Университет ИТМО

МФ КТиУ, Ф ПИиКТ

**Лабораторная работа №3**

**Дисциплина «Вычислительная математика»**

**Решение нелинейных уравнений**

**Вариант:**

аг1

**Выполнил:**

Студент группы P3212

Анищенко Анатолий Алексеевич

**Преподаватель:**

Перл Ольга Вячеславовна

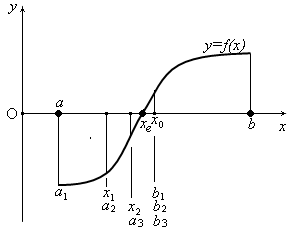
г. Санкт-Петербург

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:

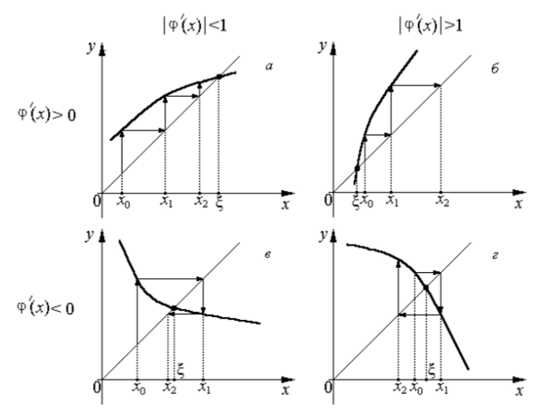
Для нахождения корня уравнения. Будем последовательно сужать отрезок поиска, и когда заданная точность будет достигнута результат будет найден.

Метод половинного деления всегда сходится. Скорость сходимости линейна.

Критерий окончания итерационного процесса: или .

Приближенное значение корня: или или

**Метод простой итерации:**

Суть метода заключается в том, что уравнение с помощью некоторых преобразований необходимо переписать в виде .

Рабочая формула метода: .

Уравнение эквивалентно уравнению для любой .

Для нахождения корня уравнения выберем некоторое начальное значение , которое должно находиться как можно ближе к корню уравнения. Дальше с помощью итерационной формулы будем находить каждое следующее приближение корня уравнения.

Условия сходимости метода простой итерации определяются теоремой:

Если в некоторой корня уравнения функция дифференцируема и удовлетворяет неравенству , где постоянная, то независимо от выбора начального приближения из указанной итерационная последовательность не выходит из этой окрестности, метод сходится со скоростью геометрической прогрессии.

Достаточное условие сходимости метода: , где – некоторая константа.

Критерий окончания итерационного процесса: 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 в качестве начального приближения решения, мы можем построить линейную аппроксимацию в окрестности и решить получающееся линейное уравнение .

, где

матрица Якоби

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

где поправка к текущему приближению

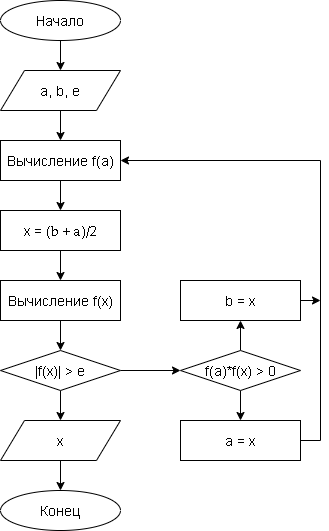
В результате получена система линейных алгебраических уравнений относительно поправки . После ее определения вычисляется следующее приближение

Существует теорема о достаточных условиях сходимости метода Ньютона.

**Листинг:**

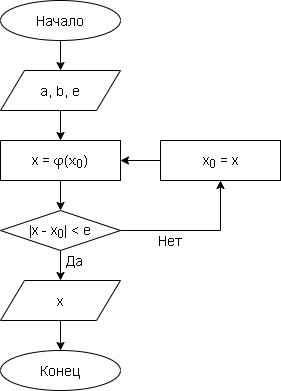
**Метод деления пополам:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | **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:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | **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**  **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");  } |

****

**Изображение выглядит как текст, карта

Автоматически созданное описаниеМетод Ньютона для СНАУ:**

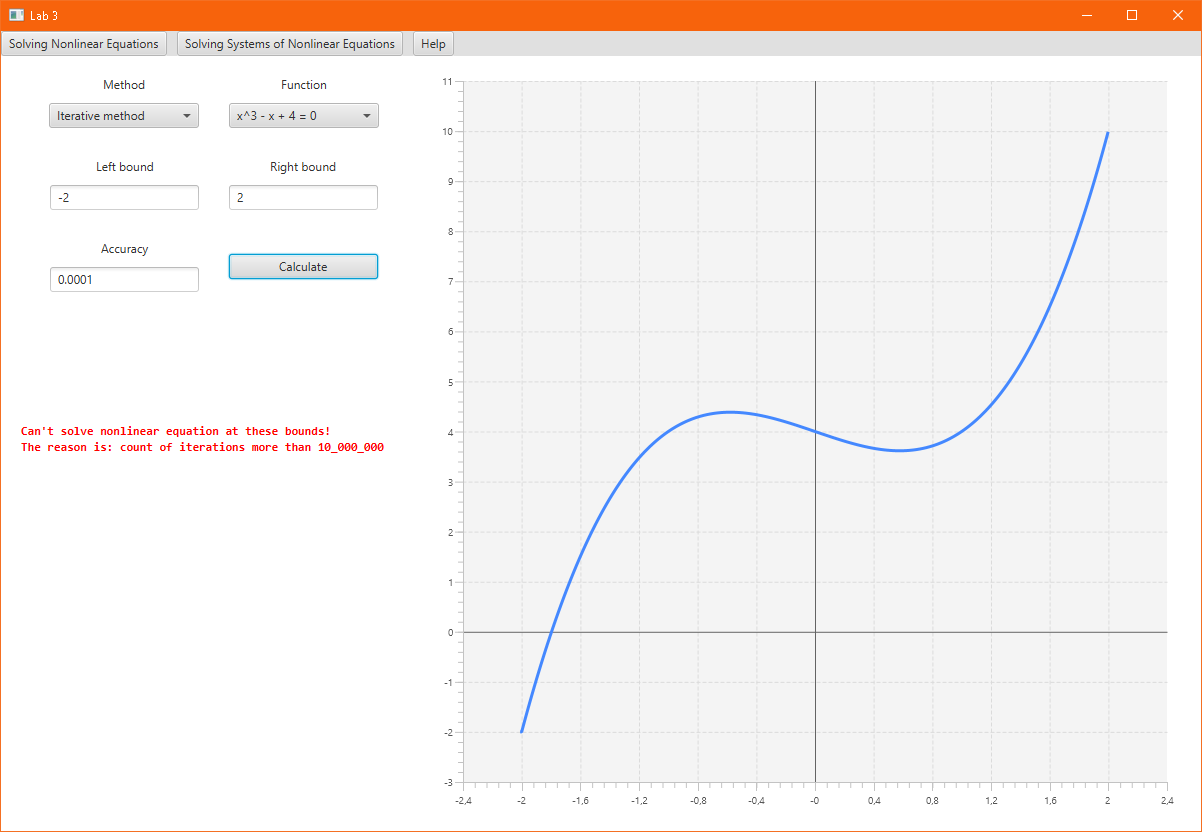
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | **static** **private** SystemOfNonlinearEquationsSolutionResult **newtonMethodSolution**(  SystemOfNonlinearEquations system,  ArrayList<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:**

Изображение выглядит как снимок экрана, компьютер

Автоматически созданное описание

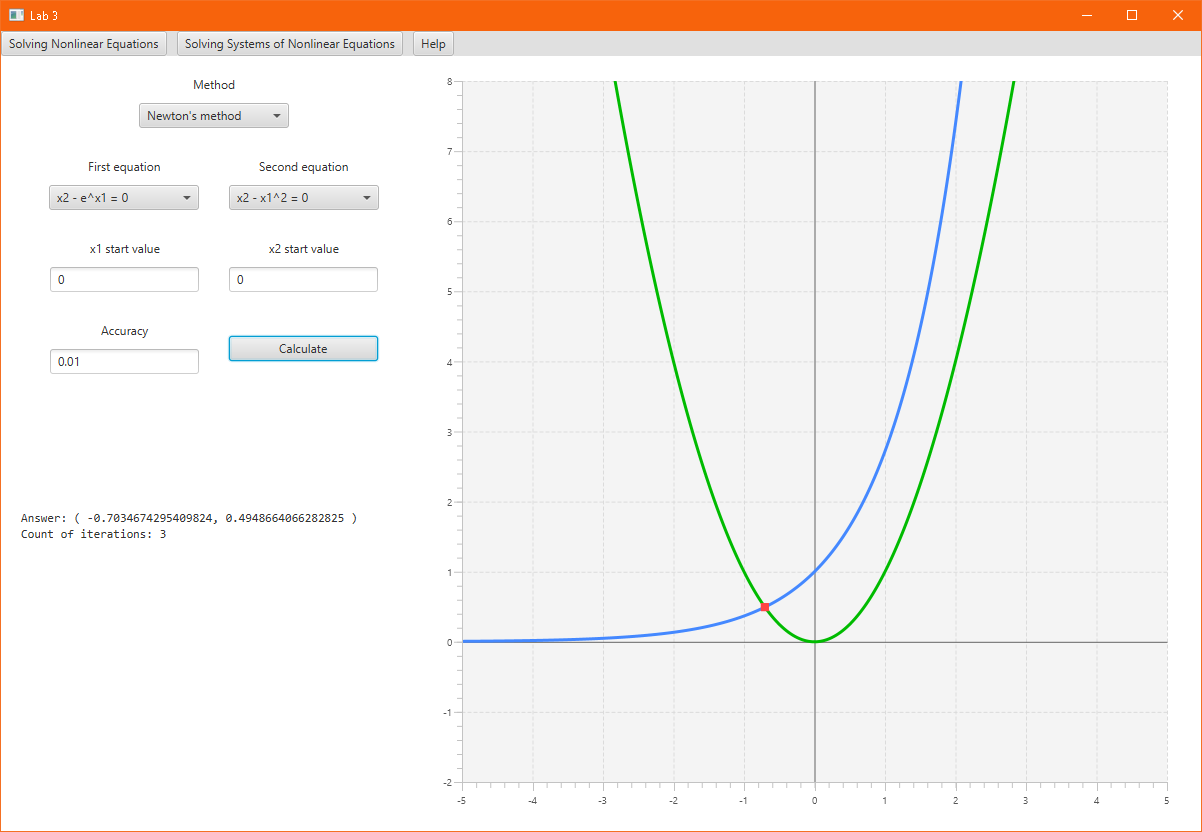
Изображение выглядит как текст, карта

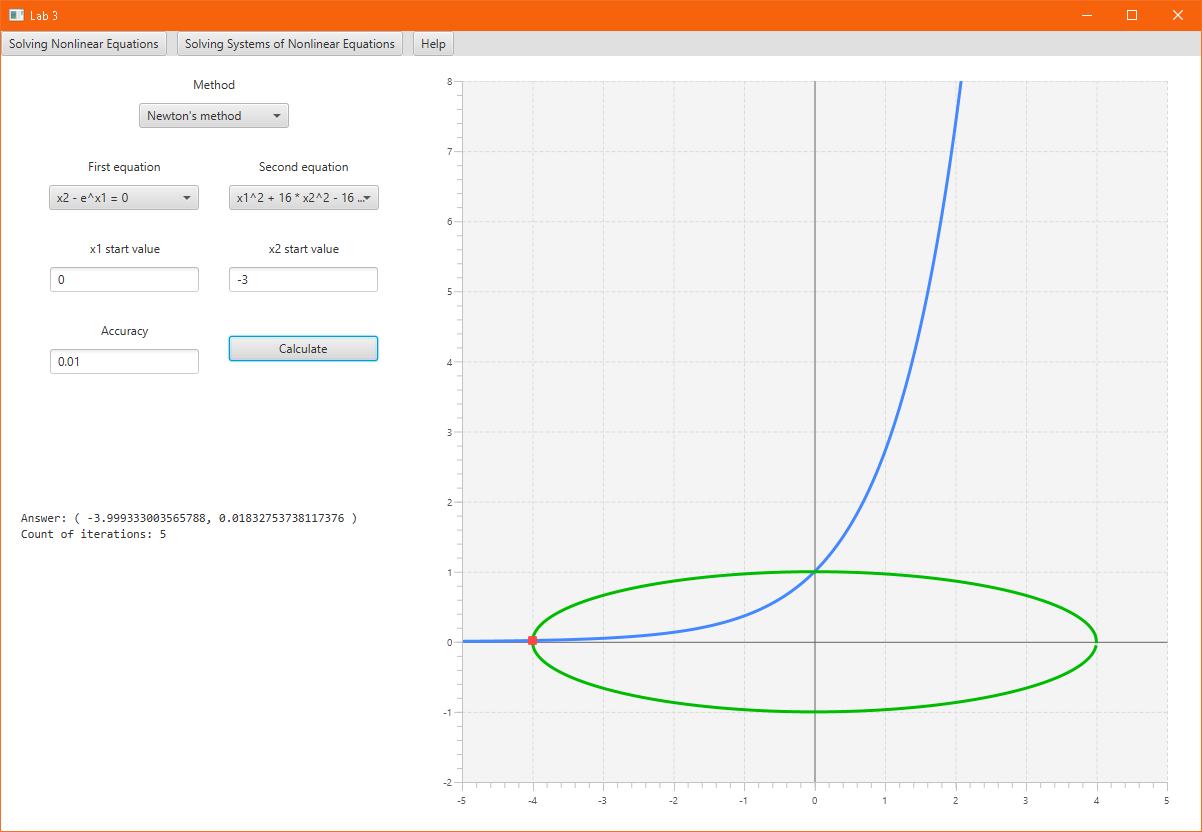
Автоматически созданное описаниеИзображение выглядит как карта, компьютер, стол, мужчина

Автоматически созданное описаниеИзображение выглядит как стол, компьютер, мужчина, дисплей

Автоматически созданное описаниеИзображение выглядит как стол, компьютер, белый, мужчина

Автоматически созданное описаниеИзображение выглядит как мужчина

Автоматически созданное описаниеИзображение выглядит как мужчина

Автоматически созданное описание

**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