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 ІО-83

Тимочко Дмитро

Залікова книжка № 8326

Перевірив Регіда П. Г.

Київ 2020р.

**Лабораторна робота №4**

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

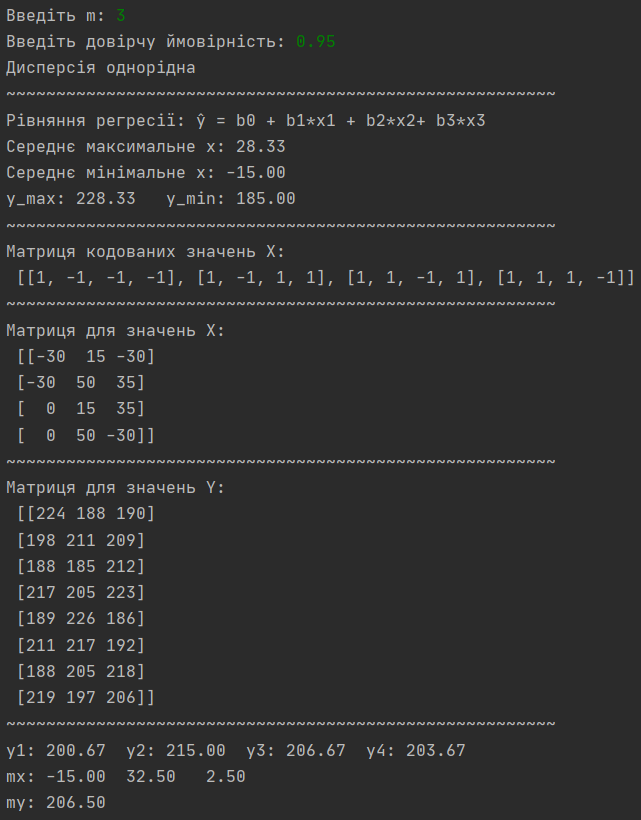
**Мета:** Провести повний трьохфакторний експеримент. Знайти рівняння регресії адекватне об'єкту.

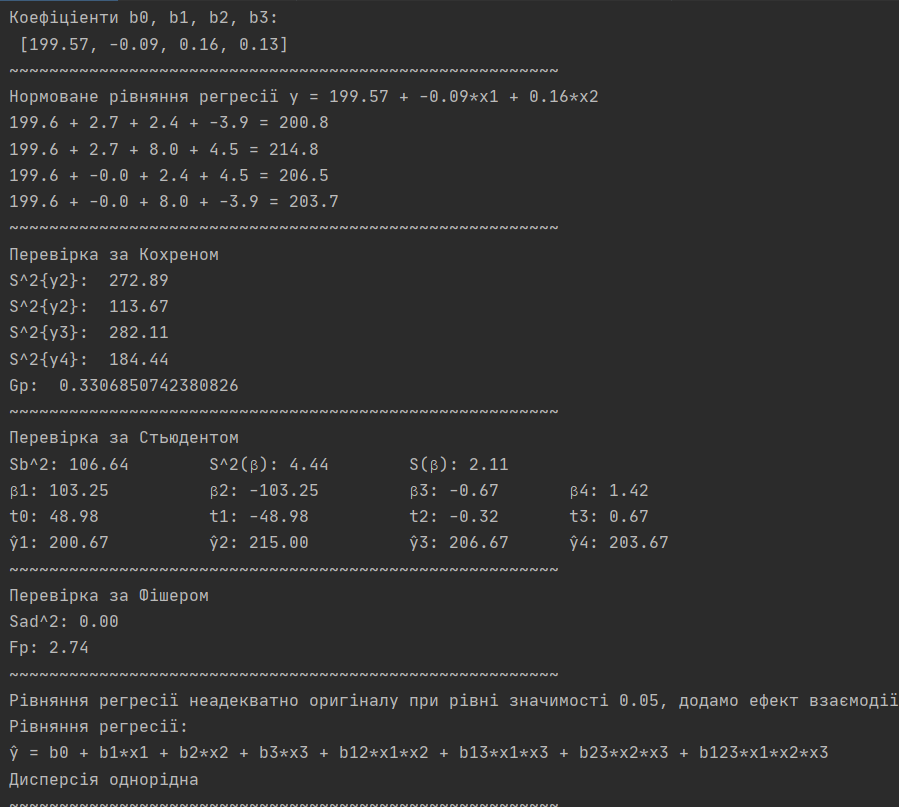
**Виконання:**

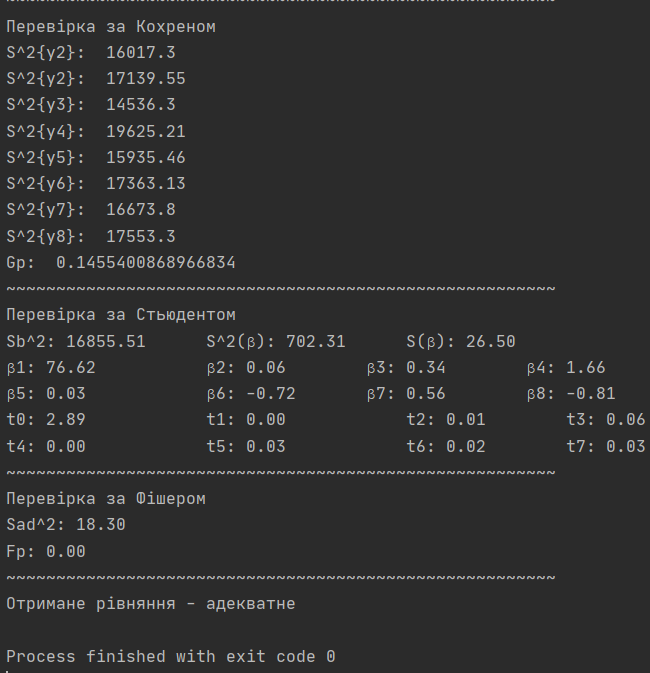
Варіант – 325.



1. Лістинг програми:
2. **from** numpy **import** \*  
   **from** math **import** \*  
   **import** numpy **as** np  
     
     
   **class** Critical\_values:  
    @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 = Critical\_values()  
     
     
   **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]  
     
     
   m = int(input(**"Введіть m: "**))  
   p = float(input(**"Введіть довірчу ймовірність: "**))  
     
   rows = N = 8  
   x1\_min, x1\_max = -30, 0  
   x2\_min, x2\_max = 15, 50  
   x3\_min, x3\_max = -30, 35  
   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\_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 = Critical\_values.get\_cohren\_value(f2, f1, q)  
    **if** Gp <= Gt:  
    print(**"Дисперсія однорідна"**)  
    check = **False  
    else**:  
    m += 1  
    print(**"Збільшуємо m на одиницю"**)  
     
   *# ==============================Оцінимо 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))  
     
     
   *# Визначимо 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)  
     
     
   *# Запишемо 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==============================*d = tbool.count(**True**) *# кількість значимих коефіціентів*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(**"~"** \* 55)  
   print(**"Рівняння регресії: ŷ = b0 + b1\*x1 + b2\*x2+ b3\*x3 "**)  
   print(**"Середнє максимальне х: {:.2f}"**.format(x\_avarage\_max))  
   print(**"Середнє мінімальне х: {:.2f}"**.format(x\_avarage\_min))  
   print(**"y\_max: {:.2f} \ty\_min: {:.2f}"**.format(y\_max, y\_min))  
   print(**"~"** \* 55)  
   print(**"Матриця кодованих значень Х: \n"**, matrix\_x\_cod\_for4)  
   print(**"~"** \* 55)  
   print(**"Матриця для значень Х: \n"**, matrix\_x\_for4)  
   print(**"~"** \* 55)  
   print(**"Матриця для значень Y: \n"**, random\_matrix\_y)  
   print(**"~"** \* 55)  
   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(**"~"** \* 55)  
   print(**"Коефіціенти b0, b1, b2, b3: \n"**, X)  
   print(**"~"** \* 55)  
   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(**"~"** \* 55)  
   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(**"~"** \* 55)  
   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\t\tβ2: {:.2f} \t\tβ3: {:.2f} \t\tβ4: {:.2f}"**.format(beta0, beta1, beta2, beta3))  
   print(**"t0: {:.2f} \t\t\tt1: {:.2f} \t\t\tt2: {:.2f} \t\tt3: {:.2f}"**.format(t0, t1, t2, t3))  
   print(**"ŷ1: {:.2f} \t\t\tŷ2: {:.2f} \t\t\tŷ3: {:.2f} \t\tŷ4: {:.2f}"**.format(yj1, yj2, yj3, yj4))  
   print(**"~"** \* 55)  
   print(**"Перевірка за Фішером"**)  
   print(**"Sad^2: {:.2f} \nFp: {:.2f}"**.format(Sad, Fp))  
   print(**"~"** \* 55)  
     
   **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(**"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 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 = Critical\_values.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(**"~"** \* 55)  
    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(**"~"** \* 55)  
    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(**"~"** \* 55)  
    print(**"Перевірка за Фішером"**)  
    print(**"Sad^2: {:.2f} \nFp: {:.2f}"**.format(S2\_ad, Fp))  
    print(**"~"** \* 55)  
    **if** Fp < Ft:  
    print(**"Отримане рівняння - адекватне"**)  
    cont = **False  
    else**:  
    cont = **True** print(**"Отримане рівняння - неадекватне. Врахування ефекту взаємодії не допомогло."**)
3. Результат виконання роботи програми:







**Висновок:** Отже, у ході виконання лабораторної роботи № 4 провели повний трьохфакторний експеримент при використанні рівняння з ефектом взаємодії. Склали матрицю планування, знайшли коефіцієнти рівняння регресії, провели 3 статистичні перевірки. Була написана текстова програма, результати наведені вище. Результати співпадають із калькулятором. Кінцева мета роботи досягнута!