Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Московский государственный технический университет имени Н.Э. Баумана

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №5
по курсу «Численные методы»
«Метод наименьших квадратов. Аппроксимация алгебраическими многочленами»

Выполнила:
студентка группы ИУ9-62Б
Самохвалова П. С.

Проверила:
Домрачева А. Б.

Москва, 2023

Цель:

Анализ метода наименьших квадратов для решения задачи аппроксимации алгебраическими многочленами.

Постановка задачи:

Дано: Функция $y_i = f(x_i), i = \overline{0, n}$, значения y_i могут иметь случайные ошибки.

x_1	x_2	...	x_{n-1}	x_n
y_1	y_2	...	y_{n-1}	y_n

Найти: Гладкую аналитическую функцию $z(x)$, доставляющую наименьшее значение величине

$$SKU = \sqrt{\sum_{i=0}^{n} (z(x_i) - y_i)^2}$$

Эту величину называют среднеквадратичным уклонением функции $z(x)$ от системы узлов. Описанный подход к решению задачи приближения функции - методом наименьших квадратов.

Тестовый пример:

Вариант 21

$$f(x) = y$$

x	1	1.5	2	2.5	3	3.5	4	4.5	5
y	0.86	0.97	0.65	0.75	1.60	0.65	1.34	1.62	1.01

Задание:

1. Аппроксимировать данную функцию по методу наименьших квадратов многочленом третьей степени (m = 4). Найти:
   а) Матрицу A и столбец b;
   б) Набор коэффициентов λ_1, λ_2, λ_3, λ_4;
   в) Среднеквадратичное отклонение Δ и относительную ошибку δ;
   г) Значения аппроксимирующего многочлена z(x) в средних точках отрезков между узловыми точками.

2. То же для решения задачи Коши, полученного в предыдущей лабораторной работе.

Описание метода:

Как правило, $z(x)$ отыскивают в виде линейной комбинации заданных функций

$$z(x) = \lambda_1\varphi_1(x) + ... + \lambda_m\varphi_m(x)$$

Параметры $\lambda_i, i = \overline{1,m}$ являются решениями линейной системы наименьших квадратов

$$A\lambda = b,$$

где λ - столбец параметров λ_i . $A = (a_{ij})$ - симметричная положительно определенная матрица (матрица Грама) с коэффициентами $a_{ij} = \sum_{k=0}^{n} \varphi_i(x_i)\varphi_j(x_k)$; b - столбец правой части системы, $b_i = \sum_{k=0}^{n} \varphi_i(x_k)y_k, \quad i,j = \overline{1,m}$

Если приближаемая функция достаточно глакая, хотя вид ее и неизвестен, аппроксимирующую функцию нередко ищут в виде алгебраического многочлена

$$z(x) = \lambda_1 + \lambda_2 x + ... + \lambda_m x^{m-1}.$$

Тогда $\varphi_i = x^{i-1}$ и элементы матрица Грама получают по формулам

$$a_{ij} = \sum_{k=0}^{n} x_k^{i+j-2},$$

а свободные члены -

$$b_i = \sum_{k=0}^{n} y_k^{i-1}, \quad i,j = \overline{1,m}$$

Абсолютной погрешностью аппроксимации служит среднеквадратичное отклонение (СКО):

$$\Delta = \frac{SKU}{\sqrt{n}} = \frac{1}{\sqrt{n}}\sqrt{\sum_{k=0}^{n}(y_k - \lambda_1 - \lambda_2 x_k - ... - \lambda_m x_k^{m-1})^2},$$

относительная ошибка

$$\delta = \frac{\Delta}{||y||} = \frac{\Delta}{\sqrt{\sum_{k=0}^{n} y_k^2}}$$

Листинг 1. Метод наименьших квадратов. Аппроксимация алгебраическими многочленами

```
def mnk(n, x, y, m):
    a = []
    for i in range(m):
        a.append([0] * m)
    b = [0] * m

    for i in range(m):
        for j in range(m):
            for k in range(n + 1):
```

```
            a[i][j] += x[k] ** (i + j)

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

print("A =")

for i in range(m):
    for j in range(m):
        print("{:20} ".format(str(a[i][j])), end="")
    print()

print()

print("b =")

for i in range(m):
    print(b[i])

print()

t = []
for i in range(m):
    t.append([0] * m)

t_tr = []

for i in range(m):
    t_tr.append([0] * m)

t[0][0] = a[0][0] ** 0.5

for j in range(1, m):
    t[0][j] = a[0][j] / t[0][0]

for i in range(1, m):
    for j in range(m):
        if i == j:
```

```
            s = 0
            for k in range(i):
                s += t[k][i] ** 2
            t[i][i] = (a[i][i] - s) ** 0.5
        elif i < j:
            s = 0
            for k in range(i):
                s += t[k][i] * t[k][j]
            t[i][j] = (a[i][j] - s) / t[i][i]

for i in range(m):
    for j in range(m):
        t_tr[i][j] = t[j][i]

xr = [0] * m
yr = [0] * m

yr[0] = b[0] / t[0][0]

for i in range(1, m):
    s = 0
    for k in range(i):
        s += t[k][i] * yr[k]
    yr[i] = (b[i] - s) / t[i][i]

xr[m - 1] = yr[m - 1] / t[m - 1][m - 1]

for i in range(m - 2, -1, -1):
    s = 0
    for k in range(i + 1, m):
        s += t[i][k] * xr[k]
    xr[i] = (yr[i] - s) / t[i][i]

la = xr[:]

print("lambda =")

for i in range(m):
    print(la[i])
```

```
print()

s = 0

for k in range(n + 1):
    s1 = 0
    for q in range(m):
        s1 += la[q] * (x[k] ** q)
    s += (y[k] - s1) ** 2

abs_delta = (1 / (n ** 0.5)) * (s ** 0.5)

s = 0

for k in range(n + 1):
    s += y[k] ** 2

otn_delta = abs_delta / (s ** 0.5)

print("abs_delta =")
print(abs_delta)

print()

print("otn_delta = ")
print(otn_delta)

print()

ym = [0] * n

for i in range(n):
    xm = (x[i] + x[i + 1]) / 2
    s = 0
    for k in range(m):
        s += (la[k] * (xm ** k))
    ym[i] = s
```

```
    print("ym = ")
    for i in range(n):
        print(ym[i])


n = 8
x = [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]
y = [0.86, 0.97, 0.65, 0.75, 1.60, 0.65, 1.34, 1.62, 1.01]
m = 4

print("1.")
print()
mnk(n, x, y, m)
print()

n = 10
x = [0.0, 0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6000000000000001,
     0.7000000000000001, 0.8, 0.9, 1.0]
y = [1.0, 1.205004334722476, 1.420072088955231, 1.6453790142373428,
     1.881243039949921, 2.128146809304331, 2.3867612737855444,
     2.6579703947081676, 2.9428970193919906, 3.242930020784433,
     3.5597528132669414]
m = 4

print("2.")
print()
mnk(n, x, y, m)
```

Результаты работы:

В результате работы программы на тестовом примере получаем:

$$A = \begin{pmatrix} 9.0 & 27.0 & 96.0 & 378.0 \\ 27.0 & 96.0 & 378.0 & 1583.25 \\ 96.0 & 378.0 & 1583.25 & 6900.75 \\ 378.0 & 1583.25 & 6900.75 & 30912.5625 \end{pmatrix}$$

$$b = \begin{pmatrix} 9.450000000000001 \\ 30.265000000000004 \\ 112.1875 \\ 451.75375 \end{pmatrix}$$

$$\lambda = \begin{pmatrix} 1.913730158729999 \\ -1.544129389129182 \\ 0.6322510822510062 \\ -0.07084175084174252 \end{pmatrix}$$

СКО $\Delta = 0.3179591701941208$

Относительная ошибка $\delta = 0.09548667876045708$

Таблица значений аппроксимирующего многочлена z(x) в средних точках отрезков между узловыми точками

x_n	$z(x_n)$
1.25	0.8330979437229404
1.75	0.7681051587301733
2.25	0.8332783189033353
2.75	0.9754861111111188
3.25	1.1415972222222188
3.75	1.278480339105326
4.25	1.3330041486291346
4.75	1.2520373376623395

Также программа была запущена на значениях, полученных при решении задачи Коши в предыдущей лабораторной работе. Было получено:

$$A = \begin{pmatrix} 11.0 & 5.500000000000001 & 3.8500000000000005 & 3.0250000000000004 \\ 5.500000000000001 & 3.8500000000000005 & 3.0250000000000004 & 2.5333 \\ 3.8500000000000005 & 3.0250000000000004 & 2.5333 & 2.2082500000000005 \\ 3.0250000000000004 & 2.5333 & 2.2082500000000005 & 1.9784050000000004 \end{pmatrix}$$

$$b = \begin{pmatrix} 24.07015680910638 \\ 14.8400426642202 \\ 11.28159240138787 \\ 9.30124507302068 \end{pmatrix}$$

$$\lambda = \begin{pmatrix} 0.999382178223316 \\ 2.0203891611643496 \\ 0.40411000986154677 \\ 0.1352029501748844 \end{pmatrix}$$

СКО $\Delta = 0.0005739961457411783$

Относительная ошибка $\delta = 7.419259034640004e - 05$

Таблица значений аппроксимирующего многочлена z(x) в средних точках отрезков между узловыми точками

x_n	$z(x_n)$
0.05	1.101428811674959
0.15000000000000002	1.3119893375766936
0.25	1.5318488902272327
0.35000000000000003	1.761818687327626
0.45	2.002709946578923
0.55	2.2553338856821727
0.6500000000000001	2.520501722338425
0.75	2.7990246742487277
0.8500000000000001	3.0917139591141316
0.95	3.3993807946356855

Выводы:

В результате выполнения лабораторной работы был изучен метод наименьших квадратов для решения задачи аппроксимации алгебраическими многочленами, была написала реализация данного метода на языке программирования Python. На тестовом примере среднеквадратичное отклонение составило $\Delta = 0.3179591701941208$, относительная ошибка составила $\delta = 0.09548667876045708$. Дальнейшее увеличение точности решения можно осуществить увеличением количества точек разбиения и повышением степени многочлена.