```kotlin
package math

import kotlin.math.*

/**
 * Предоставляет методы решения нелинейных уравнений.
 * @property MAX_ITERS максимальное число итераций.
 * @author Артемий Кульбако.
 * @version 1.6
 */
internal class NonLinearEquationSolver {

    var MAX_ITERS = 10_000_000

    /**
     * Решает нелинейное уравнение методом половинного деления.
     * @param f функция, значение которой необходимо вычислить.
     * @param borders интервал, на котором ищется корень.
     * @param accuracy точность вычислений.
     * @return результат вычислений.
     */
    internal fun bisectionMethod(f: MathFunction<Double>, borders: Pair<Double, Double>, accuracy:
Double): NonLinearEquationAnswer {
        var left = borders.first
        var right = borders.second
        var x: Double
        var xFuncValue: Double
        var i = 0
        do {
            i++
            x = (left + right) / 2
            val leftFuncValue = f.func(left)
            xFuncValue = f.func(x)
            if (leftFuncValue * xFuncValue > 0) left = x else right = x
        } while (((right - left) > accuracy || abs(xFuncValue) > accuracy) && i < MAX_ITERS)
        return NonLinearEquationAnswer(Pair(x, xFuncValue), i, i == MAX_ITERS)
    }

    /**
     * Решает нелинейное уравнение методом касательных.
     * @param f функция, значение которой необходимо вычислить.
     * @param borders интервал, на котором ищется корень.
     * @param accuracy точность вычислений.
     * @return результат вычислений.
     */
    internal fun tangentsMethod(f: MathFunction<Double>, borders: Pair<Double, Double>, accuracy: Double
): NonLinearEquationAnswer {

        val firstApproach = {it: Double -> f.func(it) * f.findDerivative(it, 2) }
        val left = firstApproach(borders.first)
        val right = firstApproach(borders.second)
        var x = borders.toList().max()!!.let { if (it > 0 ) it else (left + right) / 2 }
        var xFuncValue: Double
        var i = 0
        do {
            i++
            xFuncValue = f.func(x)
            val dX = f.findDerivative(x, 1)
            x -= xFuncValue / dX
        } while ((abs(xFuncValue) > accuracy) && i < MAX_ITERS)
        return NonLinearEquationAnswer(Pair(x, xFuncValue), i, i == MAX_ITERS)
    }

    /**
     * Решает систему нелинейных уравнение методом простых итераций. Может решить 1 или 2 уравнения.
     * @param system система функций, значения которых необходимо вычислить.
     * @param borders начальные приближения.
     * @param accuracy точность вычислений.
     * @return результат вычислений.
     * @throws IllegalArgumentException если количество уравнений в системе меньше 1 или больше 2.
     * @throws Exception если не выполняется условие сходимости метода.
     */
    internal fun iterativeMethod(system: List<MathFunction<Double>>, borders: Pair<Double, Double>,
                                 accuracy: Double): NonLinearEquationAnswer {

        fun isAccuracyAchieve(oldX: DoubleArray, newX: DoubleArray) =
            oldX.zip(newX) { x, y -> abs(x - y) }.toDoubleArray().max()!! >= accuracy

        var i = 0
        when (system.size) {
```

```
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
    = false.
140  * @author Артемий Кульбако.
141  * @version 1.3
142  */
143 data class NonLinearEquationAnswer(val root: Pair<Double, Double>, val iterCounter: Int, val
    isCalcLimitReached: Boolean = false) {
144
145     override fun toString() = """
146         Ответ = ${root.first}
147         Значение функции в точке x = ${root.second}
148         Итераций = $iterCounter
149         """.trimIndent().plus(if (isCalcLimitReached) "\nБЫЛ ДОСТИГНУТ ЛИМИТ ВЫЧИСЛЕНИЙ" else "")
150 }
```