



УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №3(2) – Приближение функций

Метод наименьших квадратов

Преподаватель: Перл Ольга Вячеславовна

Выполнили: Кульбако Артемий Юрьевич Р3212

Описание метода

Задача о приближении (аппроксимации) функции состоит в том, чтобы данную функцию $f(x)$ приближенно заменить (аппроксимировать) некоторой функцией $\varphi(x)$, значения которой в заданной области мало отличались от опытных данных.

Построение аппроксимирующей формулы состоит из 2 этапов:

1. Подбор общего вида формулы. Иногда он известен из физических соображений, если характер зависимости неизвестен, то первоначально его выбирают геометрически: экспериментальные точки наносятся на график, и примерно получается общий вид зависимости путем сравнения полученной кривой с графиками известных функций.
2. Определение значений параметров аппроксимирующей функции.

Мерой отклонения многочлена $\varphi(x)$ от заданной функции $f(x)$ на множестве точек $\{(x_i, y_i)\}$ является величина S (критерий минимизации), равная сумме квадратов разности между значениями многочлена и функции для всех точек $x_0 \dots x_n$:

$$S = \sum_{i=1}^n [\varphi(x_i) - y_i]^2 \rightarrow \min$$

Задача сводится к нахождению неизвестных параметров $a_0 \dots a_n$, являющихся параметрами аппроксимирующей функции $S = S(a_0 \dots a_n)$, минимум которой найдем, приравняв к нулю частные производные по этим переменным:

$$\frac{\partial S}{\partial a_0} = 0; \dots \frac{\partial S}{\partial a_m} = 0$$

Полученные соотношения – система уравнений (отличаются для разных видов аппроксимации), пригодна для нахождения параметров a, b, c и получения финальной функции вида:

$ax + b$ – для линейной аппроксимации

$ax^2 + bx + c$ – для квадратичной аппроксимации

ax^b – для степенной аппроксимации

$a \cdot e^{bx}$

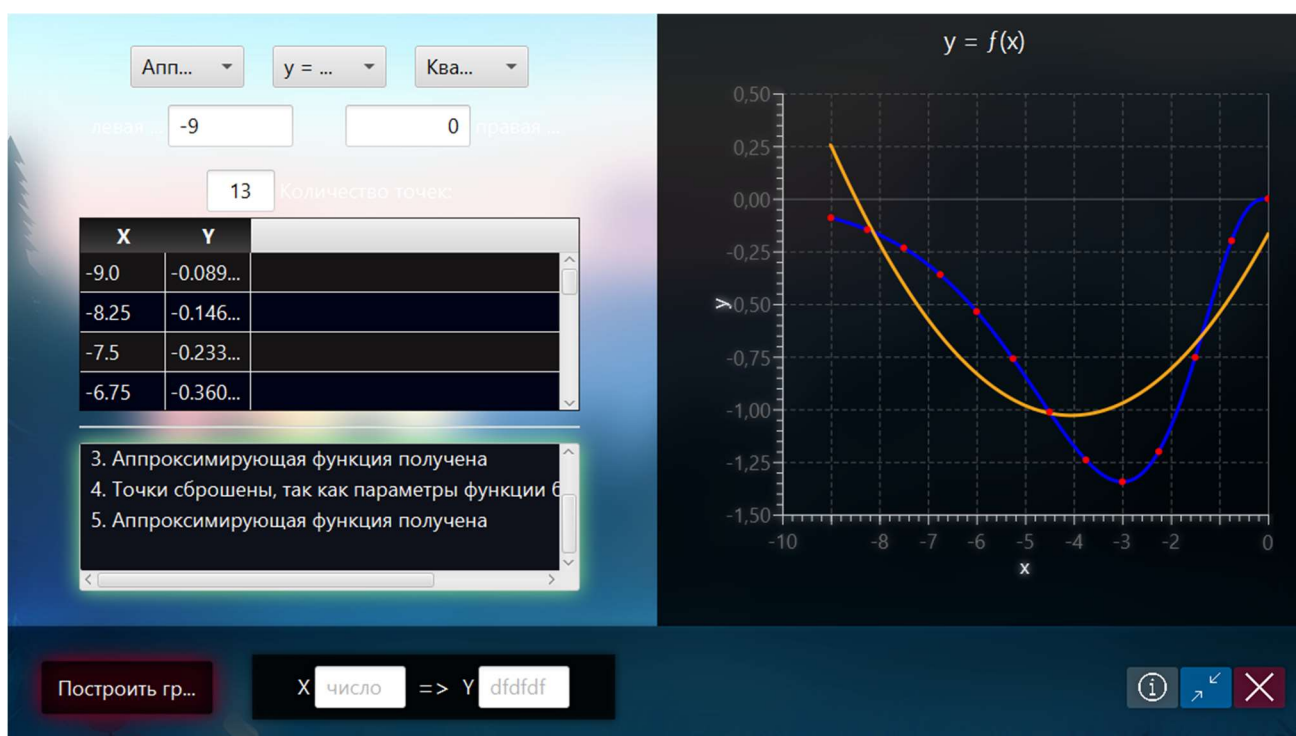
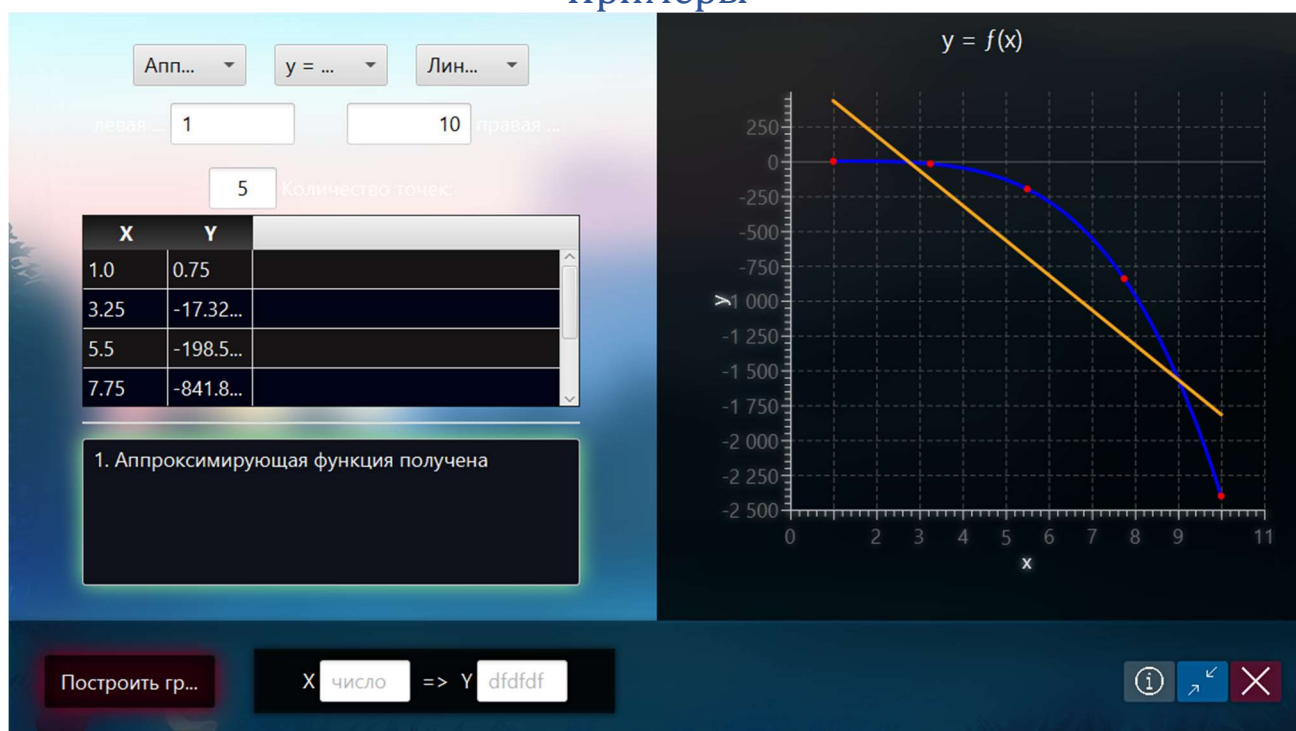
$a \cdot \ln(x) + b$ – для логарифмической аппроксимации

Вывод

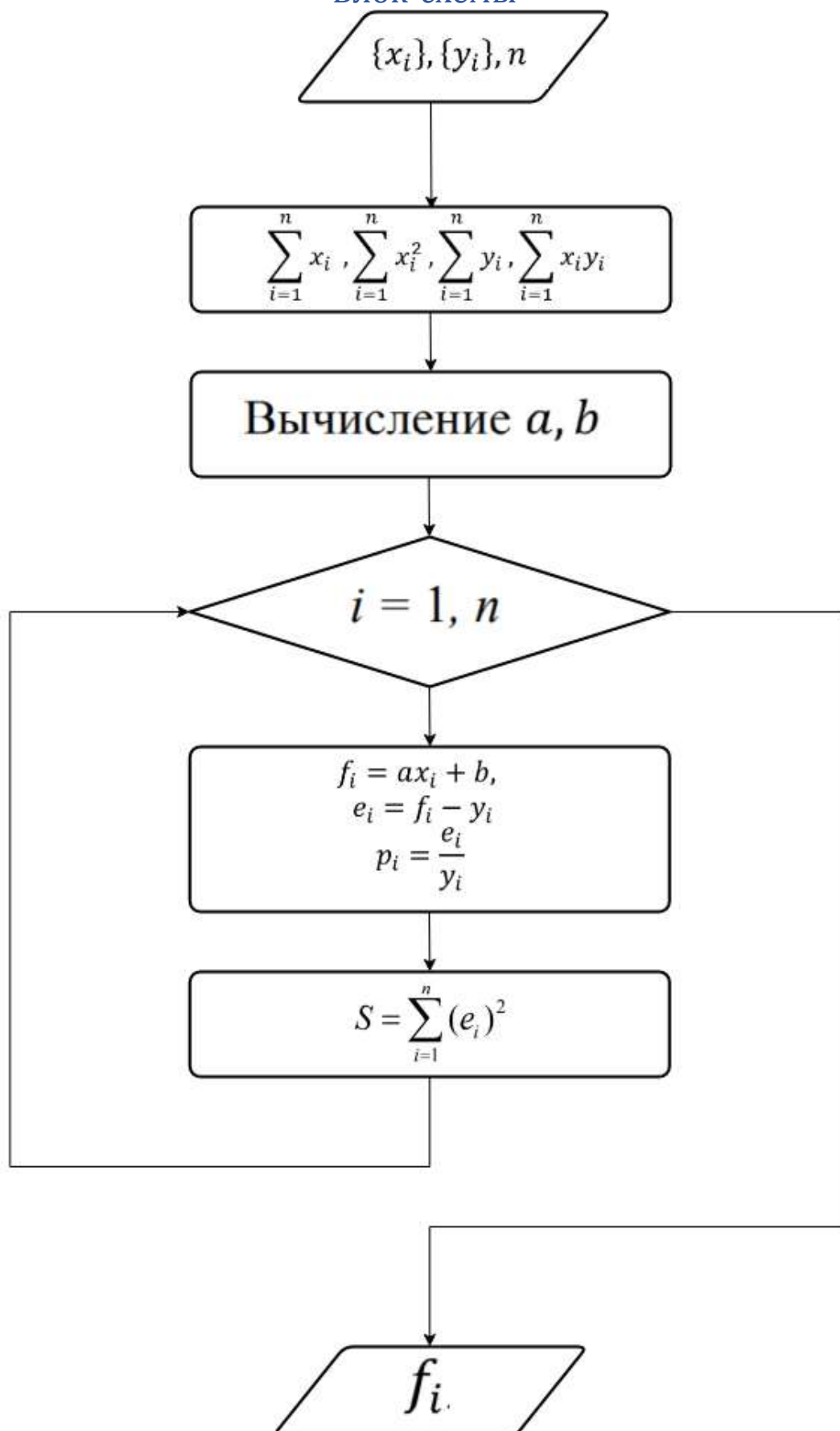
Задача в лабораторной работе – реализовать один из алгоритмов интерполяции или аппроксимации. Основное различие в требовании прохождения полученной функции через заданные узлы – в интерполяции оно должно строго соблюдаться, в аппроксимации нет.

Метод	+	-
Интерполирование многочленом Лагранжа	<ul style="list-style-type: none"> • Малая погрешность при $n < 20$. • Можно применять для таблиц с равноудалёнными узлами. 	<ul style="list-style-type: none"> • С изменением числа узлов, приходится проводить все вычисления заново. • Проявление колебательных свойств при большом числе узлов, из-за увеличения степени полинома.
Интерполирование многочленом Ньютона	<ul style="list-style-type: none"> • При добавлении новой точки отсутствует необходимость пересчитывать все коэффициенты заново. 	<ul style="list-style-type: none"> • Только для таблиц с равноудалёнными узлами.
Интерполирование кубическими сплайнами	<ul style="list-style-type: none"> • Степень многочленов не зависит от числа узлов сетки. 	<ul style="list-style-type: none"> • Функция должна быть непрерывной.
Аппроксимация методом наименьших квадратов	<ul style="list-style-type: none"> • Прост в реализации. • Подходит, когда требования к точности аппроксимирующей функции свободные. 	<ul style="list-style-type: none"> • Значительно расхождение исходной и аппроксимирующей функции для сложным функций. • Подбор общего вида формулы.

Примеры



Блок-схемы



```

1 package math
2
3 import Point
4 import math.InterpolationSolver.LeastSquaresType.*
5 import kotlin.math.E
6 import kotlin.math.ln
7 import kotlin.math.pow
8
9 class InterpolationSolver {
10
11     /**
12      * Возвращает интерполирующую функцию для заданного набора точек методом полинома Ньютона.
13      * @param points набор точек по котором будет вычисляться функция.
14      */
15     internal fun newtonPolynomial(points: List<Point>): MathFunction<Double> {
16         val order = points.size
17         val finiteDiffs = Array(order) { DoubleArray(order) { 0.0 } }
18         finiteDiffs[0] = points.map { it.y }.toDoubleArray() //Δy0 - нулевые конечные разности =
значениям функции
19         for (i in 1 until order)
20             for (j in 0 until order - i)
21                 finiteDiffs[i][j] = finiteDiffs[i - 1][j + 1] - finiteDiffs[i - 1][j]
22         val p0 = points[0]
23         val h = points[1].x - p0.x
24         return MathFunction<Double> { x ->
25             var res = p0.y
26             val q = (x[0]!! - p0.x) / h
27             var product = 1.0
28             for (i in 1 until order) {
29                 product = (product * (q + 1 - i)) / i
30                 res += product * finiteDiffs[i][0]
31             }
32             res
33         }
34     }
35
36     /**
37      * Находит аппроксимирующую функцию для заданного набора точек методом наименьших квадратов.
38      * @property LINEAR линейная аппроксимация.
39      * @property QUADRATIC квадратичная аппроксимация.
40      * @property POW степенная аппроксимация.
41      * @property EXPONENTIAL экспоненциальная аппроксимация.
42      * @property LOGARITHMIC логарифмическая аппроксимация.
43      */
44     enum class LeastSquaresType {
45
46         LINEAR {
47             override fun approximate(points: List<Point>): MathFunction<Double> {
48                 val (a, b) = LeastSquaresType.getLinearApproxCoefs(points)
49                 return MathFunction<Double> { x -> a * x[0] + b }
50             }
51             override fun toString() = "Линейная"
52         },
53
54         QUADRATIC {
55             override fun approximate(points: List<Point>): MathFunction<Double> {
56                 val sumX = points.map { it.x }.sum()
57                 val sumY = points.map { it.y }.sum()
58                 val sumXSquares = points.map { it.x.pow(2) }.sum()
59                 val sumXY = points.map { it.x * it.y }.sum()
60                 val sumXCubes = points.map { it.x.pow(3) }.sum()
61                 val sumXTesseractes = points.map { it.x.pow(4) }.sum()
62                 val sumXSquaresY = points.map { it.x.pow(2) * it.y }.sum()
63                 val n = points.size
64                 val matrix = arrayOf(doubleArrayOf(n.toDouble(), sumX, sumXSquares),
65 sumXTesseractes))
66                 val resVector = doubleArrayOf(sumY, sumXY, sumXSquaresY)
67                 var (a, b, c) = LinearSystemSolver.gaussian(matrix, resVector)
68                 a = c.also { c = a } //пришлось поменять, почему-то в неправильной последовательности
возвращает
69                 return MathFunction<Double> { x -> a * x[0].pow(2) + b * x[0] + c }
70             }
71             override fun toString() = "Квадратичная"
72         },
73
74         POW {
75             override fun approximate(points: List<Point>): MathFunction<Double> {
76                 val (x, y) = points.map { Pair(ln(it.x), ln(it.y)) }.toList().unzip()
77                 val modifiedPoints = x.zip(y) { xi, yi -> Point(xi, yi) }.toList()

```

```

78         val (b, a0) = getLinearApproxCoefs(modifiedPoints)
79         val a = E.pow(a0)
80         return MathFunction<Double> { x -> a * x[0].pow(b) }
81     }
82     override fun toString() = "Степенная"
83 },
84
85 EXPONENTIAL {
86     override fun approximate(points: List<Point>): MathFunction<Double> {
87         val (x, y) = points.map { Pair(it.x, ln(it.y)) }.toList().unzip()
88         val modifiedPoints = x.zip(y) { xi, yi -> Point(xi, yi) }.toList()
89         val (b, a0) = getLinearApproxCoefs(modifiedPoints)
90         val a = E.pow(a0)
91         return MathFunction<Double> { x -> a * E.pow(b * x[0]) }
92     }
93     override fun toString() = "Экспоненциальная"
94 },
95
96 LOGARITHMIC {
97     override fun approximate(points: List<Point>): MathFunction<Double> {
98         val (x, y) = points.map { Pair(ln(it.x), it.y) }.toList().unzip()
99         val modifiedPoints = x.zip(y) { xi, yi -> Point(xi, yi) }.toList()
100        val (a, b) = getLinearApproxCoefs(modifiedPoints)
101        return MathFunction<Double> { x-> a * ln(x[0]) + b }
102    }
103    override fun toString() = "Логарифмическая"
104 };
105
106 private companion object {
107     fun getLinearApproxCoefs(points: List<Point>): Pair<Double, Double> {
108         val sumX = points.map { it.x }.sum()
109         val sumY = points.map { it.y }.sum()
110         val sumXSquares = points.map { it.x.pow(2) }.sum()
111         val sumXY = points.map { it.x * it.y }.sum()
112         val n = points.size
113         val matrix = arrayOf(doubleArrayOf(sumXSquares, sumX), doubleArrayOf(sumX, n.toDouble
114     )))
115         val resVector = doubleArrayOf(sumXY, sumY)
116         val (a, b) = LinearSystemSolver.gaussian(matrix, resVector)
117         return Pair(a, b)
118     }
119
120 /**
121  * @return аппроксимирующую функцию для набора точек [points].
122  */
123 abstract fun approximate(points: List<Point>): MathFunction<Double>
124 }
125 }

```