Міністерство освіти і науки України

Національний технічний університет України «КПІ» імені Ігоря Сікорського

Кафедра обчислювальної техніки ФІОТ

Звіт

з лабораторної роботи №4

з навчальної дисципліни «Методи оптимізації та планування експерименту»

Тема: Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням ефекту взаємодії.

Виконав:

Студент 2 курсу кафедри ОТ ФІОТ

Навчальної групи ІО-91

Тарасенко Андрій

Перевірив:

Регіда П. Г.

Київ 2021

Мета: провести повний трьохфакторний експеримент. Знайти рівняння регресії адекватне об'єкту.

Завдання:

1. Скласти матрицю планування для повного трьохфакторного експерименту.

2. Провести експеримент, повторивши N раз досліди у всіх точках факторного простору і знайти значення відгуку Y. Знайти значення Y шляхом моделювання випадкових чисел у певному діапазоні відповідно варіанту. Варіанти вибираються за номером в списку в журналі викладача.

3. Знайти коефіцієнти рівняння регресії і записати його .

4. Провести 3 статистичні перевірки – за критеріями Кохрена , Стьюдента , Фішера .

5. Зробити висновки по адекватності регресії та значимості окремих коефіцієнтів і записати скореговане рівняння регресії .

6. Написати комп ' ютерну програму , яка усе це моделює .

Варіант:

122	10	40	-30	45	-30	-10

$x_{1min}=10;\ x_{2min}=-30;\ x_{3min}=-30;$

$x_{1max}=40;\ x_{2max}=45;\ x_{3max}=-10;$

$y_{min}=183;\ y_{max}=225$

Лістинг програми

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

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

from tabulate import tabulate
import math
import random


class Lab4:
    N = 8
    m = 3

    X0 = [1, 1, 1, 1, 1, 1, 1, 1]
    X1 = [-1, -1, -1, -1, 1, 1, 1, 1]
    X2 = [-1, -1, 1, 1, -1, -1, 1, 1]
    X3 = [-1, 1, -1, 1, -1, 1, -1, 1]
    X12 = [1, 1, -1, -1, -1, -1, 1, 1]
    X13 = [1, -1, 1, -1, -1, 1, -1, 1]
    X23 = [1, -1, -1, 1, 1, -1, -1, 1]
    X123 = [-1, 1, 1, -1, 1, -1, -1, 1]
    X = [X1, X2, X3, X12, X13, X23, X123]
    Xf = [X0, X1, X2, X3]

    def __init__(self):
        print("DOX Lab4")
        x1min = 10
        x2min = -30
        x3min = -30

        x1max = 40
        x2max = 45
        x3max = -10

        ymin = round((x1min + x2min + x3min) / 3) + 200
        ymax = round((x1max + x2max + x3max) / 3) + 200

        self.x_range = ((x1min, x1max), (x2min, x2max), (x3min, x3max))

        x = []
        x.append([random.randint(x1min, x1max) for i in range(8)])
        x.append([random.randint(x2min, x2max) for i in range(8)])
        x.append([random.randint(x3min, x3max) for i in range(8)])
        x.append([x[0][i] * x[1][i] for i in range(8)])
        x.append([x[0][i] * x[2][i] for i in range(8)])
        x.append([x[1][i] * x[2][i] for i in range(8)])
        x.append([x[0][i] * x[1][i] * x[2][i] for i in range(8)])

        y = []
        y.append([random.randint(ymin, ymax) for i in range(8)])
        y.append([random.randint(ymin, ymax) for i in range(8)])
        y.append([random.randint(ymin, ymax) for i in range(8)])

        self.yavg = [sum(x) / len(x) for x in zip(*y)]

        print("PFE Matrix:")
```

```
        print(
            tabulate(
                zip(*x + y),
                headers=[
                    "X1",
                    "X2",
                    "X3",
                    "X12",
                    "X13",
                    "X23",
                    "X123",
                    "Y1",
                    "Y2",
                    "Y3",
                ],
                floatfmt=".4f",
                tablefmt="fancy_grid",
            )
        )

        self.x = x
        self.y = y

    def normalize_x(self):
        x0 = [(self.x_range[i][0] + self.x_range[i][1]) / 2 for i in range(3)]
        dx = [x0[i] - self.x_range[i][0] for i in range(3)]
        xn = []
        for i in range(3):
            xn.append([(self.x[i][j] - x0[i]) / dx[i] for j in range(8)])
        self.xn = xn
        print("\nNormalized factors")
        print(
            tabulate(
                zip(*xn),
                headers=["X1", "X2", "X3"],
                floatfmt=".4f",
                tablefmt="fancy_grid",
            )
        )

    def calculate_b(self):
        y_sum = sum(self.yavg)
        b = [0] * 8
        b[0] = y_sum
        for i in range(8):
            b[1] += self.yavg[i] * self.xn[0][i]
            b[2] += self.yavg[i] * self.xn[1][i]
            b[3] += self.yavg[i] * self.xn[2][i]
            b[4] += self.yavg[i] * self.xn[0][i] * self.xn[1][i]
            b[5] += self.yavg[i] * self.xn[0][i] * self.xn[2][i]
            b[6] += self.yavg[i] * self.xn[1][i] * self.xn[2][i]
            b[7] += self.yavg[i] * self.xn[0][i] * self.xn[1][i] * self.xn[2][i]

        for i in range(8):
            b[i] = b[i] / self.N

        print("\nRegression equation:")
        print(
            "y = {:.3f} + {:.3f}*x1 + {:.3f}*x2 + {:.3f}*x3 + {:.3f}*x1*x2 +
{:.3f}*x1*x3 + {:.3f}*x2*x3 + {:.3f}*x1*x2*x3\n".format(
                *b
```

```
            )
        )
        self.b = b

    def cochran_test(self):
        # Calculate dispersions for every row
        d = []
        print("Dispersions:")
        for i in range(8):
            d.append(sum([((x[0] - self.yavg[i]) ** 2) for x in self.y]) / 8)
            print("d{} = {:.3f}".format(i + 1, d[-1]))
        gp = max(d) / sum(d)
        self.d = d
        print("\ngp = {:.3f}".format(gp))
        if gp < 0.7679:
            print("  Cochran’s C test passed\n")
        else:
            print("  Cochran’s C test failed\n")

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

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

        b = []
        for i in range(4):
            b.append(sum([self.yavg[j] * self.Xf[i][j] for j in range(N)]) / N)

        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(4)]
        )
        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 calc_y(self, x1, x2, x3):
        return (
            self.b[0]
            + self.b[1] * x1
            + self.b[2] * x2
            + self.b[3] * x3
            + self.b[4] * x1 * x2
            + self.b[5] * x1 * x3
            + self.b[6] * x2 * x3
            + self.b[7] * x1 * x2 * x3
        )


lab4 = Lab4()
lab4.normalize_x()
lab4.calculate_b()
lab4.cochran_test()
lab4.student_crit_check()
lab4.fisher_crit_check()
```

Результат виконання роботи

DOX Lab4
PFE Matrix:

X1	X2	X3	X12	X13	X23	X123	Y1	Y2	Y3
24	-27	-29	-648	-696	783	18792	186	202	221
15	25	-17	375	-255	-425	-6375	183	186	222
13	-29	-26	-377	-338	754	9802	218	193	210
34	12	-15	408	-510	-180	-6120	217	193	197
36	-16	-11	-576	-396	176	6336	212	197	213
33	44	-22	1452	-726	-968	-31944	212	187	183
16	40	-30	640	-480	-1200	-19200	208	223	207
22	9	-14	198	-308	-126	-2772	185	225	186

Normalized factors

X1	X2	X3
-0.0667	-0.9200	-0.9000
-0.6667	0.4667	0.3000
-0.8000	-0.9733	-0.6000
0.6000	0.1200	0.5000
0.7333	-0.6267	0.9000
0.5333	0.9733	-0.2000
-0.6000	0.8667	-1.0000
-0.2000	0.0400	0.6000

Regression equation:
y = 202.750 + -12.611*x1 + -2.609*x2 + -11.537*x3 + 2.521*x1*x2 + 44.093*x1*x3 + -0.694*x2*x3 + -14.410*x1*x2*x3

Dispersions:
d1 = 76.750
d2 = 90.250
d3 = 82.750
d4 = 76.917
d5 = 83.792
d6 = 107.125
d7 = 111.792
d8 = 83.792

gp = 0.157

```
  Cochran’s C test passed

Sb = 1.927
Beta:
b0 = 202.750
b1 = 0.417
b2 = 2.417
b3 = -4.750

t:
t0 = 105.200
t1 = 0.216
t2 = 1.254
t3 = 2.465

t1,t2 < t_tabl(t_tabl=2.306)
Factors b2,b3 can be excluded
Regression equation without excluded factors:
y = 202.750 + -11.537*x3

y1 = 214.288
y2 = 191.213
y3 = 214.288
y4 = 191.213
d = 2
S2ad = 168.842
Fp = 1.894
  F-test passed/model is adequate
```

Висновки:

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