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 №5:

Метод наименьших квадратов. Аппроксимация алгебраическими многочленами.

Студент ИУ9-61Б (Группа) — Е.А. Матвеев (И.О. Фамилия)

Проверила — А. Б. Домрачева (И.О. Фамилия)

2023 г.

Содержание

1 Цель

Целью данной работы является изучение метода наименьших квадратов аппроксимации функции алгебраическими многочленами: анализ точности аппроксимации и нахождение отклонений.

2 Постановка задачи

Дано: Набор пар точек $\{(x_i, y_i)\}_0^n$ (имеется ввиду, что существует функция $f(x) : \mathbb{R} \to \mathbb{R}$ такая, что $f(x_i) = y_i$), функция $g(x, \beta) : \mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$ и функция погрешности $S(t) : \mathbb{R}^{n+1} \to \mathbb{R}$

Требуется: Найти такой вектор β, при котором значение $S(t)$ минимально (t - это вектор $((y_1 - g(x_1, \beta)), (y_2 - g(x_2, \beta))...(y_n - g(x_n, \beta)))$.

Для достижения цели работы были поставлены следующие задачи:

1. Изучить теоретическое обоснование метода наименьших квадратов и предпосылки к его применению.
2. Реализовать алгоритм метода наименьших квадратов на языке программирования python3.
3. Рассчитать значение функции погрешности и отклонения полученные на тестовых данных.

3 Сведения о методе наименьших квадратов

Значения функции $f(x_i)$ могут быть получены с некоторой случайной погрешностью. Тогда нет смысла пытаться провести аппроксимирующую функцию точно через узлы (x_i, y_i), а достаточно провести ее довольно близко к ним, тем самым, получив значение $S(t)$ близко к минимальному.

Смысл явного отделения вектора β от функции $g(x, \beta)$ возникает, когда вводится следующее дополнительное условие: функция g ищется в множестве функций, являющихся какой-либо комбинацией (обычно линейной) функций набора $\{g_i\}_1^p$.

Могут быть и другие причины использования данного метода. Так, например, при фиксации определенного набора $\{g_i\}_1^p$ можно разработать специфические для данного набора способы вычисления значений или представления в памяти ЦВМ аппроксимирующей функции.

Значение абсолютной погрешности аппроксимации Δ вычисляется по формуле: $\Delta = \frac{S(t)}{\sqrt{(n)}}$.

Относительная ошибка находится по формуле: $\delta = \frac{\Delta}{\sqrt{\sum_0{}^n y_i^2}}$.

4 Реализация

Конкретно в этой работе в качестве $\{g_i\}_1^p$ взят набор функций: $1, x, x^2, x^3$, вектор $\beta \in \mathbb{R}^4$ задает линейную комбинацию функций из набора, т.е. $g(x, \beta) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$. В качестве $S(t)$ берется функция среднеквадратичного отклонения: $S(t) = \sqrt{\sum_0^n (y_i - g(x_i, t))^2}$.

На листинге 1 представлена реализация метода наименьших квадратов аппроксимации функции с учетом вышеперечисленных уточнений.

Листинг 1: Реализация метода наискорейшего спуска

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from math import floor

def f(x):
    d = [1.22, 1.18, 2.5, 2.35, 4.47, 6.02, 7.35, 10.9, 15.33]
    y = floor((2*x-2))
    return d[y]

def create_matrix_and_res(x, y):
    A = np.empty((m,m))
    b = np.empty(m)

    for i in range(m):
        for j in range(m):
            A[i][j] = sum([x[k]**(i + j) for k in range(n + 1)])
        b[i] = sum([y[k] * x[k] ** i for k in range(n + 1)])

    return A, b

def find_alphas(A, b):
    return (np.linalg.inv(A).dot(b.T)).T

def find_alphas2(A, b):
    T = np.zeros((m,m))
    x = np.empty(m)
    y = np.empty(m)

    # вычисляем t
    for i in range(m):
        for j in range(i):
            T[i][j] = (A[i][j] - sum([T[i][k] * T[j][k] for k in range(j)])) / T[j][j]
        T[i][i] = np.sqrt(A[i][i] - sum(T[i][k] ** 2 for k in range(i)))

    # обратный ход
    for i in range(m):
        y[i] = (b[i] - sum([T[i,k]*y[k] for k in range(i)])) / T[i,i]

    for i in range(m-1, -1, -1):
        x[i] = (y[i] - sum([T[k,i]*x[k] for k in range(i+1, m)])) / T[i,i]

    return x


m = 4

x0 = 1
xn = 5
n = 8
h = (xn-x0)/n
xs = np.linspace(x0, xn, n+1, True)
ys = [f(x) for x in xs]

A, b = create_matrix_and_res(xs, ys)

alphas = find_alphas2(A,b)
print("alphas:")
print(alphas)

MSE = np.sqrt(
        sum([
            (ys[k] - sum([alphas[i] * xs[k]**i for i in range(m)]))
             for k in range(m+1)
         ])**2
    ) / np.sqrt(n)
print("MSE")
print(MSE)

delta = MSE / np.sqrt(sum([ys[k]**2 for k in range(n+1)]))
print("delta")
print(delta)


def f_my(x):
    return sum([alphas[i] * x**i for i in range(m)])

xs_test = np.linspace(x0, xn, n+n+1, True)
res_my = np.vectorize(f_my)(xs_test)
res_true = np.vectorize(f)(xs)

plt.plot(xs_test, res_my)
plt.scatter(xs, res_true)
```

5 Тестирование

Для тестирования алгоритма взяты следующие точки $\{(x_i, y_i)\})$:

	xi	yi
0	1.0	1.22
1	1.5	1.18
2	2.0	2.5
3	2.5	2.35
4	3.0	4.47
5	3.5	6.02
6	4.0	7.35
7	4.5	10.9
8	5.0	15.33

Рисунок 1 — Точки $\{(x_i, y_i)\})$

Найден вектор $\beta : (-0.74071429, 2.5434127, -0.99744589, 0.22505051)$ при котором значение абсолютной погрешности Δ равно 0.019162287663323653, а относительная ошибка составляет 0.0008758069434217044

На рисунке 2 представлена таблица истинных значений исходной функции f в точках x_i, значение полученной аппроксимирующей функции $g(x, \beta)$ в этих точках и абсолютная разница между значениями этих функций. Дополнительно приведены значения аппроксимирующей функции в точках $\frac{x_i+x_{i+1}}{2}$.

На рисунке 3 точками отмечены узлы $\{(x_i, y_i)\}$, кривая - график полученной аппроксимирующей функции $g(x, \beta)$.

	x	f(x)	approx	absDiff
0	1.0	1.22	1.030303030302996	0.1896969696970039
1	1.25		1.3195941558441364	
2	1.5	1.18	1.5896969696969616	0.4096969696969617
3	1.75		1.8617099567099578	
4	2.0	2.5	2.1567316017316105	0.3432683982683895
5	2.25		2.495860389610403	
6	2.5	2.35	2.9001948051948223	0.5501948051948222
7	2.75		3.390833333333351	
8	3.0	4.47	3.9888744588744753	0.4811255411255244
9	3.25		4.715416666666682	
10	3.5	6.02	5.591558441558453	0.42844155844154663
11	3.75		6.6383982683982765	
12	4.0	7.35	7.877034632034636	0.5270346320346366
13	4.25		9.328566017316017	
14	4.5	10.9	11.014090909090902	0.11409090909090125
15	4.75		12.954707792207781	
16	5.0	15.33	15.171515151515136	0.15848484848486422

Рисунок 2 — Сравнение исходной и аппроксимирующей функций

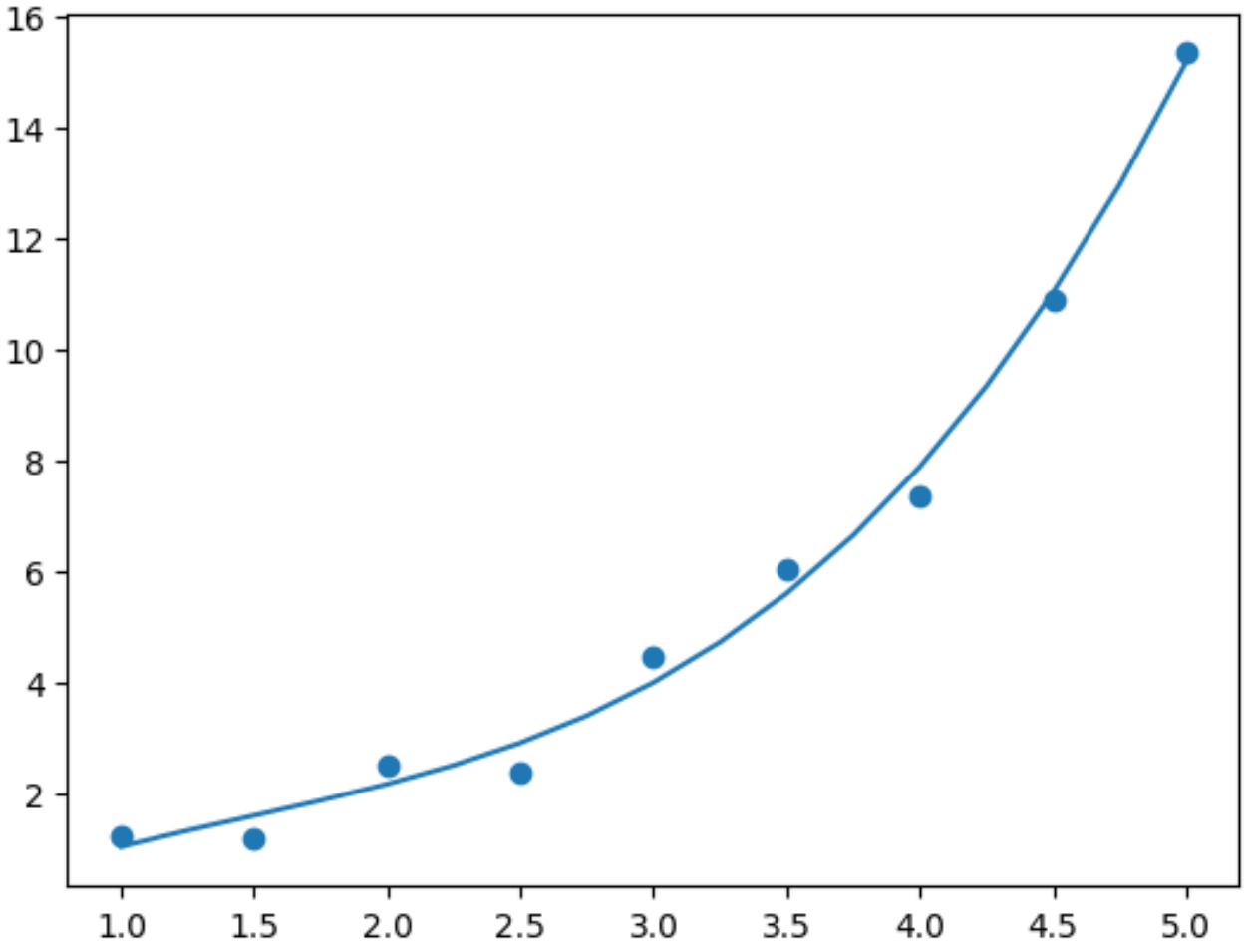

Рисунок 3 — График аппроксимирующей функции $g(x, \beta)$ и узлы $\{(x_i, y_i)\}$

6 Вывод

При выполнении лабораторной работы был изучен и реализован в программном коде метод наименьших квадратов аппроксимации функции.

Из сути метода следует, что его имеет смысл применять для описания некоторой численной зависимости. Допустим, мы имеем некоторую выборку на интервале $[a : b]$. Тогда построить аппроксимацию этой выборки с помощью рассматриваемого метода. При этом нужно понимать, что не любой набор базовых функций способен аппроксимировать любую зависимость. Так, например, с помощью любого множества многочленов не выйдет описать зависимость $x^2 + y^2 = 1$. К тому же, стоит подбирать набор базовых функций так, чтобы между ними не было зависимости (линейной, если $g(x, \beta)$ - линейная комбинация базовых функций и т.д.), в противном случае нельзя гарантировать единственность решения - вектора β.

Так же, следует сказать, что предсказания аппроксимирующей функции в точках достаточно отдаленных от интервала $[a : b]$ не несут практического смысла.

Чтобы говорить о достоверности полученной аппроксимации, нужно удостоверится, что входной набор точек в некотором смысле “хорошо” описывает аппроксимируемую зависимость.