Министерство высшего образования и науки Российской Федерации
Национальный научно-исследовательский университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по дисциплине

«Вычислительная математика».

Вариант №1.

Работу выполнил:
Афанасьев Кирилл Александрович,
Студент группы P3206.
Преподаватель:
Рыбаков Степан Дмитриевич.

Санкт-Петербург, 2024

Оглавление

Задание

1. Используя численные методы найти приближение корня нелинейного уравнения и системы нелинейных уравнений
   - 1.1. **Решение нелинейного уравнения**
     - 1.1.1. Отделить корни заданного нелинейного уравнения графически (вид уравнения представлен в табл. 6)
     - 1.1.2. Определить интервалы изоляции корней
     - 1.1.3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью $\varepsilon=10^{-2}$
     - 1.1.4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.
     - 1.1.5. Вычисления оформить в виде таблиц (1–5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой
       - 1.1.5.1. Для метода половинного деления заполнить таблицу 1
       - 1.1.5.2. Для метода хорд заполнить таблицу 2
       - 1.1.5.3. Для метода Ньютона заполнить таблицу 3
       - 1.1.5.4. Для метода секущих заполнить таблицу 4
   - 1.2. **Решение системы нелинейных уравнений**
     - 1.2.1. Отделить корни заданной системы нелинейных уравнений графически (вид системы представлен в табл. 8).
     - 1.2.2. Используя указанный метод, решить систему нелинейных уравнений с точностью до 0,001.
     - 1.2.3. Для метода простой итерации проверить условие сходимости метода.
     - 1.2.4. Подробные вычисления привести в отчете.
2. Реализовать приложение, вычисляющее решение системы линейных алгебраических уравнений (СЛАУ)
   - 2.1. № варианта определяется как номер в списке группы согласно ИСУ
   - 2.2. В программе численные методы должен быть реализован в виде отдельной подпрограммы/метода/класса, в который исходные/выходные данные передаются в качестве параметров
   - 1.1. **Для нелинейных уравнений:**
     - 1.1.1. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя
     - 1.1.2. Выполнить верификацию исходных данных. Необходимо анализировать наличие корня на введенном интервале. Если на интервале несколько корней или они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные
     - 1.1.3. Для методов, требующих начальное приближение к корню (метод Ньютона, секущих, хорд с фиксированным концом, простой итерации), выбор начального приближения x_0 (a или b) вычислять в программе
     - 1.1.4. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
     - 1.1.5. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом). Пользователь должен видеть интервалы изоляции корней.
   - 1.2. **Для систем нелинейных уравнений:**
     - 1.2.1. Организовать вывод графика функций
     - 1.2.2. Начальные приближения ввести с клавиатуры
     - 1.2.3. Организовать вывод вектора неизвестных: x_1, x_2, ..., x_n
     - 1.2.4. Организовать вывод количества итераций, за которое было найдено решения

1.2.5. Организовать вывод вектора погрешностей: r_1 , r_2, ..., r_n
1.2.6. Проверить правильность решения системы нелинейных уравнений

Описание методов

Вариант 1:

Метод Ньютона (Решение НУ + программное решение НУ):

Функция $y = f(x)$ на отрезке [a, b] заменяется касательной и в качестве приближенного значения корня принимается точка пересечения касательной с осью абсцисс.
Пусть $x_0 \in [a, b]$ - начальное приближение. Запишем уравнение касательной к графику функции $y = f(x)$в этой точке:
$y=f\ x_0 + f'(x_0)(x-x_0)$
Найдем пересечение касательной с осью x:
$x1 = x_0 - f(x_0)/f'(x_0)$
Рабочая формула метода:
$\boldsymbol{xi = x_{i-1} - f(x_{i-1})/f'(x_{i-1})}$

Метод секущих (Решение НУ):

Упростим метод Ньютона, заменив $f'(x)$ разностным приближением:
$f'(x_i) \approx f(x_i) - f(x_{i-1})\ x_i - x_{i-1}$
Рабочая формула метода:
$\boldsymbol{x_{i+1} = x_i - (x_i - x_{i-1})/(f(x_i) - f(x_{i-1}))*f(x_i)}$; $i = 1,2$ ...
Метод секущих является двухшаговым, т. е. новое приближение x_{i+1} определяется двумя предыдущими итерациями x_i и x_{i-1}.
Выбор x_0 определяется как и в методе Ньютона, x_1 выбирается рядом с начальным самостоятельно.

Метод простой итерации (Решение НУ + программное решение НУ):

Уравнение $f\ x = 0$ приведем к эквивалентному виду: $x = \varphi(x)$, выразив x из исходного уравнения. Зная начальное приближение: $x_0 \in a, b$, найдем очередные приближения:
$x_1 = \varphi(x_0) \rightarrow x_2 = \varphi\ x_1$ ...
Рабочая формула метода: $\boldsymbol{x_{i+1} = \varphi(xi)}$
Условия сходимости метода простой итерации определяются следующей теоремой.
Теорема. Если на отрезке локализации a, b функция $\varphi(x)$ определена, непрерывна и дифференцируема и удовлетворяет неравенству:
$|\varphi'(x)| < q$, где $0 \leq q < 1$, то независимо от выбора начального приближения $x_0 \in a, b$ итерационная последовательность $\{xn\}$ метода будет сходится к корню уравнения.
Достаточное условие сходимости метода:
$|\varphi'(x)| \leq q < 1$, где q – некоторая константа (коэффициент Липшица или коэффициент сжатия)
$q = \max_{[a,b]} |\varphi'\ x|$
При $q \approx 0$ - скорость сходимости высокая,
При $q \approx 1$ - скорость сходимости низкая,
При $q > 1$ - нет сходимости.
Чем меньше q, тем выше скорость сходимости.

Метод простой итерации (Решение СНУ):

Приведем систему уравнений к эквивалентному виду:
$F_1(x_1, x_2, ..., x_n)=0$
$F_2(x_1, x_2, ..., x_n)=0$

.....
$F_n(x_1, x_2, ..., x_n)=0$

$x_1=\varphi_1(x_1, x_2, ..., x_n)$
$x_2=\varphi_2(x_1, x_2, ..., x_n)$
.....
$x_n=\varphi_n(x_1, x_2, ..., x_n)$
Если выбрано начальное приближение: $\boldsymbol{X}^{(0)} = x_1^0, x_2^0, ..., x_n^0$
получим первые приближения к корням:
$x_1^{(1)}=\varphi_1(x_1^0, x_2^0, ..., x_n^0)$
$x_2^{(1)}=\varphi_2(x_1^0, x_2^0, ..., x_n^0)$
.....
$x_n^{(1)}=\varphi_n(x_1^0, x_2^0, ..., x_n^0)$
Последующие приближения находятся по формулам (k = 0, 1, 2, ...):
$x_1^{(k+1)}=\varphi_1(x_1^k, x_2^k, ..., x_n^k)$
$x_2^{(k+1)}=\varphi_2(x_1^k, x_2^k, ..., x_n^k)$
.....
$x_n^{(k+1)}=\varphi_n(x_1^k, x_2^k, ..., x_n^k)$

Метод половинного деления (Программное решение НУ):

Идея метода: начальный интервал изоляции корня делим пополам, получаем начальное приближение к корню:
$x_0 = (a_0+b_0) / 2$
Вычисляем f(x_0). В качестве нового интервала выбираем ту половину отрезка, на концах которого функция имеет разные знаки: [a_0,x_0] либо [b_0,x_0]. Другую половину отрезка [a_0, b_0], на которой функция f(x) знак не меняет, отбрасываем. Новый интервал вновь делим пополам, получаем очередное приближение к корню: $x_1=(a_1+b_1) / 2$. и т. д.
Рабочая формула метода: $\boldsymbol{x_i} = (\boldsymbol{a_i} + \boldsymbol{b_i}) / \mathbf{2}$
Приближенное значение корня: $x^*= (a_n+b_n) / 2$ или $x^*= a_n$ или $x^*= b_n$

Метод Ньютона (Программное решение СНУ):

К основе метода лежит использование разложения функций $F\ x\ ,\ x\ ,\ ...\ ,\ x$ в ряд $i12\ n$ Тейлора в окрестности некоторой фиксированной точки, причем члены, содержащие вторые (и более высоких порядков) производные, отбрасываются.
Пусть начальные приближения неизвестных системы (1) получены и равны соответственно $a1,\ a2,\ ...\ ,\ an$. Задача состоит в нахождении приращений (поправок) к этим значениям $\Delta x1,\ \Delta x2,\ ...\ ,\ \Delta xn$, благодаря которым решение системы запишется в виде
$x1=a1+\Delta x1\ ,x2=a2+\Delta x2\ ,...,xn\ =an+\Delta xn$ (2)
Проведем разложение левых частей уравнений (1) с учетом (2) в ряд Тейлора, ограничиваясь лишь линейными членами относительно приращений (3):
$F_1(x_1, x_2, ..., x_n) \approx F_1(a_1, a_2, ..., a_n) + \partial F_1/\partial x_1 * \Delta x_1 + ... + \partial F_1/\partial x_n * \Delta x_n$
$F_2(x_1, x_2, ..., x_n) \approx F_2(a_1, a_2, ..., a_n) + \partial F_2/\partial x_1 * \Delta x_1 + ... + \partial F_2/\partial x_n * \Delta x_n$
.....
$F_n(x_1, x_2, ..., x_n) \approx F_n(a_1, a_2, ..., a_n) + \partial F_n/\partial x_1 * \Delta x_1 + ... + \partial F_n/\partial x_n * \Delta x_n$
Поскольку в соответствии с (1) левые части этих выражений должны обращаться в нуль, то приравняем к нулю и правые части. Получим следующую систему линейных алгебраических уравнений относительно приращений:
$\partial F_1/\partial x_1 * \Delta x_1 + \partial F_1/\partial x_2 * \Delta x_2 + ... + \partial F_1/\partial x_n * \Delta x_n = -F_1$
$\partial F_2/\partial x_1 * \Delta x_1 + \partial F_2/\partial x_2 * \Delta x_2 + ... + \partial F_2/\partial x_n * \Delta x_n = -F_2$
.....
$\partial F_n/\partial x_1 * \Delta x_1 + \partial F_n/\partial x_2 * \Delta x_2 + ... + \partial F_n/\partial x_n * \Delta x_n = -F_n$

ЗначенияF_1, F_2,..., F_n и их производные вычисляются при $x = a_1$, $x = a_2$,..., $x = a_n$.
Определителем системы (3) является якобиан:
J = det(
$\partial F_1/\partial x_1$ …. $\partial F_1/\partial x_n$
$\partial F_2/\partial x_1$ …. $\partial F_2/\partial x_n$
……
$\partial F_n/\partial x_1$ …. $\partial F_n/\partial x_n$)
Итерационный процесс решения систем нелинейных уравнений методом Ньютона состоит в определении приращений Δx_1,Δx_2, ..., Δx_n к значениям неизвестных на каждой итерации.

Вычислительная реализация

Уравнение: $2{,}74x^3 - 1{,}93x^2 - 15{,}28x - 3{,}72$

Интервалы изоляции корней:

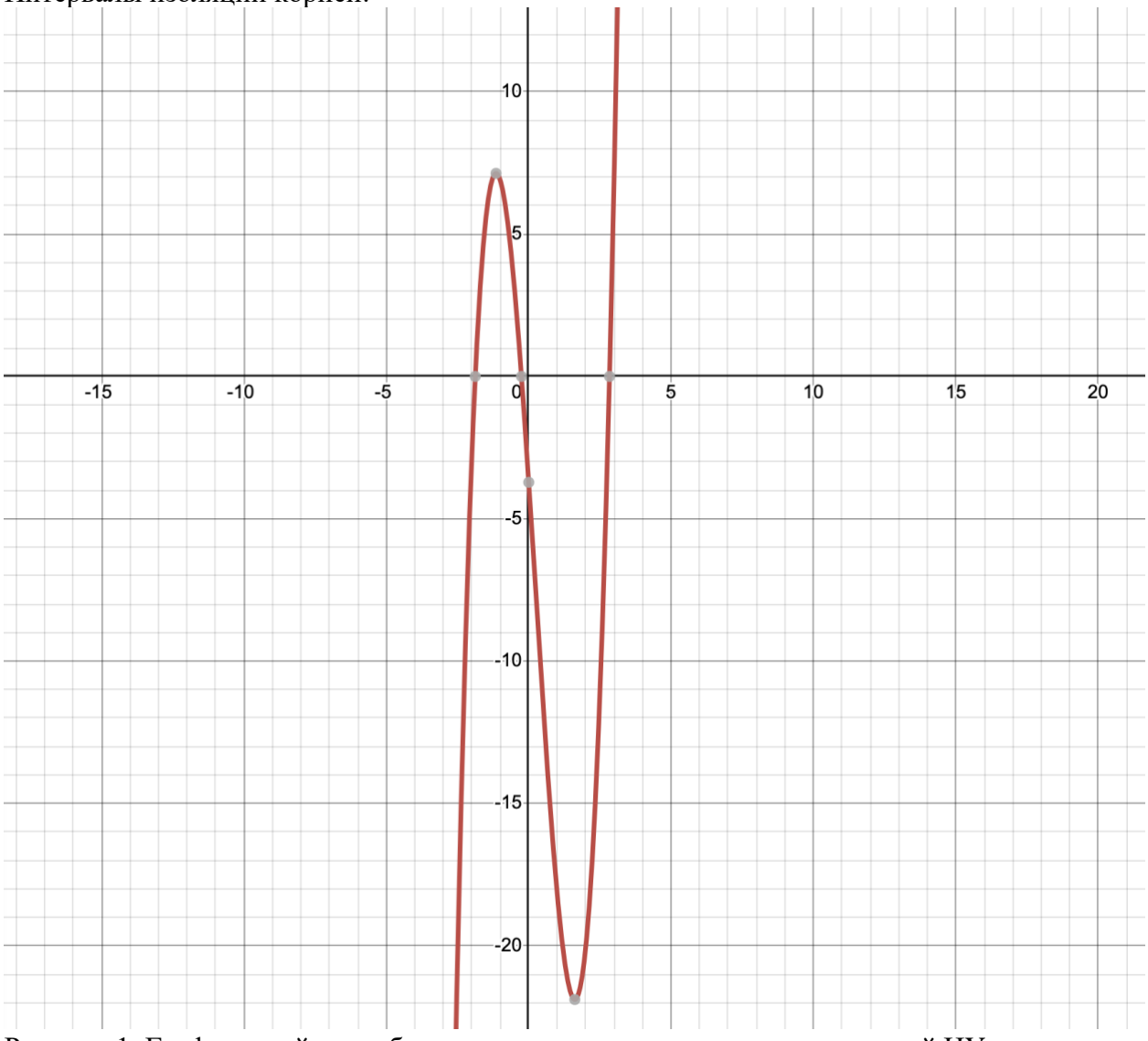

Рисунок 1. Графический способ определения интервалов изоляции корней НУ.

Крайний левый корень: [-3; -1]
Крайний правый корень: [2; 4]
Центральный корень: [-1; 1]

Таблицы приближения корней:

Таблица 1. Уточнение корня уравнения методом Ньютона

№	x_k	$f(x_k)$	$f'(x_k)$	x_{k+1}	$\lvert x_{k+1} - x_k \rvert$
1	-2	-2.8	25.32	-1.889	0.11
2	-1.889	-0.212	21.34	-1.879	0.009
3	-1.879	-0.000	20.995	-1.879	0.000

Калькулятор: https://www.desmos.com/calculator/2ct71tm3po?lang=ru

Таблица 2. Уточнение корня уравнения методом секущих

№	x_{k-1}	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	2.5	3	2.806	-1.235	0.306
2	2.806	2.5	2.845	0.272	0.039
3	2.845	2.806	2.837	-0.005	0.007
4	2.837	2.845	2.838	-0.000	0.001

Калькулятор: https://www.desmos.com/calculator/9ax74w8hk7?lang=ru

Таблица 3. Уточнение корня уравнения методом простой итерации

№	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	-1	-0.549	3.635	0.451
2	-0.549	-0.311	0.766	0.238
3	-0.311	-0.261	0.089	0.050
4	-0.261	-0.255	0.009	0.006
5	-0.255	-0.255	0.000	0.000

Калькулятор: https://www.desmos.com/calculator/zxtalnebu2?lang=ru

Система НУ: $\begin{cases} \sin(x+1) - y = 1.2 \\ 2x + \cos y = 2 \end{cases}$

Область корня: 0 < x < 1; -1 < y < 0

Эквивалентная система: $\begin{cases} y = \sin(x+1) - 1.2 \\ x = 1 - \frac{\cos y}{2} \end{cases}$

Частные производные:
$\partial\varphi_1/\partial x = \cos(x+1)$
$\partial\varphi_1/\partial y = 0$
$\partial\varphi_2/\partial x = 0$
$\partial\varphi_2/\partial y = \sin(y) / 2$

Очевидно, что процесс сходящийся, так как синус и косинус не дают значений выше 1 и суммируются с нулем.

Начальное приближение: x = 1; y = 0

Таблица 4. Уточнение корней системы методом простой итерации

№	x	y	x_{i+1}	y_{i+1}	$F_1(x_{i+1}; y_{i+1})$	$F_2(x_{i+1}; y_{i+1})$	$\lvert x_{i+1} - x_i \rvert$	$\lvert y_{i+1} - y_i \rvert$
1	1	0	0.500	-0.291	-0.290	1.000	0.500	0.291
2	0.500	-0.291	0.521	-0.203	0.088	-0.042	0.021	0.088
3	0.521	-0.203	0.510	-0.201	0.002	0.021	0.010	0.002
4	0.510	-0.201	0.510	-0.202	-0.000	-0.000	0.000	0.001

Калькулятор: https://www.desmos.com/calculator/od7xei0q3f?lang=ru

Исходный код программы

Листинг вычисления решения НУ методом половинного деления:

```
override fun solveEquation(equationParams: EquationParams): EquationResult {
    val equation = equationParams.equation
    val epsilon = equationParams.epsilon

    var iterations = 0u

    var a = equationParams.a
    var b = equationParams.b
    var fA = EquationService.calculateFunction(equation, a)

    while (b - a > epsilon / 2) {
        val x = (a + b) / 2
        val fX = EquationService.calculateFunction(equationParams.equation,
x)

        if (fA * fX > 0) {
            a = x
            fA = EquationService.calculateFunction(equation, a)
        } else {
            b = x
        }

        iterations++
    }

    val xRes = (a + b) / 2
    val yRes = EquationService.calculateFunction(equation, xRes)

    return EquationResult.ok(EquationSolvingMethod.HALF_DIVISION_METHOD,
xRes, yRes, iterations)
}
```

Листинг вычисления решения НУ методом Ньютона:

```
override fun solveEquation(equationParams: EquationParams): EquationResult {
    val f = equationParams.equation

    var iterations = 0u

    var xRes = (equationParams.a + equationParams.b) / 2
    var yRes = EquationService.calculateFunction(f, xRes)

    while (abs(yRes) > equationParams.epsilon) {
        val yFirstDer = EquationService.calculateDerivative(f, xRes)
        val ySecondDer = EquationService.calculateNDerivative(f, xRes, 2u)
```

```
        if (yRes * ySecondDer <= 0)
            throw LowEfficiencyMethodException()

        xRes -= yRes / yFirstDer
        yRes = EquationService.calculateFunction(f, xRes)
        iterations++
    }

    return EquationResult.ok(EquationSolvingMethod.NEWTON_METHOD, xRes, yRes,
iterations)
}
```

Листинг вычисления решения НУ методом простой итерации:

```
override fun solveEquation(equationParams: EquationParams): EquationResult {
    var iterations = 0u

    equation = equationParams.equation
    val a = equationParams.a
    val b = equationParams.b
    val epsilon = equationParams.epsilon

    val maximumOfDer = FunctionUtil.findMaximum(::calculateDer, a, b,
epsilon)

    val lambda = -(1 / calculateDer(maximumOfDer))

    val phi = "x + $lambda*($equation)"

    val phiDerA = EquationService.calculateDerivative(phi, a)
    val phiDerB = EquationService.calculateDerivative(phi, b)

    if (abs(phiDerA) > 1 || abs(phiDerB) > 1) throw
LowEfficiencyMethodException()

    var xLast = maximumOfDer
    var xNext = EquationService.calculateFunction(phi, xLast)
    while (abs(xNext - xLast) > epsilon) {
        if (iterations > 1000u) throw LowEfficiencyMethodException()
        xLast = xNext
        xNext = EquationService.calculateFunction(phi, xLast)
        iterations++
    }

    return EquationResult.ok(
        EquationSolvingMethod.SIMPLE_ITERATION_METHOD,
        xNext,
        EquationService.calculateFunction(equation, xNext),
        iterations
    )
}
```

Листинг вычисления решения СНУ методом Ньютона:

```
override fun solveSystem(equationSystemParams: EquationSystemParams):
EquationSystemResult {
    var iterations = 0u
```

```
    try {
        val foundSymbols = mutableSetOf<String>()
        equationSystemParams.equations.map {
            val regex = Regex("""(\b[a-z]\b|[a-z]_[0-9]*)""")
            val matches = regex.findAll(it)
            for (match in matches) {
                foundSymbols.add(it.substring(match.range))
            }
        }
        val immutableFoundSymbols = foundSymbols.toList()
        val jacobMatrix = mutableListOf<List<FunctionArgumentDerivative>>()

        for (equation in equationSystemParams.equations) {
            val jacobVector = mutableListOf<FunctionArgumentDerivative>()
            for (symbol in immutableFoundSymbols) {
                val functionWithArgumentDerivative =
FunctionArgumentDerivative(equation, symbol)
                jacobVector.add(functionWithArgumentDerivative)
            }
            jacobMatrix.add(jacobVector)
        }

        val slaeMatrix = Matrix(equationSystemParams.equations.size,
immutableFoundSymbols.size)
        val result = equationSystemParams.startApproximation.toMutableMap()
        var flag = true
        while (flag) {
            for (i in 0..<equationSystemParams.equations.size) {
                for (j in immutableFoundSymbols.indices) {
                    slaeMatrix.setMatrixElement(
                        i, j,
jacobMatrix[i][j].calculateDerivative(result).toString()
                    )
                }
            }
            val extendedVector = mutableListOf<BigDecimal>()
            for (equation in equationSystemParams.equations) {
                extendedVector.add(
                    BigDecimal(
                        -
EquationSystemService.calculateFunctionWithMultipleArguments(
                            equation, result
                        )
                    )
                )
            }
            val slaeExtendedMatrix = ExtendedMatrix(slaeMatrix)

slaeExtendedMatrix.setExtendedVector(extendedVector.toTypedArray())

            val slae = SLAE(slaeExtendedMatrix)
            val solution = slae.solveSLAE()
            if (solution.status != SLAESolutionStatus.OK) throw
SLAENotSolvedException(solution.status)
```

```
            val solutionVector = solution.solutionVector
            var nextIterationRequired = false
            for (i in immutableFoundSymbols.indices) {
                if (abs(solutionVector[i].toDouble()) >
equationSystemParams.epsilon) {
                    nextIterationRequired = true
                }
                result[immutableFoundSymbols[i]] =
result[immutableFoundSymbols[i]]!! + solutionVector[i].toDouble()
            }
            flag = nextIterationRequired
            iterations++
        }

        return EquationSystemResult.ok(
            EquationSystemSolvingMethod.NEWTON_METHOD,
            result,
            iterations,
            calculateErrorVector(equationSystemParams.equations, result)
        )
    } catch (e: Throwable) {
        throw IncorrectParametersException()
    }
}
```

GitHub: https://github.com/Zerumi/cmath2_070923_1

Результаты работы программы

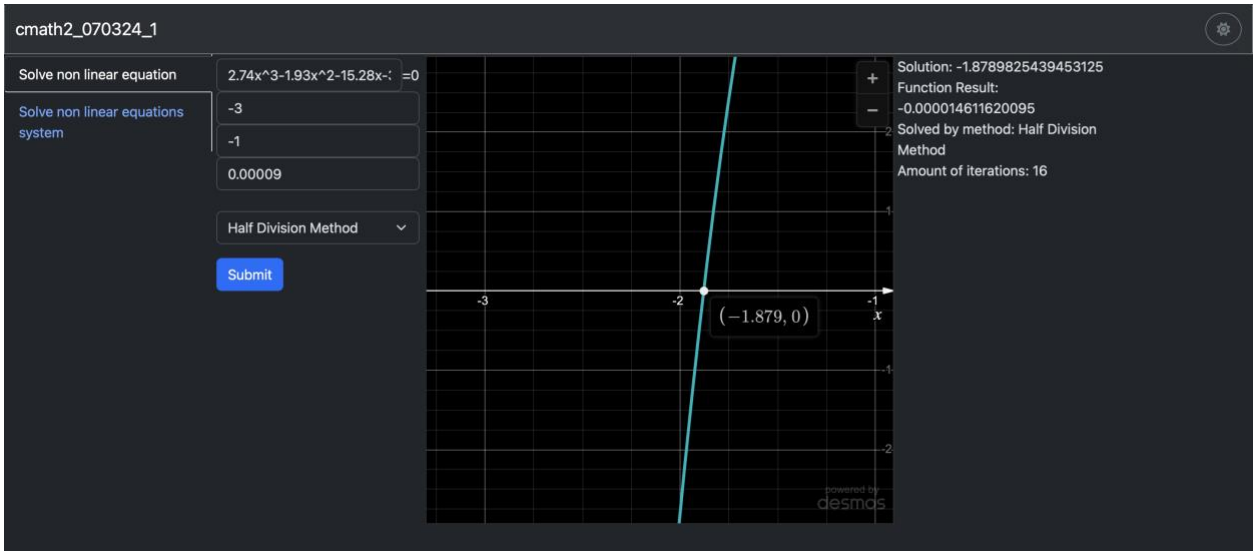

Рисунок 2. Вычисление решения НУ.

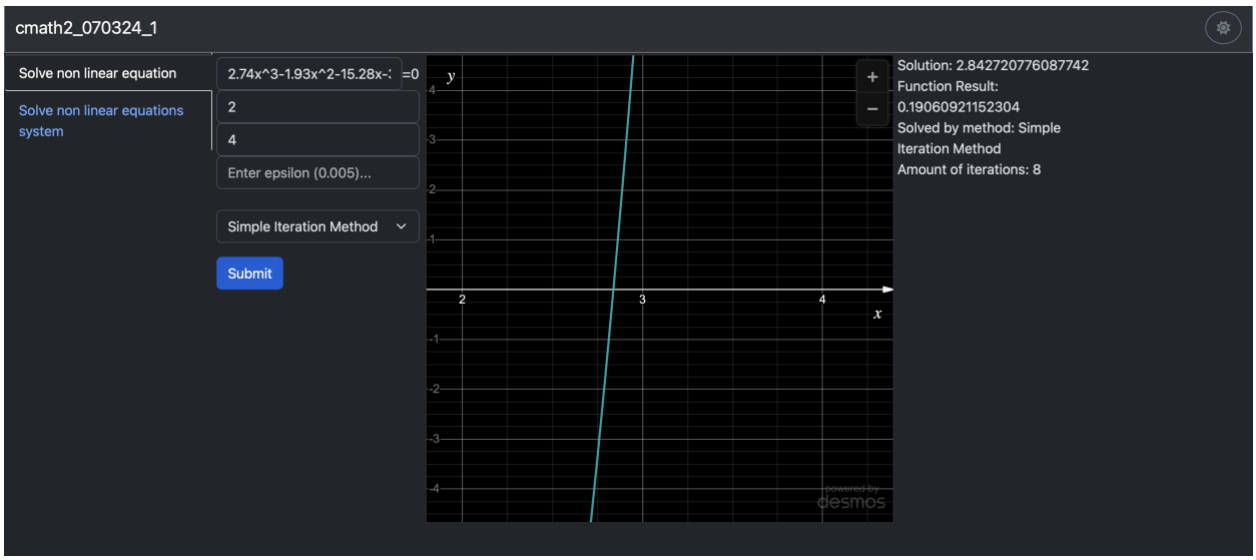

Рисунок 3. Вычисление решения НУ.

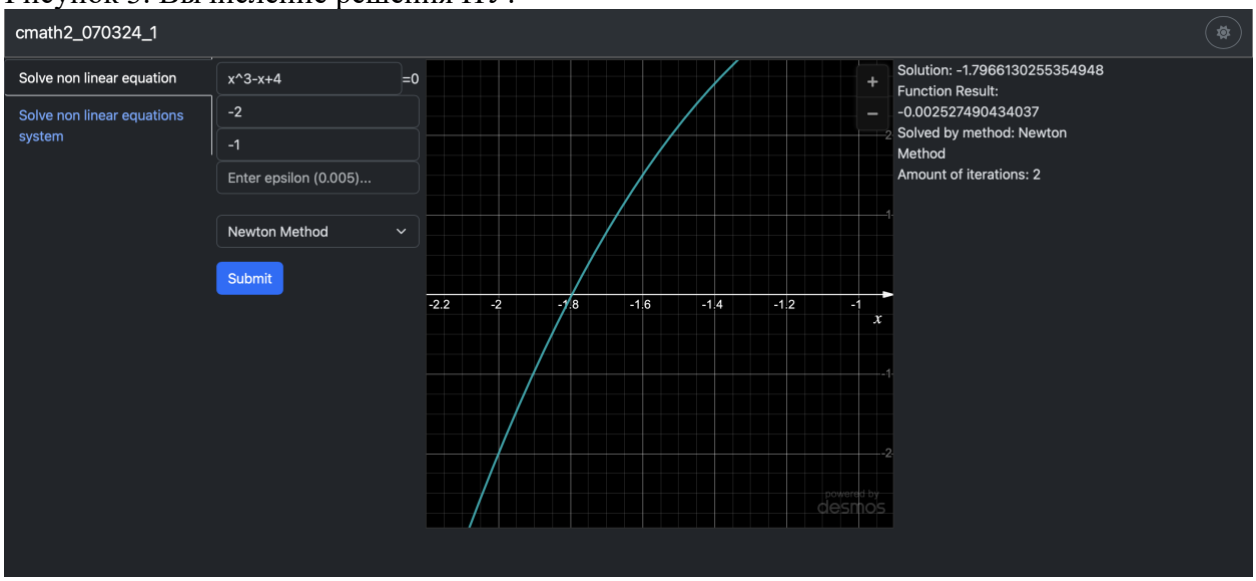

Рисунок 4. Вычисление решения НУ.

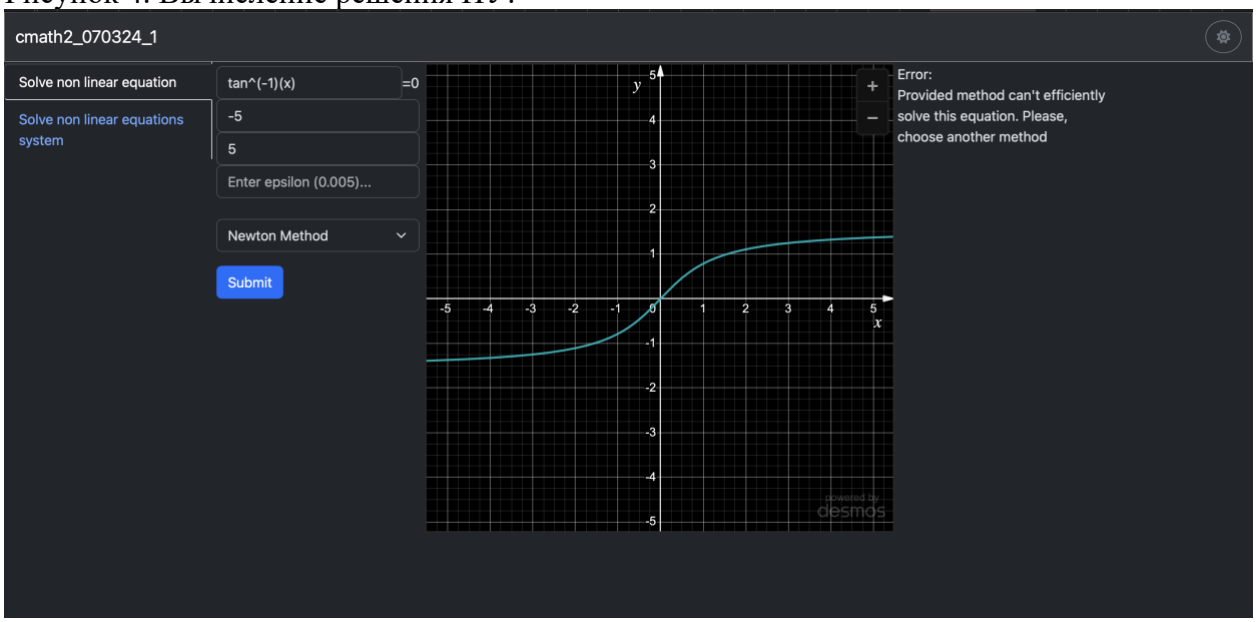

Рисунок 5. Несоблюдение критерия быстрой сходимости метода.

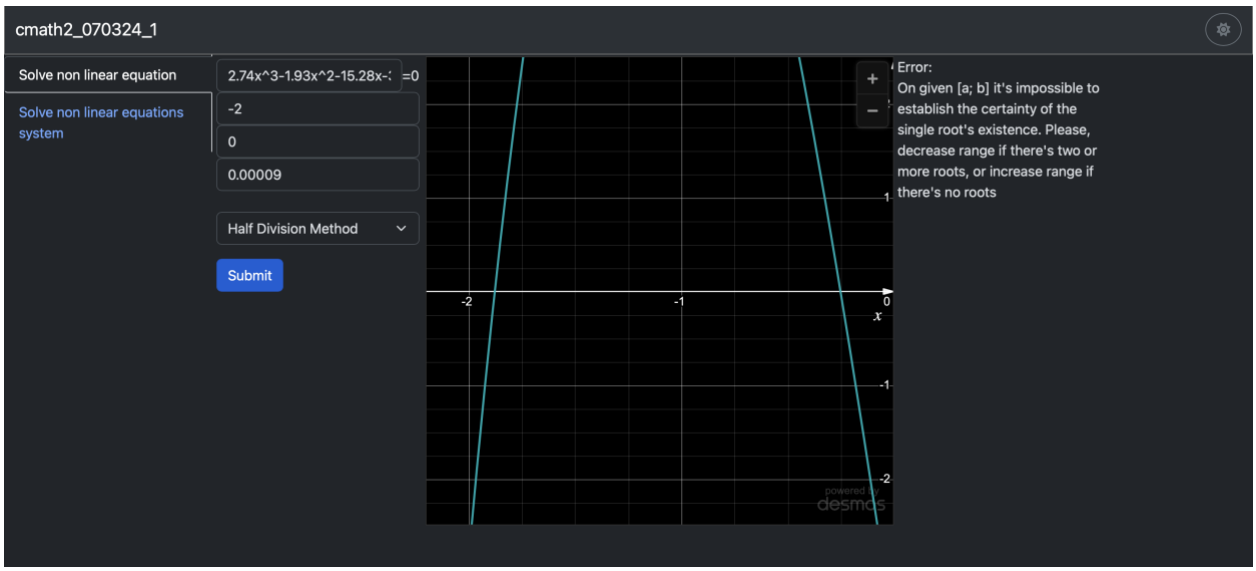

Рисунок 6. Обработка некорректного ввода данных.

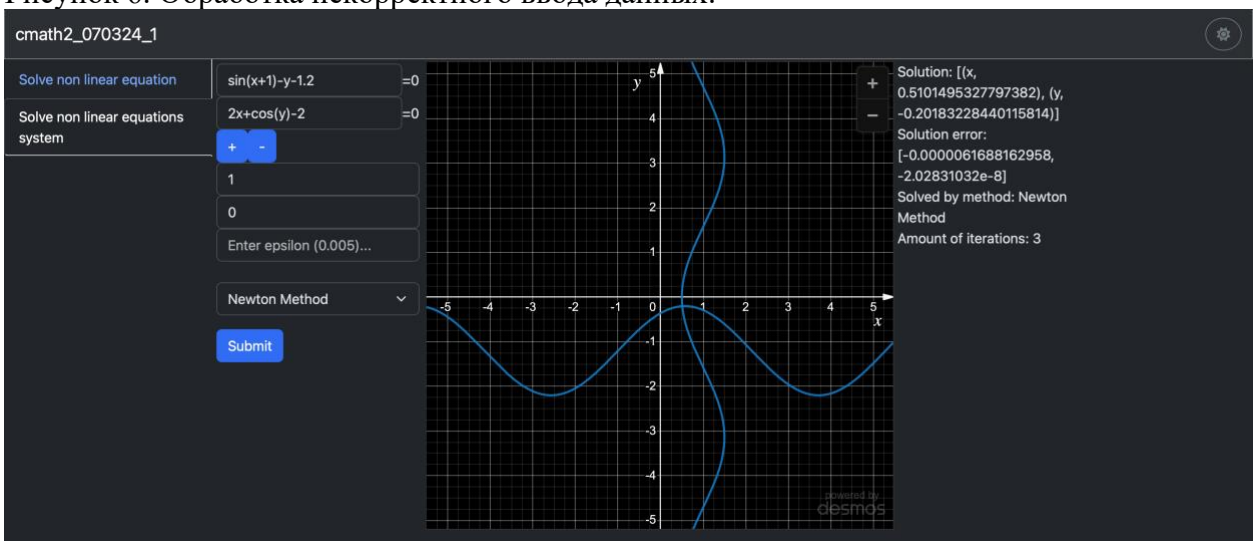

Рисунок 7. Вычисление решения СНУ.

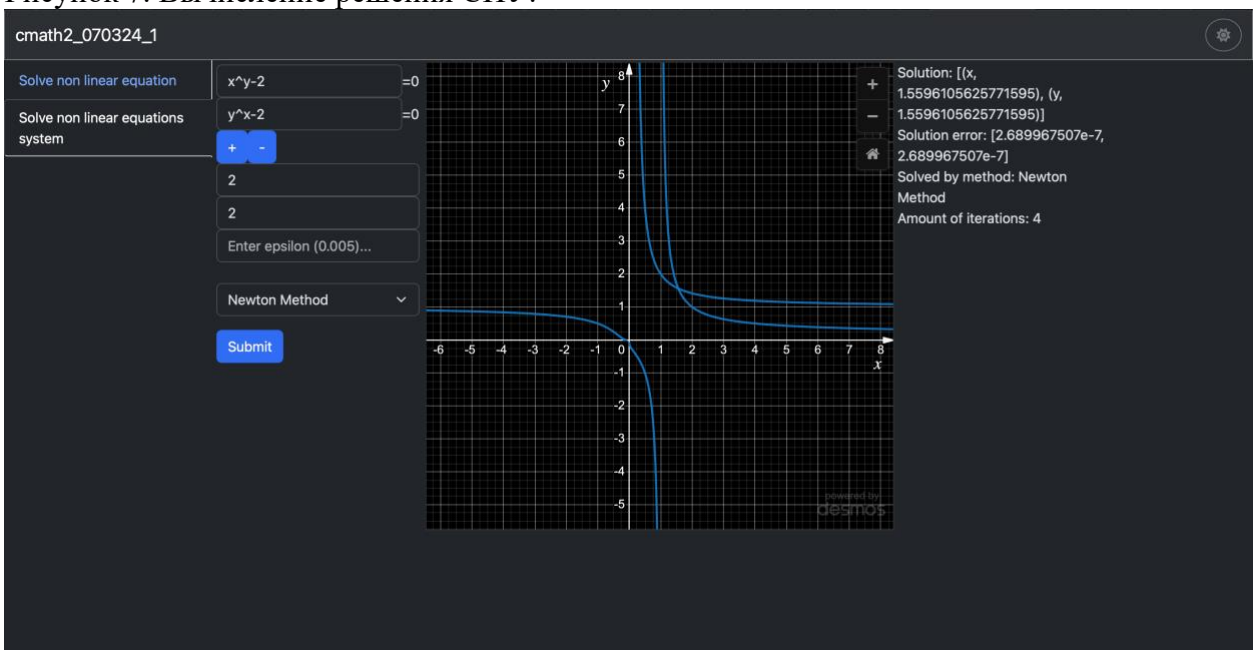

Рисунок 8. Вычисление решения СНУ.

Вывод

Во время выполнения данной лабораторной работы я ознакомился с численными методами решения нелинейных уравнений и систем нелинейных уравнений. С моральной поддержкой плейлиста “dark ambient – music to escape/dream to” (https://open.spotify.com/playlist/07lYUEyTkWP3NqIa7Kzyqx) мною было написано приложение, способное применять данные методы на практике для произвольного ввода распространённых типов НУ и СНУ.