Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа **№2**

**«Численное решение нелинейных уравнений и систем»**

по дисциплине «Вычислительная математика**»**

Вариант: **4**

**Выполнил:**

Кирячек Тимофей

Группа: Р3209

**Преподаватель:**

Наумова Надежда Александровна

Санкт-Петербург, 2025 г.

Цель работы: изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

# 1. Вычислительная реализация задачи

# 1. Решение нелинейного уравнения

Изображение выглядит как линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Для определения интервалов изоляции корней данного уравнения можно воспользоваться методом интервалов знакопеременности. Для этого нужно найти значения функции на различных интервалах и определить знак функции на каждом из них.  
  
Из графика найдем приближенные значения корней:  
x ≈ -1.2, x ≈ 0.6, x ≈ 2.4

Теперь нужно разбить ось x на 4 интервала: (-∞, -1.2), (-1.2, 0.6), (0.6, 2.4) и (2.4, +∞). На каждом из этих интервалов нужно определить знак функции.

Для этого можем вычислить значения функции в произвольной точке каждого интервала. Например, для интервала (-∞, -1.2) можно выбрать x = -2, для интервала (-1.2, 0.6) x = 0, для интервала (0.6, 2.4) x = 2, и для интервала (2.4, +∞) x = 4.

Таким образом, получим следующие значения функции:

для x = -2: f(-2) = -9.8

для x = 0: f(0) = 1.76

для x = 2: f(2) = -1.8

для x = 4: f(4) = 27.52

Знаки функции на каждом интервале будут соответственно:

|  |  |  |  |
| --- | --- | --- | --- |
| (-∞, -1.2) | (-1.2, 0.6) | (0.6, 2.4) | (2.4, +∞) |
| - | + | - | + |

Таким образом, мы получаем два интервала изоляции корней уравнения:

(-1.5, 0), (0, 2) и (2, 3).

x1 ≈

x2 ≈

x3 ≈

Крайний правый корень – **Метод простой итерации**

Проверка **условия сходимости** метода на выбранном интервале:

На отрезке начального приближения [2, 3] функция определена, непрерывна и дифференцируема.

Изображение выглядит как линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как линия, График, диаграмма, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

**итерационная последовательность сходится,** скорость сходимости низкая, критерий окончания итерационного процесса , x0 = 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | xk | xk+1 | f(xk+1) | │ xk+1- xk│ |
| 1 | 2 | 2.1318 | -1.40742 | 0.1318 |
| 2 | 2.1318 | 2.23463 | -0.98833 | 0.10283 |
| 3 | 2.23463 | 2.30698 | -0.63473 | 0,07235 |
| 4 | 2.30698 | 2.35345 | -0.37998 | 0,04647 |
| 5 | 2.35345 | 2.38127 | -0.21682 | 0,02782 |
| 6 | 2.38127 | 2.39714 | -0.12011 | 0,01587 |
| 7 | 2.39714 | 2.40593 | -0.06541 | 0,00879 |
| 8 | 2.40593 | 2.41072 | -0.03524 | 0,00479 |
| 9 | 2.41072 | **2.41330** | -0.01889 | 0,00258 |

Центральный корень – **Метод секущих**

Изображение выглядит как линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.Выберем , для этого построим график на интервале (0, 2):

Как видно из графика, на интервале нет ни одной точки, в которой обеспечивалась бы быстрая сходимость. Поэтому можно взять, например левую границу интервала

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | xk-1 | xk | xk+1 | f(xk+1) | │ xk+1- xk│ |
| 1 | 0 | 0.01 | 0.87181 | -0.75748 | 0.86181 |
| 2 | 0.01 | 0.87181 | 0.61040 | 0.062433 | 0.26141 |
| 3 | 0.87181 | 0.61040 | 0.63030 | -0.001069 | 0.01991 |
| 4 | 0.61040 | 0.63030 | **0.62997** | 0.0000013 | 0.00033 |

Крайний левый корень – **Метод половинного деления**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | a | b | x | f(a) | f(b) | f(x) | |a – b| |
| 1 | -1.500 | 0.000 | -0.750 | -2.8675 | 1.75 | 1.775 | 1.500 |
| 2 | -1.500 | -0.750 | -1.125 | -2.8675 | 1.775 | 0.19414 | 0.750 |
| 3 | -1.500 | -1.125 | -1.3125 | -2.8675 | 0.19414 | -1.13181 | 0.375 |
| 4 | -1.3125 | -1.125 | -1.21875 | -1.1318 | 0.19414 | -0.42009 | 0.18750 |
| 5 | -1.21875 | -1.125 | -1.17188 | -0.42009 | 0.19414 | -0.10113 | 0,09375 |
| 6 | -1.17188 | -1.125 | -1.14844 | -0.10113 | 0.19414 | 0.049438 | 0,04688 |
| 7 | -1.17188 | -1.14844 | -1.16016 | -0.10113 | 0.04944 | -0.02511 | 0,02344 |
| 8 | -1.16016 | -1.14844 | -1.1543 | -0.02511 | 0.04944 | 0.012349 | -0.01172 |
| 9 | -1.16016 | -1.1543 | **-1.15723** | -0.02511 | 0.01235 | -0.00633 | -0.00586 |

# 2. Решение системы нелинейных уравнений

1. , Метод Ньютона

Изображение выглядит как линия, диаграмма, График

Контент, сгенерированный ИИ, может содержать ошибки.

Отметим, что решение системы уравнений являются точки пересечения эллипса и , следовательно, система имеет не более 2 различных решений.

Построим матрицу Якоби:

*, ,*

**Корень 1:** Шаг 1: Выбираем

Шаг 2. Решаем полученную систему.

Шаг 3. Вычисляем очередные приближения:

,

, , возвращаемся к шагу 1:

,

,

Таким образом, корень 1 найден: (0.5991, 0.5662)

Аналогично находим **другой корень**:

# 2. Программная реализация задачи

**Листинг программы**

<https://github.com/TecHeReTiC3141/CalcMaths_Lab2.git>

utils.ts

const derivative = (x: number, equation: Equation) => (equation(x + 1e-6) - equation(x)) / 1e-6;  
  
const secondDerivative = (x: number, equation: Equation) =>  
 (equation(x + 1e-6) - 2 \* equation(x) + equation(x - 1e-6)) / (1e-6 \*\* 2);  
  
const MAX\_ITERS = 2000  
  
const chordMethod: MethodFunction<SecantMethodIter> = (equation, a, b, tolerance) => {  
 const fA = equation(a);  
 const fSecondDerivA = secondDerivative(a, equation);  
 const isLeftFixed = fA \* fSecondDerivA > 0;  
  
 let x0 = isLeftFixed ? b : a;  
 let x1 = Infinity;  
 const iters: SecantMethodIter[] = []  
 for (let i = 0; i < MAX\_ITERS; ++i) {  
 x1 = x0 - equation(x0) \* ((isLeftFixed ? a : b) - x0) / (equation(isLeftFixed ? a : b) - equation(x0))  
 iters.push({  
 b,  
 fB: equation(b),  
 xk: x0,  
 fXk: equation(x0),  
 xNext: x1,  
 diff: Math.abs(x1 - x0)  
 })  
 if (Math.abs(equation(x0)) <= tolerance) break;  
 x0 = x1  
 }  
 return { iters, ans: x1, a, b, equation };  
};  
  
// Метод Ньютона  
const newtonMethod: MethodFunction<NewtonMethodIter> = (equation, a, b, tolerance) => {  
 const fA = equation(a);  
 const fSecondDerivA = secondDerivative(a, equation);  
 const isLeftSuits = fA \* fSecondDerivA > 0;  
  
 const x = isLeftSuits ? a : b;  
  
 const calc = (x: number) => {  
 let curIter = 0  
 const iters: NewtonMethodIter[] = []  
 let temp  
 while (curIter++ < MAX\_ITERS && Math.abs(equation(x)) > tolerance) {  
 temp = x - equation(x) / derivative(x, equation)  
 iters.push({  
 xk: x,  
 fXk: equation(x),  
 derivXk: derivative(x, equation),  
 xNext: temp,  
 diff: Math.abs(temp - x),  
 })  
 x = temp;  
 }  
 return { iters, ans: x, a, b, equation };  
 }  
  
 const firstAns = calc(x)  
 if (firstAns.ans >= a && firstAns.ans <= b) {  
 return firstAns  
 }  
  
 const secondAns = calc(isLeftSuits ? b : a)  
 if (secondAns.ans >= a && secondAns.ans <= b) {  
 return secondAns  
 }  
  
 return calc((a + b) / 2)  
};  
  
  
const iterationMethod: MethodFunction<IterationMethodIter> = (equation, a, b, tolerance) => {  
 const maxDerivative = Math.max(...Array.from({ length: SEGMENTS }, (\_, i) =>  
 Math.abs(derivative(a + (i / (SEGMENTS - 1)) \* (b - a), equation))  
 ));  
  
 let curIter = 0;  
  
 const iters: IterationMethodIter[] = []  
 let lambda = Math.abs(1 / maxDerivative);  
  
 const isDerivPositive = Array.from({ length: SEGMENTS }, (\_, i) =>  
 derivative(a + (i / (SEGMENTS - 1)) \* (b - a), equation)  
 ).every((n) => n > 0)  
  
 if (isDerivPositive) {  
 lambda \*= -1  
 }  
 const phi = (x: number) => x + lambda \* equation(x);  
 let x = a;  
 let xPrev = Infinity;  
 while (curIter++ < MAX\_ITERS && Math.abs(x - xPrev) > tolerance) {  
 xPrev = x;  
 x = phi(x);  
 iters.push({  
 xk: xPrev,  
 xNext: x,  
 fXk: equation(xPrev),  
 diff: Math.abs(x - xPrev)  
 })  
 }  
 if (isNaN(x)) return ValidationError.noConvenge  
 return { iters, ans: x, a, b, equation };  
};  
  
const hasSingleRoot = (equation: Equation, a: number, b: number): boolean => {  
 const values = Array.from({ length: SEGMENTS }, (\_, i) => equation(a + (i / (SEGMENTS - 1)) \* (b - a)));  
 const signChanges = values.slice(1).filter((val, i) => val \* values[ i ] < 0 || values[i] === 0).length;  
 return signChanges === 1;  
};  
  
const methods: Record<EquationSolvingMethod, MethodFunction<SecantMethodIter | NewtonMethodIter | IterationMethodIter>> = {  
 [ EquationSolvingMethod.Chord ]: chordMethod,  
 [ EquationSolvingMethod.Newton ]: newtonMethod,  
 [ EquationSolvingMethod.Iteration ]: iterationMethod  
};  
  
  
export const solveEquation = (method: EquationSolvingMethod, equation: Equation, a: number, b: number, accuracy: number): SolutionData => {  
 if (!hasSingleRoot(equation, a, b)) {  
 return ValidationError.notSingleRoot  
 }  
 return methods[ method ](equation, a, b, accuracy);  
}  
  
export const solveSystem: SystemSolutionFunction = (system: SystemEquationOption, initX: number, initY: number, accuracy: number) => {  
 const { phi1, phi2 } = system;  
 const iters: NewtonSystemIter[] = [];  
 let x = initX;  
 let y = initY;  
 let xPrev = Infinity;  
 let yPrev = Infinity;  
 let curIter = 0;  
 while (curIter++ < MAX\_ITERS && Math.sqrt((x - xPrev) \*\* 2 + (y - yPrev) \*\* 2) > accuracy) {  
 xPrev = x;  
 yPrev = y;  
 x = phi1(xPrev, yPrev);  
 y = phi2(xPrev, yPrev);  
 iters.push({  
 xK: `(${xPrev.toFixed(4)}, ${yPrev.toFixed(4)})`,  
 xNext: `(${x.toFixed(4)}, ${y.toFixed(4)})`,  
 diff: Math.sqrt((x - xPrev) \*\* 2 + (y - yPrev) \*\* 2)  
 });  
 }  
 if (isNaN(x) || isNaN(y)) return ValidationError.noConvenge  
 return { iters, equations: system.equations, ans: [x, y] };  
}

**Пример работы программы:**

|  |
| --- |
| Уравнение: Выбираем уравнение из возможных вариантов  Вводим границы интервала в соответствующие поля: 2, 3  Выбираем способ решения – метод Ньютона  Указываем желаемую точность в соответствующее поле  Нажимаем на кнопку Найти решение  Результат:  Система уравнений: Выбираем систему из возможных вариантов  Вводим начальное приближение начальное приближение в соответствующие поля:  Вводим желаемую точность  Нажимаем на кнопку Найти решение |

# Вывод

В ходе выполнения лабораторной работы были изучены численные методы решения нелинейных уравнений и систем нелинейных уравнений и разработано веб-приложения для их решения. В результате работы были найдены корни заданных уравнений и систем с использованием различных численных методов, а также были построены графики функций для полного представления исследуемых интервалов.