Міністерство освіти і науки України
Національний технічний університет України «КПІ» імені Ігоря Сікорського
Кафедра обчислювальної техніки ФІОТ

Звіт

з лабораторної роботи №6

з навчальної дисципліни «Методи оптимізації та планування експерименту»

Тема: Проведення трьохфакторного експерименту при використанні рівняннярегресії з урахуванням квадратичних членів (центральний ортогональнийкомпозиційний план)

Виконав:
Студент 2 курсу кафедри ОТ ФІОТ
Навчальної групи ІО-91
Тарасенко Андрій

Перевірив:
Регіда П. Г.

Київ 2021

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 x1, x2, x3. Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1; +l; -l; 0 для x1, x2, x3.

3. Значення функції відгуку знайти за допомогою підстановки в формулу:

$$y_i = f(x_1, x_2, x_3) + \text{random}(10)-5,$$

де f(x1, x2, x3) 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.

Варіант:

122	-5	15	10	60	10	20	9,2+6,0*x1+3,2*x2+3,2*x3+6,6*x1*x1+0,1*x2*x2+4,9*x3*x3+6,9*x1*x2+0,9*x1*x3+3,5*x2*x3+0,9*x1*x2*x3

Лістинг програми

```
"""
 * Copyright © 2021 drewg3r
 * https://github.com/drewg3r/DOX-labs
main.py: main file to run the program.
"""

import random
import numpy as np
import math
from _decimal import Decimal
from functools import reduce
from itertools import compress
from scipy.stats import f, t
from tabulate import tabulate


def func(x1, x2, x3):
    coef = [9.2, 6.0, 3.2, 3.2, 6.9, 0.9, 3.5, 0.9, 6.6, 0.1, 4.9]
    return regression_equation(x1, x2, x3, coef)


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(x_norm):
    l = 1.73
    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 x.tolist()


def generate_factors_table(raw_array):
    raw_list = [row + [row[0] * row[1], row[0] * row[2], row[1] * row[2], row[0] *
row[1] * row[2]] + list(map(lambda x: x ** 2, row)) for row in raw_array]
    return list(map(lambda row: list(map(lambda el: round(el, 3), row)), raw_list))


def generate_y(m, factors_table):
    return [[round(func(row[0], row[1], row[2]) + random.randint(-5, 5), 3) for _ in
range(m)] for row in factors_table]


def cochran_criteria(m, N, y_table):
    def get_cochran_value(f1, f2, q):
        partResult1 = q / f2
        params = [partResult1, f1, (f2 - 1) * f1]
        fisher = f.isf(*params)
        result = fisher / (fisher + (f2 - 1))
        return Decimal(result).quantize(Decimal('.0001')).__float__()
    y_variations = [np.var(i) for i in y_table]
    max_y_variation = max(y_variations)
    gp = max_y_variation/sum(y_variations)
    f1 = m - 1
    f2 = N
    p = 0.95
    q = 1-p
    gt = get_cochran_value(f1, f2, q)
    print("Gp = {}, Gt = {}".format(gp, gt))
    if gp < gt:
        print("  Cochran’s C test passed")
        return True
    else:
        print("  Cochran’s C test failed")
        return False


def set_factors_table(factors_table):
    def x_i(i):
        with_null_factor = list(map(lambda x: [1] + x,
generate_factors_table(factors_table)))
        res = [row[i] for row in with_null_factor]
        return np.array(res)
    return x_i


def m_ij(*arrays):
    return np.average(reduce(lambda accum, el: accum*el, list(map(lambda el:
np.array(el), arrays))))


def find_coefficients(factors, y_vals):
    x_i = set_factors_table(factors)
    coefficients = [[m_ij(x_i(column), x_i(row)) for column in range(11)] for row in
range(11)]
    y_numpy = list(map(lambda row: np.average(row), y_vals))
    free_values = [m_ij(y_numpy, x_i(i)) for i in range(11)]
    beta_coefficients = np.linalg.solve(coefficients, free_values)
```

```
    return list(beta_coefficients)


def print_equation(coefficients, importance=[True]*11):
    x_i_names = list(compress(["", "*x1", "*x2", "*x3", "*x12", "*x13", "*x23", "*x123",
"*x1^2", "*x2^2", "*x3^2"], importance))
    coefficients_to_print = list(compress(coefficients, importance))
    equation = " ".join(["".join(i) for i in zip(list(map(lambda x: "{:+.2f}".format(x),
coefficients_to_print)), x_i_names)])
    print("Regression equation: y = " + equation)


def student_criteria(m, N, y_table, beta_coefficients):
    def get_student_value(f3, q):
        return Decimal(abs(t.ppf(q / 2, f3))).quantize(Decimal('.0001')).__float__()
    average_variation = np.average(list(map(np.var, y_table)))
    x_i = set_factors_table(natural_plan)
    variation_beta_s = average_variation/N/m
    standard_deviation_beta_s = math.sqrt(variation_beta_s)
    t_i = np.array([abs(beta_coefficients[i])/standard_deviation_beta_s for i in
range(len(beta_coefficients))])
    f3 = (m-1)*N
    q = 0.05
    t_our = get_student_value(f3, q)
    importance = [1 if el > t_our else 0 for el in list(t_i)]
    d = sum(importance)
    # print result data
    print("βs: " + ", ".join(list(map(lambda x: str(round(float(x), 3)),
beta_coefficients))))
    print("ts: " + ", ".join(list(map(lambda i: "{:.2f}".format(i), t_i))))
    print("f3 = {}; q = {}; t_table = {}".format(f3, q, t_our))
    print("d =", d)
    print_equation(beta_coefficients, importance)
    return importance


def regression_equation(x1, x2, x3, coef, importance = [True]*11):
    factors_array = [1, x1, x2, x3, x1*x2, x1*x3, x2*x3, x1*x2*x3, x1**2, x2**2, x3**2]
    return sum([el[0]*el[1] for el in compress(zip(coef, factors_array), importance)])


def fisher_criteria(m, N, d, x_table, y_table, b_coefficients, importance):
    def get_fisher_value(f3, f4, q):
        return Decimal(abs(f.isf(q, f4, f3))).quantize(Decimal('.0001')).__float__()
    f3 = (m - 1) * N
    f4 = N - d
    q = 0.05
    theoretical_y = np.array([regression_equation(row[0], row[1], row[2],
b_coefficients) for row in x_table])
    # print(theoretical_y)
    average_y = np.array(list(map(lambda el: np.average(el), y_table)))
    s_ad = m/(N-d) * sum((theoretical_y - average_y)**2)
    # print(s_ad)
    y_variations = np.array(list(map(np.var, y_table)))
    s_v = np.average(y_variations)
    f_p = float(s_ad/s_v)
    f_t = get_fisher_value(f3, f4, q)
    theoretical_values_to_print = list(zip(map(lambda x: "x1 = {0[1]:<10} x2 =
{0[2]:<10} x3 = {0[3]:<10}".format(x), x_table), theoretical_y))
    print("Fp = {}, Ft = {}".format(f_p, f_t))
    if f_p < f_t:
```

```
        print("   F-test passed/model is adequate")
    else:
        print("   F-test failed/model is NOT adequate")
    return True if f_p < f_t else False


l = 1.73
x1min = -5
x1max = 15
x2min = 10
x2max = 60
x3min = 10
x3max = 20
x_norm = [[-1, -1, -1, -1, 1, 1, 1, 1, -1.73, 1.73, 0, 0, 0, 0],
          [-1, -1, 1, 1, -1, -1, 1, 1, 0, 0, -1.73, 1.73, 0, 0],
          [-1, 1, -1, 1, -1, 1, -1, 1, 0, 0, 0, 0, -1.73, 1.73],
          [1, 1, -1, -1, -1, -1, 1, 1, 0, 0, 0, 0, 0, 0],
          [1, -1, 1, -1, -1, 1, -1, 1, 0, 0, 0, 0, 0, 0],
          [1, -1, -1, 1, 1, -1, -1, 1, 0, 0, 0, 0, 0, 0],
          [-1, 1, 1, -1, 1, -1, -1, 1, 0, 0, 0, 0, 0, 0],
          [1, 1, 1, 1, 1, 1, 1, 1, 2.9929, 2.9929, 0, 0, 0, 0],
          [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2.9929, 2.9929, 0, 0],
          [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2.9929, 2.9929]]

x_range = [[x1min, x1max], [x2min, x2max], [x3min, x3max]]
x_nat = plan_matrix5(x_norm)
m = 3
N = 14
x_norm = np.transpose(np.array(x_norm))
natural_plan = generate_factors_table(x_nat)
y_arr = generate_y(m, x_nat)
print("DOX Lab6")
print("Factors:")
print(
    tabulate(
        x_norm,
        headers=[
            "X1",
            "X2",
            "X3",
            "X12",
            "X13",
            "X23",
            "X123",
            "X1^2",
            "X2^2",
            "X3^2",
        ],
        floatfmt=".3f",
        tablefmt="fancy_grid",
    )
)

print(
    tabulate(
        y_arr,
        headers=[
            "Y1",
            "Y2",
            "Y3",
            "Y_avg",
```

```
        ],
        floatfmt=".3f",
        tablefmt="fancy_grid",
    )
)

print("Naturalized factors:")
print(
    tabulate(
        x_nat,
        headers=[
            "X1",
            "X2",
            "X3",
            "X12",
            "X13",
            "X23",
            "X123",
            "X1^2",
            "X2^2",
            "X3^2",
        ],
        floatfmt=".3f",
        tablefmt="fancy_grid",
    )
)

while not cochran_criteria(m, N, y_arr):
    m += 1
y_arr = generate_y(m, natural_plan)
coefficients = find_coefficients(natural_plan, y_arr)
print_equation(coefficients)
importance = student_criteria(m, N, y_arr, coefficients)
d = len(list(filter(None, importance)))
fisher_criteria(m, N, d, natural_plan, y_arr, coefficients, importance)
```

Результат виконання роботи

Запишемо лінійне рівняння регресії:

$$\hat{y} = \varphi(x_0, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3, x_1^2, x_2^2, x_3^2, b_0, b_1, b_2, b_3, b_{12}, b_{13}, b_{23}, b_{123}, b_{11}, b_{22}, b_{33})$$

$$\hat{y} = b_0x_0 + b_1x_i + b_2x_2 + b_3x_3 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 + b_{123}x_1x_2x_3 + b_{11}x_1^2 + b_{22}x_2^2 + b_{33}x_3^2$$

$$\varphi_i = b_0x_{0i} + b_1x_{1i} + b_2x_{2i} + b_3x_{3i} + b_{12}x_{1i}x_{2i} + b_{13}x_{1i}x_{3i} + b_{23}x_{2i}x_{3i} + b_{123}x_{1i}x_{2i}x_{3i} + b_{11}x_{1i}^2 + b_{22}x_{2i}^2 + b_{33}x_{3i}^2$$

DOX Lab6
Factors:

X1	X2	X3	X12	X13	X23	X123	X1^2	X2^2	X3^2
-1.000	-1.000	-1.000	1.000	1.000	1.000	-1.000	1.000	1.000	1.000
-1.000	-1.000	1.000	1.000	-1.000	-1.000	1.000	1.000	1.000	1.000
-1.000	1.000	-1.000	-1.000	1.000	-1.000	1.000	1.000	1.000	1.000
-1.000	1.000	1.000	-1.000	-1.000	1.000	-1.000	1.000	1.000	1.000
1.000	-1.000	-1.000	-1.000	-1.000	1.000	1.000	1.000	1.000	1.000
1.000	-1.000	1.000	-1.000	1.000	-1.000	-1.000	1.000	1.000	1.000
1.000	1.000	-1.000	1.000	-1.000	-1.000	-1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
-1.730	0.000	0.000	0.000	0.000	0.000	0.000	2.993	0.000	0.000
1.730	0.000	0.000	0.000	0.000	0.000	0.000	2.993	0.000	0.000
0.000	-1.730	0.000	0.000	0.000	0.000	0.000	0.000	2.993	0.000
0.000	1.730	0.000	0.000	0.000	0.000	0.000	0.000	2.993	0.000
0.000	0.000	-1.730	0.000	0.000	0.000	0.000	0.000	0.000	2.993
0.000	0.000	1.730	0.000	0.000	0.000	0.000	0.000	0.000	2.993

Y1	Y2	Y3
219.200	217.200	222.200
1572.200	1575.200	1572.200
-1491.800	-1500.800	-1496.800
-640.800	-641.800	-634.800
5016.200	5022.200	5022.200
8358.200	8360.200	8350.200
19198.200	19207.200	19199.200

31039.200	31038.200	31036.200
-4794.836	-4793.836	-4786.836
22870.864	22874.864	22866.864
131.981	124.981	122.981
14370.531	14373.531	14372.531
3668.675	3669.675	3675.675
11194.985	11195.985	11185.985

Naturalized factors:

X1	X2	X3	X12	X13	X23	X123	X1^2	X2^2	X3^2
-5.000	10.000	10.000	-50.000	-50.000	100.000	-500.000	25.000	100.000	100.000
-5.000	10.000	20.000	-50.000	-100.000	200.000	-1000.000	25.000	100.000	400.000
-5.000	60.000	10.000	-300.000	-50.000	600.000	-3000.000	25.000	3600.000	100.000
-5.000	60.000	20.000	-300.000	-100.000	1200.000	-6000.000	25.000	3600.000	400.000
15.000	10.000	10.000	150.000	150.000	100.000	1500.000	225.000	100.000	100.000
15.000	10.000	20.000	150.000	300.000	200.000	3000.000	225.000	100.000	400.000
15.000	60.000	10.000	900.000	150.000	600.000	9000.000	225.000	3600.000	100.000
15.000	60.000	20.000	900.000	300.000	1200.000	18000.000	225.000	3600.000	400.000
-12.300	35.000	15.000	-430.500	-184.500	525.000	-6457.500	151.290	1225.000	225.000
22.300	35.000	15.000	780.500	334.500	525.000	11707.500	497.290	1225.000	225.000
5.000	-8.250	15.000	-41.250	75.000	-123.750	-618.750	25.000	68.062	225.000
5.000	78.250	15.000	391.250	75.000	1173.750	5868.750	25.000	6123.062	225.000
5.000	35.000	6.350	175.000	31.750	222.250	1111.250	25.000	1225.000	40.322
5.000	35.000	23.650	175.000	118.250	827.750	4138.750	25.000	1225.000	559.322

Gp = 0.14108527131782944, Gt = 0.3517
  Cochran’s C test passed
Regression equation: y = -283.62 +9.69*x1 +7.21*x2 +45.91*x3 +6.89*x12 +0.90*x13 +3.49*x23 +0.90*x123 +6.24*x1^2 +0.04*x2^2 +3.48*x3^2
βs: -283.621, 9.688, 7.209, 45.91, 6.887, 0.898, 3.491, 0.901, 6.242, 0.045, 3.483
ts: 661.78, 22.61, 16.82, 107.12, 16.07, 2.10, 8.15, 2.10, 14.56, 0.10, 8.13
f3 = 28; q = 0.05; t_table = 2.0484
d = 10
Regression equation: y = -283.62 +9.69*x1 +7.21*x2 +45.91*x3 +6.89*x12 +0.90*x13 +3.49*x23 +0.90*x123 +6.24*x1^2 +3.48*x3^2
Fp = 0.5381800221735101, Ft = 2.7141
  F-test passed/model is adequate

Висновки:

Під часвиконання лабораторної роботи було проведено трьохфакторний експеримент і отримано адекватну модель –рівняння регресії, використовуючи рототабельний композиційний план.