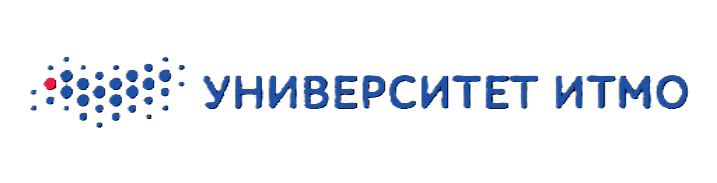
Университет ИТМО

Мегафакультет компьютерных технологий и управления

Факультет программной инженерии и компьютерной техники



ЛАБОРАТОРНАЯ РАБОТА №2

ЧИСЛЕННОЕ РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ И СИСТЕМ

Вариант №10

Группа: P3211

Студент: Орчиков Даниил Валерьевич

Преподаватель: Малышева Татьяна Алексеевна

г. Санкт-Петербург

2022

Оглавление

[Цель работы 2](#_Toc161070016)

[Вычислительная реализация задачи: 2](#_Toc161070017)

[1 часть. Решение нелинейного уравнения 2](#_Toc161070018)

[Рабочие формулы 2](#_Toc161070019)

[Решение 3](#_Toc161070020)

[2 часть. Решение системы нелинейных уравнений 4](#_Toc161070021)

[Рабочие формулы 4](#_Toc161070022)

[Решение 4](#_Toc161070023)

[Программная реализация задачи 5](#_Toc161070024)

[Рабочие формулы 5](#_Toc161070025)

[Листинг программы 6](#_Toc161070026)

[Примеры и результаты работы программы 10](#_Toc161070027)

[Пример 1 10](#_Toc161070028)

[Пример 2 10](#_Toc161070029)

[Пример 3 11](#_Toc161070030)

[Вывод 11](#_Toc161070031)

Цель работы

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов

# Вычислительная реализация задачи:

## 1 часть. Решение нелинейного уравнения

### Рабочие формулы

Метод Ньютона:

Метод половинного деления:

Метод простой итерации:

### Решение

1. Графическое отделение корней  
   Изображение выглядит как линия, График, диаграмма, Параллельный

   Автоматически созданное описание
2. Интервалы изоляции корней:  
   (-2.5; -1), (0; 1), (3.5; 4)
3. Методы, используемые для уточнения корней:
   1. Метод Ньютона
   2. Метод половинного деления
   3. Метод простой итерации
4. Первый корень (метод Ньютона):

Интервал изоляции - (-2.5; -1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |  |
| 1 | -2.5 | -23.948 | -30.875 | -1.724 | 0.776 |
| 2 | -1.724 | 5.926 | -16.197 | -1.359 | 0.365 |
| 3 | -1.359 | 1.062 | -10.527 | -1.258 | 0.101 |
| 4 | -1.258 | 0.072 | -9.105 | -1.25 | 0.008 |

1. Второй корень (метод половинного деления):

Интервал изоляции - (0; 1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № итерации | *a* | *b* | *x* | *f(a)* | *f(b)* | *f(x)* | *|a-b|* |
| 1 | 0.000 | 1.000 | 0.500 | 2.458 | -3.167 | 0.052 | 1.000 |
| 2 | 0.500 | 1.000 | 0.750 | 0.052 | -3.167 | -1.503 | 0.500 |
| 3 | 0.500 | 0.750 | 0.625 | 0.052 | -1.503 | -0.706 | 0.250 |
| 4 | 0.500 | 0.625 | 0.563 | 0.052 | -0.706 | -0.322 | 0.125 |
| 5 | 0.500 | 0.563 | 0.531 | 0.052 | -0.322 | -0.133 | 0.063 |
| 6 | 0.500 | 0.531 | 0.516 | 0.052 | -0.133 | -0.040 | 0.031 |
| 7 | 0.500 | 0.516 | 0.508 | 0.052 | -0.040 | 0.006 | 0.016 |
| 8 | 0.508 | 0.516 | 0.512 | 0.006 | -0.040 | -0.017 | 0.008 |

1. Третий корень (метод простой итерации):

Интервал изоляции - (3.5; 4)

Применим 3 способ преобразования уравнения:

Преобразованное уравнение:

Условие сходимости выполняется

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |
| 1 | 3.550 | 3.795 | -1.172 | 0.245 |
| 2 | 3.795 | 3.866 | 0.006 | 0.071 |
| 3 | 3.886 | 3.878 | 0.202 | 0.011 |
| 4 | 3.878 | 3.879 | 0.226 | 0.001 |

## 2 часть. Решение системы нелинейных уравнений

### Рабочие формулы

Метод простой итерации

### Решение

1. Графическое отделение корней  
   Изображение выглядит как линия, диаграмма, График, текст

   Автоматически созданное описание
2. Нахождение корня методом простых итераций:

Решение находится в области

Проверим условие сходимости. В области G имеем:

Начальные приближение

1 шаг

2 шаг

3 шаг

4 шаг

5 шаг

6 шаг

7 шаг

8 шаг

9 шаг

10 шаг

# Программная реализация задачи

## Рабочие формулы

## Листинг программы

Представлен только код, непосредственно выполняющий вычисления

Весь код можно посмотреть [тут (GitHub)](https://github.com/DaniilOrchikov/Computational_mathematics/tree/master/laba2)

function chordMethod(a, b, accuracy) {  
 function runA() {  
 let v = 1  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>a</th><th>b</th><th>x</th><th>f(a)</th><th>f(b)</th><th>f(x)</th><th>|x\_k - x\_{k+1}|</th></tr>"  
 let x0 = a  
 while (true) {  
 let fx0 = *f*["function"](x0)  
 let fb = *f*["function"](b)  
 x = x0 - (b - x0) / (fb - fx0) \* fx0  
 let fx = *f*["function"](x)  
 *document*.getElementById("res").innerHTML += `<tr><td>${v}</td><td>${x0}</td><td>${b}</td><td>${x}</td><td>${fx0}</td><td>${fb}</td><td>${fx}</td><td>${*Math*.abs(x0 - x)}</td></tr>`  
 if (*Math*.abs(x0 - x) < accuracy) return  
 v++  
 x0 = x  
 if (v > 100) {  
 *document*.getElementById("res").innerHTML = ""  
 *document*.getElementById("info").innerHTML = "<br>На данном интервале метод хорд не может получить решение"  
 return  
 }  
 }  
 }  
  
 function runB() {  
 let v = 1  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>a</th><th>b</th><th>x</th><th>f(a)</th><th>f(b)</th><th>f(x)</th><th>|x\_k - x\_{k+1}|</th></tr>"  
 let x0 = b  
 while (true) {  
 let fx0 = *f*["function"](x0)  
 let fa = *f*["function"](a)  
 x = x0 - (a - x0) / (fa - fx0) \* fx0  
 let fx = *f*["function"](x)  
 *document*.getElementById("res").innerHTML += `<tr><td>${v}</td><td>${a}</td><td>${x0}</td><td>${x}</td><td>${fa}</td><td>${fx0}</td><td>${fx}</td><td>${*Math*.abs(x0 - x)}</td></tr>`  
 if (*Math*.abs(x0 - x) < accuracy) return  
 v++  
 x0 = x  
 if (v > 100) {  
 *document*.getElementById("res").innerHTML = ""  
 *document*.getElementById("info").innerHTML = "<br>На данном интервале метод хорд не может получить решение"  
 return  
 }  
 }  
 }  
  
 if (root(a, b, *f*["derivative"]) || root(a, b, *f*["derivative2"])) {  
 alert("На заданном интервале применение метода может дать некорректный результат")  
 }  
 let x  
 if (*f*["derivative"]((a + b) / 2) \* *f*["derivative2"]((a + b) / 2) > 0) {  
 runA()  
 }  
 if (*f*["derivative"]((a + b) / 2) \* *f*["derivative2"]((a + b) / 2) < 0 || a > x || b < x) {  
 runB()  
 }  
 if (a > x || b < x) {  
 runA()  
 }  
}  
  
function newtonMethod(a, b, accuracy) {  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>x\_k</th><th>f(x\_k)</th><th>f'(x\_k)</th><th>x\_{k+1}</th><th>|x\_k - x\_{k+1}|</th></tr>"  
  
 function run() {  
 let v = 1  
 while (true) {  
 let fx0 = *f*["function"](x0)  
 let dfx0 = *f*["derivative"](x0)  
 x = x0 - fx0 / dfx0  
 *document*.getElementById("res").innerHTML += `<tr><td>${v}</td><td>${x0}</td><td>${fx0}</td><td>${dfx0}</td><td>${x}</td><td>${*Math*.abs(x0 - x)}</td></tr>`  
 if (*Math*.abs(x0 - x) < accuracy) return  
 v++  
 x0 = x  
 if (v > 100) {  
 *document*.getElementById("res").innerHTML = ""  
 *document*.getElementById("info").innerHTML = "<br>На данном интервале метод Ньютона не может получить решение"  
 return  
 }  
 }  
 }  
  
 if (root(a, b, *f*["derivative2"])) {  
 alert("На заданном интервале применение метода Ньютона может дать некорректный результат")  
 }  
 let x0, x  
 if (*f*["function"](a) \* *f*["derivative2"](a)) {  
 x0 = a  
 } else x0 = b  
 run()  
 if (a > x || b < x) {  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>x\_k</th><th>f(x\_k)</th><th>f'(x\_k)</th><th>x\_{k+1}</th><th>|x\_k - x\_{k+1}|</th></tr>"  
 if (*f*["function"](a) \* *f*["derivative2"](a)) {  
 x0 = b  
 } else x0 = a  
 run()  
 }  
}  
  
function simpleIterationMethod(a, b, accuracy) {  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>x\_i</th><th>x\_{i+1}</th><th>phi(x\_{i + 1})</th><th>f(x\_{k+1})</th><th>|x\_k - x\_{k+1}|</th></tr>"  
  
 function run() {  
 let v = 1  
 let x0 = a  
 while (true) {  
 x = x0 + lambda \* (*f*["function"](x0))  
 *document*.getElementById("res").innerHTML += `<tr><td>${v}</td><td>${x0}</td><td>${x}</td><td>${x0 + lambda \* (*f*["function"](x))}</td><td>${*f*["function"](x)}</td><td>${*Math*.abs(x0 - x)}</td></tr>`  
 if (*Math*.abs(x0 - x) < accuracy) return  
 v++  
 x0 = x  
 if (v > 100) {  
 *document*.getElementById("info").innerHTML += "<br>Не удалось добиться нужной точности за вменяемое количество итераций. Метод расходится"  
 return  
 }  
 }  
 }  
  
 if (root(a, b, *f*["derivative"])) {  
 alert("На заданном интервале применение метода простых итераций может дать некорректный результат")  
 }  
 let mx = Number.MIN\_VALUE  
 for (let i = a; i <= b; i += *Math*.abs(a - b) / 100)  
 if (mx < *Math*.abs(*f*["derivative"](i)))  
 mx = *Math*.abs(*f*["derivative"](i))  
 let lambda = 1 / mx  
 lambda \*= *f*["derivative"](mx) > 0 ? -1 : 1  
 *document*.getElementById("info").innerHTML = "Достаточное условие" + (lambda \* *f*["derivative"](mx) + 1 < 1 ? " " : " не ") + "выполняется"  
 *console*.log(lambda \* *f*["derivative"](mx) + 1)  
 let x  
 run()  
 if (a > x || b < x) {  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>x\_i</th><th>x\_{i+1}</th><th>phi(x\_{i + 1})</th><th>f(x\_{k+1})</th><th>|x\_k - x\_{k+1}|</th></tr>"  
 lambda \*= -1  
 *console*.log(lambda \* *f*["derivative"](mx) + 1)  
 *document*.getElementById("info").innerHTML = "Достаточное условие" + (lambda \* *f*["derivative"](mx) + 1 < 1 ? " " : " не ") + "выполняется"  
 run()  
 }  
}  
  
function root(a, b, func) {  
 const step = *Math*.abs(b - a) / 100;  
 let previousSign = *Math*.sign(func(a));  
 let currentSign;  
  
 for (let x = a; x <= b; x += step) {  
 currentSign = *Math*.sign(func(x));  
 if (currentSign !== previousSign || currentSign === 0) {  
 return x;  
 }  
 previousSign = currentSign;  
 }  
  
 return 0;  
}  
  
function hasMoreThat1Root(a, b) {  
 a = root(a, b, *f*["function"])  
 return root(a, b, *f*["function"])  
}  
  
function systemNewtonMethod(x0, y0, accuracy) {  
 *document*.getElementById("info").innerHTML = ""  
 *document*.getElementById("check").innerHTML = ""  
 *document*.getElementById("res").innerHTML = "<tr><th>№ Шага</th><th>x\_i</th><th>yi</th><th>|x\_k - x\_{k+1}|</th><th>|y\_k - y\_{k+1}|</th></tr>"  
  
 let v = 1  
 let x, y  
 while (true) {  
 let a11 = *f*["derivative1X"](x0, y0)  
 let a12 = *f*["derivative1Y"](x0, y0)  
 let a21 = *f*["derivative2X"](x0, y0)  
 let a22 = *f*["derivative2Y"](x0, y0)  
 let b1 = -*f*["function1"](x0, y0)  
 let b2 = -*f*["function2"](x0, y0)  
 let d = a11 \* a22 - a12 \* a21  
 if (*Math*.abs(d) < Number.EPSILON) {  
 *document*.getElementById("res").innerHTML = ""  
 *document*.getElementById("info").innerHTML = "Определитель матрицы равен нулю"  
 return  
 }  
 if (d === 0) break  
 let d1 = b1 \* a22 - b2 \* a12  
 let d2 = a11 \* b2 - a21 \* b1  
 let dx = d1 / d  
 let dy = d2 / d  
 *document*.getElementById("res").innerHTML += `<tr><td>${v}</td><td>${x0}</td><td>${y0}</td><td>${*Math*.abs(dx)}</td><td>${*Math*.abs(dy)}</td></tr>`  
 x = x0 + dx  
 y = y0 + dy  
 if (*Math*.abs(dx) <= accuracy && *Math*.abs(dy) <= accuracy) {  
 break  
 }  
 x0 = x  
 y0 = y  
 v++  
 if (v > 300) {  
 *document*.getElementById("info").innerHTML += "<br>Не удалось добиться нужной точности за вменяемое количество итераций."  
 return  
 }  
 }  
 *document*.getElementById("check").innerHTML = `f1(${x}, ${y}) = ${*f*["function1"](x, y)} ≈ ${*Math*.round(*f*["function1"](x, y))}`  
 *document*.getElementById("check").innerHTML += `<br>f2(${x}, ${y}) = ${*f*["function2"](x, y)} ≈ ${*Math*.round(*f*["function2"](x, y))}`  
}

# Примеры и результаты работы программы

## Пример 1

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

## Пример 2

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

## Пример 3

Изображение выглядит как текст, диаграмма, линия, График

Автоматически созданное описание

# Вывод

Во время выполнения данной лабораторной работы я познакомился с различными итерационными методами решения нелинейных уравнений и систем нелинейных уравнений и запрограммировал некоторые из них.