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

Звіт

з лабораторної роботи №3

з навчальної дисципліни «Методи оптимізації та планування експерименту»

Тема: Проведення трьохфакторного експерименту з використанням лінійного рівняння регресії.

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

Перевірив:
Регіда П. Г.

Київ 2021

Мета: провести дробовий трьохфакторний експеримент. Скласти матрицю планування, знайти коефіцієнти рівняння регресії, провести 3 статистичні перевірки.

Завдання:

1. Скласти матрицю планування для дробового трьохфакторного експерименту. Провести експеримент в усіх точках факторного простору, повторивши N експериментів, де N – кількість експериментів (рядків матриці планування) в усіх точках факторного простору – знайти значення функції відгуку У. Значення функції відгуку знайти у відповідності з варіантом діапазону, зазначеного далі (випадковим чином).

2. Знайти коефіцієнти лінійного рівняння регресії. Записати лінійне рівняння регресії.

3. Провести 3 статистичні перевірки.

4. Написати комп'ютерну програму, яка усе це виконує.

122	-5	15	10	60	10	20

Лістинг програми

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

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

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


class Lab3:
    m = 3
    X = []
    Y = []

    def __init__(self, x1min, x1max, x2min, x2max, x3min, x3max):
        print("DOX Lab3")
        self.x0 = [1, 1, 1, 1]
        self.x1 = [-1, -1, 1, 1]
        self.x2 = [-1, 1, -1, 1]
        self.x3 = [-1, 1, 1, -1]

        self.x = [self.x0, self.x1, self.x2, self.x3]
        self.X = [
            [x1min, x1min, x1max, x1max],
            [x2min, x2max, x2min, x2max],
            [x3min, x3max, x3max, x3min],
        ]
        self.y_min = 200 + (x1min + x2min + x3min) / 3
        self.y_max = 200 + (x1max + x2max + x3max) / 3

        y1 = []
        y2 = []
        y3 = []

        for i in range(4):
            y1.append(random.randint(int(self.y_min), int(self.y_max)))
            y2.append(random.randint(int(self.y_min), int(self.y_max)))
            y3.append(random.randint(int(self.y_min), int(self.y_max)))

        self.Y.append(y1)
        self.Y.append(y2)
        self.Y.append(y3)

        # Show X and Y values as a table
        table = []
        table.append([x[0] for x in self.X] + [x[0] for x in self.Y])
        table.append([x[1] for x in self.X] + [x[1] for x in self.Y])
        table.append([x[2] for x in self.X] + [x[2] for x in self.Y])
        table.append([x[3] for x in self.X] + [x[3] for x in self.Y])
        print(
            tabulate(
                table,
                headers=["X1", "X2", "X3", "Y1", "Y2", "Y3"],
                floatfmt=".4f",
                tablefmt="fancy_grid",
```

```
            )
        )

        # Calculate average for every row
        self.y1avg = sum([x[0] for x in self.Y]) / 3
        self.y2avg = sum([x[1] for x in self.Y]) / 3
        self.y3avg = sum([x[2] for x in self.Y]) / 3
        self.y4avg = sum([x[3] for x in self.Y]) / 3
        self.yavg = [self.y1avg, self.y2avg, self.y3avg, self.y4avg]

    def cochran_test(self):
        # Calculate dispersions for every row
        d1 = sum([((x[0] - self.y1avg) ** 2) for x in self.Y]) / 3
        d2 = sum([((x[1] - self.y2avg) ** 2) for x in self.Y]) / 3
        d3 = sum([((x[2] - self.y3avg) ** 2) for x in self.Y]) / 3
        d4 = sum([((x[3] - self.y4avg) ** 2) for x in self.Y]) / 3
        self.d = [d1, d2, d3, d4]
        print("Dispersions:")
        for i in range(4):
            print("d{} = {:.3f}".format(i + 1, self.d[i]))
        gp = max(self.d) / sum(self.d)
        print("\ngp = {:.3f}".format(gp))
        if gp < 0.7679:
            print("   Cochran’s C test passed")
        else:
            print("   Cochran’s C test failed")

    def student_crit_check(self):
        N = len(self.d)
        self.N = N
        sbsq = sum(self.d) / N
        self.sbsq = sbsq
        sbssq = sbsq / (self.m * N)
        sbs = math.sqrt(sbssq)

        print("Sb = {:.3f}".format(sbs))

        b0 = sum([self.yavg[i] * self.x[0][i] for i in range(0, N)]) / N
        b1 = sum([self.yavg[i] * self.x[1][i] for i in range(0, N)]) / N
        b2 = sum([self.yavg[i] * self.x[2][i] for i in range(0, N)]) / N
        b3 = sum([self.yavg[i] * self.x[3][i] for i in range(0, N)]) / N

        b = [b0, b1, b2, b3]
        t = [abs(x) / sbs for x in b]

        print("Beta:")
        for i in range(4):
            print("b{} = {:.3f}".format(i, b[i]))

        print("\nt:")
        for i in range(4):
            print("t{} = {:.3f}".format(i, t[i]))

        print()

        t_tabl = 2.306
        regr_eq = ""
        nm = []
        xs = ["x1", "x2", "x3"]
        d = 0
```

```
    yd = [0, 0, 0, 0]
    for i in range(4):
        if t[i] < t_tabl:
            nm.append(i)
            d += 1
        else:
            for j in range(4):
                if i == 0:
                    yd[j] += self.b[i]
                else:
                    yd[j] += self.b[i] * self.X[2][j]
            if i == 0:
                regr_eq += "{:.3f} + ".format(self.b[0])
            else:
                regr_eq += "{:.3f}*{} + ".format(self.b[i], xs[i - 1])
    if len(regr_eq) != 0:
        regr_eq = regr_eq[0:-2]

    nmt = ",".join(["t" + str(x) for x in nm])
    nmb = ",".join(["b" + str(x + 1) for x in nm])
    print("{} < t_tabl(t_tabl=2.306)".format(nmt))
    print("Factors {} can be excluded".format(nmb))
    print("Regression equation without excluded factors:")
    print("y = {}\n".format(regr_eq))

    for i in range(4):
        print("y{} = {:.3f}".format(i + 1, yd[i]))

    self.d = d
    self.yd = yd

def fisher_crit_check(self):
    print("d = {}".format(self.d))
    sadsq = (self.m / (self.N - self.d)) * sum(
        [(self.yd[i] - self.yavg[i]) ** 2 for i in range(self.N)]
    )
    Fp = sadsq / self.sbsq
    print("S2ad = {:.3f}\nFp = {:.3f}".format(sadsq, Fp))
    table = [5.3, 4.5, 4.1, 3.8, 3.7, 3.6, 3.3, 3.1, 2.9]
    tablec = table[self.N - self.d - 1]
    if Fp < tablec:
        print("   F-test passed/model is adequate")
    else:
        print("   F-test failed/model is NOT adequate")

def normalize_regression_factors(self):
    mx1 = sum(self.X[0]) / 4
    mx2 = sum(self.X[1]) / 4
    mx3 = sum(self.X[2]) / 4
    my = (self.y1avg + self.y2avg + self.y3avg + self.y4avg) / 4
    a1 = (
        self.X[0][0] * (self.y1avg)
        + self.X[0][1] * (self.y2avg)
        + self.X[0][2] * (self.y3avg)
        + self.X[0][3] * (self.y4avg)
    ) / 4
    a2 = (
        self.X[1][0] * (self.y1avg)
        + self.X[1][1] * (self.y2avg)
        + self.X[1][2] * (self.y3avg)
        + self.X[1][3] * (self.y4avg)
```

```
) / 4
a3 = (
    self.X[2][0] * (self.y1avg)
    + self.X[2][1] * (self.y2avg)
    + self.X[2][2] * (self.y3avg)
    + self.X[2][3] * (self.y4avg)
) / 4

a11 = sum([x ** 2 for x in self.X[0]]) / 4
a22 = sum([x ** 2 for x in self.X[1]]) / 4
a33 = sum([x ** 2 for x in self.X[2]]) / 4

a12 = sum([self.X[0][i] * self.X[1][i] for i in range(4)]) / 4
a13 = sum([self.X[0][i] * self.X[2][i] for i in range(4)]) / 4
a23 = sum([self.X[1][i] * self.X[2][i] for i in range(4)]) / 4

a21 = a12
a31 = a13
a32 = a23

# Calculating determinants of matrixes
self.b0 = np.linalg.det(
    [
        [my, mx1, mx2, mx3],
        [a1, a11, a12, a13],
        [a2, a12, a22, a32],
        [a3, a13, a23, a33],
    ]
) / np.linalg.det(
    [
        [1, mx1, mx2, mx3],
        [mx1, a11, a12, a13],
        [mx2, a12, a22, a32],
        [mx3, a13, a23, a33],
    ]
)

self.b1 = np.linalg.det(
    [
        [1, my, mx2, mx3],
        [mx1, a1, a12, a13],
        [mx2, a2, a22, a32],
        [mx3, a3, a23, a33],
    ]
) / np.linalg.det(
    [
        [1, mx1, mx2, mx3],
        [mx1, a11, a12, a13],
        [mx2, a12, a22, a32],
        [mx3, a13, a23, a33],
    ]
)

self.b2 = np.linalg.det(
    [
        [1, mx1, my, mx3],
        [mx1, a11, a1, a13],
        [mx2, a12, a2, a32],
        [mx3, a13, a3, a33],
    ]
) / np.linalg.det(
```

```
            [
                [1, mx1, mx2, mx3],
                [mx1, a11, a12, a13],
                [mx2, a12, a22, a32],
                [mx3, a13, a23, a33],
            ]
        )

        self.b3 = np.linalg.det(
            [
                [1, mx1, mx2, my],
                [mx1, a11, a12, a1],
                [mx2, a12, a22, a2],
                [mx3, a13, a23, a3],
            ]
        ) / np.linalg.det(
            [
                [1, mx1, mx2, mx3],
                [mx1, a11, a12, a13],
                [mx2, a12, a22, a32],
                [mx3, a13, a23, a33],
            ]
        )
        self.b = [self.b0, self.b1, self.b2, self.b3]
        print(
            "Regression equation: y = {:.3f} + {:.3f}x1 + {:.3f}x2 + {:.3f}x3".format(
                self.b0, self.b1, self.b2, self.b3
            )
        )

    def generate_random_y(self):
        for i in range(5):
            self.Y.append(
                [
                    random.randint(self.ymin, self.ymax),
                    random.randint(self.ymin, self.ymax),
                    random.randint(self.ymin, self.ymax),
                ]
            )

    def generate_test_y(self):

        # You can use an example Y values. Just set test_y variable to True when
creating Lab2 object
        self.Y.append([9, 15, 20])
        self.Y.append([10, 14, 18])
        self.Y.append([11, 10, 12])
        self.Y.append([15, 12, 10])
        self.Y.append([9, 14, 16])


# Creating Lab2 object with variant 122 parameters(test_y=False: using random values)
# lab2 = Lab2(x1min=-5, x1max=15, x2min=10, x2max=60, ymin=-20, ymax=80, test_y=False)
# lab2.uniformity_of_dispersion_check()
# lab2.normalize_regression_factors()
# lab2.naturalize_regression_factors()

lab3 = Lab3(-5, 15, 10, 60, 10, 20)
lab3.normalize_regression_factors()
print("\nCochran test:")
lab3.cochran_test()
```

```
print("\nStudent's t-test")
lab3.student_crit_check()
print("\nF-test")
lab3.fisher_crit_check()
```

Результат виконання роботи: нормований план експерименту та функція відгуку для точки плану, що відповідає критерію оптимальності

DOX Lab3

X1	X2	X3	Y1	Y2	Y3
-5	10	10	214	225	222
-5	60	20	205	213	231
15	10	20	209	219	225
15	60	10	218	215	209

Regression equation: y = 220.642 + -0.125x1 + -0.077x2 + -0.017x3

Cochran test:
Dispersions:
d1 = 21.556
d2 = 118.222
d3 = 43.556
d4 = 14.000

gp = 0.599
  Cochran’s C test passed

Student's t-test
Sb = 2.028
Beta:
b0 = 217.083
b1 = -1.250
b2 = -1.917
b3 = -0.083

t:
t0 = 107.065
t1 = 0.616
t2 = 0.945
t3 = 0.041

t1,t2,t3 < t_tabl(t_tabl=2.306)
Factors b2,b3,b4 can be excluded
Regression equation without excluded factors:
y = 220.642

y1 = 220.642
y2 = 220.642
y3 = 220.642
y4 = 220.642

F-test
d = 3
S2ad = 214.858
Fp = 4.355
  F-test passed/model is adequate

Відповіді на контрольні запитання

Що називається дробовим факторним експериментом?

Дробовий факторний експеримент – це частина ПФЕ, який мінімізує число дослідів, за рахунок тієї інформації, яка не дуже істотна для побудови лінійної моделі.

Для чого потрібно розрахункове значення Кохрена?

Для перевірки однорідності дисперсій.

Для чого перевіряється критерій Стьюдента?

Для знаходження тих коефіцієнтів рівняння регресії, що практично не впливають на саме рівняння.

Чим визначається критерій Фішера і як його застосовувати?

Критерій Фішера дорівнює відношенню дисперсії адекватності до дисперсії відтворюваності. Критерій використовується для перевірки отриманого рівняння регресії після відкидання малозначущих коефіцієнтів.

Висновки:

Під час виконання лабораторної роботи було проведено трьохфакторний експеримент, складено матрицю планування, знайдено коефіцієнти рівняння регресії, проведено 3 статистичні перевірки закріплено отримані знання практичним їх використанням при написанні програми, що реалізує завдання на лабораторну роботу.