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

ЛАБОРАТОРНА РОБОТА № 6
з дисципліни «Методи наукових досліджень»
на тему «Проведення трьохфакторного експерименту
при використанні рівняння регресії квадратичними членами.»

ВИКОНАВ:
студент 2 курсу
групи ІВ-92
Подкур А. О.
Залікова – 9217

ПЕРЕВІРИВ:
ас. Регіда П. Г.

Київ – 2021

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

Завдання:

Завдання до лабораторної роботи:

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	
217	20	70	5	40	20	45	3,1+6,3*x1+9,8*x2+5,5*x3+2,5*x1*x1+0,4*x2*x2+1,0*x3*x3+3,5*x1*x2+0,7*x1*x3+7,9*x2*x3+8,7*x1*x2*x3

Лістинг програми:

```
from math import fabs, sqrt
m = 3
p = 0.95
N = 15


# Variant 217 - 20 70, 5 40, 20 45

x1_min = 20
x1_max = 70
x2_min = 5
x2_max = 40
x3_min = 20
x3_max = 45


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 Experiment:
```

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

    def get_student_value(f3, significance):
        from _pydecimal import Decimal
        from scipy.stats import t
        return Decimal(abs(t.ppf(significance / 2, f3))).quantize(Decimal('.0001')).__float__()

    def get_fisher_value(f3, f4, significance):
        from _pydecimal import Decimal
        from scipy.stats import f
        return Decimal(abs(f.isf(significance, f4, f3))).quantize(Decimal('.0001')).__float__()


def generate_matrix():
    def f(X1, X2, X3):
        from random import randrange
        y = 3.1 + 6.3 * X1 + 9.8 * X2 + 5.5 * X3 + 2.5 * X1 * X1 + 0.4 * X2 * X2 + 1 * X3 * X3 + 3.5 *
X1 * X2 + \
            0.7 * X1 * X3 + 7.9 * X2 * X3 + 8.7 * 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 = Experiment.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 = Experiment.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("The regression equation:")
    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\tChecking"
          .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("Experiment planning matrix:")
        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("Cochren's test:")
        Gt = Experiment.get_cohran_value(f2, f1, q)
        if Gt > Gp:
            print("The variance is homogeneous at {:.2f}.".format(q))
            homogeneous = True
        else:
            print("The variance is not homogeneous at {:.2f}! Try to increase m.".format(q))
            m += 1

    dispersion_b2 = sum(dispersion_y) / (N * N * m)
    student_lst = list(student_test(beta))
    print("The regression equation with the Student's test")
    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\tChecking"
          .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("The regression equation is adequate to the original")
        adequate = True
    else:
        print("The regression equation is not adequate to the original\n\t We repeat the experiment")
```

Результати роботи програми:

```
The regression equation:
8.442 + 6.179 * X1 + 9.744 * X2 + 5.277 * X3 + 3.507 * X1X2 + 0.706 * X1X3 + 7.904 * X2X3+ 8.700 * X1X2X3 + 2.499 * X11^2 + 0.397 * X22^2 + 1.000 * X33^2 = ŷ
        Checking
ŷ1 = 20519.052 ≈ 20518.100
ŷ2 = 45366.566 ≈ 45365.433
ŷ3 = 151271.075 ≈ 151272.100
ŷ4 = 335281.756 ≈ 335282.600
ŷ5 = 77155.250 ≈ 77154.100
ŷ6 = 157259.431 ≈ 157258.100
ŷ7 = 518538.607 ≈ 518539.433
ŷ8 = 1138422.954 ≈ 1138423.600
ŷ9 = 18766.254 ≈ 18766.229
ŷ10 = 597856.850 ≈ 597857.283
ŷ11 = -94593.938 ≈ -94591.448
ŷ12 = 702596.157 ≈ 702594.075
ŷ13 = 107566.683 ≈ 107566.678
ŷ14 = 500642.890 ≈ 500643.303
ŷ15 = 303637.186 ≈ 303637.183
Experiment planning matrix:
      X1          X2          X3          X1X2        X1X3        X2X3        X1X2X3       X1X1        X2X2        X3X3          Yi ->
    20.000       5.000       20.000      100.000     400.000     100.000      2000.000     400.000      25.000     400.000     20519.100    20517.100    20518.100
    20.000       5.000       45.000      100.000     900.000     225.000      4500.000     400.000      25.000    2025.000     45367.100    45363.100    45366.100
    20.000      40.000       20.000      800.000     400.000     800.000     16000.000     400.000    1600.000     400.000     151274.100   151271.100   151271.100
    20.000      40.000       45.000      800.000     900.000    1800.000     36000.000     400.000    1600.000    2025.000     335284.600   335279.600   335283.600
    70.000       5.000       20.000      350.000    1400.000     100.000      7000.000    4900.000      25.000     400.000     77153.100    77156.100    77153.100
    70.000       5.000       45.000      350.000    3150.000     225.000     15750.000    4900.000      25.000    2025.000     157261.100   157254.100   157259.100
    70.000      40.000       20.000     2800.000    1400.000     800.000     56000.000    4900.000    1600.000     400.000     518538.100   518540.100   518540.100
    70.000      40.000       45.000     2800.000    3150.000    1800.000    126000.000    4900.000    1600.000    2025.000    1138425.600  1138423.600  1138421.600
     1.750      22.500       32.500       39.375      56.875     731.250      1279.688       3.062     506.250    1056.250     18764.562    18769.562    18764.562
    88.250      22.500       32.500     1985.625    2868.125     731.250     64532.812    7788.062     506.250    1056.250     597856.950   597855.950   597858.950
    45.000      -7.775       32.500     -349.875    1462.500    -252.687    -11370.937    2025.000      60.451    1056.250     -94589.115   -94591.115   -94594.115
    45.000      52.775       32.500     2374.875    1462.500    1715.188     77183.438    2025.000    2785.201    1056.250     702594.075   702594.075   702594.075
    45.000      22.500       10.875     1012.500     489.375     244.688     11010.938    2025.000     506.250     118.266     107566.678   107568.678   107564.678
    45.000      22.500       54.125     1012.500    2435.625    1217.812     54801.562    2025.000     506.250    2929.516     500646.303   500643.303   500640.303
    45.000      22.500       32.500     1012.500    1462.500     731.250     32906.250    2025.000     506.250    1056.250     303638.850   303639.850   303632.850
Cochren's test:
The variance is homogeneous at 0.05.
The regression equation with the Student's test
8.442 + 6.179 * X1 + 9.744 * X2 + 5.277 * X3 + 3.507 * X1X2 + 0.706 * X1X3 + 7.904 * X2X3+ 8.700 * X1X2X3 + 2.499 * X11^2 + 0.397 * X22^2 + 1.000 * X33^2 = ŷ
        Checking
ŷ1 = 20519.052 ≈ 20518.100
ŷ2 = 45366.566 ≈ 45365.433
ŷ3 = 151271.075 ≈ 151272.100
ŷ4 = 335281.756 ≈ 335282.600
ŷ5 = 77155.250 ≈ 77154.100
ŷ6 = 157259.431 ≈ 157258.100
ŷ7 = 518538.607 ≈ 518539.433
ŷ8 = 1138422.954 ≈ 1138423.600
ŷ9 = 18766.254 ≈ 18766.229
ŷ10 = 597856.850 ≈ 597857.283
ŷ11 = -94593.938 ≈ -94591.448
ŷ12 = 702596.157 ≈ 702594.075
ŷ13 = 107566.683 ≈ 107566.678
ŷ14 = 500642.890 ≈ 500643.303
ŷ15 = 303637.186 ≈ 303637.183
Fisher`s test
The regression equation is adequate to the original
```

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.