



УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №3(1) - Решение нелинейных уравнений

Методы половинного деления, касательных для одного уравнения; простых итераций для системы уравнений

Преподаватель: Перл Ольга Вячеславовна

Выполнили: Кульбако Артемий Юрьевич Р3212

Описание метода

Метод половинного деления:

На некотором интервале $[a; b]$ вычисляем x_i :

$$x_i = \frac{a_i + b_i}{2}$$

Если $f(x_i) \leq \varepsilon$ - завершаем итерационный процесс, иначе в качестве нового интервала берём ту половину отрезка, на концах которого функция имеет разные знаки: $[a; x_i]$ если $f(a) \cdot f(x_i) < 0$, или $[b; x_i]$ если $f(b) \cdot f(x_i) < 0$, и повторяем вычисления.

Касательных:

На некотором интервале $[a; b]$ функция $y = f(x)$ заменяется касательной, и в качестве приближенного значения принимается точка пересечения касательной с осью абсцисс.

Начальное приближение, обеспечивающее быструю сходимость, считается по формуле:

$$x_0 = \max\{f(a) \cdot f''(a); f(b) \cdot f''(b)\} > 0$$

Находим показатель λ для интервала:

$$\lambda = \frac{f(x_0)}{f'(x_0)}$$

Вычисляем x_i по формуле:

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$$

пока $|f(x_i)| > \varepsilon$.

Простой итерации для СНАУ:

Все уравнения системы

$$\begin{cases} f_1(x_1 \dots x_n) = 0 \\ \dots \\ f_n(x_1 \dots x_n) = 0 \end{cases}$$

Преобразуем к виду:

$$\begin{cases} x_1 = \varphi_1(x_1 \dots x_n) \\ \dots \\ x_n = \varphi_n(x_1 \dots x_n) \end{cases}$$

Если выбрано некоторое начальное приближение

$$x^{(0)} = (x_1^{(0)} \dots x_n^{(0)})$$

, то последующие приближения находятся по формулам:

$$\begin{cases} x_1^{(i+1)} = \varphi_1(x_1^{(i)} \dots x_n^{(i)}) \\ \dots \\ x_n^{(i+1)} = \varphi_n(x_1^{(i)} \dots x_n^{(i)}) \end{cases}$$

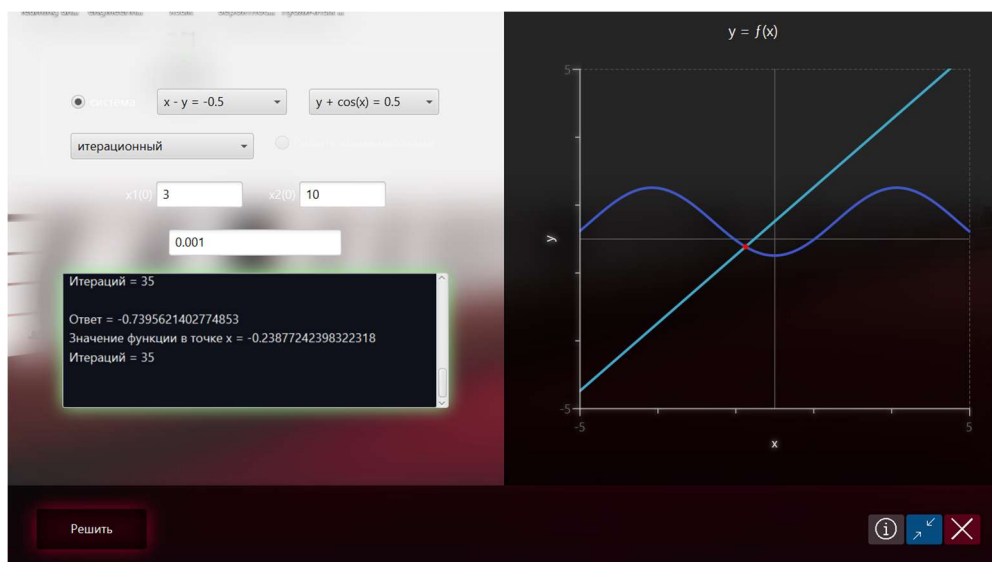
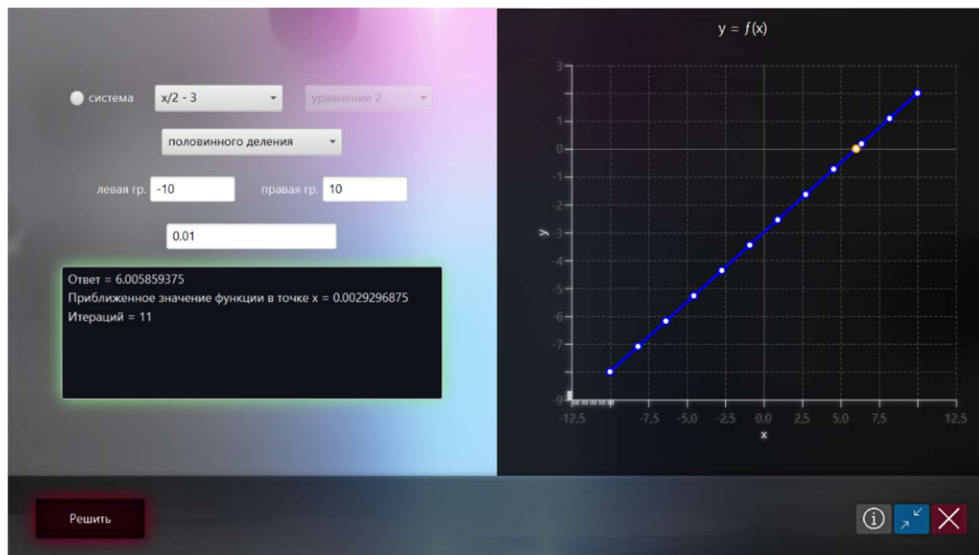
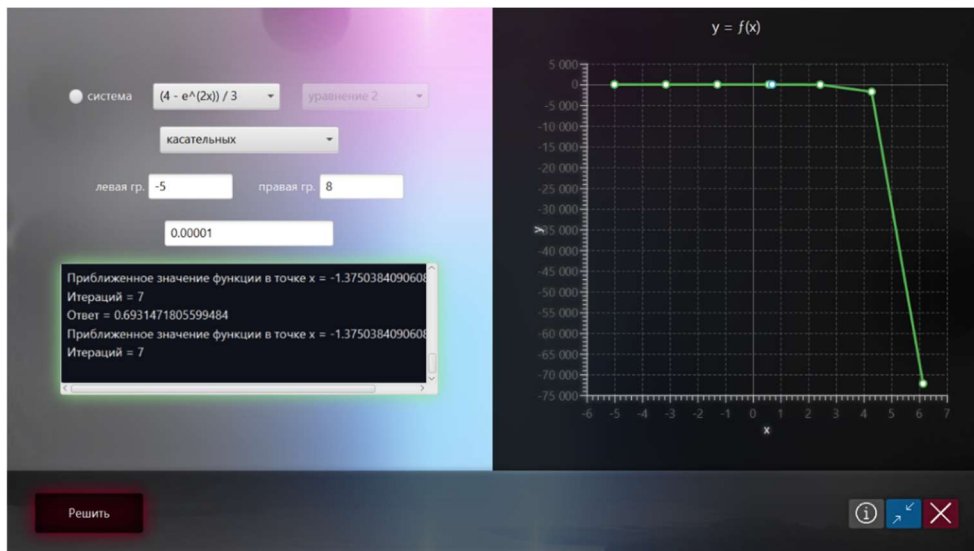
Находить новые приближения до тех пор, пока

$$\Delta^i = \max |x_n^i - x_n^{i-1}| \leq \varepsilon$$

Вывод

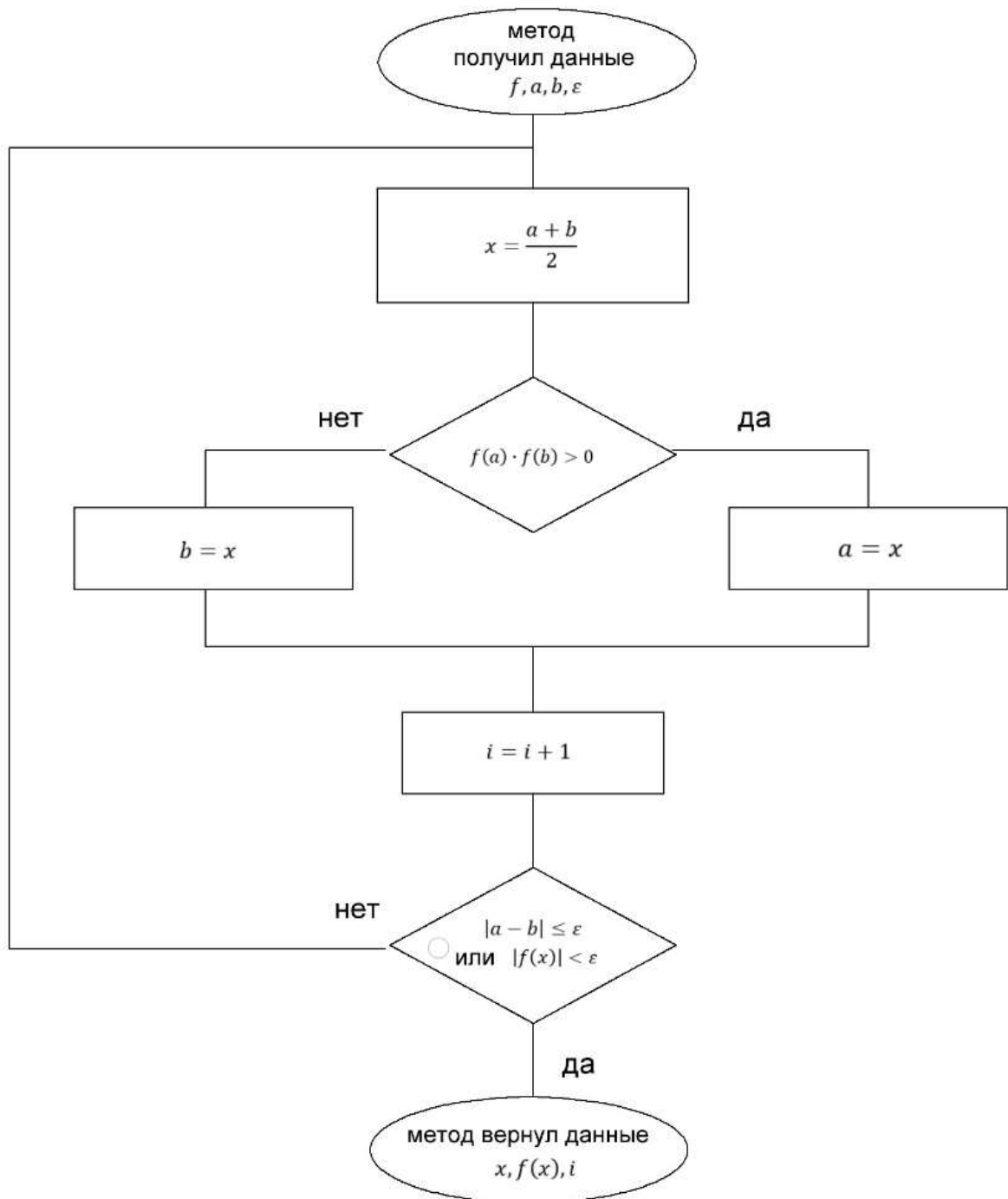
+	-
Метод половинного деления	
<ul style="list-style-type: none"> • Обладает абсолютной сходимостью (близость получаемого численного решения задачи к истинному решению) • устойчив к ошибкам округления 	<ul style="list-style-type: none"> • линейная сходимость
Метод хорд	
<ul style="list-style-type: none"> • быстрая сходимость при $f(x) \cdot f''(x) > 0$ 	<ul style="list-style-type: none"> • линейная сходимость • выбор начального приближения
Касательных (Ньютона)	
<ul style="list-style-type: none"> • квадратичная сходимость 	<ul style="list-style-type: none"> • вычисления производных • выбор начального приближения
Простых итерации	
<ul style="list-style-type: none"> • сходимость со скоростью геометрической прогрессии если в окрестности корня $0 \leq \varphi'(x) \leq 1$ и $\varphi'(x) = \text{const}$ 	<ul style="list-style-type: none"> • вычисление производных • выбор начального приближения • очень медленная сходимость при $\varphi'(x) \approx 1$
Простой итерации для СНАУ	
<ul style="list-style-type: none"> • чуть проще чем Ньютона для СНАУ 	<ul style="list-style-type: none"> • крайне сложная реализация из-за требования преобразовывать уравнения системы • вычисление производных • очень медленная сходимость при $\varphi'(x) \approx 1$
Метод касательных (Ньютона) для СНАУ	
<ul style="list-style-type: none"> • быстрее простой итерации для СНАУ 	<ul style="list-style-type: none"> • крайне сложная реализация из-за матрицы Якоби • вычисление производных

Примеры

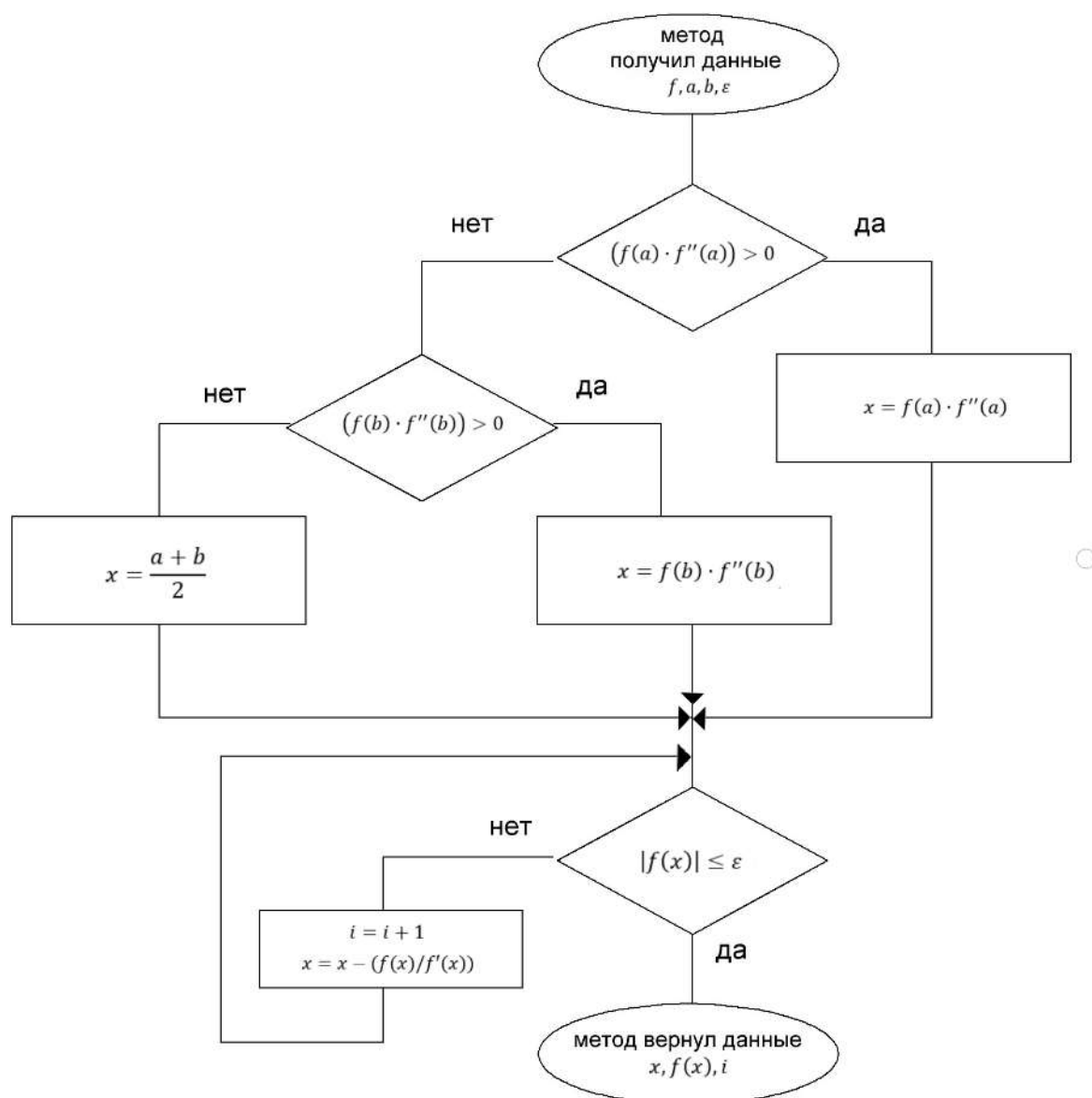


Блок-схемы

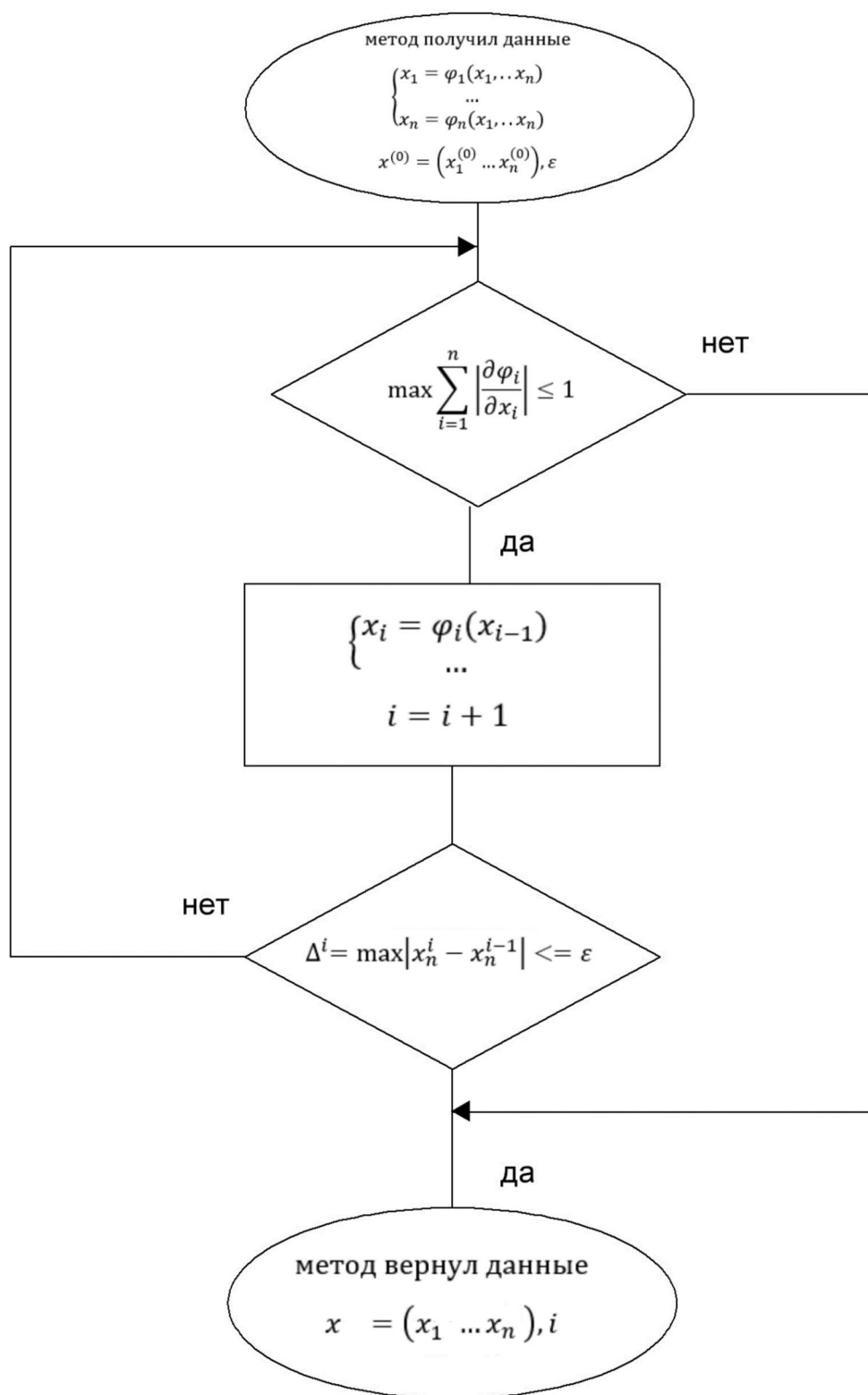
Метод половинного деления:



Метод касательных:



Метод простой итерации:



```

1 package math
2
3 import kotlin.math.*
4
5 /**
6  * Предоставляет методы решения нелинейных уравнений.
7  * @property MAX_ITERS максимальное число итераций.
8  * @author Артемий Кульбако.
9  * @version 1.6
10 */
11 internal class NonLinearEquationSolver {
12
13     var MAX_ITERS = 10_000_000
14
15     /**
16      * Решает нелинейное уравнение методом половинного деления.
17      * @param f функция, значение которой необходимо вычислить.
18      * @param borders интервал, на котором ищется корень.
19      * @param accuracy точность вычислений.
20      * @return результат вычислений.
21      */
22     internal fun bisectionMethod(f: MathFunction<Double>, borders: Pair<Double, Double>, accuracy: Double): NonLinearEquationAnswer {
23         var left = borders.first
24         var right = borders.second
25         var x: Double
26         var xFuncValue: Double
27         var i = 0
28         do {
29             i++
30             x = (left + right) / 2
31             val leftFuncValue = f.func(left)
32             xFuncValue = f.func(x)
33             if (leftFuncValue * xFuncValue > 0) left = x else right = x
34         } while (((right - left) > accuracy || abs(xFuncValue) > accuracy) && i < MAX_ITERS)
35         return NonLinearEquationAnswer(Pair(x, xFuncValue), i, i == MAX_ITERS)
36     }
37
38     /**
39      * Решает нелинейное уравнение методом касательных.
40      * @param f функция, значение которой необходимо вычислить.
41      * @param borders интервал, на котором ищется корень.
42      * @param accuracy точность вычислений.
43      * @return результат вычислений.
44      */
45     internal fun tangentsMethod(f: MathFunction<Double>, borders: Pair<Double, Double>, accuracy: Double): NonLinearEquationAnswer {
46
47         val firstApproach = {it: Double -> f.func(it) * f.findDerivative(it, 2) }
48         val left = firstApproach(borders.first)
49         val right = firstApproach(borders.second)
50         var x = borders.toList().max()!!.let { if (it > 0 ) it else (left + right) / 2 }
51         var xFuncValue: Double
52         var i = 0
53         do {
54             i++
55             xFuncValue = f.func(x)
56             val dX = f.findDerivative(x, 1)
57             x -= xFuncValue / dX
58         } while ((abs(xFuncValue) > accuracy) && i < MAX_ITERS)
59         return NonLinearEquationAnswer(Pair(x, xFuncValue), i, i == MAX_ITERS)
60     }
61
62     /**
63      * Решает систему нелинейных уравнение методом простых итераций. Может решить 1 или 2 уравнения.
64      * @param system система функций, значения которых необходимо вычислить.
65      * @param borders начальные приближения.
66      * @param accuracy точность вычислений.
67      * @return результат вычислений.
68      * @throws IllegalArgumentException если количество уравнений в системе меньше 1 или больше 2.
69      * @throws Exception если не выполняется условие сходимости метода.
70      */
71     internal fun iterativeMethod(system: List<MathFunction<Double>>, borders: Pair<Double, Double>, accuracy: Double): NonLinearEquationAnswer {
72
73         fun isAccuracyAchieve(oldX: DoubleArray, newX: DoubleArray) =
74             oldX.zip(newX) { x, y -> abs(x - y) }.toDoubleArray().max()!! >= accuracy
75
76         var i = 0
77         when (system.size) {
78

```



```

79         1 -> {
80             val derA = system[0].findDerivative(borders.first, 1)
81             val derB = system[0].findDerivative(borders.second, 1)
82             val maxDer = sequenceOf(derA, derB).max()!!
83             if (maxDer >= 1) throw Exception("Не выполняется условие сходимости метода")
84             var x = maxDer
85             val lambda = -1 / maxDer
86             do {
87                 i++
88                 val previousX = x
89                 x += lambda * system[0].func(x)
90             } while (abs(x - previousX) >= accuracy && i < MAX_ITERS)
91             return NonLinearEquationAnswer(Pair(x, system[0].func(x)), i, i == MAX_ITERS)
92         }
93         2 -> {
94             var prevX: DoubleArray
95             var newX = doubleArrayOf(borders.first, borders.second)
96             do {
97                 i++
98                 prevX = newX.clone()
99                 newX = doubleArrayOf(system[0].func(prevX[1]), system[1].func(prevX[0]))
100            } while (isAccuracyAchieve(prevX, newX) && i < MAX_ITERS)
101            return NonLinearEquationAnswer(Pair(newX[0], newX[1]), i, i == MAX_ITERS)
102        }
103    } else -> throw IllegalArgumentException("Решение систем для более чем двух пока невозможно")
104    }
105 }
106
107 /**
108  * Вычисляет производную функции.
109  * @param x точка дифференцирования.
110  * @param order порядок производной.
111  * @return результат дифференцирования в точке x.
112  * @throws IllegalArgumentException если порядок производной меньше 1 или больше 2.
113  */
114 internal fun MathFunction<Double>.findDerivative(x: Double, order: Int): Double {
115     val h = 0.0001
116     return when (order) {
117         1 -> (this.func(x + h) - this.func(x - h)) / (2 * h)
118         2 -> (this.func(x + h) - 2 * this.func(x) + this.func(x - h)) / h.pow(2)
119         else -> throw Exception("Метод расчёта производных этого порядка не реализован")
120     }
121 }
122
123 override fun equals(other: Any?): Boolean {
124     if (this === other) return true
125     if (other !is NonLinearEquationSolver) return false
126     if (MAX_ITERS != other.MAX_ITERS) return false
127     return true
128 }
129
130 override fun hashCode() = MAX_ITERS
131
132 override fun toString() = "${this.javaClass.name}(MAX_ITERS = $MAX_ITERS)"
133 }
134
135 /**
136  * Содержит результат решения системы нелинейных уравнений.
137  * @property root координаты x, y корня системы.
138  * @property iterCounter количество итераций выполненных в процессе нахождения корня.
139  * @property isCalcLimitReached показывает, был ли достигнут максимальный лимит итераций. По-умолчанию
140  * = false.
141  * @author Артемий Кульбако.
142  * @version 1.3
143  */
144 data class NonLinearEquationAnswer(val root: Pair<Double, Double>, val iterCounter: Int, val
145 isCalcLimitReached: Boolean = false) {
146     override fun toString() = ""
147         Ответ = ${root.first}
148         Значение функции в точке x = ${root.second}
149         Итераций = $iterCounter
150         """.trimIndent().plus(if (isCalcLimitReached) "\nБыл достигнут лимит вычислений" else "")
151     }

```