Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №6
З дисципліни «Методи наукових досліджень»
За темою:
«Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

ВИКОНАВ:
Студент ІІ курсу ФІОТ
Групи ІВ-91
Гутов В.В.
Номер у списку - 8

ПЕРЕВІРИВ:
асистент
Регіда П.Г.

Київ 2021 р.

Мета: Провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.

Завдання:

1. Ознайомитися з теоретичними відомостями.
2. Вибрати з таблиці варіантів і записати в протокол інтервали значень x_1, x_2, x_3. Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1;+ l; - l; 0 для $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$.
3. Значення функції відгуку знайти за допомогою підстановки в формулу:

$$y_i = f(x_1, x_2, x_3) + random(10)-5,$$

де $f(x_1, x_2, x_3)$ вибирається по номеру в списку в журналі викладача.

4. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках використовувати натуральні значення факторів.
5. Зробити висновки по виконаній роботі.

Таблиця варіантів

№ варіанту	x_1		x_2		x_3		$f(x_1, x_2, x_3)$
	min	max	min	max	min	max	
108	-30	0	-35	10	0	20	5,4+3,6*x1+6,6*x2+7,7*x3+8,0*x1*x1+0,3*x2*x2+2,5*x3*x3+5,9*x1*x2+0,3*x1*x3+7,2*x2*x3+5,3*x1*x2*x3

Програмний код

```
import random
import numpy as np
import math
from beautifultable import BeautifulTable
from numpy.linalg import solve
from scipy.stats import f, t


def main(n, m):
    x1_min = -30
    x1_max = 0
    x2_min = -35
    x2_max = 10
    x3_min = 0
    x3_max = 20
    x01 = (x1_max + x1_min) / 2
    x02 = (x2_max + x2_min) / 2
    x03 = (x3_max + x3_min) / 2

    dx1 = x1_max - x01
    dx2 = x2_max - x02
    dx3 = x3_max - x03

    xn = [[-1, -1, -1, +1, +1, +1, -1, +1, +1, +1],
          [-1, -1, +1, +1, -1, -1, +1, +1, +1, +1],
          [-1, +1, -1, -1, +1, -1, +1, +1, +1, +1],
          [-1, +1, +1, -1, -1, +1, -1, +1, +1, +1],
          [+1, -1, -1, -1, -1, +1, +1, +1, +1, +1],
          [+1, -1, +1, -1, +1, -1, -1, +1, +1, +1],
          [+1, +1, -1, +1, -1, -1, -1, +1, +1, +1],
          [+1, +1, +1, +1, +1, +1, +1, +1, +1, +1],
          [-1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
          [+1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
          [0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],
          [0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],
          [0, 0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929],
          [0, 0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

    x1 = [x1_min, x1_min, x1_min, x1_min, x1_max, x1_max, x1_max, x1_max, -
1.73 * dx1 + x01, 1.73 * dx1 + x01, x01, x01, x01, x01, x01]
    x2 = [x2_min, x2_min, x2_max, x2_max, x2_min, x2_min, x2_max, x2_max,
x02, x02, -1.73 * dx2 + x02, 1.73 * dx2 + x02, x02, x02, x02]
    x3 = [x3_min, x3_max, x3_min, x3_max, x3_min, x3_max, x3_min, x3_max,
```

```
x03, x03, x03, x03, -1.73 * dx3 + x03, 1.73 * dx3 + x03, x03]

    x1x2 = [0] * 15
    x1x3 = [0] * 15
    x2x3 = [0] * 15
    x1x2x3 = [0] * 15
    x1kv = [0] * 15
    x2kv = [0] * 15
    x3kv = [0] * 15

    for i in range(15):
        x1x2[i] = x1[i] * x2[i]
        x1x3[i] = x1[i] * x3[i]
        x2x3[i] = x2[i] * x3[i]
        x1x2x3[i] = x1[i] * x2[i] * x3[i]
        x1kv[i] = x1[i] ** 2
        x2kv[i] = x2[i] ** 2
        x3kv[i] = x3[i] ** 2

    tmp_list_a = list(zip(x1, x2, x3, x1x2, x1x3, x2x3, x1x2x3, x1kv, x2kv,
x3kv))

    plan_table = BeautifulTable()
    plan_table.columns.header = ['X1', 'X2', 'X3', 'X1X2', 'X1X3', 'X2X3',
'X1X2X3', 'X1X1', 'X2X2', 'X3X3']
    print("Planning matrix with naturalized coefficients X:")
    for i in range(len(tmp_list_a)):
        plan_table.rows.append(tmp_list_a[i])
    print(plan_table)

    def func(X1, X2, X3):
        y = 5.4 + 3.6*X1 + 6.6*X2 + 7.7*X3 + 8.0*X1*X1 + 0.3*X2*X2 +
2.5*X3*X3 + 5.9*X1*X2 + 0.3*X1*X3 + 7.2*X2*X3 + 5.3*X1*X2*X3 +
random.randint(0, 10) - 5
        return y

    y = [[func(tmp_list_a[j][0], tmp_list_a[j][1], tmp_list_a[j][2]) for _ in
range(m)] for j in range(15)]

    plan_y = BeautifulTable()
    plan_y.columns.header = ['y1', 'y2', 'y3']
    print("Planning matrix y:")
    for i in range(len(y)):
        plan_y.rows.append(y[i])
    print(plan_y)

    aver_y = []
    for i in range(len(y)):
        aver_y.append(np.mean(y[i], axis=0))
    print("Average response values:\n{}".format(aver_y))

    disp = []
    for i in range(len(y)):
        a = 0
        for k in y[i]:
            a += (k - np.mean(y[i], axis=0)) ** 2
        disp.append(a / len(y[i]))
    print("Dispersion:\n{}".format(disp))

    def finds_value(num):
        a = 0
        for j in range(15):
            a += aver_y[j] * tmp_list_a[j][num - 1] / 15
        return a
```

```
    def a(f, s):
        a = 0
        for j in range(15):
            a += tmp_list_a[j][f - 1] * tmp_list_a[j][s - 1] / 15
        return a

    my = sum(aver_y) / 15
    mx = []
    for i in range(10):
        number_lst = []
        for j in range(15):
            number_lst.append(tmp_list_a[j][i])
        mx.append(sum(number_lst) / len(number_lst))

    determinant1 = [[1, mx[0], mx[1], mx[2], mx[3], mx[4], mx[5], mx[6],
mx[7], mx[8], mx[9]],
                    [mx[0], a(1, 1), a(1, 2), a(1, 3), a(1, 4), a(1, 5), a(1,
6), a(1, 7), a(1, 8), a(1, 9), a(1, 10)],
                    [mx[1], a(2, 1), a(2, 2), a(2, 3), a(2, 4), a(2, 5), a(2,
6), a(2, 7), a(2, 8), a(2, 9), a(2, 10)],
                    [mx[2], a(3, 1), a(3, 2), a(3, 3), a(3, 4), a(3, 5), a(3,
6), a(3, 7), a(3, 8), a(3, 9), a(3, 10)],
                    [mx[3], a(4, 1), a(4, 2), a(4, 3), a(4, 4), a(4, 5), a(4,
6), a(4, 7), a(4, 8), a(4, 9), a(4, 10)],
                    [mx[4], a(5, 1), a(5, 2), a(5, 3), a(5, 4), a(5, 5), a(5,
6), a(5, 7), a(5, 8), a(5, 9), a(5, 10)],
                    [mx[5], a(6, 1), a(6, 2), a(6, 3), a(6, 4), a(6, 5), a(6,
6), a(6, 7), a(6, 8), a(6, 9), a(6, 10)],
                    [mx[6], a(7, 1), a(7, 2), a(7, 3), a(7, 4), a(7, 5), a(7,
6), a(7, 7), a(7, 8), a(7, 9), a(7, 10)],
                    [mx[7], a(8, 1), a(8, 2), a(8, 3), a(8, 4), a(8, 5), a(8,
6), a(8, 7), a(8, 8), a(8, 9), a(8, 10)],
                    [mx[8], a(9, 1), a(9, 2), a(9, 3), a(9, 4), a(9, 5), a(9,
6), a(9, 7), a(9, 8), a(9, 9), a(9, 10)],
                    [mx[9], a(10, 1), a(10, 2), a(10, 3), a(10, 4), a(10, 5),
a(10, 6), a(10, 7), a(10, 8), a(10, 9), a(10, 10)]]

    determinant2 = [my, finds_value(1), finds_value(2), finds_value(3),
finds_value(4), finds_value(5), finds_value(6), finds_value(7),
                    finds_value(8), finds_value(9), finds_value(10)]

    beta = solve(determinant1, determinant2)
    print("Regression equation:")
    print("y = {} + {} * X1 + {} * X2 + {} * X3 + {} * X1X2 + {} * X1X3 + {}
* X2X3"
          "+ {} * X1X2X3 + {} * X11^2 + {} * X22^2 + {} * X33^2"
          .format(beta[0], beta[1], beta[2], beta[3], beta[4], beta[5],
beta[6], beta[7], beta[8], beta[9], beta[10]))
    y_i = [0] * 15


    for k in range(15):
        y_i[k] = beta[0] + beta[1] * tmp_list_a[k][0] + beta[2] *
tmp_list_a[k][1] + beta[3] * tmp_list_a[k][2] + \
                 beta[4] * tmp_list_a[k][3] + beta[5] * tmp_list_a[k][4] +
beta[6] * tmp_list_a[k][5] + beta[7] * \
                 tmp_list_a[k][6] + beta[8] * tmp_list_a[k][7] + beta[9] *
tmp_list_a[k][8] + beta[10] * tmp_list_a[k][9]

    print("Experimental values:\n{}".format(y_i))

    gp = max(disp) / sum(disp)
    gt = 0.3346
```

```
    print("\nKohren check\nGp = {}".format(gp))
    if gp < gt:
        print("Dispersions are homogeneous")
    else:
        print("Dispersions are inhomogeneous")

    sb = sum(disp) / len(disp)
    sbs = (sb / (15 * m)) ** 0.5

    f3 = (m - 1) * n
    sign_coef = []
    insign_coef = []
    d = 11
    res = [0] * 11

    for j in range(11):
        t_pract = 0
        for i in range(15):
            if j == 0:
                t_pract += aver_y[i] / 15
            else:
                t_pract += aver_y[i] * xn[i][j - 1]
            res[j] = beta[j]
        if math.fabs(t_pract / sbs) < t.ppf(q=0.975, df=f3):
            insign_coef.append(beta[j])
            res[j] = 0
            d-=1
        else:
            sign_coef.append(beta[j])
    print("\nStudent criterion:")
    print("Significant regression coefficients:", [round(i, 3) for i in
sign_coef])
    print("Insignificant regression coefficients:", [round(i, 3) for i in
insign_coef])
    y_st = []
    for i in range(15):
        y_st.append(res[0] + res[1] * x1[i] + res[2] * x2[i] + res[3] * x3[i]
+ res[4] * x1x2[i] + res[5] *
                    x1x3[i] + res[6] * x2x3[i] + res[7] * x1x2x3[i] + res[8]
* x1kv[i] + res[9] *
                    x2kv[i] + res[10] * x3kv[i])
    print("Values with the coefficients:\n{}".format(y_st))

    print("\nFisher adequacy check")
    sad = m * sum([(y_st[i] - aver_y[i]) ** 2 for i in range(15)]) / (n - d)
    fp = sad / sb
    f4 = n - d
    print("fp =", fp)
    if fp < f.ppf(q=0.95, dfn=f4, dfd=f3):
        print("The mathematical model is adequate to the experimental data")
    else:
        print("The mathematical model is inadequate to the experimental
data")


main(15, 3)
```

Результати роботи програми

```
"D:\4 semestr\MND\venv\Scripts\python.exe" "D:/4 semestr/MND/lab6/main_lab6.py"
Planning matrix with naturalized coefficients X:
```

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3
-30	-35	0	1050	0	0	0	900	1225	0
-30	-35	20	1050	-600	-700	21000	900	1225	400
-30	10	0	-300	0	0	0	900	100	0
-30	10	20	-300	-600	200	-6000	900	100	400
0	-35	0	0	0	0	0	0	1225	0
0	-35	20	0	0	-700	0	0	1225	400
0	10	0	0	0	0	0	0	100	0
0	10	20	0	0	200	0	0	100	400
-40.95	-12.5	10.0	511.875	-409.5	-125.0	5118.75	1676.903	156.25	100.0
10.95	-12.5	10.0	-136.875	109.5	-125.0	-1368.75	119.902	156.25	100.0
-15.0	-51.425	10.0	771.375	-150.0	-514.25	7713.75	225.0	2644.531	100.0
-15.0	26.425	10.0	-396.375	-150.0	264.25	-3963.75	225.0	698.281	100.0
-15.0	-12.5	-7.3	187.5	109.5	91.25	-1368.75	225.0	156.25	53.29
-15.0	-12.5	27.3	187.5	-409.5	-341.25	5118.75	225.0	156.25	745.29
-15.0	-12.5	10.0	187.5	-150.0	-125.0	1875.0	225.0	156.25	100.0

Planning matrix y:

y1	y2	y3
13429.9	13428.9	13433.9
120666.9	120663.9	120659.9
5422.4	5422.4	5426.4
-23963.6	-23960.6	-23960.6
143.9	144.9	140.9
-3748.1	-3741.1	-3741.1
97.4	105.4	105.4
2698.4	2694.4	2693.4
42688.163	42688.163	42691.163
-7637.672	-7633.672	-7629.672
44222.742	44217.742	44219.742
-19023.598	-19028.598	-19022.598
-3665.485	-3669.485	-3663.485

29441.985	29447.985	29444.985
12145.525	12136.525	12141.525

```
Average response values:
[13430.9, 120663.56666666665, 5423.733333333333, -23961.599999999995, 143.23333333333335, -3743.433333333333, 102.73333333333335, 2695.4, 42689.162500000006, -7633.6725, 44220.07502083333, -19024.93
Dispersion:
[4.666666666666667, 8.222222222222221, 3.555555555555556, 2.0, 2.8888888888888893, 10.888888888888888, 14.222222222222223, 4.666666666666667, 2.0, 10.666666666666666, 4.222222222222222, 6.8888888888
Regression equation:
y = 5.967353305845056 + 3.632359212015776 * X1 + 6.6450604520606555 * X2 + 7.629952331723344 * X3 + 5.901234567901013 * X1X2 + 0.29765432098760347 * X1X3 + 7.1992592592595175 * X2X3+ 5.2999012345679
Experimental values:
[13429.875934617748, 120662.9928579734, 5423.005728170344, -23961.877348474067, 142.77538739626954, -3743.441022581677, 102.571847615231, 2695.68877097085, 42690.30833250428, -7633.835614189243, 442
```

```
Kohren check
Gp = 0.141280353200883
Dispersions are homogeneous

Student criterion:
Significant regression coefficients: [5.967, 3.632, 6.645, 7.63, 5.901, 0.298, 7.199, 5.3, 8.0, 0.302, 2.502]
Insignificant regression coefficients: []
Values with the coefficients:
[13429.875934617748, 120662.9928579734, 5423.005728170344, -23961.877348474067, 142.77538739626954, -3743.441022

Fisher adequacy check
fp = 0.6018769933767836
The mathematical model is adequate to the experimental data

Process finished with exit code 0
```

Висновок

Виконуючи дану лабораторну роботу, я провів трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план. Склав матрицю планування та знайшов коефіцієнти рівняння регресії, провів статистичні перевірки.

Результати роботи програми, наведені вище, підтверджують правильність виконання завдання. Під час виконання роботи проблем не виникло.