Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Отчет по лабораторной работе № 3 «Аппроксимация методом наименьших квадратов. Двупараметрические модели»

по курсу
«Численные методы»

Выполнил:

Студент группы ИУ9-62Б

Караник А.А.

Проверила:

Домрачева А.Б.

2024 г.

Цель работы

Целью данной работы является реализация программы, использующая метод наименьших квадратов для решения задачи аппроксимации.

Постановка задачи

1) Построить графики таблично заданной функции

x	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
y	3.21	2.95	4.06	4.03	5.39	5.97	6.51	6.77	7.79

и функции $z(x)$.

2) Найти значения x_a, x_g, x_h, y_a, y_g, y_h, $z(x_a)$, $z(x_g)$, $z(x_h)$, δ_1, …, δ_9, k такое, что $\delta_k = \min(\delta_i)$.

3) Составить систему уравнений для определения коэффициентов a и b и решить ее.

4) Найти среднеквадратичное отклонение.

Теоретические сведения

$x_a = \frac{x_0+x_n}{2}$ – среднее арифметическое двух чисел;

$x_g = \sqrt{x_0 x_n}$– среднее геометрическое двух чисел;

$x_h = \frac{2}{\frac{1}{x_0}+\frac{1}{x_n}}$ – среднее гармоническое двух чисел;

$z_1 = ax + b$; $z_2 = ax^b$; $z_3 = ae^{bx}$;

$z_4 = alnx + b$; $z_5 = \frac{a}{x} + b$; $z_6 = \frac{1}{ax+b}$;

$z_7 = \frac{x}{ax+b}$; $z_8 = ae^{\frac{b}{x}}$; $z_9 = \frac{1}{alnx+b}$;

$\delta_1 = |z(x_a) - y_a|$; $\delta_2 = |z(x_g) - y_g|$; $\delta_3 = |z(x_a) - y_g|$;

$\delta_4 = |z(x_g) - y_a|$; $\delta_5 = |z(x_h) - y_a|$; $\delta_6 = |z(x_a) - y_h|$;

$\delta_7 = |z(x_h) - y_h|$; $\delta_8 = |z(x_h) - y_g|$; $\delta_9 = |z(x_g) - y_h|$;

$$\text{СКУ} = \sqrt{\sum_{k=0}^{n} (z(x_k) - y_k)^2}$$

$$\text{СКО} = \frac{\text{СКУ}}{\sqrt{n}}$$

Реализация

```swift
/**
Laboratory work: 3
Discipline: Numerical methods
Copyright (c) 2024, Andrey Karanik
*/

import Foundation

extension Float {
    func pow(_ power: Int) -> Float {
        var calculatedValue: Float = 1.0
        if power >= 0 {
            for _ in 0..<power {
                calculatedValue *= self
            }
        } else {
            for _ in 0..<abs(power) {
                calculatedValue /= self
            }
        }
        return calculatedValue
    }
}

extension Array where Element == Float {

    func pairs() -> [(Float, Float)] {
        (1..<count).map { (self[$0 - 1], self[$0]) }
    }

    func sum() -> Float {
        reduce(into: 0) { $0 += $1 }
    }

    var minAll: (element: Float, indexes: [Int]) {
        var min: (element: Float, indexes: [Int]) = (self[0], [0])
        for i in 1..<count {
            if self[i] < min.element {
                min = (self[i], [i])

            } else if self[i] == min.element {
                min.indexes.append(i)
            }
        }
        return min
    }
}

func solve(matrixA: [[Float]], vectorB: [Float], size: Int) -> [Float] {
    var matrix: [[Float]] = .init(repeating: .init(repeating: .zero, count: size), count: size)

    for i in 0..<size {
        for j in 0...i {
            if i == j {
                let sum = (0..<j).map { matrix[j][$0].pow(2) }.sum()
                matrix[j][j] = sqrtf(matrixA[j][j] - sum)
            } else {
                let sum = (0..<j).map { matrix[i][$0] * matrix[j][$0] }.sum()
                matrix[i][j] = (matrixA[i][j] - sum) / matrix[j][j]
            }
        }
    }

    var y: [Float] = .init(repeating: .zero, count: size)
    for i in 0..<size {
        let sum = (0..<i).map { matrix[i][$0] * y[$0] }.sum()
        y[i] = (vectorB[i] - sum) / matrix[i][i]
    }

    var x: [Float] = .init(repeating: .zero, count: size)
    for i in stride(from: size - 1, through: 0, by: -1) {
        let sum = (i+1..<size).map { matrix[$0][i] * x[$0] }.sum()
        x[i] = (y[i] - sum) / matrix[i][i]
    }

    return x
}

enum ApproximateType: CaseIterable {
    case z1, z2, z3, z4, z5, z6, z7, z8, z9

    var function: (Float, Float, Float) -> Float {
```

```
        switch self {
            case .z1: return { a, b, x in a * x + b }
            case .z2: return { a, b, x in a * powf(x, b) }
            case .z3: return { a, b, x in a * expf(b * x) }
            case .z4: return { a, b, x in a * logf(x) + b }
            case .z5: return { a, b, x in a / x + b }
            case .z6: return { a, b, x in 1 / (a * x + b) }
            case .z7: return { a, b, x in x / (a * x + b) }
            case .z8: return { a, b, x in a * expf(b / x) }
            case .z9: return { a, b, x in 1 / (a * logf(x) + b) }
        }
    }

    static func get(type: Int) -> ApproximateType { allCases[type] }
}

class Polynom {

    private let coefs: [Float]

    init(coefs: [Float]) {
        self.coefs = coefs
    }

    func value(by x: Float) -> Float {
        return coefs.enumerated().map { degree, coef in
            coef * x.pow(degree)
        }.sum()
    }
}

class SLAU {

    private let a1: Float
    private let b1: Float
    private let c1: Float
    private let a2: Float
    private let b2: Float
    private let c2: Float
    private let approximateType: ApproximateType

    init(forType type: ApproximateType, arrayX: [Float], arrayY: [Float]) {
        let n: Float = Float(arrayX.count)
        approximateType = type

        switch type {

        case .z1:
            a1 = arrayX.map { $0.pow(2) }.sum()
            b1 = arrayX.sum()
            c1 = arrayX.enumerated().map { index, x in x * arrayY[index] }.sum()
            a2 = b1
            b2 = n
            c2 = arrayY.sum()

        case .z2:
            a1 = n
            b1 = arrayX.map { logf($0) }.sum()
            c1 = arrayY.map { logf($0) }.sum()
            a2 = b1
            b2 = arrayX.map { logf($0).pow(2) }.sum()
            c2 = arrayX.enumerated().map { index, x in logf(x) * logf(arrayY[index]) }.sum()

        case .z3:
            a1 = n
            b1 = arrayX.sum()
            c1 = arrayY.map { logf($0) }.sum()
            a2 = b1
            b2 = arrayX.map { $0.pow(2) }.sum()
            c2 = arrayX.enumerated().map { index, x in x * logf(arrayY[index]) }.sum()

        case .z4:
            a1 = arrayX.map { logf($0).pow(2) }.sum()
            b1 = arrayX.map { logf($0) }.sum()
            c1 = arrayX.enumerated().map { index, x in logf(x) * arrayY[index] }.sum()
            a2 = b1
            b2 = n
            c2 = arrayY.sum()

        case .z5:
            a1 = arrayX.map { (1 / $0).pow(2) }.sum()
            b1 = arrayX.map { 1 / $0 }.sum()
            c1 = arrayX.enumerated().map { index, x in (1 / x) * arrayY[index] }.sum()
            a2 = b1
            b2 = n
            c2 = arrayY.sum()

        case .z6:
            a1 = arrayX.map { $0.pow(2) }.sum()
            b1 = arrayX.sum()
            c1 = arrayX.enumerated().map { index, x in x * (1 / arrayY[index]) }.sum()
```

```
            a2 = b1
            b2 = n
            c2 = arrayY.map { 1 / $0 }.sum()

        case .z7:
            a1 = n
            b1 = arrayX.map { 1 / $0 }.sum()
            c1 = arrayY.map { 1 / $0 }.sum()
            a2 = b1
            b2 = arrayX.map { (1 / $0).pow(2) }.sum()
            c2 = arrayX.enumerated().map { index, x in (1 / x) * (1 / arrayY[index]) }.sum()

        case .z8:
            a1 = n
            b1 = arrayX.map { 1 / $0 }.sum()
            c1 = arrayY.map { logf($0) }.sum()
            a2 = b1
            b2 = arrayX.map { (1 / $0).pow(2) }.sum()
            c2 = arrayX.enumerated().map { index, x in (1 / x) * logf(arrayY[index]) }.sum()

        case .z9:
            a1 = arrayX.map { logf($0).pow(2) }.sum()
            b1 = arrayX.map { logf($0) }.sum()
            c1 = arrayX.enumerated().map { index, x in logf(x) * (1 / arrayY[index]) }.sum()
            a2 = b1
            b2 = n
            c2 = arrayY.map { 1 / $0 }.sum()
        }
    }

    func getSolution() -> (alpha: Float, beta: Float) {
        let div: Float = a1 * b2 - a2 * b1
        let alpha: Float = (c1 * b2 - c2 * b1) / div
        let beta: Float = (a1 * c2 - a2 * c1) / div
        switch approximateType {
            case .z1, .z4, .z5, .z6, .z7, .z9: return (alpha, beta)
            case .z2, .z3, .z8: return (expf(alpha), beta)
        }
    }
}

let arrayX: [Float] = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
let arrayY: [Float] = [3.21, 2.95, 4.06, 4.03, 5.39, 5.97, 6.51, 6.77, 7.79]

let n: Int = arrayX.count
let m: Int = 4

var matrixA: [[Float]] = Array(repeating: Array(repeating: .zero, count: m), count: m)
var vectorB: [Float] = Array(repeating: .zero, count: m)

for i in 0..<m {
    vectorB[i] = arrayX.enumerated().map { j, elem in
        arrayY[j] * elem.pow(i + 1)
    }.sum()

    for j in 0..<m {
        matrixA[i][j] = arrayX.map { $0.pow(i + j + 1) }.sum()
    }
}

let s = solve(matrixA: matrixA, vectorB: vectorB, size: m)

let polynom = Polynom(coefs: s)

var variance = sqrtf(arrayX.enumerated().map { index, x in
    (polynom.value(by: x) - arrayY[index]).pow(2)
}.sum())

var inaccuracy = variance / sqrtf(Float(n))
let error = inaccuracy / sqrtf(arrayY.map { $0.pow(2) }.sum())

print("Polynomial: ", s)

let middleX: (arithmetic: Float, geometric: Float, harmonic: Float) = (
    (arrayX[0] + arrayX[n - 1]) / 2,
    sqrtf(arrayX[0] * arrayX[n - 1]),
    2 / (1 / arrayX[0] + 1 / arrayX[n - 1])
)

let middleY: (arithmetic: Float, geometric: Float, harmonic: Float) = (
    (arrayY[0] + arrayY[n - 1]) / 2,
    sqrtf(arrayY[0] * arrayY[n - 1]),
    2 / (1 / arrayY[0] + 1 / arrayY[n - 1])
)

let middleZ: (arithmetic: Float, geometric: Float, harmonic: Float) = (
    polynom.value(by: middleX.arithmetic),
    polynom.value(by: middleX.geometric),
    polynom.value(by: middleX.harmonic)
)
```

```swift
let delta1 = abs(middleZ.arithmetic - middleY.arithmetic)
let delta2 = abs(middleZ.geometric - middleY.geometric)
let delta3 = abs(middleZ.arithmetic - middleY.geometric)
let delta4 = abs(middleZ.geometric - middleY.arithmetic)
let delta5 = abs(middleZ.harmonic - middleY.arithmetic)
let delta6 = abs(middleZ.arithmetic - middleY.harmonic)
let delta7 = abs(middleZ.harmonic - middleY.harmonic)
let delta8 = abs(middleZ.harmonic - middleY.geometric)
let delta9 = abs(middleZ.geometric - middleY.harmonic)

let minAll = [delta1, delta2, delta3, delta4, delta5, delta6, delta7, delta8, delta9].minAll
guard minAll.indexes.count == 1, let funcType = minAll.indexes.first else {
    fatalError("Error")
}

print("Function type:", "z\(funcType + 1)")

let type = ApproximateType.get(type: funcType)
let zFunc = type.function
let slau = SLAU(forType: type, arrayX: arrayX, arrayY: arrayY)
let (alpha, beta) = slau.getSolution()

print("a:", alpha)
print("b:", beta)

variance = sqrtf(arrayX.enumerated().map { index, x in
    (zFunc(alpha, beta, x) - arrayY[index]).pow(2)
}.sum())

let sko = variance / sqrtf(Float(n))
print("СКО:", sko)
```

Тестирование

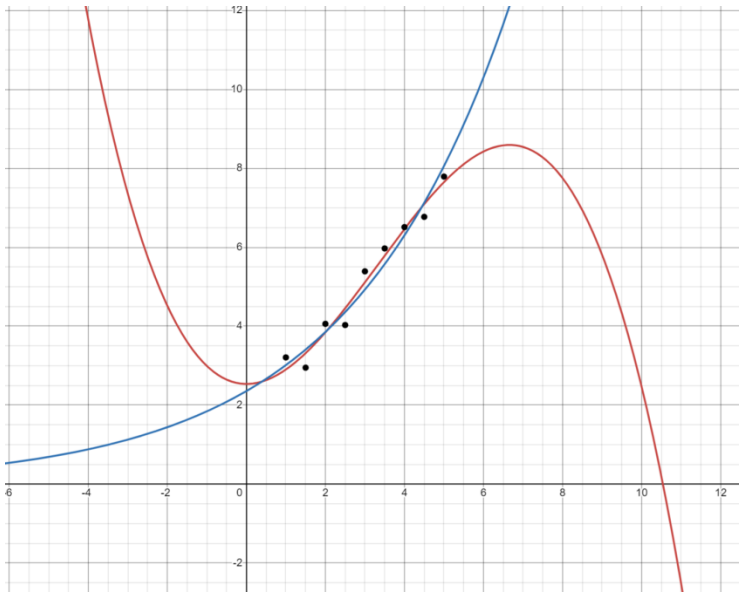
Polynomial: [2.5370317, -0.0045098485, 0.41116562, -0.041141648]

Function type: z3

a: 2.3565795

b: 0.24585526

СКО: 0.33522037



Вывод

В процессе выполнения лабораторной работы была разработана программа, использующая метод наименьших квадратов для решения задачи аппроксимации. Была построена аппроксимация заданных точек, найдена

наиболее точная функция из 9 предложенных, найдены коэффициенты и вычислено среднеквадратичное отклонение данной функции от заданных точек.