МГТУ им. Н.Э. Баумана

ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

Лабораторный практикум №4

по теме: «*Построение и программная реализация алгоритма наилучшего среднеквадратичного приближения*.»

Студент: Нгуен Фыок Санг
Группа: ИУ7И-46
Преподаватель: Градов В.М.

2020

Цель работы. Получение навыков построения алгоритма метода наименьших квадратов с использованием полинома заданной степени при аппроксимации табличных функций с весами.

Исходные данные.

1. Таблица функции с **весами** ρ_i с количеством узлов N.

x	y	ρ_i

Предусмотреть в интерфейсе удобную возможность изменения пользователем весов в таблице.

2. Степень аппроксимирующего полинома - n.

Результат работы программы.

Графики, построенные по аналогии с рис.1 в тексте Лекции №4: *точки* - заданная табличная функция, *кривые*- найденные полиномы.

Обязательно приводить таблицы, по которым работала программа.

Под близостью в среднем исходной и аппроксимирующей функций будем понимать результат оценки суммы

$$I = \sum_{i=1}^{N} p_i [y(x_i) - \varphi(x_i)\,]^2 \qquad (1)$$

$y(x)$ - исходная функция

$\varphi(x)$ - множество функций , принадлежащих линейному пространству функций

p_i - вес точки

Нужно найти наилучшее приближение, т.е

$$\sum_{i=1}^{N} p_i [y(x_i) - \varphi(x_i)\,]^2 = min \qquad (2)$$

Разложим функцию $\varphi(x)$ по системе линейно независимых функций $\varphi_k(x)$:

$$\varphi(x) = \sum_{k=0}^{N} a_k \varphi_k(x)$$

Подставляя (3) в условие (2) получим:

$$\big((y-\varphi),(y-\ \varphi)\big)=(y,y)-2\sum_{k=0}^{n}a_k(y,\varphi_k)+\sum_{k=0}^{n}\sum_{m=0}^{n}a_k a_m(\varphi_k,\varphi_m)=min$$

Дифференцируя по а$_k$ получаем:

$$\sum_{i=0}^{n}(x^k,x^m)a_m=(y,x^k)$$

Где

$$(x^k,x^m)=\sum_{i=1}^{N}p_i x_i^{k+m})$$

$$(y,x^k)=\sum_{i=1}^{N}p_i y_i x_i^k$$

СЛАУ решена методом Гаусса.

Подведем итоги.

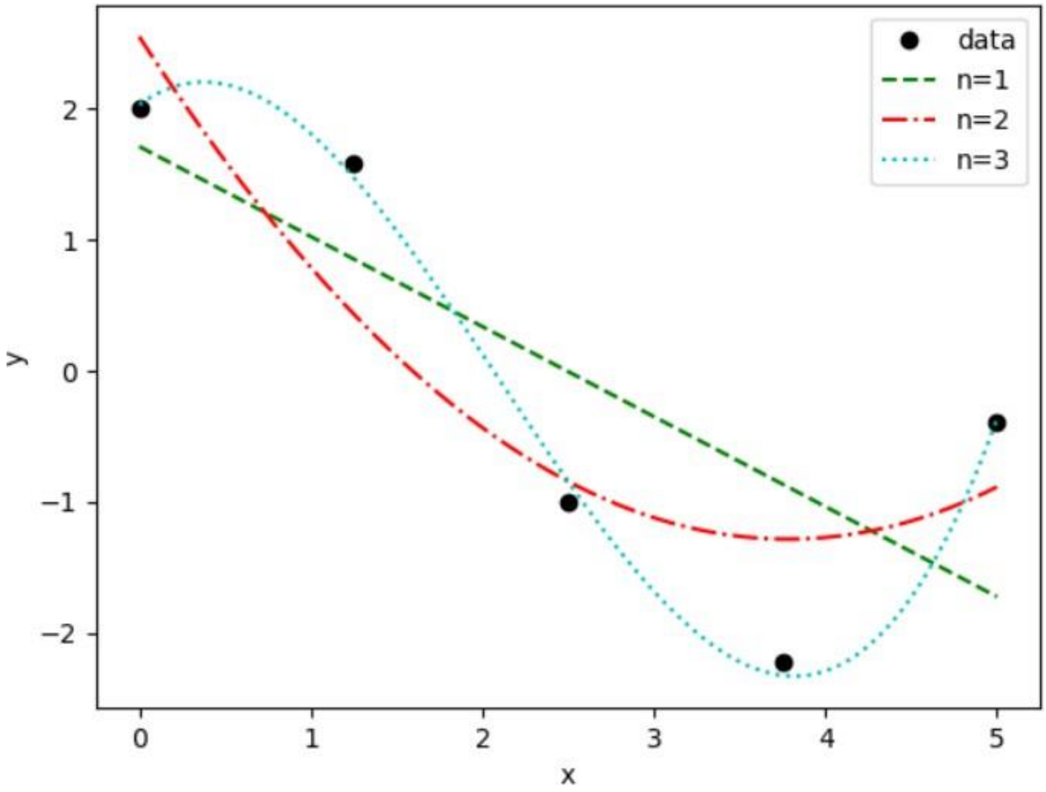
Для применения метода наименьших квадратов в случае аппроксимации полиномом следует действовать следующим образом.

1. Выбирается степень полинома n<N. Обычно степень полинома не превышает 5-6
2. Составляется система линейных алгебраических уравнений
3. В результате решения СЛАУ находятся коэффициенты полинома а$_k$

3. Результаты работы

1. Веса всех точек одинаковы и равны единице

ID	X	Y	P
1	0	2.000	1
2	1.25	1.580	1
3	2.5	-1.004	1
4	3.75	-2.213	1
5	5	-0.392	1



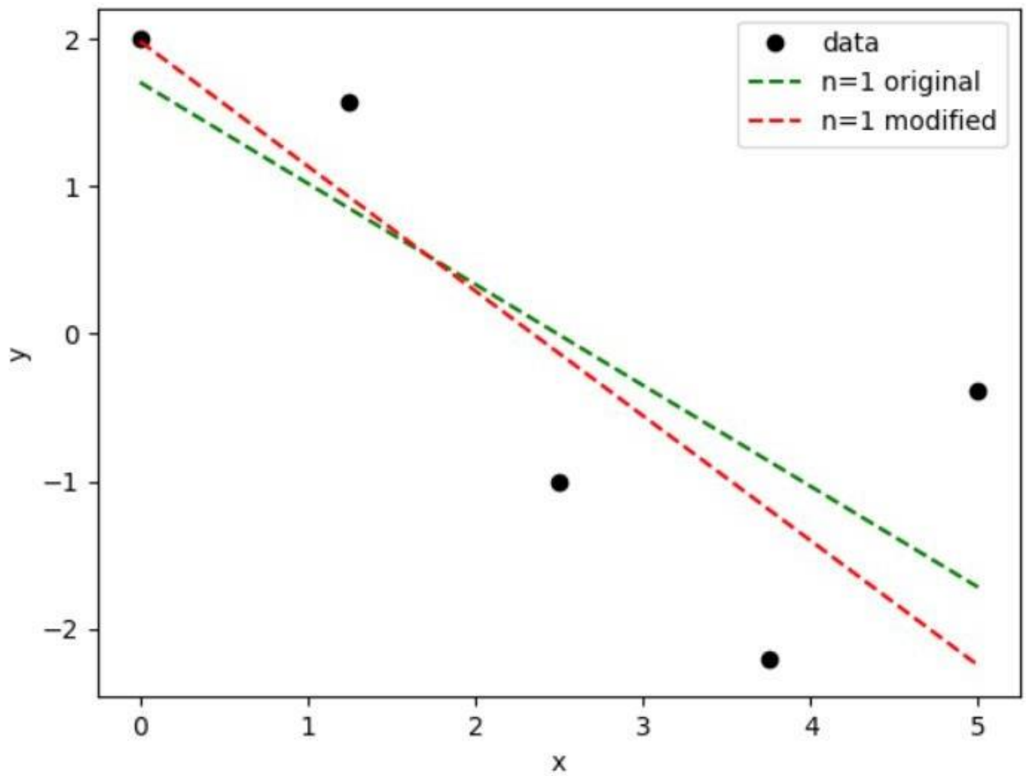
2. Веса точек разные.

Исходная таблиица

ID	X	Y	P
1	0	2.000	1
2	1.25	1.580	1
3	2.5	-1.004	1
4	3.75	-2.213	1
5	5	-0.392	1

модифицированная таблица

ID		X	Y	P
1		0	2.000	2
2		1.25	1.580	2
3		2.5	-1.004	2
4		3.75	-2.213	0.5
5		5	-0.392	0.5



Воспросы при защите лабораторной работы

1. ***Что произойдет при задании степени полинома n=N-1 (числу узлов таблицы минус 1)?***

   График полинома будет проходить через все точки в таблице, независимо от весов точек.

2. ***Будет ли работать Ваша программа при n □ N ? Что именно в алгоритме требует отдельного анализа данного случая и может привести к аварийной остановке?***

   Программа работать не будет. График будет но на графику будет отображаться только набор точек а не функция приближения. При n □ N уравнения будут линейно зависимыми т.е определитель этой СЛАУ равен нулю. Аварийная остановка может произойти при выполнения элементарных строковых операций, где может произойти ошибка деления на ноль.

3. ***Получить формулу для коэффициента полинома $a0$ при степени полинома n = 0. Какой смысл имеет величина, которую представляет данный коэффициент?***

$$\frac{\sum_{i=1}^{N} y_i p_i}{\sum_{i=1}^{N} p_i}$$

Значение: математичское ожидание

4. ***Записать и вычислить определитель матрицы СЛАУ для нахождения коэффициентов полинома для случая, когда n = N = 2.***

X_i	Y_i	P_i
X_0	Y_0	P_1
X_1	Y_1	P_2

Тогда имеем СЛАУ вида:

$$\begin{cases} (p_0+p_1)a_0+(p_0x_0+p_1x_1)a_1+(p_0x_0^2+p_1x_1^2)a_2=p_0y_0+p_1y_1 \\ (p_0x_0+p_1x_1)a_0+(p_0x_0^2+p_1x_1^2)a_1+(p_0x_0^3+p_1x_1^3)a_2=p_0y_0x_0+p_1y_1x_1 \\ (p_0x_0^2+p_1x_1^2)a_0+(p_0x_0^3+p_1x_1^3)a_1+(p_0x_0^4+p_1x_1^4)a_2=p_0y_0x_0^2+p_1y_1x_1^2 \end{cases}$$

$$\begin{aligned} \Delta= &(p_0+p_1)(p_0x_0^2+p_1x_1^2)(p_0x_0^4+p_1x_1^4)+(p_0x_0+p_1x_1)(p_0x_0^3+p_1x_1^3)(p_0x_0^2+p_1x_1^2) \\ &+(p_0x_0^2+p_1x_1^2)(p_0x_0+p_1x_1)(p_0x_0^3+p_1x_1^3) \\ &-(p_0x_0^2+p_1x_1^2)(p_0x_0^2+p_1x_1^2)(p_0x_0^2+p_1x_1^2) \\ &-(p_0+p_1)(p_0x_0^3+p_1x_1^3)(p_0x_0^3-p_1x_1^3) \\ &-(p_0x_0+p_1x_1)(p_0x_0+p_1x_1)(p_0x_0^4+p_1x_1^4)=0 \end{aligned}$$

Так как Δ = 0, система решений не имеет

5. Построить СЛАУ при выборочном задании степеней аргумента полинома $\varphi(x)=\ a_0+a_1x^m+a_2x^n$причем степени n и m в этой формуле известны

$$\begin{cases} (x^0,x^0)a_0+(x^0,x^m)a_1+(x^0,x^n)a_2=(y,x^0) \\ (x^m,x^0)a_0+(x^m,x^m)a_1+(x^m,x^n)a_2=\ (y,x^m) \\ (x^n,x^0)a_0+(x^n,x^m)a_1+(x^n,x^n)a_2=\ (y,x^n) \end{cases}$$

6. Решить задачу из вопроса 5, если степени n и m подлежат определению наравне с коэффициентами a_i

Пусть k – степень полинома

$$(x^i,x^0)a_0+(x^i,x^a)a_1+(x^i,x^b)a_2=(y,x^i)$$

Где 0 ≤i≤k; a=i < n; b=i < m

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from math import *


EPS = 10e-7
def f(x):
    return sin(x) + 2*cos(x)


class Point:
    def __init__(self, x, y, p):
        self.x = x;
        self.y = y;
        self.p = p; #the weight of the point(x, y)


    def setP(self, newP):
        self.p = newP
def initPoints(num):
    start = 0
    stop = 5
    points = []
    for x in np.linspace(start, stop, num):
        points.append(Point(x, f(x), 1))
    return points
```

```
def printTable(points):
    print("ID      X      Y      P")
    iter = 1
    for p in points:
        print('{:2}'.format(iter), end='')
        iter += 1
        print('{:8.3f}'.format(p.x), end='')
        print('{:8.3f}'.format(p.y), end='')
        print('{:8.3f}'.format(p.p), end='')
        print()
#System of linear equations is represented as a matrix of coefficients
def initCoeffMatrix(points, deg):
    num = len(points)
    matrix = [[0 for j in range(deg+2)] for i in range(deg+1)] # matrix[deg+1]*[deg+2]
    for i in range(deg+1):
        for j in range(deg+2):
            coeff = 0
            for k in range(num):
                coeff += points[k].p * points[k].x ** (i+j)
            matrix[i][j] = coeff
        augmentedCoeff = 0
        for k in range(num):
            augmentedCoeff += points[k].p * points[k].y * points[k].x ** i
        matrix[i][deg+1] = augmentedCoeff
    return matrix
```

```
#Using Gaussian elimination method
#matrix n * (n + 1)
def solve(matrix):
    n = len(matrix)
    res = [0 for i in range(n)]
    #row swapping to shape a "loose" row echelon form
    #the pivot of a non zero row is to the right OR UNDER the pivot of the row above
    for i in range(n-1):
        for j in range(i+1, n):
            if abs(matrix[i][i]) < abs(matrix[j][i]):
                tmp = matrix[i]
                matrix[i] = matrix[j]
                matrix[j] = tmp
#performing Gaussian elimination
#matrix is shaped into the "strict" row echelon form
    for i in range(n-1):
        for j in range(i+1, n):
            if abs(matrix[i][i]) < EPS:
                return res, False
            scalar = matrix[j][i] / matrix[i][i]
            for k in range(n+1):
                matrix[j][k] -= scalar * matrix[i][k]
    print(matrix[n-1])
```

```
    #Backward substitutions
    for i in range(n-1, -1, -1):
        res[i] = matrix[i][n]
        for j in range(i+1, n):
            res[i] -= matrix[i][j] * res[j]
        res[i] /= matrix[i][i]
    print(res)
    return res, True


def phi(res, x):
    val = 0
    for i in range(len(res)):
        val += res[i] * (x ** i)
    return val
def plotData(points):
    x = [p.x for p in points]
    y = [p.y for p in points]
    plt.plot(x, y, "ko", label="data")
    plt.ylabel('y')
    plt.xlabel('x')
```

```
def plotApproximationFunc(points, res, deg, mode=0):
    x = np.linspace(points[0].x, points[len(points) - 1].x, 100)
    y = [phi(res, i) for i in x]
    #the degree of the polynomial is recommended <= 6
    fmt1 = ["-b", "--g", "-.r", ":c", "-m", "--y", "-.b"]
    fmt2 = ["-g", "--r", "-.c", ":m", "-y", "--b", "-.g"]
    if mode == 0:
        plt.plot(x, y, fmt1[deg%7], label="n={:d}".format(deg))
    elif mode == 1:
        plt.plot(x, y, fmt1[deg%7], label="n={:d}".format(deg) + " original")
    elif mode == 2:
        plt.plot(x, y, fmt2[deg%7], label="n={:d}".format(deg) + " modified")

def initTable():
    num = int(input("Input the number of points : "))
    #Init points
    points = initPoints(num)
    printTable(points)
    return points
```

```
def modifyTable(points):
    num = len(points)
    while True:
        print("Please choose an option")
        print("0 - I'm fine with the current table")
        print("1 - Changing a weight value")
        option = int(input())
        if option == 0 :
            break;
        elif option == 1 :
            id = int(input("Input the id of the point you want to change : "))
            new_p = float(input("Input the new weight value : "))
            if 1 <= id <= num :
                points[id-1].setP(new_p)
                print("Success!")
            else :
                print("Failed!")
    print("The final table ")
    printTable(points)
    return points
```

```
def approximationProcess(points, mode = 0):
    degreeOfPolynomial = int(input("Input the degree of the polynomial : "))
    if degreeOfPolynomial < 0:
        print("The degree of the polynomial must be non negative")
        return False
    if degreeOfPolynomial > len(points):
        printf("The degree of the polynomial <= the number of points")
        return False
    matrix = initCoeffMatrix(points, degreeOfPolynomial)
    res, status = solve(matrix)
    if not status:
        print("The solution of SOLE is not unique!")
        return False
    plotApproximationFunc(points, res, degreeOfPolynomial, mode)
    return True

def process1():
    points = initTable()
    points = modifyTable(points)
    plotData(points)
    approximationProcess(points)
    plt.legend()
    plt.show()
```

```
def process2():
    points = initTable()
    plotData(points)
    while approximationProcess(points):
        print("Input -1 if you want to stop")
    plt.legend()
    plt.show()

def process3():
    points = initTable()
    plotData(points)
    approximationProcess(points, mode = 1)
    points = modifyTable(points)
    approximationProcess(points, mode = 2)
    plt.legend()
    plt.show()


if __name__ == "__main__":
    process1()
```