Национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной Инженерии и Компьютерной Техники

Вариант №16

Лабораторная работа №2

«Численное решение нелинейных уравнений и систем»

По дисциплине:

«Вычислительная математика»

Работу выполнила:

Студентка группы P3212

Никонова Наталья Игоревна

Преподаватель:

Малышева Татьяна Алексеевна

Санкт-Петербург

2022

Цель работы:

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

Рабочие формулы используемых методов

Для вычислительной реализации задачи:

- Метод хорд

Без выбора начального приближения $x_i = \frac{a_i f(b_i) - b_i f(a_i)}{f(b_i) - f(a_i)}$

Окончание: $|x_i - x_{i-1}| \le \varepsilon$ или $|f(x_i)| \le \varepsilon$

Выбор начального приближения: $f(a) \cdot f''(a) > 0 \to x_0 = a$, $f(b) \cdot f''(b) > 0 \to x_0 = b$

Формулы с начальным приближением: при x0=a $x_{i+1} = x_i - \frac{b - x_i}{f(b) - f(x_i)} f(x_i)$

При x0=b $x_{i+1} = x_i - \frac{a - x_i}{f(a) - f(x_i)} f(x_i)$

- Метод простой итерации

$\lambda = -\frac{1}{\max\limits_{[a,b]} f'(x)}$ $\varphi(x) = x + \lambda f(x), \varphi'(x) = 1 + \lambda f'(x)$

$q = \max\limits_{[a,b]} |\varphi'(x)|$ $x_{i+1} = \varphi(x_i)$ Достаточное условия сходимости: $|\varphi'(x)| \le q < 1,$

Критерий окончания итерационного процесса:

$|x_n - x_{n-1}| \le \varepsilon$ (при $0 < q \le 0{,}5$)

$|x_n - x_{n-1}| < \frac{1-q}{q}\varepsilon$ (при $0{,}5 < q < 1$)

- Метод Ньютона

$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$ Окончание: $|x_n - x_{n-1}| \le \varepsilon$ или $\left|\frac{f(x_n)}{f'(x_n)}\right| \le \varepsilon$ или $|f(x_n)| \le \varepsilon$

Выбор начального приближения $f(x_0) \cdot f''(x_0) > 0$ для $x0 = a, b$

Для программной реализации задачи:

- Метод половинного деления (для уравнения)

$x_i = \frac{a_i + b_i}{2}$ Критерий окончания: $|b_n - a_n| \le \varepsilon$

- Метод простой итерации (для уравнения) – см. выше в вычислительной реализации задачи
- Метод Ньютона (для системы)

$$\begin{vmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{vmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -\begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

Откуда:

$$\Delta y = \frac{g - \frac{c * f}{a}}{d - \frac{c * b}{a}} \quad \Delta x = \frac{f - b\Delta y}{a} \quad \text{где } a = \frac{\partial f(x,y)}{\partial x}, b = \frac{\partial f(x,y)}{\partial y}, c = \frac{\partial g(x,y)}{\partial x}, d = \frac{\partial g(x,y)}{\partial y}, f = f(x,y), g = g(x,y)$$

Заполненные таблицы

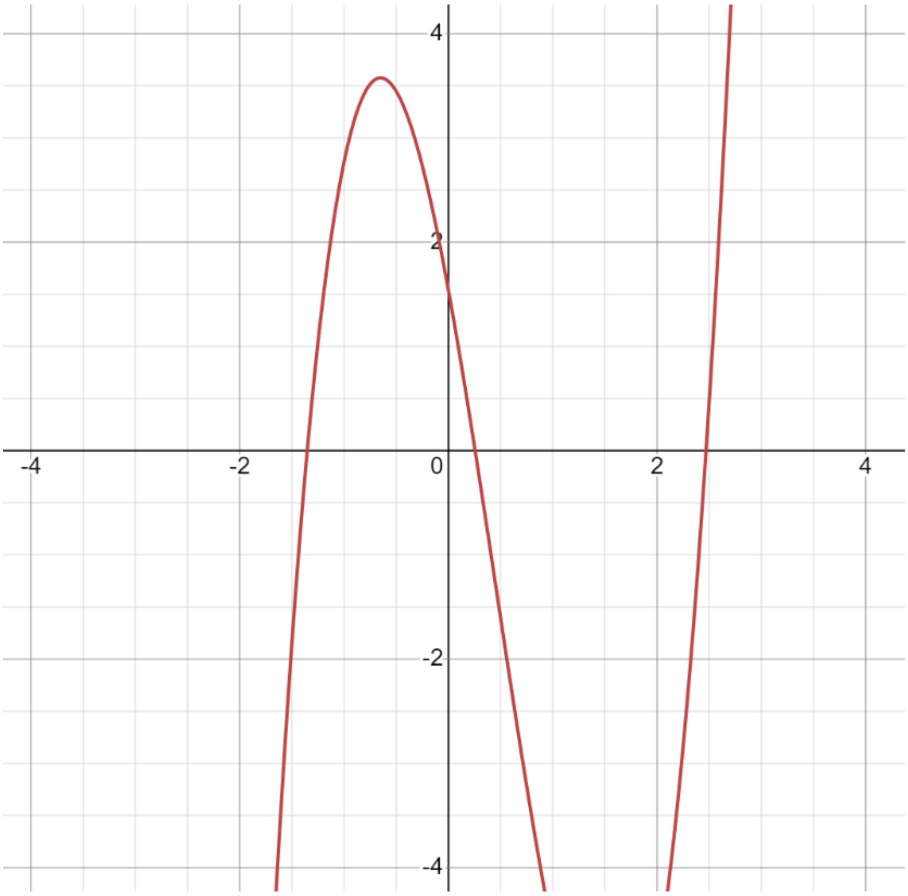
Данная функция: $f(x) = 1{,}8x^3 - 2{,}47x^2 - 5{,}53x + 1{,}539$

$$f'(x) = 5{,}4x^2 - 4{,}94x - 5{,}53$$

$$f''(x) = 10{,}8x - 4{,}94$$

$$\varepsilon = 0{,}01$$

График:



Крайний правый корень – метод хорд

Интервал изоляции – [-1.5;-1]

№ шага	a	b	x	f(a)	f(b)	f(x)	|a-b|

1	-1.500	-1.000	-1.304	-1.799	2.799	0.559	0.500
2	-1.500	-1.304	-1.350	-1.799	0.559	0.074	0.196
3	-1.500	-1.350	-1.356	-1.799	0.074	0.008 < ε	0.150

$$x_{п} = -1{,}356$$

Крайний левый корень – метод простой итерации

Интервал изоляции – [2;2,5]

№ итерации	x_k	$f(x_k)$	x_{k+1}	$\varphi(x_k)$	$\lvert x_k - x_{k+1}\rvert$
1	2.000	-5.001	2.315	2.315	0.315
2	2.315	-2.166	2.452	2.452	0.136
3	2.452	-0.340	2.473	2.473	0.021
4	2.473	-0.017	2.474	2.474	0.001<ε

$$x_{л} = 2{,}474$$

Центральный корень – метод Ньютона

Интервал изоляции – [0;0,5]

$$f(0) = 1{,}539 \quad f''(0) = -4{,}94 - \text{разных знаков}$$

$$f(0.5) = 0{,}87 \quad f''(0.5) = 0{,}46 - \text{одного знака, значит } x0 = b$$

№ итерации	x_k	$f(x_k)$	$f'(x_k)$	x_{k+1}	$\lvert x_k - x_{k+1}\rvert$
1	0.500	-1.619	-6.650	0.257	0.243
2	0.257	-0.015	-6.442	0.255	0.002< ε

$$x_{ц} = 0{,}255$$

Листинг программы

Полностью код туть: https://github.com/nanikon/computational_math/tree/master/nle

Метод половинного деления:

Проверка на корректность интервала изоляции корней:

```
override fun setAndVerifyData(a: BigDecimal, b: BigDecimal): Boolean {
    leftBorder = a
    rightBorder = b
    val left = a.multiply(BigDecimal( val: 1000)).toInt()
    val right = b.multiply(BigDecimal( val: 1000)).toInt()
    val marker = eq.firstDerivative(a)
    isCorrect = eq.function(a) * eq.function(b) <= BigDecimal.ZERO && (left ≤ .. ≤ right).all { it: Int
        marker.multiply(eq.firstDerivative(BigDecimal( val: it.toDouble() / 1000).setScale( newScale: 3, RoundingMode.HALF_UP))) >= BigDecimal.ZERO
    }
    return isCorrect
}
```

Вычисление:

```
override fun solve(approximation: BigDecimal): OneVariableResultData {
    if (!isCorrect) throw RuntimeException("Сохраненные на данный момент входные данные не валидны, расчет невозможен")
    var a = leftBorder
    var b = rightBorder
    var count = 0
    var x = BigDecimal.ZERO
    var y = approximation.plus(BigDecimal.ONE)
    while ((b.minus(a) > approximation) && y.abs() > approximation) {
        x = a.plus(b).divide(BigDecimal( val: 2), MathContext.DECIMAL64)
        y = eq.function(x)
        count++
        println("Итерация №$count: левая граница $a, правая граница $b, приближение к корню $x, значение функции в нем $y")
        if (y.multiply(eq.function(a)) < BigDecimal.ZERO) { b = x } else { a = x }
    }
    val delta = rightBorder.minus(leftBorder).divide(BigDecimal( val: 5), MathContext.DECIMAL64)
    createGraph(eq.function, { BigDecimal( val: 0) },  leftBorder: leftBorder - delta,  rightBorder: rightBorder + delta)
    return OneVariableResultData(root = x, valueRoot = y, countIteration = count)
}
```

Метод простых итераций:

Проверка на корректность интервала изоляции корней:

```
override fun setAndVerifyData(a: BigDecimal, b: BigDecimal): Boolean {
    leftBorder = a
    rightBorder = b
    val left = a.multiply(BigDecimal( val: 1000)).toInt()
    val right = b.multiply(BigDecimal( val: 1000)).toInt()
    lambda = - BigDecimal.ONE.divide((left ≤ .. ≤ right).maxOf {  it: Int
        eq.firstDerivative(BigDecimal( val: it.toDouble() / 1000).setScale( newScale: 3, RoundingMode.HALF_UP))
    }, MathContext.DECIMAL64)
    q = (left ≤ .. ≤ right).maxOf {  it: Int
        BigDecimal.ONE.plus(
            lambda.multiply(eq.firstDerivative(BigDecimal( val: it.toDouble() / 1000).setScale( newScale: 3, RoundingMode.HALF_UP)))
        ).abs()
    }
    isCorrect = q < BigDecimal.ONE
    return isCorrect
}
```

Вычисление:

```
override fun solve(approximation: BigDecimal): OneVariableResultData {
    if (!isCorrect) throw RuntimeException("Сохраненные на данный момент входные данные не валидны, расчет невозможен")
    var last: BigDecimal
    var current = leftBorder
    var count = 0
    do {
        last = current
        current = eq.function(last).multiply(lambda, MathContext.DECIMAL64).plus(last)
        count++
        println("Итерация №$count: предыдущее приближение $last, текущее приближение $current, " +
            "значение функции на предыдущем ${eq.function(last)}, модуль разницы ${current.minus(last).abs()}")
    } while (((q <= BigDecimal( val: 0.5)) && (current.minus(last).abs() >= approximation)) ||
        ((q > BigDecimal( val: 0.5))
                && (current.minus(last).abs() >= BigDecimal.ONE.minus(q).divide(q, MathContext.DECIMAL64).multiply(approximation))))
    val delta = rightBorder.minus(leftBorder).divide(BigDecimal( val: 5), MathContext.DECIMAL64)
    createGraph(eq.function, { BigDecimal( val: 0) },  leftBorder: leftBorder - delta,  rightBorder: rightBorder + delta)
    return OneVariableResultData(root = current, valueRoot = eq.function(current), countIteration = count)
}
```

Метод Ньютона (для системы):

```kotlin
override fun solve(approximation: BigDecimal): ManyVariablesResultData {
    var count = 0
    var deltaX: BigDecimal
    var deltaY: BigDecimal
    var x = x0
    var y = y0
    do {
        val a = if(firstEq.derivativeX(x, y) == BigDecimal.ZERO) approximation else firstEq.derivativeX(x, y)
        val b = firstEq.derivativeY(x, y)
        val f = - firstEq.function(x, y)
        val c = secondEq.derivativeX(x, y)
        val d = secondEq.derivativeY(x, y)
        val g = - secondEq.function(x, y)
        val cDivideA = c.divide(a, MathContext.DECIMAL64)
        deltaY = g.minus(f.multiply(cDivideA)).divide(d.minus(b.multiply(cDivideA)), MathContext.DECIMAL64)
        deltaX = f.minus(b.multiply(deltaY)).divide(a, MathContext.DECIMAL64)
        x += deltaX
        y += deltaY
        count++
        println("Итерация №$count: приближение к X $x, приближение к Y $y, приращение к X $deltaX, " +
                "приращение к Y $deltaY, значение первого уравнения ${firstEq.function(x, y)}, " +
                "второго ${secondEq.function(x, y)}")
        if (count >= 100) { throw SolutionNotExistException(count) }
    } while (deltaX.abs() >= approximation || deltaY.abs() >= approximation)
    return ManyVariablesResultData(root = listOf(x, y), errors = listOf(deltaX, deltaY), countIteration = count)
}
```

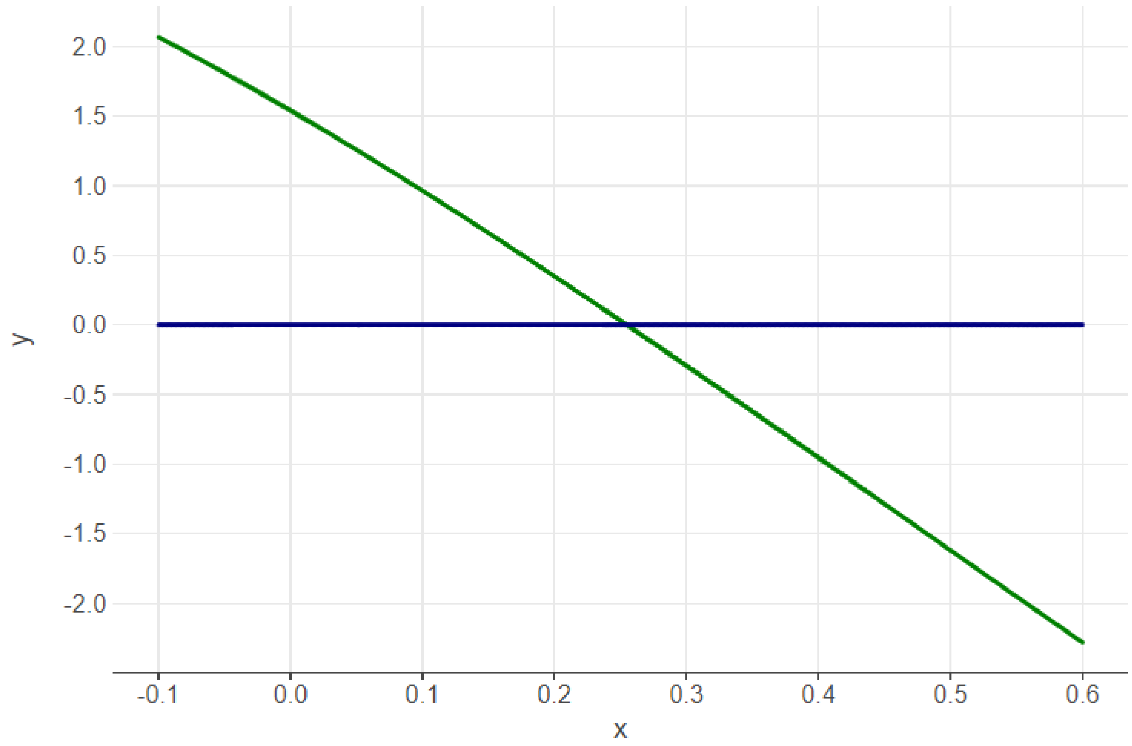
Результат выполнения программы

Решение уравнения:

```
Для решения одного уравнения введите 1, для решения системы из двух введите 2
1
Выберите функцию для решения и введите её номер:
1 - 1,8x^3 - 2,47x^2 - 5,53x + 1,539
2 - x^2 + sin(x) - 3
3 - e^-x - e^x
4 - x^3 - 1,8x^2 - 8,64x + 17,28
5 - x^10 - x^7  - 5x^3 + 2x^2 + x
1
Для использования метода половинного деления введите 1, для метода простых итераций - 2
1
Для ввода границ интервала изоляции корней и погрешности с клавиатуры введите 1, для ввода с файла - 2
1
Введите левую границу интервала:
0
Введите правую границу интервала:
0.5
Введите погрешность:
0.001
```

```
Итерация №1: левая граница 0, правая граница 0.5, приближение к корню 0.25, значение функции в нем 0.0302499999999998499256026462944646482355892658233642578125
Итерация №2: левая граница 0.25, правая граница 0.5, приближение к корню 0.375, значение функции в нем -0.7871718750000001947782213296278541747597046196460723876953125
Итерация №3: левая граница 0.25, правая граница 0.375, приближение к корню 0.3125, значение функции в нем -0.3754042968750001718256613381097253068219288252294063568115234375
Итерация №4: левая граница 0.25, правая граница 0.3125, приближение к корню 0.28125, значение функции в нем -0.1716481933593751607958015868715673235556096187792718410491943359375
Итерация №5: левая граница 0.25, правая граница 0.28125, приближение к корню 0.265625, значение функции в нем -0.07044625854492203039603431076118300602217914274660870432853698730
Итерация №6: левая граница 0.25, правая граница 0.265625, приближение к корню 0.2578125, значение функции в нем -0.020032344818115387100386014880473023458407055841234978288412094
Итерация №7: левая граница 0.25, правая граница 0.2578125, приближение к корню 0.25390625, значение функции в нем 0.00512559556961044430569871097909786031299184116960532264783978
Итерация №8: левая граница 0.25390625, правая граница 0.2578125, приближение к корню 0.255859375, значение функции в нем -0.007449222862720641562743920749397133224817268981610141
Итерация №9: левая граница 0.25390625, правая граница 0.255859375, приближение к корню 0.2548828125, значение функции в нем -0.001160770677030238246288563632735141414788898606147
Введите 1, чтобы вывести результат на консоль, и 2, чтобы вывести его в файл
1
Найденный корень уравнения: 0.2548828125
Значение функции в корне: -0.0011607706770302382462885636327351414147888986061474270172766409814357757568359375
Число итераций: 9
```

Построенный график:

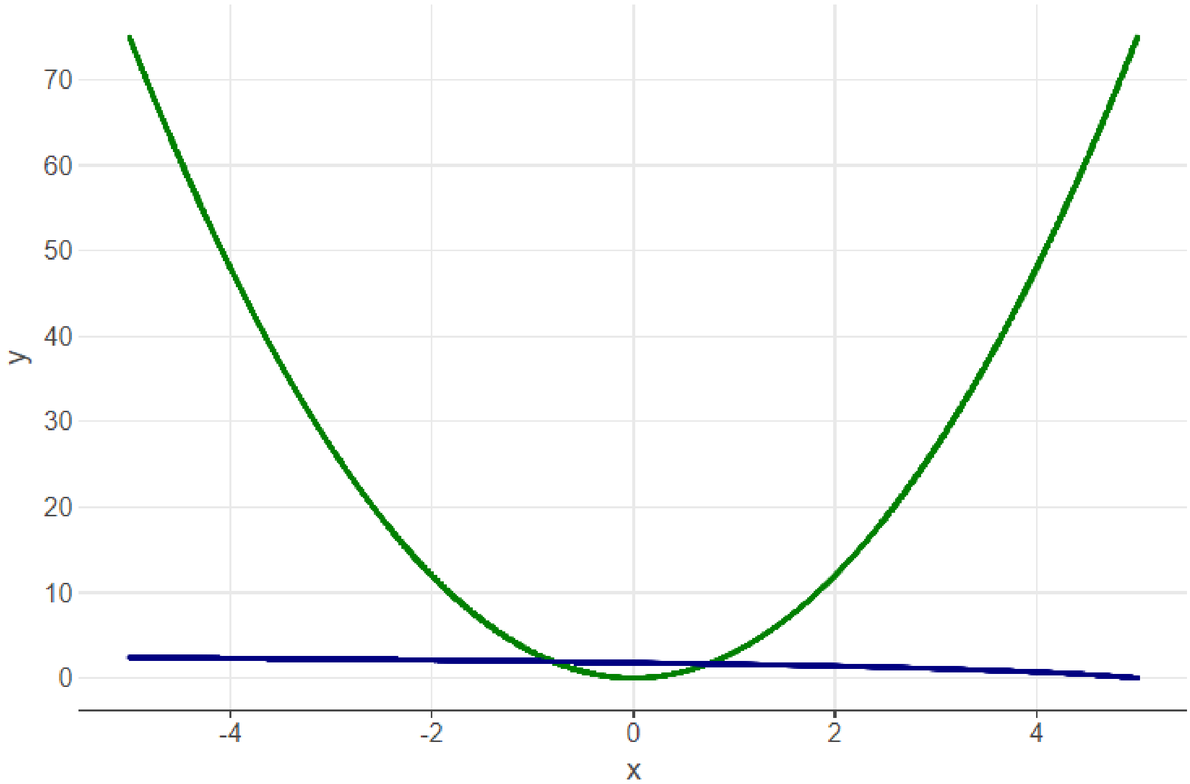


Решение системы уравнений:

```
Для решения одного уравнения введите 1, для решения системы из двух введите 2
2
Выберите первое уравнение для системы и введите его номер:
1 - x^2 + sqrt(y)^4 = 4
2 - y = 3 * x^2
3 - x + e^y = 6
4 - 6x - y = 9.89
2
Выберите второе уравнение для системы и введите его номер:
1 - x^2 + sqrt(y)^4 = 4
2 - y = 3 * x^2
3 - x + e^y = 6
4 - 6x - y = 9.89
3
Для ввода начального приближения переменных и погрешности с клавиатуры введите 1, для ввода с файла - 2
1
Введите приближение переменной X:
1
Введите приближение переменной Y:
5
Введите погрешность:
0.0001
```

```
Итерация №1: приближение к X 1.1720883687812367, приближение к Y 4.0325302126874203, приращение к X 0.1720883687812367, приращение к Y -0.9674697873125797, значение первого уравн
Итерация №2: приближение к X 1.0297890451768076, приближение к Y 3.1206491402036492, приращение к X -0.1422993236044291, приращение к Y -0.9118810724837711, значение первого урав
Итерация №3: приближение к X 0.8945748679942219, приближение к Y 2.3459439622071474, приращение к X -0.1352141771825857, приращение к Y -0.7747051779965018, значение первого урав
Итерация №4: приближение к X 0.7909782600399248, приближение к Y 1.8447430520284515, приращение к X -0.1035966079542971, приращение к Y -0.5012009101786959, значение первого урав
Итерация №5: приближение к X 0.7483945221059537 0, приближение к Y 1.6748429579453091, приращение к X -0.042583737933971 10, приращение к Y -0.1699000940831424, значение первого ур
Итерация №6: приближение к X 0.743773244244570048, приближение к Y 1.65953184793506221, приращение к X -0.004621277861383652, приращение к Y -0.01531111001024689, значение первог
Итерация №7: приближение к X 0.74373402390736670137, приближение к Y 1.6594208903376259613, приращение к X -0.00003922033720334663, приращение к Y -0.0001109575974362487, значени
Итерация №8: приближение к X 0.74373402159235340260758 0, приближение к Y 1.659420884621805575820990, приращение к X -2.315013298762420E-9, приращение к Y -5.715820385479010E-9, з
Введите 1, чтобы вывести результат на консоль, и 2, чтобы вывести его в файл
1
Вектор неизвестных:
0.74373402159235340260758 0
1.659420884621805575820990
Вектор погрешностей:
-2.315013298762420E-9
-5.715820385479010E-9
Число итераций:
8
```

Построенный график:



Выводы

В ходе выполнения лабораторной работы я познакомилась с численными методами решения нелинейных уравнений и систем уравнений. А также познакомилась с одной из библиотек языка kotlin, позволяющих строить и выводить графики.