

```

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 = 100_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, 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, borders: Pair<Double, Double>, accuracy: Double):
NonLinearEquationAnswer {
46
47         val firstApproach = {it: Double -> f.func(it) * findDerivative(f, 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 = findDerivative(f, 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 IllegalCallerException 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>, borders: Pair<Double, Double>,
accuracy: Double): NonLinearEquationAnswer {
72
73
74         fun isAccuracyAchieve(oldX: DoubleArray, newX: DoubleArray) =
75             oldX.zip(newX) { x, y -> abs(x - y) }.toDoubleArray().max()!! >= accuracy
76
77         var i = 0
78         when (system.size) {

```

```

79         1 -> {
80             val derA = findDerivative(system[0], borders.first, 1)
81             val derB = findDerivative(system[0], borders.second, 1)
82             val maxDer = sequenceOf(derA, derB).max()!!.let {
83                 if (it < 1) it else throw Exception("Не выполняется условие сходимости метода")
84             }
85             var x = maxDer
86             val lambda = -1 / maxDer
87             do {
88                 i++
89                 val previousX = x
90                 x += lambda * system[0].func(x)
91             } while (abs(x - previousX) >= accuracy && i < MAX_ITERS)
92             return NonLinearEquationAnswer(Pair(x, system[0].func(x)), i, i == MAX_ITERS)
93         }
94         2 -> {
95             var prevX: DoubleArray
96             var newX = doubleArrayOf(borders.first, borders.second)
97             do {
98                 i++
99                 prevX = newX.clone()
100                newX = doubleArrayOf(system[0].func(prevX[1]), system[1].func(prevX[0]))
101            } while (isAccuracyAchieve(prevX, newX) && i < MAX_ITERS)
102            return NonLinearEquationAnswer(Pair(newX[0], newX[1]), i, i == MAX_ITERS)
103        }
104    } else -> throw IllegalArgumentException("Решение систем для более чем двух пока невозможно")
105    }
106 }
107
108 /**
109  * Вычисляет производную функции.
110  * @param f дифференцируемая функция.
111  * @param x точка дифференцирования.
112  * @param order порядок производной.
113  * @return результат дифференцирования в точке x.
114  * @throws IllegalArgumentException если порядок производной меньше 1 или больше 2.
115  */
116 fun findDerivative(f: MathFunction, x: Double, order: Int): Double {
117     val h = 0.0001
118     return when (order) {
119         1 -> (f.func(x + h) - f.func(x - h)) / (2 * h)
120         2 -> (f.func(x + h) - 2 * f.func(x) + f.func(x - h)) / h.pow(2)
121         else -> throw Exception("Метод расчёта производных этого порядка не реализован")
122     }
123 }
124
125 override fun equals(other: Any?): Boolean {
126     if (this === other) return true
127     if (other !is NonLinearEquationSolver) return false
128     if (MAX_ITERS != other.MAX_ITERS) return false
129     return true
130 }
131
132 override fun hashCode() = MAX_ITERS
133
134 override fun toString() = "${this.javaClass.name}(MAX_ITERS = $MAX_ITERS)"
135 }
136
137 /**
138  * Содержит результат решения системы нелинейных уравнений.
139  * @property root координаты x, y корня системы.
140  * @property iterCounter количество итераций выполненных в процессе нахождения корня.
141  * @property isCalcLimitReached показывает, был ли достигнут максимальный лимит итераций. По-умолчанию
142  * = false.
143  * @author Артемий Кульбако.
144  * @version 1.3
145  */
146 data class NonLinearEquationAnswer(val root: Pair<Double, Double>, val iterCounter: Int, val
147     isCalcLimitReached: Boolean = false) {
148     override fun toString() = ""
149         Ответ = ${root.first}
150         Значение функции в точке x = ${root.second}
151         Итераций = $iterCounter
152         """.trimIndent().plus(if (isCalcLimitReached) "\nБыл достигнут лимит вычислений" else "")
153     }

```