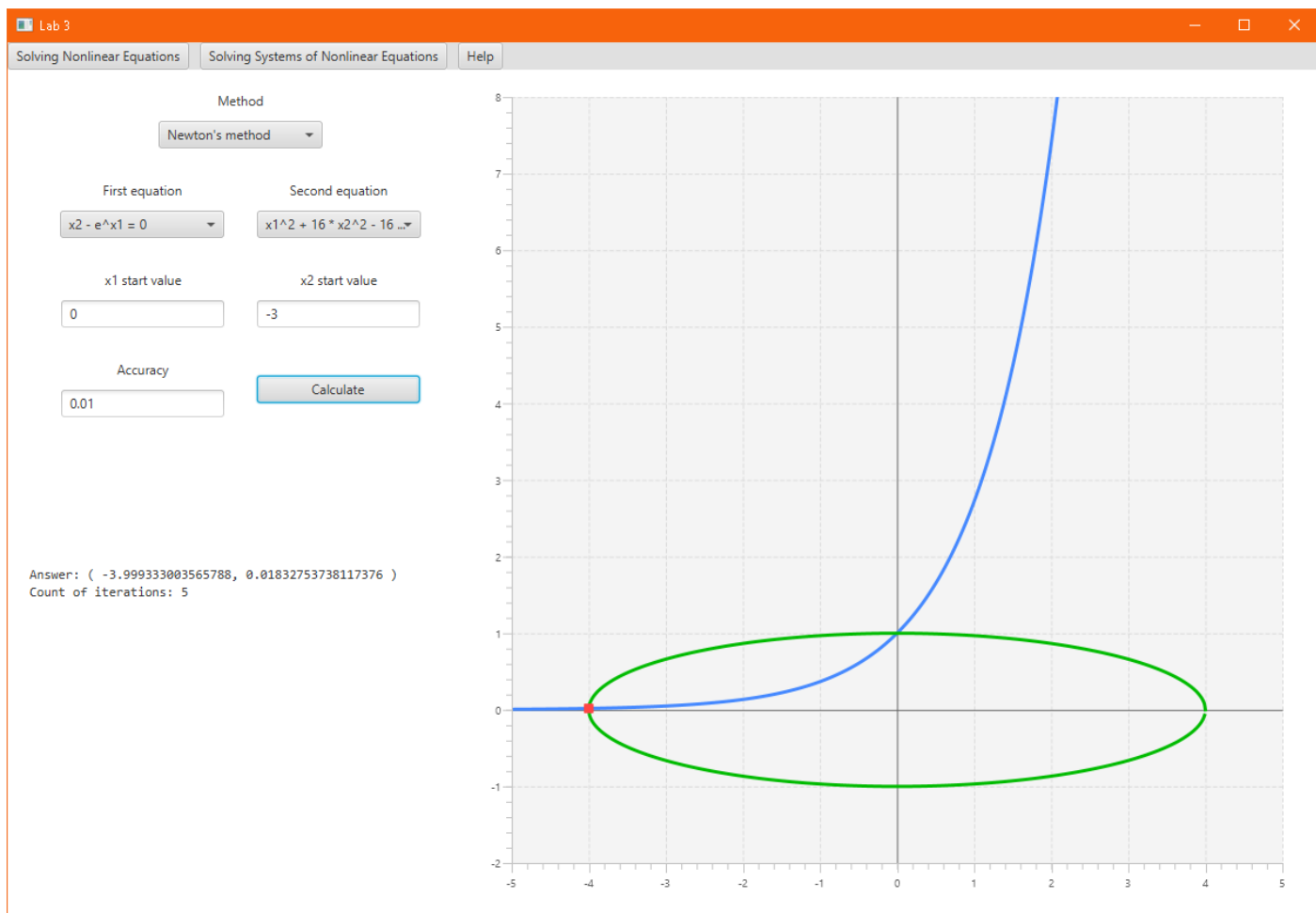
Accuracy

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
 - Требуется подсчёт 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