Міністерство освіти і науки України

Національний технічний університет України «КПІ» імені Ігоря Сікорського

Кафедра обчислювальної техніки ФІОТ

Звіт

з лабораторної роботи №5

з навчальної дисципліни «Методи оптимізації та планування експерименту»

Тема: Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план)

Виконав:

Студент 2 курсу кафедри ОТ ФІОТ

Навчальної групи ІО-91

Тарасенко Андрій

Перевірив:

Регіда П. Г.

Київ 2021

Мета: Провести трьохфакторний експеримент з урахуванням квадратичних членів, використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Завдання:

1. Взяти рівняння з урахуванням квадратичних членів.

2. Скласти матрицю планування для ОЦКП

3. Провести експеримент у всіх точках факторного простору (знайти значення функції відгуку Y). Значення функції відгуку знайти у відповідності з варіантом діапазону, зазначеного далі. Варіанти вибираються по номеру в списку в журналі викладача.

4. Розрахувати коефіцієнти рівняння регресії і записати його.

5. Провести 3 статистичні перевірки.

Варіант:

122	-8	6	-5	4	-5	8

Лістинг програми

```
"""
 * Copyright © 2021 drewg3r
 * https://github.com/drewg3r/DOX-labs

main.py: main file to run the program.
"""


import random
from tabulate import tabulate
import numpy as np
import math


def get_avg(array):
    result = []
    for i in range(len(array[0])):
        result.append(0)
        for j in range(len(array)):
            result[i] += array[j][i]
        result[i] = result[i] / len(array)
    return result


def get_dispersion(array, avg_y):
    result = []
    for i in range(len(array[0])):
        result.append(0)
        for j in range(len(array)):
            result[i] += (array[j][i] - avg_y[i]) ** 2
        result[i] = result[i] / 3
    return result


def add_sq_nums(x):
    for i in range(len(x)):
        x[i][3] = x[i][0] * x[i][1]
        x[i][4] = x[i][0] * x[i][2]
        x[i][5] = x[i][1] * x[i][2]
        x[i][6] = x[i][0] * x[i][1] * x[i][2]
        x[i][7] = x[i][0] ** 2
        x[i][8] = x[i][1] ** 2
        x[i][9] = x[i][2] ** 2
    return x


def plan_matrix5(n, m, x_norm):
    l = 1.215
    x_norm = np.array(x_norm)
    x_norm = np.transpose(x_norm)
    x = np.ones(shape=(len(x_norm), len(x_norm[0])))
    for i in range(8):
        for j in range(3):
            if x_norm[i][j] == -1:
                x[i][j] = x_range[j][0]
            else:
                x[i][j] = x_range[j][1]
    for i in range(8, len(x)):
        for j in range(3):
```

```
            x[i][j] = float((x_range[j][0] + x_range[j][1]) / 2)
    dx = [x_range[i][1] - (x_range[i][0] + x_range[i][1]) / 2 for i in range(3)]
    x[8][0] = (-l * dx[0]) + x[9][0]
    x[9][0] = (l * dx[0]) + x[9][0]
    x[10][1] = (-l * dx[1]) + x[9][1]
    x[11][1] = (l * dx[1]) + x[9][1]
    x[12][2] = (-l * dx[2]) + x[9][2]
    x[13][2] = (l * dx[2]) + x[9][2]
    x = add_sq_nums(x)
    for i in range(len(x)):
        for j in range(len(x[i])):
            x[i][j] = round(x[i][j], 3)
    return np.transpose(x).tolist()


def s_kv(y, y_aver, n, m):
    res = []
    for i in range(n):
        s = sum([(y_aver[i] - y[i][j]) ** 2 for j in range(m)]) / m
        res.append(round(s, 3))
    return res


def bs(x, y_aver, n):
    res = [sum(1 * y for y in y_aver) / n]

    for i in range(len(x[0])):
        b = sum(j[0] * j[1] for j in zip(x[:, i], y_aver)) / n
        res.append(b)
    return res


def student(x, y, y_aver, n, m):
    S_kv = s_kv(y, y_aver, n, m)
    s_kv_aver = sum(S_kv) / n

    s_Bs = (s_kv_aver / n / m) ** 0.5
    Bs = bs(x, y_aver, n)
    ts = [round(abs(B) / s_Bs, 3) for B in Bs]

    return ts


x1min = -8
x1max = 6
x2min = -5
x2max = 4
x3min = -5
x3max = 8
n = 15
m = 3
x_range = [[x1min, x1max], [x2min, x2max], [x3min, x3max]]

x_min_avg = sum([x1min, x2min, x3min]) / 3
x_max_avg = sum([x1max, x2max, x3max]) / 3

y_min = 200 + x_min_avg
y_max = 200 + x_max_avg
x_norm = [
    [-1, -1, -1, -1, 1, 1, 1, 1, -1.215, 1.215, 0, 0, 0, 0, 0],
    [-1, -1, 1, 1, -1, -1, 1, 1, 0, 0, -1.215, 1.215, 0, 0, 0],
```

```
    [-1, 1, -1, 1, -1, 1, -1, 1, 0, 0, 0, 0, -1.215, 1.215, 0],
    [1, 1, -1, -1, -1, -1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
    [1, -1, 1, -1, -1, 1, -1, 1, 0, 0, 0, 0, 0, 0, 0],
    [1, -1, -1, 1, 1, -1, -1, 1, 0, 0, 0, 0, 0, 0, 0],
    [-1, 1, 1, -1, 1, -1, -1, 1, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 1.4623, 1.4623, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1.4623, 1.4623, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1.4623, 1.4623, 0],
]

y1 = []
y2 = []
y3 = []
for i in range(15):
    y1.append(random.randint(int(y_min), int(y_max)))
    y2.append(random.randint(int(y_min), int(y_max)))
    y3.append(random.randint(int(y_min), int(y_max)))
# y1 = [196, 198, 196, 202, 202, 202, 198, 203, 203, 199, 201, 193, 203, 197, 195]
# y2 = [201, 195, 198, 203, 203, 196, 195, 198, 201, 200, 200, 202, 201, 203, 201]
# y3 = [194, 200, 193, 195, 200, 194, 193, 200, 198, 203, 195, 197, 198, 197, 203]

print("DOX Lab5")
print("Factors:")
print(
    tabulate(
        zip(*x_norm),
        headers=[
            "X1",
            "X2",
            "X3",
            "X12",
            "X13",
            "X23",
            "X123",
            "X1^2",
            "X2^2",
            "X3^2",
        ],
        floatfmt=".3f",
        tablefmt="fancy_grid",
    )
)


avg_y = get_avg([y1, y2, y3])

y_tab = [[y1[i], y2[i], y3[i], avg_y[i]] for i in range(15)]
print(
    tabulate(
        y_tab,
        headers=[
            "Y1",
            "Y2",
            "Y3",
            "Y_avg",
        ],
        floatfmt=".3f",
        tablefmt="fancy_grid",
    )
)
```

```
x = plan_matrix5(n, m, x_norm)
print("Naturalized factors:")
print(
    tabulate(
        zip(*x),
        headers=[
            "X1",
            "X2",
            "X3",
            "X12",
            "X13",
            "X23",
            "X123",
            "X1^2",
            "X2^2",
            "X3^2",
        ],
        floatfmt=".3f",
        tablefmt="fancy_grid",
    )
)

y_sum = sum(avg_y)
b = [0] * 11
b[0] = y_sum
for i in range(15):
    b[1] += avg_y[i] * x_norm[0][i]
    b[2] += avg_y[i] * x_norm[1][i]
    b[3] += avg_y[i] * x_norm[2][i]
    b[4] += avg_y[i] * x_norm[0][i] * x_norm[1][i]
    b[5] += avg_y[i] * x_norm[0][i] * x_norm[2][i]
    b[6] += avg_y[i] * x_norm[1][i] * x_norm[2][i]
    b[7] += avg_y[i] * x_norm[0][i] * x_norm[1][i] * x_norm[2][i]

b[8] = (b[1] ** 2) / b[0]
b[9] = (b[2] ** 2) / b[0]
b[10] = (b[3] ** 2) / b[0]
for i in range(11):
    b[i] = b[i] / 15

result = []
for i in range(15):
    result.append(
        b[0]
        + b[1] * x_norm[0][i]
        + b[2] * x_norm[1][i]
        + b[3] * x_norm[2][i]
        + b[4] * x_norm[3][i]
        + b[5] * x_norm[4][i]
        + b[6] * x_norm[5][i]
        + b[7] * x_norm[6][i]
        + b[8] * x_norm[7][i]
        + b[9] * x_norm[8][i]
        + b[10] * x_norm[9][i]
    )

print(
    "b0 = {:.3f}, b1 = {:.3f}, b2 = {:.3f}, b3 = {:.3f},"
    " b12 = {:.3f}, b13 = {:.3f}, b23 = {:.3f}, b123 = {:.3f}, b11 = {:.3f}, b22 =
{:.3f}, b33 = {:.3f}".format(
        b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7], b[8], b[9], b[10]
```

```
    )
)

print("Get the values of the factors from the factors table ")
for i in range(15):
    print(
        "{:.3f} + x1 * {:.3f}"
        " + x2 * {:.3f} + x3 * {:.3f}"
        " + x12 * {:.3f} + x13 * {:.3f}"
        " + x23 * {:.3f} + x123 * {:.3f}"
        " + x11 * {:.3f} + x22 * {:.3f}"
        " + x33 * {:.3f} = {:.3f}".format(
            b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7], b[8], b[9], b[10], result[i]
        )
    )

sigma = get_dispersion([y1, y2, y3], avg_y)

print("\nCochran test:\nDispersions:")
for i in range(15):
    print("d{} = {:.2f}".format(i + 1, sigma[i]))

gp = max(sigma) / sum(sigma)
print("\ngp = {:.4f}".format(gp))
f1 = m - 1
f2 = n
if gp < 0.3346:
    print("   Cochran’s C test passed")
else:
    print("   Cochran’s C test failed")

print("\n\nStudent's t-test")
sb = sum(sigma) / n
s2bs = sb / (n * m)
sbs = math.sqrt(s2bs)
print("Sb = {:.4f}\nBeta:".format(sb))

betha = []
for i in range(4):
    betha.append(0)
    for j in range(15):
        betha[i] += avg_y[j] * x_norm[i][j]
print(
    "b0 = {:.3f}\nb1 = {:.3f}\nb2 = {:.3f}\nb3 = {:.3f}".format(
        betha[0], betha[1], betha[2], betha[3]
    )
)


x_norm.insert(0, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
f1 = m - 1
f2 = n
f3 = f1 * f2
t = student(
    np.transpose(np.array(x_norm))[:, 1:],
    np.transpose(np.array([y1, y2, y3])),
    avg_y,
    15,
    3,
)
```

```
print("\nt:")

for i in range(10):
    print("t{} = {:.3f}".format(i, t[i]))
t_table = 2.042

y_ = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d = 0
nm = []
for i in range(11):
    if t[i] < t_table:
        nm.append(i)
        d += 1
    else:
        for j in range(15):
            if i == 0:
                y_[j] += b[i]
            else:
                y_[j] += b[i] * x[2][j]

print("\nCheck values without excluded factors:")
for i in range(10):
    print("y{} = {:.3f}".format(i + 1, y_[i]))

print("F-test")
print("\nd = {}".format(d))
s2ad = (m / (n - d)) * sum((y_[i] - avg_y[i]) ** 2 for i in range(4))
print("S2ad = {:.3f}".format(s2ad))

Fp = s2ad / sb
f4 = n - d
print("Fp =", Fp)

F8_table = 2.16
if Fp < F8_table:
    print("  F-test passed/model is adequate")
else:
    print("  F-test failed/model is NOT adequate")
```

Результат виконання роботи:

DOX Lab5
Factors:

X1	X2	X3	X12	X13	X23	X123	X1^2	X2^2	X3^2
-1.000	-1.000	-1.000	1	1	1	-1	1.000	1.000	1.000
-1.000	-1.000	1.000	1	-1	-1	1	1.000	1.000	1.000
-1.000	1.000	-1.000	-1	1	-1	1	1.000	1.000	1.000
-1.000	1.000	1.000	-1	-1	1	-1	1.000	1.000	1.000
1.000	-1.000	-1.000	-1	-1	1	1	1.000	1.000	1.000
1.000	-1.000	1.000	-1	1	-1	-1	1.000	1.000	1.000
1.000	1.000	-1.000	1	-1	-1	-1	1.000	1.000	1.000
1.000	1.000	1.000	1	1	1	1	1.000	1.000	1.000
-1.215	0.000	0.000	0	0	0	0	1.462	0.000	0.000
1.215	0.000	0.000	0	0	0	0	1.462	0.000	0.000
0.000	-1.215	0.000	0	0	0	0	0.000	1.462	0.000
0.000	1.215	0.000	0	0	0	0	0.000	1.462	0.000
0.000	0.000	-1.215	0	0	0	0	0.000	0.000	1.462
0.000	0.000	1.215	0	0	0	0	0.000	0.000	1.462
0.000	0.000	0.000	0	0	0	0	0.000	0.000	0.000

Y1	Y2	Y3	Y_avg
198	200	195	197.667
205	195	200	200.000
194	194	200	196.000
204	205	196	201.667
199	206	203	202.667
195	198	202	198.333
196	200	201	199.000
200	194	206	200.000
195	194	203	197.333
200	196	206	200.667
201	200	197	199.333
198	204	206	202.667
200	198	200	199.333
204	197	201	200.667
204	204	200	202.667

Naturalized factors:

X1	X2	X3	X12	X13	X23	X123	X1^2	X2^2	X3^2

-8.000	-5.000	-5.000	40.000	40.000	25.000	-200.000	64.000	25.000	25.000
-8.000	-5.000	8.000	40.000	-64.000	-40.000	320.000	64.000	25.000	64.000
-8.000	4.000	-5.000	-32.000	40.000	-20.000	160.000	64.000	16.000	25.000
-8.000	4.000	8.000	-32.000	-64.000	32.000	-256.000	64.000	16.000	64.000
6.000	-5.000	-5.000	-30.000	-30.000	25.000	150.000	36.000	25.000	25.000
6.000	-5.000	8.000	-30.000	48.000	-40.000	-240.000	36.000	25.000	64.000
6.000	4.000	-5.000	24.000	-30.000	-20.000	-120.000	36.000	16.000	25.000
6.000	4.000	8.000	24.000	48.000	32.000	192.000	36.000	16.000	64.000
-9.505	-0.500	1.500	4.752	-14.258	-0.750	7.129	90.345	0.250	2.250
7.505	-0.500	1.500	-3.753	11.258	-0.750	-5.629	56.325	0.250	2.250
-1.000	-5.968	1.500	5.968	-1.500	-8.951	8.951	1.000	35.611	2.250
-1.000	4.968	1.500	-4.968	-1.500	7.451	-7.451	1.000	24.676	2.250
-1.000	-0.500	-6.398	0.500	6.398	3.199	-3.199	1.000	0.250	40.928
-1.000	-0.500	9.398	0.500	-9.398	-4.699	4.699	1.000	0.250	88.313
-1.000	-0.500	1.500	0.500	-1.500	-0.750	0.750	1.000	0.250	2.250

b0 = 199.867, b1 = 0.581, b2 = 0.137, b3 = 0.419, b12 = -0.133, b13 = -0.756, b23 = 0.578, b123 = 0.133, b11 = 0.002, b22 = 0.000, b33 = 0.001
Get the values of the factors from the factors table
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 198.288
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 199.748
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 197.939
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 201.177
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 201.495
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 199.400
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 200.079
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 200.828
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 199.163
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 200.575
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 199.701
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 200.033
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 199.359
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 200.377
199.867 + x1 * 0.581 + x2 * 0.137 + x3 * 0.419 + x12 * -0.133 + x13 * -0.756 + x23 * 0.578 + x123 * 0.133 + x11 * 0.002 + x22 * 0.000 + x33 * 0.001 = 199.867

Cochran test:
Dispersions:
d1 = 4.22
d2 = 16.67
d3 = 8.00
d4 = 16.22
d5 = 8.22
d6 = 8.22
d7 = 4.67
d8 = 24.00

```
d9 = 16.22
d10 = 16.89
d11 = 2.89
d12 = 11.56
d13 = 0.89
d14 = 8.22
d15 = 3.56

gp = 0.1595
  Cochran’s C test passed


Student's t-test
Sb = 10.0296
Beta:
b0 = 8.717
b1 = 2.050
b2 = 6.287
b3 = -2.000

t:
t0 = 423.354
t1 = 1.231
t2 = 0.289
t3 = 0.888
t4 = 0.282
t5 = 1.600
t6 = 1.224
t7 = 0.282
t8 = 307.465
t9 = 308.291

Check values without excluded factors:
y1 = 199.853
y2 = 199.888
y3 = 199.853
y4 = 199.888
y5 = 199.853
y6 = 199.888
y7 = 199.853
y8 = 199.888
y9 = 199.871
y10 = 199.871

F-test
d = 7
S2ad = 8.552
Fp = 0.8527105062767936
  F-test passed/model is adequate
```

Висновки:

Під час виконання лабораторної роботи було проведено трьохфакторний експеримент з урахуванням квадратичних членів використовуючи центральний ортогональний композиційний план, проведено 3 статистичні перевірки, знайдено рівняння регресії, що адекватно для опису об’єкту. Закріплено отримані знання практичним їх використанням при написанні програми, що реалізує завдання на лабораторну роботу.