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» на тему

«Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

ВИКОНАВ:
студент ІІ курсу ФІОТ
групи ІВ-92
Злочевський Нікіта Вікторович
Варіант: 209

ПЕРЕВІРИВ:
Регіда П. Г.

Київ – 2021

Хід роботи

Мета: Провести 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.

209	-20	15	-30	45	-30	-15	5,4+3,4*x1+9,6*x2+6,8*x3+3,1*x1*x1+0,1*x2*x2+1,2*x3*x3+0,8*x1*x2+0,8*x1*x3+9,9*x2*x3+4,5*x1*x2*x3

Лістинг програми

```
from math import fabs, sqrt

m = 3
p = 0.95
N = 15

x1_min = -20
x1_max = 15
x2_min = -30
x2_max = 45
x3_min = -30
x3_max = -15
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 Tests:
    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 = 5.4 + 3.4 * X1 + 9.6 * X2 + 6.8 * X3 + 3.1 * X1 * X1 + 0.1 * X2 * X2 + 1.2
* X3 * X3 + 0.8 * X1 * X2 + \
            0.8 * X1 * X3 + 9.9 * X2 * X3 + 4.5 * 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 find_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_p = 0
        t_t = Tests.get_student_value(f3, q)
        for row in range(N):
            if column == 0:
                t_p += average_y[row] / N
            else:
                t_p += average_y[row] * matrix_pfe[row][column - 1]
        if fabs(t_p / dispersion_b) < t_t:
            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_p = dispersion_ad / dispersion_b2
    F_t = Tests.get_fisher_value(f3, f4, q)
    return F_p < F_t


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 = find_average(matrix_x, 0)
    average_y = find_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("Отримане 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 експеременту:")
        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("Критерій Кохрена:")
        Gt = Tests.get_cohran_value(f2, f1, q)
        if Gt > Gp:
            print("Дисперсія однорідна при рівні значимості {:.2f}.".format(q))
            homogeneous = True
        else:
            print("Дисперсія не однорідна при рівні значимості {:.2f}! Збільшуємо
m.".format(q))
            m += 1

    dispersion_b2 = sum(dispersion_y) / (N * N * m)
    student_lst = list(student_test(beta))
    print("Отримане рівняння регресії з урахуванням критерія Стьюдента")
    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("Критерій Фішера")
    d = 11 - student_lst.count(0)
    if fisher_test():
        print("Рівняння регресії адекватне  оригіналу")
        adequate = True
    else:
        print("Рівняння регресії неадекватне  оригіналу\n\t Проводимо експеримент
повторно")
```

Результат роботи програми

```
Отримане рівняння регресії
7.400 + 3.358 * X1 + 9.595 * X2 + 7.197 * X3 + 0.802 * X1X2 + 0.797 * X1X3 + 9.900 * X2X3+ 4.500 * X1X2X3 + 3.102 * X11^2 + 0.100 * X22^2 + 1.210 * X33^2 = ŷ
    Перевірка
ŷ1 = -69274.763 ≈ -69275.600
ŷ2 = -34177.310 ≈ -34176.600
ŷ3 = 110581.998 ≈ 110580.900
ŷ4 = 55565.951 ≈ 55566.400
ŷ5 = 70372.006 ≈ 70370.567
ŷ6 = 35012.459 ≈ 35012.567
ŷ7 = -102044.900 ≈ -102046.600
ŷ8 = -50328.780 ≈ -50328.933
ŷ9 = 27367.003 ≈ 27366.969
ŷ10 = -20071.799 ≈ -20070.442
ŷ11 = -1334.545 ≈ -1334.185
ŷ12 = 3787.067 ≈ 3788.030
ŷ13 = 1778.192 ≈ 1780.642
ŷ14 = 237.618 ≈ 236.491
ŷ15 = 804.222 ≈ 804.213
```

```
Матриця планування експеременту:
      X1          X2          X3          X1X2        X1X3        X2X3        X1X2X3      X1X1        X2X2        X3X3        Yi ->
   -20.000     -30.000     -30.000     600.000     600.000     900.000    -18000.000   400.000     900.000     900.000    -69274.600  -69279.600  -69272.600
   -20.000     -30.000     -15.000     600.000     300.000     450.000     -9000.000   400.000     900.000     225.000    -34174.600  -34176.600  -34178.600
   -20.000      45.000     -30.000    -900.000     600.000   -1350.000     27000.000   400.000    2025.000     900.000    110580.900  110583.900  110577.900
   -20.000      45.000     -15.000    -900.000     300.000    -675.000     13500.000   400.000    2025.000     225.000    55569.400   55562.400   55567.400
    15.000     -30.000     -30.000    -450.000    -450.000     900.000     13500.000   225.000     900.000     900.000    70374.900   70369.900   70366.900
    15.000     -30.000     -15.000    -450.000    -225.000     450.000      6750.000   225.000     900.000     225.000    35011.900   35011.900   35013.900
    15.000      45.000     -30.000     675.000    -450.000   -1350.000    -20250.000   225.000    2025.000     900.000   -102044.600 -102049.600 -102045.600
    15.000      45.000     -15.000     675.000    -225.000    -675.000    -10125.000   225.000    2025.000     225.000    -50333.600  -50326.600  -50326.600
   -32.775       7.500     -22.500    -245.812     737.438    -168.750     5530.781   1074.201     56.250     506.250    27371.303   27362.303   27367.303
    27.775       7.500     -22.500     208.312    -624.938    -168.750    -4687.031   771.451      56.250     506.250    -20068.109  -20071.109  -20072.109
    -2.500      -57.375    -22.500     143.438     56.250     1290.938    -3227.344    6.250     3291.891     506.250    -1338.852   -1331.852   -1331.852
    -2.500       72.375    -22.500    -180.938     56.250    -1628.438     4071.094    6.250     5238.141     506.250     3789.030    3787.030    3788.030
    -2.500       7.500     -35.475     -18.750     88.688     -266.062     665.156     6.250      56.250     1258.476    1781.975    1780.975    1778.975
    -2.500       7.500      -9.525     -18.750     23.812     -71.438      178.594     6.250      56.250      90.726     233.491     242.491     233.491
    -2.500       7.500     -22.500     -18.750     56.250     -168.750     421.875     6.250      56.250     506.250     801.213     805.213     806.213
Критерій Кохрена:
Дисперсія однорідна при рівні значимості 0.05.
Отримане рівняння регресії з урахуванням критерія Стьюдента
7.400 + 3.358 * X1 + 9.595 * X2 + 7.197 * X3 + 0.802 * X1X2 + 0.797 * X1X3 + 9.900 * X2X3+ 4.500 * X1X2X3 + 3.102 * X11^2 + 0.100 * X22^2 + 1.210 * X33^2 = ŷ
    Перевірка
ŷ1 = -69274.763 ≈ -69275.600
ŷ2 = -34177.310 ≈ -34176.600
ŷ3 = 110581.998 ≈ 110580.900
ŷ4 = 55565.951 ≈ 55566.400
ŷ5 = 70372.006 ≈ 70370.567
ŷ6 = 35012.459 ≈ 35012.567
ŷ7 = -102044.900 ≈ -102046.600
ŷ8 = -50328.780 ≈ -50328.933
ŷ9 = 27367.003 ≈ 27366.969
ŷ10 = -20071.799 ≈ -20070.442
ŷ11 = -1334.545 ≈ -1334.185
ŷ12 = 3787.067 ≈ 3788.030
ŷ13 = 1778.192 ≈ 1780.642
ŷ14 = 237.618 ≈ 236.491
ŷ15 = 804.222 ≈ 804.213
Критерій Фішера
Рівняння регресії адекватне  оригіналу
```

Висновок:

На цій лабораторній роботі ми провели трьохфакторний експеримент і отримали адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план. Закріпили отримані знання їх практичним використанням при написанні програми у середовищі Pycharm на мові python, що реалізує завдання лабораторної роботи.