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.

г. Санкт-Петербург, 2024

Цель работы

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

Вычислительная реализация

$$y = \frac{4x}{x^4 + 3}; x \in [-2; 0]; h = 0{,}2$$

Точки:

i	0	1	2	3	4	5	6	7	8	9	10
xi	-2,0000	-1,8000	-1,6000	-1,4000	-1,2000	-1,0000	-0,8000	-0,6000	-0,4000	-0,2000	0,0000
yi	-0,4211	-0,5334	-0,6699	-0,8185	-0,9461	-1,0000	-0,9385	-0,7669	-0,5288	-0,2665	0,0000

$$SX = -11$$

$$\text{SY} = -6{,}8897$$

$$\text{SXX} = 15{,}4$$

$$\text{SYY} = 5{,}2952$$

$$\text{SXY} = 7{,}6311$$

$$SX^2Y = -10{,}0661$$

$$a = \frac{SXY \cdot n - SX \cdot SY}{SXX \cdot n - SX \cdot SX} = 0{,}1685$$

$$b = \frac{SXX \cdot \text{SY} - SX \cdot \text{SXY}}{SXX \cdot n - SX \cdot SX} = -0{,}4578$$

Функция $\varphi_1(x) = 0{,}1685 \cdot x - 0{,}4578$

$$\delta_{\varphi_{1(x)}} = \sqrt{\frac{\sum_{i=1}^{n}\left(\varphi_{1(x_i)} - y_i\right)^2}{n}} = 0{,}2788$$

Для квадратичной аппроксимации:

$$SX^3 = -24{,}2000$$

$$SX^4 = 40{,}5328$$

Решаем систему:

$$\begin{cases} a_0 n + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} x_i y_i \\ a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} x_i^2 y_i \end{cases}$$

$$\text{c} = 0{,}0064$$

$$b = 1{,}7161$$

$$a = 0{,}7738$$

Функция $\varphi_2(x) = 0{,}7738 \cdot x^2 + 1{,}7161 \cdot x + 0{,}0064$

$$\delta_{\varphi_{2(x)}} = \sqrt{\frac{\sum_{i=1}^{n}\left(\varphi_{2(x_i)} - y_i\right)^2}{n}} = 0{,}0548$$

Очевидно, квадратичная аппроксимация лучше линейной

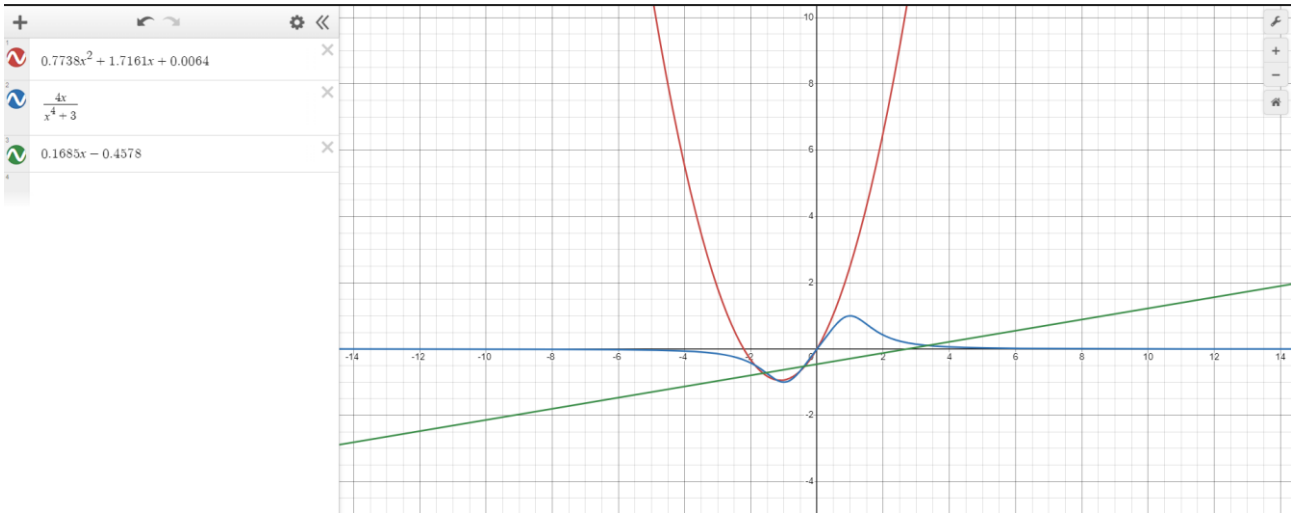

Рисунок 1. График функции и аппроксимирующих многочленов.

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,s
x5,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
```

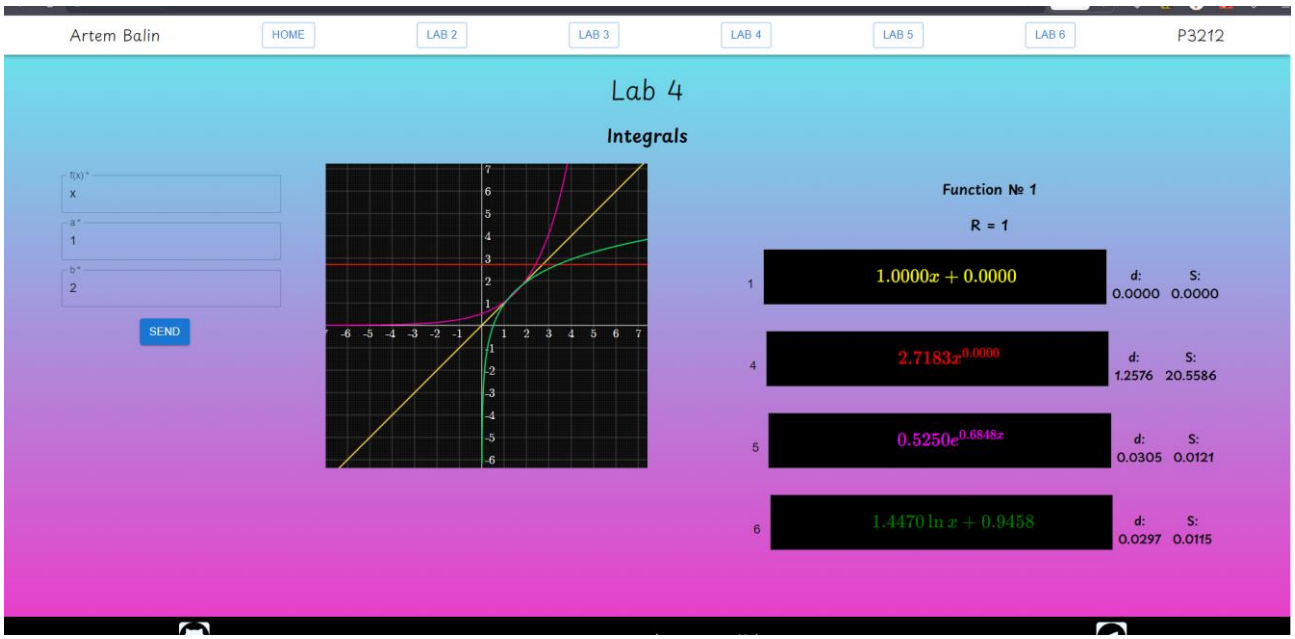

Рисунок 2. Пример выполнения программы.

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.