Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«Методи оптимізації та планування експерименту»
Лабораторна робота №6

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

Виконав:
студент групи ІО-91
Герейханов Тимур
Варіант: 105
Перевірив Регіда П. Г.

Київ 2021 р.

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

Завдання на лабораторну роботу

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	
105	-30	20	15	50	20	35	8,6+8,5*x1+7,9*x2+7,3*x3+0,2*x1*x1+0,4*x2*x2+6,1*x3*x3+7,2*x1*x2+0,8*x1*x3+5,6*x2*x3+9,3*x1*x2*x3

Роздруківка тексту програми:

```
from math import fabs, sqrt
m = 3
p = 0.95
N = 15
x1_min = -30
x1_max = 20
x2_min = 15
x2_max = 50
x3_min = 20
x3_max = 35
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 Perevirku:
    def get_cohren_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 = 8.6 + 8.5 * X1 + 7.9 * X2 + 7.3 * X3 + 0.2 * X1 * X1 + 0.4 * X2 * X2 + 6.1 * X3 * X3 + 7.2 *
X1 * X2 + \
            0.8 * X1 * X3 + 5.6 * X2 * X3 + 9.3 * 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_practice = 0
        t_theoretical = Perevirku.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 = Perevirku.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]


adekvat = False
odnorid = False
while not adekvat:
    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 odnorid:
        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 = Perevirku.get_cohren_value(f2, f1, q)
    if Gt > Gp:
        print("Дисперсія однорідна при рівні значимості {:.2f}.".format(q))
        odnorid = 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")
    adekvat = True
else:
    print("Рівняння регресії неадекватне  оригіналу\n\t Проводимо експеремент повторно")
```

Результат виконання програми:

```
Отримане рівняння регресії
4.438 + 0.382 * X1 + 0.056 * X2 + 7.444 * X3 + 7.205 * X1X2 + 0.805 * X1X3 + 5.595 * X2X3+ 0.300 * X1X2X3 + 0.200 * X11^2 + 0.400 * X22^2 + 0.899 * X33^2 - y
    Перевірка
y1 = -83012.819 = -83011.567
y2 = -139746.247 = -139745.233
y3 = -200766.375 = -200766.400
y4 = -481033.003 = -481034.067
y5 = 63012.585 = 63012.433
y6 = 111505.824 = 111505.433
y7 = 203361.029 = 203359.600
y8 = 352439.601 = 352437.933
y9 = -402838.744 = -402840.898
y10 = 338846.113 = 338848.815
y11 = 2111.063 = 2109.864
y12 = -66119.658 = -66117.904
y13 = -18492.125 = -18492.122
y14 = -44195.187 = -44194.633
y15 = -32370.479 = -32370.483
Матриця планування експерименту:
      X0          X1          X2          X3        X1X2        X1X3        X2X3      X1X2X3        X11         X22         X33          Y1 -y
  -30.000      15.000      20.000    -450.000    -600.000     300.000   -9000.000     900.000     225.000     400.000   -83009.900   -83009.900   -83014.900
  -30.000      15.000      35.000    -450.000   -1050.000     525.000  -15750.000     900.000     225.000    1225.000  -139746.900  -139745.900  -139746.900
  -30.000      50.000      20.000   -1500.000    -600.000    1000.000  -30000.000     900.000    2500.000     400.000  -200770.400  -200765.400  -200767.400
  -30.000      50.000      35.000   -1500.000   -1050.000    1750.000  -52500.000     900.000    2500.000    1225.000  -481033.400  -481031.400  -481037.400
   20.000      15.000      20.000     300.000     400.000     300.000    6000.000     400.000     225.000     400.000    63010.100    63017.100    63010.100
   20.000      15.000      35.000     300.000     700.000     525.000   10500.000     400.000     225.000    1225.000   111509.100   111503.100   111504.100
   20.000      50.000      20.000    1000.000     400.000    1000.000   20000.000     400.000    2500.000     400.000   203362.600   203355.600   203360.600
   20.000      50.000      35.000    1000.000     700.000    1750.000   35000.000     400.000    2500.000    1225.000   352439.600   352436.600   352437.600
  -48.250      32.500      27.500   -1568.125   -1326.875     893.750  -43121.438    2328.862    1056.250     756.250  -402838.756  -402838.756  -402842.756
   38.250      32.500      27.500    1243.125    1051.875     893.750   34185.938    1463.862    1056.250     756.250   338846.681   338847.681   338849.681
   -5.000       2.225      27.500     -11.125     -137.500      61.188    -305.938      25.000       4.951     756.250    2109.864    2110.864    2108.864
   -5.000      62.775      27.500    -313.875     -137.500    1726.312   -8631.562      25.000    1040.701     756.250   -66121.904   -66115.904   -66115.904
   -5.000      32.500      14.525    -162.500      -72.625     472.062   -2360.312      25.000    1056.250     210.976   -18489.122   -18493.122   -18496.122
   -5.000      32.500      40.475    -162.500     -202.375    1315.438   -6577.188      25.000    1056.250    1638.226   -44198.300   -44191.300   -44194.300
   -5.000      32.500      27.500    -162.500     -137.500     893.750   -4468.750      25.000    1056.250     756.250   -32370.150   -32370.150   -32371.150
 Критерій Кохрена:
 Дисперсія однорідна при рівні значимості 0.05.
 Отримане рівняння регресії з урахуванням критерія Стьюдента
 4.438 + 0.382 * X1 + 0.056 * X2 + 7.444 * X3 + 7.205 * X1X2 + 0.805 * X1X3 + 5.595 * X2X3+ 0.300 * X1X2X3 + 0.200 * X11^2 + 0.400 * X22^2 + 0.899 * X33^2 - y
     Перевірка
 y1 = -83012.819 = -83011.567
 y2 = -139746.247 = -139745.233
 y3 = -200766.375 = -200766.400
 y4 = -481033.003 = -481034.067
 y5 = 63012.585 = 63012.433
 y6 = 111505.824 = 111505.433
 y7 = 203361.029 = 203359.600
 y8 = 352439.601 = 352437.933
 y9 = -402838.744 = -402840.898
 y10 = 338846.113 = 338848.815
 y11 = 2111.063 = 2109.864
 y12 = -66119.658 = -66117.904
 y13 = -18492.125 = -18492.122
 y14 = -44195.187 = -44194.633
 y15 = -32370.479 = -32370.483
 Критерій Фішера
 Рівняння регресії адекватне  оригіналу
```