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»

Виконала:
студентка групи ІВ-81
Дрозд С.В.
Залікова книжка № 8111
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 – 111.

111	-25	-5	-30	45	-5	5

Код:

```
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 = -25, -5
x2_min, x2_max = -30, 45
x3_min, x3_max = -5, 5
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_cod_for4 = np.array(matrix_x_cod_for4)

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
```

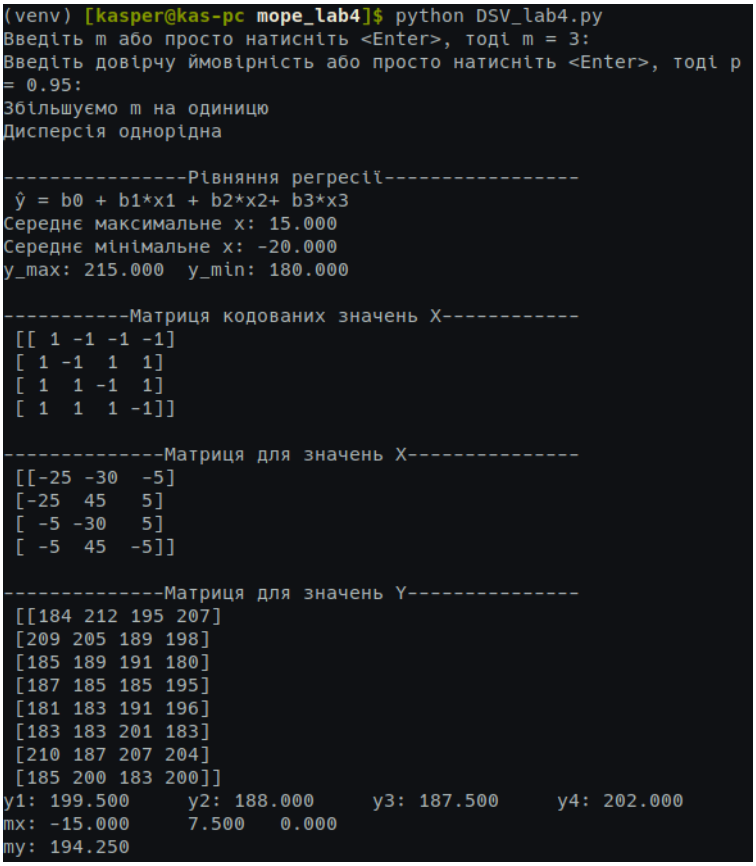
```
# кількість значимих коефіцієнтів
d = tbool.count(True)
f1 = m - 1
f2 = rows
f4 = rows - d
f3 = f1 * f2
Sad = m * (((yj1 - Yavg[0]) ** 2 + (yj2 - Yavg[3]) ** 2 + (yj3 - Yavg[5]) ** 2 + (yj4 - Yavg[6]) ** 2)) / f4
Fp = Sad / Sbetakvadr
Fp = cr.get_fisher_value(f3, f4, q)

print(f"\n{'Рівняння регресії':-^50}\n ŷ = b0 + b1*x1 + b2*x2+ b3*x3\n"
   f"Середнє максимальне x: {x_avarage_max:.3f}\nСереднє мінімальне x: {x_avarage_min:.3f}\n"
   f"y_max: {y_max:.3f} \ty_min: {y_min:.3f}\n\n"
   f"{'Матриця кодованих значень X':-^50}\n", matrix_x_cod_for4,
   f"\n\n{'Матриця для значень X':-^50}\n", matrix_x_for4,
   f"\n\n{'Матриця для значень Y':-^50}\n", random_matrix_y,
   f"\ny1: {Yavg[0]:.3f} \ty2: {Yavg[3]:.3f} \ty3: {Yavg[5]:.3f} \t"
   f"y4: {Yavg[6]:.3f}\nmx: {mx[0]:.3f} \t{mx[1]:.3f} \t{mx[2]:.3f}\n"
   f"my: {my:.3f}\n\n{'Коефіцієнти b0, b1, b2, b3':-^50}\n", X,
   f"\n\n{'Нормоване рівняння регресії':-^50}\n"
   f"y = {X[0]:.3f} {X[1]:+.3f}*x1 {X[2]:+.3f}*x2\n"
   f"{X[0]:.3f} {X[1] * x1_min:+.3f} {X[2] * x2_min:+.3f} {X[3] * x3_min:+.3f} = {y_norm1:.3f}\n"
   f"{X[0]:.3f} {X[1] * x1_min:+.3f} {X[2] * x2_max:+.3f} {X[3] * x3_max:+.3f} = {y_norm2:.3f}\n"
   f"{X[0]:.3f} {X[1] * x1_max:+.3f} {X[2] * x2_min:+.3f} {X[3] * x3_max:+.3f} = {y_norm3:.3f}\n"
   f"{X[0]:.3f} {X[1] * x1_max:+.3f} {X[2] * x2_max:+.3f} {X[3] * x3_min:+.3f} = {y_norm4:.3f}\n"
   f"\n{'Перевірка за Кохреном':-^50}\nS^2(y1): {dispersion_y[0]:.3f}\n"
   f"S^2(y2): {dispersion_y[1]:.3f}\nS^2(y3): {dispersion_y[2]:.3f}\nS^2(y4): {dispersion_y[3]:.3f}"
   f"\nGp: {Gp:.3f}\n\n{'Перевірка за Стьюдентом':-^50}\nSb^2: {Sb:.3f} \t\tS^2(β): {Sbetakvadr:.3f}"
   f" \t\tS(β): {Sbeta:.3f}\nβ1: {beta0:.3f} \t\t\tβ2: {beta1:.3f}"
   f"\nβ3: {beta2:.3f} \t\tβ4: {beta3:.3f}\nt0: {t0:.3f} \t\t\tt1: {t1:.3f}"
   f"\nt2: {t2:.3f} \t\tt3: {t3:.3f}\nŷ1: {yj1:.3f} \t\t\tŷ2: {yj2:.3f}"
   f"\nŷ3: {yj3:.3f} \t\tŷ4: {yj4:.3f}\n\n{'Перевірка за Фішером':-^50}"
   f"\nSad^2: {Sad:.3f} \nFp: {Fp:.3f}\n")

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(f"\n{'Рівняння регресії':-^50}\nŷ = b0 + b1*x1 + b2*x2"
         f" + b3*x3 + b12*x1*x2 + b13*x1*x3 + b23*x2*x3 +\n"
         f"+ 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 нашого 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 нашого 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)]
```

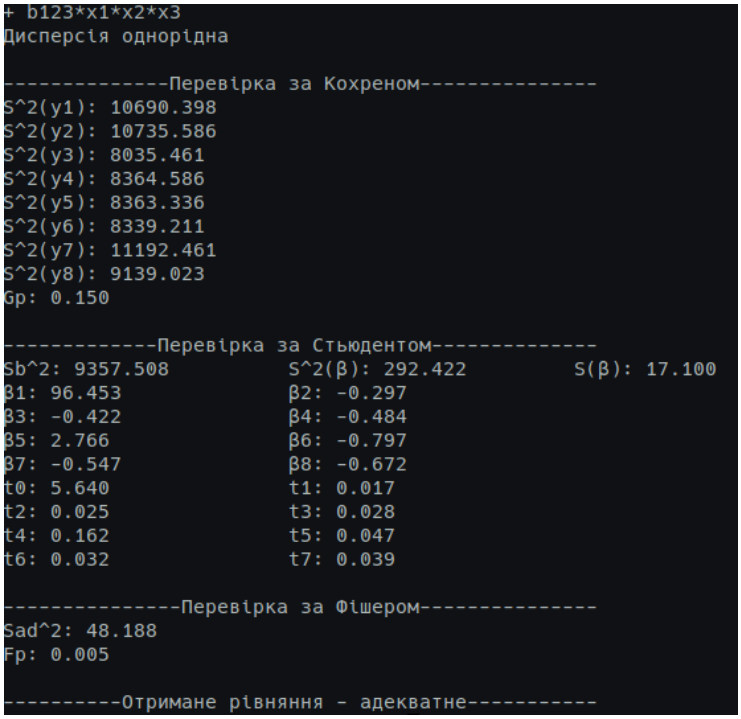
```
    # Перевірка кохрена
    S = [sum([(y1_full[i] - random_matrix_y[j][i]) ** 2 for i in range(m)]) / m for j in range(N)]
    Gp = max(S) / sum(S)
    f1 = m - 1
    f2 = N
    Gt = Crit_vals.get_cohren_value(f2, f1, q)
    if Gp > Gt:
        m += 1
        print("Дисперсія не однорідна, збільшуємо 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(f"\n{'Перевірка за Кохреном':-^50}\nS^2(y1): {S[0]:.3f}\n"
    f"S^2(y2): {S[1]:.3f}\nS^2(y3): {S[2]:.3f}\nS^2(y4): "
    f"{S[3]:.3f}\nS^2(y5): {S[4]:.3f}\nS^2(y6): {S[5]:.3f}\n"
    f"S^2(y7): {S[6]:.3f}\nS^2(y8): {S[7]:.3f}\nGp: {Gp:.3f}\n\n"
    f"{'Перевірка за Стьюдентом':-^50}\nSb^2: {S_B:.3f} \t\t"
    f"S^2(β): {S2_b:.3f}\t\tS(β): {S_b:.3f}\n"
    f"β1: {beta[0]:.3f}\t\tβ2: {beta[1]:.3f}\n"
    f"β3: {beta[2]:.3f}\t\tβ4: {beta[3]:.3f}\n"
    f"β5: {beta[4]:.3f}\t\tβ6: {beta[5]:.3f}\n"
    f"β7: {beta[6]:.3f}\t\tβ8: {beta[7]:.3f}\n"
    f"t0: {t[0]:.3f}\t\tt1: {t[1]:.3f}\n"
    f"t2: {t[2]:.3f}\t\tt3: {t[3]:.3f}\n"
    f"t4: {t[4]:.3f}\t\tt5: {t[5]:.3f}\n"
    f"t6: {t[6]:.3f}\t\tt7: {t[7]:.3f}\n\n"
    f"{'Перевірка за Фішером':-^50}\nSad^2: {S2_ad:.3f}\n"
    f"Fp: {Fp:.3f}\n")
if Fp < Ft:
    print(f"{'Отримане рівняння - адекватне':-^50}")
    cont = False
else:
    cont = True
    print(f"{'Отримане рівняння - неадекватне':-^50}\n"
```

f"Врахування ефекту взаємодії не допомогло.")

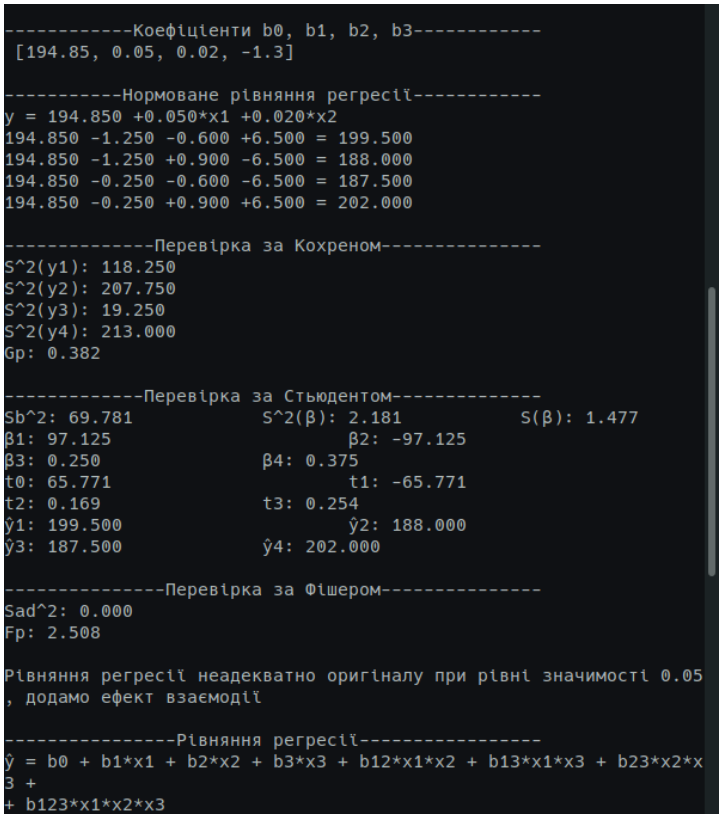
Результат виконання роботи програми:



```
(venv) [kasper@kas-pc mope_lab4]$ python DSV_lab4.py
Введіть m або просто натисніть <Enter>, тоді m = 3:
Введіть довірчу ймовірність або просто натисніть <Enter>, тоді p
= 0.95:
Збільшуємо m на одиницю
Дисперсія однорідна

----------------Рівняння регресії-----------------
 ŷ = b0 + b1*x1 + b2*x2+ b3*x3
Середнє максимальне x: 15.000
Середнє мінімальне x: -20.000
y_max: 215.000  y_min: 180.000

-----------Матриця кодованих значень X------------
 [[ 1 -1 -1 -1]
 [ 1 -1  1  1]
 [ 1  1 -1  1]
 [ 1  1  1 -1]]

--------------Матриця для значень X---------------
 [[-25 -30  -5]
 [-25  45   5]
 [ -5 -30   5]
 [ -5  45  -5]]

--------------Матриця для значень Y---------------
 [[184 212 195 207]
 [209 205 189 198]
 [185 189 191 180]
 [187 185 185 195]
 [181 183 191 196]
 [183 183 201 183]
 [210 187 207 204]
 [185 200 183 200]]
y1: 199.500     y2: 188.000     y3: 187.500     y4: 202.000
mx: -15.000     7.500   0.000
my: 194.250
```



```
+ b123*x1*x2*x3
Дисперсія однорідна

--------------Перевірка за Кохреном---------------
S^2(y1): 10690.398
S^2(y2): 10735.586
S^2(y3): 8035.461
S^2(y4): 8364.586
S^2(y5): 8363.336
S^2(y6): 8339.211
S^2(y7): 11192.461
S^2(y8): 9139.023
Gp: 0.150

-------------Перевірка за Стьюдентом--------------
Sb^2: 9357.508          S^2(β): 292.422         S(β): 17.100
β1: 96.453              β2: -0.297
β3: -0.422              β4: -0.484
β5: 2.766               β6: -0.797
β7: -0.547              β8: -0.672
t0: 5.640               t1: 0.017
t2: 0.025               t3: 0.028
t4: 0.162               t5: 0.047
t6: 0.032               t7: 0.039

---------------Перевірка за Фішером----------------
Sad^2: 48.188
Fp: 0.005

----------Отримане рівняння - адекватне-----------
```



```
------------Коефіцієнти b0, b1, b2, b3------------
 [194.85, 0.05, 0.02, -1.3]

-----------Нормоване рівняння регресії------------
y = 194.850 +0.050*x1 +0.020*x2
194.850 -1.250 -0.600 +6.500 = 199.500
194.850 -1.250 +0.900 -6.500 = 188.000
194.850 -0.250 -0.600 -6.500 = 187.500
194.850 -0.250 +0.900 +6.500 = 202.000

--------------Перевірка за Кохреном---------------
S^2(y1): 118.250
S^2(y2): 207.750
S^2(y3): 19.250
S^2(y4): 213.000
Gp: 0.382

-------------Перевірка за Стьюдентом--------------
Sb^2: 69.781            S^2(β): 2.181           S(β): 1.477
β1: 97.125                      β2: -97.125
β3: 0.250               β4: 0.375
t0: 65.771                      t1: -65.771
t2: 0.169               t3: 0.254
ŷ1: 199.500                     ŷ2: 188.000
ŷ3: 187.500             ŷ4: 202.000

---------------Перевірка за Фішером----------------
Sad^2: 0.000
Fp: 2.508

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*x
3 +
+ b123*x1*x2*x3
```

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.