Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**­­­­­Лабораторная работа №4**

**по дисциплине «Вычислительная математика»**

**«Аппроксимация функции методом наименьших квадратов»**

Вариант №3

Группа: P3212

Выполнил: Балин А. А.

Проверила: Наумова Н. А.

# Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

# Вычислительная реализация

Точки:



Функция

Для квадратичной аппроксимации:

Решаем систему:

Изображение выглядит как текст, диаграмма, Шрифт, рукописный текст

Автоматически созданное описание

Функция

Очевидно, квадратичная аппроксимация лучше линейной

Изображение выглядит как линия, График, диаграмма, снимок экрана

Автоматически созданное описание

Рисунок 1. График функции и аппроксимирующих многочленов.

# Программная часть

## Код

from math import sin, cos, tan, exp, log, sqrt, pi

from copy import deepcopy

import lab1.matrix as matrix

e = exp(1)

class Approximation:

    def \_\_init\_\_(self,table):

        assert len(table) == 2

        assert len(table[0]) == len(table[1])

        self.table = table

    def linear(self):

        n = len(self.table[0])

        sx = sum(self.table[0])

        sy = sum(self.table[1])

        sxy = sum([self.table[0][i]\*self.table[1][i] for i in range(n)])

        sx2 = sum([self.table[0][i]\*\*2 for i in range(n)])

        a = (n\*sxy - sx\*sy)/(n\*sx2 - sx\*\*2)

        b = (sy - a\*sx)/n

        S = 0

        for i in range(n):

            S += (self.table[1][i] - a\*self.table[0][i] - b)\*\*2

        x\_avg = sx/n

        y\_avg = sy/n

        r\_up = sum([(self.table[0][i] - x\_avg)\*(self.table[1][i] - y\_avg) for i in range(n)])

        r\_down = sqrt(sum([(self.table[0][i] - x\_avg)\*\*2 for i in range(n)])\*sum([(self.table[1][i] - y\_avg)\*\*2 for i in range(n)]))

        delta = sqrt((sum([(a\*self.table[0][i]+b-self.table[1][i])\*\*2 for i in range(n)]))/n)

        return a,b,S,r\_up/r\_down,delta

    def square(self):

        n = len(self.table[0])

        sx = sum(self.table[0])

        sy = sum(self.table[1])

        sx2 = sum([self.table[0][i]\*\*2 for i in range(n)])

        sx3 = sum([self.table[0][i]\*\*3 for i in range(n)])

        sx4 = sum([self.table[0][i]\*\*4 for i in range(n)])

        sy = sum(self.table[1])

        sxy = sum([self.table[0][i]\*self.table[1][i] for i in range(n)])

        sx2y = sum([self.table[0][i]\*\*2\*self.table[1][i] for i in range(n)])

        rows = [[n,sx,sx2,sy],[sx,sx2,sx3,sxy],[sx2,sx3,sx4,sx2y]]

        m = matrix.Matrix(rows)

        m.triangular\_matrix()

        solution = m.solve\_system\_gauss()

        a = solution[2]

        b = solution[1]

        c = solution[0]

        S = 0

        for i in range(n):

            S += (self.table[1][i] - a\*self.table[0][i]\*\*2 - b\*self.table[0][i] - c)\*\*2

        delta = sqrt((sum([(a\*self.table[0][i]\*\*2+b\*self.table[0][i]+c-self.table[1][i])\*\*2 for i in range(n)]))/n)

        return a,b,c,S,delta

    def qube(self):

        n = len(self.table[0])

        sx = sum(self.table[0])

        sy = sum(self.table[1])

        sx2 = sum([self.table[0][i]\*\*2 for i in range(n)])

        sx3 = sum([self.table[0][i]\*\*3 for i in range(n)])

        sx4 = sum([self.table[0][i]\*\*4 for i in range(n)])

        sx5 = sum([self.table[0][i]\*\*5 for i in range(n)])

        sx6 = sum([self.table[0][i]\*\*6 for i in range(n)])

        sxy = sum([self.table[0][i]\*self.table[1][i] for i in range(n)])

        sx2y = sum([self.table[0][i]\*\*2\*self.table[1][i] for i in range(n)])

        sx3y = sum([self.table[0][i]\*\*3\*self.table[1][i] for i in range(n)])

        rows = [[n,sx,sx2,sx3,sy],[sx,sx2,sx3,sx4,sxy],[sx2,sx3,sx4,sx5,sx2y],[sx3,sx4,sx5,sx6,sx3y]]

        m = matrix.Matrix(rows)

        m.triangular\_matrix()

        solution = m.solve\_system\_gauss()

        a = solution[3]

        b = solution[2]

        c = solution[1]

        d = solution[0]

        S = 0

        for i in range(n):

            S += (self.table[1][i] - a\*self.table[0][i]\*\*3 - b\*self.table[0][i]\*\*2 - c\*self.table[0][i] - d)\*\*2

        delta = sqrt((sum([(a\*self.table[0][i]\*\*3+b\*self.table[0][i]\*\*2+c\*self.table[0][i]+d-self.table[1][i])\*\*2 for i in range(n)]))/n)

        return a,b,c,d,S,delta

    def axb(self):

        tables\_old = deepcopy(self.table)

        n = len(self.table[0])

        for i in range(n):

            self.table[0][i] = log(self.table[0][i])

            self.table[1][i] = log(self.table[1][i])

        solution = self.linear()

        self.table = tables\_old

        a = exp(solution[0])

        b = solution[1]

        S = 0

        for i in range(n):

            S += (self.table[1][i] - a\*self.table[0][i]\*\*b)\*\*2

        delta = sqrt((sum([(a\*self.table[0][i]\*\*b-self.table[1][i])\*\*2 for i in range(n)]))/n)

        return a,b,S,delta

    def aebx(self):

        tables\_old = deepcopy(self.table)

        n = len(self.table[0])

        for i in range(n):

            self.table[1][i] = log(self.table[1][i])

        solution = self.linear()

        self.table = tables\_old

        a = exp(solution[1])

        b = solution[0]

        S = 0

        for i in range(n):

            S += (self.table[1][i] - a\*exp(b\*self.table[0][i]))\*\*2

        delta = sqrt((sum([(a\*exp(b\*self.table[0][i])-self.table[1][i])\*\*2 for i in range(n)]))/n)

        return a,b,S,delta

    def alnxplusb(self):

        tables\_old = deepcopy(self.table)

        n = len(self.table[0])

        for i in range(n):

            self.table[0][i] = log(self.table[0][i])

        solution = self.linear()

        self.table = tables\_old

        a = solution[0]

        b = solution[1]

        S = 0

        for i in range(n):

            S += (self.table[1][i] - a\*log(self.table[0][i]) - b)\*\*2

        delta = sqrt((sum([(a\*log(self.table[0][i])+b-self.table[1][i])\*\*2 for i in range(n)]))/n)

        return a,b,S,delta

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Equation:

    @staticmethod

    def get\_function(st):

        st = st.replace("^", "\*\*")

        exec("def f(x): return" + st)

        #return f

        return lambda x: eval(st)

    @staticmethod

    def get\_table(a,b,f,n):

        h = (b-a)/n

        table = [[],[]]

        for i in range(n+1):

            table[0].append(a+i\*h)

            table[1].append(f(a+i\*h))

        return table

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2. Пример выполнения программы.

## Репозиторий с исходниками

https://github.com/ta4ilka69/docs\_for\_labs/tree/main/Вычмат

# Вывод

В ходе реализации данной лабораторной работы я ознакомился с аппроксимацией функции, заданной таблицей точек, с помощью метода наименьших квадратов.