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 №4:

«ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З УРАХУВАННЯМ ЕФЕКТУ ВЗАЄМОДІЇ»

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

Київ 2020р.

Лабораторна робота №4

Тема: ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З УРАХУВАННЯМ ЕФЕКТУ ВЗАЄМОДІЇ.

Мета: Провести повний трьохфакторний експеримент. Знайти рівняння регресії адекватне об'єкту.

Виконання:

Варіант – 128.

128	-40	20	-25	10	-25	-10

1. Лістинг програми:

```
from math import *


from numpy import *
import numpy as np



class Crit_vals:
    @staticmethod
    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__()

    @staticmethod
    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__()

    @staticmethod
```

```
    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__()

cr = Crit_vals()


def dob(*args):
    res = [1 for _ in range(len(args[0]))]
    for i in range(len(args[0])):
        for j in args:
            res[i] *= j[i]

    return res


def getcolumn(arr, n):

    return [i[n] for i in arr]


inp_m = input("Введіть m, або просто натисніть 'Enter', тоді m = 3: ")
inp_p = input("Введіть довірчу ймовірність, або просто натисніть 'Enter', тоді p = 0.95: ")
m = int(inp_m) if inp_m else 3
p = float(inp_p) if inp_p else 0.95

rows = N = 8
x1_min, x1_max = -40, 20
x2_min, x2_max = -25, 10
x3_min, x3_max = -25, -10
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_cod_for4 = [
    [+1, -1, -1, -1],
```

```
    [+1, -1, +1, +1],
    [+1, +1, -1, +1],
    [+1, +1, +1, -1]
]

matrix_x_for4 = [
    [x1_min, x2_min, x3_min],
    [x1_min, x2_max, x3_max],
    [x1_max, x2_min, x3_max],
    [x1_max, x2_max, x3_min]
]
matrix_x_for4 = np.array(matrix_x_for4)

# матриця кодованих значень x
matrix_x_cod = [
    [+1, -1, -1, -1, +1, +1, +1, -1],
    [+1, -1, -1, +1, +1, -1, -1, +1],
    [+1, -1, +1, -1, -1, +1, -1, +1],
    [+1, -1, +1, +1, -1, -1, +1, -1],
    [+1, +1, -1, -1, -1, -1, +1, +1],
    [+1, +1, -1, +1, -1, +1, -1, -1],
    [+1, +1, +1, -1, +1, -1, -1, -1],
    [+1, +1, +1, +1, +1, +1, +1, +1]
]

# матриця значень x
matrix_x = [
    [1, x1_min, x2_min, x3_min, x1_min * x2_min, x1_min * x3_min, x2_min * x3_min, x1_min * x2_min * x3_min],
    [1, x1_min, x2_min, x3_max, x1_min * x2_min, x1_min * x3_max, x2_min * x3_max, x1_min * x2_min * x3_max],
    [1, x1_min, x2_max, x3_min, x1_min * x2_max, x1_min * x3_min, x2_max * x3_min, x1_min * x2_max * x3_min],
    [1, x1_min, x2_max, x3_max, x1_min * x2_max, x1_min * x3_max, x2_max * x3_max, x1_min * x2_max * x3_max],
    [1, x1_max, x2_min, x3_min, x1_max * x2_min, x1_min * x3_min, x2_min * x3_min, x1_min * x2_min * x3_min],
    [1, x1_max, x2_min, x3_max, x1_max * x2_min, x1_max * x3_max, x2_min * x3_max, x1_max * x2_min * x3_max],
    [1, x1_max, x2_max, x3_min, x1_max * x2_max, x1_max * x3_min, x2_max * x3_min, x1_max * x2_max * x3_min],
    [1, x1_max, x2_max, x3_max, x1_max * x2_max, x1_max * x3_max, x2_max * x3_max, x1_max * x2_max * x3_max]
]

check = True
while check:
    # матриця рандомних значень y
```

```
random_matrix_y = random.randint(y_min, y_max, size=(rows, m))

# сума середніх значень відгуку функції за рядками
def sum_rows(random_matrix_y):
    y = np.sum(random_matrix_y, axis=1) / m

    return y

Yavg = sum_rows(random_matrix_y)

def sum_columns(matrix_x_for4):
    mx = np.sum(matrix_x_for4, axis=0) / 4

    return mx

mx = sum_columns(matrix_x_for4)

# Нормовані коефіцієнти рівняння регресії
def sum_my(y1, y2, y3, y4):
    my = (y1 + y2 + y3 + y4) / 4

    return my

my = sum_my(Yavg[0], Yavg[3], Yavg[5], Yavg[6])

# Нормовані коефіцієнти рівняння регресії
def find_a(a, b, c, d):
    az = (a * Yavg[0] + b * Yavg[3] + c * Yavg[5] + d * Yavg[6]) / 4

    return az

a1 = find_a(x1_min, x1_min, x1_max, x1_max)
a2 = find_a(x2_min, x2_max, x2_min, x2_max)
a3 = find_a(x3_min, x3_max, x3_max, x3_min)

# Нормовані коефіцієнти рівняння регресії
def find_aa(a, b, c, d):
    aa = (a ** 2 + b ** 2 + c ** 2 + d ** 2) / 4

    return aa
```

```
a11 = find_aa(x1_min, x1_min, x1_max, x1_max)
a22 = find_aa(x2_min, x2_max, x2_min, x2_max)
a33 = find_aa(x3_min, x3_max, x3_max, x3_min)

# Нормовані коефіцієнти рівняння регресії
a12 = a21 = (x1_min * x2_min + x1_min * x2_max + x1_max * x2_min + x1_max * x2_max) / 4
a13 = a31 = (x1_min * x3_min + x1_min * x3_max + x1_max * x3_max + x1_max * x3_min) / 4
a23 = a32 = (x2_min * x3_min + x2_max * x3_max + x2_min * x3_max + x2_max * x3_min) / 4

# Матриця для визначення коефіцієнтів регресії
A = [[my, mx[0], mx[1], mx[2]], [a1, a11, a12, a13], [a2, a12, a22, a32], [a3, a13, a23, a33]]
B = [[1, my, mx[1], mx[2]], [mx[0], a1, a12, a13], [mx[1], a2, a22, a32], [mx[2], a3, a23, a33]]
C = [[1, mx[0], my, mx[2]], [mx[0], a11, a1, a13], [mx[1], a12, a2, a32], [mx[2], a13, a3, a33]]
D = [[1, mx[0], mx[1], my], [mx[0], a11, a12, a1], [mx[1], a12, a22, a2], [mx[2], a13, a23, a3]]
E = [[1, mx[0], mx[1], mx[2]], [mx[0], a11, a12, a13], [mx[1], a12, a22, a32], [mx[2], a13, a23, a33]]
X = []

# Коефіцієнти регресії
def coef_regr(a, b):
    b = linalg.det(a) / linalg.det(b)

    return b

b0 = coef_regr(A, E)
b1 = coef_regr(B, E)
b2 = coef_regr(C, E)
b3 = coef_regr(D, E)
X.append(round(b0, 2))
X.append(round(b1, 2))
X.append(round(b2, 2))
X.append(round(b3, 2))

# Нормоване рівняння регресії
def find_y_norm(a, b, c):
    y_norm = X[0] + X[1] * a + X[2] * b + X[3] * c

    return y_norm

y_norm1 = find_y_norm(x1_min, x2_min, x3_min)
```

```
    y_norm2 = find_y_norm(x1_min, x2_max, x3_max)
    y_norm3 = find_y_norm(x1_max, x2_min, x3_max)
    y_norm4 = find_y_norm(x1_max, x2_max, x3_min)

    # Перевірка однорідності дисперсії за критерієм Кохрена

    # Пошук дисперсій по рядкам
    dispersion_y = [0, 0, 0, 0]

    for i in range(m):
        dispersion_y[0] += ((random_matrix_y[0][i] - Yavg[0]) ** 2) / m
        dispersion_y[1] += ((random_matrix_y[1][i] - Yavg[3]) ** 2) / m
        dispersion_y[2] += ((random_matrix_y[2][i] - Yavg[5]) ** 2) / m
        dispersion_y[3] += ((random_matrix_y[3][i] - Yavg[6]) ** 2) / m

    ajk = dispersion_y[0] + dispersion_y[1] + dispersion_y[2] + dispersion_y[3]

    Gp = 0
    if ajk == 0:
        m += 1
        print("Збільшуємо m на одиницю")
    else:
        Gp = max(dispersion_y) / (ajk)
        f1 = m - 1
        f2 = rows
        q = 1 - p
        Gt = Crit_vals.get_cohren_value(f2, f1, q)
        if Gp <= Gt:
            print("Дисперсія однорідна")
            check = False
        else:
            m += 1
            print("Збільшуємо m на одиницю")

# Значимість коефіцієнтів за критерієм Стьюдента
f1 = m - 1
f2 = rows
f3 = f1 * f2
Ft = cr.get_student_value(f3, q)
Sb = sum(dispersion_y) / rows
```

```
Sbetakvadr = Sb / (rows * m)
Sbeta = sqrt(Sb / (rows * m))


# Визначення оцінки коефіцієнтів
def find_beta(a, b, c, d):
    beta = (Yavg[0] * a + Yavg[3] * b + Yavg[5] * c + Yavg[6] * d) / rows

    return beta

beta0 = find_beta(matrix_x_cod[0][0], matrix_x_cod[1][0], matrix_x_cod[2][0], matrix_x_cod[3][0])
beta1 = find_beta(matrix_x_cod[0][1], matrix_x_cod[1][1], matrix_x_cod[2][1], matrix_x_cod[3][1])
beta2 = find_beta(matrix_x_cod[0][2], matrix_x_cod[1][2], matrix_x_cod[2][2], matrix_x_cod[3][2])
beta3 = find_beta(matrix_x_cod[0][3], matrix_x_cod[1][3], matrix_x_cod[2][3], matrix_x_cod[3][3])


# Пошук коефіцієнта t
def find_t(a, b):
    t = a / b

    return t


t0 = find_t(beta0, Sbeta)
t1 = find_t(beta1, Sbeta)
t2 = find_t(beta2, Sbeta)
t3 = find_t(beta3, Sbeta)
t_list = [fabs(t0), fabs(t1), fabs(t2), fabs(t3)]
b_list = [b0, b1, b2, b3]

tbool = tuple(Ft < i for i in t_list)


# Рівняння з урахуванням критерію Стьюдента
def find_yj(a, b, c):
    yj = b_list[0] + b_list[1] * a + b_list[2] * b + b_list[3] * c

    return yj
```

```
yj1 = find_yj(x1_min, x2_min, x3_min)
yj2 = find_yj(x1_min, x2_max, x3_max)
yj3 = find_yj(x1_max, x2_min, x3_max)
yj4 = find_yj(x1_max, x2_max, x3_min)

# Перевірка умови за критерієм Фішера
# кількість значимих коефіцієнтів
d = tbool.count(True)
f1 = m - 1
f2 = rows
f4 = rows - d
f3 = f1 * f2
Sad = m * (((yj1 - Yavg[0]) ** 2 + (yj2 - Yavg[3]) ** 2 + (yj3 - Yavg[5]) ** 2 + (yj4 - Yavg[6]) ** 2)) / f4
Fp = Sad / Sbetakvadr
Fp = cr.get_fisher_value(f3, f4, q)

print("_" * 30)
print("Рівняння регресії: ŷ = b0 + b1*x1 + b2*x2+ b3*x3 ")
print(f"Середнє максимальне x: {x_avarage_max:.2f}")
print("Середнє мінімальне x: {:.2f}".format(x_avarage_min))
print("y_max: {:.2f} \ty_min: {:.2f}".format(y_max, y_min))
print("_" * 30)
print("Матриця кодованих значень X: \n", matrix_x_cod_for4)
print("_" * 30)
print("Матриця для значень X: \n", matrix_x_for4)
print("_" * 30)
print("Матриця для значень Y: \n", random_matrix_y)
print("_" * 30)
print("y1: {:.2f} \ty2: {:.2f} \ty3: {:.2f} \ty4: {:.2f}".format(Yavg[0], Yavg[3], Yavg[5], Yavg[6]))
print("mx: {:.2f} \t{:.2f} \t{:.2f}".format(mx[0], mx[1], mx[2]))
print("my: {:.2f}".format(my))
print("_" * 30)
print("Коефіцієнти b0, b1, b2, b3: \n", X)
print("_" * 30)
print("Нормоване рівняння регресії y = {:.2f} + {:.2f}*x1 + {:.2f}*x2".format(X[0], X[1], X[2]))
print("{:.1f} + {:.1f} + {:.1f} + {:.1f} = {:.1f}".format(X[0], X[1] * x1_min, X[2] * x2_min, X[3] * x3_min, y_norm1))
print("{:.1f} + {:.1f} + {:.1f} + {:.1f} = {:.1f}".format(X[0], X[1] * x1_min, X[2] * x2_max, X[3] * x3_max, y_norm2))
print("{:.1f} + {:.1f} + {:.1f} + {:.1f} = {:.1f}".format(X[0], X[1] * x1_max, X[2] * x2_min, X[3] * x3_max, y_norm3))
print("{:.1f} + {:.1f} + {:.1f} + {:.1f} = {:.1f}".format(X[0], X[1] * x1_max, X[2] * x2_max, X[3] * x3_min, y_norm4))
print("_" * 30)
```

```
print("Перевірка за Кохреном")
print("S^2{y2}:", round(dispersion_y[0], 2))
print("S^2{y2}:", round(dispersion_y[1], 2))
print("S^2{y3}:", round(dispersion_y[2], 2))
print("S^2{y4}:", round(dispersion_y[3], 2))
print("Gp: ", Gp)
print("_" * 30)
print("Перевірка за Стьюдентом")
print("Sb^2: {:.2f} \t\tS^2(β): {:.2f} \t\tS(β): {:.2f}".format(Sb, Sbetakvadr, Sbeta))
print("β1: {:.2f} \t\tβ2: {:.2f} \t\tβ3: {:.2f} \t\tβ4: {:.2f}".format(beta0, beta1, beta2, beta3))
print("t0: {:.2f} \t\tt1: {:.2f} \t\tt2: {:.2f} \t\tt3: {:.2f}".format(t0, t1, t2, t3))
print("ŷ1: {:.2f} \t\tŷ2: {:.2f} \t\tŷ3: {:.2f} \t\tŷ4: {:.2f}".format(yj1, yj2, yj3, yj4))
print("_" * 30)
print("Перевірка за Фішером")
print("Sad^2: {:.2f} \nFp: {:.2f}".format(Sad, Fp))
print("_" * 30)

if Fp < Ft:
    print("Рівняння регресії адекватно оригіналу при рівні значимості 0.05")
    cont = False
else:
    cont = True
    print("Рівняння регресії неадекватно оригіналу при рівні значимості 0.05, додамо ефект взаємодії")

# Ефект взаємодії

if cont == True:
    while True:
        # Нормовані коефіцієнти рівняння регресії
        # сума середніх значень відгуку функції за рядками
        def sum_rows(random_matrix_y):
            y = np.sum(random_matrix_y, axis=1) / rows

            return y

        y1_full = tuple(sum_rows(random_matrix_y))
        print("Рівняння регресії: \nŷ = b0 + b1*x1 + b2*x2 + b3*x3 + b12*x1*x2 + b13*x1*x3 + b23*x2*x3 +
b123*x1*x2*x3")

        def sum_columns(matrix_x):
```

```
    mx = np.sum(matrix_x, axis=0) / rows

    return mx

mx = sum_columns(matrix_x)
# Знайдемо детермінант для знаходження коефіцієнтів b

# Знаменник для детермінанту
forb = [[i[j] for i in matrix_x] for j in range(8)]
determinant = list(list(sum(dob(forb[i], forb[j])) for j in range(8)) for i in range(8))

# Чисельники для детермінанту
k = [sum(dob(y1_full, forb[i])) for i in range(N)]
numerators = [[determinant[i][0:j] + [k[i]] + determinant[i][j + 1:] for i in range(N)] for j in range(N)]
matrix_for_numerators = np.array(numerators)

# Рахуємо детермінант
bs1 = [np.linalg.det(i) / np.linalg.det(determinant) for i in numerators]
test = [[i[j] for i in forb] for j in range(N)]
matrix_for_test = np.array(test)
eq1 = [sum(dob(bs1, test[i])) for i in range(N)]

# Коефіцієнти регресії
def find_beta(x1, x2, x3, x4, x5, x6, x7, x8):
    beta = (y1_full[0] * x1 + y1_full[1] * x2 + y1_full[2] * x3 + y1_full[3] * x4 + y1_full[4] * x5 + y1_full[
        5] * x6 + y1_full[6] * x7 + y1_full[7] * x8) / rows

    return beta

beta0 = find_beta(matrix_x_cod[0][0], matrix_x_cod[1][0], matrix_x_cod[2][0], matrix_x_cod[3][0],
                  matrix_x_cod[4][0], matrix_x_cod[5][0], matrix_x_cod[6][0], matrix_x_cod[7][0])
beta1 = find_beta(matrix_x_cod[0][1], matrix_x_cod[1][1], matrix_x_cod[2][1], matrix_x_cod[3][1],
                  matrix_x_cod[4][1], matrix_x_cod[5][1], matrix_x_cod[6][1], matrix_x_cod[7][1])
beta2 = find_beta(matrix_x_cod[0][2], matrix_x_cod[1][2], matrix_x_cod[2][2], matrix_x_cod[3][2],
                  matrix_x_cod[4][2], matrix_x_cod[5][2], matrix_x_cod[6][2], matrix_x_cod[7][2])
beta3 = find_beta(matrix_x_cod[0][3], matrix_x_cod[1][3], matrix_x_cod[2][3], matrix_x_cod[3][3],
                  matrix_x_cod[4][3], matrix_x_cod[5][3], matrix_x_cod[6][3], matrix_x_cod[7][3])
beta4 = find_beta(matrix_x_cod[0][4], matrix_x_cod[1][4], matrix_x_cod[2][4], matrix_x_cod[3][4],
                  matrix_x_cod[4][4], matrix_x_cod[5][4], matrix_x_cod[6][4], matrix_x_cod[7][4])
beta5 = find_beta(matrix_x_cod[0][5], matrix_x_cod[1][5], matrix_x_cod[2][5], matrix_x_cod[3][5],
```

```
                matrix_x_cod[4][5], matrix_x_cod[5][5], matrix_x_cod[6][5], matrix_x_cod[7][5])
    beta6 = find_beta(matrix_x_cod[0][6], matrix_x_cod[1][6], matrix_x_cod[2][6], matrix_x_cod[3][6],
                matrix_x_cod[4][6], matrix_x_cod[5][6], matrix_x_cod[6][6], matrix_x_cod[7][6])
    beta7 = find_beta(matrix_x_cod[0][7], matrix_x_cod[1][7], matrix_x_cod[2][7], matrix_x_cod[3][7],
                matrix_x_cod[4][7], matrix_x_cod[5][7], matrix_x_cod[6][7], matrix_x_cod[7][7])

    beta_all = []
    beta_all.append(beta0)
    beta_all.append(beta1)
    beta_all.append(beta2)
    beta_all.append(beta3)
    beta_all.append(beta4)
    beta_all.append(beta5)
    beta_all.append(beta6)
    beta_all.append(beta7)

    eq2 = [sum(dob(beta_all, matrix_x_cod[i])) for i in range(N)]

    # Кохрен
    S = [sum([(y1_full[i] - random_matrix_y[j][i]) ** 2 for i in range(m)]) / m for j in range(N)]
    Gp = max(S) / sum(S)
    f1 = m - 1
    f2 = N
    Gt = Crit_vals.get_cohren_value(f2, f1, q)
    if Gp > Gt:
        m += 1
        print("Дисперсія не однорідна, збільшуємо m")
        if len(random_matrix_y[0]) < m:
            for i in range(8):
                random_matrix_y[i].append(random.randrange(y_min, y_max))
    else:
        print("Дисперсія однорідна")
        break
# Стьюдент
S_B = sum(S) / len(S)
S2_b = S_B / (m * len(S))
S_b = S2_b ** (1 / 2)
beta = tuple(sum(dob(getcolumn(matrix_x_cod, i), y1_full)) / 8 for i in range(8))
t = tuple(abs(i) / S_b for i in beta)
f3 = f1 * f2
```

```
Ft = cr.get_student_value(f3, q)

tbool = tuple(Ft < i for i in t)
bzn = tuple(bs1[i] if tbool[i] else 0 for i in range(8))
yzn = tuple(sum(dob(bzn, test[i])) for i in range(8))



# Фішер
d = tbool.count(True)
f4 = 8 - d
S2_ad = m * sum([(y1_full[i] - yzn[i]) ** 2 for i in range(8)]) / f4
Fp = S2_ad / S_B
Ft = cr.get_fisher_value(f3, f4, q)

print("_" * 30)
print("Перевірка за Кохреном")
print("S^2{y2}:", round(S[0], 2))
print("S^2{y2}:", round(S[1], 2))
print("S^2{y3}:", round(S[2], 2))
print("S^2{y4}:", round(S[3], 2))
print("S^2{y5}:", round(S[4], 2))
print("S^2{y6}:", round(S[5], 2))
print("S^2{y7}:", round(S[6], 2))
print("S^2{y8}:", round(S[7], 2))
print("Gp: ", Gp)
print("_" * 30)
print("Перевірка за Стьюдентом")
print("Sb^2: {:.2f} \t\tS^2(β): {:.2f} \t\tS(β): {:.2f}".format(S_B, S2_b, S_b))
print("β1: {:.2f} \t\t\tβ2: {:.2f} \t\tβ3: {:.2f} \t\tβ4: {:.2f}".format(beta[0], beta[1], beta[2], beta[3]))
print("β5: {:.2f} \t\t\tβ6: {:.2f} \t\tβ7: {:.2f} \t\tβ8: {:.2f}".format(beta[4], beta[5], beta[6], beta[7]))
print("t0: {:.2f} \t\t\tt1: {:.2f} \t\t\tt2: {:.2f} \t\tt3: {:.2f}".format(t[0], t[1], t[2], t[3]))
print("t4: {:.2f} \t\t\tt5: {:.2f} \t\t\tt6: {:.2f} \t\tt7: {:.2f}".format(t[4], t[5], t[6], t[7]))
print("_" * 30)
print("Перевірка за Фішером")
print("Sad^2: {:.2f} \nFp: {:.2f}".format(S2_ad, Fp))
print("_" * 30)
if Fp < Ft:
    print("Отримане рівняння - адекватне")
    cont = False
else:
```

```
cont = True
print("Отримане рівняння - неадекватне. Врахування ефекту взаємодії не допомогло.")
```

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:
Дисперсія однорідна
_____
Рівняння регресії: ŷ = b0 + b1*x1 + b2*x2+ b3*x3
Середнє максимальне x: 6.67
Середнє мінімальне x: -30.00
y_max: 206.67   y_min: 170.00
_____
Матриця кодованих значень X:
 [[1, -1, -1, -1], [1, -1, 1, 1], [1, 1, -1, 1], [1, 1, 1, -1]]
_____
Матриця для значень X:
 [[-40 -25 -25]
 [-40  10 -10]
 [ 20 -25 -10]
 [ 20  10 -25]]
_____
Матриця для значень Y:
 [[183 171 170]
 [181 177 185]
 [178 173 198]
 [195 203 193]
 [201 185 171]
 [190 192 181]
 [186 180 170]
 [202 181 204]]
_____
y1: 174.67  y2: 197.00  y3: 187.67  y4: 178.67
mx: -10.00  -7.50   -17.50
my: 184.50
_____
Коефіцієнти b0, b1, b2, b3:
 [203.76, -0.04, 0.19, 1.04]
_____
```

```
Нормоване рівняння регресії y = 203.76 + -0.04*x1 + 0.19*x2
203.8 + 1.6 + -4.8 + -26.0 = 174.6
203.8 + 1.6 + 1.9 + -10.4 = 196.9
203.8 + -0.0 + -4.8 + -10.4 = 187.8
203.8 + -0.0 + 1.9 + -26.0 = 178.9
_____
Перевірка за Кохреном
S^2(y2):  34.89
S^2(y2):  266.67
S^2(y3):  138.44
S^2(y4):  354.78
Gp:  0.44630613169299624
_____
Перевірка за Стьюдентом
Sb^2: 99.35         S^2(β): 4.14        S(β): 2.03
β1: 52.25           β2: -92.25      β3: -0.67       β4: 1.67
t0: 45.34           t1: -45.34          t2: -0.33       t3: 0.82
ŷ1: 174.67          ŷ2: 197.00          ŷ3: 187.67      ŷ4: 178.67
_____
Перевірка за Фішером
Sad^2: 0.00
Fp: 2.74
_____
Рівняння регресії неадекватно оригіналу при рівні значимості 0.05, додамо ефект взаємодії
Рівняння регресії:
ŷ = b0 + b1*x1 + b2*x2 + b3*x3 + b12*x1*x2 + b13*x1*x3 + b23*x2*x3 + b123*x1*x2*x3
Дисперсія однорідна
_____
```

```
Перевірка за Кохреном
S^2{y2}:  11572.64
S^2{y2}:  12930.55
S^2{y3}:  13481.8
S^2{y4}:  16832.72
S^2{y5}:  14186.39
S^2{y6}:  14511.8
S^2{y7}:  12456.39
S^2{y8}:  16585.3
Gp:  0.1495476204402043
_____________________________
Перевірка за Стьюдентом
Sb^2: 14069.70      S^2(β): 586.24      S(β): 24.21
β1: 69.53           β2: 0.56        β3: 1.19        β4: 1.84
β5: -1.09           β6: -0.06       β7: 1.06        β8: 0.34
t0: 2.87            t1: 0.02            t2: 0.05        t3: 0.08
t4: 0.05            t5: 0.00            t6: 0.04        t7: 0.01
_____________________________
Перевірка за Фішером
Sad^2: 170.23
Fp: 0.01
_____________________________
Отримане рівняння - адекватне
```

Висновок: Під час виконання лабораторної роботи був проведений повний трьохфакторний експеримент при використанні рівняння з ефектом взаємодії. Складено матрицю планування, знайдено коефіцієнти рівняння регресії, проведено 3 статистичні перевірки.