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 №5:

«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів
(центральний ортогональний композиційний план)»

Виконав:
студент групи ІВ-81
Юхимчук Я. М.
Перевірив Регіда П. Г.

Київ 2020р.

Лабораторна робота №5

Тема: Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план).

Мета: Провести трьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральнийортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Виконання:

Варіант – 128.

128	-2	5	-6	4	-9	8

1. Лістинг програми:

```
from _pydecimal import Decimal, ROUND_UP, ROUND_FLOOR
from math import fabs
from math import *

from numpy.linalg import solve
import numpy as np
from numpy import *
from scipy.stats import f, t, ttest_ind, norm


m = int(input("Введіть m або просто натисніть <Enter>, тоді m = 3: ") or 3)
p = float(input("Введіть довірчу ймовірність або просто натисніть <Enter>, тоді p =
0.95: ") or 0.95)
N = 15

# Задані за варіантом значення x
x1_min, x1_max = -2, 5
x2_min, x2_max = -6, 4
x3_min, x3_max = -9, 8
x_avarage_max = (x1_max + x2_max + x3_max) / 3
x_avarage_min = (x1_min + x2_min + x3_min) / 3
y_max = 200 + x_avarage_max
y_min = 200 + x_avarage_min

# Матриця кодованих значень x
matrix_x_code = [[-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],
                 [-1.125, 0, 0, 0, 0, 0, 0, +1.476, 0, 0],
                 [+1.125, 0, 0, 0, 0, 0, 0, +1.476, 0, 0],
                 [0, -1.125, 0, 0, 0, 0, 0, 0, +1.476, 0],
                 [0, +1.125, 0, 0, 0, 0, 0, 0, +1.476, 0],
                 [0, 0, -1.125, 0, 0, 0, 0, 0, 0, +1.476],
                 [0, 0, +1.125, 0, 0, 0, 0, 0, 0, +1.476],
                 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

# Розрахунки по формулі для зоряної точки
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


def x_formula(l1, l2, l3):
    a = l1 * delta_x1 + x01
    b = l2 * delta_x2 + x02
    c = l3 * delta_x3 + x03
    return [a, b, c]


# Пошук критеріїв
class Criteries:
    @staticmethod
    def get_cohren_value(size_of_selections, qty_of_selections, significance):
        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__()

    @staticmethod
    def get_student_value(f3, significance):
        return Decimal(abs(t.ppf(significance / 2,
f3))).quantize(Decimal('.0001')).__float__()

    @staticmethod
    def get_fisher_value(f3, f4, significance):
        return Decimal(abs(f.isf(significance, f4,
f3))).quantize(Decimal('.0001')).__float__()


cr = Criteries()

# Заповнюємо матрицю x
matrix_x = [[] for x_formula in range(N)]
for i in range(len(matrix_x_code)):
    if i < 8:
        x1 = x1_min if matrix_x_code[i][0] == -1 else x1_max
        x2 = x2_min if matrix_x_code[i][1] == -1 else x2_max
        x3 = x3_min if matrix_x_code[i][2] == -1 else x3_max
    else:
        x_lst = x_formula(matrix_x_code[i][0], matrix_x_code[i][1],
matrix_x_code[i][2])
        x1, x2, x3 = x_lst

    matrix_x[i] = [x1, x2, x3, x1 * x2, x1 * x3, x2 * x3, x1 * x2 * x3, x1 ** 2, x2
** 2, x3 ** 2]

print("~" * 51 + "Матриця планування експеременту" + "~" * 51)
print("|      X1           X2           X3          X1X2        X1X3         X2X3
X1X2X3       X1X1"
      "         X2X2         X3X3")
# Виводимо нашу матрицю
for i in range(N):
    print("|", end=' ')
    for j in range(len(matrix_x[0])):
        print("{:^12.3f}".format(matrix_x[i][j]), end=' ')
    print("|")
print("~" * 133)

check = True
while check:
    # Створюємо матрицю для y
    random_matrix_y = random.randint(y_min, y_max, size=(15, m))
```

```
    print("Матриця для y:\n", random_matrix_y)


    # Шукаємо середні значення y
    def sum_rows(random_matrix_y):
        y = np.sum(random_matrix_y, axis=1) / m
        return y

    Yavg = sum_rows(random_matrix_y)
    print("Середні значення y: {:.2f}\t {:.2f}\t {:.2f}\t {:.2f}\t {:.2f}\t "
          "{:.2f}\t {:.2f}\t {:.2f}\t {:.2f}\t {:.2f}\t "
          "{:.2f}\t {:.2f}\t {:.2f}\t {:.2f}\t {:.2f}".format(Yavg[0], Yavg[1],
Yavg[2], Yavg[3],
                                                             Yavg[4], Yavg[5],
Yavg[6], Yavg[7],
                                                             Yavg[8], Yavg[9],
Yavg[10], Yavg[11],
                                                             Yavg[12], Yavg[13],
Yavg[14]))


    # Шукаємо середні значення x
    def sum_columns(matrix_x):
        mx = np.sum(matrix_x, axis=0) / 15
        return mx

    mx_i = sum_columns(matrix_x)
    c = []
    for i in mx_i:
        c.append(round(i, 2))
    print("Середні x: ", c)


    # Шукаємо середнє значення my наших середніх Yavg
    def sum_my(y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15):
        my = (y1 + y2 + y3 + y4 + y5 + y6 + y7 + y8 + y9 + y10 + y11 + y12 + y13 +
y14 + y15) / 15
        return my

    my = sum_my(Yavg[0], Yavg[1], Yavg[2], Yavg[3], Yavg[4],
                Yavg[5], Yavg[6], Yavg[7], Yavg[8], Yavg[9],
                Yavg[10], Yavg[11], Yavg[12], Yavg[13], Yavg[14])


    # Пошук коефіцієнтів a
    def a(x, y):
        a = 0
        for j in range(N):
            a += matrix_x[j][x - 1] * matrix_x[j][y - 1] / N
        return a


    # Пошук коефіцієнтів a1, a2, a3...an
    def find(n):
        a = 0
        for j in range(N):
            a += Yavg[j] * matrix_x[j][n - 1] / N
        return a

    # Рахуємо коефіцієнти для b
    b = [
        [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)]
    ]

    zadany = [my, find(1), find(2), find(3), find(4), find(5), find(6), find(7),
find(8), find(9), find(10)]
    beta = solve(b, zadany)


    # Перевірка
    def check(a, b, c):
        y_norm = beta[0] + beta[1] * a + beta[2] * b + beta[3] * c + beta[4] * a * b
+ \
                 beta[5] * a * c + beta[6] * b * c + beta[7] * a * b * c + \
                 beta[8] * a * a + beta[9] * b * b + beta[10] * c * c
        return y_norm


    y_norm1 = check(x1_min, x2_min, x3_min)
    y_norm2 = check(x1_min, x2_min, x3_max)
    y_norm3 = check(x1_min, x2_max, x3_min)
    y_norm4 = check(x1_min, x2_max, x3_max)
    y_norm5 = check(x1_max, x2_min, x3_min)
    y_norm6 = check(x1_max, x2_min, x3_max)
    y_norm7 = check(x1_max, x2_max, x3_min)
    y_norm8 = check(x1_max, x2_max, x3_max)
    y_norm9 = check(x1_min * (-1.125), 0, 0)
    y_norm10 = check(x1_max * (1.125), 0, 0)
    y_norm11 = check(0, x2_min * (-1.125), 0)
    y_norm12 = check(0, x2_max * (1.125), 0)
    y_norm13 = check(0, 0, x3_min * (-1.125))
    y_norm14 = check(0, 0, x3_max * (1.125))
    y_norm15 = check(0, 0, 0)
    X = []
    X.append(y_norm1)
    X.append(y_norm2)
    X.append(y_norm3)
    X.append(y_norm4)
    X.append(y_norm5)
    X.append(y_norm6)
    X.append(y_norm7)
    X.append(y_norm8)
    X.append(y_norm9)
    X.append(y_norm10)
    X.append(y_norm11)
    X.append(y_norm12)
    X.append(y_norm13)
    X.append(y_norm14)
    X.append(y_norm15)
    print("Отримане рівняння регресії")
    print("{:.2f} + {:.2f}*x1 + {:.2f}*x2 + {:.2f}*x3 + {:.2f}*x1x2 + {:.2f}*x1x3 +
{:.2f}*x2x3 + "
          "{:.2f}*x1x2x3 + {:.2f}*x1^2 + {:.2f}*x2^2 + {:.2f}*x3^2 = ŷ\nПеревірка ŷ"
```

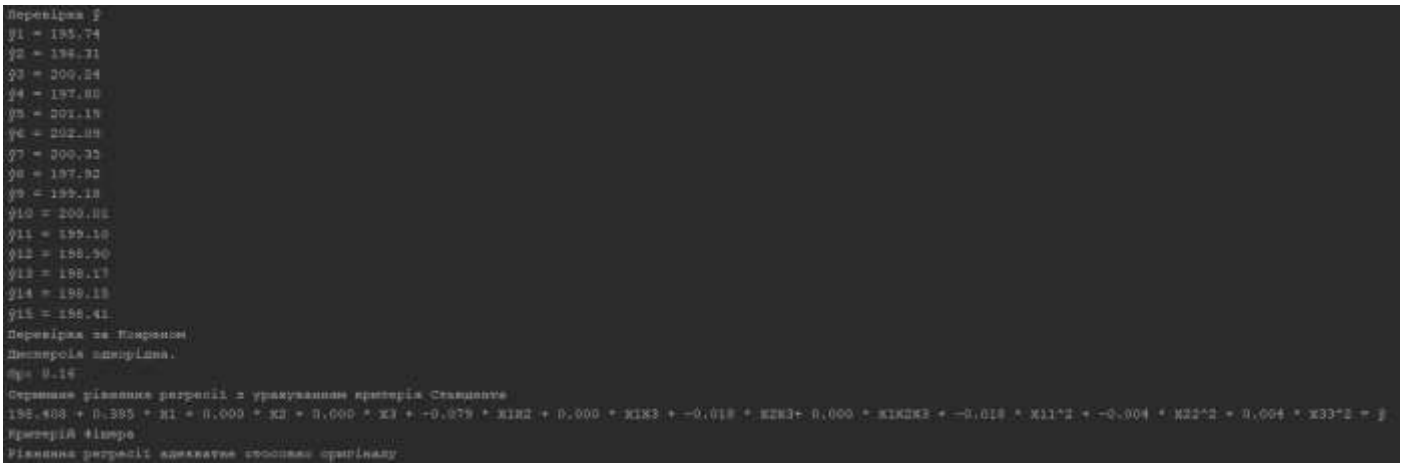
```
          .format(beta[0], beta[1], beta[2], beta[3], beta[4], beta[5], beta[6],
beta[7], beta[8], beta[9], beta[10]))

    print("ŷ1 = {:.2f}\nŷ2 = {:.2f}\nŷ3 = {:.2f}\nŷ4 = {:.2f}\nŷ5 = {:.2f}\nŷ6 =
{:.2f}\nŷ7 = {:.2f}\nŷ8 = {:.2f}\n"
          "ŷ9 = {:.2f}\nŷ10 = {:.2f}\nŷ11 = {:.2f}\nŷ12 = {:.2f}\nŷ13 = {:.2f}\nŷ14
= {:.2f}\nŷ15 = {:.2f}"
          .format(y_norm1, y_norm2, y_norm3, y_norm4, y_norm5, y_norm6, y_norm7,
                  y_norm8, y_norm9, y_norm10, y_norm11, y_norm12, y_norm13,
y_norm14, y_norm15))

    # Критерій Кохрена
    print("Перевірка за Кохреном")
    dispersion_y = [0.0 for x in range(N)]
    for i in range(N):
        dispersion_i = 0
        for j in range(m):
            dispersion_i += ((random_matrix_y[i][j] - Yavg[i]) ** 2) / m
        dispersion_y.append(dispersion_i)
    Gp = max(dispersion_y) / sum(dispersion_y)
    f1 = m - 1
    f2 = N
    q = 1 - p
    Gt = cr.get_cohren_value(f2, f1, q)
    if Gp <= Gt:
        print("Дисперсія однорідна.")
        check = False
    else:
        m += 1
        print("Отримали неоднорідну дисперсію, збільшуємо m.")
    print("Gp: {:.2f}".format(Gp))

    # Критерій Стьюдента
    f1 = m - 1
    f2 = N
    f3 = f1 * f2
    Ft = cr.get_student_value(f3, q)
    S_B = sum(dispersion_y) / len(dispersion_y)
    S2_beta = S_B / (m * N)
    S_b = S2_beta ** (1 / 2)


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


    dispersion_b2 = sum(dispersion_y) / (N)
    student_lst = list(student(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"
          .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]))


    # Критерій Фішера
    def fisher_test():
        dispersion_ad = 0
        f4 = N - d
        for row in range(len(Yavg)):
            dispersion_ad += (m * (X[i] - Yavg[row])) / (N - d)
        F_practice = dispersion_ad / dispersion_b2
        F_theoretical = cr.get_fisher_value(f3, f4, q)
        return F_practice < F_theoretical


    print("Критерій Фішера")
    d = 11 - student_lst.count(0)
    if fisher_test():
        print("Рівняння регресії адекватне стосовно оригіналу")
    else:
        print("Рівняння регресії неадекватне стосовно оригіналу")
```

2. Результат виконання роботи програми:

```
Введіть m або просто натисніть <Enter>, тоді m = 3:
Введіть довірчу ймовірність або просто натисніть <Enter>, тоді p = 0.95:
                                                    Матриця планування експерименту
|      X1          X2          X3          X1X2        X1X3        X2X3        X1X2X3      X1X1        X2X2        X3X3        |
|    -2.000       -6.000      -9.000      12.000      18.000      54.000      -108.000    4.000       36.000      81.000      |
|    -2.000       -6.000      8.000       12.000      -16.000     -48.000     96.000      4.000       36.000      64.000      |
|    -2.000       4.000       -9.000      -8.000      18.000      -36.000     72.000      4.000       16.000      81.000      |
|    -2.000       4.000       8.000       -8.000      -16.000     32.000      -64.000     4.000       16.000      64.000      |
|    5.000        -6.000      -9.000      -30.000     -45.000     54.000      270.000     25.000      36.000      81.000      |
|    5.000        -6.000      8.000       -30.000     40.000      -48.000     -240.000    25.000      36.000      64.000      |
|    5.000        4.000       -9.000      20.000      -45.000     -36.000     -180.000    25.000      16.000      81.000      |
|    5.000        4.000       8.000       20.000      40.000      32.000      160.000     25.000      16.000      64.000      |
|    -2.438       -1.000      -0.500      2.438       1.219       0.500       -1.219      5.941       1.000       0.250       |
|    5.438        -1.000      -0.500      -5.438      -2.719      0.500       2.719       29.566      1.000       0.250       |
|    1.500        -6.625      -0.500      -9.938      -0.750      3.312       4.969       2.250       43.891      0.250       |
|    1.500        4.625       -0.500      6.938       -0.750      -2.312      -3.469      2.250       21.391      0.250       |
|    1.500        -1.000      -10.062     -1.500      -15.094     10.062      15.094      2.250       1.000       101.254     |
|    1.500        -1.000      9.062       -1.500      13.594      -9.062      -13.594     2.250       1.000       82.129      |
|    1.500        -1.000      -0.500      -1.500      -0.750      0.500       0.750       2.250       1.000       0.250       |
```

```
Матриця для y:
 [[197 194 197]
 [194 200 196]
 [195 204 203]
 [194 201 200]
 [196 204 202]
 [204 204 197]
 [204 197 199]
 [196 197 200]
 [194 198 195]
 [201 204 200]
 [197 204 196]
 [198 198 200]
 [203 201 196]
 [198 199 199]
 [201 200 196]]
Середні значення y: 196.00  196.67  200.67  198.33  200.67  201.67  200.00  197.67  195.67  201.67  199.00  198.67  200.00  198.67  199.00
Середні x:  [1.5, -1.0, -0.5, -1.5, -0.75, 0.5, 0.75, 10.85, 18.55, 50.98]
Отримане рівняння регресії
198.41 + 0.39*x1 + 0.13*x2 + -0.07*x3 + -0.08*x1x2 + 0.00*x1x3 + -0.02*x2x3 + -0.00*x1x2x3 + -0.02*x1^2 + -0.00*x2^2 + 0.00*x3^2 = ŷ
```



Висновок: у ході виконання лабораторної роботи №5 було проведено трьохфакторний експеримент з урахуванням квадратичних членів, використовуючи центральний ортогональний композиційний план. Кінцева мета роботи досягнута.