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.



**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 х

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 х

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 х

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 у

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 коефіціенти 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 коефіціенти 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 коефіціенти 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 коефіціенти 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 коефіціентів 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 = []

# Коефіціенти 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 коефіціентів 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 коефіціентів

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 коефіціента 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 коефіціентів

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 х: {x\_avarage\_max:.3f}\nСереднє мінімальне х: {x\_avarage\_min:.3f}\n"

f"y\_max: {y\_max:.3f} \ty\_min: {y\_min:.3f}\n\n"

f"{'Матриця кодованих значень Х':-^50}\n", matrix\_x\_cod\_for4,

f"\n\n{'Матриця для значень Х':-^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{'Коефіціенти 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("Pівняння регресії адекватно оригіналу при рівні значимості 0.05")

cont = False

else:

cont = True

print("Pівняння регресії неадекватно оригіналу при рівні значимості 0.05, додамо ефект взаємодії")

# Ефект взаємодії

if cont == True:

while True:

# Нормовані коефіціенти 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 коефіціентів 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)]

# Коефіціенти 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

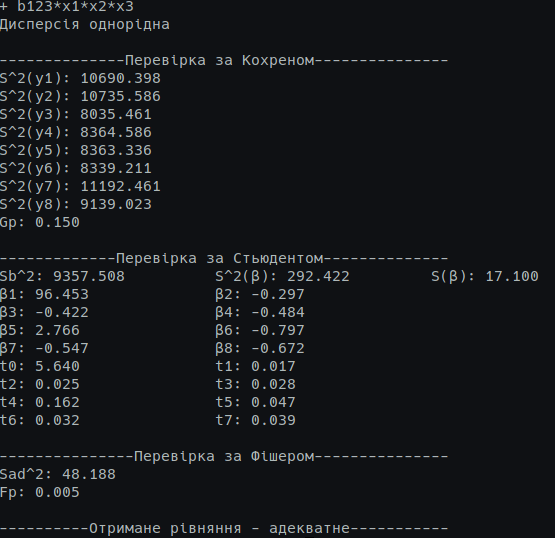
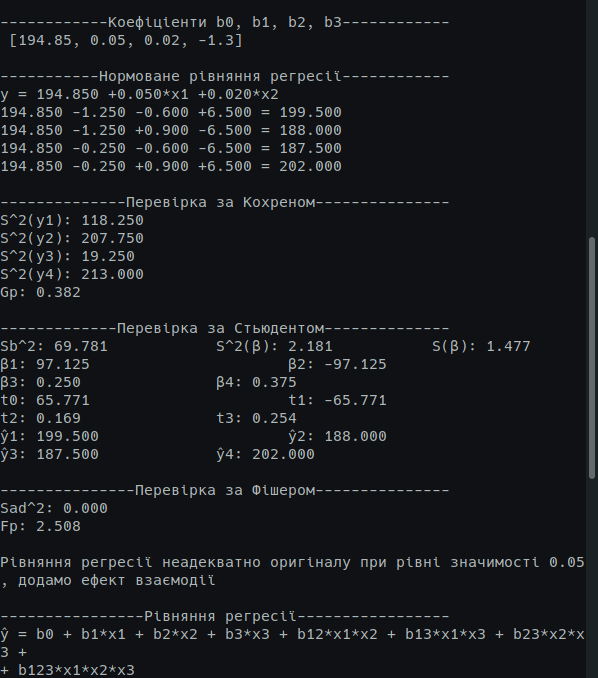
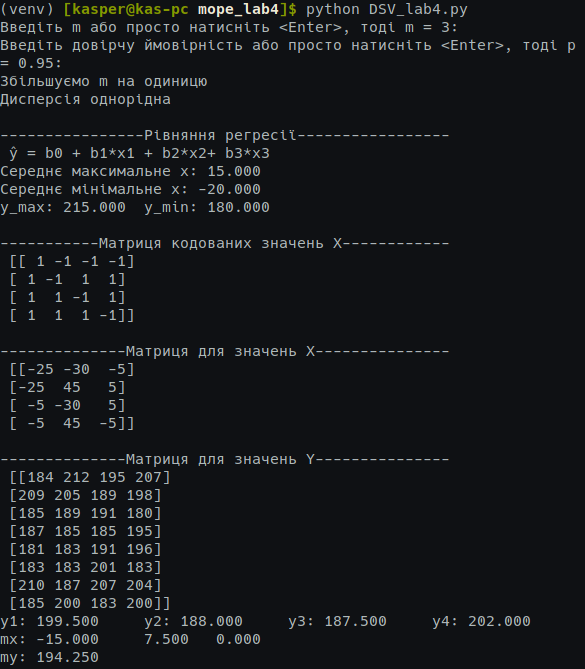
else:

cont = True

print(f"{'Отримане рівняння - неадекватне':-^50}\n"

f"Врахування ефекту взаємодії не допомогло.")

**Результат виконання роботи програми:**



**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.