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-93
Дромашко Артем
Номер у списку – 6
Варіант - 306

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.

№ варіанту	x_1		x_2		x_3		$f(x_1, x_2, x_3)$
	min	max	min	max	min	max	
306	10	40	15	50	10	30	1,7+4,9*x1+2,5*x2+3,4*x3+6,3*x1*x1+1,0*x2*x2+1,2*x3*x3+4,8*x1*x2+0,1*x1*x3+2,0*x2*x3+0,5*x1*x2*x3

Програмний код

```
from math import fabs, sqrt
# Варіант 306
m = 5
p = 0.84
N = 15

x1_min = 10
x1_max = 40
x2_min = 15
x2_max = 50
x3_min = 10
x3_max = 30

x01 = (x1_max + x1_min) / 2
x02 = (x2_max + x2_min) / 2
x03 = (x3_max + x3_min) / 2
delta_x1 = x1_max - x01
delta_x2 = x2_max - x02
delta_x3 = x3_max - x03


class Exp:
    def getCohranValue(self, qty_of_selections, significance):
        from _pydecimal import Decimal
        from scipy.stats import f
        self += 1
        partResult1 = significance / (self - 1)
        params = [partResult1, qty_of_selections, (self - 1 - 1) * qty_of_selections]
        fisher = f.isf(*params)
        result = fisher / (fisher + (self - 1 - 1))
        return Decimal(result).quantize(Decimal('.0001')).__float__()

    def get_student_value(self, significance):
        from _pydecimal import Decimal
        from scipy.stats import t
        return Decimal(abs(t.ppf(significance / 2,
self))).quantize(Decimal('.0001')).__float__()

    def get_fisher_value(self, f4, significance):
        from _pydecimal import Decimal
        from scipy.stats import f
        return Decimal(abs(f.isf(significance, f4,
```

```
self))).quantize(Decimal('.0001')).__float__()


def generate_matrix():
    def f(X1, X2, X3):
        from random import randrange
        y = 1.7 + 4.9 * X1 + 2.5 * X2 + 3.4 * X3 + 6.3 * X1 * X1 + 1 * X2 * X2 + 1.2
* X3 * X3 + 4.8 * X1 * X2 + \
            0.1 * X1 * X3 + 2 * X2 * X3 + 0.5 * X1 * X2 * X3 + randrange(0, 10) - 5
        return y

    matrix_with_y = [[f(matrix_x[j][0], matrix_x[j][1], matrix_x[j][2]) for i in
range(m)] for j in range(N)]
    return matrix_with_y


def x(l1, l2, l3):
    x_1 = l1 * delta_x1 + x01
    x_2 = l2 * delta_x2 + x02
    x_3 = l3 * delta_x3 + x03
    return [x_1, x_2, x_3]


def get_average(lst, orientation):
    average = []
    if orientation == 1:
        for rows in range(len(lst)):
            average.append(sum(lst[rows]) / len(lst[rows]))
    else:
        for column in range(len(lst[0])):
            number_lst = []
            for rows in range(len(lst)):
                number_lst.append(lst[rows][column])
            average.append(sum(number_lst) / len(number_lst))
    return average


def a(first, second):
    need_a = 0
    for j in range(N):
        need_a += matrix_x[j][first - 1] * matrix_x[j][second - 1] / N
    return need_a


def find_known(number):
    need_a = 0
    for j in range(N):
        need_a += average_y[j] * matrix_x[j][number - 1] / 15
    return need_a


def solve(lst_1, lst_2):
    from numpy.linalg import solve
    solver = solve(lst_1, lst_2)
    return solver


def check_result(b_lst, k):
    y_i = b_lst[0] + b_lst[1] * matrix[k][0] + b_lst[2] * matrix[k][1] + b_lst[3] *
matrix[k][2] + \
          b_lst[4] * matrix[k][3] + b_lst[5] * matrix[k][4] + b_lst[6] * matrix[k][5]
+ b_lst[7] * matrix[k][6] + \
```

```
          b_lst[8] * matrix[k][7] + b_lst[9] * matrix[k][8] + b_lst[10] *
matrix[k][9]
    return y_i


def student_test(b_lst, number_x=10):
    dispersion_b = sqrt(dispersion_b2)
    for column in range(number_x + 1):
        t_practice = 0
        t_theoretical = Exp.get_student_value(f3, q)
        for row in range(N):
            if column == 0:
                t_practice += average_y[row] / N
            else:
                t_practice += average_y[row] * matrix_pfe[row][column - 1]
        if fabs(t_practice / dispersion_b) < t_theoretical:
            b_lst[column] = 0
    return b_lst


def fisher_test():
    dispersion_ad = 0
    f4 = N - d
    for row in range(len(average_y)):
        dispersion_ad += (m * (average_y[row] - check_result(student_lst, row))) / (N
- d)
    F_practice = dispersion_ad / dispersion_b2
    F_theoretical = Exp.get_fisher_value(f3, f4, q)
    return F_practice < F_theoretical


matrix_pfe = [
    [-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]
]

matrix_x = [[] for x in range(N)]
for i in range(len(matrix_x)):
    if i < 8:
        x_1 = x1_min if matrix_pfe[i][0] == -1 else x1_max
        x_2 = x2_min if matrix_pfe[i][1] == -1 else x2_max
        x_3 = x3_min if matrix_pfe[i][2] == -1 else x3_max
    else:
        x_lst = x(matrix_pfe[i][0], matrix_pfe[i][1], matrix_pfe[i][2])
        x_1, x_2, x_3 = x_lst
    matrix_x[i] = [x_1, x_2, x_3, x_1 * x_2, x_1 * x_3, x_2 * x_3, x_1 * x_2 * x_3,
x_1 ** 2, x_2 ** 2, x_3 ** 2]

adequate = False
```

```
homogeneous = False
while not adequate:
    matrix_y = generate_matrix()
    average_x = get_average(matrix_x, 0)
    average_y = get_average(matrix_y, 1)
    matrix = [(matrix_x[i] + matrix_y[i]) for i in range(N)]
    mx_i = average_x
    my = sum(average_y) / 15

    unknown = [
        [1, mx_i[0], mx_i[1], mx_i[2], mx_i[3], mx_i[4], mx_i[5], mx_i[6], mx_i[7],
mx_i[8], mx_i[9]],
        [mx_i[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_i[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_i[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_i[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_i[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_i[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_i[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_i[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_i[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_i[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)]
    ]
    known = [my, find_known(1), find_known(2), find_known(3), find_known(4),
find_known(5), find_known(6),
             find_known(7),
             find_known(8), find_known(9), find_known(10)]

    beta = solve(unknown, known)
    print("Рівняння регресії:")
    print("{:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 + {:.3f}
* X1X3 + {:.3f} * X2X3"
          "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} * X33^2 =
ŷ\n\tПеревірка"
          .format(beta[0], beta[1], beta[2], beta[3], beta[4], beta[5], beta[6],
beta[7], beta[8], beta[9], beta[10]))
    for i in range(N):
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(beta, i),
average_y[i]))

    while not homogeneous:
        print("Матриця планування експерименту:")
        print("      X1          X2          X3          X1X2        X1X3
X2X3         X1X2X3       X1X1"
              "         X2X2         X3X3          Yi ->")
        for row in range(N):
            print(end=' ')
            for column in range(len(matrix[0])):
                print("{:^12.3f}".format(matrix[row][column]), end=' ')
            print("")

        dispersion_y = [0.0 for x in range(N)]
```

```
        for i in range(N):
            dispersion_i = 0
            for j in range(m):
                dispersion_i += (matrix_y[i][j] - average_y[i]) ** 2
            dispersion_y.append(dispersion_i / (m - 1))
        f1 = m - 1
        f2 = N
        f3 = f1 * f2
        q = 1 - p
        Gp = max(dispersion_y) / sum(dispersion_y)
        print("Тест Кохрена:")
        Gt = Exp.getCohranValue(f2, f1, q)
        if Gt > Gp:
            print("Дисперсія однорідна при {:.2f}.".format(q))
            homogeneous = True
        else:
            print("Дисперсія неоднорідна при {:.2f}! Спробуйте збільшити
m.".format(q))
            m += 1

    dispersion_b2 = sum(dispersion_y) / (N * N * m)
    student_lst = list(student_test(beta))
    print("Рівняння регресії з тестом студента")
    print("{:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 + {:.3f}
* X1X3 + {:.3f} * X2X3"
          "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} * X33^2 =
ŷ\n\tПеревірка"
          .format(student_lst[0], student_lst[1], student_lst[2], student_lst[3],
student_lst[4], student_lst[5],
                  student_lst[6], student_lst[7], student_lst[8], student_lst[9],
student_lst[10]))
    for i in range(N):
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(student_lst, i),
average_y[i]))

    print("Fisher`s test")
    d = 11 - student_lst.count(0)
    if fisher_test():
        print("Рівняння регресії адекватне оригіналу")
        adequate = True
    else:
        print("Рівняння регресії неадекватне оригіналу\n\t Повторюємо експеримент")
```

Результати роботи програми

```
Рівняння регресії:
-1.543 + 4.824 * X1 + 2.571 * X2 + 3.614 * X3 + 4.799 * X1X2 + 0.097 * X1X3 + 1.996 * X2X3+ 0.500 * X1X2X3 + 6.303 * X11^2 + 1.000 * X22^2 + 1.198 * X33^2 = ŷ
    Перевірка
ŷ1 = 2875.611 ≈ 2876.400
ŷ2 = 6025.007 ≈ 6024.800
ŷ3 = 9369.670 ≈ 9369.700
ŷ4 = 17416.665 ≈ 17415.700
ŷ5 = 16913.222 ≈ 16914.400
ŷ6 = 24621.017 ≈ 24621.200
ŷ7 = 33696.681 ≈ 33697.100
ŷ8 = 56802.876 ≈ 56802.300
ŷ9 = 2530.073 ≈ 2530.381
ŷ10 = 44200.083 ≈ 44199.491
ŷ11 = 5581.082 ≈ 5580.063
ŷ12 = 34494.154 ≈ 34494.888
ŷ13 = 10394.746 ≈ 10393.453
ŷ14 = 28564.235 ≈ 28565.243
ŷ15 = 19120.798 ≈ 19120.800
```

Матриця планування експерименту:

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3	Yi ->			
10.000	15.000	10.000	150.000	100.000	150.000	1500.000	100.000	225.000	100.000	2875.200	2873.200	2875.200	2881.200
10.000	15.000	30.000	150.000	300.000	450.000	4500.000	100.000	225.000	900.000	6022.200	6024.200	6027.200	6026.200
10.000	50.000	10.000	500.000	100.000	500.000	5000.000	100.000	2500.000	100.000	9365.700	9372.700	9372.700	9366.700
10.000	50.000	30.000	500.000	300.000	1500.000	15000.000	100.000	2500.000	900.000	17419.700	17414.700	17413.700	17414.700
40.000	15.000	10.000	600.000	400.000	150.000	6000.000	1600.000	225.000	100.000	16917.200	16909.200	16911.200	16916.200
40.000	15.000	30.000	600.000	1200.000	450.000	18000.000	1600.000	225.000	900.000	24618.200	24623.200	24618.200	24622.200
40.000	50.000	10.000	2000.000	400.000	500.000	20000.000	1600.000	2500.000	100.000	33699.700	33695.700	33697.700	33695.700
40.000	50.000	30.000	2000.000	1200.000	1500.000	60000.000	1600.000	2500.000	900.000	56799.700	56801.700	56802.700	56806.700
-0.950	32.500	20.000	-30.875	-19.000	650.000	-617.500	0.902	1056.250	400.000	2532.381	2531.381	2527.381	2529.381
50.950	32.500	20.000	1655.875	1019.000	650.000	33117.500	2595.903	1056.250	400.000	44199.891	44202.891	44199.891	44199.891
25.000	2.225	20.000	55.625	500.000	44.500	1112.500	625.000	4.951	400.000	5580.463	5581.463	5578.463	5578.463
25.000	62.775	20.000	1569.375	500.000	1255.500	31387.500	625.000	3940.701	400.000	34491.088	34496.088	34492.088	34496.088
25.000	32.500	2.700	812.500	67.500	87.750	2193.750	625.000	1056.250	7.290	10394.253	10395.253	10392.253	10394.253
25.000	32.500	37.300	812.500	932.500	1212.250	30306.250	625.000	1056.250	1391.290	28564.443	28562.443	28569.443	28565.443
25.000	32.500	20.000	812.500	500.000	650.000	16250.000	625.000	1056.250	400.000	19120.200	19117.200	19118.200	19126.200

```
Тест Кохрена:
Дисперсія однорідна при 0.16.
Рівняння регресії з тестом студента
-1.543 + 4.824 * X1 + 2.571 * X2 + 3.614 * X3 + 4.799 * X1X2 + 0.097 * X1X3 + 1.996 * X2X3+ 0.500 * X1X2X3 + 6.303 * X11^2 + 1.000 * X22^2 + 1.198 * X33^2 = ŷ
    Перевірка
ŷ1 = 2875.611 ≈ 2876.400
ŷ2 = 6025.007 ≈ 6024.800
ŷ3 = 9369.670 ≈ 9369.700
ŷ4 = 17416.665 ≈ 17415.700
ŷ5 = 16913.222 ≈ 16914.400
ŷ6 = 24621.017 ≈ 24621.200
ŷ7 = 33696.681 ≈ 33697.100
ŷ8 = 56802.876 ≈ 56802.300
ŷ9 = 2530.073 ≈ 2530.381
ŷ10 = 44200.083 ≈ 44199.491
ŷ11 = 5581.082 ≈ 5580.063
ŷ12 = 34494.154 ≈ 34494.888
ŷ13 = 10394.746 ≈ 10393.453
ŷ14 = 28564.235 ≈ 28565.243
ŷ15 = 19120.798 ≈ 19120.800
Тест Фішера
Рівняння регресії адекватне оригіналу
```

Висновок: під час виконання лабораторної роботи було проведено повний трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план. Лабораторна робота виконана, кінцева мета досягнута.